# 2020 Presidential Election

Yechen Cao, Jingran Zhang, Pinyi Li, Vieno Wu

## 1. Introduction

This report explains the process of constructing classification models for a train dataset through supervised learning. The dataset used in this study is retrieved from Kaggle, for which the data of the election winner of each county where demographic information is recorded. The demographic information is estimated by the U.S Census Bureau. The dataset contains four files: train_class.csv, test_class.csv, col_descriptions.txt, and sample_submission.csv. For the purpose of this specific study, we use the train_class.csv and test_class.csv and modify them accordingly in the process of building the final classification model.

The objective of this study is to build a model such that it will predict the winner of a county, based on various predictors. According to the col_description.txt, we select the predictor columns that include distinct age levels, racial composition, and education level with statistical significance to our prediction.

## 2. Data Analysis

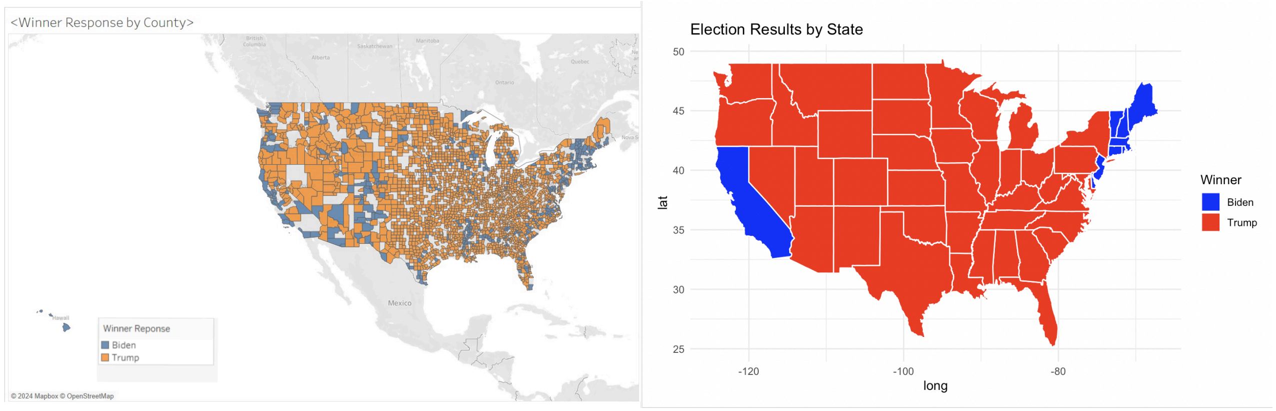

<Winner Response by County>
Winner Reponse
Biden
Trump
© 2024 Mapbox © OpenStreetMap
Election Results by State
lat
long
Winner
Biden
Trump

**Figure 1 Election winner for county and states (2020)**
The map on the left visualizes the winner of each county in 2020 where orange represents Trump and blue represents Biden. We see that there are more counties that voted for Trump than for Biden. The map on the right visualizes the mode of winners of all counties in each state where Trump is coded red and Biden is coded blue. In our training data most states had Trump as the winner, with only California and some northeastern states with Biden as the winner. This visualization does not reflect the true winner of each state in 2020, however, as our training data consists of  mostly counties that voted for Trump. If we do not generalize our model well then this plot suggests that it could be overfitting when the model learned the excessive details of the training data.

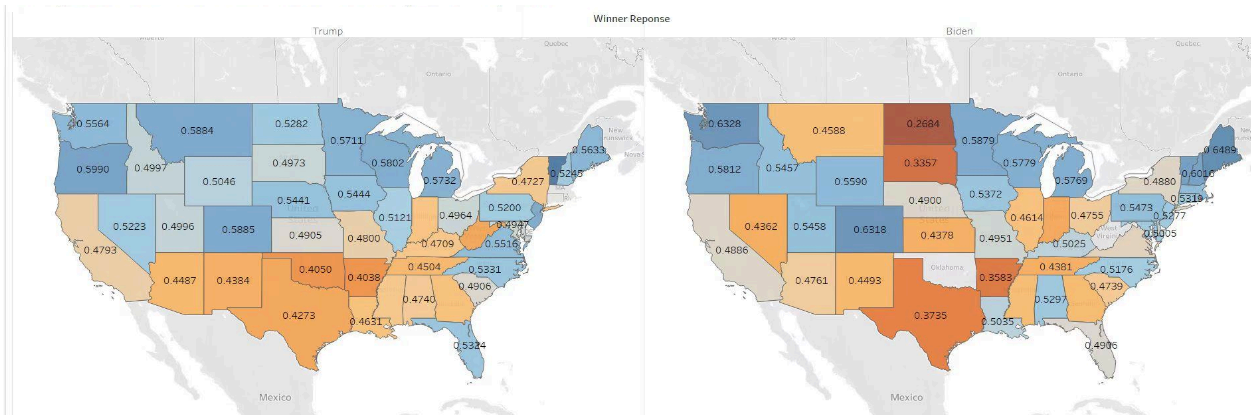
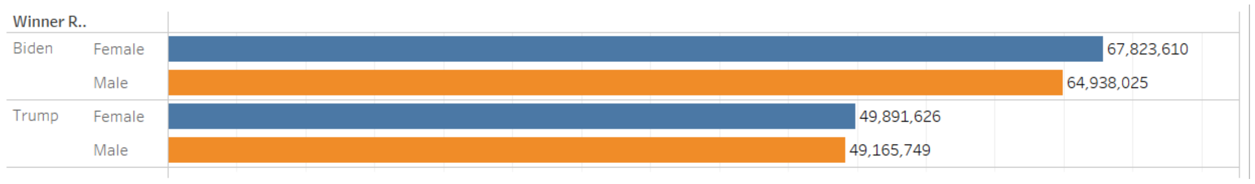


**Figure 2 Average Voter Turnout rate of Candidates' winning counties by State**
The US States map above shows the average of the voter turnout rate, computed by total votes / total population, of all of the winning counties in a State for each of the presidential candidates respectively. The redder the color, the lower the rate; the bluer the color, the higher the rate. The median rate, indicated by states of neutral color, is about 0.49. From the above two maps, we see

that overall, southern states have a lower voter turnout rate for both candidates. In terms of individual candidates though, even within the winning counties, Biden has a lower voter turnout rate in more states than that of Trump.



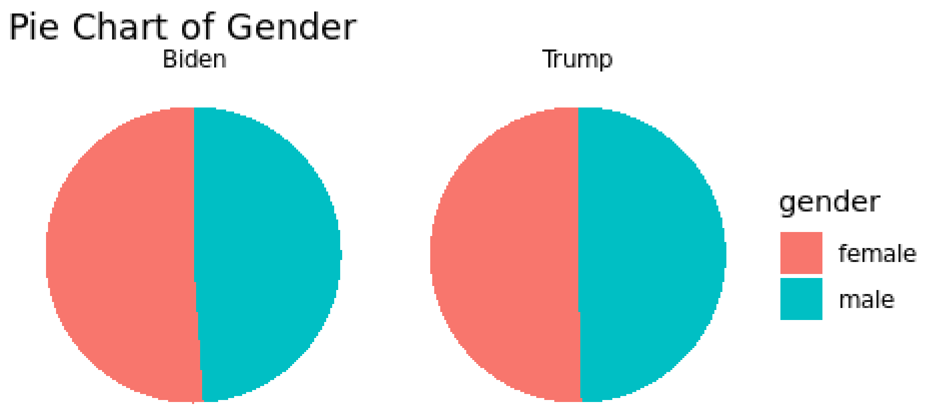


**Figure 3 Distribution of population by Gender within Candidates' winning counties**

The pie chart shows the sum of population by gender of all of the winning counties to their respective candidate. From the above bar graph in counts, we can see the total population of Biden's winning counties is more than that of Trump's winning counties. However, when we compare to that of Figure 1, we see that Trump has significantly more number of counties winning against Biden. To combine these lines of information, we can conclude that Biden is winning the counties with more population and with more female population; whereas Trump is winning the counties with smaller population yet he is winning many of those smaller counties.

