University of California

Los Angeles

Kaggle Regression Project Report

By

Mayerli Cordero-Cortez

Eric Gallardo

Avanthika Panchapekesan

Shayan Saadat

Luke Villanueva

Stats 101C:

Introduction to Statistical Models and Data Mining

Professor Miles Chen

08 September 2023

Contents

1 Introduction

The goal of this project is to develop a predictive model that leverages demographic, educational, and socioeconomic information in order to predict the percentage of voters in different counties across the United States who voted for Joe Biden in the 2020 US Presidential election. Based on background information, we originally concluded that income level, race, and education were likely to be associated with the response variable—the percent of Biden voters in different counties. Previous studies and research have consistently outlined that income, race, and education are key demographic patterns that have historically influenced voting patterns (Pew Research Center). For instance, members of a low-income community are likely to vote for more liberal policies that aim to increase minimum wages and expand affordable healthcare, two platforms that Biden ran on in the 2020 election (National Low Income Housing Coalition). On the other hand, wealthier communities, especially those associated with larger corporations, will likely support Republican candidates because this party generally advocates for lower taxes and capitalist-friendly policies (Bhattacharya).

2 Exploratory Data Analysis

2.1 Voter Demography

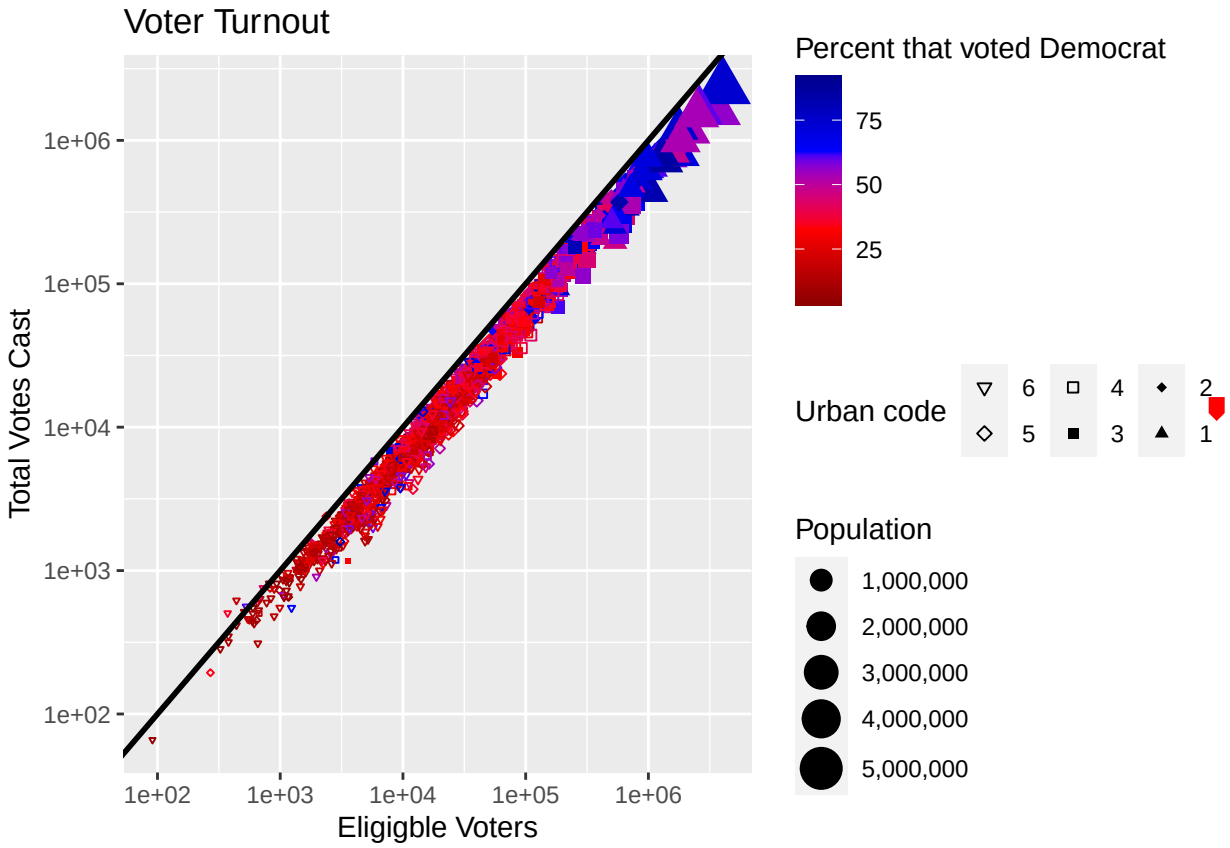


Figure 1: The above graph gives a glimpse at the entire dataset's voter population, specifically in terms of raw voter turnout. The black line represents a 100% turnout, and the shape and scale of the individual points tells us about the size of each county and how urban/rural they are, with a 1 (up-triangle) being most urban and a 6 (hollow down-triangle) being most rural. We can note a degree of inaccuracy with the vote collection because the number of eligible voters (estimated population over 18 years of age) in some of the smallest counties is smaller than the number of total votes cast.

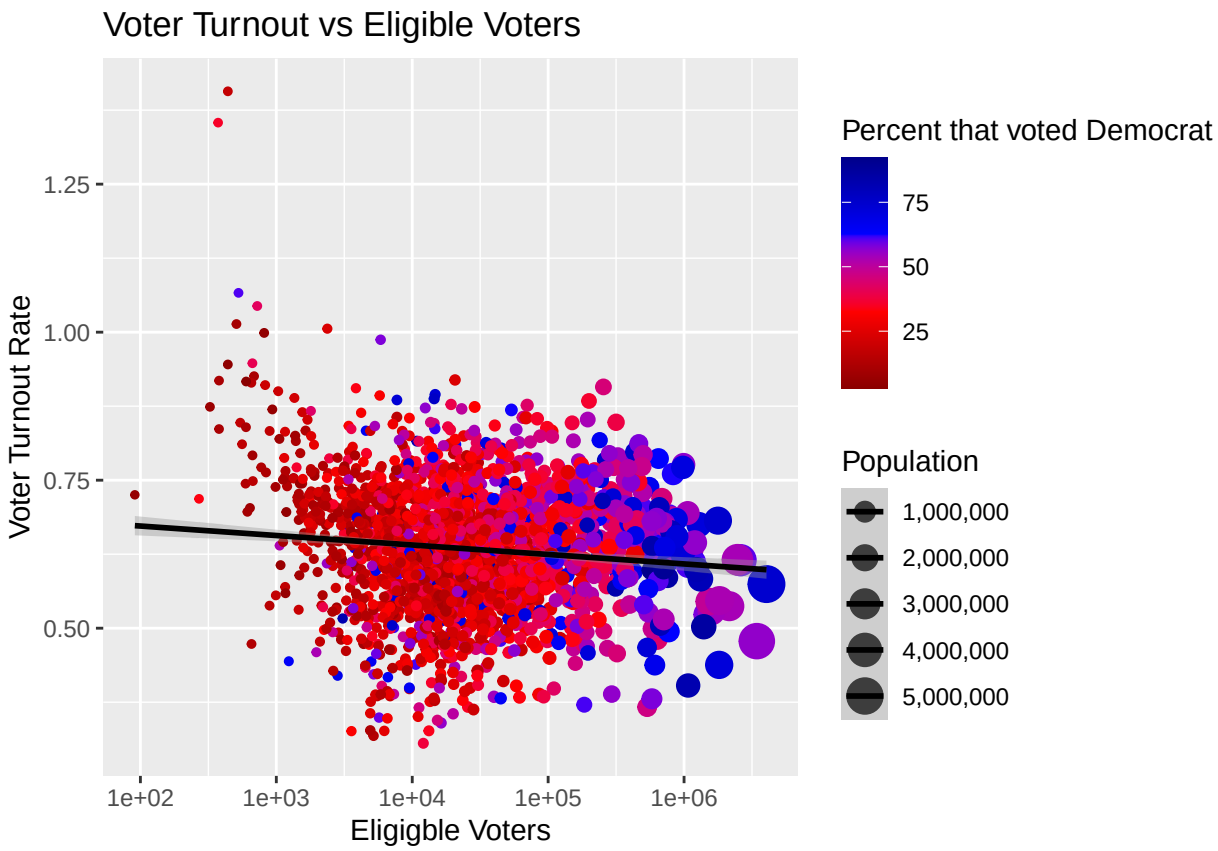


Figure 2: Similar to the initial graph, the above scatterplot gives us a glimpse into the relationship between the eligible voter base for a given county and their voter turnout. Interestingly, we see a slight decline with a fitted regression line, indicating that as the size of a county's voter base increases, the rate of voter turnout decreases slightly. Viewed another way, voter turnout rates are higher in smaller counties.

