Using a Statistical Model to Predict Loan Status

Section 1: Executive Summary

The bank currently experiences lost profit due to not accurately predicting which individuals will default on their loans. From an example filtered dataset of over 34,000 loans with a fully paid or defaulted status, it was determined loan defaults happened around 20% of the time, resulting in lost profit of over $11 million dollars for the bank. If the bank can accurately predict which loans are going to default based on key predictor data, it would be able to recover some of the lost profit, by not awarding those loans.

To solve this problem, a rigorous analysis was done to build a statistical model with predictor variables. The model outputted probabilties based on the variable values, which then was used to predict loan defaults. From the dataset mentioned above, there was over 30 potential predictor variables. To start, the data was cleansed and transformed for additional clarity, and then each predictor was analyzed for its effect on loan status. The analysis determined that total credit limit for an individual and bank loan grade were the predictors which showed the greatest statistical relationship for predicting a loan default, making them the best choice for our model predictor variables. The next step was then to pick a probability threshold for loan defaults. The probability threshold is used to classify whether the model is predicting a loan default or a fully paid loan. Probabilities from the model greater than this threshold would be a predicted loan default, while probabilities under the threshold would be a predicted fully paid loan.

Two separate methods were looked at to select the appropriate threshold. The first method was maximized loan status accuracy, which measured how accurate our model and threshold was at predicting the loan status. This was done by comparing the prediction against the known loan status from our dataset. The second method was maximized bank profit, which measured the profit the bank would make by only awarding loans the model and threshold predicted as being fully paid, and dropping the predicted defaulted loans. For loan status accuracy, the optimized threshold was 0.49 which gave an approximate 79% accuracy at predicting loan status, broken down into being 98% accurate for predicting fully paid loans, and only 6% accurate for predicting defaulted loans. The overall 79% accuracy was due to nearly 80% of loans being fully paid from our dataset. For maximized bank profit, the optimized threshold was 0.28, which was approximately 71% accurate at predicting loan status. This broke down into an approximate 75% accuracy at predicting fully paid loans, and a nearly 50% accuracy at predicting defaulted loans.

From this analysis, it is strongly recommended the bank explores using the statistical model at the 0.28 probability threshold for the purpose of maximizing profit. While the loan status accuracy is not optimized at this threshold, defaulted loans are able to be predicted much more accurately at nearly 50% as compared to 6% at the 0.49 threshold level. The 0.28 probability threshold allowed the bank to recover over $2 million dollars in lost profit in our dataset, with the bank awarding 30% less loans using this method as compared to the current lending practice.

Section 2: Introduction

For this analysis, we will look to predict loan default status based on an analysis of a dataset titled "loans50k.csv." The method for the analysis will be logistic regression, with a response variable for loan default status, and relevant predictor variable(s) identified from graphical and numerical analysis. The output of the logistic regression model will be the probability for the loan default. Functions from the R packages "ggformula", "dplyr","reader", and "gridExtra" will be utilized for this analysis.

The predictor variable(s) may be categorical or quantitative in nature. We will look to first load the dataset in R, identify additional categorical variables, and filter out rows and columns which will not be helpful for the analysis. We will then look to analyze predictor variables which may have an effect on the response variable. Then, we will attempt to fit a model based on the most relevant predictor variable(s), optimize it, and report the results.

Section 3: Preparing and Cleansing the Data

As mentioned, the first step is to load the dataset into R. We load the dataset into R with the read.csv command, and get dimensions and column names:

