2020 US Presidential Election Voter Prediction

Tiffany Hu

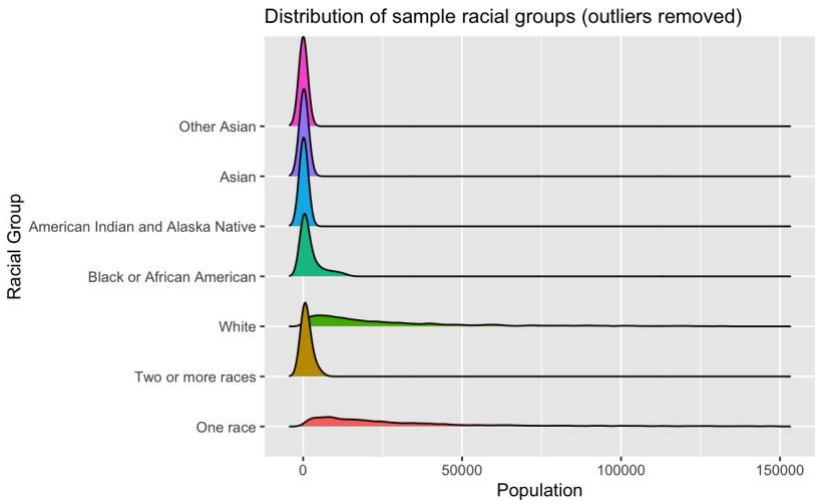
Table of contents

Introduction

The predictor variables given in the dataset are mainly about the demographic and educational background of the voters. Through information obtained from external sources, I found that all major ethnic groups lean to Democratic, except White; meaning that groups like Black, Hispanic, and Asian are more likely to vote for Democratic than Republican within a specific state (Pew Research Center, 2020). Education attainment also plays a part in determining the vote result. It is found from Current Population Survey data that individuals with a college degree or higher are about 50% more likely to vote as compared to those without (U.S. Bureau of the Census, 2020c). Moreover, the widening ideological divide between the group of voters with an education level of college or higher and those with less education indicates that the former lean towards democratic party while the latter prefers republican (Suls, 2016). The third factor that appears to be related to the U.S. presidential election result is the age of the voters. Over 70% of the individuals of ages under 30 supported Democratic candidates in 2018; the share of voters who preferred Republican increases as their ages increase (Hartig et al., 2023). Hence, it is very likely that the variables about ethnicity, education level, and age in the dataset are strongly associated with the percentage of voters in a county that voted for Biden in the 2020 US Presidential Election.

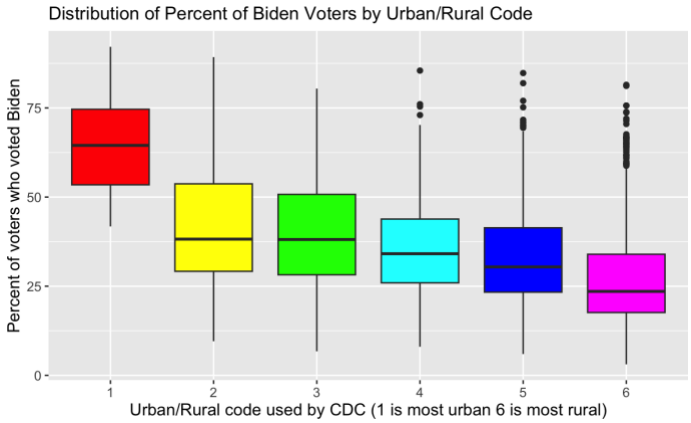
Exploratory Data Analysis

Visualization 1: Distribution of racial groups



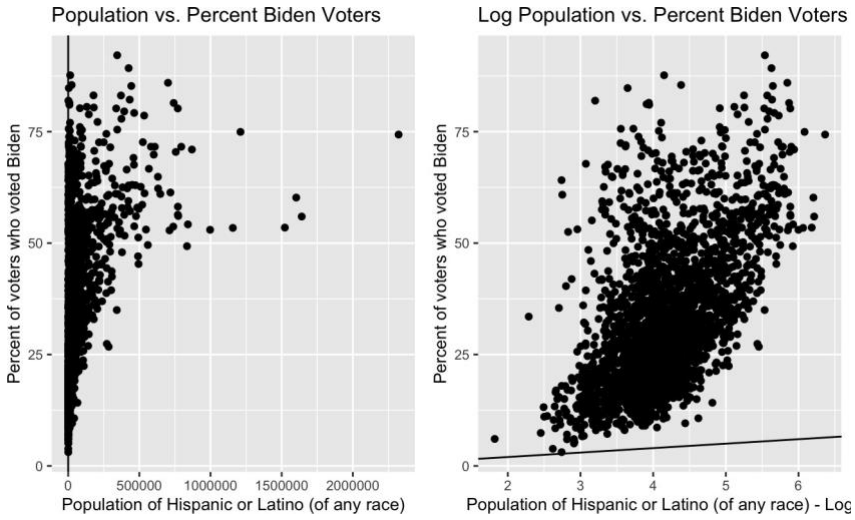
Based on background research, I took a deeper look into specific demographic groups. Visualization 1 displays the population distribution of racial groups in the dataset. Although this visual does not encompass any direct relationship to the outcome variable “Percent voters who voted Biden”, it shows a clear skew in all groups except “White” and “One race” across counties. Since skewness can have influence in model performance, we may need to transform heavily skewed columns in the preprocessing steps. Note: this distribution was made with extreme outliers removed from all columns included.

Visualization 2: Distribution of Biden voters by urban/rural codes



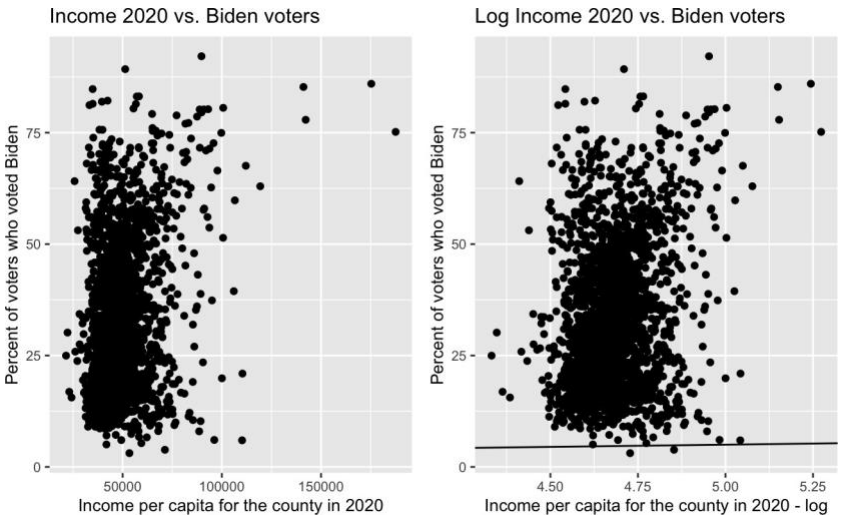
The second visualization uses side-by-side boxplots to compare the distribution of voters who voted for Biden by urban/rural code. The dataset includes six numbered urban/rural codes labeled by the CDC for each county, where 1 represents “most urban” and 6 represents “most rural”. Based on this visualization, we see that there is more skewness amongst voters as areas become more rural. Another important piece of information that this visualization encapsulates is the high percentage of Biden voters in the “most urban” counties. Overall, this visualization helps us to note that the urban/rural codes will most likely play an important role in predicting voter choices.

Visualization 3: Population (sample racial group) vs. Biden voters



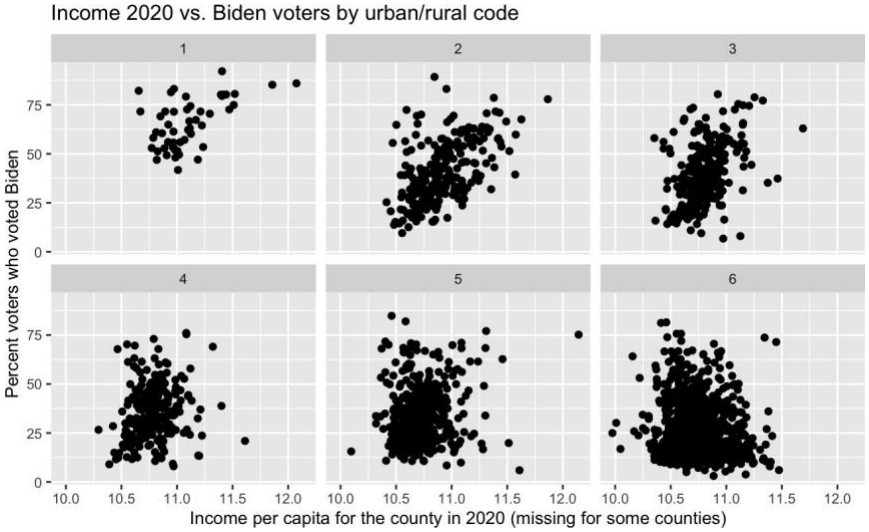
The third visualization’s goal is to further show the skewness in racial groups. This visualization takes the “Hispanic or Latino” group as an example. The left graph is a scatter plot of this group’s population against Biden voters. Compared to this, we see that the log transformed plot on the right seems to show a more distinct relationship between this predictor variable and the outcome variable. However, with the transformed plot also displaying high variance, these racial group variables may have to be removed from the model based on performance metrics.

