Statistics 101C - Kaggle Classification Project

Summer 2024

By: Albert Putranegoro, James Joyce, Benedetta Gasbarra, Mikayla Silverman,
Kyle Alexander Slaughter

Introduction:

This report presents a comprehensive analysis of voter behavior during the 2020 election, focusing on various demographic and geographic factors. The study utilizes data by the US Census Bureau, covering a range of variables such as education level, age, county size, and urban/rural classification. The dataset includes a total of 3,111 counties or county equivalents, with 75% (2,331 rows) allocated to training data and the remaining 780 rows to test data.

The primary objective is to predict the winner of each county in the 2020 US Presidential Election — either Biden or Trump — using demographic and education information for each county. Notably, Alaska has been excluded from the dataset because votes in Alaska are tabulated in 41 districts, while census information is recorded in 30 boroughs or census areas, making it difficult to match demographic data with voting districts.

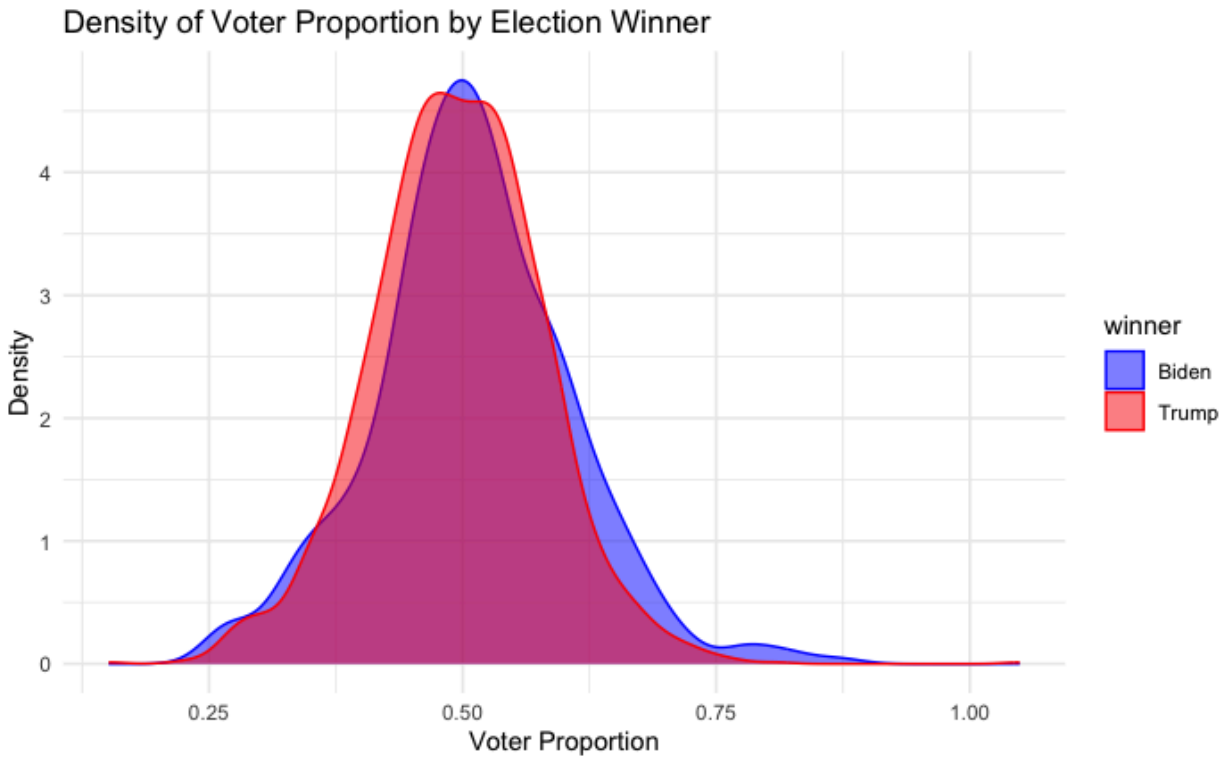
In the subsequent sections, we will delve deeper into the dataset to explore its distributions and interactions among variables. This exploratory analysis aims to uncover patterns, identify anomalies, and provide insights into key characteristics of the data. By applying regression models, we aim to uncover the relationships between these demographic and geographic characteristics and their influence on the election outcomes.

Throughout this report, we will discuss the strengths and weaknesses of each model, the tuning process, and the final selection for this classification task. These findings will aim to provide a deeper understanding of the demographic and socio-economic factors that shaped the 2020 election, highlighting the complex interplay between voter characteristics and election outcomes.

Exploratory analysis:

In this section, we conduct an exploratory analysis of the dataset to examine its key characteristics, uncover patterns, identify anomalies, and provide insights for subsequent analysis.

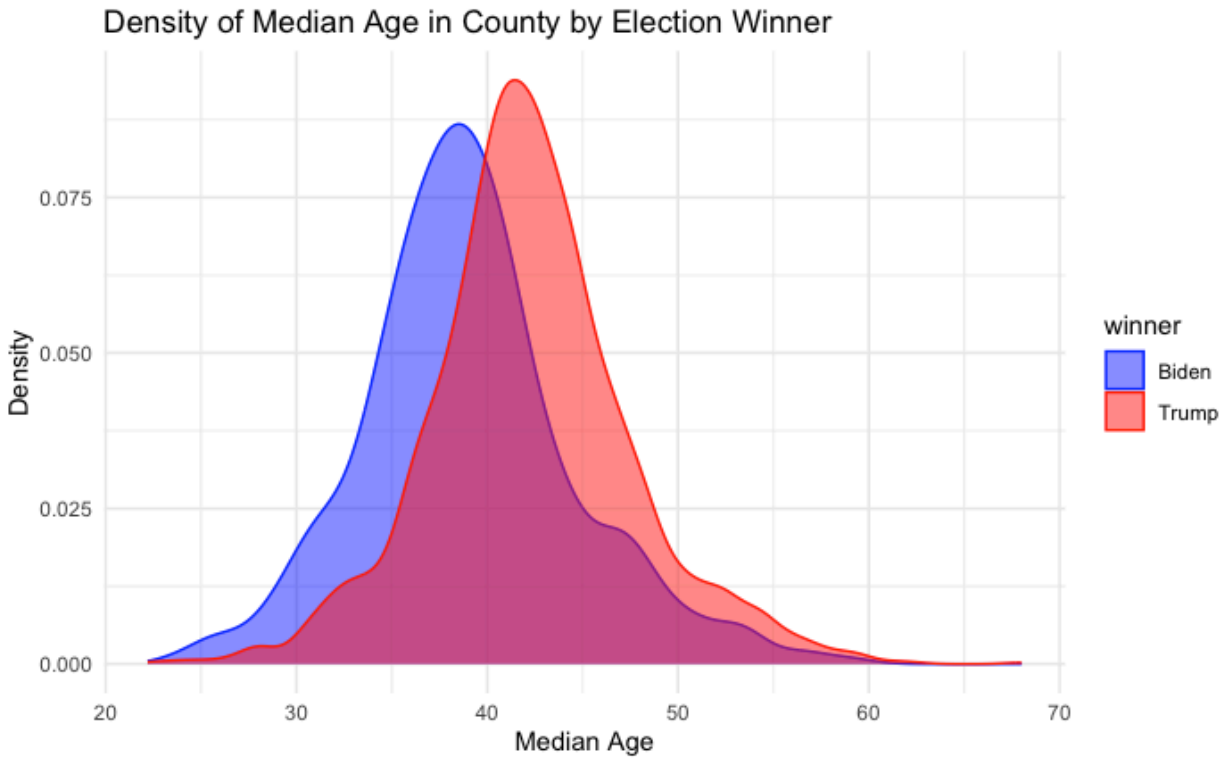
Graph 1:



Description:

Graph 1 displays the density of voter proportion by election winner. This density plot shows the distribution of voter proportions in counties, differentiated by the winning candidate, Biden or Trump. The blue curve represents counties won by Biden, while the red curve represents counties won by Trump. The central peak of around 0.5 voter proportion indicates that many counties had a relatively balanced voter split, with a slight skew towards Trump as his red curve shows a higher density at the peak. The overlap between the curves suggests that there were many counties with similar voter proportions for both candidates, reflecting a competitive election landscape.

