Lending Club - Predicting Defaults on Loans

Josh Janda

08 December, 2019

Introduction

For my project, I will be evaluating the Lending Club loan dataset. In specific, I will be evaluating loans from the 2017 Quarter 1 period. This dataset includes information of all loans given by LendingClub.com. The goal of this project is to be able to predict the probability of default of a loan given a certain set of features. Some of the most important features (in my opinion), are:

- *annual_inc*: The self-reported annual income provided by the borrower during registration.
- *chargeoff_within_12_mths*: Number of charge-offs within 12 months
- *emp_length*: Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.
- *grade*: Lending Club assigned loan grade
- *sub_grade*: Lending Club assigned loan subgrade
- *home_ownership*: The home ownership status provided by the borrower during registration or obtained from the credit report. Our values are: RENT, OWN, MORTGAGE, OTHER
- *int_rate*: Interest Rate on the loan
- *loan_amt*: The listed amount of the loan applied for by the borrower.
- *pub_rec_bankruptcies*: Number of public record bankruptcies
- *totalAcc*: The total number of credit lines currently in the borrower's credit file

While there are many more variables, 145 to be exact, I believe these are some of the most important features that will help predict whether or not a loan will default.

Moving onto our target (independent) variable, we will be trying to predict:

- *loan_status*: Current status of the loan

Overall, my goal for this project is to utilize multiple machine learning algorithms/techniques to achieve a highly accurate model for predicting loan default given attributes for a loanee.

Summary Statistics

Now that we have introduced the data and our goal for it, we will want to load the data. But first, let's start with importing all needed libraries to run this project.

```
library(vroom)
library(boot)
library(broom)
library(ggplot2)
library(knitr)
library(tidyverse)
library(zoo)
library(caret)
library(dataPreparation)
```

```
library(e1071)
library(glmnet)
library(doParallel)
library(tree)
library(randomForest)
library(gbm)
library(xgboost)
library(keras)

cl = makeCluster(detectCores(logical=FALSE))
registerDoParallel(cl)
```

Now, let's load the data.

```
lending_data = vroom('loan.csv', delim = ",", na = "")
```

```
## Observations: 2,260,668
## Variables: 145
## chr [ 36]: term, grade, sub_grade, emp_title, emp_length, home_ownership, verification_...
## dbl [106]: loan_amnt, funded_amnt, funded_amnt_inv, int_rate, installment, annual_inc, ...
## lgl [  3]: id, member_id, url
##
## Call `spec()` for a copy-pastable column specification
## Specify the column types with `col_types` to quiet this message
```

We can now take a full look at the total number of observations and variables in this dataset.

```
total_obs1 = nrow(lending_data)
total_vars1 = ncol(lending_data)
```

Total observations in this dataset: 2260668

Total variables in this dataset: 145

Target Variable Exploration

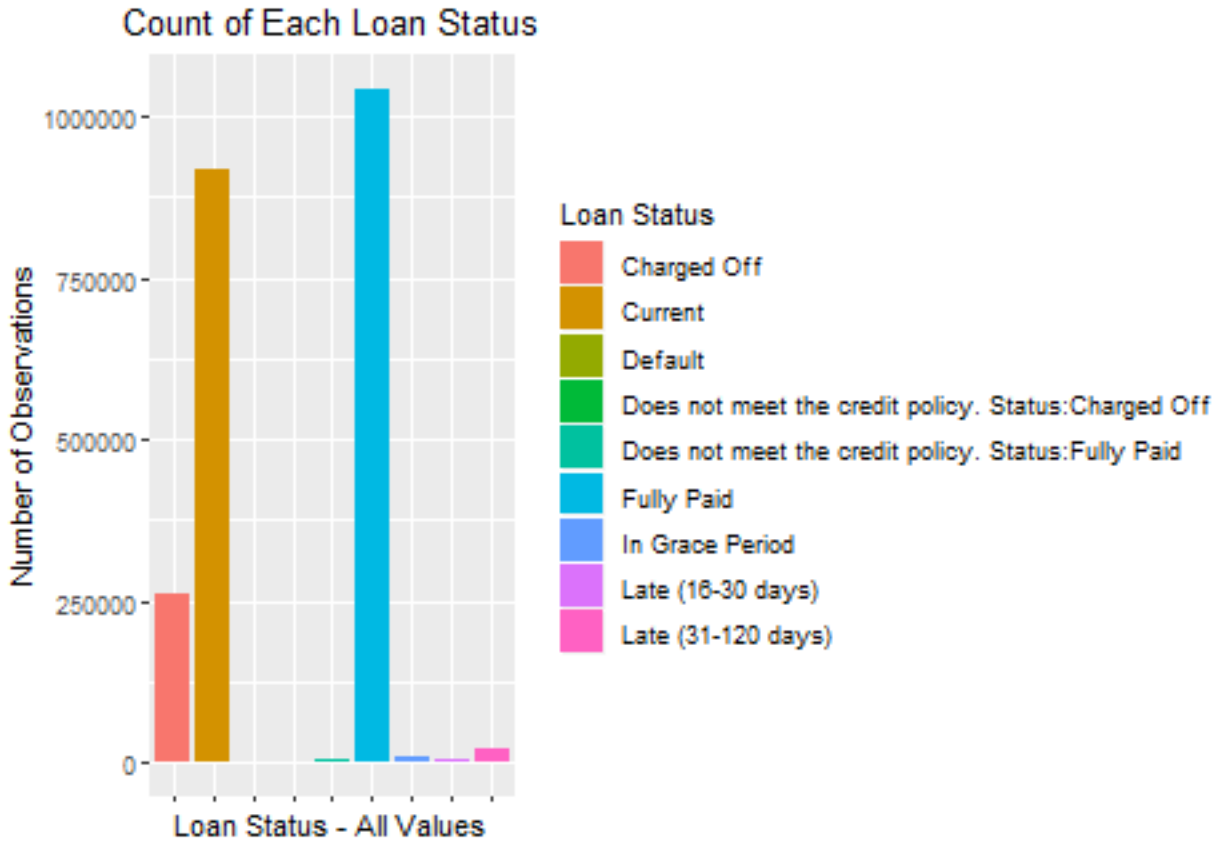
So, as stated above, there are a lot of variables given to help predict whether or not a person will default on their loan. Let's explore our target variable, *loan_status*, more closely to get an idea of what possible values this variable can take.

```
unique(lending_data$loan_status)
```

```
## [1] "Current"
## [2] "Fully Paid"
## [3] "Late (31-120 days)"
## [4] "In Grace Period"
## [5] "Charged Off"
## [6] "Late (16-30 days)"
## [7] "Default"
## [8] "Does not meet the credit policy. Status:Fully Paid"
## [9] "Does not meet the credit policy. Status:Charged Off"
```

There are a total of 9 different values that *loan_status* can take. Since we are interested in predicting whether someone will default on their loan or not, we will clean this variable later on so it only takes on values default or not default. Forn now, let's take a look at how many of each value we have using a bar graph.

```
ggplot(data = lending_data, aes(x = factor(loan_status), fill = loan_status)) +
  geom_bar() +
  theme(axis.text.x = element_blank(), plot.title = element_text(hjust = 0.5)) +
  ggtitle("Count of Each Loan Status") +
  xlab("Loan Status - All Values") +
  ylab("Number of Observations") +
  labs(fill = "Loan Status")
```



Looking at the bar graph above, we can see that the top three values that *loan_status* takes is "Charged Off", "Current", and "Fully Paid". All other values are very low in the number of observations.

Since we are interested in predicting whether or not someone will default on a loan, we can remove the values that will be not helpful in making this prediction. I will justify why I am removing each value from *loan_status* that I believe is not helpful.

- "Current": If you are current on your loan, this does not mean you will remain current. You can eventually default on the loan or pay it off, we do not know.
- "In Grace Period": If you are in the grace period, you can always go back to being current on your loan or eventually default. We do not know.
- "Late (16-30 days)": If you are late on paying your loan, you can always catch up on payments or eventually default. We do not know.

- "Late (31-120 days)": If you are late on paying your loan, you can always catch up on payments or eventually default. We do not know.

So, I will be keeping these values to eventually create a single variable that has the values "Default" or "Not Default"

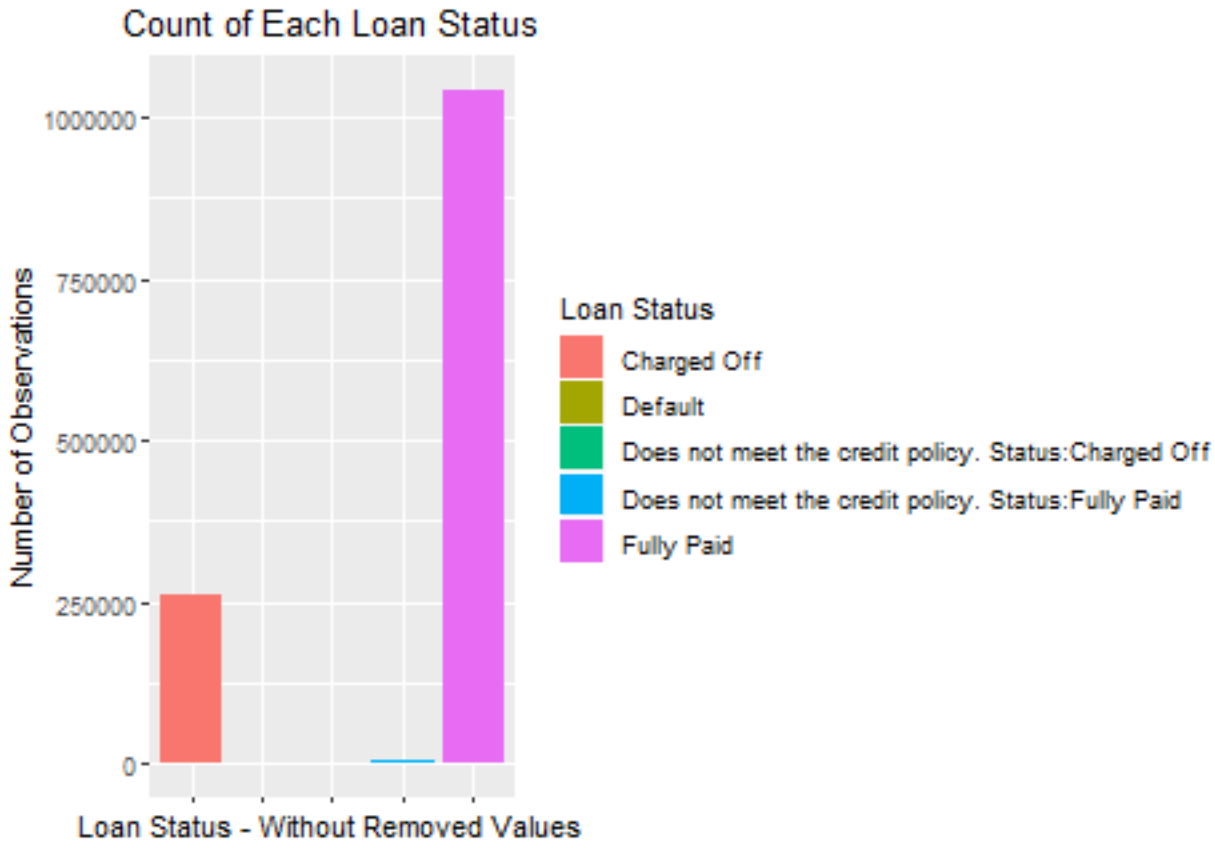
- "Charged Off": This loan was defaulted on and sent to collections, and Lending Club has charged off the loan and considered it a loss.
- "Default": This loan was defaulted on.
- "Does not meet the credit policy. Status: Charged Off": This loan was defaulted on and sent to collections, and Lending Club has charged off the loan and considered it a loss.
- "Does not meet the credit policy. Status: Fully Paid": This loan was fully paid off and was not defaulted on.
- "Fully Paid": This loan was fully pauid off and was not defaulted on.

Let's clean the data a little bit now by removing all rows that have any of the four values for *loan_status* in order to remove these values.

```
'%ni%' = Negate('%in%') #create function that does the opposite of %in%

lending_data_target_clean = lending_data %>% filter(
  loan_status %ni% c("Current", "In Grace Period", "Late (16-30 days)", "Late (31-120 days)"))

rm(list = setdiff(ls(), "lending_data_target_clean"))
total_obs2 = nrow(lending_data_target_clean)
 # clear environment
```

With those values of *loan_status* removed, we have shrunken the data to only 1306387 rows, which is much smaller and more manageable. Let's take a look at a bar plot of *loan_status* once again.

```
ggplot(data = lending_data_target_clean, aes(x = factor(loan_status), fill = loan_status)) +
  geom_bar() +
  theme(axis.text.x = element_blank(), plot.title = element_text(hjust = 0.5)) +
  ggtitle("Count of Each Loan Status") +
  xlab("Loan Status - Without Removed Values") +
  ylab("Number of Observations") +
  labs(fill = "Loan Status")
```

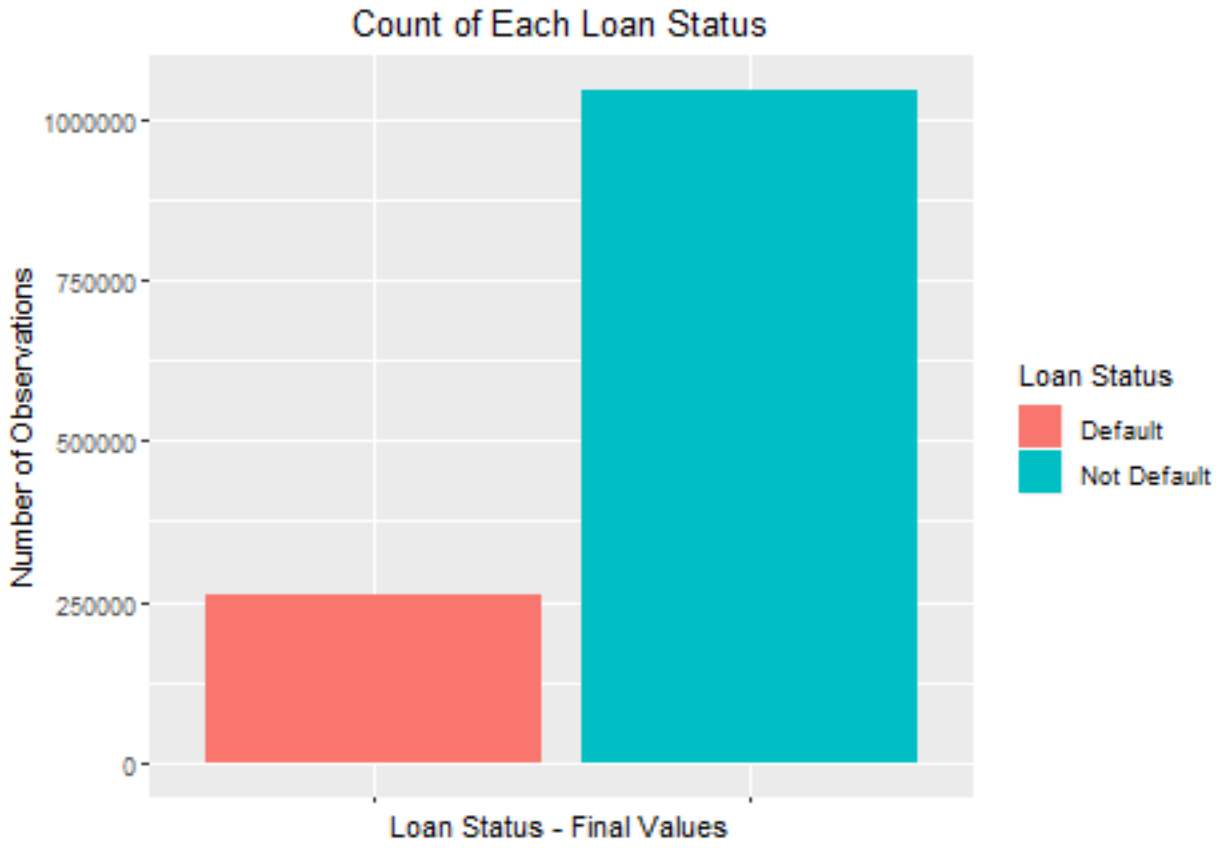


This plot looks much cleaner with the redudant levels of *loan_status* removed. Let's now create our final variable of *loan_status*, that we will use for model building. This variable will include levels of "Default" or "Not Default".

```
lending_data_target_clean$loan_status_final = ifelse(
  lending_data_target_clean$loan_status %in% c("Charged Off",
    "Default", "Does not meet the credit policy. Status:Charged Off"), "Default", "Not Default")
lending_data_target_clean = lending_data_target_clean[, !(names(lending_data_target_clean) %in% c("loan
```

Let's take a look at the bar graph one more time for our created variable *loan_status_final*.

```
ggplot(data = lending_data_target_clean, aes(x = factor(loan_status_final), fill = loan_status_final)) +
  geom_bar() +
  theme(axis.text.x = element_blank(), plot.title = element_text(hjust = 0.5)) +
  ggtitle("Count of Each Loan Status") +
  xlab("Loan Status - Final Values") +
  ylab("Number of Observations") +
  labs(fill = "Loan Status")
```



With this variable created, we will now consider this to be our target variable that we will be using for model building and further visualization.

Feature Exploration

With the data cleaned and redudant rows removed, let's move on to exploring some of the important "X" variables, or features in this dataset. I will be taking a look at the three most important features, which I have picked.

- *annual_inc*: The self-reported annual income provided by the borrower during registration.
- *int_rate*: Interest Rate on the loan
- *loan_amt*: The listed amount of the loan applied for by the borrower.

I have picked *annual_inc* as I believe this will play a crucial factor on whether someone will default on their loan or not. My belief is that the higher the income someone has, the less chance they have of defaulting on the loan. Let's explore some summary statistics on this variable and also the distribution of it through the use of a histogram.

```
summary(lending_data_target_clean$annual_inc)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's
##        0    45760    65000    76149    90000 10999200        4
```

Looking at the summary statistics of *annual_inc*, we have some interesting results to make note of. First of all, we need to pay attention to the number of "NA's" in this variable, which is 4. For these NA's, we can either remove the rows or impute a value into them. We will make this decision later on. Our minimum

annual_inc is zero, which seems off as I would not believe that a loan would be given to someone with zero income. Our maximum *annual_inc* is 10,999,200, which also seems off as that is a lot of income and I am doubtful someone would need a loan with that much income. The average *annual_inc* is 76,149, which is a reasonable average for annual incomes on people requesting loans. It should be noted that *annual_inc* is a field that the loanee provides to Lending Club, so it is safe to believe that some people did not enter in their true annual income.

Let's now take a look at the summary of *annual_inc*, when grouped by whether or not they have defaulted on their loan. We will be ignoring the NA values for this summary.

```
lending_data_target_clean %>% group_by(loan_status_final) %>% summarize(
  MinInc = min(annual_inc, na.rm = TRUE),
  MeanInc = mean(annual_inc, na.rm = TRUE),
  MaxInc = max(annual_inc, na.rm = TRUE))
```

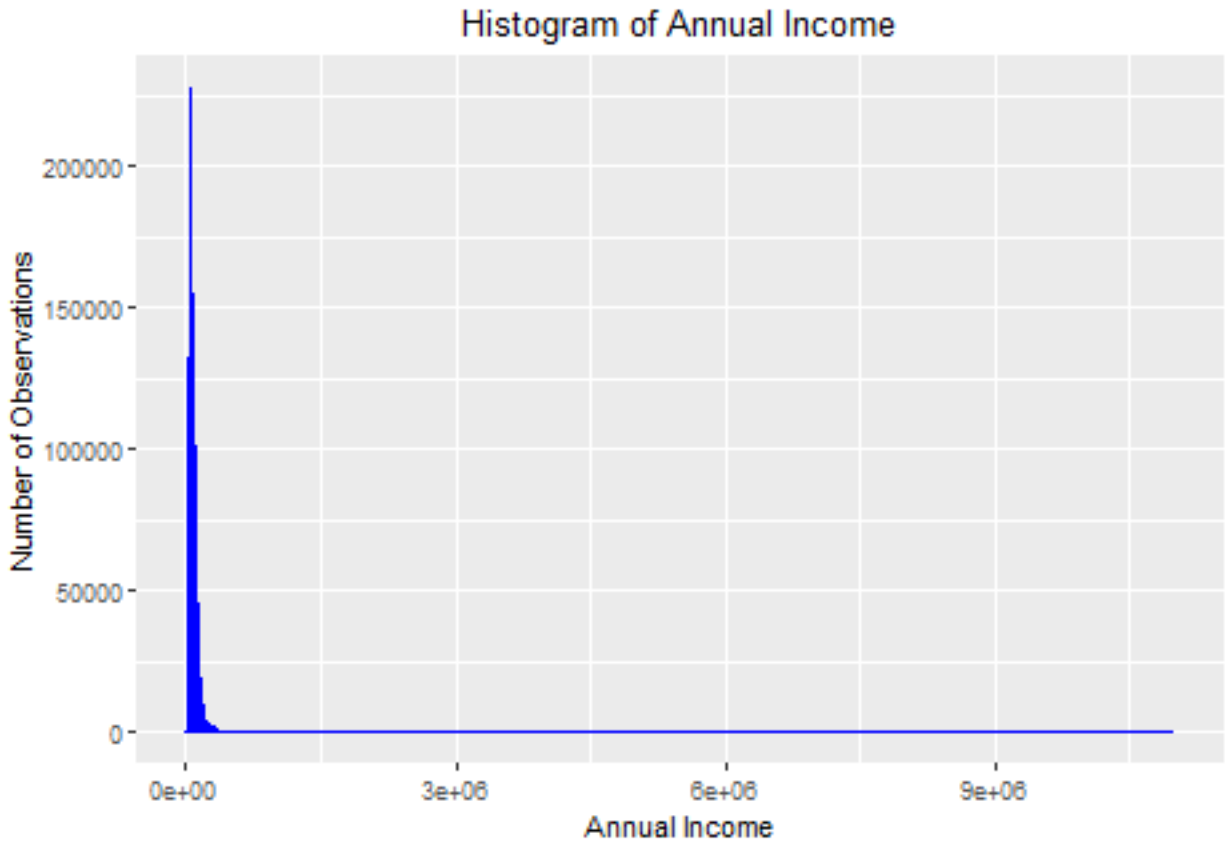
```
## # A tibble: 2 x 4
##   loan_status_final MinInc MeanInc   MaxInc
##   <chr>              <dbl>   <dbl>    <dbl>
## 1 Default                0  70325.  9500000
## 2 Not Default            0  77613. 10999200
```

Looking at the summary when grouped by whether or not the person has defaulted on their loan, we can see that the minimum income and maximum income really do not differ by much as each group has a minimum of zero and a maximum of a ridiculously high number. The mean incomes however differ by a good amount, about 7,000. This difference in means falls in limne with my belief that the higher someones income is the less chance they have of defaulting on their loan.

With summary statistics aside, let's take a look at the histogram of *annual_inc* to get an idea of the distribution.

```
ggplot(lending_data_target_clean, aes(x = annual_inc)) +
  geom_histogram(stat = "bin", bins = 1000, color = "blue") +
  xlab("Annual Income") +
  ylab("Number of Observations") +
  ggtitle("Histogram of Annual Income") +
  theme(plot.title = element_text(hjust = 0.50))
```

```
## Warning: Removed 4 rows containing non-finite values (stat_bin).
```

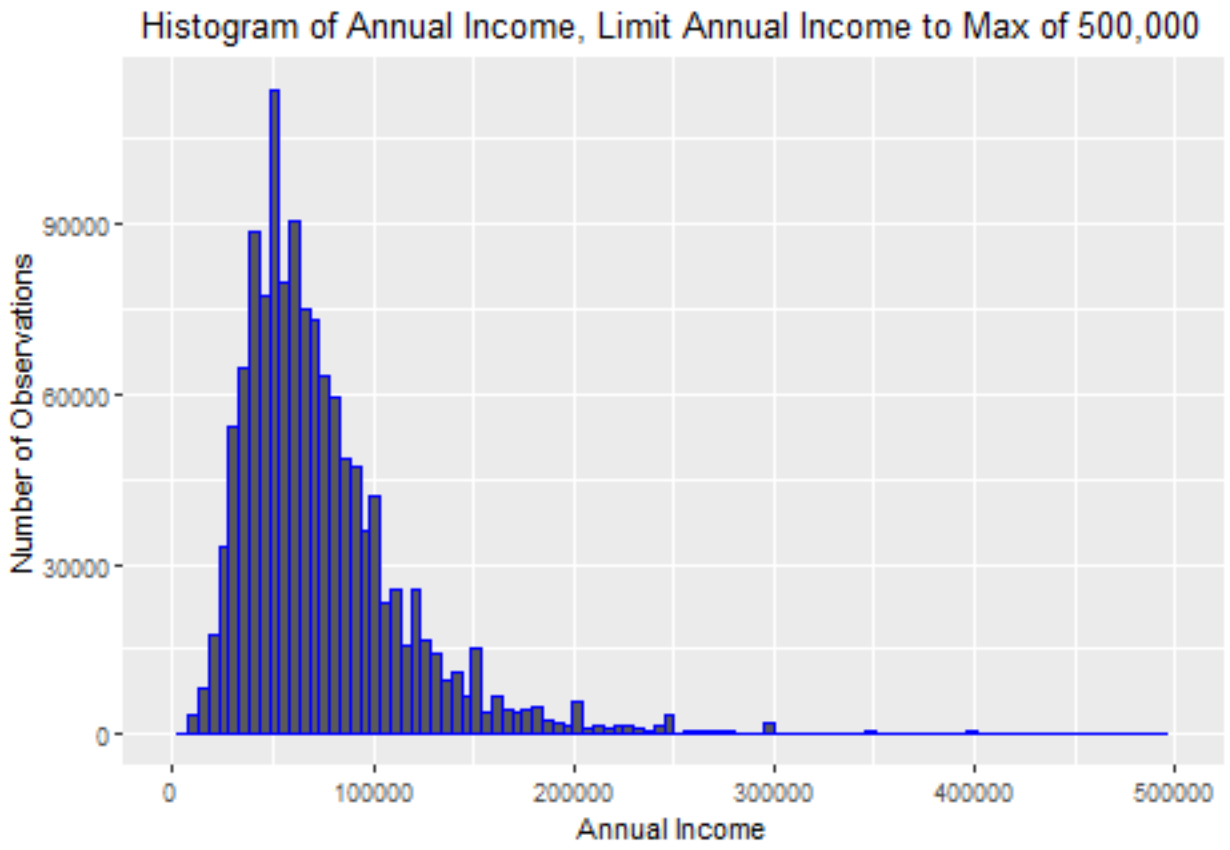


Looking at the histogram above, we don't really get a good idea of the distribution of *annual_inc* when allowing the range of the x-axis to span all of the way to the maximum value of 10,999,200. I am going to also include a histogram of *annual_inc* with an xrange of (0, 500,000) to get a better idea of the distribution. I have chosen 500,000 as that seems to be around the flatline of this histogram, also diving deeper into the data there are only 1675 observations greater than 500,000 which is only 0.1282162% of the data which is very minimal.

```
options(scipen = 10)
ggplot(lending_data_target_clean, aes(x = annual_inc)) +
  geom_histogram(stat = "bin", bins = 100, color = "blue") +
  xlim(0, 500000) +
  xlab("Annual Income") +
  ylab("Number of Observations") +
  ggtitle("Histogram of Annual Income, Limit Annual Income to Max of 500,000") +
  theme(plot.title = element_text(hjust = 0.50))
```

```
## Warning: Removed 1679 rows containing non-finite values (stat_bin).
```

```
## Warning: Removed 2 rows containing missing values (geom_bar).
```



Looking at the histogram of *annual_inc* with a limit of 500,000, we can see that most incomes surround the mean as the histogram has a very high peak. This distribution looks normal-ish if you remove outliers, however does have a fatter right tail as incomes can continue to increase while they cannot go below zero (I hope).

Moving on from *annual_inc*, let's now take a *int_rate*. I believe that this variable plays a crucial role in predicting the target variable of whether or not someone will default on their loan as interest rates are tied directly to how risky the loanee is. The more risky it is to borrow someone money (less chance that they will pay back the loan), the higher the interest rate you would charge them. Therefore, the higher the interest rate the higher the chance of someone defaulting on a loan. Let's explore some summary statistics on this variable and also the distribution of it through the use of a histogram.

```
summary(lending_data_target_clean$int_rate)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    5.31    9.75   12.74   13.26   15.99   30.99
```

Looking at the summary statistics of *int_rate*, we have some interesting results to make note of. For the case of this variable, we have no NA values which is fantastic. Our minimum *int_rate* is 5.31%, which is a good interest rate for a loan. Our maximum *int_rate* is 30.99%, which is a very high interest rate on a loan and therefore this person was most likely very risk to give a loan to. The average *int_rate* is 13.26%, which is a reasonable average interest rate for loans given to all types of people.

Let's now take a look at the summary of *int_rate*, when grouped by whether or not they have defaulted on their loan.

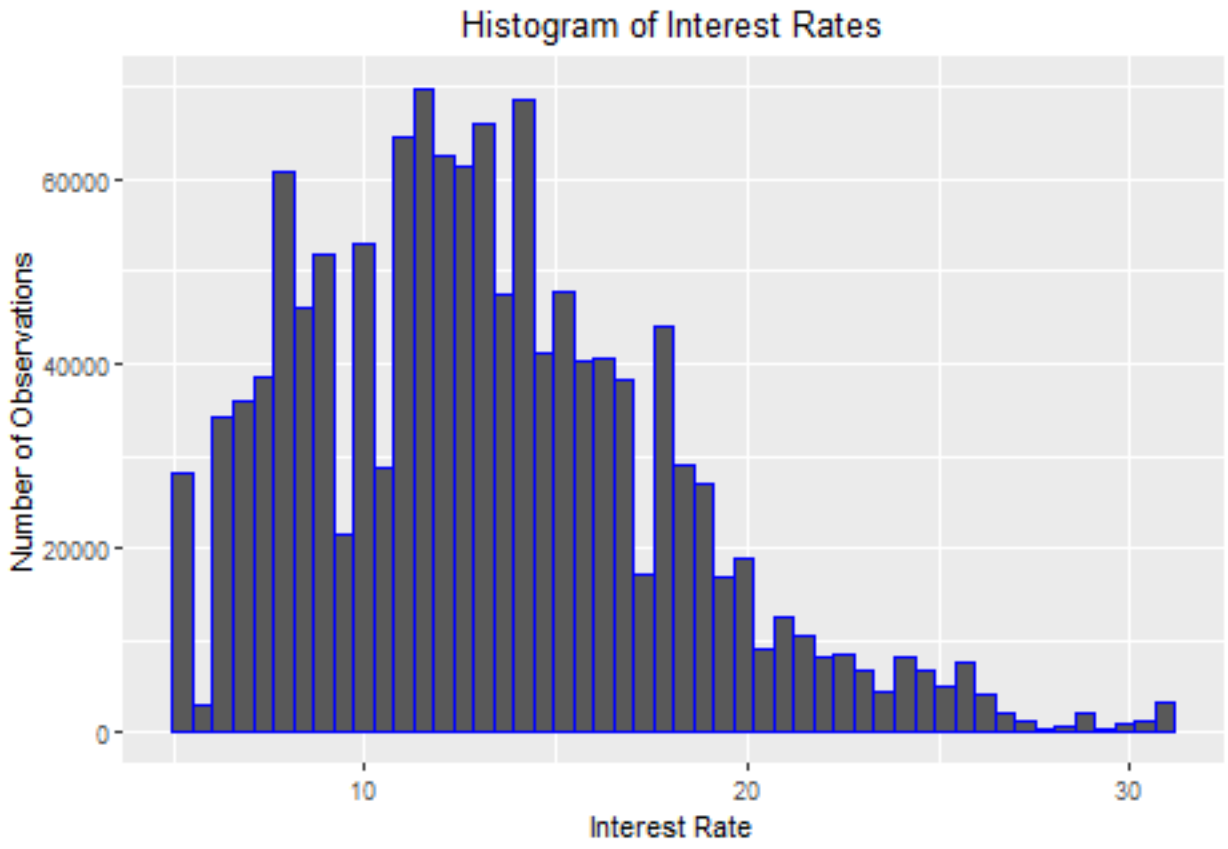
```
lending_data_target_clean %>% group_by(loan_status_final) %>% summarize(
  MinIntRate = min(int_rate, na.rm = TRUE),
  MeanIntRate = mean(int_rate, na.rm = TRUE),
  MaxIntRate = max(int_rate, na.rm = TRUE))
```

```
## # A tibble: 2 x 4
##   loan_status_final MinIntRate MeanIntRate MaxIntRate
##   <chr>                  <dbl>       <dbl>      <dbl>
## 1 Default                 5.31        15.7       31.0
## 2 Not Default             5.31        12.6       31.0
```

Looking at the statistics of *int_rate* when grouped by whether or not someone has defaulted on their loan, once again we see that the minimum and maximum values are similar (equal in this case). I believe they are equal due to this being the minimum and maximum interest rates offered by Lending Club. However, when looking at the mean *int_rate* between groups, we can see that those who did not default on their loan have a lower mean interest rate. This falls in line with my belief that those with lower interest rates are less risky and therefore have less chance to default on their loan.

With summary statistics aside, let's take a look at the histogram of *int_rate* to get an idea of the distribution.

```
ggplot(lending_data_target_clean, aes(x = int_rate)) +
  geom_histogram(stat = "bin", bins = 50, color = "blue") +
  xlab("Interest Rate") +
  ylab("Number of Observations") +
  ggtitle("Histogram of Interest Rates") +
  theme(plot.title = element_text(hjust = 0.50))
```



Looking at the histogram of *int_rate*, it appears to have a normal-ish distribution with most interest rates being near the mean. However, it is right-skewed as more risky people receive higher interest rates therefore causing the distribution to have a fatter right tail.

Lastly, let's take a look at *loan_amt*. I believe that this variable also plays a crucial role in predicting whether or not someone will default on their loan as the larger the loan amount someone takes out I believe there is less chance that they plan on (and are capable of), repaying the loan. However, it should be noted that this is my belief and there could be cases where someone with more income needs a larger loan and is therefore capable of paying it back. Let's explore some summary statistics on this variable and also the distribution of it through the use of a histogram.

```
summary(lending_data_target_clean$loan_amnt)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     500    8000   12000   14406   20000   40000
```

Looking at the summary statistics of *loan_amnt*, we have some interesting results to make note of. For the case of this variable, we have no NA values which is fantastic. Our minimum *loan_amnt* is 500, which is a rather small loan and is likely to be paid back. Our maximum *loan_amnt* is 40,000, which is a larger-than-average loan and therefore carries more risk of being defaulted on. The average *loan_amnt* is 14,406, which seems high for an average of the given loan amounts.

Let's now take a look at the summary of *loan_amnt*, when grouped by whether or not they have defaulted on their loan.

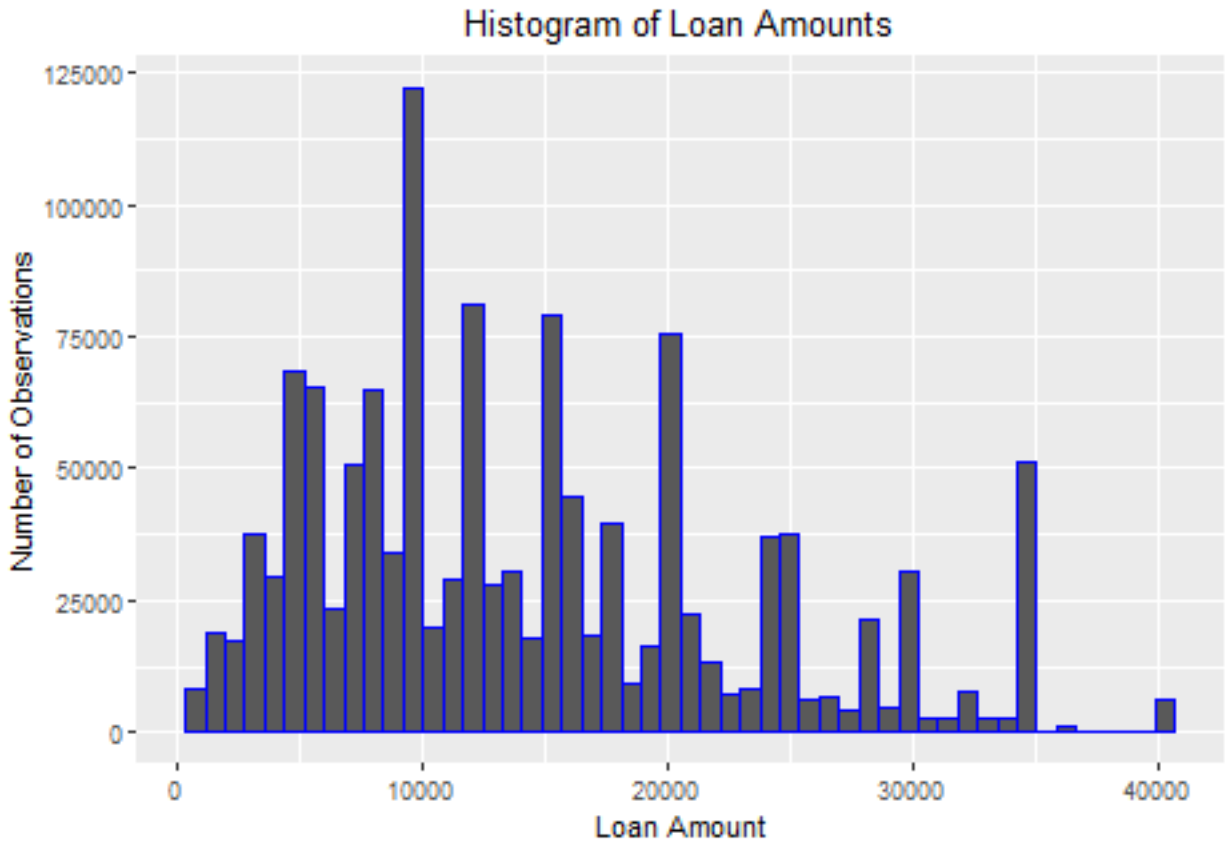
```
lending_data_target_clean %>% group_by(loan_status_final) %>% summarize(
  MinLoanAmnt = min(loan_amnt, na.rm = TRUE),
  MeanLoanAmnt = mean(loan_amnt, na.rm = TRUE),
  MaxLoanAmnt = max(loan_amnt, na.rm = TRUE))
```

```
## # A tibble: 2 x 4
##   loan_status_final MinLoanAmnt MeanLoanAmnt MaxLoanAmnt
##   <chr>                   <dbl>        <dbl>       <dbl>
## 1 Default                   500       15532.       40000
## 2 Not Default               500       14122.       40000
```

Looking at the statistics of *loan_amnt* when grouped by whether or not someone has defaulted on their loan, once again we see that the minimum and maximum values are similar (equal in this case). I believe they are equal due to this being the minimum and maximum loan amounts offered by Lending Club. However, when looking at the mean *loan_amnt* between groups, we can see that those who did not default on their loan have a lower mean loan amount. This falls in line with my belief that those with a lower loan amount have less chance of defaulting on their loan.

With summary statistics aside, let's take a look at the histogram of *loan_amnt* to get an idea of the distribution.

```
ggplot(lending_data_target_clean, aes(x = loan_amnt)) +
  geom_histogram(stat = "bin", bins = 50, color = "blue") +
  xlab("Loan Amount") +
  ylab("Number of Observations") +
  ggtitle("Histogram of Loan Amounts") +
  theme(plot.title = element_text(hjust = 0.50))
```



Looking at the histogram of *loan_amnt*, I cannot see any real distribution underlying the data. I believe that this could be due to people needing different loan amounts for different needs, therefore leaving no real pattern and distribution to follow.

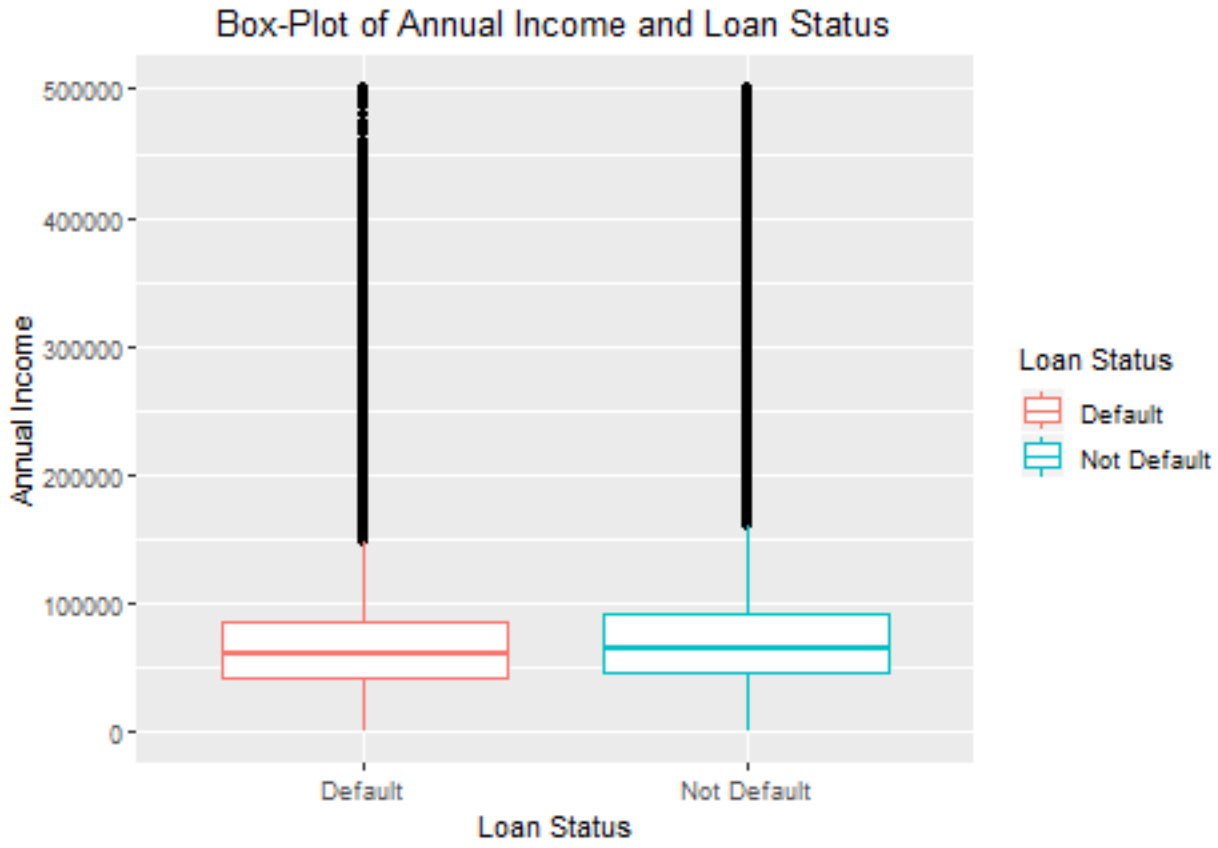
With the exploration of our target variable as well as three of my believed important variables, we are now going to take a look at the box-plot between each of these three variables, *annual_inc*, *int_rate*, and *loan_amnt*, and the output variable, *loan_status*. The box-plot will provide us a graph that will differentiate between each level of *loan_status* to take a look at the difference in mean and variance and overall density of each level.

Comparing X's vs Y

Let's first take a look at the box-plot between *annual_inc* and *loan_status*. Note that I will be once again limiting the annual income to a maximum of 500,000 to disregard outliers and obvious incorrect incomes.

```
ggplot(lending_data_target_clean, aes(x = loan_status_final, y = annual_inc, color = loan_status_final))
  geom_boxplot(outlier.colour = "black") +
  ylim(0, 500000) +
  labs(x = "Loan Status",
       y = "Annual Income",
       title = "Box-Plot of Annual Income and Loan Status",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
```

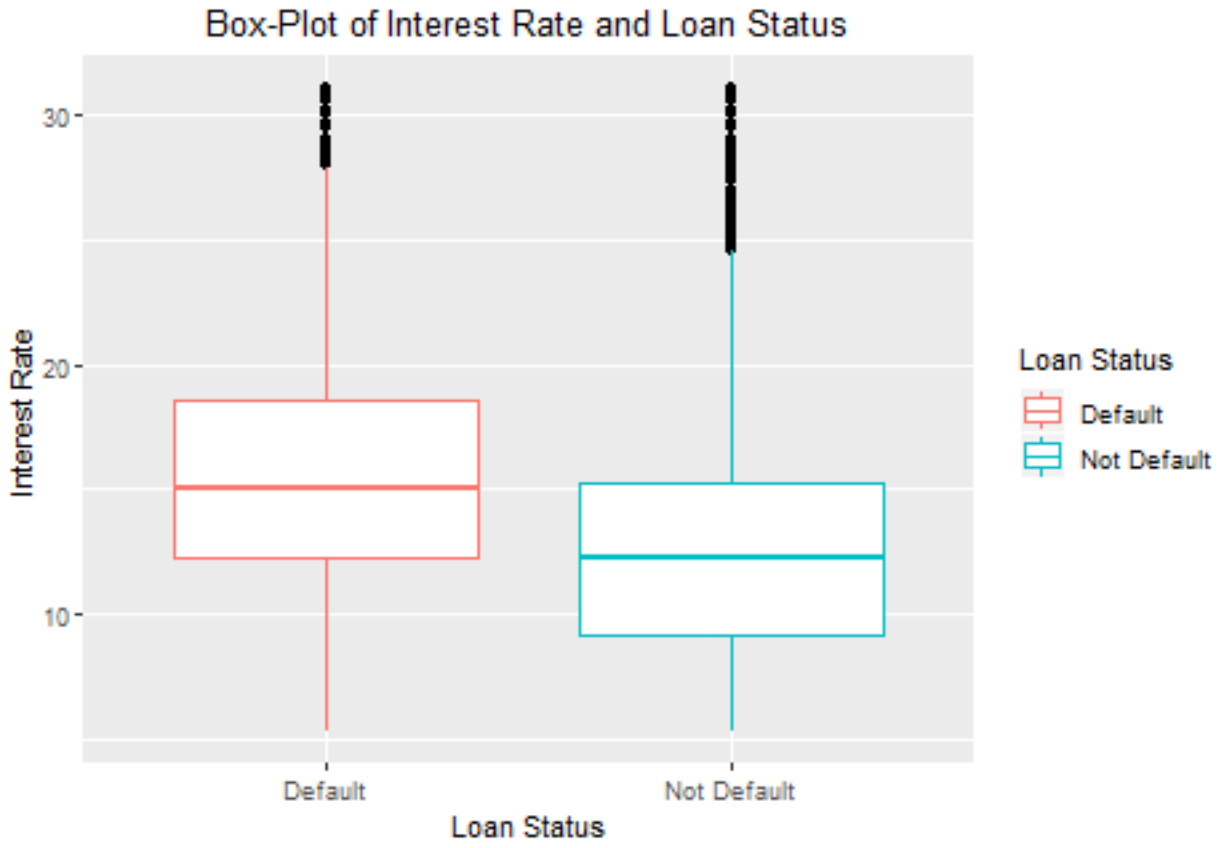
```
## Warning: Removed 1679 rows containing non-finite values (stat_boxplot).
```



Looking at the plot above, we can see that there is a very small difference between groups "Default" and "Not Default". This difference is about 7,000, as noted above when discussing the *annual_inc* variable. We can also see that there are many outliers in this variable, as noted by the observations that are black.

Let's now take a look at the box-plot between *int_rate* and *loan_status*.