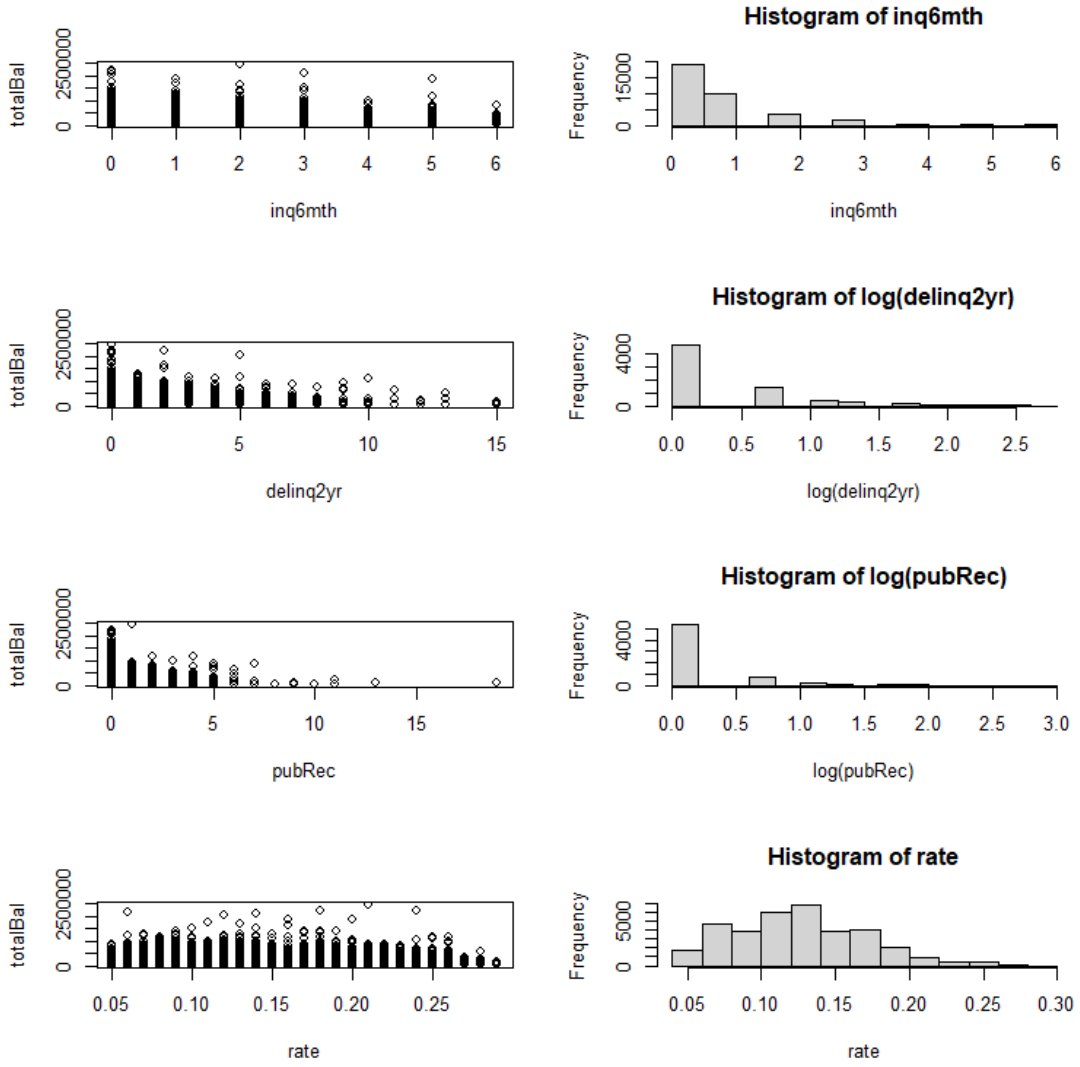
```
loan_data=read.csv("loans50k.csv")
dim.data.frame(loan_data)

## [1] 50000    32

colnames(loan_data)

##  [1] "loanID"      "amount"      "term"        "rate"        "payment"
##  [6] "grade"       "employment"  "length"      "home"        "income"
## [11] "verified"    "status"      "reason"      "state"       "debtIncRat"
## [16] "delinq2yr"   "inq6mth"     "openAcc"     "pubRec"      "revolRatio"
## [21] "totalAcc"    "totalPaid"   "totalBal"    "totalRevLim" "accOpen24"
## [26] "avgBal"      "bcOpen"      "bcRatio"     "totalLim"    "totalRevBal"
## [31] "totalBcLim"  "totalIlLim"
```

We note the dataframe dimensions of 50,000 rows and 32 columns, with each column having data which may potentially be useful in predicting loan defaults.

Next, we look to build our response variable and add it to our dataframe as a new field. The response variable will be called "res_var", and will have a status of "Good" if the loan "status" is "Fully Paid", or "Bad" if the loan "status" is "Charged Off" or "Default." The response variable will be created via a "for" loop. We will filter out all other rows in our loan_data dataframe with a different loan status than the 3 above. This reduces the dataframe to 34,655 rows.

Furthermore, we also look to immediately eliminate columns which will not be useful for our analysis. These columns are loanID, employment, and state. LoanID is just a transaction log and serves no purpose, while employment has thousands of unique values and we already have income data as a column, while state has many unique values and we already have the loan interest rate as a column.

The next step is to identify numerical variables which may actually be categorical. Numerical variables which may potentially be categorical were plotted next:



Based off the scatter plots verse a quantitative variable (totalBal) and histograms, we conclude that variables delinq2yr,pubRec, and inq6mth act as categorical, since they are non-continuous. Careful consideration was given to whether rate should be considered categorical given its scatterplot, but its histogram ultimately indicates it is continuous in nature. Furthermore, interest rate can be used on a numerical basis, for example, calculating interest.

Now that we identified all possible categorical variables, the next step is for some feature engineering on those categories. The goal of the feature engineering is to combine or eliminate categorical levels which are infrequent to simplify the dataset. To do this, we created a dataframe for each categorical variable seen in our loan_data dataframe. The categorical dataframe included number of observations of each categorical level, as well as the associated percentage. The example results for the “reason” and “status” categorical

dataframes are shown below, along with example code how each categorical dataframe was created:

```
#Example code to create df_term categorical dataframe with observation and pe
rcent.
df_term<-loan_data %>% group_by(term) %>% summarise(N=n())
#create a function so we can get a percent of each level in the categorical v
ariable.
percent_function<- function(col_data) {
  col_length=length(col_data)
  return_vec<-c()
  for (i in 1:col_length) {
  temp_val=paste0(round(col_data[i]/sum(col_data)*100,2),"%")
  return_vec<-c(return_vec,temp_val)
}
  return(return_vec)
}
#add the percent to the categorical dataframe.
df_term<-mutate(df_term,Percent=percent_function(df_term$N))
```

```
## # A tibble: 13 x 3
##    reason                N Percent
##  * <chr>             <int> <chr>
##  1 car                 281 0.81%
##  2 credit_card        7843 22.63%
##  3 debt_consolidation 21046 60.73%
##  4 home_improvement    2021 5.83%
##  5 house                126 0.36%
##  6 major_purchase       619 1.79%
##  7 medical              378 1.09%
##  8 moving               215 0.62%
##  9 other               1571 4.53%
## 10 renewable_energy      27 0.08%
## 11 small_business       317 0.91%
## 12 vacation             209 0.6%
## 13 wedding                2 0.01%
```

```
## # A tibble: 3 x 3
##   status          N Percent
## * <chr>       <int> <chr>
## 1 Charged Off  7579 21.87%
## 2 Default         2 0.01%
## 3 Fully Paid  27074 78.12%
```

As a result of these categorical dataframes, the following feature engineering took place to simplify our dataset:

- Remove delinq2yr and pubRec variables from the dataset. Each were redundant and had nearly 80% of its observations for one categorical level, making them unlikely to be useful.

