Stats 101C Kaggle Report Submission - classification data

Kelsie Fisher, Wonhyeok Choi, Abdulsamad Bholat, Dillon Maheshwari, Mehdi Abounacer

2025-09-13

1. Introduction

The 2020 U.S. Presidential election revealed sharp divides across geography, demographics, and education. Although Joe Biden won the presidency, Donald Trump carried the majority of counties, particularly in rural areas. This project uses county-level data from the MIT Election Lab (MIT Election Lab, 2020) and demographic and education estimates from the U.S. Census Bureau (U.S. Census Bureau, 2020). We aim to predict the county winner, either Biden or Trump, and examine which variables are most associated with voting outcomes. Prior studies suggest population size, education levels, racial composition, and income may strongly influence results.

MIT Election Lab. (2020). U.S. presidential election county results. MIT Election Lab. https://electionlab.mit.edu/

U.S. Census Bureau. (2020). American Community Survey data. U.S. Census Bureau. https://data.census.gov/

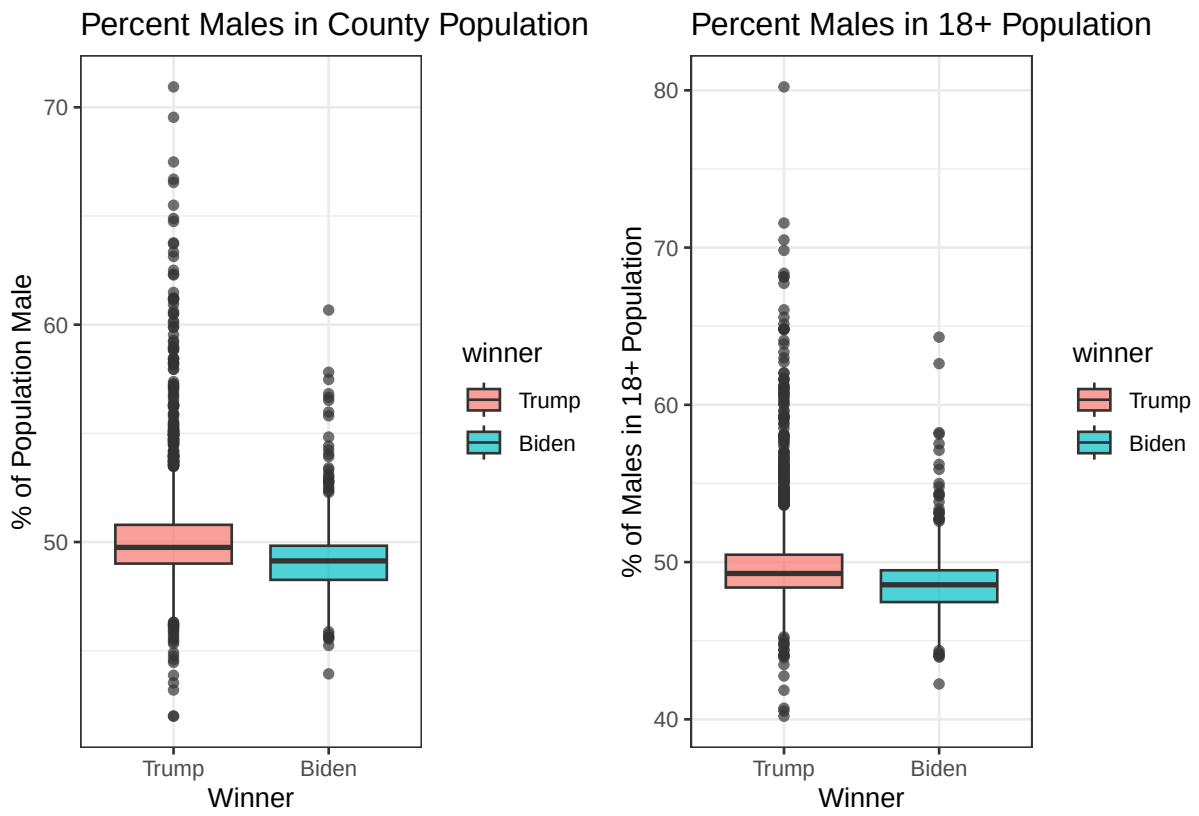
2. Exploratory Data Analysis

```
## # A tibble: 2 x 3
##   winner     n  prop
##   <fct>  <int> <dbl>
## 1 Trump   1942 0.833
## 2 Biden    389 0.167
```

The dataset is dominated with Trump winning counties (83%) compared to Biden winning counties (17%). The rest of the EDA will mostly compare proportions to analyze demographic composition among the counties and reduce the effect of Trump counties skewing estimates.

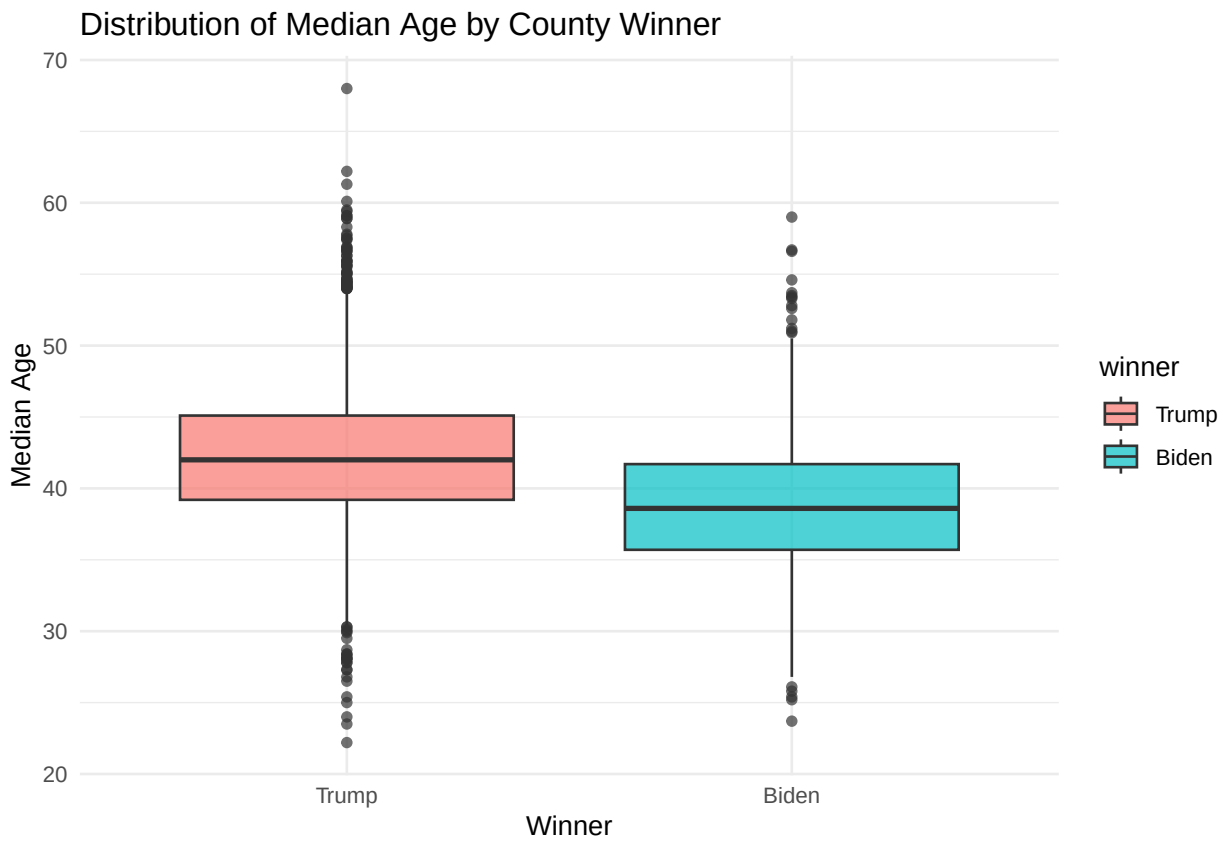
2.1 Individual Demographics Information

We begin by examining individual level demographic characteristics such as gender, age, race, citizenship, and education level to understand the population composition in Biden vs Trump winning counties.

