Predicting the 2020 US Presidential Election

Brandon Nava

August 3, 2024

Contents

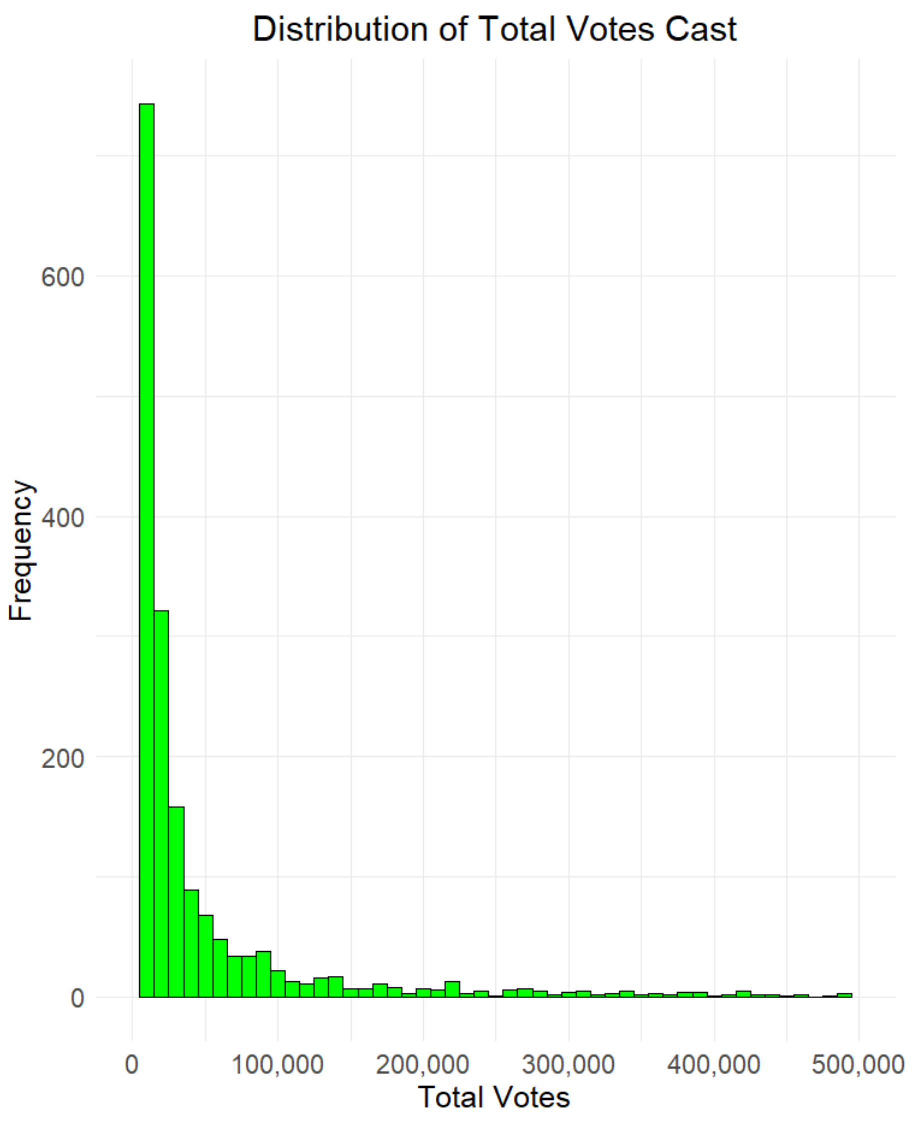
1 Introduction

Being able to understand factors that can influence electoral outcomes can be crucial for political analysts and researchers. The outcome of the 2020 US Presidential Election presented unique challenges and insights into voter behavior across different counties. In this study, I aim to predict the winner of a county using demographic and educational data. Variables such as total population, race, age, and education levels typically have significant associations with the election. I plan to test different models and find which one can best predict electoral outcomes.