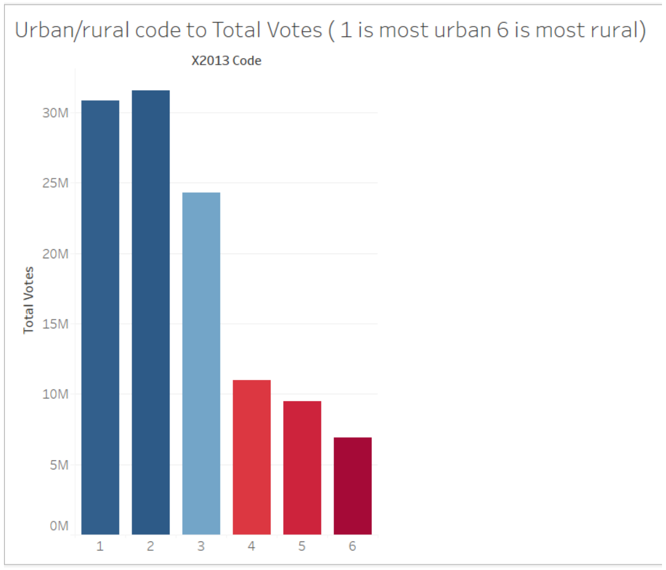


**Figure 4 Voter distribution over urban/rural counties (1 is most urban, 6 is most rural)**

The bar chart above shows the number of votes from six different levels of counties in terms of their urbanization measure. From the above, a general trend can be concluded that the more urban the county, the more voters it has. In addition to the population difference intuitively, we may also attribute the fact that urban counties are more expressive than that of rural counties. Notice the counties with the most urban (1) label do not have a higher number of votes than the 2nd highest urban counties. This could be due to the fact that the most urbanized counties, like LA for example, is also an international city where not everyone living there has the right to vote.

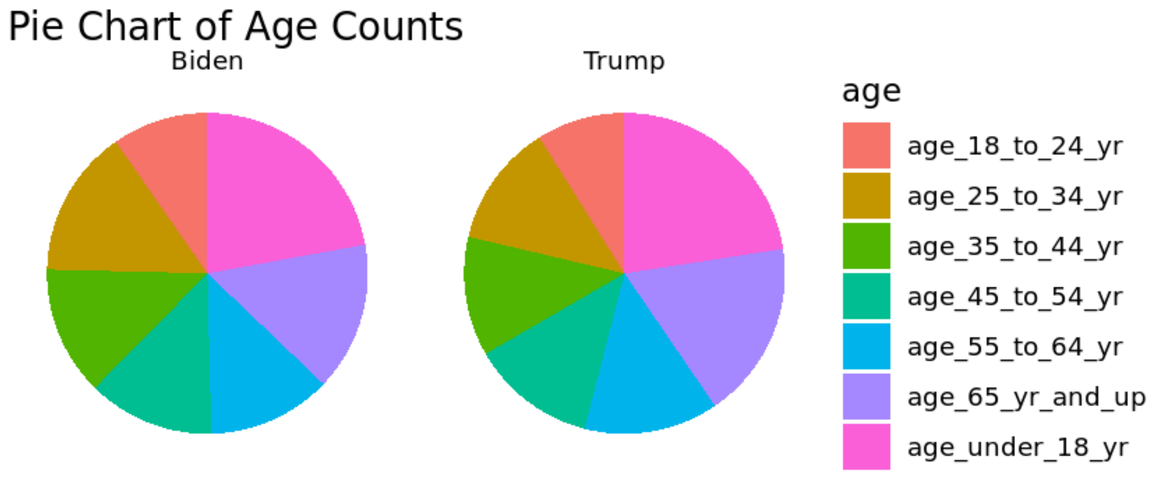


**Figure 5 Distribution of population by age within Candidates' winning counties**

The two graphs above show the distribution of population by age within each candidates' winning counties from age under 18 years to age 65 years and up. We have already seen that Biden's supporting counties have more population in count than that of Trump, which is again reflected here when comparing on the same scale. We also see that in Biden's chart, there are a lot more people under 18 than any of the other age groups as well as age 25-34 suggesting a younger population than that of Trump.

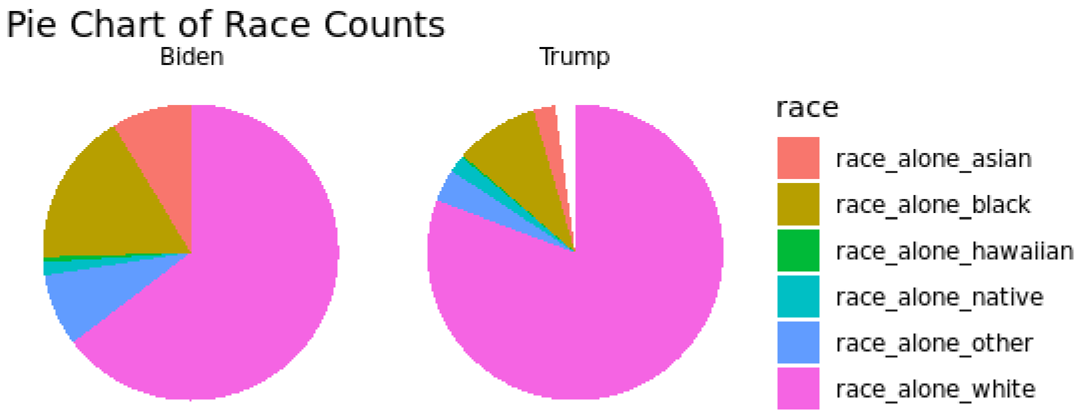


**Figure 6 Distribution of population by Race in Candidate's winning counties.**

The figure above shows that both candidates have the white population as the main racial group of population. However if we examine other racial groups, we see that Biden has a significantly larger population of minority racial groups such as Asian and Black and others in his winning counties than that of Trump. These could potentially show the correlation between population makeup and their supporting rate when measuring their winning chances. Combining with previous voter turnout data though, we can also see that white population tends to have a higher voter turnout rate.

