STATS 101C Summer 2023

Kaggle Classification Competition
Team 12 Report

Introduction: Context and Background Information

In this Kaggle Classification competition report, we delve into the realm of mortgage loan predictions based on data from Wells Fargo National Bank in 2019. Our objective is to predict the actions that will be taken on loan applications. By examining the variable, we believe that loan_type, loan_purpose, loan_amount, income, rate_spread, and applicant_credit_scoring_model are pivotal factors influencing loan decisions. For instance, a higher credit score may enhance one's prospects of loan approval, and an applicant's income can serve as a crucial indicator of their capacity to meet loan repayment obligations. These variables, among others, hold the potential to unravel essential insights into the loan decision-making process. In the upcoming sections, we will discuss our approach for analyzing and selecting predictors, the recipe we created, the various candidate models we've explored, the tuning techniques we've applied, and ultimately, the model we have chosen for making predictions.

Citations

- https://www.forbes.com/advisor/personal-loans/personal-loan-requirements/#:~:text=Most%20personal%20loan%20lenders%20review,Minimum%20credit%20score%20of%20670.
- https://recipes.tidymodels.org/reference/step_impute_bag.html
- https://www.tidymodels.org/find/parsnip/
- https://stacks.tidymodels.org/articles/basics.html
- https://recipes.tidymodels.org/reference/step_YeoJohnson.html
- Stats 101C textbook: Intro to Statistical Learning with R
- Stats 101C: Introduction to Statistical Models and Data Mining Lecture PowerPoints

Exploratory Data Analysis

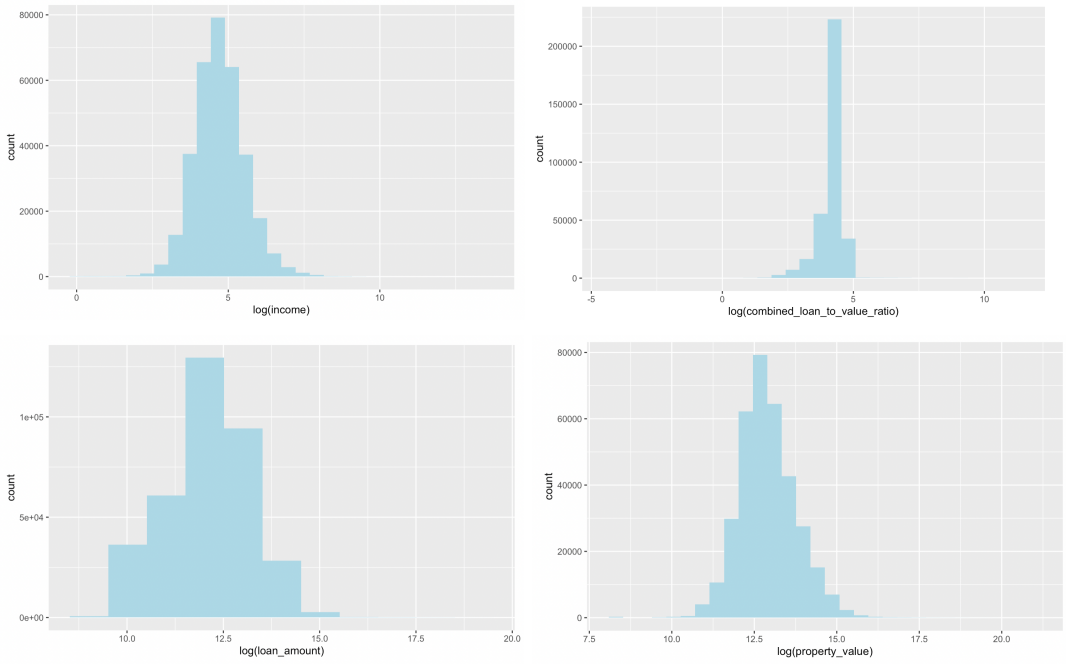
Correlation

- Correlation between selected numerical predictors after dropping NAs.
- There is no significant correlation between variables, the highest correlation value is less than 0.50.

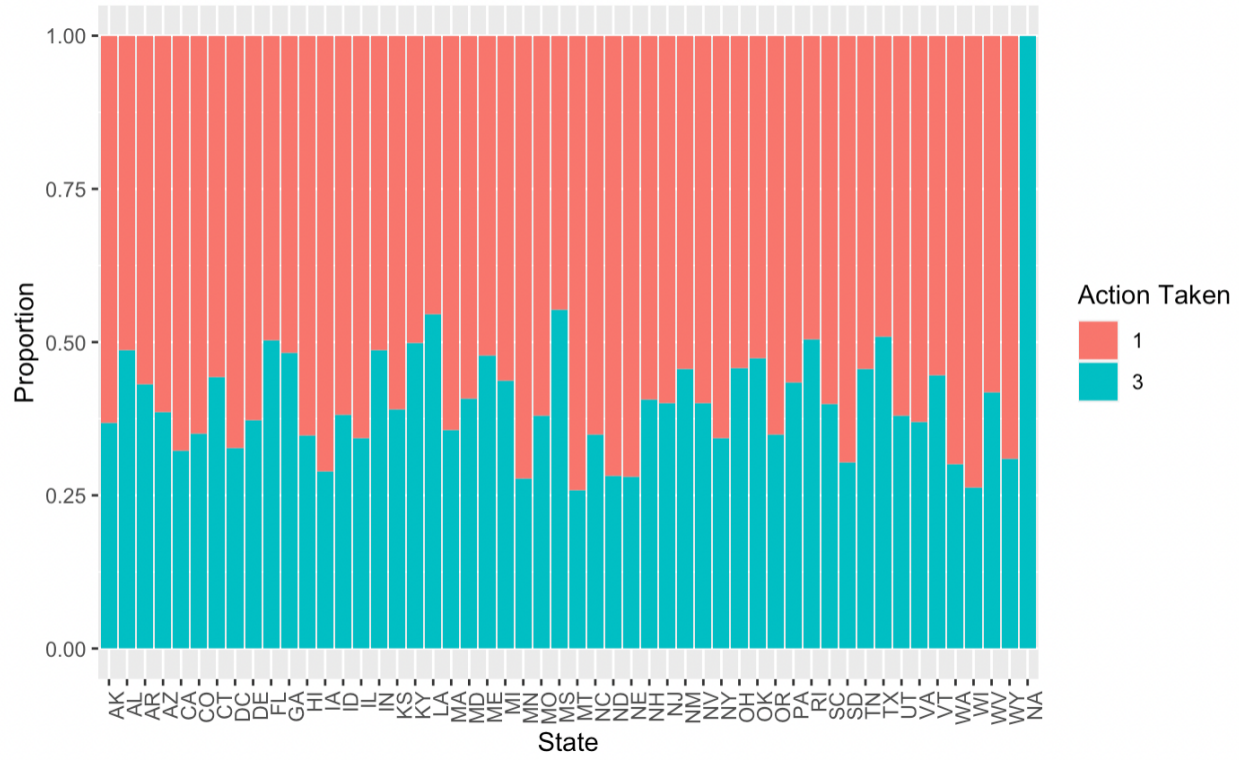
	income	loan_amount	property_value	combined_loan_to_value_ratio
income	1.000000000	0.126441204	0.07630996	-0.001011569
loan_amount	0.126441204	1.000000000	0.49097303	-0.003644335
property_value	0.076309965	0.490973035	1.00000000	-0.010443975
combined_loan_to_value_ratio	-0.001011569	-0.003644335	-0.01044398	1.000000000