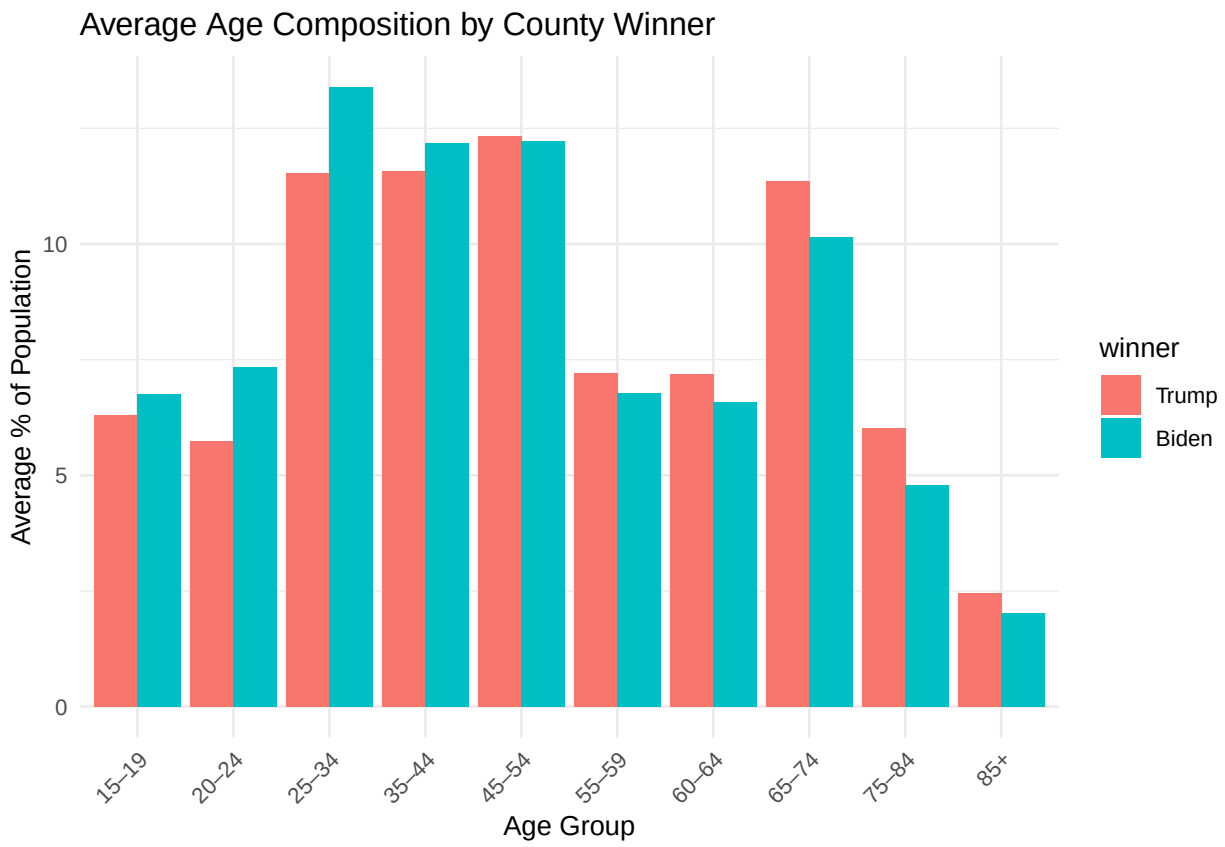


The boxplots for male total population percentages are similar across Trump and Biden winning counties, with both centered slightly below 50% male. Biden counties show a somewhat tighter distribution, while Trump counties have more spread and outliers and a higher max at 70%. We also restricted to the voting age population (18+) and found also identicical results, which suggests gender composition does not meaningfully differ between the general population and those who can vote.

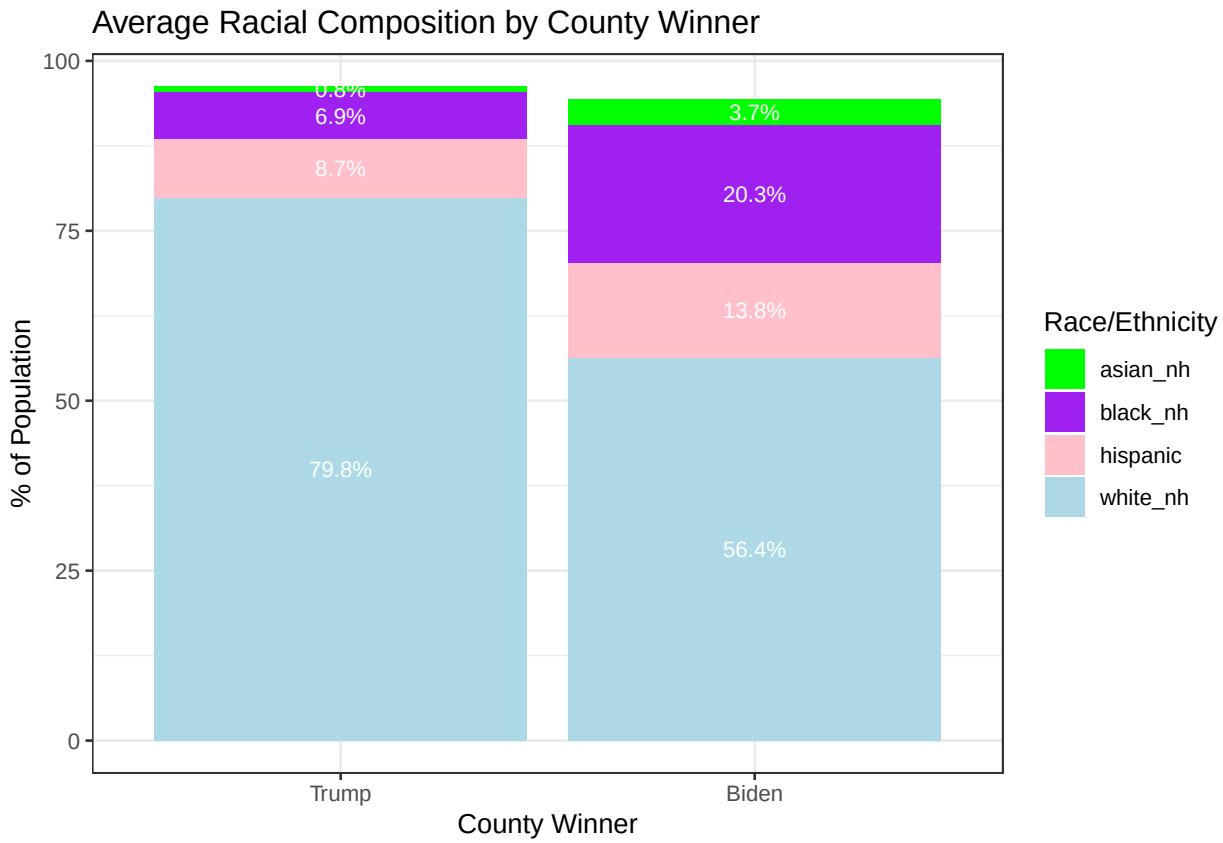
```
## # A tibble: 2 x 4
##   winner median_age mean_age     n
##   <fct>       <dbl>    <dbl> <int>
## 1 Trump        42       42.3  1942
## 2 Biden        38.6     39.0   389
```



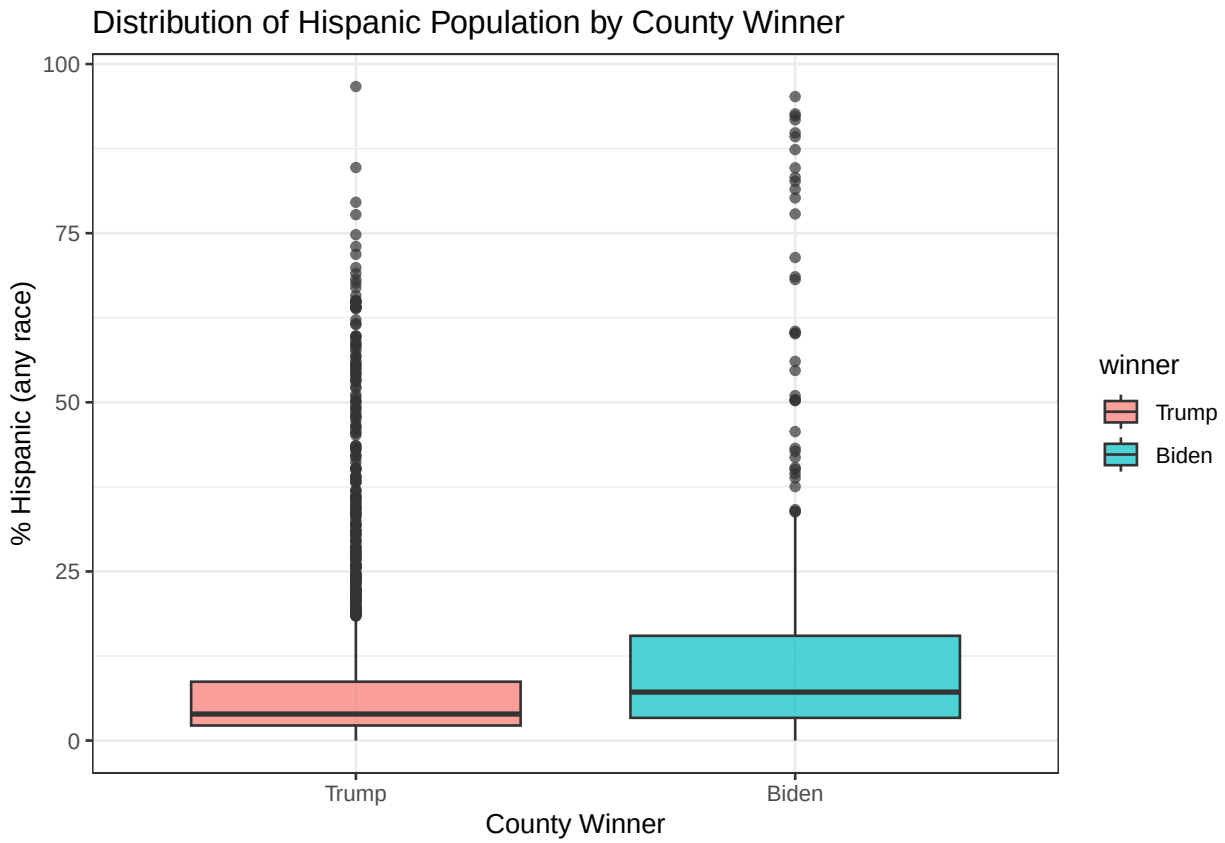
The boxplots for age show that Trump counties tend to be slightly older at a median age of 42 years old, with a wider spread compared to Biden counties. Biden counties cluster more tightly around a slightly younger median age of 38.6, though both groups overlap considerably. This suggests age composition may play a slight role in county voting differences, but further exploration is necessary.