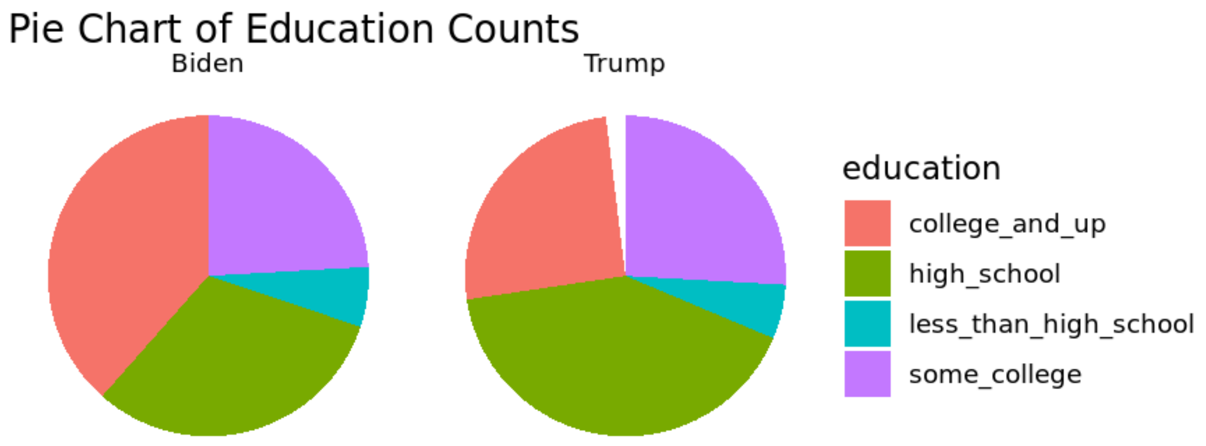


**Figure 7 Distribution of population in each Candidates' winning counties by Education**

Based on the above pie charts, we can conclude that both candidates have about the same ratio of population with education level as "some_college" and "less_than_high_school"; in terms of the remaining two categories, "college_and_up" and "high_school", we see that Biden has a significantly higher ratio of education level "college_and_up" to the population in his winning counties, whereas Trump has a higher ratio of education level of "high school". This suggests that Biden's winning counties have higher education levels in general compared to that of Trump.

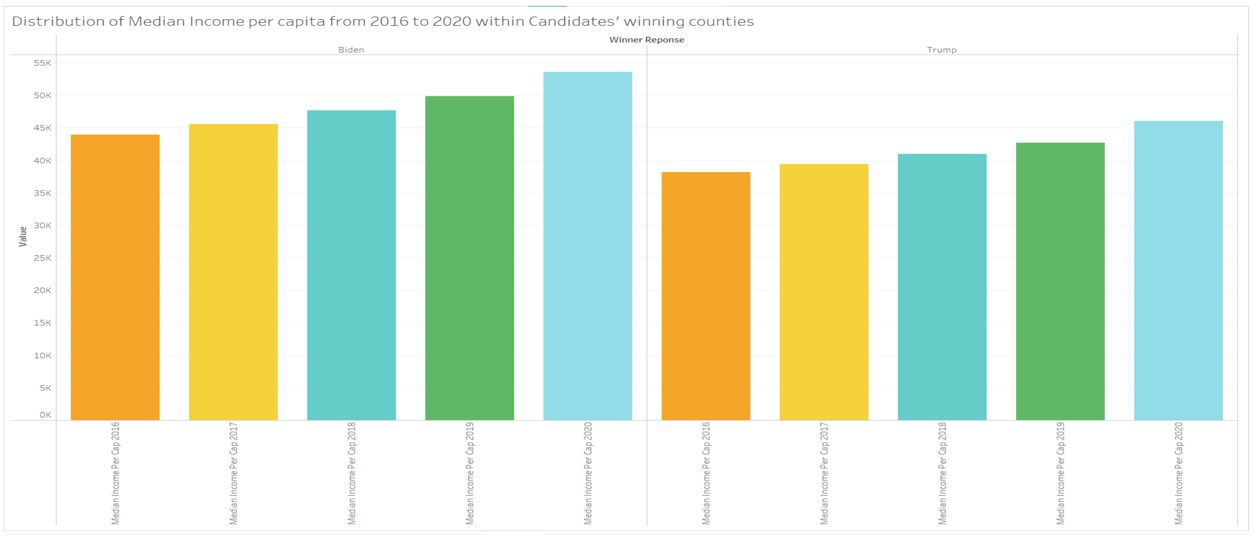


**Figure 8 Median income per capita in Candidates' winning counties from 2016 to 2020**

The above bar graphs represent the median income per capita within those winning counties by candidate. We can observe a general trend such that Biden's winning counties have a higher median income than that of Trump's in all of the years. In addition, Biden's winning counties also demonstrate a high number of increases from year to year. Higher income could have resulted in a higher amount of election donation. This data could have also suggested that population with less income favor Trump over Biden more.

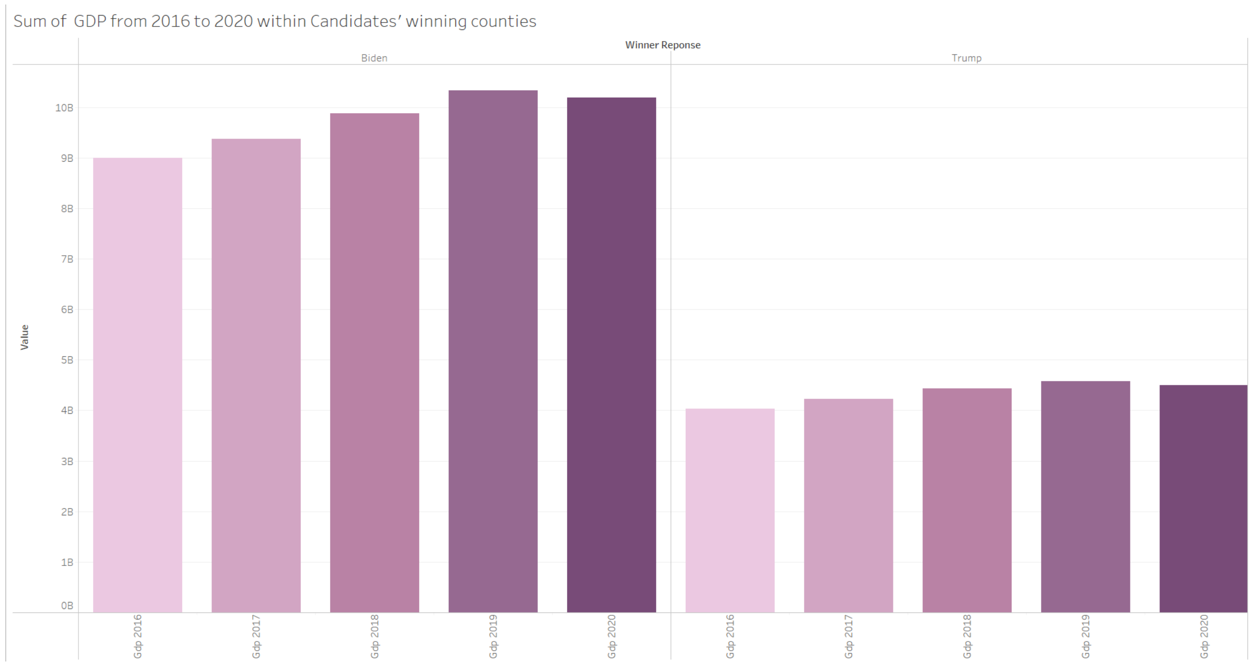


**Figure 9 Sum of GDP in each Candidates' winning counties by Education from 2016-2020**

The bar graphs above show the sum of GDP from all of the winning candidates attributed to each presidential candidate from the years 2016 to 2020. Overall, we see that the GDP has been growing until dropping in 2020, which could have been due to the impact of COVID-19. A noticeable point is that Biden's winning counties, though smaller in number, combined have over doubled the GDPs than that of Trump's winning counties. They suggest that Biden's winning counties are much more prosperous financially and these giant counties have a higher population.

