Stats 101C Kaggle Competition Final Project: Classification

TEAM 11

Aldo Untoro, Antonio Lucchesi, Edward Halim, Ernest Salim, Justin Ko

Table Of Contents

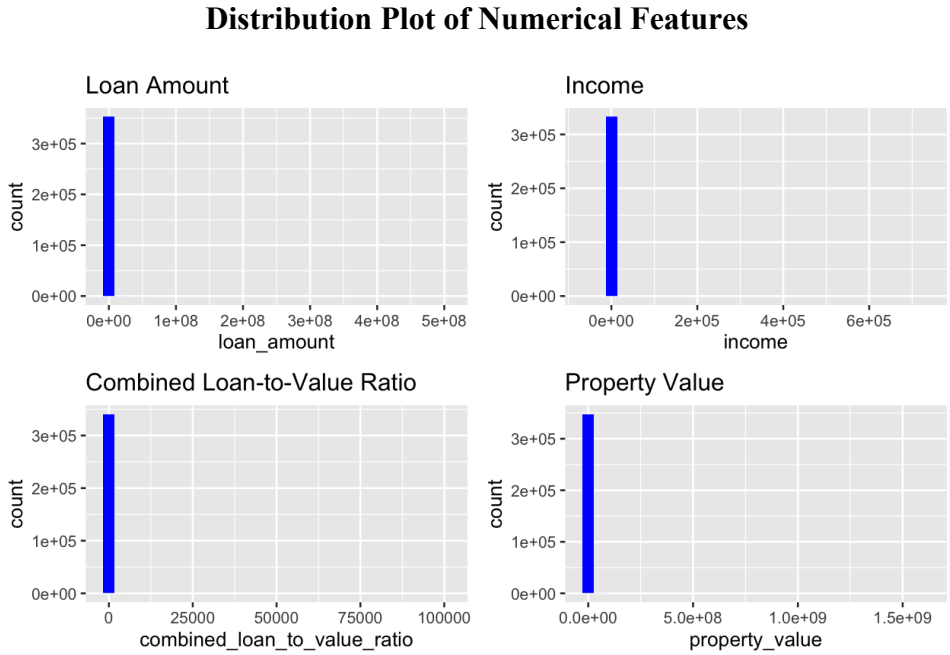
Introduction

Utilizing the comprehensive dataset from the Consumer Financial Protection Bureau, which encompasses detailed home mortgage application data, our objective is to predict the actions taken on these applications. Each entry in this dataset represents an individual application and contains a plethora of variables, 72 columns of variables. There are several columns which might capture pivotal parameters such as income, property value, sex of applicant, age of applicant, race of applicant, state, loan term, loan amount, and loan purpose.

The association between mortgage approvals and financial indicators has been firmly established in the literature. For instance, an encyclopedia study conducted in , employing a data-driven approach, delineated that applicants with higher income levels and more valuable properties are generally more likely to receive mortgage approvals (Dolf de Roos. “Real Estate Riches.” Wiley, 2001). This finding highlights the significant roles that financial stability and property value play in determining mortgage outcomes. Drawing from this, we hypothesize a strong association between these variables and the mortgage application outcomes in our dataset. In the ensuing sections, we will validate this hypothesis, exploring the multidimensional relationships between these parameters and the application outcomes through rigorous data analysis and classification model development.

Exploratory Data Analysis

Numerical Features



Description:

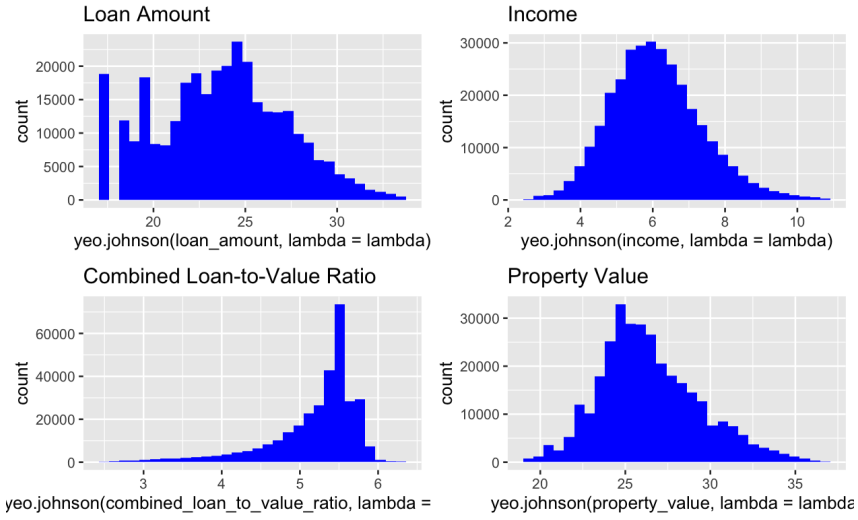
In the figure above, histograms show the univariate distributions of four numeric columns as they exist when we first read in the data. We cannot gather much information from this visualization other than the fact that there are clearly outliers for all of these features that interfere with the distributions.

Distribution Plot of Numerical Features after Outliers Removal

Description:

Once we identified the outliers, we removed them using basic R functions that implement quantile methods. If certain points are above or below a specific threshold (in our case above 99.5% and below 0.05% percentile), then we can safely remove those points because we consider those as outliers. After the dataset had approximately 2.8% of its total observations removed, we can now see that the distribution can be seen clearly and these features can be used for our predictor variables.

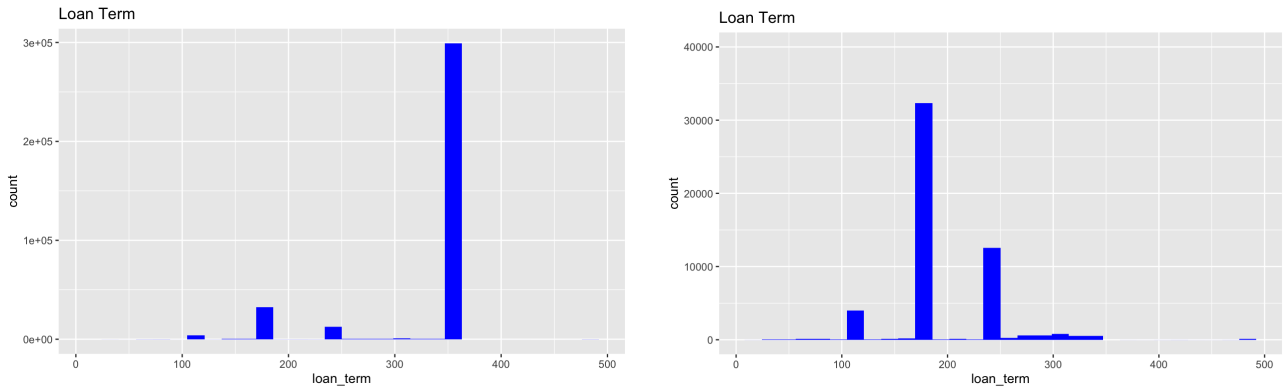
Distribution Plot of Numerical Features after Outliers Removal and Yeo-Johnson Transformation



Description:

After we remove the outliers, we are able to preprocess several features further with the purpose of making the distribution more centered and equally distributed. Here, we chose Yeo-Johnson transformation and it shows significant results for all features, except for the predictor combined_loan_to_value_ratio. We can see from the plot above that the column combined_loan_to_value_ratio is more skewed to the left compared to our initial distribution without any transformation. Thus, in this project, we are using log transformation for the features, but not for combined_loan_to_value_ratio.

