Predicting Loan Defaults with Logistic Regression

Josh Jarvey

6/5/2020

# 1.0 Executive Summary

Products within the banking industry are all typically based on a risk assessment, that is to say each offering comes with some sort of risk of “winning” or “losing” on that transaction. For instance, providing investment services to a consumer has an associated risk of increasing or decreasing in value based on numerous factors. When it comes to loans for home, auto, school, personal, etc. these products are no different - they have some associated risk of “paid in full” (with interest collected) or the loan will “default” - the win vs. loss metaphor mentioned earlier.

Historically, the ability to assess risk on loans was something more of an art than a science. Financial bankers would meet with consumers in person, and while it’s true that they would go over various details of finance and stability, the ultimate decision was grounded in intuition and “gut-feel” of that banker (in accordance to bank guidelines of course). Some were very successful at closing the right ones and making a profit for their organization, while others were not as successful. There’s also the ethical dilemma of bias being introduced into gut-feel and intuitive decision making, which can lead to missed revenues, but also potential legal troubles that organizations are wise to mitigate.

In today’s age, the science has caught up with the art - and in many cases has even surpassed it. Where you have had one or two individuals trying to “crunch the numbers” and come up with a decision in the times past, we now have computers to do this work. But where the computers and science really excel at this work is finding the hidden relationships between a consumer’s relevant “parameters” to the loan and aiding in the decision of win vs. loss. It crunches the numbers among thousands upon thousands of historical loan’s parameters, and understanding those outcomes it can provide a sound assessment of risk that’s rooted in science and not intuition.

In this project we will explore the use of one of these scientific tools called Regression. Being that our end goal is to have a decision to “give loan” or “decline loan”, we will use the Logistic version of the regression analysis, which is better suited for a yes/no decision. A more scientific approach to decision making will not only save time and money to improve the bottom line, but will also help to mitigate bias and the associated organizational risk that comes with it.

# 2.0 Introduction

In risk-based decision making, it can be hard for any one human to take into account all the necessary information and variables that go into making the *right* decisions that minimize losses and maximizes revenues. This is where Regression analysis can come into play, because it is designed specifically to tackle the analysis of many variables, at the same time, to predict an outcome. Since the end result of the product in this study is a successful financial loan, we know that the outcome will be one of two choices: either “paid in full” as a success, or “loan default” as a failure. Because of this binary in nature outcome, we will use the Logistic version of the Regression analysis, which is suited well for handling this type of task.

The remainder of this report will outline the steps taken to build, train, and test different candidate models to see which performs the best and will ultimately be released into production to serve its purpose of aiding in the loan decision making process.

As with any data project, we start out by summarizing the dataset of 50,000 observations both numerically and visually to explore it for anomalies - missing data, outliers, etc. - and determine how to best overcome these data quality issues. In the next step we’ll explore the dataset’s 30 unique variables. Some of these variables will be significant toward solving the problem, whereas some will not and will need to be removed from the model or lumped together in an “other” category. Once we’ve landed on the appropriate variables, we’ll take a look at the distribution of its values as data may need to be transformed depending on what is discovered.

In the second half of the project, we will describe fitting the model to the data and accessing its performance. From here we will test at optimizing for prediction accuracy in addition maximizing profit. Finally, we will end with a conclusion and summarize model performance, along with any room for enhancements.

# 3.0 Data Preparation and Cleaning

To get started in this step, first we will load the necessary packages we need in order to perform the data exploration and cleaning.

lapply(c("readr","dplyr","car","mice","ggformula","ggpubr"), require, character.only = TRUE)

Now we can prepare our response variable based on the status column of the existing dataset. We will discard any “in progress” loans, and only keep those that are “complete” - these three status reflect completed loans, leaving 34,655 observations remaining:

loans = loans[loans$status %in% c("Fully Paid", "Charged Off", "Default"),]

Because our response variable must be binary in order to use logistic regression, we will condense the results into a 2-factor variable called “outcome”.

loans = loans %>% mutate(outcome = as.factor(if\_else(status == "Fully Paid","Good","Bad")))

## 3.1 Feature Selection

In an effort to keep parsimony, we will a look at each of these variables in the dataset and evaluate if we expect them to add overall value to the model. To get us started, we can drop the “status” variable since we used it already to create our new response variable.

loans = subset(loans, select = -status)

