Team 10 Regression Project Report

1. Introduction

In this project, we aim to choose the best model that most efficiently describes the relationship between the 124 predictor variables and the response variable, predicting the percentage of voters in a county that voted for President Biden in the 2020 US Presidential Election. Based on the 26th Amendment, we may believe that the population of 18 years and over to be associated with the response variable. Also the 2020 Presidential Election Census (Census.gov) report states that voter turnout was highest among those ages 65 to 74 at 76.0%, while lowest among those ages 18 to 24 at 51.4%. We believe that there is an association between the age group variables and the response variable. We will investigate the relationship between variables further in the report.

2. Exploratory Data Analysis

2.1 The relationship between the predictor variables

2.1.1 Relationship with the total population

We used a tree diagram to investigate the relationship between variables that have a relationship with the total population.

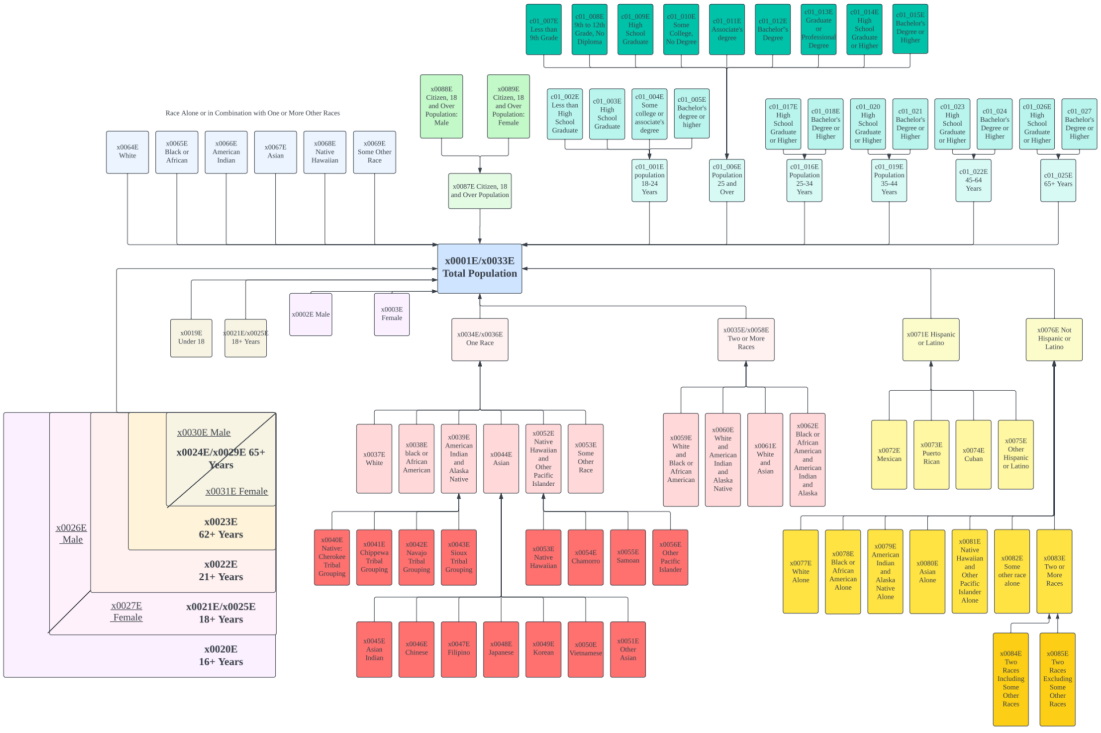


Figure 1: Tree diagram Showing the relationship between predictor variables. From the tree diagram, we can see that the total population, the root, can be divided into some groups of nodes, which can be further divided into several leaf nodes. For example, the total population can be divided into an one-race group and a two-or-more-races group. Then one race group can be divided into 6 leaf groups of different races, which can be further divided into more specific leaf nodes of races.

2.1.2 Relationship between Race and Education level

We used a correlation plot to investigate the relationship between races and also between the race and the education level.

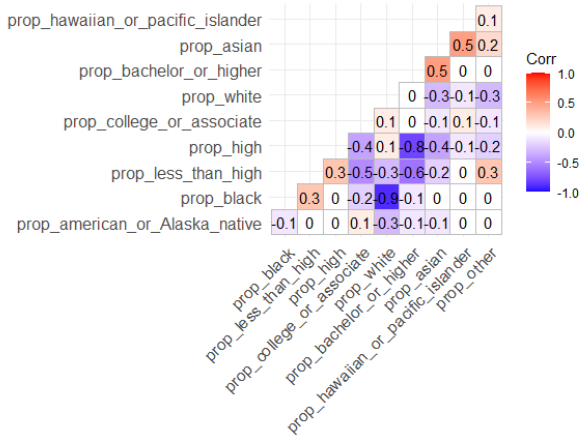


Figure 2: Correlation plot Showing the relationship between race and education level. From this plot, we can see that there is a very strong negative correlation between Black and White. Also it can be seen that the proportion of Asians and high level of education have positive correlation.

