Classification Report

Eric Chu, Van-An Lam, Ava Adlao, William Martin-McClay, Pavan Sah

Set-Up

If you want to knit the PDF but don't want to run the code chunks, set eval = FALSE.

```
knitr::opts_chunk$set(echo = FALSE, eval = TRUE)

set.seed(161)
```

Import Libraries and Data

```
library(tidyverse)
library(tidymodels)
library(ggplot2)
library(readr)
library(patchwork)

train <- read_csv("../Data/train_class.csv")
test <- read_csv("../Data/test_class.csv")
```

Introduction

Many models are created to attempt to predict presidential election winners. The purpose of this model is to predict the winning presidential candidate per county using the 2020 election as the training and testing data. The data includes over 3,000 counties across the US. All data was acquired from the US Census Bureau.

We believe a county's type (i.e. urban or rural) is its most substantial predictor, as urban counties tend to exclusively vote Democrat. Furthermore, we also believe the variables associated with the county's result is the age and education level, as younger and more educated voters tend to vote Democrat, with vice versa demographics typically voting Republican.

Exploratory Data Analysis

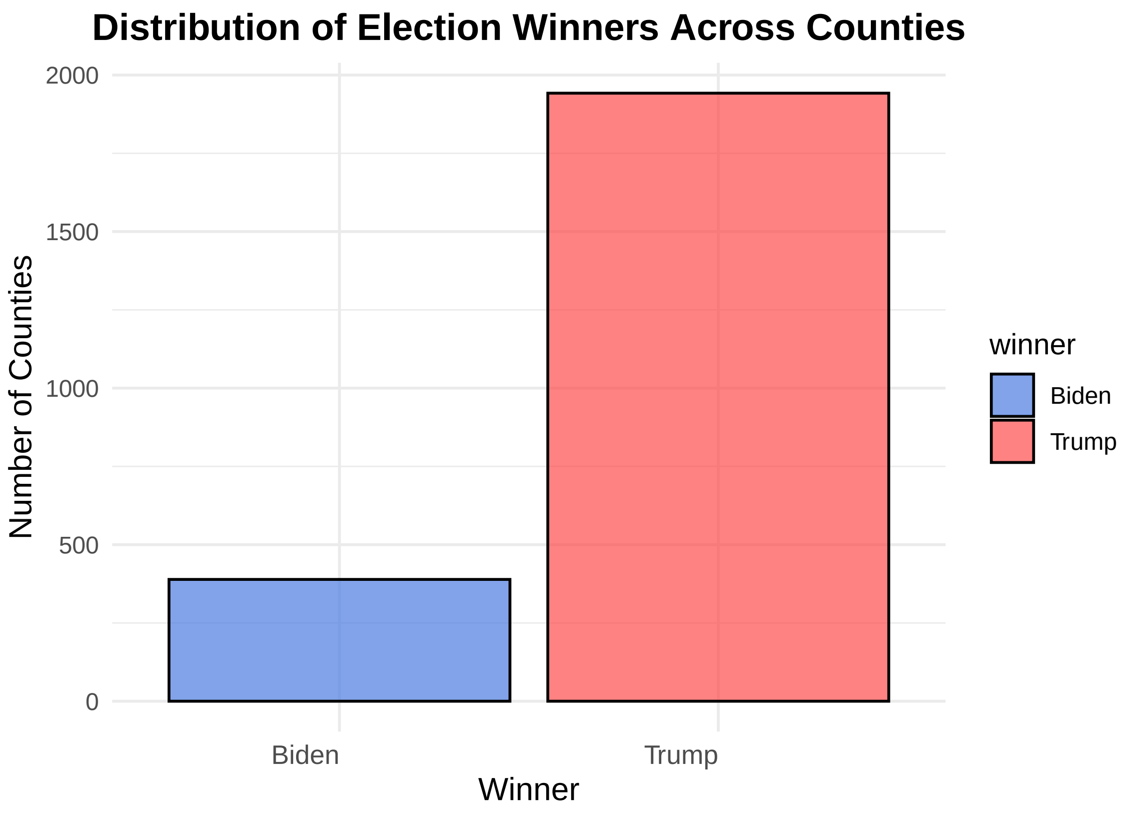

Figure 1: **Distribution of Election Winners Across Counties**

There is a clear imbalance in the number of counties won by each candidate as the plot shows a significant skew toward Trump, with him receiving nearly 2000 votes compared to Biden's approximately 500 votes. This suggests that Trump was more successful in a larger number of counties. This does not, however, account for the population density of each county.

