

# Statistical Analysis and Prediction of the 2020 U.S. Presidential Election

**Author:**

Ching-Yi Wang (Ian)

Date: September 7, 2023

# 1 Introduction

**Library:** ggplot2, dplyr, tidymodels, xgboost, glmnet, ranger, , keras


In our regression project report, we explore the multitude of factors that may influence the response variable, "Percent voters who voted Biden," within a dataset containing over 120 variables. While individual predictors, such as the "Total population: One race: Asian" variable and "Age Under 34 years old," are of particular interest, our analysis encompasses the entirety of available data. Employing advanced modeling techniques, notably XGBoost models, we aim to construct predictive models that illuminate the complex landscape of voter behavior during the 2020 United States presidential election. This holistic approach enables us to uncover valuable insights, shedding light on the multifaceted dynamics that contributed to the election's outcome. Through rigorous analysis, we seek to understand the comprehensive set of variables that played pivotal roles in shaping voter preferences and electoral results.

## Reference:

1. XGBoost Documentation.  
<https://xgboost.readthedocs.io/en/stable>
2. A Guide on XGBoost Hyperparameters Tuning.  
<https://www.kaggle.com/code/prashant111/a-guide-on-xgboost-hyperparameters-tuning>
3. Tidymodels Stacks.  
<https://stacks.tidymodels.org>
4. Tidymodels Tune.  
<https://tune.tidymodels.org>
5. Tidymodels Parsnip.  
<https://parsnip.tidymodels.org>

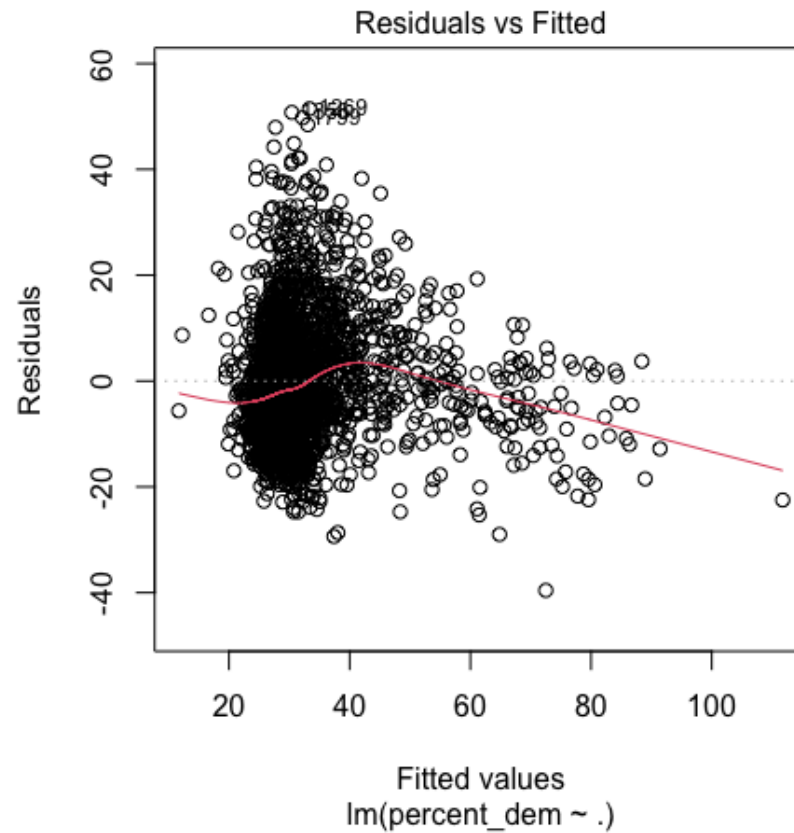
## 2 Exploratory Data Analysis

Figure 1: Summary Table of Percent Demographic

	percent_dem
count	2331.0
mean	33.277053137352400
std	15.940397918393900
min	3.09090909090909
25%	21.012394057216300
50%	29.9594145861518
75%	42.3081334092653 
max	92.1496939214069

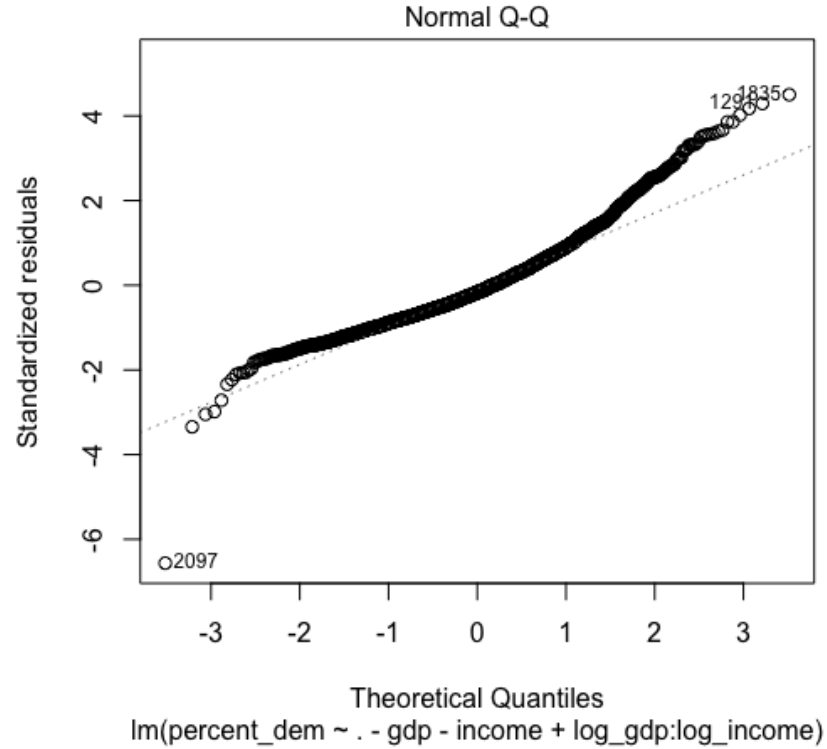
**Summary:** According to the summary of the percentage democratic vote, the average 33.3% of the population will tend to vote for Joe Biden as president during the election in 2020. However, the actual percentage will significantly vary due to the different states where different states will have different political opinions such as the maximum percentage who will vote for Joe Biden is 92.1% and the minimum percentage is 3.1%. Therefore, we expected a certain degree of skewness might be exist in the voting distribution and the skewness is possibly a right skew because the mean is 33.3% and the median is 29.6% where large outliers pull the distribution to a larger percentage, so the data summary suggests that the voting distribution is not normally distributed.