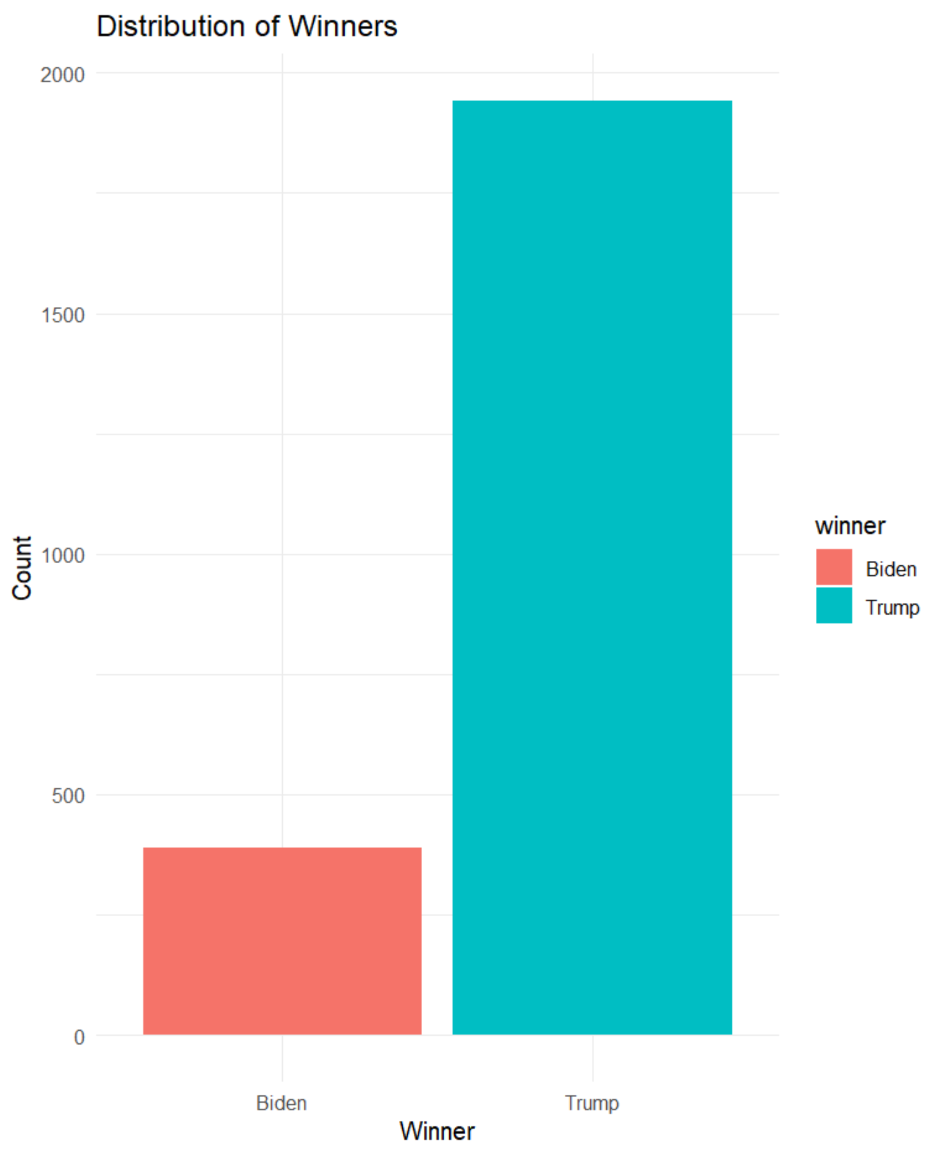
2 Data Analysis

2.1 Relationships & Visualizations

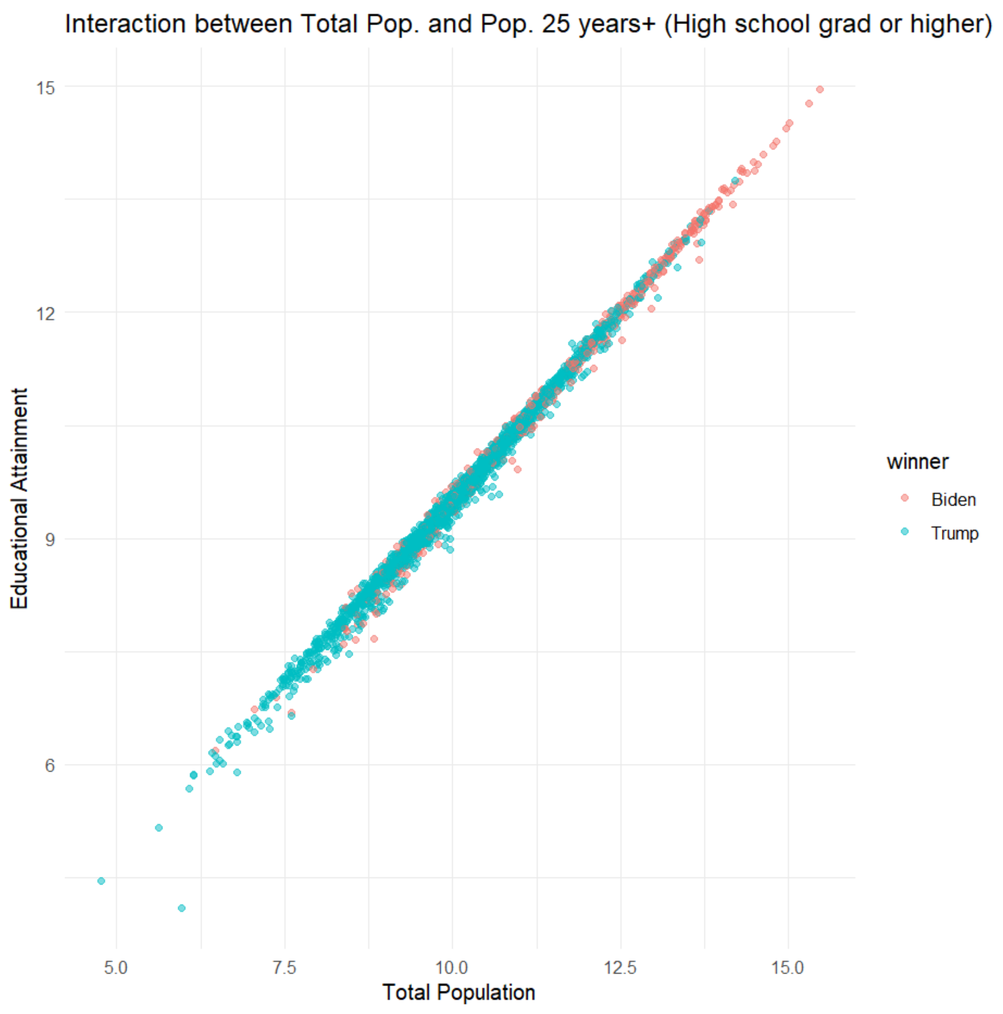
We will first explore the potential relationships between the demographic variables and election outcome. This will be done by transforming variables and using visualizations to explore possible interactions.