Below are a list of the other variables we’ll remove from the dataset in the order of which they occur:

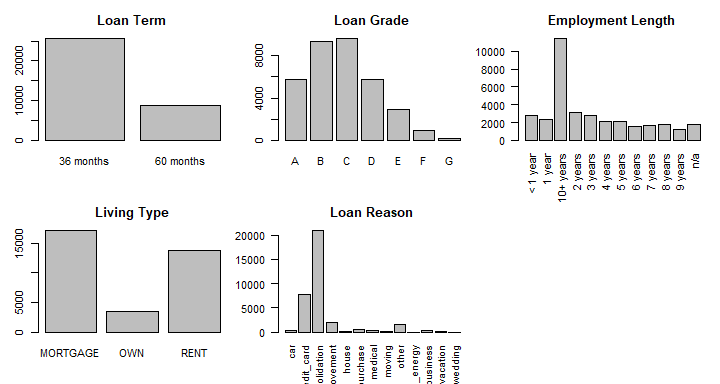
* “**loanID**”: The id in which a loan can be retrieved from the database. This adds no information.
* “**employment**”: A free form entry field that offers no consistency between values. Riddled with short hand and spelling mistakes.
* “**verified**”: Whether the source of income has been verified. If the bank does not have this verified, we would not make the loan.
* “**state**”: With loans in an overwhelming majority of the states, this seems more noise than signal.
* “**debtIncRat**”: Any ratios are a calculation of existing data within this set, so it will be removed since it provides no extra information.
* “**revolRatio**”: Any ratios are a calculation of existing data within this set, so it will be removed since it provides no extra information.
* “**totalAcc**”: Total accounts (including closed) - We’re only interested in open accounts that are currently active.
* “**bcRatio**”: Any ratios are a calculation of existing data within this set, so it will be removed since it provides no extra information.

## 3.2 Feature Engineering

Next we will take a look at the remaining variables with our str() function to check the variable types in our dataset. We see right away there are variables that are not coded appropriately (as a character, instead of as a factor), so we will make those adjustments now before moving forward.

loans = mutate\_at(loans, vars(term,grade,length,home,reason), as.factor)

Now that we have the variable’s types coded correctly, we can review the number and counts (i.e. general distribution) of categories within each of the converted variables using a barplot as a visual.



First, we take a look at the loan term variable. While 36 month loans account for the vast majority, there are only two categories for this variable, so we wont make any adjustments on this one.

Next, we take a look at the grade variable, which ranges in values from A to G. In reviewing the summary, we notice the F and G groups have a considerably less count than the other grades, so we’ll combine these two groups into the “E” grade and form a new category “E or less”.

loans$grade = recode(loans$grade,"c('E','F','G')='E or less'")

The employment length variable has many different categories, and the 10+ year is skewing the distribution, so let’s combine into three categories as follows (we also have NA, and will deal with that in the next section):

* “0-4 years”
* “5-9 years”
* “10+ years”

We now review the home variable, which explains the type of living arrangement: rent vs. mortgage vs. owning. There are only three categories here, and the frequencies for each category look to be above 5% of the total number of observations, so we wont be making any adjustments here.

Finally, we have loan reason. Similar to the employment length and grade variable, we will need to combine categories using the recode() function, as some of these contain a very low number of observations. There also seems to be similar in nature categories such as “house” and “home improvement”, so those will be a natural combination. The combined groups will be as follows:

* “**Home Expense**” = home\_improvement, house, moving, renewable\_energy
* “**Other**” = car, major\_purchase, medical, other, small\_business, vacation, wedding

## 3.2 Missing Data & Imputation

There are two variables that contain missing values - length (of employment) & bcOpen. Because “length” is a categorical variable, we won’t be able to impute its value, so these 1,823 observations will be dropped.

loans = loans[which(loans$length != "n/a"),]

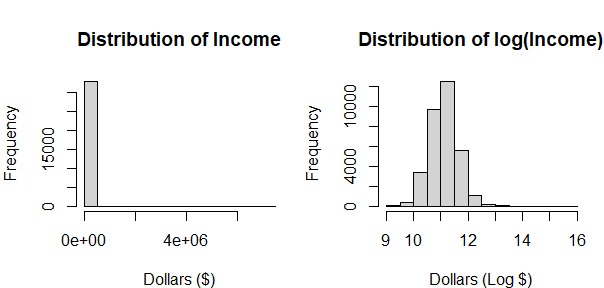
BcOpen, however, is a quantitative variable, therefore we can try an imputation process to determine “best fit” values for these missing data. We create a custom function impute\_bcOpen() that takes in the original dataframe, performs the imputation process using the mice() package, and replaces the NA values with the imputed values.