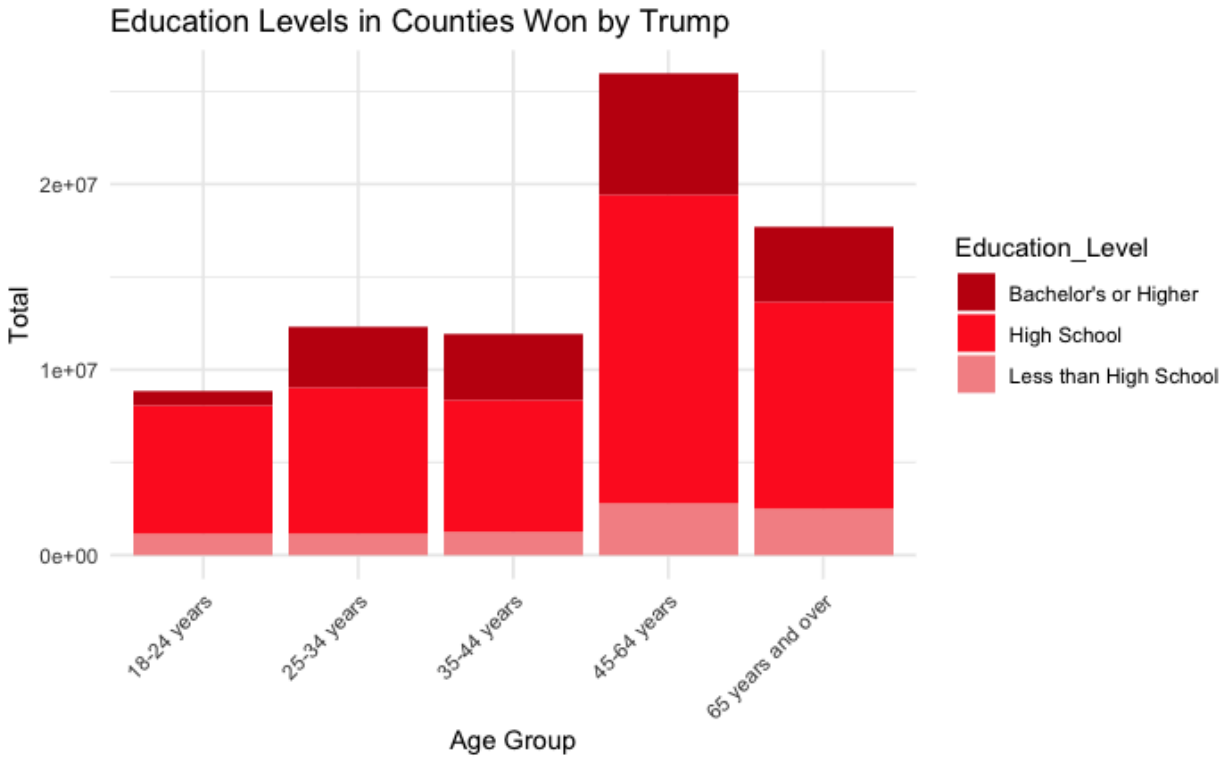
Graph 2:



Description:

Graph 2 shows the median age density in counties by the election winner. This density plot illustrates the distribution of median ages in counties, separated by the winning candidate (Biden or Trump). The blue curve represents counties won by Biden, while the red curve represents counties won by Trump. This graph shows that counties with a median age of around 40-45 tend to favor Trump, as indicated by the peak of the red curve. Conversely, Biden's support appears stronger in counties with a median age of around 35-40, as shown by the peak of the blue curve. The overlap between the two curves indicates areas where both candidates had similar median age demographics.

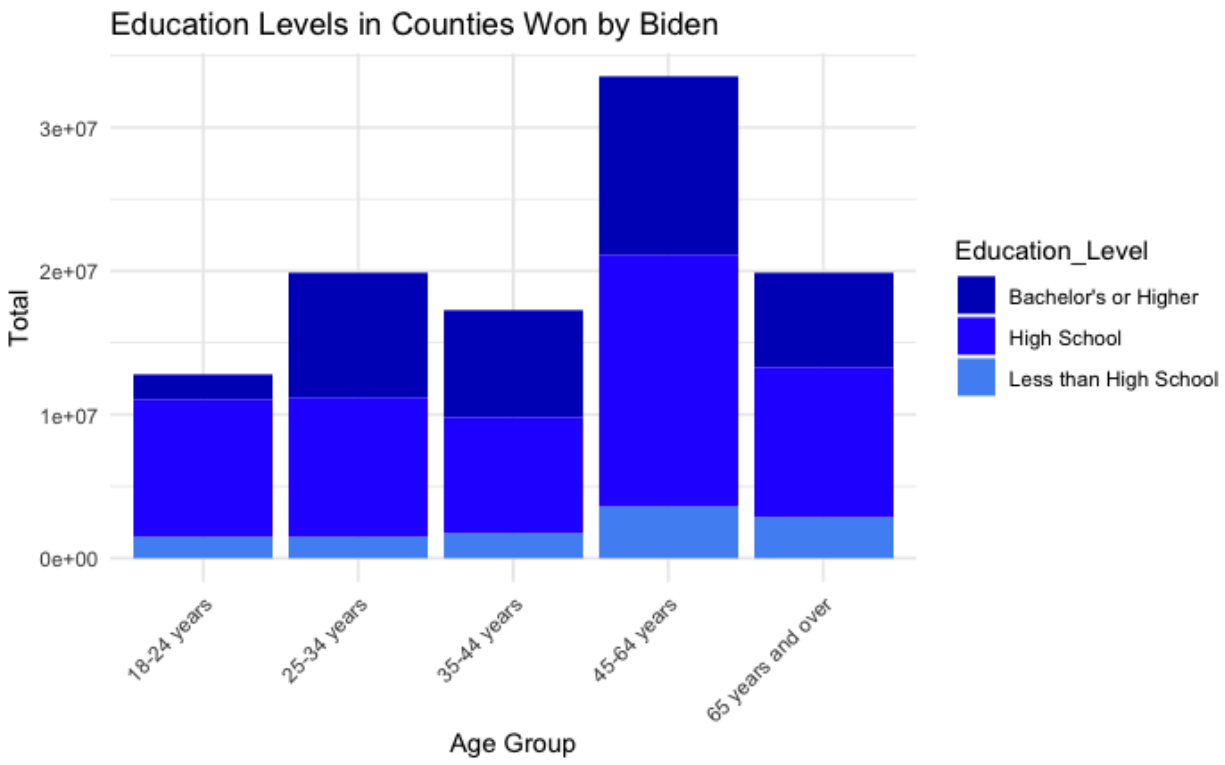
Graph 3:



Description:

Graph 3 presents the education levels within different age groups in counties won by Trump. The x-axis represents age groups, while the y-axis shows the total number of individuals within each education level. Notably, the 45-64 age group has the highest total, with a significant portion having only a high school education. In contrast, younger age groups (18-24) have fewer individuals across all education levels than older groups.

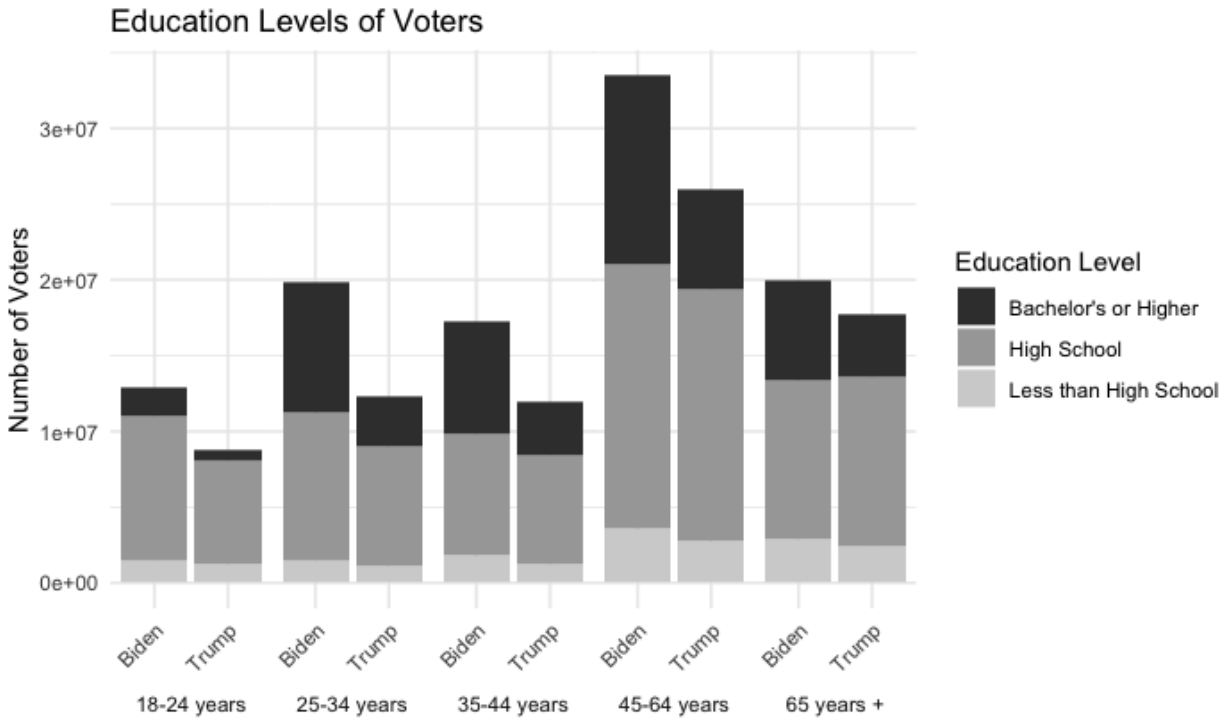
Graph 4:



Description:

Graph 4 illustrates the education levels within different age groups in counties won by Biden. The x-axis represents age groups, and the y-axis shows the total number of individuals within each education level. Similar to Trump's counties, the 45-64 age group has the highest total, but a notable portion holds a bachelor's degree or higher. This suggests that Biden's counties have more educated individuals than Trump's.