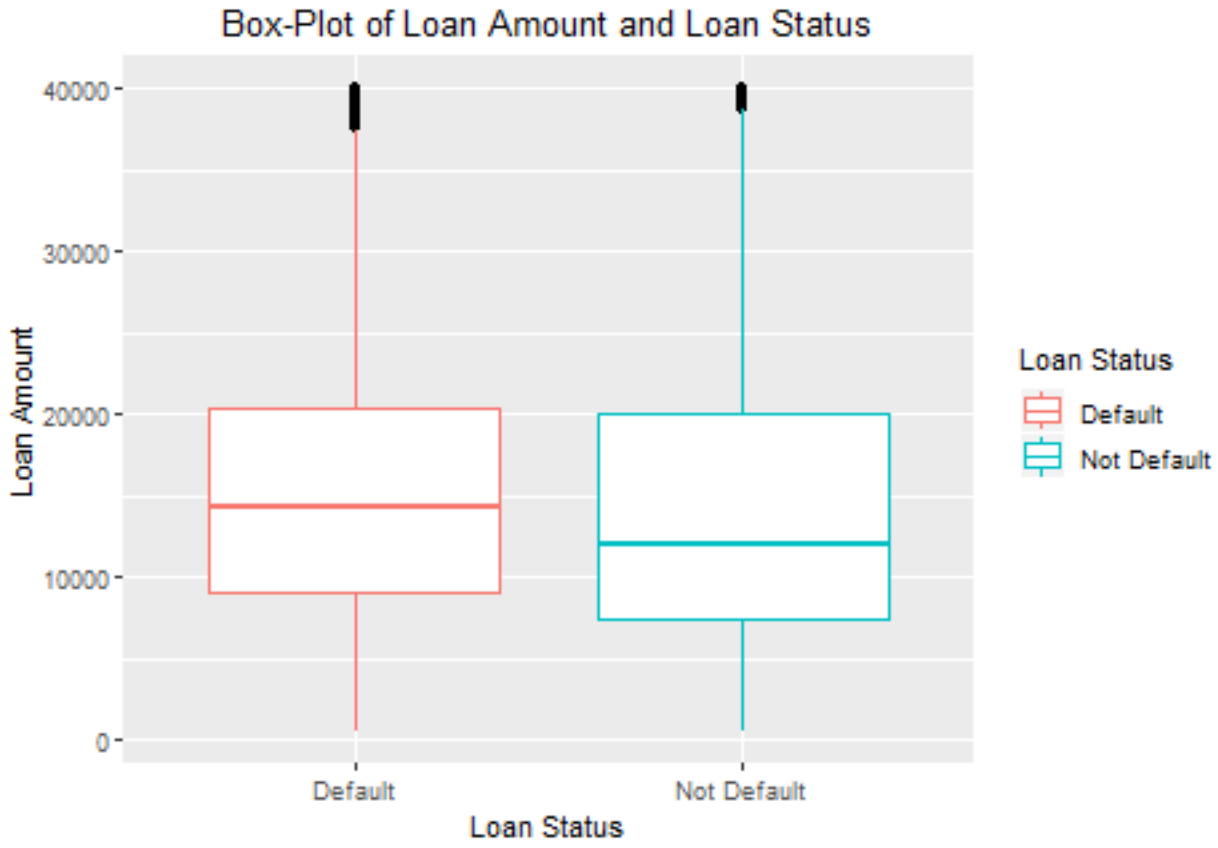
```
ggplot(lending_data_target_clean, aes(x = loan_status_final, y = int_rate, color = loan_status_final)) +
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "Interest Rate",
       title = "Box-Plot of Interest Rate and Loan Status",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
```



Looking at the plot above, we can see that there is a large difference of interest rates between groups of "Default" and "Not Default". When looking at the outliers, we can see that the outliers of the "Default" group start at about an interest rate of 28% while then outliers of the "Not Default" group start at about an interest rate of 24.5%. This tells us that overall, the "Not Default" group has lower interest rates and a lower interest rate tends to point towards a loan that will not be defaulted on.

Lastly, let's take a look at the box-plot between *loan_amnt* and *loan_status*.

```
ggplot(lending_data_target_clean, aes(x = loan_status_final, y = loan_amnt, color = loan_status_final))
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "Loan Amount",
       title = "Box-Plot of Loan Amount and Loan Status",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
```



Looking at the plot above, we can see that the loan amounts between each group does not have a huge difference. The outliers start at about the same amount of 37,000. However, the mean loan amount for the "Not Default" group is about 2,000 lower. The spread of the observations in the "Not Default" group is a little bit larger though.

All in all, I believe that these three variables will play an important role in predicting whether or not someone will default on their loan. This is shown through summary statistics, visualizations of variables, as well as box-plots between each variable and the target variable *loan_status*.

With our initial target variable exploration and three important variables aside, we will now dive deeper into the dataset by exploring all variables. Recall that there are 145 variables in this dataset, which I plan to reduce through exploration.

Missing Values

For starters, let's get a look at the total number of variables that contain NA values.

```
total_vars_with_na = length(colnames(lending_data_target_clean)
                            [colSums(is.na(lending_data_target_clean)) > 0])
```

Our Lending Club data has 112 variables that have at least one NA value. This is a large amount of variables, so let's get some of the most obvious ones out of the way first. I am going to set a rule beforehand that any variable that has more than half of the rows as NA as a redundant variable and therefore will be removed. I am setting this rule as if there is over half the data missing it will cause more harm to our model accuracy due to having to do mass data imputation which could end up being innacurate. Let's filter the data and display all variables being removed.

```
cols_being_removed = colnames(lending_data_target_clean)[colSums(
  is.na(lending_data_target_clean)) > nrow(lending_data_target_clean) / 2]
colnames(lending_data_target_clean)[colSums(
  is.na(lending_data_target_clean)) > nrow(lending_data_target_clean) / 2]
```

```
##  [1] "id"
##  [2] "member_id"
##  [3] "url"
##  [4] "desc"
##  [5] "mths_since_last_delinq"
##  [6] "mths_since_last_record"
##  [7] "next_pymnt_d"
##  [8] "mths_since_last_major_derog"
##  [9] "annual_inc_joint"
## [10] "dti_joint"
## [11] "verification_status_joint"
## [12] "open_acc_6m"
## [13] "open_act_il"
## [14] "open_il_12m"
## [15] "open_il_24m"
## [16] "mths_since_rcnt_il"
## [17] "total_bal_il"
## [18] "il_util"
## [19] "open_rv_12m"
## [20] "open_rv_24m"
## [21] "max_bal_bc"
## [22] "all_util"
## [23] "inq_fi"
## [24] "total_cu_tl"
## [25] "inq_last_12m"
## [26] "mths_since_recent_bc_dlq"
## [27] "mths_since_recent_revol_delinq"
## [28] "revol_bal_joint"
## [29] "sec_app_earliest_cr_line"
## [30] "sec_app_inq_last_6mths"
## [31] "sec_app_mort_acc"
## [32] "sec_app_open_acc"
## [33] "sec_app_revol_util"
## [34] "sec_app_open_act_il"
## [35] "sec_app_num_rev_accts"
## [36] "sec_app_chargeoff_within_12_mths"
## [37] "sec_app_collections_12_mths_ex_med"
## [38] "sec_app_mths_since_last_major_derog"
## [39] "hardship_type"
## [40] "hardship_reason"
## [41] "hardship_status"
## [42] "deferral_term"
## [43] "hardship_amount"
## [44] "hardship_start_date"
## [45] "hardship_end_date"
## [46] "payment_plan_start_date"
## [47] "hardship_length"
## [48] "hardship_dpd"
```

```
## [49] "hardship_loan_status"
## [50] "orig_projected_additional_accrued_interest"
## [51] "hardship_payoff_balance_amount"
## [52] "hardship_last_payment_amount"
## [53] "debt_settlement_flag_date"
## [54] "settlement_status"
## [55] "settlement_date"
## [56] "settlement_amount"
## [57] "settlement_percentage"
## [58] "settlement_term"
```

Now, let's remove the variables.

```
lending_data_vars_rem = lending_data_target_clean[, !(names(
  lending_data_target_clean) %in% cols_being_removed)]

rm(list = setdiff(ls(), "lending_data_vars_rem"))

total_vars_3 = ncol(lending_data_vars_rem)
```

With those variables removed, we are left with 87 in our data. Let's continue our exploration.

Categorical / Text Variable

With missing values put aside and our number of variables drastically reduced, let's now focus on categorical / text variables and see which variables will be of most use for our end goal of predicting whether or not someone will default on their loan.

```
total_char_vars = length(lending_data_vars_rem
                         %>% select_if(is.character))
```

There are a total of 22 in our data. Let's take a look at each of these variables.

```
colnames(lending_data_vars_rem %>% select_if(is.character))
```

```
##  [1] "term"                 "grade"                "sub_grade"
##  [4] "emp_title"            "emp_length"           "home_ownership"
##  [7] "verification_status"  "issue_d"              "pymnt_plan"
## [10] "purpose"              "title"                "zip_code"
## [13] "addr_state"           "earliest_cr_line"     "initial_list_status"
## [16] "last_pymnt_d"         "last_credit_pull_d"   "application_type"
## [19] "hardship_flag"        "disbursement_method"  "debt_settlement_flag"
## [22] "loan_status_final"
```

Let's go through each variable and give a quick description of it.

- *term* - The number of payments on the loan. Values are in months and can be either 36 or 60.
- *grade* - Lending Club assigned loan grade
- *sub_grade* - Lending Club assigned loan subgrade
- *emp_title* - The job title supplied by the Borrower when applying for the loan.*
- *emp_length* - Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.

- *home_ownership* - The home ownership status provided by the borrower during registration. Our values are: RENT, OWN, MORTGAGE, OTHER.
- *verification_status* - Indicates if income was verified by LC, not verified, or if the income source was verified
- *issue_d* - The month which the loan was funded
- *pymnt_plan* - Indicates if a payment plan has been put in place for the loan
- *purpose* - A category provided by the borrower for the loan request.
- *title* - The loan title provided by the borrower
- *zip_code* - The first 3 numbers of the zip code provided by the borrower in the loan application.
- *addr_state* - The state provided by the borrower in the loan application
- *earliest_cr_line* - The month the borrower's earliest reported credit line was opened
- *initial_list_status* - The initial listing status of the loan. Possible values are - W, F
- *last_pymt_d* - Last month payment was received
- *last_credit_pull_d* - The most recent month Lending Club pulled credit for this loan
- *application_type* - Indicates whether the loan is an individual application or a joint application with two co-borrowers
- *hardship_flag* - Flags whether or not the borrower is on a hardship plan
- *disbursement_method* - The method by which the borrower receives their loan. Possible values are: CASH, DIRECT_PAY
- *debt_settlement_flag* - Flags whether or not the borrower, who has charged-off, is working with a debt-settlement company.
- *loan_status_final* - Loan status of the borrower. Can be Default or Not Default.

Now that we have an idea of what each of these text variables are, let's go through each and see what can be done with it and how it correlates with whether or not someone will default on their loan.

I will start by making a quick summary function for each variable.

```
summary_func = function(variable) {
  levels = unique(lending_data_vars_rem[, variable])
  number_nas = sum(is.na(lending_data_vars_rem[, variable]))
  if (nrow(levels) >= 30) {
    summary_df = data.frame("Number of Levels" = nrow(levels),
                            "Number of NA's" = number_nas
                           )
    kable(summary_df)
  }else{
    summary_df = data.frame("Levels" = levels,
                            "Number of NA's" = number_nas
                           )
    kable(summary_df)
  }
}
```

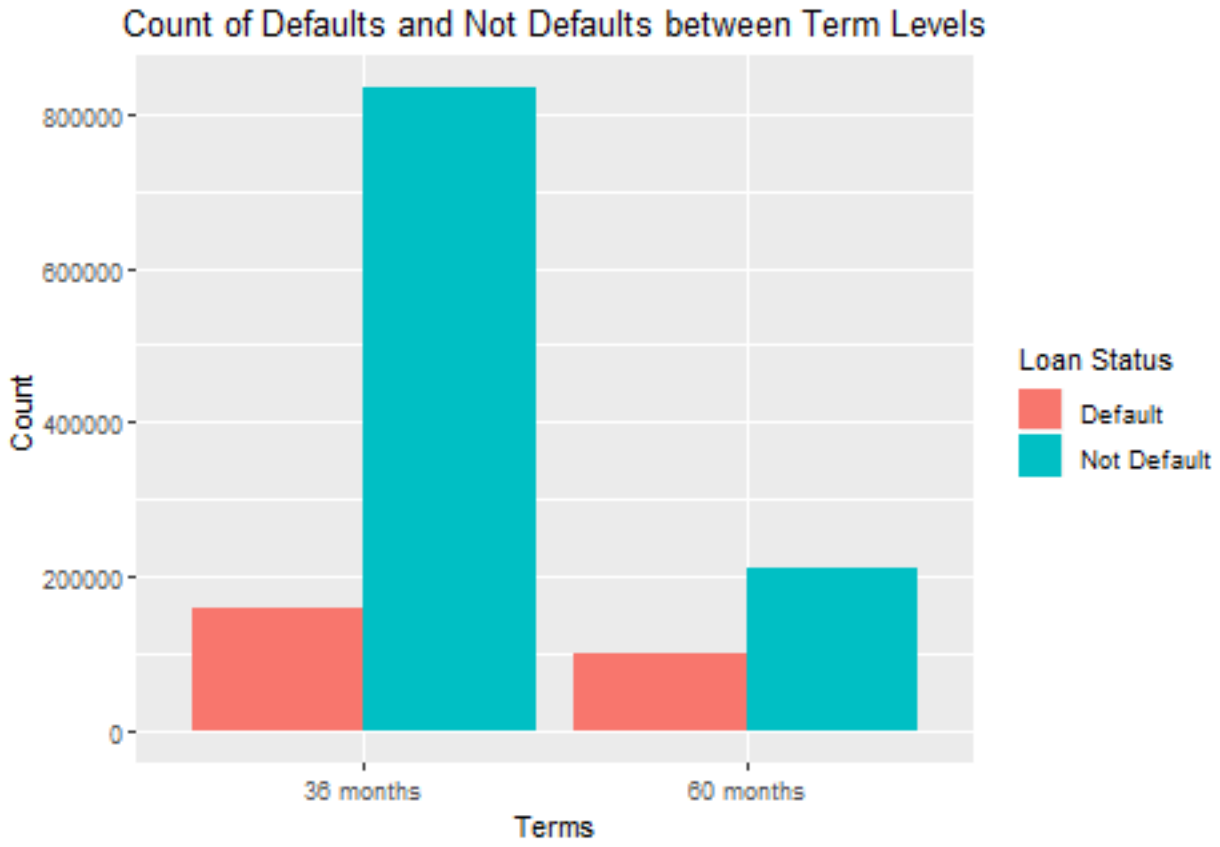
Term, is an assigned variable by Lending Club. Let's use our summary function to get a list of all values this variable can take.

```
summary_func("term")
```

term	Number.of.NA.s
36 months	0
60 months	0

So, we can see that *term* takes on two levels which are `36 months` and `60 months`. Also, *term* contains no missing values. This variable will be useful for predicting whether or not someone will default on their loan as I believe those with more payments (60 months), will be more likely to not fully pay back their loan. Let's take a look at a grouped barplot to see how the Default and Not Default observations differ between each level of *term*.

```
options(scipen = 10)
freq_table = as.data.frame(lending_data_vars_rem %>% count(term, loan_status_final))
ggplot(freq_table, aes(factor(term), n, fill = loan_status_final)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Terms", y = "Count", fill = "Loan Status",
       title = "Count of Defaults and Not Defaults between Term Levels") +
  theme(plot.title = element_text(hjust = 0.5))
```




```
default_rate_36_month = freq_table$n[1]/(sum(lending_data_vars_rem$term == "36 months"))
default_rate_60_month = freq_table$n[3]/(sum(lending_data_vars_rem$term == "60 months"))
def_rate_60to30 = default_rate_60_month / default_rate_36_month
```

So, we can see above that a majority of the loans have a term for 36 months. Between each term, 36 and 60 months, we can see that ther number of defaults is between 150,000 and 175,000. However, since the majority of loans are 36 months we can say that the rate of defaults on 60 month term loans is much higher than the rate of defaults on a 36 month term loan. The default rate of 36 month loans is 0.1612349 and the default rate of 60 month loans is 0.3256254, so the default rate of 60 month loans is 2.0195713 times as much which is much higher. This confirms my belief that those with 60 month terms are more likely to default on their loan.

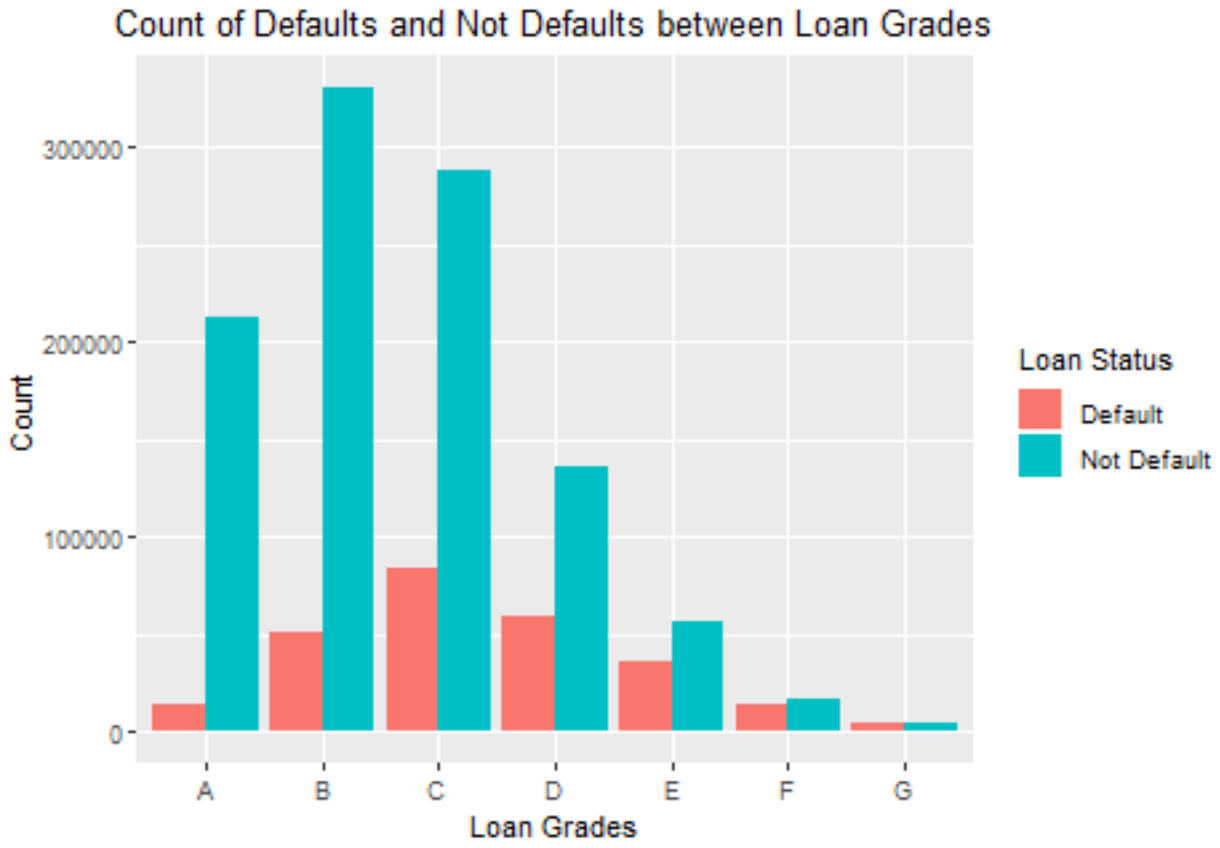
Grade, is an assigned variable by Lending Club. Let's use our summary function to get a list of all values this variable can take.

```
summary_func("grade")
```

grade	Number.of.NA.s
D	0
C	0
A	0
B	0
E	0
G	0
F	0

So, we can see that *grade* takes on 7 levels which are `A, B, C, D, E, F, G`. Also, *grade* contains no missing values. This variable will be useful for predicting whether or not someone will default on their loan as loans with worse grades (G being the worst), the more likely someone will default on their loan. Let's take a look at a grouped barplot to see how the Default and Not Default observations differ between each level of *grade*.

```
options(scipen = 10)
freq_table = as.data.frame(lending_data_vars_rem %>% count(grade, loan_status_final))
ggplot(freq_table, aes(factor(grade), n, fill = loan_status_final)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Loan Grades", y = "Count", fill = "Loan Status",
       title = "Count of Defaults and Not Defaults between Loan Grades") +
  theme(plot.title = element_text(hjust = 0.5))
```



As we can see above, the loan grade with the most amount of loans is B. Looking at the number of defaults between each loan grade, we can see that as the loan grade worsens the default rate increases. When we get to loan grades F and G, the default rate is near 50%. This confirms my belief that as the loan grades worsen, the borrower is less likely to repay their loan and therefore will default on it.

Sub_Grade, is an assigned variable by Lending Club. Let's use our summary function to get a list of all values this variable can take.

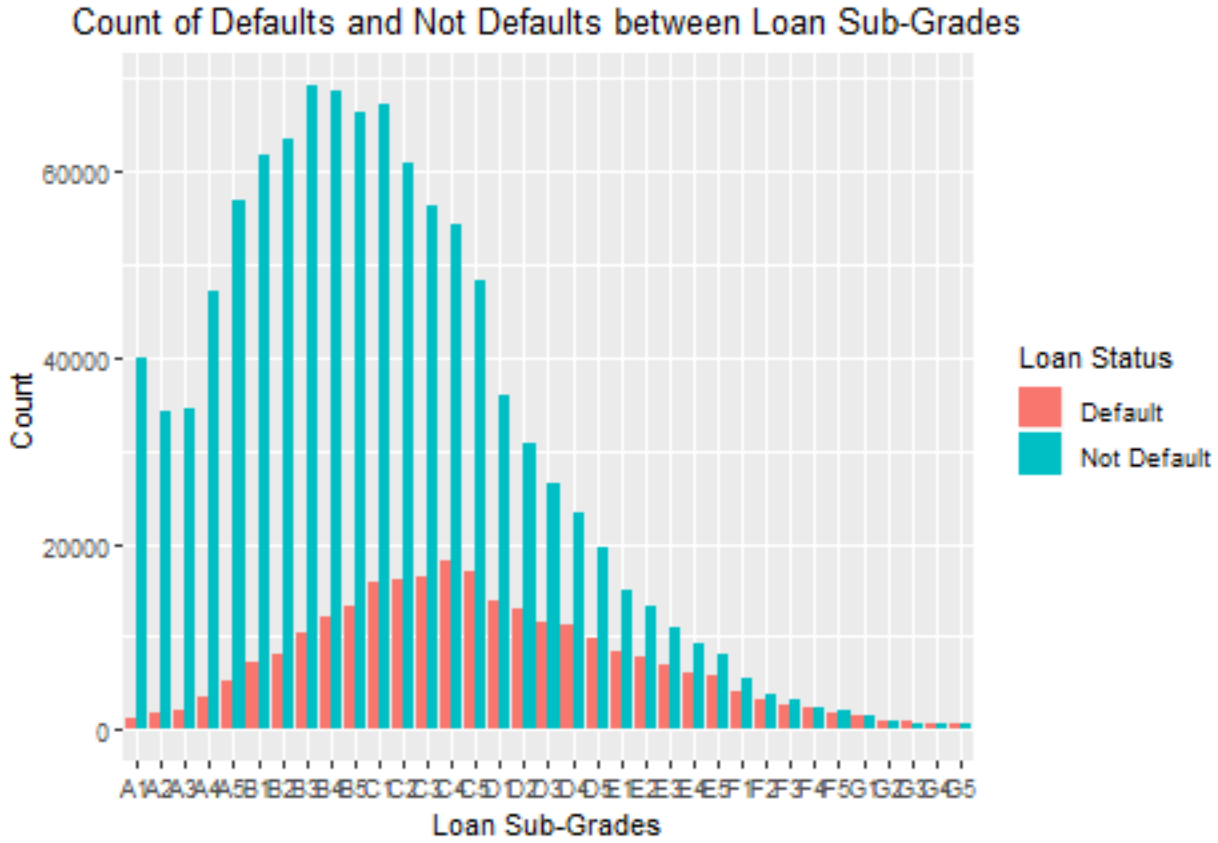
```
summary_func("sub_grade")
```

Number.of.Levels	Number.of.NA.s
35	0

There are 35 levels in the *sub_grade* variable. This is because for each grade loan, there are 5 sub-grades in that loan grade. This variable will be useful for predicting loan default as sub-grades within loan grades also are a good predictor of how risky someone and therefore the chance they may default on their loan. Let's take a look at a grouped barplot to see how the Default and Not Default observations differ between each level of *sub_grade*.

```
options(scipen = 10)
freq_table = as.data.frame(lending_data_vars_rem %>% count(sub_grade, loan_status_final))
ggplot(freq_table, aes(factor(sub_grade), n, fill = loan_status_final)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Loan Sub-Grades", y = "Count", fill = "Loan Status",
```

```
      title = "Count of Defaults and Not Defaults between Loan Sub-Grades") +
theme(plot.title = element_text(hjust = 0.5))
```



Similiarly to the barplot above of *grade*, we can see that borrowers with a loan sub-grade between B1-B5 are the most common. Also similiarly to above, as the loan sub-grade worsens the default rate of loans is near 50%. This confirms my belief that as the loan grade and sub-grade worsens, the more risky a borrower is and therefore the higher chance that they default on their loan.

Emp_Title, is a user-entered variable in this dataset. Let's use our summary function to get a list of all values this variable can take.

```
summary_func("emp_title")
```

Number.of.Levels	Number.of.NA.s
355809	82715

There are 355,809 levels this variable can take. There are also 82,715 missing values of this variable in our dataset. While employment titles are important, I do not believe they will be of much use for our end goal of predicting loan default. Also, being that this variable is entered in by the borrower, we cannot trust that all values are accurate and we are better off removing this variable.

Emp_Length, is a user-entered variable in this dataset. Let's use our summary function to get a list of all values this variable can take.

```
summary_func("emp_length")
```

emp_length	Number.of.NA.s
5 years	0
< 1 year	0
10+ years	0
3 years	0
4 years	0
1 year	0
8 years	0
n/a	0
2 years	0
6 years	0
9 years	0
7 years	0

There are 12 levels this variable can take, being from less than 1 year all the way to 10+ years as well as an 'n/a' variable. While they are not explicitly missing, there are actually 75491 missing values of this variable in our dataset. While employment length is important, I do not believe it will be of much use for our end goal of predicting loan default. Also, being that this variable is entered in by the borrower, we cannot trust that all values are accurate and we are better off removing this variable.

Emp_Length, is a user-entered variable in this dataset. Let's use our summary function to get a list of all values this variable can take.

```
summary_func("home_ownership")
```

home_ownership	Number.of.NA.s
MORTGAGE	0
RENT	0
OWN	0
ANY	0
NONE	0
OTHER	0

There are 6 levels this variable can take, which are `mortgage, rent, own, any, none, and other`. While there are no true NA values, we must take note of the levels `any`, `none`, and `other`. Since these levels are ambiguous, I believe that this variable should be removed to prevent confusion later on. Also, being that this variable is user-entered we cannot trust it's accuracy.

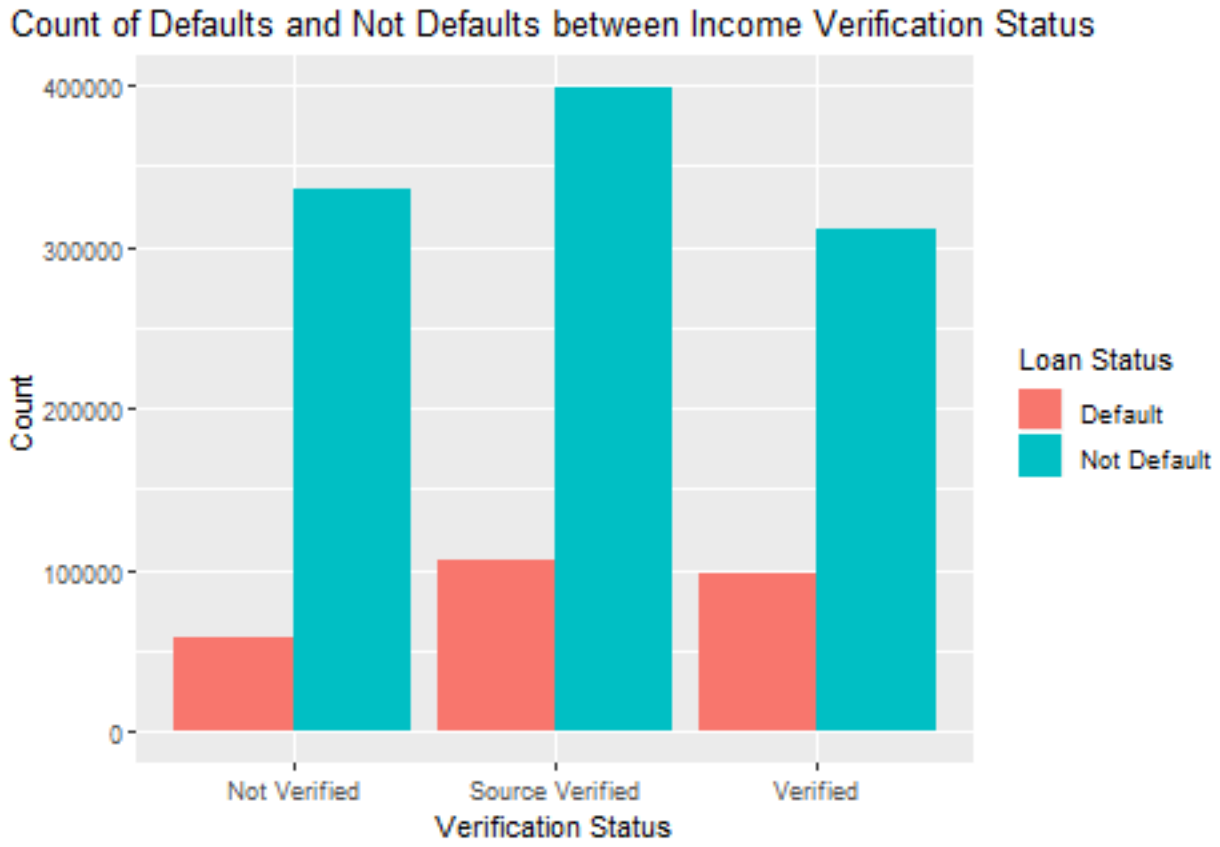
Verification_status, is a Lending Club assigned variable in this dataset. Let's use our summary function to get a list of all values this variable can take.

```
summary_func("verification_status")
```

verification_status	Number.of.NA.s
Source Verified	0
Verified	0
Not Verified	0

There are 3 levels this variable can take, which are `source verified`, `verified`, and `not verified`. Since income is an important factor on whether someone will default on their loan or not, I believe it is important if the borrower has had their income verified. I believe that those who have not had their income verified will be more likely to default on their loan. Let's take a look at a grouped barplot to see how the Default and Not Default observations differ between each level of *verification_status*.

```
options(scipen = 10)
freq_table = as.data.frame(lending_data_vars_rem %>% count(verification_status, loan_status_final))
ggplot(freq_table, aes(factor(verification_status), n, fill = loan_status_final)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Verification Status", y = "Count", fill = "Loan Status",
       title = "Count of Defaults and Not Defaults between Income Verification Status") +
  theme(plot.title = element_text(hjust = 0.5))
```



From the looks of the graph, it seems that those who have had their income verified are actually more likely to default on their loan. I believe this variable will be useful in predicting whether or not someone will default on their loan.

Issue_d, is a Lending Club assigned variable in this dataset. Let's use our summary function to get a list of all values this variable can take.

```
summary_func("issue_d")
```

Number.of.Levels	Number.of.NA.s
139	0

Unsurprisingly, there are a large amount of levels for this variable as it is the issue date of the loan. My belief is that the issue date will have no correlation to whether or not someone will default on their loan as the month a loan was issued should not be a cause of loan default. I will be removing this variable.

Pymnt_plan, is a Lending Club assigned variable in this dataset. Let's use our summary function to get a list of all values this variable can take.

```
summary_func("pymnt_plan")
```

pymnt_plan	Number.of.NA.s
n	0

This variable takes on one level, `n`, which indicates that the borrower is not on a payment plan for the loan. Since this payment plan is for borrowers who are behind on their loan, this variable will not be useful for us as we are only analyzing loans that are defaulted on, charged off, or fully paid. I will be removing this variable.

Purpose, is a user-entered variable in this dataset. Let's use our summary function to get a list of all values this variable can take.

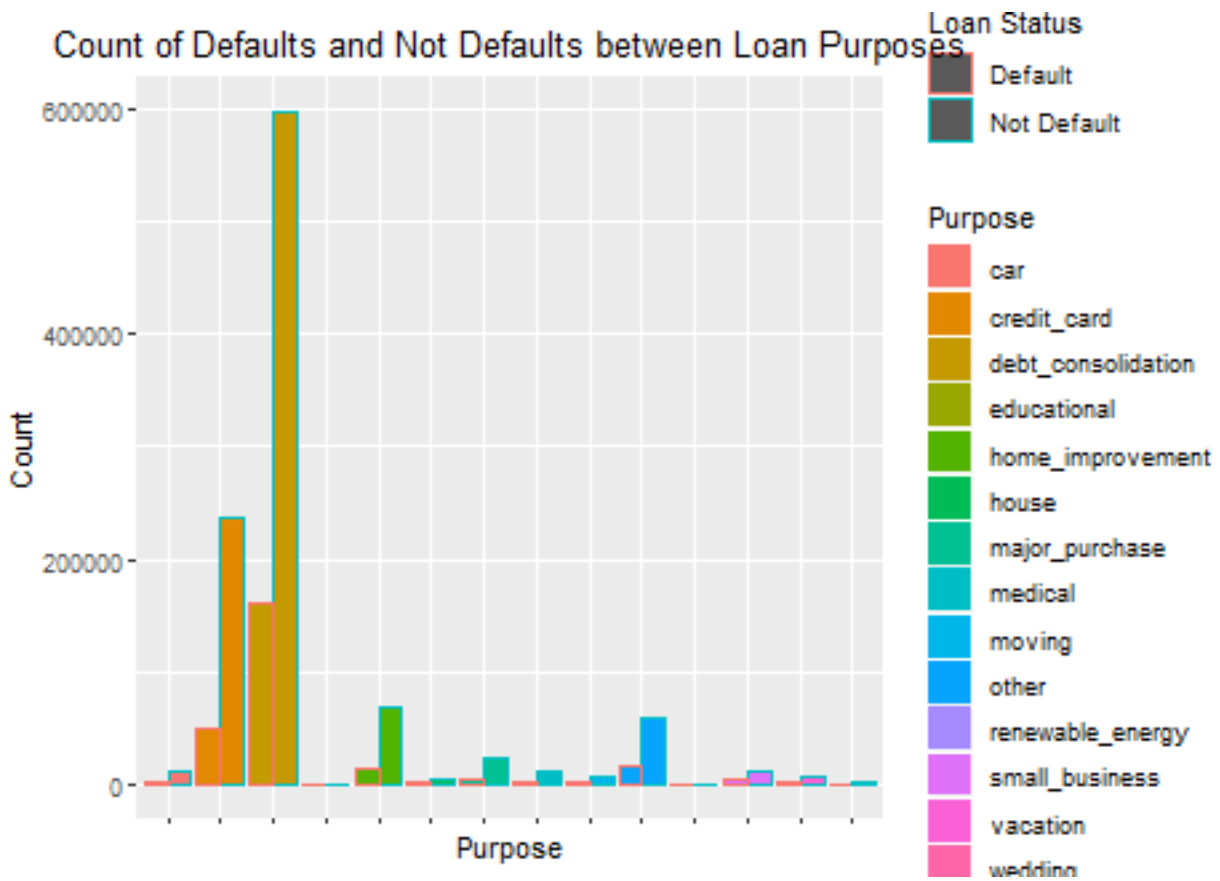
```
summary_func("purpose")
```

purpose	Number.of.NA.s
debt_consolidation	0
credit_card	0
other	0
house	0
car	0
home_improvement	0
moving	0
small_business	0
vacation	0
medical	0
major_purchase	0
renewable_energy	0
wedding	0
educational	0

This variable takes on 14 levels, which cover different areas of purposes for their loan. I believe this variable will indeed play a role in predicting loan default as those who are taking out loans to consolidate debt or pay off credit cards may be at more risk of default. Let's look at a grouped bar plot to see the default rate between levels of this variable.

```
options(scipen = 10)
freq_table = as.data.frame(lending_data_vars_rem %>% count(purpose, loan_status_final))
ggplot(freq_table, aes(factor(purpose), n, fill = factor(purpose), color = loan_status_final)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Purpose", y = "Count", fill = "Purpose",
       title = "Count of Defaults and Not Defaults between Loan Purposes",
       color = "Loan Status") +
```

```
  theme(axis.text.x = element_blank(), plot.title = element_text(hjust = 0.5))
```



We can see that the most popular purposes for a borrowers loan are credit card debt and debt consolidation. Both of these purposes have a large amount of defaults compared to the other purposes. Overall, I believe that purposes will play a role in predicting loan default.

Title, is a user-entered variable in this dataset. Let's use our summary function to get a list of all values this variable can take.

```
summary_func("title")
```

Number.of.Levels	Number.of.NA.s
61453	15425

There are 61,453 levels in this variable and 15,425 NA values. Since this variable is user-entered and is just the "title" of their loan, I do not believe it will play an important role in predicting loan default so I will be removing this variable.

Zip_code, is a user-entered variable in this dataset. Let's use our summary function to get a list of all values this variable can take.

```
summary_func("zip_code")
```

Number.of.Levels	Number.of.NA.s
946	1

While I do believe location will play a role in predicting loan default, I do not believe zip codes will be as important as states. With 946 levels of zip codes, I believe we should focus on states instead. I will be removing this variable.

Addr_state, is a user-entered variable in this dataset. Let's use our summary function to get a list of all values this variable can take.

```
summary_func("addr_state")
```

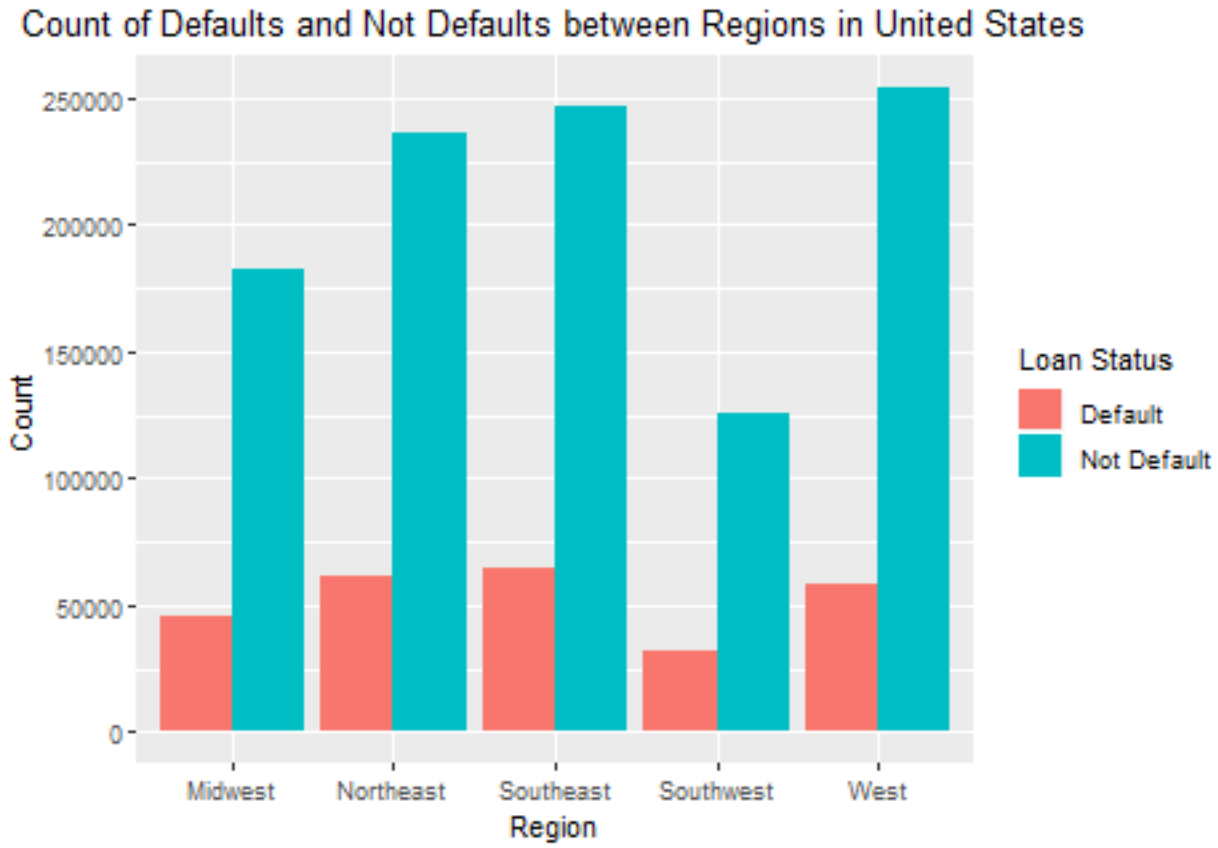
Number.of.Levels	Number.of.NA.s
51	0

There are 51 levels in this variable, which makes sense since there are 50 states as well as Washington DC. There are no NA values. For this variable, I will be creating a new variable that corresponds to the region of the United States instead of the actual state. Let's begin this process.

```
west_region = c("WA", "OR", "CA", "NV", "UT", "HI", "AK", "CO",
                "WY", "MT", "ID")
southwest_region = c("AZ", "NM", "TX", "OK")
midwest_region = c("ND", "SD", "NE", "KS", "MN", "IA", "MO",
                   "WI", "IL", "IN", "MI", "OH")
southeast_region = c("AR", "LA", "MS", "AL", "TN", "FL", "GA",
                     "SC", "NC", "VA", "DC", "KY", "WV")
northeast_region = c("MD", "PA", "NY", "DE", "NJ", "CT", "RI",
                     "MA", "NH", "VT", "DC", "ME")

lending_data_vars_rem$state_regions = ifelse(
  lending_data_vars_rem$addr_state %in% west_region, "West",
  ifelse(lending_data_vars_rem$addr_state %in% southwest_region, "Southwest",
         ifelse(lending_data_vars_rem$addr_state %in% midwest_region, "Midwest",
                ifelse(lending_data_vars_rem$addr_state %in% southeast_region, "Southeast", "Northeast")
)
```

With the new variable created, let's remove the old variable *addr_state* and then take a look at the grouped bar plot to see default rates between regions.

```
lending_data_vars_rem = lending_data_vars_rem[, !(names(lending_data_vars_rem) %in% c("addr_state"))]

options(scipen = 10)
freq_table = as.data.frame(lending_data_vars_rem %>% count(state_regions, loan_status_final))
ggplot(freq_table, aes(factor(state_regions), n, fill = loan_status_final)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Region", y = "Count", fill = "Loan Status",
       title = "Count of Defaults and Not Defaults between Regions in United States") +
  theme(plot.title = element_text(hjust = 0.5))
```



Looking at the plot above, all regions have a similiar number of defaults. The region that seems to have the largest default rate is the Southwest, which includes states such as Texas. Overall, there does not seem to be much of a difference between default rates of regions but I believe that location will still play a role in predicting loan defaults so I will be keeping this variable.

Earliest_cr_line, is a Lending Club assigned variable in this dataset. Let's use our summary function to get a list of all values this variable can take.

```
summary_func("earliest_cr_line")
```

Number.of.Levels	Number.of.NA.s
739	29

Since this variable represents the month that the earliest credit line was opened for the borrower, we can therefore figure out how old their earliest credit line is. Since this is important for assessing risk of a borrower, I believe it will play a role in predicting loan default. I will be converting this variable to a numeric one which gives us the number of days old the borrowers credit line is. To avoid potentional modeling discrepancies later, I will be measuring the age from the date of December 31st, 2015 since that is when this data is through. I will then drop the original *earliest_cr_line* variable.

```
lending_data_vars_rem$earliest_cr_line_fixed = as.Date(paste("01", lending_data_vars_rem$earliest_cr_li
lending_data_vars_rem = select(lending_data_vars_rem, -c(earliest_cr_line)) #drop variable
end_of_data_date = as.Date("2015-12-31", format = "%Y-%m-%d")
```

```
lending_data_vars_rem$credit_age = end_of_data_date - lending_data_vars_rem$earliest_cr_line_fixed
```

We now have the variable *credit_age* which gives us the length of their earliest credit line in days. With this new variable created, we need to now focus on the NA values we have. I will be replacing all NA values with the mean of the *credit_age* variable, and then finally converting this variable to a numeric type and removing our original variabe *earliest_cr_line*.

```
credit_age_mean = mean(lending_data_vars_rem$credit_age, na.rm = TRUE)

lending_data_vars_rem$credit_age = na.aggregate(lending_data_vars_rem$credit_age)
lending_data_vars_rem$credit_age = as.numeric(lending_data_vars_rem$credit_age)
lending_data_vars_rem = lending_data_vars_rem[, !(names(lending_data_vars_rem) %in% c("earliest_cr_line
```

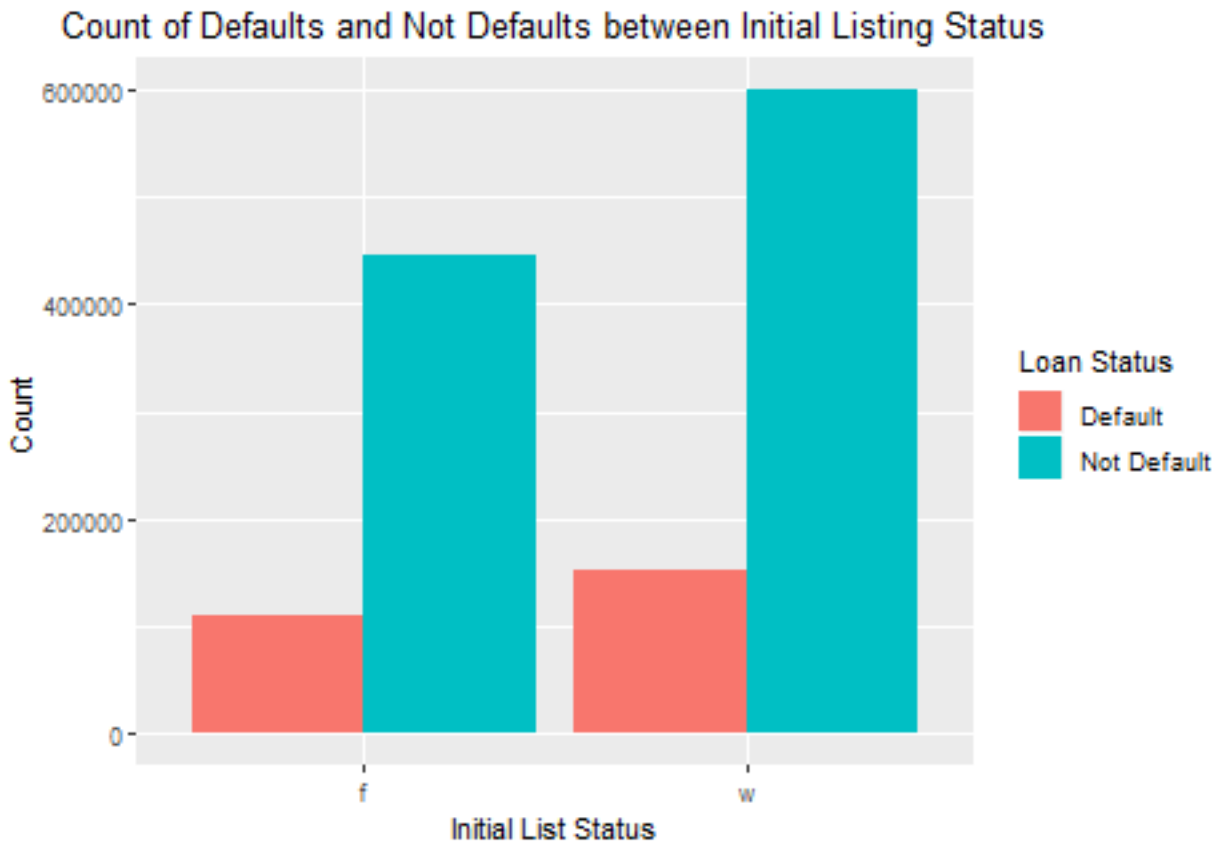
Initial_list_status is a Lending Club assigned variable. Let's take a look at the summary function output to see the levels of this variable.

```
summary_func("initial_list_status")
```

initial_list_status	Number.of.NA.s
w	0
f	0

The two levels for *initial_list_status* are `w`, and `f`. These stand for `whole`, and `fractional`. Whole loans are loans that can be purchased by investors in their entirety. Fractional loans are loans that are able to be purchased by investors in fractions. Let's look at the grouped bar plot for this variable.

```
options(scipen = 10)
freq_table = as.data.frame(lending_data_vars_rem %>% count(initial_list_status, loan_status_final))
ggplot(freq_table, aes(factor(initial_list_status), n, fill = loan_status_final)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Initial List Status", y = "Count", fill = "Loan Status",
       title = "Count of Defaults and Not Defaults between Initial Listing Status") +
  theme(plot.title = element_text(hjust = 0.5))
```



Looking at the bar plot, there is not much of a difference in default rates between groups. Due to this variable being meant for investors to analyze a loan, I do not believe this variable will be of relevance to our goal of predicting loan default. I will be removing this variable.

Last_pymnt_d is a Lending Club assigned variable. Let's take a look at the summary function output to see the levels of this variable.

```
summary_func("last_pymnt_d")
```

Number.of.Levels	Number.of.NA.s
136	2273

There are 136 levels in this variable and 2273 NA values. Since this variable is associated with being late on a loan, I do not believe it will be useful for predicting a default on a loan. I will be removing this variable.

Last_credit_pull_d is a Lending Club assigned variable. Since this variable is associated with the last time Lending Club pulled the credit on a borrower, I do not believe it will be of much use for predicting loan default. I will be removing this variable.