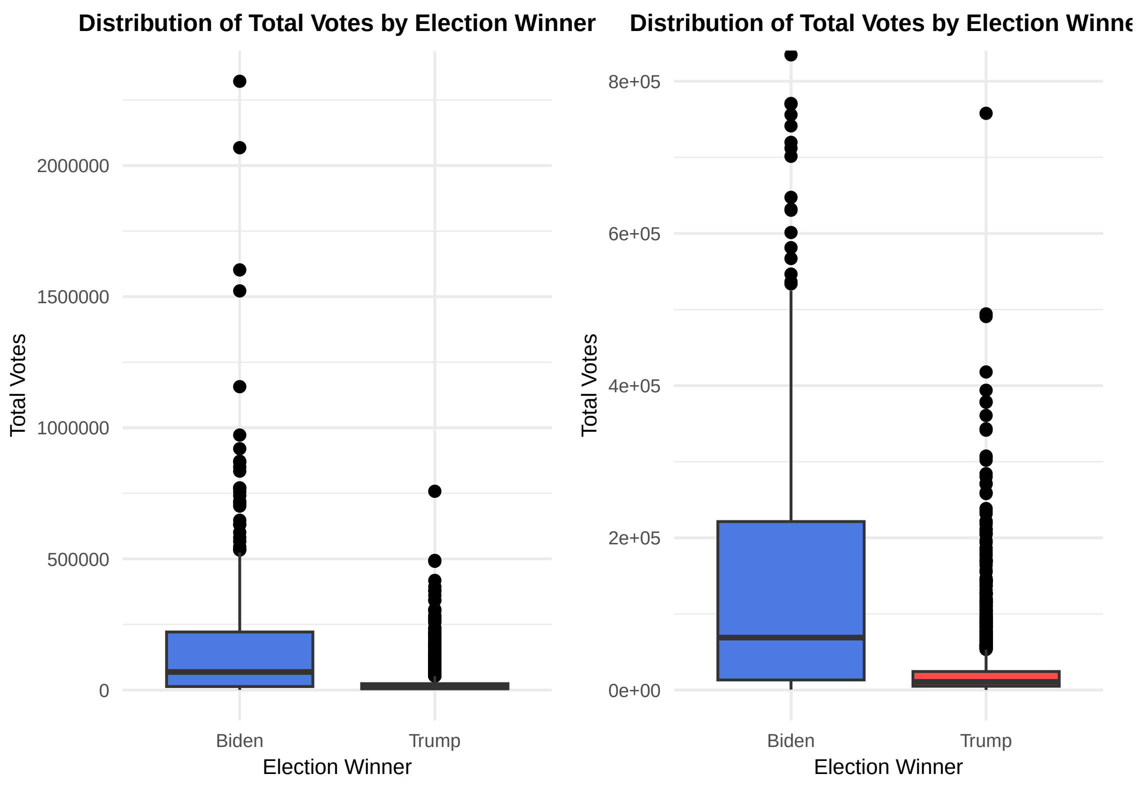

Figure 2: **Distribution of Total Votes by Election Winners**

There is also a clear imbalance in the number of total votes for each candidate. Biden's boxplot is wider and higher, which indicates that in the counties where Biden won, the total number of votes cast was generally higher. This pattern indicates that Biden's support was likely stronger in more densely populated areas or regions with higher voter turnout, compared to the counties won by Trump. To investigate the difference in the distribution of wins vs. total votes, we want to investigate demographic or economic characteristics that are associated with the counties won by Trump or Biden.

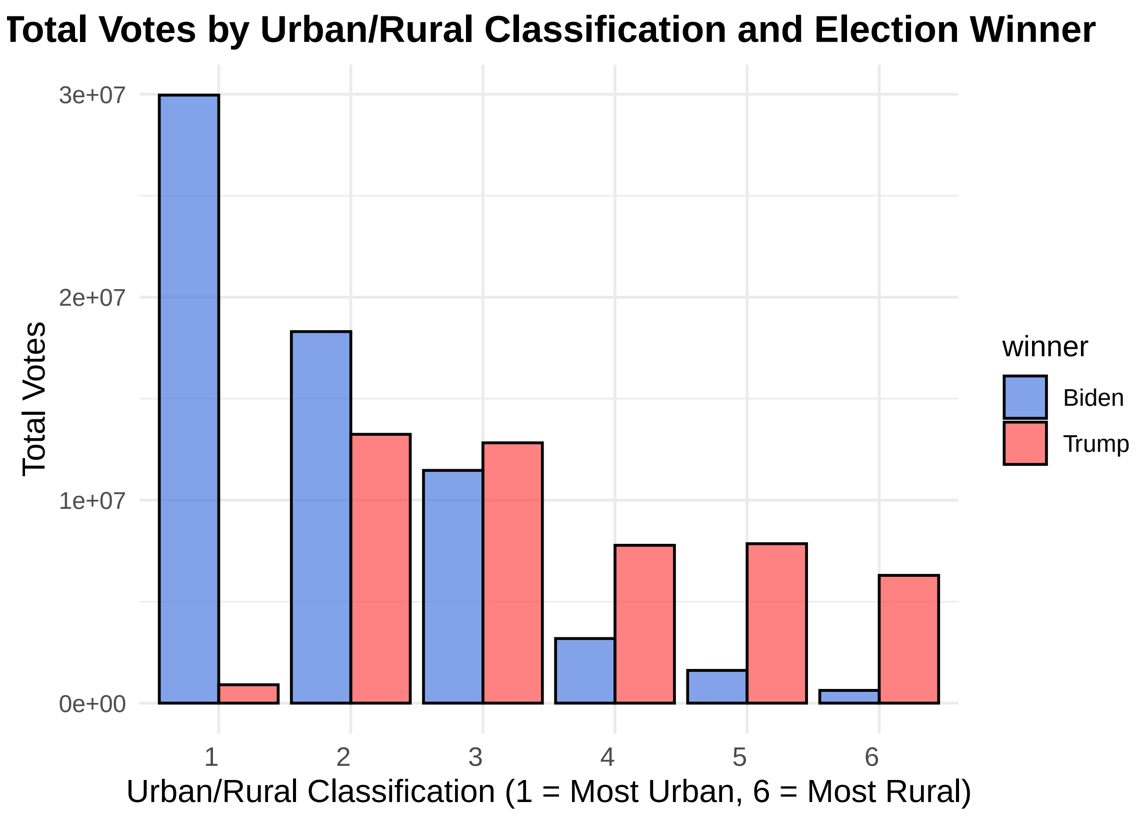

Figure 3: **Total Votes by Urban/Rural Classification and Election Winners**

This bar plot helps us compare the total votes cast for each candidate in different urban/rural categories. This visual, especially at the extremes of 1 = Most Urban and 6 = Most Rural, highlights and supports the fact that more urban areas have higher votes for Biden, whereas more rural areas have higher votes for Trump.

