2019 Mortgage Loan Classification Report

STATS 101C Final Project

Ahmed Awadalla

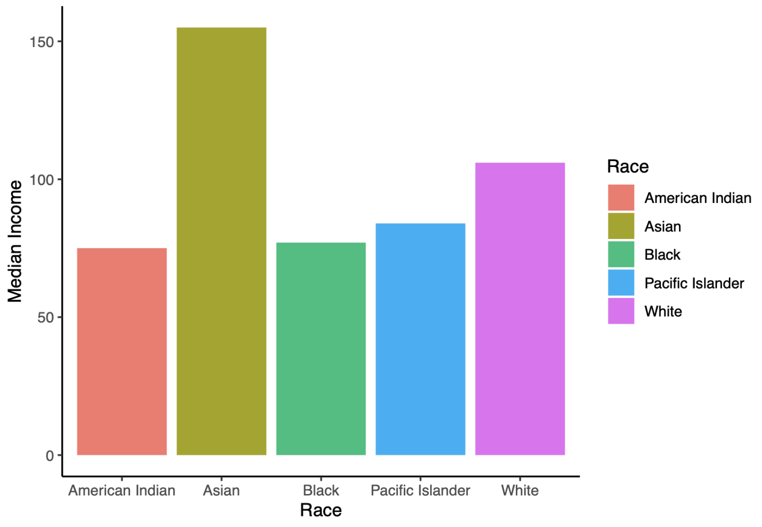
Introduction

By analyzing Wells Fargo's HMDA submissions in 2019 and the Consumer Financial Protection Bureau's Loan/Application Register, I seek to predict which loans are approved (originated) and which loans are denied. Disregarding several incontrovertible variables in the data—such as an applicant's debt to income ratio, origination charges, and reason for denial—I reference the Federal Reserve's "How Much Does Racial Bias Affect Mortgage Lending?" and Harvard Business Review's "AI Can Make Bank Loans More Fair" to help determine underlying factors that may influence what action will be taken on a loan application. As such, I firmly believe that an individual's race, income, age, credit scoring model, and property value are strongly associated with the response variable.

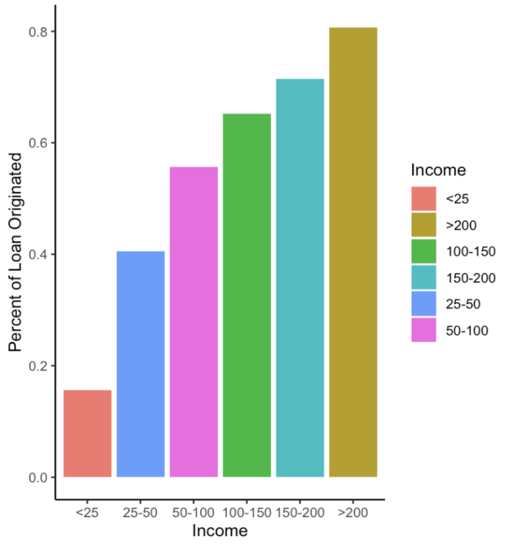
Exploratory Data Analysis

According to the Federal Reserve, standard factors "can explain" most racial disparities in denial rates; thus, since a loan application is heavily dependent on an individual's financial stability, I first explore the median income per race, and how income influences whether a loan is approved.

Median Income per Race

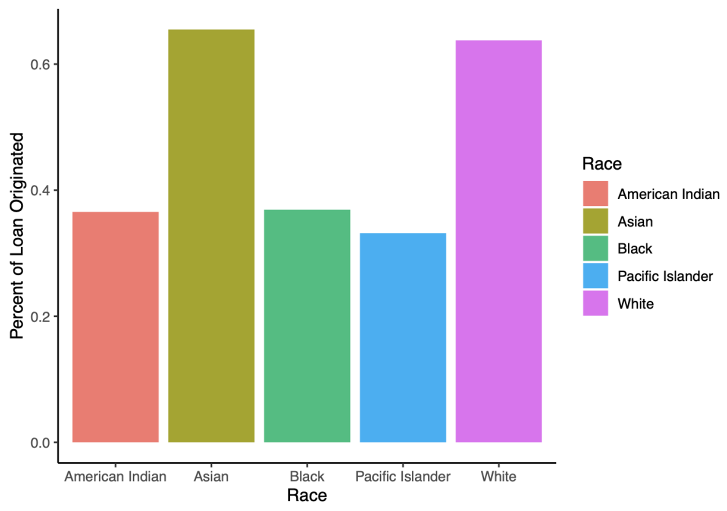


Loans Originated per Income



By comparing each race's median income, Asian applicants possess a significantly higher median income compared, followed by White, Pacific Islander, Black, and American Indian. Juxtaposing this visualization with the percent of loans accepted, I can see that the percent of loans originated increases as income increases; following this trend, I am confident that Asians possess the highest percent of loans accepted. I conclude that income is a significant factor. Further exploring the Federal Reserve's report into racial bias, I seek to investigate the relationship between the ethnicity/race of an individual's application with the action taken.