- For status category, remove “Default” level since it only had 2 observations, which was only 0.01% of the data.
- For grade category, combine “F” and “G” levels into “Other-Risky” level, since both were relatively infrequent observations (less than 3.5% combined.)
- For reason and inq6mth categories, combine any level which had observations less than 2% of all observations into “other” level for each category.

Now that we have done our feature engineering, the next steps is to identify number of rows and columns with missing values. We do this using colSums(is.na(loan_data)) and length(loan_data[rowSums(is.na(loan_data))>0,]$amount) . Below is a snapshot of missing values by column, as well as the number and percent of rows with missing values:

```
##      amount        term        rate     payment       grade      length
##           0           0           0           0           0           0
##        home      income    verified      status      reason  debtIncRat
##           0           0           0           0           0           0
##     inq6mth     openAcc  revolRatio    totalAcc   totalPaid    totalBal
##           0           0          15           0           0           0
## totalRevLim   accOpen24      avgBal      bcOpen     bcRatio    totalLim
##           0           0           0         360         384           0
## totalRevBal  totalBcLim  totalIlLim     res_var
##           0           0           0           0

## [1] "The number of rows with missing values is 384 which accounts for appr
oximately 1% of all the data."
```

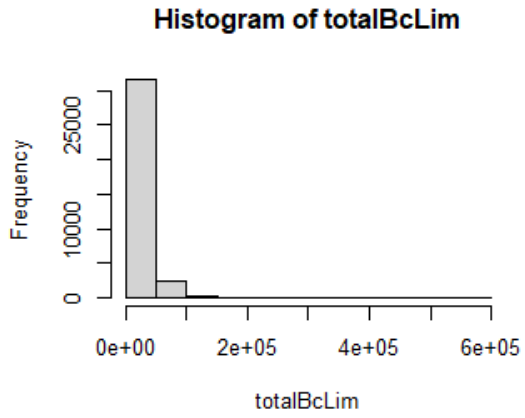
As you can see there is only 3 columns with missing values, revolRatio, bcOpen, and bcRatio. Furthermore, the total number of rows with missing data is 384, meaning the missing values between rows are related(i.e. for every instance of a missing value for revolRatio or bcOpen there will be an associated missing value for bcRatio.) The 384 rows which contain the missing values accounts for only 1% of the rows in the entire loan data set. Given the small percentage amount, I think deleting the data is acceptable here since 1% of data is not likely to make an impact on our model.

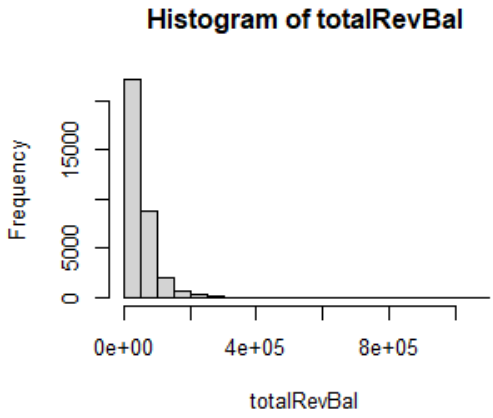
```
#Filter our rows with NA values.
loan_data<-loan_data[rowSums(is.na(loan_data))==0,]
```

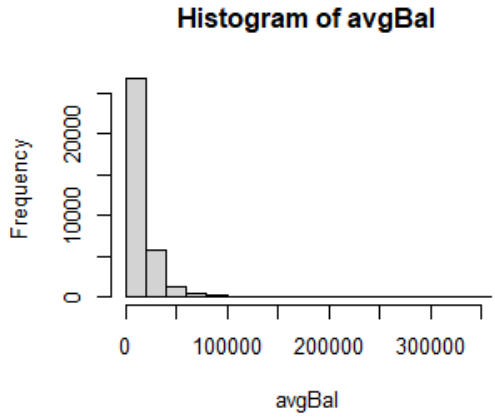
After feature engineering and data cleansing, the final dimensions of our loan dataframe is 34,269 rows, and 28 columns. Of the 28 columns, 10 are categorical, while 18 are quantitative.

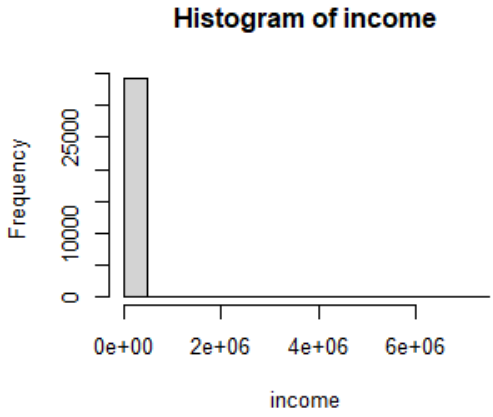
Section 4: Exploring and Transforming the Data

The first objective here is to identify quantitative variables which show a high amount of skewness. From a model perspective, extreme skewness can influence model predictive results thus ultimately making them not usable. Each quantitative variable was plotted via a histogram, and its distribution was analyzed. Most of the quantitative variables plotted showed at least mild skewness, however the below are examples plots of variables which showed extreme enough skewness to be considered for data transformation.