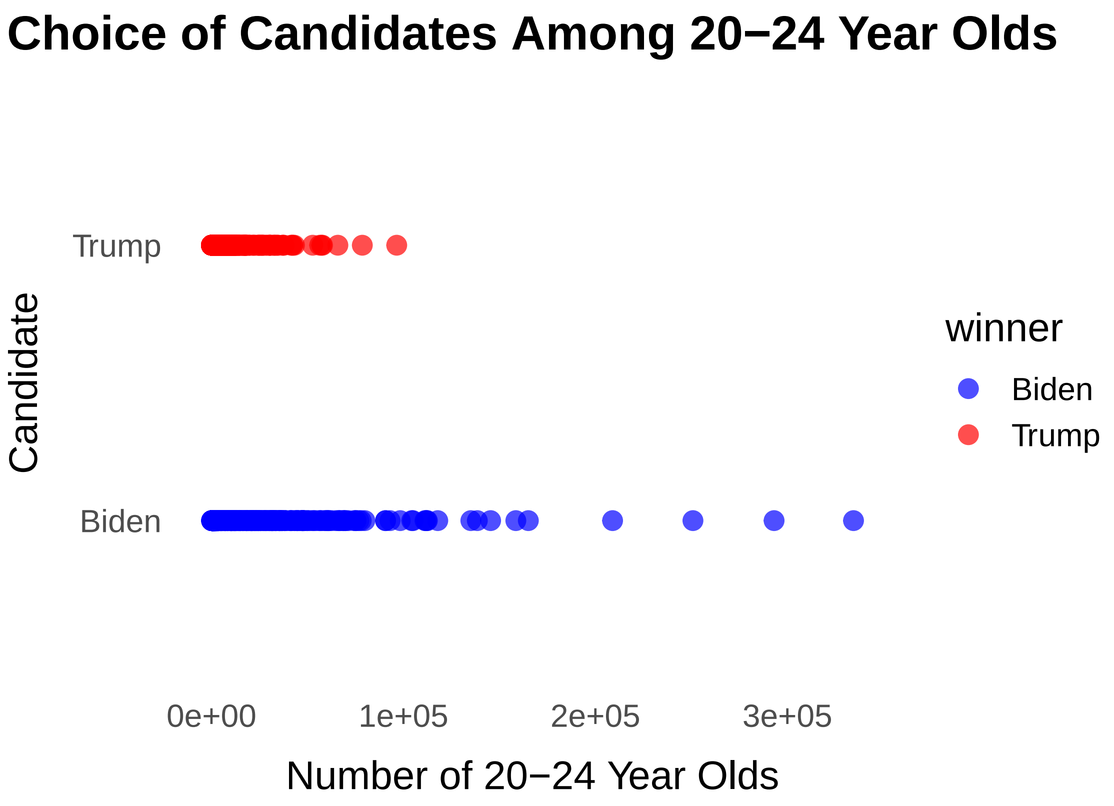

Figure 4: **Choice of Candidates Among 20-24 Year Olds**

The plot illustrates the preference for presidential candidates among 20-24 year-olds. As expected, this age group heavily skewed towards Biden, shown by the concentration of blue points. However, there was still a notable level of support for Trump, indicating that the Republican party retains some vitality and youth support, more than we initially expected.

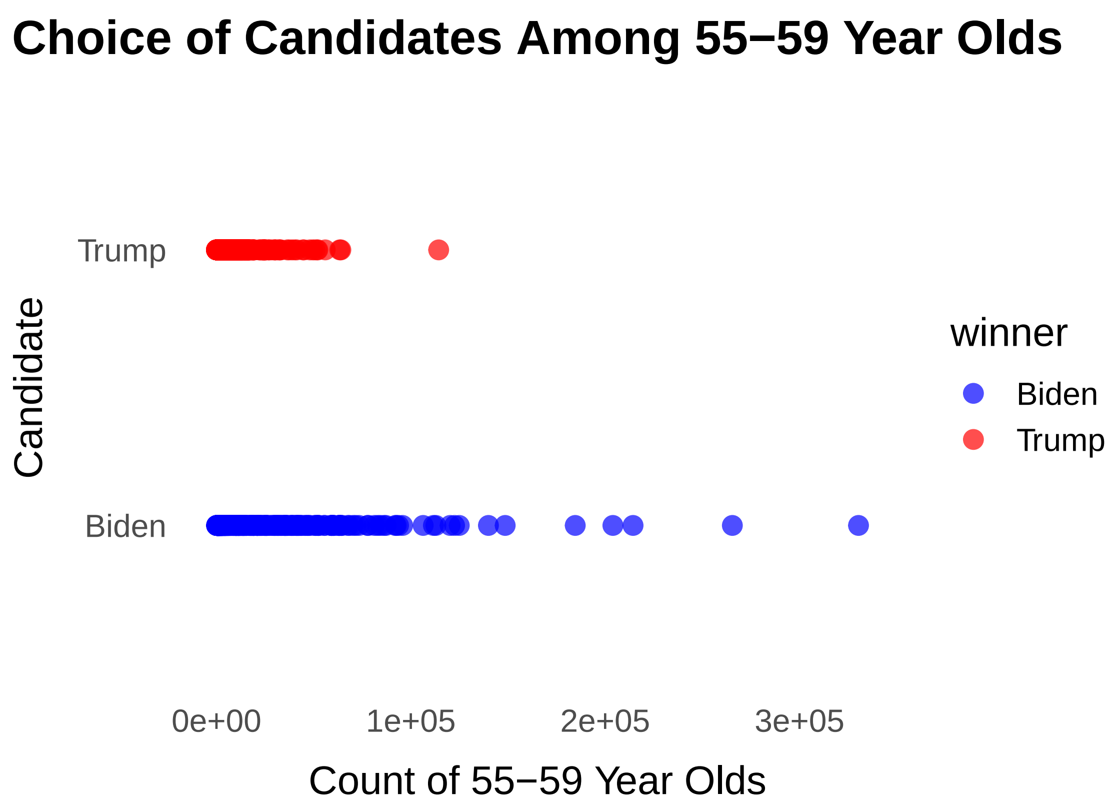

Figure 5: **Choice of Candidates Among 55-59 Year Olds**

The plot illustrates candidate preferences among individuals aged 55-59. Biden is the clear leader in this age group, which aligns with the broader trend of Democratic candidates often winning the popular vote. This observation highlights the tendency of Democratic candidates to perform well across various demographics.