Visualization 4: Income per capita 2020 vs. Biden voters



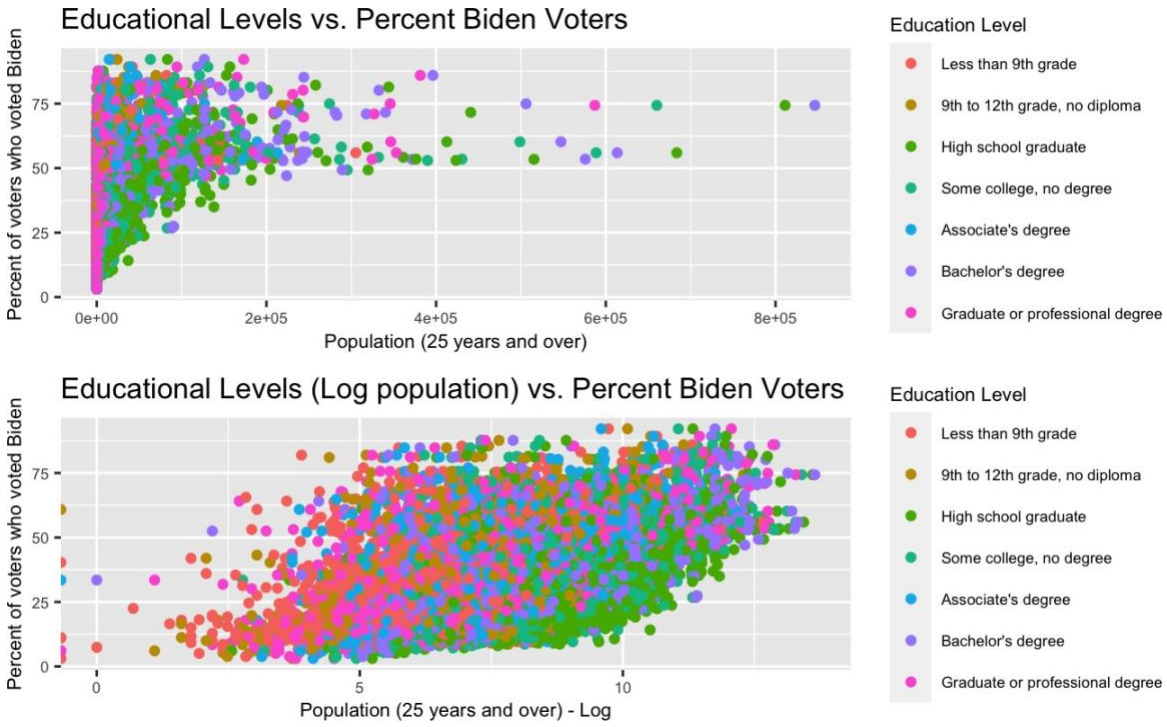
Similar to the previous visualization, the fourth visualization displays the before and after differences in log transformations for the "Income per capita for the county in 2020" predictor variable. Unlike the previous visualization, we see that transforming this variable does not cause a clear difference in variance or linearity. Based on these results, we may have to test transformations other than the log transformation or remove the "Income per capita" variables from the final model.

Visualization 5: Income per capita 2020 vs. Biden voters by urban/rural code



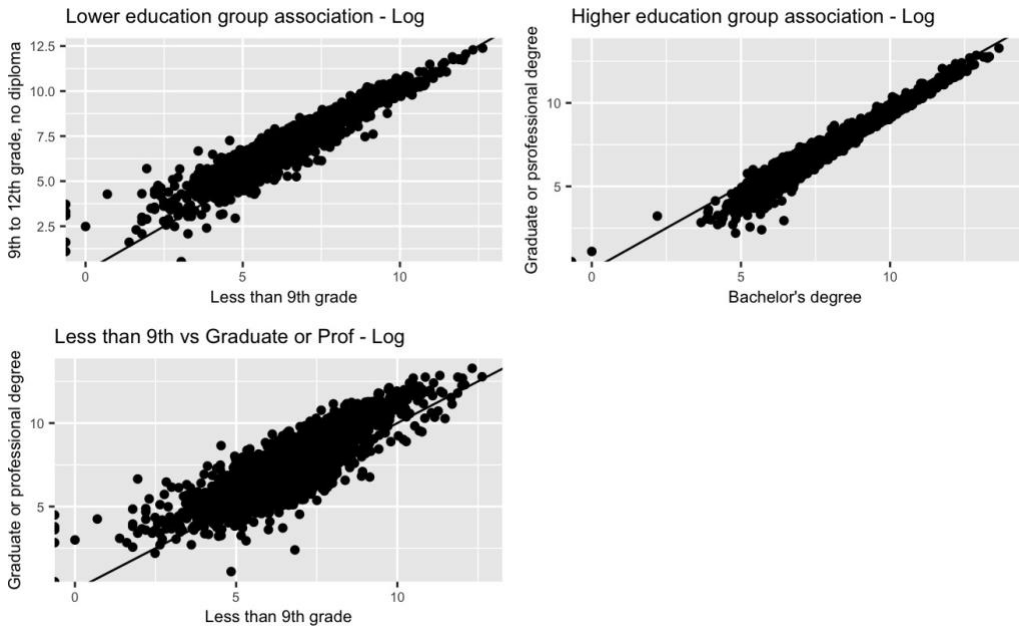
The fifth visualization divides the “Income per capita for the county in 2020” variable by the county’s urban/rural code. The second visualization showed that there was a clear difference in voting choices and distribution based on the county’s code. Similarly, this visualization shows higher variability as counties become more rural, and again, an overall higher percentage of Biden voters in the “most urban” counties. We can also clearly see more data points in more rural counties. Meanwhile, it is noted that potential outliers with high income in the “most urban” group.

Visualization 6: Education level vs. Biden Voters



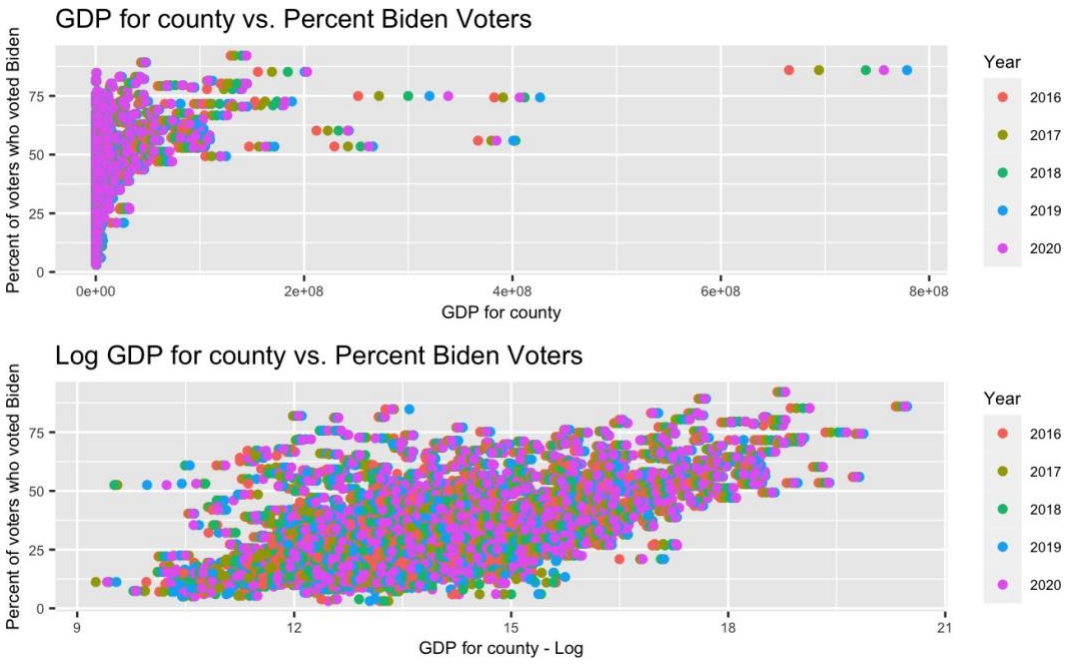
The sixth visualization plots the percent of Biden voters against all the education levels in the dataset. The dataset included education levels for adults 25 years and over. In the log transformed scatter plot, we can see that the distribution patterns look similar across different education levels. This may require us to investigate collinearity amongst these variables, and it is also noted that there is a large population of high school graduates in many counties.