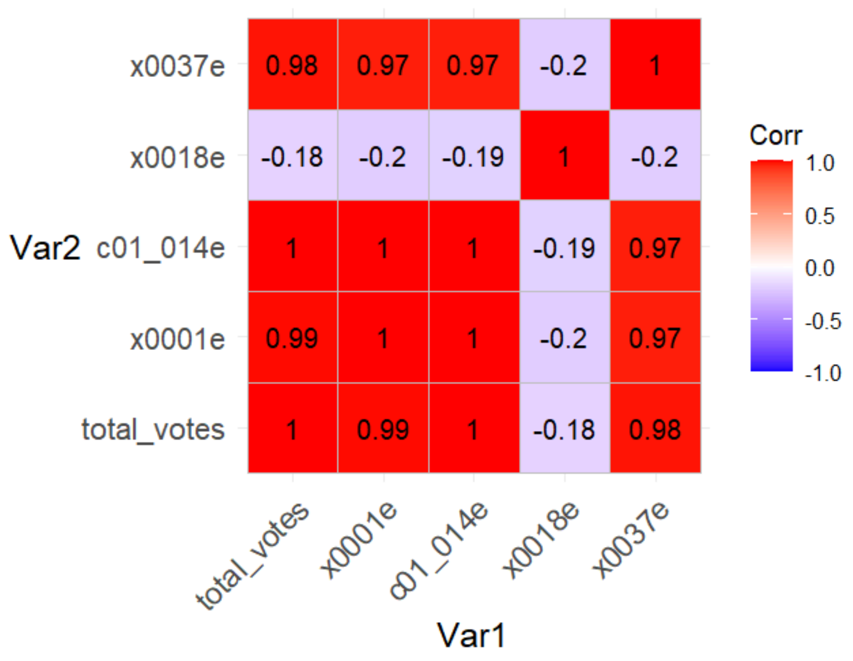
Based on the way the histogram is skewed to the right, this tells us that many counties have lower voter turnout. This is a crucial pattern that can help us make sense of voter behavior.



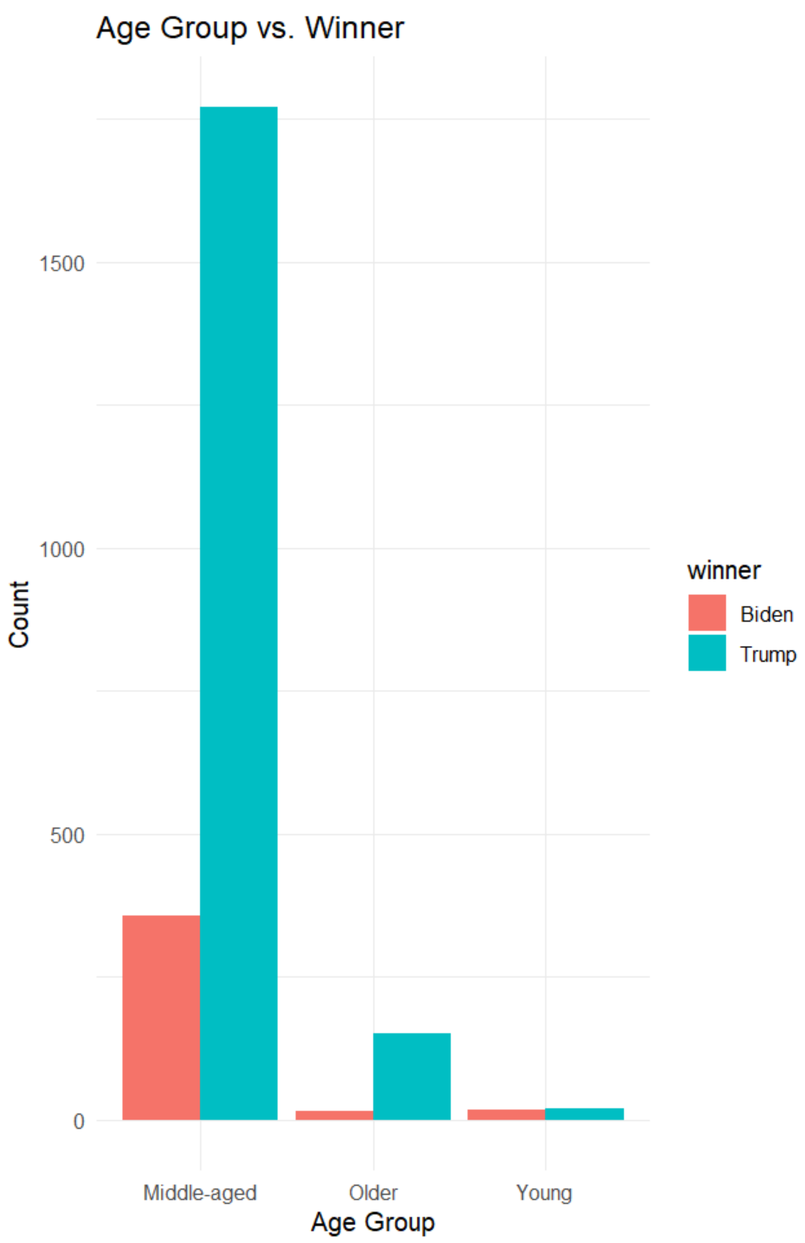
The bar plot shows the distribution of winners in the dataset, specifically comparing the number of counties won by Biden versus Trump. Although the number of counties won by Trump is higher, we have to consider population and total number of votes. This shows us that we have to consider other factors in political analysis.