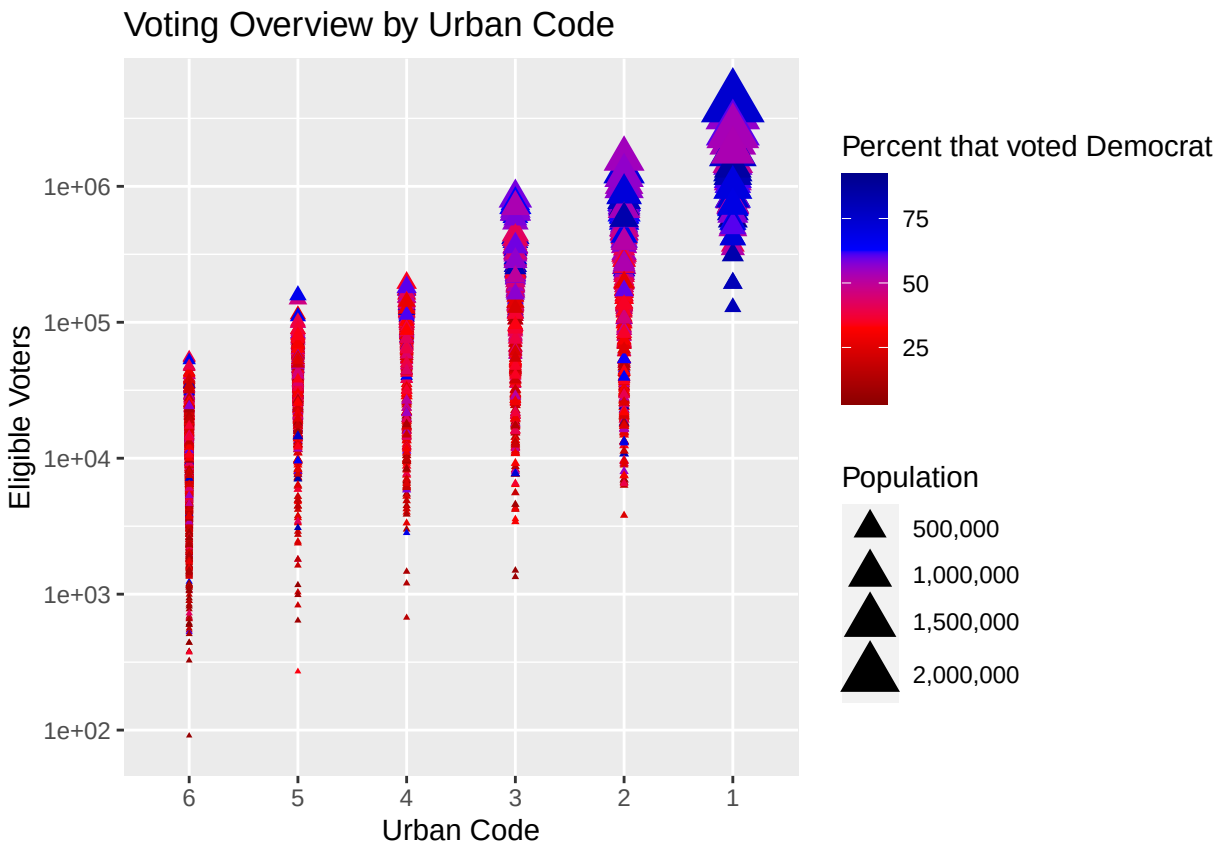


Figure 3: This scatterplot tells us about the tendencies of communities to vote more democratic as their populations increase. The increase in population is also directly correlated to the urban code, which makes sense as urban areas are by nature more densely populated. We can also note that rural areas are not only smaller, but are also more likely to vote republican. It is also interesting to note that regardless of the urban code, as the number of eligible voters increases, so does the percent that voted democrat.

2.2 Correlation Analysis

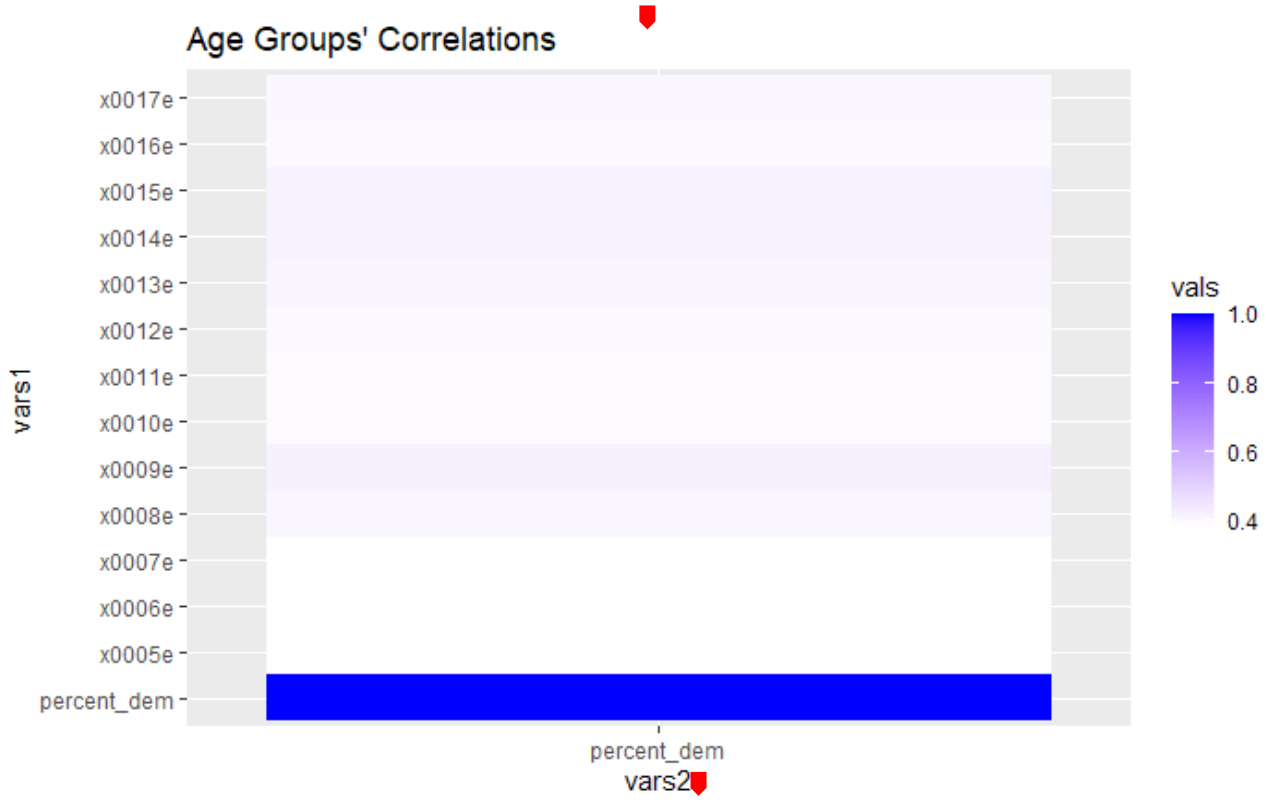


Figure 4: The predictor variables regarding age groups were subsetted. There is no clear correlation between `percent_dem` and any age category.

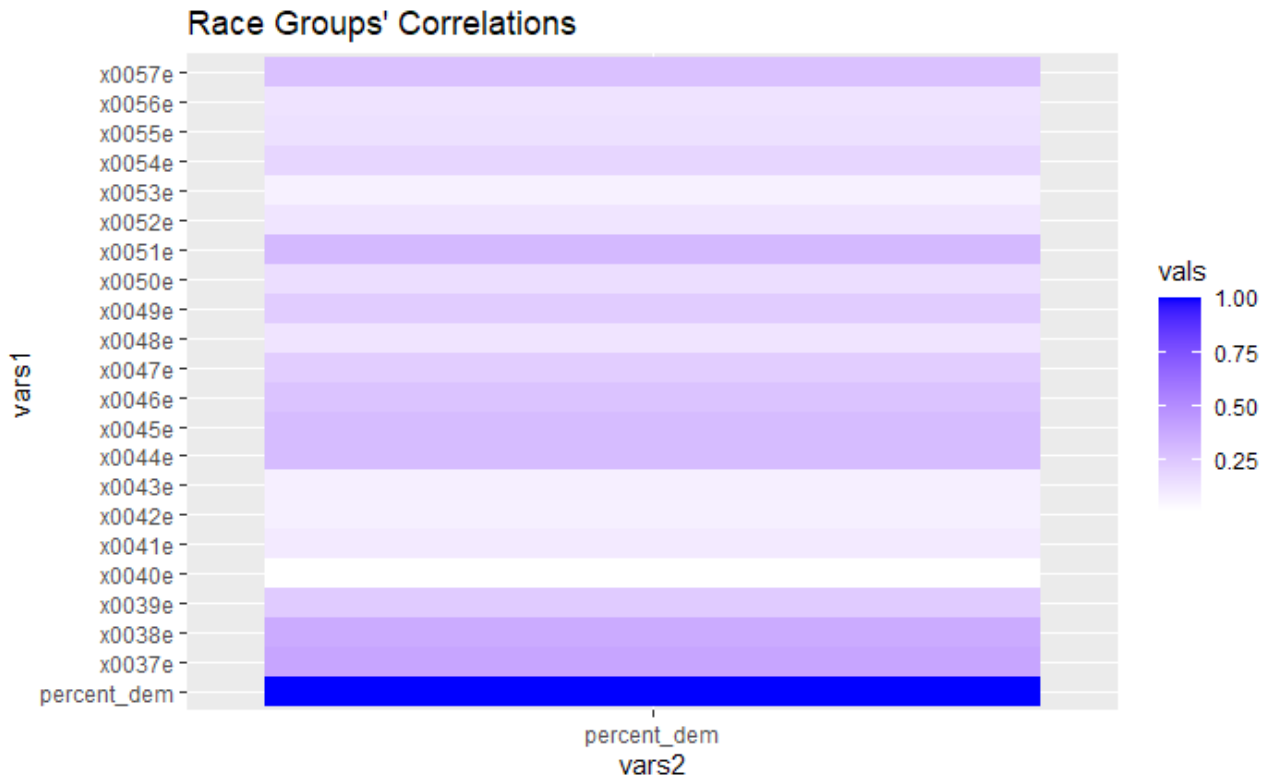


Figure 5: The predictor variables regarding race groups were subsetted. There is some mild correlation between certain races and `percent_dem`. However, nothing is too strong.

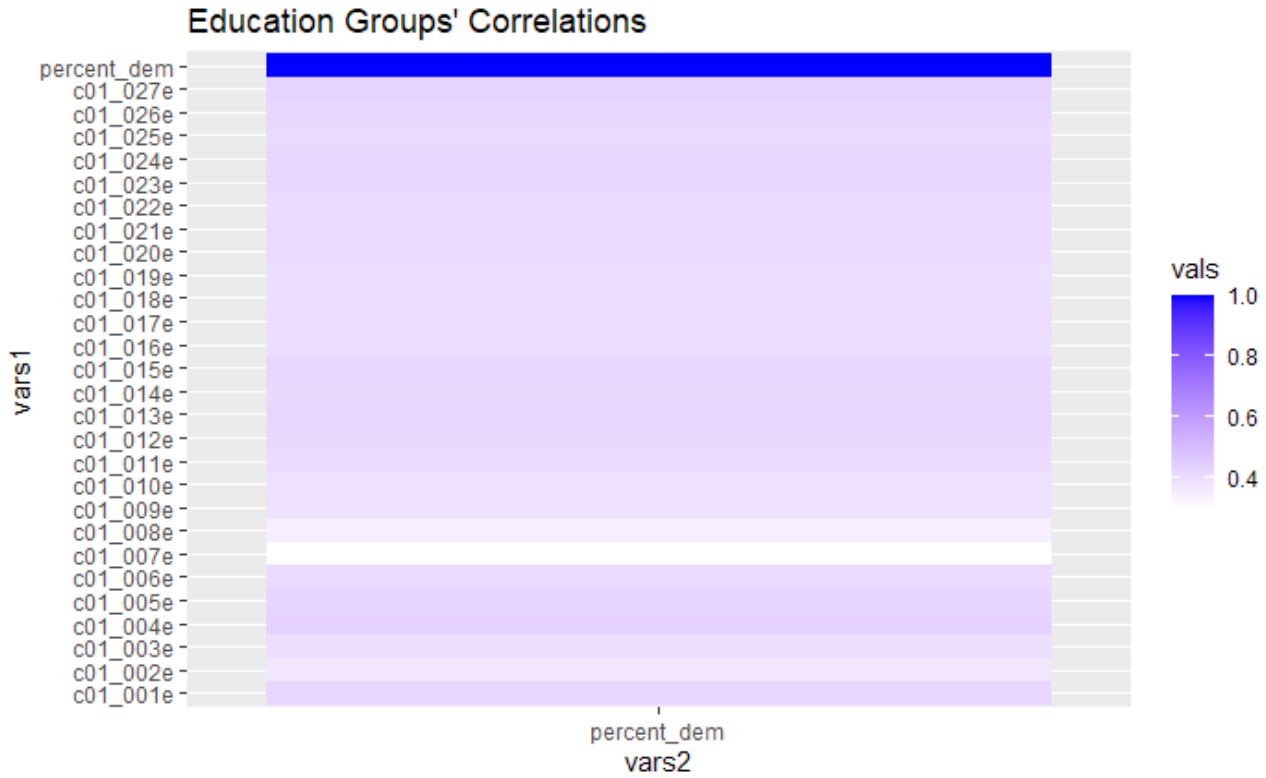


Figure 6: The predictor variables regarding educational level groups were subsetted. There is no strong correlation between `percent_dem` and any educational group.