Distribution of Loan Term



Description:

The dataset contained one final numerical feature that we decided not to include to the rest of numerical features distribution assessment. Reason being the distribution of loan_term felt very discrete and suffered from imbalanced data allocation, with almost 90% of the whole dataset clustered around the value of 360.

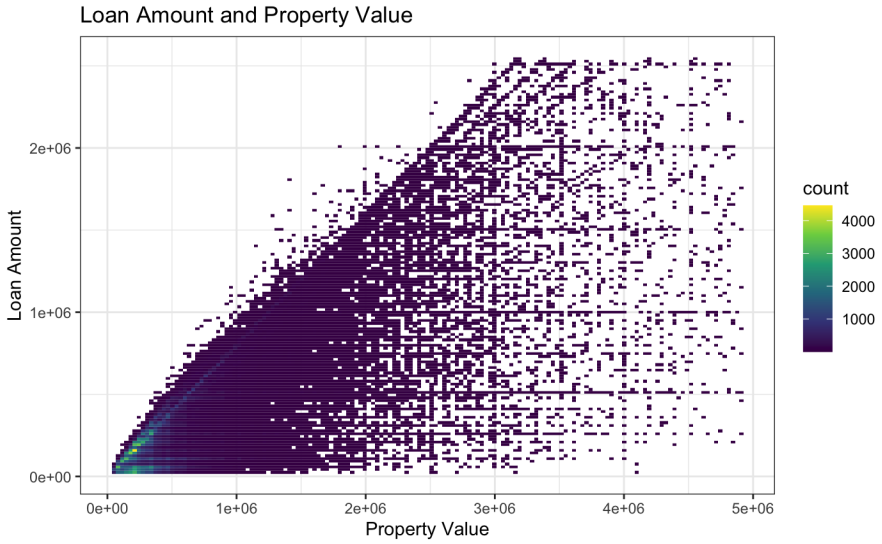
The visualization on the right is the closer vignette of the left distribution plot to get a better outlook on the smaller subset of the dataset.

Collinearity Matrix of Numerical Features

Description:

From our collinearity plot above, we can see that multicollinearity is not a problem when we plot the relationship between these 4 features. The relationship between each predictor is still tolerable and does not cause overfitting due to the unnecessarily complexity of the model. This means that we can keep the columns and we do not require to use step_corr() on these predictors in our recipe.

Scatterplot of Property Value against Loan Amount

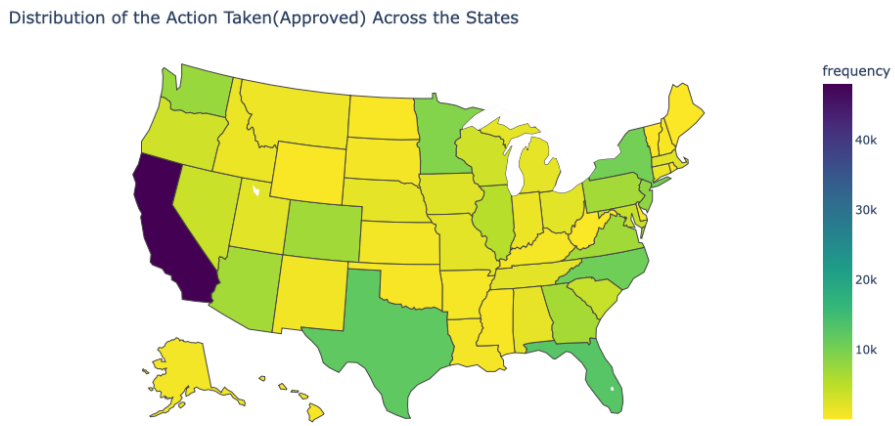


Description:

The figure above plots the relationship between Loan Amount and Property Value with color representing the density. Most points are clustered towards the bottom left as most loan applicants have loan amounts and property values below 1e+06 (1 million). We can observe a positive relationship such that an increase in property value will tend to indicate an increase in loan amount. Additionally, the absence of points on the upper left corner of the plot illustrate the strict cut-off in which higher loan amounts require a higher minimum property value.

Categorical Features

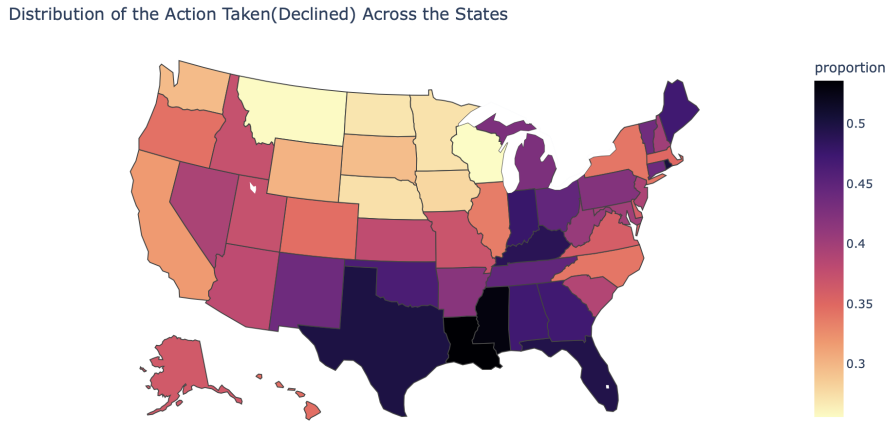
Spatial Frequency Distribution of Target Variable (# approved)



Description:

In the heat map above, we can get an idea of the number of approved loans within each state. The darker states have more approved loans, which could likely be a result of having more applications in general, or may reflect to some degree if a loan is more likely to be accepted depending on a state. As indicated by the chart above, California has emerged as the state with the most approved applications. This could be attributed to the fact that California receives a substantial number of applications.

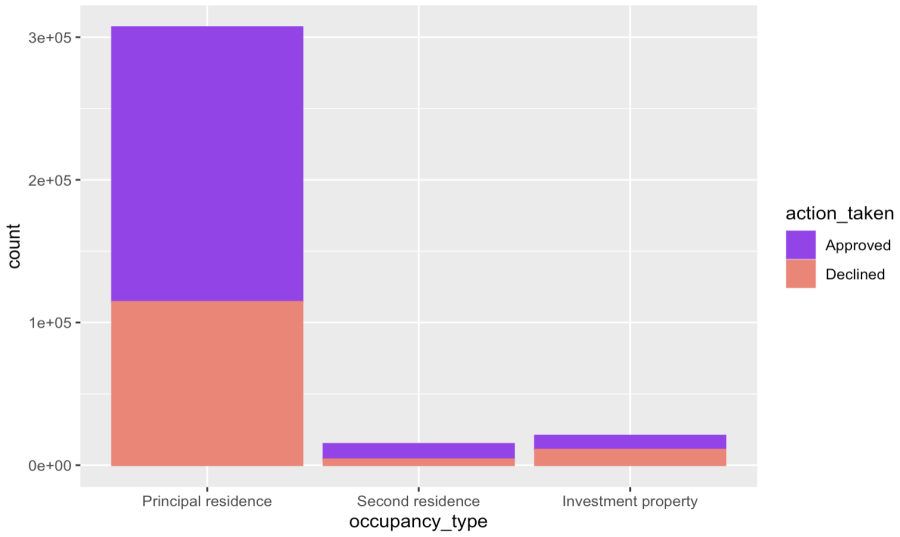
Spatial Proportion Distribution of Target Variable (% denied)



Description:

While the earlier heat map illustrated the number of accepted loans by state, here we observe the proportion of accepted loans by state, giving us a more clear picture on how likely a loan is going to be accepted or denied given the state. Here, the darker states have a higher proportion of denied applications and the lighter states have a higher proportion of accepted applications. For example, loan applications from Louisiana and Mississippi are more likely to have been denied compared to applications from other states.