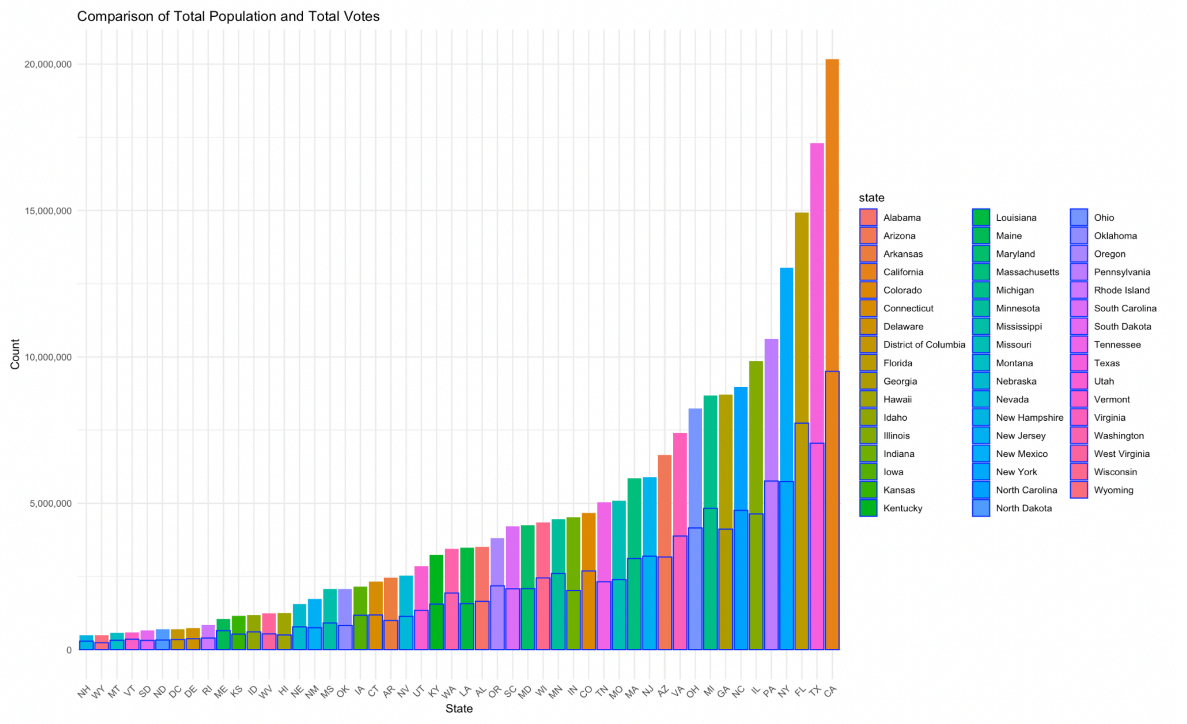
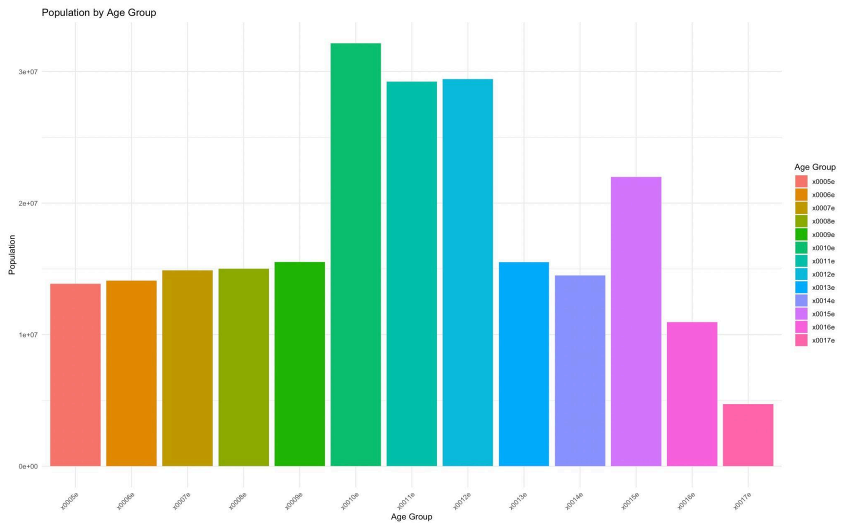
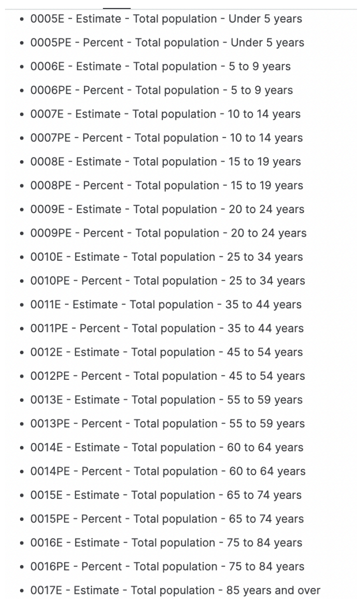


**Figure 10 Total population vs. Total votes**

The graph displays a comparison of total population and total votes across all U.S. states, showing variations in voter turnout relative to population size. The vote counts appear to be around half of the population counts in most states. States like California and Texas, with large populations, exhibit higher vote counts.





- 0005E - Estimate - Total population - Under 5 years
- 0005PE - Percent - Total population - Under 5 years
- 0006E - Estimate - Total population - 5 to 9 years
- 0006PE - Percent - Total population - 5 to 9 years
- 0007E - Estimate - Total population - 10 to 14 years
- 0007PE - Percent - Total population - 10 to 14 years
- 0008E - Estimate - Total population - 15 to 19 years
- 0008PE - Percent - Total population - 15 to 19 years
- 0009E - Estimate - Total population - 20 to 24 years
- 0009PE - Percent - Total population - 20 to 24 years
- 0010E - Estimate - Total population - 25 to 34 years
- 0010PE - Percent - Total population - 25 to 34 years
- 0011E - Estimate - Total population - 35 to 44 years
- 0011PE - Percent - Total population - 35 to 44 years
- 0012E - Estimate - Total population - 45 to 54 years
- 0012PE - Percent - Total population - 45 to 54 years
- 0013E - Estimate - Total population - 55 to 59 years
- 0013PE - Percent - Total population - 55 to 59 years
- 0014E - Estimate - Total population - 60 to 64 years
- 0014PE - Percent - Total population - 60 to 64 years
- 0015E - Estimate - Total population - 65 to 74 years
- 0015PE - Percent - Total population - 65 to 74 years
- 0016E - Estimate - Total population - 75 to 84 years
- 0016PE - Percent - Total population - 75 to 84 years
- 0017E - Estimate - Total population - 85 years and over

**Figure 11 Population by Age Groups**

The graph illustrates the distribution of the U.S. population across various age groups. Notably, the age groups "25 to 54 years" are the most populous, suggesting a mature population. And it also suggests that 25 to 54 years could be contributing the most for the voting.