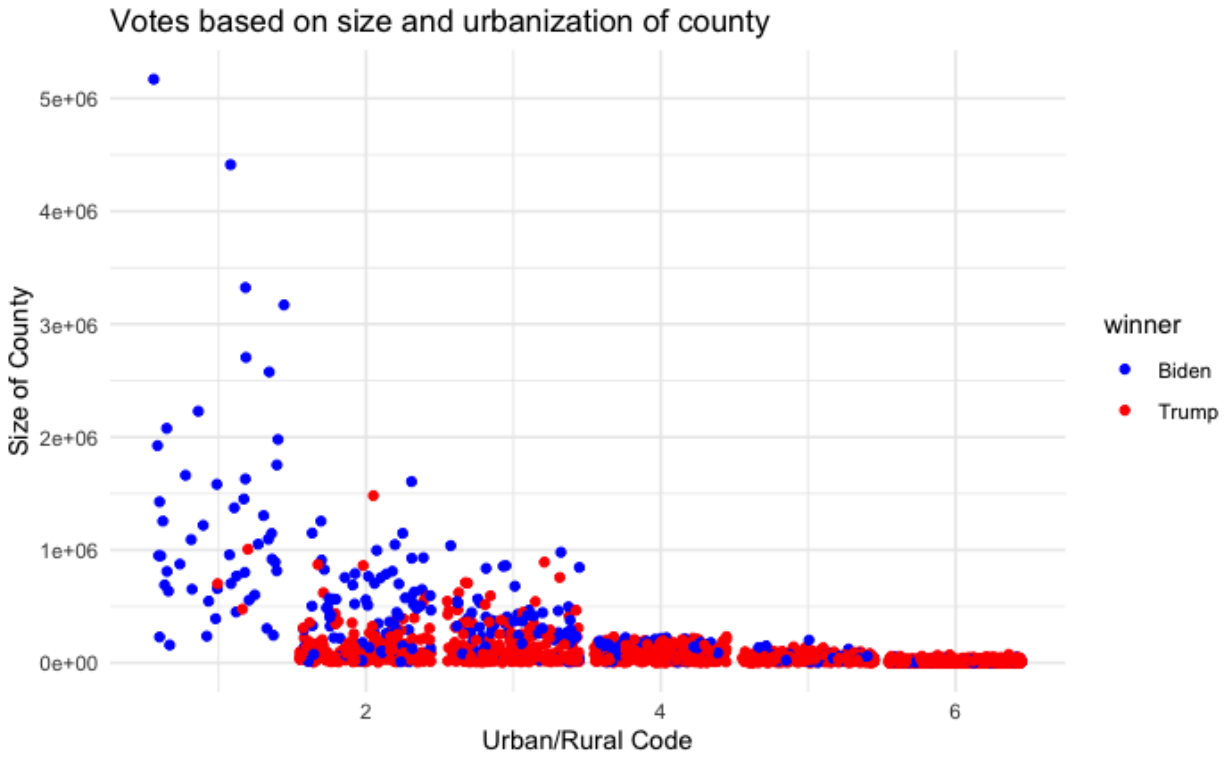
Graph 5:



Description:

Graph 5 compares the education levels of voters in counties that voted for Biden and Trump, separated by age groups. The x-axis represents the different age groups, and the y-axis shows the number of voters. In counties won by Biden, there were more voters with a bachelor's degree or higher in every age group. Conversely, Trump counties have more voters with only a high school education, especially in older age groups (45-64 years).

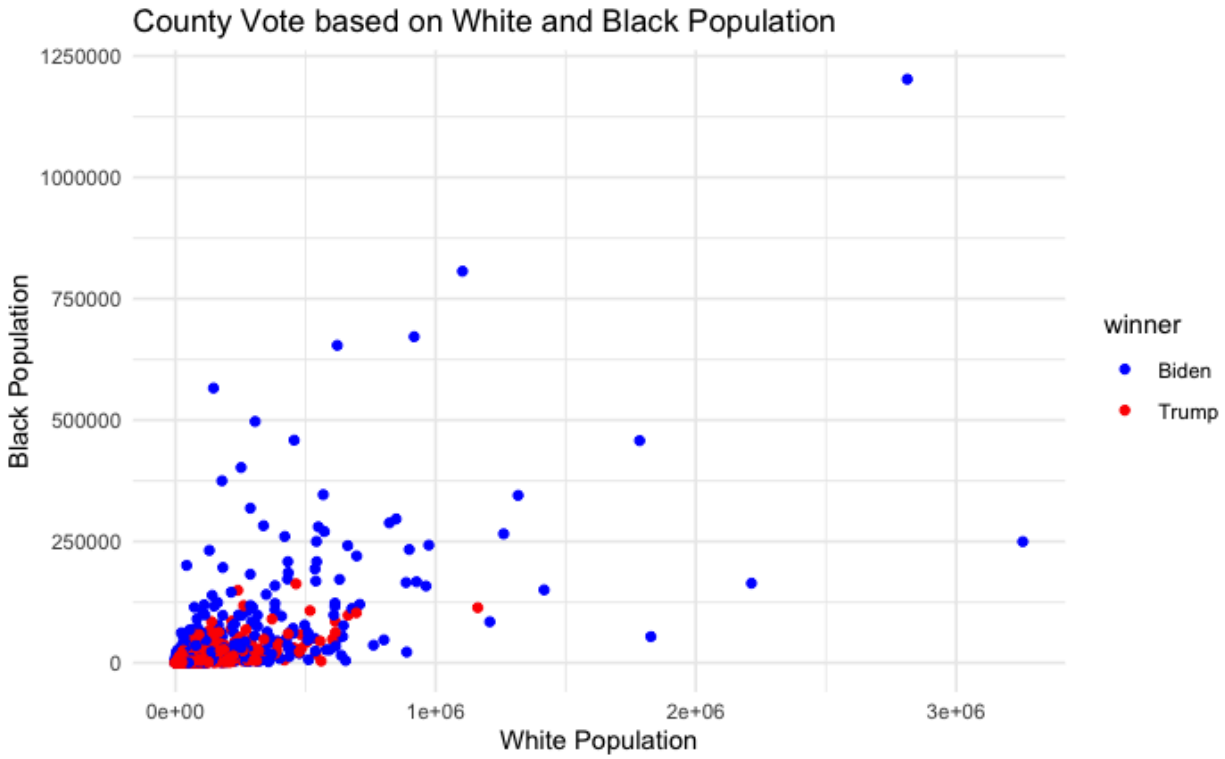
Graph 6:



Description:

Graph 6 is a scatter plot showing the relationship between county size and urban/rural code, indicating the winning candidate. The y-axis represents the size of the county, and the x-axis represents the urban/rural code, with a score of 1 indicating a large central metropolitan area and 6 representing rural areas. Blue dots represent counties won by Biden, and red dots represent counties won by Trump. The graph reveals that Biden won more populous and urban counties, while Trump won more rural and less populated counties.The graph shows that Biden won more often in larger counties with lower urban/rural codes, while Trump won more often in less populated counties with higher urban/rural codes.

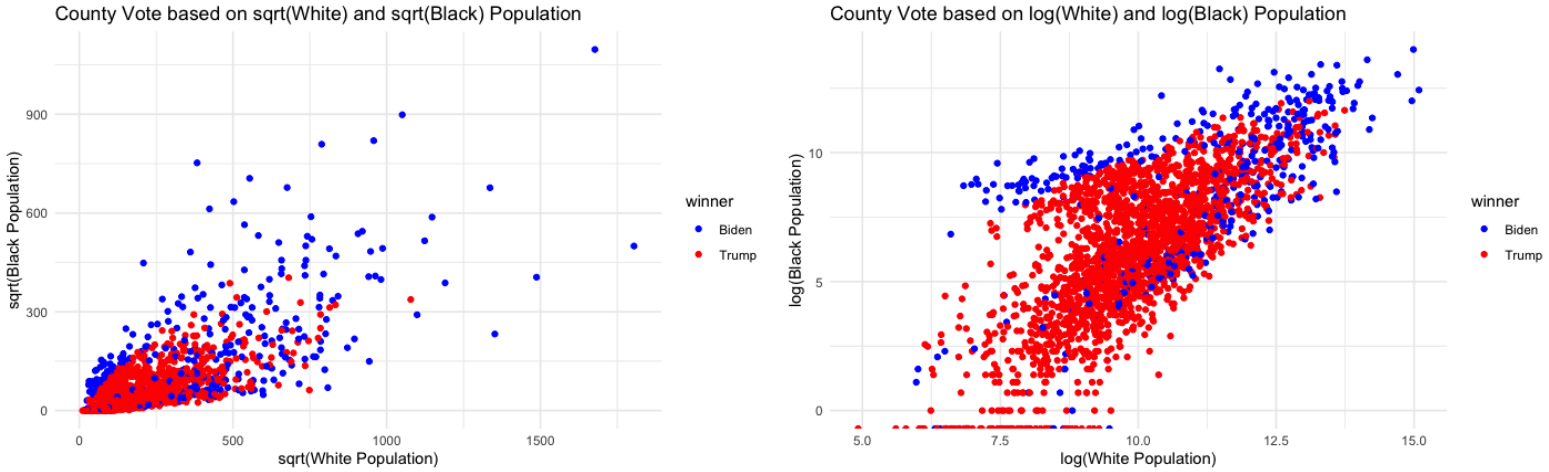
Graph 7:



Description:

***Graph* 7** is a scatter plot that displays the relationship between county vote and the white and black populations. The x-axis represents the white population, and the y-axis represents the black population. Blue dots represent counties won by Biden, and red dots represent counties won by Trump. This graph shows that counties with higher black populations tend to be won by Biden, while counties with higher white populations tend to be won by Trump.