Percent of Loans Originated by Race

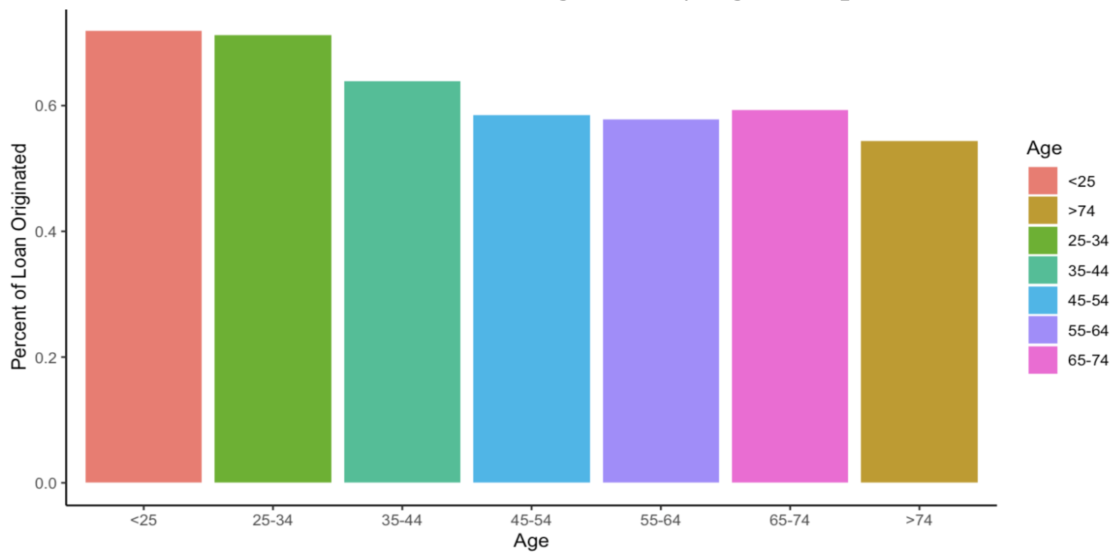


I can see that there is a clear difference in the percent of loans originated based on race. Most notably, Asians and Whites have a significantly higher percent of loans accepted compared to other races.

However, this trend is not explicitly indicative of racial bias; as I mentioned, Asians and Whites typically possessed a higher median income and therefore, a higher percent of loans accepted.

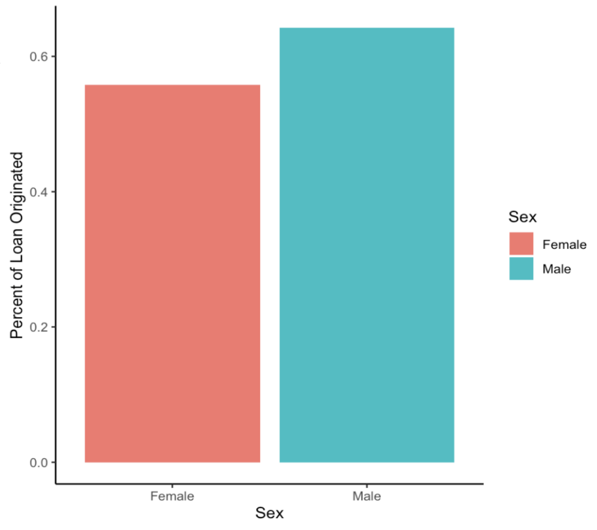
In addition to race/ethnicity, according to Harvard Business Review's "AI Can Make Bank Loans More Fair", several significant factors in loan denial are an individual's age and gender; thus, I shall compare the percent of loans originated with an applicant's age and sex.

Percent of Loans Originated by Age Group



There is a clear decrease in percent of loans originated as the age range of the applicant increases. Between <25 and 25-34, the percentage is roughly similar, with roughly 70% of applicants approved. However, there is a significant drop in the 35-44 range, which continually decreases as the age increases. While I witness a slight increase between 65-74, it is relatively insignificant, and the percent of accepted loans sharply decreases again to around 55% for individuals older than 74. Thus, banks tend to be biased against older applicants.

Percent of Loans Originated by Sex

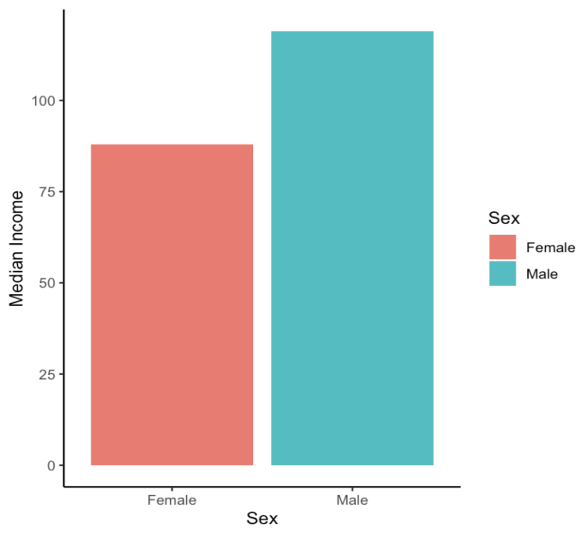


Similarly, I see a noticeable difference in the overall percent of loans accepted between male and female primary applicants.

Male applicants are accepted roughly 60% of the time (around 65%), while female individuals are accepted slightly less than 60% of the time. While not necessarily evidence of gender bias, I am confident that women generally have a lower percentage of accepted loans compared to men.

Thus, exploring income's association with sex:

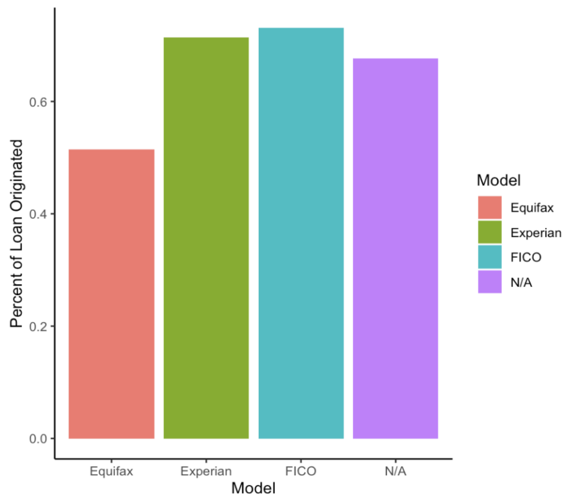
Relationship between Sex and Income



I can see that there male primary applicants have a significantly higher median income than female primary applicants, hence explaining the difference in the overall percent of loans accepted in our earlier plot.

This reiterates the significance of an individual's income when determining whether or not their loans are approved (originated) or denied. However, to add predictive power, I retain an applicant's race, sex, and ethnicity in our final model.

