1. Introduction

In this project, we aim to choose the best model that will predict the action taken on the loan application based on the loan data from Wells Fargo National Bank in 2019. We try to predict which loans are approved and which ones are denied. Employment status, gross monthly income, desired term, loan purpose, and loan amount are required elements listed in the Wells Fargo Personal Loans Application Checklist. From this official website, we hypothesize predictor variables income, loan_term, loan_purpose, and loan_amount are associated with the response variable, action_taken.

2. Exploratory Data Analysis

2.1 The relationship between the predictor variables

2.1.1 The relationship between Age Group and Gender

We used a bar chart to investigate the relationship between age group and gender.

- Gender code descriptions: 1:Male, 2:Female, 3:Information not provided by applicant in mail internet or telephone application, 4:Not applicable, 6:Applicant selected both male and female

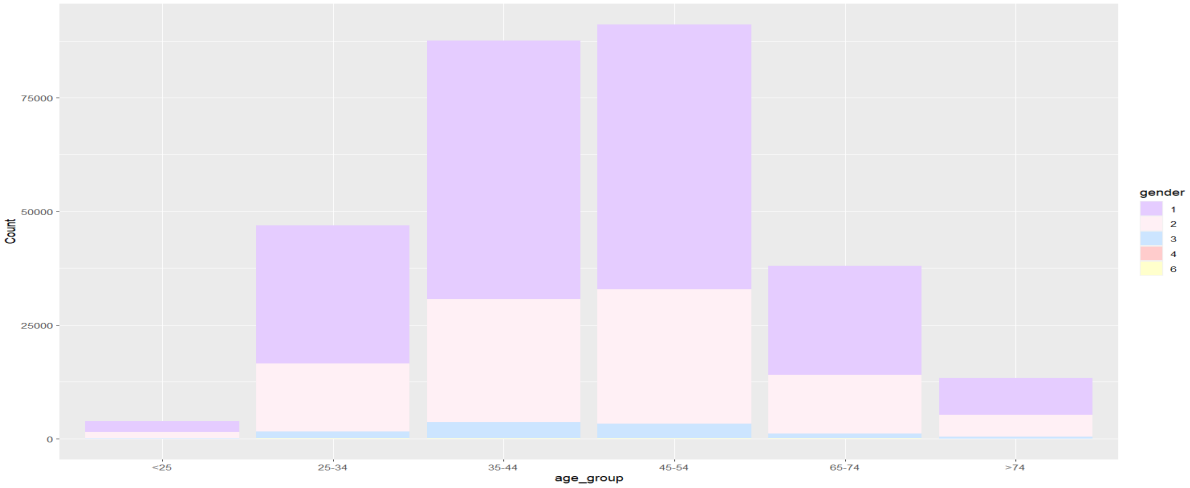


Figure 1: Bar chart Showing cross-analysis of age group and gender. From this bar chart, we can see that the age groups from 35 to 54 make up the largest population. It can also be seen that males account for the largest population of all age groups, followed by females.

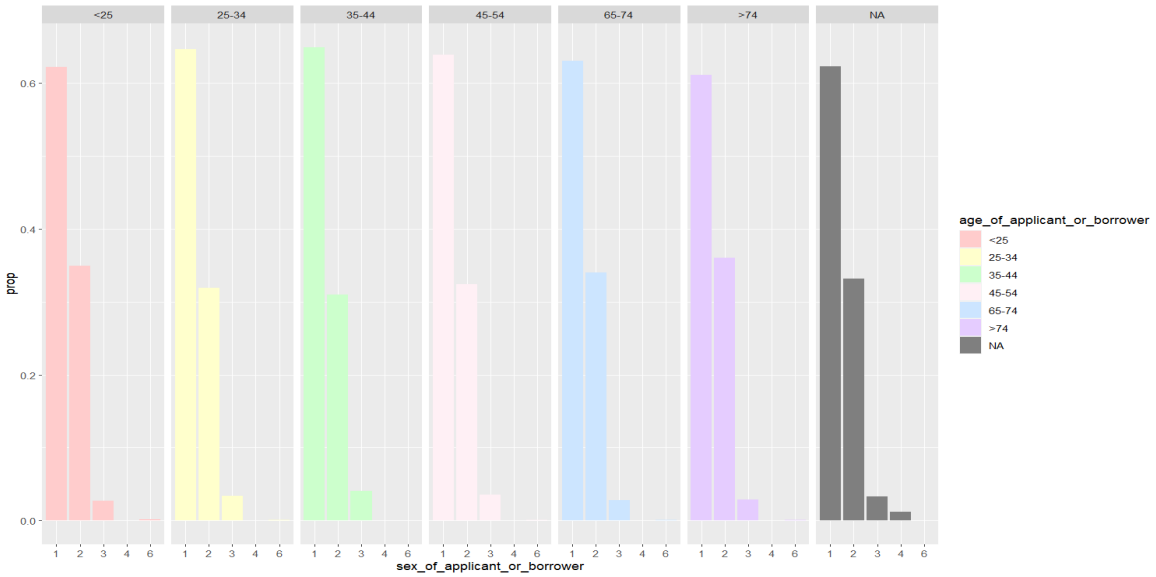


Figure 2: Bar chart Showing cross-analysis of the proportions of each age group and gender. From this bar chart, we can see that the gender variables have almost identical distribution regardless of age group. Through these results, we can infer the possibility that the correlation between the age group variable and the gender variable is not high.

2.1.2 The relationship between Loan type and Loan amount

We created a box plot to investigate the relationship between loan type and loan amount variables. However it was difficult to see the relationship between the two variables because there are outliers that have a large influence on the distribution of the loan amount variable.

- Loan type Descriptions:
  1: Conventional (not insured or guaranteed by FHA VA RHS or FSA)
  2: Federal Housing Administration insured (FHA)
  3: Veterans Affairs guaranteed (VA)
  4: USDA Rural Housing Service or the Farm Service Agency guaranteed (RHS or FSA)

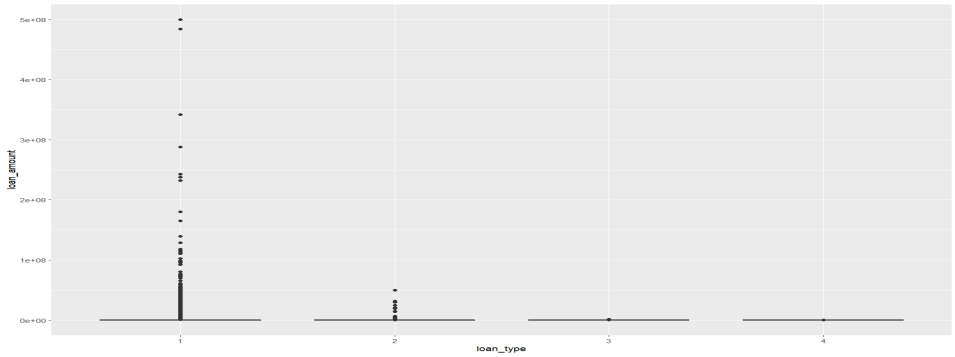


Figure 3: box plot Showing relationship of the loan type and loan amount. From this box plot, it can be identified that outliers with great influence exist in loan amount variable.

To explore the relationship between the loan type and the loan amount, we adjusted the loan amount variable to 0.05-0.95 quantiles.

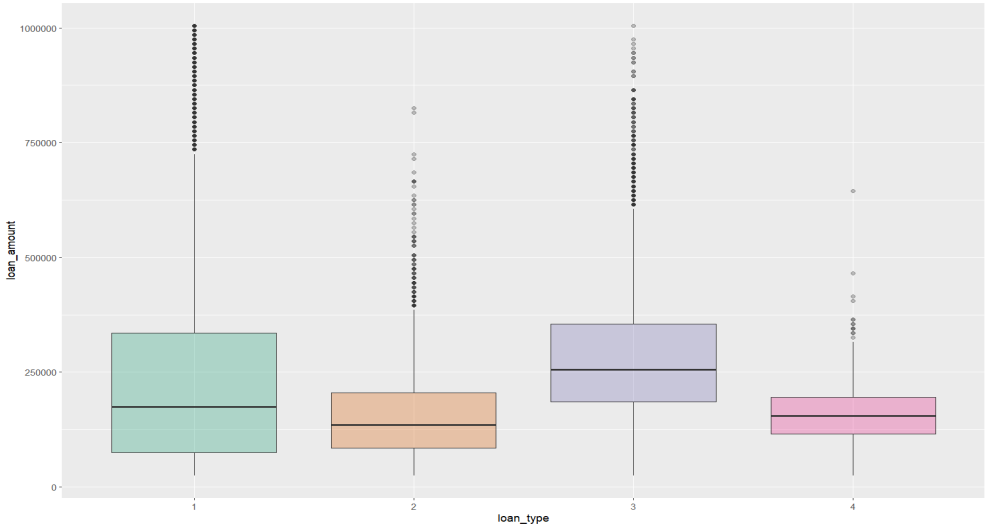


Figure 4: box plot Showing relationship of the loan type and loan amount(with 0.05-0.95 quantile.) From this box plot, we can see that when the loan type is 3:Veterans Affairs guaranteed, the average loan amount is the highest, and when the loan type is 2:Federal Housing Administration insured, the average loan amount is the lowest.

2.2 The relationship between response variable and predictor variables

2.2.1 The relationship between Income and Action taken

We created a bar plot to investigate the relationship between income and action taken variables. However it was difficult to see the relationship between the two variables because there are outliers that have a large influence on the distribution of the income variable.

