Classification Project Report - Presidential Election Winner for Each County Prediction

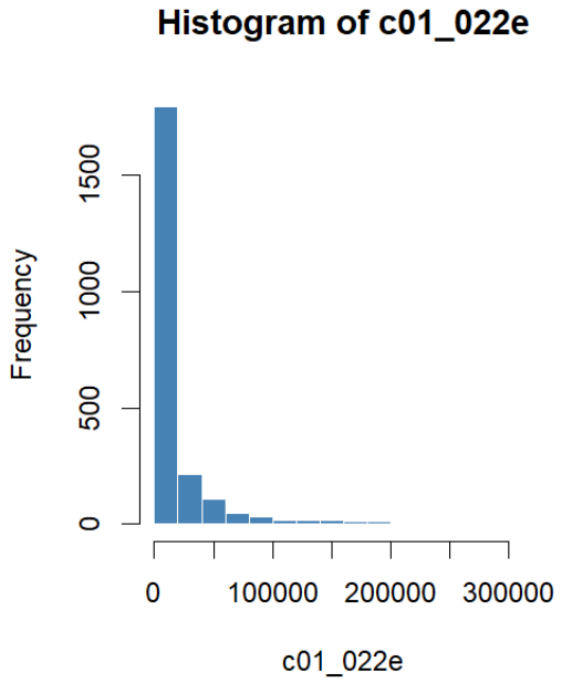
Summer 2025 Stats 101C, Team 10

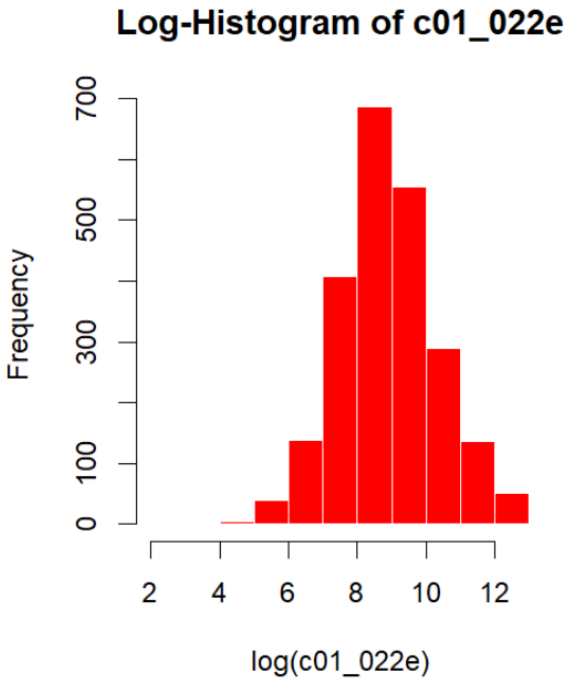
By Alfred Mastan, Quinn Koch, Joseph Choi, Johnathan Pham, Kwanhee Yoon

Introduction

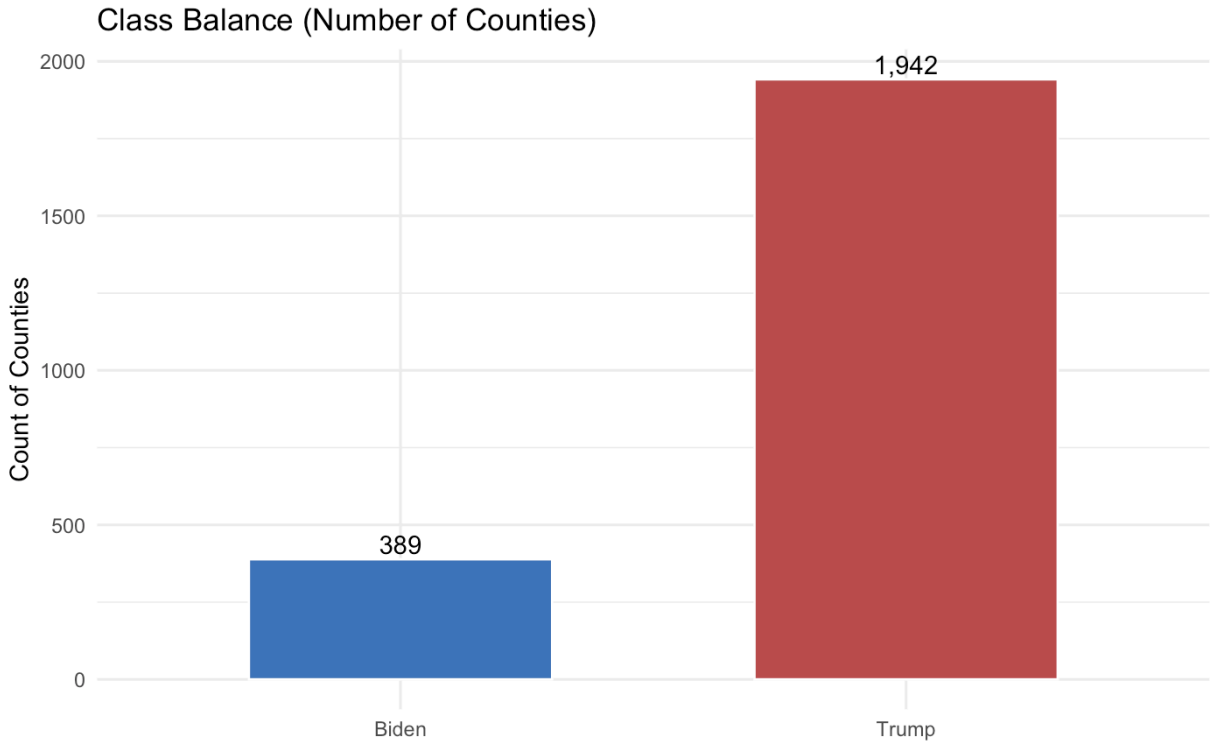
The goal of this research project is to predict the county-level winner in the 2020 U.S. Presidential Election, either Biden or Trump, based on the demographic and education estimates from the U.S. Census Bureau. To make the counties comparable by size, we converted the raw Census data into rates. More specifically, we took the subcategory columns (age-by-sex, race, and two-or-more race subgroups, Hispanic/Non-Hispanic, etc.) and divided them by their respective within-group totals. We then normalized the remaining groups by total population and removed any identifiers and raw totals. By doing so, we create interpretable percentages instead of raw counts, allowing more stable model fitting and preventing any scale effects that may impact our predictions.