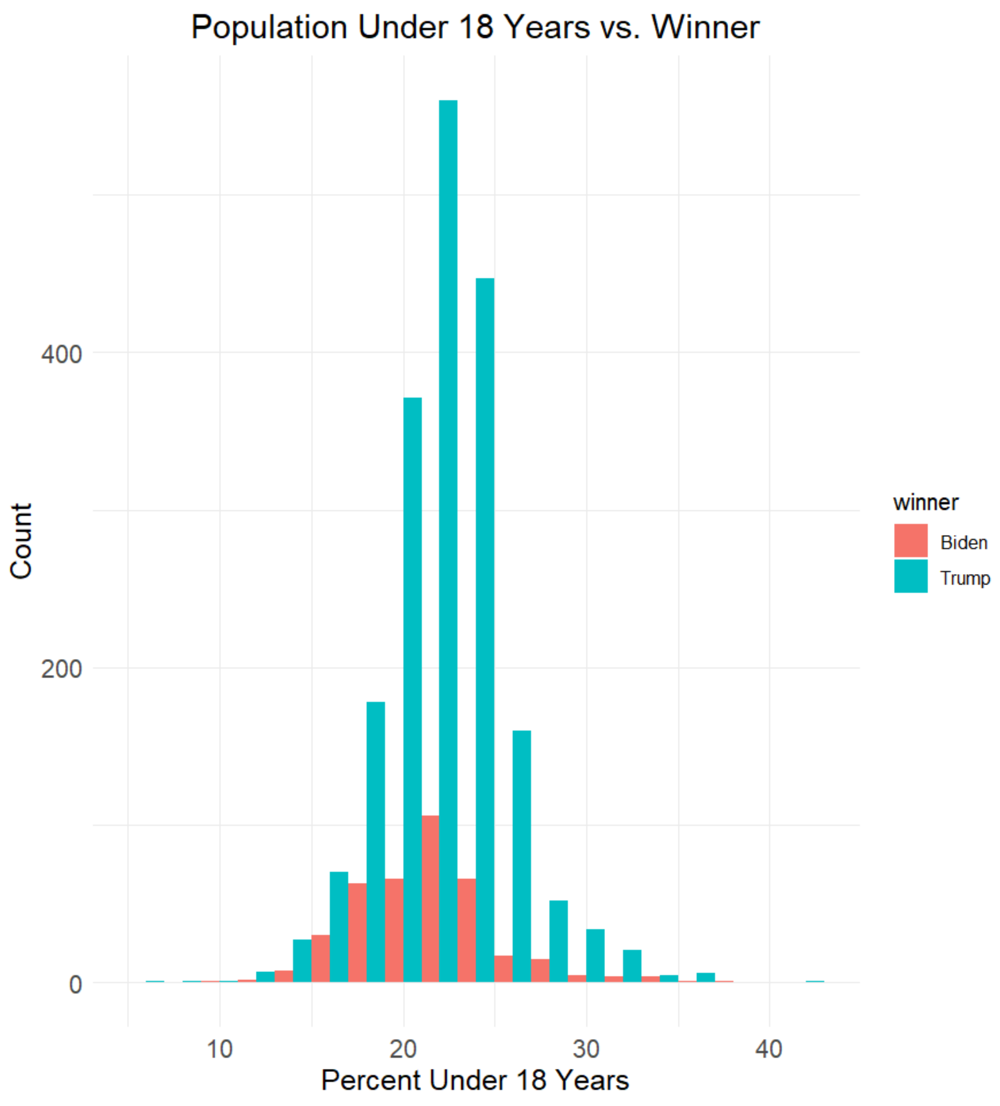
There is a clear linear relationship between the total population and education levels. The separation may indicate different patterns in how educational attainment and population size influence voting outcomes.



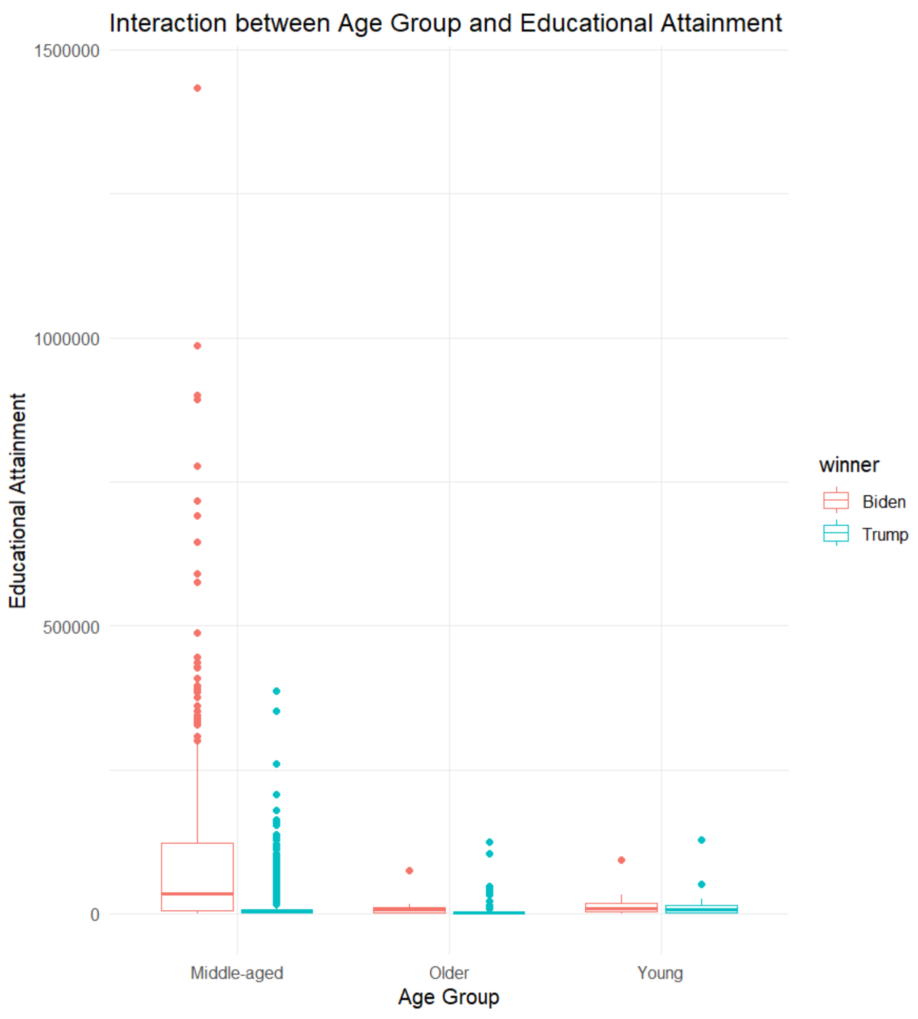
There is a perfect correlation between total votes and the number of people with at least a high school education (c01_014e), suggesting that education levels increase with the population. There is also a slight negative correlation between median age (x0018e) and total votes, indicating that counties with a higher median age may have slightly fewer total votes.