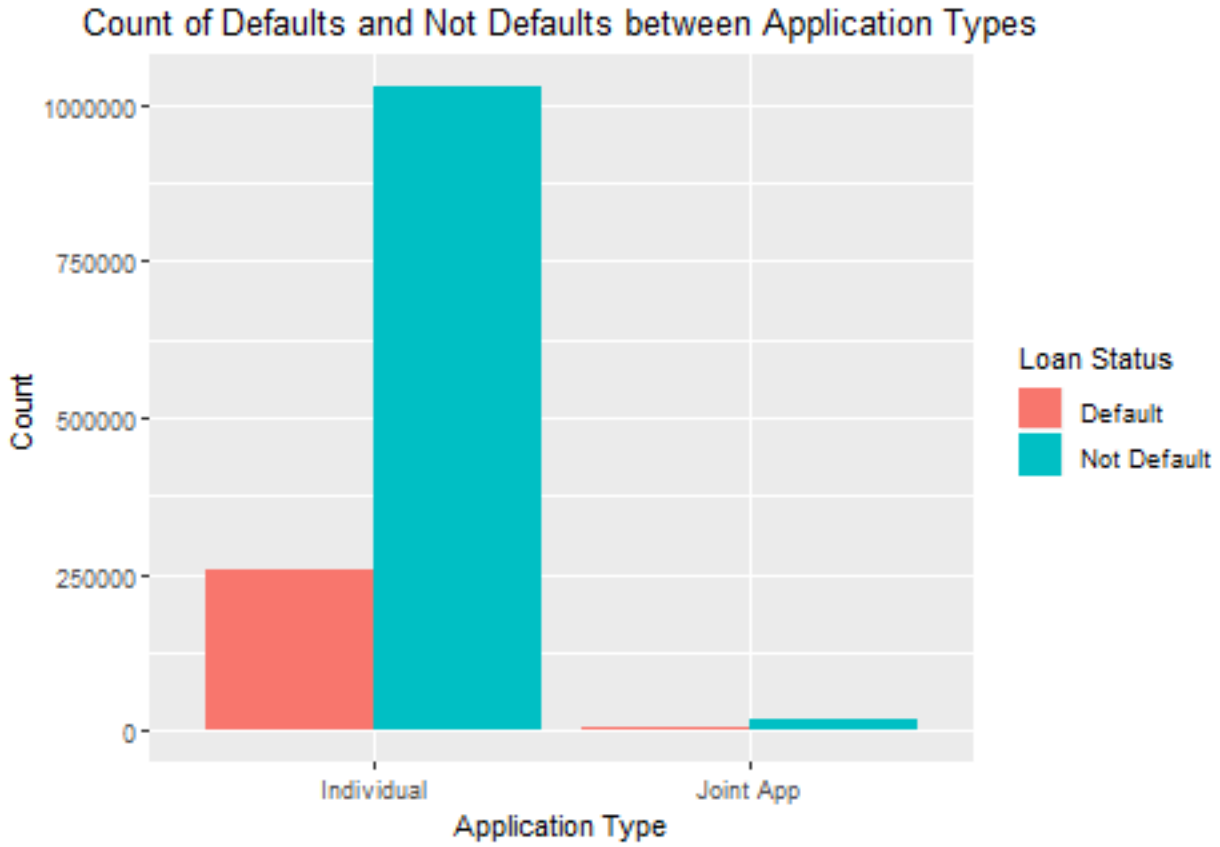
Application_type is a Lending Club assigned variable. Let's take a look at the summary function output to see the levels of this variable.

```
summary_func("application_type")
```

application_type	Number.of.NA.s
Joint App	0
Individual	0

There are two levels in this variable, `Joint App` or `Individual`, where Joint App is a joint application with two co-borrowers and an individual application is with one borrower. Let's take a look at the grouped bar plot.

```
options(scipen = 10)
freq_table = as.data.frame(lending_data_vars_rem %>% count(application_type, loan_status_final))
ggplot(freq_table, aes(factor(application_type), n, fill = loan_status_final)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Application Type", y = "Count", fill = "Loan Status",
       title = "Count of Defaults and Not Defaults between Application Types") +
  theme(plot.title = element_text(hjust = 0.5))
```



Looking at the plot above, the application type observations are very imbalanced. Joint application loans are very uncommon in our data. The default rates are very similiar between groups, with individual having a default rate of .25 and joint app having a default rate of .33. Since these are very similiar, with account of imbalanced observations between groups, I will be removing this variable.

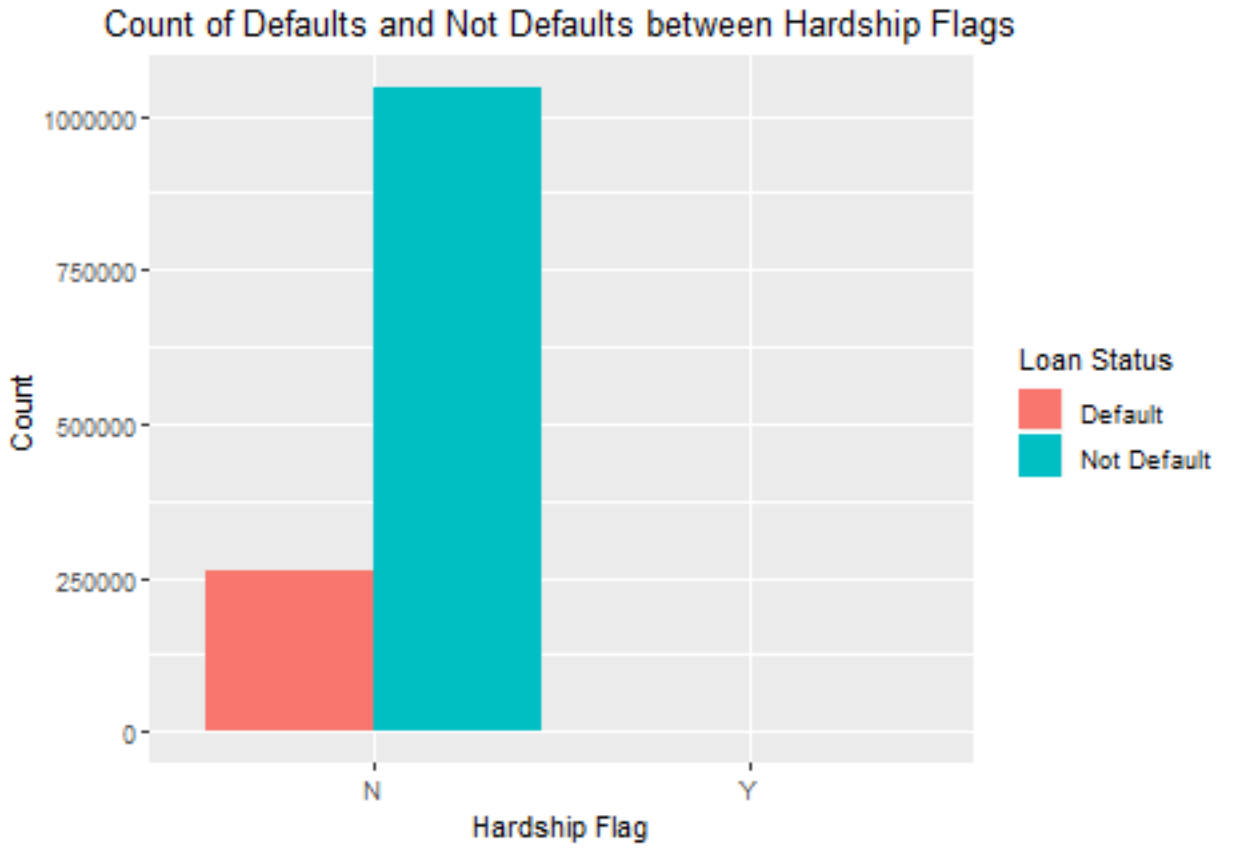
Hardship_flag is a Lending Club assigned variable. Let's take a look at the summary function output to see the levels of this variable.

```
summary_func("hardship_flag")
```

hardship_flag	Number.of.NA.s
N	0
Y	0

There are two levels in this variable, `Y` or `N`, indicating yes or no. A hardship flag indicates that the borrower is in a hardship program, which is mostly used for catching up on debt by waived interest fees. Let's look at a grouped bar plot.

```
options(scipen = 10)
freq_table = as.data.frame(lending_data_vars_rem %>% count(hardship_flag, loan_status_final))
ggplot(freq_table, aes(factor(hardship_flag), n, fill = loan_status_final)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Hardship Flag", y = "Count", fill = "Loan Status",
       title = "Count of Defaults and Not Defaults between Hardship Flags") +
  theme(plot.title = element_text(hjust = 0.5))
```



Once again, we can see right away that the variable is very imbalanced. Digging deeper into the graph, there is actually only 1 observation for the `Y` group. With this huge imbalance, this variable will not be helpful and I will be removing it.

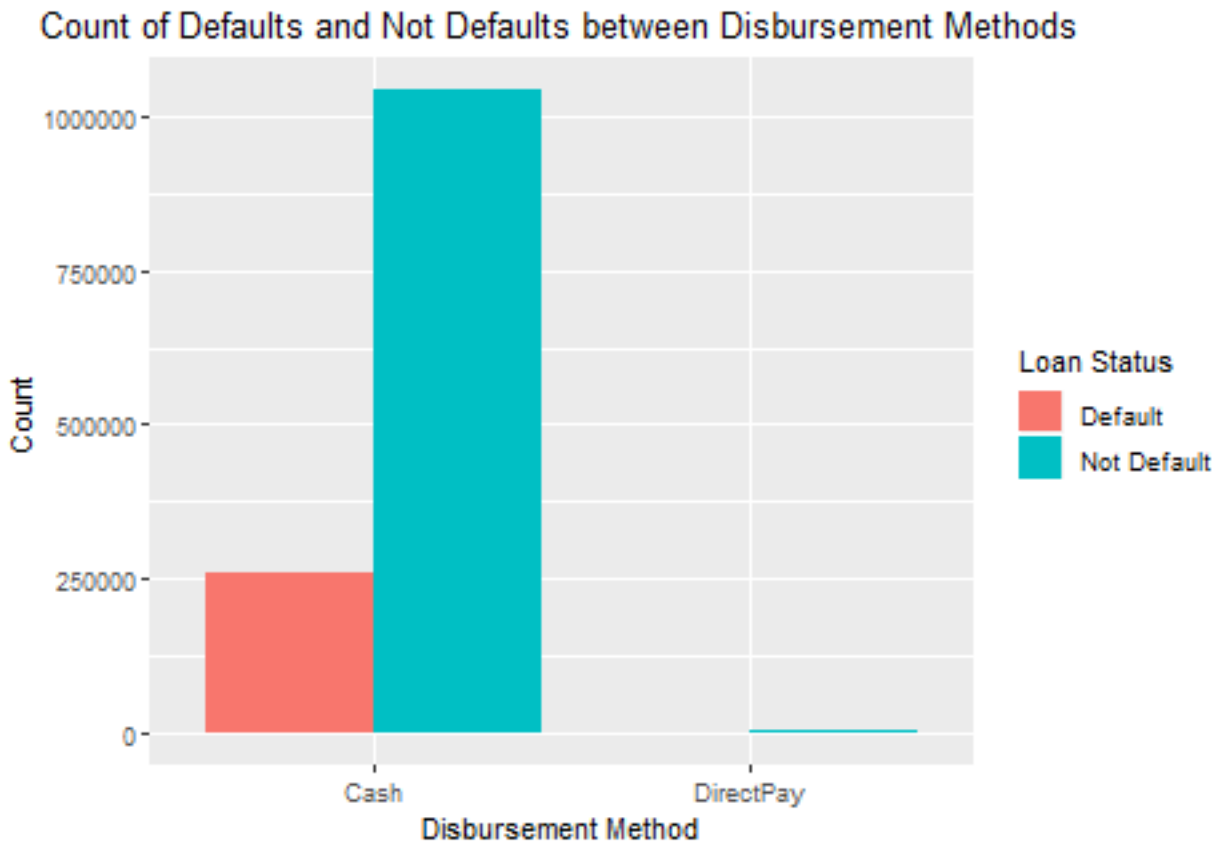
Disbursement_method is a Lending Club assigned variable. Let's take a look at the summary function output to see the levels of this variable.

```
summary_func("disbursement_method")
```

disbursement_method	Number.of.NA.s
Cash	0
DirectPay	0

There are two levels in this variable, `Cash` or `DirectPay`. The cash level means the borrower received the money from their loan. The direct pay level means that borrowers are using up to 80% of their loan for debt right away. Let's take a look at a grouped bar plot.

```
options(scipen = 10)
freq_table = as.data.frame(lending_data_vars_rem %>% count(disbursement_method, loan_status_final))
ggplot(freq_table, aes(factor(disbursement_method), n, fill = loan_status_final)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Disbursement Method", y = "Count", fill = "Loan Status",
       title = "Count of Defaults and Not Defaults between Disbursement Methods") +
  theme(plot.title = element_text(hjust = 0.5))
```



Once again, we see that this variable is affected by highly imbalanced observations. Only 5813 observations have received a `DirectPay` loan. With this much of an imbalance, I will be removing this variable.

Debt_settlement_flag is a Lending Club assigned variable. Let's take a look at the levels with the summary function.

```
summary_func("debt_settlement_flag")
```

debt_settlement_flag	Number.of.NA.s
N	0
Y	0

This variable indicates whether or not the borrower, who has already defaulted on their loan, is working with a debt-settlement company. Therefore, it is safe to say that this variable will not be of use as the observations that are in Debt Settlement have already defaulted.

That is the end of the categorical variable review. For each variable, we have taken a look at the levels and total NA values and justified whether each variable was useful or not through visualizations. We have created new variables, as well as removed some. Let's go ahead and remove those that need to be removed now.

```
remove_list = c("emp_title",
                "emp_length",
                "home_ownership",
                "issue_d",
                "pymnt_plan",
                "title",
                "zip_code",
                "initial_list_status",
                "last_credit_pull_d",
                "application_type",
                "hardship_flag",
                "disbursement_method",
                "debt_settlement_flag",
                "last_pymnt_d")

lending_data_vars_rem = lending_data_vars_rem[, !(names(lending_data_vars_rem) %in% remove_list)]
total_char_vars_final = length(lending_data_vars_rem %>% select_if(is.character))
```

Overall, we have reduced our number of categorical variables from 22 to 7.

Numerical Variables

In this section, we will be taking a look at all variables that are of type numeric. Let's get a quick look at a list of all these variables.

```
total_num_vars = length(lending_data_vars_rem %>% select_if(is.numeric))
```

There are a total of 66 numeric variables in our data. Let's take a look at each of these variables.

```
numeric_vars = lending_data_vars_rem %>% select_if(is.numeric)
colnames(lending_data_vars_rem %>% select_if(is.numeric))
```

```
##  [1] "loan_amnt"                 "funded_amnt"
##  [3] "funded_amnt_inv"           "int_rate"
##  [5] "installment"               "annual_inc"
##  [7] "dti"                       "delinq_2yrs"
```

```
##  [9] "inq_last_6mths"             "open_acc"
## [11] "pub_rec"                    "revol_bal"
## [13] "revol_util"                 "total_acc"
## [15] "out_prncp"                  "out_prncp_inv"
## [17] "total_pymnt"                "total_pymnt_inv"
## [19] "total_rec_prncp"            "total_rec_int"
## [21] "total_rec_late_fee"         "recoveries"
## [23] "collection_recovery_fee"    "last_pymnt_amnt"
## [25] "collections_12_mths_ex_med" "policy_code"
## [27] "acc_now_delinq"             "tot_coll_amt"
## [29] "tot_cur_bal"                "total_rev_hi_lim"
## [31] "acc_open_past_24mths"       "avg_cur_bal"
## [33] "bc_open_to_buy"             "bc_util"
## [35] "chargeoff_within_12_mths"   "delinq_amnt"
## [37] "mo_sin_old_il_acct"         "mo_sin_old_rev_tl_op"
## [39] "mo_sin_rcnt_rev_tl_op"      "mo_sin_rcnt_tl"
## [41] "mort_acc"                   "mths_since_recent_bc"
## [43] "mths_since_recent_inq"      "num_accts_ever_120_pd"
## [45] "num_actv_bc_tl"             "num_actv_rev_tl"
## [47] "num_bc_sats"                "num_bc_tl"
## [49] "num_il_tl"                  "num_op_rev_tl"
## [51] "num_rev_accts"              "num_rev_tl_bal_gt_0"
## [53] "num_sats"                   "num_tl_120dpd_2m"
## [55] "num_tl_30dpd"               "num_tl_90g_dpd_24m"
## [57] "num_tl_op_past_12m"         "pct_tl_nvr_dlq"
## [59] "percent_bc_gt_75"           "pub_rec_bankruptcies"
## [61] "tax_liens"                  "tot_hi_cred_lim"
## [63] "total_bal_ex_mort"          "total_bc_limit"
## [65] "total_il_high_credit_limit" "credit_age"
```

For numeric variables, we are only going to be worrying about columns with NA values. Let's take care of missing values first. After that, we will be normalizing the data to prevent drastic effects from outliers in the data.

We will first start with variables with over 100,000 missing values. Doing mass imputation on these variables can sway our results, so I believe that keeping these variables will hurt our model accuracy more than help it.

```
num_over_100k_missing = colnames(numeric_vars)[colSums(is.na(numeric_vars)) > 100000]
num_over_100k_missing
```

```
## [1] "mo_sin_old_il_acct"    "mths_since_recent_inq" "num_tl_120dpd_2m"
```

The variables with over 100,000 missing values are `mo_sin_old_il_acct`, `mths_since_recent_inq`, and `num_tl_120dpd_2m`. These variables represents the number of months since oldest bank installment account opened, number of months since most recent inquiry, and the number of accounts currently 120 days past due. These variables do not seem of much relevance to our end goal as well, so let's remove them.

```
lending_data_vars_rem = lending_data_vars_rem[, !(names(lending_data_vars_rem) %in%
                                                    num_over_100k_missing)]
```

Now, let's look at variables with over 50,000 missing values. While doing imputation on this number of missing values isn't as bad, it can still sway our model and cause more harm than good. Let's go through them and see which ones should definitely be removed.

```
numeric_vars = lending_data_vars_rem %>% select_if(is.numeric)
num_over_50k_missing = colnames(numeric_vars)[colSums(is.na(numeric_vars)) > 50000]
num_over_50k_missing
```

```
##  [1] "tot_coll_amt"               "tot_cur_bal"
##  [3] "total_rev_hi_lim"           "acc_open_past_24mths"
##  [5] "avg_cur_bal"                "bc_open_to_buy"
##  [7] "bc_util"                    "mo_sin_old_rev_tl_op"
##  [9] "mo_sin_rcnt_rev_tl_op"      "mo_sin_rcnt_tl"
## [11] "mort_acc"                   "mths_since_recent_bc"
## [13] "num_accts_ever_120_pd"      "num_actv_bc_tl"
## [15] "num_actv_rev_tl"            "num_bc_sats"
## [17] "num_bc_tl"                  "num_il_tl"
## [19] "num_op_rev_tl"              "num_rev_accts"
## [21] "num_rev_tl_bal_gt_0"        "num_sats"
## [23] "num_tl_30dpd"               "num_tl_90g_dpd_24m"
## [25] "num_tl_op_past_12m"         "pct_tl_nvr_dlq"
## [27] "percent_bc_gt_75"           "tot_hi_cred_lim"
## [29] "total_bal_ex_mort"          "total_bc_limit"
## [31] "total_il_high_credit_limit"
```

There are quite a few of these variables. Let's quick a get summary of each and see if they are worth keeping.

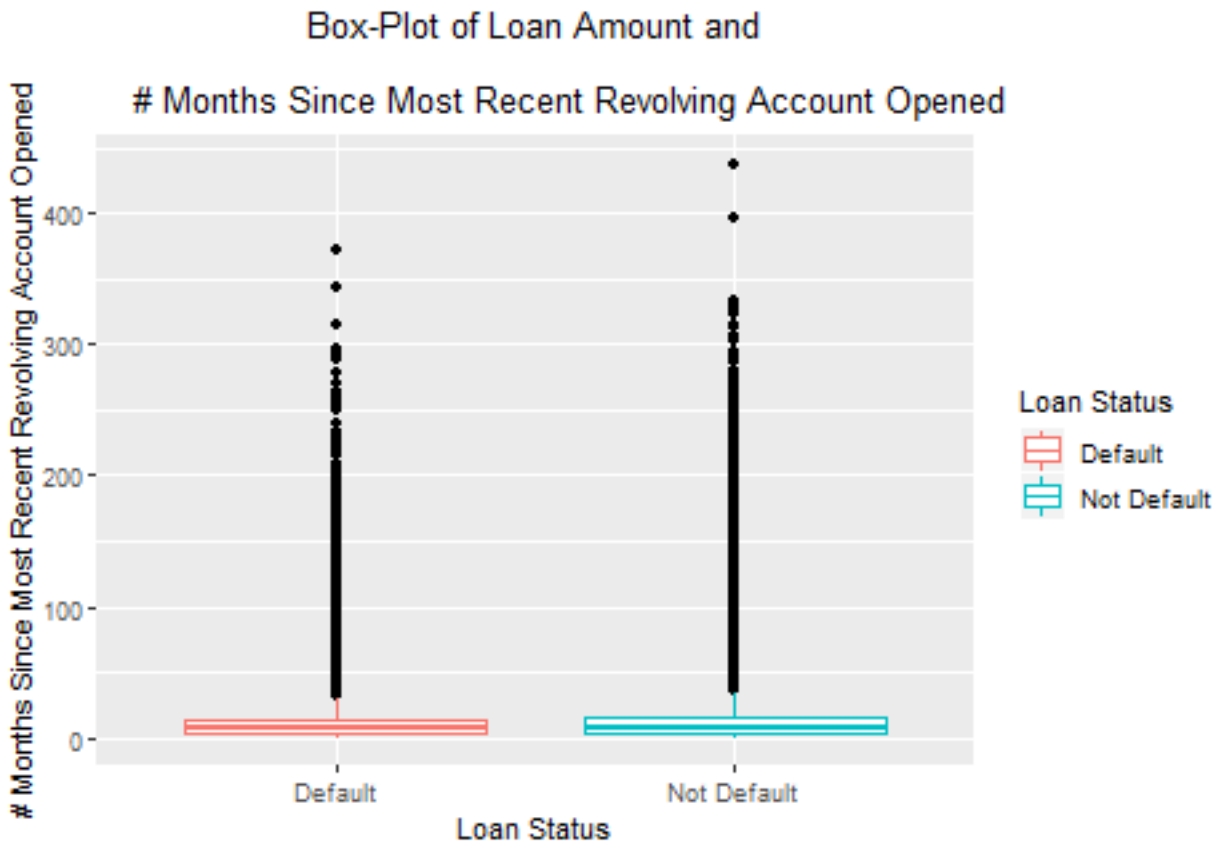
- "tot_coll_amt" - Total collection amounts ever owed
- "tot_cur_bal" - Total current balance of all accounts
- "total_rev_hi_lim" - Total revolving high credit/credit limit
- "acc_open_past_24mths" - Number of trades opened in past 24 months.
- "avg_cur_bal" - Average current balance of all accounts
- "bc_open_to_buy" - Total open to buy on revolving bankcards.
- "bc_util" - Ratio of total current balance to high credit/credit limit for all bankcard accounts.
- "mo_sin_old_rev_tl_op" - Months since oldest revolving account opened
- "mo_sin_rcnt_rev_tl_op" - Months since most recent revolving account opened
- "mo_sin_rcnt_tl" - Months since most recent account opened
- "mort_acc" - Number of mortgage accounts.
- "mths_since_recent_bc" - Months since most recent bankcard account opened.
- "num_accts_ever_120_pd" - Number of accounts ever 120 or more days past due
- "num_actv_bc_tl" - Number of currently active bankcard accounts
- "num_actv_rev_tl" - Number of currently active revolving trades
- "num_bc_sats" - Number of satisfactory bankcard accounts
- "num_bc_tl" - Number of bankcard accounts
- "num_il_tl" - Number of installment accounts
- "num_op_rev_tl" - Number of open revolving accounts
- "num_rev_accts" - Number of revolving accounts
- "num_rev_tl_bal_gt_0" - Number of revolving trades with balance >0
- "num_sats" - Number of satisfactory accounts
- "num_tl_30dpd" - Number of accounts currently 30 days past due (updated in past 2 months) - These borrowers are late on their loans, so not relevant.
- "num_tl_90g_dpd_24m" - Number of accounts 90 or more days past due in last 24 months - These borrowers are late on their loans, so not relevant.
- "num_tl_op_past_12m" - Number of accounts opened in past 12 months
- "pct_tl_nvr_dlq" - Percent of trades never delinquent
- "percent_bc_gt_75" - Percentage of all bankcard accounts > 75% of limit.
- "tot_hi_cred_lim" - Total high credit/credit limit

- "total_bal_ex_mort" - Total credit balance excluding mortgage
- "total_bc_limit" - Total bankcard high credit/credit limit
- "total_il_high_credit_limit" - Total installment high credit/credit limit

Looking at these variable descriptions, a lot of these variables are very similar. Let's take a look at the groups of variables that are similar to one another. We will first start with variables *mo_sin_rcnt_rev_tl_op*, *mo_sin_rcnt_rev_tl_op*, and *mths_since_recent_bc*. These three variables are all related to the number of months since most recent account opened, type of account varying. Let's take a look at their box plot to see if their distributions are similar between default groups.

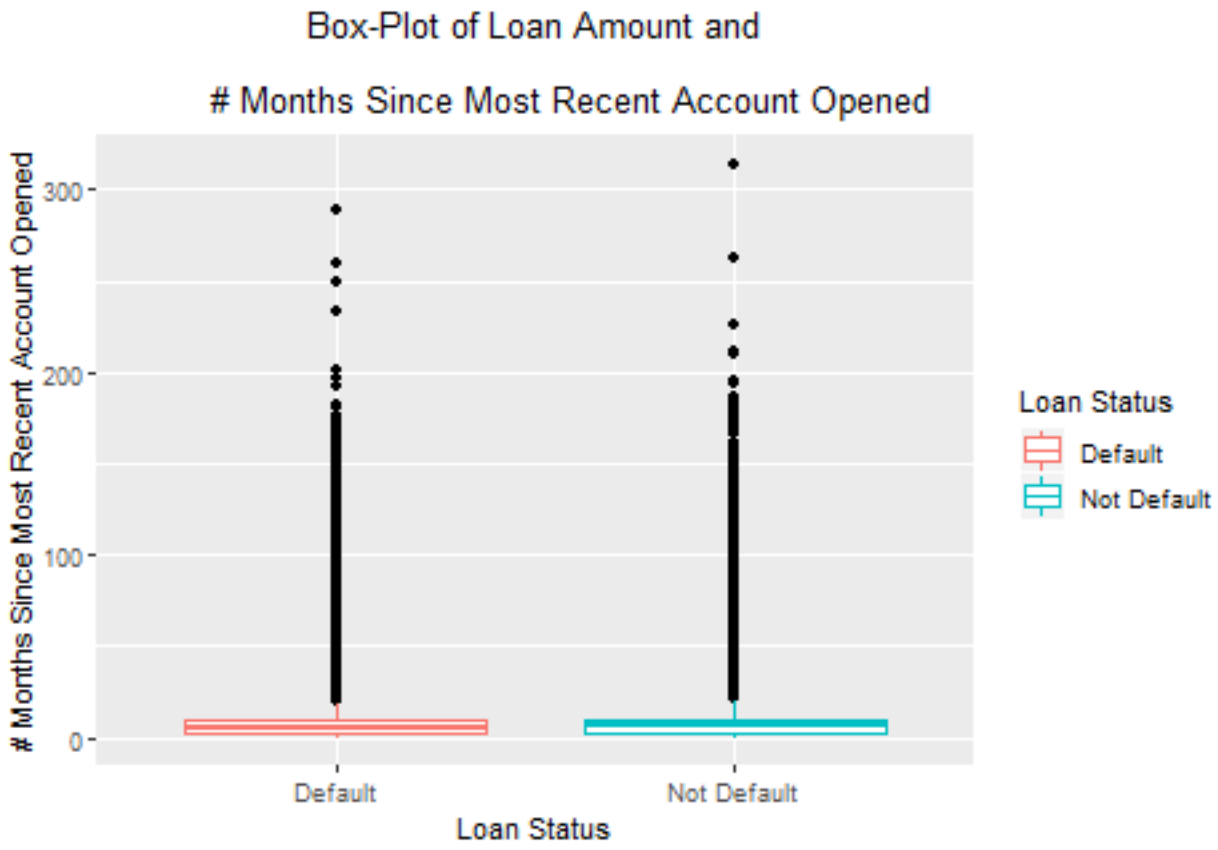
```
plot_mo_revolving = ggplot(lending_data_vars_rem, aes(x = loan_status_final, y = mo_sin_rcnt_rev_tl_op,
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "# Months Since Most Recent Revolving Account Opened",
       title = "Box-Plot of Loan Amount and\n
       # Months Since Most Recent Revolving Account Opened",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))

plot_mo_recent = ggplot(lending_data_vars_rem, aes(x = loan_status_final, y = mo_sin_rcnt_tl, color = l
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "# Months Since Most Recent Account Opened",
       title = "Box-Plot of Loan Amount and\n
       # Months Since Most Recent Account Opened",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
means = aggregate(mths_since_recent_bc ~ loan_status_final, lending_data_vars_rem, mean)
plot_mo_bc = ggplot(lending_data_vars_rem, aes(x = loan_status_final, y = mths_since_recent_bc, color =
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "# Months Since Most Recent Bankcard Opened",
       title = "Box-Plot of Loan Amount and\n
       # Months Since Most Recent Bankcard Opened",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))

plot_mo_revolving
```

```
## Warning: Removed 70277 rows containing non-finite values (stat_boxplot).
```



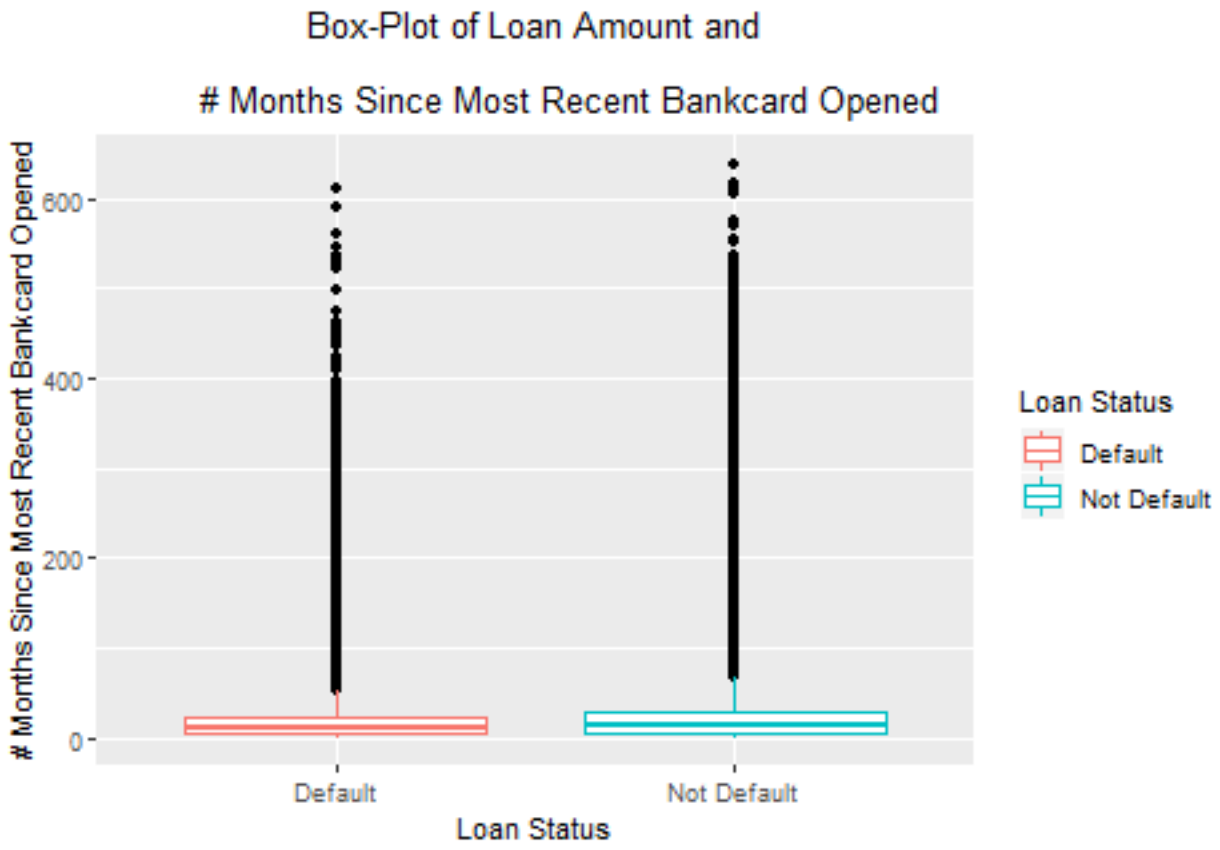

```
plot_mo_recent
```

```
## Warning: Removed 70276 rows containing non-finite values (stat_boxplot).
```




```
plot_mo_bc
```

```
## Warning: Removed 62491 rows containing non-finite values (stat_boxplot).
```



Looking at the box plots for all three of these variables, we can see right away that all their distributions are very similar. We can also see that for all three varaibles, the mean and spread between the `Default` and `Not Default` groups are almost identical. This leads me to believe that these variables will not be very useful. Just to be sure, let's take a quick look at summary statistics before removing the variables.

```
means1 = aggregate(mths_since_recent_bc ~ loan_status_final, lending_data_vars_rem, mean)
means2 = aggregate(mo_sin_rcnt_tl ~ loan_status_final, lending_data_vars_rem, mean)
means3 = aggregate(mo_sin_rcnt_rev_tl_op ~ loan_status_final, lending_data_vars_rem, mean)
means_mo_rec = cbind(means1, means2[2], means3[2])
kable(means_mo_rec, col.names = c("Loan Status",
                                  "Mean Recent BC",
                                  "Mean Recent Acc",
                                  "Mean Recent Revolving Acc"))
```

Loan Status	Mean Recent BC	Mean Recent Acc	Mean Recent Revolving Acc
Default	20.64556	6.882228	11.33643
Not Default	24.58948	8.086805	13.53321

```
sd1 = aggregate(mths_since_recent_bc ~ loan_status_final, lending_data_vars_rem, sd)
sd2 = aggregate(mo_sin_rcnt_tl ~ loan_status_final, lending_data_vars_rem, sd)
sd3 = aggregate(mo_sin_rcnt_rev_tl_op ~ loan_status_final, lending_data_vars_rem, sd)
sd_mo_rec = cbind(sd1, sd2[2], sd3[2])
kable(sd_mo_rec, col.names = c("Loan Status",
                                  "SD Recent BC",
```

```
                                "SD Recent Acc",
                                "SD Recent Revolving Acc"))
```

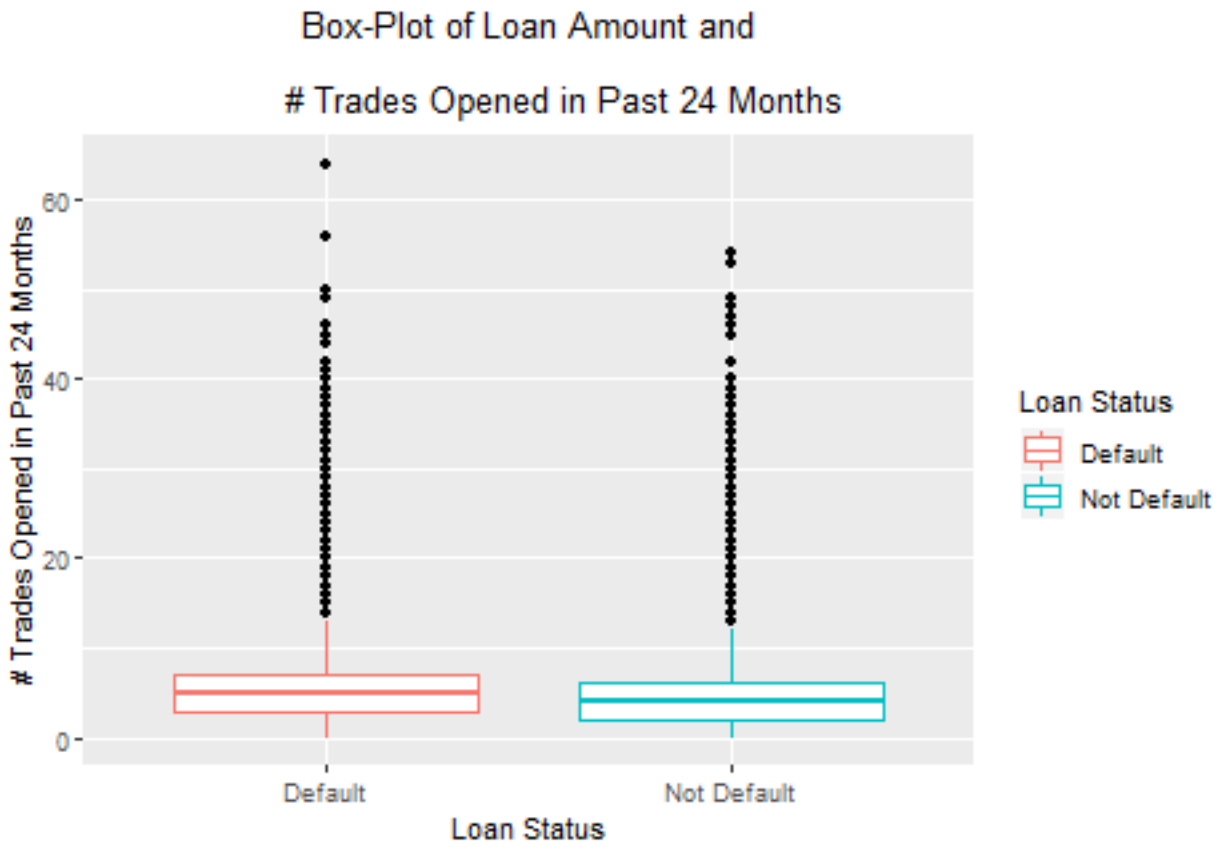
Loan Status	SD Recent BC	SD Recent Acc	SD Recent Revolving Acc
Default	27.85722	7.890417	14.49898
Not Default	31.28686	8.889145	16.68775

So, confirming with our belief from the box-plots, there is not much of a difference between groups in these three variables to be useful in our model. I will be removing these variables.

Next, let's take a look at the similar variables relating to the number of accounts a borrower has. These variables are *acc_open_past_24mths*, *num_actv_bc_tl*, *num_actv_rev_tl*, *num_bc_sats*, *num_bc_tl*, *num_il_tl*, *num_op_rev_tl*, *num_rev_accts*, *num_rev_tl_bal_gt_0*, *num_sats num_tl_op_past_12m*. The descriptions of these variables can be found above at the beginning of the section. Let's first take a look at all of their box plots.

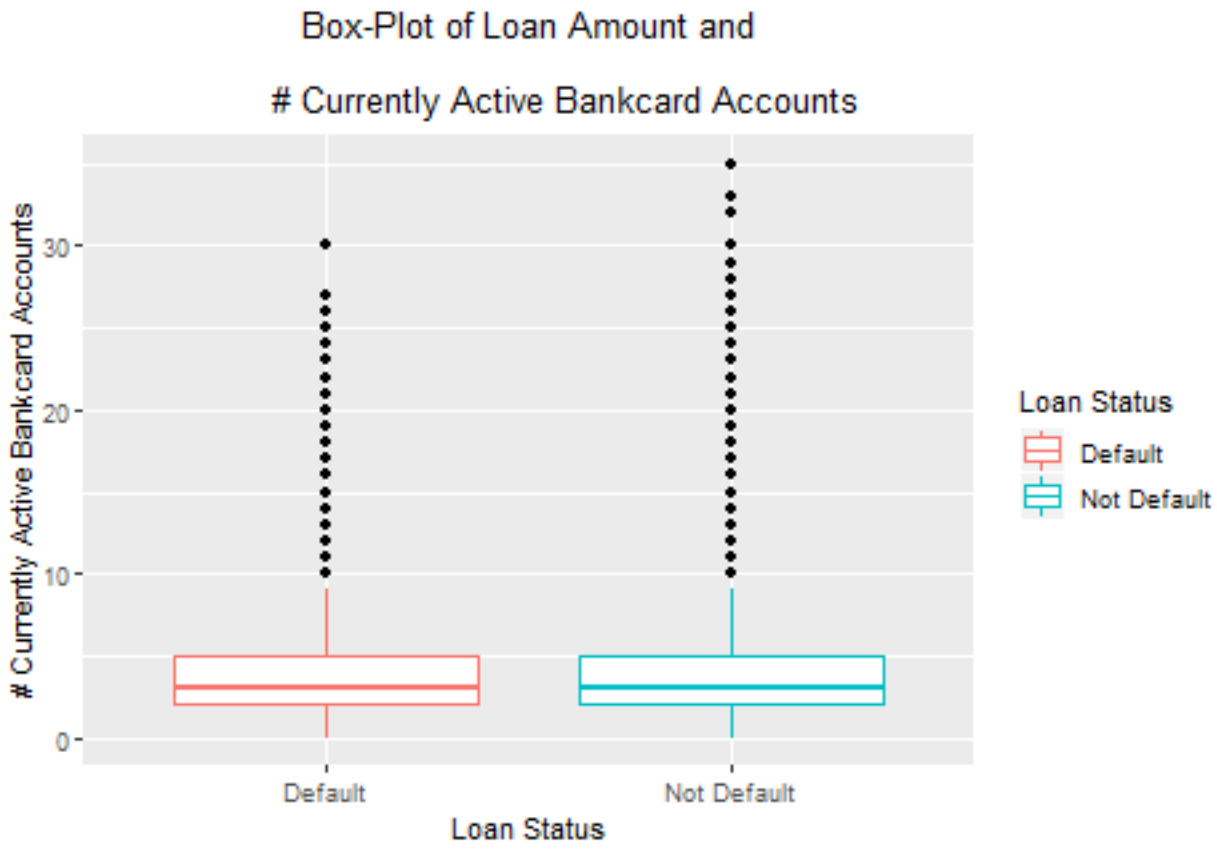
```
ggplot(lending_data_vars_rem, aes(x = loan_status_final, y = acc_open_past_24mths, color = loan_status_
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "# Trades Opened in Past 24 Months",
       title = "Box-Plot of Loan Amount and\n
       # Trades Opened in Past 24 Months",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
```

```
## Warning: Removed 50030 rows containing non-finite values (stat_boxplot).
```



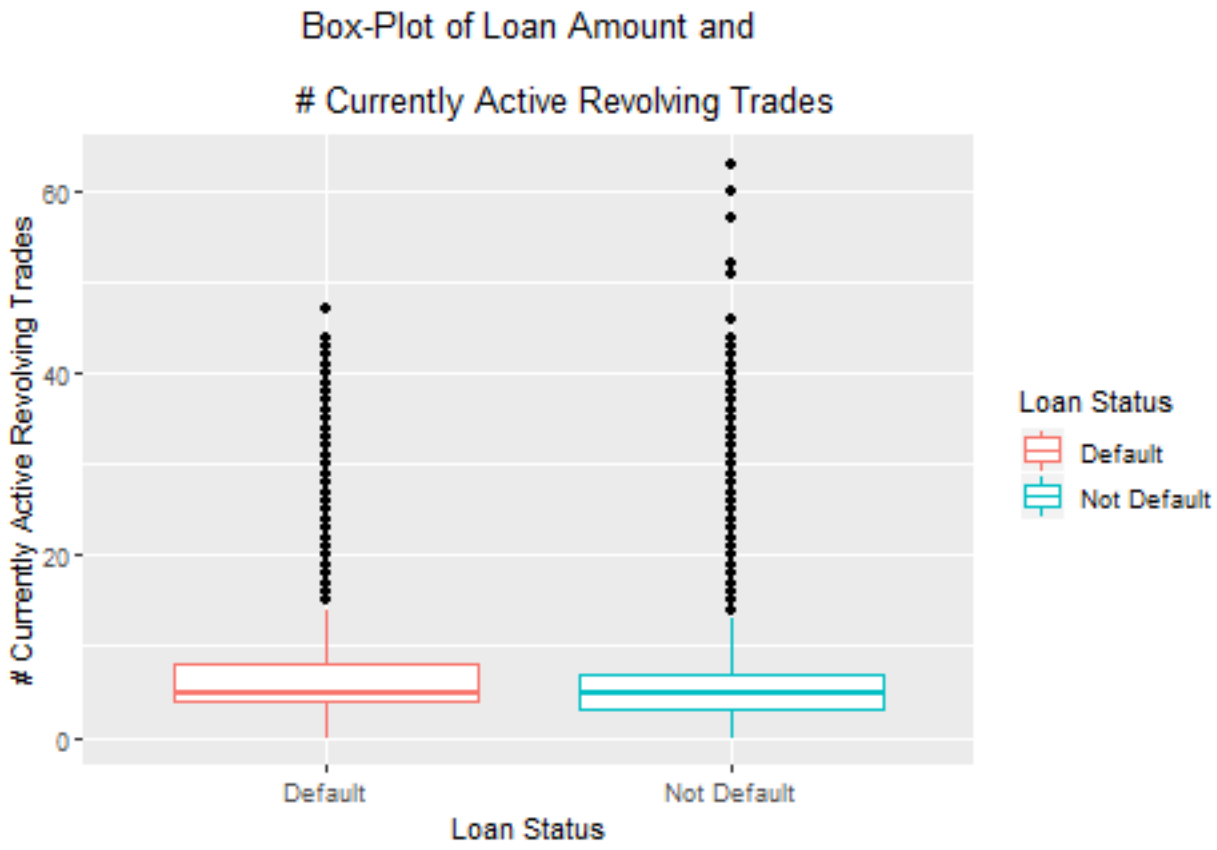

```
ggplot(lending_data_vars_rem, aes(x = loan_status_final, y = num_actv_bc_tl, color = loan_status_final))
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "# Currently Active Bankcard Accounts",
       title = "Box-Plot of Loan Amount and\n
       # Currently Active Bankcard Accounts",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
```

```
## Warning: Removed 70276 rows containing non-finite values (stat_boxplot).
```



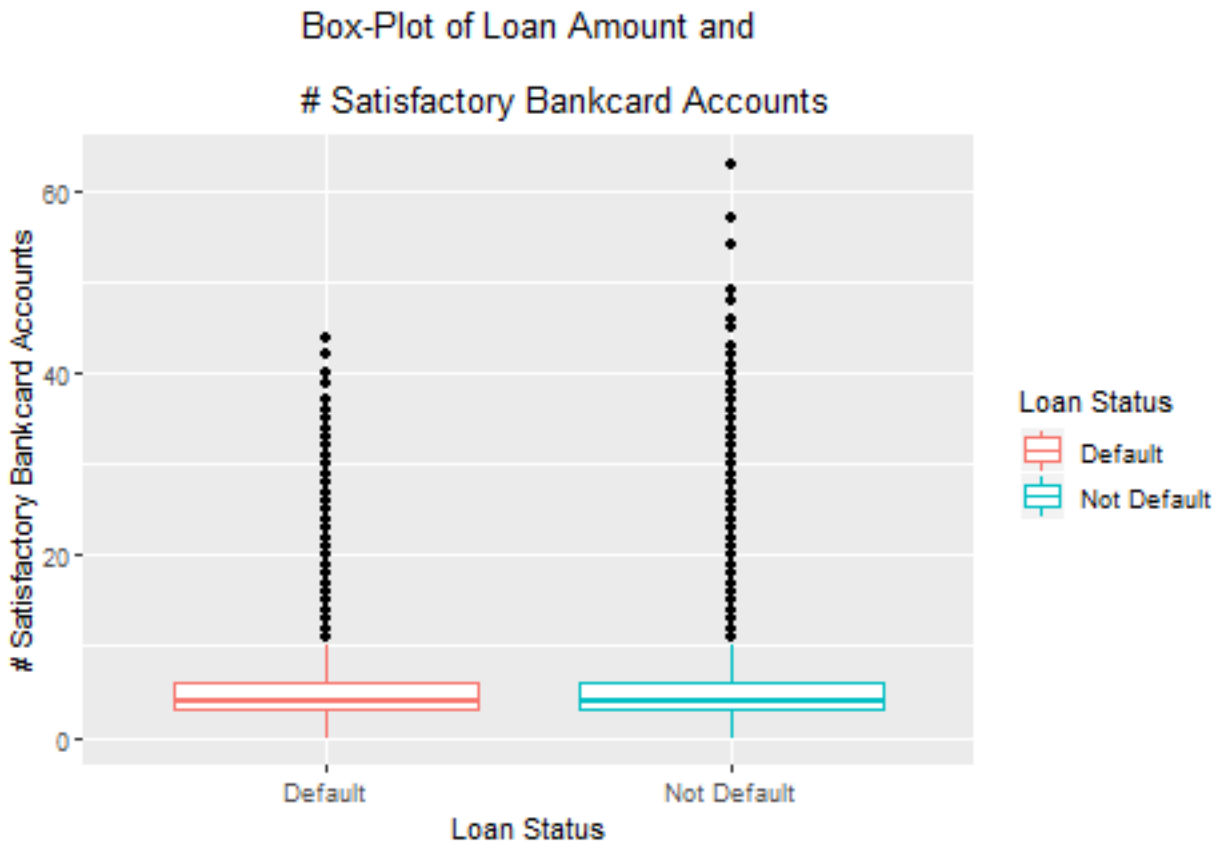

```
ggplot(lending_data_vars_rem, aes(x = loan_status_final, y = num_actv_rev_tl, color = loan_status_final
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "# Currently Active Revolving Trades",
       title = "Box-Plot of Loan Amount and\n
       # Currently Active Revolving Trades",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
```

```
## Warning: Removed 70276 rows containing non-finite values (stat_boxplot).
```