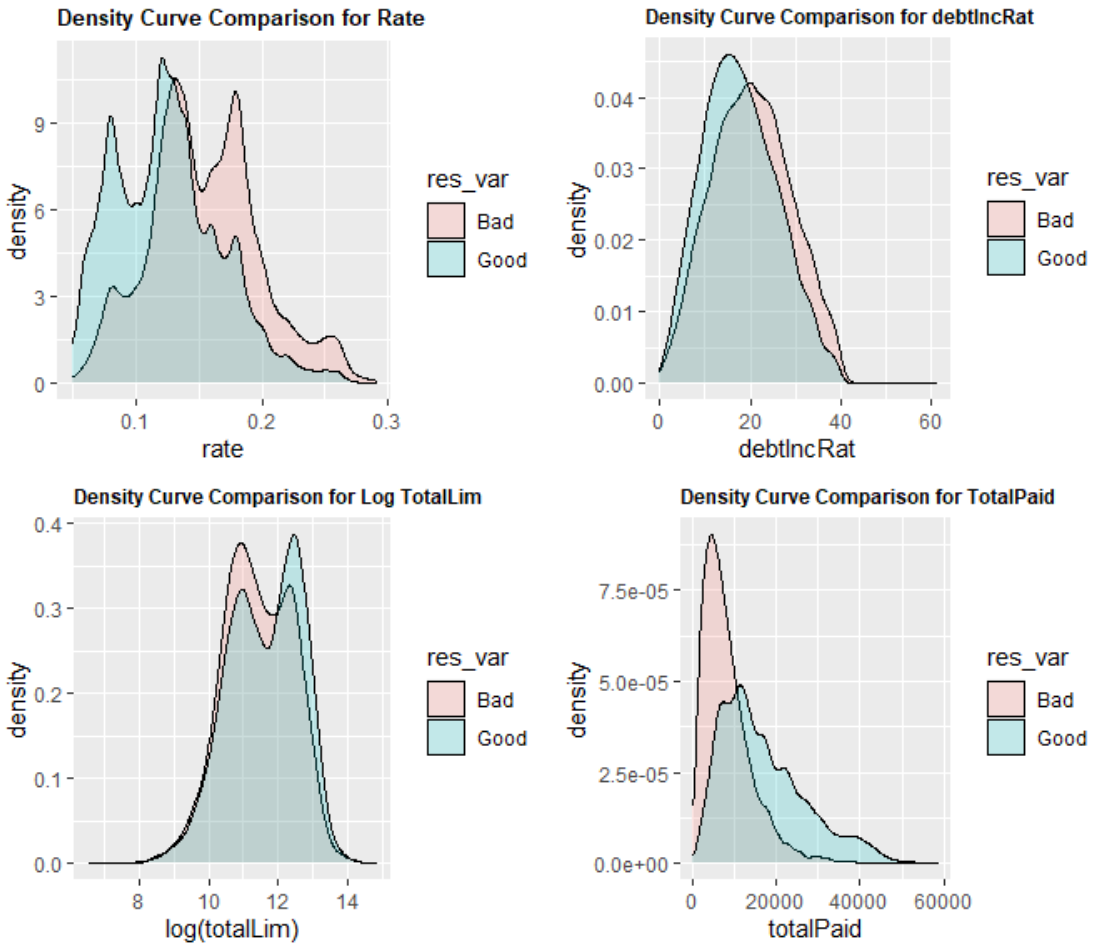




Variables which showed extreme skewness when histogram plotting include totalBcLim, totalRevBal, avgBal, Income, totalLim, totalIlLim, bcOpen, totalRevLim, and totalBal. As a result, a log transformation will be applied to these variables going forward, since it is typically used for data skewness for its effectiveness and simplicity. We also verify that the log transformation does not introduce NA values into the data set.

Next, I explored the relationship of each quantitative variable to our response variable, res_var, which had two statuses of "Bad" or "Good" for Loan status. I did this via a density curve comparison for each of our quantitative variables, split into two groups,"Good" and "Bad" loan status. Below are plots of significance from the density curve analysis.

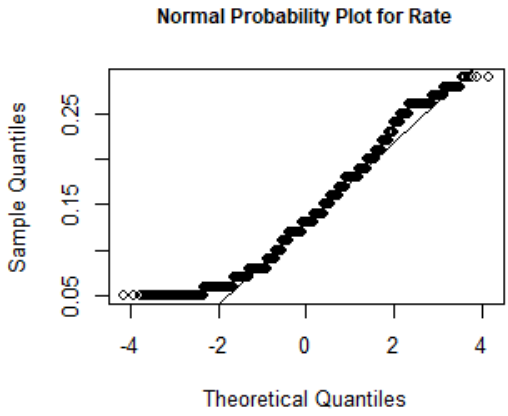
```
#Example code to create one density curve comparison for rate and loan status
, grid.arrange used for side-by-side plotting.
  example<-ggplot(loan_data) + geom_density(aes(x = rate, fill = res_var), al
pha = 0.2)+ggtitle("Density Curve Comparison for Rate")+
  theme(plot.title = element_text(size = 10, face = "bold"))
```