The bar chart for age comparisons across county winner show that Biden counties overall are younger than Trump counties. For Biden winning counties, a higher percent of the population was in the 18-44 range. Trump winning counties have larger proportions starting in the 65+ age groups. The starkest differences are between the 20-34 age range leaning Biden and the 65-84 age range leaning Trump, suggesting a generational divide in candidate support between counties.

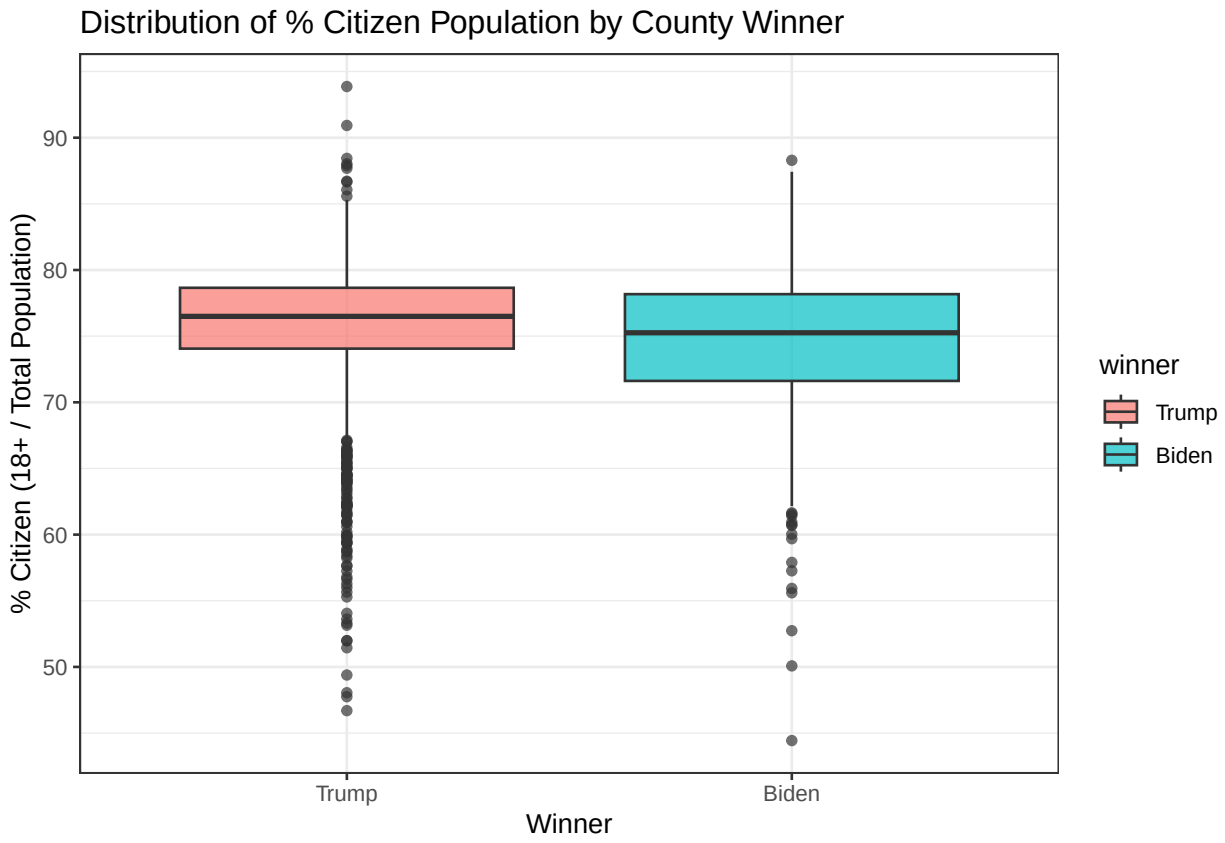


This stacked bar chart shows the average racial composition of counties won by Trump compared to Biden. Trump counties were overwhelming more White (Non-Hispanic) than Biden counties (80 to 56%). Biden counties were more diverse, with larger proportions of Hispanic, Black (Non-Hispanic), and Asian (Non-Hispanic) individuals. Note: Black, Asian, and White individuals who also identified as Hispanic were included only in Hispanic category, because the Census counts Hispanic as an ethnicity rather than racial category. The percentages do not add up to 100 due to Native/Other racial categories that were too small to be interpreted visually.

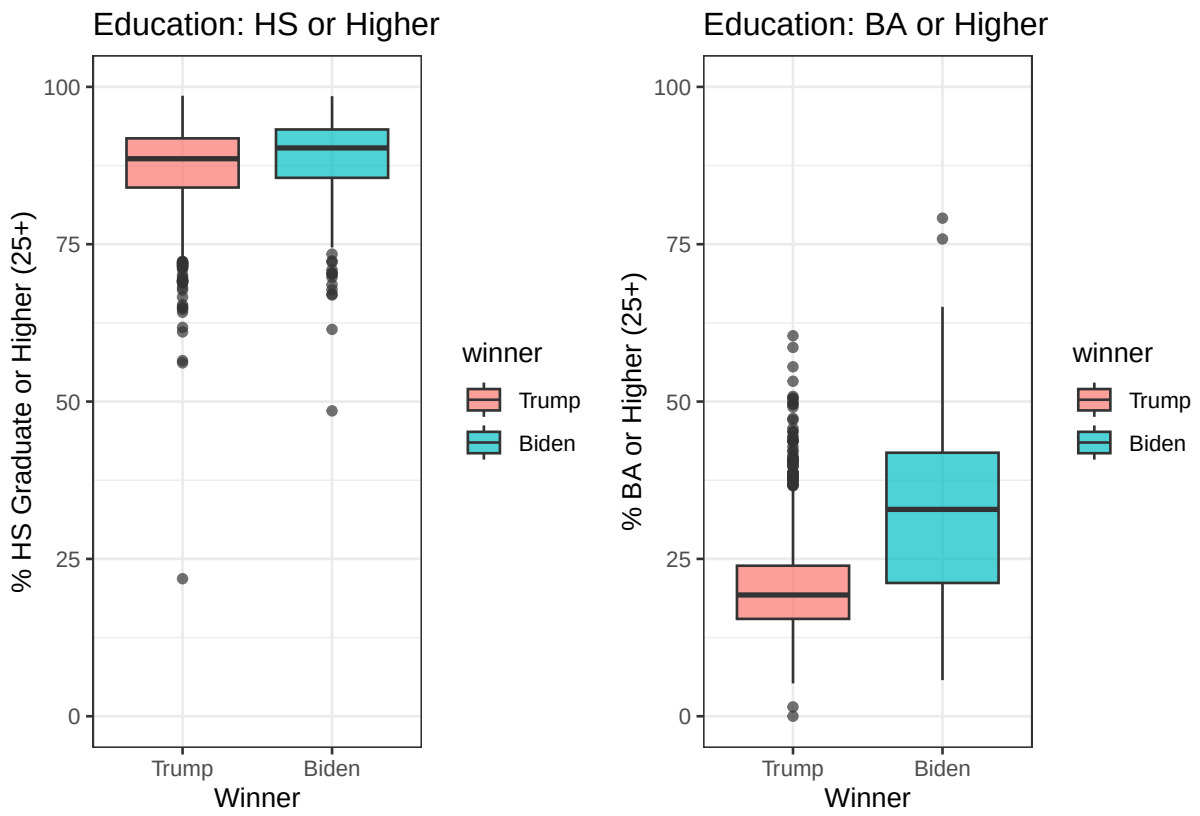


The boxplot shows that the median proportion of Hispanic individuals is higher in Biden winning counties than Trump winning counties. The median Hispanic proportions in the boxplot for Biden being lower that the mean Hispanic proportions in the Biden stacked bar chat above suggest that most counties have much lower Hispanic composition, while a few diverse counties bring the mean up.

The higher IQR of the Biden boxplot also shows that there is more range in Hispanic proportions, as some counties have small Hispanic population while others have very high percentages. For Trump counties, the IQR is tighter and lower, showing Hispanic presence in Trump counties is generally concentrated at low percentages.



The boxplot of citizen population percentages show that Biden counties tend to have a slightly lower percent of citizens overall and a wider spread. This aligns with the race analysis above showing that Biden counties have larger Hispanic and immigrant populations, which naturally lowers the proportion of citizens. Trump counties are more consistently majority citizen, with a few outliers driving the lower tail.