2.3 Income, Raical, and Educational Analysis

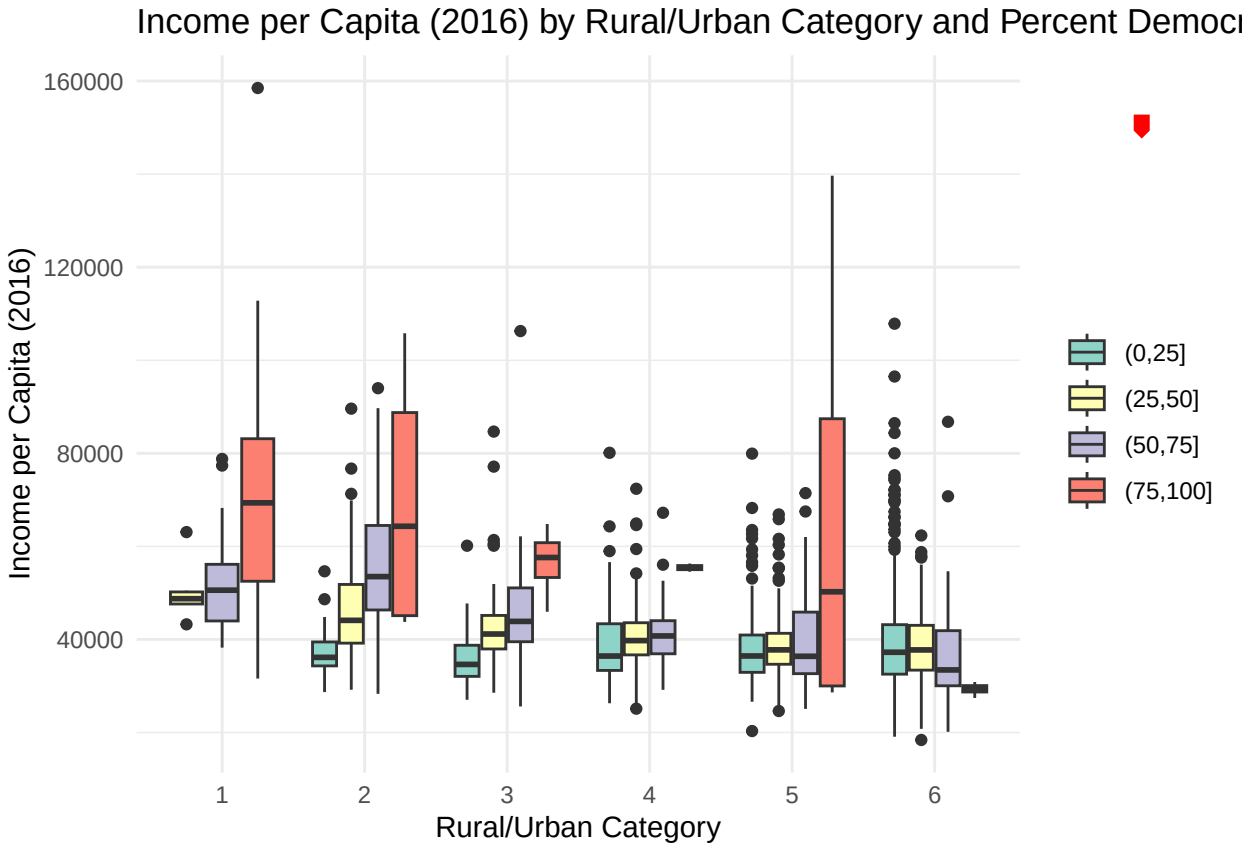


Figure 7: This graph shows the relationship between income, geographical location (whether individuals are located in an urban or rural community), as well as percent of democratic votes. A higher number on the rural/urban category signifies a more rural location. We can conclude based on the above graph that the percentage of democratic voters and income per capacity somewhat slowly decreases, as the rural/urban category number increases.

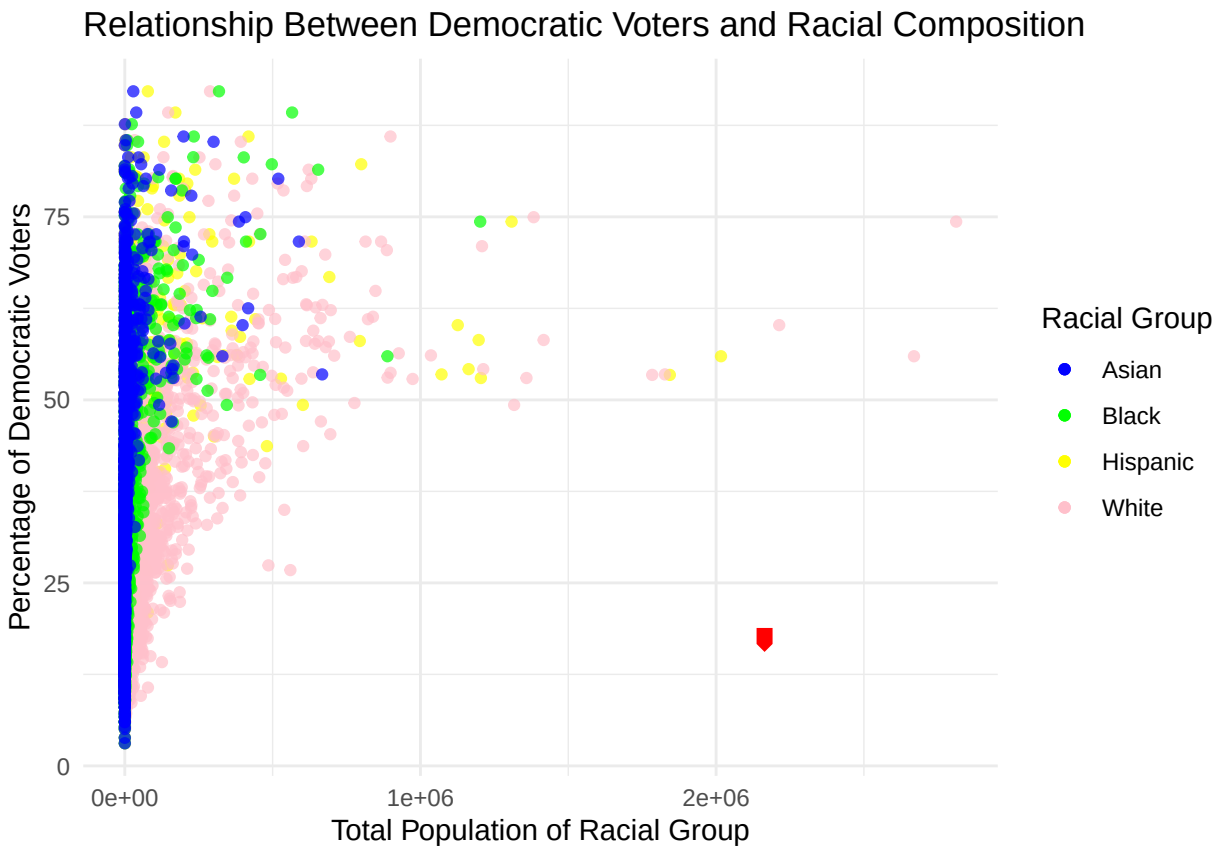


Figure 8: This graph shows the relationship between democratic voters and different racial groups. While there is no significant trend in communities where the total population of a certain racial group is low, communities where there are a higher percentage of a certain racial group are likely to also have a higher percentage of democratic voters.

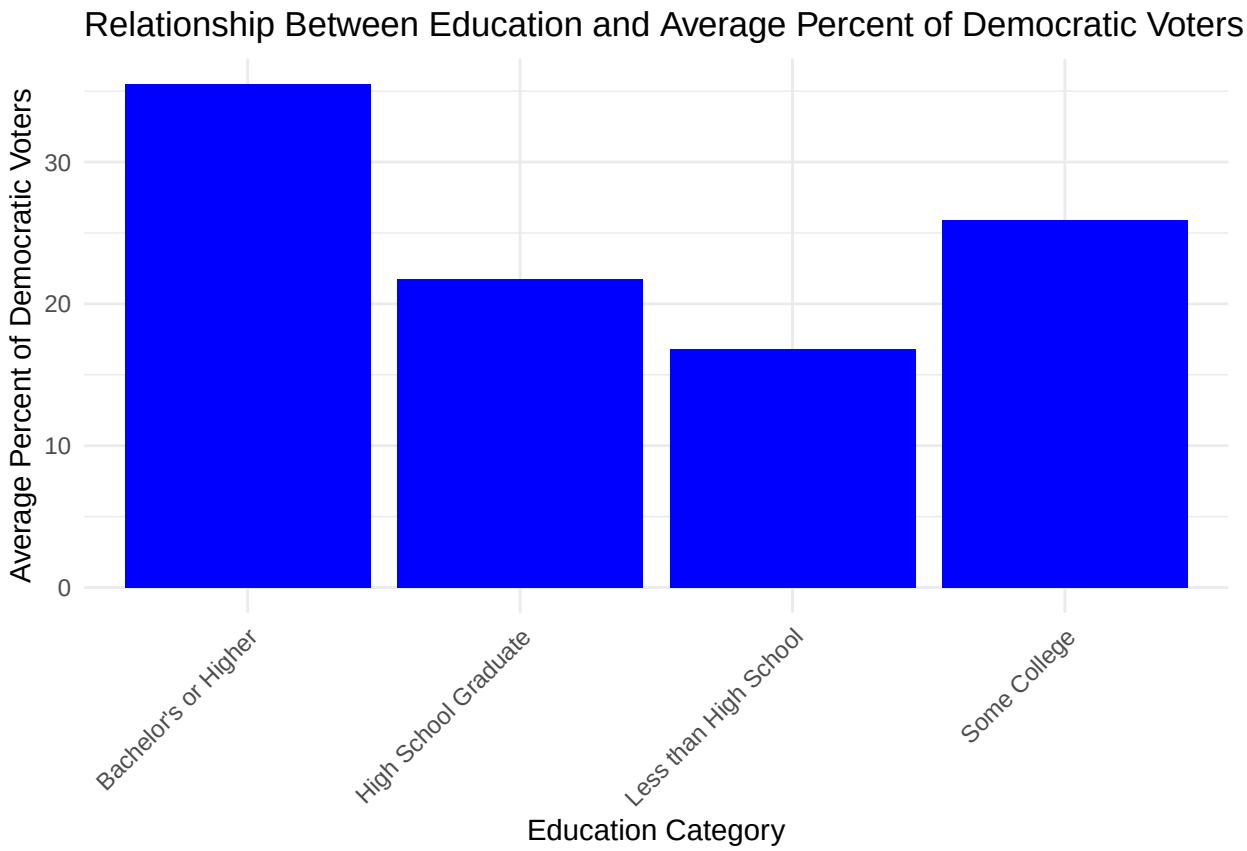


Figure 9: This graph shows the relationship between education and the average percent of democratic voters. We can conclude that those holding a bachelor's degree or higher are the most likely, out of all education groups, to vote for a democratic candidate.