Figure 2: Residual Plot



**Summary:** The residual plot suggests nonlinearity with most data points on the left indicating a non-linear relationship between predictors and the response. Outliers with high Cook's distances skew the plot and cause the x-axis unbalance, this indicates the data distribution issues and existence of potential bias in the dataset.

Figure 3: Normal Q-Q Plot



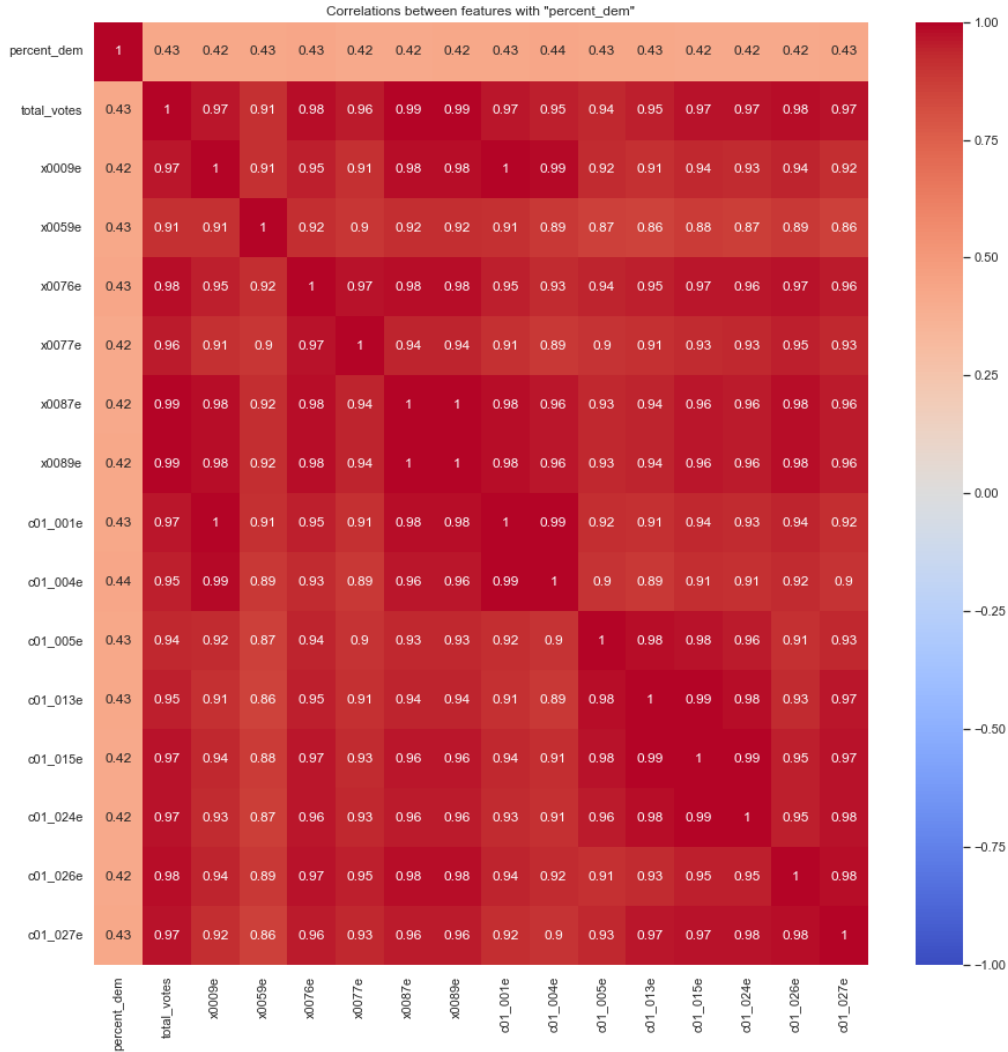
**Summary:** The normal QQ plot shows a combination of left downward skew (negative skewness) and right upward skew (positive skewness), which display the dataset is tends to be more complex than normal distribution. This pattern suggests that the dataset's distribution deviates from a symmetry or normal distribution. According to the plot, we know that the dataset are more concentrated to extreme high outlier values by observing the right upward skew. In addition, the left downward skew suggests that the dataset has a concentration on the extreme low outliers by observing the left downward skew. Therefore, this plot shows there has a certain degree of skewness and outliers that impact the normal distribution of the plot.