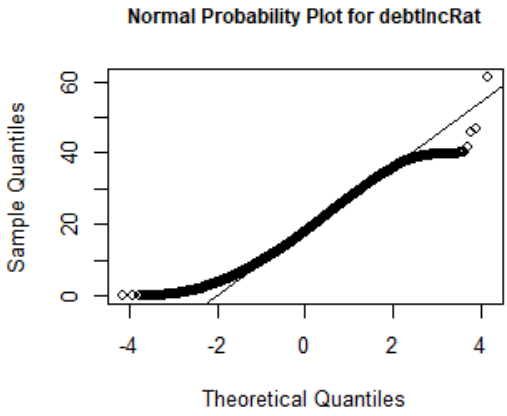


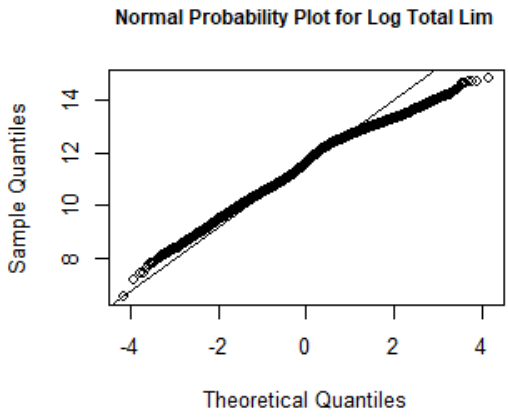
For density curves, we are looking for non-overlapping regions and shifts in distribution between “Good” and “Bad” loans. You can see significant non-overlapping regions in the graphs for “ToalPaid” and “rate.” Conversely, you can see the distribution peaks are notably shifted in the log(totalLim) graph for “Good” and “Bad” loans. However, we note that the distributions for these graphs do show skewness and non-normality. For the debtIncRat graph, there is more overlapping region and less significant distribution shift between “Good” and “Bad” loans, but each density curve seems to follow a more classic normal distribution.

It is noted that predictor variables are not technically required to be normally distributed for logistic regression, but it is ideal for the predictor variables to have lesser skewness, since it will give more consistent model results. From this, we need to find a way to compare skewness amongst our four quantiative variables.

As a result of this need to identify skewness, the next section will be a normal probability plot comparison for the quantitative variables of rate, debtIncRat,log(totalLim), and as our potential predictors. TotalPaid cannot be a predictor, but its graph was included in this analysis.







As expected, each plot showed evidence of skewness. However, the normal probability plots for log(totalLim) and debtIncRat do not seem to show significant differences in skewness. Given that the shift for distribution central values between "Good" and "Bad" loans is much more apparent for log(totalLim), this may mean it would be a more effective predictor than debtIncRat if skewness is similar. As a result, we will also eliminate debtIncRat as a potential predictor variable, leaving us only with log(totalLim) and rate.

Next, we will use proportional bar chart comparisons for our categorical variables, with the fill being our response variable,res_var. This will allow us to see if there are any significant proportional differences between "Good" or "Bad" loans among the categorical levels for each categorical variable. Below are plots of significance from the categorical proportional analysis.

```
#Example code to create one proportional bar chart for Verified and loan stat
us, grid.arrange used for side-by-side plotting.
  example<-loan_data %>%
    group_by(verified) %>%
    count(res_var) %>%
    mutate(prop = n/sum(n)) %>%
    ggplot(aes(x = verified, y = prop)) +
    geom_col(aes(fill = res_var), position = "dodge") +
```

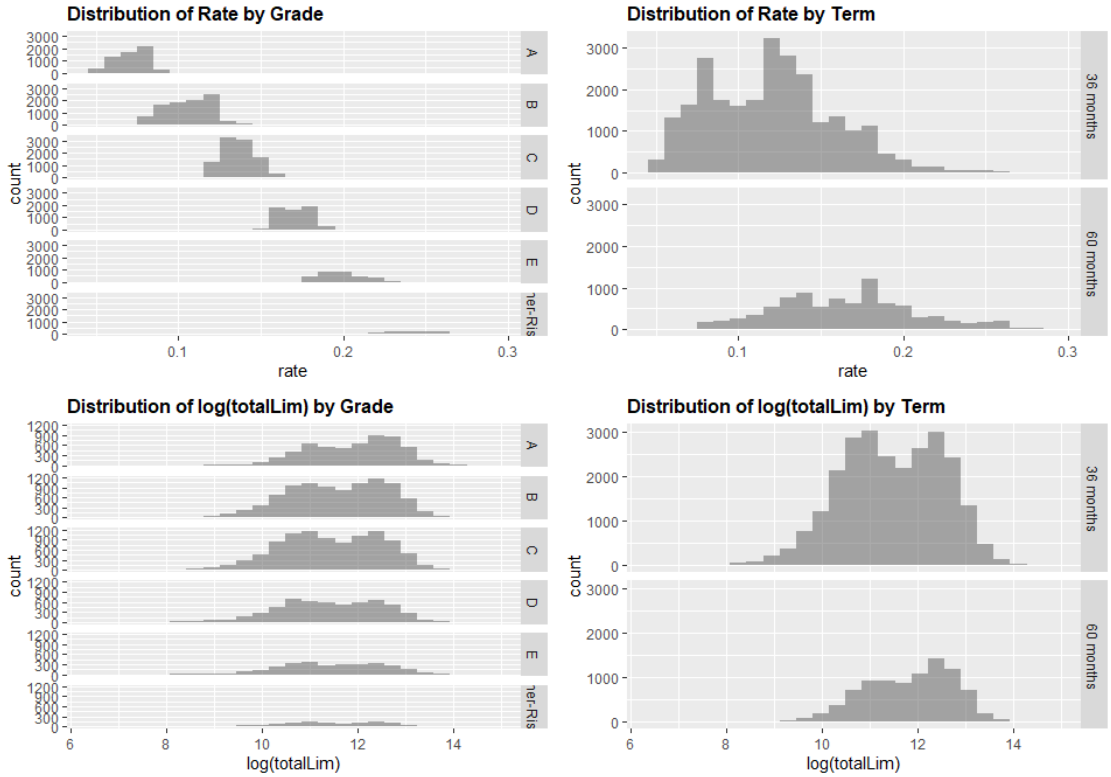
```
    geom_text(aes(label = scales::percent(prop),
                  y = prop,
                  group = res_var),
              position = position_dodge(width = 0.9),
              vjust = 1.5)+ggtitle ("Loan Status Proportion Analysis for Verified")+
   theme(plot.title = element_text(size = 12, face = "bold"))
```