Percent of Loans Originated per Scoring Model

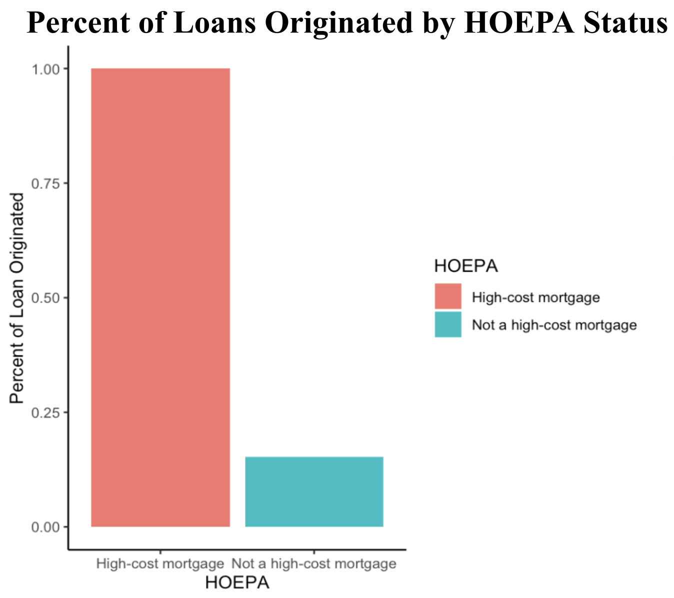


Analyzing the percent of loans approved for each credit scoring mode, applicants with Equifax Beacon 5.0 are the lowest by far, with a percentage of approval roughly equal to 50%.

Experian Fair Isaav and FICO Risk Score Classic 04 appear to be roughly similar, hovering around 75-80%. People who did not report a model have around a 70% percentage of approval. Thus, I conclude that an individual's credit scoring model is a significant predictor.

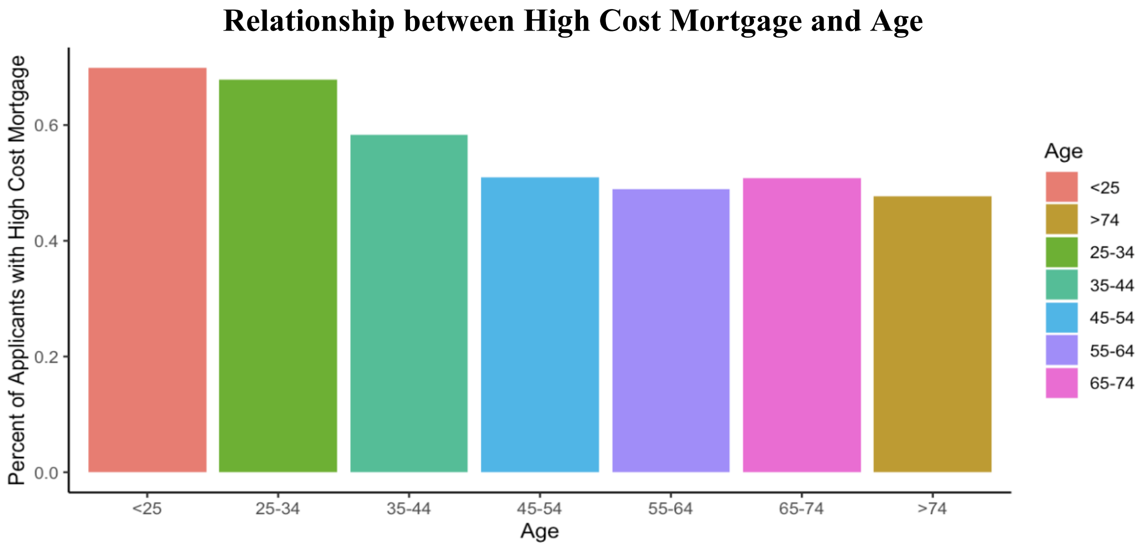
For this next section, in order to analyze their potential relationships and fit them into a model, I convert several numeric variables into factors. Most notably, we convert age, sex, ethnicity, and HOEPA status into factors. Several of these conversions are are shown below:

```
train2 %>%
mutate(age_of_applicant_or_borrower = as.factor(age_of_applicant_or_borrower),
hoepa_status = as.factor(hoepa_status), sex_of_applicant_or_borrower =
as.factor(sex_of_applicant_or_borrower))
```



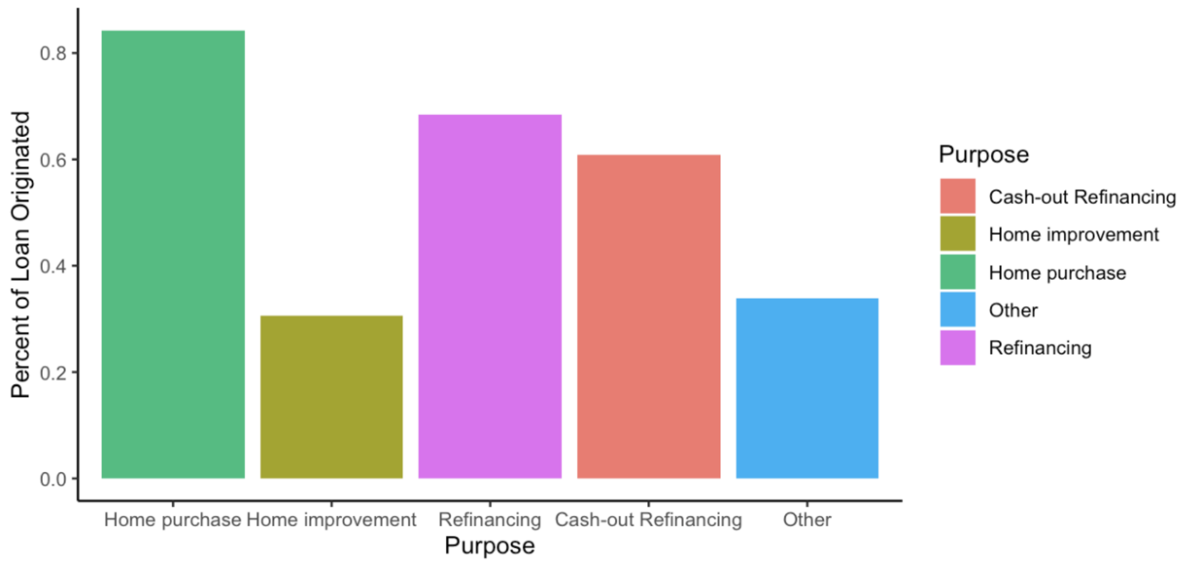
I notice that there is a significant difference in the percentage of loans originated between applicants with a high-cost mortgage and not a high-cost mortgage. Hence, I believe that this is a significant factor to include in my model.