3 Preprocessing and Recipes

3.1 Recipe 1: org_rec

```
org_rec <- recipe(percent_dem ~ ., data = train) %>%
  step_rm(id) %>%
  step_corr(all_numeric_predictors(), threshold = tune()) %>%
  step_mutate_at(all_numeric_predictors(), fn = function(x){x + 1}) %>%
  step_impute_knn(all_numeric_predictors())
```

In this recipe, we first removed the variable "id" from the dataset, which we did not find was necessary for modeling. Furthermore, we calculated the correlation between all the numeric predictors and the response variables to measure the linear relationship between variables and identify highly correlated predictors. Removing highly correlated predictors would increase model accuracy because it removes possible collinearity. We also applied a mutation to all the numeric predictors in this recipe by adding 1 to each numeric predictor. This step was necessary for certain scaling mutations because the scaling operations do not work with 0 values. Lastly, this recipe also imputed missing values in all of the numeric predictors using the KNN imputation method, which helped us replace missing values with values from other similar data points in the dataset.

3.2 Recipe 2: log_norm_org_rec

```
log_norm_org_rec <- org_rec %>%
  step_log(all_numeric_predictors(), base = 10) %>%
  step_normalize(all_numeric_predictors())
```

This recipe first applied a log transformation with base 10 to all of the numeric predictors and then standardizes them. We included this step to stabilize variance and also because we were dealing with variables that were highly skewed right.

3.3 Recipe 3: simple_rec

```
simple_rec <- recipe(percent_dem ~ ., data = train) %>%
  step_mutate(pct_male = x0002e / x0001e * 100) %>%
  # step_mutate(pct_female = x0003e / x0001e * 100) %>%
  step_mutate(pct_under15 = (x0005e + x0006e + x0007e) * 100 / x0001e) %>%
  step_mutate(pct_15thru19 = x0008e * 100 / x0001e) %>%
  step_mutate(pct_20thru24 = x0009e * 100 / x0001e) %>%
  step_mutate(pct_25thru34 = x0010e * 100 / x0001e) %>%
  step_mutate(pct_35thru44 = x0011e * 100 / x0001e) %>%
  step_mutate(pct_45thru54 = x0012e * 100 / x0001e) %>%
  step_mutate(pct_55thru64 = (x0013e + x0014e) * 100 / x0001e) %>%
  step_mutate(pct_65thru84 = (x0015e + x0016e) * 100 / x0001e) %>%
  # step_mutate(pct_over85 = (x0017e) * 100 / x0001e) %>%
  step_rename(median_age = x0018e) %>%
  step_mutate(pct_u18 = x0019e * 100 / x0001e) %>%
  step_mutate(pct_o16 = x0020e * 100 / x0001e) %>%
  # step_mutate(pct_o18 = x0021e * 100 / x0001e) %>%
  # step_mutate(pct_o21 = x0022e * 100 / x0001e) %>%
  step_mutate(pct_o62 = x0023e * 100 / x0001e) %>%
  # step_mutate(pct_o65 = x0024e * 100 / x0001e) %>%
  step_mutate(pct_m_over18 = x0026e * 100 / x0001e) %>%
  # step_mutate(pct_f_over18 = x0027e * 100 / x0001e) %>%
  step_mutate(pct_m_over65 = x0030e * 100 / x0001e) %>%
  # step_mutate(pct_f_over65 = x0031e * 100 / x0001e) %>%
  step_mutate(pct_1race = x0034e * 100 / x0001e) %>%
  # step_mutate(pct_over1race = x0035e * 100 / x0001e) %>%
  step_mutate(pct_white = x0037e * 100 / x0001e) %>%
  step_mutate(pct_black = x0038e * 100 / x0001e) %>%
  # step_mutate(pct_natamer =
  #(x0039e + x0040e + x0041e + x0042e + x0043e) * 100 / x0001e) %>%
  step_mutate(pct_asian = x0044e * 100 / x0001e) %>%
```

```
  step_mutate(pct_indian = x0045e * 100 / x0001e) %>%
  step_mutate(pct_chinese = x0046e * 100 / x0001e) %>%
  step_mutate(pct_filipino = x0047e * 100 / x0001e) %>%
  step_mutate(pct_japan = x0048e * 100 / x0001e) %>%
  step_mutate(pct_korean = x0049e * 100 / x0001e) %>%
  step_mutate(pct_viet = x0050e * 100 / x0001e) %>%
  # step_mutate(pct_asian_other = x0051e * 100 / x0001e) %>%
  # step_mutate(pct_pac_isl = x0052e * 100 / x0001e) %>%
  # step_mutate(pct_nat_haw = x0053e * 100 / x0001e) %>%
  # step_mutate(pct_chamorro = x0054e * 100 / x0001e) %>%
  step_mutate(pct_samoan = x0055e * 100 / x0001e) %>%
  # step_mutate(pct_pacisl_other = x0056e * 100 / x0001e) %>%
  step_mutate(pct_native_haw = (x0052e + x0053e + x0054e + x0055e + x0056e)
              * 100 / x0001e) %>%
  step_mutate(pct_otherrace = x0057e * 100 / x0001e) %>%
  # step_mutate(pct_white_black = x0059e * 100 / x0001e) %>%
  step_mutate(pct_white_native = x0060e * 100 / x0001e) %>%
  step_mutate(pct_white_asian = x0061e * 100 / x0001e) %>%
  # step_mutate(pct_black_native = x0062e * 100 / x0001e) %>%
  step_mutate(pct_whitecombo = x0064e * 100 / x0001e) %>%
  # step_mutate(pct_blackcombo = x0065e * 100 / x0001e) %>%
  # step_mutate(pct_nativecombo = x0066e * 100 / x0001e) %>%
  # step_mutate(pct_asiancombo = x0067e * 100 / x0001e) %>%
  step_mutate(pct_hawcombo = x0068e * 100 / x0001e) %>%
  # step_mutate(pct_othercombo = x0069e * 100 / x0001e) %>%
  step_mutate(pct_hispanic = x0071e * 100 / x0001e) %>%
  step_mutate(pct_mexican = x0072e * 100 / x0001e) %>%
  # step_mutate(pct_puerto = x0073e * 100 / x0001e) %>%
  step_mutate(pct_cuban = x0074e * 100 / x0001e) %>%
  # step_mutate(pct_hisp_other = x0075e * 100 / x0001e) %>%
  # step_mutate(pct_not_hisp = x0076e * 100 / x0001e) %>%
  step_mutate(pct_nhisp_owhite = x0077e * 100 / x0001e) %>%
```

```
  step_mutate(pct_nhisp_oblack = x0078e * 100 / x0001e) %>%
  step_mutate(pct_nhisp_onatamer = x0079e * 100 / x0001e) %>%
  step_mutate(pct_nhisp_oasian = x0080e * 100 / x0001e) %>%
  step_mutate(pct_nhisp_ohawi = x0081e * 100 / x0001e) %>%
  # step_mutate(pct_nhisp_other= x0082e * 100 / x0001e) %>%
  # step_mutate(pct_nhisp_twoother = x0083e * 100 / x0001e) %>%
  step_rename(housing_units = x0086e) %>%
  step_mutate(pct_18t24_nohsdegree = c01_002e * 100 / c01_001e) %>%
  step_mutate(pct_18t24_hsdegree = c01_003e * 100 / c01_001e) %>%
  step_mutate(pct_18t24_somecollege = c01_004e * 100 / c01_001e) %>%
  step_mutate(pct_18t24_bachdegree = c01_005e * 100 / c01_001e) %>%
  step_mutate(pct_o25_nohsdegree = (c01_007e + c01_008e) * 100 / c01_006e) %>%
  step_mutate(pct_o25_hsdegree = c01_009e * 100 / c01_006e) %>%
  step_mutate(pct_o25_somecollege = (c01_010e + c01_011e) * 100 / c01_006e) %>%
  step_mutate(pct_o25_bachdegree = c01_012e * 100 / c01_006e) %>%
  # step_mutate(pct_o25_graddegree = c01_013e * 100 / c01_006e) %>%
  step_mutate(pct_25t34_hsormore = c01_017e * 100 / c01_016e) %>%
  # step_mutate(pct_25t34_bachormore = c01_018e * 100 / c01_016e) %>%
  # step_mutate(pct_35t44_hsormore = c01_020e * 100 / c01_019e) %>%
  step_mutate(pct_35t44_bachormore = c01_021e * 100 / c01_019e) %>%
  step_mutate(pct_45t64_hsormore = c01_023e * 100 / c01_022e) %>%
  step_mutate(pct_45t64_bachormore = c01_024e * 100 / c01_022e) %>%
  step_mutate(pct_o65_hsormore = c01_026e * 100 / c01_025e) %>%
  step_mutate(pct_o65_bachormore = c01_027e * 100 / c01_025e) %>%
  # Removed pct_18t24_bachdegree in impute
  step_impute_knn(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, pct_18t24_nohsdegree,
                  pct_18t24_hsdegree, pct_18t24_somecollege, pct_18t24_bachdegree) %>%
  step_mutate_at(total_votes, median_age, housing_units, 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, fn = function(x){x + 1}) %>%
  step_log(total_votes, median_age, housing_units, 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,
           base = 10) %>%
  step_rm(id, contains("x00"), contains("c01"), gdp_2016, gdp_2017, gdp_2018,
          gdp_2019, pct_18t24_nohsdegree, pct_18t24_somecollege)
```

This is our main recipe which comprises multiple elements, mostly through data preprocessing and feature engineering. We created new variables based on demographic characteristics while renaming some variables, imputed missing values, and applied transformations. The majority of the variables in this recipe were transformed on the basis of population percentages with respect to age demographics, race and education level. Some base 10 log transformations were made in this recipe to some variables, including total_votes, median_age, housing_units, as well as GDP and income per capita variables. All values in this data set are now between 0 and 100 and all missing values were imputed using k-nearest neighbors. Additionally, variables were omitted from this recipe, if the relationship could be entirely explained by another variable or a set of variables. Other variables were removed on the basis of observing p-values in linear regression summaries to help reduce the possibility of overfitting.

