Stats 101C Classification Project Report

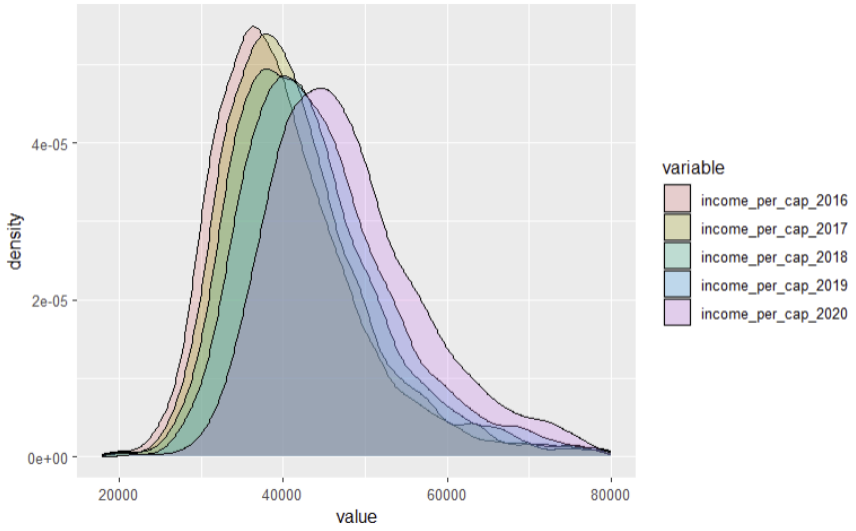
Maxwell Chu, Bella Gordon, Tran Nguyen, Vanda Suklar

Table of Contents

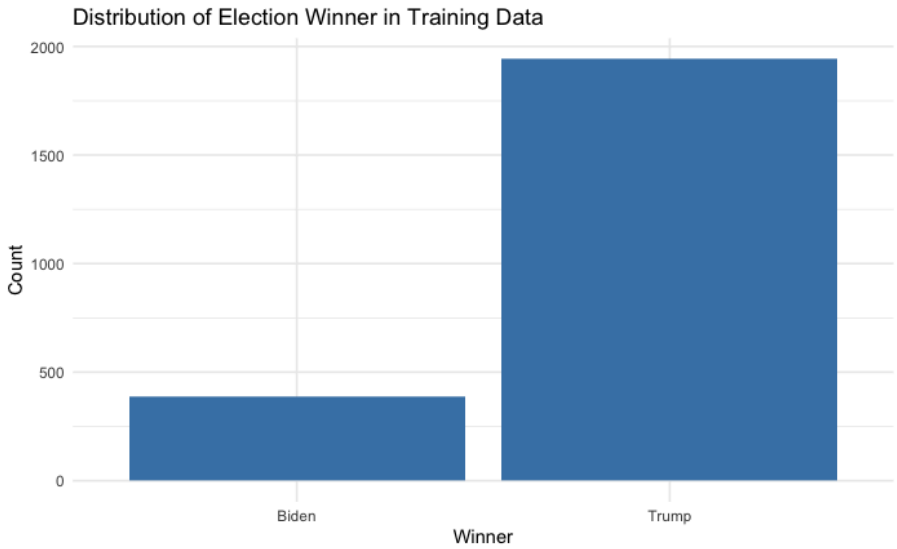
Introduction

The 2020 US Presidential Election was a pivotal moment in American politics, shaped by a complex interplay of demographic, economic, and social factors across the nation's diverse counties. This project aims to predict county-level election outcomes using a range of variables associated with voting behavior. Key predictors include demographic composition (age, race, ethnicity), educational attainment, and economic indicators such as income per capita and GDP. Additionally, historical voting patterns, the urban-rural divide, and population density are considered crucial factors. Employment rates and industry presence may also influence political leanings. By analyzing these multifaceted variables, we seek to uncover the underlying patterns that shaped the election's outcome, providing insights into the American electoral dynamics at the county level.