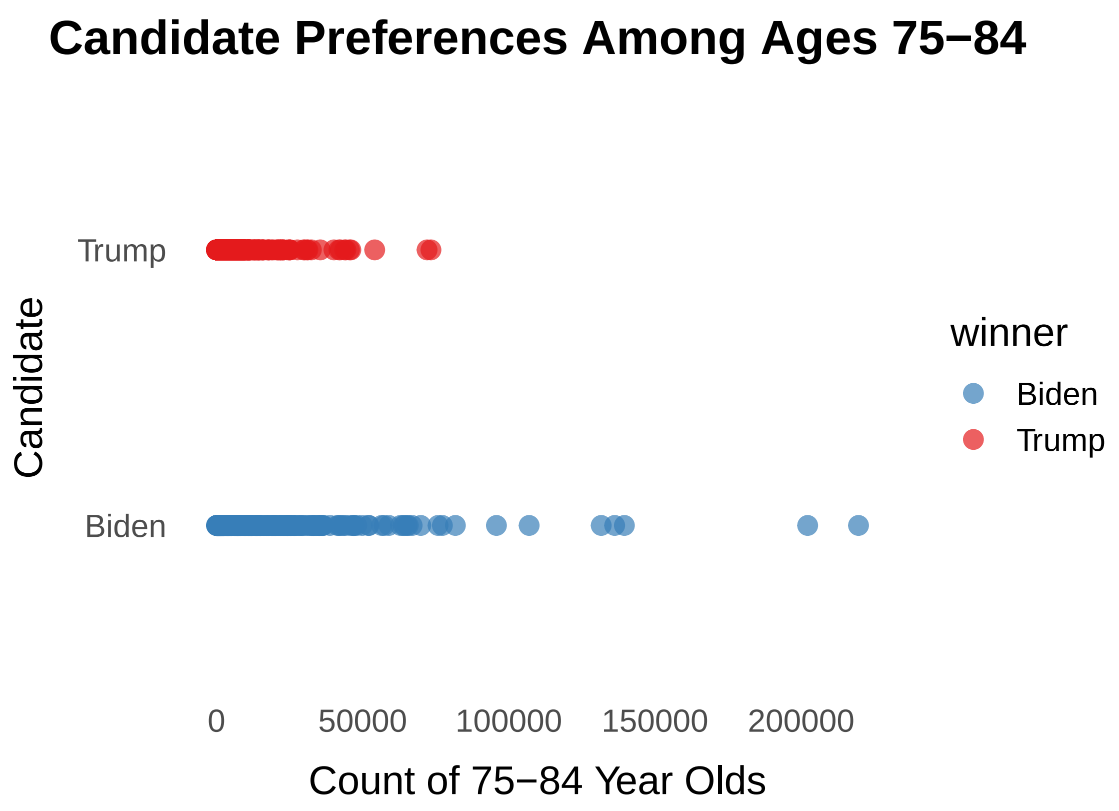

Figure 6: **Choice of Candidates Among 75-84 Year Olds**

This plot visualizes the voting preferences of individuals aged 75-84. Contrary to my initial assumption that this age group would lean towards Trump, the data shows a more balanced distribution between Trump and Biden. While Biden maintains a clear edge, the preferences are notably more even compared to other age groups.

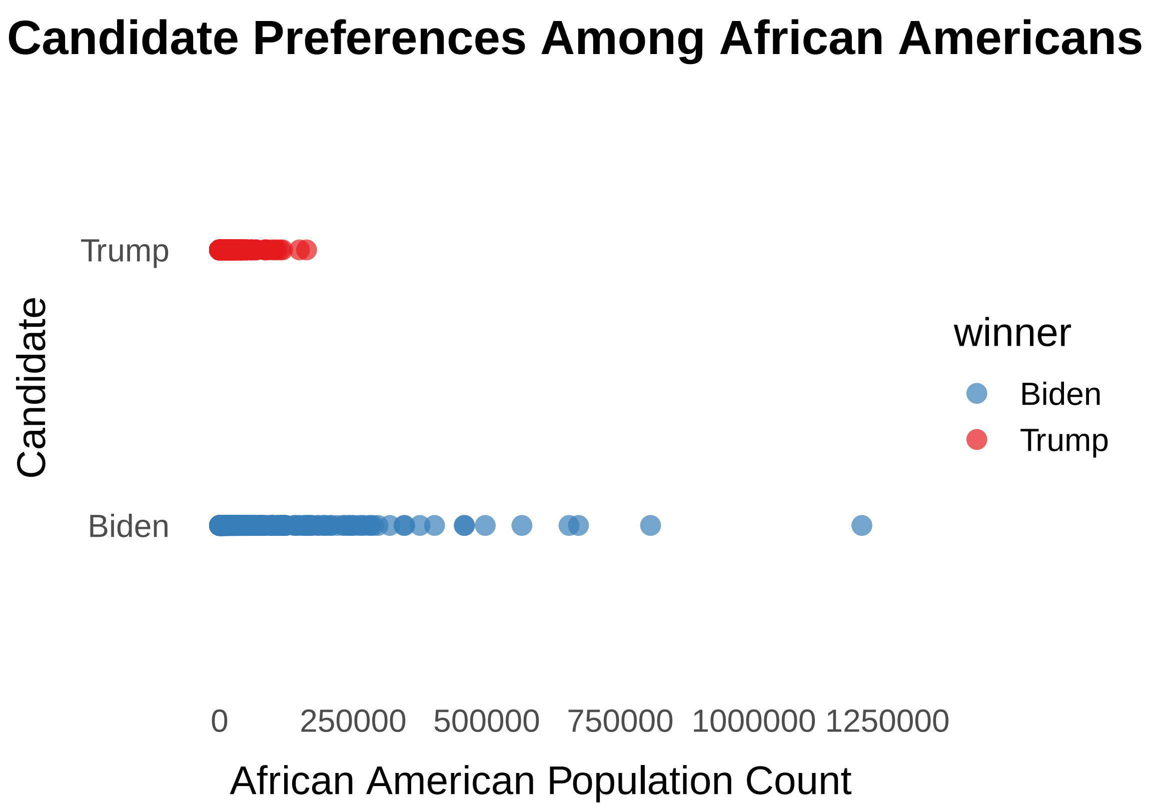

Figure 7: **Candidate Preferences Among African American Voters**

This plot explores how the African American population leans in the general election. Reflecting news stories about the significant impact of the Black vote in Biden's primaries, the data shows a strong preference for Biden among this demographic.