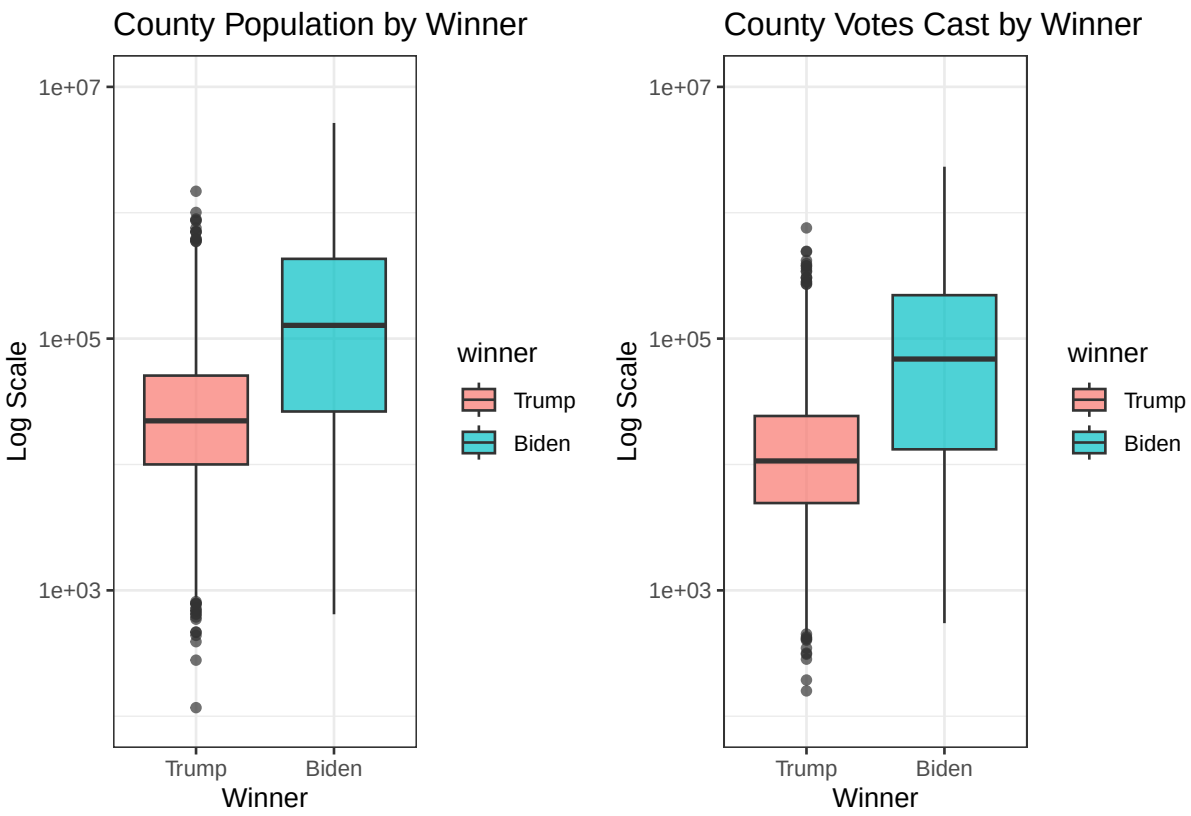


The boxplots show the education level proportions of those 25+, comparing Biden to Trump winning counties. For those who have their high school diploma or higher, the distributions are relatively similar, with both Trump and Biden winning counties having medians above 85%. However, when examining the percentage of individuals 25+ with atleast a bachelor's degree, Biden counties have a much higher median proportion of college graduates compared to Trump counties This suggests that while high school education is similar across groups, higher education levels were seen much more in Biden counties, so education is likely a strong predictor.

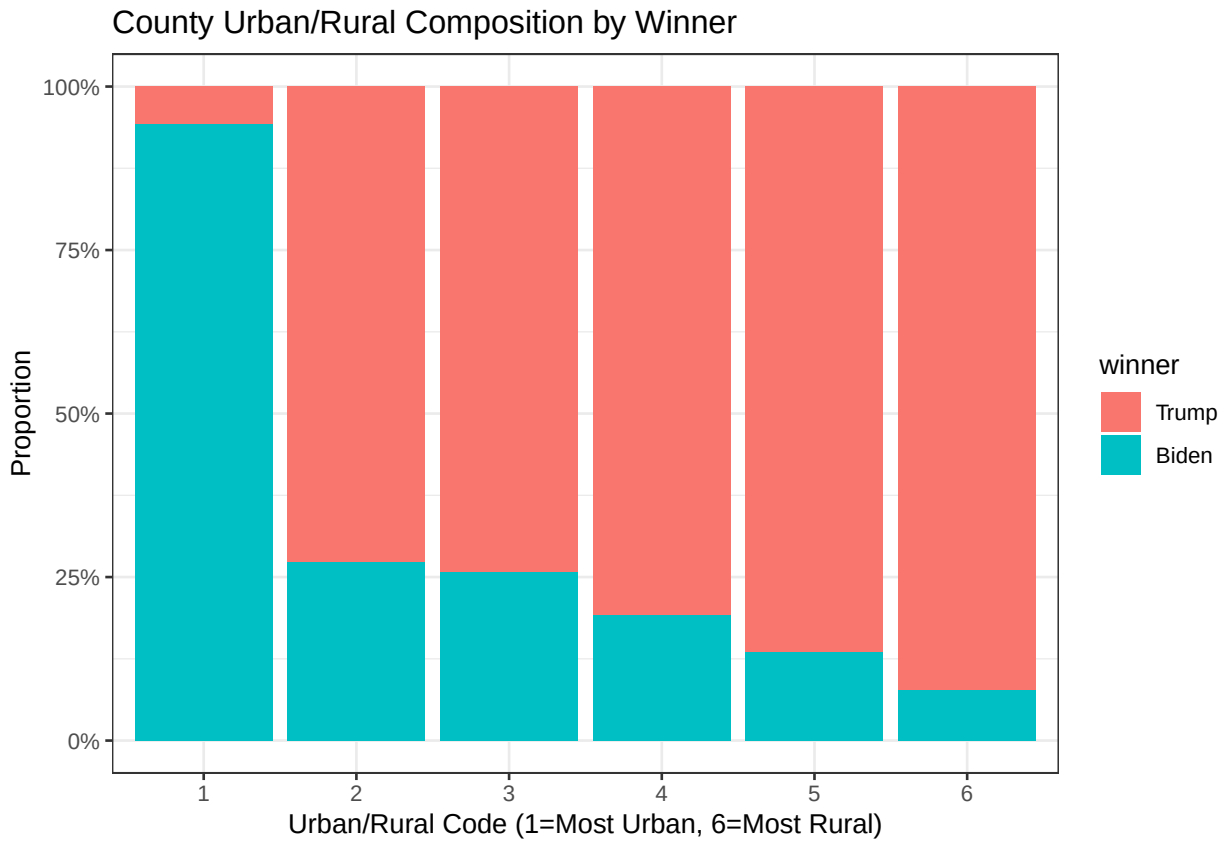
2.2 County Demographics Information

We next look at county level characteristics, such as population size, GDP, and income per capita, to see how broader geographic information correlate with voting outcomes.

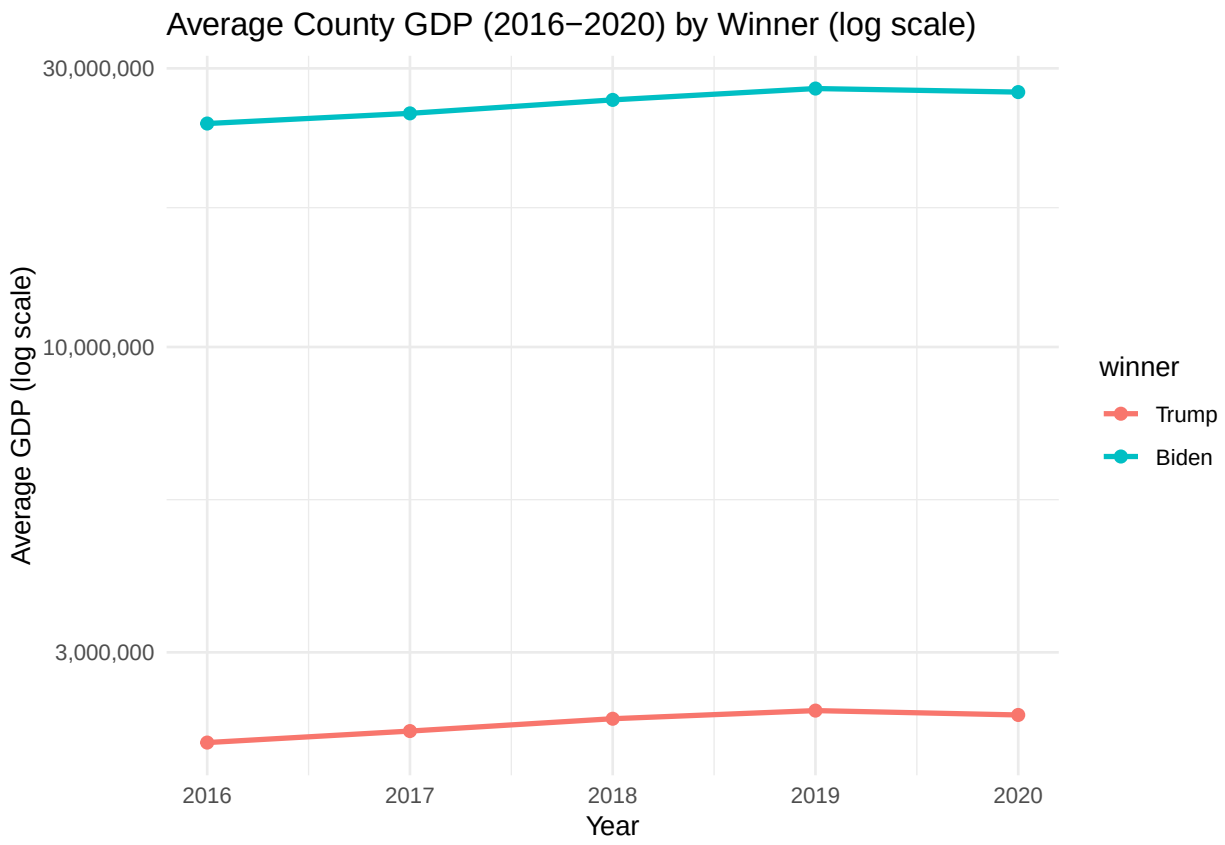
We applied log transformations to variables GDP and income to reduce skewness and allow for easier comparisons across counties.