impute\_bcOpen <- function(df, seed){  
 #create a vector of the indices that are "NA"  
 index\_NA = which(is.na(df$bcOpen))  
   
 #complete the imputation process  
 imputation = mice(loans, seed = seed)  
  
 #iterate through each missing value and calculate the average of the imputed set, replace NA with the imputed value.  
 for(i in 1:length(index\_NA)){  
 sum = 0  
 sum = sum + imputation$imp$bcOpen$`1`[i]  
 sum = sum + imputation$imp$bcOpen$`2`[i]  
 sum = sum + imputation$imp$bcOpen$`3`[i]  
 sum = sum + imputation$imp$bcOpen$`4`[i]  
 sum = sum + imputation$imp$bcOpen$`5`[i]  
 df$bcOpen[index\_NA[i]] = sum/5  
 }  
 return(df)  
}

## Warning: Number of logged events: 25

# 4.0 Data Transformation

In preparation for model fitting, we need to take a look at the distribution of our quantitative variables to understand their shape. Should a distribution appear skewed in either direction, we can attempt a transformation to make it more “normal” in shape.

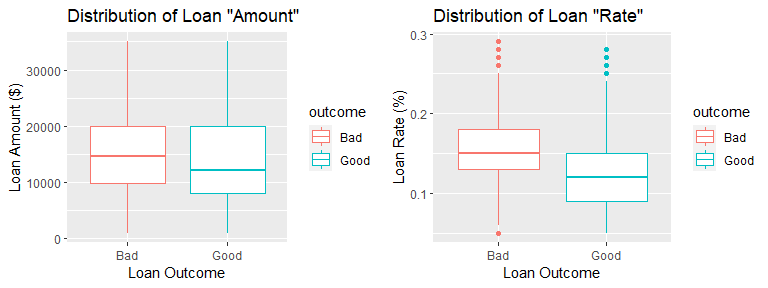


A prime example of skewed data is the “income” variable. In the first histogram we see a severely right-skewed distribution, so we attempt a log() transformation. The results of the log(income) show in the graph to the right - these are now much more normally distributed, so we will keep this transformation on the “income” variable.

A look at the remaining dollars based variables suggest a log() transformation applied to each one with the exception of (loan) amount.

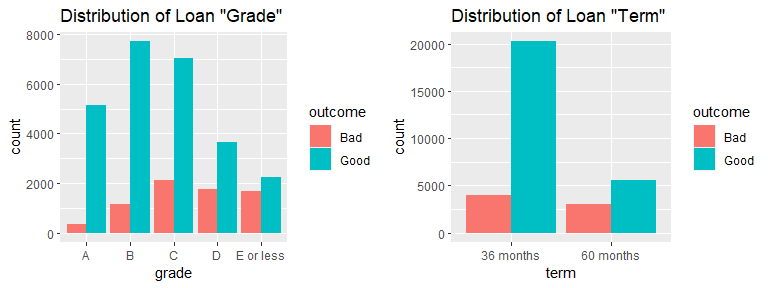
## 4.1 Data Exploration

Our final step before model fitting is to explore some of the data within the variables to gain some intuition of distribution as it relates to the our outcome variable of “Good vs. Bad”.



In the first boxplot we can see loan amounts (in $). Loans that have a “bad” outcome seem evenly distributed in their amounts, whereas “good” loans appear slightly skewed to the right. We perform a Wilcoxon Sum Rank test to check for significance between these distributions, and at the 95% level of confidence, there is enough evidence to conclude that the bad loan’s median amount is at least $1,000 greater than the good loan’s median amount (p=2.2e-16).

The second boxplot display’s loan percentage rate distribution. Both data sets appear to be generally normally distributed with a few outliers, however, similar to the above, the “bad” loans appear to have a higher median. We perform a Wilcoxon Sum Rank test to check for significance between these distributions, and at the 95% level of confidence, there is enough evidence to conclude that the bad loan’s median rate is at least 2.99% greater than the good loan’s median rate (p=2.2e-16).