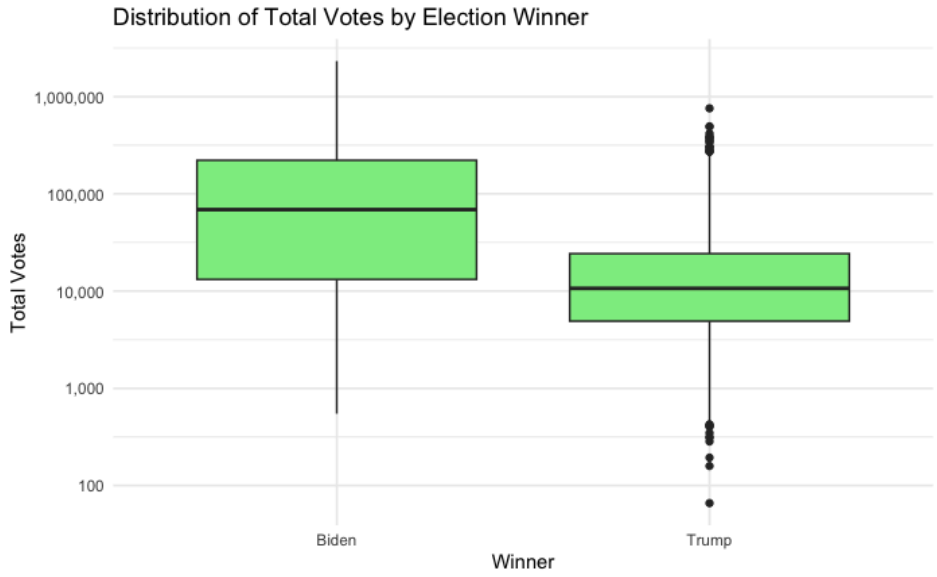
Exploratory Data Analysis



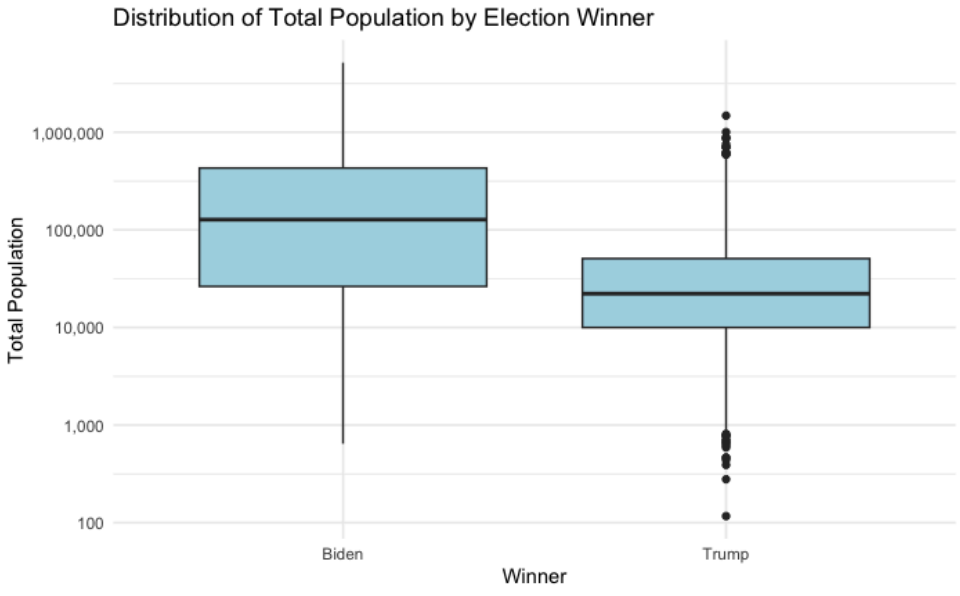
Visual 1: This density plot shows the distribution of income per capita from 2016 to 2020. Each curve represents a different year, highlighting the right-skewed distribution, with most values clustered around a similar range and a few counties with higher incomes. The plot indicates slight increases in income over time, providing insight into economic trends in the dataset.



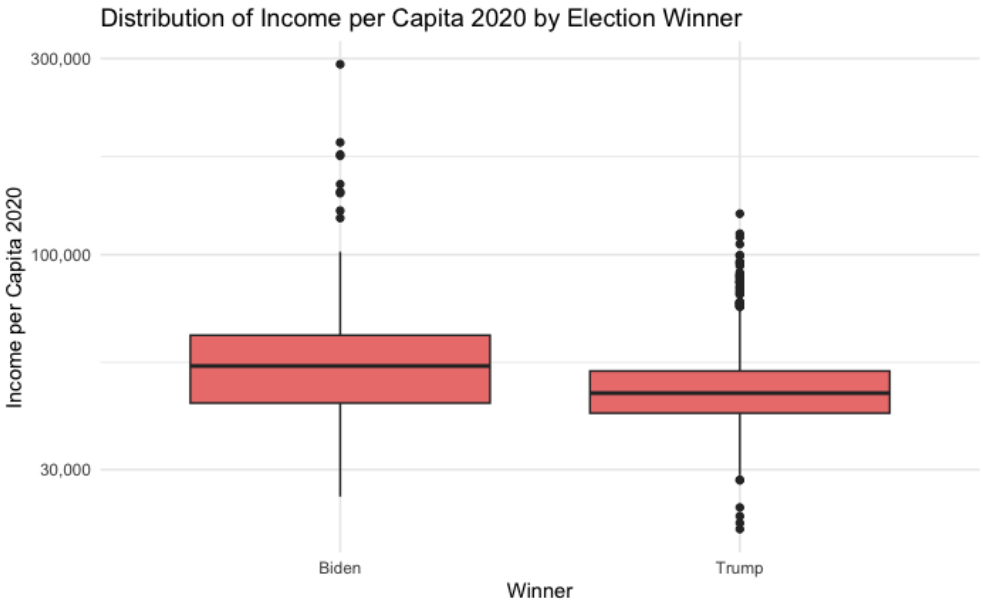
Visual 2: This bar chart shows the count of counties won by each candidate. The significant disparity, with more counties won by Trump, highlights the distribution of electoral support across different regions. This visualization sets the stage for understanding the data's class imbalance, a crucial factor in classification modeling.



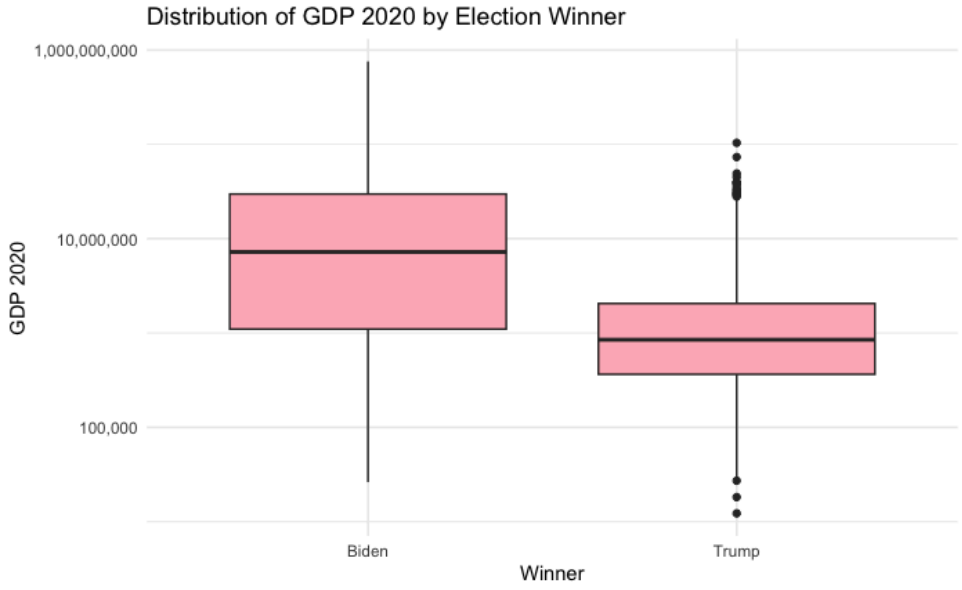
Visual 3: The box plot illustrates the distribution of total votes across counties by election winner, using a logarithmic scale to manage large value ranges. It reveals that counties with higher voter turnout tend to favor Biden, suggesting a potential correlation between voter turnout and election outcomes.