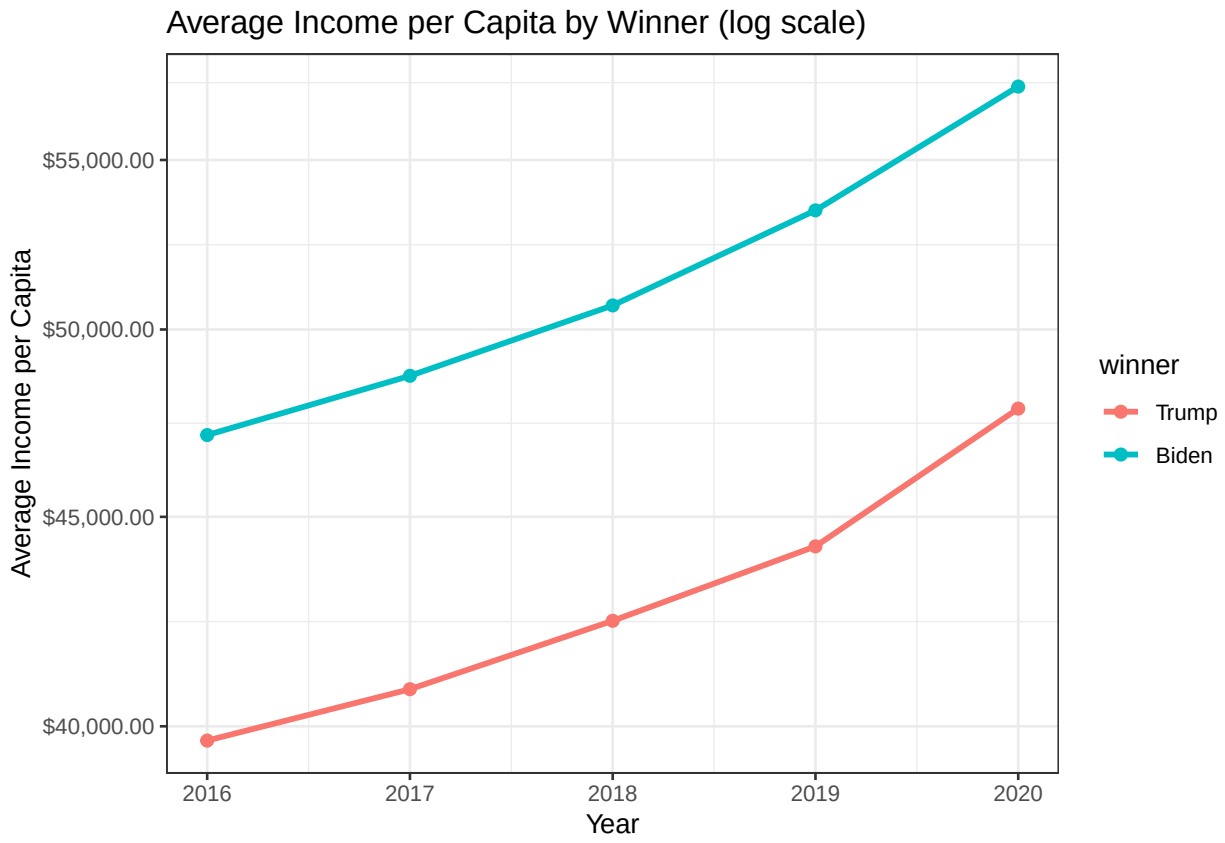
The boxplots show that Biden-winning counties tend to be significantly larger than Trump-winning counties, as seen in both the total population size or total votes cast (actual voter turnout). Biden winning counties center at a much higher population and vote count than Trump counties. This pattern reflects the urban vs rural divide as stated in our introduction, where Biden voters are concentrated in densely populated counties, while Trump wins in smaller, spread out areas.



This stacked bar chart shows the that the distribution of winners is clearly divided by the CDC's urban/rural codes. Biden dominates the vast majority (over 90%) of the most urban counties (1), while Trump dominates as counties become increasingly more rural (codes 2–6). This supports the conclusion that urban areas lean Democratic, while rural areas are strongly Republican. Although Trump dominates 5 of the 6 county codes, the large population numbers of the counties in code 1 balance out the rural support.



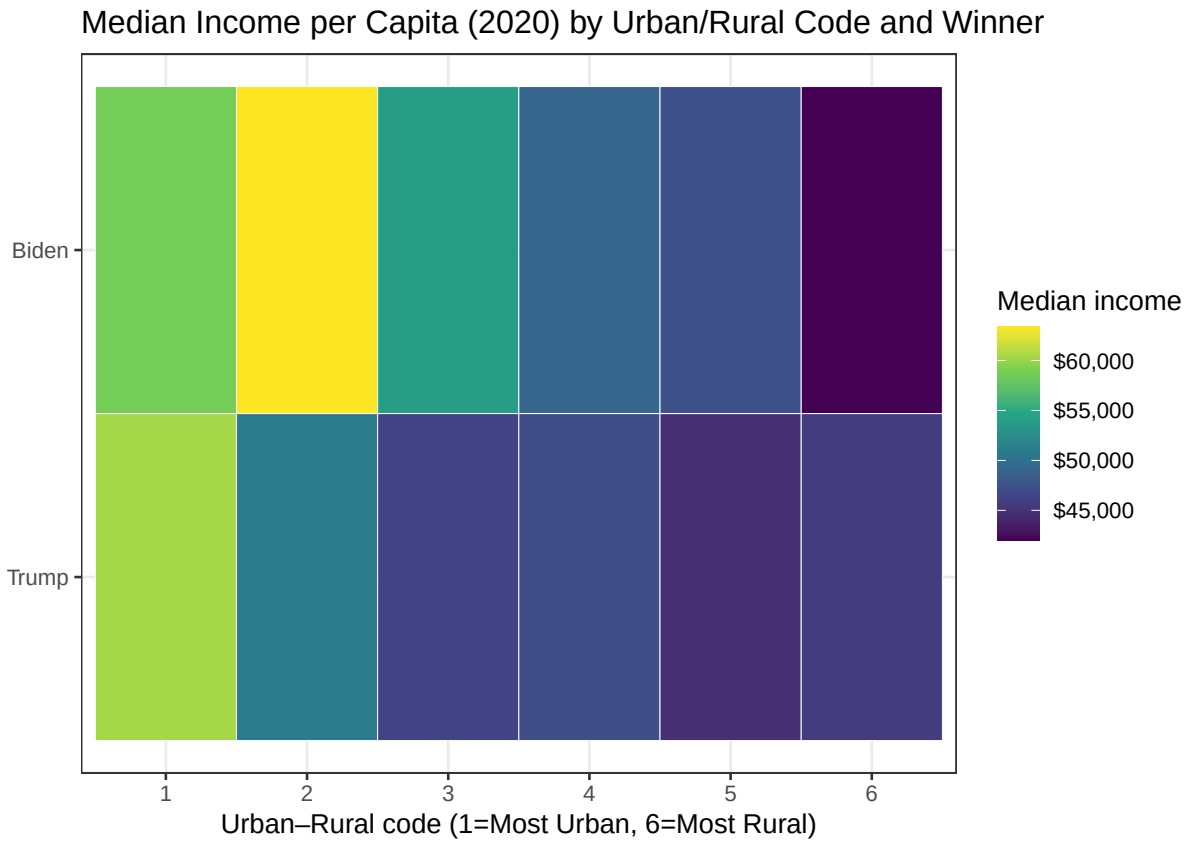
The line plot of average county GDP from 2016–2020 shows that Biden winning counties have much higher overall GDP levels than Trump counties for all 5 years. Both Biden and Trump counties follow a very similar pattern of steady GDP growth until 2019, followed by a slight decrease in 2020 (possibly explained by COVID-19). This suggests that overall GDP level is likely a strong predictor of election outcomes, however yearly changes in GDP following up to the election likely is not a major predictive factor.



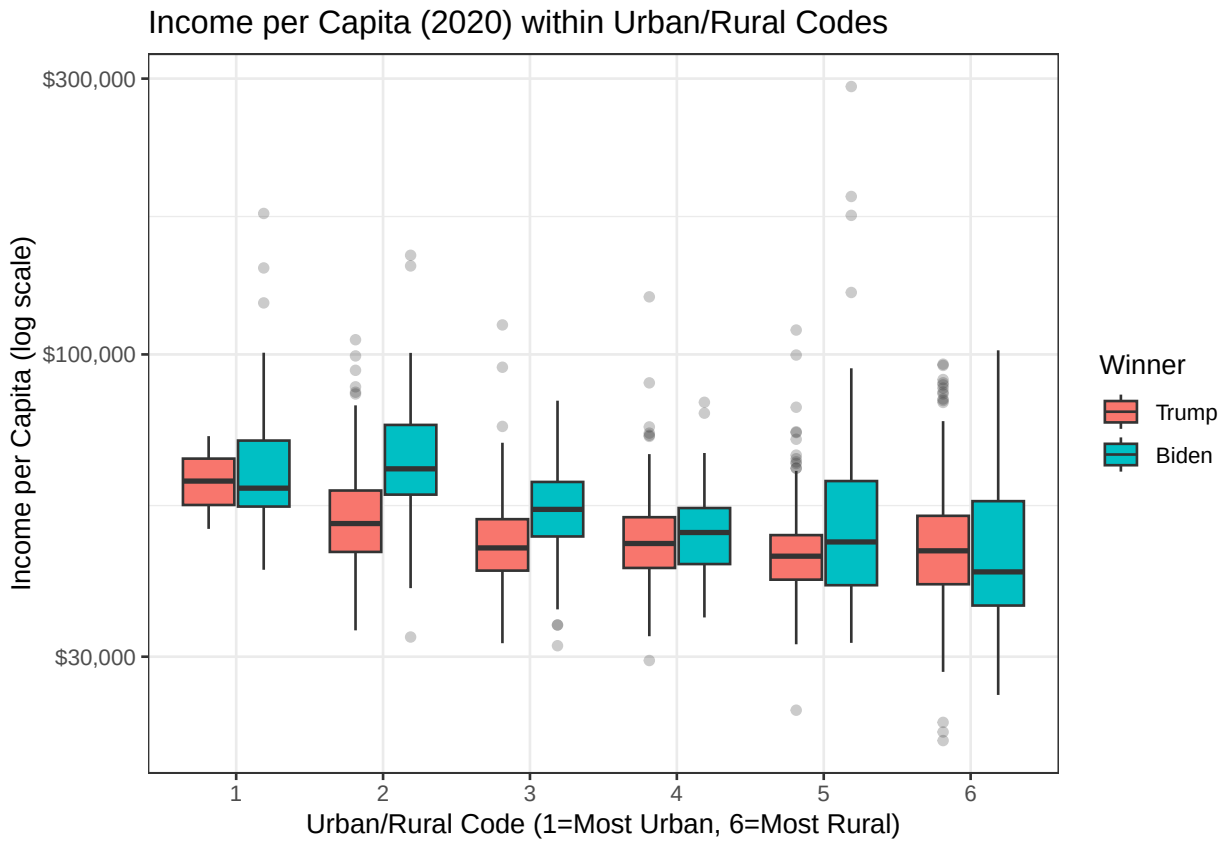
The line plot of average county income per capita from 2016–2020 shows that Biden winning counties have much higher overall income levels than Trump counties for all 5 years. We also used a log transformation to account for the large variation in incomes. Similar to GDP, both Biden and Trump counties follow a very similar pattern of steady income growth. This suggests that average income per capita is likely a strong predictor of election outcomes, however yearly changes in income are not predictive.