 Now we can look at the first bar graph that shows the distribution of bad vs. good outcomes based on letter grade of the loan. We see some disproportion, with lesser graded loans having a higher number of bad outcomes. To test if the distribution is truly different among categories, we will perform a chi squared goodness of fit test. At a 95% significance level, there is enough evidence to support that these distributions of bad loans are different by loan grade (p=2.2e-16).

Finally, we review the proportions of bad vs. good outcomes for the 36 and 60 month loan terms. Visually, there appears to be a much lower proportion of bad loan outcomes in the 36 month term vs. the 60 month term, so we perform a proportion test. At a 95% level of confidence, there is enough evidence to support the proportion of bad loans in the 36 month term group is at least 18.07% less than the proportion of bad loans in the 60 month term group (p=2.2e-16).

# 5.0 Building the Logistic Regression Model

In this part, we will begin the steps of building our logistic regression model that will be used to predict the outcome of a loan - either successful repayment or default. We’ll look at building a holdout sample for testing vs. training, determining model selection, and assessing model performance.

## 5.1 Train/Test Data Split

Before we build our model, we will split our single data frame into two distinct data frames - one that will be used to fit (i.e. train) the model, and the other that will be used to test the model’s accuracy.

## 5.2 Model Building and Variable Selection

Next we check for collinearity among variables, and remove the variable that has the highest “VIF” score above 10 using the vif() function from the “car” package. After we remove the variable we rebuild the model and check VIF scores again, removing the next highest VIF score above 10. This process is repeated until there are no more variables above 10.

fullmodel = glm(outcome~., data = train, family = "binomial")  
vif(fullmodel)

Now we build the model using a stepwise technique with the forward direction. This technique starts with a null (i.e. blank) model and is bounded in potential variables by the full model we discovered in the previous step. With each iteration, the algorithm assesses how adding one variable to the null model from the full model will affect the AIC score of the null model. Each variable is considered and tested within the iteration, and the variable that decreases AIC the most is the one that is selected and added to the model. This process continues until the algorithm detects that adding any of the remaining variables would increase the AIC score, which indicates the process has reached its “best” version of the model it can find.

nullmodel = glm(outcome~1, data = train, family = "binomial")  
  
step(nullmodel, scope = list(lower = nullmodel, upper = fullmodel), direction = "forward")

The model that the process has identified as the “best” model is listed: outcome ~ grade + term + accOpen24 + home + income + payment + bcOpen + delinq2yr + avgBal + totalIlLim + rate + totalRevLim + totalRevBal + inq6mth.

We look at the model summary and check variable significance. TotalIlLim and inq6mth does not appear to be significant, therefore we will remove it from our model.

The final model we land on is listed in the summary output below. Here we see each variables coefficient, error, and significance on the overall model. The AIC score is 24,569.

##   
## Call:  
## glm(formula = outcome ~ grade + term + accOpen24 + home + income +   
## payment + bcOpen + delinq2yr + avgBal + rate + totalRevLim +   
## totalRevBal, family = "binomial", data = train)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.7707 0.3187 0.5125 0.7230 1.8462   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -1.184e+00 4.918e-01 -2.408 0.016050 \*   
## gradeB -3.649e-01 8.046e-02 -4.536 5.74e-06 \*\*\*  
## gradeC -7.502e-01 9.756e-02 -7.690 1.48e-14 \*\*\*  
## gradeD -9.336e-01 1.278e-01 -7.303 2.82e-13 \*\*\*  
## gradeE or less -9.957e-01 1.669e-01 -5.967 2.42e-09 \*\*\*  
## term60 months -6.817e-01 3.937e-02 -17.316 < 2e-16 \*\*\*  
## accOpen24 -6.637e-02 5.379e-03 -12.339 < 2e-16 \*\*\*  
## homeOWN -9.321e-02 5.869e-02 -1.588 0.112279   
## homeRENT -2.152e-01 4.325e-02 -4.976 6.50e-07 \*\*\*  
## income 3.155e-01 4.279e-02 7.373 1.67e-13 \*\*\*  
## payment -4.949e-04 8.139e-05 -6.081 1.20e-09 \*\*\*  
## bcOpen 3.628e-02 9.435e-03 3.846 0.000120 \*\*\*  
## delinq2yr -6.716e-02 1.682e-02 -3.994 6.50e-05 \*\*\*  
## avgBal 1.159e-01 2.226e-02 5.208 1.91e-07 \*\*\*  
## rate -3.635e+00 1.093e+00 -3.325 0.000885 \*\*\*  
## totalRevLim 8.775e-02 2.596e-02 3.381 0.000723 \*\*\*  
## totalRevBal -1.078e-01 2.473e-02 -4.361 1.30e-05 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 27238 on 26264 degrees of freedom  
## Residual deviance: 24535 on 26248 degrees of freedom  
## AIC: 24569  
##   
## Number of Fisher Scoring iterations: 5

