Project_1

Eli (Ilya) Bolotin

04/05/2019

Section 1 - Executive Summary

Given that lending is a major source of long-term revenue for most banks, their business truly depends on making wise lending decisions. By correctly predicting which loans are “good” versus “bad”, the bank can avoid accumulating costly liabilities and maximize profit. The purpose of this report is to (1) provide a summary of a model created to increase bank profitability and (2) discuss the suitability of this model for bank operations.

The original dataset used to create the predictive model consists of 50,000 loans and 30 customer and loan-related fields. After data cleaning and preparation, the usable dataset consists of approximately 32,500 loans. The model was trained on roughly 26,000 of these loans and tested on the remaining ~6500.

The results of the predictive model are encouraging: the model predicts loan status with **80%** accuracy compared to actual (observed) classifications. By employing the model at this level of accuracy, the bank stands to improve profits by as much as *58%* compared to existing lending standards. However - the model could be further adjusted to more stringent standards of loan qualification. At the adjusted classification threshold, the bank would deny more potentially bad loans and improve profits by as much as *123% (or $3.7M)* compared to existing lending practices.

Implementing this model would be a wise decision by the bank. However, it would be a wiser decision to invest more resources into improving current model accuracy. The reason is: at 100% (theoretical) model accuracy, the bank would stand to improve profits by as much as *630%*.

The recommendation of this report is that bank management continue to invest resources in developing this model to at least 90% accuracy. Margins would improve well beyond the 123% (maximum) profit increases afforded under the current model.

Subsequent to the recommendation above, below are a few specific areas of improvement that would facilitate in developing a better model:

1. The dataset used to train this model contains nearly 80% of loans that are classified as “Good”. This means the model is much better at classifying “Good” loans than “Bad” loans. However, since maximum profit is determined by reducing “Bad” loans, it would be beneficial to have more data on “Bad” loans.
2. It would be beneficial to have more data in general.
3. The current model was trained on data that was focused on customer and loan characteristics/features. This data did not provide “qualitative” financial measures

such as if the loan meets the bank's desired risk profile… Or how much profit is historically earned on similar loans. Or, for example, if loan size meets the bank's preference for larger loans versus smaller loans. By finding and using such data in model design, it would be possible to improve accuracy significantly.

Section 2 - Introduction

The goal of this project is to create a logistic model that predicts whether loan applicants are likely to default on their loans. The dataset used to train this model includes 50,000 loans and 30 variables [1]. Let's review the stages of this project and what each stage entails. The stages are:

1. Data preparation and cleaning
2. Data exploration and transformation
3. Logistic modeling
4. Optimizing for accuracy and profit

In the **first** stage (preparing and cleaning), we define the response variable, drop unnecessary variables, create useful features, check for feature redundancy (correlation), and identify & resolve missing values. In the **second** stage, we identify, analyze and correct for feature skewness, evaluate the final features via plotting, and select final predictors for modeling. In the **third** stage, we split the data into training & test sets, create the logistic regression model, and predict loan status for loans in the test dataset. In the **final** stage of optimization, we optimize the model for both accuracy and profit, and compare & analyze the differences.

This project is preceded by an executive summary and followed by a results summary.

Section 3 - Data Preparation and Cleaning

Step 1: Prepare the response variable

The response variable will be based on the existing "status" column in dataset, and will have two possible values: "Good" and "Bad". Definitionally, these values mean [1]:

- "Good" loans are all those that are fully paid.
- "Bad" loans are loans that have a status of "charged off" or "default".
- Loans that are "late", "current" (being paid), or in grace period are removed from the data.

```
# Create 'not in' operator
'%ni%' <- Negate('%in%')

# Remove loans that are late, current, or in grace period
df <- df[df$status %ni% c("Current","Late (16-30 days)","Late (31-120 days)",
"In Grace Period"),]

# Create response variable
df <- df %>% mutate(response_var = case_when(
  status == "Fully Paid" ~ "Good",
  status == "Charged Off" | status == "Default" ~ "Bad")
  )
```

Step 2: Eliminate variables that are not useful as predictors and create new features

The original dataset has 30 variables. Not all of them are useful for modeling purposes in their current form. These variables include:

- **loanID** - This number is an arbitrary identifier. It does not have inherent qualities that influence the quality of loan making.
- **employment** - This is the job title of applicant. Job titles can be useful because they are indicators of economic/social standing and therefore correlated to wealth. However, the employment values are too specific to be useful. If they were grouped into broader categories such as "entry-level" or "management", they might have been more useful.
- **totalPaid** - Total repaid to bank. Cannot be known before loan is issued.
- **status** - Response variable. We are going to use this in creating the response variable.
- **payment** - Monthly loan payment amount is an absolute number. Without a denominator, it's not relative to anything. To make this number useful, we create a ratio of monthly payment to monthly income.
- **amount** and **income** (2 separate cols) - Delete because we have a monthly payment to monthly income ratio. Both are derivatives of these columns.
- **revolRatio** - The proportion of revolving credit in use. Very similar to bcRatio, which is the ratio of total credit balance to credit card limits. To avoid redundancy, we remove this.
- **bcOpen** and **bcRatio** - bcOpen is total unused credit. Which is the inverse of bcRatio. We're going to use bcOpen in a custom feature, so we can delete these two features after doing so.