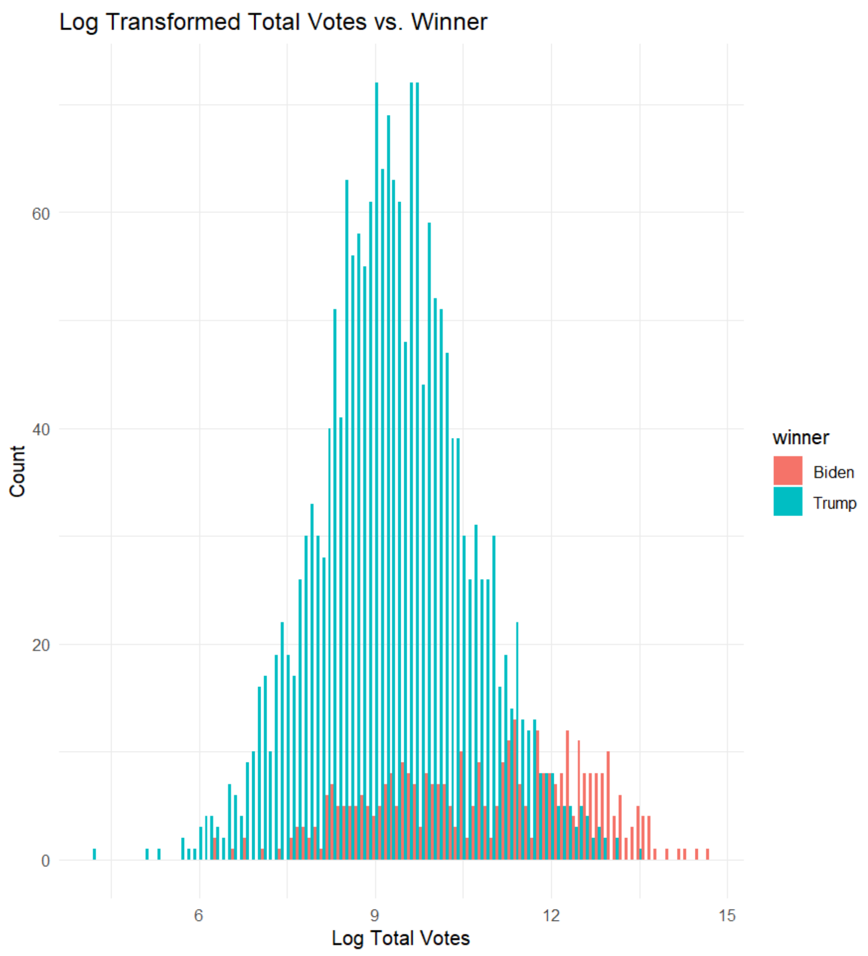
Among these middle-aged counties, Trump won significantly more counties compared to Biden. Trump also won more "Older" counties compared to Biden, though the difference is less pronounced than in the "Middle-aged" category. The distribution between Biden and Trump is more balanced in this age group, though the number of counties is small.



The majority of counties have between 20% and 25% of their population under 18 years, suggesting that many counties have a similar demographic structure with respect to the younger population. Biden's wins are more concentrated around the central range of 20-25%. The highest count of counties falls around the 20-22% range for Trump. Both Biden and Trump have wins in counties with a broader range of percentages, but Trump dominates in the overall number of counties won.



Biden's educational attainment values are generally higher compared to Trump's in this age group. This might suggest that counties won by Biden in the middle-aged category tend to have higher educational attainment. Both older and young groups have lower educational attainment values compared to the middle-aged group.



The log-transformed total votes exhibit a roughly normal distribution with a peak around a log value of 9. Trump's wins are more frequent in counties with lower total votes, indicating stronger performance in smaller or less populous counties. Biden's performance improves as the total vote count increases, suggesting stronger support in more populous counties.

3 Preprocessing/Recipes

I prepared the data by converting population counts into percentages for demographics to standardize comparisons across counties. I then removed unnecessary columns to simplify the dataset and to avoid multicollinearity. The target variable `winner` was converted to a factor for classification tasks. I standardized numerical features by centering and scaling them to normalize their range. This was done in order to improve the overall performance of the model. In order to deal with missing values I imputed them with column means to maintain dataset integrity. I finally split data into training and validation sets for performance evaluation and model tuning before starting my model training and evaluation.