Exploratory Data Analysis

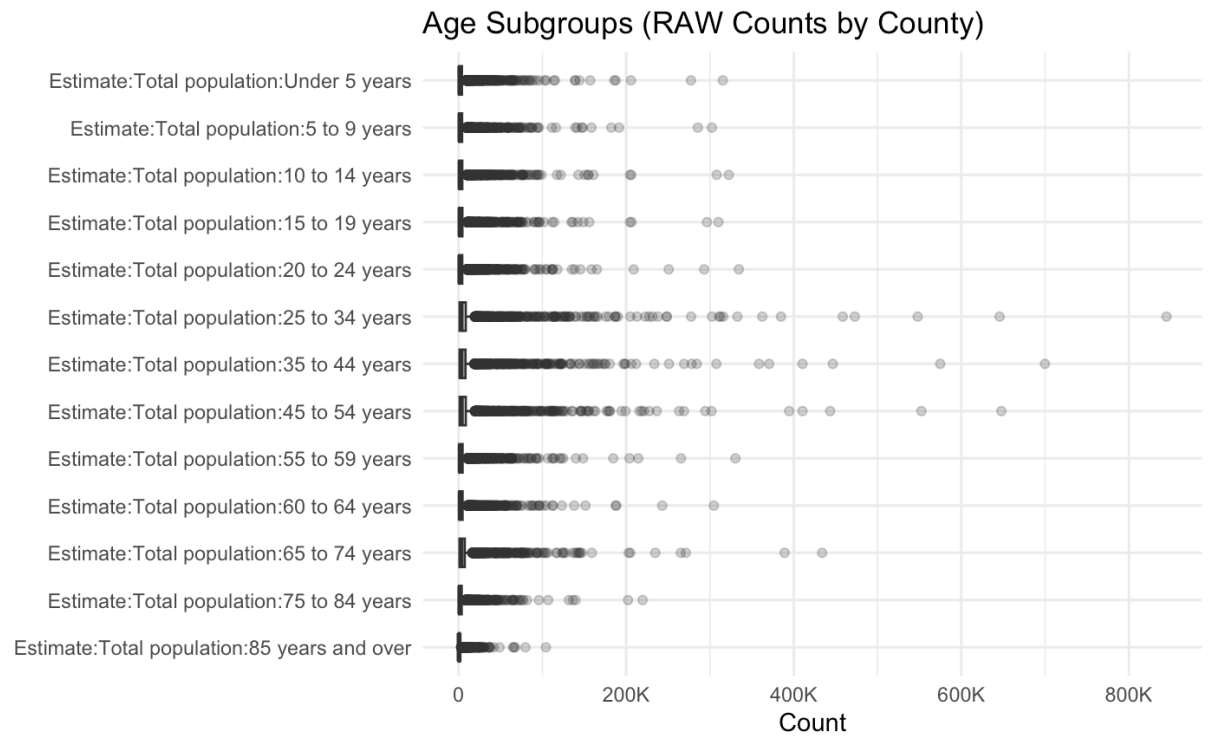


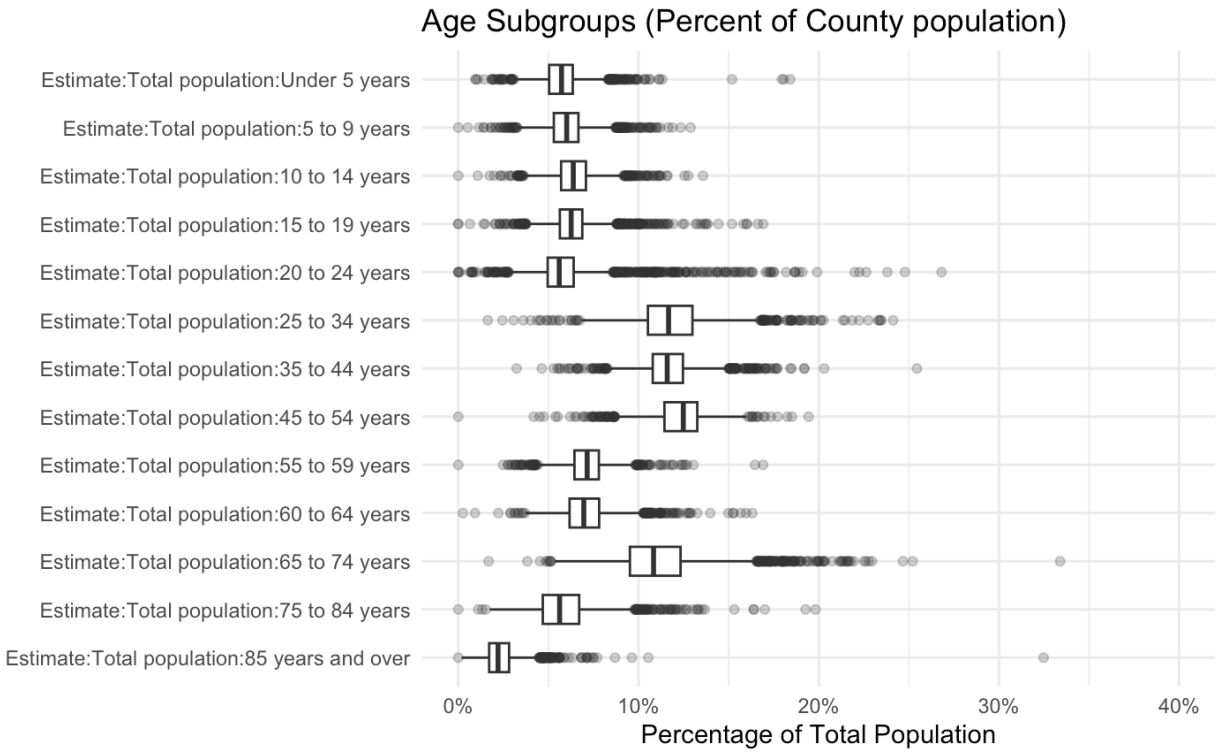


We created histograms for all the columns and see that the left graphs (in blue) are based on the original data, which have the majority right-skewed. We then applied a log transformation to every graph, which resulted in a more symmetric distribution (in red). This works well for us because we now know to apply a log transformation in our models to reduce leverage and stabilize variance in models sensitive to scale, improving the fit.



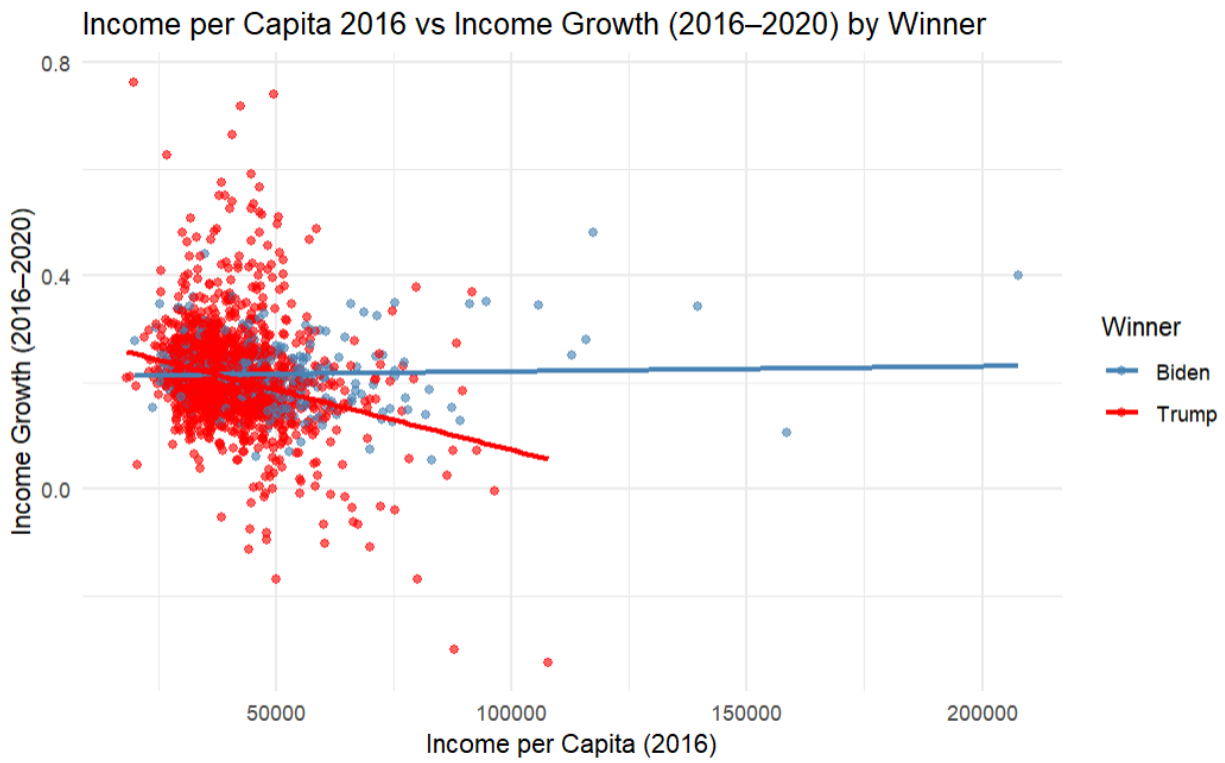
From this bar graph, we can see that there is an imbalance with far more Trump-winning counties than Biden. In this specific scenario with votes, we want to show that we are not going to alter anything (no over or undersampling) because it may cause some bias in our predictions. Instead, we will keep the original distribution and will tune our models to accurately predict the winner.