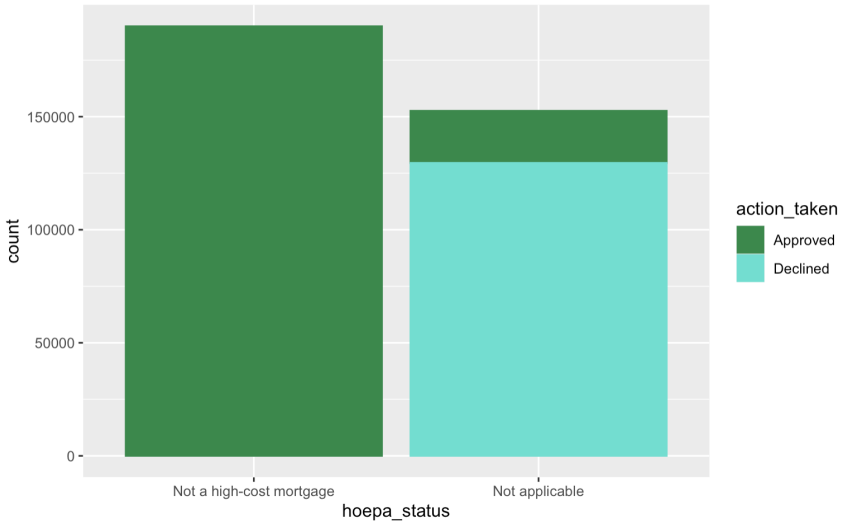
Bar Plot: Occupancy Type & Action Taken



Description:

From the bar plots above, it is clear that the vast majority of applicants have occupancy type as their principal residence. It can also be observed that among those applying for a secondary residence, there is a rather high proportion of approvals relative to other occupancy types.

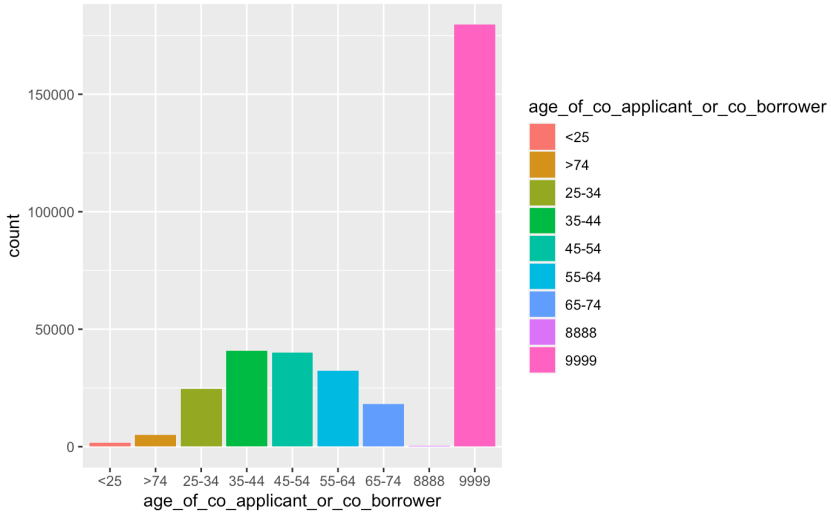
Stacked Frequency Plot: Action Taken & Mortgage Status



Description:

Based on our frequency plot, it becomes readily apparent that individuals possessing a low-cost mortgage status are more prevalent than those with no Hoepa status. Our research further supports this observation by revealing that individuals classified with a "not applicable" mortgage status tend to have a diminished likelihood of being approved.

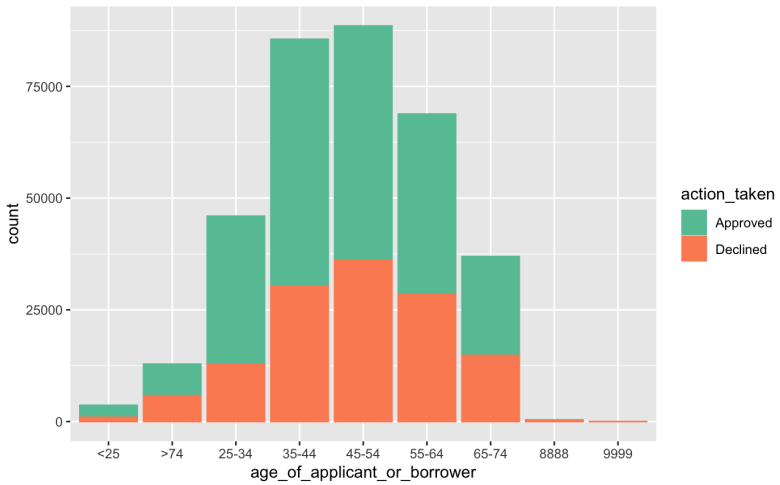
Problematic Distribution of Co-Applicant/Co-Borrower Age



Description:

Here we see a distribution for the age of co-applicant / co-borrower. On the surface, this feature did not show many NA values, however when we look more closely we can observe an extremely high frequency of the 9999 observation. After viewing the provided variable details, we can discover that the observations of 9999 represent missing values. This high proportion of missing values in this column indicates that it will likely need to be manually removed later on in preprocessing.

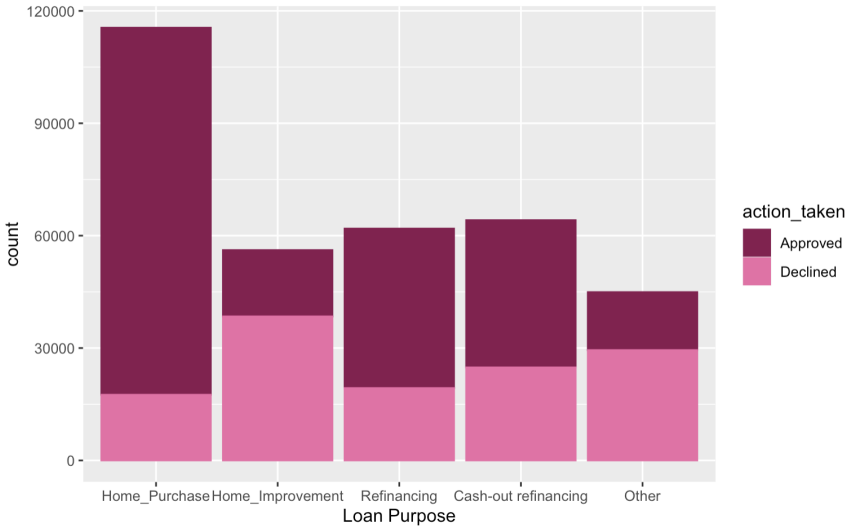
Stacked Frequency Plot: Action Taken & Age of Applicant



Description:

Above we can visualize the number of applicants grouped by their age categories in decade intervals. The graph also shows how many applicants were approved or denied within each age group. We can gather that most applicants are in the 35 to 64 range, and although there are less applicants in the 24-34 range, these applicants appear to have a higher proportion of approvals compared to other groups. Moreover, we can also remove the “default value” for NA which are 8888 and 9999 in our preprocessing data with step_mutate and step_impute in our recipe.