2.1.3 Relationship between Income, Education level, and urban level.

We used a scatter plot to investigate the relationship between Income, Education, and urban level. 1 is most urban and 6 is most rural.

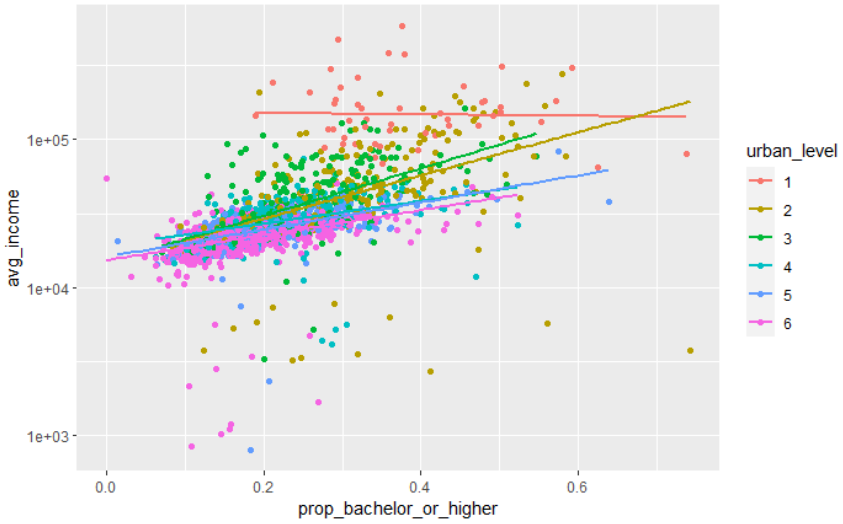


Figure 3: Scatter plot Showing the relationship between Income, Education, and urban level. From this plot, we can see that the closer to urban the higher the income, and the closer to rural, the lower the income. In addition, it can be seen that the income increases as the proportion of high education levels increases.

2.2 The relationship between response variable and predictor variables

2.2.1 The urban level

We used a boxplot to investigate the relationship between the percentage of voters who voted Biden and the urban level. 1 is most urban and 6 is most rural.

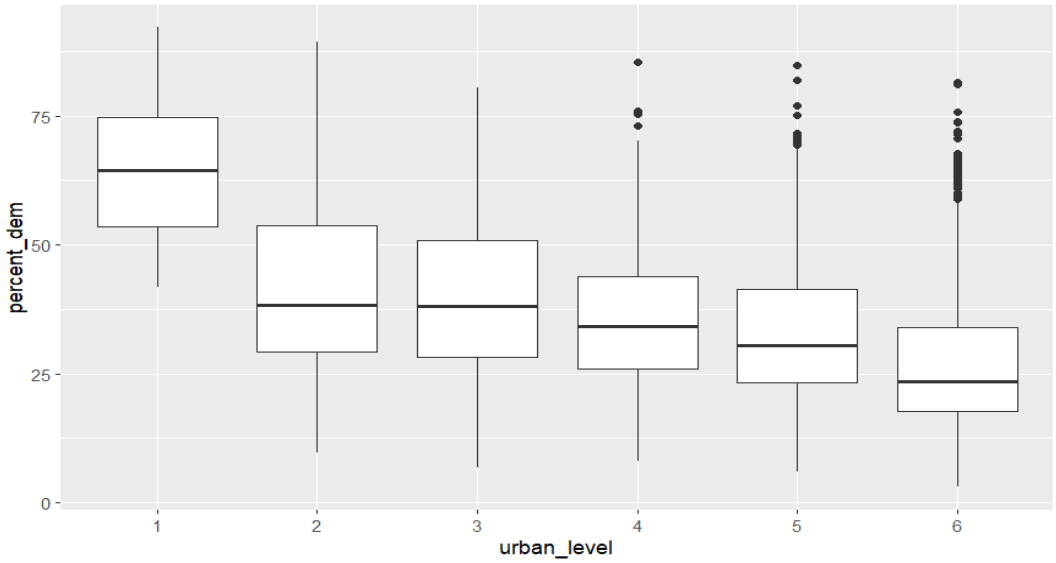


Figure 4: Box plot Comparing the percentage of voters who voted for Biden between urban_level. From this boxplot, we can see that the percentage of voters who voted for Biden decreases as the area gets closer to the rural. We also notice that this variable x2013_code is a categorical variable. We will convert it to a factor.

2.2.2 The Income

We used a scatter plot to investigate the relationship between the percentage of voters who voted Biden and the income from 2016 to 2020.