Visualization 7: Association between education levels



Continuing from findings in the sixth visualization, the seventh visualization plots the relationship between some sample education levels. The first (top left) plots the relationship between the two least educated groups, the second (top right) plots the relationship between the two most educated groups, and lastly the third (bottom left) plots the relationship between the least and most educated groups. In all three pairs, we see a strong linear relationship between the log transformed variables. This may indicate issues with collinearity, and may warrant further testing and removal of variables.

Visualization 8: GDP vs. Biden Voters by year



This last visualization has the same format as the sixth visualization, and this plots the relationship between the county GDP and percent of Biden voters. Each color represents a different year from 2016 to 2020. Both the raw plot and the log transformed plot shows almost an exact overlap in voter choice and GDP across the years. This may also hint at a need for removal of redundant/unnecessary variables.

Preprocessing / Recipe

1. Removed the `id` column. The `id` column is an identifier for each entry in the data. It doesn't contain any valuable information. Before any recipe steps, the `name` column is also removed from the training set since this column is not relevant to the prediction and is not included in the test set.
2. Imputed missing values in predictor variables using the knn imputation method with step_impute_knn(). It estimates missing values based on the values of their nearest neighbors in the feature space. It is useful when there are missing values in predictor variables.
3. A total of 5 candidate recipes were created, each adding degrees of complexity. The `basic_recipe` handles skewed variables and imputed values as mentioned in step 2. The following `normalized_recipe` adds a normalizing step to all numeric predictors onto the basic recipe. Then the `interaction_recipe` tests interactions between income variables that raised areas of concern in the EDA steps. The `cor_var_recipe` then removes columns with collinearity and also removes variables containing single values and zero variance. The next step describes the `custom_recipe` that I built to handle irrelevant variables and create new columns that better encapsulate the data given.
4. Created multiple new variables by performing various mathematical operations on existing variables with step_mutate(). It aims to calculate various percentage values or transformations of the original features.
    a. 65 new variables are created by dividing corresponding estimated population groups over the estimated total population.They are percentage-based variables, which can be valuable features for modeling, as they provide a standardized way to compare population characteristics. Those new variables are pct_male, pct_female, pct_male_18over, pct_female_18over, pct_21andover, pct_62andove, pct_female_65over, pct_male_65over, pct_16over, pct_18over, pct_0to19, pct_20to34, pct_20to34, pct_35to54, pct_55to64, pct_65andovr, pct_18o_citizen_m, pct_18o_citizen_f, pct_18o_citizen, pct_onerace, pct_morerace, pct_white, pct_black, pct_indian, pct_asian_c, pct_api_c, pct_other_c, pct_his, pct_cherokee, pct_chippewa, pct_navajo, pct_sioux, pct_indian, pct_chinese, pct_filipino, pct_japanese, pct_korean, pct_viet, pct_othera, pct_hawaii, pct_chamorro, pct_samoan, pct_otherpi, pct_mexican, pct_puertorican, pct_cuban, pct_otherh, pct_whiteandblack, pct_whiteandindian, pct_whiteandasian, pct_indianandblack, pc_nothispanic_w, pc_nothispanic_b, pc_nothispanic_i, pc_nothispanic_a,

pc_nothispanic_hpi, pc_nothispanic_o, pc_nothispanic_more, pc_nothispanic_more_o, pc_nothispanic_three.

b. 19 new variables are created by dividing the estimated population of age groups with different levels of education over the estimated total population of that age group.They calculate the percentage of individuals with specific educational backgrounds within different age brackets. Some steps prevent the 0 in the denominator by using an ifelse() to return 1 instead of 0 in the denominator. These variables are valuable for understanding educational disparities across age groups and can be useful for modeling demographic and socioeconomic trends in the population. These variables are: pct_lesshighschool_18to24, pct_highschool_18to24, pc_bachelor_18to24, pc_lesshighschool_25over, pc_highschool_25over, pc_college_25over, pc_bachelorover_25over, pc_highschoolover_25over, pc_bachlor_25over, pc_graduate_25over, pc_highschool_25to34, pc_bachelor_25to34, pc_highschool_35to44, pc_bachelor_35to44, pc_highschool_45to64, pc_bachelor_45to64, pc_highschool_65over.

c. mean_income_per_cap is created to calculate the mean of income per capita over a five-year period.It is a key economic indicator that provides insight into the average income level of the population over time. It can be important for modeling socioeconomic factors and understanding economic trends.

d. mean_gdp is created by calculating the mean GDP over a five-year period. It's another crucial economic indicator that provides insights into the average economic output per person over time. It is useful for modeling economic conditions and trends.

e. pc_vote_population is created by dividing total votes by total population. This variable helps assess the level of civic engagement and voter turnout within the population.

f. pc_vote_eligible is created by dividing total votes by eligible voters. It provides insight into the voter turnout rate among those eligible to vote. It is a key metric in assessing the effectiveness of voter outreach and political engagement efforts.

5. Removed variables that are used to create new variables in step 3 with step_rm(). It's designed to avoid redundancy.
6. Converted the numerical variable x2013_code to factor variables and then create dummy variables from it with step_num2factor() and step_dummy(). This is done to handle the categorical variable.
7. Removed zero-variance predictors with step_zv(). Since predictors with zero variance do not provide any information for modeling.

8. To decide which recipe would work best in our pipeline, I used a workflow set and workflow map to fit multiple recipes against a basic linear regression model. The best performing recipe was then selected to be used for all other candidate models that we built. The result of this step is explained in the next section.

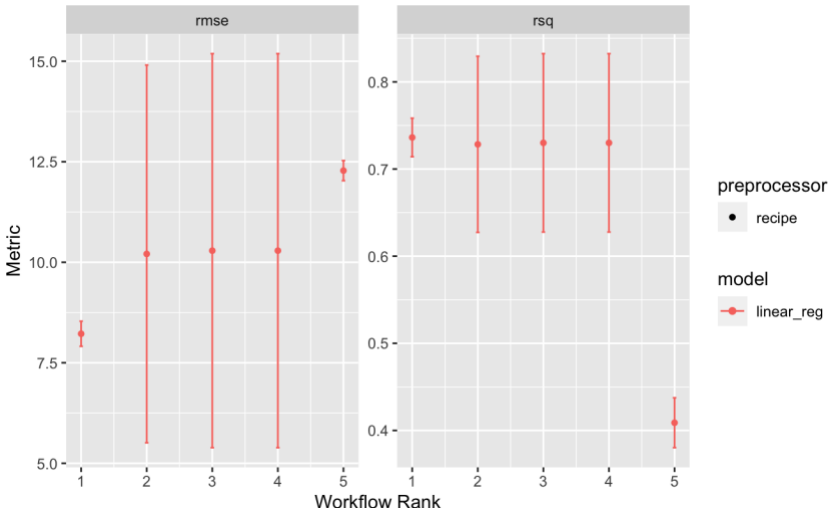
Candidate models / Model evaluation / Tuning

Model_Identifier	Type_of_Model	Engine	Recipe_Variables	Hyperparameters
Model 1	Linear Regression	lm (R base)	basic_recipe, normalized_recipe, interaction_recipe, cor_var_recipe, custom_recipe	None
Model 2	Random forest	ranger (R package)	custom_recipe	trees = 100, importance = 'impurity'
Model 3	K-Nearest neighbors	kknn (R package)	custom_recipe	mode = 'regression', neighbors = tune('k')
Model 4	Support Vector Machine	kernlab (R package)	custom_recipe	cost = tune('cost'), rbf_sigma = tune('sigma')
Model 5	Boosted Trees - Xgboost Engine	xgboost (R package)	custom_recipe	mode = 'regression', trees = 100, learn_rate = 0.05, tree_depth = tune('tree_depth'), mtry = tune('mtry'), min_n = tune('min_n'), loss_reduction = tune('loss_red')
Model 6	Boosted trees - Lightgbm Engine	lightgbm (R package)	custom_recipe	mode = 'regression', trees = 100, learn_rate = 0.05, tree_depth = tune('tree_depth'), mtry = tune('mtry'), min_n = tune('min_n'), loss_reduction = tune('loss_red')

Linear Regression

The 'linear_regression_spec' is a specification for a linear regression model used for modeling the relationship between the predictor variables and the target variable to find the best-fitting linear relationship. In this case, the model is configured to use the 'lm' engine, which stands for 'linear model,' with a standard implementation.