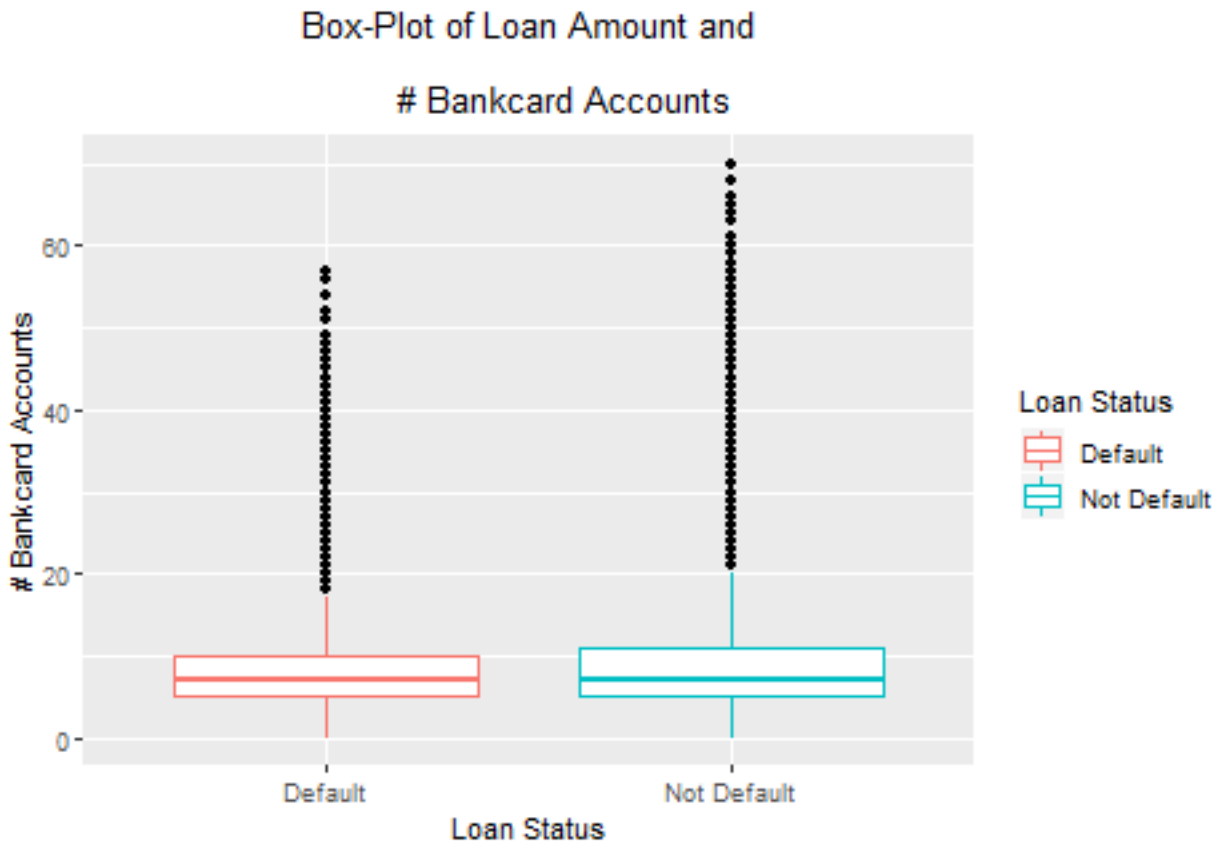

```
ggplot(lending_data_vars_rem, aes(x = loan_status_final, y = num_bc_sats, color = loan_status_final)) +
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "# Satisfactory Bankcard Accounts",
       title = "Box-Plot of Loan Amount and\n
       # Satisfactory Bankcard Accounts",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
```

```
## Warning: Removed 58590 rows containing non-finite values (stat_boxplot).
```



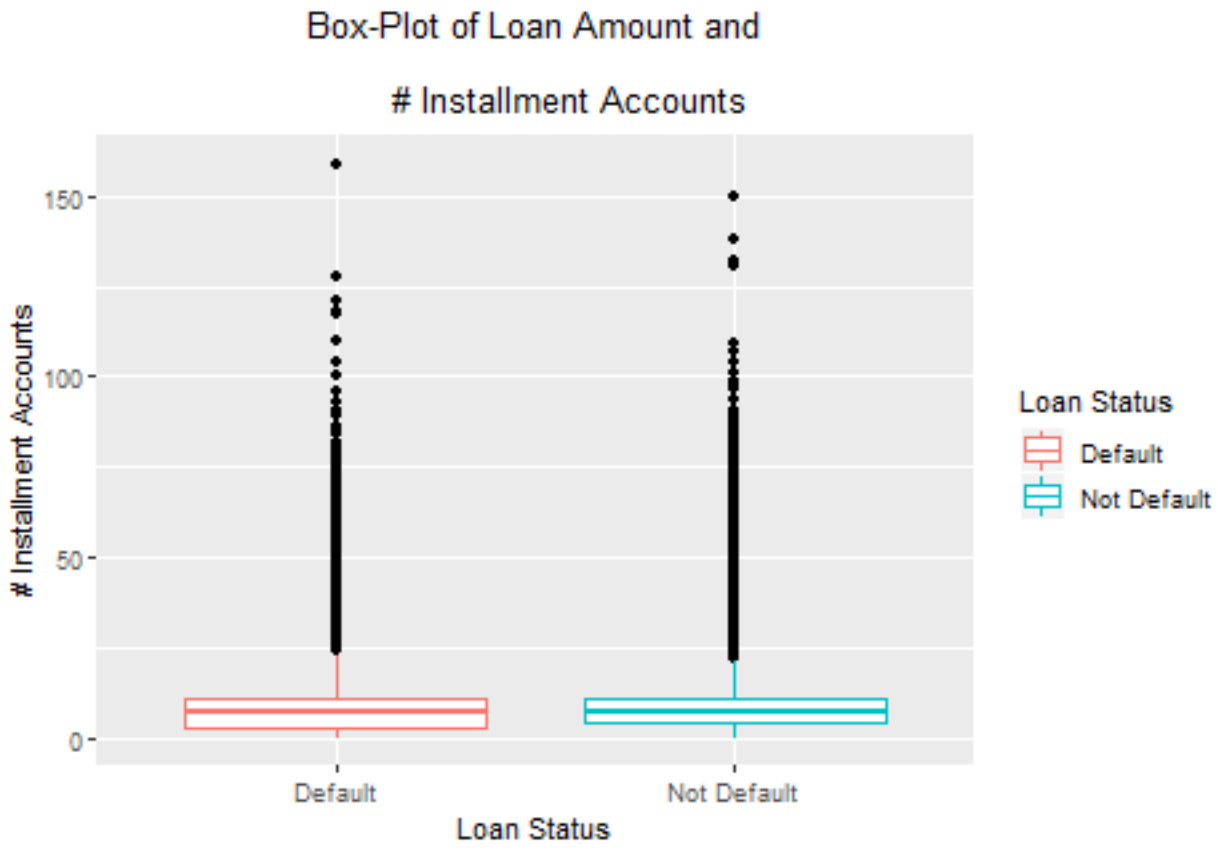

```
ggplot(lending_data_vars_rem, aes(x = loan_status_final, y = num_bc_tl, color = loan_status_final)) +
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "# Bankcard Accounts",
       title = "Box-Plot of Loan Amount and\n
       # Bankcard Accounts",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
```

```
## Warning: Removed 70276 rows containing non-finite values (stat_boxplot).
```



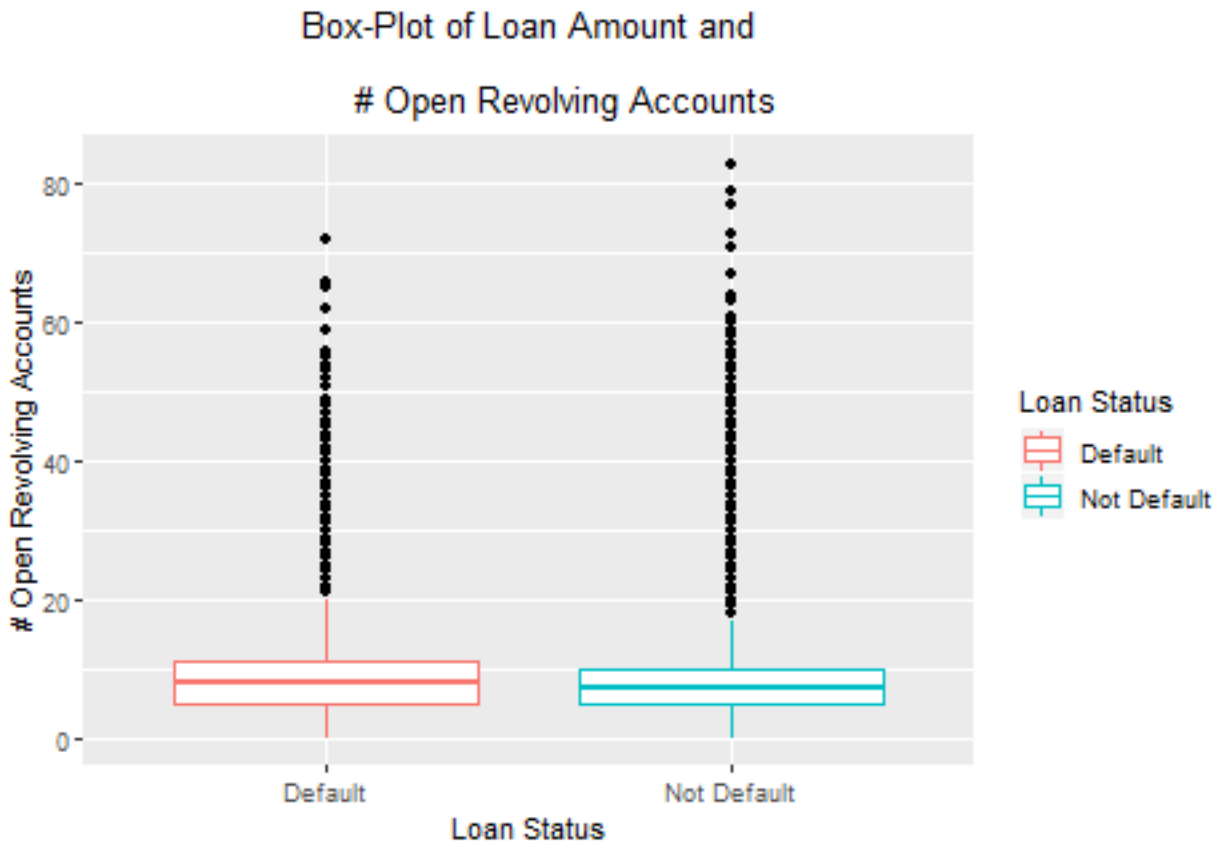

```
ggplot(lending_data_vars_rem, aes(x = loan_status_final, y = num_il_tl, color = loan_status_final)) +
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "# Installment Accounts",
       title = "Box-Plot of Loan Amount and\n
       # Installment Accounts",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
```

```
## Warning: Removed 70276 rows containing non-finite values (stat_boxplot).
```



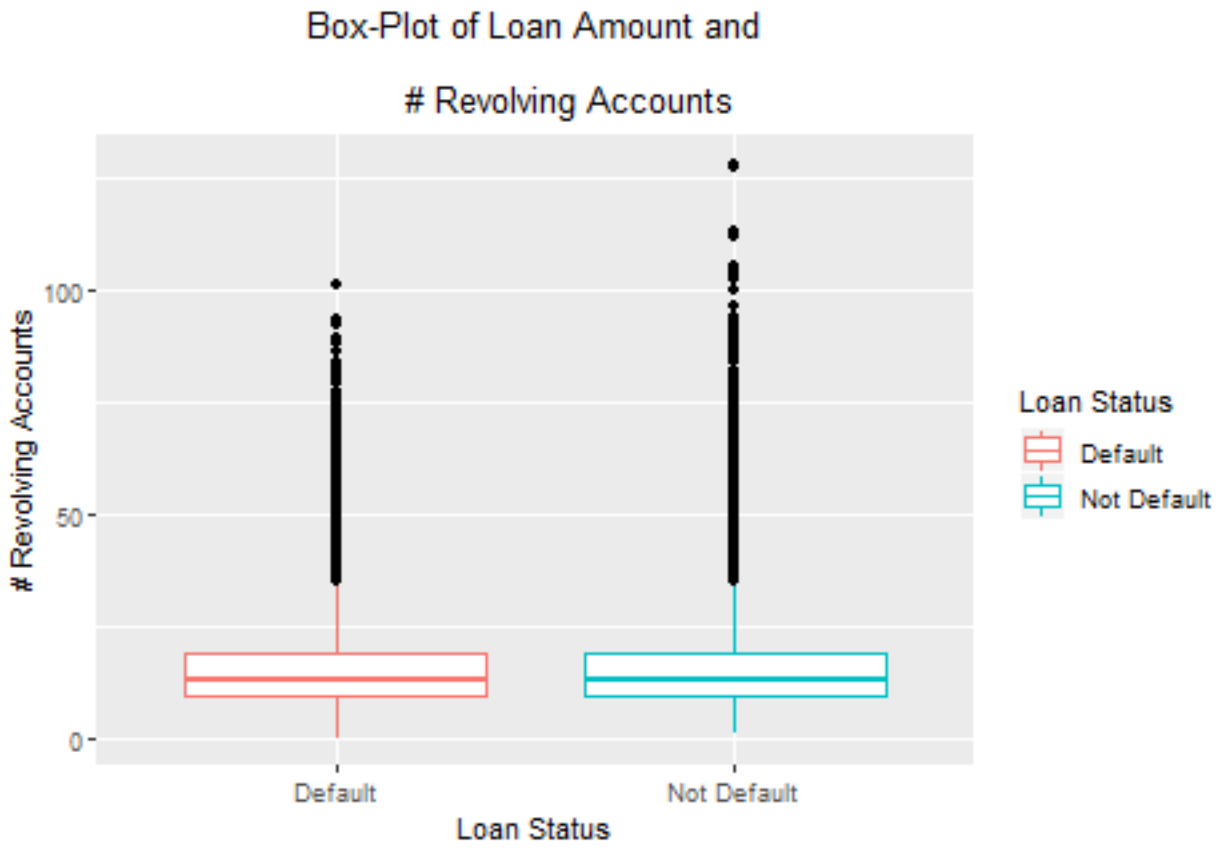

```
ggplot(lending_data_vars_rem, aes(x = loan_status_final, y = num_op_rev_tl, color = loan_status_final))
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "# Open Revolving Accounts",
       title = "Box-Plot of Loan Amount and\n
       # Open Revolving Accounts",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
```

```
## Warning: Removed 70276 rows containing non-finite values (stat_boxplot).
```



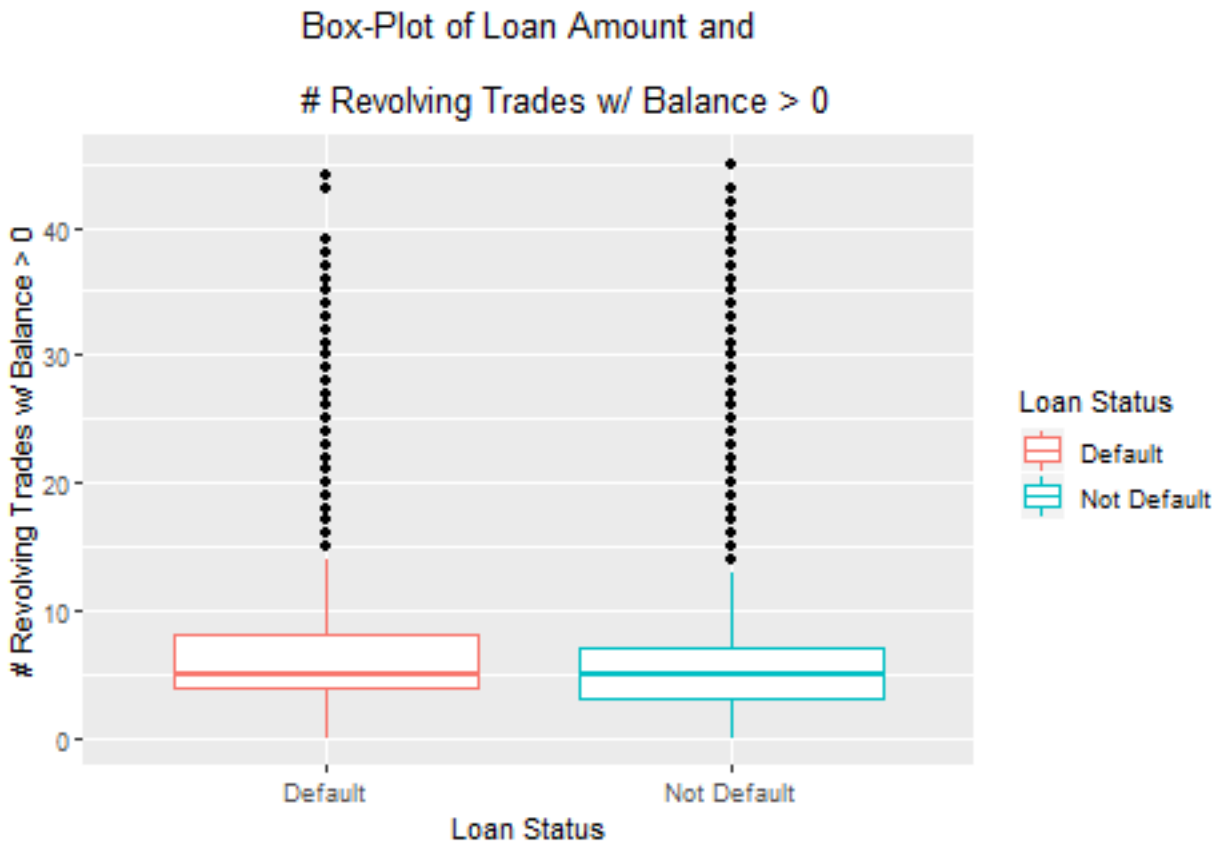

```
ggplot(lending_data_vars_rem, aes(x = loan_status_final, y = num_rev_accts, color = loan_status_final))
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "# Revolving Accounts",
       title = "Box-Plot of Loan Amount and\n
       # Revolving Accounts",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
```

```
## Warning: Removed 70277 rows containing non-finite values (stat_boxplot).
```



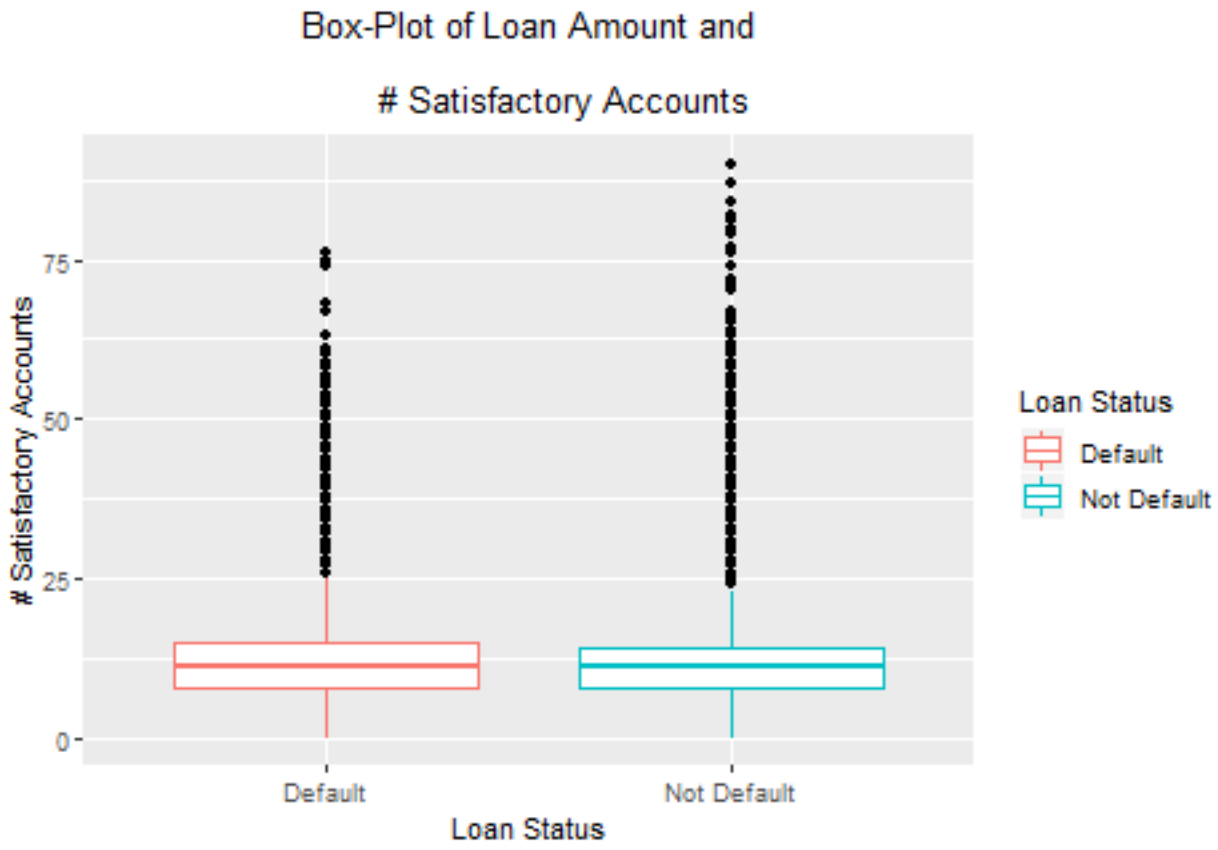

```
ggplot(lending_data_vars_rem, aes(x = loan_status_final, y = num_rev_tl_bal_gt_0, color = loan_status_f
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "# Revolving Trades w/ Balance > 0",
       title = "Box-Plot of Loan Amount and\n
       # Revolving Trades w/ Balance > 0",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
```

```
## Warning: Removed 70276 rows containing non-finite values (stat_boxplot).
```



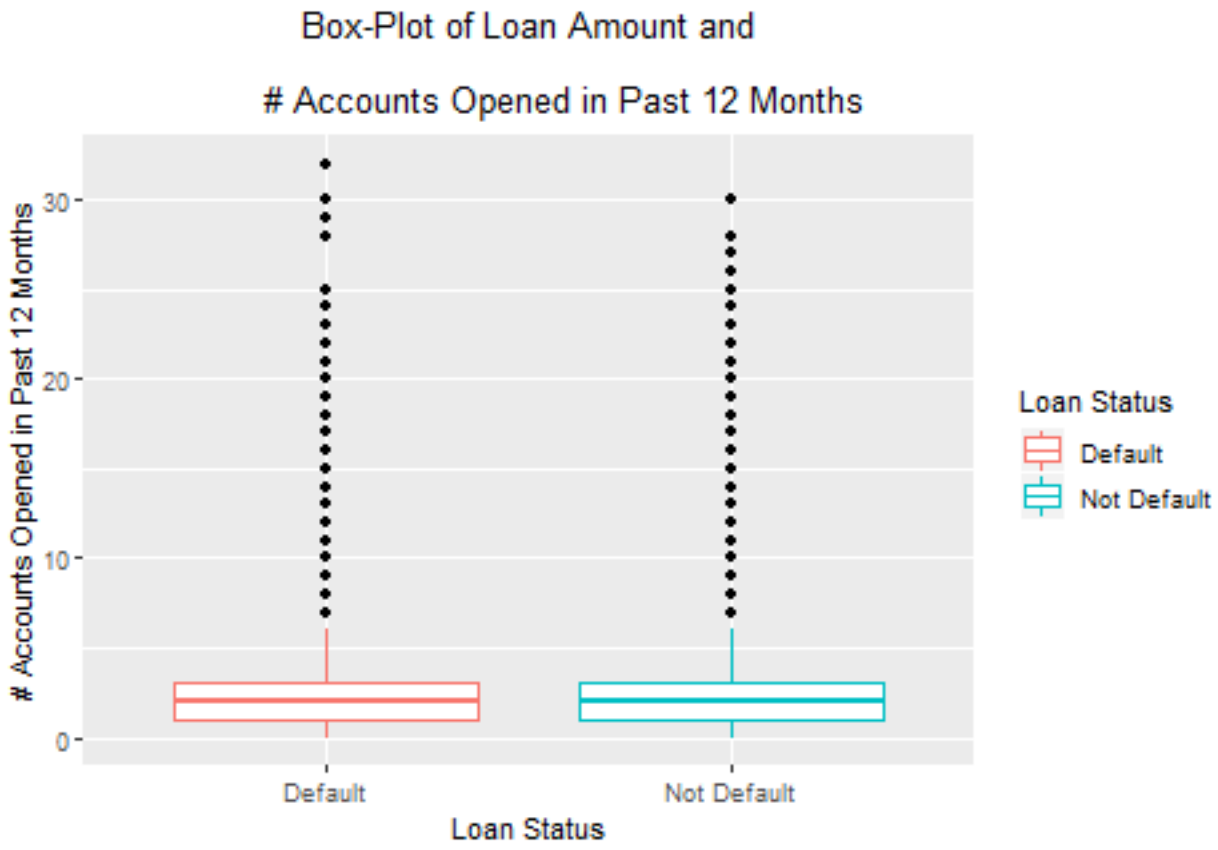

```
ggplot(lending_data_vars_rem, aes(x = loan_status_final, y = num_sats, color = loan_status_final)) +
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "# Satisfactory Accounts",
       title = "Box-Plot of Loan Amount and\n
       # Satisfactory Accounts",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
```

```
## Warning: Removed 58590 rows containing non-finite values (stat_boxplot).
```




```
ggplot(lending_data_vars_rem, aes(x = loan_status_final, y = num_tl_op_past_12m, color = loan_status_fi
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "# Accounts Opened in Past 12 Months",
       title = "Box-Plot of Loan Amount and\n
       # Accounts Opened in Past 12 Months",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
```

```
## Warning: Removed 70276 rows containing non-finite values (stat_boxplot).
```



Looking at the box-plots of all of these variables, we can see that their distributions are very similar and have similar means and spreads for both groups. Let's move forward to combining these variables into one variable, *num_accs*, which will be equal to the total number of accounts a borrower has. This will be created by adding together each of these variables.

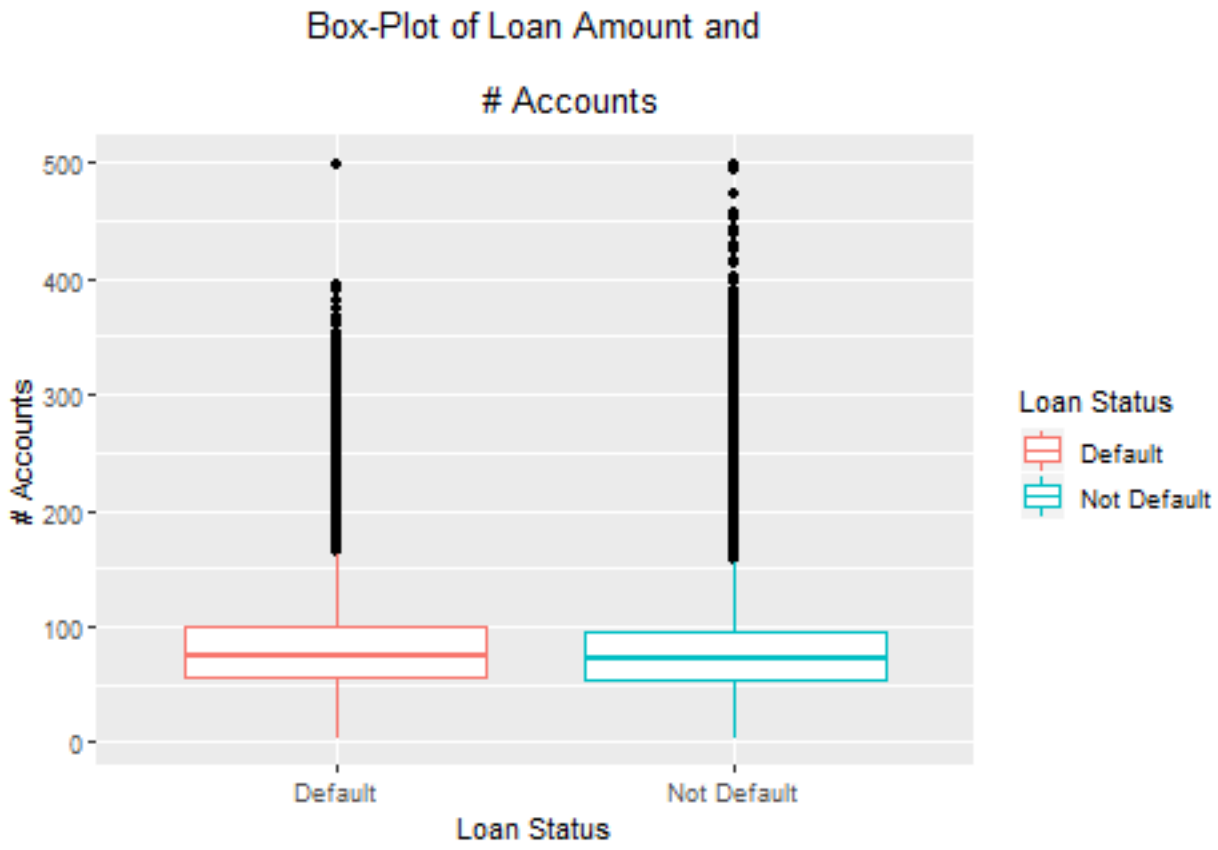
```
num_accs = lending_data_vars_rem$acc_open_past_24mths +
  lending_data_vars_rem$num_actv_bc_tl +
  lending_data_vars_rem$num_actv_rev_tl +
  lending_data_vars_rem$num_bc_sats +
  lending_data_vars_rem$num_bc_tl +
  lending_data_vars_rem$num_il_tl +
  lending_data_vars_rem$num_op_rev_tl +
  lending_data_vars_rem$num_rev_accts +
  lending_data_vars_rem$num_rev_tl_bal_gt_0 +
  lending_data_vars_rem$num_sats +
  lending_data_vars_rem$num_tl_op_past_12m

lending_data_vars_rem$num_accs = num_accs
```

Let's now take a look at the box-plot for this new variable.

```
ggplot(lending_data_vars_rem, aes(x = loan_status_final, y = num_accs, color = loan_status_final)) +
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "# Accounts",
       title = "Box-Plot of Loan Amount and\n
```

```
       # Accounts",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
```

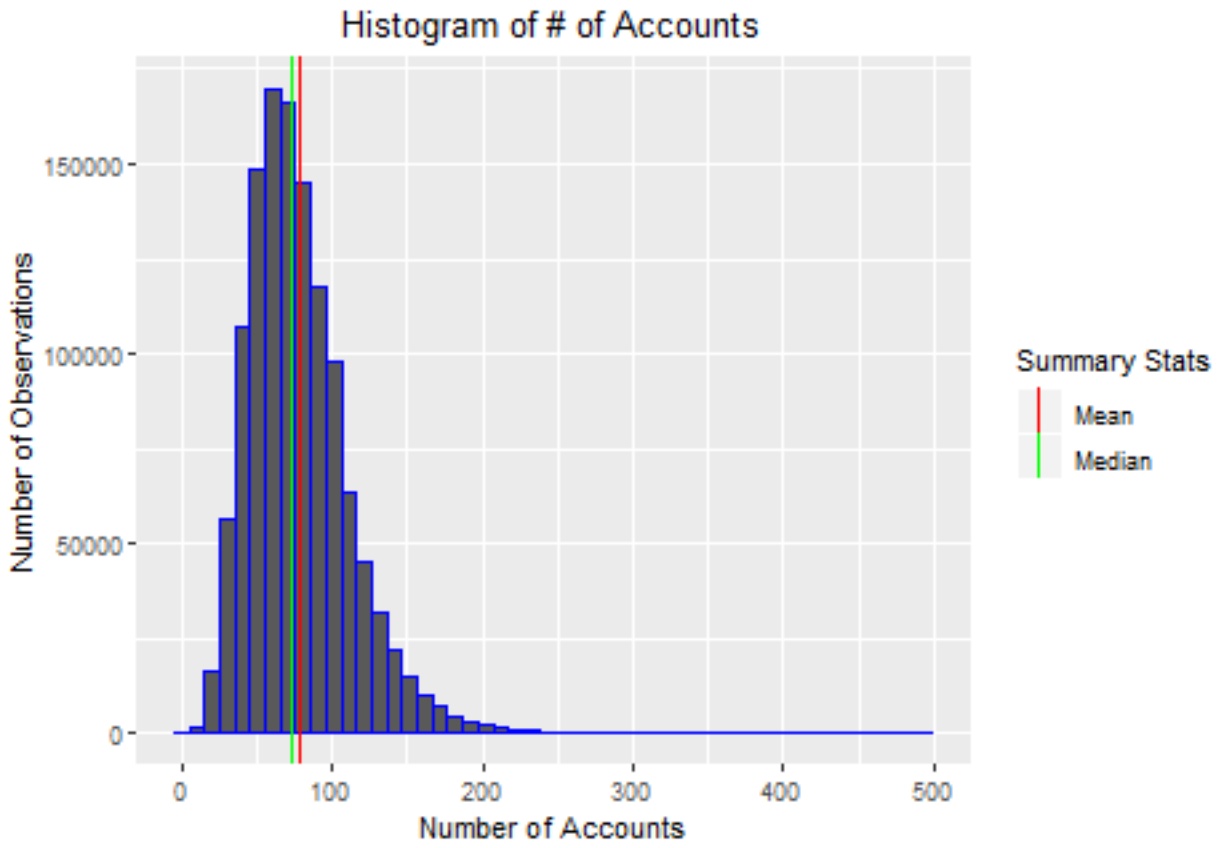
```
## Warning: Removed 70277 rows containing non-finite values (stat_boxplot).
```



With this combined accounts variable, we can see that the distribution stayed relatively the same as all of the other variables. Now, let's take care of the missing values. This variable has 70277 missing values which we must fill. Let's take a look at the histogram of this variable to see if we should impute the mean or median for missing values.

```
num_accs_mean = mean(num_accs, na.rm = TRUE)
num_accs_median = median(num_accs, na.rm = TRUE)
ggplot(lending_data_vars_rem, aes(x = num_accs)) +
  geom_histogram(stat = "bin", bins = 50, color = "blue") +
  geom_vline(aes(xintercept = num_accs_mean,
             color = "Mean")) +
  geom_vline(aes(xintercept = num_accs_median,
             color = "Median")) +
  xlab("Number of Accounts") +
  ylab("Number of Observations") +
  ggtitle("Histogram of # of Accounts") +
  theme(plot.title = element_text(hjust = 0.50)) +
  scale_color_manual(name = "Summary Stats",
                     labels = c("Mean", "Median"),
                     values = c("red", "green"))
```

```
## Warning: Removed 70277 rows containing non-finite values (stat_bin).
```



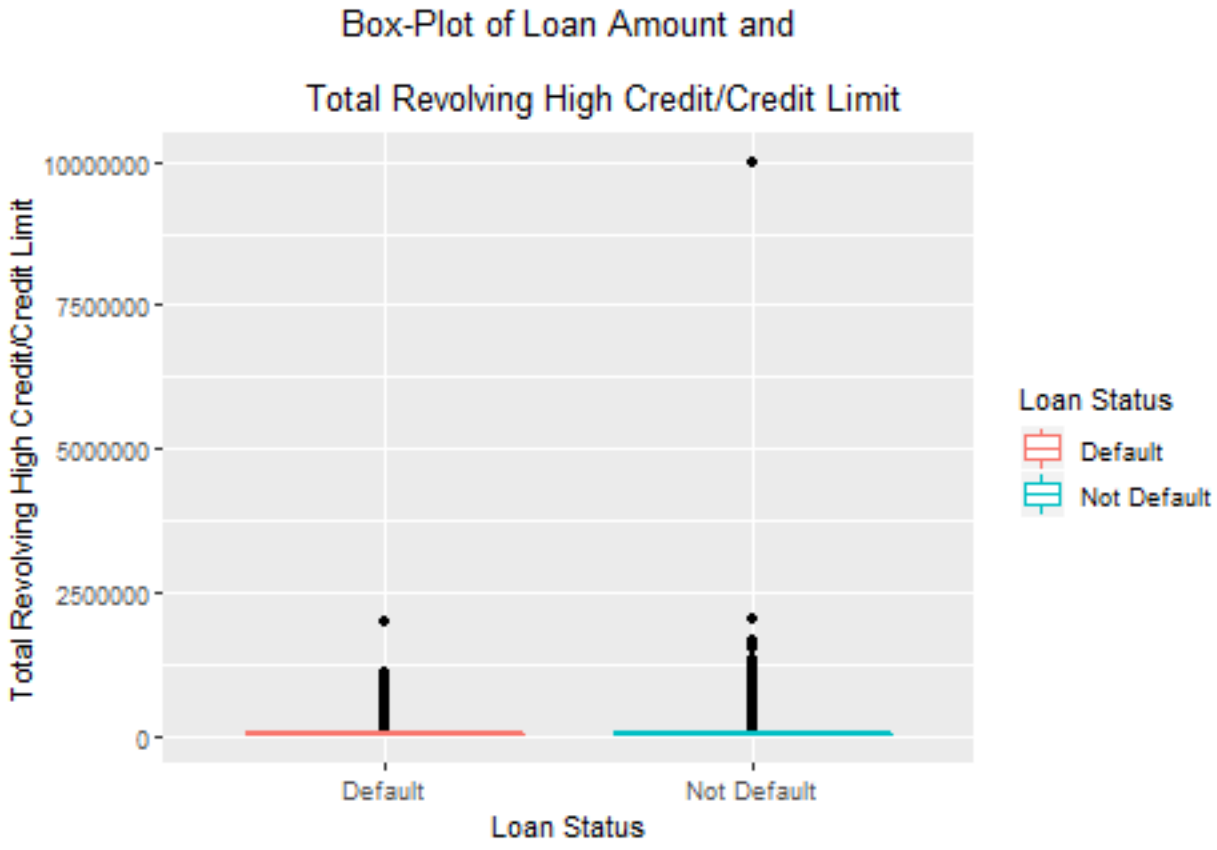
Looking at the histogram, I believe our best method of imputation will be the mean as it is most centered with the data. Let's go ahead and impute these missing values.

```
lending_data_vars_rem$num_accs[
  is.na(lending_data_vars_rem$num_accs)] = num_accs_mean
```

With the new variable created and missing values filled, let's move on to the next similar set of variables. These variables are *total_rev_hi_lim*, *tot_hi_cred_lim*, *total_bc_limit*, and *total_il_high_credit_limit*. These variables are all related to the total high credit/credit limits of accounts. Let's take a look at the box-plots for each variable.

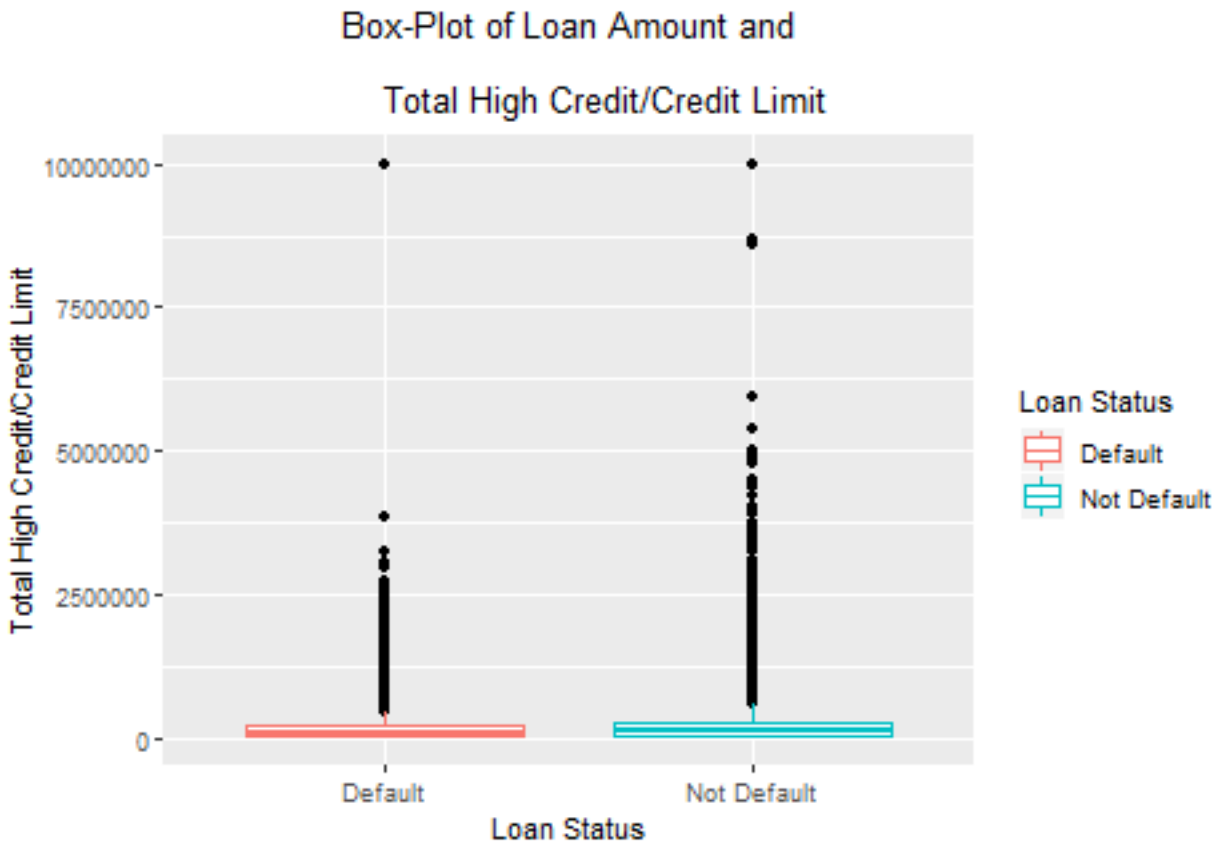
```
ggplot(lending_data_vars_rem, aes(x = loan_status_final,
        y = total_rev_hi_lim, color = loan_status_final)) +
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "Total Revolving High Credit/Credit Limit",
       title = "Box-Plot of Loan Amount and\n
       Total Revolving High Credit/Credit Limit",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
```

```
## Warning: Removed 70276 rows containing non-finite values (stat_boxplot).
```



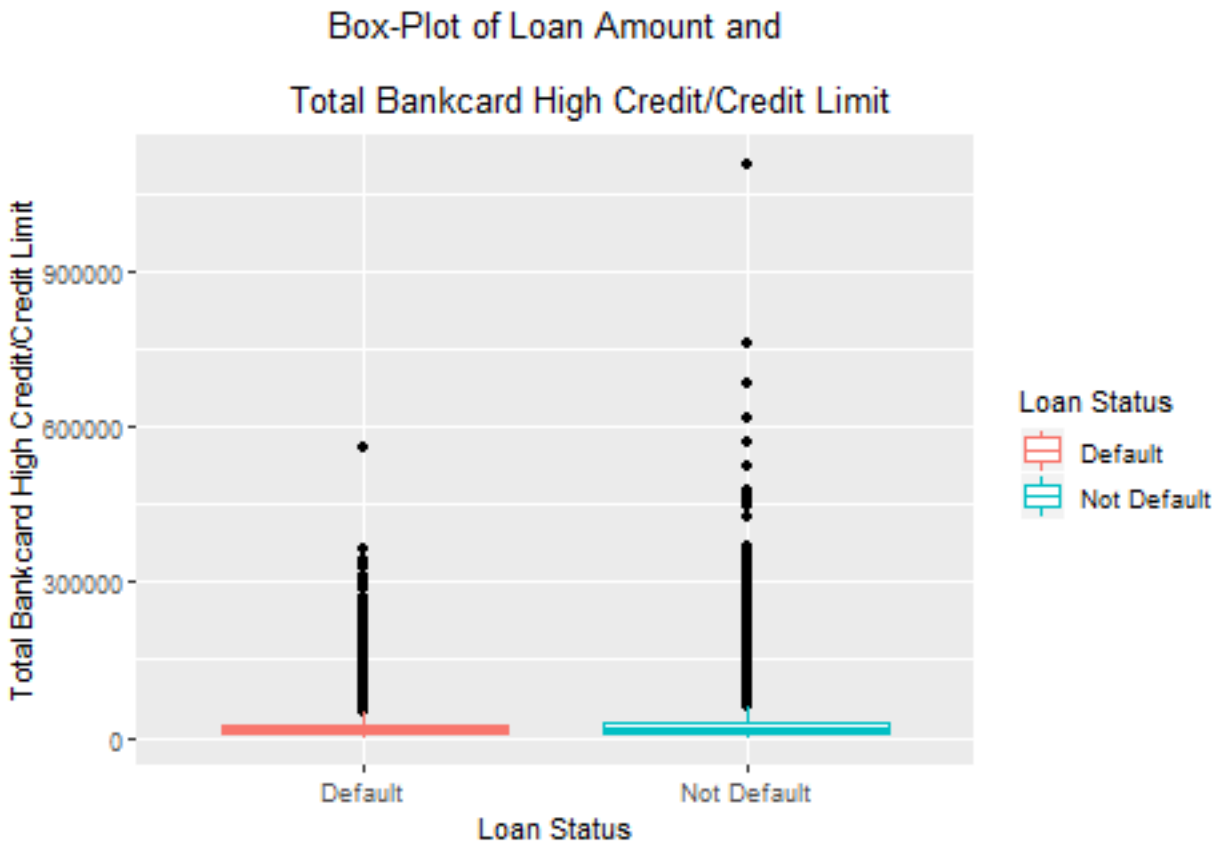

```
ggplot(lending_data_vars_rem, aes(x = loan_status_final,
         y = tot_hi_cred_lim, color = loan_status_final)) +
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "Total High Credit/Credit Limit",
       title = "Box-Plot of Loan Amount and\n
       Total High Credit/Credit Limit",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
```

```
## Warning: Removed 70276 rows containing non-finite values (stat_boxplot).
```



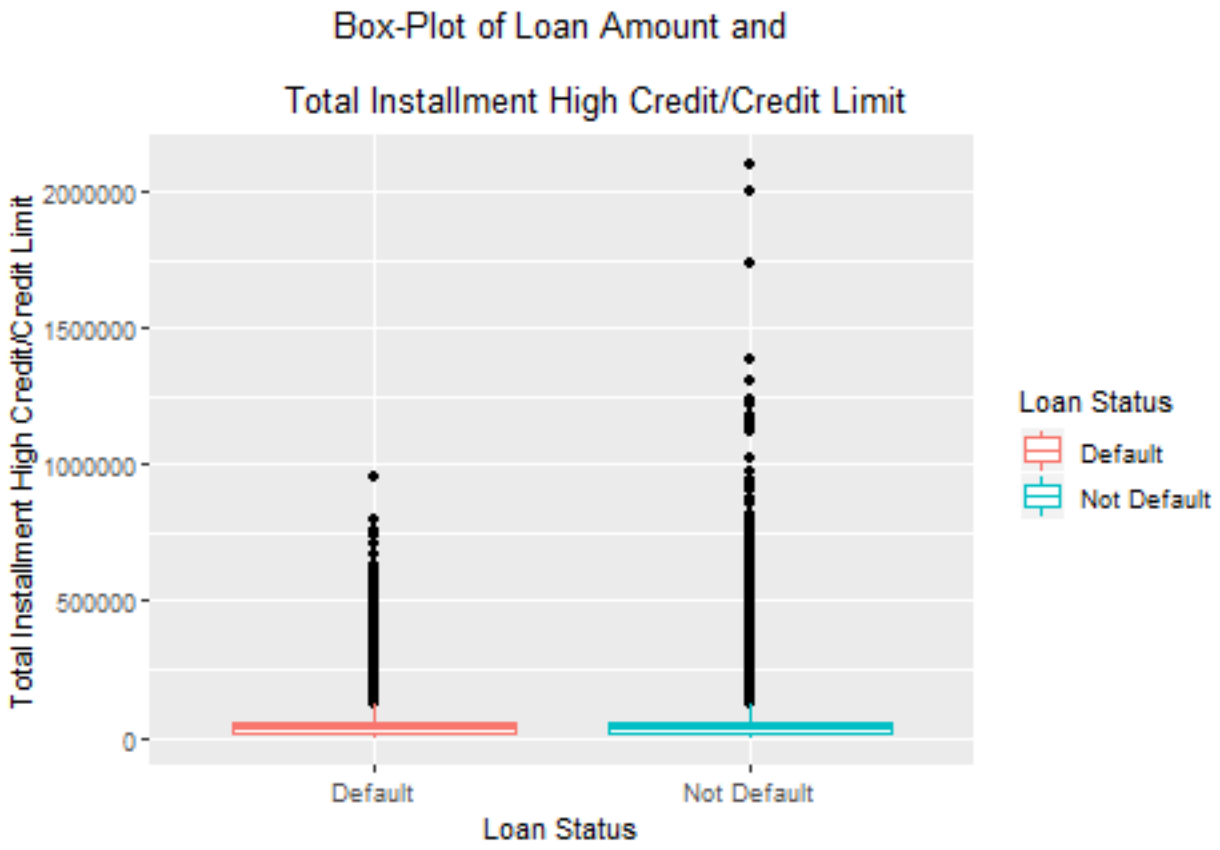

```
ggplot(lending_data_vars_rem, aes(x = loan_status_final,
          y = total_bc_limit, color = loan_status_final)) +
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "Total Bankcard High Credit/Credit Limit",
       title = "Box-Plot of Loan Amount and\n
       Total Bankcard High Credit/Credit Limit",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
```

```
## Warning: Removed 50030 rows containing non-finite values (stat_boxplot).
```




```
ggplot(lending_data_vars_rem, aes(x = loan_status_final,
  y = total_il_high_credit_limit, color = loan_status_final)) +
  geom_boxplot(outlier.colour = "black") +
  labs(x = "Loan Status",
       y = "Total Installment High Credit/Credit Limit",
       title = "Box-Plot of Loan Amount and\n
       Total Installment High Credit/Credit Limit",
       color = "Loan Status") +
  theme(plot.title = element_text(hjust = 0.50))
```

```
## Warning: Removed 70276 rows containing non-finite values (stat_boxplot).
```



Looking at the plots, these variables have similar distributions between each group of loan status. However, the means and spreads between each group seem very similar and therefore could not be useful in predicting loan default. Let's get a numerical summary of each variable.

```
means1 = aggregate(total_rev_hi_lim ~ loan_status_final, lending_data_vars_rem, mean)
means2 = aggregate(tot_hi_cred_lim ~ loan_status_final, lending_data_vars_rem, mean)
means3 = aggregate(total_bc_limit ~ loan_status_final, lending_data_vars_rem, mean)
means4 = aggregate(total_il_high_credit_limit ~ loan_status_final, lending_data_vars_rem, mean)
means_mo_rec = cbind(means1, means2[2], means3[2], means4[2])
kable(means_mo_rec, col.names = c("Loan Status",
                                  "Mean Revolving HC/CL",
                                  "Mean HC/CL",
                                  "Mean BC HC/CL",
                                  "Mean Installment HC/CL"))
```

Loan Status	Mean Revolving HC/CL	Mean HC/CL	Mean BC HC/CL	Mean Installment HC/CL
Default	28891.71	146425.2	18481.02	41991.38
Not Default	33682.12	181303.7	22351.13	42086.52

```
sd1 = aggregate(total_rev_hi_lim ~ loan_status_final, lending_data_vars_rem, sd)
sd2 = aggregate(tot_hi_cred_lim ~ loan_status_final, lending_data_vars_rem, sd)
sd3 = aggregate(total_bc_limit ~ loan_status_final, lending_data_vars_rem, sd)
sd4 = aggregate(total_il_high_credit_limit ~ loan_status_final, lending_data_vars_rem, sd)
sd_mo_rec = cbind(sd1, sd2[2], sd3[2], sd4[2])
```

```
kable(sd_mo_rec, col.names = c("Loan Status",
                               "SD Revolving HC/CL",
                               "SD HC/CL",
                               "SD BC HC/CL",
                               "SD Installment HC/CL"))
```

Loan Status	SD Revolving HC/CL	SD HC/CL	SD BC HC/CL	SD Installment HC/CL
Default	27983.33	151252.3	18376.83	41514.73
Not Default	38483.57	183112.1	22130.39	43577.13

Looking at the numerical summary, we can see that actually those who have not defaulted on their loan consistently have a higher high credit/credit limit. Therefore, this variable could be useful for predicting loan default. Let's go ahead and combine these variables into one variable, *hc_cl*, which gives us the borrowers total high credit/credit limit.

```
hc_cl = lending_data_vars_rem$total_rev_hi_lim +
  lending_data_vars_rem$tot_hi_cred_lim +
  lending_data_vars_rem$total_bc_limit +
  lending_data_vars_rem$total_il_high_credit_limit
lending_data_vars_rem$hc_cl = hc_cl
```

Now, let's go ahead and impute the mean for the missing values in this new variable.

```
hc_cl_mean = mean(hc_cl, na.rm = TRUE)
lending_data_vars_rem$hc_cl[
  is.na(lending_data_vars_rem$hc_cl)] = hc_cl_mean
```

That concludes similar variable combinations. Let's go through the rest of the variables and decide on removal or imputation.