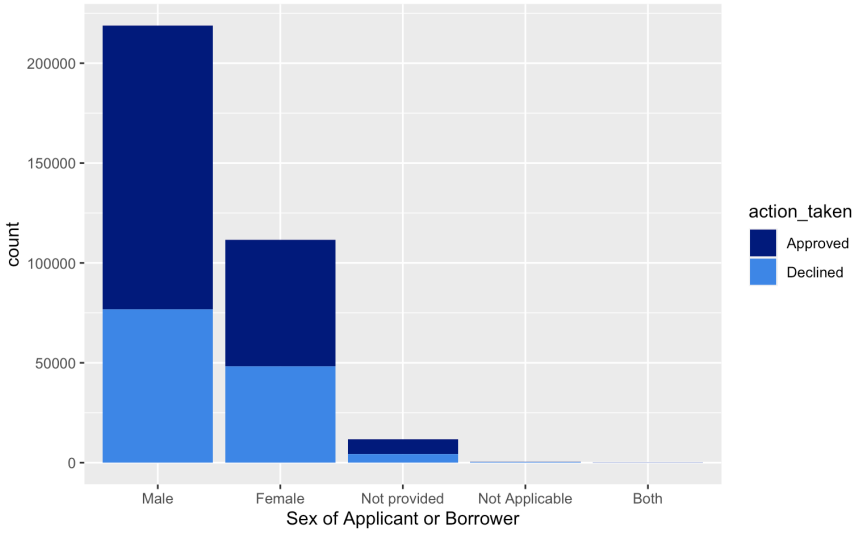
Stacked Frequency Bar Plot: Action Taken & Loan Purpose



Description:

The figure above gives the frequency of observations under each category of Loan Purpose. Home Purchase is by far the most common reason for loan applications. Aside from Home Purchase, there are a relatively similar number of occurrences in each group. Home Purchase, Refinancing, and Cash-out refinancing tend to have rather high approval proportions while Home Improvement and Other Purpose tend to have a higher proportion of denied applications.

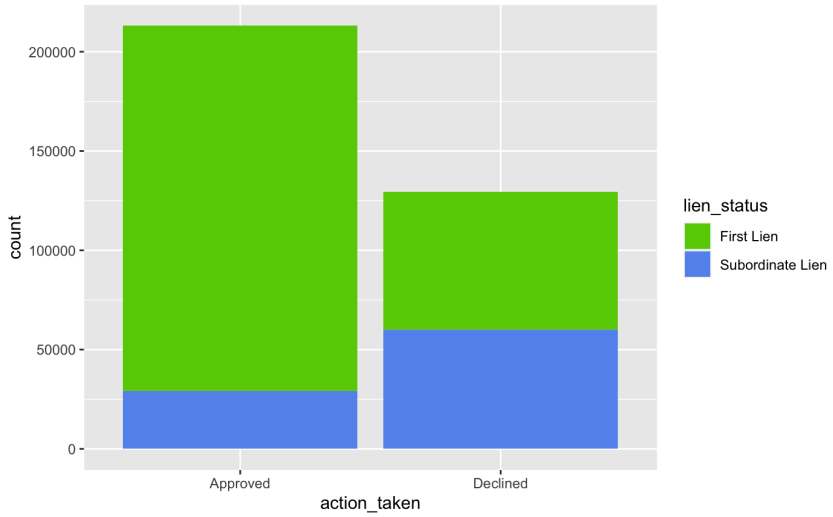
Stacked Bar Plot: Action Taken & Sex



Description:

The above stacked bar chart delineates the outcome of loan applications based on the gender of applicants. It manifests that a significant portion of both "Male" and "Female" applicants secure approval, with males having nearly double the applications and enjoying a marginally higher success rate than females. Conversely, the "Not provided" category witnesses a notable rate of declinations, hinting at a potential preference for applications with specified gender details. The "Not applicable" and "Both" categories represent a minority, indicating their lesser prevalence in the dataset.

Stacked Bar Plot: Lien Status & Action Taken



Description

Unlike the previous plots, this plot groups action_taken into two bars so we can better observe how much of the applications in total were approved and denied. Further, this plot illustrates how many applications were secured under a first lien or a subordinate lien given within the two action taken groups. As we can see, of the approved applications, a large majority were secured under a first lien. Of the declined applications, almost half were secured under a subordinate lien.

Preprocessing / Recipes

Initial Preprocessing

In the extensive dataset comprising 72 features, a substantial proportion of these features contain a significant number of missing values, with some reaching as high as 90% or even 100% of the entire column. To ensure the reliability and credibility of our exploratory data analysis, it became imperative to eliminate these particular features, as they did not contribute any meaningful information or value to the analysis or data prediction. Utilizing an initial preprocessing method **step_filter_missing** from the **recipes** package with a threshold of 0.1, we completely removed all features which had a number of missing observations exceeding 10% of the whole column. In reality, features either had lots of NA values or very few NA values, such that we could have technically gotten the same results using a threshold of up to about 0.52. Additionally, in controlling the outliers, we construct a function that could generate and handle the outliers by removing several values with the threshold above 99.5% and below 0.05% percentile.

```
remove_outliers <- function(df) {
  filter <- rep(TRUE, nrow(df))

  for (column_name in names(df)) {
    if (is.numeric(df[[column_name]])) {
      P_1 <- quantile(df[[column_name]], 0.005, na.rm = TRUE)
      P_2 <- quantile(df[[column_name]], 0.995, na.rm = TRUE)

      filter <- filter & (df[[column_name]] >= P_1 & df[[column_name]] <=
P_2 | is.na(df[[column_name]]))
    }
  }

  df <- df[filter, ]
  return(df)
}
```

Another discernible pattern that we found was the occurrence of zero variances throughout the entirety of the dataset. This trend also posed no value for establishing a high-performing classification model at the end of the day. Thus, **step_nzv** was utilized to eliminate features that suffered from the near-zero variance pattern in their data. Instead of using **step_zv**, **step_nzv** could simultaneously handle the problem of class imbalance that plagued some of the dataset's features.

Numerical Features

Due to the existence of remaining NA values, our recipe included **step_impute_mean** to replace the NA observations in the numeric columns. From the exploratory data analysis that we have assessed, we were confident in using **step_YeoJohnson** as a replacement for **step_log** that we were initially thinking of utilizing. A notable reason for doing this particularly with the income variable is that although the training data had positive incomes, the test data had negative incomes that presented issues when applying a log transformation. Ultimately, the YeoJohnson transformation was only applied to loan_amount, income, and property_value due to the lack of beneficial significance the other numerical features gained after transformation.

Moreover, because of the stark differences in the scale of these numerical features, we resorted to utilizing **step_normalize** to convert all numerical features' magnitude within the same scale.