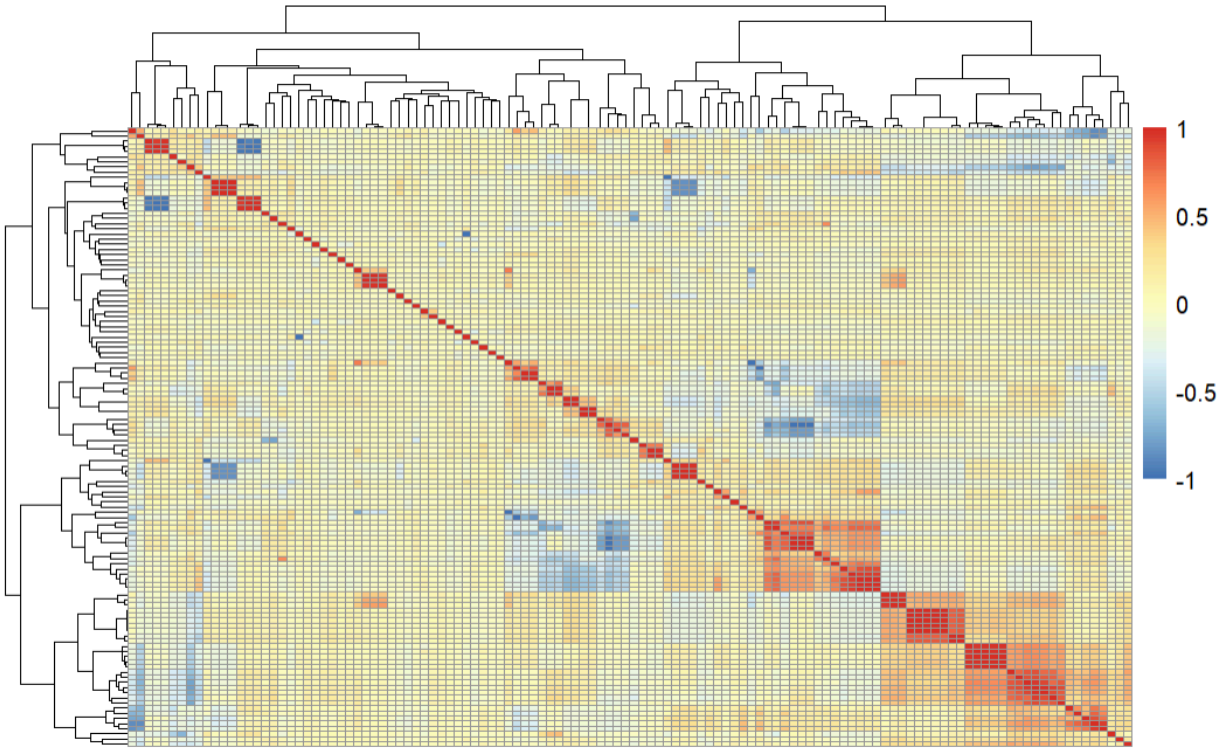


Here we have two boxplots: age subgroups raw counts and the same subgroups expressed in percent of each county's population. Looking at the raw data boxplot, there is an extreme scale difference due to the county size so the raw counts are incomparable across counties. This

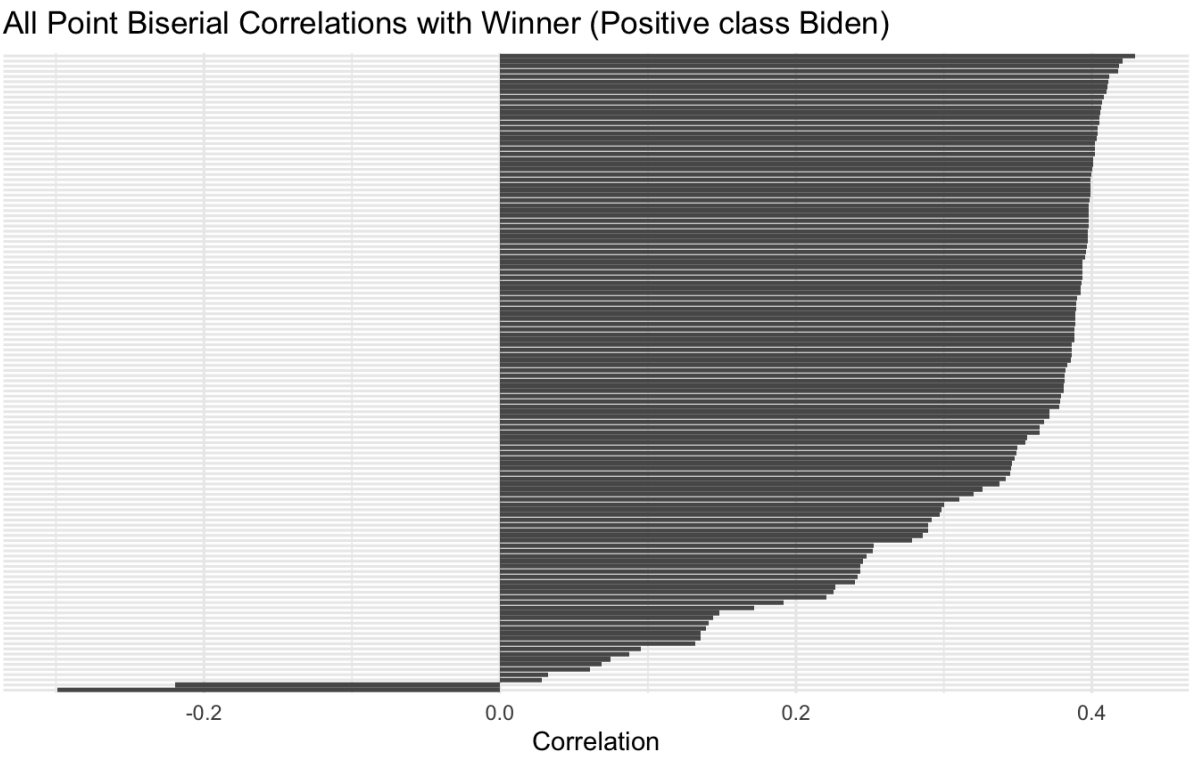
justifies our method of expressing the same variables in a percentage of the population, removing size effects from our models. The subgroups become more comparable to use now, but we still need to use some transformation to treat extreme values and normalize the predictors.



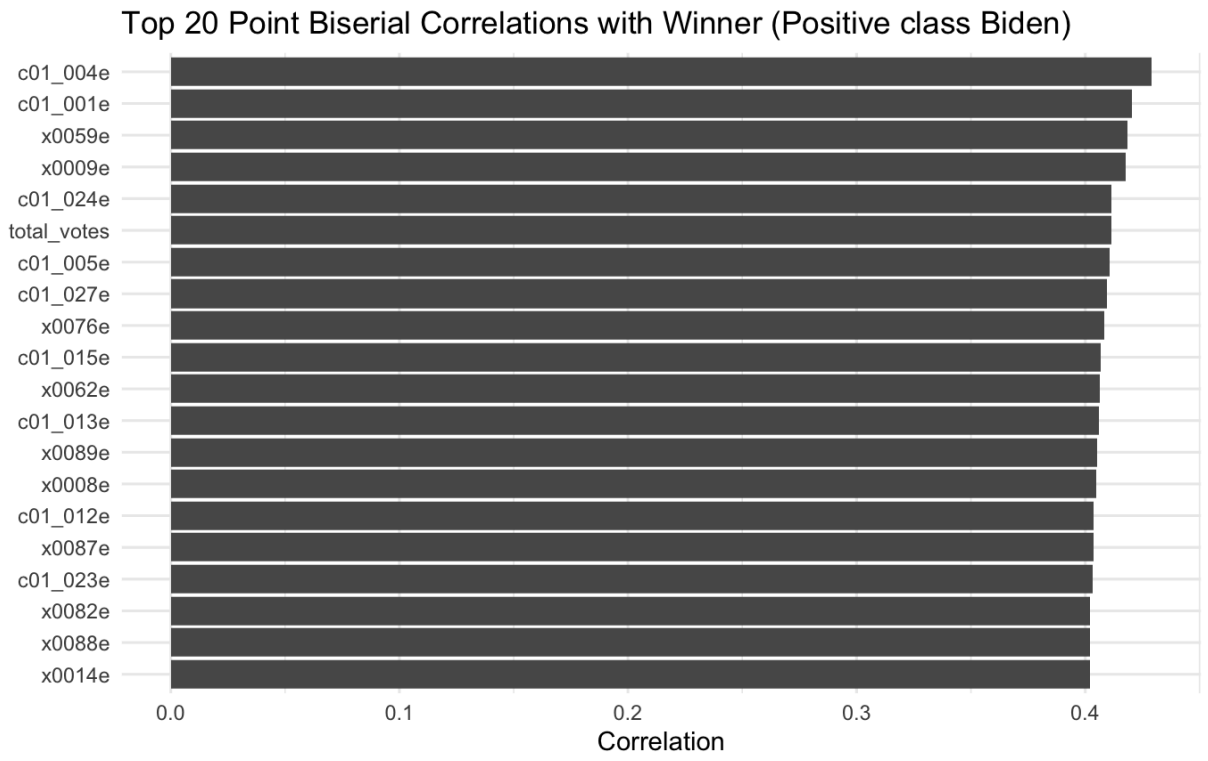
Income per capita growth from 2016 to 2020 emerges as a meaningful feature in distinguishing counties. While overall medians are close for income per capita growth between Biden and Trump-leaning areas, Trump counties display greater variability, with both rapid gains and negative growth. This spread indicates that income growth captures economic divergence across regions, making it a potentially valuable predictor of political outcomes beyond static income levels.



This is a clustered correlation heat map of the training data. The spots that are more red indicate a stronger positive correlation, the spots that are yellow indicate weak or no correlation, and the spots that are more blue indicate a stronger negative correlation. Although most of the map is yellow, we can see some areas where it is red and blue, which may suggest multicollinearity in certain subsets of predictors. With this information, we may try using step_corr() in our recipe to remove highly correlated predictors and see if it will improve performance.