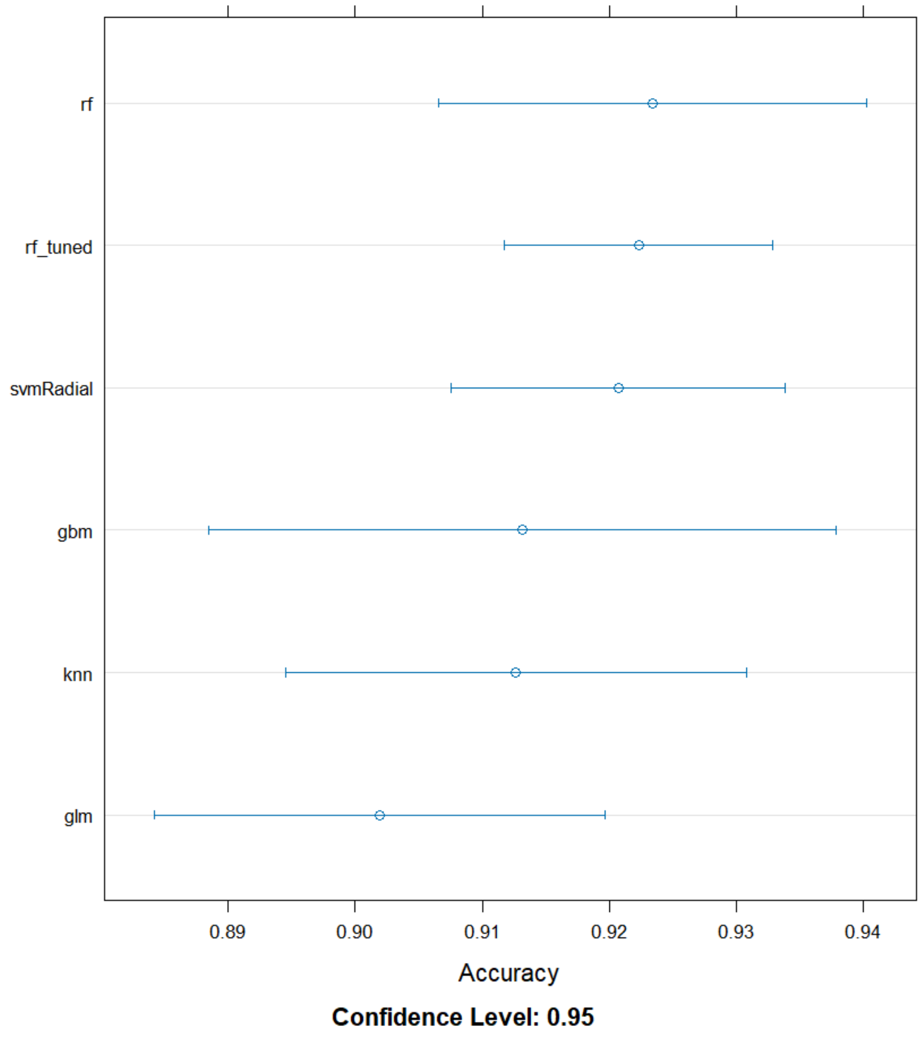
5 Model Evaluation

One of the models I used was the Generalized Linear Model (GLM), since it was a simple model and serves well for classification tasks. The K-Nearest Neighbors Model (KNN) was used for its straightforward algorithm which predicts based on close training examples and is useful for finding patterns and relationships. I also used the Random Forest Model (RF) that makes decision trees to improve predictive accuracy and accommodate possible overfitting. The Support Vector Machine (SVP) is effective for high dimensional spaces so it was a suitable contender for the model evaluation. I also used the Gradient Boosting machine since it builds models sequentially and corrects previous models and is effective for machine learning tasks. Here is a table with details on the different models.

Model Identifier	**Type of model**	**Engine**	**Variables**	**Hyperparameters**
General Linear Model	Logistic Regression	glm	all	default
GBM Model	Gradient Boosting	gbm	all	default
Random Forest Model	Random Forest	ranger	all	default
Support Vector Machine	SVM with RBF Kernel	e1071	all	default
RF_Tuned	Tuned Random Forest	ranger	all	ntree = 1000, mtry = sqrt(number of variables)
KNN Model	K-Nearest Neighbors	caret	all	default

The models were evaluated using the 5-fold cross-validation method. The accuracy mean was the main metric I used to determine the potential performance of the models. In the end, the Tuned Random Forest Model showed to be the best model for fitting the full data for training and making final predictions. I included a table with a summary of the performances along with some graphics that also support this finding.

Model Identifier	**Mean Accuracy**	**Median Accuracy**	**Max Accuracy**	**Min Accuracy**
General Linear Model	0.9019280	0.9059140	0.9171123	0.8820375
GBM Model	0.9212267	0.9166667	0.9463807	0.9090909
Random Forest Model	0.9185515	0.9166667	0.9436997	0.9010695
Support Vector Machine	0.9201586	0.9171123	0.9356568	0.9115282
RF_Tuned	0.9238950	0.9222520	0.9358289	0.9115282
KNN Model	0.9121157	0.9037433	0.9383378	0.9032258



6 Discussion

The final model selected for generating predictions was the Tuned Random Forest Model. This was based on its performance in terms of overall mean accuracy and upper quartile values, as evidenced by the cross-validation results. The tuning of hyperparameters played a crucial role in enhancing its predictive capability. Its robustness against overfitting, especially when the hyperparameters were properly tuned, was able to handle a large number of input variables without overfitting. One of the weaknesses came down to its intensive computation. With this dataset in particular, this model was slower to train and took longer to execute. This model could be improved through the use of more extensive hyperparameter optimization. By also considering additional factors such as more voter demographics, socioeconomic indicators, and historical voting patterns, we could provide more predictive power for this model.