4 Models

4.1 Model Evaluation

ID	Type	Engine	Recipe	Hyper-parameters	Metric Score (RMSE) — Cross-Validation	Standard Error
1	`linear`	`lm`	`log_norm_org_rec`	`threshold`	8.50	0.191
2	`linear`	`lm`	`simple_rec`	`X`	7.95	0.133
3	`boosted_tree`	`xgboost`	`simple_rec`	`learn_rate, loss_reduction`	7.34	0.144
4	`rand_forest`	`ranger`	`simple_rec`	`mtry`	7.20	0.116
5	`decision_tree`	`rpart`	`simple_rec`	`cost_complexity`	10.00	0.339
6	`decision_tree`	`rpart`	`log_norm_org_rec`	`cost_complexity, threshold`	10.94	0.139
7	final stack: `Linear, boosted_tree, rand_forest`	`lm, Xgboost, ranger`	`log_norm_org_rec, simple_rec, simple_rec`	`threshold`	6.73	X

4.2 Model Descriptions:

1. The first model utilized is a linear model that adopts the log and normalization transformation of all the numeric predictors, also referred to as `log_norm_org_rec`. The final stack will employ this model.

2. The second model employed is once more a linear model, albeit subject to preprocessing with the `simple_rec` technique, which involves the transformation of all numerical variables linked to population into percentage values. Furthermore, specific variables are strategically omitted in order to mitigate the risk of overfitting, a departure from the approach adopted in the `log_norm_org_rec` model.

3. The third model employed is a boosted tree model that follows the `simple_rec` preprocessing step with meticulously tuned hyperparameters. Specifically, the learning rate has been set to

0.30, and the loss reduction parameter has been fine-tuned to 140. This particular model has been earmarked for inclusion in the final stacking ensemble.

4. The fourth model utilized is a random forest model, implemented following the `simple_rec` preprocessing step, incorporating a tuned hyperparameter for `mtry` that is set to 23. This model will be used in the final stack.

5. The fifth model employed is a decision tree model. It is important to note that a hyperparameter related to cost complexity has been carefully tuned but has not been definitely set. Consequently, this model is not slated for inclusion in the final stack.

6. The sixth model used is a decision tree model that adheres to `log_norm_org_rec` with tuned, but not set hyperparameters for cost complexity and threshold.

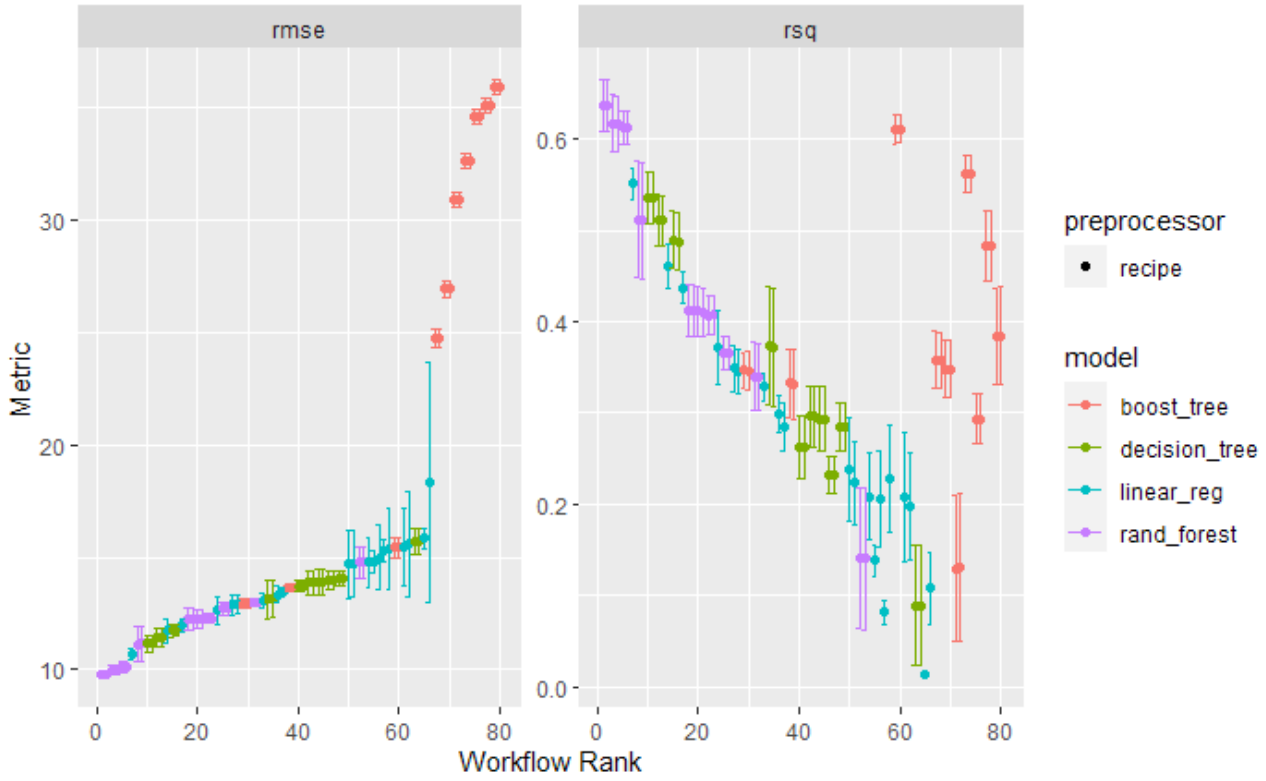
7. The seventh model, which was the highest performing model that we submitted, is composed of a stack or ensemble comprising models 1, 3 and 4. Specifically, it leverages linear regression in conjunction with the `log_norm_org_rec` preprocessing technique, and integrates the boosted tree and random forest models, both employing the `simple_rec` preprocessing approach.

Based on the 5-fold cross-validation stratified by percent_dem RMSE results listed in the table above, the main models that will be focused on are models 1, 3 and 4 or the models used in the final stack – model 7. Model 4 or the random forest model had the lowest RMSE of all basic models. The worst performing models were the decision tree models. It is noteworthy that the `log_norm_org_rec` preprocessing technique was employed for both the random forest and boosted tree models; however, the RMSE outcomes for these models were inferior in comparison to their counterparts employing the `simple_rec` preprocessing method.

Additionally, in comparing `simple_rec` and `log_norm_org_rec`, it was observed that linear models generally exhibited superior predictive performance when utilizing the `simple_rec` approach on testing data. It should be noted that the `log_norm_org_rec` methodology was selected for the linear model within the final ensemble stack due to its significantly improved predictive accuracy, surpassing that of a stack employing a linear model preprocessed with the `simple_rec` technique.

It is important to observe that the lower RMSE results for the ensemble stack model are attributed to the fact that the RMSE reported values for these models are derived from the training data rather than from cross-validation of the training data. This distinction arises from the construction of the model ensembles.

Among the models submitted within the competition's time frame, Model 7, denoted as the final stack, emerged as the highest performer. This outcome aligns logically with its composition as an amalgamation of three foundational models, all characterized by the lowest RMSE values.



This figure above showcases the performance comparisons of the four basic models with tuned, but not fixed hyperparameters for the `recipes of org_rec` and `log_norm_org rec`. Note that the performance for these models shows an RMSE around 10 or higher for all reported models.

wflow_id <chr>	.config <chr>	preproc <chr>	model <chr>	.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>
log_norm_org_rec_linear_reg	Preprocessor01_Model1	recipe	linear_reg	rmse	standard	10.68830	5	0.1627724
log_norm_org_rec_decision_tree	Preprocessor04_Model1	recipe	decision_tree	rmse	standard	11.15615	5	0.2103714
org_rec_decision_tree	Preprocessor04_Model1	recipe	decision_tree	rmse	standard	11.15903	5	0.2101828
log_norm_org_rec_decision_tree	Preprocessor05_Model1	recipe	decision_tree	rmse	standard	11.44316	5	0.2430588
org_rec_decision_tree	Preprocessor05_Model1	recipe	decision_tree	rmse	standard	11.44547	5	0.2430136
log_norm_org_rec_linear_reg	Preprocessor05_Model1	recipe	linear_reg	rmse	standard	11.72469	5	0.3033622
log_norm_org_rec_decision_tree	Preprocessor03_Model1	recipe	decision_tree	rmse	standard	11.72559	5	0.1567967
org_rec_decision_tree	Preprocessor03_Model1	recipe	decision_tree	rmse	standard	11.74031	5	0.1525837
log_norm_org_rec_linear_reg	Preprocessor07_Model1	recipe	linear_reg	rmse	standard	11.96686	5	0.1814592
log_norm_org_rec_linear_reg	Preprocessor06_Model1	recipe	linear_reg	rmse	standard	12.64211	5	0.3820061