- Description: This is a simple linear regression model that predicts the RMSE of new data based on a linear combination of the features. Here I tried using 5 recipes as mentioned above (`basic_recipe`, `normalized_recipe`, `interaction_recipe`, `cor_var_recipe`, `custom_recipe`) to see which one performed the best. From the autoplot below, the best performing recipe is the `custom_recipe` which has a RMSE of approximately 8.5. Hence for all other models, I applied this particular recipe for fitting.
- Tuning: No tuning was performed



Random Forest

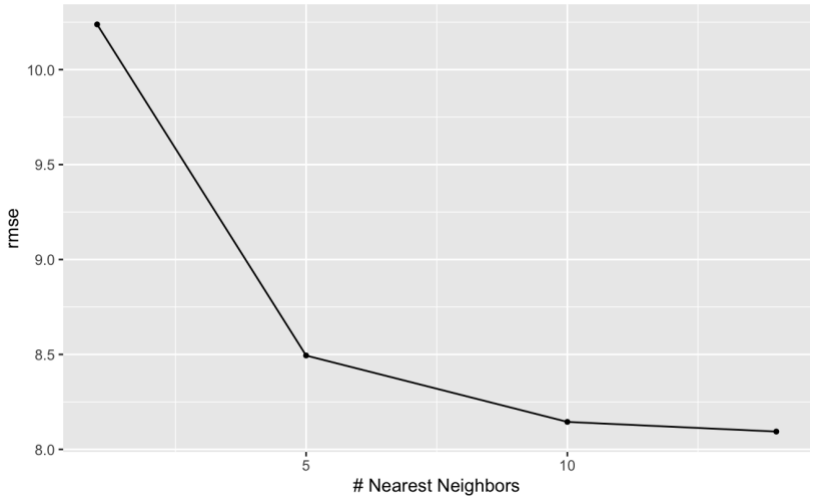
The 'random_forest_spec' serves as a specification for a regression model leveraging the power of the Random Forest algorithm. This algorithm combines the predictions from numerous decision trees to yield highly accurate predictions. In this specific configuration, the model employs 100 decision trees ('trees = 100') and is tailored for regression tasks, where it predicts continuous numeric values. The chosen engine is 'ranger', (set_engine("ranger", importance = "impurity")) known for its efficient implementation. Furthermore, the 'importance' parameter is set to 'impurity,' signifying that the model assesses variable importance by measuring impurity reduction.

- RMSE: 7.3
- Tuning: No tuning was performed
- Description: This is a random forest model that predicts the RMSE of new data by averaging the predictions of a set of decision trees.

K-Nearest Neighbors

The 'knn_spec' functions as a specification for a regression model employing the K-nearest neighbors algorithm, which is versatile and can be applied to both regression and classification tasks. In this instance, the 'mode = "regression"' parameter signifies that the K-nearest neighbors model is specifically tailored for regression tasks. It will generate numeric predictions for each input data point based on the values of its k-nearest neighbors. The 'neighbors = tune("k")' parameter indicates that the model will explore different values of k to identify the one that yields the optimal performance. By setting the engine to 'set_engine("kknn"),' the modeling flexibility is enhanced.

- RMSE: 8.0
- Tuning: The number of neighbors was tuned using a grid search with 4 values, more than 10 neighbors give the best results
- Description: This is a K-nearest neighbors model that predicts the RMSE of new data by finding the K most similar training examples and averaging their RMSE

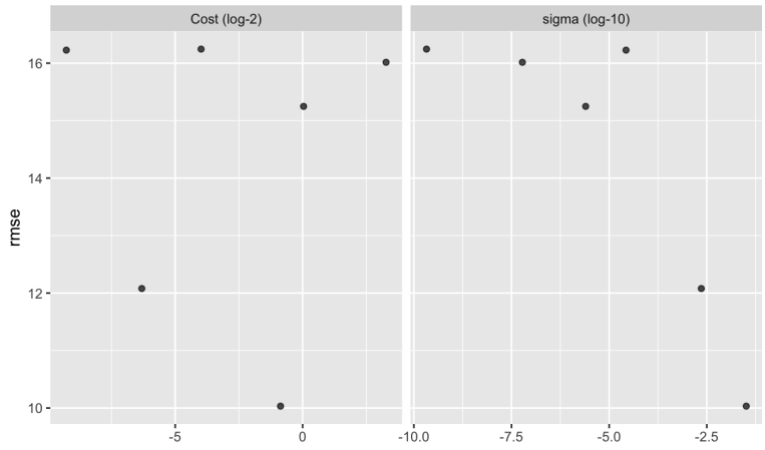


Support Vector Machine (SVM)

The 'svm_spec' provides a specification for a regression model based on the Support Vector Machine (SVM) algorithm, specifically employing the Radial Basis Function (RBF) kernel, which is applicable to both regression and classification tasks. In this context, when 'set_mode("regression")' is applied, the model is configured to predict a numeric value for each input data point. To optimize its performance, the model employs hyperparameter tuning for both the 'cost' and 'rbf_sigma' parameters. The 'cost' parameter governs the trade-off between maximizing the margin and minimizing classification errors, while 'rbf_sigma' influences the model's flexibility. By utilizing the 'tune' function for both 'cost' and 'rbf_sigma,' the hyperparameters are adjusted to find values that enhance model performance.

- RMSE: 10
- Tuning: The Cost(log-2) and sigma(log-10) hyper parameters were tuned using a grid search with 6 values each
- Description: This is a support vector machine model that predicts the RMSE of new data by finding the hyperplane that minimizes the error between the predicted and actual RMSE. We see

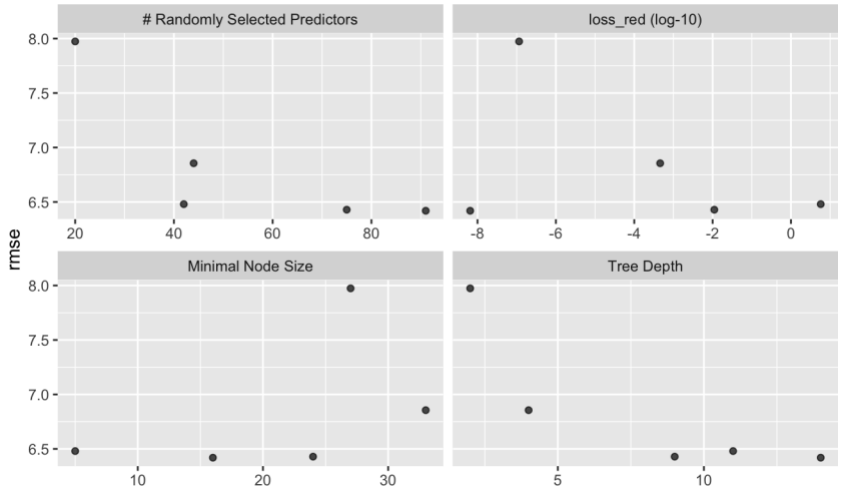
the Cost parameter performs best when it is set at approximately -1(log-2) and sigma parameter approximately -3(log-10).



Boosted trees (XGBoost)

The 'bt_xgboost_spec' serves as a specification for a regression model founded on the Gradient Boosting algorithm using XGBoost implementation. This approach amalgamates predictions from multiple weak decision trees in a sequential fashion to construct a strong predictive model. In this scenario, when 'mode = "regression"' is specified, the model is set to predict a numeric value for each input data point. Several key hyperparameters come into play, including the number of trees ('trees'), the learning rate ('learn_rate'), the maximum depth of individual trees ('tree_depth'), the number of randomly selected features at each split ('mtry'), the minimum required data points in a terminal node ('min_n'), and the threshold for loss reduction ('loss_reduction'). These hyperparameters play a vital role in regulating the model's complexity, training duration, and predictive performance. The tuning grids for 'tree_depth,' 'mtry,' 'min_n,' and 'loss_reduction' parameters aim to identify the optimal combination of these hyperparameters during the training process. The use of 'set_engine("xgboost")' underscores the model's high efficiency and accuracy.

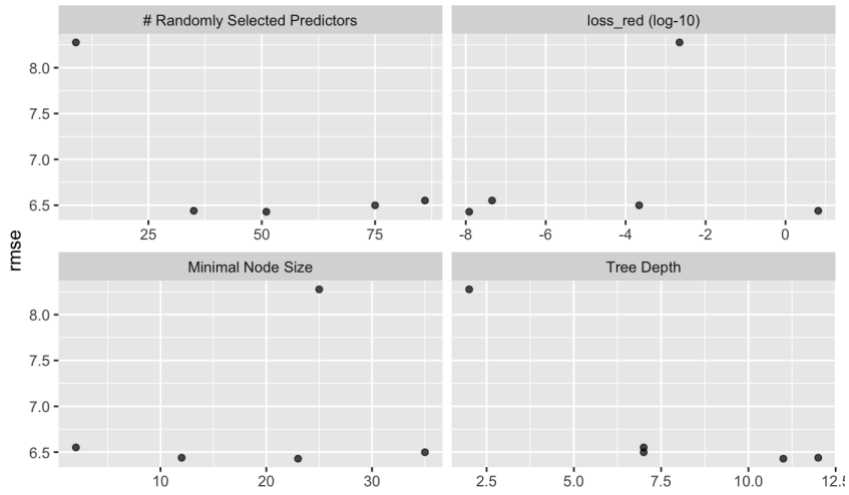
- RMSE: 6.3
- Tuning: The tree depth, mtry, min_n, and loss_reduction hyperparameters were tuned using a grid search with 5 values each
- Description: This is an XGBoost model that predicts the RMSE of new data by iteratively adding decision trees to a model in a way that minimizes the error between the predicted and actual RMSEs.