This dataset also contains variables prefixed by "total". These figures represent either *credit limits* (total limits on credit cards or installment accounts) or *credit balances*. Such columns are most meaningful when used in ratios. It's the ratio of balance to credit limit that is possibly influential as a predictor. We will create a few ratios to make use of this data, and then drop the following columns:

- totalIlLim
- totalBcLim
- totalLim
- TotalRevLim
- totalBal
- totalRevBal

Before dropping the columns above, we need to create custom predictors for use in our model:

- **mpayToInc_ratio** - ratio of montly (loan) payment to monthly income. Assumption is that a high ratio is bad.
- **opencredit_ratio** - ratio of total unused credit on credit cards to total credit limits of credit cards. High ratio is good.
- **openAccRatio** - ratio of number of recently opened accounts to number of open credit lines. Low is good.

Create custom features before dropping (above) columns

```
# monthly payment to monthly income
df['mpayToInc_ratio'] <- df['payment']/(df['income']/12)

# total unused credit / total credit limit
df['opencredit_ratio'] <- df['bcOpen']/df['totalBcLim']

# recently open accounts to total accounts
df['openAcc_ratio'] <- (df['accOpen24']/df['openAcc'])
```

Next, let's drop undesired columns from our data

```
# Will drop totalPrepaid later
df <- select(df,-c(employment, totalPaid, loanID, status, payment, amount, in
come, totalIlLim, openAcc, totalBcLim, totalLim, totalRevLim, totalRevBal, to
talBal, bcOpen, bcRatio, revolRatio))
```

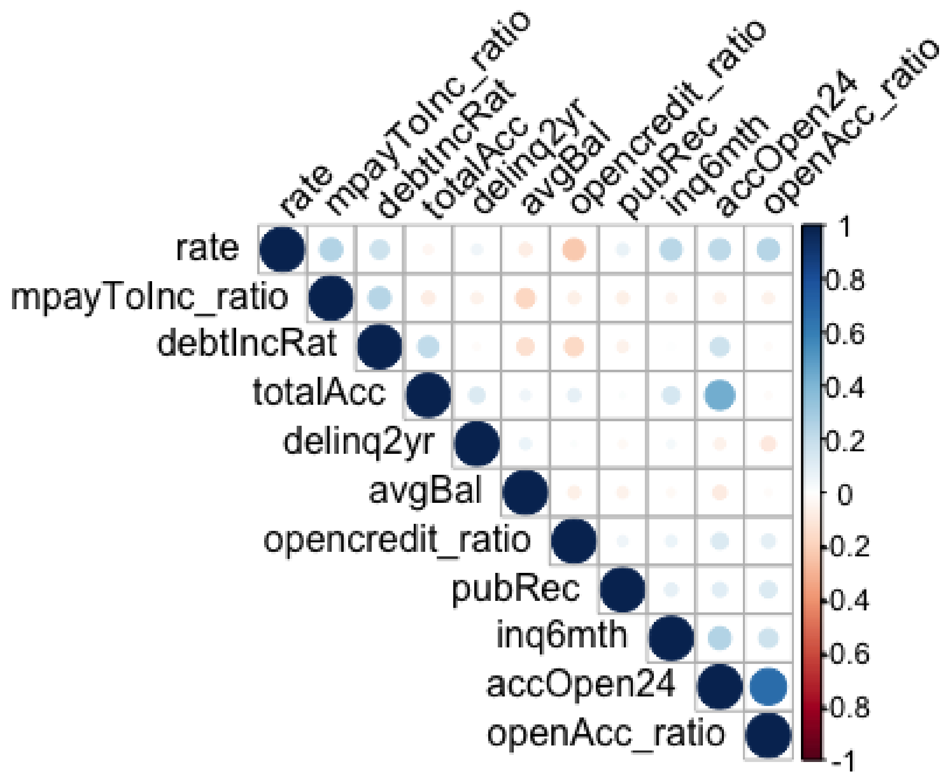
With the above columns dropped, some of the remaining variables may be redundant. We can check if this is the case using correlation analysis. In preparation, let's select only numerical features from the dataset.