Because of how strong HOEPA is associated with my response variable, I want to investigate which demographic factors typically possess a high-cost mortgage to construct an interaction term.

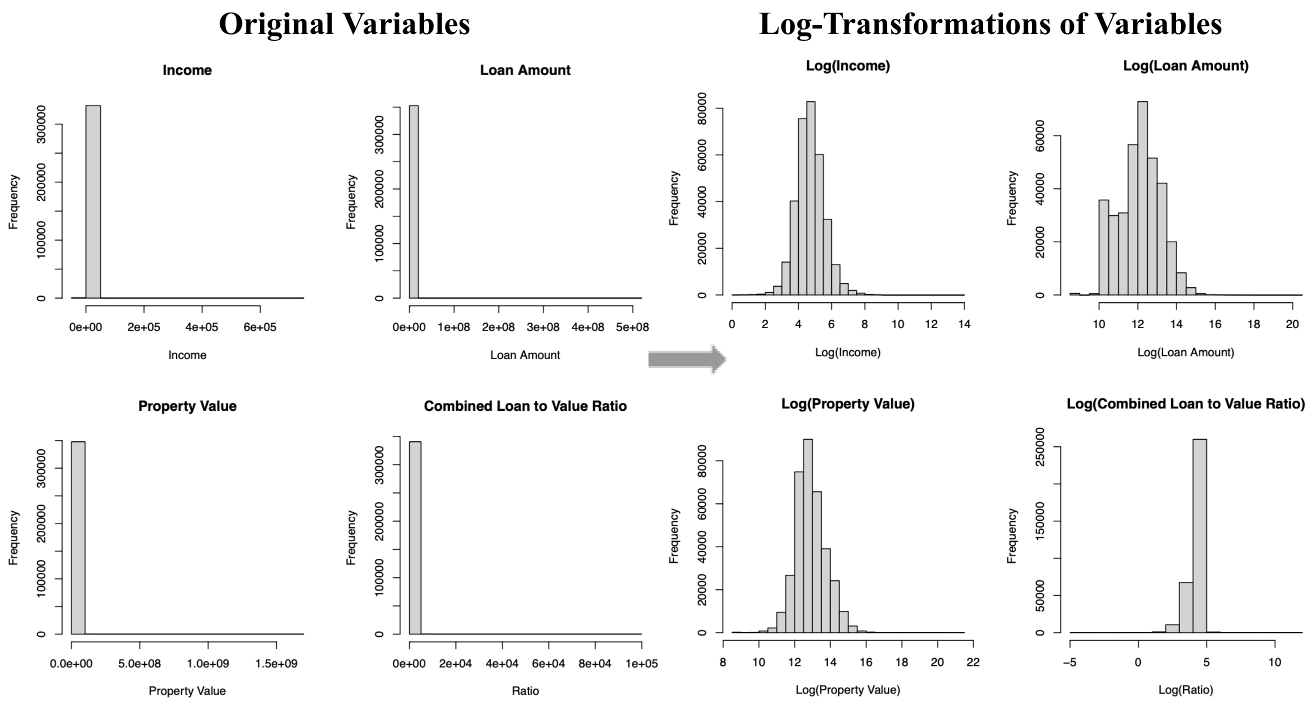


Unsurprisingly, the patterns in this bar chart closely resemble the “Percent of Loans Originated per Age Group”, suggesting that high cost mortgages strongly interact with Age—an interaction I will include in my model. For example, I see that individuals <25 and between 25-34 tend to have a high cost mortgage (~65-70%), reflecting their percentage of loans approved (~70%).

Percent of Loans Accepted for Each Purpose



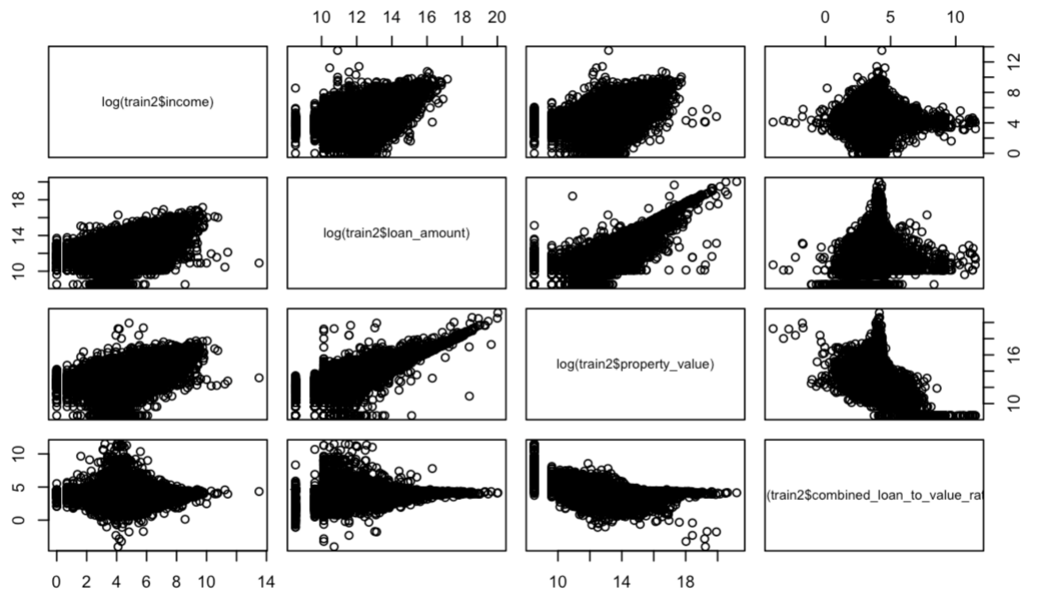
From the plot, more than 80% of home purchase and ~75% of refinancing loans are accepted, suggesting that there is a high probability an application is accepted when the applicant is utilizing the loan for those two specific reasons. On the other hand, home improvement loans are only accepted ~30% of the time. Therefore, loan purpose is a significant predictor when determining whether or not a loan is accepted.