Finally, when it comes to loan_term, **step_cut** was used to discretize the continuous value to categorical factor. By setting the boundaries to 150, 210, 270, 330, and 360, it successfully clustered the data to their respective groups.

Categorical Features

To properly process our nominal predictors, we first converted various features to factor since most numerical data are actually meant to be represented as categorical variables. For the train data, we also convert the target variable action_taken to factor with the reason being classification models are trying to predict categorical values instead of numeric ones.

```
factor_conversion <- function(feat) {
  if (n_distinct(feat) <= 20) {
    feat <- as.factor(feat)
  }
  return(feat)
}

train <- train %>%
  mutate(across(everything(), factor_conversion))

test <- test %>%
  mutate(across(everything(), factor_conversion))
```

The code chunk above shows how we convert features to factor. We take all variables and convert to factor all variables with 20 or less unique values. Next, we can start to process the

predictors by converting values in age_of_applicant_or_borrower that are equal to 8888 or 9999, and set them to NA to be imputed later. We did this by using **step_mutate** and the reason was because age is not realistically possible to be that high and these values are specified to represent NA according to the documentation. Later, we impute the NAs by utilizing **step_impute_mode** that replaces all NAs with the mode of the column. Furthermore, we used **step_dummy** to convert all factor features to binary terms. Finally, we took account for the binary tables that might have zero variance, and thus, we used **step_zv** to remove those predictor columns that might overfit our predictions.

Our initial preprocessing was done and all went well until we tried to predict the test dataset. During the fitting, we ran into an error that said there exists a state called "PR", which is not a state code and interrupted our predictions. Thus, we hard coded another **step_mutate** that replaces a state called "PR" and replaces it with NA so that it can be imputed using **step_impute_mode** just like what we mentioned above.

Recipe

```
rec <- recipe(action_taken ~ ., data = train_eval) %>%
  # Direct features manipulation
  step_mutate(state = factor(state)) %>%
  step_mutate(state = if_else(state == "PR", NA, state)) %>%
  step_mutate(age_of_applicant_or_borrower =
if_else(age_of_applicant_or_borrower %in% c(8888, 9999), NA,
age_of_applicant_or_borrower)) %>%
  # Missing data imputation
  step_impute_mean(all_numeric_predictors()) %>%
  step_impute_mode(all_nominal_predictors()) %>%
  # Discretizing
  step_cut(loan_term, breaks = c(150, 210, 270, 330, 390),
include_outside_range = TRUE) %>%
  # Numerical features transformation
  step_YeoJohnson(loan_amount, income, property_value) %>%
  step_normalize(all_numeric_predictors()) %>%
  # Categorical features transformation
  step_dummy(all_nominal_predictors()) %>%
  # Removing unnecessary features
  step_zv(all_predictors())
```

Candidate models / Model evaluation / Tuning

For this project, we used classification machine learning models with random grid search tuning, as random grid search tuning had provided significant and reliable results as reflected in our previous regression project. For this method, the random grid size differs between models as certain algorithms took a much longer time to train and evaluate.

```
grid <- grid_random(extract_parameter_set_dials(bag_model), size = x)

fit <- mlp_wf %>%
  tune_grid(resamples = folds,
            grid = grid,
            metrics = metric_set(accuracy, roc_auc),
            control = control_grid(verbose = TRUE))
show_notes(.Last.tune.result)
```

Decision Tree

Our first candidate model is the **Decision Tree** model that implements **rpart** engine.

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

Since the **Decision Tree** showed relatively excellent results, considering how easy, lightweight, and interpretable the training phase was, we could leverage this model to become the base benchmark of our upcoming model training.

Random Forest

Next up was a **Random Forest** model that implements the **Ranger** engine, allowing us to extremely extend the size of the trees and integrating ensemble methods in hopes of getting a better performance.

```
rand_forest_spec <- rand_forest(
  trees = tune(),
  min_n = tune()
```

```
) %>%
  set_engine("ranger") %>%
  set_mode("classification")
```

Despite the better performance, the **Random Forest** model required a very exhaustive and extensive learning time. Increasing the time required for learning from minutes to hours. We decided to not use **Random Forest** as the tradeoff was not ultimately worth the increased metric once we progressed to creating stacks and workflow sets.

Bagged Trees

Next, we resorted to a less exhaustive ensemble tree model: **Bagged Trees**, which is a more general form of **Random Forest** and equipped with bootstrapping. The model uses **rpart** as its engine.

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

Compared to the previous ensemble model, **Bagged Trees** didn't require a lot of time to train as each tree only utilized a subset of the whole dataset for individual training. Topped with a slightly better performance against **Random Forest**, this proved to be a more reliable candidate than **Random Forest**.

Boosted Trees

In addition to the previous model, we were curious about the predictive power of **Boosted Trees**, which was regarded as one of the best models on the basis of public sentiment. The **Boosted Trees** model implements the **LightGBM** engine instead of the usual **XGBoost** because **LightGBM** is way faster in terms of training time by utilizing numerical discretizing and parallel processing, considering how enormous our dataset is. **XGBoost** was in fact attempted, however hyperparameter tuning was never able to be completed due to the extreme time demands.

```
boosted_tree_spec <- boost_tree(
  mtry = tune(), trees = tune(), tree_depth = tune(),
  learn_rate = tune(), min_n = tune(), loss_reduction = tune()
) %>%
```

```
  set_engine("lightgbm") %>%
  set_mode("classification")
```

Multilayer Perceptron (Single-Layer Feedforward NN)

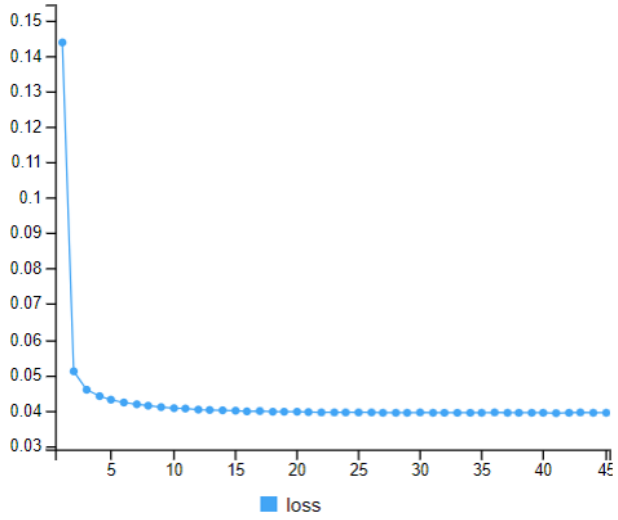
Outside of the realm of decision trees-based models, we continued on venturing other supervised methodologies like deep learning and **Neural Networks**. Instead of developing the **neural network** across multiple sequential layers, parsnip provides a convenient **single-layer feedforward neural network (MLP)** model via **Keras**, one of the most prominent engines in the field of deep learning.

During the tuning phase, what made the model training different was that we did not have to tune the epochs, because the **MLP** model tends to be slower in the tuning process compared to tree ensemble models. If we chose to tune the epochs, random epochs might occur and processing thousands of epochs would be a redundant process as the loss metric could reach a plateau.