Step 3: Identify numeric and categorical features

```
# Get all numeric columns
is.numeric <- sapply(df, is.numeric)
numeric_cols <- which(is.numeric)
num_numeric_cols <- length(numeric_cols)
names_numeric_cols <- names(numeric_cols)

# Get all non-numeric (categorical) columns
not.numeric <- !(is.numeric)
cat_cols <- which(not.numeric)
num_cat_cols <- length(cat_cols)
names_cat_cols <- names(cat_cols)
```

Step 4: Perform correlation analysis to check for variable redundancy.

```
# plot correlation matrix
corrplot(corr_table, type = "upper", order = "hclust",
         tl.col = "black", tl.srt = 45)
```



This chart indicates that overall, the numeric features in this dataset do not demonstrate high correlation. This means that our variables do not "overlap" too much, and have the propensity to be independant predictors of the response variable.

Step 5: Evaluating and fixing missing values

This data has missing values. Missing values can be naturally occurring (the absence of a value), or they can be hardcoded by the author/collector of the data. As we will see, both types are found in this dataset.

First, let's check for "standard" NAs.

```
# Get columns with "standard" NAs
na_columns <- sapply(df, function(y) sum(length(which(is.na(y)))))

# Calculate sum of all "standard" NAs
total_standard_na <- sum(na_columns)

# Calculate number of rows with "standard" NAs
total_standard_na_rows <- nrow(df[(!complete.cases(df)),])
```

Next check for "non-standard" NAs. These are hardcoded values such as "n/a" or string blanks.

```
# Get non-standard NAs
total_nonstandard_na <- get_other_na(df)
```

```
# Calculate total NA in dataset
total_na <- total_nonstandard_na + total_standard_na

# Calculate *count* of rows with all variants of NA
total_na_rows <- length(which(apply(df, 1, function(x) (any(x %in% c("", "n/a
") | is.na(x))))))

# Calculate *percent* of rows that have an NA
percent_missing <- total_na_rows / nrow(df)
```

Result: In summary, there are a total of 2225 NAs. 1830 of these are non-standard. 395 are standard NAs."

Let's get a better sense of missing values by looking at detailed tables:

The analysis above is quite telling:

- There are 2 kinds of missing values in this data: standard and non-standard NAs, as defined above. There are a total of 2225 *total* NAs in this data, spread out in 2177 *rows*. These rows amount to 6.3% of the total rows.
- Overall, missing data is fairly sparse. Few rows have missing values, and those that do have few NAs.
- The primary sources of missingness is the opencredit_ratio columns (which is actually based on the presently deleted variable 'bcOpen').

Conclusion: we can delete records with NA (instead of imputing them) with relatively low impact on predictive capability.

Next: delete all rows/records that contain any NAs

```
# Delete standard NAs
df <- na.omit(df)
rows_to_delete <- which(apply(df, 1, function(x) (any(x %in% c("", "n/a")))))

# Delete non-standard NAs
df <- df[-c(rows_to_delete),]
```

Result We have successfully deleted 2177 rows with NAs from our data. Now we are ready to transform and explore our data.

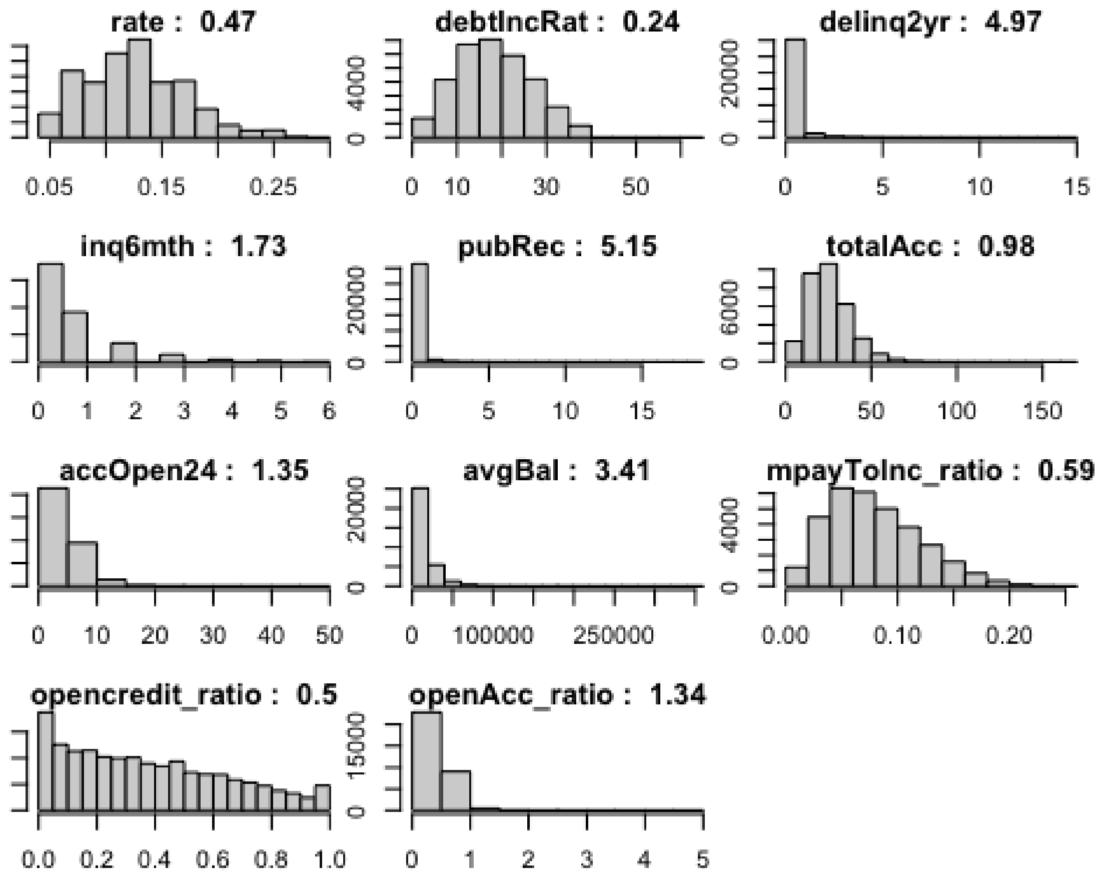
Section 4 - Data transformation and exploration