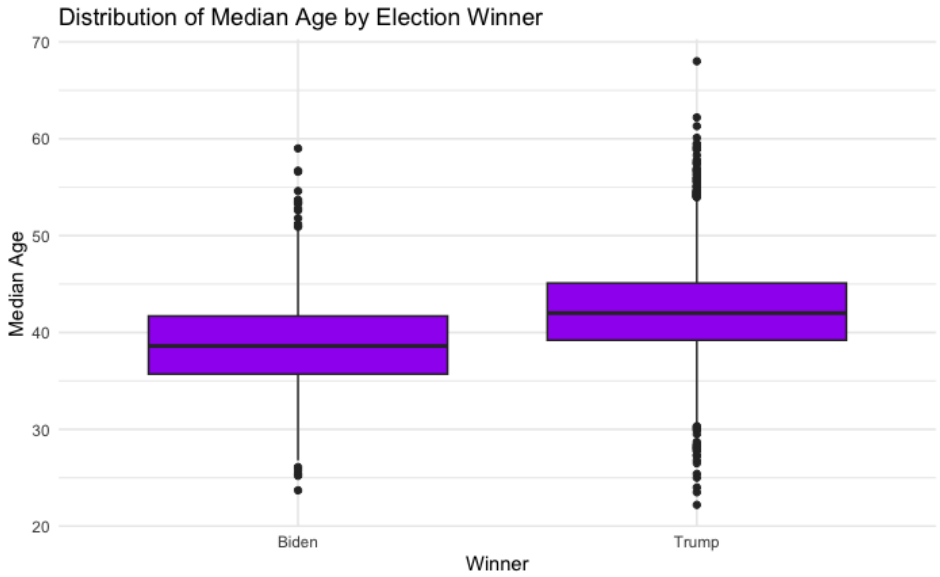
Visual 4: This box plot displays the distribution of the total population in counties by election winner, also on a logarithmic scale. It indicates that counties with larger populations are more likely to vote for Biden, suggesting that urban areas with denser populations may lean Democratic.



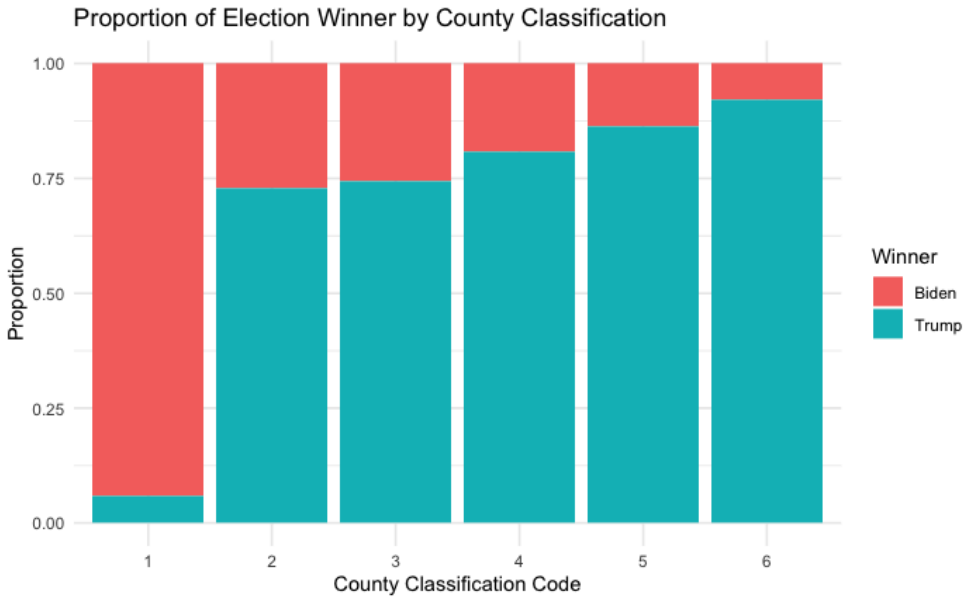
Visual 5: The visualization depicts the income per capita distribution by election winner, showing that counties with higher income levels tend to vote for Biden. This trend suggests economic factors, such as wealth and income disparities, may influence political preferences and electoral outcomes.



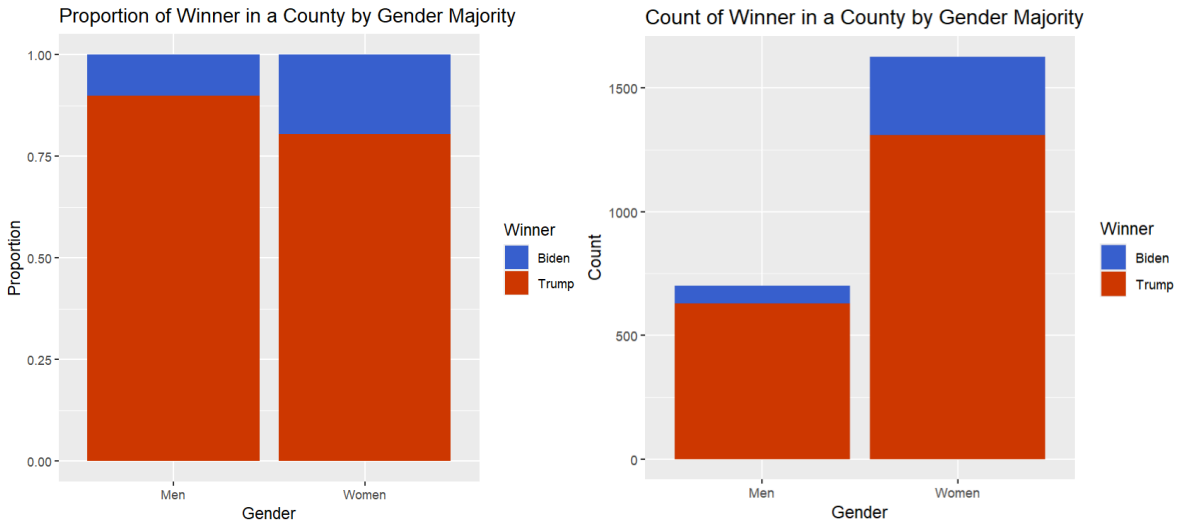
Visual 6: This box plot compares GDP distribution across counties by election winner, revealing that Biden-supporting counties generally have higher GDP. The use of a logarithmic scale helps illustrate the wide range of economic conditions, highlighting the influence of economic prosperity on voting behavior.