7 Appendix & Contributions

Here is the final script I used in R. I made all the contributions for this project.

```
# Nava, Brandon - Classification project

setwd("C:/Users/RGRIND")

# Prepare packages
install.packages("tidyverse")
install.packages("readr")
install.packages("caret")
install.packages("GGally")
install.packages("ggplot2")
install.packages("randomForest")
install.packages("glmnet")
install.packages("xgboost")
install.packages("e1071")
install.packages("ggcorrplot")
install.packages("caretEnsemble")
install.packages("gbm")
install.packages("dplyr")

library(gbm)
library(dplyr)
library(caretEnsemble)
library(ggcorrplot)
library(tidyverse)
library(readr)
library(caret)
library(GGally)
library(ggplot2)
library(randomForest)
library(glmnet)
library(xgboost)
library(e1071)

# Load in train data
train <- read_csv("train_class.csv")
```

```
# Histogram of total votes
ggplot(train, aes(x = total_votes)) +
  geom_histogram(binwidth = 10000, fill = "green", color =
"black") +
  theme_minimal() +
  labs(title = "Distribution of Total Votes Cast", x = "Total
Votes", y = "Frequency") +
  theme(plot.title = element_text(hjust = 0.5, size = 16),
        axis.title.x = element_text(size = 14),
        axis.title.y = element_text(size = 14),
        axis.text.x = element_text(size = 12),
        axis.text.y = element_text(size = 12)) +
  scale_x_continuous(labels = scales::comma, limits = c(0,
500000))

# Bar plot of winners
ggplot(train, aes(x = winner, fill = winner)) +
  geom_bar() +
  theme_minimal() +
  labs(title = "Distribution of Winners", x = "Winner", y =
"Count")

# Log value of interaction between population and high school
grads 25 yrs old+
ggplot(train, aes(x = log(x0001e + 1), y = log(c01_014e + 1),
color = winner)) +
  geom_point(alpha = 0.5) +
  theme_minimal() +
  labs(title = "Interaction between Total Pop. and Pop. 25
years+ (High school grad or higher)",
       x = "Total Population", y = "Educational Attainment")

## Correlation Matrix
# Select numeric variables for correlation matrix
numeric_vars <- train %>% select(where(is.numeric))

# Generate correlation matrix
corr_matrix <- cor(numeric_vars, use = "complete.obs")


# Select a subset of features
```

```
subset_vars <- c('total_votes', 'x0001e', 'c01_014e', 'x0018e',
'x0037e')

# Filter correlation matrix
subset_corr_matrix <- corr_matrix[subset_vars, subset_vars]

# Plot subset correlation matrix
ggcorrplot(subset_corr_matrix, lab = TRUE) +
  theme(plot.title = element_text(hjust = 0.5, size = 16),
        axis.title.x = element_text(size = 14),
        axis.title.y = element_text(size = 14),
        axis.text.x = element_text(size = 12),
        axis.text.y = element_text(size = 12))

## Age groups bar plot
train_age <- train %>%
  mutate(age_group = case_when(
    x0018e < 30 ~ 'Young',
    x0018e >= 30 & x0018e < 50 ~ 'Middle-aged',
    TRUE ~ 'Older'
  ))

ggplot(train_age, aes(x = age_group, fill = winner)) +
  geom_bar(position = "dodge") +
  theme_minimal() +
  labs(title = "Age Group vs. Winner", x = "Age Group", y =
"Count")

## Age groups box plot and education
ggplot(train_age, aes(x = age_group, y = c01_012e, color =
winner)) +
  geom_boxplot() +
  theme_minimal() +
  labs(title = "Interaction between Age Group and Educational
Attainment", x = "Age Group", y = "Educational Attainment")



## Normalize the 'x0019e' column to get the percentage of
population under 18 years
train_percent <- train %>%
  mutate(percent_under_18 = (x0019e / x0001e) * 100)

# Plot Population Under 18 Years vs. Winner
ggplot(train_percent, aes(x = percent_under_18, fill = winner))
+
  geom_histogram(position = "dodge", binwidth = 2) +
  theme_minimal() +
  labs(title = "Population Under 18 Years vs. Winner",
       x = "Percent Under 18 Years",
       y = "Count") +
  theme(plot.title = element_text(hjust = 0.5, size = 16),
        axis.title.x = element_text(size = 14),
        axis.title.y = element_text(size = 14),
        axis.text.x = element_text(size = 12),
        axis.text.y = element_text(size = 12))

# Log transformation of total votes
train_log <- train %>%
  mutate(log_total_votes = log(total_votes + 1))

ggplot(train_log, aes(x = log_total_votes, fill = winner)) +
  geom_histogram(position = "dodge", binwidth = 0.1) +
  theme_minimal() +
  labs(title = "Log Transformed Total Votes vs. Winner", x =
"Log Total Votes", y = "Count")



### Prepare data for candidate models

# Load test data
test <- read_csv("test_class.csv")

# Check for missing values in the training set
sum(is.na(train))

# Check for missing values in the test set
sum(is.na(test))

# Handle missing values by filling with the median
train <- train %>%
  mutate(across(everything(), ~ifelse(is.na(.), median(., na.rm
= TRUE), .)))

test <- test %>%
  mutate(across(everything(), ~ifelse(is.na(.), median(., na.rm
= TRUE), .)))

# Create percentage features for various demographics
new_train <- train %>%
  mutate(
    percent_male = (x0002e / x0001e) * 100,
    percent_female = (x0003e / x0001e) * 100,
    percent_white = (x0037e / x0001e) * 100,
    percent_black = (x0038e / x0001e) * 100,
    percent_asian = (x0044e / x0001e) * 100,
    percent_hispanic = (x0071e / x0001e) * 100,
    percent_high_school_grad = (c01_009e / c01_006e) * 100,
    percent_bachelor_degree = (c01_012e / c01_006e) * 100
  )

test <- test %>%
  mutate(