Predictors Selection

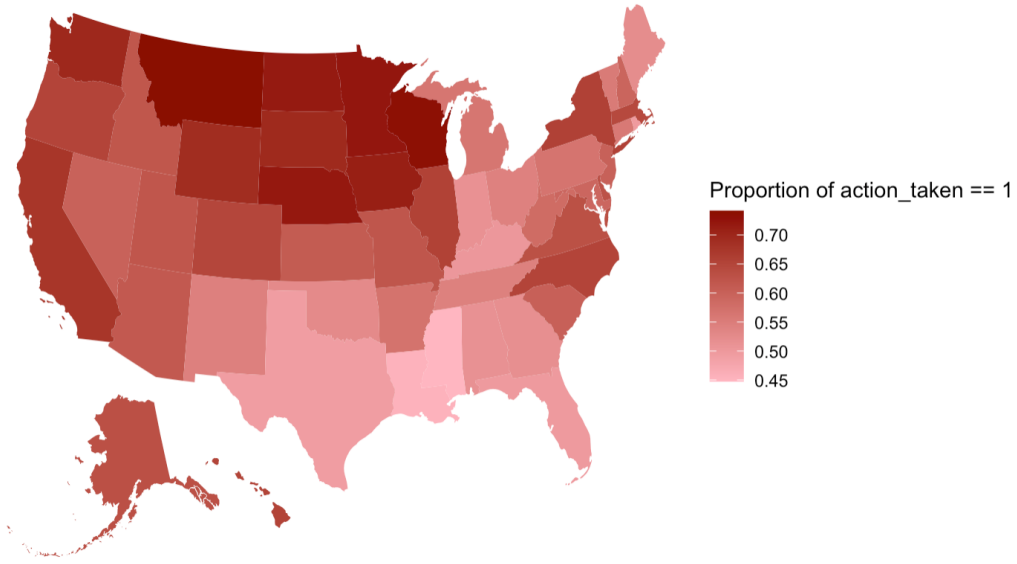
- For numerical variables, we decided to use “income”, “loan_amount”, “property_value”, and “combined_loan_to_value_ratio”.
- All 4 of these variables need transformation. These 4 plots below are log transformed for these 4 numeric variables. Now they are showing normal distribution with slight skewness.
- “Total_points_and_fees” and “multifamily_affordable_units” are removed because nearly all the values are NA. These variables are meaningless for the modeling process.



- **State**: There are subtle variations in the proportion of loan originations across different states. Notably, California (CA) stands out with the highest number of loan applications, as indicated by the distribution histogram. All NA values provided in 2019 were denied. However, a more comprehensive understanding of the likelihood of loan originations is provided by the map plot. It becomes apparent that states in the northern region tend to have a higher proportion of loan originations, while states in the southern region exhibit a lower proportion in comparison.



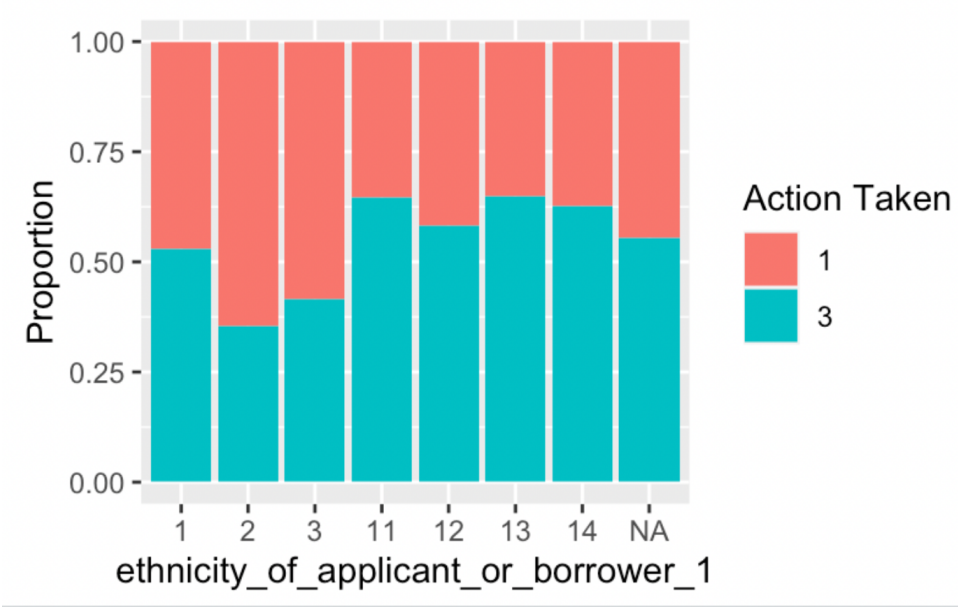
Proportion of action_taken == 1 by State

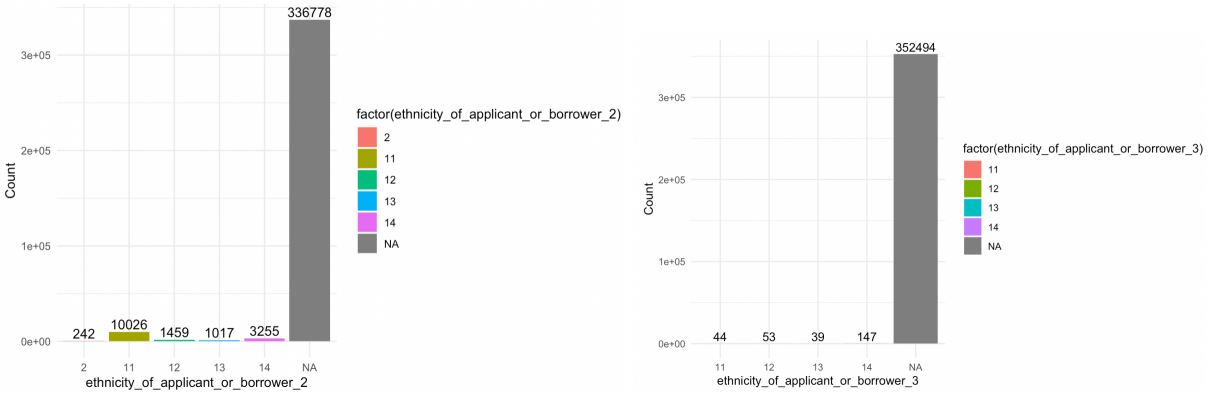


- **Ethnicity** : There are 5 predictors related to ethnicity of borrowers. Upon visualizing ethnicity_of_borrower_or_applicant_1 with a barplot, it becomes apparent that there are significant differences in the likelihood of loans being originated based on ethnicity. However, starting from predictors 2, the number of missing values (NAs) increases

substantially. We have decided to only focus our analysis on the "ethnicity_of_applicant_or_borrower_1".

- For "ethnicity_of_applicant_or_borrower_1", borrower with ethnicity 2 is most likely to be originated, where ethnicity 11 and 13 are equally likely to be rejected.