The next stage in this analysis is to understand whether any variables are strongly skewed. The reason to do this is because a skewed distribution may result in model coefficients that are sizable enough to reduce the model's accuracy.

So, in the following steps, we check for skewness and then transform the skewed variables.

Step 1: Evaluate and transform *numeric* feature skewness

```
# Calculate skewness before transformation
skewness_before <- check_skewness(df, names_numeric_cols)
```



Judging from the histograms above, we can confirm that numeric features in this dataset are right-skewed. Furthermore, many features are non-trivially skewed (not in a small amount). These features could benefit from transformation/scaling. Let's compute median skewness *before* transformation.

Calculate median skewness before transformation

```
# Calculate median skewness of numeric features
median_skewness_before <- median(skewness_before$skewness)

# get details about skewness with regard to a threshold of 1
get_skewed_features_threshold <- get_skewness_details(skewness_before)
count_skewed_above_threshold <- get_skewed_features_threshold[[1]]
features_skewed_above_threshold <- as.vector(get_skewed_features_threshold[[2
]])
```

The median skewness of numeric features **before** transformation is 1.34.

Transform skewed features
```
# transform skewed features by taking square root (of abs value and then mult
iply by sign).
df[features_skewed_above_threshold] <- sapply(df[features_skewed_above_thresh
old], function(x) (sign(x) * sqrt(abs(x))))
```

After transformation, let's check if median skewness has decreased.

Check median skewness after transformation
```
median_skewness_after <- median(skewness_after$skewness)
median_skewness_after
```

Result: skewness of numeric columns has decreased from 1.34 to 0.23.

Step 2: Explore and analyze final features

The dataset has now been prepared, cleaned, and transformed. Next, we need to analyze the features with respect to the response variable. More specifically, we need to review the distributions of features for 'Good' and 'Bad' loans. Let's start by looking at how many records constitute both 'Good' and 'Bad' loan groups.

Compare proportions of the response variable (good and bad loans)
```
# First check the size of our good/bad loan groups
good_count <- nrow(df[df$response_var=='Good',])
bad_count <- nrow(df[df$response_var=='Bad',])
total_count = nrow(df)

good_percent <- (good_count / total_count) * 100
bad_percent <- (bad_count / total_count) * 100
```

Nearly 80% of the records in this loan data are classified as "Good" loans. This means that our model will be able to better recognize good loans than bad loans.