Figure 4: Scale-Location Plot



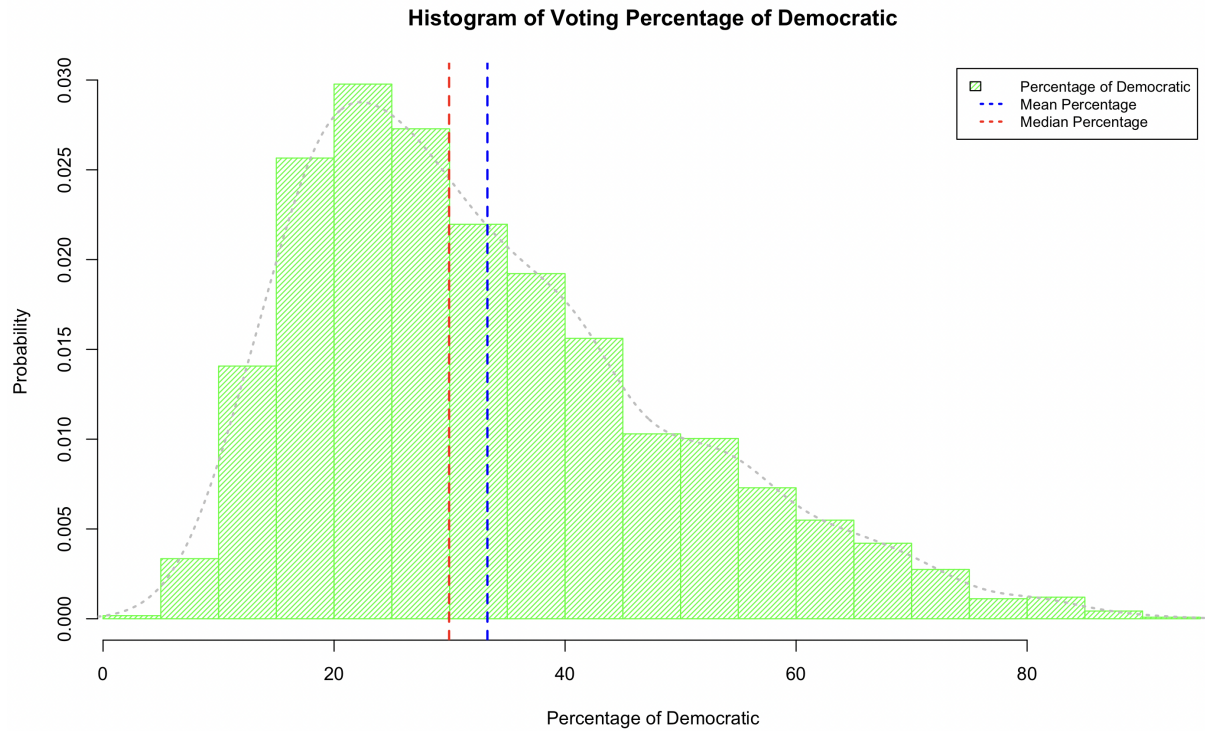
**Summary:** The scale location performs a fan shape distribution pattern which indicates the existence of heteroscedasticity where the variance of the residuals is not constant and varies asymmetrically with changes in the predictor variables. Therefore, the model will tend to have a low reliability and tend to have a skewness in this data distribution due to the non-constant variance.

Figure 5: Heat Map



**Summary:** According to the heatmap made by the correlation of all variables with percent democratic, we can observe that every variables does not have a strong correlation with percent democratic variables (max = 0.43), instead the other variables seems to highly correlated to each other which might cause the problem of collinearity. Therefore, the prediction might be altered due to the multicollinearity.

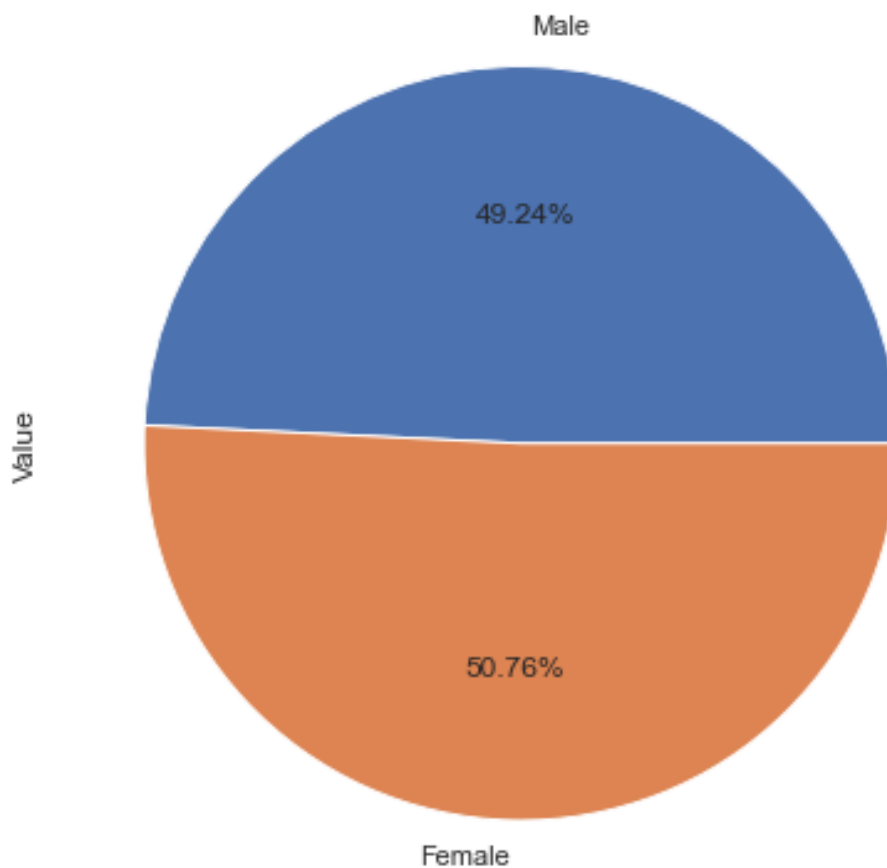
Figure 6: Histogram of Percentage Democratic



**Summary:** Based on the histogram of the "Percent Demographic" variable distribution, it shows that the majority of values cluster within the 20% to 40% range. However, there are some notable data points that exceed the 80% mark which might indicate that some city is extreme democratic voting behavior where most people will vote for Joe Biden in election, while others tend toward the lower end, approaching 0% which indicate there are some extreme non-democratic behavior city that will not vote for Joe Biden. The presence of values significantly exceeding the 80% threshold suggests the potential existence of extreme outliers within this dataset.