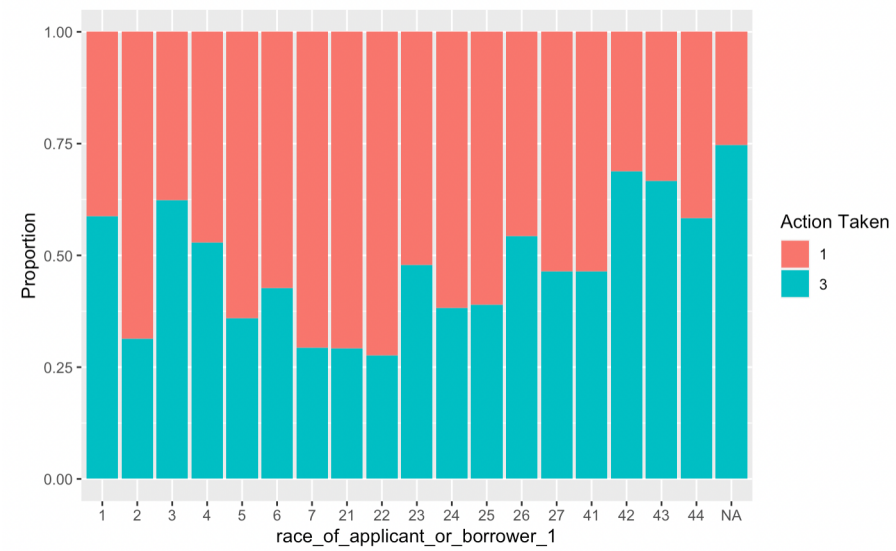


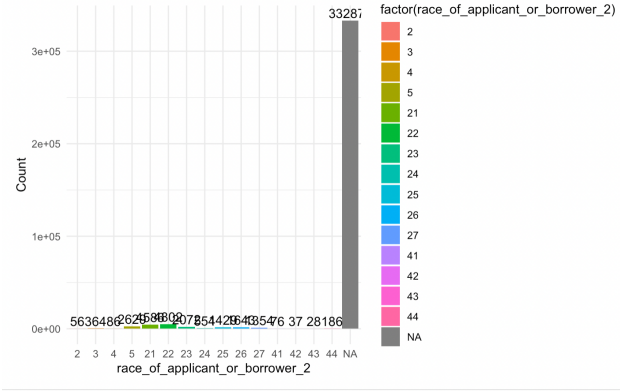


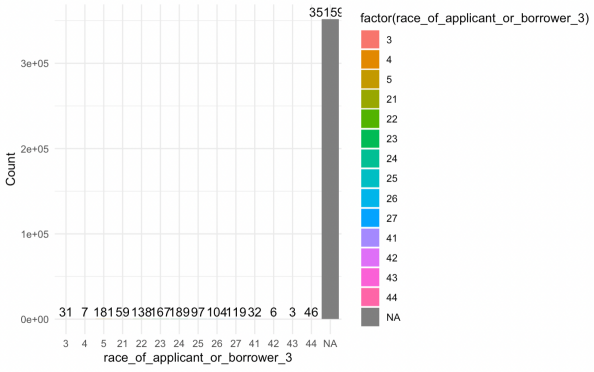
- **Race** : There are also 5 predictors related to race of borrowers. Upon visualizing race_of_borrower_or_applicant_1 with a barplot, it becomes apparent that there are significant differences in the likelihood of loans being originated based on race. However,

starting from predictors 2, the number of missing values (NAs) increases substantially. We have decided to only focus our analysis on the "race_of_applicant_or_borrower_1".

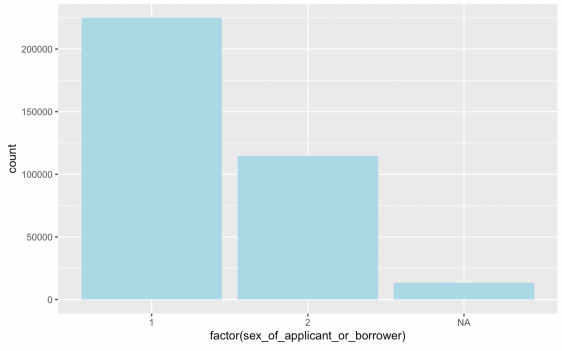
- For “race_of_applicant_or_borrower_1", borrowers with race 2, 7, 21, and 22 are equally likely to have originated, where race 1, 3, 42, 32, 44, and race unknown are likely to be rejected.

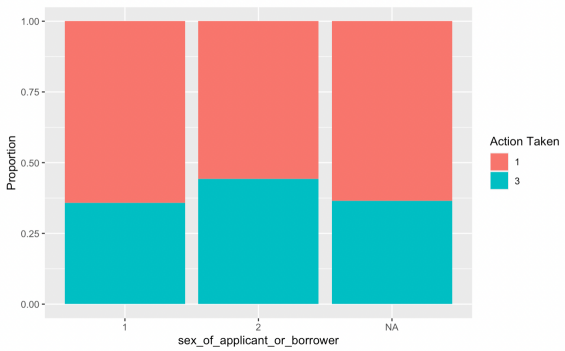




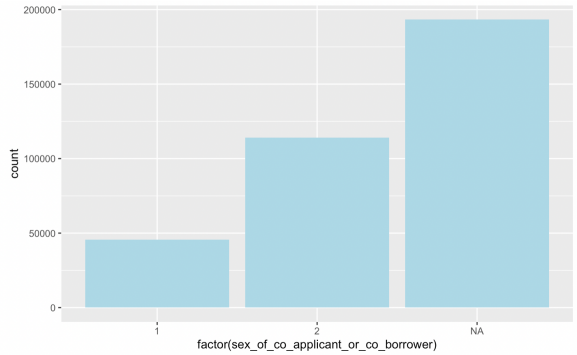


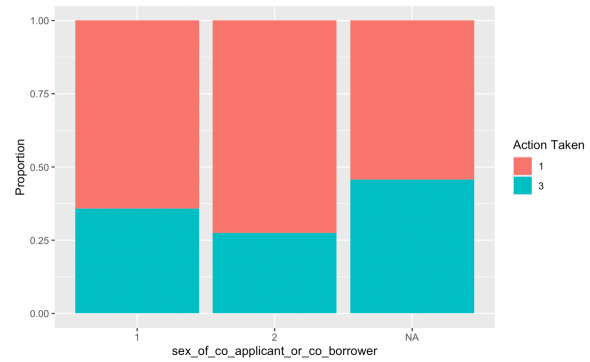
- **Sex** : There are 2 predictors related to sex of borrowers.
- For “sex_of_applicant_or_borower”, there are around twice as many borrowers with sex 1 than borrowers with sex 2, and there are around 10% more applicants with sex 1 have originated than sex 2.



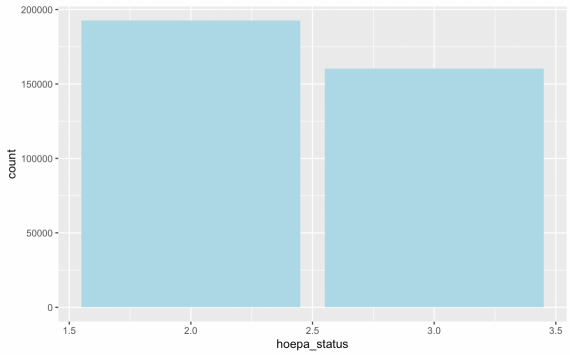


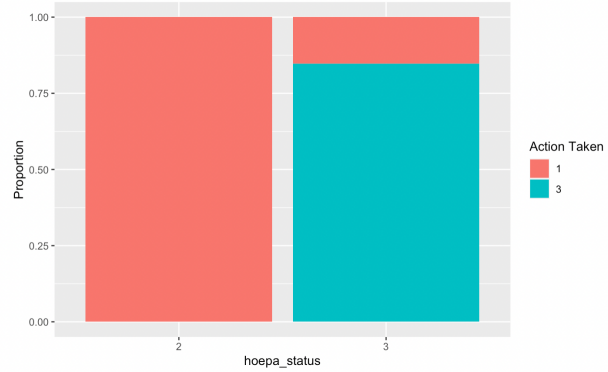
- For “sex_of_co_applicant_or_co_borrower”, the distribution changes to the opposite: there are around twice as many borrowers with sex 2 than borrowers with sex 1. NA values dominated the amount of co borrowers. The proportion of originated applications also changed that borrowers with sex 2 are more likely to be originated.