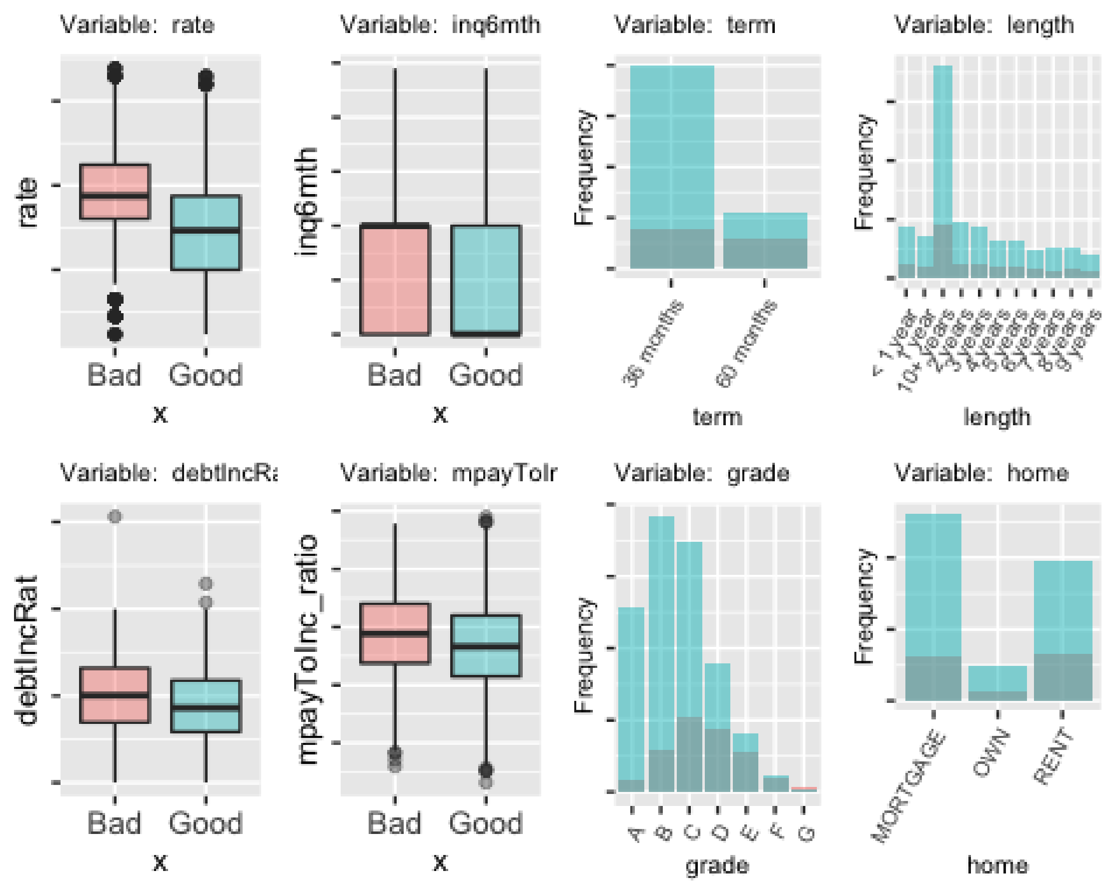
Next, we want to find numeric variables that have significantly different distributions for "Good" and "Bad" loan groups. We will perform Welch's T-test to determine significance at the 5% significance level.

Step 3: Perform hypothesis test of numeric features
```
# Create vector to track significant numeric features
significant_numeric_features <- c()

# Loop through numeric features and perform hypothesis test for each feature
for(feature in names_numeric_cols) {
  t.test <- t.test(df[df$response_var == 'Good',feature], df[df$response_var
== 'Bad',feature])
  if(t.test$p.value < 0.05) {
    significant_numeric_features <- c(significant_numeric_features, feature)
  }
}
```

Next, let's evaluate the feature set using box plots for numeric features and bar plots for categorical features.

Step 4: Plot & analyze numeric and categorical features

```
# Create container of box plots for numerical features
box_plots <- list()
i = 0

# Loop through numeric features and generate ggplot box plot
for (col in significant_numeric_features) {
    p <- ggplot(df, aes_string(x=df$response_var, y=col, fill=df$response_var
)) +
    geom_boxplot(alpha=0.4) + ylab(col) + ggtitle(paste("Variable: ", col)) +
    guides(fill=FALSE) +
    theme(plot.title = element_text(size=8), text = element_text(size=10),
    axis.text.x = element_text(size=10, angle=0), axis.text.y = element_blank
())
    box_plots <- c(box_plots, list(p))
    i = i + 1
}

# Create container of bar plots for categorical features
bar_plots <- list()

# Loop through categorical features and generate ggplot barplots
for (col in names_cat_cols) {
  if(col != 'response_var') {
    p <- ggplot(df, aes_string(x=col, fill=df$response_var)) +
    geom_bar(alpha=.5, position="identity") + xlab(col) +
    ylab("Frequency") + ggtitle(paste("Variable: ", col)) +
    guides(fill=FALSE) +
    theme(plot.title = element_text(size=8), text = element_text(size=8),
    axis.text.x = element_text(size=7, angle=60, hjust=1), axis.text.y = elem
ent_blank())
    bar_plots <- c(bar_plots, list(p))
    i = i + 1
  }
}

# define significant categorical features
significant_categorical_features <- names_cat_cols[names_cat_cols != "respons
e_var"]

# Plot boxplots and barplots in grid
multiplot(plotlist=c(box_plots[1], box_plots[2], box_plots[4], box_plots[8],
bar_plots[1], bar_plots[2], bar_plots[3], bar_plots[4]), cols=4)
```



Results of box plots: The boxplots show significant differences in response variable groups. For example, the interest rate of good loans is significantly lower than for bad loans. Also, the debt-to-income ratio of borrowers of "bad" loans is significantly higher than borrowers of good loans. Not surprisingly, borrowers of bad loans have had credit inquiries in the last 6 months.

Results of bar plots: there are a few interesting barplots here. The first is the barplot of the "home" feature (which reveals if the borrower owns/rents/has mortgage). The highest amount of bad loans belong to borrowers that rent. And the highest amount of good loans belong to borrowers that have a mortgage. People that own homes have the least of any type of loan.