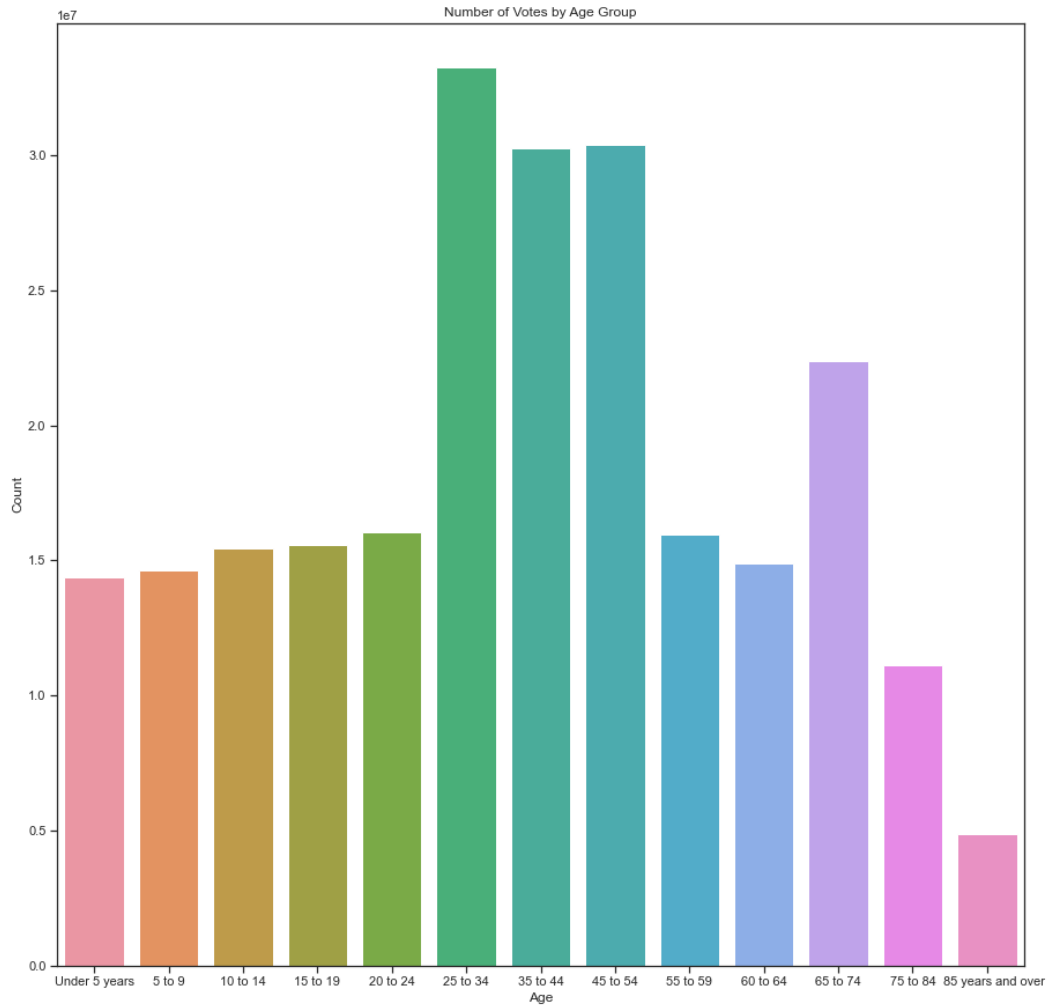


Figure 7: Pie Plot: Gender



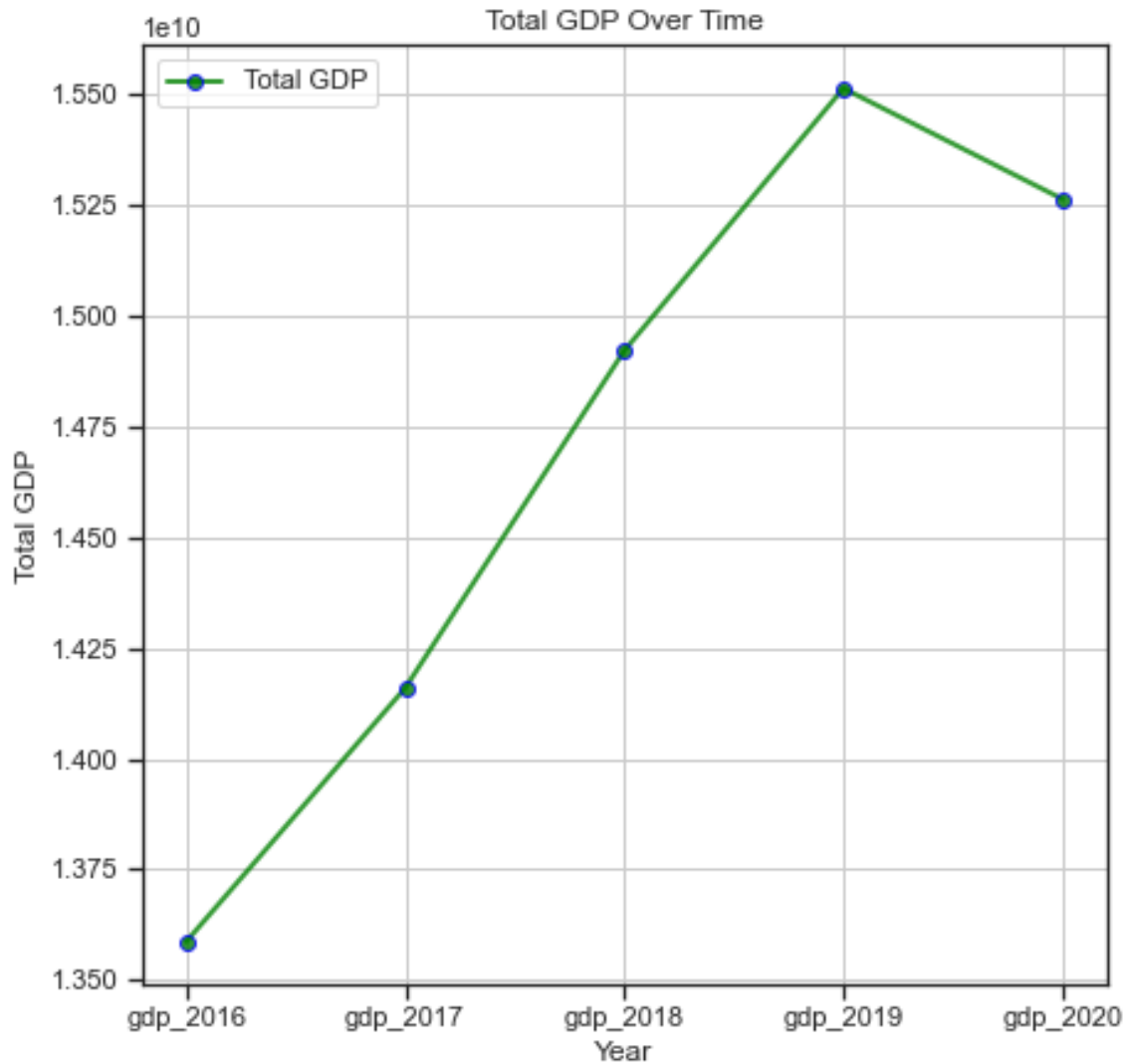
**Summary:** In the graph depicting gender distribution, a nearly equal representation emerges, with males constituting 49.24% and females comprising 50.76% of the dataset. This balanced distribution underscores a significant parity between the number of males and females within the sample. The near 50-50 split suggests a fair and equitable representation of both genders in the dataset, reflecting a commendable effort to ensure gender diversity and neutrality.