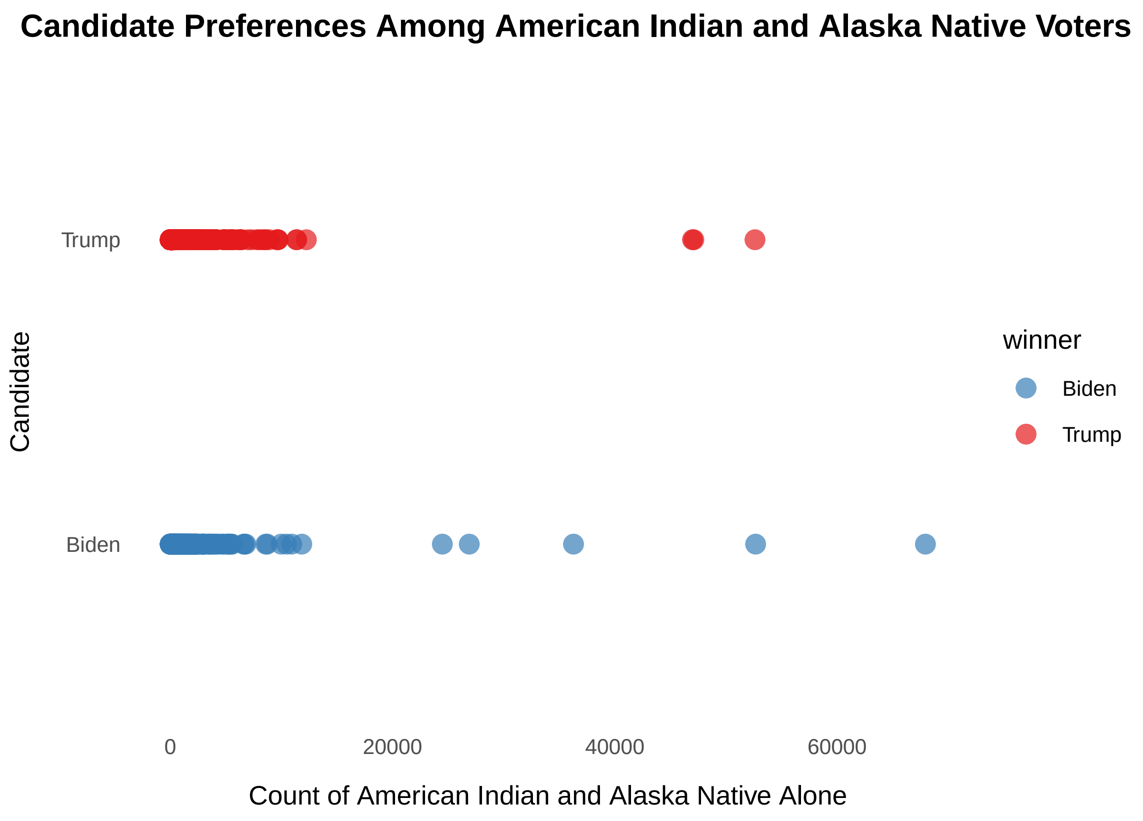

Figure 8: **Candidate Preferences Among American Indian and Alaska Native Voters**

The plot reveals an unexpected balance in voting preferences among American Indian and Alaskan Native populations, contrary to the assumption that this group would lean more liberal. The vote appears to be more evenly split between the candidates.

Preprocessing/Recipes

Minimal preprocessing was done on our best model. The only preprocessing step was to correctly identify the categorical variables. Little preprocessing was done because in the context of boosted trees, hyperparamter tuning is more importance than preprocessing. When the final model was created, this was a decision made to save time, but when we revisited the model after hyperparamter tuning we attempted to use some preprocessing to improve the score, but it turned out to have lower performance.

Candidate Models, Evaluation, and Tuning

Candidate Models and Evaluation

Table 1. Summary of Candidate Models

Model Num	Model Type	Engine	Author	Hyperparameters
Model 1	Logistic Regression	glmnet	Van-An	$mixture = 0.333, penalty = 10e^{-10}$
Model 2	KNN (untuned)	kknn	Eric	N/A
Model 3	GBM	ranger	Ava	trees = 59, min_n = 3, learn_rate = 1.22, loss_reduction = 1.27
Model 4	KNN tuned	kknn	Eric	$neighbors = 13$
Model 5	xgboost	xgboost	Liam	mtry = 90, trees = 1000, min_n = 7, tree_depth = 2, learn_rate = 0.04185, loss_reduction = $5.8758 * 10e^{-08}$, sample_size = 0.7967
Model 6	xgboost with recipe	xgboost	Eric	mtry = 20, trees = 1000, min_n = 3, tree_depth = 6, learn_rate = $4.1084 * 10e^{-10}$, loss_reduction = $2.2940 * 10e^{-6}$, sample_size = 0.3345
Model 7	Random Forest	ranger	Pavan	mtry = 98, trees = 50, min_n = 1

Table 2. Description of Candidate Models

Model Num	Description
Model 1	A logistic multiple linear regression that uses 'voting_recipe' and 10-fold cross validation, using proportions for each variable category.
Model 2	A KNN classification model that uses 'knn_basic_recipe', 10-fold cross validation, but is not tuned.
Model 3	A GBM model that uses 'gbm_recipe', 10-fold cross validation, and uses a grid search for hyperparameter tuning.
Model 4	The same KNN model as `Model 2` but with hyperparameter tuning conducted.
Model 5	An xgboost tree that is **not** pre-processed but tuned. Also uses 10-fold cross validation.
Model 6	The same xgboost model as `Model 5` but with `xgb_recipe` containing `step_*()` functions for preprocessing.
Model 7	A random forest model using 10-fold cross validation and is tuned. Did not include populations under 18 (they cannot vote).

Table 3. Recipes of Candidate Models

Model Num	Model Type	Author	Recipe/Formula
Model 1	Logistic	Van-An	`voting_recipe <- recipe(winner ~ ., data = train) %>% step_impute_bag(starts_with("income"), starts_with("gdp")) %>% step_mutate(male_prop = x0002e / x0001e, female_prop = x0003e / x0001e, black_prop = x0038e / x0001e, white_prop = x0037e / x0001e, native_prop = x0039e / x0001e, asian_prop = x0044e / x0001e, pacific_prop = x0052e / x0001e, other_prop = x0057e / x0001e, hispanic_prop = x0071e / x0001e, lessHS_prop = (c01_002e + c01_007e + c01_008e) / (c01_001e + c01_006e), gradHS_prop = (c01_003e +c01_009e) / (c01_001e + c01_006e), someAssoc_prop = (c01_004e +c01_010e) / (c01_001e + c01_006e), bachUp_prop = (c01_005e +c01_015e) / (c01_001e + c01_006e), under18_prop = x0019e / x0001e, over18_prop = x0025e / x0001e`