Another interesting categorical predictor is "term". The 36-month term 'good' loans far outnumber 60-month term 'good' loans. However, both terms are very close when it comes to 'bad' loans. This could indicate that term is not really a good predictor for bad loans.

In the final step before modeling, let's get a count of total features.

Step 4: Calculate total feature count

```
total_significant_feature_count <- length(significant_numeric_features) + len
gth(significant_categorical_features)
total_column_count <- length(names(df))
```

The total number of significant features in our dataset is 17. This is 1 less than the total predictor/feature count of 18. This is due to the hypothesis tests that we performed on numeric features.

Section 5 - The Logistic Model

With the data prepared, cleaned, transformed, analyzed, and features selected - the next step is to create a logistic model.

Step 1: Split the dataset into training and test sets

```
# Create updated dataframe which contains significant features (numeric + cat
egorical) and response variable
df_updated <- df[,c(significant_numeric_features, significant_categorical_fea
tures, "response_var")]

# Get rownames (indices) of df_updated
df_updated_indices <- as.numeric(row.names(df_updated))

# Get 'totalPaid' column from original dataset
totalPaid <- original[df_updated_indices, "totalPaid"]

# Get 'amount' from original dataframe
amount <- original[df_updated_indices,"amount"]

# Add 'totalPaid' column to df_updated
df_updated <- cbind(df_updated, totalPaid, amount)

# Set seed
set.seed(42)

# Get total count of rows in dataaset
total_rows <- nrow(df_updated)

# Get quantile at 80% distribution
num_training_rows <- round(total_rows * 0.80)

# Generate sample of random indices for data frame
training_indices <- sample(as.vector(df_updated_indices), num_training_rows,
replace = FALSE)

# Get remaining 20% of random indices to represent test indices
test_indices <- setdiff(df_updated_indices, training_indices)

# Create training data
df_training <- df_updated[df_updated_indices %in% training_indices,]
df_training <- df_training[,names(df_training) %ni% c('totalPaid','amount')]

# Create test data
df_test <- df_updated[df_updated_indices %in% test_indices,]
```

Step 2: Create logistic regression model using training data

With the data split into training and test sets, the next step is to train a logistic model on the training dataset.

```
# Create logistic regression model
glm_model <- glm(response_var~., data = df_training, family="binomial")
```

Step 3: Use this model to predict the response variable (Good or Bad loans) for the test data.

```
# Make predictions
predictions <- predict(glm_model, df_test, type = "response")
df_predictions <- as.data.frame(predictions)
```

Now that predictions have been computed, it is possible to evaluate the model's accuracy by comparing it to the observed response variable.

Step 4: Create a classification table of predictions vs observed loan status *for test data*.

The classification threshold in this model is 0.5. Meaning - loan status is classified as "Good" if Y (output/response variable) is greater than 0.5. Loans are classified as "Bad" if Y is less than or equal to 0.5.

```
# Print table and accuracy
ctable

##           observed
## predicted  bad good  Sum
##      bad   170  127  297
##      good 1241 4958 6199
##      Sum  1411 5085 6496

## TNR (%) - Recall:  12.05
## TPR (%) - Recall:  97.5

## Overall accuracy (%):  78.94
```

At the 0.5 classification threshold, model accuracy is about 80%. This means our model correctly predicted 80% of loan statuses for the loans in the test data.

The decision of whether this model is effective or ineffective depends on the business requirements for loan making. Perhaps, 80% accuracy in loan status classification is quite good. Perhaps, it is not good enough. Furthermore, this accuracy is only at the level of test data. We would need to run this model against additional test sets, perhaps even deploy it in a development sandbox to understand it's general applicability to predicting loan status.