Figure 8: Bar Plot:Age



**Summary:** According to the bar plot, the age category of 25-34 stands out with the highest count, surpassing all other age groups. Following closely as the second highest are individuals within the 45-54 age bracket, while the 35-44 age group follows in third place. The order of the top five is completed by the 65-74 age group. Meanwhile, the remaining age groups exhibit similar, nearly equal, population counts, making them comparable in terms of representation.

Figure 9: Line Plot: GDP 2016 to 2020



From the wikipedia U.S. presidential election resource, we know the last president is a republican party. According to the line plot of the total GDP from the last presidential term, we can see the GDP has a positive slope in the first 3 years of the term, but after reaching the peak in 2019 we can see the overall GDP trend begin to significantly decrease. Therefore, the republican votes might have impact due to overall GDP reduction where people might tend to vote for a new political party during the presidential election in 2020.

### 3 Preprocessing / Recipe

**Summary:** From the voting data we began with 125 variables including the id variable. The goal of the competition is to predict the percentage of democrats, "**percent dem**", from a selection of the remaining 124 variables. To begin with our recipe we decided to predict "**percent dem**" by all other variables in the training data. Our next step was to remove the id variable which we are able to do because of the random method implementation in our model creation process. In order to enhance our model, and improve performance we decided to add 7 variables. These 7 variables are "**pct minority**", "**pct white**", "**pct minor**," "**pct male**", "**pct female**", "**pct college**", "**pct old**".

These variables were created in order to more accurately describe the demographic variables of a county relatively since the data set contains counties with largely varying population sizes. "**Pct minority**" addresses the percentage of Black or African American, American Indian and Alaska Native, Asian, Native Hawaiian and Other Pacific Islander, Some other race groups in the information. These groups were the minority groups as White/Caucasian people made up greater than 50 % of the population in the United States in 2020. We then broke these groups down further to only "**pct white**", the percentage of white people in a community. "**Pct minor**" is the percentage of a community not of voting age in the United States (under 18 years old). We split the population of these counties into "**pct male**", and "**pct female**" to understand gender representation. "**Pct college**" is used to represent the education level of a community, and the "**pct old**" used to represent the population 45 years or older.

The thing these groups have in common is that they tend to have influences on the voting path. Keep in mind these following claims about voting patterns are in the scope of 2020 United States of America. Minority populations, college educated individuals, and females are more likely to vote Democrat than other groups. Meanwhile white individuals, males, and individuals over the age of 45 are all more likely to not vote Democrat in elections. The proportion of minors in an area could be helpful for understanding the age demographic, and the voting population of a county as a younger age likely indicates younger families.

Aside from this our final steps in our preprocessing were using a **step impute mean(income per cap 2016:gdp 2020)** line in our code. This identifies missing values in the columns addressing income per capita and GDP by county, and it fills them in with the mean value of the respective column. The last step was normalizing all of the predictor variables in our data set in order to scale them due to the large range amongst variables.

## 4 Candidate models/Model evaluation/Tuning

### Model Candidates:

We first started with a set of preliminary models without any tuning of hyperparameters to get an idea of how the models would perform with the given data set. We started off with linear regression, random forest, neural net, and boosted tree with the mode set to “regression”. The initial prepping for the models were all similar with just removing the “id” variable and imputing missing values by mean. We wrote off using a linear regression model with just the “lm” engine as its performance was significantly worse than the random forest model used in Homework 3. So, we decided to use the “glmnet” engine to allow tuning. When testing the neural net model, we found that it overfitted the data greatly. We also saw that the boosted tree model with the default model parameters performed better than the random forest model, and so we decided to start tuning the boosted tree model to increase performance. Eventually we adjusted the ranges of the hyperparameters during tuning and added new variables to get our best performing model. Below is a chart of our models.

Model Identifier	Model Type	Engine	Recipe	Hyperparamters
Random Forest	Random Forest	ranger	Remove "id", Impute by mean, Normalize	"mtry"=mtry(c(1,123)), "trees"=trees(c(200,1500)), "min_n"=min_n(c(20,40)))
Boosted Tree	Boosted Tree	xgboost	Remove "id", Impute by mean, Normalize	mtry(range = c(15, 25)), min_n(range = c(5, 15)), trees(range = c(15, 25)), tree_depth(range = c(5, 15)), learn_rate(range = c(-5, -1)), loss_reduction(range = c(-5, -1)), size = 200
Linear Regression (Lasso)	Linear Regression	glmnet	Remove "id", Impute by mean, Normalize, Add Variables	penalty(), levels = 10
Neural Net	Keras Model Sequential		Omit missing values, Remove highly correlated variables	layer_dense(units = 64, activation = "relu", input_shape = dim(X)[2]) %>% layer_dense(units = 32, activation = "relu") %>% layer_dense(units = 16, activation = "relu") %>% layer_dense(units = 1)
Final Model	Boosted Tree	xgboost	Remove "id", Impute by mean, Normalize, Add new Variables	mtry(range = c(35, 80)), min_n(range = c(10, 20)), trees(range = c(300, 600)), tree_depth(range = c(10, 20)), learn_rate(range = c(-5, -1)), loss_reduction(range = c(-5, -1)), sample_size = sample_prop(), size = 25

One thing to note is that we also tried using `xgboost()` which is very similar to `boost tree's xgboost`, with parameters also very similar, but just with different names, like `eta`, `n_rounds`, `gamma`, etc.