- Action taken code descriptions: 1:Loan originated 2:Application approved but not accepted 3:Application denied 4:Application withdrawn by applicant 5: File closed for incompleteness 6: Purchased loan 7: Preapproval request denied 8: Preapproval request approved but not accepted

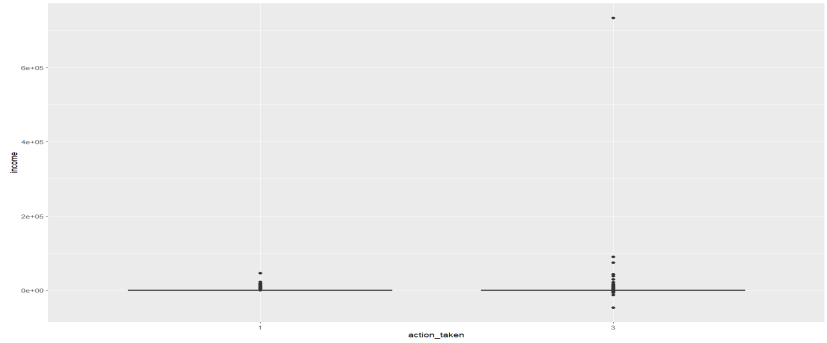


Figure 5:Box plot showing the relationship between average income and action_taken. It is difficult to identify the relationship between the two variables due to outliers with great influence.

To explore the relationship between the income and action taken, we adjusted the income variable to 0.05-0.95 quantiles.

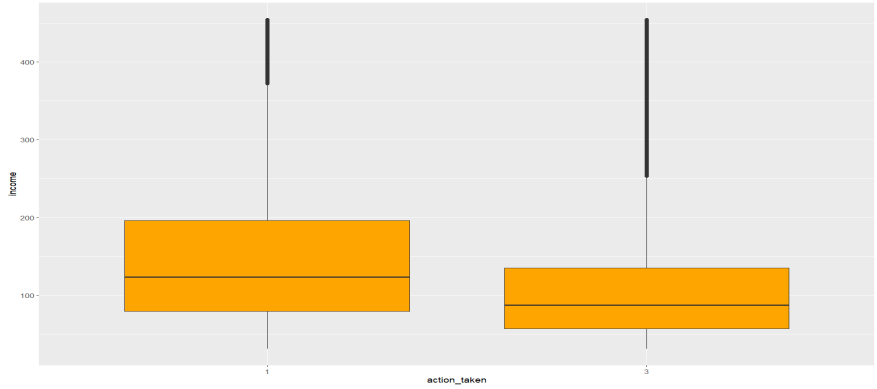


Figure 6: Box plot Showing the relationship between average income(with 0.05-0.95 quantiles) and loan originated. From this box plot, we can see that the average income of those whose loans originated is higher than that of those whose applications were denied.

2.2.2 The relationship between loan amount and action taken

We created a bar plot to investigate the relationship between loan amount and action taken variables. However it was difficult to see the relationship between the two variables because there are outliers that have a large influence on the distribution of the loan amount variable. Therefore we adjusted the income variable to 0.05-0.95 quantiles.

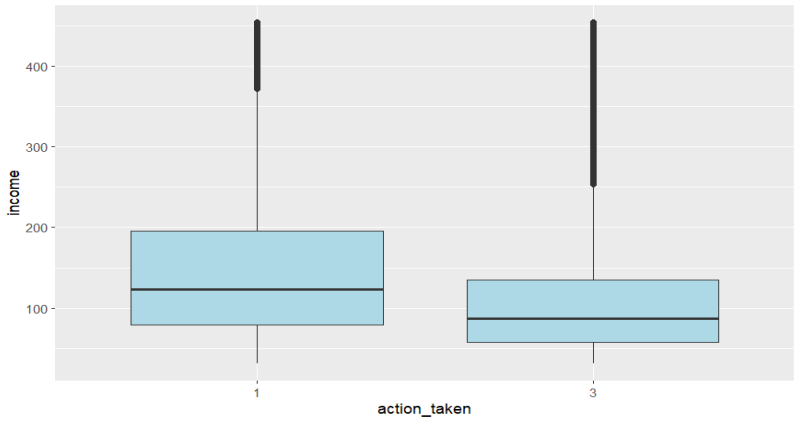


Figure 7: Box plot showing the relationship between loan_amount and action_taken. From the box plot, we can see that the median loan amount of loan originated is higher than median loan amount of application denied. The interquartile range of loan amount for loan originated is higher than the interquartile range of loan amount for application denied.

2.2.3 The relationship between loan purpose and action taken

We used a bar chart to investigate the relationship between loan purpose and action taken.

- Loan purpose code descriptions: 1:Home purchase 2: Home improvement 31: Refinancing 32: Cash-out refinancing 4: Other purpose 5: Not applicable

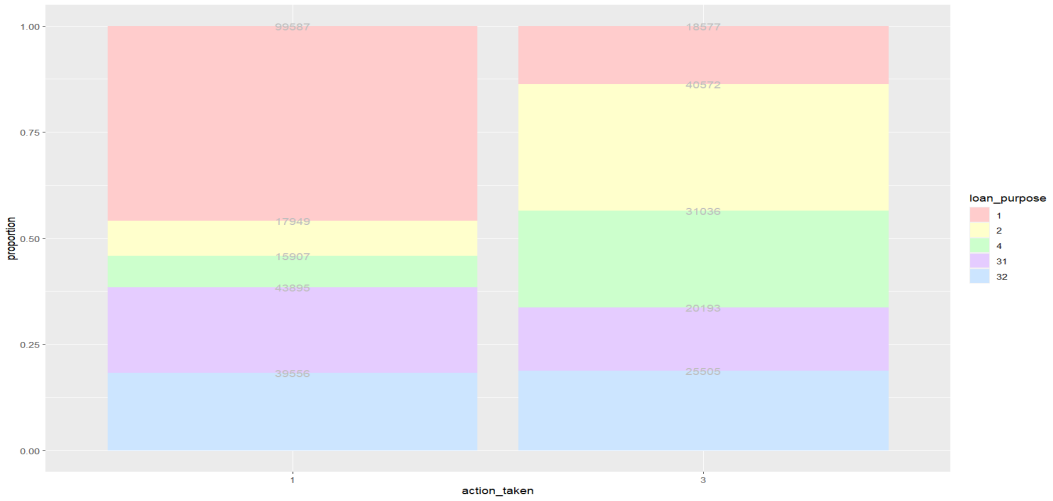


Figure 8: Bar chart Showing the relationship between loan purpose and action_taken. From the bar chart, it can be seen that there is a noticeable difference in the purpose of the loan between loan originated and application denied. In the case of application denied, the proportion of home purchase was the highest, followed by refinancing, while in the case of loan originated, the proportion of home improvement was the highest and the proportion of home purchase was the lowest. Since the proportion of each loan purpose varies noticeably depending on the action taken, this predictor can be a good indicator for predicting action taken variables.

2.2.4 The relationship between Ethnicity of Applicant or Borrower: 1 and Action taken

We used a bar chart to investigate the relationship between ethnicity(between Hispanic or Latino and Not Hispanic or Latino and among Hispanic or Latino) and action taken.

- Loan purpose code descriptions:

1: Hispanic or Latino 11: Mexican 12:Puerto Rican 13:Cuban 14: Other Hispanic or Latino
2: Not Hispanic or Latino 3: Information not provided by applicant in mail, internet or telephone application 4: Not applicable. If the Applicant or Borrower did not select any ethnicity(ies) and only provided ethnicity(ies) in the Ethnicity of Applicant or Borrower: Free Form Text Field for Other Hispanic or Latino, this data field may be left blank.

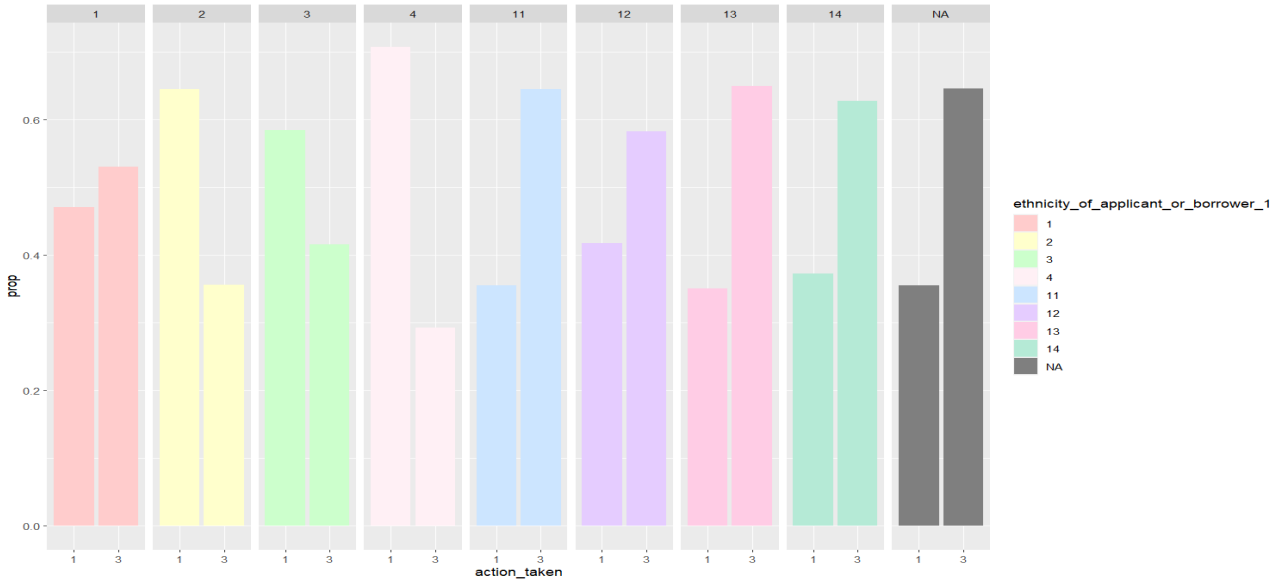


Figure 9: Bar chart Showing the relationship between ethnicity and action_taken. From the bar chart, it can be seen that the proportion of type of action taken between hispanic and non hispanic applicants is noticeably different. Regardless of nationality, Hispanic or Latino had a higher proportion of loan denied, while Non-Hispanic applicants had a higher proportion of loan originated. Therefore Ethnicity of Applicant or Borrower: 1 predictor is likely to be a useful indicator to predict action taken variables.