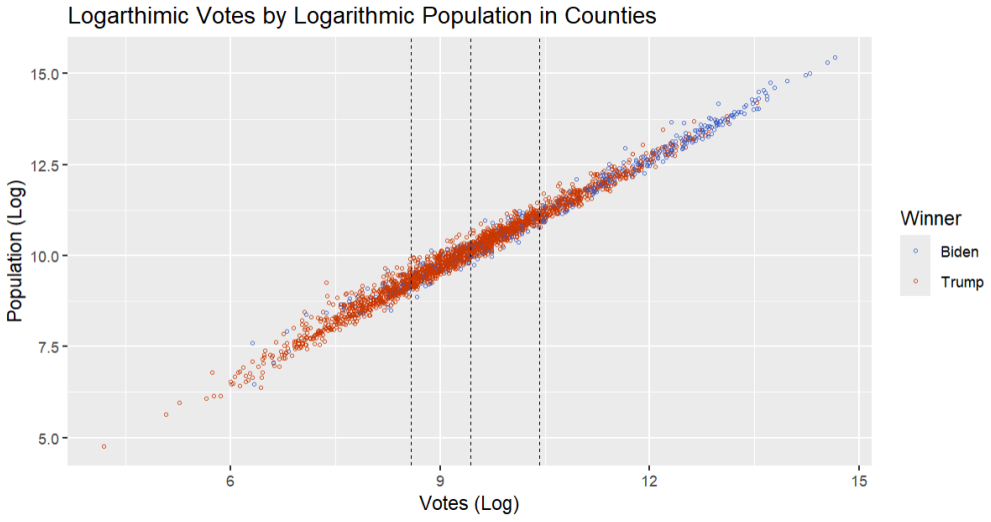
Visual 7: The box plot shows the distribution of median age in counties by election winner. The plot suggests that Trump-supporting counties tend to have a slightly older median age, indicating age demographics' potential role in shaping political affiliations and voting behavior.



Visual 8: This stacked bar chart represents the proportion of election winners across different county classification codes. It highlights the political leanings in various county types, with rural areas favoring Trump and urban areas leaning towards Biden. This visualization underscores the urban-rural divide in American politics.

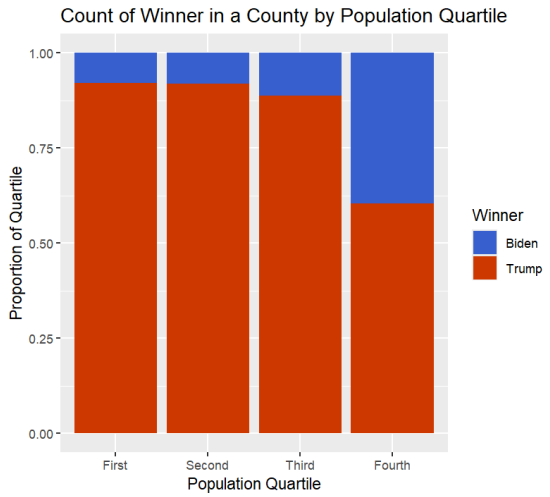


Visual 9: These two bar plots depict proportion and amount respectively of the winner in male-majority counties versus female-majority counties. The left bar graph shows us that Biden won in female-majority counties about twice as much as in male-majority counties. When looking at the right bar graph, we also see that women are the majority in counties over twice as much as men, which could have an impact on the results we see in the left bar graph. In combination, these two bar graphs indicate a potential btu inconclusive impact of the gender makeup of a county on the winner in that county.