Thus, we first set the epochs to 100, ran the benchmark model, and observed how many epochs the model expected to reach a plateau, then we extracted the value and set the value statically to 45. Activation function was also set to **softmax** as we were dealing with a classification problem.

```
mlp_spec <- mlp(
   penalty = tune(),
   epochs = 45,
   hidden_units = tune(),
   activation = "softmax") %>%
set_mode("classification") %>%
set_engine("keras")
```

In terms of performance, **MLP** produced a somewhat excellent output, two disadvantages that the model has is the lack of interpretability and how difficult it is to scale the model to multiple layers, instead of relying on only one layer. We initially wanted to try **Keras**' **Sequential** model, but it was hindered by the fact that we were using tidymodels package to ultimately create an ensemble of models, and **tidymodels** provides support no further than single-layer feedforward **neural network**.



The visualization above illustrates the gradual decrease of loss metric during the fitting of MLP models. After 35-45 iterations, it is apparent that the loss has reached a plateau and further iterations are unnecessary.

Bagged MLP

In addition, parsnip makes it available for us to implement bagging to the previous model that we had experimented with. By integrating the **Bagged MLP** model, we could easily tune this model the same way we tuned the previous single **MLP** model, and we were left with a static epoch value of 30.

```
bagged_mlp_spec <- bag_mlp(
  hidden_units = tune(),
  penalty = tune(),
  epochs = 30) %>%
set_mode("classification") %>%
set_engine("nnet")
```

Conversely, this model was hindered by rigorous fitting of multiple single-layer **neural network**, in which each network required a lengthy amount of training time. Moreover, the final metric slightly decreased from its previous counterpart, thus making this model somewhat trivial.

Logistic Regression (Elastic-net Regularization)

Apart from tree-based and deep-learning-based models, we also tried utilizing a simple **Logistic Regression** with the **glmnet** engine. Because of its simplicity and interpretability, the model tends to be associated with less-powerful predictive nature and high overfitting likelihood. Therefore, elastic-net regularization was integrated to the model to counter future overfitting scenarios.

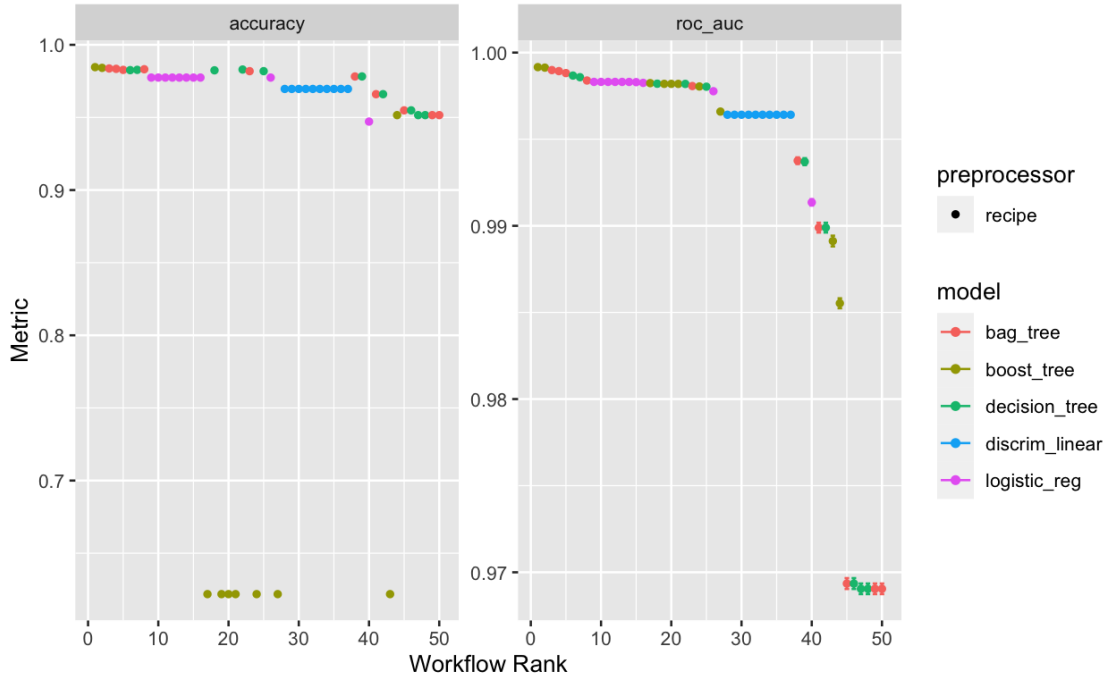
```
logistic_reg_spec <- logistic_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet") %>%
  set_mode("classification")
```

Linear Discriminant Analysis

Finally, an interesting model that we applied for this project is the **LDA** Classification Model. This model implements linear discriminant that is flexible and curated for classification. Using the **mda** engine gave slightly better results compared to the **MASS** engine, and while the **MASS LDA** model does not have tunable parameters, the **mda LDA** has a tunable penalty parameter. Nonetheless, hyperparameter tuning presented no significant difference in model accuracy between different penalty specifications so we resorted to the default form.

```
lda_spec <- discrim_linear() %>%
  set_engine("mda") %>%
  set_mode("classification")
```

Hyperparameters Tuning Performance



We ultimately decided to assess final random hyperparameters tuning and evaluation on **Decision Tree**, **Bagged Trees**, **Boosted Trees**, **Logistic Regression**, **Linear Discriminant Analysis**, **MLP (Single-Layer)**, and **Bagged MLP** using workflowsets to generate visualizations across 5 cross-validation folds, allowing us to form a more candid judgment on each model. The above visualization only encompassed 5 model evaluations (excluding MLP and Bagged MLP). It was apparent that from the random search tuning, **Boosted Trees** was the best-performing model.

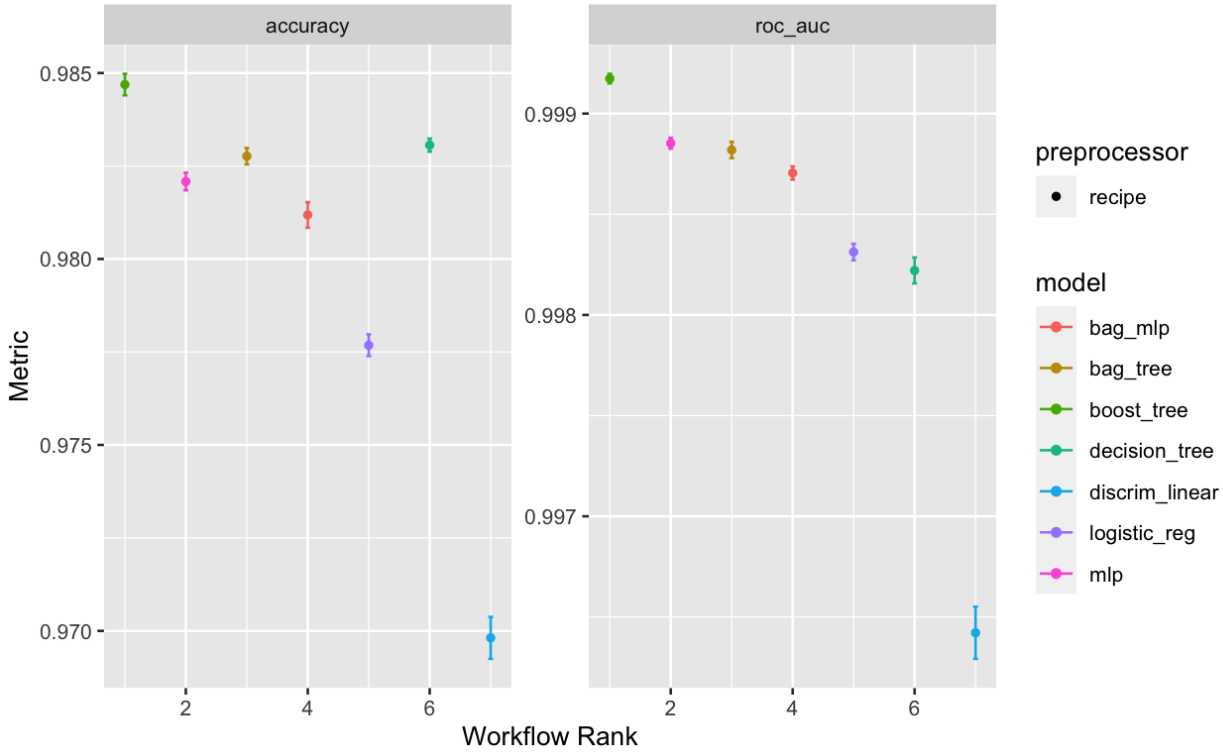
Eventually, we collected all the best parameters from the tuning result and proceeded to fit those models across 10 cross-validation folds for a more accurate score and to compile everything to construct the final model ensemble.