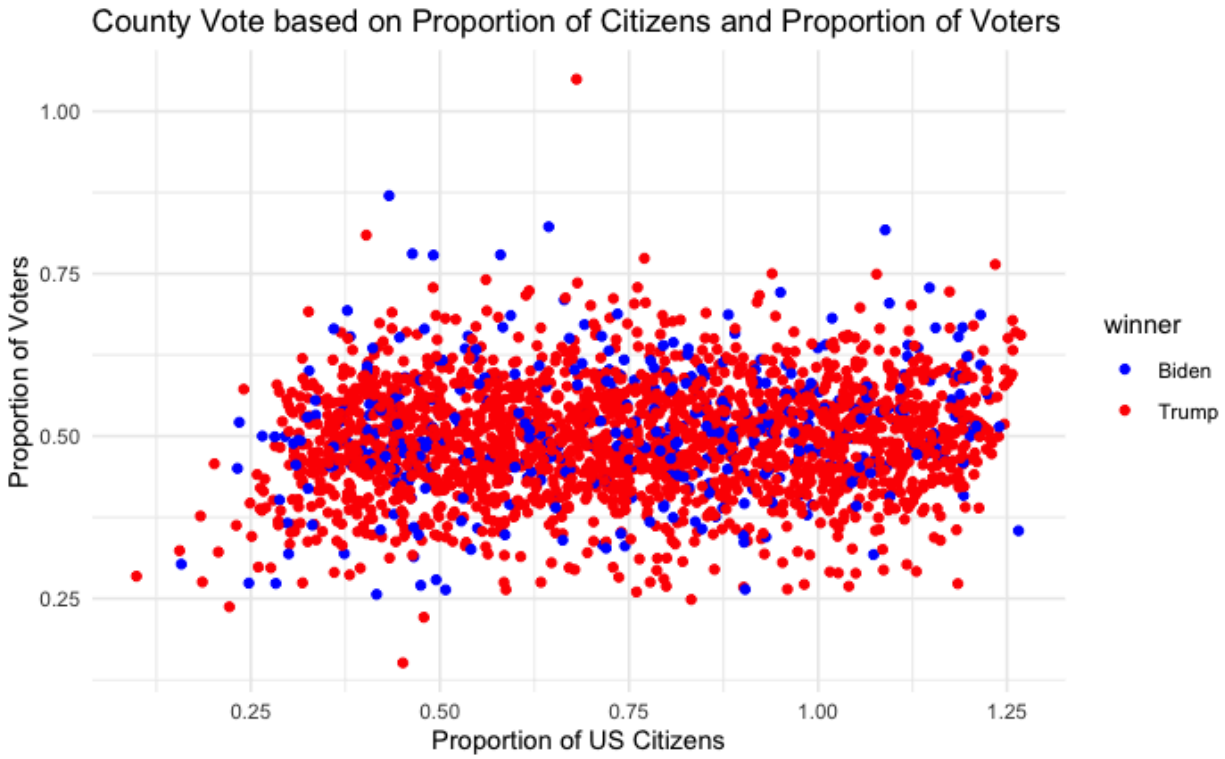
Graph 8-9:



Description:

Graphs 8-9 represent scatter plots showing the relationship between county vote and the log-transformed and square root transformed white and black populations. The top plot uses a logarithmic scale for both white and black populations, while the bottom plot employs a square root transformation. Blue dots represent counties won by Biden, and red dots represent counties won by Trump. These transformations highlight the distribution more clearly, showing a positive correlation between higher black population and Biden's wins, and lower populations with Trump's wins.

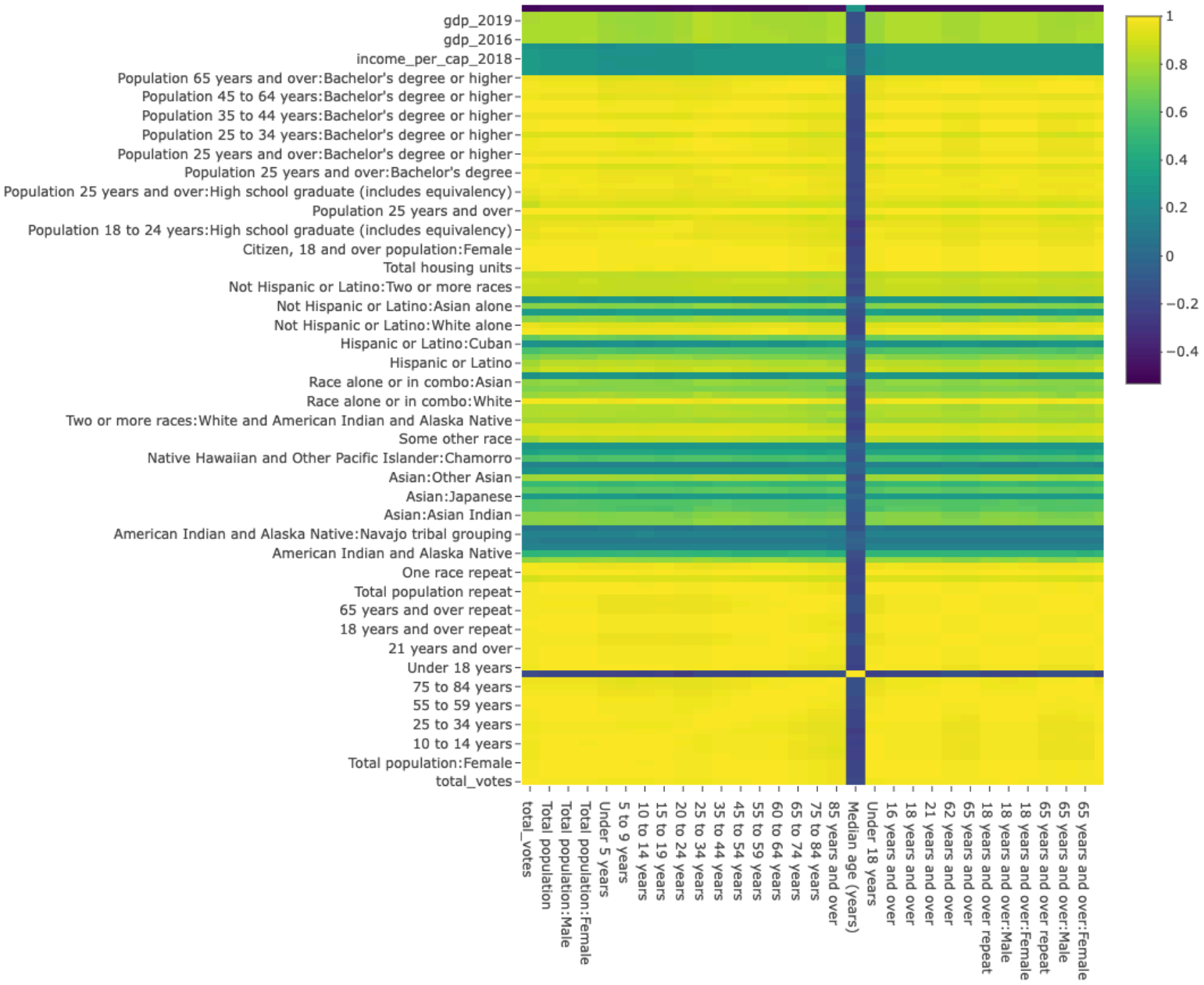
Graph 10



Description:

Graph 10 explores the county vote based on the proportion of citizens and the proportion of voters. Each point represents a county, with the x-axis showing the proportion of US citizens and the y-axis showing the proportion of voters. Blue dots represent counties won by Biden, and red dots represent counties won by Trump. The scatter plot indicates no significant pattern between the proportion of a county that is US citizens and the proportion of the county population that votes in determining whether Biden or Trump won that county. This suggests that factors beyond citizenship and voter proportion may play a crucial role in influencing election outcomes. The proportion of US citizens in each county was calculated by the total population of citizens over 18 divided by the total population of citizens, which produced proportions over 1 which indicates some issue in the population counts.