First, we notice the clear trend in the “grade” category how the proportion of bad loans changes by grade level. Just comparing loan grade “A” and the “Other-Risky” levels, you can see the striking difference in proportion between “Good” and “Bad” loans (approximate 43% difference.) This is a strong indicator this category would be informative as a potential predictor variable. Next is the “term” category, there is a clear proportion difference between 36 and 60 months level (approximate 19% difference), and this category is also likely to be informative as a predictor variable. Finally, for the other two categories, “inq6mnth” and “verified”, there is a noticeable difference between proportion levels between the most right and left levels (approximate 10% difference.) However for the purposes of this analysis, we will consider grade and loan term as our categorical variables of most interest.

Finally, now that we have identified our categorical and quantitative variables of most interest, we need to explore the relationships between each predictor variable. We will explore this relationship via a faceted histogram.

```
#Example code to create one faceted histogram for rate and grade, grid.arrange used for side-by-side plotting.
```

```
example<-gf_histogram(~rate, data=loan_data) %>%
gf_facet_grid(grade~.)+ggtitle("Distribution of Rate by Grade")+
 theme(plot.title = element_text(size = 12, face = "bold"))
```



One immediate observation is how the distribution of rate changed with term and grade. This implies that rate,term,and grade are all related to each other. This may not be ideal from a modeling perspective if you have predictor variables which are related to each other, since they may not be adding new valuable information to our model. Conversely if you look at the distribution of log(totalLim), this variable does not seem to significantly change by term or grade category levels. As a result, I am most interested in exploring the log(totalLim) quantitative variable going forward, along with the term and grade categories, for our logistic regression model.

Section 5: The Logistic Model

Now that we have our potential predictors, the next step is to actually build the logistic model. To do this we will split our loan dataset into two portions: training and test. 80% of the data will be randomly selected for the training dataset, while the remaining 20% will go into the test dataset. This is done with the following code:

```
set.seed(321) #set seed so same random data is always selected.
sample_size<-round(length(loan_data$amount)*0.8) #get the sample size needed
for 80% of the data.
training_data<-loan_data[sample(nrow(loan_data), size=sample_size), ] #create
the training dataset.
test_data<-setdiff(loan_data,training_data) #the remaining data goes into the
```

```
test dataset, which is the difference between the entire dataset and training
dataset.
```

Now that we have our datasets, the next step is to actually build our glm model. The first step to build this model is to get the response variable in binary format which is a requirement to use the "glm" function in R studio.

After we have the binary response variable the next step is to actually create the glm model. For this step we will create a glm model involving all our predictor variables in the training dataset, and compare the AIC of each model to determine which model is best.

```
#example code to build a glm model using the log_total_lim and term predictor
variables.
glm_model_1<-glm(status_bin~log_total_lim+term,data=training_data,family="bin
omial")
```

```
##                                                Predictors      AIC
## 1                                term and log(totalLim) 27526.66
## 2                               grade and log(totalLim) 26692.47
## 3                                        log(totalLim) 28733.57
## 4                                              grade 26746.96
## 5                                               term 27873.57
## 6                       term and grade and log(totalLim) 26446.07
## 7 term and grade and log(totalLim) and interaction terms 26458.21
```

Based off the AIC, the best model involving our predictor variables is the one with all three in their first-order form; term, grade, and log(totalLim.) This is the model with the lowest AIC. Our next check is to see if the model suffers from collinearity using the vif formula.

```
require(HH)
vif(glm_model_6)
```

```
##    log_total_lim           gradeB           gradeC           gradeD
##         1.083694         1.954570         2.069708         1.868071
##           gradeE gradeOther-Risky   term 60 months
##         1.619313         1.305783         1.329737
```

Since there is no VIF that is over 10, we conclude this model does not suffer from collinearity and is suitable for further use.