We will start with *tot_coll_amt*. Let's take a look at the summary statistics of this variable.

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean Total Collection Amount" = mean(tot_coll_amt, na.rm=TRUE),
            "SD Total Collection Amount" = sd(tot_coll_amt, na.rm=TRUE),
            "Max Total Collection Amount" = max(tot_coll_amt, na.rm=TRUE))
```

```
## # A tibble: 2 x 4
##   loan_status_final `Mean Total Collec~ `SD Total Collec~ `Max Total Colle~
##   <chr>                           <dbl>             <dbl>             <dbl>
## 1 Default                          239.              1776.            146917
## 2 Not Default                      252.             12565.           9152545
```

Looking at the means, the groups in this variable are very similar. However, looking at the standard deviation we can see that something is off. The maximum total collection amount for the `Not Default` group is over 9 million. This signifies that this variable is not verified, and therefore will not be accurate and useful. I will be removing this variable.

Next, let's take a look at *tot_cur_bal*.

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean Total Current Balance" = mean(tot_cur_bal, na.rm=TRUE),
            "SD Total Current Balance" = sd(tot_cur_bal, na.rm=TRUE),
            "Max Total Current Balance" = max(tot_cur_bal, na.rm=TRUE))
```

```
## # A tibble: 2 x 4
##   loan_status_final `Mean Total Curren~ `SD Total Curren~ `Max Total Curre~
##   <chr>                           <dbl>             <dbl>             <dbl>
## 1 Default                       118736.           134279.           3437283
## 2 Not Default                   146783.           162177.           8000078
```

For this variable, we can see that the means between groups are largely different. However, with such a high maximum value this tells me that this variable was not verified and therefore will not be accurate and useful. I will be removing this variable.

Next, let's take a look at *avg_cur_bal*.

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean Average Current Balance" = mean(avg_cur_bal, na.rm=TRUE),
            "SD Average Current Balance" = sd(avg_cur_bal, na.rm=TRUE),
            "Max Average Current Balance" = max(avg_cur_bal, na.rm=TRUE))
```

```
## # A tibble: 2 x 4
##   loan_status_final `Mean Average Curr~ `SD Average Curr~ `Max Average Cur~
##   <chr>                           <dbl>             <dbl>             <dbl>
## 1 Default                         10915.            13198.            355824
## 2 Not Default                     14144.            16916.            958084
```

For this variable, we can see that the average current balance between groups differs by a good amount. Looking at the maximum current balance however tells me that this variable was also not verified and therefore will not be accurate and useful. Also, imputing values for an aggregated variable can cause issues. I will be removing this variable.

Next, let's take a look at *bc_open_to_buy*

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean Total Open to Buy on Bankcards" = mean(bc_open_to_buy, na.rm=TRUE),
            "SD Total Open to Buy on Bankcards" = sd(bc_open_to_buy, na.rm=TRUE),
            "Max Total Open to Buy on Bankcards" = max(bc_open_to_buy, na.rm=TRUE))
```

```
## # A tibble: 2 x 4
##   loan_status_final `Mean Total Open t~ `SD Total Open t~ `Max Total Open ~
##   <chr>                           <dbl>             <dbl>             <dbl>
## 1 Default                          7619.            11920.            327512
## 2 Not Default                     10762.            15957.            559912
```

For this variable, we can see that the average current balance between groups differs by a small amount. Looking at the maximum current balance however tells me that this variable was also not verified and therefore will not be accurate and useful. I will be removing this variable.

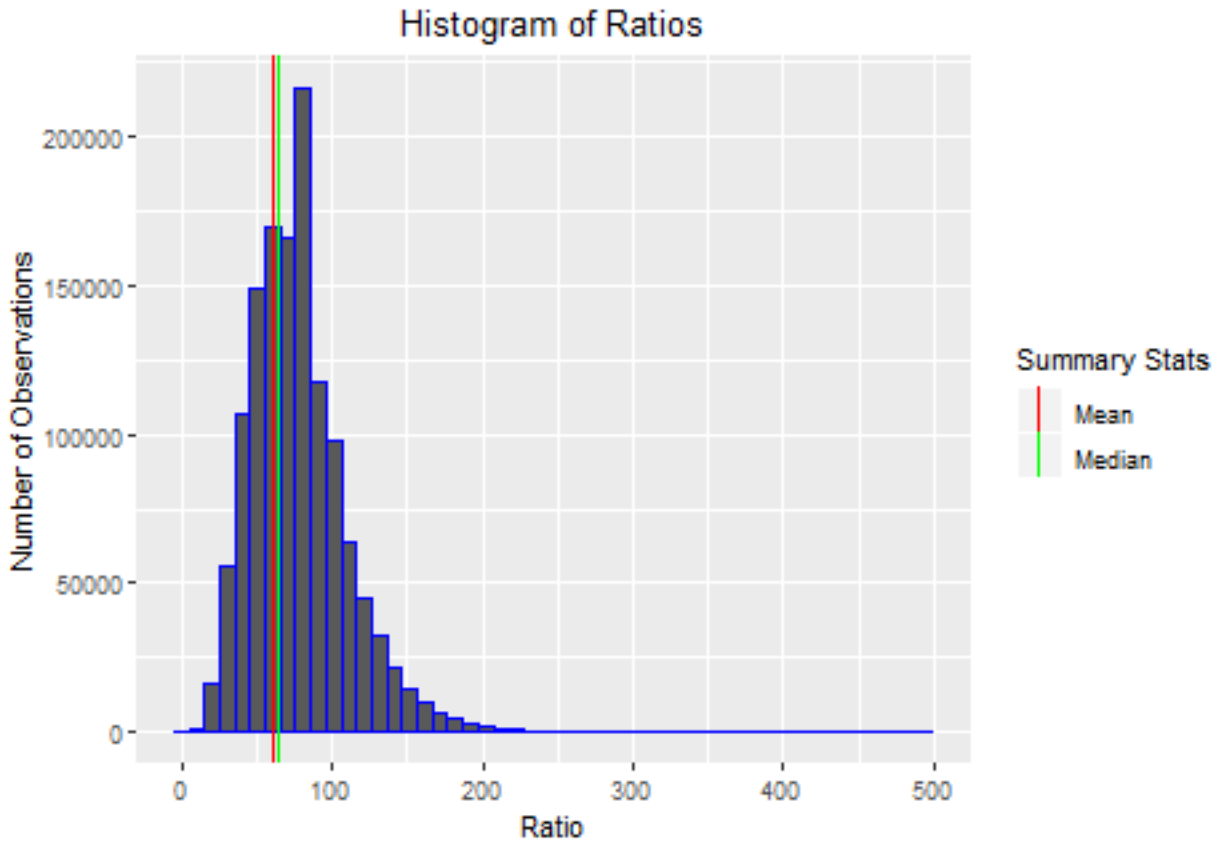
Next, let's take a look at *bc_util*. Recall this variable is the ratio of total current balance to high credit/credit limit for all bankcard accounts.

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean Ratio" = mean(bc_util, na.rm=TRUE),
            "SD Ratio" = sd(bc_util, na.rm=TRUE),
            "Max Ratio" = max(bc_util, na.rm=TRUE))
```

```
## # A tibble: 2 x 4
##   loan_status_final `Mean Ratio` `SD Ratio` `Max Ratio`
##   <chr>                    <dbl>      <dbl>       <dbl>
## 1 Default                   63.8       27.6        255.
## 2 Not Default               59.1       28.4        340.
```

Looking at the means between groups, the difference is not very high but the standard deviations are consistent. Those who have defaulted on their loan have a higher ratio of current balance to credit limit. This variable seems to me that it would be useful for predicting loan default as it is related to paying off balances. Let's take a look at the distribution of this variable for imputation.

```
bc_util = lending_data_vars_rem$bc_util
bc_util_mean = mean(bc_util, na.rm = TRUE)
bc_util_median = median(bc_util, na.rm = TRUE)
ggplot(lending_data_vars_rem, aes(x = num_accs)) +
  geom_histogram(stat = "bin", bins = 50, color = "blue") +
  geom_vline(aes(xintercept = bc_util_mean,
             color = "Mean")) +
  geom_vline(aes(xintercept = bc_util_median,
             color = "Median")) +
  xlab("Ratio") +
  ylab("Number of Observations") +
  ggtitle("Histogram of Ratios") +
  theme(plot.title = element_text(hjust = 0.50)) +
  scale_color_manual(name = "Summary Stats",
                     labels = c("Mean", "Median"),
                     values = c("red", "green"))
```



Looking at the distribution, let's go ahead and impute the median as it more centered.

```
lending_data_vars_rem$bc_util[
  is.na(lending_data_vars_rem$bc_util)] = bc_util_median
```

Next, let's take a look at *mo_sin_old_rev_tl_op*.

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean Mos Since Oldest Rev. Acc. Opened" =
              mean(mo_sin_old_rev_tl_op, na.rm=TRUE),
            "SD Mos Since Oldest Rev. Acc. Opened" =
              sd(mo_sin_old_rev_tl_op, na.rm=TRUE),
            "Max Mos Since Oldest Rev. Acc. Opened" =
              max(mo_sin_old_rev_tl_op, na.rm=TRUE))
```

```
## # A tibble: 2 x 4
##   loan_status_final `Mean Mos Since Ol~ `SD Mos Since Ol~ `Max Mos Since O~
##   <chr>                           <dbl>             <dbl>             <dbl>
## 1 Default                          172.              94.6               842
## 2 Not Default                      184.              94.1               852
```

Looking at the summary statistics, we can see that the mean months since the oldest revolving account opened are similar between loan status groups. The standard deviations of each group are almost identical, as well as the maximum. This tells me that there is not much of a difference between groups, and therefore this variable will not be very useful for predicting loan default. I will be removing this variable.

Next, let's take a look at *mort_acc*.

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean # of Mortgage Accs." =
              mean(mort_acc, na.rm=TRUE),
            "SD # of Mortgage Accs" =
              sd(mort_acc, na.rm=TRUE),
            "Max # of Mortgage Accs" =
              max(mort_acc, na.rm=TRUE))
```

```
## # A tibble: 2 x 4
##   loan_status_final `Mean # of Mortgag~ `SD # of Mortgag~ `Max # of Mortga~
##   <chr>                           <dbl>             <dbl>             <dbl>
## 1 Default                          1.38              1.83                29
## 2 Not Default                      1.75              2.04                51
```

Looking at the summary statistics, the mean # of mortgage accounts between loan status groups are similar. This makes sense to me as those who default or don't default on a loan will still own a house or have some sort of housing. Therefore, there will not be much of a difference between groups making this variable not very useful for predicting loan default. I will be removing this variable.

Next, let's take a look at *num_accts_ever_120_pd*

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean # Accs. Ever 120 Days Past Due" =
              mean(num_accts_ever_120_pd, na.rm=TRUE),
            "SD # Accs. Ever 120 Days Past Due" =
              sd(num_accts_ever_120_pd, na.rm=TRUE),
            "Max # Accs. Ever 120 Days Past Due" =
              max(num_accts_ever_120_pd, na.rm=TRUE))
```

```
## # A tibble: 2 x 4
##   loan_status_final `Mean # Accs. Ever~ `SD # Accs. Ever~ `Max # Accs. Eve~
##   <chr>                           <dbl>             <dbl>             <dbl>
## 1 Default                         0.536              1.34                34
## 2 Not Default                     0.503              1.32                51
```

Looking at the summary statistics, the mean and standard deviation are almost identical between groups. With this result, this variable will not be of much use for predicting loan default. I will be removing this variable.

Next, let's take a look at *pct_tl_nvr_dlq*

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean Perc. Trades Never Delinquent" =
              mean(pct_tl_nvr_dlq, na.rm=TRUE),
            "SD Perc. Trades Never Delinquent" =
              sd(pct_tl_nvr_dlq, na.rm=TRUE),
            "Max Perc. Trades Never Delinquent" =
              max(pct_tl_nvr_dlq, na.rm=TRUE))
```

```
## # A tibble: 2 x 4
##   loan_status_final `Mean Perc. Trades~ `SD Perc. Trades~ `Max Perc. Trade~
```

```
##   <chr>                          <dbl>             <dbl>             <dbl>
## 1 Default                         94.0              8.79               100
## 2 Not Default                     94.2              8.71               100
```

Looking at the summary statistics, the mean and standard deviation are almost identical between groups. With this result, this variable will not be of much use for predicting loan default. I will be removing this variable.

Next, let's take a look at *percent_bc_gt_75*.

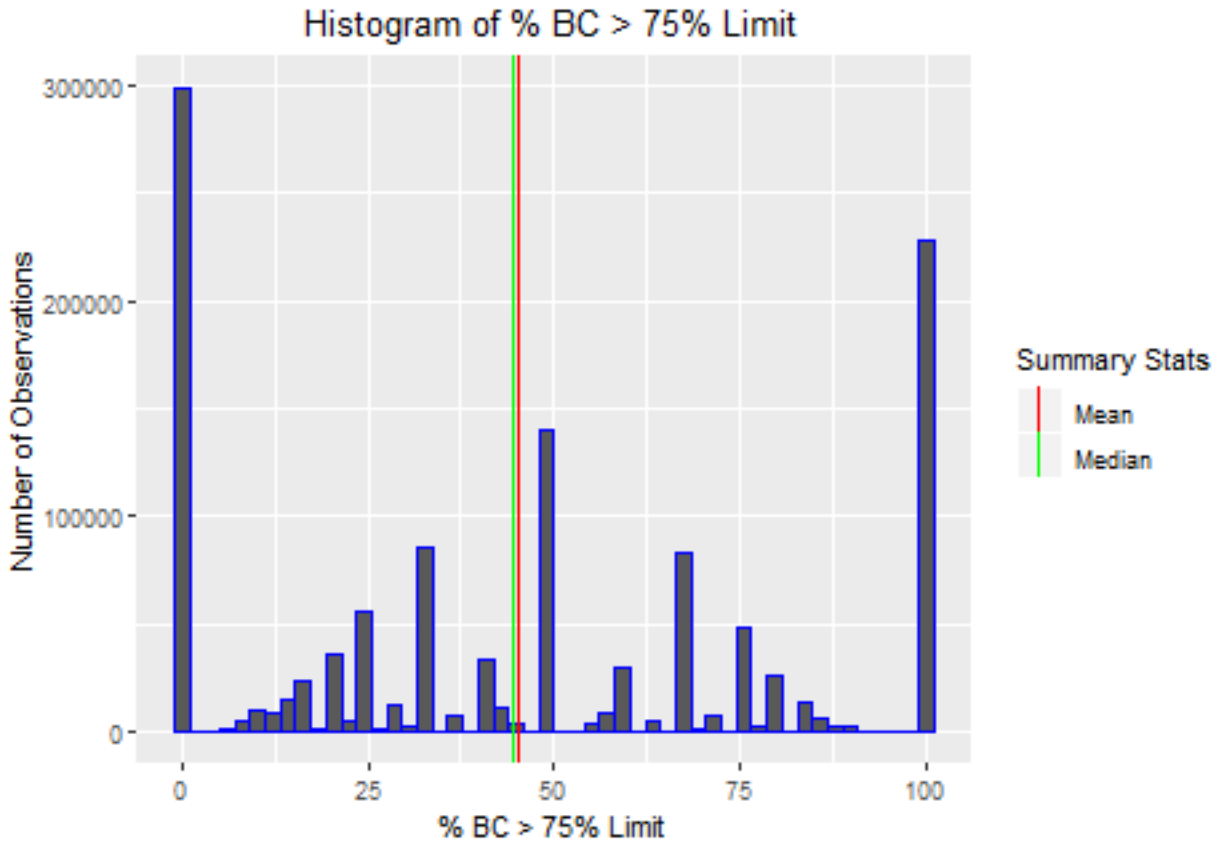
```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean Perc. BC > 75% of Limit" =
              mean(percent_bc_gt_75, na.rm=TRUE),
            "SD Perc. BC > 75% of Limit" =
              sd(percent_bc_gt_75, na.rm=TRUE),
            "Max Perc. BC > 75% of Limit" =
              max(percent_bc_gt_75, na.rm=TRUE))
```

```
## # A tibble: 2 x 4
##   loan_status_final `Mean Perc. BC > 7~ `SD Perc. BC > 7~ `Max Perc. BC > ~
##   <chr>                           <dbl>             <dbl>             <dbl>
## 1 Default                          50.2              35.8               100
## 2 Not Default                      44.1              35.9               100
```

Looking at the summary statistics, the mean between groups is different by a few percentage points. The standard deviation between groups is almost identical. The maximum for both groups is 100%. Since this variable tells us the percent of a borrowers bankcards greater than 75% of the limit, I believe this will be useful for predicting loan default as those with higher useage will have a harder time paying back their cards and loan.

```
percent_bc_gt_75 = lending_data_vars_rem$percent_bc_gt_75
percent_bc_gt_75_mean = mean(percent_bc_gt_75, na.rm = TRUE)
percent_bc_gt_75_median = median(percent_bc_gt_75, na.rm = TRUE)
ggplot(lending_data_vars_rem, aes(x = percent_bc_gt_75)) +
  geom_histogram(stat = "bin", bins = 50, color = "blue") +
  geom_vline(aes(xintercept = percent_bc_gt_75_mean,
             color = "Mean")) +
  geom_vline(aes(xintercept = percent_bc_gt_75_median,
             color = "Median")) +
  xlab("% BC > 75% Limit") +
  ylab("Number of Observations") +
  ggtitle("Histogram of % BC > 75% Limit") +
  theme(plot.title = element_text(hjust = 0.50)) +
  scale_color_manual(name = "Summary Stats",
                     labels = c("Mean", "Median"),
                     values = c("red", "green"))
```

```
## Warning: Removed 63797 rows containing non-finite values (stat_bin).
```



There is not much of a distribution in this variable. However, I believe that imputing the median in this variable will be best since this variable seems to have value to predicting loan default. Let's go ahead and fill the missing values with the median.

```
lending_data_vars_rem$percent_bc_gt_75[
  is.na(lending_data_vars_rem$percent_bc_gt_75)] = percent_bc_gt_75_median
```

Next, let's take a look at *total_bal_ex_mort*. Since we already removed *tot_cur_bal*, this variable might be of more use since there will not be a large loan included which is a mortgage.

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean Total Bal. Excl. Mortgage" =
              mean(total_bal_ex_mort, na.rm=TRUE),
            "SD Total Bal. Excl. Mortgage" =
              sd(total_bal_ex_mort, na.rm=TRUE),
            "Max Total Bal. Excl. Mortgage" =
              max(total_bal_ex_mort, na.rm=TRUE))
```

```
## # A tibble: 2 x 4
##   loan_status_final `Mean Total Bal. E~ `SD Total Bal. E~ `Max Total Bal. ~
##   <chr>                           <dbl>             <dbl>             <dbl>
## 1 Default                        49301.            44732.           1896461
## 2 Not Default                    49694.            48354.           3408095
```

Looking at the summary statistics, the mean and standard deviation are almost identical between groups. Also, looking at the maximum values between groups we can see that this variable is not verified. With this result, this variable will not be of much use for predicting loan default. I will be removing this variable.

We have now gone through all of the variables that have greater than 50,000 missing values. Let's remove all of these variables that I have stated to be removed.

```
remove_list2 = c("mo_sin_rcnt_rev_tl_op",
      "mo_sin_rcnt_tl",
      "mo_sin_rcnt_rev_tl_op",
      "mths_since_recent_bc",
      "num_actv_bc_tl",
      "num_actv_rev_tl",
      "num_bc_sats",
      "num_bc_tl",
      "num_il_tl",
      "num_op_rev_tl",
      "num_rev_accts",
      "num_rev_tl_bal_gt_0",
      "num_sats",
      "num_tl_op_past_12m",
      "total_rev_hi_lim",
      "tot_hi_cred_lim",
      "total_bc_limit",
      "total_il_high_credit_limit",
      "tot_coll_amt",
      "tot_cur_bal",
      "avg_cur_bal",
      "bc_open_to_buy",
      "mo_sin_old_rev_tl_op",
      "mort_acc",
      "num_accts_ever_120_pd",
      "pct_tl_nvr_dlq",
      "total_bal_ex_mort",
      "num_tl_30dpd",
      "num_tl_90g_dpd_24m"
)

lending_data_vars_rem = lending_data_vars_rem[, !(names(lending_data_vars_rem) %in% remove_list2)]
total_vars_4 = ncol(lending_data_vars_rem) - 1
```

We have now reduced the number of columns to 43. We now want to focus on fixing variables with any amount of missing values. Let's go ahead and get a list of these variables.

```
numeric_vars = lending_data_vars_rem %>% select_if(is.numeric)
num_any_missing = colnames(numeric_vars)[colSums(is.na(numeric_vars)) > 0]
num_any_missing
```

```
##  [1] "annual_inc"                "dti"
##  [3] "delinq_2yrs"               "inq_last_6mths"
##  [5] "open_acc"                  "pub_rec"
##  [7] "revol_util"                "total_acc"
##  [9] "collections_12_mths_ex_med" "acc_now_delinq"
## [11] "acc_open_past_24mths"      "chargeoff_within_12_mths"
## [13] "delinq_amnt"               "pub_rec_bankruptcies"
## [15] "tax_liens"
```

- "annual_inc" - The self-reported annual income provided by the borrower during registration.

- "dti" - Debt (Excluding Mortgage & LC Loan) to Income Ratio.
- "delinq_2yrs" - The number of 30+ days past-due incidences of delinquency in the borrower's credit file for the past 2 years
- "inq_last_6mths" - The number of inquiries in past 6 months (excluding auto and mortgage inquiries)
- "open_acc" - The number of open credit lines in the borrower's credit file.
- "pub_rec" - Number of derogatory public records
- "revol_util" - Total credit revolving balance
- "total_acc" - The total number of credit lines currently in the borrower's credit file
- "collections_12_mths_ex_med" - Number of collections in 12 months excluding medical collections
- "acc_now_delinq" - The number of accounts on which the borrower is now delinquent.
- "acc_open_past_24mths" - Number of trades opened in past 24 months.
- "chargeoff_within_12_mths" - Number of charge-offs within 12 months
- "delinq_amnt" - The past-due amount owed for the accounts on which the borrower is now delinquent.
- "pub_rec_bankruptcies" - Number of public record bankruptcies
- "tax_liens" - Number of tax liens

Annual_inc was one of the first variables we looked at. Let's go ahead and impute the mean for this variable and move on to the next.

```
ann_inc_mean = mean(lending_data_vars_rem$annual_inc, na.rm = TRUE)
lending_data_vars_rem$annual_inc[
  is.na(lending_data_vars_rem$annual_inc)] = ann_inc_mean
```

Next, let's take a look at *dti*.

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean DTI" =
              mean(dti, na.rm=TRUE),
            "SD DTI" =
              sd(dti, na.rm=TRUE),
            "Max DTI" =
              max(dti, na.rm=TRUE),
            "Total NA Values" =
              sum(is.na(dti)))
```

```
## # A tibble: 2 x 5
##   loan_status_final `Mean DTI` `SD DTI` `Max DTI` `Total NA Values`
##   <chr>                  <dbl>    <dbl>     <dbl>              <int>
## 1 Default                 20.1     11.3       999                 62
## 2 Not Default             17.8     10.8       999                250
```

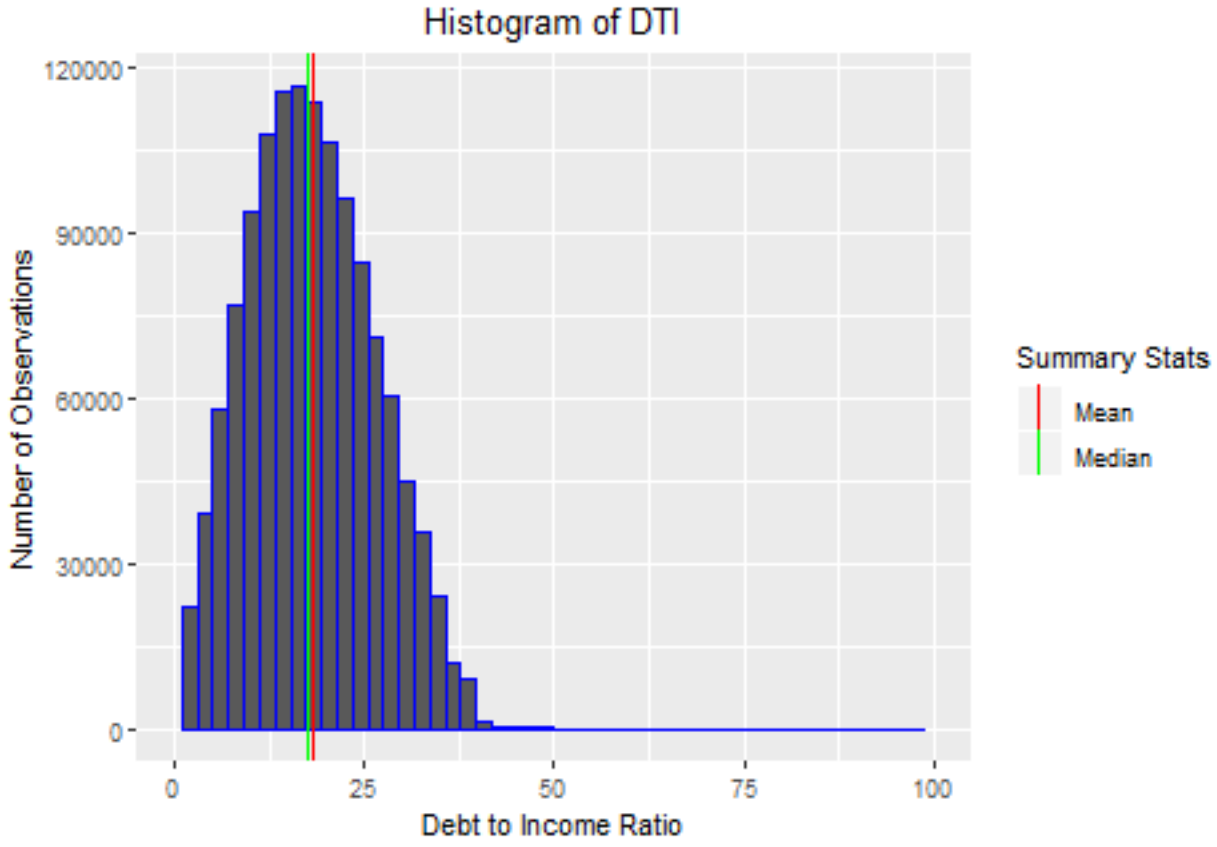
Looking at the summary statistics between groups, we can see that those who have not defaulted on their loan have a lower debt to income ratio. I believe that this variable will be useful for predicting loan default as it gives us a great statistic for judging how much debt they have to pay off compared to their income. Let's take a look at the histogram for this variable for possible imputation values. Note that the maximum of this variable is 999. Under the hood the minimum is actually -1. I believe that these values are outliers and should be ignored. We will take care of outliers later on.

```
dti_mean = mean(lending_data_vars_rem$dti,
                na.rm = TRUE)
dti_median = median(lending_data_vars_rem$dti,
                    na.rm = TRUE)
```

```
ggplot(lending_data_vars_rem, aes(x = dti)) +
  geom_histogram(stat = "bin", bins = 50, color = "blue") +
  xlim(0, 100) +
  geom_vline(aes(xintercept = dti_mean,
             color = "Mean")) +
  geom_vline(aes(xintercept = dti_median,
             color = "Median")) +
  xlab("Debt to Income Ratio") +
  ylab("Number of Observations") +
  ggtitle("Histogram of DTI") +
  theme(plot.title = element_text(hjust = 0.50)) +
  scale_color_manual(name = "Summary Stats",
                     labels = c("Mean", "Median"),
                     values = c("red", "green"))
```

```
## Warning: Removed 790 rows containing non-finite values (stat_bin).
```

```
## Warning: Removed 2 rows containing missing values (geom_bar).
```



Looking at the distribution, let's go ahead and fill the missing values with the median as the data is more centered around the median.

```
lending_data_vars_rem$dti[
  is.na(lending_data_vars_rem$dti)] = dti_median
```

Next, let's take a look at *delinq_2yrs*.

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean # Delinquencies" =
              mean(delinq_2yrs, na.rm=TRUE),
            "SD # Delinquencies" =
              sd(delinq_2yrs, na.rm=TRUE),
            "Max # Delinquencies" =
              max(delinq_2yrs, na.rm=TRUE),
            "Total NA Values" =
              sum(is.na(delinq_2yrs)))
```

```
## # A tibble: 2 x 5
##   loan_status_fin~ `Mean # Delinqu~ `SD # Delinquen~ `Max # Delinque~
##   <chr>                      <dbl>            <dbl>             <dbl>
## 1 Default                    0.352            0.939                27
## 2 Not Default                0.309            0.860                39
## # ... with 1 more variable: `Total NA Values` <int>
```

Looking at the summary statistics, there is not much of a difference between groups. However, there is some difference and the number of NA values is small at only 29 values. Let's go ahead and fill the missing values with the mean of this variable.

```
delinq_2yrs_mean = mean(
  lending_data_vars_rem$delinq_2yrs,
  na.rm = TRUE)

lending_data_vars_rem$delinq_2yrs[
  is.na(lending_data_vars_rem$delinq_2yrs)] = delinq_2yrs_mean
```

Next, let's take a look at *inq_last_6mths*.

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean # Inquiries" =
              mean(inq_last_6mths, na.rm=TRUE),
            "SD # Inquiries" =
              sd(inq_last_6mths, na.rm=TRUE),
            "Max # Inquiries" =
              max(inq_last_6mths, na.rm=TRUE),
            "Total NA Values" =
              sum(is.na(inq_last_6mths)))
```

```
## # A tibble: 2 x 5
##   loan_status_fin~ `Mean # Inquiri~ `SD # Inquiries` `Max # Inquirie~
##   <chr>                      <dbl>            <dbl>             <dbl>
## 1 Default                    0.793            1.05                 33
## 2 Not Default                0.635            0.938                31
## # ... with 1 more variable: `Total NA Values` <int>
```

Looking at the summary statistics, there is somewhat of a difference between groups. The number of NA values is also small. Let's go ahead and fill the missing values with the mean of this variable.

```
inq_last_6mths_mean = mean(
  lending_data_vars_rem$inq_last_6mths,
  na.rm = TRUE)

lending_data_vars_rem$inq_last_6mths[
  is.na(lending_data_vars_rem$inq_last_6mths)] = inq_last_6mths_mean
```

Next, let's take a look at *open_acc*. Recall that I created a variable indicating the total number of accounts a borrower has/had. Let's go ahead and remove this variable to avoid high collinearity issues.

Next, let's take a look at *pub_rec*.

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean # Public Records" =
              mean(pub_rec, na.rm=TRUE),
            "SD # Public Records" =
              sd(pub_rec, na.rm=TRUE),
            "Max # Public Records" =
              max(pub_rec, na.rm=TRUE),
            "Total NA Values" =
              sum(is.na(pub_rec)))
```

```
## # A tibble: 2 x 5
##   loan_status_fin~ `Mean # Public ~ `SD # Public Re~ `Max # Public R~
##   <chr>                       <dbl>            <dbl>            <dbl>
## 1 Default                     0.247            0.658               86
## 2 Not Default                 0.207            0.587               63
## # ... with 1 more variable: `Total NA Values` <int>
```

Looking at the summary statistics, there is somewhat of a difference between groups. The number of NA values is also small. Since this variable indicates the number of public derogatory records for the borrower, this may be useful for predicting loan default as those who have defaulted on their loan have a higher average and maximum. Let's go ahead and fill the missing values with the mean of this variable.

```
pub_rec_mean = mean(
  lending_data_vars_rem$pub_rec,
  na.rm = TRUE)

lending_data_vars_rem$pub_rec[
  is.na(lending_data_vars_rem$pub_rec)] = pub_rec_mean
```

Next, let's take a look at *revol_util*.

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean Revolving Credit Utilization" =
              mean(revol_util, na.rm=TRUE),
            "SD Revolving Credit Utilization" =
              sd(revol_util, na.rm=TRUE),
            "Max Revolving Credit Utilization" =
              max(revol_util, na.rm=TRUE),
            "Total NA Values" =
              sum(is.na(revol_util)))
```

```
## # A tibble: 2 x 5
##   loan_status_fin~ `Mean Revolving~ `SD Revolving C~ `Max Revolving ~
##   <chr>                       <dbl>            <dbl>            <dbl>
## 1 Default                      54.9             23.8             367.
## 2 Not Default                  51.2             24.6             892.
## # ... with 1 more variable: `Total NA Values` <int>
```

Looking at this variable, the difference between groups is very small. While I do believe this variable could be useful, since the difference between groups is small I do not believe it will benefit our model much. I will be removing this variable.

Next, let's take a look at *total_acc*. Recall that I created a variable indicating the total number of accounts a borrower has/had. Let's go ahead and remove this variable to avoid high collinearity issues.

Next, let's take a look at *collections_12_mths_ex_med*.

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean Coll. Last 12 Months" =
              mean(collections_12_mths_ex_med, na.rm=TRUE),
            "SD Coll. Last 12 Months" =
              sd(collections_12_mths_ex_med, na.rm=TRUE),
            "Max Coll. Last 12 Months" =
              max(collections_12_mths_ex_med, na.rm=TRUE),
            "Total NA Values" =
              sum(is.na(collections_12_mths_ex_med)))
```

```
## # A tibble: 2 x 5
##   loan_status_fin~ `Mean Coll. Las~ `SD Coll. Last ~ `Max Coll. Last~
##   <chr>                       <dbl>            <dbl>            <dbl>
## 1 Default                    0.0216            0.161                9
## 2 Not Default                0.0158            0.142               20
## # ... with 1 more variable: `Total NA Values` <int>
```

Looking at this variable, the difference between groups is very small. While I do believe this variable could be useful, since the difference between groups is small I do not believe it will benefit our model much. I will be removing this variable.

Next, let's take a look at *acc_now_delinq*.

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean # Accs. Delinquent" =
              mean(acc_now_delinq, na.rm=TRUE),
            "SD # Accs. Delinquent" =
              sd(acc_now_delinq, na.rm=TRUE),
            "Max # Accs. Delinquent" =
              max(acc_now_delinq, na.rm=TRUE),
            "Total NA Values" =
              sum(is.na(acc_now_delinq)))
```

```
## # A tibble: 2 x 5
##   loan_status_fin~ `Mean # Accs. D~ `SD # Accs. Del~ `Max # Accs. De~
##   <chr>                       <dbl>            <dbl>            <dbl>
## 1 Default                   0.00569           0.0823                6
## 2 Not Default               0.00491           0.0761               14
## # ... with 1 more variable: `Total NA Values` <int>
```

Looking at this variable, the difference between groups is very, very small. While I do believe this variable could be useful, since the difference between groups is so small I do not believe it will benefit our model much. I will be removing this variable.

Next, let's take a look at *acc_open_past_24mths*. Recall that I created a variable indicating the total number of accounts a borrower has/had. Let's go ahead and remove this variable to avoid high collinearity issues.

Next, let's take a look at *chargeoff_within_12_mths*.

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean # Charge-Offs within 12 Mos" =
              mean(chargeoff_within_12_mths, na.rm=TRUE),
            "SD # Charge-Offs within 12 Mos" =
              sd(chargeoff_within_12_mths, na.rm=TRUE),
            "Max # Charge-Offs within 12 Mos" =
              max(chargeoff_within_12_mths, na.rm=TRUE),
            "Total NA Values" =
              sum(is.na(chargeoff_within_12_mths)))
```

```
## # A tibble: 2 x 5
##   loan_status_fin~ `Mean # Charge-~ `SD # Charge-Of~ `Max # Charge-O~
##   <chr>                      <dbl>            <dbl>            <dbl>
## 1 Default                  0.00972            0.113                8
## 2 Not Default              0.00889            0.109               10
## # ... with 1 more variable: `Total NA Values` <int>
```

Looking at this variable, the difference between groups is very, very small. While I do believe this variable could be useful, since the difference between groups is so small I do not believe it will benefit our model much. I will be removing this variable.

Next, let's take a look at *delinq_amnt*.

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean Delinquent Amount" =
              mean(delinq_amnt, na.rm=TRUE),
            "SD Delinquent Amount" =
              sd(delinq_amnt, na.rm=TRUE),
            "Max Delinquent Amount" =
              max(delinq_amnt, na.rm=TRUE),
            "Total NA Values" =
              sum(is.na(delinq_amnt)))
```

```
## # A tibble: 2 x 5
##   loan_status_fin~ `Mean Delinquen~ `SD Delinquent ~ `Max Delinquent~
##   <chr>                      <dbl>            <dbl>            <dbl>
## 1 Default                     19.6             989.           249925
## 2 Not Default                 13.9             767.           185408
## # ... with 1 more variable: `Total NA Values` <int>
```

Looking at the summary statistics, we can see that those who have defaulted on their loan on average have a higher average deliquent amount. The standard deviation is also much higher for those who have defaulted. Let's go ahead and fill the missing values with the mean of this variable.

```
delinq_amnt_mean = mean(
  lending_data_vars_rem$delinq_amnt,
  na.rm = TRUE)

lending_data_vars_rem$delinq_amnt[
  is.na(lending_data_vars_rem$delinq_amnt)] = delinq_amnt_mean
```

Next, let's take a look at *pub_rec_bankruptcies*.

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean # Pub. Rec. Bankruptcies" =
              mean(pub_rec_bankruptcies, na.rm=TRUE),
            "SD # Pub. Rec. Bankruptcies" =
              sd(pub_rec_bankruptcies, na.rm=TRUE),
            "Max # Pub. Rec. Bankruptcies" =
              max(pub_rec_bankruptcies, na.rm=TRUE),
            "Total NA Values" =
              sum(is.na(pub_rec_bankruptcies)))
```

```
## # A tibble: 2 x 5
##   loan_status_fin~ `Mean # Pub. Re~ `SD # Pub. Rec.~ `Max # Pub. Rec~
##   <chr>                       <dbl>            <dbl>            <dbl>
## 1 Default                     0.154            0.405               11
## 2 Not Default                 0.130            0.371               12
## # ... with 1 more variable: `Total NA Values` <int>
```

Looking at the summary statistics, there is not much of a difference between groups. However, it can be seen that those who have defaulted on their loan have a higher average of public record bankruptcies. Since this variable can be useful for predicting loan default (those who have declared bankruptcy are typically unable to pay), I will be keeping it and filling the NA values with the mean.

```
pub_rec_bankruptcies_mean = mean(
  lending_data_vars_rem$pub_rec_bankruptcies,
  na.rm = TRUE)

lending_data_vars_rem$pub_rec_bankruptcies[
  is.na(lending_data_vars_rem$pub_rec_bankruptcies)] = pub_rec_bankruptcies_mean
```

Lastly, let's take a look at *tax_liens*.

```
lending_data_vars_rem %>% group_by(loan_status_final) %>%
  summarise("Mean # Tax Liens" =
              mean(tax_liens, na.rm=TRUE),
            "SD # Tax Liens" =
              sd(tax_liens, na.rm=TRUE),
            "Max # Tax Liens" =
              max(tax_liens, na.rm=TRUE),
            "Total NA Values" =
              sum(is.na(tax_liens)))
```

```
## # A tibble: 2 x 5
```

```
##   loan_status_fin~ `Mean # Tax Lie~ `SD # Tax Liens` `Max # Tax Lien~
##   <chr>                      <dbl>            <dbl>            <dbl>
## 1 Default                   0.0599            0.451               85
## 2 Not Default               0.0499            0.384               63
## # ... with 1 more variable: `Total NA Values` <int>
```

Looking at the summary statistics, there is not much of a difference between groups. However, it can be seen that those who have defaulted on their loan have a higher average of tax liens. Also, those who have defaulted on their loan have a higher max number of tax liens. Since this variable can be useful for predicting loan default, I will be keeping it and filling the NA values with the mean.

```
tax_liens_mean = mean(
  lending_data_vars_rem$tax_liens,
  na.rm = TRUE)

lending_data_vars_rem$tax_liens[
  is.na(lending_data_vars_rem$tax_liens)] = tax_liens_mean
```

We are now complete with looking at variables with greater than zero missing values. Let's go ahead and remove the variables stated above. I will also be removing an extra variable, *policy_code*, as it is meaningless. This variable relates to the loan policy, in which all of these loans are policy 1. Therefore, it will not be useful in our models.

```
remove_list_3 = c("open_acc",
        "revol_util",
        "total_acc",
        "collections_12_mths_ex_med",
        "acc_now_delinq",
        "acc_open_past_24mths",
        "chargeoff_within_12_mths",
        "policy_code")

lending_data_vars_rem = lending_data_vars_rem[, !(names(lending_data_vars_rem)
                                                  %in% remove_list_3)]
total_vars_5 = ncol(lending_data_vars_rem) - 1
```

Overall, we have drastically reduced the number of features in our dataset from 144 to 35. We also now have a grand total of 0 variables with missing values.

In this section, we have cleaned and filled both character and numeric variables. We have removed variables that are not useful, and kept or engineered variables that will be of much use for our end goal of predicting loan default through the use of numerical and visual aid. We will now shift our focus to modeling our data and creating a useful prediction model. Before we move on, let's make a copy of our final data and name it to `lending_data_final` to confirm we are working with finalized data.

```
lending_data_final = lending_data_vars_rem

rm(list = setdiff(ls(), "lending_data_final"))

#vroom_write(lending_data_final,
#            "lending_data_final.csv",
#            delim = ",")#write to file to avoid large markdown files..
```

Let's move on to modeling our data.