2.2.5 The relationship between Property value, Loan term and Action taken

We used a box plot to investigate the relationship between property value, loan term and action taken variables. However it was difficult to see the relationship between the three variables because there are outliers that have a large influence on the distribution of the property value variable.

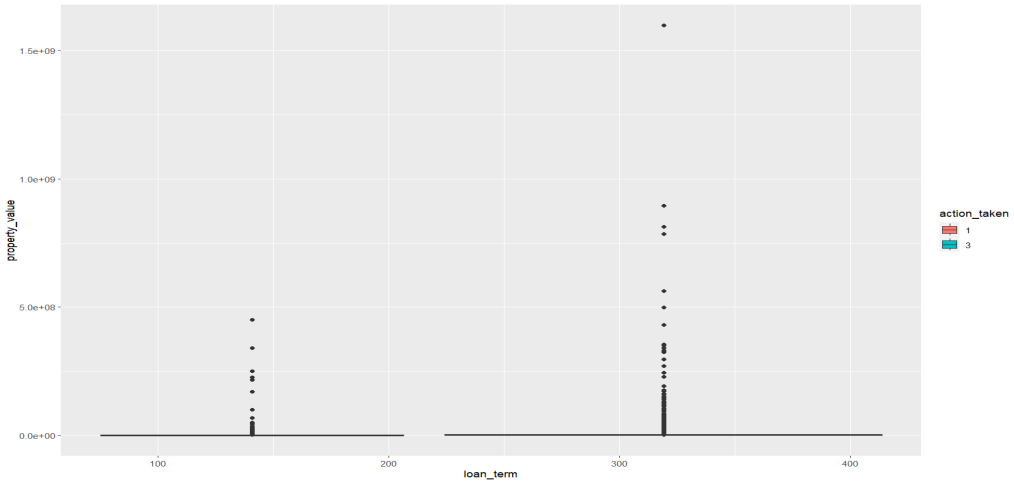


Figure 10: Box plot Showing the relationship between property value, loan term and action_taken. From this plot, it can be identified that outliers with great influence exist in property value variables.

To explore the relationship between property value, loan term and action taken variables, we adjusted the property value variable to 0.05-0.95 quantiles.

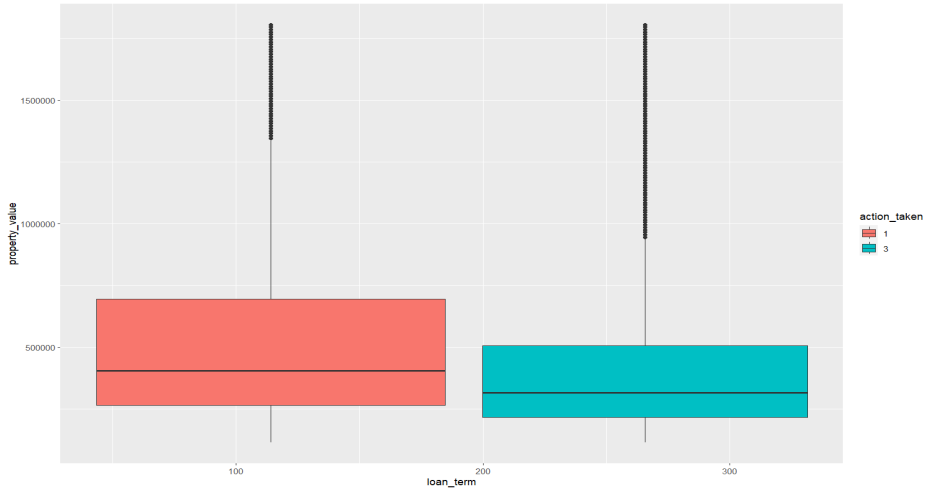


Figure 11: Box plot Showing the relationship between property value(with 0.05-0.95 quantiles) , loan term and action_taken. From this plot, we can see that the average property value of the loan originated cases is higher than the average property value of the application denied cases. Also we can see that there are more loan originated cases when the loan term is shorter than when the loan term is longer. In addition, it can be seen that the average property value is higher when the loan term is short.

2.2.6 The relationship between Interest only payments, Balloon payment and Action taken

We used a bar chart to investigate the relationship between interest only payments, balloon payment and action taken.

- Interest only payments code descriptions:
  1: Interest-only payments 2: No interest-only payments
- Balloon payment code descriptions:
  1: Balloon payment 2: No balloon payment

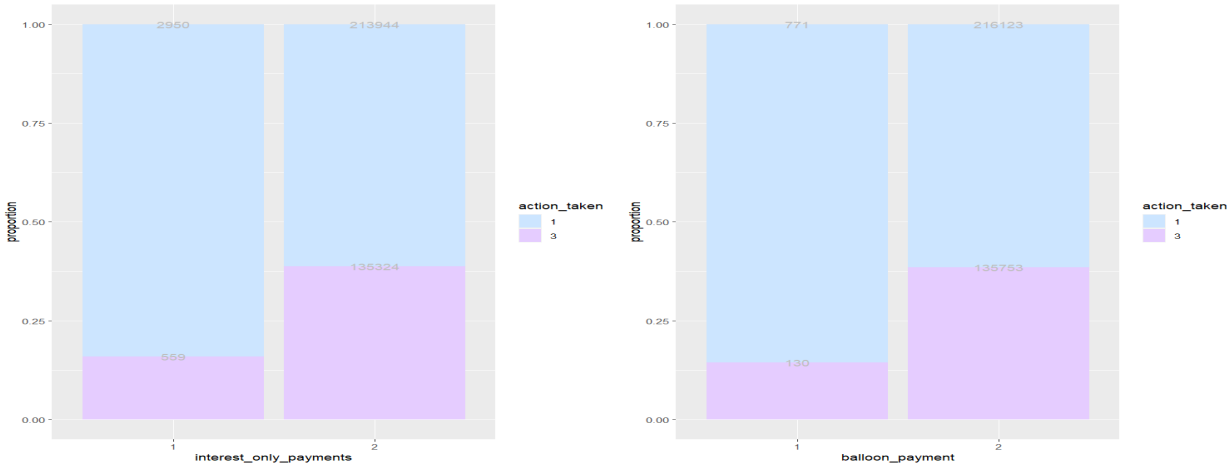


Figure 12: Bar chart Showing the relationship between interest only payments, balloon payment and action taken. From these bar charts, we can see that the relationship between interest only payment and action taken and the relationship between balloon payment and action taken is very similar. In the case of interest only payment variable, the proportion of loan originated is very high when interest only payment, and the proportion of loan originated is still high even when no interest only payment, but there is no difference as high as when interest only payment. Likewise, both balloon payment and no balloon payment have a higher proportion of loan originated than proportion of application denied, but in the case of balloon payment, the proportion of loan originated is much higher.

3. Preprocessing/Recipes

3.1 Recipe 1

We remove unnecessary variables with their reasons as follows:

- Id: each id is assigned just for order and it is irrelevant to the response variable.
- Legal_entity_identifier_lei: it is one unique value that is assigned to every instance.
- Activity year: because we focused on the loan data of Wells Fargo National Bank in 2019
- Preapproval: it is a unique value as it remains the same for all data points (preapproval not requested).
- State: contains NA values that are impossible to impute.
- Ethnicity_of_applicant_or_borrower_2, ethnicity_of_applicant_or_borrower_3, ethnicity_of_applicant_or_borrower_4, ethnicity_of_applicant_or_borrower_5: there are too many NA values that are impossible to impute
- Ethnicity_of_co_applicant_or_borrower_2, ethnicity_of_co_applicant_or_borrower_3, ethnicity_of_co_applicant_or_borrower_4, ethnicity_of_co_applicant_or_borrower_5: there are too many NA values that are impossible to impute

- Race_of_applicant_or_borrower_2, race_of_applicant_or_borrower_3, race_of_applicant_or_borrower_4, race_of_applicant_or_borrower_5, race_of_co_applicant_or_co_borrower_2, race_of_co_applicant_or_co_borrower_3, race_of_co_applicant_or_co_borrower_4, race_of_co_applicant_or_co_borrower_5: there are too many NA values that are impossible to impute
- Total_points_and_fees: contains one unique value NA that are impossible to impute
- Prepayment_penalty_term: contains NA values that are impossible to impute
- Introductory_rate_period: contains NA values that are impossible to impute
- Multifamily_affordable_units: contains only one unique value NA that is impossible to impute
- Automated_underwriting_system_2, automated_underwriting_system_3, automated_underwriting_system_4, automated_underwriting_system_5: contain too many NA values that are impossible to impute.

We converted loan_purpose, ethnicity_of_applicant_or_borrower_1, ethnicity_of_co_applicant_or_co_borrower_1, race_of_applicant_or_borrower_1, and race_of_co_applicant_or_co_borrower_1 into factors.
We replaced the missing values with mode for variables 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, age_of_applicant_62, and age_of_co_applicant_62.
We replaced the missing values with mean value for variables income, combined_loan_to_value_ratio, loan_term, and property_value.

3.2 Recipe 2
We added a step to variables age_of_applicant_or_borrower, negative_amortization, and manufactured_home_secured_property_type, that allows the factor to add new levels instead of converting them into NA values.

In addition to the variables removed in recipe 1, we removed other unnecessary variables with their reasons as follows:

- Reverse_mortgage: it is one unique value that is assigned to every instance. It remains the same for every data point.