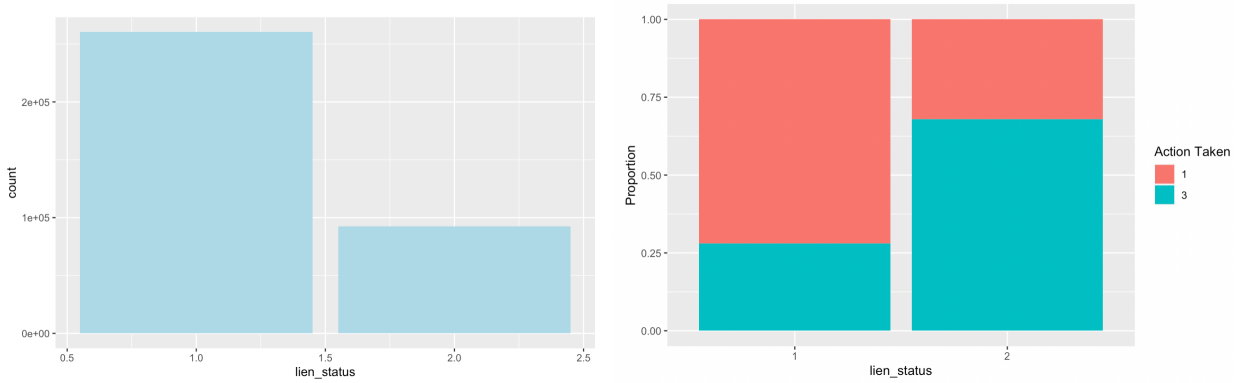


- **hoepa_status** : There is no significant difference on the amount of borrowers with hoepa status 2 or 3. However, it is surprising that borrowers with status 2 all originated in 2019; whereas only a very small percentage of borrowers with status 3 have originated. This is a predictor we need to focus on.

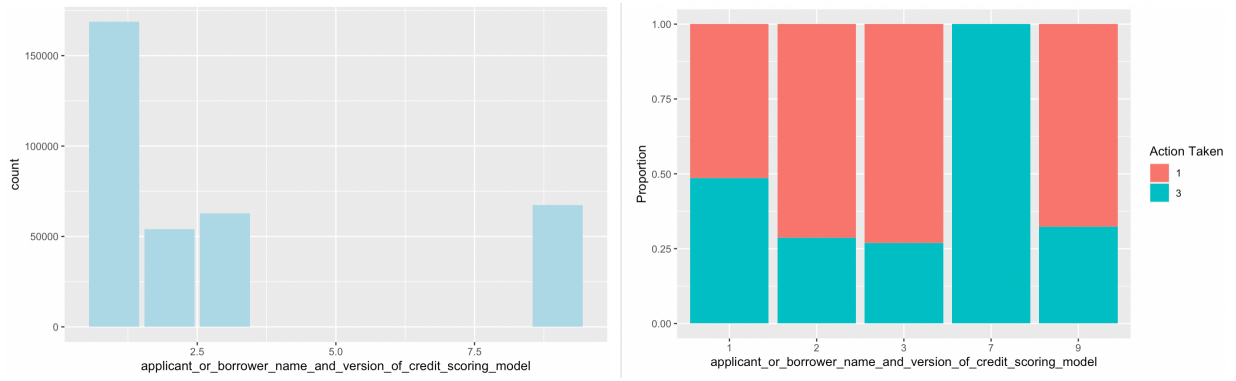




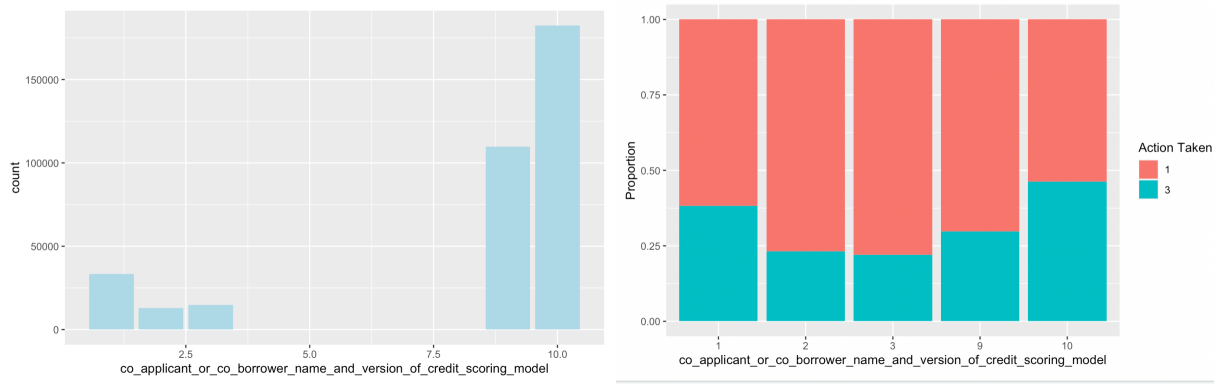
- **lien_status** : There are around 3 times as many borrowers with lien status 1 than 2. And borrowers with lien status 1 are also around 3 times more likely to be originated.



- **Applicant_or_borrower_name_and_version_of_credit_scoring_model** : There are 5 categories of this predictor, 1 being the significant highest amount and 7 being the least. All borrowers with 7 are rejected, 1 has the second highest chance of being rejected. The rest categories are equally likely to have originated.



- **co_applicant_or_co_borrower_name_and_version_of_credit_scoring_model** : There are also 5 categories. 9 and 10 have the significantly highest number of borrowers. It seems like the number of borrowers does not affect the chance of originating. 2 and 3 have the highest proportion of originated.



Preprocessing / Recipes

In order to improve the quality of the result produced by the data, preprocessing is a crucial step during the process. Below are the steps we did for preparing and cleaning the data:

1. **Understand the data.** We first went through all the columns and figured out what kind of data they are. Numerical or categorical. Whether they have missing values or NAs in the columns. Whether there are columns with all the same values(in this case, we decide to not include them, because they will not produce any difference). We take a brief look at the numerical values to see if any possible transformations are needed. Then we went through all the categorical predictors and checked how many unique values they have, this step provided us with the information of how many levels each categorical predictors have.
2. **Change categorical predictors.** After some inspection and analyzing of the data sets, we started to create the recipe. First we add all the predictors we needed. Then we used **`step_mutate`** to manipulate the categorical predictors. Since all the categorical predictor columns have all numeric values, the first thing we did was to change these predictors into factors, this way they will have levels and can be used as categorical columns for the prediction.
3. **Fill in missing values.** Since many of the predictors(both categorical and numerical) have missing values(NAs) in the columns. In order to fix this problem, we used **`step_impute_mode`** columns **`step_impute_mean`** to fill in the missing values.
4. **Further transformation for numerical values.** Having a really large and complicated dataset causes some problems. For the numerical predictors' values, a lot of them are

measured in different units or scales, in order to produce better result, we decided to normalize all of the numeric data using **`step_normalize`**, this help us bring all variables to a common scale and make it easier to use them later on. Then we also used **`step_YeoJohnson`** on all the numerical values to stabilize variance and reduce skewness.

Candidate Models / Model Evaluation / Tuning

Candidate Models

Our group experimented with several models, such as logistic regression, decision tree, random forest, boosted trees, LDA(linear discriminant Analysis). For the computation efficiency, we decide to do the cross validation for each model first and then choose the best model to tune it.

Model Identifier	Type of Model	Engine	Recipe	`tunable()` Hyperparameters in R
`logistic_reg()`	logistic regression	`glm`	`recipe2`	

`decision_tree()`	decision tree	`rpart`	`recipe2`	`tree_depth`
				`min_n`
				`cost_complexity`

`rand_forest()`	random forest	`ranger`	`recipe2`	`mtry`
				`trees`
				`min_n`

`boost_tree()`	boosted tree	`xgboost`	`bt_recipe <- recipe2 %>% step_dummy(all_nominal_predictors())`	`tree_depth`
				`trees`
				`learn_rate`
				`mtry`
				`min_n`
				`loss_reduction`
				`sample_size`
				`stop_iter`

`discrim_linear()`	linear discriminant	`MASS`	`recipe2`	

1. Logistic Regression model:
   Logistic Regression model, which is the type of generalized linear model, is a classification algorithm for counting the probability of binary response variables.