Model Num	Model Type	Author	Recipe/Formula
Model 2	KNN	Eric	`knn_basic_recipe <- recipe(winner ~ ., data = train) %>% step_log('total_votes', offset = 1) %>% step_rm('id') %>% # Don't use these as predictors step_dummy('x2013_code') %>% step_impute_mean(all_numeric()) %>% step_normalize(all_numeric()) %>% step_interact(terms = ~x0002e:x0003e) %>% step_interact(terms = ~ x0005e:x0006e:x0007e:x0008e:x0009e:x0010e:x0011e:x0012e:x00 %>% step_interact(terms = ~ x0019e:x0020e:x0021e:x0022e:x0023e:x0024e) %>% step_interact(terms = ~ x0025e:x0026e:x0027e:x0029e:x0030e:x0031e) %>% step_interact(terms = ~ x0034e:x0035e:x0036e:x0037e:x0038e:x0039e:x0040e:x0041e:x00 %>% step_interact(terms = ~ x0058e:x0059e:x0060e:x0061e:x0062e:x0064e:x0065e:x0066e:x00 %>% step_interact(terms = ~ x0071e:x0072e:x0073e:x0074e:x0075e) %>% step_interact(terms = ~ x0076e:x0077e:x0078e:x0079e:x0080e:x0081e:x0082e:x0083e:x00 %>% step_interact(terms = ~ x0087e:x0088e:x0089e) %>% step_interact(terms = ~ c01_001e:c01_002e:c01_003e:c01_004e:c01_005e:c01_006e:c01_0 %>% step_interact(terms = ~ income_per_cap_2016:income_per_cap_2017:income_per_cap_2018 %>% step_interact(terms = ~ gdp_2016:gdp_2017:gdp_2018:gdp_2019:gdp_2020)`
Model 3	GBM	Ava	`gbm_recipe <- recipe(winner ~ ., data = train) %>%   step_normalize(all_predictors())`
Model 4	KNN tuned	Eric	same recipe as **Model 2**
Model 5	xgboost	Liam	`add_formula(winner~.)`
Model 6	xgboost with recipe	Eric	`xgb_recipe <- recipe(winner ~ ., data = train) %>% step_log('total_votes', offset = 1) %>% step_rm('id') %>% # Don't use these as predictors step_dummy('x2013_code') %>% # Make sure to normalize after encoding step_impute_mean(all_numeric()) %>% step_normalize(all_numeric()) # if we don't drop id then don't forget to avoid normalizing it with all_of('id')`

Model Num	Model Type	Author	Recipe/Formula
Model 7	random forest	Pavan	`rf_recipe <- recipe(winner ~ ., data = train) %>% step_dummy('x2013_code') %>% step_rm('x0005E', 'x0006E', 'x0019E') %>% step_impute_median(income_per_cap_2016, income_per_cap_2017, income_per_cap_2018, income_per_cap_2019, income_per_cap_2020, gdp_2016, gdp_2017, gdp_2018, gdp_2019, gdp_2020)`

Table 4. Performance of Candidate Models

Metric	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7
Accuracy	0.933	0.85355	0.9960	0.89539	0.94560	0.94142	0.91205
Std. Error	0.0044	0.0058	0.0044	0.005247	0.0052123	0.0065	0.003404

Visualizing Performance of Candidate Models

Accuracy Graphs

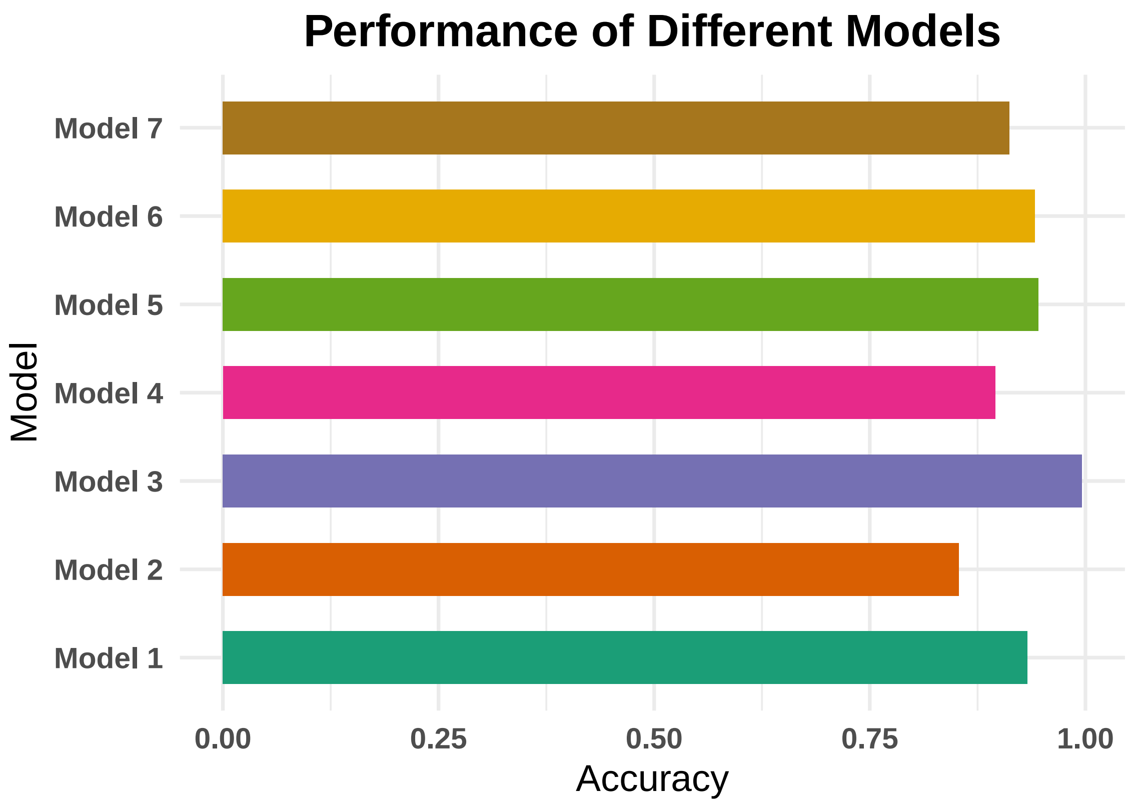

Figure 9: **Accuracy Scores Across Candidate Models**

This metric measures the proportion of correctly classified instances among the total instances, indicating overall correctness.