Since I believe that each individual's income, loan amount, property value, and combined loan-to-value ratio are associated with what action will be taken on a loan application, I transformed these predictors to fit into the linear model, following the normal distribution requirement.

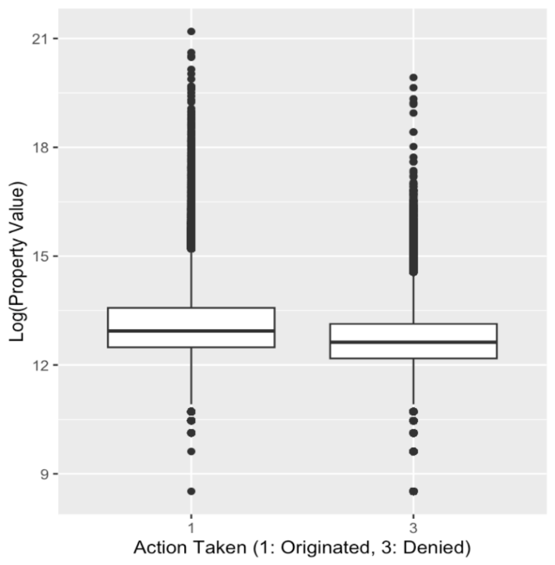
Each applicant possesses a unique economic status—with some significantly better off—so many plots are skewed to the right, requiring a log transformation to create a linear relationship with the target variable.

Interaction of Log-Transformed Variables



In the interaction plot, there is a clear positive correlation between the loan amount and an applicant's property value. As the property value increases, so does the loan amount. Likewise, as the individual's income increases, I see an overall increase in their property value and the loan amount. Therefore, I conclude that the loan amount is dependent on an applicant's income and property value, and there is also a relationship between the two aforementioned predictors.

Action Taken by Property Value



Analyzing the difference between when a loan is accepted and when it is denied, I notice that the median log property value of applicants is slightly higher for originated loans.

However, there is a multitude of outliers suggesting that plenty of individuals with a high property value are still denied, skewing the data.

Thus, while the property value of successful applicants is slightly higher than denied applicants, I believe that there are too many outliers for any significant predictive power.

Preprocessing / Recipes

When constructing the recipe for the candidate models, I decided to predict the outcome variable `action_taken` based on roughly 70% of the base predictors. Using `step_rm()`, I subsequently removed more than 20 repetitive “noisy” variables or columns filled with non-existent values. By implementing this formula, I optimized my model’s computational efficiency by significantly reducing the number of predictors, improved my model’s accuracy, and possessed further flexibility regarding what predictors to remove or create.

While modeling the data, I realized several predictors in the test and train datasets contained mismatched types. Thus, we converted `ethnicity_of_applicant_or_borrower_4` and `race_of_co_applicant_or_co_borrower_4` into numerics. Furthermore, as mentioned in the Exploratory Data Analysis section, I converted several numeric predictors—specifically the `age_of_applicant` variables, `state`, and `race`—into factors to properly implement these variables and further optimize computation.

Due to the prevalence of non-existent values in several of the numeric predictors, I utilized `step_impute_median()` to replace these NAs with the column median. Likewise, I used `step_impute_mode()` to replace non-existent values in the nominal predictors with the column mode. These two steps are essential, as they guaranteed I did not run into any issues with NAs. Finally, because the training dataset contained several factor levels with very few observations, I implemented the `step_other()` function to “collapse” these levels into a level called "other", placing these infrequent values into a different category and ensuring they do not negatively affect our model’s accuracy.

Following this step, I generated a dozen different interactions with `step_interact()`. Referencing our Exploratory Data Analysis, I selected each interaction based on their relation to the response variable, `action_taken`. Thus, I implemented an interaction between an applicant’s race with their specific ethnicity, sex, and income. Furthermore, I explored the interactions between loan amount, an individual’s income, loan purpose, occupancy, and what state they resided in. In total, I constructed nearly a dozen different interactions in order to account for the key sub-categories and characteristics of someone applying for a loan: their motivation or purpose for applying, economic status, age, sex, race/ethnicity, and location. This allowed us to predict what action will be taken on a loan application using the interactions for each specific applicant’s personal attributes in tandem with their reason to apply.

Finally, using `step_mutate()`, I constructed nearly 10 new predictors to enhance our model’s predictive power. Because age and length of loan affected whether or not a loan originated, I developed two predictors which indicated if a loan term was longer than 300 days, and whether or not an applicant was over 60 years—since percent of accepted loans sharply decreases as age increases. In order to generalize and categorize the numeric variables, specifically income and loan term, I also created bracket predictors by splitting these columns into factors. I grouped all the specific ethnicities and races together, so my final model analyzed the six main classifications: American Indian, Asian, Black, Pacific Islander, Latino, and White. Ultimately, I increased my final model's accuracy by including these parameters.

Candidate Models

I have constructed 8 candidate models to predict what action will be taken on a loan application: which loans are approved (originated) and which loans are denied. The models are:

1. **Logistic Regression**: Model 1
   A logistic regression model is utilized to predict the outcome of a categorical response variable; in this case, a binary classification (approve/deny). It makes the assumption that there is a logistic relationship between target and the features. By setting Mixture=1, I implemented a pure lasso model—which estimates the coefficients by maximizing the log-likelihood function and reducing low coefficients to zero.

2. **Random Forest:** Model 2, Model 4, Model 6
   A random forest model is essentially a group of decision trees. To determine the result, it will aggregate the decision for the target from all sets of decision trees. Furthermore, it randomly selects a set of features to split on, thus preventing overfitting. It also prevents overfitting by defining the proper number of trees as well.

3. **Decision Tree:** Model 3
   A decision tree splits the feature set into nodes and the target decision into leaves. In each node, the decision tree makes a decision based on the feature and the criteria defined when training the decision tree. Similar to other tree models, a decision tree is able to prevent overfitting through the use of tree depth and penalty functions.