2. Decision Tree model:
   This model partition the data into small parts and then the tree uses explanatory variables to split binarily among each part, which create the branches. The binary splitting process continues until all the observations fall into a certain group, which is also called a terminal node. The predictions are made by following the branches and reaching the terminal node.

3. Random forest model:
   Random forest is the ensemble technique extending from the decision tree for classification. The model created multiple decision trees generated by bootstrapping with random sampling of predictors. Unlike single decision trees that tend to use the strong predictors from the data set in the top split, the random forest restricts each split to consider only the subset of predictors. Thus, we diversify the bagged tree to increase the accuracy and reduce overfitting.

4. Boost tree model(XGboost):
   We set the engine of "xgboost" to specify the Extreme Gradient Boosting(XGboost) model we want to use from the boosting algorithm. XG boost is the ensemble learning technique based on gradient boosting that trees grow sequentially based on the previous tree to form the better model. Specifically, XG boost includes some regularization that contributes to better control of the model and reduces the risk of overfitting. Noticing that R output error when there are the nominal predictors in the XG boost model, we add the additional `step_dummy` to convert all the nominal predictors to on the recipe.

5. Linear discriminant:

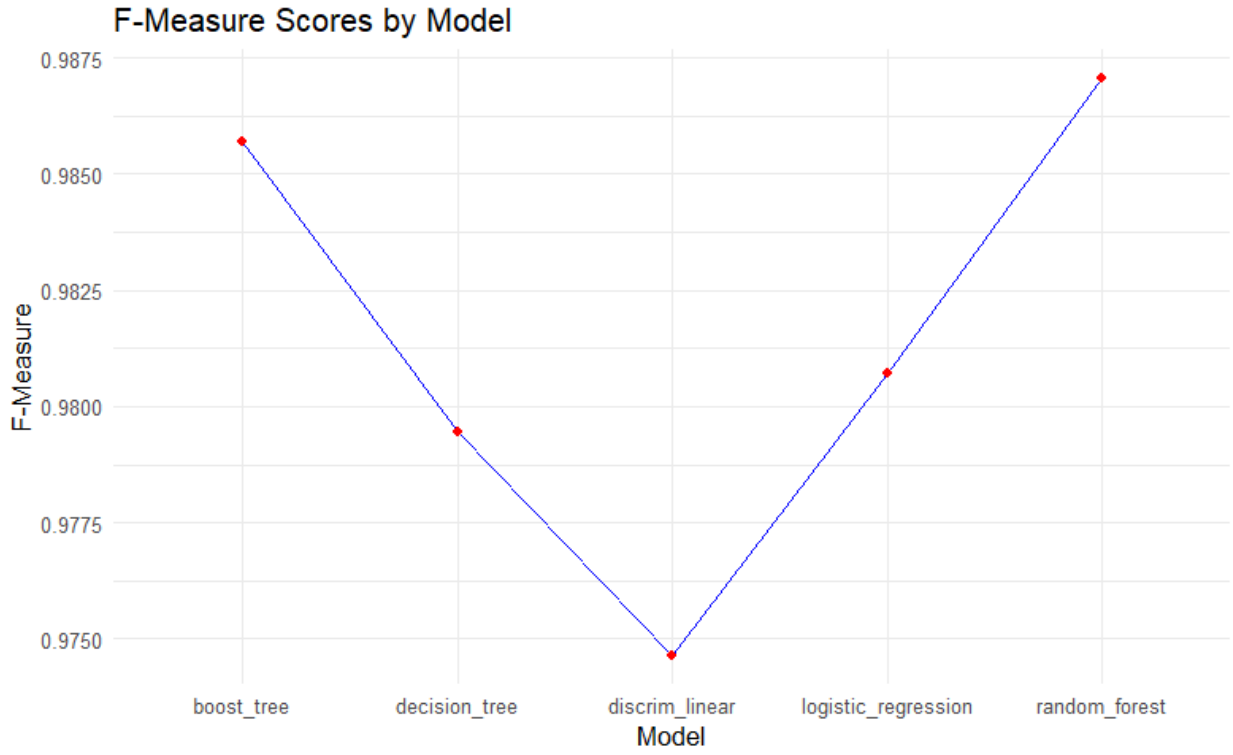
   Linear discriminant analysis(LDA) is one of the estimates of the density function to approximate the Bayes classifier. Based on the Bayes classifier, LDA is able to trade the total error rate with more accurate prediction for one of response classes that the researchers are more interested in.

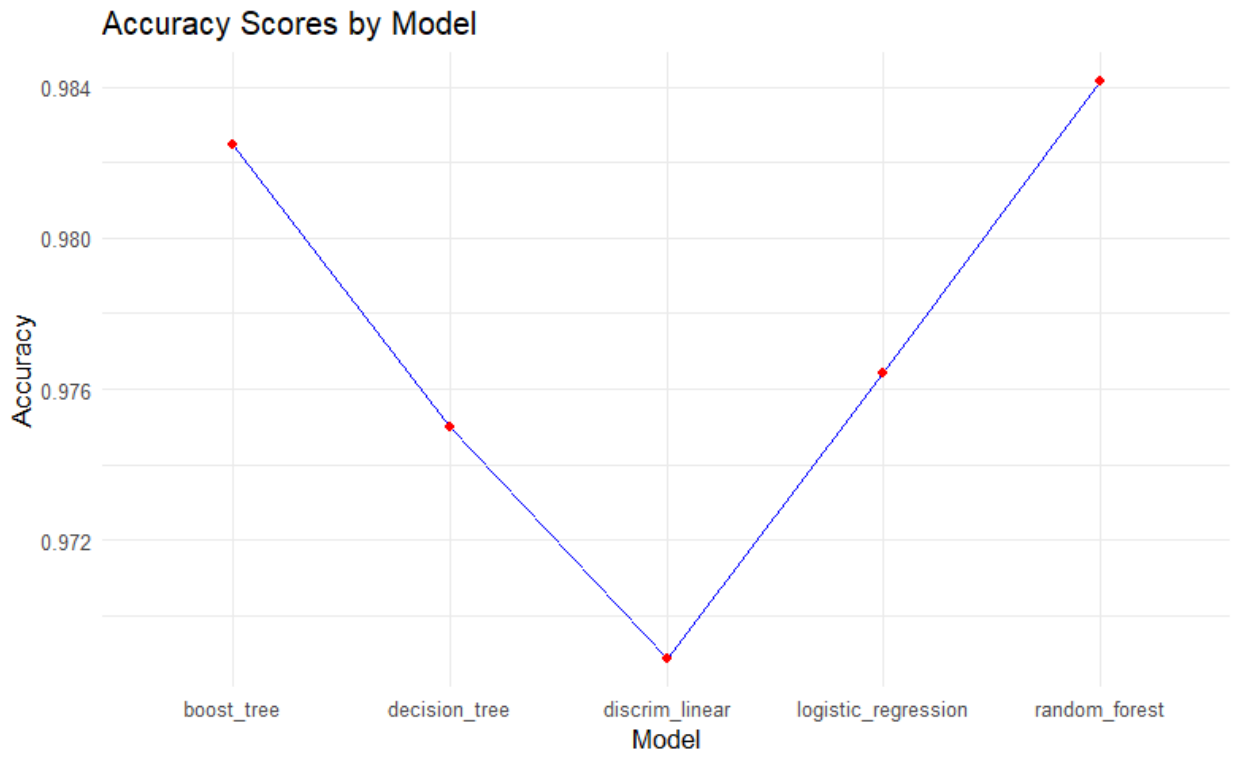
Model Evaluation

The following table is the accuracy using 10 fold cross validation to measure the performance of the candidate models.