Final Hyperparameters for Each Model

MODEL	ENGINE	HYPERPARAMETERS
Decision Tree	rpart	tree_depth = 13 cost_complexity = 7.058241e-05 min_n = 26
Bagged Trees	rpart	tree_depth = 9 min_n = 4

		cost_complexity = 6.711050e-09
Boosted Trees	Lightgbm	trees = 1257 tree_depth = 8 min_n = 11 learn_rate = 6.525540e-02
MLP (Single-Layer)	Keras	penalty = 0.0000000366 epochs = 45 hidden_units = 8
Bagged MLP (Single-Layer)	nnet	hidden_units = 7 penalty = 1.18e-10 epochs = 30
Logistic Regression	glmnet	penalty = 2.989194e-08 mixture = 0.2268432
Linear Discriminant Analysis (LDA)	mda	Default Hyperparameters

Tuned Model 10 CV Performance



From the autoplot that the workflowsets have generated, **Boosted Trees** (**Light GBM**) performed the best in both metrics, whereas other models underperformed within a very small margin.

Because of how small the difference window is between models, we could employ ensemble methods to combine their predictive power to form a unified final model which was more robust and resilient to unseen patterns and to prevent the model from relying on one algorithm only.

Table of Performance

MODEL ID	METRIC SCORE (ACCURACY)	METRIC SCORE (ROC_AUC)	SE (ACCURACY)
Decision Tree	0.9830613	0.9982210	1.064927e-04
Bagged Trees	0.9827636	0.9988195	1.350721e-04
Boosted Trees	0.9846901	0.9991739	1.759626e-04
MLP (Single-Layer)	0.9820834	0.9988530	1.444326e-04
Bagged MLP (Single-Layer)	0.9811844	0.9987050	2.089531e-04
Logistic Regression	0.9776816	0.9983124	1.777869e-04
Linear Discriminant Analysis	0.9698121	0.9964216	3.426538e-04

Discussion of Final Model

Final Model

It's feasible to exclusively utilize the top-performing **Boosted Trees** model for generating final predictions on the test dataset. However, it's generally advisable not to solely depend on a single model, considering that each model possesses its own strengths and weaknesses depending on the underlying patterns in unseen data. Consequently, to bolster the dependability and predictive capacity of our model, we can establish an ensemble of models by assigning varying weights to each model. This approach allows the model to aggregate predicted values from each member and consider their significance in creating the ultimate prediction.

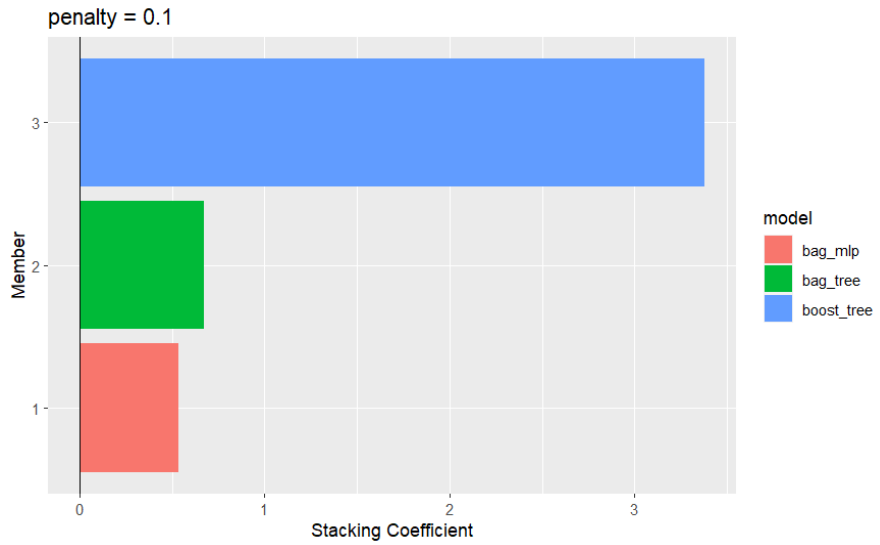
The Tidymodels package features a specialized library, **stacks**, dedicated to constructing such model ensembles. This empowers us to combine our previously trained 8 models (workflows in this scenario) into a unified model stack.

```
wfl <- workflow_set(
    preproc = list(mod = rec),
    models = list(tree = tree_spec,
                  bagtree = bagtree_spec,
                  boosttree = boosttree_spec,
                  logistic = logistic_spec,
                  mlp = mlp_spec,
                  bagmlp = bagmlp_spec,
                  lda = lda_spec),
    cross = TRUE
  ) %>% option_add(control = control_grid(save_workflow = TRUE,
                                          save_pred = TRUE, verbose = TRUE))

model_stack <- stacks() %>%
  add_candidates(wfl_fitted) %>%
  blend_predictions()
```

Furthermore, the **blend_predictions** function from the data stack was utilized to assign weights to each member depending on their error metrics. The higher the weight, the better the member is in terms of performance.

```
set.seed(2023)
model_stack <- stacks() %>%
  add_candidates(wfl_fitted) %>%
  blend_predictions()
```



The visualization depicts the weight of each model during the final ensembling. The general rule of thumb is the higher the weight, the better the model in predicting the unseen data. In this case, **Boosted Trees** took up the biggest proportion of the final ensemble model, solidifying itself to be the most significant member of the ensemble. In addition, the other two models: **Bagged Trees** and **Bagged MLP** shared almost the same proportion in terms of magnitude, gaining them a pretty significant right-of-say. On the other hand, models like **Decision Tree**, **Logistic Regression**, **MLP**, and **LDA** were not included in the ensemble due to their slightly worse performance.