Boosted trees (LightGBM)

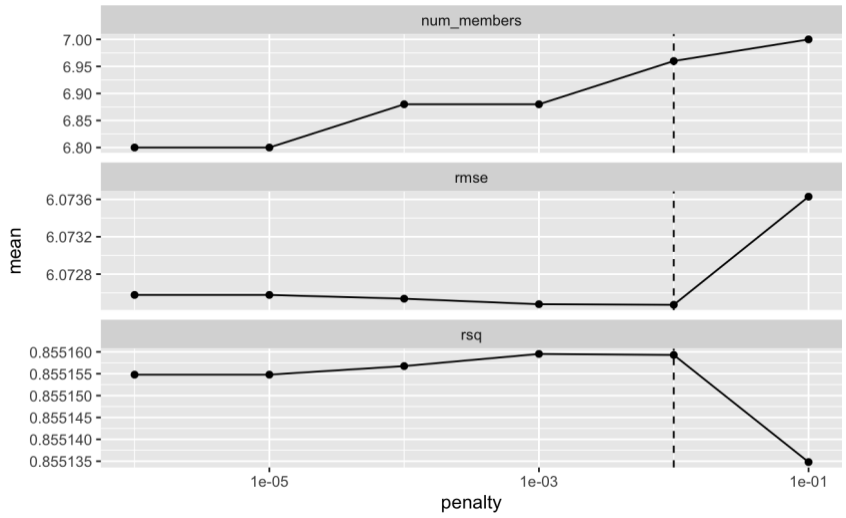
The 'bt_lightgbm_spec' represents a specification for a regression model founded on the Gradient Boosting algorithm, utilizing the efficient LightGBM implementation. Similar to the 'bt_xgboost_spec,' this approach also combines predictions from multiple weak decision trees in a sequential manner to craft a robust predictive model. The regression task, hyperparameters, and tuning grid closely mirror those of the 'bt_xgboost_spec,' with the only distinction being the choice of engine, which is set to 'set_engine("lightgbm").' LightGBM is high-performance and memory-efficient, and its speed and ability to handle large datasets.

- RMSE: 6.3
- Tuning: The tree_depth, mtry, loss_reduction, and min_n hyper parameters were tuned using a grid search with 5 values each
- Description: This is a LightGBM model that predicts the RMSE of new data by adding decision trees to a model in a way that minimizes the error between the predicted and actual RMSEs.



Stacked Model

- RMSE: 6.27
- Description: This is a stacked model that combines the predictions of the top four candidate models to predict the RMSE of new data.



The stacking coefficient tells us how much weight to give to each individual model in the stacked model. The plot above helps make sure that the model has the right trade-off between minimizing the number of members and optimizing performance. The model with the highest stacking coefficient is given the most weight, and the model with the lowest stacking coefficient is given the least weight. So in the visualization below, we see that the boost_tree is given the most importance when determining the predictions.

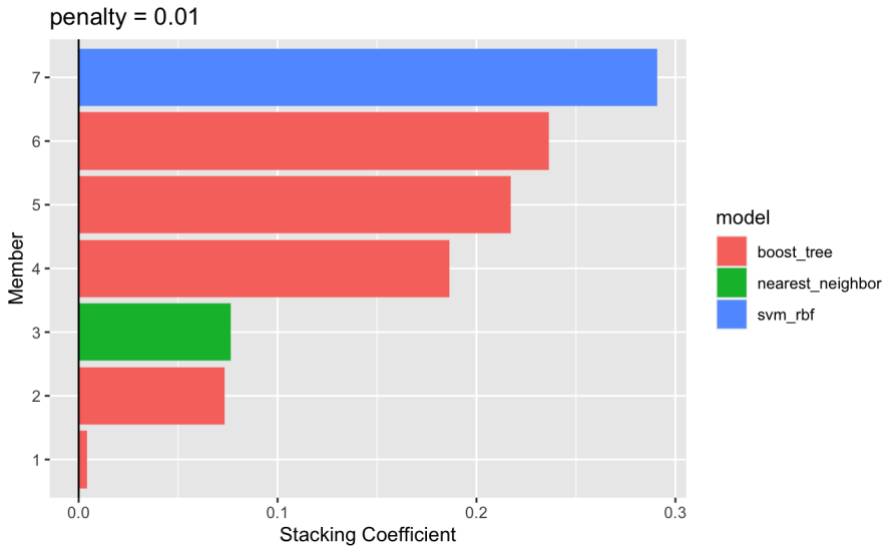


Table Summary

Model name	Model Identifier	RMSE
Linear regression	`linear_regression_spec`	8.5
KNN	`knn_spec`	8.0
SVM	`svm_spec`	10
Random forest	`random_forest_spec`	7.5
LightGBM	`bt_lightgbm_spec`	6.3
XGBoost	`bt_xgboost_spec`	6.3
Stacked model	`test_stack`	6.27 (based on private Kaggle leaderboard)

Discussion of final model

For the final model, I opted for a model stack that combined four tunable models: k-nearest neighbors, support vector machine, boosted trees with xgboost engine, and boosted trees with lightgbm engine.

One of the notable strengths of the chosen approach lies in the optimized tuning process based on candidate models. This method allows for harnessing the best predictions from these models, resulting in a more robust and accurate overall prediction. Additionally, model stacking proves effective in mitigating overfitting by capitalizing on the diversity inherent in the candidate models. The likelihood of all base models making the same overfitting mistakes is reduced.

However, it's important to acknowledge the weaknesses of this approach. The increased complexity, though somewhat beyond control, is a factor that should be considered. At times, the process felt more like a guessing game than a controlled and customized model-building endeavor. Furthermore, model stacking leads to limited interpretability, making it challenging to extract insights and understand the underlying relationships in the data. It's crucial to be mindful of the potential for diminishing returns, as adding more candidate models to the stack may not always result in proportionate improvements in performance. There comes a point where the additional complexity may not be justified by the marginal gains in predictive power, and the process can devolve into yet another guessing game.

In terms of potential improvements, with more time and computational resources at disposal, it would have expanded to include a wider array of candidate models in the testing. Additionally, exploring various combinations of fewer or more models could have provided further insights into the optimal approach. It would have been beneficial to allocate more time towards developing multiple model recipes and ensuring a comprehensive understanding of all the different columns in the dataset.