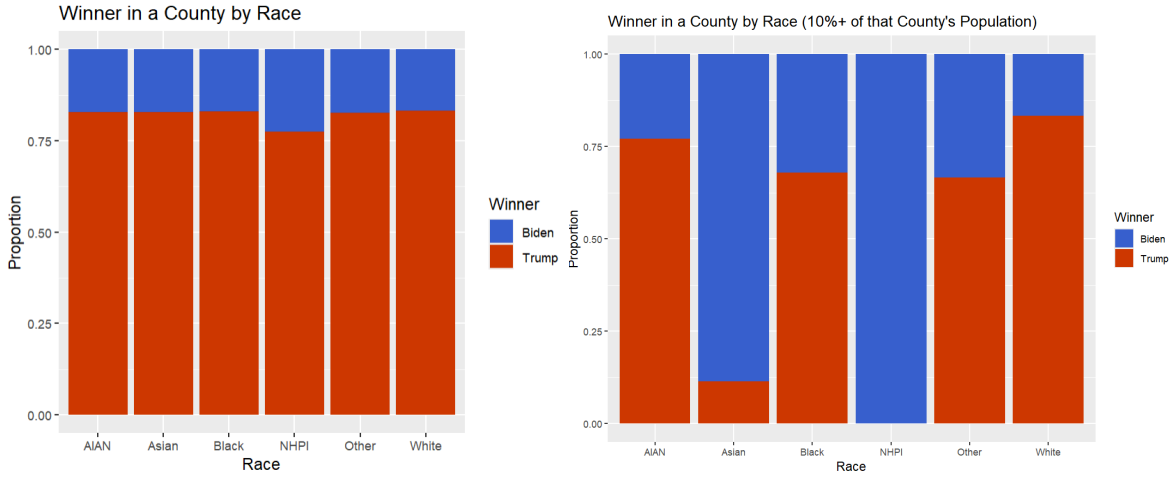


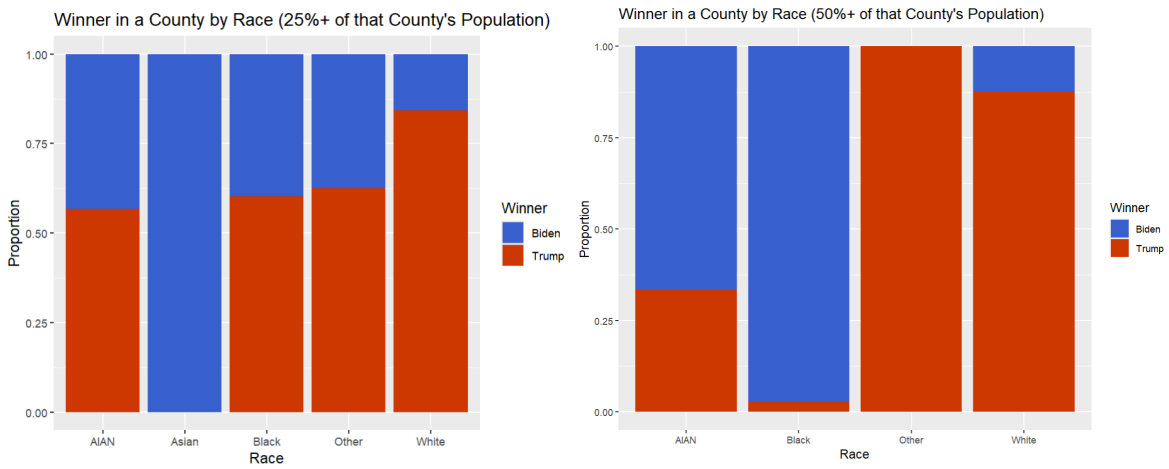
Visual 10: This scatterplot visualizes the logarithmic population and logarithmic votes counted in a county with each point colored red or blue to indicate if Trump or Biden won in that county respectively. Each vertical line separates the points into quartiles, with the second line marking the median of the data. In the plot, we can see that Trump generally won more in counties with both smaller amounts of votes and a smaller population while Biden won more in counties with

both higher amounts of votes and a larger population – indicating that both population and votes casted could be an effective predictive variable for the winner of a county.

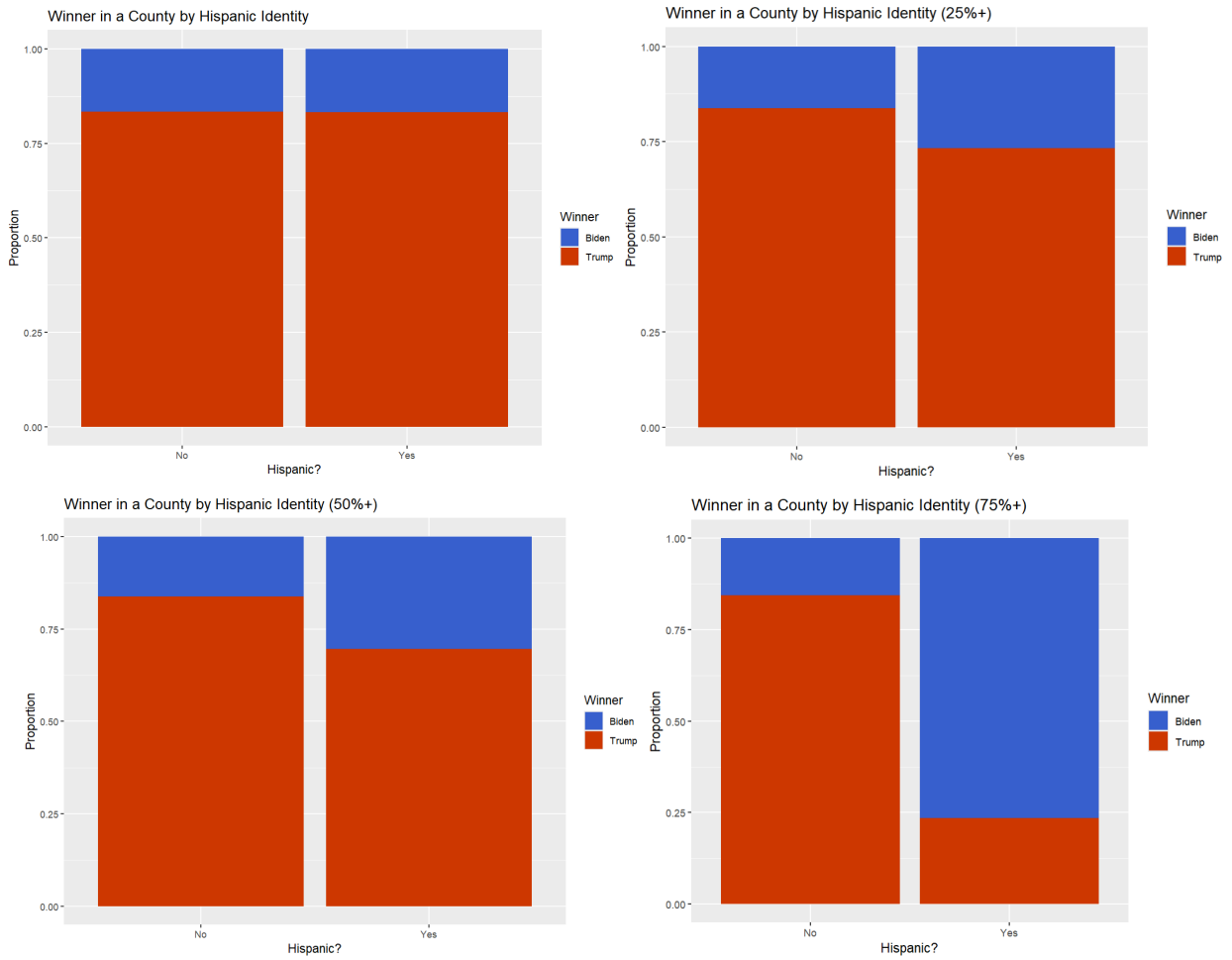


Visual 11: This bar graph coheres with the previous visual as it shows the proportion of counties within each population quartile that had Biden or Trump as its winner. The results here support our initial findings from visual 10 as we see again that as the population in counties increases, Biden tends to win more, strengthening the viability of total population as a predictive variable.