Standard Error Graphs

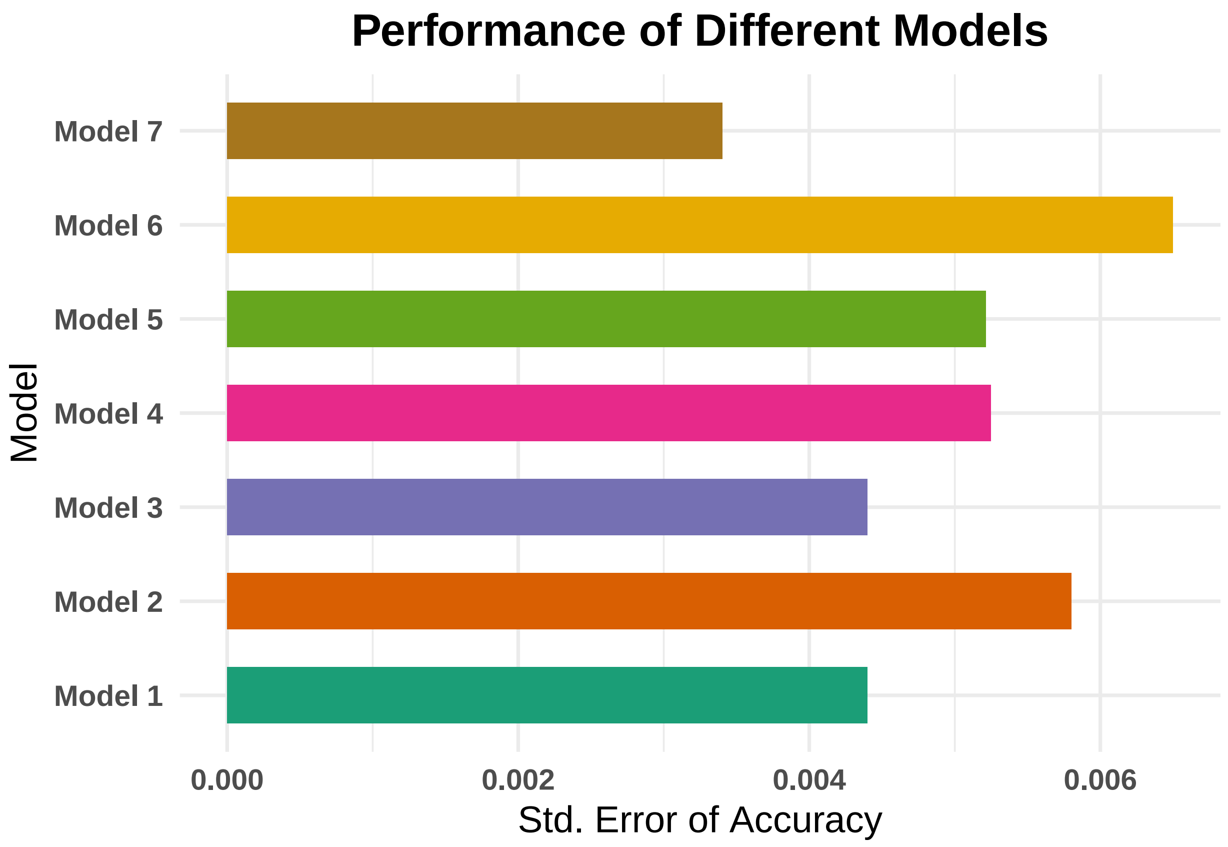

Figure 10: **Standard Error of Accuracy Scores Across Candidate Models**

Discussion of Final Model

The final model was created while following a tutorial from one of Rstudio's finest, Julia Silge. We choose an xgboost model because it is a very common way to extract predictions from tabular data. Our first attempt at this model turned out to be our best, so there weren't many iterations made, despite our best attempts.

Model 5 was chosen as the final model due to the fact that it had the best performance in terms of accuracy.

Despite the fact that it does not contain any pre-processing steps, it still performed the best among all the candidate models.

Hypertuning Process

We didn't opt to do any preprocessing, instead spending our time on tuning almost every hyperparameter:tree_depth, min_n, loss_reduction, sample_size, mtry, and learn_rate. This provided a learning experience over our model for the regression project. In the regression project, we used tune_grid() to tune our models, but learned that when tuning more than two parameters, it took a very long time. Facing six parameters meant we needed a new approach. This is why we chose to use grid_latin_hypercube(). This allowed the tuning process to take under 20 minutes with size = 10. This process led to following graph, and it ended up being our best model. After seeing the success of the first iteration of the model, we retried it with size = 50, hoping to find an even better combination of parameters. However, the combinations that were found performed worse.