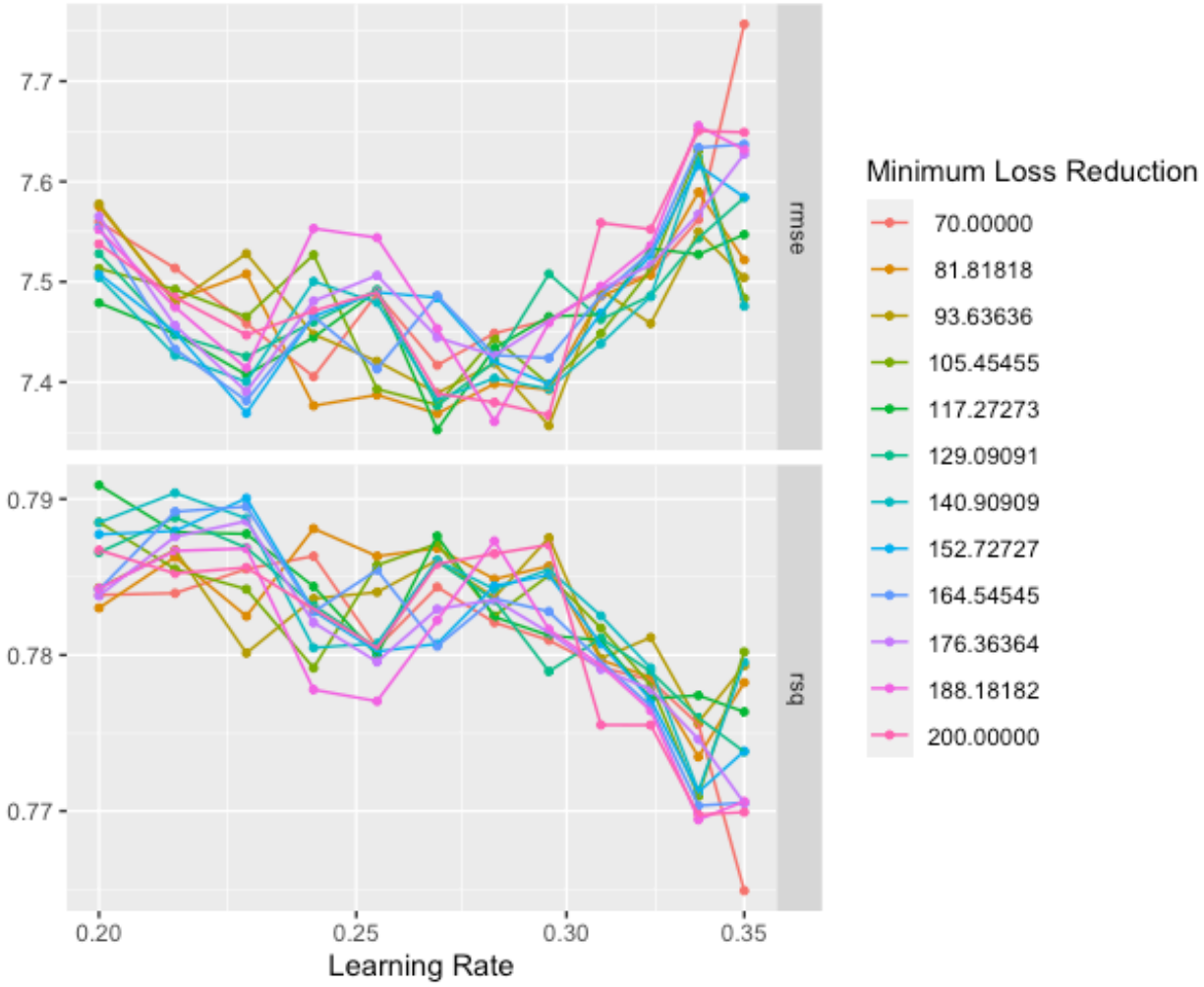
1-10 of 60 rows Previous 1 2 3 4 5 6 Next

This chart provides an overview of metric performance for select models, highlighting the superior performance of `log_norm_org_rec` and linear regression when considering both the `log_norm_org_rec` and 'org_rec' configurations among the models under examination.

4.3 Model Tuning

With regards to the non-stack models that are using simple recipe, the `boosted_tree`, `rand_forest`, `decision_tree` models have tuning parameters. Since the performance of the decision tree in the cross validation was lackluster, only the hyperparameters for `boost_tree` and `rand_forest` were optimized.

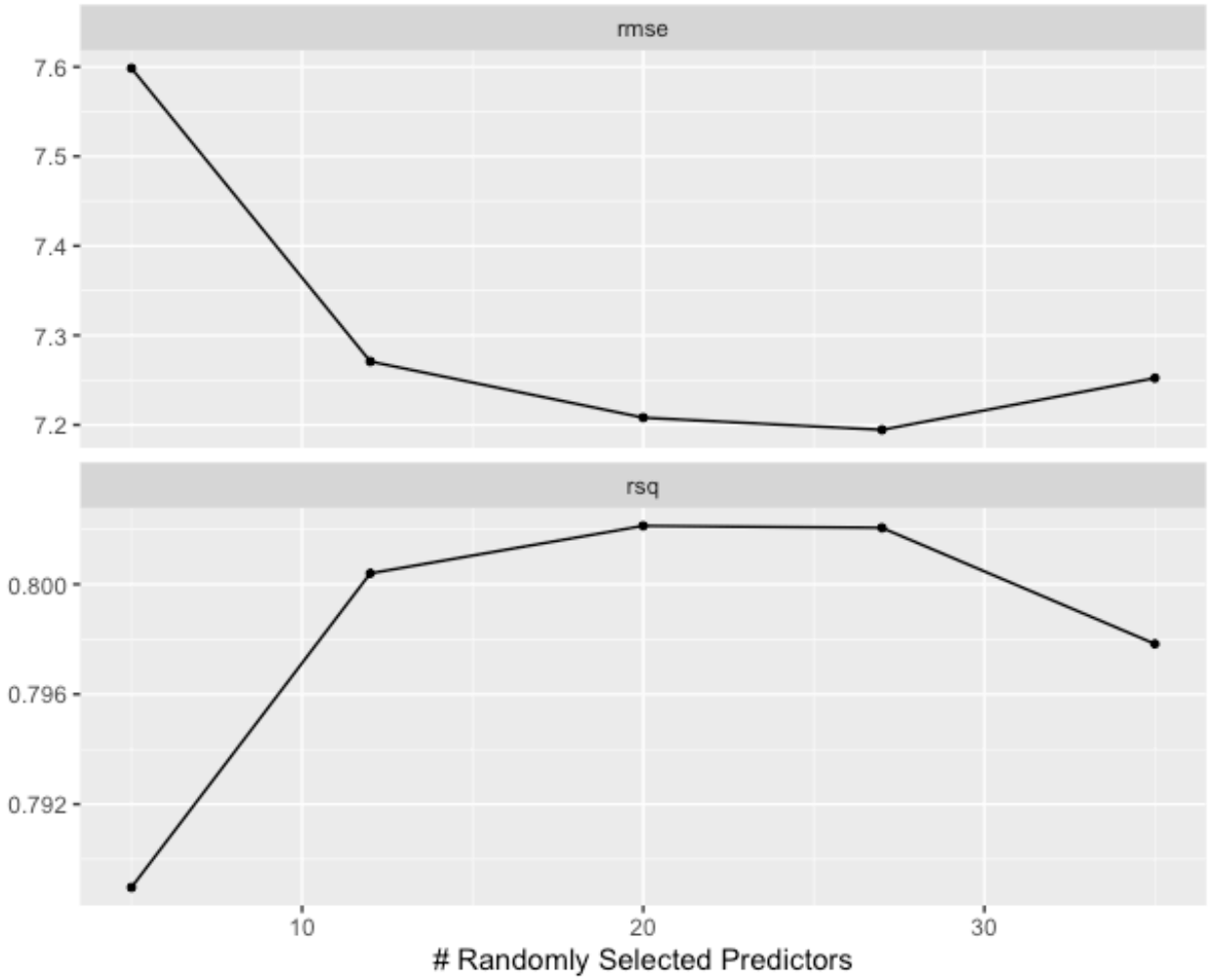
When tuning for the boosted tree model, the tuning grid performance for the parameters were as follow:



As evident from the analysis, the optional tuning parameter for the learning rate demonstrates optimal performance at approximately 0.30, as RMSE starts to increase with larger rates. Additionally, with regards to loss reduction, the parameter is relatively independent of influencing the RMSE of the model. As a result, a learning rate of 0.30 was selected for the boosted tree model within the stack used for the submission.

For random forest, the parameter chosen from the optimization for simple recipe for `mtry` is 23. The

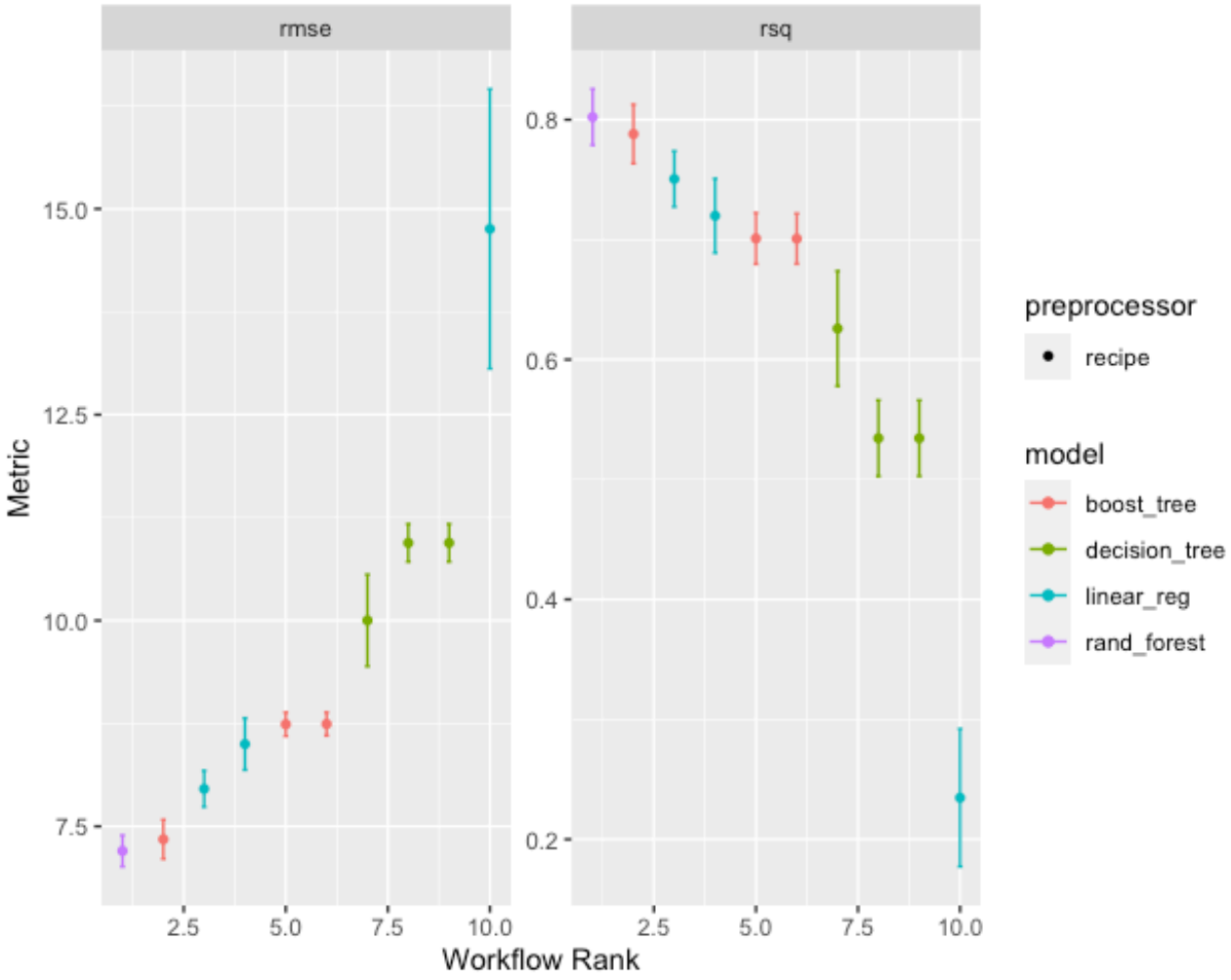
RMSE can be shown to improve with parameter value set to 23 over other values when tuning for an optimal parameter in the following chart:



}

For the threshold hyperparameter of `log_norm_org_rec`, a tuning grid is used in the stack itself, but no particular value of the parameter is set or optimized prior to usage in the stack.