Strengths and Weaknesses

Strengths:

1. Since we relied on a stacked ensembling method, our model choices were not limited to one, but rather the algorithm assigned weighted coefficients to the best performing models that we had tuned individually and combined, making it easier for us to compare the final result. Moreover, utilizing stacks inherently helps the model in reducing the impact of individual model uncertainties or errors.
2. Our final model reflects consistency when we compare the public dataset and private dataset. Thus, we trust our model to be reliable and the performance of our model is not heavily influenced by random variations. We firmly believe that our model is robust and is neither overfitting nor underfitting to either dataset, which also indicates that our model has generalized well to unseen data.

Weaknesses:

1. Our approach relied on a single generalized preprocessor recipe for each model during training, which potentially limits the predictive prowess of each individual model.
2. We did not fully explore the spectrum of available models that could have been included in the final ensemble, thereby impeding the realization of the ultimate potential of our model.
3. The final model tuning and fitting took an absurdly long time, especially during the fitting of **MLP** and **Bagged MLP**, combined with the enormous size and dimension of the dataset, made the whole evaluation process way more tedious than it should have been.

Areas of Improvement

1. In light of our time constraints and rigorous experimentation of 7 models, incorporating additional new models presents an opportunity to better enhance the final predictive accuracy.
2. Despite our best efforts to remove, clean, and process feature engineering, there might be more unseen patterns that we have not captured completely from the initial dataset, hindering the model's ability to understand the full complexity and relationship of the dataset's features.

Appendix: Final annotated script

```
# Loading required libraries
library(tidyverse)
library(tidymodels)

library(rpart)
library(ranger)
library(baguette)
library(bonsai)
library(lightgbm)
library(xgboost)
library(keras)
library(tensorflow)
library(nnet)
library(reticulate)
library(mda)
library(discrim)
library(stacks)

# Reading the datasets
train <- read_csv("train2.csv")
test <- read_csv("test2.csv")

# Removing problematic features
train <- recipe(action_taken ~ ., data = train) %>%
  step_rm(id, age_of_co_applicant_or_co_borrower) %>%
  step_filter_missing(all_predictors(), threshold = 0.1) %>%
  step_nzv(all_predictors()) %>%
  prep() %>%
  juice()

# Removing numerical outliers
remove_outliers <- function(df) {
  filter <- rep(TRUE, nrow(df))

  for (column_name in names(df)) {
    if (is.numeric(df[[column_name]])) {
      P_1 <- quantile(df[[column_name]], 0.005, na.rm = TRUE)
      P_2 <- quantile(df[[column_name]], 0.995, na.rm = TRUE)

      filter <- filter & (df[[column_name]] >= P_1 & df[[column_name]] <= P_2 |
is.na(df[[column_name]]))
    }
  }
  df <- df[filter, ]
  return(df)
```

```
}
train <- remove_outliers(train)

# Converting numerical to factors
factor_conversion <- function(feat) {
  if (n_distinct(feat) <= 20) {
    feat <- as.factor(feat)
  }
  return(feat)
}
train <- train %>%
  mutate(across(everything(), factor_conversion))
test <- test %>%
  mutate(across(everything(), factor_conversion))


# ------------------------------------------------------------


# Creating 10 cross-validation folds
set.seed(2023)
folds <- vfold_cv(data = train, v = 10, strata = action_taken)

# Preprocessing recipe
rec <- recipe(action_taken ~ ., data = train) %>%
  step_mutate(state = factor(state)) %>%
  step_mutate(state = if_else(state == "PR", NA, state)) %>%
  step_mutate(age_of_applicant_or_borrower = if_else(
    age_of_applicant_or_borrower %in% c(8888, 9999),
    NA,
    age_of_applicant_or_borrower)) %>%
  step_impute_mean(all_numeric_predictors()) %>%
  step_impute_mode(all_nominal_predictors()) %>%
  step_cut(loan_term, breaks = c(150, 210, 270, 330, 390), include_outside_range =
TRUE) %>%
  step_YeoJohnson(loan_amount, income, property_value) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors())


# ------------------------------------------------------------


# Model specifications
# Model 1 - Decision Tree
tree_spec <- decision_tree(
```

```
  tree_depth = 13,
  cost_complexity = 7.058241e-05,
  min_n = 26
) %>%
  set_engine("rpart") %>%
  set_mode("classification")

# Model 2 - Bagged Trees
bagtree_spec <- bag_tree(
  tree_depth = 9,
  min_n = 4,
  cost_complexity = 6.711050e-09
) %>%
  set_engine("rpart") %>%
  set_mode("classification")

# Model 3 - Boosted Trees
boosttree_spec <- boost_tree(
  trees = 1257,
  tree_depth = 8,
  min_n = 11,
  learn_rate = 6.525540e-02
) %>%
  set_engine("lightgbm") %>%
  set_mode("classification")

# Model 4 - Logistic Regression with Elastic-Net Regularization
logistic_spec <- logistic_reg(
  penalty = 2.989194e-08 ,
  mixture = 0.2268432
) %>%
  set_engine("glmnet") %>%
  set_mode("classification")

# Model 5 - MLP (Single-Layer FNN)
mlp_spec <- mlp(
    epochs = 45,
    hidden_units = 8,
    penalty = 0.0000000366,
    activation = "softmax"
  ) %>%
  set_engine("keras") %>%
  set_mode("classification")

# Model 6 - Bagged MLP (Single-Layer FNN)
bagmlp_spec <- bag_mlp(
    hidden_units = 7,
```

```
    penalty = 1.18e-10,
    epochs = 30
  ) %>%
  set_engine("nnet") %>%
  set_mode("classification")

# Model 7 - Linear Discriminant Analysis
lda_spec <- discrim_linear() %>%
  set_engine("mda") %>%
  set_mode("classification")


# ------------------------------------------------------------


# Creating workflow
wfl <- workflow_set(
  preproc = list(mod = rec),
  models = list(tree = tree_spec,
                bagtree = bagtree_spec,
                boosttree = boosttree_spec,
                logistic = logistic_spec,
                mlp = mlp_spec,
                bagmlp = bagmlp_spec,
                lda = lda_spec),
  cross = TRUE
) %>% option_add(control = control_grid(save_workflow = TRUE,
                                        save_pred = TRUE, verbose = TRUE))

# Fitting workflow
set.seed(2023)
wfl_fitted <- wfl %>% workflow_map("fit_resamples", resamples = folds, verbose =
TRUE)

# Constructing model stack
set.seed(2023)
model_stack <- stacks() %>%
  add_candidates(wfl_fitted) %>%
  blend_predictions() %>%
  fit_members()


# ------------------------------------------------------------


# Making predictions
pred <- model_stack %>%
```

```
  predict(test)

# Creating prediction tibble and export
pred <- test %>%
  select(id) %>%
  bind_cols(pred)

write_csv(pred, "pred.csv")
```

Appendix: Team member contributions

Aldo Untoro - EDA / EDA Explanations / Preprocessing / Model Training and Evaluation (Random Forest, C5, Boosted Trees)

Antonio Lucchesi - EDA & Descriptions / Preprocessing / Model Training and Evaluation (LDA, MLP, Boosted Trees)

Edward Halim - EDA / Preprocessing / Model Training and Evaluation (MLP, Bagged MLP), Final Model Ensemble

Ernest Salim - EDA / Preprocessing / Model Training and Evaluation (Tree Ensemble Models, Logistic Regression), Final Model Ensemble

Justin Ko - EDA / Preprocessing / Project Manager