## 5.3 Model Evaluation

Now that we’ve decided on a final model, we’ll assess its accuracy on correctly predicting loan outcome on the training dataset.

The table below shows the actual loan outcomes in the row labels, whereas the column labels represent the model’s prediction on outcome. The square Where the row label and column label match is considered a “correct” prediction for our model, and is counted in our numerator for the accuracy calculation. Mismatching labels show where the model “incorrectly” predicted the outcome. For example, there were 5101 loans that the model predicted as having a “good” outcome, however those loans were actually marked as having a “bad” outcome.

## predGood  
## Bad Good Sum  
## Bad 506 5101 5607  
## Good 387 20271 20658  
## Sum 893 25372 26265

## [1] "Proportion correctly predicted = 0.79"

To calculate accuracy, we take the sum of the correctly predicted observations and divide it by the total number of observations. In this case (506 + 20271)/26265 = 0.79, or 79% accurate.

We complete this same process with the holdout testing sample, and determine the model performs almost as accurate on previously “unseen” data - 0.78, or 78% accurate.

## predGood  
## Bad Good Sum  
## Bad 137 1317 1454  
## Good 95 5018 5113  
## Sum 232 6335 6567

## [1] "Proportion correctly predicted = 0.78"

Digging one step deeper on our accuracy measures, we look at both the accuracy of the “bad” outcome loans and the “good” outcome loans separately. For “good” outcome loans, the models accuracy is a whopping 98%! However, the model only predicts “bad” outcome loans with an accuracy of about 9% - not necessarily admirable. We used a default threshold of 0.50 on the predicted probabilities and when to kick them into the good or bad bucket, however this unbalanced result does seem to indicate another look at the threshold is in order.

# 6.0 - Balancing The Threshold For Better Results

In the last section we noticed a pretty unbalanced accuracy between the good and bad outcomes. In this section we will explore the accuracy trade off between good and bad outcomes, and see if we can identify an optimal threshold value using a trade-off curve.

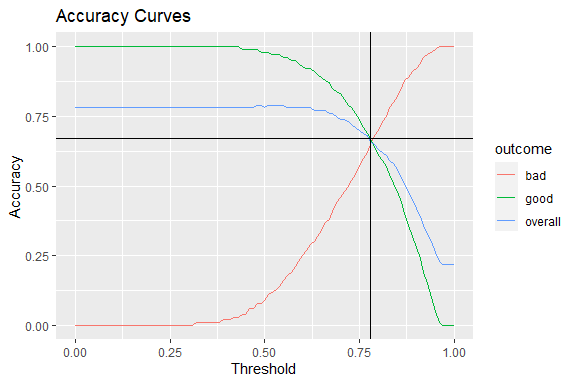
To get started, we create a custom function that repurposes the code used to build the contingency tables in the previous section. The function will take the vector of predicted probability output from the model, along with a threshold value. It will then calculate and return the current threshold used, the accuracy score (for overall), and the type of accuracy it calculated. A function for the good and bad outcome is also created.

calc\_overall\_accuracy <- function(probabilties, threshold){  
 predGood = cut(probabilties, breaks=c(-Inf, threshold, Inf),labels=c("Bad", "Good"))  
   
 cTab = table(test$outcome, predGood)   
 addmargins(cTab)  
 overall\_acc = round(sum(diag(cTab))/sum(cTab),2)  
   
 return(list(threshold,overall\_acc,"overall"))  
}

An empty dataframe containing three columns is created, and a for-loop is used to populate the data frame for each value of a possible threshold setting from 0.01 to 1.00. Below are the first six rows:

## threshold accuracy outcome  
## 1 0.00 0.78 overall  
## 2 0.00 1.00 good  
## 3 0.00 0.00 bad  
## 4 0.01 0.78 overall  
## 5 0.01 1.00 good  
## 6 0.01 0.00 bad

Now we can plot the resulting dataframe onto a single line plot. With threshold along the x-axis, each variable’s accuracy value is plotted along the y-axis and given a unique color. Where the lines intercept, this is our optimal selection of the threshold value for the accuracy measure.