## 5 Model Evaluation and Tuning

On a wider scale, our models included mainly random forest (rand forest) and xgboost (through boost tree()), narrowing down to xgboost. We had also tried neural networks, however, it did not work too well with the workflow and we suspected it was overfitting a lot. The linear regression (linear reg) model was also tried, but we quickly found it was too simple and weak relative to our complex dataset and other models. The table below shows our top 3 performing xgboost models using the latin hypercube sampling and regular (expand.grid())

XGBOOST - Using Latin Hypercube Sampling									
	MSE	mtry	trees	min_n	tree_depth	learn_rate	loss_reduction	sample_size	stop_iter
1	41.37898	50	487	18	14	0.023846	0.0012889	0.861323	5
2	43.40664	61	521	12	12	0.028164	0.0290256	0.583513	5
3	44.06075	54	452	12	13	0.023225	0.0021997	0.547753	5
Using normal grid									
4	58.3289	100	100	20	30	0.23	0.002	1	5
5	60.5283	100	125	20	10	0.23	0.1	1	5
6	63.6804	20	25	12	10	0.23	0.1	1	5



### The code for Latin Hypercube Sampling (LHS) for XGBoost:

```

1 boost_grid <- grid_latin_hypercube(
2   mtry(range = c(35, 80)),
3   min_n(range = c(10, 20)),
4   trees(range = c(200, 600)),
5   tree_depth(range = c(10, 20)),
6   learn_rate(range = c(-5, -1)),
7   loss_reduction(range = c(-5, -1)),
8   sample_size = sample_prop(),
9   size = 25
10 )

```

We talk more in depth about the pros and cons of LHS in the discussion.

One crucial way we roughly measured overfitting was using the validation/test MSE score from Kaggle. If we had gotten an RMSE around 6.433, we'd want to obtain a MSE score around  $6.433^2 = 41.38$ . If we did get  $\pm 1$ , we'd consider it a good fit, neither over or under fitting. Another metric to note is the std error, which was consistently between 0.09 to 0.12.

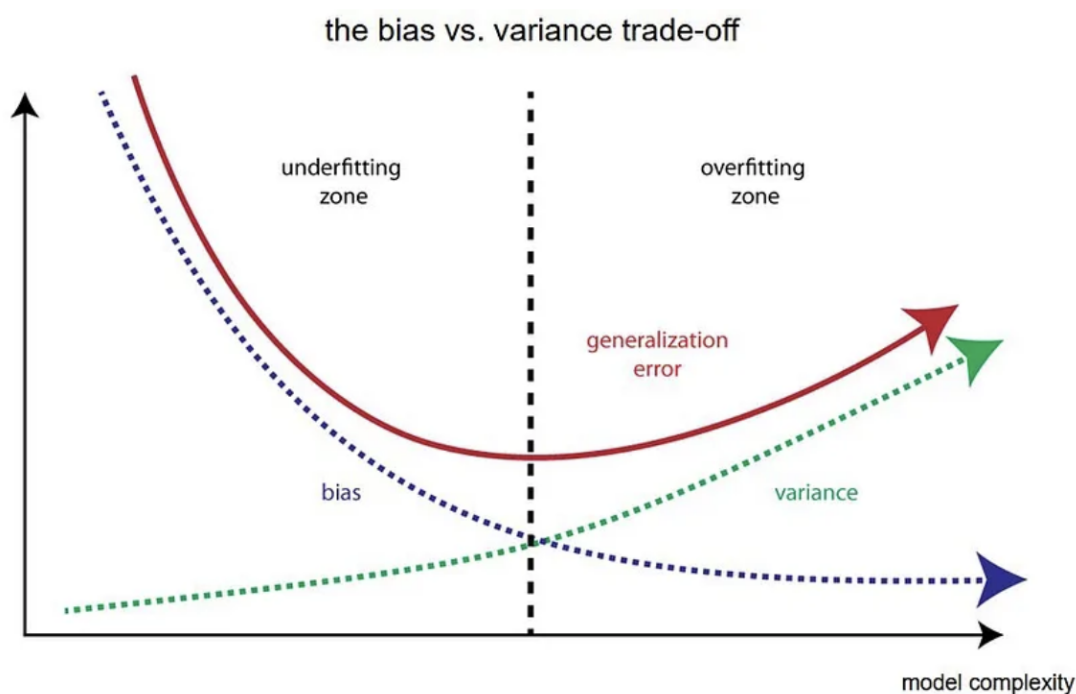
Other results using random forest are in the table below

Random Forest - Using Latin Hypercube Sampling				
	MSE	mtry	trees	min_n
1	57.9121	50	487	18
2	58.5225	61	521	12
3	58.6756	54	452	12

Below are some results for linear regression, using the glmnet engine, since it allowed for tuning the penalty and mixture parameters

Linear Regression - Using Latin Hypercube Sampling				
	MSE	penalty	mixture	std_err
1	88.7364	0.866	0.895	0.123
2	92.1600	0.0512	0.765	0.160
3	95.4529	2.42	0.550	0.115