Section 6 - Optimizing the Threshold for Accuracy

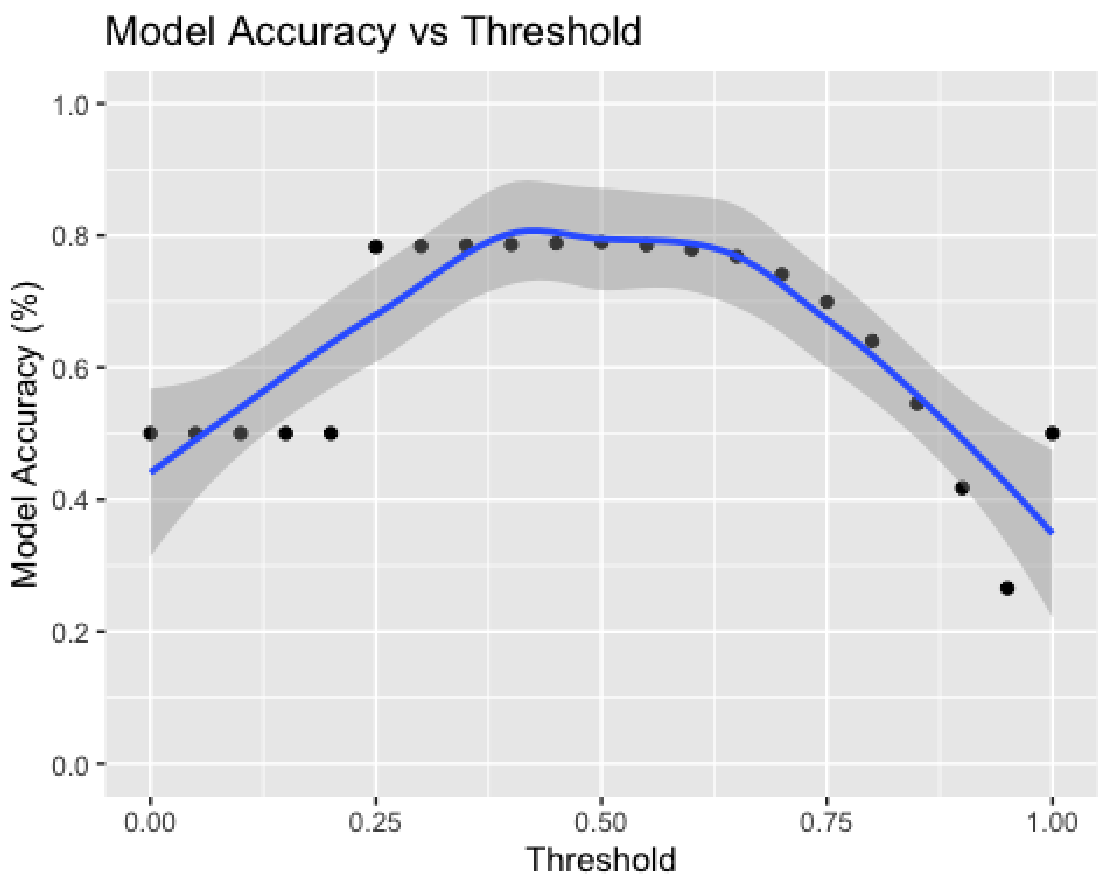
A big question about our model is whether the classification threshold of 0.5 is optimal. Perhaps, a smaller or greater threshold would improve accuracy. To find out, we can plot a scatterplot of accuracy vs threshold.

Step 1: Generate list of threshold and Y values and combine them into single dataframe

```r
# Generate list of threshold values
t_list = seq(from = 0, to = 1, by = 0.05)

# Create classification table and accuracy plot
y_list = create_classification(t_list)
accuracy_plot <- as.data.frame(cbind(t_list, y_list))
```

Step 2: Plot Accuracy vs Threshold

```r
# Generate plot
ggplot(accuracy_plot, aes(x=t_list, y=y_list)) +
  geom_point()+
  geom_smooth(method="loess") +
  labs(title = "Model Accuracy vs Threshold", x = "Threshold", y = "Model Acc
uracy (%)") +
  scale_y_continuous(breaks=seq(0,1,.20), limits=c(0,1))

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



The plot above indicates that the 0.5 threshold results in the maximum accuracy producable with this model.

Section 7 - Optimizing the Threshold for Profit

While we know that this model's *accuracy* is highest at the 0.5 classification threshold, we do not yet know if this is the optimal threshold under which the model also maximizes the bank's *profit*.

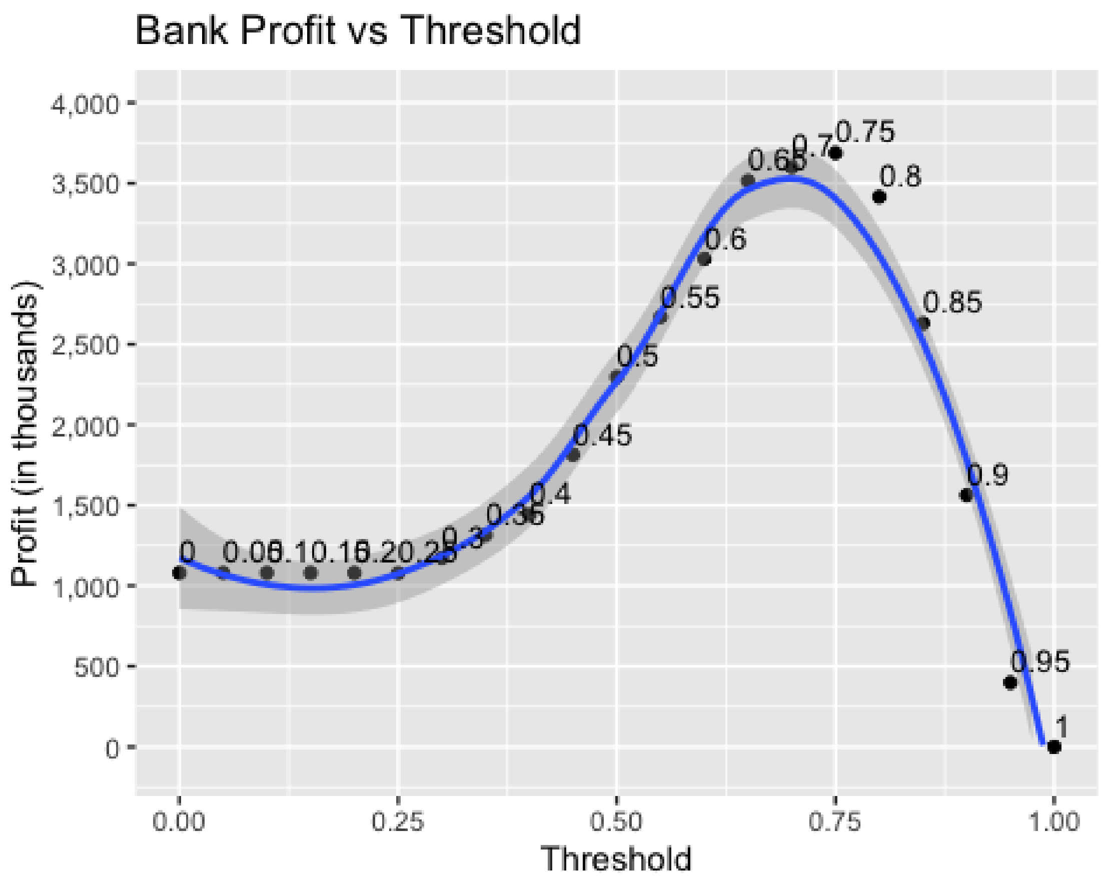
Step 1: Compute profit at the default 0.5 threshold