Visual 12: These four bar graphs depict the winner of a county by race when that race makes up four different proportions of the county: 0%+, 10%+, 25%+, and 50%+. The races included in the graphs match those included in the "One race" variable in the dataset, with them being American Indian and Alaska Native (shortened as AIAN in the graphs), Asian, Black, Native Hawaiin and Pacific Islander (shortened as NHPI), White, and Other. NHPI is not in the 25%+ and 50%+ graphs due to there being no counties with that minimum proportion of NHPI people in the county, as is the same for Asian in the 50%+ graph. In the initial graph, the proportion of counties that Biden and Trump won for each race is approximately the same, with Biden winning slightly more only in counties with NHPI people – creating a primary suggestion that race is not a predictive factor for the winner of a county. However, when we look at the 10%+ graph, these results dramatically shift; Biden wins considerably more in counties where AIAN, Asian, Black, NHPI, and Other people make up at least 10% of the population (with him even winning every NHPI 10%+ county), with only White having about the same proportions as the first graph. When we look at the 25%+ graph, these changes become more dramatic. Counties with Asian people making up 25%+ of the population are only won by Biden while counties with AIAN, Black, and Other people making up 25% of the population were won by Biden approximately 33% of the time. In looking at our final graph (50%+), the results do, once again, change considerably. In almost every county in which Black people are the majority of the population, Biden wins. Additionally, Biden wins about 66% of the counties in which AIAN people are the majority of the population (which is double that of the 25%+ graph). We also see a break in the pattern from the previous two graphs as the counties with Other people making up the majority of the population were all won by Trump. Throughout all four graphs, Trump and Biden win about the same proportion of counties regardless of the proportion of White people in those counties. All in all, these four graphs suggest that race is an important predictive demographic factor for the winner of a county, with Biden tending to win more in counties with higher proportions of non-White people (with the caveat of Other results being somewhat inconclusive and the proportion of White people having seemingly no impact).



Visual 13: These four graphs depict the winner of a county by Hispanic identity when people of that identity (yes = Hispanic, no = not Hispanic) make up a certain proportion of that county, with the proportions being 0%+, 25%+, 50%+, and 75%+. Much like with visual 12, we see that there is not an apparent visual difference between how often Biden and Trump won respectively within counties where Hispanic and non-Hispanic people make up 0%+ of the county. Once again, however, when we look at the second graph (25%+), we see a pattern potentially start to emerge; in counties where Hispanic people make up at least a quarter of the population, Biden tends to win about 10% more than in counties where non-Hispanic people make up at least a quarter of the population (the latter bar of which has approximately the same proportions as in the first graph). In the 50%+ bar graph, this pattern continues: counties in which non-Hispanic people make up the majority of a county have the same proportions as in the first and second graphs, and in counties where a majority of the population is Hispanic, Biden tends to win more (this time in more than a quarter of the counties). Our final graph (75%+) depicts a dramatic strengthening of this pattern with counties in which Hispanic people make up 75%+ of the population having Biden as their winner 75% of the time (almost triple the proportion seen in the second and third graph). Counties where non-Hispanic people make up 75%+ of the population have approximately the same proportions of Biden and Trump winning as the three previous

graphs. Overall, we can use these graphs to infer that the amount of Hispanic people in a county relative to the total population of a county could be a valuable predictive variable for determining the winner of that county.

Preprocessing

Handling Missing Values: For all models, we applied different imputation methods tailored to the characteristics of the data. Specifically, for Models #1 (K-Nearest Neighbors - KNN) and #2 (Random Forest), we employed step_impute_mean() to replace missing values with the mean of their respective columns. This approach is effective in maintaining the dataset's overall distribution, especially for models sensitive to the scale of data, such as KNN. It ensures that all available data can be used for model training, while the mean imputation helps mitigate potential biases from missing values. For Models #3 (Boosted Tree), #4 (Naive Bayes), and #5 (Bagged Tree), we utilized step_impute_median() for imputing missing values with the median of their respective columns, as these models benefit from robust measures of central tendency, particularly in the presence of skewed distributions. The median is less sensitive to outliers, making it a more stable choice for these models, especially when dealing with variables like income and GDP, which can have extreme values.

Log Transformation: For models #3, #4, and #5, we applied a log transformation (step_log()) to almost all numeric predictors, except for the x0018e variable (median age). This transformation addresses the issue of right-skewed distributions by compressing the range of the data, making it more normally distributed. This step is particularly beneficial for models like Boosted Trees and Naive Bayes, where the assumptions of normality or even distribution can improve model performance.

Test Data Preprocessing: For consistency, the same preprocessing steps (imputation and transformations) were applied to the test data. The median imputation for numeric variables and mode imputation for categorical variables ensured that the test data remained aligned with the training data's preprocessing. This uniformity prevents data leakage and ensures that the model's evaluation reflects real-world performance.

Exclusion of Non-Predictive Features: We excluded the ‘id’ and ‘name’ columns from the feature set. These columns do not contain predictive information relevant to the target variable (winner). Including them could lead to overfitting, as they may inadvertently capture noise specific to the training data, which doesn't generalize to new data.