Furthermore, we converted the following variables into factors in addition to recipe 1:

- Ethnicity_of_applicant_or_borrower_collected_on_the_basis_of_visual_observation_or_surname
- Ethnicity_of_co_applicant_or_co_borrower_collected_on_the_basis_of_visual_observation_or_surname
- Race_of_applicant_or_borrower_collected_on_the_basis_of_visual_observation_or_surname

- race_of_co_applicant_or_co_borrower_collected_on_the_basis_of_visual_observation_or_surname, sex_of_applicant_or_borrower
- Sex_of_co_applicant_or_co_borrower
- Sex_of_applicant_or_borrower_collected_on_the_basis_of_visual_observation_or_surname
- Sex_of_co_applicant_or_co_borrower_collected_on_the_basis_of_visual_observation_or_surname
- 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_model
- Balloon_payment
- Interest_only_payments
- Negative_amortization
- Other_non_amortizing_features
- Manufactured_home_secured_property_type
- Manufactured_home_land_property_interest
- Total_units
- Submission_of_application
- Initially_payable_to_your_institution
- Automated_underwriting_system_1
- Open_end_line_of_credit
- Business_or_commercial_purpose

3.3 Recipe 3

In addition to Recipe 1, we add step_dummy(all_factor_predictors(), one_hot = TRUE), which convert categorical data into one hot encoding.

4. Candidate models/Model evaluation/Tuning

4.1 Candidate models

To predict the action taken, either loan originated or application denied, we started by building seven models first: Logistic linear regression, Knn, Random forest, Decision tree, Neural network, Xgboost, and Naive Bayes. However when we tried to train our **Neural Network model**, the loss was NaN. This means the gradients explode. The actual gradient value was larger than the maximum value that the computer can store. We can decrease the learning rate or use other optimizers such as Adam or RMSprop. However, since the tidy model does not support this, we decided to omit the Neural Network method from our candidate models. Also

when we tried to train our **Knn model**, we encountered the problem that our laptops are not able to handle such a large data set and it overflew. Even though we tried a small grid size, we were still unable to obtain a result. Thus we decided to omit the Knn model. Therefore, we decided on five models: Logistic linear regression, Random forest, Decision tree, Xgboost, and Naive Bayes as our final candidate models.

4.1.1 Candidate Model Description

- Logistic linear regression
  Logistic linear regression uses the logistic function to predict the probability that an instance belongs to a particular class. It is mostly used in binary classification models. It can model the relationship between one or more independent variables. It can also predict the probability of an outcome that falls between 0 and 1 using a logistic function.

- Random Forest
  Random forest constructs multiple decision trees. It seeks to find the best subset for the data and combines the multiple decision trees into a single result as an output. It can overcome the problem of overfitting and is able to handle large data sets.

- Decision Tree
  Decision tree uses a graph similar to the shape of the tree to make predictions. Each root represents a test, each branch represents the outcome of the test, and each leaf node represents the class. The instance is classified into different subsets as it goes down the tree.

- Xgboost
  Xgboost is an implementation of gradient boosted trees. It incorporates regularization in order to prevent the problem of overfitting. It builds decision trees and each tree aims to correct errors made by previous trees. It is known for its high efficiency and accuracy.

- Naive Bayes
  Naive Bayes is a model based on the Bayes' Theorem. It assumes the variables in the dataset are mutually independent, which is a rare case in the real world and thus called naive bayes model. It is known for its easy implementation and computational efficiency.

4.2 Candidate models with default hyperparameters

We first trained models with the default hyperparameters to compare the performance of each model without tuning. When it comes to recipe selection, all three recipes were applied to

each model and the recipe that showed the best performance for each model was selected. After we tuned the models, we compared the results to get the best model.

4.2.1 Summary of baseline models (without tuning)

	Type of model	Engine	Recipe	Hyperparameters (default)
Candidate model 1	Logistic linear regression	glm	Recipe 2	-
Candidate model 2	Random Forest	ranger	Recipe 1	Trees Min_n
Candidate model 3	Decision tree	rpart	Recipe 2	Mode Engine Cost_complexity Tree_depth Min_n
Candidate model 4	Xgboost	xgboost	Recipe 3	Trees Min_n Tree_depth Learn_rate Loss_reduction Sample_size Stop_iter
Candidate model 5	Naive Bayes	naivebayes	Recipe 2	Smoothness Laplace

Table 1: Summary of baseline models. Different recipes are used depending on the models. All models are trained with default hyperparameters.

4.2.2 Summary of performance of baseline models using V-fold cross-validation

	Metric	Mean Score	Std_Error
Candidate Model 1: Logistic linear regression	accuracy	0.9754633	-
	roc_auc	0.9976347	-
Candidate Model 2: Random Forest	accuracy	0.9841770	-
	roc_auc	0.9990606	-
Candidate Model 3: Decision tree	accuracy	0.9749729	2.296777e-04
	roc_auc	0.9920051	1.875761e-05
Candidate Model 4: Xgboost	accuracy	0.9827370	1.304429e-04
	roc_auc	0.9989795	1.040053e-05

Candidate Model 5: Naive Bayes	accuracy	0.8541969	0.0072336184
	roc_auc	0.9662598	0.0003616065

Table 2: Summary of performance of baseline models. The accuracy and roc_auc scores were measured using a 2-fold cross-validation. From this table, we can see that the Random Forest model shows the best performance in both accuracy and roc_auc, while the Naive Bayes model shows the worst performance in both accuracy and roc_auc. These results are our baselines. We attempt to search for better hyperparameters that outperforms their respective baselines.

4.2.3 Baseline model bar chart comparison

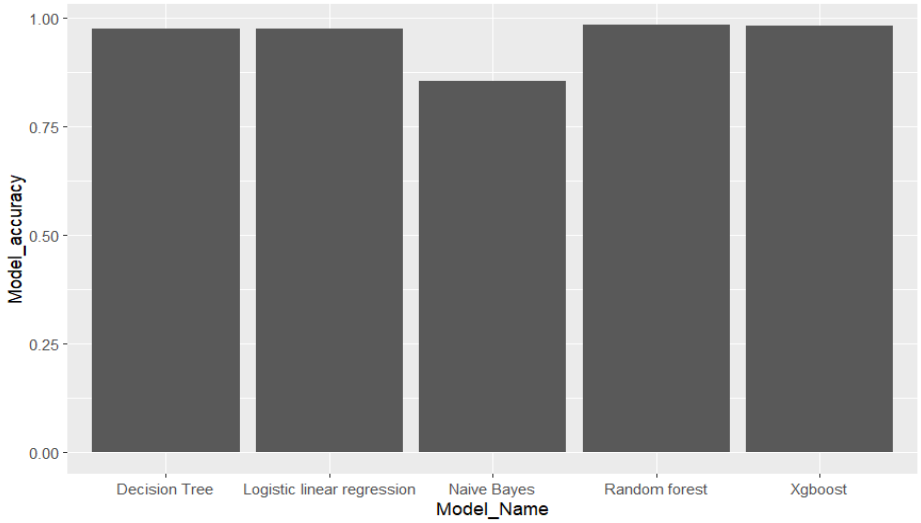


Figure 1: Bar chart Comparing the performance of baseline models using accuracy. We can see that the Random forest model has the highest accuracy score(highest performance) and the Naive Bayes model has the lowest accuracy score(lowest performance.)

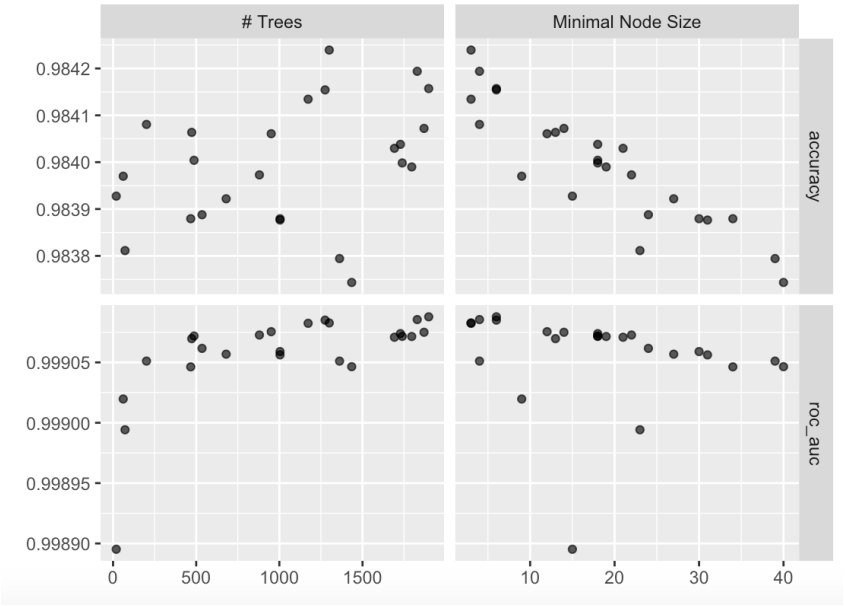
4.3 Candidate models with tuning

We tuned each candidate model to find the hyperparameters that best predict the action taken on the loan application. Each model had the same recipe as their baseline model, and the performance of the random forest model was measured using 2 fold cross validation and the rest of the models were measured using 5 fold cross validation.

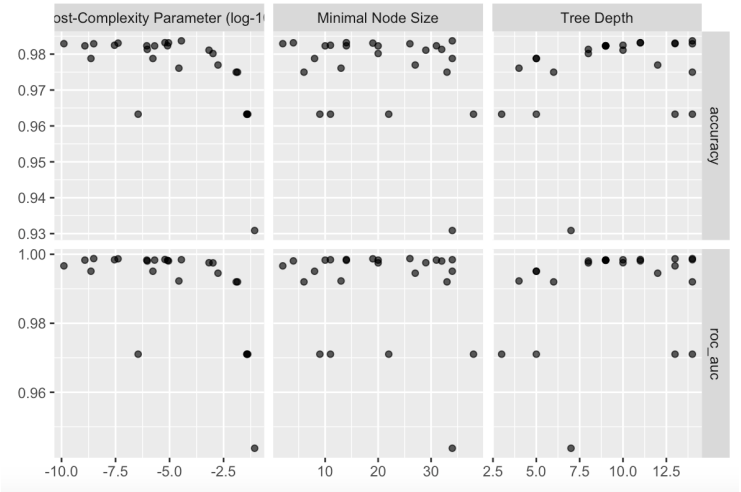
4.3.1 Tuning of the hyperparameters for candidate models

- Linear logistic model
  Since the Linear logistic model does not have hyperparameters that can be tuned, we did not tune this model.