<table>
<tr><th>Model Identifier</th><th>Metric score(accuracy)</th></tr>
<tr><td>logistic_reg()</td><td>
<table>
<tr><th>.metric
<chr></th><th>.estimator
<chr></th><th>mean
<dbl></th><th>n
<int></th><th>std_err
<dbl></th><th>.config
<chr></th></tr>
<tr><td>accuracy</td><td>binary</td><td>0.9764156</td><td>10</td><td>3.775006e-04</td><td>Preprocessor1_Model1</td></tr>
<tr><td>f_meas</td><td>binary</td><td>0.9807173</td><td>10</td><td>3.128128e-04</td><td>Preprocessor1_Model1</td></tr>
<tr><td>roc_auc</td><td>binary</td><td>0.9978569</td><td>10</td><td>5.769886e-05</td><td>Preprocessor1_Model1</td></tr>
</table>
</td></tr>
<tr><td>decision_tree()</td><td>
<table>
<tr><th>.metric
<chr></th><th>.estimator
<chr></th><th>mean
<dbl></th><th>n
<int></th><th>std_err
<dbl></th><th>.config
<chr></th></tr>
<tr><td>accuracy</td><td>binary</td><td>0.9749729</td><td>10</td><td>0.0002854398</td><td>Preprocessor1_Model1</td></tr>
<tr><td>f_meas</td><td>binary</td><td>0.9794410</td><td>10</td><td>0.0002383796</td><td>Preprocessor1_Model1</td></tr>
<tr><td>roc_auc</td><td>binary</td><td>0.9920046</td><td>10</td><td>0.0001121041</td><td>Preprocessor1_Model1</td></tr>
</table>
</td></tr>
<tr><td>rand_forest()</td><td>
<table>
<tr><th>.metric
<chr></th><th>.estimator
<chr></th><th>mean
<dbl></th><th>n
<int></th><th>std_err
<dbl></th><th>.config
<chr></th></tr>
<tr><td>accuracy</td><td>binary</td><td>0.9841448</td><td>9</td><td>2.977174e-04</td><td>Preprocessor1_Model1</td></tr>
<tr><td>f_meas</td><td>binary</td><td>0.9870603</td><td>9</td><td>2.445766e-04</td><td>Preprocessor1_Model1</td></tr>
<tr><td>roc_auc</td><td>binary</td><td>0.9991158</td><td>9</td><td>2.802524e-05</td><td>Preprocessor1_Model1</td></tr>
</table>
</td></tr>
<tr><td>boost_tree()</td><td>
<table>
<tr><th>.metric
<chr></th><th>.estimator
<chr></th><th>mean
<dbl></th><th>n
<int></th><th>std_err
<dbl></th><th>.config
<chr></th></tr>
<tr><td>accuracy</td><td>binary</td><td>0.9824563</td><td>10</td><td>2.369996e-04</td><td>Preprocessor1_Model1</td></tr>
<tr><td>f_meas</td><td>binary</td><td>0.9856914</td><td>10</td><td>1.948128e-04</td><td>Preprocessor1_Model1</td></tr>
<tr><td>roc_auc</td><td>binary</td><td>0.9989248</td><td>10</td><td>2.972031e-05</td><td>Preprocessor1_Model1</td></tr>
</table>
</td></tr>
<tr><td>discrim_linear()</td><td>
<table>
<tr><th>.metric
<chr></th><th>.estimator
<chr></th><th>mean
<dbl></th><th>n
<int></th><th>std_err
<dbl></th><th>.config
<chr></th></tr>
<tr><td>accuracy</td><td>binary</td><td>0.9688783</td><td>10</td><td>3.081954e-04</td><td>Preprocessor1_Model1</td></tr>
<tr><td>f_meas</td><td>binary</td><td>0.9746545</td><td>10</td><td>2.561234e-04</td><td>Preprocessor1_Model1</td></tr>
<tr><td>roc_auc</td><td>binary</td><td>0.9961356</td><td>10</td><td>6.054366e-05</td><td>Preprocessor1_Model1</td></tr>
</table>
</td></tr>
</table>

Below, you'll find graphs that compare the various models performance based on cross validation listed above:





Among the five models evaluated, linear discriminant analysis consistently exhibited the poorest performance, with F-measure scores falling below 0.9750 and accuracy scores below 0.970. On the other hand, both the random forest model and the XGBoost tree model, which are extensions of the single decision tree model, displayed superior overall performance. However, the standalone decision tree model did not achieve the desired level of performance, with an accuracy score around 0.975 and an F-measure score below 0.98, ranking it as the second-worst performer among the five models.

The logistic regression model, on the other hand, achieved an F-measure score of approximately 0.98 and an accuracy score of around 0.976. Although its performance falls short when compared to the XGBoost tree model and random forest model, it offers the advantage of quicker computation times. In scenarios where computational efficiency takes precedence over absolute accuracy, the logistic regression model may be a suitable choice.

As mentioned previously, the XGBoost tree model and the random forest model stand out as the top two performers. The XGBoost tree model achieved an impressive F-measure score of approximately 0.9855, along with an accuracy score of 0.982. In comparison, the random forest model delivered even stronger results, with an F-measure score of around 0.987 and an accuracy score of 0.984. Consequently, based on the initial cross-validation results, the random forest model emerges as the preferred choice for prediction tasks due to its superior performance, boasting the highest F-measure and accuracy scores among the evaluated models.

Tuning

Given that we recognize random forest as the model that achieved the highest accuracy previously, our strategy now revolves around optimizing the random forest model by tuning its parameters to identify the most suitable values. We're concentrating on adjusting two parameters: "`trees`" and "`min_n`," with each parameter undergoing testing at three different levels. For "`trees`," we're evaluating the values 1, 1000, and 2000, but we've opted to exclude the "trees = 1" option because it lacks reference significance. In the case of "`min_n`," we're

conducting tests with the values 2, 21, and 40. Below, you will find a table displaying the outcomes following the tuning process.

A tibble: 18 × 8

trees <int>	min_n <int>	.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
1000	2	accuracy	binary	0.9843180	9	1.709229e-04	Preprocessor1_Model1
1000	2	f_meas	binary	0.9872064	9	1.405420e-04	Preprocessor1_Model1
1000	2	roc_auc	binary	0.9991304	9	1.463762e-05	Preprocessor1_Model1
2000	2	accuracy	binary	0.9843306	9	1.496922e-04	Preprocessor1_Model2
2000	2	f_meas	binary	0.9872165	9	1.233056e-04	Preprocessor1_Model2
2000	2	roc_auc	binary	0.9991318	9	1.508406e-05	Preprocessor1_Model2
1000	21	accuracy	binary	0.9841952	9	1.141687e-04	Preprocessor1_Model3
1000	21	f_meas	binary	0.9871009	9	9.401237e-05	Preprocessor1_Model3
1000	21	roc_auc	binary	0.9991211	9	1.494694e-05	Preprocessor1_Model3
2000	21	accuracy	binary	0.9842109	9	1.189587e-04	Preprocessor1_Model4
2000	21	f_meas	binary	0.9871135	9	9.807850e-05	Preprocessor1_Model4
2000	21	roc_auc	binary	0.9991215	9	1.460315e-05	Preprocessor1_Model4
1000	40	accuracy	binary	0.9840755	9	1.132885e-04	Preprocessor1_Model5
1000	40	f_meas	binary	0.9869992	9	9.371213e-05	Preprocessor1_Model5
1000	40	roc_auc	binary	0.9991050	9	1.498767e-05	Preprocessor1_Model5
2000	40	accuracy	binary	0.9840818	9	1.209461e-04	Preprocessor1_Model6
2000	40	f_meas	binary	0.9870040	9	9.987635e-05	Preprocessor1_Model6
2000	40	roc_auc	binary	0.9991066	9	1.444959e-05	Preprocessor1_Model6

Furthermore, we would like to incorporate the outcomes obtained with parameters set to `trees = 500` and `min_n = 2, 21, 40` to compare with the models above. Here are the results.