The black lines pinpoint this optimal value for threshold which occurs at 0.78, with the corresponding accuracies as follows:

* **Overall** Accuracy: 0.66
* **Bad** Outcome Accuracy: 0.65
* **Good** Outcome Accuracy: 0.66

# 7.0 - Optmizing for Profit

In this section we will explore the trade off between prediction threshold selection and loan profit. Similar to the previous section, we will calculate “profit” (defined as the total amount paid on the loan, minus the original loan amount) as a function of all possible threshold values between 0-1. Once profit is calculated for each threshold value, we will plot the curve and decide which threshold value maximizes profit returned.

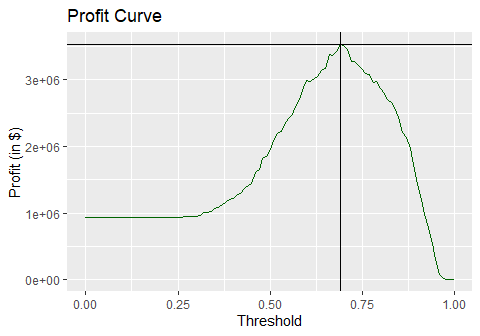
## 7.1 - Calculating Maximum Profit

To get us started, we will create a custom function that takes in both the predicted probabilities from the “test” holdout sample as well as a threshold value, and will return the calculated profit and the threshold value used. The function only focuses on those loans that the model *predicted* as having a “good” outcome per the given threshold, and predicted “bad” outcome loans are removed under the assumption those would be denied if the model was in production. Don’t forget - there are some “bad” loans that the model *predicted* as “good”, therefore the sum of all of these *predicted* “good” loans will include positive and negative profits.

calc\_profit <- function(probabilties, threshold){  
 predGood = cut(probabilties, breaks=c(-Inf, threshold, Inf),labels=c("Bad", "Good"))  
 test\_no\_bad = test[which(predGood == "Good"),c("amount","totalPaid")]  
 test\_no\_bad$profit = round(test\_no\_bad$totalPaid - test\_no\_bad$amount,2)  
 return(list(threshold,sum(test\_no\_bad$profit)))  
}

Now we calculate the profit for each potential threshold value. The first six rows of the profit table are displayed below, along with the plotted profit curve:

## threshold profit  
## 1 0.00 927573.3  
## 2 0.01 927573.3  
## 3 0.02 927573.3  
## 4 0.03 927573.3  
## 5 0.04 927573.3  
## 6 0.05 927573.3



The threshold that optimizes profit of the *predicted* “good” outcome loans is 0.69, with a total of about $3.5M.

## 7.2 - Model vs. Current State - Comparision of Profits

Now that we have determined the potential maximum profit when the model is in use and is “filtering” out the predicted bad loans, we can compute the ratio of profit between what the model produces vs. current state. In the code below, we again calculate the maximum profit produced by the model, and divide this by the same profit using the entire test data set (i.e. “current state”):

## [1] 3.807092

This turns out to be about 3.8 times greater. When we multiply by 100, we get about a 380% increase in overall profit when the model is in production (optimized for profit) compared to the baseline - wow!

Now we compare both the model’s profit number and baseline’s profit number against a “perfect” model’s (i.e. that could predict “bad” outcome loans with 100% accuracy) profit number - this would be considered our total potential theoretical profits.

profit\_perfect = sum(test$totalPaid[test$outcome == "Good"] - test$amount[test$outcome == "Good"])  
round(profit\_with\_model/profit\_perfect,2)

## [1] 0.3

round(profit\_without\_model/profit\_perfect,2)

## [1] 0.08

round(profit\_with\_model/profit\_perfect-profit\_without\_model/profit\_perfect,2)

## [1] 0.22

In the code above, we calculate profit under the guise of the “perfect” model (filter out all the bad outcome loans in the test dataset) - this is about $11.7M. Next, we divide the previously calculated profit from both the model and the current state by this ~$11.7M figure, and we see that the model accounts for ~30% of the total theoretical profits whereas the current state only accounts for ~8% of this theoretical profit. We can conclude that the use of the model increases profits by a whopping 22%!