Graph 11:

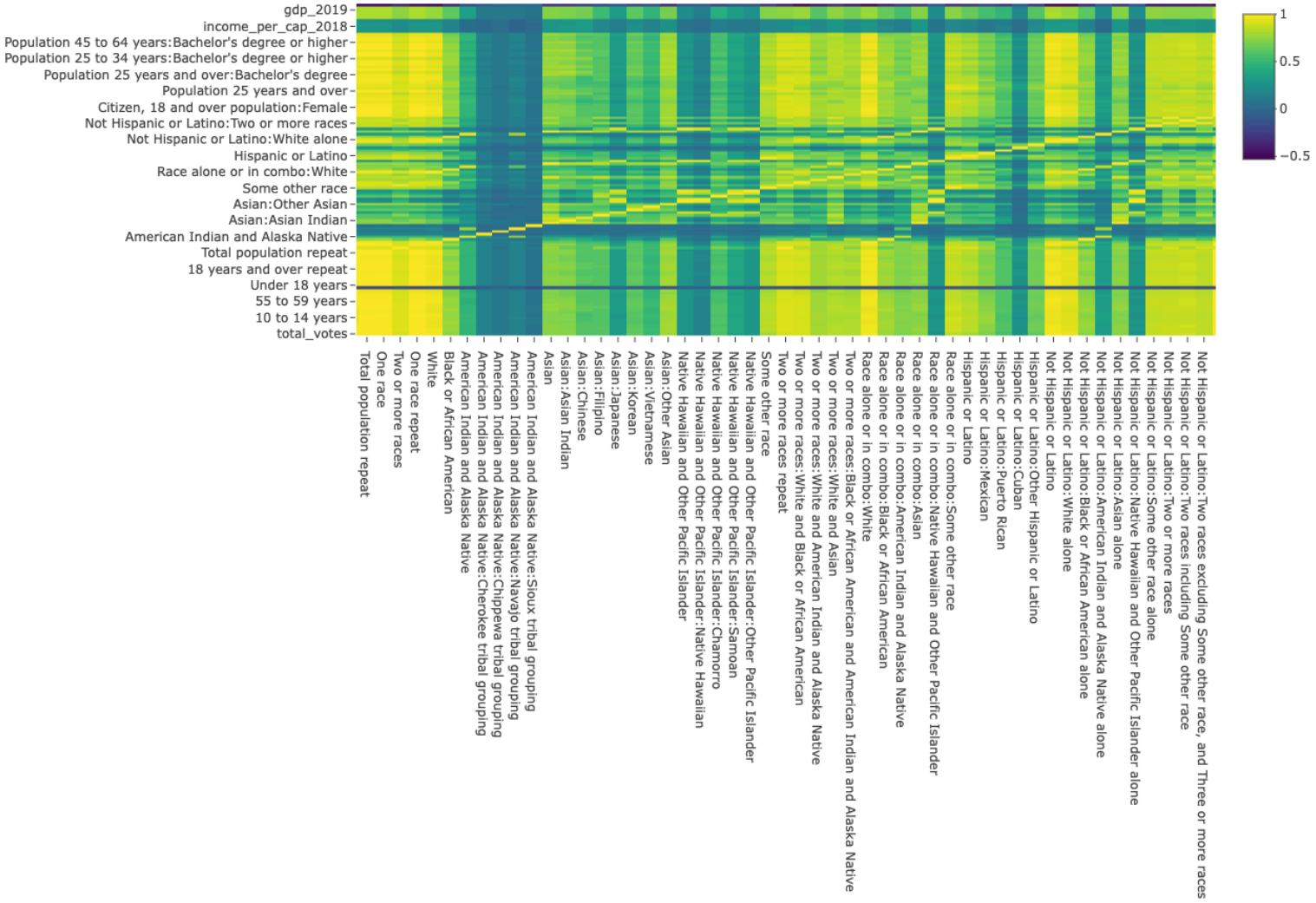


Description:

Graph 11 is a correlation plot of the total population ages along the x-axis with all the variables on the y-axis. It reveals a high correlation among different age groups, particularly evident in the bottom third of the plot where age-related variables are concentrated. The dark blue line, which represents the median age in the county, does not have as strong a correlation between the other age variables.

Additionally, there is minimal correlation between age and income or GDP variables at the very top of the graph, indicating that age distribution does not significantly impact economic indicators. Moreover, the top third of the plot shows a significant correlation between age and education variables, indicating a strong association between different age groups and education levels. In contrast, the correlation between age variables and race variables in the middle third of the plot is less pronounced, suggesting that age distribution does not vary significantly across different racial groups.

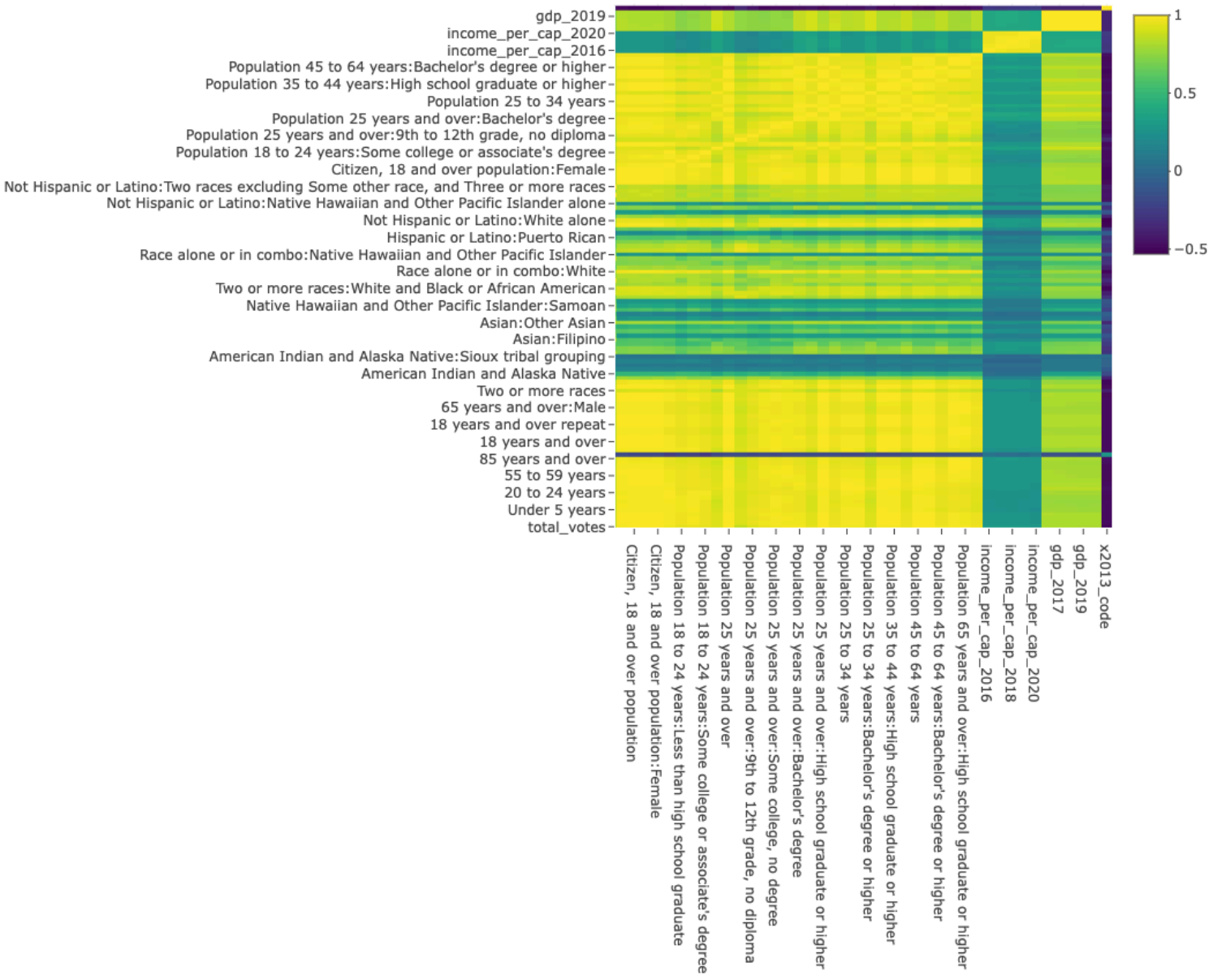
Graph 12:



Description:

Graph 12 is a correlation plot that focuses on the relationship between race variables and other variables. The higher presence of blue and green in the graph indicates less correlation between the various racial groups and all the variables. Including age, education, and other racial groups.