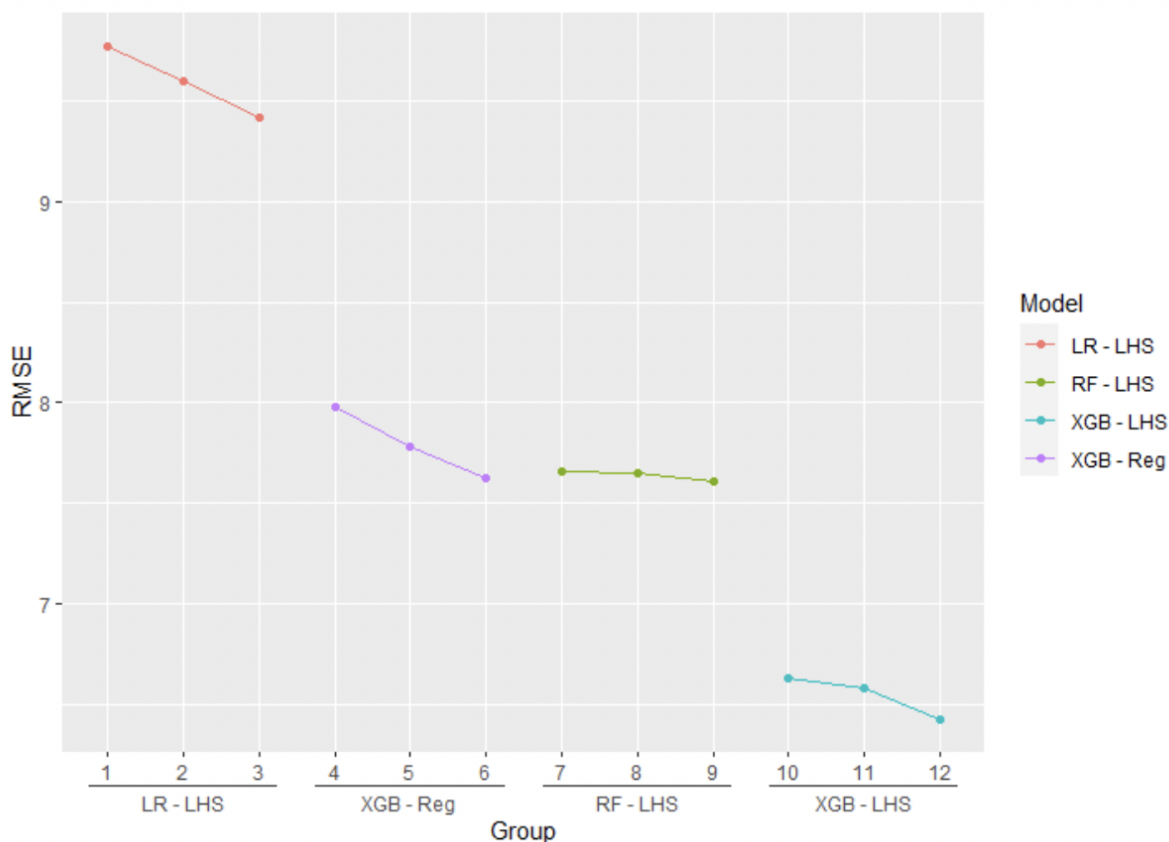
For all of these models, we consistently used 10 fold cross-validation. We did play around with the folds ranging from 5 to 15, and decided 10 was the best in terms of our metric. With higher numbers like 15, our validation test would've been too small. Even though 15 gave the lower rmse at times, we suspected it was due to the bias variance tradeoff that is also present for validating model performance. We also tried to keep this in mind, with a useful image shown below.



**Reference from:**

<https://towardsdatascience.com/bias-and-variance-but-what-are-they-really-ac539817e171>

The graph below shows a comparison of models and their RMSE values



Comparing our model to the other candidate models shows the evaluation to our selection:

#### **XGBoost vs. Random Forest:**

Strengths of Random Forest: Random Forest is relatively easier to tune compared to XGBoost and provides some degree of interpretability. It can handle high-dimensional data as well.

Weaknesses of Random Forest: XGBoost often outperforms Random Forest when it comes to predictive accuracy, especially with large and complex datasets.

#### **XGBoost vs. Linear Regression:**

Strengths of Linear Regression: Linear Regression is interpretable and straightforward. It can work well if there are linear relationships between predictors and the target variable.

Weaknesses of Linear Regression: Linear Regression may not capture the complex, non-linear relationships that XGBoost can capture in our dataset.

#### **XGBoost vs. Neural Networks:**

Strengths of Neural Networks: Neural Networks are highly flexible and can capture intricate patterns in data, but they require large amounts of data and computational resources.

Weaknesses of Neural Networks: Neural Networks can be overkill for the dataset size, and they may not offer significant advantages in terms of predictive accuracy compared to XGBoost, and can overfit the training set a lot, resulting in a misleading test/validation score.

## 6 Discussion of Final Model

Using the grid latin hypercube has helped us a lot in several ways. Latin hypercube sampling, through `grid latin hypercube()`, introduces randomness into the hyperparameter search, which can help explore a broader and more diverse set of hyperparameter combinations compared to a regular grid search (`expand.grid()`). And due to our highly dimensional space, this method offers efficiency and a balanced exploration of the hyperparameter space since it doesn't require evaluating all possible combinations or overemphasizing certain parameters like in a normal grid, saving time and computational resources. And its biggest advantage is that it is well-suited for parallelization since each sample can be evaluated independently, taking advantage of multiple computing resources, which we employed through `doParallel()`.

Selection of the model was much easier when we decided to further explore `xgboost`. Before this decision, we tried several combinations with `linear reg()`, `rand forest()` as well as a bit of neural networks. Comparing all these models with `xgboost`, we found that `xgboost` had performed consistently better in the `rmse` metric as well as validation/test score available on Kaggle. As for `xgboost` itself, we had first explored a hyperparameter grid through `expand.grid`. However, we switched to using Latin hypercube sampling due to the advantages listed above.

The strengths of our XGBoost model include:

**High Predictive Accuracy:** XGBoost is well-suited for datasets with a large number of variables like ours. It excels at capturing complex, non-linear relationships between predictors and the target variable, which can lead to high predictive accuracy.

**Parallel Processing:** XGBoost can leverage parallel processing capabilities, making it faster to train than some other algorithms. This is advantageous when dealing with a large dataset. Hyperparameter flexibility: You can fine-tune the model by adjusting hyperparameters to achieve optimal performance, which can be particularly useful when dealing with a dataset with many variables.

Some weaknesses include:

**Hyperparameter Tuning:** Tuning XGBoost’s hyperparameters can be challenging and computationally expensive, given the size of the dataset and the large number of variables. Finding the optimal set of hyperparameters may require significant computational resources and time.

**Interpretability:** XGBoost is not highly interpretable. Understanding the precise reasons behind a particular prediction can be complex, especially with a large number of predictors.

**Data Quality:** XGBoost’s performance is sensitive to the quality of the input data. Noisy data, present abundantly in our datasets, can lead to suboptimal results. Ensuring data quality through preprocessing and cleaning is essential in this case.

**Data Size:** XGBoost may not perform as well on very small datasets due to its complexity, as it does on moderate to large. Our dataset contains 2,300 observations which is considered small.

**Overfitting:** We might not have fully kept the bias-variance tradeoff in mind, because we utilised a large number of trees in our grid, causing slight overfitting.