Appendix I: Final annotated R script

```
# Load packages library(tidyverse)
library(tidymodels) library(tidyselect)
library(recipes) library(caret)
library(baguette) library(rpart)
library(e1071) library(stacks)
library(xgboost) library(lightgbm)
library(bonsai) library(kknn)
library(kernlab) library(gridExtra)
library(ggridges)



#Load train and test data train <-
read_csv('train.csv') test <-
read_csv('test.csv')

# Cross-validation data set.seed(1)
train_folds <- vfold_cv(train, v = 10, strata = percent_dem)


#EDA (add later)


# Identifying skewed columns:

racial_groups <- train %>% select(x0034e:x0039e, -x0036e, x0044e,
    x0051e)

racial_groups_tidy <- racial_groups %>%
    pivot_longer(cols = x0034e:x0051e, names_to = "race", values_to = "population")

#Calculate the IQR for each race group racial_groups_tidy <-
racial_groups_tidy %>%
    group_by(race) %>% mutate(
```

```
    Q1 = quantile(population, 0.25), Q3 =
    quantile(population, 0.75), IQR = Q3 - Q1
  ) %>%
  ungroup()

# Define a threshold to determine outliers (e.g., 1.5 times the IQR) outlier_threshold <- 1.5

# Remove outliers
racial_groups_tidy <- racial_groups_tidy %>%
  filter(population >= (Q1 - outlier_threshold * IQR) & population <= (Q3 + outlier_threshold * IQR))

# Create the density ridgeline plot with outliers removed ggplot(racial_groups_tidy, aes(x = population, y = race, fill = race))
+
  geom_density_ridges() +
  labs(x = "Population", y = "Racial group", title = "Distribution of sample racial groups (outliers removed)")+
  theme(text = element_text(size = 11), plot.title = element_text(size =
          12), axis.title.x = element_text(hjust = 0.5),
          axis.title.y = element_text(hjust = 0.5)) + scale_y_discrete(name = "Racial Group",
                                labels = c("One race", "Two or more races", "White", "Black or African American", "American
Indian and Alaska Native", "Asian", "Other Asian")) +
  guides(fill = "none")



# Box plot for the relationship between total votes and percent_Dem ggplot(train, aes(x =
as_factor(x2013_code), y = percent_dem))+
  geom_boxplot(fill = rainbow(6)) +
  labs(x = "Urban/Rural code used by CDC (1 is most urban 6 is most rural)", y = "Percent of voters who voted Biden", title =
"Distribution of Percent of Biden Voters by Urban/Rural Code")+
  theme(text = element_text(size = 12), plot.title = element_text(size
          = 13))



# Scatterplot sample to show that columns such as population groups require transformations

## scatter plot for the relationship between total votes and percent_Dem hisp_lat <- ggplot(train, aes(x =
total_votes, y = percent_dem))+
  geom_abline() +
```

```
    geom_point() +
    labs(x="Population of Hispanic or Latino (of any race)", y="Percent of voters who voted Biden", title="Population vs.
Percent Biden Voters")+
    theme(text=element_text(size=10), plot.title = element_text(size
                = 12))

##scatter plot for the relationship between total votes and percent_Dem hisp_lat_log <- ggplot(train, aes(x =
log10(total_votes), y= percent_dem)) +
    geom_abline() + geom_point()+
    labs(x="Population of Hispanic or Latino (of any race) - Log", y="Percent of voters who voted Biden", title="Log Population
vs. Percent Biden Voters")+
    theme(text=element_text(size=10), plot.title = element_text(size
                = 12))

grid.arrange(hisp_lat, hisp_lat_log, ncol = 2)



# scatter plot for relationship between income_per_cap_2020 and percent_dem income_2020 <- ggplot(train, aes(x =
income_per_cap_2020, y = percent_dem)) +
    geom_abline() + geom_point()+
    labs(x="Income per capita for the county in 2020", y="Percent of voters who voted Biden", title="Income 2020 vs. Biden
voters")+
    theme(text=element_text(size=10), plot.title = element_text(size
                = 12))

# scatter plot for relationship between income_per_cap_2020 and percent_dem  income_2020_log <-ggplot(train, aes(x =
log10(income_per_cap_2020), y = percent_dem)) +
    geom_abline() + geom_point()+
    labs(x="Income per capita for the county in 2020 - log", y="Percent of voters who voted Biden", title="Log Income 2020 vs.
Biden voters")+
    theme(text=element_text(size=10), plot.title = element_text(size
                = 12))

# Combine above two

grid.arrange(income_2020, income_2020_log, ncol = 2)



# scatter plot for relationship between income_per_cap_2020 and percent_dem, grouped by area type
ggplot(train, aes(x = log(income_per_cap_2020), y = percent_dem)) + geom_point() +
```

```
    facet_wrap(~x2013_code) +
    labs(x='Income per capita for the county in 2020 (missing for some counties)', y='Percent voters who voted Biden', title=
'Income 2020 vs. Biden voters by urban/rural code')+
    theme(text=element_text(size=10), plot.title = element_text(size
                = 12))



## create a plot based on different education group education<-train%>%
    select(percent_dem, c01_007e:c01_013e)

education_tidy <- education %>%
    pivot_longer(c("c01_007e", "c01_008e", "c01_009e", "c01_010e", "c01_011e", "c01_012e", "c01_013e"), names_to=
"education", values_to="population")

education_levels <- ggplot(education_tidy, aes(x = population, y = percent_dem, color=education))+
    geom_point() +
    labs(x = "Population (25 years and over)",
            y="Percent of voters who voted Biden",
            title = "Educational Levels vs. Percent Biden Voters") + scale_color_discrete(name="Education Level",
                                        labels=c("Less than 9th grade", "9th to 12th grade, no diploma", "High school graduate",
"Some college, no degree", "Associate's degree", "Bachelor's degree", "Graduate or professional degree"))+
    theme(text=element_text(size=8), plot.title = element_text(size =
                12))

#Log transform

education$c01_007e          <-          log(education$c01_007e)
education$c01_008e          <-          log(education$c01_008e)
education$c01_009e          <-          log(education$c01_009e)
education$c01_010e          <-          log(education$c01_010e)
education$c01_011e          <-          log(education$c01_011e)
education$c01_012e          <-          log(education$c01_012e)
education$c01_013e <- log(education$c01_013e)

education_tidy_log <- education %>%
    pivot_longer(c("c01_007e", "c01_008e", "c01_009e", "c01_010e", "c01_011e", "c01_012e", "c01_013e"), names_to=
"education", values_to="population")

education_levels_log <- ggplot(education_tidy_log, aes(x = population, y = percent_dem, color=education))+
    geom_point() +
```

```
    labs(x="Population (25 years and over) - Log", y="Percent of voters
         who voted Biden",
         title = "Educational Levels (Log population) vs. Percent Biden Voters") + scale_color_discrete(name="Education Level",
                                        labels=c("Less than 9th grade", "9th to 12th grade, no diploma", "High school graduate",
"Some college, no degree", "Associate's degree", "Bachelor's degree", "Graduate or professional degree"))+
    theme(text=element_text(size=8), plot.title = element_text(size =
            12))

# Combine above two
grid.arrange(education_levels, education_levels_log, ncol = 1)


# Sample of similar groups and opposing groups

lower_ed <- ggplot(train, aes(x = log(c01_007e), y = log(c01_008e))) + geom_abline() +
    geom_point() +
    labs(x="Less than 9th grade", y="9th to 12th grade, no diploma", title="Lower education group association - Log")+
    theme(text = element_text(size = 8))

higher_ed <- ggplot(train, aes(x = log(c01_012e), y = log(c01_013e))) + geom_abline() +
    geom_point() +
    labs(x="Bachelor's degree", y="Graduate or professional degree", title="Higher education group association - Log")+
    theme(text = element_text(size = 8))

low_high_ed <- ggplot(train, aes(x = log(c01_007e), y = log(c01_013e))) + geom_abline() +
    geom_point() +
    labs(x="Less than 9th grade", y="Graduate or professional degree", title="Less than 9th vs Graduate or Prof - Log")+
    theme(text = element_text(size = 8))

# Combine above three
grid.arrange(lower_ed, higher_ed, low_high_ed, ncol = 2, nrow = 2)


## create a plot based on GDP by year gdp <- train %>%
    select(percent_dem, gdp_2016:gdp_2020)
```

```
gdp_tidy <- gdp %>%
    pivot_longer(cols = c("gdp_2016","gdp_2017","gdp_2018","gdp_2019","gdp_2020"), names_to="Year", values_to=
"GDP")

gdp_plot<-ggplot(gdp_tidy, aes(x=GDP, y=percent_dem, color=Year)) + geom_point() +
    labs(x="GDP for county",
            y="Percent of voters who voted Biden",