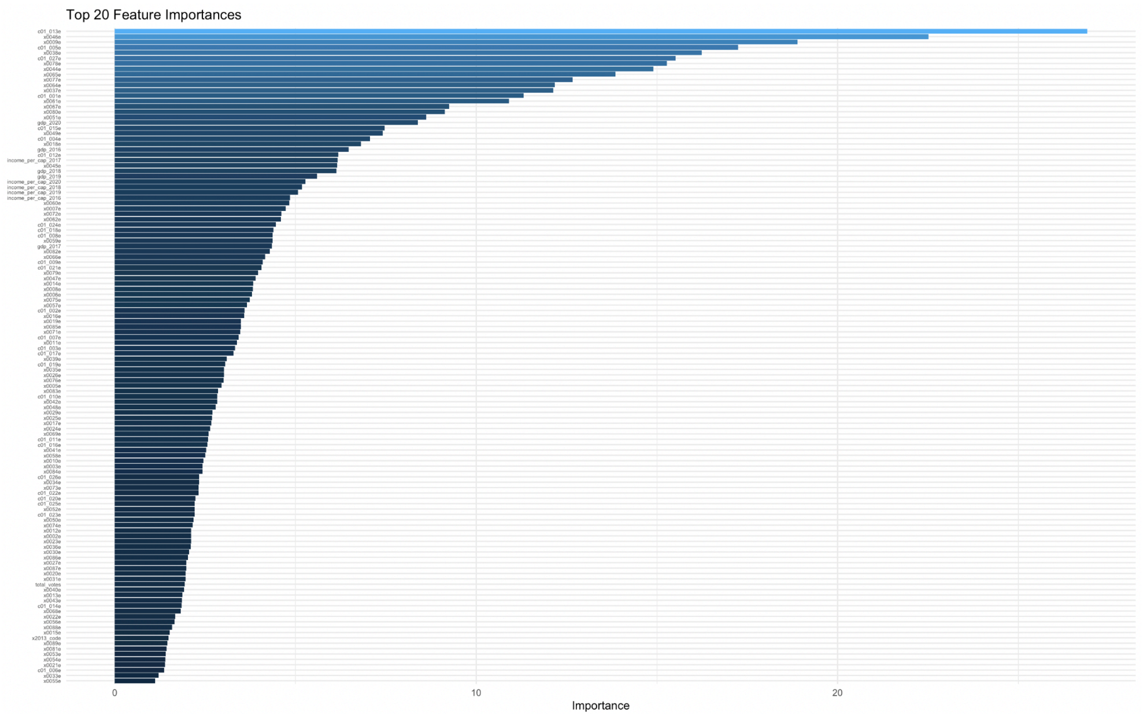


**Figure 12 Predictor Importance for Random Forest Model**

We used the importance function of the Random Forest model to identify and possibly reduce predictors with little importance in terms of statistical importance to the prediction of Winner. Despite removing id and names, and focusing on the other predictors, we get the up 20 predictors: "c01_013e" "x0046e" "x0009e" "c01_005e" "x0038e" "c01_027e" "x0078e" "x0044e" "x0065e" "x0077e" "x0064e" "x0037e" "c01_001e" "x0061e" "x0067e" "x0080e" "x0051e" "gdp_2020" "c01_015e" "x0049e". And, we can see that race is especially important in this model. After running rf bayes tuned model and stacking model, we found out that it actually improves the performance of the model. We decided to keep this method.

## 3. Preprocessing

For preprocessing we first select the appropriate variables and then define a recipe that will be used in modeling workflows.

We attempted to exclude the predictors that are repeated from the training data. After comparing column names in col_description and values in the dataset, we found that the columns -x0033e (Total population), -c01_016e (Estimate:Total:Population 25 to 34 years), -c01_019e (Estimate:Total:Population 35 to 44 years), -x0025e (Total population:18 years and over), -x0029e (Total population:65 years and over), -c01_025e (duplicate of Estimate:Total:Population 65 years and over), -x0036e (Estimate:Total population:One race), -x0058e (Estimate:Total population:Two or more races) each have an identical copy in the dataset. We first thought of removing them from the dataset, but an initial fit on the models indicates that the removal of these columns does not affect the prediction so we kept them. Then we ran the importance function with a random forest model to identify possible significant columns and select our predictors accordingly. As mentioned in previous data analysis, the columns “c01_013e", "x0046e", "x0009e", "c01_005e", "x0038e", "c01_027e", "x0078e", "x0044e", "x0065e", "x0077e", "x0064e", "x0037e", "c01_001e", "x0061e", "x0067e", "x0080e", "x0051e", "gdp_2020", "c01_015e", "x0049e" are identified to be significant in prediction so we keep these columns as our predictors. Our filtered training data contains 21 variables and 2,331 observations.

We defined a recipe that includes step_dummy, step_impute_mean, and step_normalize. Step_dummy converts all categorical predictors into dummy encoded variables. This step is necessary as most machine learning models require numeric inputs. Step_impute_mean imputes missing values in numeric predictors with the mean value of each predictor, which handles the missing values in the data if any. Step_normalize() scales the numeric columns to have a mean of 0 and a standard deviation of 1. This step ensures convergence of the algorithms and allows for equal contribution from all numeric predictors.

## 4. Model Selection

Table 1 Summary of Candidate Models

| Model Identifier | Type of Model | Engine | Recipe Steps | Hyperparameters |
|---|---|---|---|---|
| svm_rbf | Support vector machine with gaussian radial basis function kernel | kernlab | Mean imputation<br>Normalize predictors<br>Dummy encoding | cost<br>rbf_sigma |
| rf_model_1 | Random forest | ranger | Same as svm_rbf | mtry<br>trees<br>min_n |
| rf_model_2 | Random forest | ranger | Same as svm_rbf | mtry<br>trees<br>min_n |
| logistic_model | Logistic Regression | glmnet | Same as svm_rbf | penalty<br>mixture |
| knn_model | K- Nearest Neighbors | kknn | Same as svm_rbf | mixture |
| model_stack | Combination of several model above | / | / | / |

There are six models used in our classification analysis. For each model analysis, the model is first defined and a recipe for data preprocessing is created. The workflow is constructed by combining the model and the recipe. We used bayesian optimization for hyperparameter tuning. A 10 fold cross-validation is performed to identify the best model parameters based on accuracy and roc_auc. After selecting the best model, the workflow is finalized and evaluated through cross-validation. The final model is fitted to the training data, and predictions are made on the test data. Additionally, grid search is performed for random forest tuning.

**SVM RBF Model**
Support vector machine with gaussian radial basis function as the kernel uses nonlinear function of predictors to do classification analysis. It can optimize the loss function and make it only affected by very large model residuals. The two hyperparameters, cost and RBF sigma, in the model (svm_rbf) are tuned through grid search. Cost specifies the cost of predicting a sample within or on the wrong side of the margin. RBF sigma specifies the width of the RBF kernel. The range of cost for tuning is set to (-5, 2), and the range of RBF sigma for tuning is set to (-7, -1).

**Random Forest Model**
Random forest constructs a multitude of decision trees at training time and then combines the outputs into a single result. For this model (rf_model), firstly, three hyperparameters, trees, mtry, and min_m, are tuned through grid search. Trees denotes the number of trees contained in the model; mtry specifies the number of predictors that will be randomly sampled at each split when creating the tree models; min_n specifies the minimum number of data points in a node that are required for the node to be split further. The range of trees for tuning is set to (50, 200), the range of mtry for tuning is set to (1, 5), and the range of min_n for tuning is set to (5, 20). The model has relatively good accuracy in predicting test data.

Beside grid search (grid_random), we also tried grid_regular and bayes tuning method. For bayes tuning, we set the initial to be 10 and the iteration to be 20. We fixed the trees to 1000, and set the range of mtry to be (1, ) and the range of min_n to be (1, 10). After applying prediction to the model with the highest accuracy, we find out that there is not much difference in accuracy between different tuning methods for our random forest model. However, when we are comparing the accuracy and roc_auc of other models, random forest is one of the models with the highest accuracy and highest roc_auc. Therefore, we decided to use this model as one of our final models.