Some improvements could have been to use nested cross validation, though would be very time and resource consuming, could’ve given a better and realistic test score. Looking at our results, it looks like we slightly overfit some of the data. And due to xgboost’s tendency to overfit if a lot of noisy data is present, we should’ve done more effective data preprocessing, cleaning and feature engineering. We also could have replaced most of the data with percentages so it is easier for our model to compare, at the same time removing noise from unneeded variables. We could have also tried some ensemble methods, like stacking or blending. Combining the predictions of multiple models, including XGBoost, may have led to better results.

Additional data, but not trivial like percent of democrats/republicans, would be useful. Some examples may include geographical data of the county, like location, urban or rural classification. Other demographic data like average household size, marital status, would correlate with political preferences. Economic data like unemployment rates, median household income, poverty levels, and economic growth, would factor a large part. Other aspects of living like healthcare, education, immigration, and environmental concerns, will also affect one’s choice in political party.

## 7 R Code for Final Model

```
1 library(tidyverse)
2 library(tidymodels)
3 library(doParallel)
4 library(xgboost)
5
6 # Read the training data
7 train <- read_csv("train.csv")
8 train <- train %>% select(-name)
9
10 # Define the boosted tree model with tuning hyperparameters
11
12 boost_model <-
13   boost_tree(
14     # Number of variables to consider at each split
15     mtry = tune(),
16     # Number of boosting iterations (trees)
17     trees = tune(),
18     # Minimum number of observations in terminal nodes
19     min_n = tune(),
20     # Maximum depth of the trees
21     tree_depth = tune(),
22     # Learning rate for each tree
23     learn_rate = tune(),
24     # Minimum loss reduction required to make a split
25     loss_reduction = tune(),
26     # Fraction of training data to sample for each tree
27     sample_size = tune(),
28     # Stopping condition: stop at 5 iterations
29     stop_iter = c(5)
30   ) %>%
31   # Set the mode of the model to regression
32   set_mode("regression") %>%
33   # Set the underlying engine to xgboost
34   set_engine("xgboost")
35
36 # Set the seed and define the recipe
37 set.seed(1)
38 # Create a recipe for data preprocessing
39 boost_recipe <-
40   recipe(percent_dem ~ ., data = train) %>%
41   # Remove the id variable from the dataset
42   step_rm(id) %>%
43   # Create a new variable pct_minority combine all the minority race population
44   # together and divide by total population
```

```

44  step_mutate(pct_minority = (x0038e + x0039e + x0044e + x0052e + x0057e ) /
      x0001e) %>%
45  # Create a new variable pct_white by divide total white population to total
      population
46  step_mutate(pct_white = x0037e / x0001e) %>%
47  # Create a new variable pct_minor by divide total minor population to total
      population
48  step_mutate(pct_minor = x0019e / x0001e) %>%
49  # Create a new variable pct_male by divide total male to total population
50  step_mutate(pct_male = x0088e / x0001e) %>%
51  # Create a new variable pct_female by divide total female to total population
52  step_mutate(pct_female = x0089e / x0001e) %>%
53  # Create a new variable pct_college by combine all population with higher
      education together and divide by total population
54  step_mutate(pct_college = (c01_004e + c01_005e + c01_011e + c01_015e + c01_018e
      + c01_021e + c01_024e + c01_027e) / x0001e) %>%
55  # Create a new variable pct_old by combine all population who age is above 45
      years old, and divide by total population
56  step_mutate(pct_old = (x0012e + x0013e + x0014e + x0015e + x0016e + x0017e) /
      x0001e) %>%
57  # Impute missing values in column gdp in 2016 to 2020 by mean
58  step_impute_mean(income_per_cap_2016:gdp_2020) %>%
59  # Normalize all predictor variables
60  step_normalize(all_predictors())
61
62 # Create the workflow
63 boost_wf <-
64   workflow() %>%
65   add_recipe(boost_recipe) %>%
66   add_model(boost_model)
67
68 # Set the seed and define the folds
69 set.seed(1)
70 boost_folds <- vfold_cv(train, v = 10, strata = percent_dem)
71
72 # Define the hyperparameter grid
73 set.seed(1)
74 boost_grid <- grid_latin_hypercube(
75   mtry(range = c(35, 80)), # number of predictors randomly sampled at each split
      when creating the tree models higher mtry less rmse, but careful for
      overfitting
76
77   min_n(range = c(10, 20)), # minimum number of data points in a node that is
      required for the node to be split further
78
79   trees(range = c(300, 600)), # number of trees, higher trees less rmse, but
      careful for overfitting

```

```

80
81   tree_depth(range = c(10, 20)), # maximum depth of the tree (i.e. number of
      splits)
82   learn_rate(range = c(-5, -1)), # rate at which the boosting algorithm adapts
83
84   loss_reduction(range = c(-5, -1)), # reduction in the loss function which can
      use in further split
85
86   sample_size = sample_prop(), # proportion of data that is use to the fitting
87
88   size = 50 # number for trees that fit in model's ensemble
89 )
90
91 # Register parallel processing which can make it faster
92 doParallel::registerDoParallel()
93 set.seed(1)
94 boost_res <- tune_grid(
95   boost_wf,
96   resamples = boost_folds,
97   grid = boost_grid,
98   control = control_grid(save_pred = TRUE)
99 )
100
101 #Update the model with the best rmse
102 boost_model <- update(boost_model, parameters = select_best(boost_res, "rmse")[,
      1:7])
103
104 # Select the best parameters
105 boost_wf <-
106   workflow() %>%
107   add_recipe(boost_recipe) %>%
108   add_model(boost_model)
109
110 # Fit the model to cross-validation folds
111 boost_crossval_fit <- boost_wf %>%
112   fit_resamples(resamples = boost_folds)
113
114 # Collect and display metrics
115 boost_crossval_fit %>%
116   collect_metrics() %>%
117   filter(.metric == "rmse")
118
119 # Read the test data
120 test <- read_csv("test.csv")
121
122 # Set the seed and fit the final model
123 set.seed(1)

```



```
124 boost_fit <-
125   boost_wf %>%
126   fit(data = train)
127
128 # Make predictions
129 boost_predictions <- boost_fit %>%
130   predict(test) %>%
131   cbind(test %>% select(id))
132
133 # Write predictions to a CSV file
134 write_csv(x = boost_predictions %>% rename(percent_dem = .pred),
135           file = "boost_predictions.csv")
```