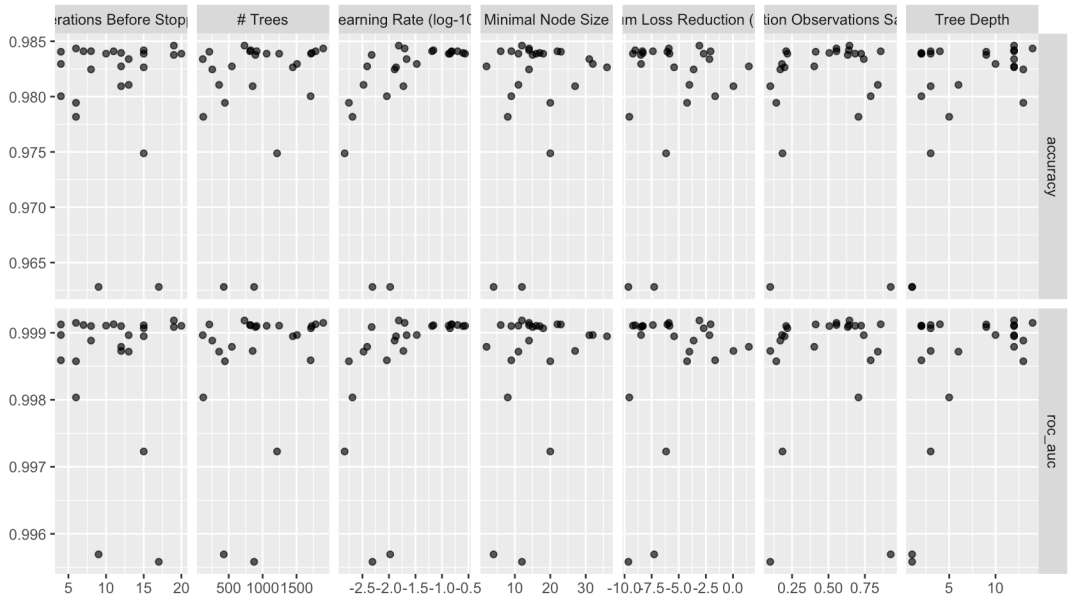
- Random Forest
  We tuned the trees and the min_n hyperparameters with grid size 10 and used the default value for the mtry hyperparameter. After tuning, this model showed best performance with an accuracy score: 0.984 when trees is 1300 and min_n is 3 and showed best performance with a roc_auc score: 0.999 when trees is 1898 and min_n is 6.



- Decision tree
  We tuned the cost complexity, tree depth, and the min_n hyperparameters with grid size 25. After tuning, this model showed best performance with accuracy score: 0.984 when cost complexity is 3.53e-5, tree depth is 14 and min n is 34 and showed best performance with a roc_auc score: 0.999 when cost complexity is 3.09e-9, tree depth is 14, and min n is 26.



- Xgboost
  We tuned 7 hyperparameters of the Xgboost model: trees, min_n, tree_depth, learn rate, loss_reduction, sample_size, and Stop_iter and used the default value for mtry hyperparameter. After tuning, this model showed best performance (accuracy: 0.985, roc_auc: 0.999) when trees is 732, min_n is 12, tree_depth is 12, learn rate is 0.01531878, loss_reduction is 8.004341e-04, sample_size is 0.6441571, and Stop_iter is 19.



- Naive Bayes
  We tuned the smoothness and laplace hyperparameters with grid size 25. After tuning, this model showed best performance (accuracy: 0.834, roc_auc: 0.949) when smoothness is 1.19 and laplace is 2.18.

4.3.2 Summary of Tuned Models

	Type of model	Engine	Recipe	Hyperparameters	
Candidate model 1	Logistic linear regression	glm	Recipe 2	-	
Candidate model 2	Random Forest	ranger	Recipe 1	accuracy	Trees = 1300 Min_n = 3
				roc_auc	Trees = 1898 Min_n = 6
Candidate model 3	Decision tree	rpart	Recipe 2	accuracy	Cost_complexity = 3.53e-5 Tree_depth =14 Min_n = 34
				roc_auc	Cost_complexity = 3.09e-09 Tree_depth =14 Min_n = 26
Candidate model 4	Xgboost	xgboost	Recipe 3	Trees = 732 Min_n = 12 Tree_depth = 12 Learn_rate = 0.01531878 Loss_reduction = 8.004341e-04 Sample_size = 0.6441571 Stop_iter = 19	
Candidate model 5	Naive Bayes	naivebayes	Recipe 2	Smoothness = 1.19 Laplace = 2.18	

Table3: Summary of tuned models. Different recipes are used depending on the models and tuned hyperparameters were applied to all models.

4.3.3 Summary of best performance of tuned models using V-fold cross-validation

	Cross-validation	Metric	Mean Score	Std_Error
Candidate Model 1: Logistic linear regression	-	accuracy	-	-
		roc_auc	-	-
Candidate Model 2: Random forest	2-fold	accuracy	0.984	6.81e-5
		roc_auc	0.999	1.56e-5
Candidate Model 3: Decision tree	5-fold	accuracy	0.984	1.81e-4
		roc_auc	0.999	2.95e-5
Candidate Model 4: Xgboost	5-fold	accuracy	0.985	4.53e-5
		roc_auc	0.999	1.62e-5
Candidate Model 5: Naive Bayes	5-fold	accuracy	0.834	0.0227
		roc_auc	0.949	2.50e-4

Table 4: Summary of performance of tuned models.

4.3.4 Bar chart: Performance comparison of Tuned models

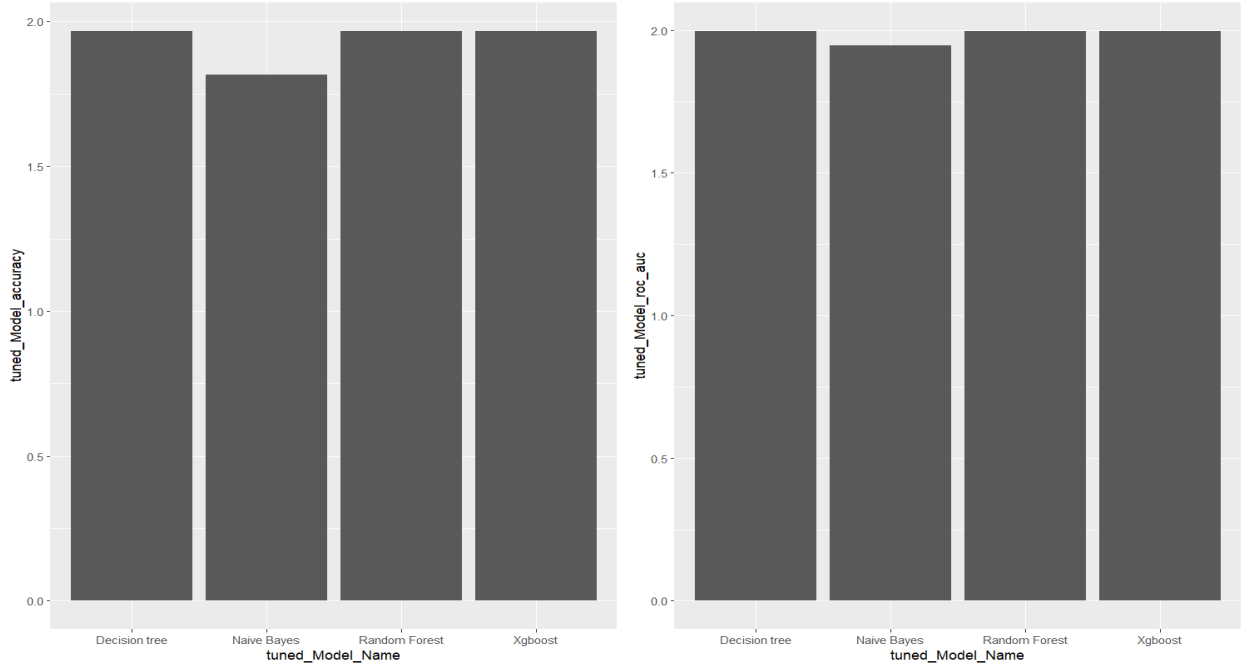


Figure 2: Bar chart Comparing the performance of baseline models using accuracy and roc_auc.
We can see that the Xgboost model has the highest accuracy score(highest performance) and the Naive Bayes model has the lowest accuracy score(lowest performance.) Also we can see that all models, except the Naive Bayes model, show a pretty good roc_auc score.

4.3.5 Performance Comparison of Baseline and Tuned Models

	Type	Metric	Mean Score	Standard error
Candidate Model 1: Logistic linear regression	Baseline	accuracy	0.9754633	-
		roc_auc	0.9976347	-
	Tuned	accuracy	-	-
		roc_auc	-	-
Candidate Model 2: Random forest	Baseline	accuracy	0.9841770	-
		roc_auc	0.9990606	-
	Tuned	accuracy	0.984	6.81e-5
		roc_auc	0.999	1.56e-5
Candidate Model 3: Decision tree	Baseline	accuracy	0.9749729	2.296777e-04
		roc_auc	0.9920051	1.875761e-05
	Tuned	accuracy	0.984	1.81e-4
		roc_auc	0.999	2.95e-5
Candidate Model 4: Xgboost	Baseline	accuracy	0.9827370	1.304429e-04
		roc_auc	0.9989795	1.040053e-05
	Tuned	accuracy	0.985	4.53e-5
		roc_auc	0.999	1.62e-5
Candidate Model 5: Naive Bayes	Baseline	accuracy	0.8541969	0.0072336184
		roc_auc	0.9662598	0.0003616065
	Tuned	accuracy	0.834	0.0227
		roc_auc	0.949	2.50e-4

Table 5: Performance Comparison of Baseline and Tuned Models. Except for the Naive Bayes model, the tuned model shows the same or better performance than the baseline in all models.