The next step for our selected model is to create a confusion matrix to estimate the accuracy of our model with the test dataset. For our model, we will use a 50% probability threshold to classify if a loan will default, meaning if our logistic model has a probability greater than 50% in a given output we will classify that as a loan default.

```
test_df<-data.frame(log_total_lim=test_data$log_total_lim,grade=test_data$gra
de, term=test_data$term)
predict_df<-predict(glm_model_6,test_df,type="response")

DF_comparison <- data.frame(response = test_data$res_var, predicted = round(p
redict_df))
```

```
comparison_tbl<-xtabs(~ predicted + response, data = DF_comparison)
addmargins(comparison_tbl)

##          response
## predicted  Bad Good  Sum
##       0   1397 5327 6724
##       1     74   56  130
##       Sum 1471 5383 6854

percent_accuracy<-round((comparison_tbl[1,2]+comparison_tbl[2,1])/nrow(DF_com
parison)*100,2)


print(paste0("The accuracy of this model is ", percent_accuracy," % overall")
)

## [1] "The accuracy of this model is 78.8 % overall"
```

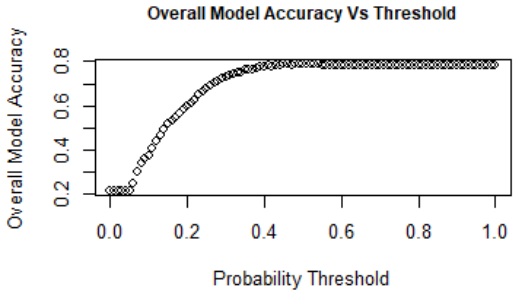
We note that our model has an accuracy of 78.8% overall. By looking at the confusion matrix, we can see the model is overestimating the number of good loans while underestimating the number of bad loans. The difference in accuracy for each type is striking, for bad loans the model is correctly predicting only 74 out of the 1471 bad loans which is only 5% accurate. While for good loans the model is correctly predicting 5327 out of 5383 good loans, which is nearly 99% accurate. This is the reason the model is nearly 80% accurate overall, as good loans make up 5383/6854=78.5% of the overall data. The next step is to find the threshold for optimal model accuracy.

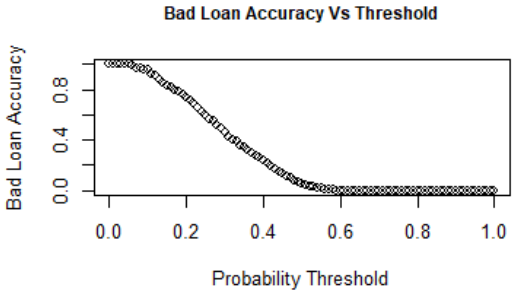
Section 6: "Optimizing the Threshold for Accuracy"

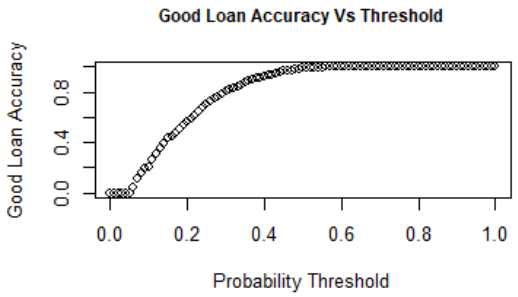
To find the optimized model accuracy for probabilities between 0 and 1, we repeat the steps above, but in a "for" loop for the possible probability thresholds between 0 and 1. We will then have three scatterplots: one for overall model accuracy, one for model accuracy for bad loans, and one for model accuracy for good loans all by threshold level. Furthermore we will also find the threshold which gives the maximum overall model accuracy, and create a dataframe comparing these thresholds.

```
x_data<-seq(0, 1, by = 0.01) #vector for probability thresholds between 0 and
1 in increments of 0.01.
accuracy_vec<-c() #initialize accuracy vector.
bad_vec<-c() #initialize vector for accuracy for bad loans
good_vec<-c() #initialize vector for accuracy for bad loans
for (i in x_data) {
  temp_df<-predict_df
  temp_df[temp_df>i]<-1
  temp_df[temp_df<i]<-0
  temp_comparison <- data.frame(response = test_data$status, predicted = temp
_df)
  temp_comparison_tbl<-xtabs(~ predicted + response, data = temp_comparison)
  if(nrow(temp_comparison_tbl)<2 & i<0.5){
     temp_accuracy=temp_comparison_tbl[1,1]/6854
```

```
    temp_bad=temp_comparison_tbl[1,1]/1471
    temp_good=0
    }
  else if(nrow(temp_comparison_tbl)<2 & i>0.5) {
    temp_accuracy=temp_comparison_tbl[1,2]/6854
    temp_bad=0
    temp_good=temp_comparison_tbl[1,2]/5383
  }
  else {
    temp_accuracy=(temp_comparison_tbl[1,2]+ temp_comparison_tbl[2,1])/6854
    temp_bad=temp_comparison_tbl[2,1]/1471
    temp_good=temp_comparison_tbl[1,2]/5383
  }
    accuracy_vec<-c(accuracy_vec,temp_accuracy)
    bad_vec<-c(bad_vec,temp_bad)
    good_vec<-c(good_vec,temp_good)
} #use x-data, accuracy_vec, bad_vec, good_vec for plots and dataframe.
```







Analyzing the overall accuracy plot yields some interesting conclusions. We can see the model accuracy is low in the lower probability threshold range, then rises dramatically and levels off. This is partly because our predicted probabilities from our model is in the range of 0.04 to 0.69, so any threshold up to 0.04 predicted all bad loans, and any threshold over 0.69 predicated all good loans. We note the maximum accuracy is practically the same as previously seen at a probability threshold of 0.49. At this threshold, good loan accuracy is 98.5%, while bad loan accuracy is 6.45%.