Splitting Data

Since our target variable has two levels of `Default` and `Not Default`, I will be utilizing Stratified sampling in order to maintain similar percentages of each level in our train and test data set. This is so our train or test set is not imbalanced with either of the target variable levels. Let's go ahead and set a seed to maintain consistency, and then split the data. I will be splitting the data into a 70% train set and 30% test set.

```
set.seed(490)

lending_data_final[sapply(lending_data_final, is.character)] = lapply(
  lending_data_final[sapply(lending_data_final, is.character)],
  as.factor
)#coerce all character variables to factors..

lending_data_final$loan_status_final = relevel(
  lending_data_final$loan_status_final, ref = "Not Default"
)#this changes our reference level so our model is predicting Defaults.

train_index = createDataPartition(
  lending_data_final$loan_status_final,
  p = .70,
  list = FALSE)

lending_data_train = lending_data_final[train_index,]
lending_data_test = lending_data_final[-train_index,]

rm(train_index)
```

With our data split, we can onto standardizing our data. Recall I briefly mentioned dealing with outliers in the data. I will be standardizing our data to be in a smaller range for numeric variables (mean of zero and variance of one). Let's go ahead and do that.

```
num_columns = colnames(lending_data_test %>% select_if(is.numeric))
scales = build_scales(lending_data_train,
                      num_columns,
                      verbose = TRUE)
```

```
## [1] "build_scales: I will compute scale on  29 numeric columns."
## [1] "build_scales: it took me: 1.07s to compute scale for 29 numeric columns."
```

```
lending_data_train_scaled = fastScale(lending_data_train,
                                      scales = scales,
                                      verbose = TRUE)
```

```
## [1] "fastScale: I will scale 29 numeric columns."
## [1] "fastScale: it took me: 0.23s to scale 29 numeric columns."
```

```
lending_data_test_scaled = fastScale(lending_data_test,
                                      scales = scales,
                                      verbose = TRUE)
```

```
## [1] "fastScale: I will scale 29 numeric columns."
## [1] "fastScale: it took me: 0.09s to scale 29 numeric columns."
```

Logistic/Probit Regression

Our first of many algorithms we will use is Logistic Regression. We will be fitting a few different logistic regression models, but our first model will be with what I expect to be the most important variable which is `annual_inc`.

```
model_eval = function(model, data) {
  num_vars = length(model$coefficients) - 1
  predictions = predict(model,
                        data,
                        type = "response")
  label_pred = ifelse(predictions > .50,
                        "Default",
                        "Not Default")
  accuracy = mean(label_pred == data$loan_status_final)
  error = mean(label_pred != data$loan_status_final)
  data.frame(num_vars,
             accuracy,
             error)
}#simple model evaluation function
```

```
log_model_1 = glm(loan_status_final ~ annual_inc,
                  data = lending_data_train_scaled,
                  family = binomial)
```

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

```
log_mod1_eval = model_eval(log_model_1,
                           lending_data_train_scaled)
```

Our next model I will build is using the variable `int_rate`. I believe that this variable is also very useful for predicting loan default. Let's go ahead and build the model.

```
log_model_2 = glm(loan_status_final ~ int_rate,
                  data = lending_data_train_scaled,
                  family = binomial)

log_mod2_eval = model_eval(log_model_2,
                           lending_data_train_scaled)
```

Our last single variable model will be using the variable `loan_amnt`. I believe that this variable is as well very useful for predicting loan default, as discussed in the beginning of the report. Let's go ahead and build the model.

```
log_model_3 = glm(loan_status_final ~ loan_amnt,
                  data = lending_data_train_scaled,
                  family = binomial)

log_mod3_eval = model_eval(log_model_3,
                           lending_data_train_scaled)
```

Now, let's go ahead and create a model that contains all three variables `annual_inc`, `int_rate`, and `loan_amnt`. Since these were the three most important features I had originally discussed, I believe that this model will be great for predicting loan defaults. Let's create this model.

```
log_model_4 = glm(loan_status_final ~ annual_inc + int_rate + loan_amnt,
                  data = lending_data_train_scaled,
                  family = binomial)
```

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

```
log_mod4_eval = model_eval(log_model_4,
                           lending_data_train_scaled)
```

Our last model we will build will introduce some categorical variables. I will be including the three features above, as well as the variables `grade`, and `term`. I chose these categorical variables as I believed they both contain valuable information on predicting loan default. Let's go ahead and build the model.

```
log_model_5 = glm(loan_status_final ~ annual_inc +
                    int_rate + loan_amnt +
                    term + grade,
                  data = lending_data_train_scaled,
                  family = binomial)
```

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

```
log_mod5_eval = model_eval(log_model_5,
                           lending_data_train_scaled)
```

So for all five of these models, I used the following variables:

- *annual_inc*
- *int_rate*
- *loan_amnt*
- *term*
- *grade*

These relationships make sense as I believe that these variables are all some of the most important variables for predicting loan default.

Annual income is directly tied to being able to payoff loans.

Interest rates can show underlying risk, and therefore can those with higher rates have a higher chance of default.

Those with a higher loan amount will typically have more trouble paying it off, leading to default.

Terms were shown previously that those with shorter terms have a lower default rate. Therefore, lower term means less risk of default.

Grades are the grade of the loan, and it was shown that the worse grade the higher chance of default.

Now, let's create a table that shows us the accuracy rate as well as the number of variables in each model.

```
log_models_df = data.frame("Model 1" =
                             c("# Vars" = log_mod1_eval$num_vars,
                               "Accuracy" = log_mod1_eval$accuracy),
                           "Model 2" =
                             c("# Vars" = log_mod2_eval$num_vars,
                               "Accuracy" = log_mod2_eval$accuracy),
                           "Model 3" =
                             c("# Vars" = log_mod3_eval$num_vars,
                               "Accuracy" = log_mod3_eval$accuracy),
                           "Model 4" =
                             c("# Vars" = log_mod4_eval$num_vars,
                               "Accuracy" = log_mod4_eval$accuracy),
                           "Model 5" =
                             c("# Vars" = log_mod5_eval$num_vars,
                               "Accuracy" = log_mod5_eval$accuracy)
)
kable(log_models_df)
```

	Model.1	Model.2	Model.3	Model.4	Model.5
# Vars	1.0000000	1.0000000	1.0000000	3.0000000	10.0000000
Accuracy	0.7991046	0.7976972	0.7991046	0.7985644	0.7995409

Looking at the table above, we can see that all models perform very similarly. However, the best model when evaluated on the training data is Model 5. This model included variables *annual_inc*, *int_rate*, *loan_amnt*, *term*, and *grade*. So, in this case our best model was the one with all variables. Let's shift our focus to this best model now.

Logistic Regression - Best Model

Now that we have chosen our best model to be model 5, let's get a better intuition of this model. First, let's take a quick look at the summary and then let's go through each X and it's effect on Y.

```
summary(log_model_5)
```

```
##
## Call:
## glm(formula = loan_status_final ~ annual_inc + int_rate + loan_amnt +
##     term + grade, family = binomial, data = lending_data_train_scaled)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4272  -0.7040  -0.5337  -0.3444   8.4904
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -2.469348   0.014870 -166.06   <2e-16 ***
## annual_inc    -0.232439   0.005245  -44.32   <2e-16 ***
## int_rate       0.187621   0.008246   22.75   <2e-16 ***
## loan_amnt      0.079696   0.003317   24.03   <2e-16 ***
## term60 months  0.376250   0.006806   55.28   <2e-16 ***
## gradeB         0.657530   0.013465   48.83   <2e-16 ***
## gradeC         1.072974   0.016635   64.50   <2e-16 ***
```

```
## gradeD         1.287887   0.021933   58.72   <2e-16 ***
## gradeE         1.414849   0.027677   51.12   <2e-16 ***
## gradeF         1.486036   0.035263   42.14   <2e-16 ***
## gradeG         1.528498   0.044599   34.27   <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: 917473  on 914470  degrees of freedom
## Residual deviance: 846673  on 914460  degrees of freedom
## AIC: 846695
##
## Number of Fisher Scoring iterations: 5
```

Each coefficient can be interpreted in terms of log odds. Log odds is the logarithm of the odds $\frac{p}{1-p}$ where p is probability. We can obtain the odds by taking the exponential of this value. We can then directly obtain the probability by this formula, where O is the odds.

$\frac{O}{1-O}$

Intercept - If the annual income, interest rate, and loan amount is all zero and the borrower has a term of 36 months and a loan grade of A then the log odds is -2.4693477.

Annual_inc - If the annual income of a borrower increases by 1 unit, then the expected change in log odds is -0.2324389.

Int_rate - If the interest rate of a borrower increases by 1 unit, then the expected change in log odds is 0.187621.

Loan_amnt - If the loan amount of a borrower increases by 1 unit, then the expected change in log odds is 0.0796956.

Term60 Months - If the loan is a 60 month term, then the log odds ratio between the 60 month and 36 month groups is 0.3762505.

GradeB - If the loan is of grade B, then the log odds ratio between the B and A loan grade groups is 0.6575302.

GradeC - If the loan is of grade B, then the log odds ratio between the C and A loan grade groups is 1.0729743.

GradeD - If the loan is of grade B, then the log odds ratio between the D and A loan grade groups is 1.2878874.

GradeE - If the loan is of grade B, then the log odds ratio between the E and A loan grade groups is 1.4148495.

GradeF - If the loan is of grade B, then the log odds ratio between the F and A loan grade groups is 1.4860359.

GradeG - If the loan is of grade B, then the log odds ratio between the G and A loan grade groups is 1.5284984.

Looking back at the summary of the model, we can see that all variables are significantly different from zero at the "0" level. This assumes that the probability the coefficients of these variables are zero is zero itself.

Since we are doing logistic regression, instead of using an F statistic for evaluating the model validity as a whole, we actually use the Likelihood Ratio Test. The Likelihood Ratio Test (LRT) follows a χ^2 distribution with some degrees of freedom. The LRT assumes a null hypothesis that all coefficients are equal to zero,

and the alternative is that at least of of the coefficients is not equal to zero. If we reject the null, that means that our model has at least one coefficient different from zero and therefore our model is valid as a whole.

Let's go ahead and perform the test.

```
base_log_mod = glm(loan_status_final ~ 1,
                   data = lending_data_train_scaled,
                   family = binomial)
anova_ref = anova(base_log_mod, log_model_5, test = "LRT")
anova(base_log_mod, log_model_5, test = "LRT")
```

```
## Analysis of Deviance Table
##
## Model 1: loan_status_final ~ 1
## Model 2: loan_status_final ~ annual_inc + int_rate + loan_amnt + term +
##     grade
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1    914470     917473
## 2    914460     846673 10    70800 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We can see that we get a χ^2 statistic of 70799.5497349 on 10 degrees of freedom. This yields us a p-value of approximately zero, so we therefore reject the null hypothesis that all coefficients is equal to zero. So what we have learned from this test and the χ^2 statistic is that this model is valid as a whole.

Logistic Regression - Best Model - True Pos & False Pos Rates

Next, let's go ahead and take a look at the true positive and false positive rates. For this, we will need the confusion matrix of the model.

```
log_mod5_pred = predict(log_model_5,
                    lending_data_train_scaled,
                    type = "response")

log_mod5_label_pred = ifelse(log_mod5_pred > .50,
                      "Default",
                      "Not Default")

log_mod5_conf_mtx = confusionMatrix(table(
  predicted = log_mod5_label_pred,
  actual = lending_data_train_scaled$loan_status_final)[2:1, 1:2],
  positive = "Default")
```

Now that we have the confusion matrix, we can obtain the true positive and false positive rates. Recall the formula for the true positive rate is the number of true positives divided by the total number of positives in the population. Let's get the true positive rate now.

```
log_mod5_tp_rate = log_mod5_conf_mtx$log_mod5_conf_mtx$byClass[1]
log_mod5_tp_rate
```

```
## NULL
```

So, the true positive rate for this model is .022094. In other words, we are able to correctly predict that someone is going to "Default" on the loan at a very low rate. Since our goal is to predict loan defaults, this is not where we want to be. Let's check our false positive rate.

The false positive rate formula is the total number of incorrect positive predictions divided by the total number of negatives in the population.

```
log_mod5_fp_rate = 1 - log_mod5_conf_mtx$log_mod5_conf_mtx$byClass[2]
log_mod5_fp_rate
```

```
## numeric(0)
```

So, the false positive rate for this model is .005007. In other words, the rate that we incorrectly predict the positive label is very low. Since we are worried about predicting loan defaults, we need to focus more on the true positive rate. We would like to maximize this rate with our model as we are more worried about correctly classifying loan defaults.

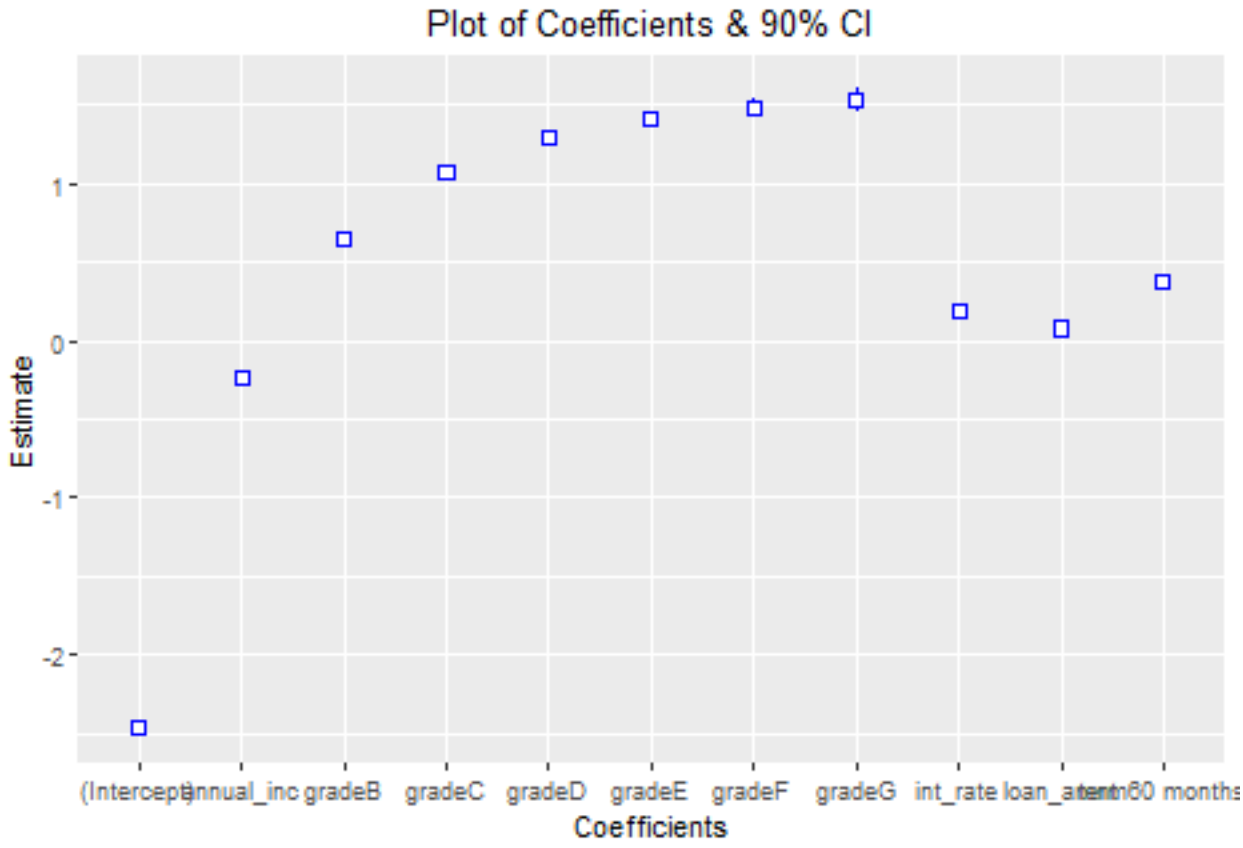
Overall, these rates tell me that my regression is predicting the majority class. Since the true positive is near 0, my model is almost always correctly predicting the negative label. However, the model is overpredicting the negative label and therefore causing a low true positive rate. Since we want to correctly predict loan defaults, this model is not performing how I would like it to.

Logistic Regression - Best Model - Coefficients & Std. Errors

With true and false positive rates aside, let's dive deeper into the coefficients. We will be plotting the estimated coefficients and their standard errors.

Since the test statistics of the estimated coefficients follow a normal distribution, we can compute confidence intervals for the coefficient using the normal distribution. I will be computing 90% confidence intervals for each coefficient and plotting it.

```
z = qnorm(.05, lower.tail = FALSE)
coef_confint = data.frame(
  "Coefficient" = rownames(summary(log_model_5)$coefficients),
  "Estimate" = summary(log_model_5)$coefficients[, 1],
  "Lower Bound" = summary(log_model_5)$coefficients[, 1] -
    z*summary(log_model_5)$coefficients[, 2],
  "Upper Bound" = summary(log_model_5)$coefficients[, 1] +
    z*summary(log_model_5)$coefficients[, 2]
)

ggplot(coef_confint, aes(Coefficient, Estimate)) +
  geom_point() +
  geom_pointrange(aes(ymin = Lower.Bound, ymax = Upper.Bound),
                  fill = "white", shape = 22, color = "blue") +
  labs(x = "Coefficients", y = "Estimate",
       title = "Plot of Coefficients & 90% CI") +
  theme(plot.title = element_text(hjust = 0.5))
```



Looking ast the plot, we can see that the only variable that has visible confidence interval lines is "gradeG". However, the probalility of this variable having a coefficient equal to zero is approximately zero. We can conclude that all variables are significant at the "0" level.

Logistic Regression - Best Model - Probit

With coefficient visualization complete, let's move forward. We will now be re-evaluating our model using probit regression. We will then be comparing the accuracy of this new model with our original model which used logit regression.

```
log_model_5_probit = glm(loan_status_final ~ annual_inc +
                     int_rate + loan_amnt +
                     term + grade,
                   data = lending_data_train_scaled,
                   family = binomial(link = "probit"))
```

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

```
log_model_5_probit_eval = model_eval(log_model_5_probit,
                                     lending_data_train_scaled)
```

```
log_acc_compare_df = data.frame(
  "Logistic Regression" = c("Accuracy" = log_mod5_eval$accuracy),
  "Probit Regression" = c("Accuracy" = log_model_5_probit_eval$accuracy)
)
kable(log_acc_compare_df)
```

	Logistic.Regression	Probit.Regression
Accuracy	0.7995409	0.7995114

Comparing our accuracy rates above, we can see that logistic regression slightly outperforms probit regression. Therefore, by the use of accuracy rate only, I can say that logistic regression performs better for this model.

This concludes the section on logistic regression.

K Nearest Neighbors

For this section, we will be creating a K-Nearest Neighbors classification model. We will be training a few models with different K's, and then choosing the model that performs the best on our test data. After so, we will predict results, take a look at the accuracy and compare to our results with logistic regression. I will be using the same variables that I had used in the best logistic regression model.

KNN - Model Training

For model building, I will be utilizing 10-fold cross-validation with a tune length of 20. In other words, I will be tuning on 20 different values of K using 10-fold cross-validation for each value of K. I will be choosing the best K by looking at the model with the lowest cross-validated area under the curve. The higher the AUC, the better the model. After the model is chosen, I will evaluate it on the test data to see how it performs on unseen data.

First, we will be getting a sample of the data to reduce our training and prediction times. I will be sampling 25% of the full data for use, and then splitting that into train and test sets. I will also be scaling that data.

```
set.seed(490)

sample_idx = createDataPartition(
  lending_data_final$loan_status_final,
  p = .25,
  list = FALSE)

lending_sampled = lending_data_final[sample_idx,]

trn_idx = createDataPartition(
  lending_sampled$loan_status_final,
  p = .70,
  list = FALSE)

lending_data_samp_trn = lending_sampled[trn_idx,]
lending_data_samp_tst = lending_sampled[-trn_idx,]

scales = build_scales(lending_data_train,
                      num_columns,
                      verbose = TRUE)
```

```
## [1] "build_scales: I will compute scale on  29 numeric columns."
## [1] "build_scales: it took me: 0.35s to compute scale for 29 numeric columns."
```

```
lending_data_samp_scaled_trn = fastScale(lending_data_samp_trn,
                                         scales = scales,
                                         verbose = TRUE)
```

```
## [1] "fastScale: I will scale 29 numeric columns."
## [1] "fastScale: it took me: 0.04s to scale 29 numeric columns."
```

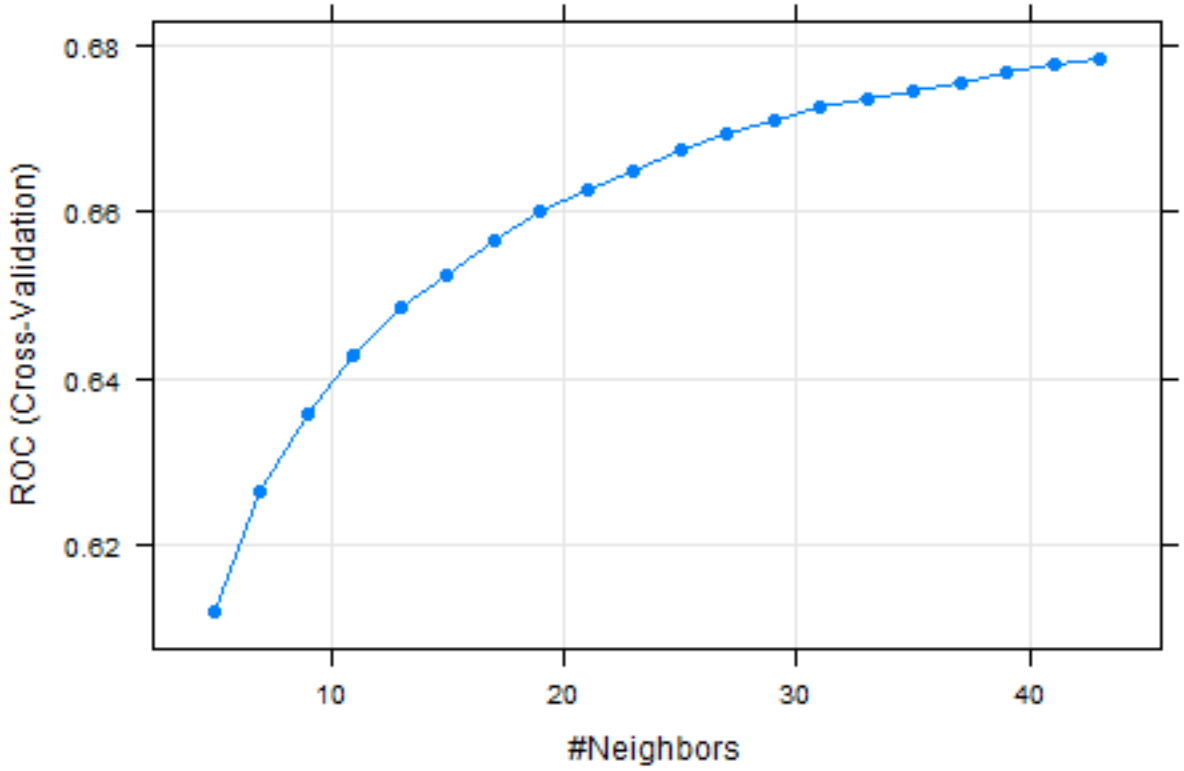
```
lending_data_samp_scaled_tst = fastScale(lending_data_samp_tst,
                                         scales = scales,
                                         verbose = TRUE)
```

```
## [1] "fastScale: I will scale 29 numeric columns."
## [1] "fastScale: it took me: 0.02s to scale 29 numeric columns."
```

```
set.seed(490)

levels(lending_data_samp_scaled_trn$loan_status_final
       )[levels(lending_data_samp_scaled_trn$loan_status_final
                )=="Not Default"] <- "NotDefault" #removes space in Not Default level name..
                                                  #knn cannot handle spaces in factor names..

# ctrl = trainControl(method = "cv", number = 10, classProbs = TRUE,
#                     summaryFunction = twoClassSummary,
#                     verboseIter = TRUE)
# cv_knn = train(loan_status_final ~ annual_inc +
#                int_rate + loan_amnt +
#                term + grade, data = lending_data_samp_scaled_trn,
#                method = "knn",
#                trControl = ctrl,
#                metric = "ROC",
#                tuneLength = 20)
#The above is trained. Using an RDS file to avoid lengthy retraining time.

cv_knn = readRDS("cv_knn.rds")
```

With the models trained, let's take a look at the cross-validation scores across all levels of K.

```
plot(cv_knn,
     pch = 16)
```




```
best_k = cv_knn$bestTune[1]
```

As we can see above, as we increase the number of neighbors (K), our cross-validated ROC increases. Our KNN with the highest cross-validated ROC is K = 43. Let's go ahead and use this model to predict on our test subset and get an idea on how it performs on unseen data.

```
lending_data_samp_scaled_tst_subset = lending_data_samp_scaled_tst %>%
   select(annual_inc, int_rate, loan_amnt, term, grade, loan_status_final)
levels(lending_data_samp_scaled_tst_subset$loan_status_final
       )[levels(lending_data_samp_scaled_tst_subset$loan_status_final
                )=="Not Default"] <- "NotDefault"

# knn_preds = predict(cv_knn, lending_data_samp_scaled_tst_subset)
#takes awhile to predict.. using saved object
knn_preds = readRDS("knn_preds.rds")

knn_acc = mean(knn_preds == lending_data_samp_scaled_tst_subset$loan_status_final)
knn_err = mean(knn_preds != lending_data_samp_scaled_tst_subset$loan_status_final)

knn_tst_results = data.frame("Accuracy" = c("K = 43" = knn_acc),
                             "Error Rate" = c("K = 43" = knn_err)
                            )

kable(knn_tst_results)
```

	Accuracy	Error.Rate
K = 43	0.7968217	0.2031783

Looking at the table above, we can see that with K=43 we get an accuracy of 0.7968217 and error rate of 0.2031783 on the test data. This is not the best result, but still good. Let's go ahead and compare this KNN model with our best logistic model built above.

KNN - Model Evaluation

```
lending_data_samp_scaled_full = rbind(
   lending_data_samp_scaled_trn,
   lending_data_samp_scaled_tst)
lending_data_samp_scaled_full_subset = lending_data_samp_scaled_full %>%
   select(annual_inc, int_rate, loan_amnt, term, grade, loan_status_final)
levels(lending_data_samp_scaled_full_subset$loan_status_final
       )[levels(lending_data_samp_scaled_full_subset$loan_status_final
                )=="Not Default"] <- "NotDefault"

# knn_preds_full = predict(cv_knn, lending_data_samp_scaled_full_subset)
#takes awhile to predict.. using saved object
knn_preds_full = readRDS("knn_preds_full.rds")

knn_acc_full = mean(knn_preds_full == lending_data_samp_scaled_full_subset$loan_status_final)
knn_err_full = mean(knn_preds_full != lending_data_samp_scaled_full_subset$loan_status_final)

knn_logistic_results = data.frame("Accuracy" = c("K = 43" = knn_acc_full,
                                                 "Logistic" = log_mod5_eval$accuracy),
                                  "Error Rate" = c("K = 43" = knn_err_full,
                                                   "Logistic" = 1 - log_mod5_eval$accuracy))

kable(knn_logistic_results)
```

	Accuracy	Error.Rate
K = 43	0.8005922	0.1994078
Logistic	0.7995409	0.2004591

When predicting on our full data, our KNN classification model with K = 43 actually slightly outperforms our best logistic regression model. It should be taken into account that the logistic regression model accuracy is from predictions made on training data, so that could be boosting it's accuracy somewhat. Overall, we can see that our KNN classification model is slightly better when comparing to logistic regression on full data.

This concludes the section on K Nearest Neighbors.

Ridge and Lasso Regressions

For this section, I will be fitting Ridge and Lasso regressions using the standardized data. Recall that the use of standardized data is to minimize effects on outliers in the data.

Fitting Ridge & Lasso Regressions and Utilizing Cross-Validation for Model Selection

```r
set.seed(490)

lending_data_train_scaled = lending_data_train_scaled %>% select(-sub_grade)
lending_data_test_scaled = lending_data_test_scaled %>% select(-sub_grade)

train_x = model.matrix(loan_status_final ~ ., data = lending_data_train_scaled)
train_y = lending_data_train_scaled$loan_status_final
lambdas = 10^seq(5, -10, length = 100)

# cv_ridge = cv.glmnet(train_x, train_y,
#                      alpha = 0, lambda = lambdas,
#                      type.measure = "auc",
#                      family = "binomial",
#                      parallel = TRUE)
cv_ridge = readRDS("cv_ridge.rds")#so we don't have to retrain again..

# cv_lasso = cv.glmnet(train_x, train_y,
#                      alpha = 1, lambda = lambdas,
#                      type.measure = "auc",
#                      family = "binomial",
#                      parallel = TRUE)
cv_lasso = readRDS("cv_lasso.rds")#so we don't have to retrain again..
```

With our models created using cross-validation, we will have fitted a large amount of models. Each of these models are created with a different value of lambda, and then evaluated using cross-validation to get an average area under the curve for each lambda. Area under the curve represents how well our model can distinguish between our two output groups, *Default* and *Not Default*. So, we will want to choose the lambda that has the highest area under the curve. However, to account for some error in our models we will be selecting the lambda that is within 1 standard error of our best lambda with the highest area under the curve. Let's go ahead and select that best lambda and display it. Note that we have fitted numerous models using both ridge & lasso regression, so we will be evaluating each of these algorithms in each part.

Selecting λ Within One Standard Error

```r
ridge_best_lambda = cv_ridge$lambda.1se

lasso_best_lambda = cv_lasso$lambda.1se
```

Our chosen lambda for ridge regression is 0.

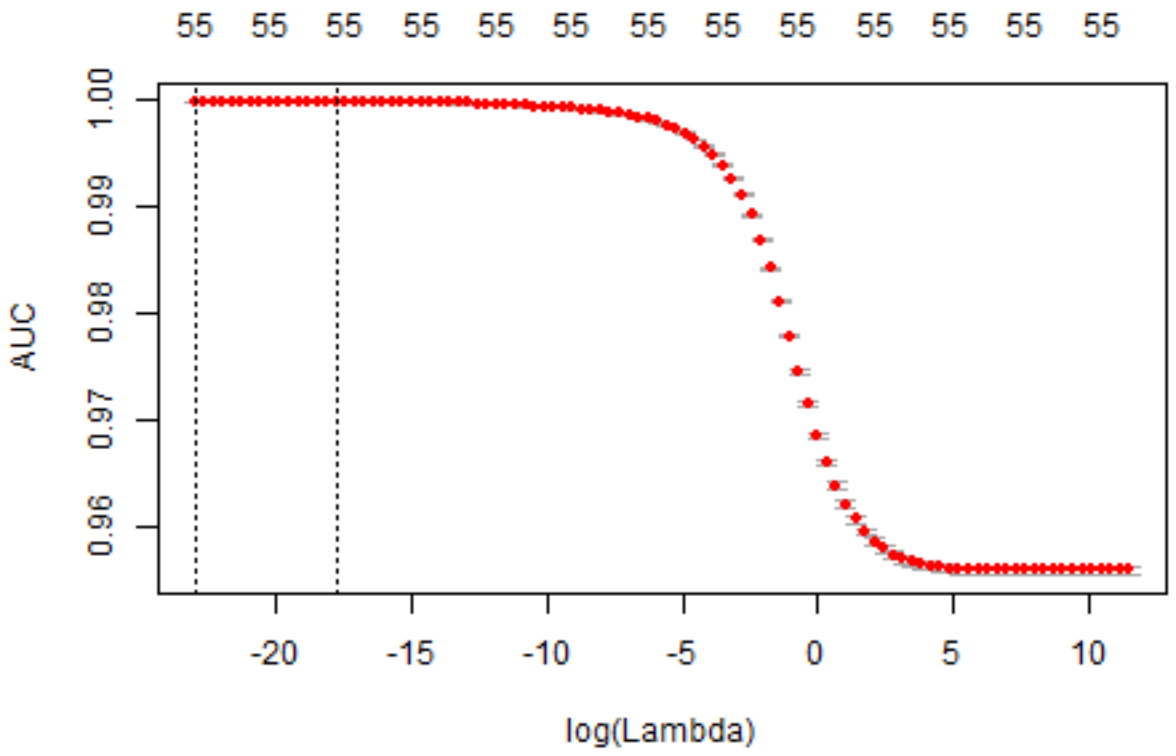
Our chosen lambda for lasso regression is 0.0000025.

Note that both of these numbers are very close to zero. With a lambda very close to zero, there is not much shrinkage happening to the coefficients. This is somewhat expected due to data standardization before hand, which itself shrinks the coefficients due to smaller ranges for variables.

Now, let's go ahead and take a look at a graph of our cross validation results. We will be plotting Area Under the Curve vs Log(Lambda) for both Ridge and LASSO regression. The plot will show how Log(Lambda) changes our Area Under the Curve results, and will also show where our chosen Log(Lambda) is.

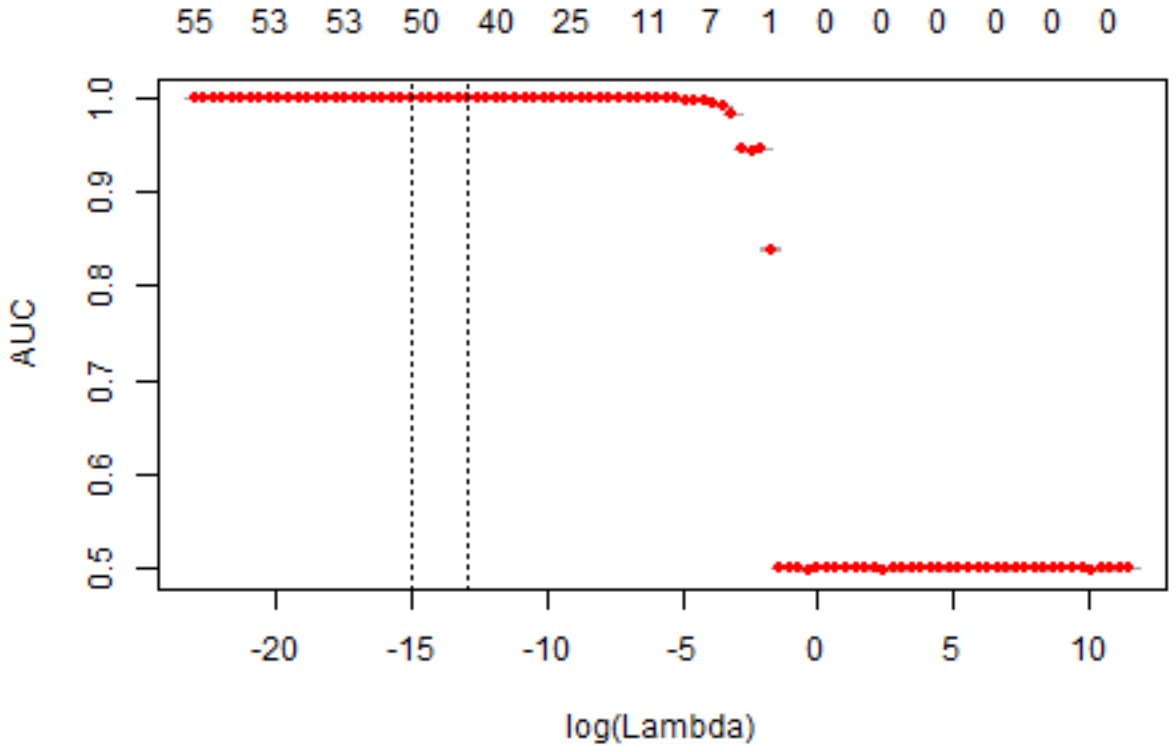
Displaying Cross-Validation Results

```r
plot(cv_ridge)
```



Looking at the plot above, we can see that as Log(Lambda) increases, our cross-validated Area Under the Curve decreases to an almost stable line. Our minimum log(Lambda) is around -25ish, and our log(Lambda) within one standard error of our minimum is around -18ish, which is what the veritical lines represent. Since our best lambda is so close to zero, we get an insight that standardizing our coefficients may not be as useful for building models.

```
plot(cv_lasso)
```

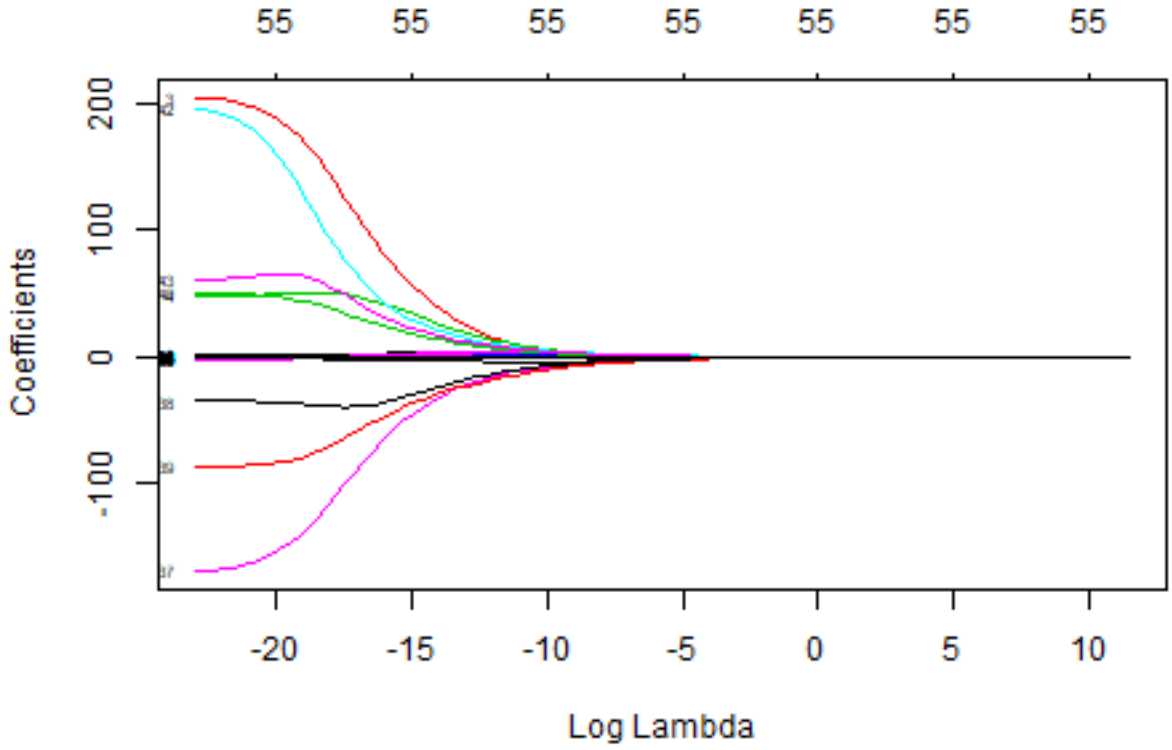


Looking at the plot above, we get a much more oddly shaped curve for Area Under the Curve vs. log(Lambda). We can see that our chosen log(Lambda) is at around -13ishish, with an AUC around .99. However, when log(Lambda) increases, our curve suddenly discontinues and flattens out at an AUC of about .50 at log(Lambda) of -2ish. We should also note that our chosen model LASSO model contains 44 variables, down from 55, which is very significant. Since LASSO regression performs variable selection through the shrinkage of coefficients to zero, we can see that our LASSO model is removing variables that are not helpful in predicting loan defaults.

Now, let's take a look at how our coefficients vary by a given Log(Lambda).

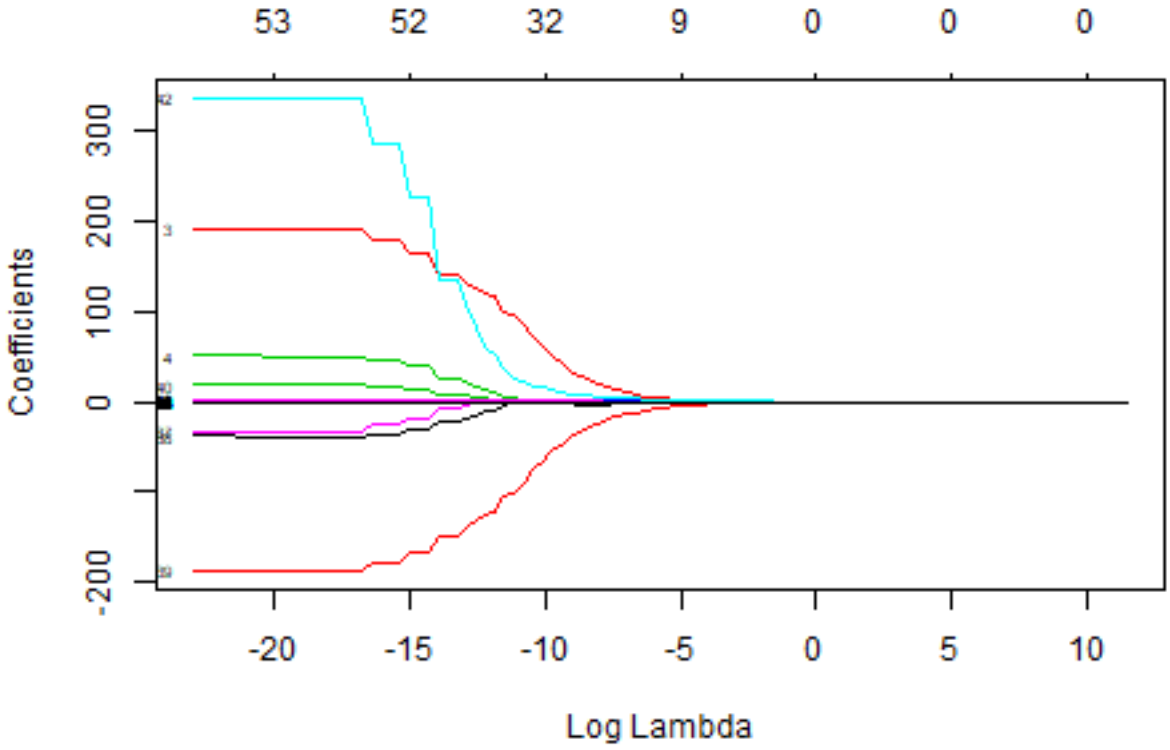
Displaying Coefficients by Log(Lambda)

```
plot(cv_ridge$glmnet.fit, "lambda", label=TRUE)
```



As we can see above, our coefficients for our ridge regression shrink towards zero as Log(Lambda) increases. This is what we expect as Lambda is a tuning parameter that when increased, shrinks coefficients towards zero. However, given our best lambda we know that shrunken coefficients are not best for our data.

```
plot(cv_lasso$glmnet.fit, "lambda", label=TRUE)
```



Looking at the plot above, our coefficients for ridge regression shrink towards zero as Log(Lambda) increases. This is what we expect as Lambda is a tuning parameter that when increased, shrinks coefficients towards zero. However, given our best lambda we know that shrunken coefficients are not best for our data. We should point out though for LASSO regression, as mentioned above, our chosen model only contains 44 variables. As Log(Lambda) increases, the number of variables ends up to be zero which is not very useful for predicting loan defaults.

Let's take a look at the coefficients we will be using for our best Ridge and LASSO regression model, with $\lambda = 0$ and $\lambda = 0.0000025$ respectively.

Coefficients of Chosen Best λ

```
kable(as.matrix(coef(cv_ridge, s = ridge_best_lambda)),
      col.names = "Chosen Coefficient by Best Lambda")
```

	Chosen Coefficient by Best Lambda
(Intercept)	70.0831576
(Intercept)	0.0000000
loan_amnt	2.6939947
funded_amnt	136.3237970
funded_amnt_inv	50.1532253
term60 months	0.4887488
int_rate	-0.0891829
installment	0.3141208
gradeB	0.4536277
gradeC	0.7688097

	Chosen Coefficient by Best Lambda
gradeD	0.8755355
gradeE	0.8265468
gradeF	0.3837312
gradeG	0.4960292
annual_inc	-0.2922892
verification_statusSource Verified	0.0124210
verification_statusVerified	-0.0402432
purposecredit_card	0.1820602
purposedebt_consolidation	0.2267836
purposeeducational	0.2578678
purposehome_improvement	0.4416899
purposehouse	0.3204971
purposemajor_purchase	-0.1729333
purposemedical	0.3127355
purposemoving	0.2410744
purposeother	0.4027772
purposerenewable_energy	0.9161694
purposesmall_business	-0.8120615
purposevacation	0.1124385
purposewedding	-0.4101720
dti	-0.0221644
delinq_2yrs	-0.0119433
inq_last_6mths	0.0219937
pub_rec	0.0951071
revol_bal	-0.0604293
out_prncp	-0.0131741
out_prncp_inv	-0.0129982
total_pymnt	-108.1748044
total_pymnt_inv	-39.0275395
total_rec_prncp	-68.0540556
total_rec_int	36.9316265
total_rec_late_fee	0.2954547
recoveries	86.0329083
collection_recovery_fee	53.5878946
last_pymnt_amnt	-1.3383929
bc_util	0.0305077
delinq_amnt	-0.0237897
percent_bc_gt_75	0.0255465
pub_rec_bankruptcies	-0.0297805
tax_liens	-0.0935700
state_regionsNortheast	0.0209662
state_regionsSoutheast	-0.1371139
state_regionsSouthwest	0.0086813
state_regionsWest	-0.1129260
credit_age	0.0454790
num_accs	0.0831041
hc_cl	-0.2116973

Recall that the coefficients above represent log-odds since our model is logistic. Overall, as expected with our chosen lambda, our models coefficients are not shrunken to close to zero. However, due to standardizing our data our coefficients are still very small. I believe since our data was standardized, this affected how much our coefficients needed to be standardized further, which is why our chosen lambda is near zero.

Now, let's take a look at our coefficients for our LASSO regression model. Recall that our best lambda chose a model with only 44 coefficients, so some of our coefficients are equal to zero.

```
kable(as.matrix(coef(cv_lasso, s = ridge_best_lambda)),
      col.names = "Chosen Coefficient by Best Lambda")
```

	Chosen Coefficient by Best Lambda
(Intercept)	134.5434843
(Intercept)	0.0000000
loan_amnt	2.2425399
funded_amnt	191.3504327
funded_amnt_inv	50.1273651
term60 months	-0.0936713
int_rate	-0.0262678
installment	-1.2367938
gradeB	0.4917988
gradeC	0.8450949
gradeD	1.0104370
gradeE	1.0217126
gradeF	0.4486670
gradeG	0.7300633
annual_inc	-0.2383592
verification_statusSource Verified	0.0534898
verification_statusVerified	0.0397004
purposecredit_card	0.3373859
purposedebt_consolidation	0.3780058
purposeeducational	-1.6215410
purposehome_improvement	0.6274657
purposehouse	0.7002768
purposemajor_purchase	-0.0474649
purposemedical	0.4175656
purposemoving	0.0370245
purposeother	0.5446065
purposerenewable_energy	1.2049273
purposesmall_business	-1.2753834
purposevacation	0.0547560
purposewedding	-0.5160894
dti	0.0016136
delinq_2yrs	-0.0081992
inq_last_6mths	0.0222754
pub_rec	0.1028004
revol_bal	-0.0758370
out_prncp	0.0000000
out_prncp_inv	0.0000000
total_pymnt	-33.0544257
total_pymnt_inv	-38.5484877
total_rec_prncp	-186.7855145
total_rec_int	17.9934768
total_rec_late_fee	0.2170879
recoveries	335.1373510
collection_recovery_fee	-0.1011354
last_pymnt_amnt	-0.9692113
bc_util	0.0277919

	Chosen Coefficient by Best Lambda
delinq_amnt	-0.0475488
percent_bc_gt_75	-0.0132257
pub_rec_bankruptcies	-0.0230544
tax_liens	-0.0902872
state_regionsNortheast	-0.0051101
state_regionsSoutheast	-0.1921681
state_regionsSouthwest	0.0168019
state_regionsWest	-0.1276982
credit_age	0.0406010
num_accs	0.0722284
hc_cl	-0.2742018

Same as with the Ridge regression, our coefficients represent log-odds. With our chosen lambda, there is not much shrinkage occuring on the coefficients. With our data being standardized before hands, our coefficients are still very close to zero still. I believe since our data was standardized, this affected how much our coefficients needed to be standardized further, which is why our chosen lambda is near zero.

With these chosen models and coefficients in mind, let's now predict on our test data and get an accuracy and error rate to see how our model performs on unseen test data.

Error & Accuracy Rate on Test Data

```
test_x = model.matrix(loan_status_final ~ ., data = lending_data_test_scaled)
test_y = lending_data_test_scaled$loan_status_final

ridge_test_predictions = predict(cv_ridge,
                                 newx = test_x, s = "lambda.1se",
                                 type = "class")
ridge_test_accuracy = mean(ridge_test_predictions == test_y)
ridge_test_error = mean(ridge_test_predictions != test_y)

lasso_test_predictions = predict(cv_lasso,
                                 newx = test_x, s = "lambda.1se",
                                 type = "class")
lasso_test_accuracy = mean(lasso_test_predictions == test_y)
lasso_test_error = mean(lasso_test_predictions != test_y)

model_test_df = data.frame("Ridge Regression Measures" = c(
                             "Accuracy on Test Data" = ridge_test_accuracy,
                             "Error on Test Data" = ridge_test_error),
                           "LASSO Regression Measures" = c(
                             "Accuracy on Test Data" = lasso_test_accuracy,
                             "Error on Test Data" = lasso_test_error))

kable(model_test_df)
```

	Ridge.Regression.Measures	LASSO.Regression.Measures
Accuracy on Test Data	0.999579	0.9994642
Error on Test Data	0.000421	0.0005358

Looking at our results above, our accuracy on the test data for Ridge regression is 0.999579. Our error rate on the test data for Ridge regression is 0.000421.

Our accuracy on the test data for LASSO regression is 0.9994642. Our error rate on the test data for LASSO regression is 0.0005358.

Our LASSO model slightly outperforms the Ridge regression model, while also utilizing fewer variables. This is important as decreasing the number of predictors in our models lowers our variance in the model and offers better results on unseen data, which is seen above.

Comparing Results Between All Models

Overall, both of these models are a significant improvement over our original logistic regression model, where we saw an accuracy rate of about .79 (on the train data, we will be re-evaluating this model on test data for this portion). However, for that model we only used a limited amount of variables. For this model, we utilized all variables. Let's predict on a Logistic model that utilizes all variables to get a better idea of performance. For this model, we will only consider lambda = 0 as this does not impose any coefficient shrinkage.

Below, we will consider the LASSO, Ridge, chosen Logistic from report 2, as well as a Logistic model with all variables and compare their results.