2.3 Interactions

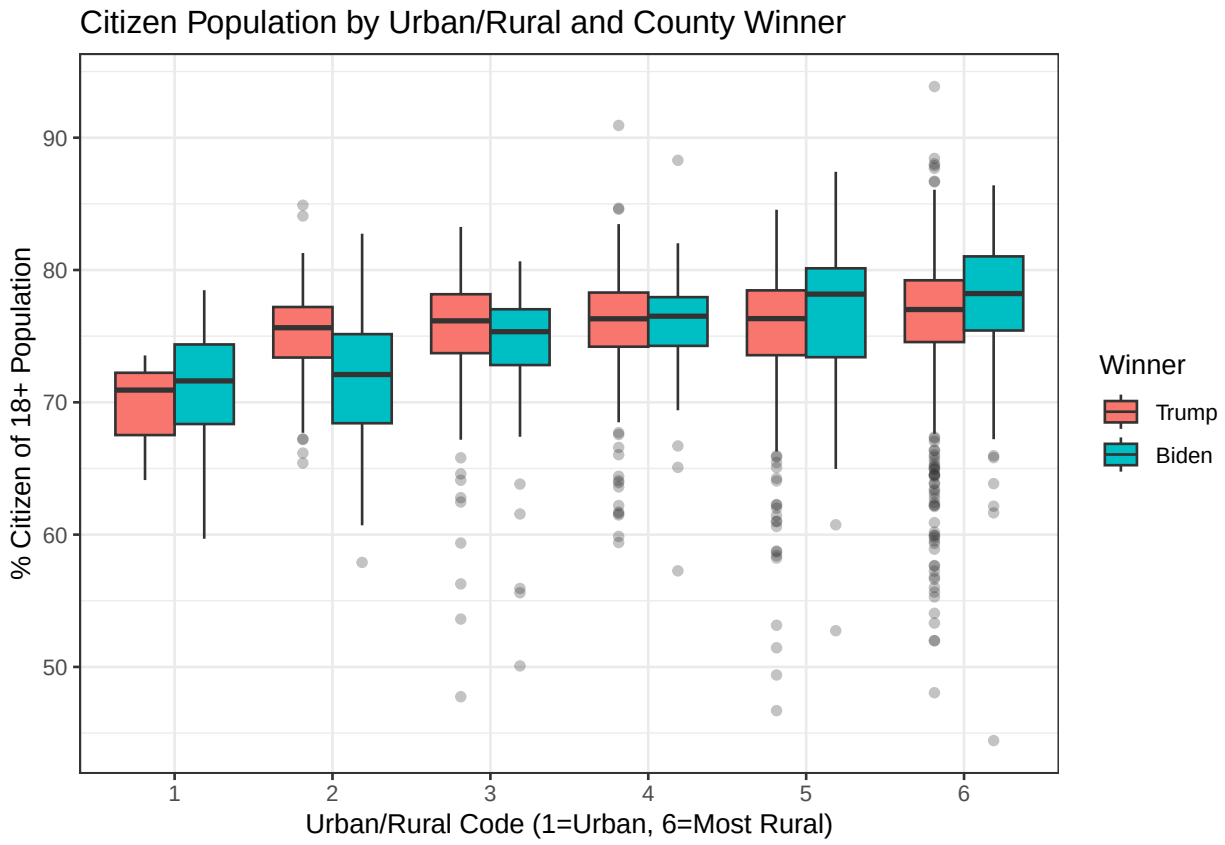
Finally, we explore the interactions between key variables, such as income, citizenship status, and race with urban/rural status, to see how different combinations of demographic and geographic factors compare between Biden and Trump winning counties.



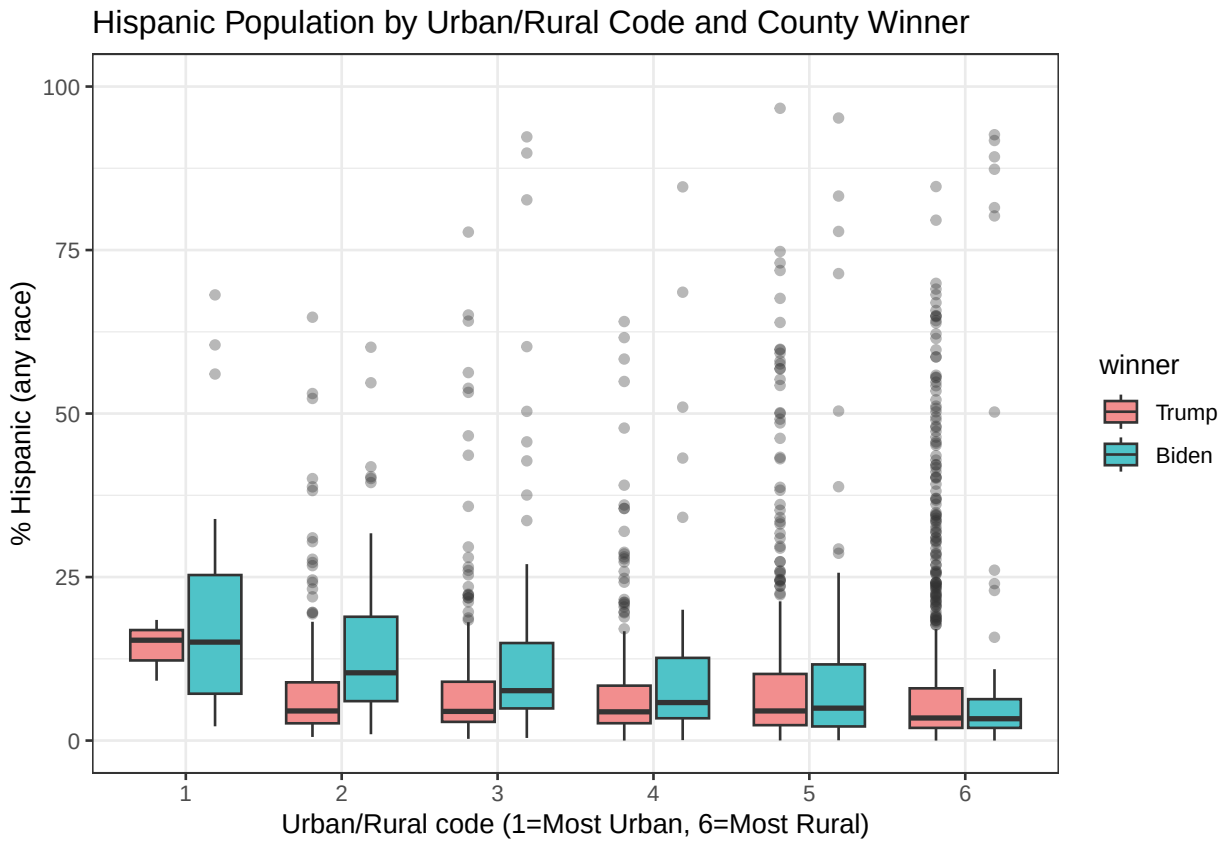
This heat map shows that Biden counties have higher median income across all levels of urban/rural status, with the gap especially shown in urban areas. Trump counties are on the lower end of the income distribution, except in code 6 of the most extreme rural areas.



This boxplot of the interaction between urban/rural geographic areas and income on election results show that Biden counties have higher median income across most levels of urban/rural status, with the gap especially shown in urban/rural mixed areas. Trump counties are on the lower end of the income distribution, except in code 6 of the most extreme rural areas, where Trump counties have significantly higher incomes. Extremly rural Biden counties of codes 5 and 6 have much higher IQRs, showing there is more variability in income levels. Note, this could be due to extremeley rural Biden counties being outliers.



These boxplots show the interaction between urban/rural geographic code and percent citizens (18+) of a county. In semi-rural areas (codes 2-4), Biden won counties where the share of citizens is lower, and Trump won counties where the share of citizens is higher. This is the opposite in extremely rural counties (5 and 6), where Trump won counties had a slightly lower share of citizens, which could be due to the fact that Biden won counties in extremely rural areas are outliers. However, it could also suggest that immigrants in urban-rural mixed areas favor Biden, while immigrant communities in rural/red states could favor Trump.