4. **Single Layer Neural Network:** Model 5
   A single layer neural network is an artificial neural network that can host a multitude of hidden layers; these hidden layers draw parallels and assign weighted values in order to classify inputs to derive a specific output. This framework reiterates on itself in order to construct a more concise and accurate model

5. **Support Vector Classification:** Model 7
   A support vector classification fits a hyperplane to the data such that it is able to classify each data to the proper target class based on this separation. If the data is not linearly

separable for a hyperplane, it uses a kernel function to transform the feature set to a higher dimension to make it separable for a hyperplane.

6. **Boosted Tree Model:** Model 8
A boosted tree is built in sequential order such that it is able to learn from previous iterations for classification purposes. On top of this, boosted tree can be tuned to prevent overfitting through the use tree depth and penalty functions.

Model	Model Type	Engine	Recipe/Variables	Hyperparameters

Model 1	Logistic Regression	glmnet	Removed 23 variables
			Added 6 interactions
			Added 1 variable
				Penalty = 0.01
				Mixture = 1

Model 2	Random Forest	ranger	Removed 23 variables Added 6 interactions Added 1 variable	Mtry = 18
Model 3	Decision Tree	rpart	Removed 23 variables Added 6 interactions	cost_complexity= 30, tree_depth = 100 , min_n = 25
Model 4	Random Forest	ranger	Removed 23 variables Added 6 interactions	mtry = 75

Model 5	Neural Network Classification	kernas	Removed 15 variables	Penalty = 0
				Hidden = 10

Model 6	Random Forest	ranger	Removed 26 variables, 9 interactions, 10 mutates	mtry= 18
				trees = 650

Model 7	Support Vector Classification	kernlab	Removed 24 variables, 5 interactions, 5 mutates	cost = 0.1
				rbf_sigma = 1

Model 8	Boost Tree Classification	xgboost	Removed 26 variables, 9 interactions, 10 mutates	learn_rate = 0.1
				trees = 450

Table 1: All Attempted Candidate Models

Model Evaluation and Tuning

For hyperparameter tuning, I decided to try a different approach than the regression competition. First and foremost, I randomly searched for a range of hyperparameters which best suited the various models—most notably for random forest and decision tree classification models. Afterwards, from the initial large number of possible hyperparameters, I would determine which hyperparameters performed the best based on an accuracy metric.

Using the hyperparameter values from random search, I conducted an abridged grid search around a small neighborhood of values to find the best combination of values. While I was concerned with this method overfitting—especially to a certain set of hyperparameters—I ultimately avoided this issue using cross-validation.

By implementing 3-5 folds using the tune() function from the tune package, I continually tested each hyperparameter under a different circumstance and measured the performance of the candidate models.

Model Identifier	Accuracy
Model 1: LR	0.9755
Model 2: RF	0.9881
Model 3: DT	0.9633
Model 4: RF	0.9728
Model 5: SLNN	0.9651
Model 6: RF	0.9848
Model 7: SVC	0.9557
Model 8: BT	0.9887

Table 2: Candidate Model Evaluation

Further elaborating on my method, because Boost trees contain several hyperparameters—namely learning rate and max depth—for the tree itself, I decided to:

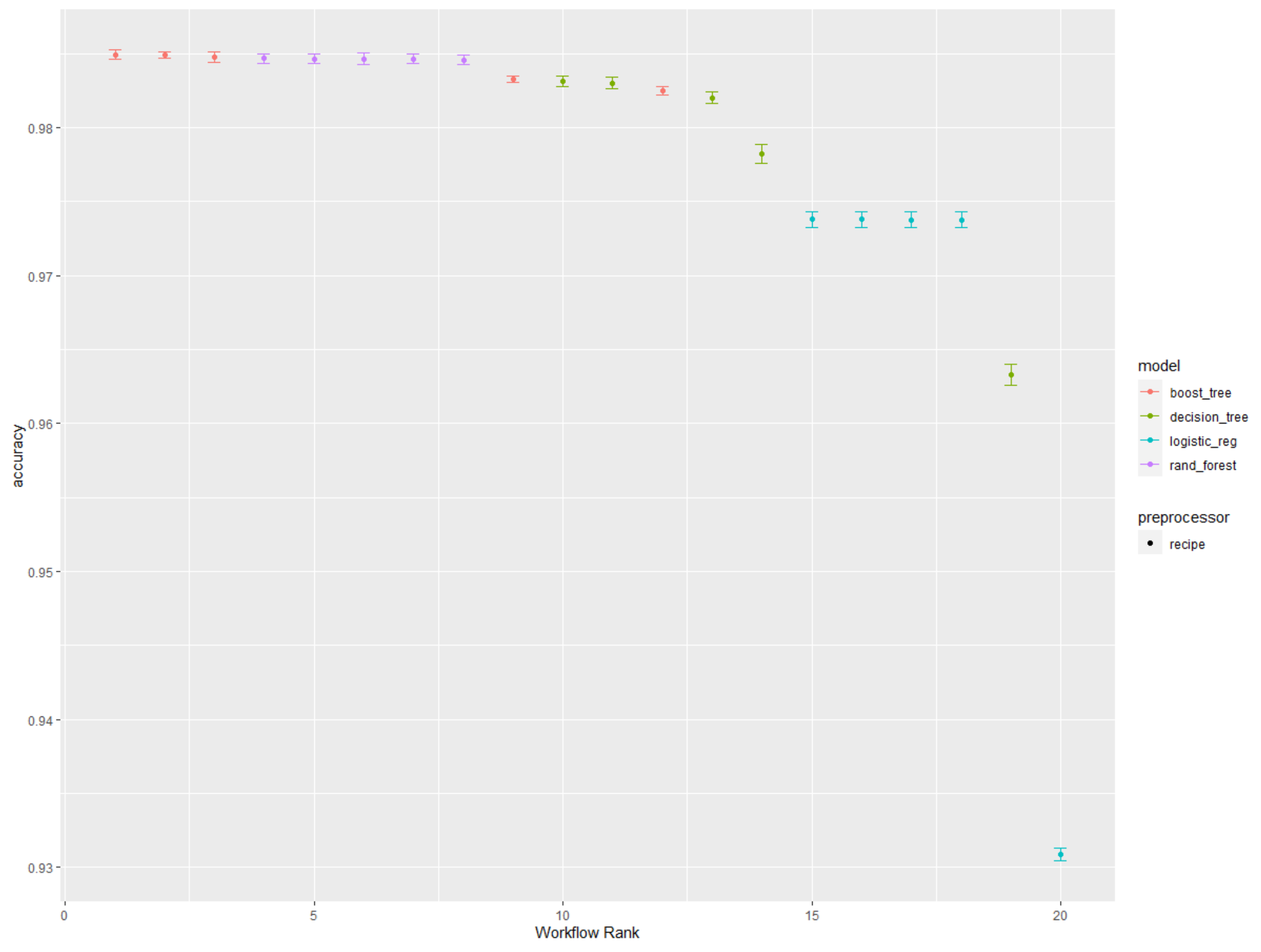
1. Randomly sample various learning rate values between 0.001 and 0.4 and max depth values between 1 and 20, optimizing the computation speed.