```
test_log_predictions = predict(cv_ridge,
                               newx = test_x, s = 0,
                               type = "class")

test_log_accuracy = mean(test_log_predictions == test_y)
test_log_error = mean(test_log_predictions != test_y)

log_mod5_eval = model_eval(log_model_5,
                           lending_data_test_scaled)

model_full_test_df = data.frame("Ridge" = c(
                            "Accuracy on Test Data" = ridge_test_accuracy,
                            "Error on Test Data" = ridge_test_error),
                              "LASSO" = c(
                            "Accuracy on Test Data" = lasso_test_accuracy,
                            "Error on Test Data" = lasso_test_error),
                              "Logistic" = c(
                            "Accuracy on Test Data" = test_log_accuracy,
                            "Error on Test Data" = test_log_error),
                              "Original Logistic" = c(
                            "Accuracy on Test Data" = log_mod5_eval$accuracy,
                            "Error on Test Data" = log_mod5_eval$error))#include log5 model
kable(model_full_test_df)
```

	Ridge	LASSO	Logistic	Original.Logistic
Accuracy on Test Data	0.999579	0.9994642	0.9997448	0.7995642
Error on Test Data	0.000421	0.0005358	0.0002552	0.2004358

We can see above that our logistic regression model with full variables performs a small amount better than both the Ridge and LASSO regression models. This confirms my belief that the shrinkage is not helping our models very much, and therefore is not needed for building an accurate model. Our original logistic model built in report 2 is lagging behind, with an accuracy rate of .799. This is understandable as our original

model did not utilize all variables in our data.

Overall, LASSO and Ridge regression do not prove to have an advantage over a logistic model with all variables. I do not believe that shrinkage is of much use for our model building, as our data is standardized so shrinkage is already somewhat happening by the standardized variables.

Next, we will move on to a different type of model which is Decision Tree. This model offers similar model interpretation as linear regression, but with extra perks such as considering nonlinear relationships.

Decision Trees

Using Cross-Validation to Prune Decision Tree

Below, we will be building a Decision Tree model utilizing Cross-Validation in order to compare accuracy results on different tuning parameters for the decision tree. I will be utilizing 10-fold cross-validation, same as the above models.

```
set.seed(490)

#default_tree = tree(loan_status_final ~ ., data = lending_data_train_scaled)#create initial tree for
#further pruning and cross-validation
##This is commented out as model has already been trained.

# cv_tree = cv.tree(default_tree,
#                   FUN=prune.misclass,
#                   K=10)
#This is commented out as model has already been trained.

default_tree = readRDS("default_tree.rds")#so we don't have to retrain again..

cv_tree = readRDS("cv_tree.rds")#so we don't have to retrain again..
```

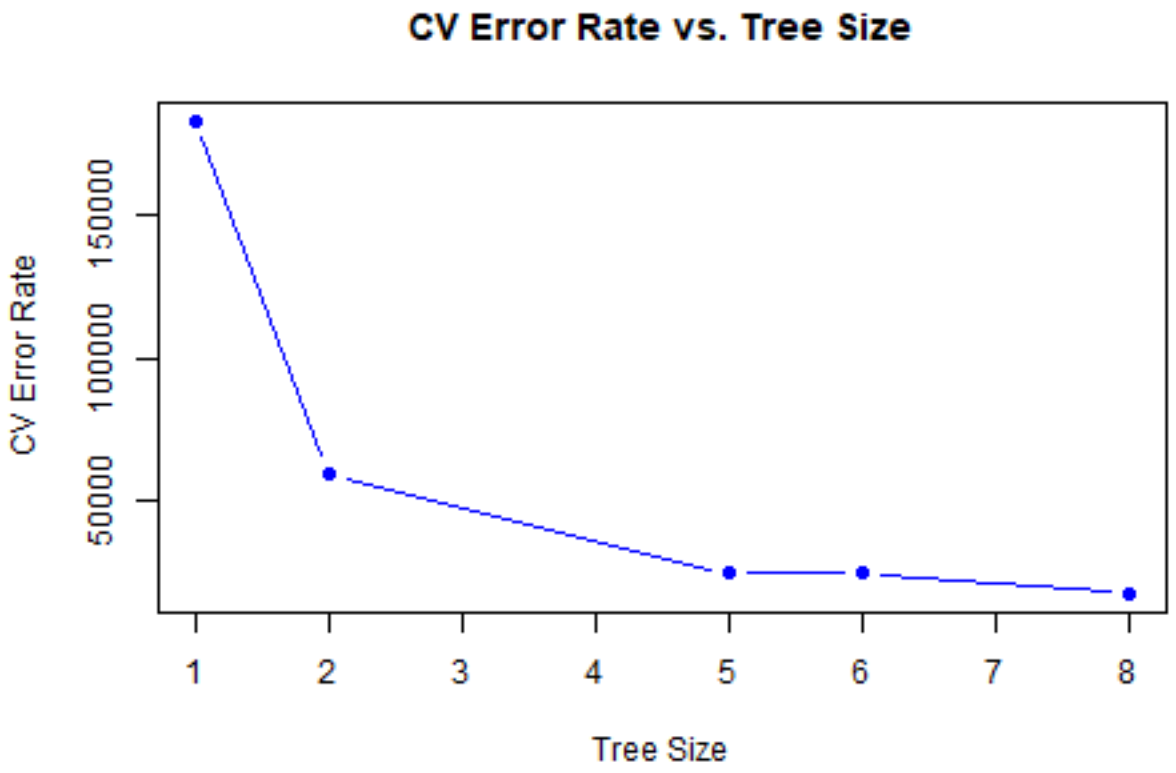
The code above will utilize 10-fold cross-validation to prune my decision tree. We use the `prune.misclass` function in order for our cross-validation to be set to minimize classification error rather than model deviance.

With our model tuned using pruning and cross-validation, let's go ahead and plot our tree.

Plotting the Pruned Tree

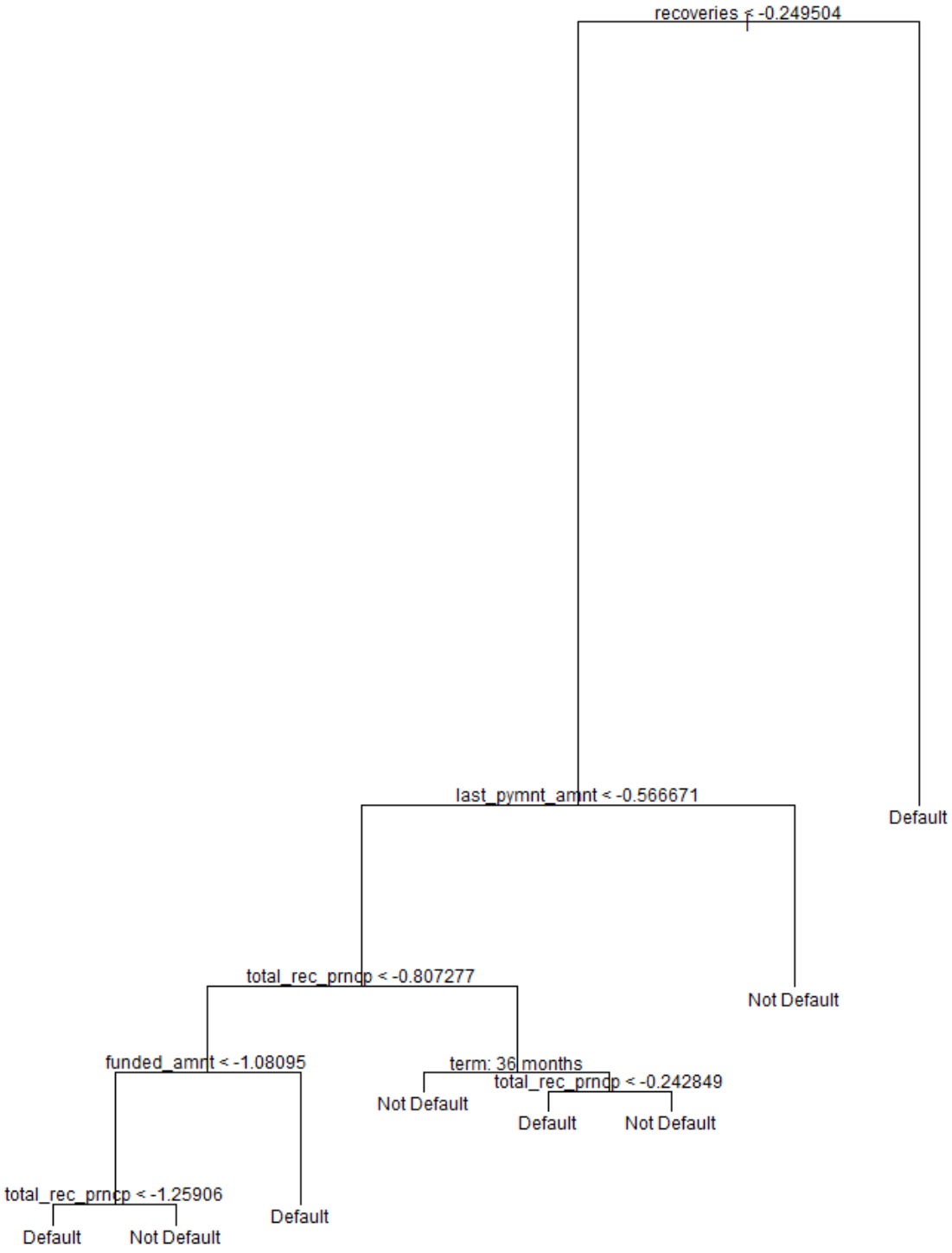
In order to determine our best tree, we first need to plot our cross-validation error rate against tree size. Let's go ahead and plot that.

```
plot(cv_tree$size,
     cv_tree$dev,
     type = "b",
     pch = 16,
     col = "blue",
     xlab = "Tree Size",
     ylab = "CV Error Rate",
     main = "CV Error Rate vs. Tree Size")
```



Looking at the plot above, we can see that the tree size with the lowest Cross-Validation error rate is the tree of size 8. With that tree size in mind, let's go ahead and plot this exact tree.

```
set.seed(490)

prune_cv_tree = prune.misclass(default_tree,
                               best = 8)


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


recoveries < -0.249504
Default
last_pymnt_amnt < -0.566671
Not Default
total_rec_prncp < -0.807277
funded_amnt < -1.08095
term: 36 months
Not Default
total_rec_prncp < -0.242849
Default
Not Default
total_rec_prncp < -1.25906
Default
Default
Not Default

Interpreting Pruned Tree

Above is a plot of our best tree, which was created through the use of cross-validation and pruning on classification error. Our tree has a total size of 8, meaning there are 8 splits. Let's go through each of the selected variables in use by our tree, going from top-to-bottom splits. Recall our variables are standardized, so the numbers split on are also standardized.

- *recoveries* - Our first split. If Recoveries < -.25, then you split to the next node. If Recoveries > -.25, you are classified as Default.
- *last_pymnt_amnt* - If < -.567, split to next node. If > -.567, then you are classified as Not Default.
- *total_rec_prncp* - If < -.807, split to next node. If > -.807, split to next node.
  - *term60.months* - If not 60 month term, classified as Not Default. If 60 month term, split to next node.
    - *total_rec_prncp* - If < -.243, classify as Default. If > -.243, classify as Not Default.
  - *funded_amnt* - If < -1.08, split to next node. If > -1.08, classify as Default.
    - *total_rec_prncp* If < -1.26, classify as Default. If > -1.26, classify as Not Default.

As we can see above, the variable *total_rec_prncp* was split on a total of three times. Recall this variable is defined as the principal received to date. So, this variable has a good amount of importance for classifying loan defaults. With our tree interpreted and splits looked at, let's take a look at the confusion matrix to see how well our model classifies. We will be looking at the confusion matrix for predictions on our test data.

```
set.seed(490)

tree_pred = predict(prune_cv_tree,
                    lending_data_test_scaled,
                    type = "class")

tree_test_accuracy = mean(tree_pred == lending_data_test_scaled$loan_status_final)

tree_test_error = mean(tree_pred != lending_data_test_scaled$loan_status_final)

tree_conf_mtx = confusionMatrix(table(
      predicted = tree_pred,
      actual = lending_data_test_scaled$loan_status_final),
      positive = "Default")

tree_conf_mtx$table
```

```
##              actual
## predicted     Not Default Default
##   Not Default      312781    6927
##   Default             401   71807
```

Looking at the confusion matrix above on our test data, we are correctly classifying Not Default and Default a very large amount of the time. In order to get the results in numbers, let's take a look at the accuracy and error rate of our chosen tree.

Comparing Results Between Tree & Regression Models

```
model_full_test_df = data.frame("Decision Tree" = c(
                                "Accuracy on Test Data" = tree_test_accuracy,
                                "Error on Test Data" = tree_test_error),
                                  "Ridge" = c(
                                "Accuracy on Test Data" = ridge_test_accuracy,
                                "Error on Test Data" = ridge_test_error),
                                  "LASSO" = c(
                                "Accuracy on Test Data" = lasso_test_accuracy,
                                "Error on Test Data" = lasso_test_error),
                                  "Logistic" = c(
                                "Accuracy on Test Data" = test_log_accuracy,
                                "Error on Test Data" = test_log_error),
                                  "Original Logistic" = c(
                                "Accuracy on Test Data" = log_mod5_eval$accuracy,
                                "Error on Test Data" = log_mod5_eval$error))#include log5 model
kable(model_full_test_df)
```

	Decision.Tree	Ridge	LASSO	Logistic	Original.Logistic
Accuracy on Test Data	0.9813021	0.999579	0.9994642	0.9997448	0.7995642
Error on Test Data	0.0186979	0.000421	0.0005358	0.0002552	0.2004358

Looking at the results above, we can see that our chosen Decision Tree only outperforms our model from report 2 when evaluated on the test data. We get an accuracy score of 0.9813021 and an error score of 0.0186979 on the decision tree. This tells me that logistic regression models, as well as LASSO and Ridge regression models outperform this decision tree.

With the given accuracy of our best decision tree, I believe there is still room for improvement. What if we were to create multiple trees and average their predictions? That's exactly what we will be doing in the next section.

Boot Function

Fitting 100 Trees

```
set.seed(490)

bootstrap_tree_func = function(data, index) {
  data1 = data[index, ]
  place_tree = tree(loan_status_final ~ ., data = data1)
  prune_tree = prune.misclass(place_tree,
                              k = 8)
  predictions = predict(prune_tree,
                        data1,
                        type = "class")
  mean(predictions != data1$loan_status_final)
}

# boot_trees = boot(lending_data_train_scaled,
#                   bootstrap_tree_func,
#                   R = 5)
boot_trees = readRDS("boot_trees.rds")#so we dont have to retrain..
```

The code above uses the `boot` function to fit 100 trees of the same size as the tree created in part 3. The `boot` function resamples the training data with replacement and for each resample, fits a tree and computes the prediction error rate on the sample. Let's go ahead and look at our average prediction error rate over these 100 trees.

Average Prediction Error Over 100 Trees

```
bootstrap_avg_error = mean(boot_trees$t)
```

Our average prediction error over 100 trees is 0.0185183. This is very similar to our prediction error rate obtained above in our original tree, which was 0.0186979. Let's go ahead and compare our bootstrap aggregated model performance with our original tree using a table.

Comparing Performance Between Bagged Trees and Single Tree

```
tree_comparison = data.frame("Bagged Trees" = c(
                              "Accuracy Rate" = 1 - bootstrap_avg_error,
                              "Error Rate" = bootstrap_avg_error),
                             "Single Tree" = c(
                              "Accuracy Rate" = tree_test_accuracy,
                              "Error Rate" = tree_test_error)
                            )
kable(tree_comparison)
```

	Bagged.Trees	Single.Tree
Accuracy Rate	0.9814817	0.9813021
Error Rate	0.0185183	0.0186979

As we can see above, using the bootstrap aggregated tree method gives us results that slightly outperform just a single decision tree. This gives us a result that is understandable, that if we fit numerous random trees we can obtain better results by averaging their predictions rather than predicting with a single tree.

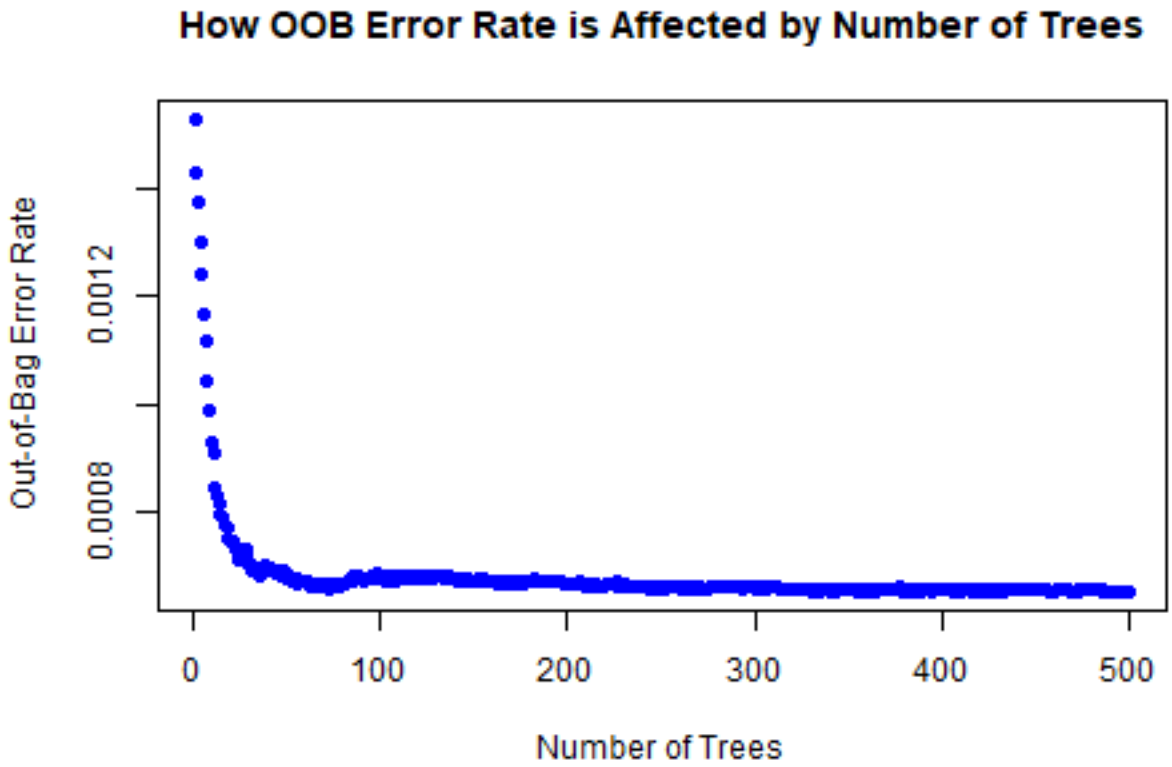
Bagging

```
set.seed(490)

# bag_default = randomForest(loan_status_final ~ .,
#                            data = lending_data_train_scaled,
#                            mtry = ncol(lending_data_train_scaled) - 1,
#                            importance = TRUE)
#commenting out to save time on training
bag_default = readRDS("bag_default.rds")
```

The code above creates a bagged classification model. For bagging, I set the parameter `mtry` equal to the number of variables in the model. This parameter lets the model consider all variables at each split. The number of trees being fit is equal to 500, which is the default parameter of the model. With the model fit, let's take a look at the out-of-bag error rate.

Plot out-of-bag error rate

```
plot(bag_default$err.rate[,1],
     xlab = "Number of Trees",
     ylab = "Out-of-Bag Error Rate",
     main = "How OOB Error Rate is Affected by Number of Trees",
     pch = 16,
     col = "blue")
```



Looking at the plot above, we can see that the out-of-bag error rate exponentially decays as the number of trees increases. This is expected, as our results will continue to improve as we aggregate our error rate over a larger number of trees. Recall that the out-of-bag error rate is the error rate on each training sample x_i using only the trees that did not contain x_i in their training sample. So, the out-of-bag error rate is an estimate of the error rate of the model on unseen data. Note that the performance starts to decay around # of trees = 100, so choosing the 100 trees for our model would work well. However, I will be keeping the parameter of total number of trees equal to 500. Overall, our plot shows that as the number of trees increase our OOB error rate decreases towards zero which is great.

Let's now take a look at the error rate table to get an idea of how our model is making predictions.

Plot Confusion Matrix

```
bag_default$confusion
```

```
##             Not Default Default   class.error
## Not Default      730748      10 0.00001368442
## Default             589  183124 0.00320608776
```

Looking at the table above, we can see that our model is correctly classifying "Default" and "Not Default" for almost every observation. However, we do misclassify 10 actual "Not Default" as "Default", and 589 actual "Default" as "Not Default". This shows us that our model is still predicting the majority class of "Not Default". However, with misclassifications being still very low it is not a large issue.

For the "Not Default" class, we get a classification error of about 0.0000137. For the "Default" class, we get a classification error of about 0.0032061. These are both very low, which shows our model is useful (at least on out-0f-bag predictions).

Let's move on and make a quick comparison between the error rate in this bagging model compared to the error rates of our LASSO/Ridge models. We will be comparing error rates on the test data.

Compare Error Rates

```
bag_predictions = predict(bag_default, lending_data_test_scaled)
bag_test_error = mean(bag_predictions != lending_data_test_scaled$loan_status_final)
bag_test_accuracy = mean(bag_predictions == lending_data_test_scaled$loan_status_final)

bag_test_error_comparison = data.frame("Bagging" = c("Accuracy on Test Data" = bag_test_accuracy,
                                                     "Error on Test Data" = bag_test_error),
                                       "Ridge" = c("Accuracy on Test Data" = ridge_test_accuracy,
                                                   "Error on Test Data" = ridge_test_error),
                                       "LASSO" = c("Accuracy on Test Data" = lasso_test_accuracy,
                                                   "Error on Test Data" = lasso_test_error))
kable(bag_test_error_comparison)
```

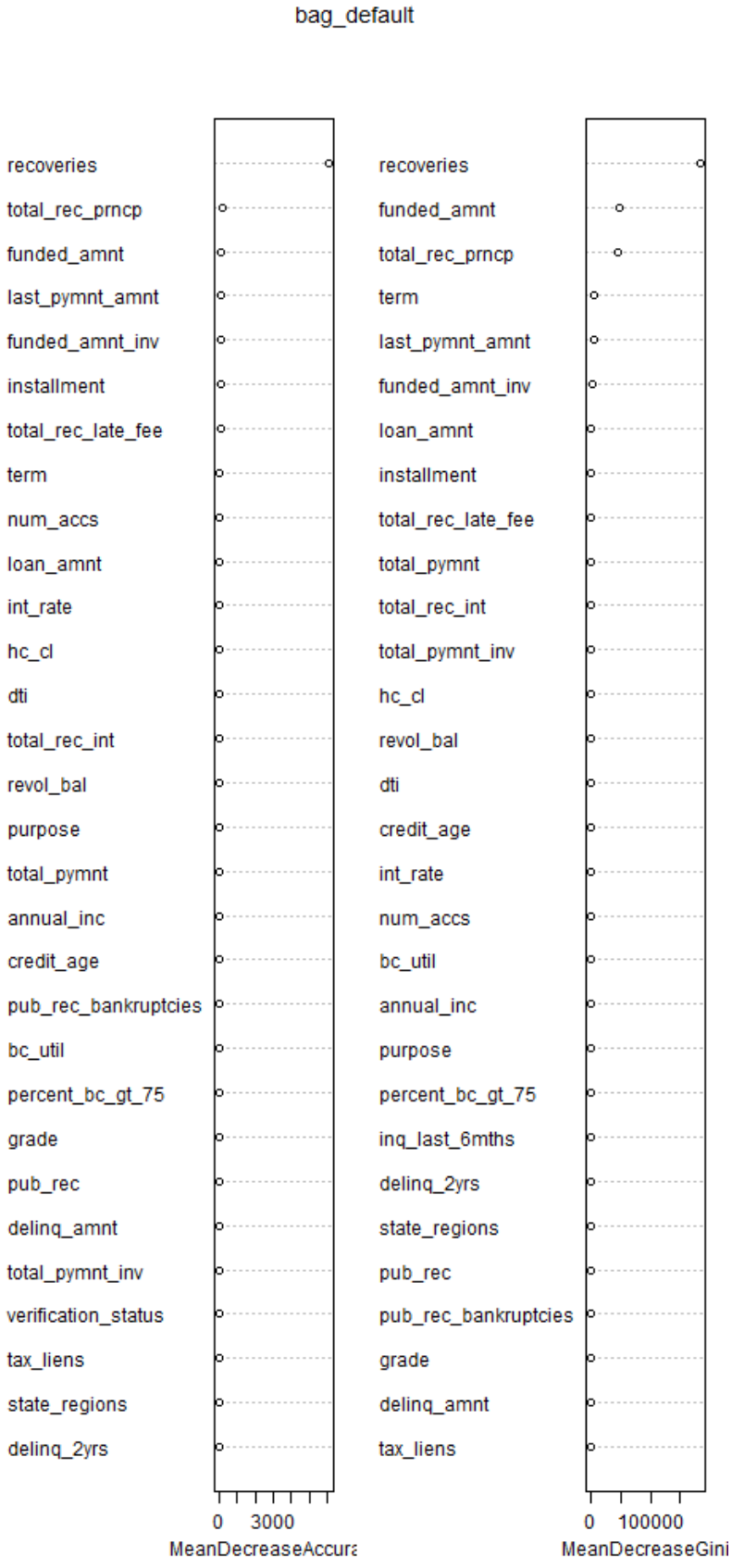
	Bagging	Ridge	LASSO
Accuracy on Test Data	0.9992473	0.999579	0.9994642
Error on Test Data	0.0007527	0.000421	0.0005358

Looking at the table above, we can see that our bagging model is being slightly outperformed by the Ridge and LASSO models, albeit by only ~.0003 and .0002 respectively. This difference is very minimal, which shows that this model is performing almost on par with these models. Overall, our bagging model performs very well on the held out test data and is competitive with parametric models of linear functional form while also requiring much less training time.

With error rate comparison aside, let's take a look at feature importance in this model.

Plot Importance Matrix

```
options(scipen = 10)
varImpPlot(bag_default)
```


bag_default
recoveries
total_rec_prncp
funded_amnt
last_pymnt_amnt
funded_amnt_inv
installment
total_rec_late_fee
term
num_accs
loan_amnt
int_rate
hc_cl
dti
total_rec_int
revol_bal
purpose
total_pymnt
annual_inc
credit_age
pub_rec_bankruptcies
bc_util
percent_bc_gt_75
grade
pub_rec
delinq_amnt
total_pymnt_inv
verification_status
tax_liens
state_regions
delinq_2yrs
0
3000
MeanDecreaseAccura
recoveries
funded_amnt
total_rec_prncp
term
last_pymnt_amnt
funded_amnt_inv
loan_amnt
installment
total_rec_late_fee
total_pymnt
total_rec_int
total_pymnt_inv
hc_cl
revol_bal
dti
credit_age
int_rate
num_accs
bc_util
annual_inc
purpose
percent_bc_gt_75
inq_last_6mths
delinq_2yrs
state_regions
pub_rec
pub_rec_bankruptcies
grade
delinq_amnt
tax_liens
0
100000
MeanDecreaseGini

Looking at the plot above, the top three important variables are `recoveries`, `total_rec_prncp`, and `funded_amnt`.

The first important variable is `recoveries` (post charge off gross recovery), which has the largest mean decrease in accuracy, meaning that the variable itself decreases accuracy of our model, but the largest mean decrease in gini. With such a large mean decrease in gini, this variable is highly important as it is overwhelmingly decreasing the average error rate in our model.

The next important variables is `total_rec_prncp` (principal received to date), which has a large mean decrease in gini with a small tradeoff in mean decrease in accuracy. This variable is also overwhelmingly decreasing our error rate with a small tradeoff in decrease in accuracy.

Lastly, another important variable is `funded_amnt` (total amount committed to that loan at that point in time). This variable also has a very large mean decrease in gini with a small tradeoff in mean decrease in accuracy. This variable is also overwhelmingly decreasing our error rate with a small tradeoff in decrease in accuracy.

Overall, these three variables are largely decreasing our mean error rates while not causing a sharp drop in mean accuracy rate.

Looking at other variables, there is some importance but they mostly hover around 0 mean decrease in accuracy and 0 mean decrease in gini and are not worth mentioning.

With our bagging model evaluated and commented on let's now move onto another form of ensemble learning, Random Forest. Random Forest is very similar to bagging, but with a small change in the number of variables considered at each node. With less variables considered at each node, we will be able to de-correlate the trees and hopefully improve our accuracy and error rate. Let's go ahead and create the model.

Random Forest

For this section, we will first be tuning our model using different values of `mtry`. In order to be efficient, we will be using a small sample of our data for tuning then building the full model with optimized parameters using all data.

```
set.seed(490)

rf_sample_index = createDataPartition(
  lending_data_final$loan_status_final,
  p = .075,
  list = FALSE)

rf_tune_sample = lending_data_final[rf_sample_index,]

rf_train_index = createDataPartition(
  rf_tune_sample$loan_status_final,
  p = .70,
  list = FALSE)

rf_tune_train = rf_tune_sample[rf_train_index,]
rf_tune_test = rf_tune_sample[-rf_train_index,]

num_columns_rf = colnames(rf_tune_train %>% select_if(is.numeric) %>% select(-c(out_prncp, out_prncp_inv

scales_rf = build_scales(rf_tune_train,
                         num_columns_rf,
```

```
                        verbose = TRUE)

rf_tune_train_scale = fastScale(rf_tune_train,
                                scales = scales_rf,
                                verbose = TRUE)
rf_tune_test_scale = fastScale(rf_tune_test,
                               scales = scales_rf,
                               verbose = TRUE)

rf_tune_train_scale = rf_tune_train_scale %>% select(-sub_grade)
rf_tune_test_scale = rf_tune_test_scale %>% select(-sub_grade)

rf_X_train = rf_tune_train_scale %>% select(-loan_status_final)
rf_Y_train = rf_tune_train_scale %>% select(loan_status_final)

cl = detectCores(logical=FALSE)
registerDoParallel(cl)

oob_and_test_errs = foreach (i = 1:34, .combine = 'rbind', .multicombine = TRUE,
         .packages = 'randomForest') %dopar% { #try mtry from 1 to sqrt(p-1) + 1
    rf = randomForest(loan_status_final ~ .,
                      data = rf_tune_train_scale,
                      mtry = i,
                      ntree = 500,
                      importance = TRUE)

    oob_err = rf$err.rate[500] #get error of all trees

    test_preds = predict(rf, rf_tune_test_scale)

    test_err = mean(test_preds != rf_tune_test_scale$loan_status_final)

    data.frame("mtry" = i,
               "oob_err" = oob_err,
               "test_err" = test_err)
}
##THIS CODE IS NOT RUN DUE TO EXTENSIVE TRAINING TIME AND
##UNEEDED DATA USED FOR TUNING
```

```
oob_and_test_errs = readRDS("oob_and_test_errs.rds")
```

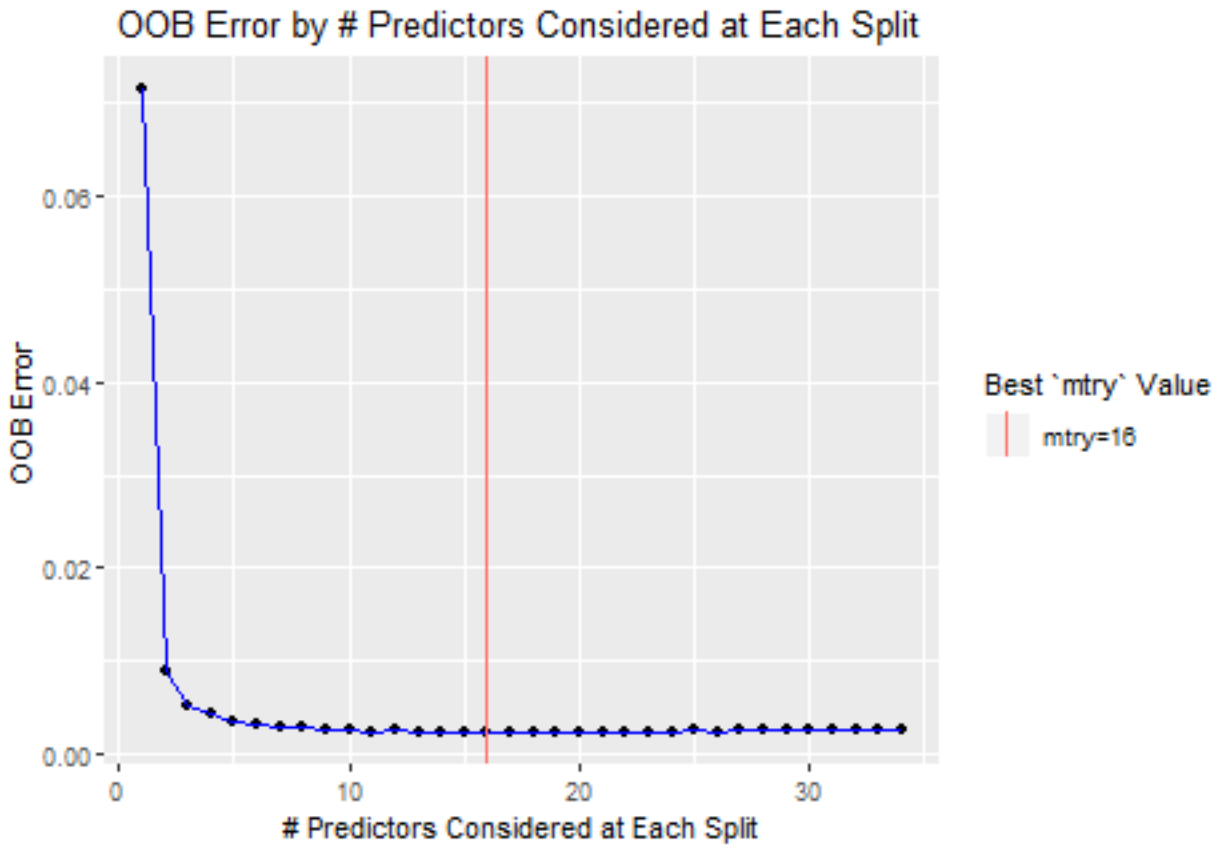
The code above creates a Random Forest model with the number of variables considered at each node being equal to the square root of the total number of variables in our data. So, for each node our model will only consider 5. Similar to the bagging model above, the number of trees being fit is equal to 500.

With the model fit, let's go ahead and take a look at the out-of-bag error rate compared to the number of predictors considered in each split.

Plot OOB Error

```
best_mtry = oob_and_test_errs$mtry[which.min(oob_and_test_errs$oob_err)]

ggplot(data = oob_and_test_errs, aes(x = as.numeric(mtry), y = oob_err)) +
  geom_point(color = "black") +
```

```
  geom_line(color = "blue") +
  geom_vline(aes(xintercept = best_mtry, color = "mtry=16")) +
  labs(x = "# Predictors Considered at Each Split",
       y = "OOB Error",
       title = "OOB Error by # Predictors Considered at Each Split",
       color = "Best `mtry` Value") +
  theme(plot.title = element_text(hjust = 0.5))
```



Looking at the plot above, we can see that as we increase from `mtry` = 1 to `mtry` = 2, we have a very sharp drop in the OOB error. However, it starts to smooth out at about `mtry` = 5 (which is about the square root of the total number of predictors). Overall, the `mtry` with the lowest OOB is `mtry` = 16, so we will be using that value to compute our final model.

Use OOB Error to Tune `mtry`

```
# rf_min_oob_err_indx = which.min(oob_and_test_errs$oob_err)
# rf_best_mtry = oob_and_test_errs$mtry[rf_min_oob_err_indx]
# best_rf = randomForest(loan_status_final ~ .,
#                        data = lending_data_train_scaled,
#                        mtry = rf_best_mtry,
#                        ntree = 500,
#                        importance = TRUE)
#extensive training time, commented out
best_rf = readRDS("best_rf.rds")
```

Overall, using our OOB error we have determined `mtry`=16 to be our best model. Our code above trains a

random forest model on our full data (rather than the sample used for tuning), using `mtry` = rf_best_mtry, where rf_best_mtry is the `mtry` corresponding to the minimum OOB error.

With our model tuned and trained, let's go ahead and take a look at the error rate table.

Plot Error Rate Table

```
best_rf$confusion
```

```
##             Not Default Default    class.error
## Not Default      730757       1 0.000001368442
## Default             492  183221 0.002678090282
```

Looking at the table above, we can say that we are overwhelmingly predicting the correct classes for `Default` and `Not Default`. However, we have misclassified 1 observation that is `Not Default` as `Default`. We have also misclassified 492 observations that are `Default` as `Not Default`. This shows we are still having an issue of our model misclassifying the data as the majority class, but with such a small number of misclassifications it is not a large issue.

For the "Not Default" class, we get a classification error of about 0.0000014. For the "Default" class, we get a classification error of about 0.0026781. These are both very low, which shows our model is useful (at least on out-0f-bag predictions).

Overall, our model is correctly classifying on the training data an overwhelmingly amount which is great. Now, let's take a look at how our model performs on the test data.

Compare Model to Previous Models

```
rf_predictions = predict(best_rf, lending_data_test_scaled)
rf_test_error = mean(rf_predictions != lending_data_test_scaled$loan_status_final)
rf_test_accuracy = mean(rf_predictions == lending_data_test_scaled$loan_status_final)

rf_test_error_comparison = data.frame("RandomForest" = c("Accuracy on Test Data" = rf_test_accuracy,
                                                         "Error on Test Data" = rf_test_error),
                                      "Bagging" = c("Accuracy on Test Data" = bag_test_accuracy,
                                                    "Error on Test Data" = bag_test_error),
                                       "Ridge" = c("Accuracy on Test Data" = ridge_test_accuracy,
                                                   "Error on Test Data" = ridge_test_error),
                                       "LASSO" = c("Accuracy on Test Data" = lasso_test_accuracy,
                                                   "Error on Test Data" = lasso_test_error))
kable(rf_test_error_comparison)
```

	RandomForest	Bagging	Ridge	LASSO
Accuracy on Test Data	0.9993519	0.9992473	0.999579	0.9994642
Error on Test Data	0.0006481	0.0007527	0.000421	0.0005358

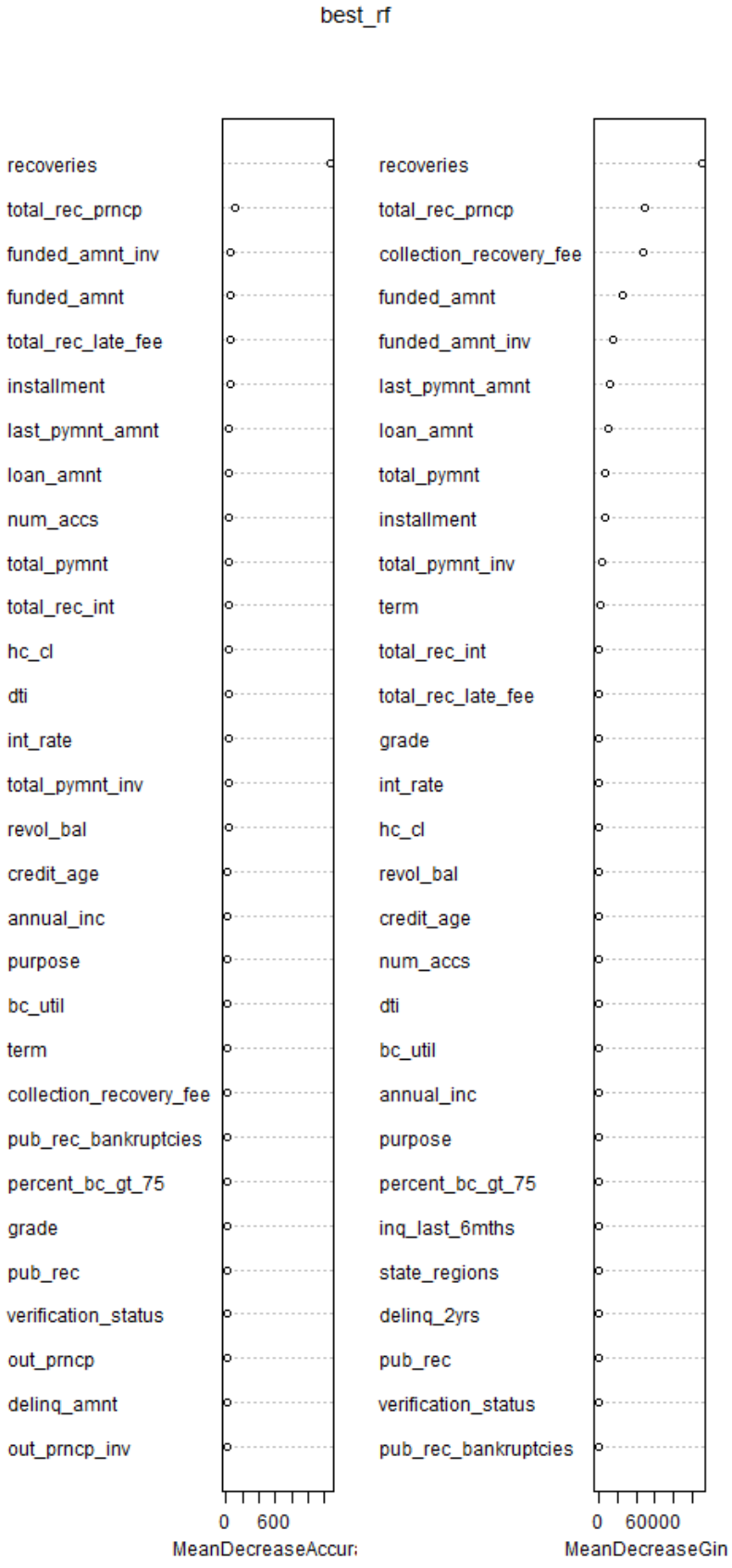
Looking at the table above, our Random Forest model with tuned `mtry` is slightly outperforming our Bagged model by about .0001. While all improvements are welcomed, this is not a huge improvement. Recall that our bagged model is the same as our random forest, but with `mtry` equal to the total number of predictors. So we can see that decreasing the total number of predictors considered at each split increases our accuracy. Our Random Forest model is being outperformed by our Ridge and LASSO regression models, similar to our bagged model, hinting that our data may be more of a linear form.

Overall, our model performs very well on unseen data but other algorithms may show better results. Now,

let's go ahead and plot the importance matrix to get a feel of the variables driving our model.

Plot Importance Matrix

```
options(scipen = 10)
varImpPlot(best_rf)
```


best_rf
recoveries
total_rec_prncp
funded_amnt_inv
funded_amnt
total_rec_late_fee
installment
last_pymnt_amnt
loan_amnt
num_accs
total_pymnt
total_rec_int
hc_cl
dti
int_rate
total_pymnt_inv
revol_bal
credit_age
annual_inc
purpose
bc_util
term
collection_recovery_fee
pub_rec_bankruptcies
percent_bc_gt_75
grade
pub_rec
verification_status
out_prncp
delinq_amnt
out_prncp_inv
0
600
MeanDecreaseAccur
recoveries
total_rec_prncp
collection_recovery_fee
funded_amnt
funded_amnt_inv
last_pymnt_amnt
loan_amnt
total_pymnt
installment
total_pymnt_inv
term
total_rec_int
total_rec_late_fee
grade
int_rate
hc_cl
revol_bal
credit_age
num_accs
dti
bc_util
annual_inc
purpose
percent_bc_gt_75
inq_last_6mths
state_regions
delinq_2yrs
pub_rec
verification_status
pub_rec_bankruptcies
0
60000
MeanDecreaseGin

Looking at the plots above, our top two most important variables are `recoveries` and `total_rec_prncp`.

For `recoveries`, the total mean decrease in gini (classification error) is a little over 100,000. The total mean decrease in accuracy is a little over 1,200. This tradeoff for a huge mean decrease in error for a small mean decrease in accuracy is showing that this variable is highly important in our model.

For `total_rec_prncp`, the total mean decrease in gini (classification error) is about 50,000. The total mean decrease in accuracy is about 75. This tradeoff for a huge mean decrease in error for a small mean decrease in accuracy is showing that this variable is highly important in our model.

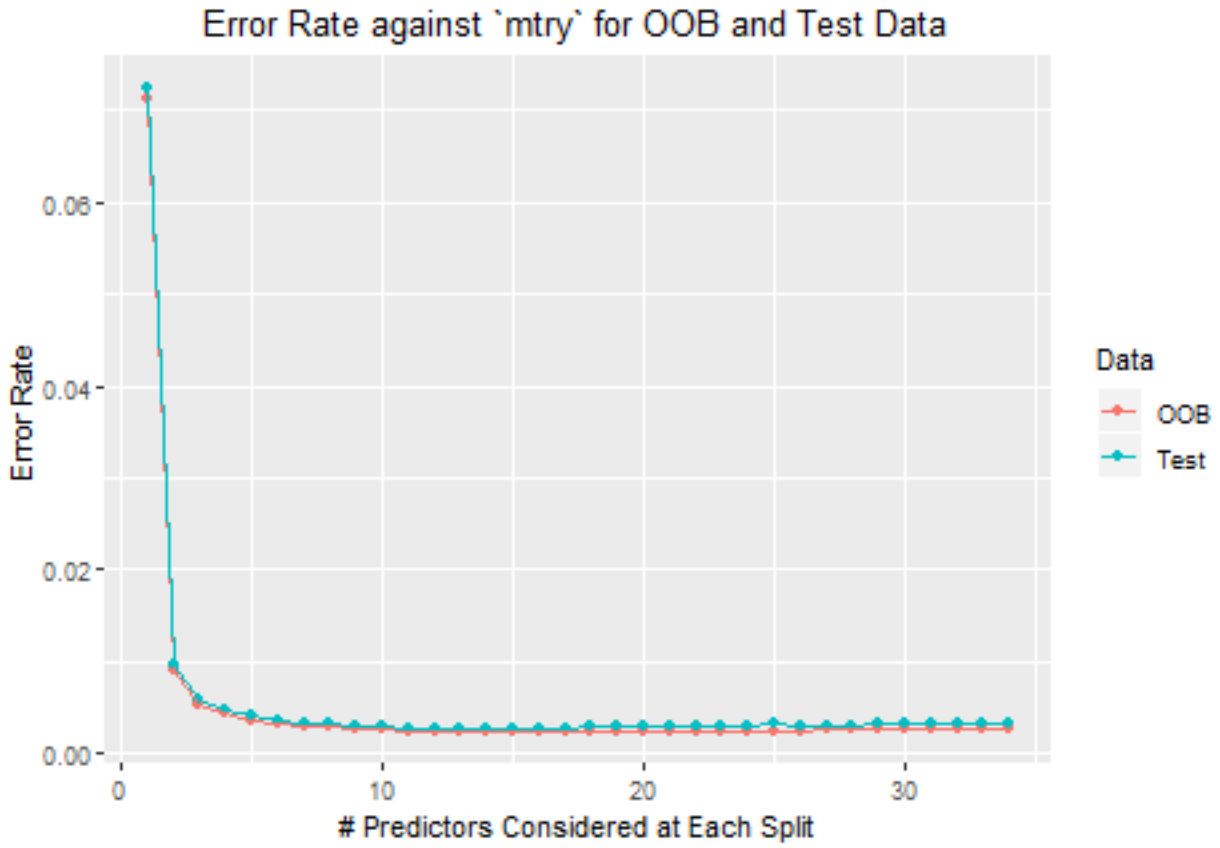
Comparing to the importance matrix for our bagged model, `funded_amnt` is lagging behind `collection_recovery_fee` in the mean decrease in gini. However, it can still be said that `funded_amnt` is an important variable.

Overall, our top two important variables in the Random Forest model are `recoveries` and `total_rec_prncp`, with other variables of importance being seen in the plot above.

With variable importance analyzed, let's move back to our original tuning method and plot the OOB Error and Test Error against `mtry`.

Plot OOB Error and Test Error Against `mtry`

```
ggplot(oob_and_test_errs, aes(mtry, y = value, color = variable)) +
  geom_point(aes(y = oob_err, col = "OOB")) +
  geom_point(aes(y = test_err, col = "Test")) +
  geom_line(aes(y = oob_err, col = "OOB")) +
  geom_line(aes(y = test_err, col = "Test")) +
  labs(x = "# Predictors Considered at Each Split",
       y = "Error Rate",
       title = "Error Rate against `mtry` for OOB and Test Data",
       color = "Data") +
  theme(plot.title = element_text(hjust = 0.5))
```



Looking at the plot above, we can see that the OOB and Test Errors follow a very similar pattern. This helps show that using out-of-bag error as an estimate for the test error is useful, as they follow similar patterns. Tuning on OOB error helps simulate test error, so will ultimately create better performing models on unseen data. For the tuning I did, both OOB and Test error selected `mtry` = 16 to be the best parameter for `mtry`.

Overall, our OOB and Test error follow a similar pattern when being used to tune `mtry` and can be said that utilizing OOB Error to estimate test error is useful.

Now that we have created both Bagged and Random Forest models, let's move on to boosted models.