`trees = 500` and `min_n = 2`

.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
accuracy	binary	0.9842582	9	2.803731e-04	Preprocessor1_Model1
f_meas	binary	0.9871562	9	2.302231e-04	Preprocessor1_Model1
roc_auc	binary	0.9991139	9	2.755025e-05	Preprocessor1_Model1

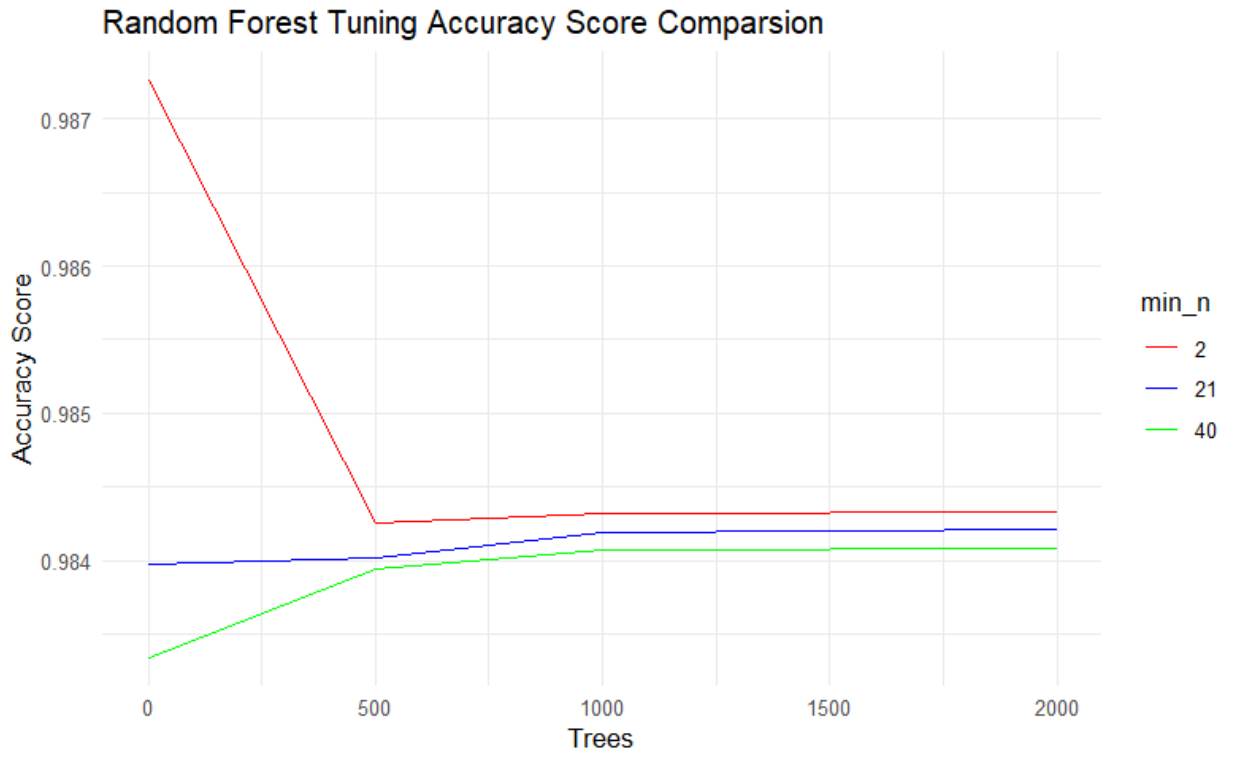
`trees = 500` and `min_n = 21`

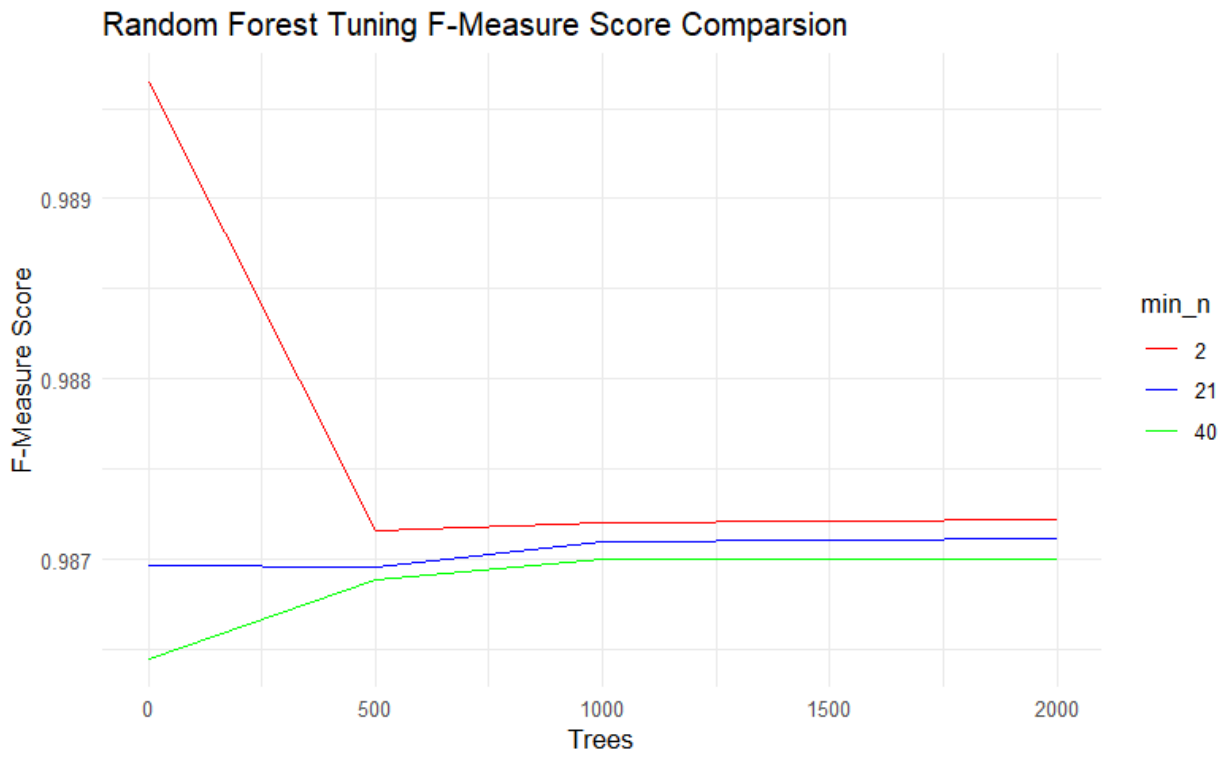
.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
accuracy	binary	0.9840251	9	3.032432e-04	Preprocessor1_Model1
f_meas	binary	0.9869603	9	2.490938e-04	Preprocessor1_Model1
roc_auc	binary	0.9991072	9	2.879753e-05	Preprocessor1_Model1

`trees = 500` and `min_n = 40`

.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
accuracy	binary	0.9839432	9	2.770087e-04	Preprocessor1_Model1
f_meas	binary	0.9868897	9	2.281708e-04	Preprocessor1_Model1
roc_auc	binary	0.9990949	9	2.961359e-05	Preprocessor1_Model1

Evaluating the Tuned Random Forest Model:





In instances where the number of `trees` is set to be greater than 1000, it is observed that adjustments to the parameters `min_n` and `trees` have a negligible impact on performance. Therefore, our focus shifts to optimizing `min_n` and `trees` when trees are smaller or equal to 1000. When `trees` are less than 1000, `min_n = 21` and `min_n = 40` have been observed to enhance performance in conjunction with an increase in `trees`. Specifically, `min_n = 21` has been found to consistently yield better overall performance than `min_n = 40`. However, it is noteworthy that `min_n = 2` consistently results in the best performance, even as `trees` increase, which is in contrast to the behavior observed with the other two `min_n` values. Given the nearly identical accuracy and F-measure scores between models with `trees` = 500 and `trees` = 1000, and taking into account computational runtime, we opt to utilize the random forest model with `trees` = 500 and `min_n` = 2.