Candidate Models, Model Evaluation, & Tuning

ID	*Type of Model*	*Model Engine*	*Recipe*	*Hyperparameters*
1	K-Nearest Neighbors	kknn	knn_recipe <- recipe(winner ~ ., data = train) %>% step_impute_mean(all_numeric_predictors()) %>% step_corr(threshold = 0.9)	neighbors = 45 weight_func = "triweight" dist_power = 1
2	Random Forest	partykit	prf_recipe <- recipe(winner ~ ., data = train) %>% step_impute_mean(all_numeric_predictors()) %>% step_interact(terms = ~ total_votes * x0001e)	mtry = 50 trees = 600 min_n = 15
3	Boosted Tree	XGboost	bt_recipe <- recipe(winner ~ ., data = train) %>% step_impute_median(contains("income"), contains("gdp")) %>% step_log(all_numeric_predictors(), -x0018e, offset = 1)	mtry = 97 trees = 298 tree_depth = 15 learn_rate = 0.0827 sample_size = 0.9615
4	Naive Bayes	klaR	nb_recipe <- recipe(winner ~ ., data = train) %>% step_impute_median(contains("income"), contains("gdp")) %>% step_log(all_numeric_predictors(), -x0018e, offset = 1) %>% step_corr(all_numeric_predictors(), threshold = tune())	smoothness = 0.5165 Laplace = 1.9339 threshold = 0.6349
5	Bagged Tree	rpart	bag_recipe <- recipe(winner ~ ., data = train) %>% step_impute_median(contains("income"), contains("gdp")) %>% step_log(all_numeric_predictors(), -x0018e, offset = 1)	cost_complexity = 2.1472e-10 tree_depth = 15 min_n = 6 class_cost = 1.2418

Candidate model #1: A k-nearest neighbors classification model with kknn as its engine; it performed well overall but is weak in comparison to the rest of our candidate models with a public accuracy score of 0.87866. This model included the step_impute_mean() mentioned in the Pre-Processing section in its recipe. By including this, we correct the issue of having missing values in the data while still utilizing the other information included in the row for training. step_corr() was included with a threshold of 0.9 in an attempt to account for multicollinearity between variables in the data. Within the setup of the model, we set neighbors to 45 so that the model would use the closest 2% of the data in the training set when trying to classify test points. The "triweight" function was selected as our weight function since, when we were testing the model and its minimal misclassification model, "triweight" consistently produced better results. Similar reasoning was applied in setting dist_power to 1.

Candidate model #2: A random forest classification model with partykit as its engine; this model also performed generally well but is somewhat weaker than most of our candidate models with a public accuracy score of 0.88354. This model also included the step_impute_mean() step in its recipe to correct the issue of having missing values in the data while still utilizing the other information included in the row for training. We also included step_interact() with x0001e (the total population of the county) and total_votes to utilize the potential interaction effect of the amount of voters and the total population of the county in classifying test points. In the setup of the random forest model, mtry was set to 50 to randomly sample 40% of the independent variables when trying to sort the points down a tree. We chose to use 600 trees to try to increase the total number of paths that the points can be sorted down due to the high amount of variables in the data and included in the model's recipe. min_n was set to 15 to include a light amount of data points in a node before splitting to increase the complexity and flexibility of the model.

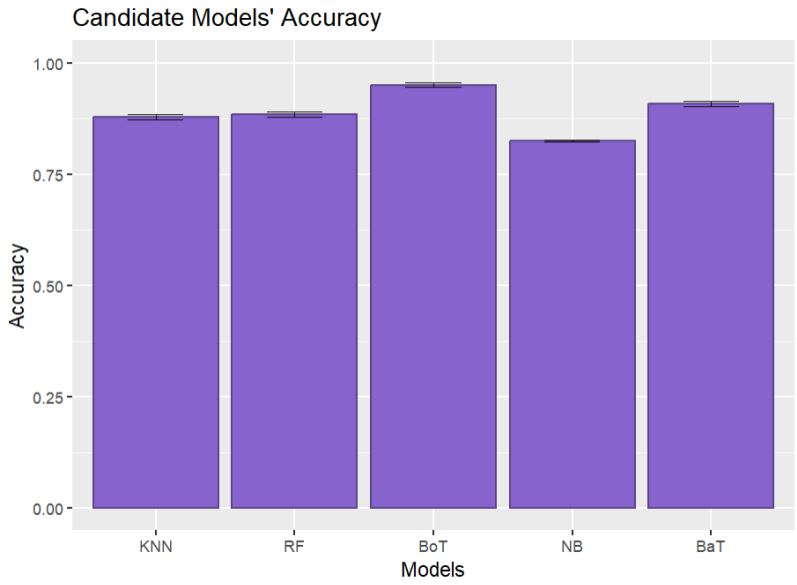
Candidate model #3: A boosted decision tree model using XGboost as its computational engine. It was our best-performing model with a public accuracy of 0.94979. The recipe replaced missing values in the "income_per_capita" and "gdp" columns with the median of those variables (as opposed to the mean, since the variables are right-skewed), then log10-transformed any right-skewed numeric variables. Five hyperparameters were tuned using tune_grid.

Candidate model #4: A naive bayes classifier model using klaR as its computational engine. Since this model type assumes the predictors are independent (hence its "naivety"), multicollinearity directly reduces its accuracy. Thus, the recipe was the same as model #3's, but with an added step_corr with a to-be-tuned threshold parameter, which seemed to impact this type of model more than it did for others. The model did not perform as well as the others in terms of accuracy, due to unfamiliarity with the model and lack of understanding of the specific modeling process for it.

Candidate model #5: A bagged decision tree model using rpart as its computational engine. The recipe used was identical to that of model #3. Its accuracy was middle-of-the-road, at 0.90794. Four hyperparameters were tuned using tune_grid.

Model ID	***Type of Model***	***Accuracy (Public)***	***Standard Error***
1	K-Nearest Neighbors	0.87866	0.00602602
2	Random Forest	0.88354	0.005980927
3	Boosted Tree	0.94979	0.004905849
4	Naive Bayes	0.82426	0.002114560

5	Bagged Tree	0.90794	0.005522992



In this bar graph, we can see that all five of our candidate models have a strong accuracy, with the lowest accuracy being about 0.8243, the highest being about 0.9498, and our average accuracy being about 0.8890. Our models' accuracies are also fairly consistent and reliable, having a standard error of at most 0.006.