**Logistic Regression**
Logistic regression is a supervised machine learning algorithm widely used for binary classification tasks, which utilizes the logistic function to transform a linear combination of input features into a probability value ranging between 0 and 1. This probability indicates the likelihood that a given input corresponds to one of the two outcomes. Logistic regression has two hyperparameters, penalty and mixture. Penalty represents the total amount of regularization and mixture represents the proportion of L1 regularization (lasso) in the model. The hyperparameters are tuned using grid search. The range of penalty is set to (-6, -1) and the range of mixture is between (0, 1).

**k-nearest neighbors**

k-nearest neighbors algorithm (k-NN) is a non-parametric supervised learning method. k-NN performs a type of classification by calculating the distance between the query point and all points in the training data, selecting the k closest data points, and assigning the most common class label among these neighbors to the query point. For the model (KNN), we implemented the kknn engine for classification and used bayes tuning. The hyperparameter involved in tuning is neighbors, which specifies the number of nearest neighbors that need to be considered to assign an object to the class. The range of neighbors is set to (1, 30). KNN does give out the highest accuracy among all models. However, it doesn't outperform the random forest model when submitted to kaggle, suggesting possible overfitting in KNN.

**Stacking Model**

We use tidy model stacks to ensemble the output of several models together and generate a new model to better fit the data. In our stacking model, we use two random forest models, svm rbf model and logistic regression model as candidate members to predict the output. As we can see from the table, the accuracy for model stacking is much higher than the others. Therefore, we select the stacking model as one of our final models.

**Summary table**

We evaluated each model with fit_resamples and set the metric to accuracy. This function fits the model to each resample (our cross-validation folds) and computes performance metrics, allowing us to assess the model's generalization ability. The accuracy and standard error of each model are extracted and compared. The details are shown in the table below.

Table 3 Summary of Accuracy and Standard Error of Candidate Models

| Model Identifier | Metric (Accuracy) | SE of metric |
|---|---|---|
| random_forest_1 | 0.913 | 0.00383 |
| random_forest_2 | 0.913 | 0.00373 |
| SVM-rbf | 0.888 | 0.00580 |
| KNN | 0.915 | 0.00451 |
| Logistic | 0.881 | 0.00572 |
| Stack_model | 0.994 | NA |

## 5. Final Model

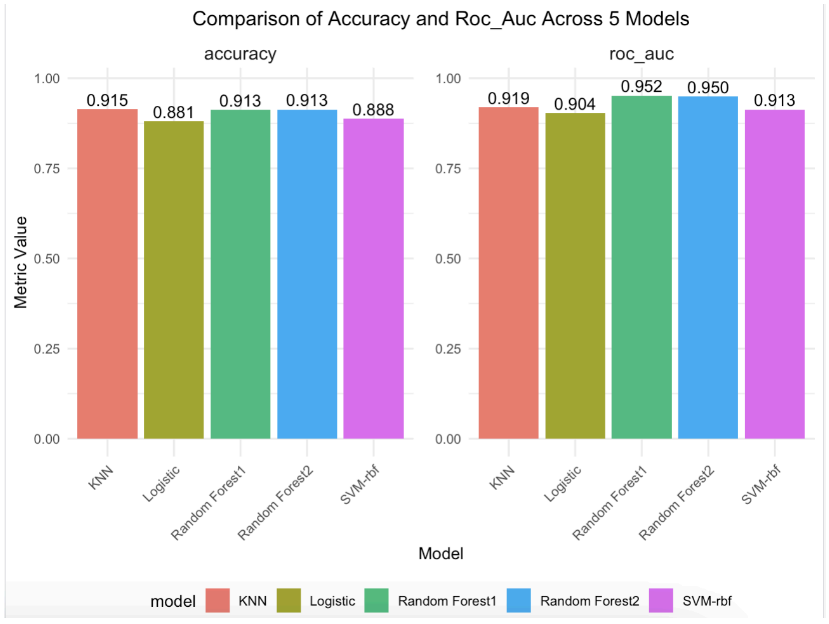


**Figure 13 Comparison of Accuracy and Roc-auc Across 5 Models**

According to the public scoreboard, our best performing models are the Bayesian random forest and stacking model where the random forest model is heavily weighted. The stacking model achieved an accuracy of 0.95. This accuracy is lower than the training accuracy of 0.994 so we suspect there is overfitting of the data. From figure 10 we see that the two random forest models with bayesian tuning and grid tuning and KNN with bayesian tuning have the highest accuracy and roc_auc scores. However, KNN model performed poorly on the test dataset compared to the training data, suggesting that it might have overfitted the data. Conversely, the Bayesian random forest model had a better accuracy on the test dataset than the training dataset, showing its ability to generalize well to unseen data.

Our final model selection, taking into account potential overfitting to the dataset, would be the Random forest model (without stacking) with a public score of 0.9456. Random forest model is a powerful algorithm for binary classification as the nature of the model allows for careful learning of data by the construction of multiple decision trees, each trained on different subsets of the data and features. This approach reduces overfitting and increases generalization by averaging the predictions of all trees, guaranteeing accurate classification. However, with a small dataset of 2,331 observations, there is still possible overfitting over the training data. We address this issue by excluding the best performing model on the training dataset (KNN) and selecting a model that does not have the best performance score in the public score.

Some potential improvements to our model building include more effective preprocessing techniques since our current preprocessing shows no significant improvement compared to fitting the tuned model on the original dataset. Our model is also prone to overfitting due to the complexity of the parameters and insufficient training data. Lastly, our model is based primarily on the train dataset, where there is an imbalance between counts of Trump as the winner and counts of Biden as the winner. This could lead to overfitting the training data to produce a biased model that favors Trump as the predicted outcome.