5. Discussion of Final Model

5.1 Conclusion

Based on the performance comparison of the tuned models table, we can see that Candidate Model 4, which is Xgboost after tuning, produces the highest accuracy of 0.985 and roc_auc of 0.999 with lowest std_error.

Strengths:

1. There are a lot of NA values in the dataset and Xgboost can handle NA values. Then we can apply the Xgboost model to unprocessed data.
2. The Xgboost model tends to outperform other models in most cases.

Weaknesses:

1. Compared to other models, Xgboost takes longer to tune and run. Sometimes it is impossible to do parameter tuning with a large grid size.
2. Xgboost has lower interpretability compared to other models such as linear models or decision tree models. The Xgboost model is usually more complex and less interpretable.

5.2 Improvements

Since it took a long time for our laptops to run the code, we did not apply large grid sizes or tune a lot of hyperparameters for the models. In order to shorten the time that it takes to run the code, we decrease the size of cross validation, which might lead to biased results. Also, we know that Neural Network model is an excellent tool for classification, However, due to the limitation to tidy models, we cannot change the optimizers or add layers. Sigmoid activation function does not exist, which is essential for binary classification. In the future, we can try using tensorflow or pytorch to build a deep neural network.

~~6. Script Verification Video Link~~

~~https://drive.google.com/file/d/1haKNbc7z_OYHvVHlr4gy5tsCtAqEftXf/view?usp=sharing~~

7. Appendix (Team Member Contribution)

Team Member Name	Contribution
Phillips (Ruoyan) Li	Designed code for the baseline model and tuned model. Designed all the recipes. Ran the hyperparameter tuning code for algorithms.
Vincent (Haojie) Liu	Designed code for the baseline model and tuned model. Ran the hyperparameter tuning code for algorithms.
Cyril (Yizhuo) Chang	Contributed in running hyperparameter tuning code. Annotated the code. Mascot of the team
Jinyoung Hwang	Contributed in writing the report. Ran the hyperparameter tuning code for algorithms. Annotated the code. Generated data visualization plots in the report.
Kristy (Yingyue) Lai	Contributed in writing the report. Ran the hyperparameter tuning code for algorithms. Coordinated with team members to combine scripts and results into a systematic report.

Import library

```
library(cowplot)
library(dplyr)
library(tidymodels)
library(tidyverse)
library(rpart)
library(xgboost)
library(glmnet)
library(LiblineaR)
library(kernlab)
library(kknn)
```

Load Data

```
set.seed(777777)

test <- read.csv("test2.csv")
train <- read.csv("train2.csv")

# convert action_taken variable to factor
cleaned_train <-
  train %>%
  mutate(action_taken = as.factor(action_taken))
```

Relationship between sex and age

```
#select variables and convert the variables to factor
age_sex <- cleaned_train %>%
  select(sex_of_applicant_or_borrower, age_of_applicant_or_borrower,action_taken) %>%
  mutate(sex_of_applicant_or_borrower = factor(sex_of_applicant_or_borrower))%>%
  mutate(age_of_applicant_or_borrower= factor(age_of_applicant_or_borrower,
                                              levels=c('<25','25-34','35-44','45-54','65-74','>74')))

#make table
age_sex_dat <- data.frame(table(age_sex$age_of_applicant_or_borrower,age_sex$sex_of_applicant_or_borr
#set the name for each column
names(age_sex_dat) <- c("age_group","gender","Count")

#create bar chartShowing cross-analysis of age group and gender
ggplot(age_sex_dat, aes(x=age_group, y=Count, fill=gender)) +
  geom_bar(stat="identity")+
  scale_fill_manual(values=c("#E5CCFF","#FFF0F5","#CCE5FF","#FFCCCC","#FFFFCC"))
```

```
# calculate proportion
toPlot<-age_sex%>%
  group_by(sex_of_applicant_or_borrower,age_of_applicant_or_borrower)%>%
  summarise(n = n())%>%
  group_by(age_of_applicant_or_borrower)%>%
  mutate(prop = n/sum(n))

# create Bar chart Showing cross-analysis of the proportions of each age group and gender
ggplot(data = toPlot, aes(sex_of_applicant_or_borrower, prop, fill = age_of_applicant_or_borrower)) +
```

```
  geom_col() +
  facet_grid(~age_of_applicant_or_borrower)+
  scale_fill_manual(values=c("#FFCCCC","#FFFFCC","#CCFFCC","#FFF0F5","#CCE5FF","#E5CCFF","#FFCCE5"))
```

Relationship between loan type and loan amount

```
type_amount <- cleaned_train %>%
  #select loan_type and loan_amount
  select(loan_type,loan_amount) %>%
  #convert loan_type variable to factor
  mutate(loan_type=factor(loan_type))

#create boxplot
ggplot(type_amount, aes(x=loan_type, y=loan_amount))+
  geom_boxplot()
```

```
#create boxplot with 0.05-0.95 quantile of loan_amount
ggplot(type_amount, aes(x=loan_type, y=loan_amount,fill=loan_type))+
  geom_boxplot(alpha=0.3) +
  theme(legend.position="none") +
  scale_fill_brewer(palette="Dark2")+
  scale_y_continuous(limits = quantile(type_amount$loan_amount, c(0.05, 0.95)))
```

Relationship between action_taken and income

```
#select action_taken, income variable
income_action <- cleaned_train %>%
  select(action_taken, income) %>%
  #drop NA value
  drop_na(income)

#create boxplot
ggplot(income_action , aes(x=action_taken, y=income))+
  geom_boxplot()
```

```
# create boxplot with 0.05-0.95 Quentile of income variable
ggplot(income_action , aes(x=action_taken, y=income))+
  geom_boxplot(fill="orange") +
  scale_y_continuous(limits = quantile(income_action $income, c(0.05, 0.95)))
```

Relationship between loan amount and action taken

```
# create boxplot with 0.05-0.95 Quentile of loan amount variable
ggplot(income_action , aes(x=action_taken, y=income))+
  geom_boxplot(fill="lightblue") +
  scale_y_continuous(limits = quantile(income_action $income, c(0.05, 0.95)))
```

Relationship btw loan purpose and action taken

```
#select variables and convert the variables to factor
lone_purpose_action <- cleaned_train %>%
  select(loan_purpose, action_taken)%>%
```

```
  mutate(loan_purpose= as.factor(loan_purpose))

#create the proportion bar chart
ggplot(data = lone_purpose_action, aes(x = action_taken, fill = loan_purpose)) +
  geom_bar(position = "fill") + ylab("proportion") +
  stat_count(geom = "text",
             aes(label = stat(count)),
             position="fill", colour="gray")+
  scale_fill_manual(values=c("#FFCCCC","#FFFFCC","#CCFFCC","#E5CCFF","#CCE5FF"))
```

Relationship between ethnicity_of_applicant_or_borrower_1 and action taken

```
#select variables and convert the variables to factor
ethnicity_action <-
cleaned_train %>%
  select(ethnicity_of_applicant_or_borrower_1, action_taken)%>%
  mutate(ethnicity_of_applicant_or_borrower_1= as.factor(ethnicity_of_applicant_or_borrower_1))

# calculate proportion
toPlot1<-ethnicity_action%>%
  group_by(action_taken,ethnicity_of_applicant_or_borrower_1)%>%
  summarise(n = n())%>%
  group_by(ethnicity_of_applicant_or_borrower_1)%>%
  mutate(prop = n/sum(n))

# create plot
ggplot(data = toPlot1, aes(action_taken, prop, fill = ethnicity_of_applicant_or_borrower_1)) +
  geom_col() +
  facet_grid(~ethnicity_of_applicant_or_borrower_1)+
  #scale_fill_brewer(palette="Pastel1")
  scale_fill_manual(values=c("#FFCCCC","#FFFFCC","#CCFFCC","#FFF0F5","#CCE5FF",
                             "#E5CCFF","#FFCCE5","#B5EAD6","#C7DBDA","#FFB8B1"))
```

Relationship between property_value and action taken

```
#create box plot
ggplot(data = cleaned_train, aes(x = loan_term, y = property_value, fill = action_taken)) +
  geom_boxplot()
```

```
#create box plot with 0.05-0.95 quantile of property variable
ggplot(data = cleaned_train, aes(x = loan_term, y = property_value, fill = action_taken)) +
  geom_boxplot()+
  scale_y_continuous(limits = quantile(cleaned_train$property_value, c(0.05, 0.95), na.rm=TRUE))
```

Relationship between Interest only payments, Balloon payment and Action taken

```
# select interest_only_payments, balloon_payment,action_taken variables and
#convert variables to factor
interest_balloon_action <- cleaned_train %>%
  select(interest_only_payments, balloon_payment,action_taken)%>%
  mutate(interest_only_payments= as.factor(interest_only_payments))%>%
  mutate(balloon_payment= as.factor(balloon_payment))
```

```r
#create bar chart between interest_only_payments and  action_taken
p1 <- ggplot(data = interest_balloon_action, aes(x = interest_only_payments, fill = action_taken)) +
  geom_bar(position = "fill") + ylab("proportion") +
  stat_count(geom = "text",
             aes(label = stat(count)),
             position="fill", colour="gray")+
  scale_fill_manual(values=c("#CCE5FF","#E5CCFF"))

#create bar chart between balloon_payment and  action_taken
p2 <- ggplot(data = interest_balloon_action, aes(x = balloon_payment, fill = action_taken)) +
  geom_bar(position = "fill") + ylab("proportion") +
  stat_count(geom = "text",
             aes(label = stat(count)),
             position="fill", colour="gray")+
  scale_fill_manual(values=c("#CCE5FF","#E5CCFF"))

# two bar charts
plot_grid(p1, p2)
```

Import Library

```
library(dplyr)
library(tidymodels)
library(tidyverse)
library(rpart)
library(xgboost)
library(glmnet)
library(LiblineaR)
library(kernlab)
library(kknn)
library(agua)
library(naivebayes)
library(discrim)
```

Load Data

```
set.seed(666666)

test <- read.csv("test2.csv")
train <- read.csv("train2.csv")

cleaned_train <- train %>%
  mutate(action_taken = factor(action_taken))

train_folds <- vfold_cv(cleaned_train, v=2, strata = 'action_taken')
```