            title="GDP for county vs. Percent Biden Voters") + scale_color_discrete(name="Year",
                                        labels = c("2016", "2017", "2018", "2019", "2020")) +
    theme(text=element_text(size=8), plot.title = element_text(size =
              12))

#Log transform

gdp$gdp_2016        <-      log(gdp$gdp_2016)
gdp$gdp_2017        <-      log(gdp$gdp_2017)
gdp$gdp_2018        <-      log(gdp$gdp_2018)
gdp$gdp_2019        <-      log(gdp$gdp_2019)
gdp$gdp_2020 <- log(gdp$gdp_2020)

gdp_tidy_log <- gdp %>%
    pivot_longer(cols = c("gdp_2016","gdp_2017","gdp_2018","gdp_2019","gdp_2020"), names_to="Year", values_to=
"GDP")

gdp_plot_log<-ggplot(gdp_tidy_log, aes(x=GDP, y=percent_dem, color=Year)) + geom_point() +
    labs(x="GDP for county - Log",
            y="Percent of voters who voted Biden",
            title="Log GDP for county vs. Percent Biden Voters") + scale_color_discrete(name="Year",
                                        labels = c("2016", "2017", "2018", "2019", "2020")) +
    theme(text=element_text(size=8), plot.title = element_text(size =
              12))

# Combine above two

grid.arrange(gdp_plot, gdp_plot_log, ncol = 1)


# Preprocessing / Recipes


# Remove non-numeric column ("id") and handle missing values numeric_vars<-train[, !names(train)
%in% c("name", "id")]

numeric_vars_na_omit <- na.omit(numeric_vars)  # Remove rows with missing values
```

```
# Calculate the correlation matrix for numeric variables correlation_matrix <-
cor(numeric_vars_na_omit)

# Identify highly correlated variables

highly_correlated <- findCorrelation(correlation_matrix, cutoff = 0.75)

# Calculate skewness for each column in the dataframe df skew_values <-
sapply(numeric_vars, skewness)

# Set a threshold for skewness threshold <- 1.0

# Get column names with high skewness
high_skew_columns <- names(skew_values[abs(skew_values) > threshold]) high_skew_columns <-
as.character(high_skew_columns) high_skew_columns <- na.omit(high_skew_columns)

education_cols = c("c01_003e" , "c01_004e", "c01_005e","c01_006e" , "c01_007e", "c01_008e" ,"c01_009e",
"c01_010e","c01_011e","c01_012e","c01_013e", "c01_014e","c01_015e", "c01_016e", "c01_017e", "c01_018e"
,"c01_019e", "c01_020e" , "c01_021e", "c01_022e", "c01_023e", "c01_024e", "c01_025e", "c01_026e", "c01_027e")

cols_to_drop_1 = c("x0001e", "x0018e","x0019e", "x0020e", "x0021e", "x0022e", "x0023e","x0024e" , "x0025e", "x0026e", "x0027e",
"x0029e", "x0030e", "x0031e",
"x0034e", "x0058e", "x0062e", "x0064e", "x0065e",                    "x0066e", "x0067e", "x0068e", "x0069e",
"x0076e", "x0077e","x0078e",
"x0079e","x0080e","x0081e","x0082e","x0083e",
"x0002e","x0003e","x0005e","x0006e","x0007e","x0008e","x0009e","x0010e","x0011e","x 0012e","x0013e","x0014e","x0015e", "x0016e",
"x0017e")


# Remove "name" column from train train <-
subset(train, select = -name)



# Create all recipes basic_recipe <-
    recipe(percent_dem ~ ., data = train) %>%
    step_rm(id) %>%
    step_log(all_of(high_skew_columns), base = 10, offset = 0.001) %>% step_impute_knn(all_predictors())

# Add normalization step
normalized_recipe <-
```

```
    basic_recipe %>% step_normalize(all_numeric_predictors())

# Add interaction columns
interaction_recipe<-
    normalized_recipe %>%
    step_interact(terms = ~ income_per_cap_2016:x2013_code +
                            income_per_cap_2017:x2013_code +
                            income_per_cap_2018:x2013_code +
                            income_per_cap_2019:x2013_code +
                            income_per_cap_2020:x2013_code)

# Account for correlation/collinearity and zero/low variance columns cor_var_recipe<-
    interaction_recipe%>% step_corr(all_predictors(), threshold = 0.75)
    %>% step_zv(all_predictors())%>% step_nzv(all_predictors())

# Create custom recipe with multiple new columns calculating percentages of different groups based on total population


code2013 <- c("one", "two", "three", "four", "five", "six") custom_recipe<-


recipe(percent_dem ~ ., data = train)%>%
    step_rm(id) %>%
    step_impute_knn(all_predictors())%>% step_mutate(pct_male =
    x0002e/x0001e)%>% step_mutate(pct_female=x0003e/x0001e)  %>%
    step_mutate(pct_male_18over=x0026e/x0001e)%>%
    step_mutate(pct_female_18over = x0027e / x0001e) %>%
    step_mutate(pct_21andover=x0022e/x0001e)%>%
    step_mutate(pct_62andove=x0023e/x0001e)%>%
    step_mutate(pct_female_65over = x0031e / x0001e) %>%
    step_mutate(pct_male_65over=x0030e/x0001e)%>%
    step_mutate(pct_16over=x0020e/x0001e)%>% step_mutate(pct_18over=
    x0025e/x0001e)%>%
    step_mutate(pct_0to19=(x0005e+x0006e+x0007e+x0008e)/x0001e)%>% step_mutate(pct_20to34=
    (x0009e+x0010e)/x0001e)%>% step_mutate(pct_35to54=(x0011e+x0012e)/x0001e)%>%
    step_mutate(pct_55to64=(x0013e+x0014e)/x0001e)%>% step_mutate(pct_65andovr=(x0015e+
    x0016e+x0017e)/x0001e)%>% step_mutate(pct_18o_citizen_m=x0088e/x0001e)%>%
    step_mutate(pct_18o_citizen_f=x0089e/x0001e)%>% step_mutate(pct_18o_citizen=x0087e/x0001e)
    %>% step_mutate(pct_onerace=x0034e/x0001e)  %>%
    step_mutate(pct_morerace = x0035e / x0001e) %>%
```

```
step_mutate(pct_white = x0037e / x0001e) %>% step_mutate(pct_black = x0038e /
x0001e) %>% step_mutate(pct_indian = x0039e / x0001e) %>%
step_mutate(pct_asian = x0044e / x0001e) %>% step_mutate(pct_api = x0052e /
x0001e) %>% step_mutate(pct_otherrace = x0057e / x0001e) %>%
step_mutate(pct_white_c = x0064e / x0001e)  %>% step_mutate(pct_black_c =
x0065e / x0001e)  %>% step_mutate(pct_indian_c = x0066e / x0001e)  %>%
step_mutate(pct_asian_c = x0067e / x0001e)  %>% step_mutate(pct_api_c =
x0068e / x0001e)  %>% step_mutate(pct_other_c = x0069e / x0001e)  %>%
step_mutate(pct_his = x0071e / x0001e) %>% step_mutate(pct_cherokee =
x0040e/x0001e) %>% step_mutate(pct_chippewa = x0041e/x0001e) %>%
step_mutate(pct_navajo = x0042e/x0001e) %>% step_mutate(pct_sioux =
x0043e/x0001e) %>% step_mutate(pct_indian = x0045e/x0001e) %>%
step_mutate(pct_chinese = x0046e/x0001e) %>% step_mutate(pct_filipino =
x0047e/x0001e) %>% step_mutate(pct_japanese = x0048e/x0001e) %>%
step_mutate(pct_korean = x0049e/x0001e) %>% step_mutate(pct_viet =
x0050e/x0001e) %>% step_mutate(pct_othera = x0051e/x0001e) %>%
step_mutate(pct_hawaii = x0053e/x0001e) %>% step_mutate(pct_chamorro =
x0054e/x0001e) %>% step_mutate(pct_samoan = x0055e/x0001e) %>%
step_mutate(pct_otherpi = x0056e/x0001e) %>% step_mutate(pct_mexican =
x0072e/x0001e) %>% step_mutate(pct_puertorican = x0073e/x0001e) %>%
step_mutate(pct_cuban = x0074e/x0001e) %>% step_mutate(pct_otherh =
x0075e/x0001e) %>% step_mutate(pct_whiteandblack = x0059e/x0001e) %>%
step_mutate(pct_whiteandindian = x0060e/x0001e) %>%
step_mutate(pct_whiteandasian = x0061e/x0001e) %>%
step_mutate(pct_indianandblack = x0062e/x0001e) %>%
step_mutate(pc_nothispanic_w = x0077e / x0001e) %>%
step_mutate(pc_nothispanic_b = x0078e / x0001e) %>%
step_mutate(pc_nothispanic_i = x0079e / x0001e) %>%
step_mutate(pc_nothispanic_a = x0080e / x0001e) %>%
step_mutate(pc_nothispanic_hpi = x0081e / x0001e) %>%
step_mutate(pc_nothispanic_o = x0082e / x0001e) %>%
step_mutate(pc_nothispanic_more = x0083e / x0001e) %>%
step_mutate(pc_nothispanic_more_o = x0084e / x0001e) %>%
step_mutate(pc_nothispanic_three = x0085e / x0001e) %>%
step_mutate(pct_lesshighschool_18to24 = c01_002e / ifelse(c01_001e != 0, c01_001e, 1))  %>%
```

```
    step_mutate(pct_highschool_18to24 = c01_003e/ifelse(c01_001e != 0, c01_001e, 1))
%>%
    step_mutate(pc_college_18to24 = c01_004e/ifelse(c01_001e != 0, c01_001e, 1)) %>% step_mutate(pc_bachelor_18to24 =
    c01_005e/ifelse(c01_001e != 0, c01_001e, 1)) %>% step_mutate(pc_lesshighschool_25over = (c01_007e +
    c01_008e)/c01_006e) %>% step_mutate(pc_highschool_25over = c01_009e/c01_006e) %>%
    step_mutate(pc_college_25over = (c01_010e + c01_011e)/c01_006e) %>% step_mutate(pc_bachelorover_25over =
    c01_015e/c01_006e) %>% step_mutate(pc_highschoolover_25over = c01_014e/c01_006e) %>%
    step_mutate(pc_associate_25over = c01_011e / c01_006e ) %>% step_mutate(pc_bachlor_25over = c01_012e /
    c01_006e ) %>% step_mutate(pc_graduate_25over = c01_013e / c01_006e ) %>% step_mutate(pc_highschool_25to34 =
    c01_017e/ c01_016e) %>% step_mutate(pc_bachelor_25to34 = c01_018e/ c01_016e) %>%
    step_mutate(pc_highschool_35to44 = c01_020e/ c01_019e) %>% step_mutate(pc_bachelor_35to44 = c01_021e/
    c01_019e) %>% step_mutate(pc_highschool_45to64 = c01_023e/ c01_022e) %>% step_mutate(pc_bachelor_45to64 =
    c01_024e/ c01_022e) %>% step_mutate(pc_highschool_65over = c01_026e/ c01_025e) %>%