Boosting

For this model, we will be ditching the ensemble learning and moving onto boosting. Boosted learning is a way to compute a large amount of weak learners and let them learn from each other, and ultimately create a good model from these. Let's create the model.

```
set.seed(490)

lending_data_train_scaled$loan_status_binary = ifelse(
  lending_data_train_scaled$loan_status_final == "Default",
  1, 0
)

# boost_default = gbm(loan_status_binary ~ . - loan_status_final,
#                     data = lending_data_train_scaled,
#                     distribution = "bernoulli",
```

```
#                     n.trees = 5000,
#                     interaction.depth = 4)
#commented out due to extensive training time

boost_default = readRDS("boost_default.rds")
```

The model above creates a boosted decision tree model utilizing parameters given in class. Since my problem is a classification problem, I have set the distribution to `bernoulli` as this is a distribution with outcomes either 0 or 1. The outcome `0` indicates "Not Default", and the outcome `1` indicates "Default". The number of trees being fit is equal to 5000, so 5000 trees are being fit to ultimately create one strong tree that is highly accurate. The interaction depth being equal to 4 means that there will be 4 splits in each tree.

With the model created, let's take a look at the error rate table (confusion matrix) of the model.

Plot Error Rate Table

```
boost_test_preds = predict(boost_default,
                           lending_data_test_scaled,
                           n.trees = 5000,
                           type = "response")
class_boost_test_preds = as.factor(ifelse(boost_test_preds > .50,
                                "Default",
                                "Not Default"))
boost_conf_matr = confusionMatrix(class_boost_test_preds,
                                  lending_data_test_scaled$loan_status_final,
                                  positive = "Default")
```

```
## Warning in confusionMatrix.default(class_boost_test_preds,
## lending_data_test_scaled$loan_status_final, : Levels are not in the same
## order for reference and data. Refactoring data to match.
```

```
boost_conf_matr$table
```

```
##              Reference
## Prediction    Not Default Default
##   Not Default      313132     562
##   Default              50   78172
```

Looking at the table above, we can see that overall our table is performing very well on the test data. However, we still have the issue of our model predicting "Not Default" on true "Default" much more than the opposite. However, these misclassifications are very small compared to the number of observations so is not much of an issue.

Let's now compare this model to our Bagging, Random Forest, and LASSO/Ridge regression models.

Compare Test Accuracy/Error to Other Models

```
boost_test_error = mean(class_boost_test_preds != lending_data_test_scaled$loan_status_final)
boost_test_accuracy = mean(class_boost_test_preds == lending_data_test_scaled$loan_status_final)

boost_test_error_comparison = data.frame("Boosted" = c("Accuracy on Test Data" = boost_test_accuracy,
                                                       "Error on Test Data" = boost_test_error),
                                         "RandomForest" = c("Accuracy on Test Data" = rf_test_accuracy,
```

```
                                                    "Error on Test Data" = rf_test_error),
                                          "Bagging" = c("Accuracy on Test Data" = bag_test_accuracy,
                                                        "Error on Test Data" = bag_test_error),
                                          "Ridge" = c("Accuracy on Test Data" = ridge_test_accuracy,
                                                      "Error on Test Data" = ridge_test_error),
                                          "LASSO" = c("Accuracy on Test Data" = lasso_test_accuracy,
                                                      "Error on Test Data" = lasso_test_error))
kable(boost_test_error_comparison)
```

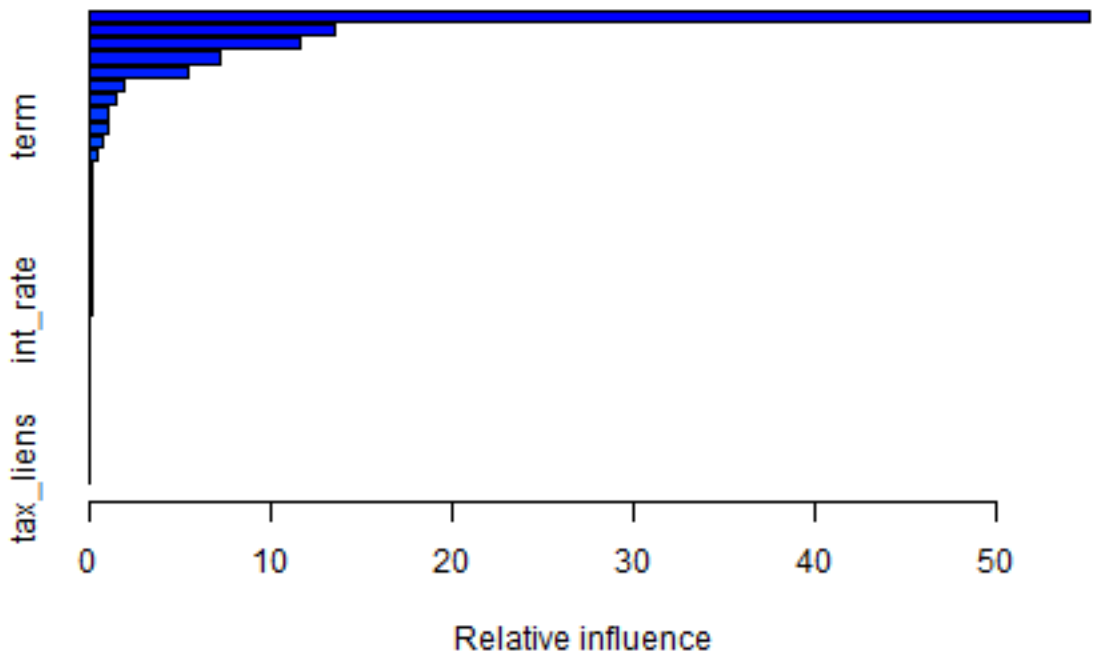
	Boosted	RandomForest	Bagging	Ridge	LASSO
Accuracy on Test Data	0.9984384	0.9993519	0.9992473	0.999579	0.9994642
Error on Test Data	0.0015616	0.0006481	0.0007527	0.000421	0.0005358

Looking at the table above, we can see that our boosted model is outperformed by the random forest, bagging, ridge, and lasso models. I believe this is due to some parameters being used, so we will try and improve this later on through the use of cross-validated grid search for parameter tuning. It also may be due to the classifier I have created, which only classifies observations as "Default" if their predicted probability is greater than .50. Overall, this model still performs extremely well on unseen data. Just not as good as other models, which we will hopefully improve upon.

Let's take a quick look at the importance matrix for this model before moving onto XGBoost.

Plot Importance Matrix

```
summary(boost_default)
```



```
##                                         var        rel.inf
## recoveries                         recoveries 55.14324090555
## funded_amnt                       funded_amnt 13.43194168299
## total_rec_prncp               total_rec_prncp 11.59171840410
## dti                                       dti  7.14209846214
## last_pymnt_amnt               last_pymnt_amnt  5.43624914616
## loan_amnt                           loan_amnt  1.90768230422
## revol_bal                           revol_bal  1.48079758575
## installment                       installment  1.02572400643
## term                                     term  0.94223392119
## total_pymnt                       total_pymnt  0.65117017896
## total_rec_int                   total_rec_int  0.34787988048
## funded_amnt_inv               funded_amnt_inv  0.16851631881
## total_rec_late_fee         total_rec_late_fee  0.15196187359
## num_accs                           num_accs  0.13416215896
## grade                                 grade  0.07123814635
## purpose                             purpose  0.05618367298
## bc_util                             bc_util  0.05198993780
## percent_bc_gt_75           percent_bc_gt_75  0.05139891797
## state_regions                 state_regions  0.04805571803
## annual_inc                       annual_inc  0.04324720977
## hc_cl                                 hc_cl  0.03745165444
## int_rate                           int_rate  0.03442608085
## credit_age                       credit_age  0.02742120174
## verification_status     verification_status  0.01446246209
## total_pymnt_inv             total_pymnt_inv  0.00424753884
## collection_recovery_fee collection_recovery_fee  0.00388978681
## pub_rec                             pub_rec  0.00045185042
## out_prncp                         out_prncp  0.00007351441
## pub_rec_bankruptcies   pub_rec_bankruptcies  0.00004080699
## inq_last_6mths               inq_last_6mths  0.00003629973
## delinq_2yrs                     delinq_2yrs  0.00000837146
## out_prncp_inv                 out_prncp_inv  0.00000000000
## delinq_amnt                     delinq_amnt  0.00000000000
## tax_liens                         tax_liens  0.00000000000
```

Looking at the table and plot above, we see once again that the three most important variables are `recoveries`, `funded_amnt`, and `tot_rec_prncp`. These three variables have the largest relative influence, so therfore these variables are the largest contributing factor to our model.

Relative influence can be described as how much a variable accounts for reduction to the loss function given all of these features. With our models agreeing on variable importance, this leads us to believe that these variables truly have high influence on a default outcome.

Overall, these three variables have been chosen due to having the largest reduction to the loss function in our model and therefore having the largest influence on a loan default outcome.

Before moving on from boosted decision trees, I will be utilizing cross-validation for parameter tuning to see if results can be improved on this model.

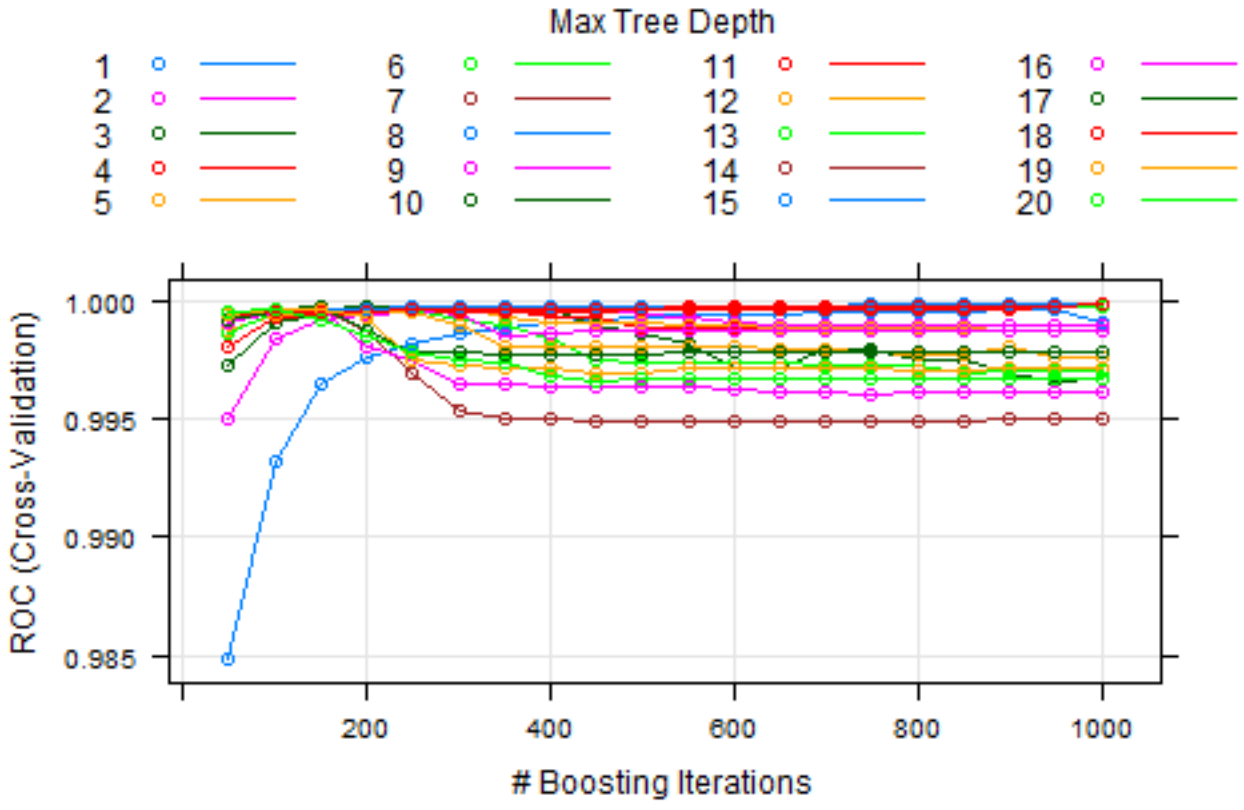
```
# cl = makeCluster(detectCores(logical=FALSE))
# registerDoParallel(cl)
#
# bst_X_trn = model.matrix(loan_status_final ~ ., data = rf_tune_train_scale)
# bst_Y_trn = rf_tune_train_scale$loan_status_final
```

```
# levels(bst_Y_trn)[levels(bst_Y_trn)=="Not Default"] = "NotDefault"
#
# set.seed(490)
# boost_ctrl = trainControl(method = "cv", number = 5,
#                           summaryFunction = twoClassSummary,
#                           classProbs = TRUE)
#
# boost_cv = train(x = bst_X_trn, y = bst_Y_trn,
#                  method = "gbm", trControl = boost_ctrl,
#                  verbose = FALSE,
#                  metric = "ROC",
#                  tuneLength = 20)
#commented out due to extensive training
boost_cv = readRDS("boost_cv.rds")
```

The above code performs a cross-validated grid search of the variable `boost_params`, in order to tune our parameters for our boosted decision tree model. We are tuning on `interaction.depth`, and `n.trees`. `interaction.depth` defines the comlexity of the tree, and `n.trees` defines the number of trees to fit (or number of iterations).

With our models fit and tuned using five-fold cross-validation, let's go ahead and quickly plot our tuning output and choose the best model.

```
plot(boost_cv)
```



The plot above is quite a mess, but we can see that for the most part at the number of boosting iterations increased, ROC increased as well. Let's choose our best model using ROC.

```
boost_cv$bestTune
```

```
##     n.trees interaction.depth shrinkage n.minobsinnode
## 299     950                15       0.1             10
```

So, our best model has `n.trees` = 950, `interaction.depth` = 15, `shrinkage` = 0.1, and `n.minobsinnode` = 10. This model had the greatest ROC score of about 1.

With our best model chosen, let's go ahead and look at the confusion matrix of our model on the test data.

Confusion Matrix of Tuned Boosted Model

```
lending_data_test_scaled_relabeled = lending_data_test_scaled
levels(lending_data_test_scaled_relabeled$loan_status_final)[
  levels(lending_data_test_scaled_relabeled$loan_status_final)=="Not Default"] = "NotDefault"
boost_cv_test = model.matrix(loan_status_final ~ ., data = lending_data_test_scaled_relabeled)

boost_cv_preds = predict.train(boost_cv, boost_cv_test)

boost_conf_matr = confusionMatrix(boost_cv_preds,
                                  lending_data_test_scaled_relabeled$loan_status_final,
                                  positive = "Default")
boost_conf_matr$table
```

```
##             Reference
## Prediction   NotDefault Default
##   NotDefault      25442       3
##   Default        287740   78731
```

Looking at the confusion matrix, our best model is not doing as well as we thought. We are overwhelmingly misclassifying our data. This leads me to believe that `n.trees` is not large enough, or we did not use a large enough sample of our data. Overall, we observed better results in our original boosted model.

Compare Model to Previous Models

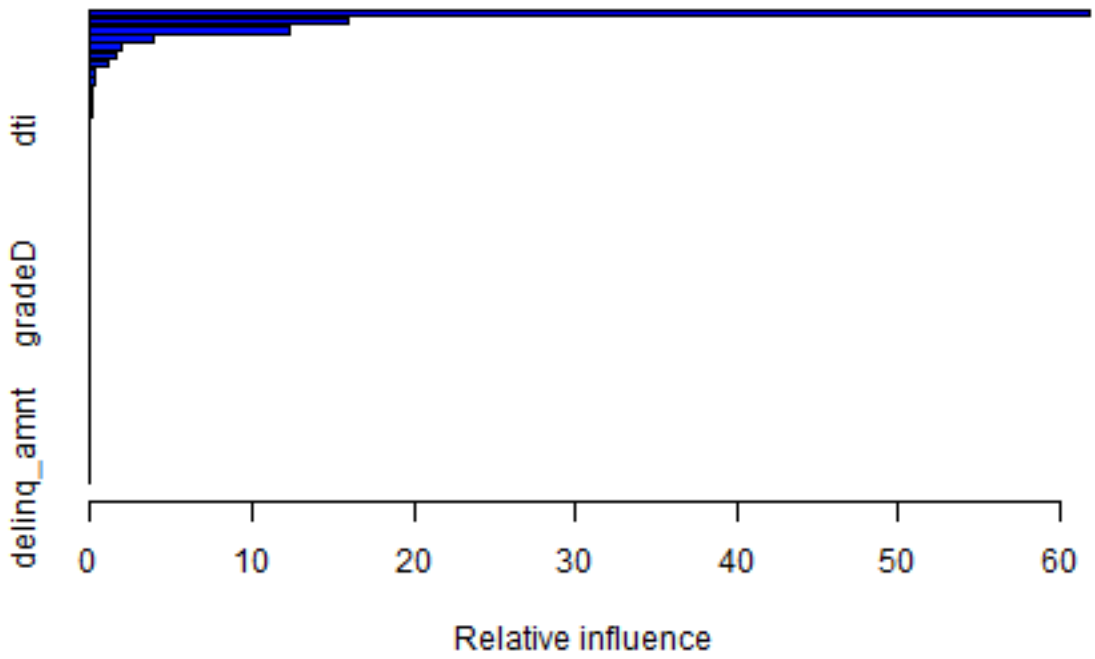
```
boost_cv_test_error = mean(boost_cv_preds !=
                           lending_data_test_scaled_relabeled$loan_status_final)
boost_cv_test_accuracy = mean(boost_cv_preds ==
                           lending_data_test_scaled_relabeled$loan_status_final)

boost_cv_test_error_comparison = data.frame("BoostCV" = c("Accuracy on Test Data" = boost_cv_test_accura
                                                          "Error on Test Data" = boost_cv_test_error),
                                            "Boosted" = c("Accuracy on Test Data" = boost_test_accuracy,
                                                          "Error on Test Data" = boost_test_error),
                                            "RandomForest" = c("Accuracy on Test Data" = rf_test_accuracy,
                                                               "Error on Test Data" = rf_test_error),
                                            "Bagging" = c("Accuracy on Test Data" = bag_test_accuracy,
                                                          "Error on Test Data" = bag_test_error),
                                            "Ridge" = c("Accuracy on Test Data" = ridge_test_accuracy,
                                                        "Error on Test Data" = ridge_test_error),
                                            "LASSO" = c("Accuracy on Test Data" = lasso_test_accuracy,
                                                        "Error on Test Data" = lasso_test_error))
kable(boost_cv_test_error_comparison)
```

	BoostCV	Boosted	RandomForest	Bagging	Ridge	LASSO
Accuracy on Test Data	0.2658044	0.9984384	0.9993519	0.9992473	0.999579	0.9994642
Error on Test Data	0.7341956	0.0015616	0.0006481	0.0007527	0.000421	0.0005358

As our error shows, our cross-validated model is not doing very well. As mentioned above, I believe it did not utilize enough trees to create a strong learner. Overall, all other models have outperformed this one.

Plot Importance Matrix

```
summary(boost_cv)
```




```
##                                                  var
## recoveries                                 recoveries
## total_rec_prncp                       total_rec_prncp
## funded_amnt                               funded_amnt
## funded_amnt_inv                       funded_amnt_inv
## last_pymnt_amnt                       last_pymnt_amnt
## term60 months                           term60 months
## loan_amnt                                   loan_amnt
## installment                               installment
## total_rec_late_fee                 total_rec_late_fee
## total_rec_int                           total_rec_int
## total_pymnt                               total_pymnt
## int_rate                                     int_rate
```

```
## annual_inc                                                annual_inc
## hc_cl                                                          hc_cl
## dti                                                              dti
## credit_age                                                credit_age
## num_accs                                                    num_accs
## revol_bal                                                  revol_bal
## bc_util                                                      bc_util
## total_pymnt_inv                                       total_pymnt_inv
## percent_bc_gt_75                                     percent_bc_gt_75
## pub_rec                                                      pub_rec
## inq_last_6mths                                         inq_last_6mths
## gradeB                                                       gradeB
## collection_recovery_fee                           collection_recovery_fee
## purposevacation                                     purposevacation
## delinq_2yrs                                             delinq_2yrs
## state_regionsSouthwest                           state_regionsSouthwest
## purposecredit_card                                 purposecredit_card
## purposedebt_consolidation                       purposedebt_consolidation
## pub_rec_bankruptcies                               pub_rec_bankruptcies
## verification_statusSource Verified verification_statusSource Verified
## purposemoving                                           purposemoving
## gradeD                                                         gradeD
## purposerenewable_energy                         purposerenewable_energy
## gradeC                                                         gradeC
## state_regionsSoutheast                           state_regionsSoutheast
## purposemedical                                     purposemedical
## tax_liens                                               tax_liens
## purposemajor_purchase                           purposemajor_purchase
## state_regionsWest                                   state_regionsWest
## purposeother                                           purposeother
## verification_statusVerified                   verification_statusVerified
## state_regionsNortheast                           state_regionsNortheast
## gradeF                                                         gradeF
## purposehome_improvement                         purposehome_improvement
## gradeE                                                         gradeE
## purposesmall_business                           purposesmall_business
## gradeG                                                         gradeG
## (Intercept)                                             (Intercept)
## purposeeducational                                 purposeeducational
## purposehouse                                             purposehouse
## purposewedding                                         purposewedding
## out_prncp                                                   out_prncp
## out_prncp_inv                                           out_prncp_inv
## delinq_amnt                                               delinq_amnt
##                                          rel.inf
## recoveries                        61.85580035480533
## total_rec_prncp                   15.96141641826293
## funded_amnt                       12.30711840606650
## funded_amnt_inv                    3.96846544670114
## last_pymnt_amnt                    1.90703822180650
## term60 months                      1.64828070454052
## loan_amnt                          1.09201990178748
## installment                        0.34082490134928
## total_rec_late_fee                 0.29582861463821
```

```
## total_rec_int                        0.18337410482938
## total_pymnt                          0.11807051747280
## int_rate                             0.04916265346919
## annual_inc                           0.03581271517503
## hc_cl                                0.03367702577411
## dti                                  0.03359751823838
## credit_age                           0.03140093012266
## num_accs                             0.02777973953334
## revol_bal                            0.02695153987017
## bc_util                              0.02516401724972
## total_pymnt_inv                      0.02315692770292
## percent_bc_gt_75                     0.00833548836783
## pub_rec                              0.00239841803305
## inq_last_6mths                       0.00235176993542
## gradeB                               0.00233650433865
## collection_recovery_fee              0.00204886332702
## purposevacation                      0.00203381924854
## delinq_2yrs                          0.00184510956311
## state_regionsSouthwest               0.00167973954342
## purposecredit_card                   0.00159194453882
## purposedebt_consolidation            0.00136227622972
## pub_rec_bankruptcies                 0.00128180786302
## verification_statusSource Verified   0.00127219256518
## purposemoving                        0.00083796948062
## gradeD                               0.00080777545374
## purposerenewable_energy              0.00071789427353
## gradeC                               0.00070712429886
## state_regionsSoutheast               0.00064041012146
## purposemedical                       0.00056582688020
## tax_liens                            0.00043148566507
## purposemajor_purchase                0.00039542981275
## state_regionsWest                    0.00036195136781
## purposeother                         0.00032080035195
## verification_statusVerified          0.00024865272561
## state_regionsNortheast               0.00017407356892
## gradeF                               0.00013857553465
## purposehome_improvement              0.00012757628437
## gradeE                               0.00003845499117
## purposesmall_business                0.00000739617236
## gradeG                               0.00000001006756
## (Intercept)                          0.00000000000000
## purposeeducational                   0.00000000000000
## purposehouse                         0.00000000000000
## purposewedding                       0.00000000000000
## out_prncp                            0.00000000000000
## out_prncp_inv                        0.00000000000000
## delinq_amnt                          0.00000000000000
```

Looking at the table and plot above, we see once again that the three most important variables are `recoveries`, `funded_amnt`, and `tot_rec_prncp`. These three variables have the largest relative influence. Relative influence can be described as how much a variable accounts for reduction to the loss function given all of these features. With our models agreeing on variable importance, this leads us to believe that these variables truly have high influence on a default outcome.

Overall, these three variables have been chosen due to having the largest reduction to the loss function in

our model and therefore having the largest influence on a loan default outcome.

With boosted decision trees visited and evaluated, let's move onto the XGBoost classification model. XGBoost also utilizes gradient boosting, but now includes an additional regularization parameter.

XGBoost

For this model, we will be fitting an XGBoost classification algorithm.

```
set.seed(490)

Y_train = as.matrix(lending_data_train_scaled[, "loan_status_final"])
Y_train = ifelse(Y_train == "Default", 1, 0)
X_train = model.matrix(loan_status_final ~ ., data = lending_data_train_scaled)


Y_test = as.matrix(lending_data_test_scaled[, "loan_status_final"])
Y_test = ifelse(Y_test == "Default", 1, 0)
X_test = model.matrix(loan_status_final ~ ., data = lending_data_test_scaled)

dtrain = xgb.DMatrix(data = X_train, label = Y_train)
dtest = xgb.DMatrix(data = X_test, label = Y_test)

# xgb_default = xgboost(data = dtrain,
#                       max_depth = 5,
#                       eta = 0.1,
#                       nrounds = 60,
#                       lambda = 0,
#                       print_every_n = 10,
#                       objective = "binary:logistic",
#                       nthread = 4)
#commented out due to training time
xgb_default = readRDS("xgb_default.rds")
```

The code above fits an XGBoost classification model with a maximum tree depth of 5, learning rate of .10 (this helps our model be less prone to overfitting), a regularization weight of zero, and 60 boosting iterations. Let's go ahead and take a look at the error rate table of this model.

Plot Error Rate Table

```
xgb_test_preds = predict(xgb_default,
                         dtest)
xgb_class_preds = as.factor(ifelse(xgb_test_preds > .10,#want to minimize type 1 error
                         "Default",
                         "Not Default"))

xgb_conf_matr = confusionMatrix(xgb_class_preds,
                                lending_data_test_scaled$loan_status_final,
                                positive = "Default")
```

```
## Warning in confusionMatrix.default(xgb_class_preds,
## lending_data_test_scaled$loan_status_final, : Levels are not in the same
## order for reference and data. Refactoring data to match.
```

```
xgb_conf_matr$table
```

```
##              Reference
## Prediction    Not Default Default
##   Not Default      313065     372
##   Default             117   78362
```

Looking at the table above, we are still facing the issue of misclassifying "Default" as "Not Default". I have chosen a cutoff of .10 as the classifier due to wanting to minimize type 1 error, or minimize how many actual "Default" we classify as "Not Default". Overall, we have a very small number of misclassifications and this model seems to be performing very well on unseen data. Let's go ahead and compare it to other models we have created.

Comparison to Other Models

```
xgb_test_error = mean(xgb_class_preds != lending_data_test_scaled$loan_status_final)
xgb_test_accuracy = mean(xgb_class_preds == lending_data_test_scaled$loan_status_final)

xgb_test_error_comparison = data.frame("XGB" = c("Accuracy on Test Data" = xgb_test_accuracy,
                                                 "Error on Test Data" = xgb_test_error),
                                       "Boosted" = c("Accuracy on Test Data" = boost_test_accuracy,
                                                     "Error on Test Data" = boost_test_error),
                                       "RandomForest" = c("Accuracy on Test Data" = rf_test_accuracy,
                                                          "Error on Test Data" = rf_test_error),
                                       "Bagging" = c("Accuracy on Test Data" = bag_test_accuracy,
                                                     "Error on Test Data" = bag_test_error),
                                       "Ridge" = c("Accuracy on Test Data" = ridge_test_accuracy,
                                                   "Error on Test Data" = ridge_test_error),
                                       "LASSO" = c("Accuracy on Test Data" = lasso_test_accuracy,
                                                   "Error on Test Data" = lasso_test_error))
kable(xgb_test_error_comparison)
```

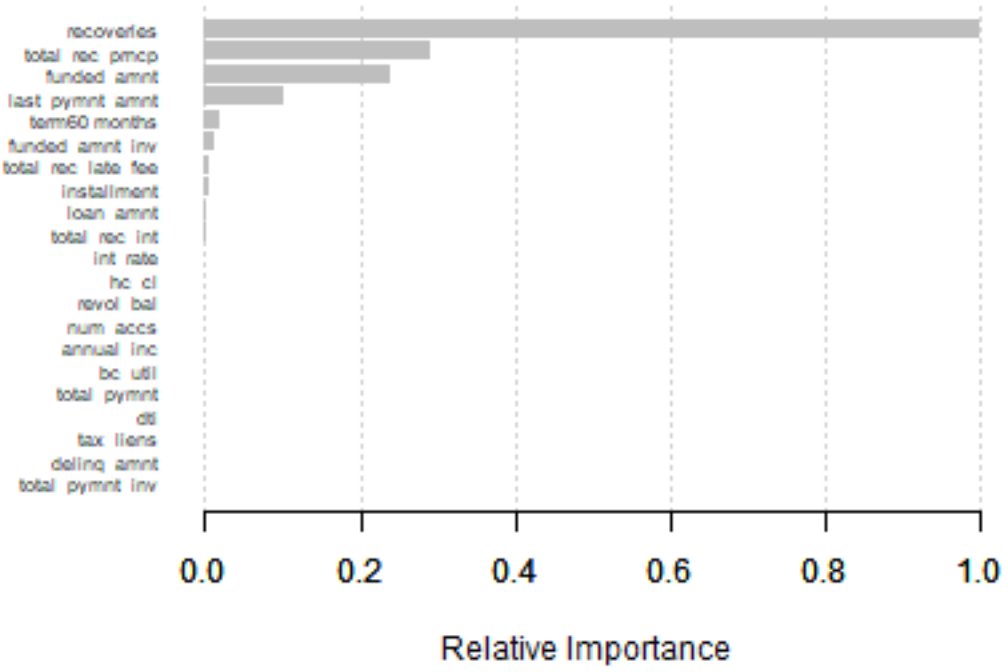
	XGB	Boosted	RandomForest	Bagging	Ridge	LASSO
Accuracy on Test Data	0.9987523	0.9984384	0.9993519	0.9992473	0.999579	0.9994642
Error on Test Data	0.0012477	0.0015616	0.0006481	0.0007527	0.000421	0.0005358

Looking at the table above, we can see that our XGB model outperforms our boosted model. However, our XGB model is still being outperformed by our random forest, bagging, ridge, and lasso models. While being outperformed, this model is not being outperformed by much still making it a viable and accurate model. We must also take into consideration training time, in which XGBoost excels in. We also must take into consideration our classifier we built, which only classifies as "Default" if the predicted probability is greater than .10. With some tuning, our model may become more accurate.

Let's take a quick look at the importance matrix for this model.

Plot Importance Matrix

```
importance = xgb.importance(colnames(X_train), model = xgb_default)
xgb.plot.importance(importance, rel_to_first=TRUE, xlab="Relative Importance")
```



Looking at the table and plot above, we see once again that the three most important variables are `recoveries`, `funded_amnt`, and `tot_rec_prncp`. These three variables have the largest relative importance. Relative importance can be described as a metric computed by three different metrics, `gain`, `coverage`, and `frequency`. With our models agreeing on variable importance, this leads us to believe that these variables truly have high influence on a default outcome.

With XGBoost trained and evaluated, I will be moving forward from decision trees into Neural Networks. Let's go ahead and train a neural network on the data now.

Neural Net

```
levels(lending_data_train_scaled$loan_status_final
       )[levels(lending_data_train_scaled$loan_status_final)=="Not Default"] = "NotDefault"
levels(lending_data_test_scaled$loan_status_final
       )[levels(lending_data_test_scaled$loan_status_final)=="Not Default"] = "NotDefault"

train_labels = lending_data_train_scaled$loan_status_final
train_labels = ifelse(train_labels == "Default",
                      1, 0)
train_data = model.matrix(loan_status_final ~ ., data = lending_data_train_scaled)

test_labels = lending_data_test_scaled$loan_status_final
test_labels = ifelse(test_labels == "Default",
```

```
                      1, 0)
test_data = model.matrix(loan_status_final ~ ., data = lending_data_test_scaled)
```

```
set.seed(490)
nn_model = keras_model_sequential() %>%
            layer_dense(units = 1000, activation = "relu",
                        input_shape = dim(train_data)[2]) %>%
            layer_dense(units = 500, activation = "relu") %>%
            layer_dense(units = 250, activation = "relu") %>%
            layer_dense(units = 100, activation = "relu") %>%
            layer_dense(units = 50, activation = "relu") %>%
            layer_dense(units = 2, activation = "softmax")
nn_model %>% compile(
  loss = 'sparse_categorical_crossentropy',
  optimizer = optimizer_sgd(),
  metrics = c('accuracy')
)

early_stop = callback_early_stopping(monitor = "val_loss",
                                     patience = 20)
epochs = 100
nn_model_fit = nn_model %>% fit(
  train_data,
  train_labels,
  epochs = epochs,
  validation_split = .25,
  callbacks = list(early_stop)
)
keras_save(nn_model, "nn_model.h5")
#not run due to training time..
```

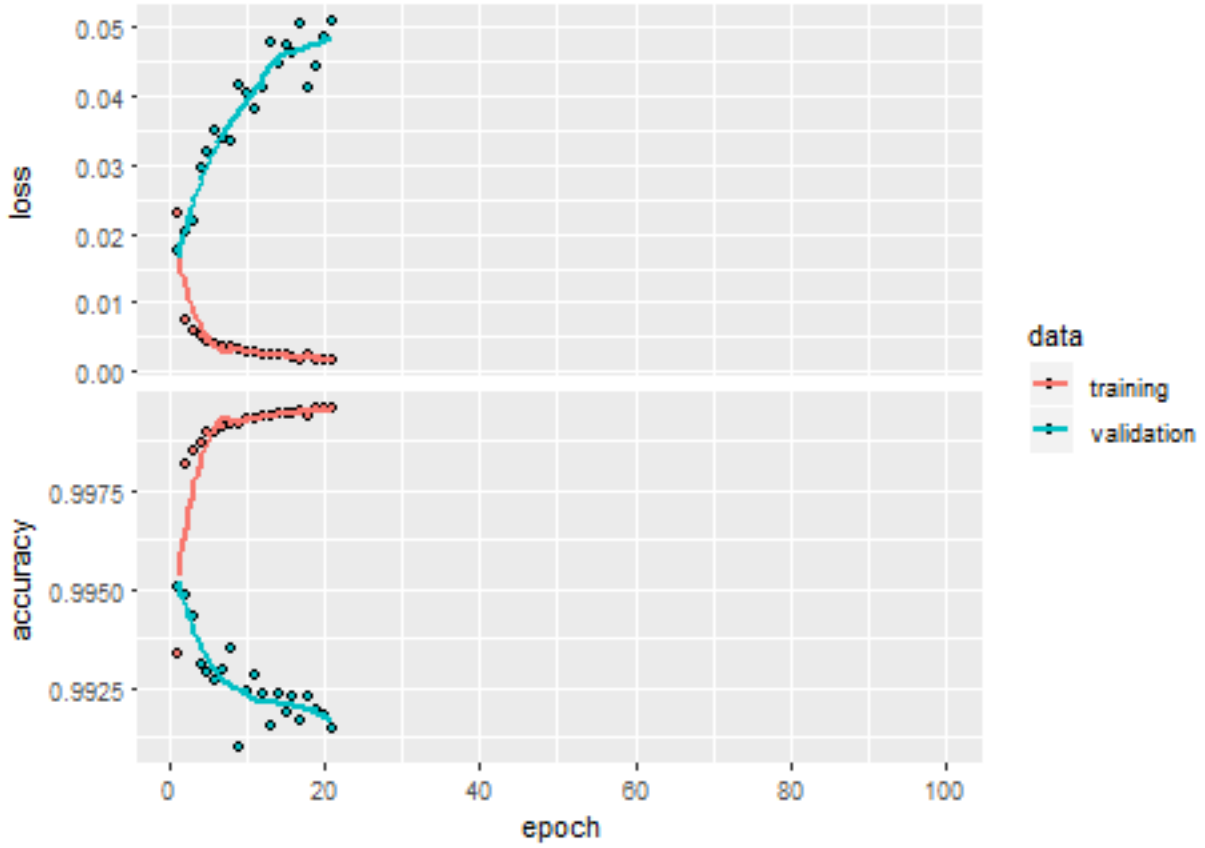
```
nn_model = load_model_hdf5("nn_model.h5")
nn_model_fit = readRDS("nn_model_fit.rds")
```

The code above trains a sequential neural network with five hidden layers, with 1000, 500, 250, 100, and 50 respectively. The loss function being used is `sparse_categorical_crossentropy`, which allows us to use dummy variables as labels (integers) and keep cross entropy as our loss function. Cross-entropy is a measure of the difference between two probability distributions for a given random variable. It can be written as $-\sum_x p(x) * log(q(x))$ where p(x) is the wanted probability (either 0 or 1 for each given training instance) and q(x) is the actual probability. For the hidden layers, I am using the `relu` activation function. The `relu` activation function is linear for all positive values, and zero for all negative values. This will help with our model avoiding overfitting. I am using `softmax` for the activation function of the output layer, which normalizes K numbers into K probabilities. We are using accuracy for our model fitness measurement.

For each epoch, I am generating a loss and accuracy for our training sample and for our validation sample. Our validation sample is 25% of our training data. I am utilizing early stopping for when our model reaches a minimum and stops gaining in accuracy performance. With the model trained and explained, let's go ahead and plot our model against the number of epochs to tune for the number of epochs.

Plotting Model Tuned on Epochs

```
plot(nn_model_fit)
```



Looking at the plot above, we can see that our validation loss has stopped improving around epoch 19-20. So, our model has stopped training at epoch number 21. With the model trained and tuned, let's go ahead and take a look at the accuracy on the test data and see if it similar to the observed validation data.

Test Data Performance

Let's first predict on the test data and look at the confusion matrix of our model.

```
set.seed(490)
nn_test_predictions = nn_model %>% predict_classes(test_data)

nn_class_predictions = factor(ifelse(nn_test_predictions == 1,
                                     "Default",
                                     "NotDefault"),
                                     levels = c("NotDefault", "Default"))

nn_test_accuracy = mean(nn_class_predictions == lending_data_test_scaled$loan_status_final)
nn_test_error = mean(nn_class_predictions != lending_data_test_scaled$loan_status_final)
nn_conf_mtx = confusionMatrix(nn_class_predictions,
                                     lending_data_test_scaled$loan_status_final,
                                     positive = "Default")
nn_conf_mtx$table
```

```
##             Reference
## Prediction   NotDefault Default
##   NotDefault     310773     235
##   Default          2409   78499
```

Looking at the confusion matrix above, we are overwhelmingly correctly predicting `Default` and `Not Default` for our test data. However, we are still predicting `Not Default` on a number of `Defaults`. For this model, it seems that our misclassifications are more even across both levels rather than heavily predicting the majority class like our other model. This shows that this model has less susceptible to imbalanced data. Overall, our model is performing very well on the test data. Let's now compare it's performance to a few other models we have created.

```
nn_test_err_comparison= data.frame("NeuralNet" = c("Accuracy on Test Data" = nn_test_accuracy,
                                                   "Error on Test Data" = nn_test_error),
                                   "XGB" = c("Accuracy on Test Data" = xgb_test_accuracy,
                                             "Error on Test Data" = xgb_test_error),
                                   "Boosted" = c("Accuracy on Test Data" = boost_test_accuracy,
                                                 "Error on Test Data" = boost_test_error),
                                   "RandomForest" = c("Accuracy on Test Data" = rf_test_accuracy,
                                                      "Error on Test Data" = rf_test_error),
                                   "Ridge" = c("Accuracy on Test Data" = ridge_test_accuracy,
                                               "Error on Test Data" = ridge_test_error),
                                   "LASSO" = c("Accuracy on Test Data" = lasso_test_accuracy,
                                               "Error on Test Data" = lasso_test_error))
kable(nn_test_err_comparison)
```

	NeuralNet	XGB	Boosted	RandomForest	Ridge	LASSO
Accuracy on Test Data	0.9932537	0.9987523	0.9984384	0.9993519	0.999579	0.9994642
Error on Test Data	0.0067463	0.0012477	0.0015616	0.0006481	0.000421	0.0005358

Looking at the table above, we can see that our neural network is being outperformed by a lot of previous models. Our best performing models are still Ridge/LASSO regressions, which continues to suggest that our data is of linear form and linear models work best. However, possibly with more layers/nodes in our neural network we could possibly outperform the Ridge and LASSO regressions. We must also consider the support vector machine, which separates data by hyperplanes which could possibly outperform our regression models.

Overall, our neural network performs very well however is outeprformed by a number of previous models which suggests our data may be linear.

With our neural network trained and evaulated, I will be moving onto the last model type being trained in this report. This model is the Support Vector Classifier, let's go ahead and train this model and then evaluate.

Support Vector Classifier

Note, due to training time for this algorithm I will be using a smaller training sample.

```
set.seed(490)

svc_sample_index = createDataPartition(
  lending_data_final$loan_status_final,
  p = .075,
  list = FALSE)

svc_tune_sample = lending_data_final[svc_sample_index,]
```

```
svc_train_index = createDataPartition(
  svc_tune_sample$loan_status_final,
  p = .70,
  list = FALSE)

svc_tune_train = svc_tune_sample[svc_train_index,]
svc_tune_test = svc_tune_sample[-svc_train_index,]

num_columns_svc = colnames(svc_tune_train %>% select_if(is.numeric) %>% select(-c(out_prncp, out_prncp_

scales_svc = build_scales(svc_tune_train,
                      num_columns_svc,
                      verbose = TRUE)

svc_tune_train_scale = fastScale(svc_tune_train,
                                 scales = scales_svc,
                                 verbose = TRUE)
svc_tune_test_scale = fastScale(svc_tune_test,
                                scales = scales_svc,
                                verbose = TRUE)

svc_tune_train_scale = svc_tune_train_scale %>% select(-sub_grade)
svc_tune_test_scale = svc_tune_test_scale %>% select(-sub_grade)

svc_train_labels = svc_tune_train_scale$loan_status_final
svc_train_labels = ifelse(svc_train_labels == "Default",
                      1, 0)
svc_train_data = model.matrix(loan_status_final ~ ., data = svc_tune_train_scale)

svc_test_labels = svc_tune_test_scale$loan_status_final
svc_test_labels = ifelse(svc_test_labels == "Default",
                      1, 0)
svc_test_data = model.matrix(loan_status_final ~ ., data = svc_tune_test_scale)


##THIS CODE IS NOT RUN DUE TO EXTENSIVE TRAINING TIME AND
##UNEEDED DATA USED FOR TUNING
```

```
set.seed(490)

svc_model = svm(svc_train_data, svc_train_labels,
                scale = FALSE,
                kernel = 'radial',
                gamma = 1,
                cost = 1)
#not run due to training time..
```

```
svc_model = readRDS("svc_model.rds")
```

The code above creates a support vector classifier using a sample of our original data with a `radial` kernel, a gamma value of `1`, and a cost of `1`. The radial kernel allows non-linearity in the data, and is more flexible than a linear kernel where data must be separable by a linear hyperplane. The `radial` kernel has a functional form of $K(x_i, x_i') = exp(-\gamma \sum_{j=1}^{p}(x_{ij} - x_{ij}')^2)$, where γ is a tuning parameter which accounts for

the smoothness of the decision boundary and controls the variance of the model. Since we have set our value of $\gamma = 1$, our decision boundary will be smoother and have a lower variance. Lastly, the cost parameter C, in essense, bounds the sum of the total violations ε_i where $\varepsilon_i > 0$ to be $\sum_{i=1}^{n} \varepsilon_i < C$. With the code above ran and the model trained, let's go ahead and evaluate it on test data.

Test Data Performance

Let's first predict on the test data and look at the confusion matrix of our model.

```
#svc_test_predictions = predict(svc_model, test_data)
svc_test_predictions = readRDS("svc_test_predictions.rds")
svc_class_preds = as.factor(ifelse(svc_test_predictions > .40,#want to minimize type 1 error
                          "Default",
                          "NotDefault"))
svc_test_error = mean(svc_class_preds != lending_data_test_scaled$loan_status_final)
svc_test_accuracy = mean(svc_class_preds == lending_data_test_scaled$loan_status_final)

svc_conf_mtx = confusionMatrix(svc_class_preds,
                               lending_data_test_scaled$loan_status_final,
                               positive = "Default")
```

```
## Warning in confusionMatrix.default(svc_class_preds,
## lending_data_test_scaled$loan_status_final, : Levels are not in the same
## order for reference and data. Refactoring data to match.
```

```
svc_conf_mtx$table
```

```
##             Reference
## Prediction   NotDefault Default
##   NotDefault     312874   69527
##   Default           308    9207
```

Overall, this model performs poorly compared to others but could perform well if tuned correctly given time.

This wraps up the model training and evaluation phase, and all models have now been trained and evaluated on the test data. There have been snippets of model comparisons, but not all models have been compared against one another yet. Let's go ahead and perform our final model comparison and choose which model has performed best on our data out of all used.

Comparing All Models

```
final_test_comparison = data.frame("SVC" = c("Accuracy on Test Data" = svc_test_accuracy,
                                             "Error on Test Data" = svc_test_error),
                                   "NeuralNet" = c("Accuracy on Test Data" = nn_test_accuracy,
                                                   "Error on Test Data" = nn_test_error),
                                   "BoostCV" = c("Accuracy on Test Data" = boost_cv_test_accuracy,
                                                 "Error on Test Data" = boost_cv_test_error),
                                   "XGB" = c("Accuracy on Test Data" = xgb_test_accuracy,
                                             "Error on Test Data" = xgb_test_error),
                                   "Boosted" = c("Accuracy on Test Data" = boost_test_accuracy,
                                                 "Error on Test Data" = boost_test_error),
```

```
                                  "RandomForest" = c("Accuracy on Test Data" = rf_test_accuracy,
                                                     "Error on Test Data" = rf_test_error),
                                  "Bagging" = c("Accuracy on Test Data" = bag_test_accuracy,
                                                "Error on Test Data" = bag_test_error),
                                  "DecisionTree" = c("Accuracy on Test Data" = tree_test_accuracy,
                                                     "Error on Test Data" = tree_test_error),
                                  "Ridge" = c("Accuracy on Test Data" = ridge_test_accuracy,
                                              "Error on Test Data" = ridge_test_error),
                                  "LASSO" = c("Accuracy on Test Data" = lasso_test_accuracy,
                                              "Error on Test Data" = lasso_test_error),
                                  "BestLogistic" = c("Accuracy on Test Data" = log_mod5_eval$accuracy,
                                                     "Error on Test Data" = log_mod5_eval$error))
kable(t(final_test_comparison))
```

	Accuracy on Test Data	Error on Test Data
SVC	0.8218113	0.1781887
NeuralNet	0.9932537	0.0067463
BoostCV	0.2658044	0.7341956
XGB	0.9987523	0.0012477
Boosted	0.9984384	0.0015616
RandomForest	0.9993519	0.0006481
Bagging	0.9992473	0.0007527
DecisionTree	0.9813021	0.0186979
Ridge	0.9995790	0.0004210
LASSO	0.9994642	0.0005358
BestLogistic	0.7995642	0.2004358

Looking at the table above, we can see that our ultimate best model when measuring performance on unseen test data is the Ridge logistic regression model. This ultimately confirms the belief that our data is linear, and is best fit with a linear functional form model (parametric).

Our accuracy for this model is 0.999579 and the error rate is 0.000421.

This model slightly outperforms the LASSO model, telling us that the variable reduction attribute to LASSO regression is not useful in our case as many of our variables have at least some importance.

Choosing this Ridge logistic regression model is useful in numerous ways, especially when predicting loan defaults, due to it's direct interpretability of the coefficients which allows us to read into the model and determine what is driving loan defaults.

Conclusion

This wraps up the machine learning project of predicting loan defaults utilizing the Lending Club dataset.

For recaps, I have visited the data and summarized and plotted numerous variables in order to determine importance, correlation, and usefulness to the model and ultimately cleaned the data through those steps. With cleaned data, I have then split the data using a stratification technique which keeps the ratio of `Default` and `Not Default` observations very similar in both the training and testing dataset. The data was then standarized so the numeric variables had a mean of zero and variance of 1 in order to prevent outlier effects from causing trouble in our models.

I then have trained numerous models in order to predict on our data and ultimately determine the best model for predicting loan default. These models are:

- Logistic Regression
- Ridge Logistic Regression
- Lasso Logistic Regression
- K-Nearest-Neighbors
- Decision Tree
- Bootstrap-Aggregated Decision Trees
- Random Forest
- Boosted Decision Trees
- XGBoost
- Neural Network
- Support Vector Classifier

These models all proved to be mostly accurate in answering our original questions of "What drives a loan default?" and "Are we able to create models utilizing loanee attributes to successfully predict loan defaults?" Through the use of coefficient interpretation and variable importance plots, we have answered these questions in most models excluding the Neural Network and Support Vector Classifier.

Ultimately, we have chosen a parametric model of linear functional form to be the best and most accurate model in predicting loan defaults. This model is the Ridge Logistic Regression. Recall that the Ridge Logistic Regression has an objective function of $\sum_i^n (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^p \beta_j^2$. This function is similar to the original objective function of a linear regression, but has a regularization term which shrinks coefficients towards zero. This model has proven to be the best in answering our question of interest.

This project has shown the advantages and disadvantages of classification models, and which classification model performs best on this data. We now have a greater understanding of what is driving loan defaults, and how accurate of models we can create to predict and correctly classify loan defaults which directly can help companies from a business perspective.