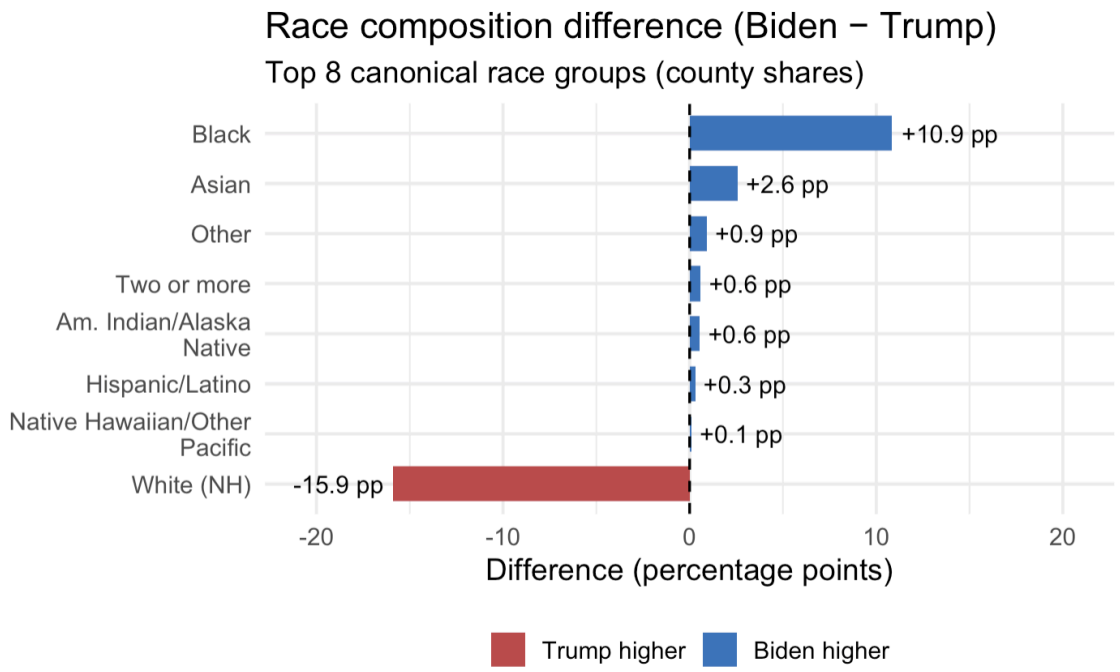
Here, we plotted the point biserial correlations (which allows us to create a correlation coefficient with a binary class) of all predictors with the target variable. As we can see, there are no standout variables, with nearly half of the predictors having correlations with the target close to 0.4, but none exceeding 0.5. Due to the small number of predictors having low correlation values with the target, we chose not to select features based on correlation, as removing low correlation values would yield small efficiency gains compared to including all predictors.



	variable	winner_num	DESCRIPTION
1	c01_004e	0.42918169	Estimate:Total:Population 18 to 24 years:Some college or associate's degree
2	c01_001e	0.42056653	Estimate:Total:Population 18 to 24 years
3	x0059e	0.41829155	Estimate:Total population:Two or more races:White and Black or African American
4	x0009e	0.41781016	Estimate:Total population:20 to 24 years
5	c01_024e	0.41153916	Estimate:Total:Population 45 to 64 years:Bachelor's degree or higher
6	total_votes	0.41129087	Total Votes Cast
7	c01_005e	0.41074015	Estimate:Total:Population 18 to 24 years:Bachelor's degree or higher
8	c01_027e	0.40968265	Estimate:Total:Population 65 years and over:Bachelor's degree or higher
9	x0076e	0.40828826	Estimate:Total population:Not Hispanic or Latino
10	c01_015e	0.40692059	Estimate:Total:Population 25 years and over:Bachelor's degree or higher
11	x0062e	0.40654366	Estimate:Total population:Two or more races:Black or African American and American Indian and Alaska Native
12	c01_013e	0.40582177	Estimate:Total:Population 25 years and over:Graduate or professional degree
13	x0089e	0.40506619	Estimate:Citizen, 18 and over population:Female
14	x0008e	0.40499072	Estimate:Total population:15 to 19 years
15	c01_012e	0.40381112	Estimate:Total:Population 25 years and over:Bachelor's degree
16	x0087e	0.40380220	Estimate:Citizen, 18 and over population
17	c01_023e	0.40311361	Estimate:Total:Population 45 to 64 years:High school graduate or higher
18	x0082e	0.40219142	Estimate:Total population:Not Hispanic or Latino:Some other race alone
19	x0088e	0.40212251	Estimate:Citizen, 18 and over population:Male
20	x0014e	0.40210376	Estimate:Total population:60 to 64 years

In the above plot and its accompanying table (containing variable descriptions), we isolate the top 20 variables most correlated with winner = ‘Biden’. Broadly speaking, we see that categories pertaining to young and college educated people dominate the top performing variables. A notable exception would be the total_votes variable performing highly. These findings agree with

the conventional wisdom of American politics that Democrats perform well in major city centers and places with young, highly educated people.



Based on this bar graph, we can see race composition differences (Biden - Trump) in average county race shares for the top 8 race/ethnicity groups. Across 2,331 counties, Biden-won counties had a +10.9 percentage points higher Black share vs Trump-won counties with a +15.9 points higher non-Hispanic White share. We also see small differences in Biden-won counties for Asian, Hispanic/Latino, and some other groups. The size and consistency of these gaps can indicate that county racial composition has a strong association with county outcomes. Although we can treat race groups as solid predictors because of compositional shares, we still need to control other predictors like urban-rural or income to reduce confounding.

Preprocessing

Recipes

Recipe 1: generic_recipe

```
generic_recipe <- recipe(winner ~ . , data = train) %>% #create baseline recipe with all predictors in dataset
  # Normalize counts related to total population into percentages (so everything has the same "weight")
  step_mutate(across(c(x0026e, x0027e), ~ . / x0025e), # 18years and over (Male/Female)
         across(c(x0030e, x0031e), ~ . / x0029e), # 65years and over (Male/Female)
         across(c(x0040e, x0041e, x0042e, x0043e), ~ . / x0039e), # American Indian sub-races
         across(44:50, ~ . / x0044e), # Asian sub-races
         across(52:55, ~ . / x0052e), # Hawaiian sub-races
         across(c(x0037e, x0038e, x0039e, x0044e, x0052e, x0057e), ~ . / x0036e), # One race (White/Black/American
Indian/Asian/Hawaiian)
         across(c(x0059e, x0060e, x0061e, x0062e), ~ . / x0058e), # Two or more races
         across(c(62:67), ~ . / x0001e), # Combinations
```

```
        across(c(x0072e, x0073e, x0074e, x0075e), ~ . / x0071e), # Hispanic or Latino
        across(c(x0077e, x0078e, x0079e, x0080e, x0081e, x0082e), ~ . / x0076e), # Not Hispanic or Latino
        across(c(x0084e, x0085e), ~ . / x0083e), # Not Hispanic or Latino (Two or more races)
        across(c(x0088e, x0089e), ~ . / x0087e), # Citizen over 18 (Male/Female)
        across(c(c01_002e, c01_003e, c01_004e, c01_005e), ~ . / c01_001e), # Educations 18-24 yrs
        across(c(93:101), ~ . / c01_006e), # Educations 25+ yrs
        across(c(c01_017e, c01_018e), ~ . / c01_016e), # Educations 25-34 yrs
        across(c(c01_020e, c01_021e), ~ . / c01_019e), # Educations 35-44 yrs
        across(c(c01_023e, c01_024e), ~ . / c01_022e), # Educations 45-64 yrs
        across(c(c01_026e, c01_027e), ~ . / c01_025e) # Educations 65+ yrs
        ) |>
  # Normalize rest by total population
  step_mutate(across(c(x0025e, x0029e, x0036e, x0039e, x0058e, x0071e, x0076e, x0083e, x0087e,
                c01_001e, c01_006e, c01_016e, c01_019e, c01_022e, c01_025e), ~ . / x0001e)) |>
  step_mutate(across(c(4:25), ~ . / x0001e)) |> #X0002E-X0024E
  step_rm(id, x0001e, x0033e, x0034e, x0035e, x0024e) |>
  step_mutate(income_per_cap_16_20 = (income_per_cap_2020 - income_per_cap_2016)/income_per_cap_2016,
              gdp_grow_16_20 = (gdp_2020 - gdp_2016)/gdp_2016) |>
  step_impute_median(all_numeric_predictors())
```

For the *generic_recipe*, we applied three main preprocessing operations - step_mutate(), step_rm(), and step_impute_median() to prepare the data for model training.

- **step_mutate():** The EDA shows that most of the variables were heavily skewed and vary in ranges. To avoid biased importance between variables, we normalize them using the respective total population for each subpopulation. As an example, *total_population-male* and *total_population-female* are divided by *total_population*, and *population_25-34-highschool_graduate* and *population_25-34-bachelor* are divided by *population_25-34*. We also calculated the growth rate for the *income_per_cap* and *gdp_grow* between the years 2020 and 2016.
- **step_rm():** To remove duplicated columns and prevent high correlation, as it's being used as a denominator for many other variables.
- **step_impute_median():** Since the *income_per_cap* and *gdp_grow* from 2016 to 2020 have NA values, we impute them with the median to make it more robust to possible outliers.