```
profit <- getProfit(0.5)
profit

## [1] 2296726
```

In denying all bad loans at the 0.5 threshold, our model suggests that bank profit is about $2.6M.

Step 2: Find the optimal threshold for maximizing profit.

```
profit_list = getProfit(t_list)
profit_threshold_list <- as.data.frame(cbind(t_list, profit_list))

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



The plot above indicates that the model maximizes bank profitability at the 0.7 threshold ($3,659,443).

Step 3: Compare profitability to ideal model

Now that we know the maximum profitability under this model, we need to compare this to the profitability generated by a "perfect" model that correctly predicts all loan statuses.

```
data.frame(this_model, ideal_model)

##   this_model ideal_model
## 1       2.33       10.07
```

The table above indicates that our model generates an increase in profitability of 123% (first column). Compare this to a "perfect" model that correctly predicts 100% of bad loans, which the bank would then deny. Under this perfect model, the bank stands to improve it's profitability by 630% (second column).

Step 4: Compare thresholds of optimal profitability vs optimal accuracy

We determined that the optimal *accuracy* threshold (at which the model correctly classifies the most good/bad loans) is 0.5. We also determined that the optimal *profitability* threshold is 0.7. Let's produce a classification table to show accuracy for the optimal profitability threshold.

Print classification table and accuracy for 0.7 threshold

```
##          observed
## predicted  bad good  Sum
##      bad   668  940 1608
##      good  743 4145 4888
##      Sum  1411 5085 6496

## TNR (%) - Recall:  47.34
## TPR (%) - Recall:  81.51

## [1] 0.7409175
```

Print classification table and accuracy for 0.5 threshold

```
##          observed
## predicted  bad  good   Sum
##      bad   170   127   297
##      good 1241  4958  6199
##      Sum  1411  5085  6496
##      Sum  2822 10170 12992

## TNR (%) - Recall:  12.05
## TPR (%) - Recall:  97.5

## [1] 0.7894089
```

Overall conclusion: This model is ~5% less accurate at the 0.7 threshold than at the 0.5 threshold. At the same time, at the 0.7 threshold, this model generates $1,063,726 more profit than at the 0.5 threshold.

Section 8 - Results Summary

The analysis above leads to the following conclusions:

1. This model classifies loans with 80% accuracy (maximum).
2. The most *accurate* threshold is 0.5.
3. The most *profitable* threshold is 0.7.

Why do the thresholds for maximum accuracy and profitability differ? The reason is that our model resembles a parabolic function with respect to threshold. Think of the threshold as a filter. The higher the threshold, the more stringent the filter. As you increase the threshold, the quantity of loans *classified* as "good" decrease and quantity of loans classified as "bad" increase. Note that not all loans **classified** as "bad" are actually bad in the observed data (same thing for "good" loans).

The point is that a higher threshold will result in more loans classified as "bad" and thus denied by the bank. This has the effect of maximizing margins from a profit standpoint (total paid back to the bank minus loan amount). Compare the classification tables in the last part of section 7. The 0.7 threshold table has a higher amount of bad loans (1614) excluded than at the 0.5 threshold (291).

What does this mean?

Even though the 0.7 threshold is less accurate than the 0.5 threshold, it results in more stringent classification of good loans and thus eliminates more bad loans, which increases profit for the bank.

Sources

[1] Project Assignment: https://datascienceuwl.github.io/Project2018/