The naive bayes model performed the worst in terms of accuracy, due to unfamiliarity with the model. It should be noted that R provided a warning while predicting with the test data and while tuning hyperparameters which stated that the predicted probability for all [two] classes was 0. For certain observations, the classifier simply defaulted to picking Biden. Given more time to research and understand this type of classifier, we could fix this issue, better customize the recipe used, and improve the accuracy of this model.

The KNN, random forest, and bagged tree models all performed similarly, with accuracies within 0.04 of one another. With an overall accuracy of 0.9498, the boosted decision tree model performed the best of the five candidates. We believe one reason for this is that the boosted tree is able to handle the presence of imbalanced classes in the data set. This imbalance was quite great, with over 83% of observations having Trump as its winner. Models such as the random forest may not have been quite as adept at dealing with this imbalance as the boosted decision tree was.

Final Model

Our final model is candidate #3, the boosted decision tree, with a public accuracy of 0.94979 and private accuracy of 0.90203. The model performed a fair amount better than any of the other four candidate models. The low standard error indicates the performance metrics and model to be trustworthy.

Since it is a boosted decision tree, it handles the class imbalance well in the training data set. Moreover, the model iteratively corrects the mistakes of past decision trees, allowing it to capture complex relationships between the predictor variables and the response. A hyperparameter value for mtry of 97 sufficiently prevents overfitting to the training set, allowing the model to generalize fairly well to new data (as evidenced by the relatively private accuracy).

That said, the model could be improved in several ways. No interaction terms were used in the model's recipe, meaning there could be relationships between the predictor variables that could have been leveraged more. Also, there was an oversight in failing to convert the last variable in the data set into a categorical variable, which indicated with a number from one to six how rural or urban a county is. Since this variable could only take on six values, it should have been treated as a categorical, rather than numerical predictor.

Additional data such as surveyed partisanship from each county could aid in model accuracy greatly, since people would likely vote for the presidential candidate who is a part of the party that the voter aligns themself with. Other data, such as more economic factors and geographical data, could also introduce new relationships to capture with our models.

Appendix: Final Annotated Script

```
library(tidyverse)
library(tidymodels)
library(xgboost)

train <- read_csv("train_class.csv")
train <- train %>% select(-id, -name)
test <- read_csv("test_class.csv")
set.seed(3) # for reproducibility
train_folds <- vfold_cv(train, v = 5, strata = winner)

# model spec with xgboost for classification and hyperparameter
tuning with tune()
bt_model <- boost_tree(
  trees = tune(),
  mtry = tune(),
  min_n = 2,
  learn_rate = tune(),
  tree_depth = tune(),
  sample_size = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

bt_recipe <- recipe(winner ~ ., data = train) %>%
  step_impute_median(contains("income"), contains("gdp")) %>% #
replacing missing values with medians of their respective
variables
  step_log(all_numeric_predictors(), -x0018e, offset = 1) #
log10 transformation for all but column "x0018e", which is the
only one not skewed

bt_wflow <- workflow() %>%
  add_model(bt_model) %>%
  add_recipe(bt_recipe)

bt_params <- extract_parameter_set_dials(bt_wflow) %>%
  update(mtry = finalize(mtry(), x = train %>% select(-winner)))
%>%
  update(trees = deg_free(c(5, 300))) %>%
  # update(min_n = deg_free(c(2, 15))) %>%
```

```
  update(learn_rate = learn_rate(range = c(-1.30103, -0.60206)))
%>%
  update(tree_depth = deg_free(c(7, 15))) # updated
hyperparameter ranges based on previous results

# ***this takes like 30 mins to run
bt_tune_fit <-
  bt_wflow %>%
  tune_grid(
     resamples = train_folds,
     grid = grid_random(bt_params, size = 50)
  )

bt_tune_bayes_fit <- # uses results of tune_grid as a baseline
to optimally tune hyperparameters
  bt_wflow %>%
  tune_bayes(
     resamples = train_folds,
     param_info = bt_params,
     iter = 50,
     initial = bt_tune_fit,
     metrics = metric_set(roc_auc, accuracy),
     objective = exp_improve(trade_off = 0.03),
     control = control_bayes(uncertain = 5)
  )

bt_best_model <- bt_tune_fit %>% select_best(metric = "roc_auc")
# select best model
bt_final_wflow <- bt_wflow %>% finalize_workflow(bt_best_model)
# finalize workflow with best model
bt_final_fit <- bt_final_wflow %>% fit(data = train) # fit and
predict, then make resulting data frame
bt_final_pred <- bt_final_fit %>% predict(new_data = test)
bt_final_res <- bind_cols(id = test$id, winner =
bt_final_pred$.pred_class)

write_csv(bt_final_res, "submission.csv")
```

Appendix: Team Member Contributions

- Max
    - Created Kaggle teams and outlined project reports
    - EDA visual 1
    - Script creation
    - Models #3, #4, #5, descriptions, and some comparisons
    - Wrote Final Model section
- Bella
    - Submitted ten sets of predictions to Kaggle
    - Created EDA Visuals 9-13 and wrote their descriptions
    - Created Candidate Models 1 and 2 and wrote their descriptions
    - Created the Candidate Models' Accuracy graph and wrote a brief analysis of it
- Tran
    - Wrote the introduction and the preprocessing
    - Submitted predictions on Kaggle
    - Adjusted the font and layout of the report
- Vanda
    - Created EDA Visuals 2 - 8 and wrote their descriptions
    - Wrote the description for EDA Visual 1
    - Refined and extended the description of Preprocessing, ensuring that it accurately reflects the preprocessing steps for each model type and justifies their choices
    - Edited the introduction to enhance readability and precision
    - Standardized the formatting and layout of the report