Recipe 2: log_recipe

```
log_recipe <- generic_recipe %>%
  step_log(all_numeric_predictors(), offset = 1) %>%
  step_normalize(all_numeric_predictors())
```

- **step_log():** To further combat the skewness in the data after the percentage normalization, we use the logarithmic transformation to help models like KNN and Logistic Regression capture the pattern in the data. While a tree-based model like XGBoost doesn't require further transformation, we also ran it through to see whether it could give any assistance and improve the score.
- **step_normalize():** After the log transformation, we also ensure that the data is centered and normalized into one standard deviation to allow KNN and Logistic Regression to capture the patterns correctly.

Candidate Models, Model Evaluation, & Tuning

Candidate Models Tested:

Model	Type of Model	Engine	Recipe	Hyperparameters
Generic XGBoost	XGBoost	xgboost	1	trees: 889 min_n: 2 tree_depth: 4
Log XGBoost	XGBoost	xgboost	2	trees: 889 min_n: 2 tree_depth: 4
Generic KNN	KNN	kknn	1	neighbors (k) = 15
Log KNN	KNN	kknn	2	neighbors (k) = 15
Generic Logistic Regression	Logistic Regression	glmnet	1	penalty: 2.920543e-03
Log Logistic Regression	Logistic Regression	glmnet	2	penalty: 2.920543e-03

Generic XGBoost Model

trees <int>	min_n <int>	tree_depth <int>	.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
889	2	4	roc_auc	binary	0.9728114	5	0.004164384	Preprocessor1_Model01
445	10	13	roc_auc	binary	0.9711922	5	0.004577317	Preprocessor1_Model03
1555	6	11	roc_auc	binary	0.9703528	5	0.004307555	Preprocessor1_Model02
667	23	1	roc_auc	binary	0.9636525	5	0.005965302	Preprocessor1_Model06
1777	18	2	roc_auc	binary	0.9636222	5	0.006052268	Preprocessor1_Model05
2000	27	10	roc_auc	binary	0.9574698	5	0.006659990	Preprocessor1_Model07
223	35	8	roc_auc	binary	0.9567295	5	0.006956562	Preprocessor1_Model09
1111	31	15	roc_auc	binary	0.9554993	5	0.006391398	Preprocessor1_Model08
1333	40	5	roc_auc	binary	0.9467206	5	0.006852935	Preprocessor1_Model10
1	14	7	roc_auc	binary	0.9135554	5	0.009199450	Preprocessor1_Model04

XGBoost uses a boosting algorithm where it builds trees sequentially, where each tree corrects errors from the previous one for the model to capture complex nonlinear patterns. With the generic preprocessing recipe and 5-fold cross-validation, our best configuration achieved a mean ROC-AUC ≈ 0.973 and a small standard error ≈ 0.0042, which indicates stable performance across folds. This result establishes XGBoost as a strong tree-based baseline.

Log XGBoost Model

trees <int>	min_n <int>	tree_depth <int>	.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
889	2	4	roc_auc	binary	0.9728114	5	0.004164384	Preprocessor1_Model01
445	10	13	roc_auc	binary	0.9711922	5	0.004577317	Preprocessor1_Model03
1555	6	11	roc_auc	binary	0.9703328	5	0.004317276	Preprocessor1_Model02
1777	18	2	roc_auc	binary	0.9636556	5	0.006039937	Preprocessor1_Model05
667	23	1	roc_auc	binary	0.9635666	5	0.005893150	Preprocessor1_Model06
2000	27	10	roc_auc	binary	0.9574698	5	0.006659990	Preprocessor1_Model07
223	35	8	roc_auc	binary	0.9567295	5	0.006956562	Preprocessor1_Model09
1111	31	15	roc_auc	binary	0.9554993	5	0.006391398	Preprocessor1_Model08
1333	40	5	roc_auc	binary	0.9467206	5	0.006852935	Preprocessor1_Model10
1	14	7	roc_auc	binary	0.9135554	5	0.009199450	Preprocessor1_Model04

With the log-transformed recipe, our best configuration again achieved ROC-AUC = 0.973, statistically indistinguishable from the generic version. As we expected, the log preprocessing did not provide measurable improvements, as XGBoost can already handle skewed and nonlinear feature distributions effectively.

Generic KNN Model

neighbors <int>	.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
15	roc_auc	binary	0.9603166	5	0.002670456	Preprocessor1_Model10
13	roc_auc	binary	0.9586759	5	0.002587007	Preprocessor1_Model09
11	roc_auc	binary	0.9556648	5	0.003113294	Preprocessor1_Model08
10	roc_auc	binary	0.9540858	5	0.003576516	Preprocessor1_Model07
8	roc_auc	binary	0.9506158	5	0.003745317	Preprocessor1_Model06
7	roc_auc	binary	0.9487292	5	0.003144664	Preprocessor1_Model05
5	roc_auc	binary	0.9383960	5	0.002911962	Preprocessor1_Model04
4	roc_auc	binary	0.9317162	5	0.003783834	Preprocessor1_Model03
3	roc_auc	binary	0.9173289	5	0.005149965	Preprocessor1_Model02
1	roc_auc	binary	0.8181708	5	0.010538981	Preprocessor1_Model01

We also trained a K-Nearest Neighbors (KNN) classifier using the generic recipe. KNN predicts based on the majority vote among nearby points, which allows it to capture local patterns but makes it sensitive to noise. Performance peaked at ROC-AUC ≈ 0.960 with k = 15. Smaller neighbor counts tended to overfit, while larger values smoothed predictions at the expense of flexibility.

Log KNN Model

neighbors <int>	.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
15	roc_auc	binary	0.9594661	5	0.002688396	Preprocessor1_Model10
13	roc_auc	binary	0.9587333	5	0.002758130	Preprocessor1_Model09
11	roc_auc	binary	0.9578176	5	0.002780992	Preprocessor1_Model08
10	roc_auc	binary	0.9555928	5	0.002849778	Preprocessor1_Model07
8	roc_auc	binary	0.9518674	5	0.003559131	Preprocessor1_Model06
7	roc_auc	binary	0.9483176	5	0.003011255	Preprocessor1_Model05
5	roc_auc	binary	0.9399071	5	0.004371752	Preprocessor1_Model04
4	roc_auc	binary	0.9360005	5	0.006471600	Preprocessor1_Model03
3	roc_auc	binary	0.9234128	5	0.004474684	Preprocessor1_Model02
1	roc_auc	binary	0.8281281	5	0.013140815	Preprocessor1_Model01

After applying log transformation and normalization (Recipe 2), the performance remained nearly identical, with ROC-AUC ≈ 0.959. This shows that, although scaling transformations can theoretically benefit distance-based models, in this dataset, they did not yield significant gains.

Generic Logistic Regression Model

penalty <dbl>	.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
2.920543e-03	roc_auc	binary	0.9586175	5	0.005936503	Preprocessor1_Model08
2.752832e-04	roc_auc	binary	0.9573494	5	0.005656865	Preprocessor1_Model07
1.149985e-10	roc_auc	binary	0.9566028	5	0.005893395	Preprocessor1_Model01
1.444037e-09	roc_auc	binary	0.9566028	5	0.005893395	Preprocessor1_Model02
1.299519e-08	roc_auc	binary	0.9566028	5	0.005893395	Preprocessor1_Model03
1.169299e-07	roc_auc	binary	0.9566028	5	0.005893395	Preprocessor1_Model04
1.572047e-06	roc_auc	binary	0.9566028	5	0.005893395	Preprocessor1_Model05
1.871555e-05	roc_auc	binary	0.9565962	5	0.005892793	Preprocessor1_Model06
5.079465e-02	roc_auc	binary	0.9350875	5	0.008703543	Preprocessor1_Model09
7.795161e-01	roc_auc	binary	0.5000000	5	0.000000000	Preprocessor1_Model10

We fitted regularized logistic regression models with the generic recipe. Logistic regression estimates class probabilities as a linear function of predictors, offering simplicity and interpretability. The best configuration achieved ROC-AUC ≈ 0.959, though performance was sensitive to the penalty parameter. This provides a reliable baseline, but the linear specification may fail to capture nonlinear effects.