    step_mutate(pc_bachelor_65over = c01_027e/ c01_025e) %>% step_mutate(mean_income_per_cap =
    (income_per_cap_2016 + income_per_cap_2017+
income_per_cap_2018+income_per_cap_2019+income_per_cap_2020)/5)  %>% step_mutate(mean_gdp = (gdp_2016+
    gdp_2017+ gdp_2018 + gdp_2019+ gdp_2020)/5) %>% step_mutate(pc_vote_population = total_votes / x0001e) %>%
    step_mutate(pc_vote_eligible = total_votes/x0087e) %>%
    step_rm(x0002e, x0009e, x0010e, x0003e, x0005e, x0006e, x0007e, x0008e, x0033e, x0011e, x0012e, x0013e, x0014e,
x0015e, x0016e, x0017e) %>%
    step_rm(x0020e, x0021e, x0022e, x0023e, x0024e, x0025e, x0029e, x0087e, x0088e, x0089e) %>% #age
    step_rm(x0034e, x0035e, x0036e, x0037e, x0038e, x0039e, x0040e, x0052e, x0044e,x0057e, x0058e, x0064e,
x0065e,x0066e, x0067e, x0068e, x0069e, x0071e) %>%
#race
    step_rm(c01_002e, c01_015e,c01_011e,c01_010e, c01_009e, c01_008e, c01_007e, c01_005e, c01_004e, c01_003e)%>%
    step_rm(c01_017e, c01_018e, c01_019e, c01_020e, c01_021e, c01_022e, c01_023e, c01_024e, c01_025e, c01_026e,
c01_027e)%>%
    step_rm(x0041e, x0042e, x0043e, x0045e, x0046e, x0047e, x0048e, x0049e, x0050e, x0051e, x0053e, x0054e, x0055e,
x0056e)%>%
    step_rm(matches("^x007[2-9]e$|^x008[0-5]e$"))  %>%
    step_rm(x0019e, x0026e, x0027e, x0031e, x0030e, x0060e, x0061e, x0062e, x0059e)
%>%
    step_rm(c01_001e, c01_006e, c01_012e, c01_013e, c01_016e)%>% step_rm(c01_014e)%>%
    step_rm(total_votes)%>% step_num2factor(x2013_code, levels =
    code2013) %>% step_dummy(x2013_code)%>%
    step_zv(all_predictors())
```

```
# Candidate models / Model evaluation / tuning

#1. Linear Regression (baseline testing for recipes)

# Initialize candidate models for testing

# Linear Regression (baseline testing for selecting best recipe) linear_regression_spec <-
    linear_reg() %>%
    set_engine("lm")

#2. Random forest


# Random forest (basic)

random_forest_spec <-
    rand_forest(trees = 100) %>%
    set_mode("regression") %>% set_engine("ranger", importance =
    "impurity")

#3. K-nearest neighbors


# K-nearest neighbors knn_spec <-
    nearest_neighbor(
        mode="regression", neighbors =
        tune("k")
    ) %>%
    set_engine("kknn")

#4. Support vector machine


# Support vector machine svm_spec <-
    svm_rbf(
```

```
    cost=tune("cost"), rbf_sigma =
    tune("sigma")
) %>%
set_engine("kernlab") %>%
set_mode("regression")

#5. Boosted trees - xgboost engine


# Boosted trees - xgboost engine bt_xgboost_spec<-
  boost_tree(
    mode = "regression", trees =
    100,
    learn_rate = 0.05,
    # Define a tuning grid tree_depth =
    tune("tree_depth"), mtry=tune("mtry"),
    min_n=tune("min_n"), loss_reduction =
    tune("loss_red")
  ) %>%
  set_engine("xgboost")


#6. Boosted trees - lightgbm engine


# Boosted trees - lightgbm engine bt_lightgbm_spec<-
  boost_tree(
    mode = "regression", trees =
    100,
    learn_rate = 0.05,
    # Define a tuning grid tree_depth =
    tune("tree_depth"), mtry=tune("mtry"),
    min_n=tune("min_n"), loss_reduction =
    tune("loss_red")
  ) %>%
  set_engine("lightgbm")


# Workflow sets (recipes x models)
```

```
# Run this part to select ideal recipe (baseline)

# Set up preprocess object containing all candidate recipes preproc<- list(basic=
basic_recipe,
                              normal=normalized_recipe, interact =
                              interaction_recipe, cor_var=
                              cor_var_recipe, cust=custom_recipe)

keep_pred <- control_resamples(save_pred = TRUE, save_workflow = TRUE)



# Create a workflow_set to test all candidate recipes against baseline linear regression model
lm_models <- workflow_set(preproc = preproc,
                                        models=list(lm= linear_regression_spec), cross = FALSE)

# Use workflow_map to call in v-fold cross validation data lm_models <-
    lm_models %>% workflow_map("fit_resamples",
                              # Options to `workflow_map()`: seed=1,
                              verbose=TRUE,
                              # Options to `fit_resamples()`:
                              resamples = train_folds, control = keep_pred)



# This shows that the custom recipe performs best lm_models %>%
    collect_metrics() %>%
    filter(.metric == "rmse")


autoplot(lm_models)


# Random forest individual fit rf_workflow

<-
```

```
    workflow() %>% add_recipe(custom_recipe) %>%
    add_model(random_forest_spec)

rf_res <- rf_workflow %>%
    fit_resamples(resamples = train_folds,
                                control = control_resamples(save_pred = TRUE))



rf_res %>% collect_metrics()%>%
    filter(.metric == "rmse")



# KNN individual fit knn_wflow <-
    workflow() %>%
    add_model(knn_spec)%>%
    add_recipe(custom_recipe)

knn_res <- tune_grid(
        knn_wflow,
        resamples = train_folds, metrics =
        metric,
        grid=4,
        control = control_stack_grid()
    )



show_best(knn_res, metric = "rmse")



autoplot(knn_res)



# SVM individual fit svm_wflow <-
    workflow() %>%
```

```
    add_model(svm_spec)%>%
    add_recipe(custom_recipe)

svm_res <- tune_grid(
        svm_wflow,
        resamples = train_folds, grid = 6,
        metrics = metric,
        control = control_stack_grid()
    )


show_best(svm_res, metric = "rmse")


autoplot(svm_res)



#Boosted trees - xgboost fit bt_xgboost_workflow <- workflow() %>%
    add_model(bt_xgboost_spec) %>%
    add_recipe(custom_recipe)

## Define a tuning grid
# grid_xgboost <- grid_regular(
#       tree_depth(range = c(1, 30), trans = NULL),   # Adjust the range based on your knowledge of the problem
#       mtry(range = c(2, 20), trans = NULL),  # Adjust the range based on your knowledge of the problem
#       min_n(range = c(5, 50), trans = NULL),   # Adjust the range based on your knowledge of the problem
#       loss_reduction(range = c(0, 0.1), trans = NULL),   # Adjust the range based on your knowledge of the problem
#       levels = 5  # or another number to control the grid size # )

bt_xgboost_res <- tune_grid(
        bt_xgboost_workflow, resamples =
        train_folds, grid = 5,
        metrics = metric,
```

```
        control = control_stack_grid()
    )


show_best(bt_xgboost_res, metric = "rmse")


autoplot(bt_xgboost_res)



#Boosted trees - lightgbm fit bt_light_workflow <- workflow() %>%
    add_model(bt_lightgbm_spec) %>%
    add_recipe(custom_recipe)

bt_light_res <- tune_grid(
        bt_light_workflow, resamples =
        train_folds, grid = 5,
        metrics = metric,
        control = control_stack_grid()
    )


show_best(bt_light_res, metric = "rmse")


autoplot(bt_light_res)


# Stacked model

# Define stack object adding the top four candidate models test_stack <-
    stacks() %>% add_candidates(knn_res)%>%
    add_candidates(svm_res)%>%
    add_candidates(bt_xgboost_res) %>%
    add_candidates(bt_light_res)
```

```
test_stack <- test_stack %>%
    blend_predictions()


autoplot(test_stack)


autoplot(test_stack, type = "weights")



# Fit the models based on evaluation of the stack function test_stack <-
    test_stack %>% fit_members()




# Generate final predictions


test_res_stacked <-
    test_stack %>%
    predict(test) %>%
    cbind(test %>% select(id)) %>% select(id, .pred)

test_res_stacked <- test_res_stacked %>%
    rename(percent_dem = .pred) %>%
    write_csv("stacked_predictions.csv")

print(head(test_res_stacked, n = 15))
```

Appendix II: References

1. Suls, R. (2016). Educational divide in vote preferences on track to be wider than in recent elections. Retrieved September 2, 2023, https://www.pewresearch.org/short-reads/2016/09/15/educational-divide-in-vote-preferences-on-track-to-be-wider-than-in-recent-elections/
2. U.S. Bureau of the Census. (2020c). Table A-2. Reported Voting and Registration by Region, Educational Attainment and Labor Force Status for the Population 18 and Over: November 1964 to 2020. Retrieved September 2, 2023, https://www.census.gov/data/tables/time-series/demo/voting-and-registration/voting-historical-time-series.html
3. Pew Research Center. (2020). *The Changing Racial and Ethnic Composition of the U.S. Electorate*. Retrieved September 2, 2023, https://www.pewresearch.org/2020/09/23/the-changing-racial-and-ethnic-composition-of-the-u-s-electorate/
4. Hartig, H., Daniller, A., Keeter, S., & Green, T. V. (2023). Republican Gains in 2022 Midterms Driven Mostly by Turnout Advantage. Retrieved September 2, 2023, https://www.pewresearch.org/politics/2023/07/12/republican-gains-in-2022-midterms-driven-mostly-by-turnout-advantage/