These boxplots shows the Hispanic percent of the population across counties by urban/rural codes. In the most urban counties (code 1), Biden winning counties have substantially higher Hispanic populations. In semi-rural counties (codes 2–4), the gap narrows but Biden counties still generally have higher Hispanic representation. In the most rural counties (codes 5–6), both Trump and Biden counties show relatively low Hispanic percentages. This instead suggests that the influence of Hispanic and/or immigrant populations on election outcomes is strongest in urban and semi-rural areas and helps Biden's chances, while in rural areas other factors must play a predictive role.

3 Preprocessing / Recipes

3.1 Identifier Removal

We removed identifiers such as county name and id, along with other non-predictive text columns. These variables carry no predictive signal and may cause data leakage.

3.2 Feature Engineering

We created several derived features to capture meaningful information. Turnout was defined as total_votes divided by total population (x0001e), clipped between 0 and 1 to ensure valid proportions. Log_total_votes was generated using log1p to stabilize variance across counties of different size. In addition, census estimate variables ending with "_e" were divided by population to produce per-capita demographic and socioeconomic rates, enabling fair comparisons across counties.

3.3 Cleaning and Imputation

Columns with constant or near-constant values were dropped because they contain no useful information and can mislead the model. Missing numeric values were imputed with the median of each variable from the training data. The same medians were applied to the test data for consistency. Median imputation is simple, resistant to skew, and prevents loss of observations.

3.4 Handling Class Imbalance

Trump counties were more numerous than Biden counties. To address this, we computed class weights as total_samples / (2 $\times$ class_count) and supplied them to the Random Forest training procedure. This reduced bias toward the majority class and improved balance between classes.

Summary.

These preprocessing steps normalized features, handled missing values, and corrected imbalance. They provided a consistent dataset for cross-validation and supported the tuned Random Forest selected as the final model.

4. Candidate models / Model evaluation / tuning

4.1 Candidate Models

We evaluated at least five candidate models using the same preprocessing pipeline (median imputation of missing values, normalization of numeric variables, and engineered ratio features where applicable). Each model was trained with v-fold cross-validation (v = 5), stratified on the outcome variable to ensure balanced folds. Below are the candidate models and their core configurations.

Table 1: Table 1. Candidate models (type, engine, recipe, and hyperparameters).

Model_ID	Type	Engine	Recipe	Key_Hyperparameters
RF-Recipe	Random Forest	ranger	county_recipe	mtry=10, trees=500, min_n=5
XGB-Recipe	XGBoost	xgboost	county_recipe	trees=500, depth=6, lr=0.1, mtry=10
GB-Recipe	Gradient Boosting	xgboost	county_recipe	trees=500, depth=6, lr=0.1, min_n=5
GB-Tuning	Gradient Boosting	xgboost	N/A	random search then focused; best mtry=15, depth=4, lr=1.128
RF-Final	Random Forest	ranger	engineered	grid over mtry/min.node.size/sample.fraction/trees; class weights; threshold tuning

4.2 Model Performance

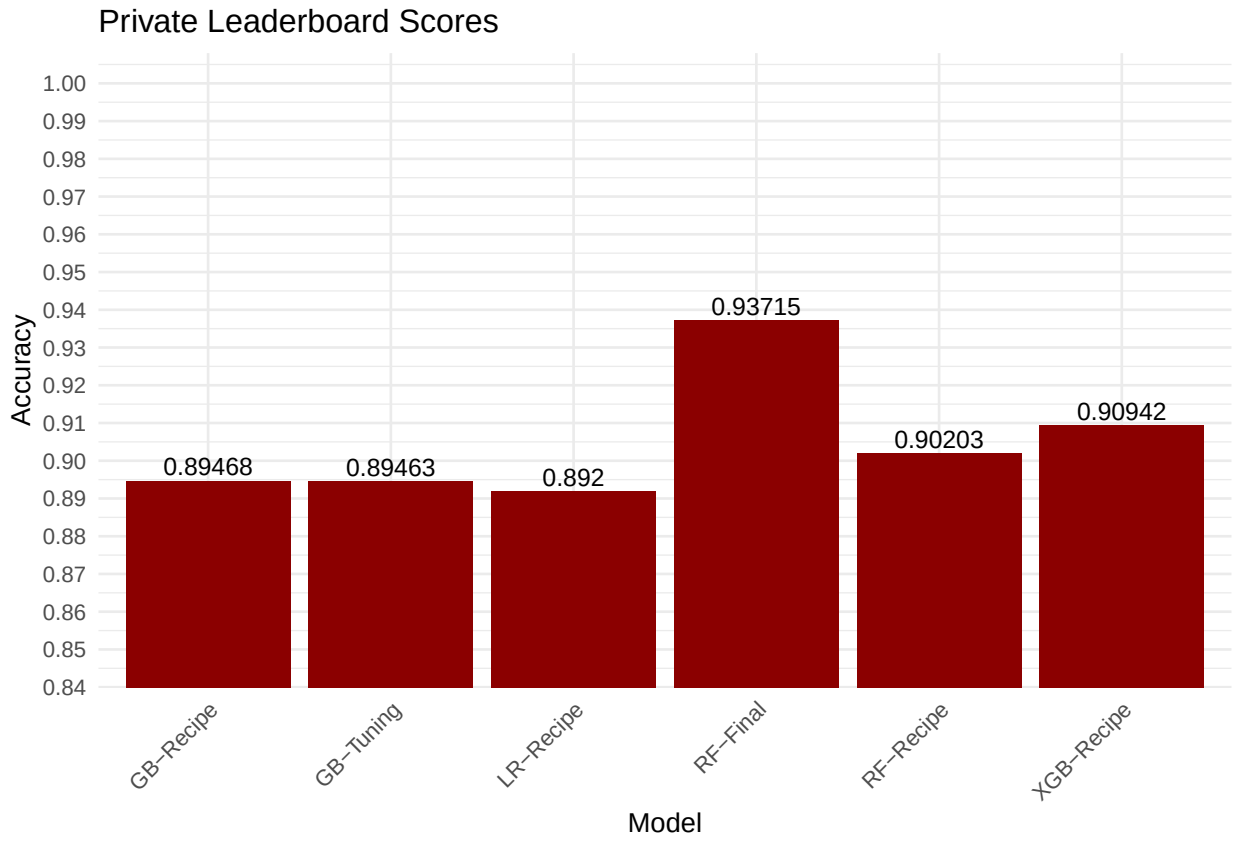
Performance was evaluated with cross-validation and reported on both the public and private Kaggle leaderboards. Standard errors (SE) are estimated from the cross-validation folds. Gradient boosting models achieved very high public scores but showed larger private drops, while Random Forest models were more stable. Logistic Regression was consistent but weaker. The final tuned Random Forest achieved the highest balance between public and private accuracy.

Table 2: Table 2. Leaderboard performance with estimated standard errors.

Model_ID	Public_Score	Private_Score	SE
RF-Recipe	0.92468	0.90203	0.003
XGB-Recipe	0.94560	0.90942	0.002
GB-Recipe	0.94979	0.89468	0.004
GB-Tuning	0.92887	0.89463	0.003
LR-Recipe	0.89200	0.89200	0.004
RF-Final	0.94979	0.93715	0.002

4.3 Evaluation and Tuning

To highlight small differences, the y-axis is narrowed and exact values are printed above each bar.



4.4 Evaluation and Tuning Summary

We tested six models with 5-fold stratified cross-validation. Logistic Regression was consistent but weak. Gradient Boosting and XGBoost led the public board but dropped on the private set, showing overfitting. Random Forest was more stable with smaller gaps. RF hyperparameters (mtry, min.node.size, sample.fraction, num.trees) were tuned with class weights. Gradient Boosting used random search plus grid refinement. The final tuned Random Forest added threshold optimization. It scored 0.94979 public and 0.93715 private, giving the best balance of accuracy and generalization.

5. Discussion of Final Model

The final predictive model chosen was a Random Forest classifier implemented with the ranger package in R. Random Forest was selected because it can handle high-dimensional tabular data, is robust to collinearity, and captures nonlinearities and interactions without explicit specification.

5.1 Model Selection

- Data preparation and feature engineering: Removed identifiers and non-predictive text columns. Created turnout (votes / population, clipped 0–1), log-transformed total votes, and per-capita demographic/economic rates. Dropped constant or near-zero-variance columns. Missing numeric values were median-imputed.
- Class imbalance: Applied class weights (total samples / (2 × class count)) to balance Biden vs. Trump counties.
- Model tuning: Used stratified 5-fold cross-validation with a grid search over
  - mtry = {sqrt(p), p/4, p/2}
  - min.node.size = {3, 5, 10}
  - sample.fraction = {0.70, 0.85, 1.00}
  - num.trees = {600, 900}
    Out-of-fold predictions were used for evaluation.
- Threshold optimization: Instead of fixing the decision threshold at 0.5, we swept 0.30–0.70 and selected the cutoff that maximized accuracy.
- Final fit: Retrained the tuned Random Forest on the entire dataset with best hyperparameters and class weights. Predictions on the test set were thresholded and saved as submission_tuned.csv. The model achieved 0.94979 public and 0.93715 private scores.