Create Recipe 1

```
recipe1 <- recipe(action_taken ~ ., data = cleaned_train) %>%
  # step_novel(age_of_applicant_or_borrower) %>%

  step_rm(id) %>%

  step_rm(legal_entity_identifier_lei) %>%

  # unique(test$activity_year)  unique(train$activity_year) 2018
  step_rm(activity_year) %>%

  step_mutate(loan_purpose = factor(loan_purpose, levels = c("1", "2", "31", "32", "4", "5"))) %>%

  # unique(train$preapproval) unique(test$preapproval)  2
  step_rm(preapproval) %>%

  # contain NA value impossible to impute
  step_rm(state) %>%

  step_mutate(ethnicity_of_applicant_or_borrower_1 = factor(ethnicity_of_applicant_or_borrower_1, lev

  step_mutate(ethnicity_of_co_applicant_or_co_borrower_1 = factor(ethnicity_of_co_applicant_or_co_bor
```

```
  # impute nominal variable by most common values
  step_impute_mode(ethnicity_of_applicant_or_borrower_1, ethnicity_of_co_applicant_or_co_borrower_1)

  step_rm(ethnicity_of_applicant_or_borrower_2) %>%
  step_rm(ethnicity_of_applicant_or_borrower_3) %>%
  step_rm(ethnicity_of_applicant_or_borrower_4) %>%
  step_rm(ethnicity_of_applicant_or_borrower_5) %>%
  step_rm(ethnicity_of_co_applicant_or_co_borrower_2) %>%
  step_rm(ethnicity_of_co_applicant_or_co_borrower_3) %>%
  step_rm(ethnicity_of_co_applicant_or_co_borrower_4) %>%
  step_rm(ethnicity_of_co_applicant_or_co_borrower_5) %>%

  step_mutate(race_of_applicant_or_borrower_1 = factor(race_of_applicant_or_borrower_1, levels = c("1

  step_mutate(race_of_co_applicant_or_co_borrower_1 = factor(race_of_co_applicant_or_co_borrower_1, l

  step_impute_mode(race_of_applicant_or_borrower_1, race_of_co_applicant_or_co_borrower_1) %>%

  step_rm(race_of_applicant_or_borrower_2, race_of_applicant_or_borrower_3,
          race_of_applicant_or_borrower_4, race_of_applicant_or_borrower_5,
          race_of_co_applicant_or_co_borrower_2, race_of_co_applicant_or_co_borrower_3,
          race_of_co_applicant_or_co_borrower_4, race_of_co_applicant_or_co_borrower_5) %>%

  step_impute_mode(age_of_applicant_62, age_of_co_applicant_62) %>%

  step_impute_mean(income) %>%

  step_rm(total_points_and_fees, prepayment_penalty_term) %>%

  step_impute_mean(combined_loan_to_value_ratio, loan_term) %>%

  step_rm(introductory_rate_period) %>%

  step_impute_mean(property_value) %>%

  step_rm(multifamily_affordable_units) %>%

  # too many NA values
  step_rm(automated_underwriting_system_2, automated_underwriting_system_3,
          automated_underwriting_system_4, automated_underwriting_system_5)
```

Create Recipe 2

```
recipe2 <- recipe(action_taken ~ ., data = train) %>%
  step_novel(age_of_applicant_or_borrower) %>%

  step_rm(id) %>%

  step_rm(legal_entity_identifier_lei) %>%

  # unique(test$activity_year)  unique(train$activity_year) 2018
  step_rm(activity_year) %>%
```

```
step_mutate(loan_type = factor(loan_type, levels = c("1", "2", "3", "4"))) %>%

step_mutate(loan_purpose = factor(loan_purpose, levels = c("1", "2", "31", "32", "4", "5"))) %>%

# unique(train$preapproval) unique(test$preapproval)  2
step_rm(preapproval) %>%

step_mutate(construction_method = factor(construction_method, levels = c("1", "2")),
            occupancy_type = factor(occupancy_type, levels = c("1", "2", "3"))) %>%

# contain NA value impossible to impute
step_rm(state) %>%

step_mutate(ethnicity_of_applicant_or_borrower_1 = factor(ethnicity_of_applicant_or_borrower_1, lev

step_mutate(ethnicity_of_co_applicant_or_co_borrower_1 = factor(ethnicity_of_co_applicant_or_co_bor

# impute nominal variable by most common values
step_impute_mode(ethnicity_of_applicant_or_borrower_1, ethnicity_of_co_applicant_or_co_borrower_1)

step_rm(ethnicity_of_applicant_or_borrower_2) %>%
step_rm(ethnicity_of_applicant_or_borrower_3) %>%
step_rm(ethnicity_of_applicant_or_borrower_4) %>%
step_rm(ethnicity_of_applicant_or_borrower_5) %>%
step_rm(ethnicity_of_co_applicant_or_co_borrower_2) %>%
step_rm(ethnicity_of_co_applicant_or_co_borrower_3) %>%
step_rm(ethnicity_of_co_applicant_or_co_borrower_4) %>%
step_rm(ethnicity_of_co_applicant_or_co_borrower_5) %>%

step_mutate(ethnicity_of_applicant_or_borrower_collected_on_the_basis_of_visual_observation_or_surn

step_mutate(ethnicity_of_co_applicant_or_co_borrower_collected_on_the_basis_of_visual_observation_o

step_mutate(race_of_applicant_or_borrower_1 = factor(race_of_applicant_or_borrower_1, levels = c("1

step_mutate(race_of_co_applicant_or_co_borrower_1 = factor(race_of_co_applicant_or_co_borrower_1, l

step_impute_mode(race_of_applicant_or_borrower_1, race_of_co_applicant_or_co_borrower_1) %>%

step_rm(race_of_applicant_or_borrower_2, race_of_applicant_or_borrower_3,
        race_of_applicant_or_borrower_4, race_of_applicant_or_borrower_5,
        race_of_co_applicant_or_co_borrower_2, race_of_co_applicant_or_co_borrower_3,
        race_of_co_applicant_or_co_borrower_4, race_of_co_applicant_or_co_borrower_5) %>%

step_mutate(race_of_applicant_or_borrower_collected_on_the_basis_of_visual_observation_or_surname =

step_mutate(race_of_co_applicant_or_co_borrower_collected_on_the_basis_of_visual_observation_or_sur

step_mutate(sex_of_applicant_or_borrower = factor(sex_of_applicant_or_borrower, levels = c("1", "2"

step_mutate(sex_of_co_applicant_or_co_borrower = factor(sex_of_co_applicant_or_co_borrower, levels

step_mutate(sex_of_applicant_or_borrower_collected_on_the_basis_of_visual_observation_or_surname =
```

```
step_mutate(sex_of_co_applicant_or_co_borrower_collected_on_the_basis_of_visual_observation_or_surn

step_impute_mode(age_of_applicant_62, age_of_co_applicant_62) %>%

step_impute_mean(income) %>%

step_mutate(hoepa_status = factor(hoepa_status, levels = c("1", "2", "3"))) %>%

step_mutate(lien_status = factor(lien_status, levels = c("1", "2"))) %>%

step_mutate(applicant_or_borrower_name_and_version_of_credit_scoring_model = factor(applicant_or_bo

step_mutate(co_applicant_or_co_borrower_name_and_version_of_credit_scoring_model = factor(co_applic

step_rm(total_points_and_fees, prepayment_penalty_term) %>%

step_impute_mean(combined_loan_to_value_ratio, loan_term) %>%

step_rm(introductory_rate_period) %>%

step_mutate(balloon_payment = factor(balloon_payment),
            interest_only_payments = factor(interest_only_payments),
            negative_amortization = factor(negative_amortization),
            other_non_amortizing_features = factor(other_non_amortizing_features)) %>%

step_impute_mean(property_value) %>%

step_mutate(manufactured_home_secured_property_type = factor(manufactured_home_secured_property_typ
            manufactured_home_land_property_interest = factor(manufactured_home_land_property_inter

step_mutate(total_units = factor(total_units)) %>%

step_rm(multifamily_affordable_units) %>%

step_mutate(submission_of_application = factor(submission_of_application),
            initially_payable_to_your_institution = factor(initially_payable_to_your_institution),
            automated_underwriting_system_1 = factor(automated_underwriting_system_1)) %>%

# too many NA values
step_rm(automated_underwriting_system_2, automated_underwriting_system_3,
        automated_underwriting_system_4, automated_underwriting_system_5) %>%

# unique(train$reverse_mortgage) unique(test$reverse_mortgage)  2
step_rm(reverse_mortgage) %>%

step_mutate(open_end_line_of_credit = factor(open_end_line_of_credit),
            business_or_commercial_purpose = factor(business_or_commercial_purpose)) %>%

step_novel(negative_amortization, manufactured_home_secured_property_type)
```