2. After randomly searching through these rates, my function would recommend a set of hyperparameters (e.g. learning rate = 0.05, max_depth = 9)

3. Finally, I conducted an abridged grid search around the neighborhood of these values. For example, I would test:
    a. Learning rate: [0.035, 0.04, 0.045, 0.05, 0.055, 0.06]
    b. Max_depth: [6, 7, 8, 9, 10, 11, 12]

From the example above, random search greatly reduced the time needed for tuning each model. If I had only used grid search, I would have to test a large combination of values between 0.001 and 0.4 for learning rate, and 1 and 20 for max depth. However, with random search, I only had to use grid search for a few values—learning rate- [0.035, 0.04, 0.045, 0.05, 0.055, 0.06] and max depth- [6, 7, 8, 9, 10, 11, 12]. This allowed me to identify the best performing set of hyperparameters in a shorter amount of time.

Autoplot Comparing the Performance of the Different Models



To preface, the sheer amount of features and data in this competition made it difficult to tune the hyperparameters in the time it involved. As mentioned above, I used hyperparameter training to tune each model. Overview for each model:

1. **Random Forest Model:** Naturally, random forest model was one of the slowest models to tune and train due to complexity and depth. Using hyperparameter training and determining the optimal number of features, I found it preferred a higher amount of trees to capture variance across the dataset.

   While this reduced my model's potential misclassification reliance from an incorrect tree, I risked overfitting the data. To mitigate this issue, I used cross validation to ensure that the random forest model was also repeatedly trained for "unseen" data. However, this large amount of trees also increased the time it took to train and the overall complexity.

2. **Decision Tree Model**: The performance of my decision tree model is rather similar to the aforementioned random forest model. Given its ability to handle non-linear data and outliers, my tree model generally performed better for the competition. However, while testing my model on Kaggle, I realized that Decision Trees had a tendency to overfit the train data—resulting in a decreased performance against unseen test data. For this competition, I tuned for cost_complexity and tree_depth.

3. **Support Vector Model Classification:** In terms of training speed, my support vector model took the longest, due to its need to find a hyperplane to be able to separate features into distinct classes in the target. However, this contributed to my model's misclassification—if the features were mapped extremely close to the hyperplane, the model typically chose the closest target class, leading to a misclassification. I tuned the cost parameter in order to determine a margin of error for the hyperplane.

4. **Boosted Tree Model:** Likewise, the boosted tree model performed rather similarly to the prior tree model during cross validation. As such, it handled outliers and non-linear data relatively better than many of the other models. However, like the decision tree model, it struggled predicting the private test data on Kaggle, despite the use of cross validation.

5. **Logistic Regression:** Logistic regression, although it trained rather quickly, did not perform as well in this competition. I suspect there does not exist a log relationship between the features and the target variable. Despite the use of the penalty, I believe that there exists multicollinearity and dependence between each feature in the dataset. Therefore, with these given issues, trees performed better for me in this competition.

6. **Neural Network Model:** Using Python as a base, I manufactured a single layer neural network model to generate a classification model. Utilizing the keras engine and a modified version of my original recipe, I applied the mlp function to the workflow with certain parameters. By setting epochs to 100 and the hidden units to 10, I generated a decent classification model with an accuracy of .9651. Although it may be beneficial to set each parameter to tune(), it not only required a significant amount of time due to the density of the data and scale, but the model also utilized an exceptional amount of computing power. One of the main oversights is the variables at play; my model was missing some key variables, but also a great deal of the variables were crowded with NAs which could account for a lower score.

Discussion of Final Model

In conclusion, my leading model was a random forest set to 650 trees and a mtry of 18; this produced an accuracy of .9881 which was fairly even with my other models. The random forest was able to utilize its splits and numerous trees by adding additional randomness that prioritizes the best variables. The nature of this process reduces the risk of overfitting as it minimizes the noise some variables could produce. This in turn allows me to comprehend which specific variable plays a more influential part within the data. All this culminates into a highly accurate model that can handle various types of information.

However, this does not mean that it is without its faults; due to the high complexity, I found this model difficult to train and extremely time consuming. Being able to extrapolate the data can prove quite challenging, as it does not display a simple correlation between variables. Furthermore, this process made refining a recipe difficult, since it required additional computing time. Another weakness is associated with the data; random forest models are disproportionately biased to the majority, so a high density of one variable type may skew the data, thus lowering the accuracy of the model. This could all culminate in overturning the hyperparameters to fit the training data, and leading to over-fitting.

For future reference, one aspect that I can further improve on is the recipe itself and the interaction I have in place. Due to the nature of a random forest, it is hard to associate certain aspects of the data, so for following implications, using the feature importance score more accurately would help better improve my recipe. My model would also improve if adjusted for the weights of certain variables; figures like “loan type” and “preapproval” had a definite majority within their category which could skew the model. This could help boost the accuracy when approaching a minority within the dataset. It also could be possible to implement a stack to bolster my results as it incorporates different models.