---

## 6. Appendix: Final annotated script & Team member contribution

```
#Using Mac to run the code

#Model1
# load all required packages
library(ROSE)
library(ISLR)
library(caret)
library(kernlab)
library(tidymodels)
library(tidyverse)
library(tune)
library(dials)
library(ranger)
library(workflows)
library(rsample)
library(yardstick)
library(kknn)
library(lattice)
library(stacks)
tidymodels_prefer()

# load the original training data
train <- read_csv('train_class.csv',show_col_types = FALSE)

# delete the id, name and total_population to remove non-predictors and ovoid overfitting.
train <- train %>%
  select(-id, -name, -x0001e)

#create recipe from training data
vote_recipe1 <- recipe(winner ~ . , data = train) %>%
  step_impute_mean(all_numeric_predictors(), -all_outcomes()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_dummy(all_nominal(), -all_outcomes())

# set engine and mode for rain forest
rf_initial_model <- rand_forest(trees = 100) %>%
  set_engine('ranger', importance = 'impurity') %>%
  set_mode('classification')

# Workflow to fit initial model
initial_rf_wkfl <- workflow() %>%
  add_model(rf_initial_model) %>%
  add_recipe(vote_recipe1)

# Fit model to get feature importance
set.seed(100)
initial_fit <- fit(initial_rf_wkfl, data = train)

# Extracting feature importance
importance_df <- vip::vi(initial_fit, method = "model")
# Check the structure of importance_df
print(str(importance_df))

#select the top 20 important predictors
top_features <- importance_df$Variable[1:20]

# select only the response variable and top 20 predictors
train <- train %>%
  select(winner, all_of(top_features))

#load the test data
test <- read_csv('test_class.csv')

# select only the id column and the top 20 predictors from training
test <- test %>%
  select(id, all_of(top_features))

# set up cvfolds with v = 10
set.seed(100)
folds <- vfold_cv(train,v=10)

# Update recipe to include only the top features
vote_recipe <- recipe(winner ~ . , data = train) %>%
  step_impute_mean(all_numeric_predictors(), -all_outcomes()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_dummy(all_nominal(), -all_outcomes())

#create the new model with ranger and classification
rf_model1 <- rand_forest() %>%
  set_engine('ranger')%>%
  set_mode('classification')


# Update the workflow with the refined recipe
rf_wkfl1 <- workflow() %>%
  add_model(rf_model1)%>%
  add_recipe(vote_recipe)

#Convert

#update random forest model with mtry, min_m and trees
rf_model1 <- rand_forest(
  mtry = tune(),
  min_n = tune(),
  trees = 1000
) %>%
  set_engine("ranger") %>%
  set_mode("classification")

#update workflow with tuning model
rf_tune_wkfl1 <- rf_wkfl1 %>%
  update_model(rf_model1)

#set up tuning parameters
rf_params1 <- parameters(
  mtry(range = c(1, ncol(train) - 1)),
  min_n(range = c(1, 10))
)

# Set up Bayesian Optimization
bayes_opt1 <- tune_bayes(
  rf_model1,
  vote_recipe,
  resamples = vfold_cv(train, v = 5),
  param_info = rf_params1,
  initial = 10,
  iter = 20,
  metrics = metric_set(roc_auc, accuracy)
)
#check the model results
bayes_opt1 %>%
  collect_metrics()

#select the best model based on their roc_auc results
best_params1 <- select_best(bayes_opt1, metric = "roc_auc")

#create final workflow with the best model
final_rf_wkfl1 <- rf_tune_wkfl1 %>%
  finalize_workflow(best_params1)
final_rf_wkfl1

#resample the model with the newest workflow and folds
set.seed(100)
final_rf_res1 <- fit_resamples(
  final_rf_wkfl1,
  resamples =folds,
  metrics = metric_set(accuracy,roc_auc),control=control_resamples(save_workflow = TRUE)
)

collect_metrics(final_rf_res1)

# fit the training data to the workflow
final_rf_fit1 <-final_rf_wkfl1 %>%
  fit(data=train)

#make predictions based on the model
rf_predictions1 <- final_rf_fit1 %>%
  predict(new_data=test)

#create results table
results_rf1 <- test %>%
  select(id) %>%
  bind_cols(rf_predictions1)%>%
  rename(id = id, winner = .pred_class)

head(results_rf1,15)

#output results
write_csv(results_rf1,'rf_bayes_class_original_top20_final.csv')
```

**#Model2**

```
# load all required packages
library(ROSE)
library(ISLR)
library(caret)
library(kernlab)
library(tidymodels)
library(tidyverse)
library(tune)
library(dials)
library(ranger)
library(workflows)
library(rsample)
library(yardstick)
library(kknn)
library(lattice)
library(stacks)
tidymodels_prefer()

# load the original training data
train <- read_csv('train_class.csv',show_col_types = FALSE)

# delete the id, name and total_population to remove non-predictors and ovoid overfitting.
train <- train %>%
  select(-id, -name, -x0001e)

#create recipe from training data
vote_recipe1 <- recipe(winner ~ . , data = train) %>%
  step_impute_mean(all_numeric_predictors(), -all_outcomes()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_dummy(all_nominal(), -all_outcomes())

# set engine and mode for rain forest
rf_initial_model <- rand_forest(trees = 100) %>%
  set_engine('ranger', importance = 'impurity') %>%
  set_mode('classification')

# Workflow to fit initial model
initial_rf_wkfl <- workflow() %>%
  add_model(rf_initial_model) %>%
  add_recipe(vote_recipe1)

# Fit model to get feature importance
set.seed(100)
```

```
initial_fit <- fit(initial_rf_wkfl, data = train)

# Extracting feature importance
importance_df <- vip::vi(initial_fit, method = "model")
# Check the structure of importance_df
print(str(importance_df))

#select the top 20 important predictors
top_features <- importance_df$Variable[1:20]

# select only the response variable and top 20 predictors
train <- train %>%
  select(winner, all_of(top_features))

#load the test data
test <- read_csv('test_class.csv')

# select only the id column and the top 20 predictors from training
test <- test %>%
  select(id, all_of(top_features))

# set up cvfolds with v = 10
set.seed(100)
folds <- vfold_cv(train,v=10, repeats = 3)

# Set control for tuning
ctrl_grid <- control_stack_grid()

# Set up resampling
set.seed(100)
cv_folds <- vfold_cv(train, v = 10, repeats = 3)

# Update recipe to include only the top features
vote_recipe <- recipe(winner ~ . , data = train) %>%
  step_impute_mean(all_numeric_predictors(), -all_outcomes()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_dummy(all_nominal(), -all_outcomes())

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

```
# Update the workflow with the refined recipe
rf_wkfl <- workflow() %>%
  add_model(rf_model)%>%
  add_recipe(vote_recipe)

#update model with tuning parameters mtry, min_n, trees
rf_tune_model <- rand_forest(
  mtry = tune(),     # Indicate tuning
  min_n = tune(),    # Indicate tuning
  trees = 1000       # Fixed number of trees
) %>%
  set_engine("ranger") %>%
  set_mode("classification")

#update workflow with tuning model
rf_tune_wkfl <- rf_wkfl %>%
  update_model(rf_tune_model)

#set up tuning parameters
rf_params <- parameters(
  mtry(range = c(1, ncol(train) - 1)),
  min_n(range = c(1, 10))
)

# Set up Bayesian Optimization
bayes_opt <- tune_bayes(
  rf_tune_model,
  vote_recipe,
  resamples = vfold_cv(train, v = 5),
  param_info = rf_params,
  initial = 10,  # Number of randomly selected points to evaluate before the Bayesian optimization starts
  iter = 20,     # Number of iterations of Bayesian optimization
  metrics = metric_set(roc_auc, accuracy)
)

bayes_opt %>%
  collect_metrics()

#select the best model based on roc_auc
best_params <- select_best(bayes_opt, metric = "roc_auc")

#update workflow with the selected model
final_rf_wkfl <- rf_tune_wkfl %>%
  finalize_workflow(best_params)
```

```
final_rf_wkfl

# resample the final workflow with folds
set.seed(100)
folds <- vfold_cv(train,v=10)

final_rf_res <- fit_resamples(
  final_rf_wkfl,
  resamples =folds,
  metrics = metric_set(accuracy,roc_auc),control=control_resamples(save_workflow = TRUE)
)

## random forest 2

#set up second random forest model
rf_model2 <- rand_forest() %>%
  set_engine('ranger')%>%
  set_mode('classification')

# Update the workflow with the refined recipe
rf_wkfl2 <- workflow() %>%
  add_model(rf_model2)%>%
  add_recipe(vote_recipe)

#update the second model with tuning parameters mtry, trees, min_n
rf_tune_model2 <- rand_forest(
  mtry = tune(),
  trees = tune(),
  min_n = tune()
) %>%
  set_engine('ranger')%>%
  set_mode('classification')