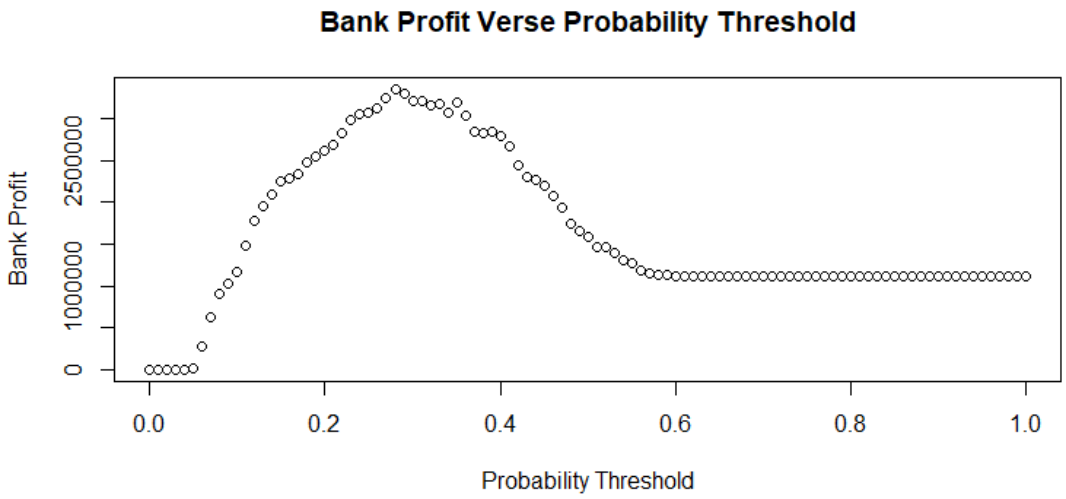
For the main curve, the model accuracy dramatically improves up to a probability threshold of 0.40 then levels off. This is directly related to the tradeoff in accuracy verse

good and bad loans as seen in their opposite trends. Unsurprisingly the accuracy for the overall model closely follows the accuracy for the good loans, since the good loans make up the majority of the data as discussed above. The next step is to optimize the threshold for profit for the bank.

Section 7: "Optimizing the Threshold for Profit"

Optimizing the threshold for profit is similar as we have done for optimizing the threshold for accuracy. We define bank profit as the sum of the totalPaid variable minus the sum of the amount variable. We will compare three scenario's for bank profit:

- Scenario 1: The banks profit given it's current process and no model. The bank does not know which loans are bad, so all loans are included in the profit calculation.
- Scenario 2: The banks profit in the ideal scenario and perfect model. This is assuming the bank can accurately identify all loans which will be bad, and as a result, will only calculate profit using the loans in good standing.
- Scenario 3: The banks profit using our logistic model, but with the probability threshold optimized for maximum profit.




```
profit_df<- data.frame(x=x_data,profit=profit_vec)
profit_3<-profit_df[which.max(profit_df$profit),1:2]
profit_3[1:1]
```

```
##       x
## 29 0.28
```

```
prof_vec<-c(profit_1,profit_2,profit_3[1,2])
profit_comp<-data.frame(Scenario=c("Current","Ideal","Optimized Model Thresho
ld"),Profit_dollars=prof_vec)
profit_comp
```

```
##                   Scenario Profit_dollars
## 1                   Current        1111883
```

```
## 2                      Ideal       12139182
## 3 Optimized Model Threshold        3352848
```

The results here are interesting to analyze. We note the bank has a current profit of approximately 1.1 million dollars, while in the ideal scenario the bank's profit would be approximately 12.1 million dollars, and using our model at a optimized threshold the banks profit would be approximately 3.3 million dollars.

The optimized probability threshold for our model for profit came in at 0.28. This is interesting because that threshold is very different from the one for max accuracy which came in around 0.5. We can see from the scatterplot how the bank profit rises and falls from the 0.28 limit. The difference in these thresholds is directly related to the model's difficulty to predict bad loans at it's optimized threshold level for accuracy. As mentioned previously, there is a clear tradeoff between the accuracy of bad loan predictions verse overall model accuracy. As a result, the 0.28 threshold level seems to be the "sweet spot" for bank profit where overall model accuracy is still high enough while also maintaining a high enough accuracy for bad loans.

Section 8: "Results Summary"

In summary, we built a first order logistic classification model involving the term, grade, and log(totalLim) predictors. This model gave us the lowest AIC out of all potential models involving our predictors. We then made sure the model did not suffer from collinearity. After that, we optimized the model both for accuracy and also for profit. We noted there is a clear tradeoff between the accuracy of bad loan predictions verse overall model accuracy, primarily due to the dataset having around 80% of loans being classified as good. We found the optimized model threshold for accuracy was 0.49, while the optimized threshold for profit was 0.28.

My recommendation on the optimal threshold the model should use would be 0.28. At this threshold the overall model accuracy is about 71%, while the accuracy for bad loans is around 50%, and the accuracy for good loans is around 75%. While I realize the overall accuracy is 8% below the optimized level, this is primarily driven by the accuracy for good loans. At the 0.49 threshold level, the accuracy for good loans is nearly 100%, while for bad loans it is less than a percentage. At this level you can be highly confident you are predicting good loans, but you would have extremely low confidence in predicting bad loans. At the 0.28 level, you can at least have reasonable confidence in predicting both loan types, and as a result I believe that is the optimal threshold level.