Log Logistic Regression Model

penalty <dbl>	.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
2.752832e-04	roc_auc	binary	0.9663347	5	0.004360331	Preprocessor1_Model07
2.920543e-03	roc_auc	binary	0.9627067	5	0.005417137	Preprocessor1_Model08
1.149985e-10	roc_auc	binary	0.9618288	5	0.003849854	Preprocessor1_Model01
1.444037e-09	roc_auc	binary	0.9618288	5	0.003849854	Preprocessor1_Model02
1.299519e-08	roc_auc	binary	0.9618288	5	0.003849854	Preprocessor1_Model03
1.169299e-07	roc_auc	binary	0.9618288	5	0.003849854	Preprocessor1_Model04
1.572047e-06	roc_auc	binary	0.9618288	5	0.003849854	Preprocessor1_Model05
1.871555e-05	roc_auc	binary	0.9618288	5	0.003849854	Preprocessor1_Model06
5.079465e-02	roc_auc	binary	0.9374020	5	0.008763042	Preprocessor1_Model09
7.795161e-01	roc_auc	binary	0.5000000	5	0.000000000	Preprocessor1_Model10

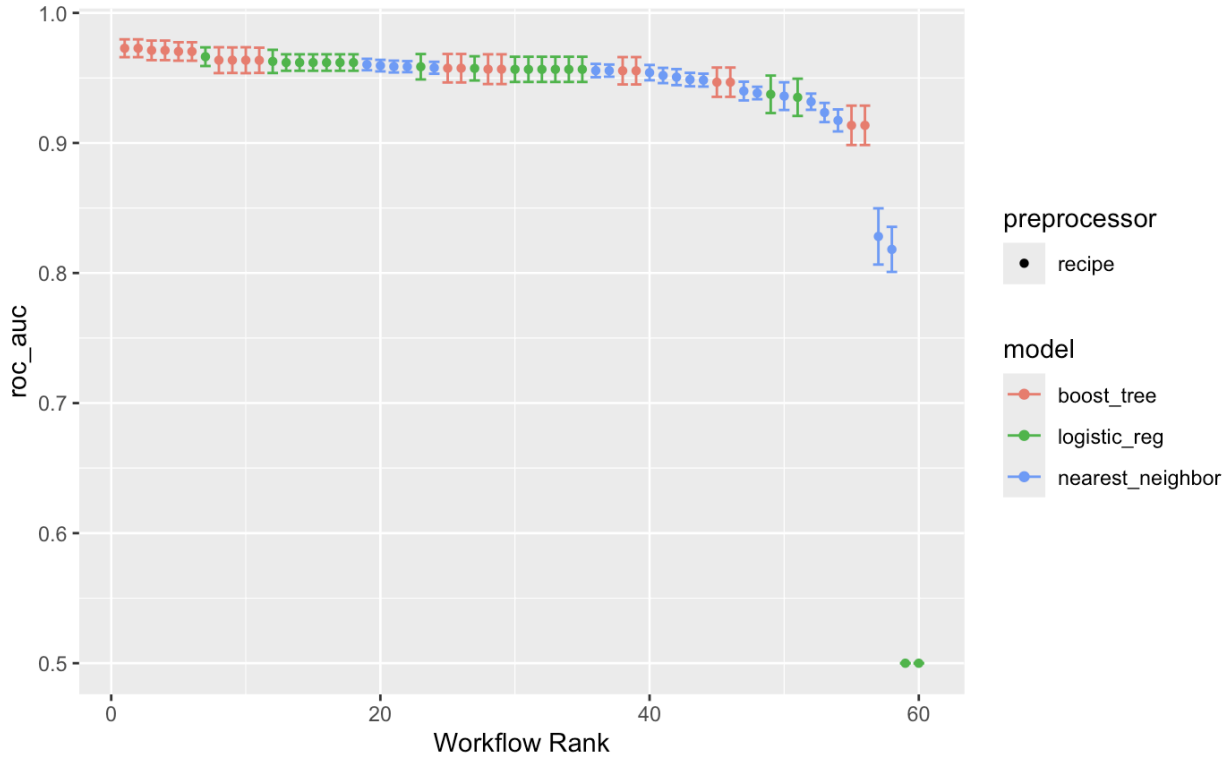
When applying the log transformation before logistic regression (Recipe 2), performance improved slightly. The best model achieved an ROC-AUC ≈ 0.966, reflecting a modest but consistent gain. This improvement likely results from reducing skewness in predictor distributions, which enhances the fit of linear decision boundaries.

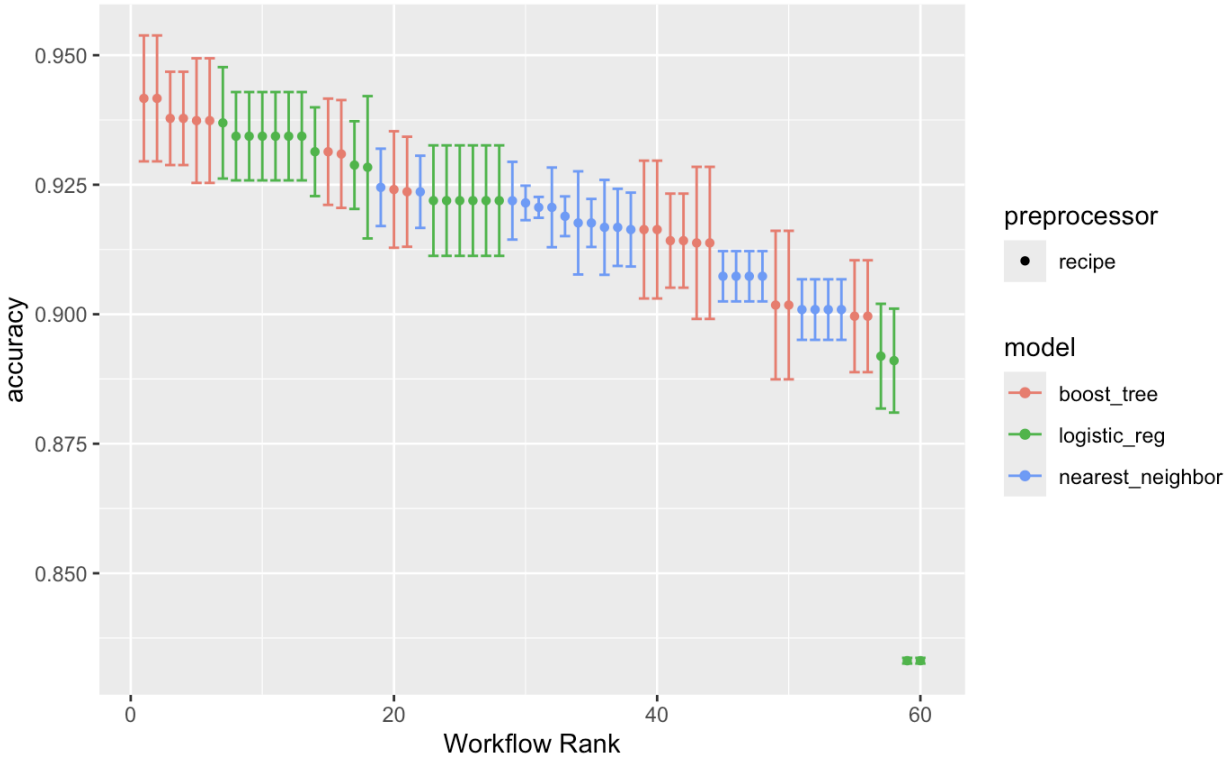
Discussion of Candidate Models

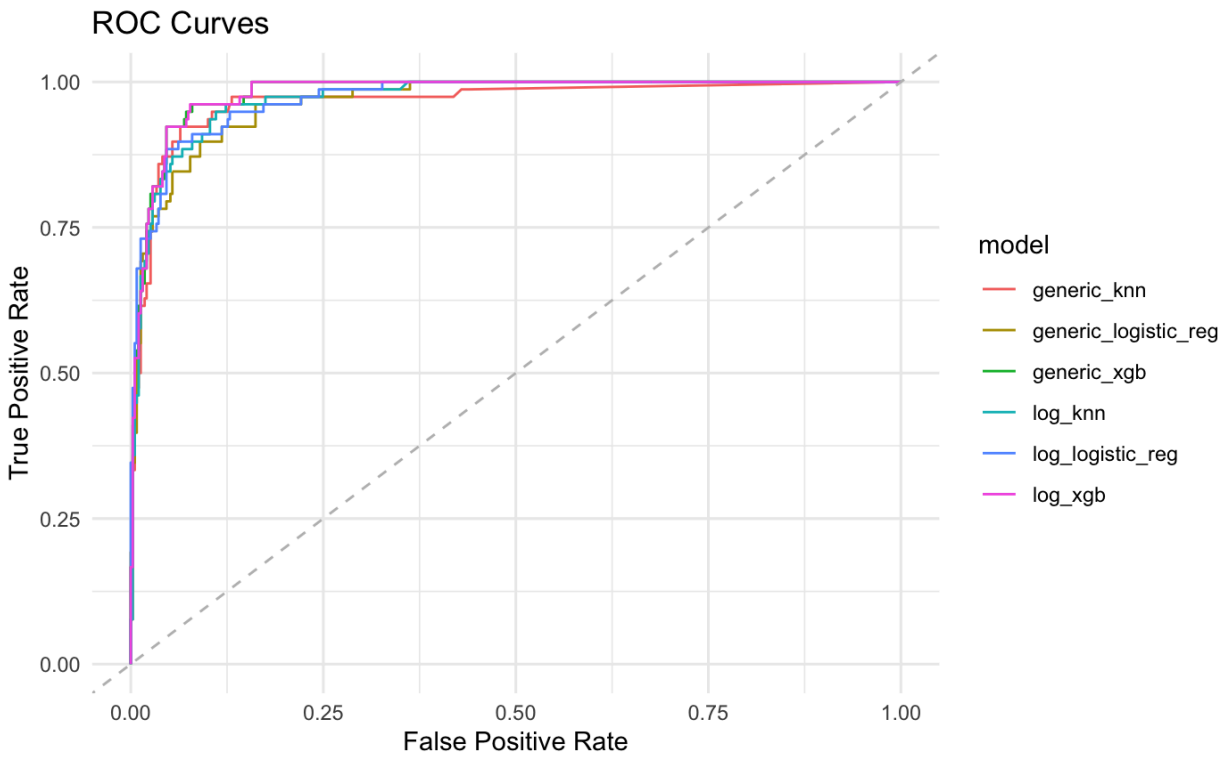
We tested several classification models with two preprocessing strategies: a generic recipe and a log recipe. Using five-fold cross-validation with stratification on the outcome, the Generic XGBoost workflow clearly performed best. It achieved the highest ROC-AUC ≈ 0.973 with a small standard error of about 0.0042, beating both Logistic Regression and K-Nearest Neighbors under either recipe. On an additional 80/20 hold-out split, the same workflow reached an accuracy ≈ 0.944, showing that its performance generalized well beyond cross-validation. Logistic regression and KNN trailed behind in mean AUC, and as expected, the log-transformed recipe did not improve results for XGBoost. This tells us that boosted trees already capture nonlinearities and interactions effectively through their splits, making the simpler generic recipe the right choice in this case.