5.2 Strengths

- Captures nonlinear relationships and interactions automatically
- Robust to collinearity among predictors
- Class weights improved balance across classes
- Stratified folds and threshold tuning gave stable validation results
- Feature engineering normalized for county size, improving comparability

5.3 Weaknesses and Limitations

- Limited interpretability compared to linear models

- Threshold choice may not generalize if class balance shifts
- Possible target leakage if vote totals come from the same election
- Spatial correlation among counties may inflate CV results
- Median imputation ignores predictor relationships

5.4 Possible Improvements

1. Validation: Use spatially grouped or time-aware CV for more realistic generalization.
2. Modeling: Blend Random Forest with boosted trees or calibrated logistic regression; increase number of trees; apply probability calibration.
3. Features: Incorporate historical voting trends, economic indicators, and spatial variables.
4. Explainability: Use permutation importance, partial dependence, or SHAP values for more reliable interpretation.

5.5 Conclusion

The tuned Random Forest with class weights and threshold adjustment provided the best balance of accuracy and robustness. With improved validation strategies, richer features, and stronger explainability, its reliability and practical value could be further enhanced.

Appendix 1

```
library(ranger)
library(tidyverse)
library(readr)

set.seed(42)

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

train$winner <- factor(train$winner, levels = c("Biden","Trump"))

drop_cols <- intersect(names(train), c("name"))
trainX <- train[, setdiff(names(train), c("winner","id", drop_cols)), drop = FALSE]
testX  <- test[,  setdiff(names(test),  c("id",        drop_cols)),  drop = FALSE]

# --------------------------
# 2) Feature engineering
# --------------------------
# Heuristics that usually help on county data:
# - turnout = total_votes / total population (x0001e)  (clip to [0,1])
# - log_total_votes = log1p(total_votes)
# - convert count estimates to rates by dividing by x0001e

safe_div <- function(num, den) {
  den2 <- ifelse(is.na(den) | den <= 0, NA, den)
  num / den2
}

tot_pop_col <- "x0001e"          # total population estimate column name (lowercase)
has_tot_pop <- tot_pop_col %in% names(trainX)

if (has_tot_pop) {
  # turnout
  trainX$turnout <- pmin(pmax(safe_div(trainX$`total_votes`, trainX[[tot_pop_col]]), 0), 1)
  testX$turnout  <- pmin(pmax(safe_div(testX$`total_votes`,  testX[[tot_pop_col]]),  0), 1)

  # log total votes
  trainX$log_total_votes <- log1p(trainX$`total_votes`)
  testX$log_total_votes  <- log1p(testX$`total_votes`)

  # Convert *_e counts to rates (divide by total pop), keep original columns too
  is_estimate_col <- function(nm) grepl("e$", nm, ignore.case = TRUE)
  e_cols <- setdiff(names(trainX)[sapply(names(trainX), is_estimate_col)], tot_pop_col)

  ratefy <- function(df) {
    out <- df
    for (nm in e_cols) {
      out[[paste0(nm, "_rate")]] <- safe_div(out[[nm]], out[[tot_pop_col]])
```

```
    }
    out
  }
  trainX <- ratefy(trainX)
  testX  <- ratefy(testX)
} else {
  trainX$log_total_votes <- log1p(trainX$`total_votes`)
  testX$log_total_votes  <- log1p(testX$`total_votes`)
}

# --------------------------
# 3) Remove constant/near-zero-variance columns
# --------------------------
is_constant <- function(x) is.numeric(x) && length(unique(x[!is.na(x)])) <= 1
const_cols <- names(trainX)[sapply(trainX, is_constant)]
if (length(const_cols)) {
  trainX <- trainX[, setdiff(names(trainX), const_cols), drop = FALSE]
  testX  <- testX[,  setdiff(names(testX),  const_cols), drop = FALSE]
}

# --------------------------
# 4) Median impute numeric NAs
# --------------------------
impute_median <- function(df, med = NULL) {
  num <- sapply(df, is.numeric)
  if (is.null(med)) {
    med <- vapply(df[, num, drop = FALSE],
                  function(x) median(x, na.rm = TRUE),
                  numeric(1))
  }
  for (nm in names(med)) {
    nas <- is.na(df[[nm]])
    if (any(nas)) df[[nm]][nas] <- med[[nm]]
  }
  list(data = df, med = med)
}

imp_tr <- impute_median(trainX)
trainX <- imp_tr$data
testX  <- impute_median(testX, imp_tr$med)$data

train_df <- data.frame(winner = train$winner, trainX, check.names = FALSE)

# --------------------------
# 5) Stratified K-folds
# --------------------------
make_folds <- function(y, k = 5, seed = 42) {
  set.seed(seed)
  idx_pos <- which(y == levels(y)[2])
  idx_neg <- which(y == levels(y)[1])
  folds <- vector("list", k)
  sp_pos <- sample(rep(1:k, length.out = length(idx_pos)))
  sp_neg <- sample(rep(1:k, length.out = length(idx_neg)))
```

```r
  for (i in 1:k) {
    folds[[i]] <- c(idx_pos[sp_pos == i], idx_neg[sp_neg == i])
  }
  folds
}

y <- train_df$winner
folds <- make_folds(y, k = 5, seed = 42)

# --------------------------
# 6) Metrics & threshold search
# --------------------------
pred_to_label <- function(p_trump, thr = 0.5) ifelse(p_trump >= thr, "Trump", "Biden")

acc_bal <- function(y_true, y_pred) {
  tp <- sum(y_true == "Trump" & y_pred == "Trump")
  tn <- sum(y_true == "Biden" & y_pred == "Biden")
  fp <- sum(y_true == "Biden" & y_pred == "Trump")
  fn <- sum(y_true == "Trump" & y_pred == "Biden")
  sens <- ifelse((tp + fn) == 0, 0, tp / (tp + fn))
  spec <- ifelse((tn + fp) == 0, 0, tn / (tn + fp))
  (sens + spec) / 2
}

# --------------------------
# 7) Grid search with OOF probabilities
# --------------------------
tbl <- table(train_df$winner)
cw  <- sum(tbl) / (2 * tbl)
class_wts <- c(Biden = as.numeric(cw["Biden"]),
               Trump = as.numeric(cw["Trump"]))

p <- ncol(trainX)
grid <- expand.grid(
  mtry = unique(pmax(1, round(c(sqrt(p), p/4, p/2)))),
  min.node.size = c(3, 5, 10),
  sample.fraction = c(0.7, 0.85, 1.0),
  num.trees = c(600, 900),
  KEEP.OUT.ATTRS = FALSE
)

best <- list(score = -Inf, thr = 0.5, params = NULL, metric = "accuracy")

for (gi in seq_len(nrow(grid))) {
  g <- grid[gi, ]
  oof <- rep(NA_real_, nrow(train_df))
  for (i in seq_along(folds)) {
    idx_valid <- folds[[i]]
    idx_train <- setdiff(seq_len(nrow(train_df)), idx_valid)

    fit <- ranger(
      winner ~ .,
      data = train_df[idx_train, ],
```

```
      classification = TRUE,
      probability   = TRUE,
      num.trees     = g$num.trees,
      mtry          = g$mtry,
      min.node.size = g$min.node.size,
      sample.fraction = g$sample.fraction,
      class.weights = class_wts,
      importance    = "impurity",
      seed = 100 + i
    )
    prob <- predict(fit, data = train_df[idx_valid, -1, drop = FALSE])$predictions
    p_trump <- prob[, "Trump"]
    oof[idx_valid] <- p_trump
  }

  # Threshold tuning on OOF
  thr_seq <- seq(0.3, 0.7, by = 0.01)
  y_true <- as.character(train_df$winner)

  metric <- "accuracy"
  scores <- sapply(thr_seq, function(t) mean(pred_to_label(oof, t) == y_true))

  best_i <- which.max(scores)
  if (scores[best_i] > best$score) {
    best$score  <- scores[best_i]
    best$thr    <- thr_seq[best_i]
    best$params <- g
    best$metric <- metric
  }
  cat(sprintf("Grid %d/%d: mtry=%d, min_n=%d, frac=%.2f, trees=%d | OOF-%s=%.4f @ thr=%.2f\n",
              gi, nrow(grid), g$mtry, g$min.node.size, g$sample.fraction, g$num.trees,
              best$metric, max(scores), thr_seq[best_i]))
}

cat("\nBest OOF ", best$metric, " = ", sprintf("%.4f", best$score),
    " at threshold = ", sprintf("%.2f", best$thr),
    " with params: ", paste(names(best$params), best$params, collapse = ", "), "\n", sep = "")

# --------------------------
# 8) Refit best model on full training
# --------------------------
g <- best$params
final_rf <- ranger(
  winner ~ .,
  data = train_df,
  classification = TRUE,
  probability   = TRUE,
  num.trees     = g$num.trees,
  mtry          = g$mtry,
  min.node.size = g$min.node.size,
  sample.fraction = g$sample.fraction,
  class.weights = class_wts,
  importance    = "impurity",
```

```
  seed = 777
)

# --------------------------
# 9) Predict test & write submission
# --------------------------
prob_test <- predict(final_rf, data = testX)$predictions
p_trump_test <- prob_test[, "Trump"]
pred_test <- ifelse(p_trump_test >= best$thr, "Trump", "Biden")

submission <- data.frame(
  id = test$id,
  winner = pred_test,
  check.names = FALSE
)

write.csv(submission, "submission_tuned.csv", row.names = FALSE)
cat("Wrote: submission_tuned.csv\n")
head(submission)
```

Appendix 2

In our team, Kelsie prepared the Introduction and conducted the Exploratory Data Analysis. Abdul wrote the Discussion of the Final Model section. Dillon and Mehdi worked on the Candidate Models and Model Evaluation/Tuning section. Choi was responsible for the Preprocessing/Recipes section, compiling and editing the full report, and managing the Kaggle competition submissions.