    percent_male = (x0002e / x0001e) * 100,
    percent_female = (x0003e / x0001e) * 100,
    percent_white = (x0037e / x0001e) * 100,
    percent_black = (x0038e / x0001e) * 100,
    percent_asian = (x0044e / x0001e) * 100,
    percent_hispanic = (x0071e / x0001e) * 100,
    percent_high_school_grad = (c01_009e / c01_006e) * 100,
    percent_bachelor_degree = (c01_012e / c01_006e) * 100
  )

# Remove irrelevant features after creating percentages
new_train <- new_train %>%
  select(-x0002e, -x0003e, -x0037e, -x0038e, -x0044e, -x0071e,
         -c01_009e, -c01_012e, -x0001e)

test <- test %>%
  select(-x0002e, -x0003e, -x0037e, -x0038e, -x0044e, -x0071e,
         -c01_009e, -c01_012e, -x0001e,)
```

```
# Convert categorical variables to factors
new_train$winner <- as.factor(new_train$winner)

# Remove 'name' column from new_train and test
new_train <- new_train %>% select(-name)

# Preserve the target variable 'winner' and remove it from
predictors
winner <- train$winner
new_train <- new_train %>% select(-winner)

# Identify and remove constant columns
constant_vars_train <- sapply(new_train, function(x)
length(unique(x)) == 1)
constant_vars_test <- sapply(test, function(x) length(unique(x))
== 1)
new_train <- new_train[, !constant_vars_train]
test <- test[, !constant_vars_test]


# Check for multicollinearity and remove highly correlated
variables
cor_matrix <- cor(new_train)
high_corr_vars <- findCorrelation(cor_matrix, cutoff = 0.9)
new_train <- new_train[, -high_corr_vars]
test <- test[, -high_corr_vars]

# Re-standardize features
preProc <- preProcess(new_train, method = c("center", "scale"))
train_normalized <- predict(preProc, new_train)
test_normalized <- predict(preProc, test)

# Add back the target variable
train_normalized$winner <- train$winner

# Split the data into training and validation sets
set.seed(123)
trainIndex <- createDataPartition(train_normalized$winner, p =
.8, list = FALSE)
train_set <- train_normalized[trainIndex, ]
validation_set <- train_normalized[-trainIndex, ]
```

```
# Train models with refined data
train_control <- trainControl(method = "cv", number = 5,
allowParallel = TRUE)
models_list <- caretList(
  winner ~ ., data = train_set,
  trControl = train_control,
  methodList = c("glm", "knn", "rf", "svmRadial", "gbm")
)


# Collect resampling results
results <- resamples(models_list)

# Summary of resampling results
summary(results)

# Plotting the resampling results
bwplot(results, metric = "Accuracy")
dotplot(results, metric = "Accuracy")

# Create a table summarizing the performance of each model
results_df <- summary(results)$statistics$Accuracy
results_df <- as.data.frame(results_df)
results_df$model <- rownames(results_df)
colnames(results_df) <- c("Min", "Q1", "Median", "Mean", "Q3",
"Max", "model")

print(results_df)

# For tuning hyperparameters, use the 'train' function with
'tuneGrid'
# Example for tuning the Random Forest model
tune_grid_rf <- expand.grid(mtry = c(2, 4, 6, 8, 10))
train_control_tune <- trainControl(method = "cv", number = 5)

# Train the Random Forest model with hyperparameter tuning
set.seed(123)
rf_model_tuned <- train(
  winner ~ ., data = train_set,
  method = "rf",
  trControl = train_control_tune,
  tuneGrid = tune_grid_rf
)

# Add the tuned model to the list of models
models_list$rf_tuned <- rf_model_tuned

# Collect resampling results again
results <- resamples(models_list)

# Summary of resampling results with the tuned model
summary(results)

# Plotting the resampling results with the tuned model
bwplot(results, metric = "Accuracy")
dotplot(results, metric = "Accuracy")

# Create an updated summary table
results_df <- summary(results)$statistics$Accuracy
results_df <- as.data.frame(results_df)
results_df$model <- rownames(results_df)
colnames(results_df) <- c("Min", "Q1", "Median", "Mean", "Q3",
"Max", "model")

# Print the updated summary table
print(results_df)


# Combine training and validation sets for final training
final_train_set <- train_normalized

# Train the final Random Forest model with the tuned
hyperparameters
final_rf_model <- train(
  winner ~ .,
  data = final_train_set,
  method = "rf",
  tuneGrid = rf_model_tuned$bestTune,
  trControl = trainControl(method = "none")
)
```

```
# Impute missing values with the mean for numeric columns
for(i in seq_along(test)) {
  if(is.numeric(test[[i]])) {
    test[[i]][is.na(test[[i]])] <- mean(test[[i]], na.rm = TRUE)
  }
}

# Verify imputation
sum(is.na(test))  # Should return 0 if all missing values are
imputed

# Normalize the test dataset
test_normalized <- predict(preProc, test)

# Verify normalization
sum(is.na(test_normalized))  # Should return 0

# Make predictions on the test dataset
final_predictions <- predict(final_rf_model, newdata =
test_normalized)

# Ensure the number of predictions matches the number of rows in
the test dataset
if (length(final_predictions) == nrow(test)) {
  # Add ID column from test dataset
  test_ids <- test$id

  # Create a submission data frame
  submission <- data.frame(id = test_ids, PredictedWinner =
final_predictions)

  # Save the submission data frame to a CSV file
  write_csv(submission, "final_submission.csv")

  # Check the first few rows of the submission file to ensure
it's correct
  head(submission)
} else {
  cat("The number of predictions does not match the number of
test rows.\n")
}
```