Model Performance:

Model	ROC-AUC	SE of Metric (if applicable)
Generic XGBoost	0.9728114	0.004164384
Log XGBoost	0.9728114	0.004164384
Generic KNN	0.9603166	0.002670456
Log KNN	0.9594661	0.002688396
Generic Logistic Regression	0.9586175	0.005936503
Log Logistic Regression	0.9663347	0.004360331


1.0
0.9
0.8
0.7
0.6
0.5
roc_auc
0
20
40
60
Workflow Rank
preprocessor
recipe
model
boost_tree
logistic_reg
nearest_neighbor


0.950
0.925
0.900
0.875
0.850
accuracy
0
20
40
60
Workflow Rank
preprocessor
recipe
model
boost_tree
logistic_reg
nearest_neighbor



Discussion of Final Model

The results showed that the Generic XGBoost workflow was the clear winner. It reached a cross-validated ROC-AUC of 0.973 with a very small standard error of about 0.0042 across five folds. On the other hand, Logistic Regression and KNN performed worse under both recipes, and as expected, the log-transformed XGBoost was not better than the generic version. On the 80/20 hold-out split, the Generic XGBoost model achieved an accuracy of 0.944, which shows that the strong performance in cross-validation also generalized to new, unseen data. However, on the training data itself, the workflow reached an accuracy of 1.000, which might indicate overfitting.

The Generic XGBoost workflow had several strengths. First, it gave the best overall ROC-AUC of all workflows, meaning it was the best at telling the classes apart. Second, the small standard error showed that the results were not noisy and were consistent across folds. Third, the simple preprocessing was already enough for strong results, which means we don't need the extra log and normalization steps.

However, there were also some weaknesses. We only searched over a grid of 10 hyperparameter values for each workflow. This saved time but may have missed even better settings, especially because XGBoost can be very sensitive to hyperparameters. Another issue is that we picked the winning model solely from ROC-AUC, where other metrics like accuracy and Brier score could be used to further evaluate the models. Finally, the perfect training accuracy of 1.000 is an

indicator that the model might be overfitting, even though our cross-validation and hold-out results looked healthy. This overfitting hypothesis was somewhat confirmed when the private leaderboard was released, as our submitted XGBoost model predictions dropped our team score from 1st place to 11th place. While this XGBoost model consistently performed well through several layers of testing and cross-validation, the nature of this classification problem means that a few key counties can highly influence the model performance, so the randomness of the train/test split is more influential to the model training as well. In hindsight, a more stable model, especially an ensemble of models, might have performed better on the final Kaggle test dataset.

In conclusion, the Generic XGBoost workflow was our strongest configuration. It had the best cross-validation score with ROC-AUC = 0.973 and SE ≈ 0.0042 across five folds, and it also performed well on the hold-out split with accuracy = 0.944. Its preprocessing was simple, its results were stable, and it clearly outperformed Logistic Regression and KNN. Although hyperparameter search, metric usage, and overfitting evaluation could be improved, this workflow already shows strong and reliable predictive power for the classification task.

Appendix: Final Annotated Script

Note: best_model_xgb.csv is what we submitted to Kaggle. The .Rmd file also exports 6 additional files, the best predictions from each model/recipe.

```
```{r, echo = FALSE}
library(knitr)
```


```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

+ Load the tidyverse and tidymodels

```{r}
library(tidyverse)
library(tidymodels)
library(lubridate)      # year(), month()
library(workflowsets)   # workflow_set(), workflow_map()
library(dials)          # parameter ranges, grids
library(pROC)
library(dplyr)
```


Read in data

```{r}

train <- read_csv("train_class.csv")
test <- read_csv("test_class.csv")
sample <- read_csv('sample_solutions.csv')
col_descriptions <- read_csv('col_descriptions.csv')

```

```{r}
```
```

```
train <- train %>%
  select(-name) %>%
  mutate(winner = factor(winner))

```


Data Preprocessing:

+ Convert variables into percentages of total [within relevant category]. This is sort of a normalization but leaves each predictor more interpretable than step_norm()

```{r}

# Preliminary data cleaning and feature engineering to be implemented in the recipe later
train |> select(-winner) |>
  # Normalize counts related to total population into percentages (so everything have same "weight")
  mutate(across(c(x0026e, x0027e), ~ . / x0025e), # 18years and over (Male/Female)
         across(c(x0030e, x0031e), ~ . / x0029e), # 65years and over (Male/Female)
         across(c(x0040e, x0041e, x0042e, x0043e), ~ . / x0039e), # American Indian sub-races
         across(44:50, ~ . / x0044e), # Asian sub-races
         across(52:55, ~ . / x0052e), # Hawaiian sub-races
         across(c(x0037e, x0038e, x0039e, x0044e, x0052e, x0057e), ~ . / x0036e), # One race (White/Black/American Indian/Asian/Hawaiian)
         across(c(x0059e, x0060e, x0061e, x0062e), ~ . / x0058e), # Two or more races
         across(c(62:67), ~ . / x0001e), # Combinations
         across(c(x0072e, x0073e, x0074e, x0075e), ~ . / x0071e), # Hispanic or Latino
         across(c(x0077e, x0078e, x0079e, x0080e, x0081e, x0082e), ~ . / x0076e), # Not Hispanic or Latino
         across(c(x0084e, x0085e), ~ . / x0083e), # Not Hispanic or Latino (Two or more races)
         across(c(x0088e, x0089e), ~ . / x0087e), # Citizen over 18 (Male/Female)
         across(c(c01_002e, c01_003e, c01_004e, c01_005e), ~ . / c01_001e), # Educations 18-24 yrs
         across(c(93:101), ~ . / c01_006e), # Educations 25+ yrs
         across(c(c01_017e, c01_018e), ~ . / c01_016e), # Educations 25-34 yrs
         across(c(c01_020e, c01_021e), ~ . / c01_019e), # Educations 35-44 yrs
         across(c(c01_023e, c01_024e), ~ . / c01_022e), # Educations 45-64 yrs
         across(c(c01_026e, c01_027e), ~ . / c01_025e) # Educations 65+ yrs
         ) |>
  mutate(across(c(x0025e, x0029e, x0036e, x0039e, x0058e, x0071e, x0076e, x0083e, x0087e,
                  c01_001e, c01_006e, c01_016e, c01_019e, c01_022e, c01_025e), ~ . / x0001e)) |> # Normalize rest by total population
  mutate(across(c(4:25), ~ . / x0001e)) |> #X0002E-X0024E
  select(-id, -x0001e, -x0033e, -x0034e, -x0035e, -x0024e)

```

Model Cross Validation

+ Fold the training data into a 5-fold cross-validation set.