Overall, my random forest model was fairly accurate, only having a margin of error of roughly 2%. There are many things that could be implemented to improve the model to further increase its accuracy. Given additional data such as credit history, number of prior loans, amount of collateral/liquid asset, or even any outstanding debt would refine my model even further as those are key factors in loan approval. Nevertheless, given the current data and model, the random forest I created is exceptional.

Appendix

Final Annotated Script

```
library(tidyverse)
library(tidymodels)

set.seed(25)

train2 <- read_csv("train2.csv")
test2 <- read_csv("test2.csv")
train2$action_taken <- as.factor(train2$action_taken)
train_folds <- vfold_cv(train2, v = 10)

# Begin of data processing

# convert two columns below to numeric to match the test_set
train2$ethnicity_of_applicant_or_borrower_4 <-
as.numeric(train2$ethnicity_of_applicant_or_borrower_4)
train2$race_of_co_applicant_or_co_borrower_4 <-
as.numeric(train2$race_of_co_applicant_or_co_borrower_4)

# ensure that factors across different categorical variables are
consistent across train and test
all_levels_state <- union(unique(train2$state),
unique(test2$state))
all_levels_age_applicant <-
union(unique(train2$age_of_applicant_62),
unique(test2$age_of_applicant_62))
```

```
all_levels_age_co_applicant <-
union(unique(train2$age_of_co_applicant_62),
unique(test2$age_of_co_applicant_62))

train2$state <- factor(train2$state, levels = all_levels_state)
train2$age_of_applicant_62 <- factor(train2$age_of_applicant_62,
levels = all_levels_age_applicant)
train2$age_of_co_applicant_62 <-
factor(train2$age_of_co_applicant_62, levels =
all_levels_age_co_applicant)

# unselected columns due to high multicollinearity or high
amounts of NA
unselected_cols <- c("age_of_applicant_or_borrower",
"age_of_co_applicant_or_co_borrower", "total_units",
                     "legal_entity_identifier_lei",
"ethnicity_of_applicant_or_borrower_4",
                     "ethnicity_of_applicant_or_borrower_5",
"ethnicity_of_co_applicant_or_co_borrower_3",

"ethnicity_of_co_applicant_or_co_borrower_4",
"ethnicity_of_co_applicant_or_co_borrower_5",
                     "race_of_applicant_or_borrower_2",
"race_of_applicant_or_borrower_3",
                     "race_of_applicant_or_borrower_4",
"race_of_co_applicant_or_co_borrower_5",

"total_points_and_fees","prepayment_penalty_term",
"introductory_rate_period",
                     "multifamily_affordable_units",
"automated_underwriting_system_2",
"automated_underwriting_system_3",
                     "automated_underwriting_system_4",
"automated_underwriting_system_5",
"race_of_applicant_or_borrower_5",
                     "race_of_co_applicant_or_co_borrower_4")

# recipe for classification model, random forest
recipe_obj <-
```

```
  recipe(action_taken ~ ., data = train2) %>%
  step_rm(all_of(unselected_cols)) %>%
  # impute by median to handle any potential outliers
  step_impute_median(all_numeric(), -all_outcomes()) %>%
  step_other(c(state, age_of_applicant_62,
age_of_co_applicant_62), threshold = 0) %>%
  step_log(all_numeric_predictors(), signed= T) %>%
  # step impute by mode for categorical variables
  step_impute_mode(c(state, age_of_applicant_62,
age_of_co_applicant_62)) %>%
  # create interactions for ethnicity~race, income~race
  step_interact(~
ethnicity_of_applicant_or_borrower_1:race_of_applicant_or_borrow
er_1   ) %>%
  step_interact(~ income:race_of_applicant_or_borrower_1) %>%
  step_interact(~  sex_of_applicant_or_borrower:loan_amount) %>%
  step_interact(~ business_or_commercial_purpose:loan_purpose)
%>%
  # create interactions for loan_amount to different factors,
income, loan_to_value
  step_interact(~ loan_amount:income) %>%
  step_interact(~ loan_amount:combined_loan_to_value_ratio) %>%
  step_interact(~ state:loan_purpose) %>%
  step_interact(~
ethnicity_of_applicant_or_borrower_1:loan_type) %>%
  # key interaction between loan_type and loan_purpose
  step_interact(~ loan_type:loan_purpose) %>%
  step_interact(~
race_of_applicant_or_borrower_1:sex_of_applicant_or_borrower)
%>%
  step_interact(~ occupancy_type:loan_amount) %>%
  # create ratios for loan amount so model can interpret
proportions instead of hard-coded values
  step_mutate(loan_prop = loan_amount/income) %>%
  step_mutate(loan_ratio = loan_amount / income) %>%
  # following mutations classifies loan length, age, income to
reduce variance
  step_mutate(long_loan = ifelse(loan_term > 360, 1, 0))  %>%
  step_mutate(old = ifelse(age_of_applicant_62 == 1, 1, 0)) %>%
  step_mutate(old_cosigner = ifelse(age_of_co_applicant_62 == 1,
1, 0)) %>%
  step_mutate(income_classify = ifelse(income < 30000, "low",
ifelse(income < 80000, "medium", "high"))) %>%
  step_mutate(loan_classify = ifelse(loan_term <= 180, "short",
ifelse(loan_term <= 360, "medium", "long"))) %>%
  step_mutate(race_of_applicant_or_borrower_1 =
              replace(race_of_applicant_or_borrower_1,
                      race_of_applicant_or_borrower_1 %in%
21:27, 2)) %>%
  step_mutate(race_of_applicant_or_borrower_1 =
              replace(race_of_applicant_or_borrower_1,
                      race_of_applicant_or_borrower_1 %in%
41:44, 4))

# fit to random forest model with 650 trees
rf_spec <-
  rand_forest(mtry = 18, trees = 650) %>%
  set_engine("ranger",
           importance = 'impurity') %>%
  set_mode("classification")

workflow_obj <-
  workflow() %>%
  add_recipe(recipe_obj) %>%
  add_model(rf_spec)

model_fit <-
  workflow_obj %>%
  fit(data = train2)

predictions <-
  model_fit %>%
  predict(test2) %>%
  cbind(test2 %>% select(id))

write_csv(predictions, "rf_team15_election.csv")
```