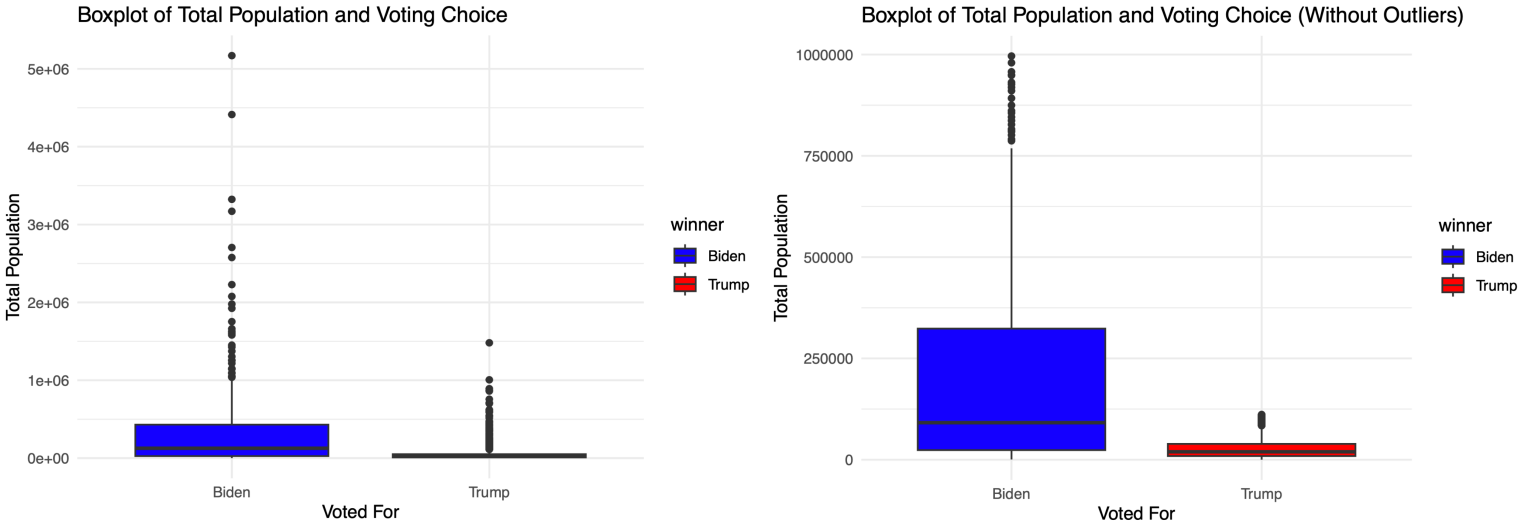
Graph 13:



Description:

Graph 13 is a correlation plot of education levels based on ages and the GDP/income along the x-axis with the other variables. The graph indicates a high correlation between education variables and age, reinforcing the strong relationship between age groups and education levels observed in the previous graphs. There is also a low correlation between age variables and race variables, suggesting that racial demographics do not significantly affect age distribution. Furthermore, the plot shows minimal correlation between income/GDP variables and age, indicating that economic indicators do not strongly influence age distribution.

Graph 14-15:



Description:

Graph 14 is a boxplot that illustrates the relationship between the total population and voting choice, distinguishing between those who voted for Biden and Trump. The data shows that counties where Biden won tend to have significantly higher populations than those where Trump won. Additionally, there are several outliers with extremely high populations in the Biden-voting counties, whereas Trump-voting counties exhibit fewer outliers and generally lower population sizes.

In ***graph 15*** the boxplot presents the same relationship between total population and voting choice but excludes outliers to provide a clearer view of the central

data. With outliers removed, it becomes evident that counties where Biden won generally have larger populations than those where Trump won. The interquartile range (IQR) for Biden-voting counties is much larger, indicating a wider spread of population sizes in these counties. In contrast, Trump-voting counties have a much smaller IQR, reflecting more consistent, smaller populations. The median population for Biden-voting counties remains higher, reinforcing the observation that more populous counties tended to vote for Biden.

These visualizations collectively offer valuable insights into the demographics and socio-economic factors influencing the election results in different counties. They highlight significant trends such as the age distribution, education levels, urbanization, and racial composition of the voting population, which are crucial for understanding voter behavior and election outcomes.

Preprocessing:

In the original dataset, we removed the name column as it was not conducive to our data processing needs, which required numerical data.

```
# Removing the 'name' column
train <- train %>% select(-name)
```

Initially, we attempted to remove outliers as part of our exploratory analysis to visualize the data more clearly.

```
# 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)
}
```

Afterwards, we developed a recipe for the analysis.

```
# Removing the 'name' column
train <- train %>% select(-name)

# recipe
rec <- recipe(as.formula(paste(target_var, "~ .")), data = train) %>%
  step_rm(all_of(id_var)) %>%
  step_dummy(all_nominal_predictors()) %>% # Convert categorical variables to dummies
  step_zv(all_predictors()) %>% # Remove variables with zero variance
  step_normalize(all_numeric_predictors()) # Normalize numeric variables
```

However, after running multiple iterations of different models and comparing the results of models with and without outliers, we concluded that the models performed best when outliers were included.

Furthermore we agreed to use 5-fold cross-validation on the training dataset, with a seed of 146 to ensure consistency.

```
46  # Cross-validation
47  set.seed(146)
48  cv_folds <- vfold_cv(train, v = 5)
```

Consequently, we decided to proceed with several different models using the refined dataset while retaining the outliers in all our models.

Candidate Models, Models Evaluations and Tuning :

Initially, we divided the work and selected five different candidate models to evaluate their performance, intending to narrow down the options based on their results. The five models we chose include Boosted Trees, Random forest, Support Vector Machine with a radial basis function (SVM), Decision Tree, Logistic Regression, and Bagged Trees. Each group member conducted their model analysis, and we then shared our findings and compared metrics such as Accuracy and Confusion Matrix. We selected the two best-performing models for this specific dataset based on these comparisons. Below is a detailed analysis of the models we chose.

Support Vector Machine with a radial basis function (SVM):

The Support Vector Machine (SVM) model with a radial basis function kernel was implemented to classify the counties based on demographic and socio-economic factors. This model was chosen for its robustness in handling non-linear relationships between the features and the target variable.

```
svm_model <- svm_rbf() %>%
  set_engine("kernlab") %>%
  set_mode("classification")
```

The SVM model underwent hyperparameter tuning, where parameters such as cost (C) and gamma (γ) were optimized to improve model performance. With that, a workflow was created, and the "kernlab" engine was utilized. The final SVM model showed a high accuracy of 0.9357602 and a kappa statistic of 0.7439412, indicating strong agreement between the predicted and actual classifications.

```
svm_workflow <- workflow() %>%
  add_model(svm_model) %>%
  add_formula(winner ~ .)
```

Despite these promising results, further tuning did not significantly enhance the model's performance, and the best results were obtained with the initial tuning parameters. Thus we obtained the following confusion matrix:

```
> gdp_svm_conf_matrix
          Truth
Prediction Biden Trump
     Biden    53     0
     Trump    30   384
```

In summary, the SVM model proved to be a reliable and effective tool for classification in this project, providing valuable insights into the demographic and socio-economic factors influencing election outcomes. Its strong performance metrics make it a suitable choice. Despite the good results, the metrics indicate that while the model performed adequately, it was not as robust or accurate as other models we evaluated.

Logistic Regression

Logistic Regression was our second possible model. It was applied to the dataset to predict the election outcomes based on the linear relationships between the predictor variables and the binary target variable (winner). The model included an initial recipe to normalize predictors and tune the penalty and mixture variables. A workflow was created, and the engine we utilized for this model was "glm."

```
log_reg_workflow <- workflow() %>%
  add_model(log_reg) %>%
  add_formula(winner ~ .)
```

```
log_reg <- logistic_reg() %>%
  set_engine("glm")
```

Despite extensive tuning, the best-tuned model achieved an accuracy of 0.9115336, slightly lower than the untuned model's of 0.9357602. This suggested that the simpler logistic regression model was more effective without additional tuning. Therefore, the final logistic regression model used all variables, except those with missing values (income and GDP columns), and demonstrated strong performance with an accuracy of 0.9357602 and a kappa of 0.7547355.

After applying a different recipe, we also implemented a different engine to see if the model would give different and better results.

```
rec <- recipe(winner ~ ., data = train_data) %>%
  step_naomit(all_predictors(), all_outcomes()) %>%
  step_dummy(all_nominal_predictors(), -all_outcomes()) %>%
  step_center(all_numeric_predictors()) %>%
  step_scale(all_numeric_predictors())
```

The new engine utilized for the logistic regression model was “glmnet.” The mode was run to see the results.

```
# Define logistic regression model with regularization
log_reg_model <- logistic_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet")
```

The tuning process involved adjusting the penalty and mixture parameters to identify the best hyperparameters. Despite this, the tuned model did not outperform the untuned model. The table below shows the tuning results with various penalty and mixture values, where all configurations resulted in an accuracy of 0.9115336:

penalty <dbl>	mixture <dbl>	.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
1.000000e-10	1	accuracy	binary	0.9115336	5	0.005227086	Preprocessor1_Model091
1.291550e-09	1	accuracy	binary	0.9115336	5	0.005227086	Preprocessor1_Model092
1.668101e-08	1	accuracy	binary	0.9115336	5	0.005227086	Preprocessor1_Model093
2.154435e-07	1	accuracy	binary	0.9115336	5	0.005227086	Preprocessor1_Model094
2.782559e-06	1	accuracy	binary	0.9115336	5	0.005227086	Preprocessor1_Model095

These results indicate that the regularization parameters (penalty and mixture) did not significantly impact the model's performance, reinforcing the decision to use the untuned logistic regression model. The logistic regression model's simplicity and robustness made it a strong candidate for predicting election outcomes, as reflected by its high accuracy and kappa scores. Despite the good results, the metrics indicate that while the model performed adequately, it was not as robust or accurate as other models we evaluated.