```{r}

set.seed(123)
train_folds <- vfold_cv(train, v = 5, strata = winner)

```

```{r}

head(train) #show head of full expanded training dataset

```

+ Create recipes for model fitting:

```{r}
generic_recipe <- recipe(winner ~ . , data = train) %>% #create baseline recipe with all predictors in dataset
  # Normalize counts related to total population into percentages (so everything have same "weight")
  step_mutate(across(c(x0026e, x0027e), ~ . / x0025e), # 18years and over (Male/Female)
         across(c(x0030e, x0031e), ~ . / x0029e), # 65years and over (Male/Female)
         across(c(x0040e, x0041e, x0042e, x0043e), ~ . / x0039e), # American Indian sub-races
```

```
         across(44:50, ~ . / x0044e), # Asian sub-races
         across(52:55, ~ . / x0052e), # Hawaiian sub-races
         across(c(x0037e, x0038e, x0039e, x0044e, x0052e, x0057e), ~ . / x0036e), # One race (White/Black/American
Indian/Asian/Hawaiian)
         across(c(x0059e, x0060e, x0061e, x0062e), ~ . / x0058e), # Two or more races
         across(c(62:67), ~ . / x0001e), # Combinations
         across(c(x0072e, x0073e, x0074e, x0075e), ~ . / x0071e), # Hispanic or Latino
         across(c(x0077e, x0078e, x0079e, x0080e, x0081e, x0082e), ~ . / x0076e), # Not Hispanic or Latino
         across(c(x0084e, x0085e), ~ . / x0083e), # Not Hispanic or Latino (Two or more races)
         across(c(x0088e, x0089e), ~ . / x0087e), # Citizen over 18 (Male/Female)
         across(c(c01_002e, c01_003e, c01_004e, c01_005e), ~ . / c01_001e), # Educations 18-24 yrs
         across(c(93:101), ~ . / c01_006e), # Educations 25+ yrs
         across(c(c01_017e, c01_018e), ~ . / c01_016e), # Educations 25-34 yrs
         across(c(c01_020e, c01_021e), ~ . / c01_019e), # Educations 35-44 yrs
         across(c(c01_023e, c01_024e), ~ . / c01_022e), # Educations 45-64 yrs
         across(c(c01_026e, c01_027e), ~ . / c01_025e) # Educations 65+ yrs
         ) |>
  step_mutate(across(c(x0025e, x0029e, x0036e, x0039e, x0058e, x0071e, x0076e, x0083e, x0087e,
                  c01_001e, c01_006e, c01_016e, c01_019e, c01_022e, c01_025e), ~ . / x0001e)) |> # Normalize rest
by total population
  step_mutate(across(c(4:25), ~ . / x0001e)) |> #X0002E-X0024E
  step_rm(id, x0001e, x0033e, x0034e, x0035e, x0024e) |>
  step_mutate(income_per_cap_16_20 = (income_per_cap_2020 - income_per_cap_2016)/income_per_cap_2016,
              gdp_grow_16_20 = (gdp_2020 - gdp_2016)/gdp_2016) |>
  step_impute_median(all_numeric_predictors())

#apply log and normalization transformation to all numeric predictors
log_recipe <- generic_recipe %>%
  step_log(all_numeric_predictors(), offset = 1) %>%
  step_normalize(all_numeric_predictors())

recipes <- list( #store 2 recipes
  generic  = generic_recipe,
  log      = log_recipe
)

```

+ Create models, tuning, and workflows:

```{r}
# Prepare models and their tunable parameters
logistic_model = logistic_reg( #Logistic regression model
  penalty = tune()
) %>%
  set_engine('glmnet') %>%
  set_mode('classification')

xgb_model <- boost_tree( #xgboost model
  trees          = tune(),
  tree_depth     = tune(),
  min_n          = tune(),
) %>%
  set_mode("classification") %>%
  set_engine("xgboost")

knn_model <- nearest_neighbor( #K-nearest-neighbors model
  neighbors=tune()
) %>%
  set_engine("kknn") %>%
  set_mode("classification")

models <- list( #store 3 model types we will try (6 total with the 2 recipes)
  xgb = xgb_model,
  logistic_reg = logistic_model,
  knn = knn_model
)
```

+ Create workflow set

```{r}

models_wkfl <- workflow_set(
  preproc = recipes,
  models = models,
  cross = TRUE
)

```

+ Perform cross validation on all model candidates

```{r}

models_fit <- models_wkfl %>% #fit models with 5 fold cv and all recipe combinations
  workflow_map('tune_grid',
               seed = 123,
               verbose = TRUE,
               resamples = train_folds,
               grid = 10
               )

```

+ Extract performance metrics from cross validation

```{r}

performance <- models_fit %>% #get table of accuracy values for each model/recipe combination
  collect_metrics() %>%
  arrange(mean) %>%
  select(wflow_id, mean) %>%
  print()

# Collect metrics for all workflows
all_metrics <- models_fit %>%
  collect_metrics()

# See available metrics
all_metrics %>% distinct(.metric)

best_model <- all_metrics %>%
  filter(.metric == "roc_auc", wflow_id == 'generic_xgb') %>%
  arrange(desc(mean))

best_model <- best_model[1, ]

best_model

```

```{r}

# Extract the full tuning result for that workflow
best_recipe <- models_fit %>%
  extract_workflow_set_result(best_model$wflow_id)

# Pull the best parameter values
best_params <- best_recipe %>%
  select_best(metric = "roc_auc")

# Get the workflow structure and finalize
final_wf <- models_fit %>%
  extract_workflow(best_model$wflow_id) %>%
  finalize_workflow(best_params)

final_wf

```

+ Evaluate the final model using train and test split

```{r}
set.seed(123)
split <- initial_split(train, prop = 0.8, strata = winner)
train_split <- training(split)
test_split  <- testing(split)

# Fit on the full training set
split_fit <- final_wf %>%
  fit(train_split)

# get predictions on training data
test_preds <- predict(split_fit, test_split) %>%
  bind_cols(test_split)

# compute RMSE on test splitted data
accuracy(test_preds, truth = winner, estimate = .pred_class)
```
+ Refit the final model on the full training data

```{r}
# Fit on the full training set
final_fit <- final_wf %>%
  fit(train)

# get predictions on training data
train_preds <- predict(final_fit, train) %>%
  bind_cols(train)

# compute RMSE on training data
accuracy(train_preds, truth = winner, estimate = .pred_class)
```

+ Use `final_fit` to make predictions for the test data

```{r}

pred <- final_fit %>%
  predict(test) %>%
  bind_cols(test)

```

+ Print the first 15 rows of ids and predictions.

```{r}

pred %>%
  select(id, .pred_class) %>%
  head(15)

```

+ Export best model predictions

```{r}

submission <- pred %>% #prepare data table to format of kaggle submission
  select(id, .pred_class) %>%
  rename(winner = .pred_class)

write_csv(submission, 'best_model_xgb.csv') # Export submission file

```

+ Fit and export other models predictions

```{r}
roc_curve_data <- tibble()
test_preds <- tibble()

# Extract the full tuning result for that workflow
for (id in models_fit$wflow_id) {
  # Get the best recipe from the model
  best_recipe <- models_fit %>%
    extract_workflow_set_result(id)

  # Pull the best parameter values
  best_params <- best_recipe %>%
    select_best(metric = "roc_auc")

  # Get the workflow structure and finalize
  final_wf <- models_fit %>%
    extract_workflow(id) %>%
    finalize_workflow(best_params)

  # Fit on the full training set
  final_fit <- final_wf %>%
    fit(train)

  split_fit <- final_wf %>% #fit model into our mini test dataset (derived from training data)
  fit(train_split)

  test_probs <- predict(split_fit, test_split, type = "prob") %>% #store probabilities for ROC curve
    bind_cols(test_split %>% select(winner))

  roc_curve_data <- roc_curve_data %>% #fill in tibble of ROC curve data with this model
    bind_rows(
      roc_curve(test_probs, truth = winner, .pred_Biden) %>%
        mutate(model = id)
  )

  # get predictions on training data
  pred <- final_fit %>%
    predict(test) %>%
    bind_cols(test)

  #prepare data table to format of kaggle submission
  submission <- pred %>%
    select(id, .pred_class) %>%
    rename(winner = .pred_class)

  # Export submission file
  out_file <- sprintf("sub_%s_%s.csv", id, Sys.Date())
  write_csv(submission, out_file)
  cat(sprintf("\n[Wrote] %s\n", out_file))
  print(head(submission, 15))
}
```

Create ROC curve for best models

```{r}

#plot ROC curves with best of each model
ggplot(roc_curve_data, aes(x = 1 - specificity, y = sensitivity, color = model)) +
  geom_path() +
  geom_abline(lty = 2, color = "gray") +
  labs(title = "ROC Curves", x = "False Positive Rate", y = "True Positive Rate") +
  theme_minimal()


```

Appendix: Team Member Contributions

Data Mining/Manipulation/Pre-Preprocessing: Alfred Mastan, Quinn Koch, Joseph Choi
Model Selection/Hyperparameter Tuning: Kwanhee Yoon, Johnathan Pham, Alfred Mastan
Report: Alfred Mastan, Quinn Koch, Joseph Choi, Johnathan Pham, Kwanhee Yoon