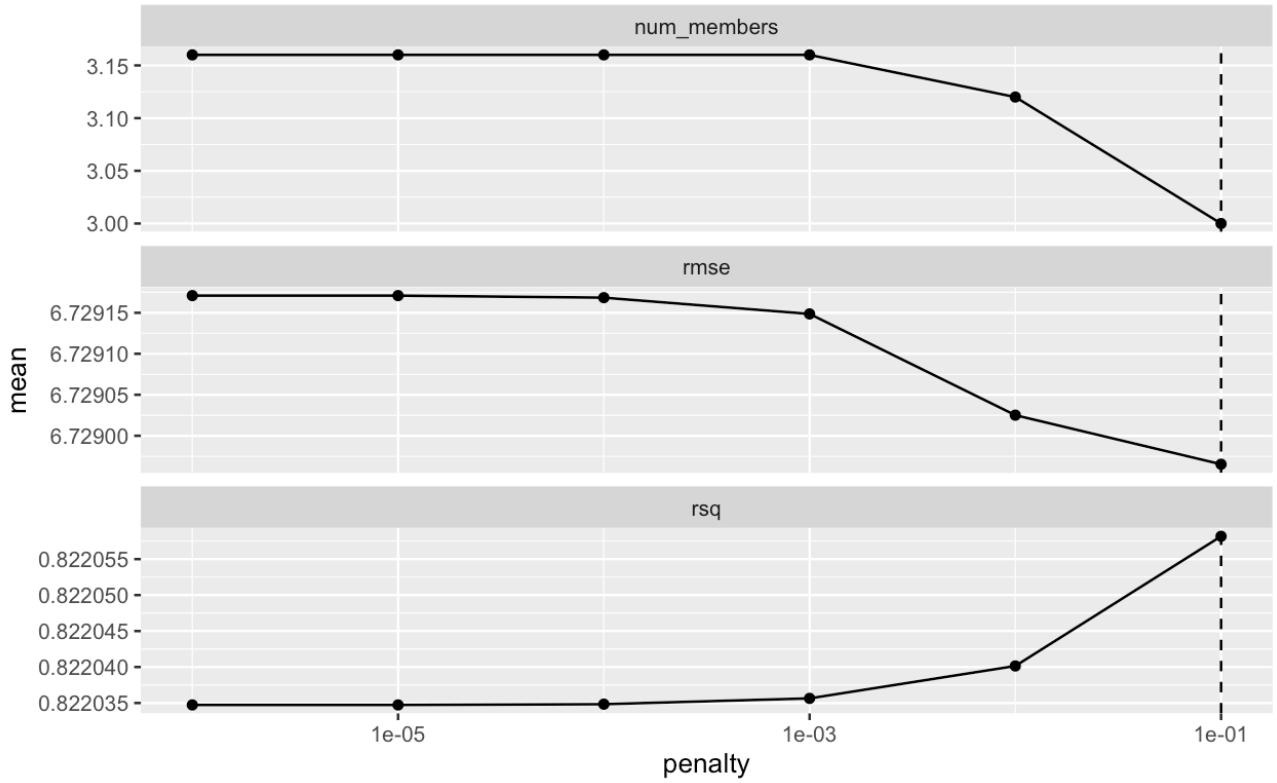
Upon tuning the boosted tree and random forest to specific values, the resulting RMSE plot exhibits the pattern depicted below:



The plot above shows the models reported with the reported 3 possible recipes: `org_rec`, `log_norm_org_rec`, and `simple_rec`. It becomes apparent from this plot that the utilization of a simple recipe, coupled with the optimization of hyperparameters, results in a noteworthy reduction in RMSE cross-validation scores.

The RMSE values depicted in the plot correspond to the values detailed in the table within this section. Furthermore, this graphical representation underscores that the performance of the boosted tree and random forest models markedly outperformed the other models in consideration.

Utilizing the final stack with the set optimized tuned hyperparameters for the candidate models, resulted in a higher predictive performance compared to that achieved by the individual models comprising the stack.



The graph displayed above illustrates the RMSE values associated with the final stack model. These RMSE values are from cross validation values internal to the model itself. These RMSE values are lower than the reported cross validation RMSE values of any individual model alone.

5 Results

5.1 Selection of Final Model

Initially, we considered models including Linear Regression, Random Forest, Boosted Tree, and ensemble using recipes like `simple_rec` and `log_norm_org_rec`. Ultimately, we decided to use a Stack Ensemble model since it aggregates predictions from various base models, which helped us enhance our predictive performance and resilience. We found that this stack method, which uses fine-tuning of hyperparameters for models like Boosted Tree and Random Forest, gave us the strongest predictive performance using `simple_rec` for those specific models. When it came to deciding the preprocess procedures for the linear model, it was apparent that use of the `log_norm_org_rec` gave more accurate predictions than the `simple_rec` despite having a higher RMSE score in the cross validation model report.

5.2 Issues and Weaknesses Final model

Some issues of this model include problems with the random forest model's handling of missing data. Ideally, we would've removed the observations with missing values to help improve predictive performance, but this method did not work with random forest. While using k-nearest neighbor imputation methods was fruitful, more research into robust imputation approaches is required to reduce the impact of missing data.

Furthermore, feature engineering could be improved. The informativeness of predictor variables can significantly impact a model's performance. Determining which predictors to include and how to transform certain predictors is likely to increase model predictive performance and reduce error.

One challenge faced, especially in working with our random forest and boosted tree model was correcting for overfitting. The utilization of linear models for `simple_rec` in was exactly to minimize the likelihood of overfitting while also removing predictors with the largest p-values. The random forest and boosted tree models would then be tested with the removed predictors to analyze their RMSE. This cycle of removing predictors and creating linear models would repeat until the lowest RMSE was found for the random forest and boosted tree models. This isn't necessarily the best way to minimize overfitting and other methods should be researched into in the future.

5.3 Future Steps and Final Reflections

Future potential possibilities that the team came across were to include more complex models, such as a highly optimized neural network. By including more powerful models, the RMSE could have gone lower during the stages of cross-validation. One possible model was to combine the reliability of the linear regression model, the accuracy of the boosted tree model, and the in-depth computational power of the neural network in one stacked model. By doing this, it was inferred that the resulting RMSE could be much lower than previous prediction submissions. Additionally, a more rigorous tuning grid for the parameters for the boosted tree could have been used, as well as tuning for epoch cycles and hidden units in a potential neural network. Moreover, an engine for the neural network could have been used to fully extend the potential of the model, such as the brulee library.