#update the workflow with tuning model
rf_tune_wkfl2 <- rf_wkfl2 %>%
  update_model(rf_tune_model2)

#set up tuning parameters
rf_grid2 <- grid_latin_hypercube(
  mtry(range = c(1, 5)),
  trees(range = c(50, 200)),
  min_n(range = c(1, 10)),
  size = 20
)
```

```
#apply grid tuning
rf_tuning2 <- tune_grid(
  rf_tune_wkfl2,
  resamples = folds,
  grid = rf_grid2,
  metrics = metric_set(accuracy, roc_auc)
)

rf_tuning2 %>%
  collect_metrics()

#select the best model with roc_auc results
best_rf_model2 <- rf_tuning2 %>%
  select_best(metric = 'roc_auc')

#update the final workflow with the best model
final_rf_wkfl2 <- rf_tune_wkfl2 %>%
  finalize_workflow(best_rf_model2)

#resample the data with the final workflow and folds
set.seed(100)
final_rf_res2 <- fit_resamples(
  final_rf_wkfl2,
  resamples =folds,
  metrics = metric_set(accuracy,roc_auc),control=control_resamples(save_workflow = TRUE)
)


## Model 3 SVM-rbf with bayes optimization
# Define the SVM model with tuning parameters
svm_rbf_model <- svm_rbf() %>%
  set_engine("kernlab") %>%
  set_mode('classification') %>%
  set_args(cost = tune(), rbf_sigma = tune())

# Define the recipe
vote_recipe_svm <- recipe(winner ~ ., data = train) %>%
  step_impute_mean(all_numeric(), -all_outcomes()) %>%
  step_normalize(all_numeric()) %>%
  step_nzv(all_predictors())

# Define the workflow
```

```
svm_rbf_wkfl <- workflow() %>%
  add_model(svm_rbf_model) %>%
  add_recipe(vote_recipe_svm)

# Define the parameter set
svm_param <- extract_parameter_set_dials(svm_rbf_wkfl) %>%
  update(
    rbf_sigma = rbf_sigma(range = c(-7, -1)),
    cost = cost(range = c(-5, 2))
  )

# Define the initial grid using Latin hypercube
svm_grid <- grid_latin_hypercube(
  svm_param,
  size = 30
)

# Set seed for reproducibility
set.seed(100)

# Initial tuning with grid search
svm_initial <- tune_grid(
  svm_rbf_wkfl,
  resamples = folds,
  grid = svm_grid,
  metrics = metric_set(accuracy, roc_auc)
)

# Collect initial metrics
collect_metrics(svm_initial)

# Bayesian optimization control settings
ctrl <- control_bayes(verbose = FALSE)

# Bayesian optimization
set.seed(100)
svm_bo <- tune_bayes(
  svm_rbf_wkfl,
  resamples = folds,
  metrics = metric_set(accuracy, roc_auc),
  initial = svm_initial,
  param_info = svm_param,
  iter = 25,
  control = ctrl
```

```
)

# Display best results
svm_bo %>% show_best(metric = 'roc_auc', n = 5)

# Select the best model from Bayesian optimization
best_bayes <- svm_bo %>% select_best(metric = "roc_auc")

# Finalize the workflow with the best parameters
final_svm_rbf_wkfl <- finalize_workflow(svm_rbf_wkfl, best_bayes)

# Perform resampling with the final workflow
set.seed(100)

final_svm_rbf_res <- fit_resamples(
  final_svm_rbf_wkfl,
  resamples = folds,
  metrics = metric_set(accuracy, roc_auc),
  control=control_stack_resamples()
)

# Collect final resampling metrics
collect_metrics(final_svm_rbf_res)

## logistic model
# Define the logistic regression model with glmnet, suitable for tuning
logistic_tuned <- logistic_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet") %>%
  set_mode("classification")


# Combine the recipe and model into a workflow
workflow_tuned <- workflow() %>%
  add_model(logistic_tuned) %>%
  add_recipe(vote_recipe)

# Define a grid of hyperparameters
penalty_vals <- penalty(range = c(-6, -1), trans = log10_trans()) # Log transformation
mixture_vals <- mixture()

# Create a regular grid
tuning_grid <- grid_regular(
  penalty_vals,
  mixture_vals,
  levels = 10
)

tune_results <- tune_grid(
  workflow_tuned,
  resamples = folds,
  grid = tuning_grid,
  metrics = metric_set(roc_auc, accuracy)
)

tune_results %>% collect_metrics()

# Select the best model based on ROC AUC
best_model3 <- select_best(tune_results, metric = "roc_auc")

# Finalize the workflow with the best model
final_workflow <- finalize_workflow(workflow_tuned, best_model3)

# Fit the finalized workflow to the resampled training data
set.seed(100)  # Ensure reproducibility
final_results <- fit_resamples(
  final_workflow,
  resamples = cv_folds,
  metrics = metric_set(roc_auc, accuracy),
  control=control_resamples(save_workflow = TRUE)
)


# stacking
#set up for model stacking
stack_control <- control_resamples(
  save_pred = TRUE,
  save_workflow = TRUE
)

set.seed(100)
# Random Forest Model with bayes tuning resample
rf_res1 <- fit_resamples(
  final_rf_wkfl,
  resamples =folds,
  metrics = metric_set(roc_auc),
  control= stack_control
)
#random forest model with grid tuning resample
```

```
rf_res2 <- fit_resamples(
  final_rf_wkfl2,
  resamples =folds,
  metrics = metric_set(roc_auc),
  control=stack_control
)
#svm with bayes tuning resample
svm_res <- fit_resamples(
  final_svm_rbf_wkfl,
  resamples = folds,
  metrics = metric_set(roc_auc),
  control=stack_control
)
#logistics model resample
log_res <- fit_resamples(
  final_workflow,
  resamples = folds,
  metrics = metric_set(roc_auc),
  control=stack_control
)


# Stack the models
stack <- stacks() %>%
  add_candidates(rf_res1) %>%
  add_candidates(rf_res2) %>%
  add_candidates(log_res) %>%
  add_candidates(svm_res)

# Blend predictions
stack <- stack %>%
  blend_predictions() %>%
  fit_members()

#check the model by predicting the training data
train_predictions <- predict(stack, new_data = train)
results_df <- train %>%
  bind_cols(train_predictions)

results_df$winner <- as.factor(results_df$winner)
results_df$.pred_class <- as.factor(results_df$.pred_class)

# Calculate accuracy
accuracy_result <- accuracy(data = results_df, truth = "winner", estimate = ".pred_class")
```

```
# Print accuracy
print(accuracy_result)

# predict the test dataset
new_prediction <- predict(stack, new_data = test)

results_stack <- test %>%
  select(id) %>%
  bind_cols(new_prediction)

head(results_stack)

#output results
write_csv(results_stack,'prediction_stack_bayes20_final.csv')
```

**Team Member Contribution:**

Yechen Cao: I am responsible for some of the visualizations, and improving the random forest model with different tuning methods and stacking models. I also writing reports for those parts.

Vieno Wu: I am responsible for result visualization and discussion.I am responsible for some of the data preprocessing and visualization. I also wrote the corresponding parts in this report.

Pinyi Li: I am responsible for some visualizations, and writing for the linear regression and svm rbf model. I also write reports for the models.

Jingran Zhang: I am responsible for renaming columns, data reshaping, visualizations, and model testing. I am also responsible for the visualizations part of the report writing.