Discussion of Final Model

The final model we choose is the random forest model with `trees = 500` and `min_n = 2` for generating the predictions because it maintains a significantly high level of accuracy and F-measure scores while managing computational runtime effectively.

The random forest model typically provides higher accuracy because it combines multiple decision trees and thus reduces the risk of overfitting. In addition, XG boost tree model and random forest model both belong to ensemble learning methods but we need to transform all nominal predictors to the dummies for the XG boost model while random forest model incorporates mixed types of data and works smoothly on numeric and categorical predictors. Moreover, the random forest model doesn't necessarily require normalization and standardization, which means that we are able to extend our model easier as we don't need to scale the data if we are given the new numeric predictors. Random forest model is able to capture the complex, non-linear, and high-dimensional interaction between the data compared to the logistic regression and LDA.

Despite the fact that random forest models provide the highest accuracy among the five models, random forest models do exhibit limitations. The rise in flexibility correlates with a reduction in interpretability. Having a large number of trees and data would also result in significant computational expenses. While they're more resistant to overfitting than single decision trees, random forest models can still overfit if not tuned correctly, and the tuning process can be time-consuming.

Some possible improvements include adjusting or tuning more aspects of the hyperparameters. This includes exploring more levels of hyperparameter tuning, going beyond the initial 3 levels. We can also consider the idea of tuning other models, like boosting trees, to broaden the range of possible candidate models. In addition, we can also try testing more advanced models like stacked models, to potentially achieve a higher accuracy and F-measures score. We can also include additional predictors like state. These predictors were not included in the current model because it would introduce a significant number of factor levels leading to computational complexity and resource requirements.

Appendix A: Final Annotated Script

Youtube Link for Script Verification Recording: https://youtu.be/6bxAD7wl0Cc

```
# Load the library
library(tidymodels)
library(tidyverse)
library(ISLR)
library(xgboost)
library(stacks)


# Import data
# Convert response variable as factor
train2 <- read_csv("train2.csv")
train2$action_taken <- as.factor(train2$action_taken)

# Recipe of the model
recipe2 <- recipe(action_taken ~
                    combined_loan_to_value_ratio +
                    loan_type +
                    income +
                    loan_purpose +
                    construction_method+
                    occupancy_type+
                    property_value +
                    loan_amount +
                    ethnicity_of_applicant_or_borrower_1 +
                    ethnicity_of_co_applicant_or_co_borrower_1 +
                    race_of_applicant_or_borrower_1 +
                    race_of_co_applicant_or_co_borrower_1 +
                    sex_of_applicant_or_borrower+
                    sex_of_co_applicant_or_co_borrower+
                    age_of_applicant_or_borrower+
                    age_of_co_applicant_or_co_borrower+
                    automated_underwriting_system_1 +
                    hoepa_status+
                    lien_status+
applicant_or_borrower_name_and_version_of_credit_scoring_model+
co_applicant_or_co_borrower_name_and_version_of_credit_scoring_m
odel+
```

```
                    balloon_payment+
                    interest_only_payments+
                    negative_amortization+
                    other_non_amortizing_features+
                    manufactured_home_secured_property_type+
                    manufactured_home_land_property_interest+
                    submission_of_application+
                    initially_payable_to_your_institution+
                    open_end_line_of_credit+
                    business_or_commercial_purpose, data =
train2) %>%
# convert selected numeric/categorical predictors into factor
# So these predictors are treated as discrete categories rather
than continuous numeric values.
  step_mutate(loan_type = as.factor(loan_type),
              loan_purpose = as.factor(loan_purpose),
              construction_method =
as.factor(construction_method),
              occupancy_type = as.factor(occupancy_type),
             ethnicity_of_applicant_or_borrower_1 =
as.factor(ethnicity_of_applicant_or_borrower_1),
             ethnicity_of_co_applicant_or_co_borrower_1 =

as.factor(ethnicity_of_co_applicant_or_co_borrower_1),
             race_of_applicant_or_borrower_1 =
as.factor(race_of_applicant_or_borrower_1),
             race_of_co_applicant_or_co_borrower_1 =

as.factor(race_of_co_applicant_or_co_borrower_1),
             sex_of_applicant_or_borrower =
as.factor(sex_of_applicant_or_borrower),
             sex_of_co_applicant_or_co_borrower =
as.factor(sex_of_co_applicant_or_co_borrower),
             age_of_applicant_or_borrower =
as.factor(age_of_applicant_or_borrower),
             age_of_co_applicant_or_co_borrower=
as.factor(age_of_co_applicant_or_co_borrower),
             automated_underwriting_system_1 =
as.factor(automated_underwriting_system_1),
             hoepa_status = as.factor(hoepa_status),
```

```
              lien_status = as.factor(lien_status),
applicant_or_borrower_name_and_version_of_credit_scoring_model=
as.factor(applicant_or_borrower_name_and_version_of_credit_scori
ng_model),

co_applicant_or_co_borrower_name_and_version_of_credit_scoring_m
odel =
as.factor(co_applicant_or_co_borrower_name_and_version_of_credit
_scoring_model),
              balloon_payment = as.factor(balloon_payment),
              interest_only_payments=
as.factor(interest_only_payments),
              negative_amortization =
as.factor(negative_amortization),
              other_non_amortizing_features =
as.factor(other_non_amortizing_features),
              manufactured_home_secured_property_type =

as.factor(manufactured_home_secured_property_type),
              manufactured_home_land_property_interest =

as.factor(manufactured_home_land_property_interest),
              submission_of_application =
as.factor(submission_of_application),
              initially_payable_to_your_institution =

as.factor(initially_payable_to_your_institution),
              open_end_line_of_credit =
as.factor(open_end_line_of_credit),
              business_or_commercial_purpose =
as.factor(business_or_commercial_purpose)) %>%
  # filling in missing values for categorical predictors by
using the most frequent values
  step_impute_mode(all_nominal_predictors()) %>%
  # filling in missing values for numeric predictors by using
mean values
  step_impute_mean(all_numeric_predictors()) %>%
  # standardize numeric variables
  step_normalize(all_numeric_predictors())  %>%
  # Yeo Johnson transformation on predictors
  step_YeoJohnson(all_numeric_predictors())
```

```r
# random forest models
rf_model <- rand_forest(trees = 500, min_n = 2) %>%
  set_mode("classification") %>%
  set_engine("ranger")

# Workflow
rf_workflow <- workflow() %>%
  add_recipe(recipe2) %>%
  add_model(rf_model)


# fit the data
rf_workflow_fit <- rf_workflow %>%
  fit(data = train2)

# import test data
test <- read_csv("test2.csv")

#  make prediction on the test data
#  bind the predicted values to the id column in the test data
prediction <- rf_workflow_fit %>% predict(new_data = test) %>%
  bind_cols(id = test$id)


# exporting prediction as csv file for Kaggle submission
prediction %>%
   rename(action_taken = .pred_class) %>%
   write_csv("rf_predictions.csv")
```

Appendix B: Team Member Contributions

Zhuowei Deng:

- Creating the recipe
- Creating and comparing candidate models
- Report: Introduction, Appendix A
- Report: Helped edit “Discussion of Final Model” and “Tuning”

Liaohan Wang

- Predictor selection and exploration
- Visualizations
- Report : Exploratory Data Analysis

Yifan Jiang

- Preprocessing data and the recipe
- Initial model
- Report: Preprocessing / Recipes

Tou-Chia Chang

- Creating the recipes and models
- Models Evaluation
- Report: Candidate Models/ Models Evaluation/ Tuning
- Report: Helped edit “Discussion of Final Model”