6 Appendix

6.1 Final Annotated Script

```
### NOTE: our results were generated with a seed from a Mac,
### so they may vary on Windows machines.

### Loading necessary libraries
library(tidyverse)
library(tidymodels)
library(stacks)
library(ranger)
library(xgboost)

### Reading necessary data
trainFilepath <- paste0(getwd(),"/train.csv")
testFilepath <- paste0(getwd(),"/test.csv")

train <- read.csv(trainFilepath)
test <- read.csv(testFilepath)

## Remove name from training data
train$name <- NULL

### Creating a cross-validation training split for the stack
# stratified by percent_dem
# Seed for reproducibility
set.seed(101)
train_split <- vfold_cv(train, v = 5, strata = percent_dem)

### Creating necessary recipes
## Original Recipe -- Takes all the raw entries in the training data,
## filters with tuning parameter to remove highly correlated variables
```

```
# and then shifts all predictors by one value to remove 0 entries in data.
# Imputes missing data using knn.
org_rec <- recipe(percent_dem ~ ., data = train) %>%
  step_rm(id) %>%
  step_corr(all_numeric_predictors(), threshold = tune()) %>%
  step_mutate_at(all_numeric_predictors(), fn = function(x){x + 1}) %>%
  step_impute_knn(all_numeric_predictors())

# This recipe takes the log and normalizes all the predictors in
# original recipe. This recipe will be used in our stack for
# the linear model regression/workflow.
log_norm_org_rec <- org_rec %>%
  step_log(all_numeric_predictors(), base = 10) %>%
  step_normalize(all_numeric_predictors())

## Simple Recipe -- This recipe uses a population percentage based
# transformation for predictors revolving around age, ethnicity and education.
# Like in the original recipe, missing values are imputed using knn, numeric
# predictors dealing with income and gdp are shifted up by 1 and have the log
# taken of to transform the values since these predictors are heavily
# right-skewed. Raw data predictors are removed, some transformed predictors
# are removed as indicated by being commented out, and some gdp and education
# predictors are removed at the end. This recipe will be used randomforest and
# boosted tree models in the stack.
simple_rec <- recipe(percent_dem ~ ., data = train) %>%
  step_mutate(pct_male = x0002e / x0001e * 100) %>%
  # step_mutate(pct_female = x0003e / x0001e * 100) %>%
  step_mutate(pct_under15 = (x0005e + x0006e + x0007e) * 100 / x0001e) %>%
  step_mutate(pct_15thru19 = x0008e * 100 / x0001e) %>%
  step_mutate(pct_20thru24 = x0009e * 100 / x0001e) %>%
  step_mutate(pct_25thru34 = x0010e * 100 / x0001e) %>%
  step_mutate(pct_35thru44 = x0011e * 100 / x0001e) %>%
```

```
  step_mutate(pct_45thru54 = x0012e * 100 / x0001e) %>%
  step_mutate(pct_55thru64 = (x0013e + x0014e) * 100 / x0001e) %>%
  step_mutate(pct_65thru84 = (x0015e + x0016e) * 100 / x0001e) %>%
  # step_mutate(pct_over85 = (x0017e) * 100 / x0001e) %>%
  step_rename(median_age = x0018e) %>%
  step_mutate(pct_u18 = x0019e * 100 / x0001e) %>%
  step_mutate(pct_o16 = x0020e * 100 / x0001e) %>%
  # step_mutate(pct_o18 = x0021e * 100 / x0001e) %>%
  # step_mutate(pct_o21 = x0022e * 100 / x0001e) %>%
  step_mutate(pct_o62 = x0023e * 100 / x0001e) %>%
  # step_mutate(pct_o65 = x0024e * 100 / x0001e) %>%
  step_mutate(pct_m_over18 = x0026e * 100 / x0001e) %>%
  # step_mutate(pct_f_over18 = x0027e * 100 / x0001e) %>%
  step_mutate(pct_m_over65 = x0030e * 100 / x0001e) %>%
  # step_mutate(pct_f_over65 = x0031e * 100 / x0001e) %>%
  step_mutate(pct_1race = x0034e * 100 / x0001e) %>%
  # step_mutate(pct_over1race = x0035e * 100 / x0001e) %>%
  step_mutate(pct_white = x0037e * 100 / x0001e) %>%
  step_mutate(pct_black = x0038e * 100 / x0001e) %>%
  # step_mutate(pct_natamer =
  #       (x0039e + x0040e + x0041e + x0042e + x0043e) * 100 / x0001e) %>%
  step_mutate(pct_asian = x0044e * 100 / x0001e) %>%
  step_mutate(pct_indian = x0045e * 100 / x0001e) %>%
  step_mutate(pct_chinese = x0046e * 100 / x0001e) %>%
  step_mutate(pct_filipino = x0047e * 100 / x0001e) %>%
  step_mutate(pct_japan = x0048e * 100 / x0001e) %>%
  step_mutate(pct_korean = x0049e * 100 / x0001e) %>%
  step_mutate(pct_viet = x0050e * 100 / x0001e) %>%
  # step_mutate(pct_asian_other = x0051e * 100 / x0001e) %>%
  # step_mutate(pct_pac_isl = x0052e * 100 / x0001e) %>%
  # step_mutate(pct_nat_haw = x0053e * 100 / x0001e) %>%
  # step_mutate(pct_chamorro = x0054e * 100 / x0001e) %>%
```

```
  step_mutate(pct_samoan = x0055e * 100 / x0001e) %>%
  # step_mutate(pct_pacisl_other = x0056e * 100 / x0001e) %>%
  step_mutate(pct_native_haw =
                (x0052e + x0053e + x0054e + x0055e + x0056e) * 100 / x0001e) %>%
  step_mutate(pct_otherrace = x0057e * 100 / x0001e) %>%
  # step_mutate(pct_white_black = x0059e * 100 / x0001e) %>%
  step_mutate(pct_white_native = x0060e * 100 / x0001e) %>%
  step_mutate(pct_white_asian = x0061e * 100 / x0001e) %>%
  # step_mutate(pct_black_native = x0062e * 100 / x0001e) %>%
  step_mutate(pct_whitecombo = x0064e * 100 / x0001e) %>%
  # step_mutate(pct_blackcombo = x0065e * 100 / x0001e) %>%
  # step_mutate(pct_nativecombo = x0066e * 100 / x0001e) %>%
  # step_mutate(pct_asiancombo = x0067e * 100 / x0001e) %>%
  step_mutate(pct_hawcombo = x0068e * 100 / x0001e) %>%
  # step_mutate(pct_othercombo = x0069e * 100 / x0001e) %>%
  step_mutate(pct_hispanic = x0071e * 100 / x0001e) %>%
  step_mutate(pct_mexican = x0072e * 100 / x0001e) %>%
  # step_mutate(pct_puerto = x0073e * 100 / x0001e) %>%
  step_mutate(pct_cuban = x0074e * 100 / x0001e) %>%
  # step_mutate(pct_hisp_other = x0075e * 100 / x0001e) %>%
  # step_mutate(pct_not_hisp = x0076e * 100 / x0001e) %>%
  step_mutate(pct_nhisp_owhite = x0077e * 100 / x0001e) %>%
  step_mutate(pct_nhisp_oblack = x0078e * 100 / x0001e) %>%
  step_mutate(pct_nhisp_onatamer = x0079e * 100 / x0001e) %>%
  step_mutate(pct_nhisp_oasian = x0080e * 100 / x0001e) %>%
  step_mutate(pct_nhisp_ohawi = x0081e * 100 / x0001e) %>%
  # step_mutate(pct_nhisp_other= x0082e * 100 / x0001e) %>%
  # step_mutate(pct_nhisp_twoother = x0083e * 100 / x0001e) %>%
  step_rename(housing_units = x0086e) %>%
  step_mutate(pct_18t24_nohsdegree = c01_002e * 100 / c01_001e) %>%
  step_mutate(pct_18t24_hsdegree = c01_003e * 100 / c01_001e) %>%
  step_mutate(pct_18t24_somecollege = c01_004e * 100 / c01_001e) %>%
```

```
  step_mutate(pct_18t24_bachdegree = c01_005e * 100 / c01_001e) %>%
  step_mutate(pct_o25_nohsdegree = (c01_007e + c01_008e) * 100 / c01_006e) %>%
  step_mutate(pct_o25_hsdegree = c01_009e * 100 / c01_006e) %>%
  step_mutate(pct_o25_somecollege = (c01_010e + c01_011e) * 100 / c01_006e) %>%
  step_mutate(pct_o25_bachdegree = c01_012e * 100 / c01_006e) %>%
  # step_mutate(pct_o25_graddegree = c01_013e * 100 / c01_006e) %>%
  step_mutate(pct_25t34_hsormore = c01_017e * 100 / c01_016e) %>%
  # step_mutate(pct_25t34_bachormore = c01_018e * 100 / c01_016e) %>%
  # step_mutate(pct_35t44_hsormore = c01_020e * 100 / c01_019e) %>%
  step_mutate(pct_35t44_bachormore = c01_021e * 100 / c01_019e) %>%
  step_mutate(pct_45t64_hsormore = c01_023e * 100 / c01_022e) %>%
  step_mutate(pct_45t64_bachormore = c01_024e * 100 / c01_022e) %>%
  step_mutate(pct_o65_hsormore = c01_026e * 100 / c01_025e) %>%
  step_mutate(pct_o65_bachormore = c01_027e * 100 / c01_025e) %>%
  # Removed pct_18t24_bachdegree in impute
  step_impute_knn(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, pct_18t24_nohsdegree,
                  pct_18t24_hsdegree, pct_18t24_somecollege,
                  pct_18t24_bachdegree) %>%
  step_mutate_at(total_votes, median_age, housing_units, 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, fn = function(x){x + 1}) %>%
  step_log(total_votes, median_age, housing_units, 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, base = 10) %>%
  step_rm(id, contains("x00"), contains("c01"), gdp_2016, gdp_2017, gdp_2018,
          gdp_2019, pct_18t24_nohsdegree, pct_18t24_somecollege)
```

```r
## Metrics for stack
wf_metrics <- metric_set(rmse)

### Stack Ensemble: Linear Regression (log_norm_org_recipe),
### Boosted Tree(simple_rec), Random Forest(simple_rec) -- Best Performer
set.seed(10)
ctrl_grid <- control_stack_grid()
ctrl_res <- control_stack_resamples()

## Linear Model -- Uses a tuning grid of 5 to determine threshold to remove
## correlated variables
linear_model_stk <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")

linear_reg_wf <- workflow() %>%
  add_model(linear_model_stk) %>%
  add_recipe(log_norm_org_rec)

linear_reg_res <- linear_reg_wf %>%
  tune_grid(
    resamples = train_split,
    metrics = wf_metrics,
    grid = 5,
    control = ctrl_grid
  )



## Random Forest -- Uses parameter of mtry = 22, the value for this parameter
#was determined by optimizing a tuning grid for rmse.
# The model performs cross validation to become part of the stack.
r_forest_model_stk <- rand_forest(mtry = 22) %>%
```

```
  set_engine("ranger") %>%
  set_mode("regression")

r_forest_model_wf <- workflow() %>%
  add_model(r_forest_model_stk) %>%
  add_recipe(simple_rec)

r_forest_reg_res <- r_forest_model_wf %>%
  fit_resamples(
    resamples = train_split,
    metrics = wf_metrics,
    control = ctrl_res
  )

##  Boosted Tree -- Parameters of learn_rate and loss_reduction were determined
# by optimizing a tuning grid. Learn_rate was significant in finding a Boosted
# Tree model with low mrse, but loss_reduction was less important.
# This model performs cross validation as part of its stack.
boost_tree_model_stk <- boost_tree(learn_rate = .301, loss_reduction = 140) %>%
  set_engine("xgboost") %>%
  set_mode("regression")

boost_tree_model_wf <- workflow() %>%
  add_model(boost_tree_model_stk) %>%
  add_recipe(simple_rec)

boost_tree_reg_res <- boost_tree_model_wf %>%
  fit_resamples(
    resamples = train_split,
    metrics = wf_metrics,
    control = ctrl_res
  )
```

```r
## Candidates are added to the stack
election_data_stack <- stacks() %>%
  add_candidates(linear_reg_res) %>%
  add_candidates(r_forest_reg_res) %>%
  add_candidates(boost_tree_reg_res)

## Model is blended to add weights to each model(7 in total)
election_model_stack <- election_data_stack %>%
  blend_predictions()

## Training data is fitted to the stack
election_model_stack <- election_model_stack %>%
  fit_members()

## Predictions are generated for the test data
election_model_test_preds <-
  election_model_stack %>% predict(new_data = test)

## ID is column bound with predictions
test_preds_stack <- test %>%
  select(id) %>%
  bind_cols(election_model_test_preds) %>%
  rename(percent_dem = .pred)

head(test_preds_stack, n=10)
```

6.2 Team Member Contributions

- *Mayerli Cordero-Cortes*: Data Evaluation, Research Write-Up, Final Drafting of Report
- *Eric Gallardo*: Model Construction, Model Tuning, Data Visualization, Research Write-Up, and Recipe Creation
- *Avanthika Panchapakesan*: Data Evaluation, Data Visualization, Background Research,

Research Write-Up

- *Shayan Saadat*: Data Evaluation, Data Visualization, Research Write-Up, Final Drafting of Report
- *Luke Villanueva*: Data Evaluation, Data Visualization, Recipe Creation and Model Construction, Research Write-Up

References

2020. National Low Income Housing Coalition. August 2020. https://nlihc.org/resource/democratic-party-and-republican-party-platforms-address-affordable-housing.

Bhattacharya, Ananya. 2022. "American Billionaires' Political Spending Overwhelmingly Leans Republican." *Quartz*, November. https://qz.com/american-billionaires-political-spending-overwhelmingl-1849751449.

Doherty, Carroll. 2020. "In Changing u.s. Electorate, Race and Education Remain Stark Dividing Lines." Pew Research Center - U.S. Politics &Amp; Policy. Pew Research Center. June 2020. https://www.pewresearch.org/politics/2020/06/02/in-changing-u-s-electorate-race-and-education-remain-stark-dividing-lines/.