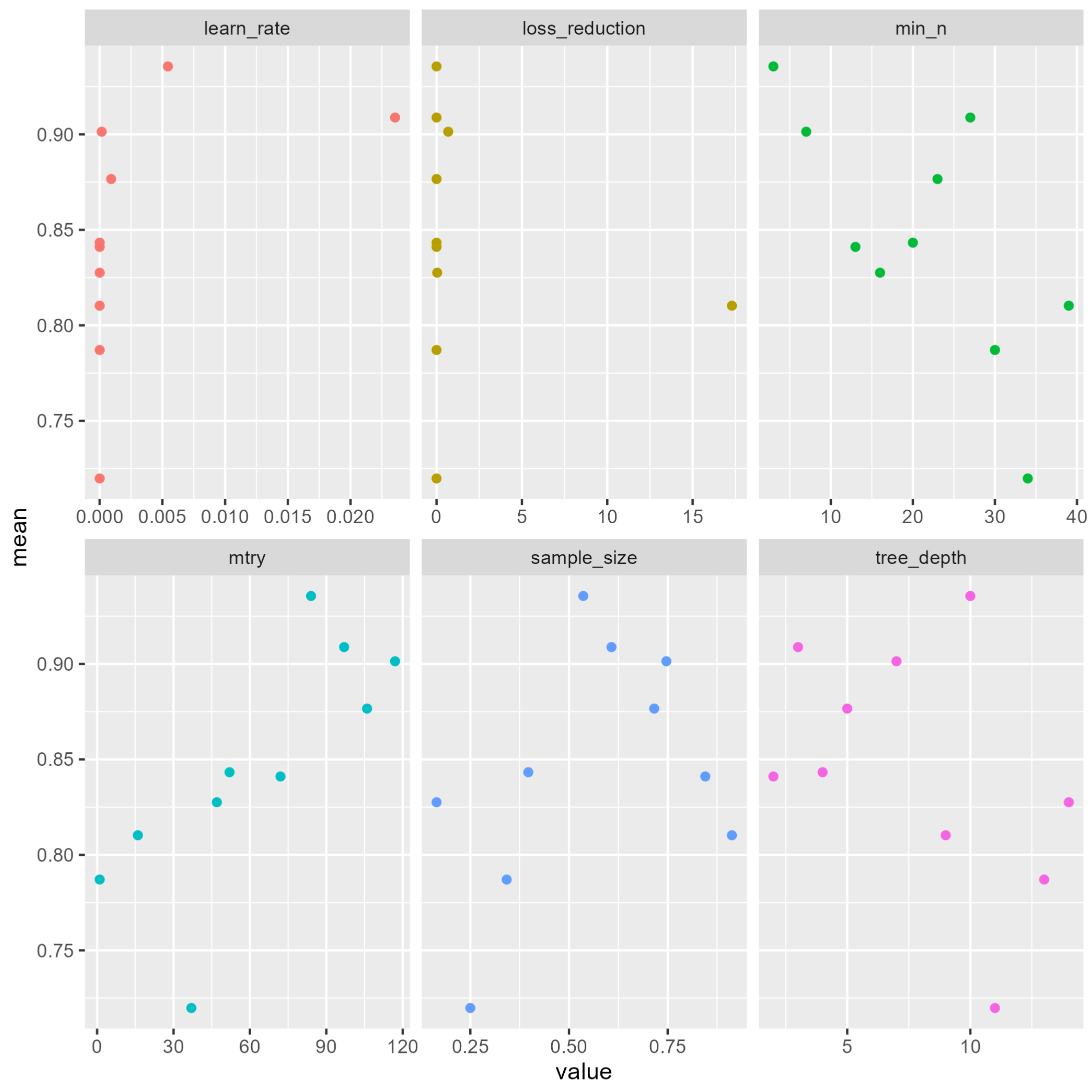

Figure 11: **Hypertuning Process**

Strengths and Weaknesses of the Final Model

There are several ways to improve the model. Namely, creating a better and more elaborate recipe could improve the model. We can add more `step_*()` pre-processing functions to improve the model. There are some things that aren't taken into account, including:

- duplicate columns
- columns that are almost duplicates and could cause huge interaction effects/multicollinearity
- the fact that the `total_votes` feature is skewed and could be log transformed
- Normalization
- Encoding a feature (e.g. `x2013_code`)
- Avoid using non-predictor columns as predictors (e.g. `id`)

While these issues were attempted to be resolved in **Model 6**, this somehow resulted in a slightly less accurate model. These attempts could further be improved in future recipes. This isn't completely unexpected from a boosted_trees model; it suggests that maybe we should only return to preprocessing for this model when other, bigger improvements or changes have been made.

I think the main weakness of this model is that it was not part of an ensemble model. This xgboost model was well tuned for the conditions, and further testing showed that more preprocessing steps, or more (size) testing didn't seem to improve it. In this sense, I think the xgboost model was almost as good as it could have been. However, we think we would have seen better predictive performance using an ensemble model. While one of our group members, Liam, attempted to get an ensemble model going, he did not finish before the competition deadline. Many of our group members were busy with other classes and responsibilities, and this competition served as a reminder that a good model takes a lot of time to make. We believe this was our biggest barrier to a better model.

Additional Data

Potential useful additional data includes, but is not limited to:

- More granular information regarding education and age groups
    - (e.g. how many people over the age of 65 have master's or have not finished high school)
- complete data regarding income per capita and GDP
- each county's 25th and 75th percentile in income
- Proportion of ethnic minority voters relative to proportion of white voters
    - i.e. do national ethnic minorities make up the majority of the voting population in that county?

Appendix: Final Annotated Script

```
# Import Libraries & Data
library(tidyverse)
library(tidymodels)
library(xgboost)
library(vip)

train <- read_csv("../Data/train_class.csv")
test <- read_csv("../Data/test_class.csv")

# Remove name column and convert character columns into features
train <- train %>%
  select(-'name') %>%
#  mutate(across('x2013_code', as.factor)) %>%
  mutate_if(is.character, factor)

test <- test %>%
#  mutate(across('x2013_code', as.factor)) %>%
  mutate_if(is.character, factor)

# Set engine and tuning parameters
xgb_spec <- boost_tree(trees = 1000,
           tree_depth = tune(), min_n = tune(),
           loss_reduction = tune(), sample_size = tune(),
           mtry = tune(),learn_rate = tune()
) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

# Create tuning grid
xgb_grid <- grid_latin_hypercube(
  tree_depth(),
  min_n(),
  loss_reduction(),
  sample_size = sample_prop(),
  learn_rate(),
  finalize(mtry(), train),
  size = 10
)

# Create workflow with no pre-processing
# i.e. simply having the formula
xgb_wf <- workflow() %>%
  add_formula(winner~.) %>%
  add_model(xgb_spec)

# Create cross validation folds using default of 10
set.seed(101)
vb_folds <- vfold_cv(train, strata = winner)

set.seed(202)
# Execute tuning, this will take a while
xgb_res <- tune_grid(
  xgb_wf,
  resamples = vb_folds,
  grid = xgb_grid,
  control = control_grid(save_pred = TRUE, verbose = TRUE)
)

# Obtain results
xgb_res %>%
  collect_metrics() %>%
  filter(.metric == "roc_auc") %>%
  select(mean,mtry:sample_size) %>%
  pivot_longer(mtry:sample_size,
               names_to = "parameter",
               values_to = "value") %>%
  ggplot(aes(value, mean, color = parameter)) +
  geom_point(show.legend = FALSE) +
facet_wrap(~parameter, scales = "free_x")

# Print the optimized hypertuned parameters
show_best(xgb_res, metric = "roc_auc")
best_auc <- select_best(xgb_res, metric = "roc_auc")
final_xgb <- finalize_workflow(xgb_wf, best_auc)
final_xgb

# Create predictions & save them to a .csv
final_res <- final_xgb %>%
  fit(data = train)
predictions <- final_res %>%
  predict(new_data = test)

pred_table_2 <- bind_cols(test %>% select(id), predictions)

write_csv(pred_table_2, "xgb_predictions_tuned.csv")
```

Appendix: Team Member Contributions

- Eric: Created tables in report, added some insight in discussions, created KNN models **Model 3** and **Model 2** and **4**, created a recipe for **Model 5** and used that to create **Model 6**, documented findings on EDA of column types
- Van-An: Creation of **Model 1** and other (logistic regression + linear discriminant analysis (LDA)) models that were not selected to be candidate models
- Ava: Creation of some exploratory data analysis visuals, creation of a candidate model, compiled information for the report, created tables in report, created **Model 2**
- Liam: Created **Model 5**, attempted stack, wrote about **Model 5** on report
- Pavan: Half of exploratory data analysis, created the random forest **Model 7**