Bagged Trees

Bagged Trees, or Bootstrap Aggregating, was initially chosen for its potential to yield high predictive accuracy. The model's robustness comes from training multiple decision trees on different bootstrapped samples and averaging their predictions. This approach helps reduce variance and prevent overfitting, making the model more stable and reliable. The Bagged Trees model demonstrated its strength by performing fairly well (with an accuracy score above 0.90) on the training data with minimal tuning. However, the primary disadvantage was the limited number of hyperparameters available for tuning: tree depth and the minimum number of samples required for a split. The model was created, and the engine used was “rpart.” Then, a workflow was created, and the model was tuned to the best parameters.

```
# Specify Engine; specify hyper=parameters to tune; set mode
bagged_tree <- bag_tree() %>%
  set_engine("rpart") %>%
  set_mode("classification") %>%
  set_args(tree_depth = tune(), min_n = tune())

# Workflow
bagged_tree_workflow <-
  workflow() %>%
  add_formula(winner ~ .) %>%
  add_model(bagged_tree)
```

The tuning process aimed to find the optimal values for tree_depth and min_n. However, after these hyperparameters were optimized on the training data, there was little room for further improvement. Additionally, the model faced overfitting issues, as evidenced by the inconsistent relationship between the optimal tuning parameters on the cross-validation set and the test set results.

```
# Grid of hyper-parameters to tune
tuning_grid <- grid_regular(
  tree_depth(range = c(1, 10)),
  min_n(range = c(1, 10)),
  levels = 10
)

#  Tune on cross-validation set
tuned_res <- tune_grid(
  bagged_tree_workflow,
  resamples = class_folds,
  grid = tuning_grid
)

# Best hyper-parameters
best_params <- show_best(tuned_res, metric = "accuracy")
best_params
```

Despite these challenges, the Bagged Trees model achieved an accuracy of 0.8944605 and a kappa statistic of 0.5613463. These metrics indicate that while the model performed adequately, it was not as robust or accurate as other models we evaluated.

Decision Tree

The Decision Tree model was employed to predict the election outcomes based on various demographic and socio-economic factors. This model was chosen for its simplicity and interpretability, allowing us to easily understand the decision rules that lead to different classifications. The model uses a tree-like structure where each internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label.

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

```r
# hyperparameters for tuning
grid <- grid_regular(
  cost_complexity(),
  min_n(),
  levels = 10
)
```

Using the default presets, the cost_complexity parameter was adjusted between [−10,−1] after a log-10 transformation. The min_n parameter was also set to the default range of [2,40]. Controlling the degree of pruning in the decision tree facilitates the identification of an ideal value that strikes a balance between generalization and model complexity. The engine utilized in this model was “rpart”

The tuning process aimed to optimize the cost complexity and min n parameters. These two hyperparameters are crucial as they control the pruning of the tree, which reduces overfitting, and the minimum number of samples required for a split, respectively. The best hyperparameters found were cost complexity of 0.01 and min n of 2.

The finalized workflow was created and this configuration yielded a score of 0.90376 in Kaggle. Despite its accuracy of 0.8854564.

```r
# Finalize the workflow with the best hyperparameters
final_cart_workflow <- cart_workflow %>%
  finalize_workflow(best_result)
```

Decision Tree models have a tendency to overfit, which results in poor generalization by capturing noise in the training set. Despite its accuracy of 0.8854564, decision tree models have a tendency to overfit, which results in poor generalization by capturing noise in the training set. High variation in the model might lead to noticeably different trees from tiny changes in the data. The other models had better accuracy and scores in Kaggle, including when adjusting the hyperparameters. Due to these results, further efforts were made on other models.

Random Forest

The random forest model was also employed due to its ability to effectively handle high-dimensional datasets like ours. Additionally, it's less sensitive to outliers of a dataset and can still capture trends in our dataset. To build our model in R, we utilized tidymodels and ranger packages to train our data, fit our model, and ultimately predict the outcomes of each row of data. We used the rangers package as our set engine in the tidymodels format to build the model. In the rangers package, the hyperparameters we chose to tune were minimum node size, the number of randomly drawn candidate variables. We set the number trees at 100 to prevent the model from overfitting on the training. We chose to do a grid search for each of these with a level of 5 so we can get a variety of possible

```
set.seed(146)
folds <- vfold_cv(train, v = 5)


rf_grid <- grid_regular(
  mtry(range = c(2, 20)),
  min_n(range = c(5, 20)),
  levels = 5
)

tune_res <- tune_grid(
  wf,
  resamples = folds,
  grid = rf_grid,
  metrics = metric_set(accuracy)
)
```

Additionally, we implemented cross-fold validation when calculating the performance metric of each combination of parameters to ensure that we chose parameters with the highest accuracy for the data. After tuning our model, we got minimum node size of 12 and 20 randomly drawn candidate variables achieved the highest accuracy. When obtaining the performance metrics based on the training data, we achieve an accuracy of .909 and a standard error of .00383. When observing the score from Kaggle, this model got an accuracy of .91127, which ultimately was the second best model we submitted. We didn't go with this as our

final model due to the testing accuracy being lower to other models we tested on Kaggle. In the future, using other packages like the carets package to train our model where there's more hyperparameters to tune could lead to stronger results that aren't captured when using the rangers package.

Boosted Trees:

Boosted Trees was selected as one of our candidate models due to its effectiveness in handling complex datasets and its ability to improve model accuracy through iterative training. This model combines the predictions of multiple weak learners to create a strong predictive model, making it a powerful tool for classification tasks. The Boosted Trees model was implemented using the LightGBM engine, and a workflow was created.

```
37  # Hyperparameter tuning for LightGBM
38  boost_tree_tuned <- boost_tree(
39    trees = tune(),
40    tree_depth = tune(),
41    min_n = tune(),
42    learn_rate = tune()) %>%
43    set_engine("lightgbm") %>%
44    set_mode("classification")
```

The hyperparameters for the LightGBM model, including the number of trees, tree depth, minimum number of samples required for a split (min_n), and learning rate, were tuned to optimize model performance.

```r
#hyperparameter tuning
tune_res <- tune_grid(
  workflow() %>%
    add_recipe(rec) %>%
    add_model(boost_tree_tuned),
  resamples = cv_folds,
  grid = 20,
  # specify the metric
  metrics = metric_set(accuracy),
  control = control_grid(save_pred = TRUE))

#best performing metrics
best_params <- select_best(tune_res,
                           metric = "accuracy")
```

The grid search explored a variety of parameter combinations to find the optimal settings. The best hyperparameters identified through this process were 1722 trees, a tree depth of 12, a minimum of 18 samples required for a split, and a learning rate of 0.05771423. These values were selected based on their performance in cross-validation, maximizing the model's accuracy.

The final Boosted Trees model was configured using the best hyperparameters identified from the tuning process. The workflow was finalized and fitted to the entire training dataset.

```r
# Finalize the workflow with the best hyperparameters
final_workflow <- workflow() %>%
  add_recipe(rec) %>%
  add_model(boost_tree(
    trees = best_params$trees,
    tree_depth = best_params$tree_depth,
    min_n = best_params$min_n,
    learn_rate = best_params$learn_rate
  ) %>%
  set_engine("lightgbm") %>%
  set_mode("classification"))
# Fit the final workflow to the entire training data
bt_fitted <- final_workflow %>%
  fit(data = test)
```

The Boosted Trees model demonstrated strong performance metrics, achieving an accuracy of 0.9206312 and a kappa statistic of 0.6813206. On Kaggle's private leaderboard, the model scored 0.90388, and on the public leaderboard, it achieved 0.94560.

.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
accuracy	binary	0.9206312	5	0.006963004	Preprocessor1_Model1
kap	binary	0.6813206	5	0.036171151	Preprocessor1_Model1

These results confirmed that the Boosted Trees model was the best-performing model among those we evaluated, thus we chose this one as our final model.