Create Recipe 3

```
recipe3 <- recipe(action_taken ~ ., data = cleaned_train) %>%
  # step_novel(age_of_applicant_or_borrower) %>%

  step_rm(id) %>%

  step_rm(legal_entity_identifier_lei) %>%

  # unique(test$activity_year)  unique(train$activity_year) 2018
  step_rm(activity_year) %>%

  # unique(train$preapproval) unique(test$preapproval)  2
  step_rm(preapproval) %>%

  # contain NA value impossible to impute
  step_rm(state) %>%

  # impute nominal variable by most common values
  # step_impute_mode(ethnicity_of_applicant_or_borrower_1, ethnicity_of_co_applicant_or_co_borrower_1

  step_rm(ethnicity_of_applicant_or_borrower_2) %>%
  step_rm(ethnicity_of_applicant_or_borrower_3) %>%
  step_rm(ethnicity_of_applicant_or_borrower_4) %>%
  step_rm(ethnicity_of_applicant_or_borrower_5) %>%
  step_rm(ethnicity_of_co_applicant_or_co_borrower_2) %>%
  step_rm(ethnicity_of_co_applicant_or_co_borrower_3) %>%
  step_rm(ethnicity_of_co_applicant_or_co_borrower_4) %>%
  step_rm(ethnicity_of_co_applicant_or_co_borrower_5) %>%
  # step_impute_mode(race_of_applicant_or_borrower_1, race_of_co_applicant_or_co_borrower_1) %>%

  step_rm(race_of_applicant_or_borrower_2, race_of_applicant_or_borrower_3,
          race_of_applicant_or_borrower_4, race_of_applicant_or_borrower_5,
          race_of_co_applicant_or_co_borrower_2, race_of_co_applicant_or_co_borrower_3,
          race_of_co_applicant_or_co_borrower_4, race_of_co_applicant_or_co_borrower_5) %>%

  step_impute_mode(age_of_applicant_62, age_of_co_applicant_62) %>%

  step_impute_mean(income) %>%

  step_rm(total_points_and_fees, prepayment_penalty_term) %>%

  step_impute_mean(combined_loan_to_value_ratio, loan_term) %>%

  step_rm(introductory_rate_period) %>%

  step_impute_mean(property_value) %>%

  step_rm(multifamily_affordable_units) %>%

  # too many NA values
  step_rm(automated_underwriting_system_2, automated_underwriting_system_3,
          automated_underwriting_system_4, automated_underwriting_system_5) %>%
```

```
  step_dummy(all_factor_predictors(), one_hot = TRUE)
```

logistic linear regression no tuning

```
# create model
logistic_model<- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

#create workflow
logistic_workflow <- workflow() %>%
  add_model(logistic_model) %>%
  add_recipe(recipe2, blueprint = hardhat::default_recipe_blueprint(allow_novel_levels = TRUE))

#create cross-validation
logistic_crossval_fit <- logistic_workflow %>%
  fit_resamples(resamples = train_folds)

logistic_accuracy = logistic_crossval_fit %>% collect_metrics() %>% filter(.metric == "accuracy")
logistic_crossval_fit %>% collect_metrics()
```

knn baseline (knn tune model - time inefficient, will not consider)

```
# create model
knn_model<- nearest_neighbor() %>%
  set_engine("kknn") %>%
  set_mode("classification")

#create workflow
knn_workflow <- workflow() %>%
  add_model(knn_model) %>%
  add_recipe(recipe, blueprint = hardhat::default_recipe_blueprint(allow_novel_levels = TRUE))

#create cross-validation
knn_crossval_fit <- knn_workflow %>%
  fit_resamples(resamples = train_folds)

knn_rmse = knn_crossval_fit %>% collect_metrics() %>% filter(.metric == "rmse")
knn_crossval_fit %>% collect_metrics()
```

```
# create model
knn_model<- nearest_neighbor(
  neighbors = tune(),
  weight_func = tune(),
  dist_power = tune()
) %>%
  set_engine("kknn") %>%
  set_mode("classification")

# create workflow
knn_workflow <- workflow() %>%
```

```
  add_model(knn_model) %>%
  add_recipe(recipe)

model_param = extract_parameter_set_dials(knn_model)

knn_tune <- knn_workflow %>%
  tune_grid(train_folds,
            grid = model_param %>% grid_random(size = 5)
  )

knn_tune %>%
  show_best()
```

random forest

```
# create model
rf_model<- rand_forest() %>%
  set_engine("ranger") %>%
  set_mode("classification")

# create workflow
rf_workflow <- workflow() %>%
  add_model(rf_model) %>%
  add_recipe(recipe1, blueprint = hardhat::default_recipe_blueprint(allow_novel_levels = TRUE))

# create cross validation
rf_crossval_fit <- rf_workflow %>%
  fit_resamples(resamples = train_folds)

rf_accuracy = rf_crossval_fit %>% collect_metrics() %>% filter(.metric == "accuracy")

rf_crossval_fit %>% collect_metrics()
show_notes(.Last.tune.result)
```

random forest tune

```
rf_model<- rand_forest(
  trees = tune(),
  min_n = tune()) %>%
  set_engine("ranger") %>%
  set_mode("classification")

rf_workflow <- workflow() %>%
  add_model(rf_model) %>%
  add_recipe(recipe, blueprint = hardhat::default_recipe_blueprint(allow_novel_levels = TRUE))

model_param = extract_parameter_set_dials(rf_workflow)

train_folds <- vfold_cv(train, v=2, strata = 'action_taken')

# set up the tuned grid
```

```
rf_tune <- rf_workflow %>%
  tune_grid(train_folds,
            grid = model_param %>% grid_random(size = 10))

# generate plot and show best models
autoplot(rf_tune)

show_best(rf_tune)

# save tuned results as rds file for future use
saveRDS(rf_tune, "rf_tune")
```

decsion tree baseline

```
# Create a decision tree classifier model
dec_tr_model <- decision_tree() %>%
  set_engine("rpart") %>%
  set_mode("classification")

# Create a workflow object to define the modeling workflow
dec_tr_workflow <- workflow() %>%
  add_model(dec_tr_model) %>%
  add_recipe(recipe2)

# Fit the decision tree model on the training set and perform cross-validation
dec_tr_crossval_fit <- dec_tr_workflow %>%
  fit_resamples(resamples = train_folds)

# Extract the accuracy metric from the cross-validation results
dec_tr_accuracy <- dec_tr_crossval_fit %>%
  collect_metrics() %>%
  filter(.metric == "accuracy")

# Print the collected metrics
dec_tr_crossval_fit %>%
  collect_metrics()
```

decision tree tune

```
dec_tr_model<- decision_tree(
  cost_complexity = tune(),
  tree_depth = tune(),
  min_n = tune()) %>%
  set_engine("rpart") %>%
  set_mode("classification")

dec_tr_workflow <- workflow() %>%
  add_model(dec_tr_model) %>%
  add_recipe(recipe)

model_param = extract_parameter_set_dials(dec_tr_model)
```

```
dec_tr_tune <- dec_tr_workflow %>%
  tune_grid(train_folds,
            grid = model_param %>% grid_random(size = 25)
  )

dec_tr_tune %>%
  show_best()
```

xgboost baseline

```
# Load the required package
library(xgboost)

# Set the random seed for reproducibility
set.seed(666666)

# Create a boost tree classifier model
boost_tree_model <- boost_tree() %>%
  set_engine("xgboost") %>%
  set_mode("classification")

# Create a workflow object to define the modeling workflow
boost_tree_workflow <- workflow() %>%
  add_model(boost_tree_model) %>%
  add_recipe(recipe3, blueprint = hardhat::default_recipe_blueprint(allow_novel_levels = TRUE))

# Fit the boost tree model on the training set and perform cross-validation
boost_crossval_fit <- boost_tree_workflow %>%
  fit_resamples(resamples = train_folds)

# Extract the accuracy metric from the cross-validation results
boost_accuracy <- boost_crossval_fit %>%
  collect_metrics() %>%
  filter(.metric == "accuracy")

# Print the collected metrics
boost_crossval_fit %>%
  collect_metrics()
```

xgboost tune

```
library(xgboost)
set.seed(666666)

#tuned parameters
boost_tree_model<- boost_tree(
  trees = 732,
  min_n = 12,
  tree_depth = 12,
  learn_rate = 0.01531878,
```

```
  loss_reduction = 8.004341e-04,
  sample_size = 0.6441571    ,
  stop_iter = 19
) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

boost_tree_workflow <- workflow() %>%
  add_model(boost_tree_model) %>%
  add_recipe(recipe, blueprint = hardhat::default_recipe_blueprint(allow_novel_levels = TRUE))

boost_fit <- boost_tree_workflow %>%
  fit(cleaned_train)

boost_result <- boost_fit %>%
  predict(test)

final <- test %>%
  select(id) %>%
  bind_cols(boost_result)


write_csv(final, "boost_result.csv")
```

naive model baseline

```
library(agua)  # Load the agua package, possibly your custom package
library(naivebayes)  # Load the naivebayes package for naive Bayes classifier
library(discrim)  # Load the discrim package, possibly your custom package

# Create a naive Bayes classifier model
naive_model <- naive_Bayes() %>%
  set_engine("naivebayes") %>%
  set_mode("classification")

naive_workflow <- workflow() %>%
  add_model(naive_model) %>% # Add the naive Bayes model to the workflow
  add_recipe(recipe2)

# Fit the naive Bayes model on the training set and perform cross-validation
naive_crossval_fit <- naive_workflow %>%
  fit_resamples(resamples = train_folds)

# Extract the accuracy metric from the cross-validation results
naive_accuracy = naive_crossval_fit %>% collect_metrics() %>% filter(.metric == "accuracy")
naive_crossval_fit %>% collect_metrics()
```

naive bayes tune

```
# create model
naive_model <- naive_Bayes(
```

```
  smoothness = tune(),
  Laplace = tune(),
) %>%
  set_engine("naivebayes") %>%
  set_mode("classification")

# create workflow
naive_workflow <- workflow() %>%
  add_model(naive_model) %>%
  add_recipe(recipe)

# extract tuned parameters
model_param = extract_parameter_set_dials(naive_model)

naive_tune <- naive_workflow %>%
  tune_grid(train_folds,
            grid = model_param %>% grid_random(size = 25)
  )

naive_tune %>%
  show_best()
```

```
df <- data.frame(Model_Name = c("Logistic linear regression", "Random forest", "Decision Tree", "Xgbo
                 Model_accuracy = c(logistic_accuracy$mean, rf_accuracy$mean, dec_tr_accuracy$mean, b
                 Model_roc_auc = c(logistic_accuracy$std_err, rf_accuracy$std_err,dec_tr_accuracy$std
                                   boost_accuracy$std_err, naive_accuracy$std_err))

# plot their accuracy and standard deviation using bar plot
par(mfrow = c(1, 2))
ggplot(df, aes(x = Model_Name, y = Model_accuracy)) +
geom_bar(stat = "identity")
ggplot(df, aes(x = Model_Name, y = Model_roc_auc)) +
geom_bar(stat = "identity")
```