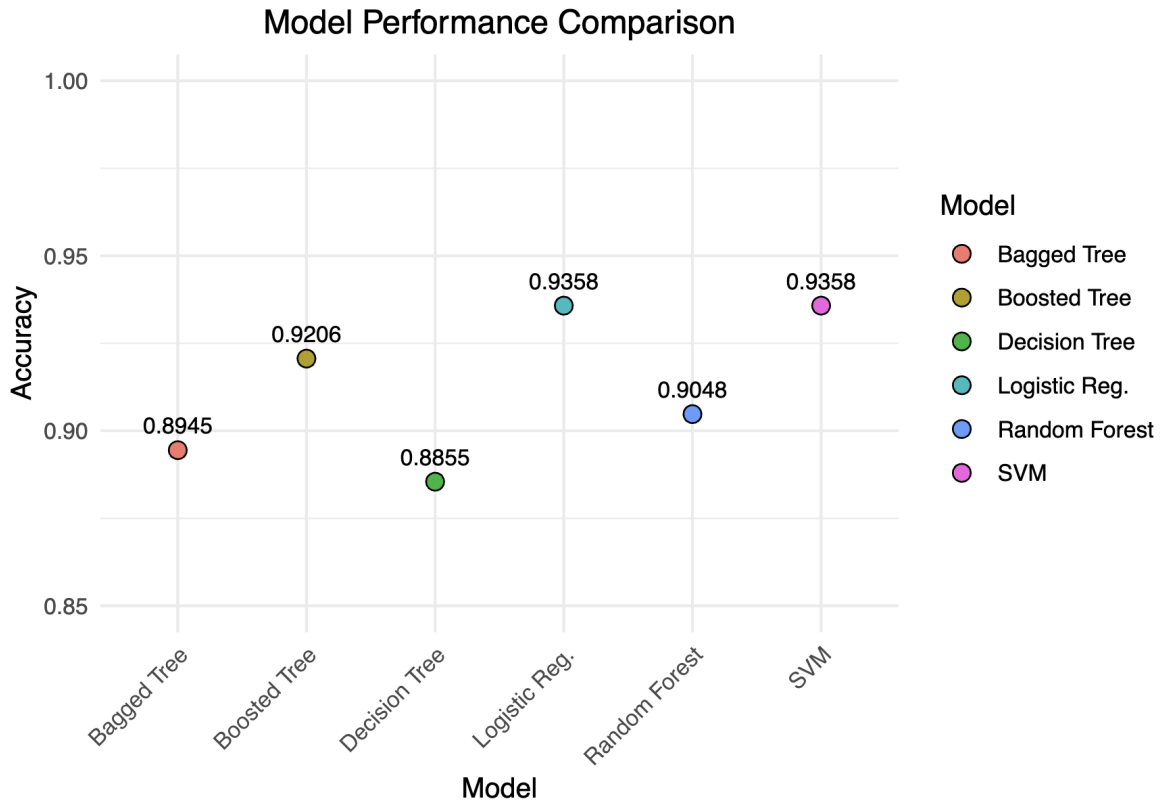
Table of finalized models:

Model identifier	***Type of Model***	***Engine***	***Recipe Used/Listed Variables***	***Hyperparameters***
decision_tree()	Decision Tree	"rpart"	All variables except "name"	min_n = 2 cost_complexity = 0.01
log_reg	Logistic Regression	"glm"	All variables w/o income_per_cap_2016 to income_per_cap_2020, gdp_2016, to gdp_2020 due to presence of NA	No parameters
svm_model	Support vector machine with a radial basis function	"kernlab"	All variables w/o income_per_cap_2016 to income_per_cap_2020, gdp_2016, to gdp_2020 due to presence of NA	Penalty = 0 Mixture = 0
random_forest	Random Forest	"ranger	Imputed NA using the median, Took the log of gdp_ 2016 to gdp_2020, and income_per_cap_2016 - income_per_cap_2020, Normalized all numeric predictors	Number of Trees: 100 Min_n: 12 Mtry: 20
bag_tree	Bagged trees model	"rpart"	All variables except "name"	tree_depth = 9, min_n = 1
boost_tree_tuned	Boost trees	"lightgbm"	All variables except "name"	trees = 1722, tree_depth = 12, min_n = 18, learn_rate = 0.05771423

Table of finalized metrics:

Model identifier	*Metric Score: Accuracy*	*Metric Score: kap*
log_reg	0.9357602	0.7547355
svm_model	0.9357602	0.7439412
decision_tree	0.8854564	0.5604668
random_forest	0.9047628	0.6079628
bag_tree	0.8944605	0.5613463
boost_tree_tuned	0.9206312	0.6813206

Graph 16:



Description:

Graph 16 represents a visual summary of the accuracy of all the models evaluated on the 10-fold cross-validation set. The x-axis represents the different models, while the y-axis shows the accuracy scores. The graph shows that the Boosted Trees model did not achieve the highest accuracy, scoring lower than the Logistic Regression and SVM models, each of which achieved an accuracy of 0.9358. These models effectively captured the linear relationships between the predictor and target variables. However, despite extensive tuning, they overfit the data, evident from their poor performance on the Kaggle competition's public and private leaderboards. In contrast, when evaluated on the test data, the Boosted Trees model emerged as the best-performing model among the candidates. Its iterative training process resulted in a high accuracy of 0.9206. More importantly, it demonstrated robustness and reliability with a score of 0.90388 on Kaggle's private leaderboard and 0.94560 on the public leaderboard. Given its superior performance metrics and consistent results across various validation methods, the Boosted Trees model was selected as the final model for predicting election outcomes.

Final Model Discussion:

In evaluating various models for predicting election outcomes, the Boosted Trees model emerged as the best-performing model. This choice was based on its ability to handle complex datasets effectively and iteratively improve model accuracy by combining the predictions of multiple weak learners.

Weaknesses: Despite its strengths, the Boosted Trees model also has some weaknesses. One notable issue is the computational complexity and the time required for training, especially with large datasets like this one; in fact, it took about 20 minutes for the model to run entirely and make predictions. Additionally, the model can be prone to overfitting if not properly tuned, although this was mitigated through our extensive hyperparameter tuning process.

Areas for Improvement: One area for potential improvement could have been a more thorough exploration of other models, such as the random forest model with the following parameters: Number of Trees: 100, Min_n: 12, Mtry: 20. Although the random forest model was not initially considered one of the top-performing models based on accuracy and the confusion matrix, it performed well on Kaggle's private leaderboard with a score of 0.91127. This suggests that the rf_model could have been a strong contender with further tuning and optimization.
Another point of consideration is that one version of the Boosted Trees model scored 0.91312 on Kaggle's private leaderboard. However, we were unaware of this until the competition results were released, and it scored lower on the public leaderboard and the accuracy tests than our chosen final model. This highlights the importance of continuous monitoring and validation of model performance.
Thus, the Boosted Trees model's ability to iteratively improve predictions and handle complex datasets makes it a reliable and powerful predictor of election outcomes. This comprehensive approach and consistent validation underscore the Boosted Trees model's value in our analysis, making it the final model of choice for this project.

Appendix: Final Annotated Script

```r
# Loading required libraries
library(tidyverse)
library(tidymodels)
library(ranger)
library(lightgbm)
library(bonsai)

# Reading the data sets:
train <- read_csv("train_class.csv")
test <- read_csv("test_class.csv")

# Removing the 'name' column
train <- train %>% select(-name)

# Feature engineering and pre-processing:
rec <- recipe(as.formula(paste(target_var, "~ .")),
              data = train) %>%
  step_rm(all_of(id_var)) %>%
  # Categorical variables to dummies
  step_dummy(all_nominal_predictors()) %>%
  # Remove variables with zero variance
  step_zv(all_predictors()) %>%
  # Normalize numeric variables
  step_normalize(all_numeric_predictors())

```

```r

# Setting the best hyper-parameters for LightGBM
best_params <- list(
  trees = 1722,
  tree_depth = 12,
  min_n = 18,
  learn_rate = 0.05771423
)

# Finalize the workflow
final_workflow <- workflow() %>%
  add_recipe(rec) %>%
  add_model(boost_tree(
    trees = best_params$trees,
    tree_depth = best_params$tree_depth,
    min_n = best_params$min_n,
    learn_rate = best_params$learn_rate
  ) %>%
    set_engine("lightgbm") %>%
    set_mode("classification"))

# Fit the final workflow to the test data
bt_fitted <- final_workflow %>%
  fit(data = train)

```

```r

# Fit the final workflow to the test data
bt_fitted <- final_workflow %>%
  fit(data = train)

# Making predictions
predictions <- predict(bt_fitted, new_data = test)

# Creating prediction tibble
predictions <- test %>%
  select(all_of(id_var)) %>%
  bind_cols(predictions)
write_csv(predictions, "final_predictions.csv")

```

Appendix: Team Member Contribution

Albert:

- Decision tree model, report writing of decision tree model.

Benedetta:

- Boost trees model R code, final mode code and R script, report organization, structure, editing, and writing of introduction, graph descriptions, boosted tree model, final model, improvements and script.

James:

- Random Forest model R code, report writing of random forest model

Kyle:

- Bagged trees model, editing script file, graph 14/15, plot of model performance, editing, proofreading,

Mikayla:

- SVM model code, logistic regression model code, exploratory analysis graphs, report organization.