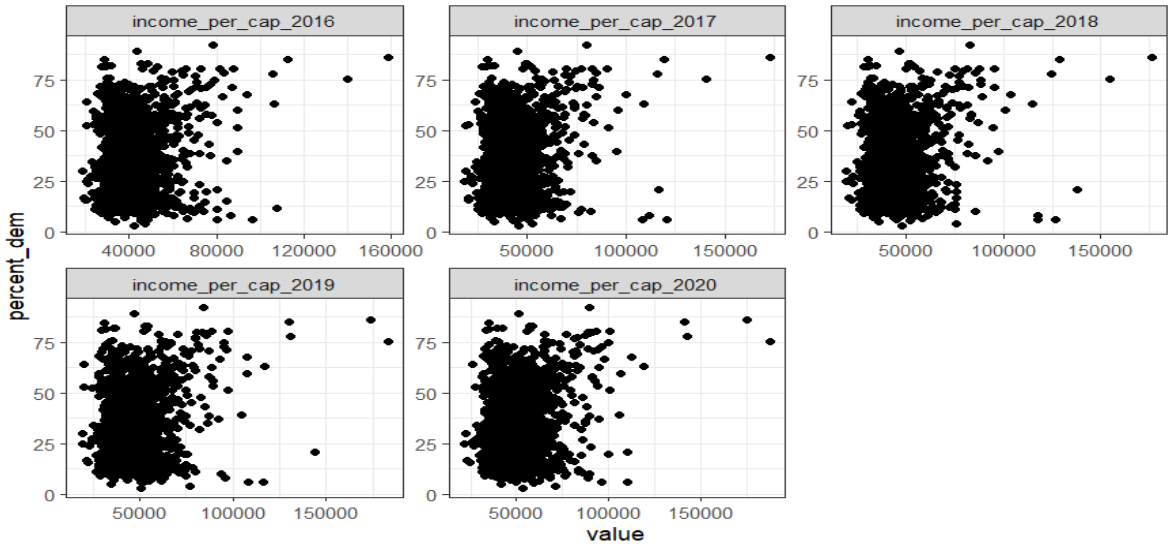


Figure 5: Scatter plot Comparing the percentage of voters who voted for Biden between Income from 2016 to 2020. From this scatter plot, we can see that the income data are mostly clustered below the 100000 threshold with only a few points above the 100000 threshold. There does not exist any apparent mathematical relationship between income and percent_dem.

The relationship between the percentage of voters who voted Biden and the average income from 2016 to 2020. The income distribution is clustered to the left, so for visibility, we converted it to log10.

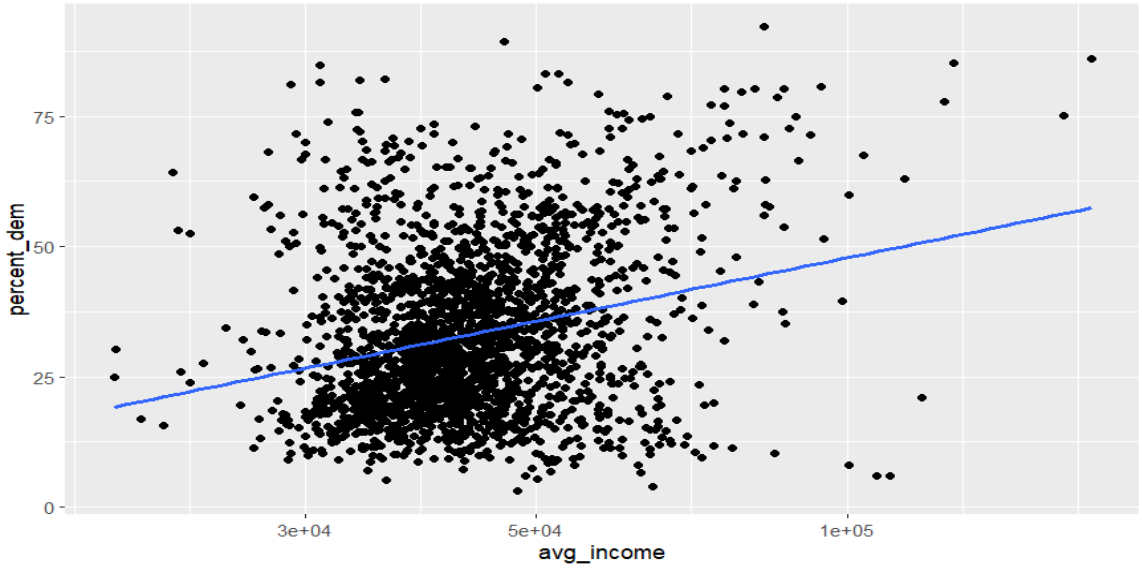


Figure 6: Scatter plot Comparing the percentage of voters who voted for Biden between average Income. From this scatter plot, we can see that the higher the average income, the higher the percentage of voters who voted for Biden. We can also identify some potential outliers from this scatter plot.

2.2.3 The GDP

We used a scatter plot to investigate the relationship between the percentage of voters who voted Biden and the GDP from 2016 to 2020.

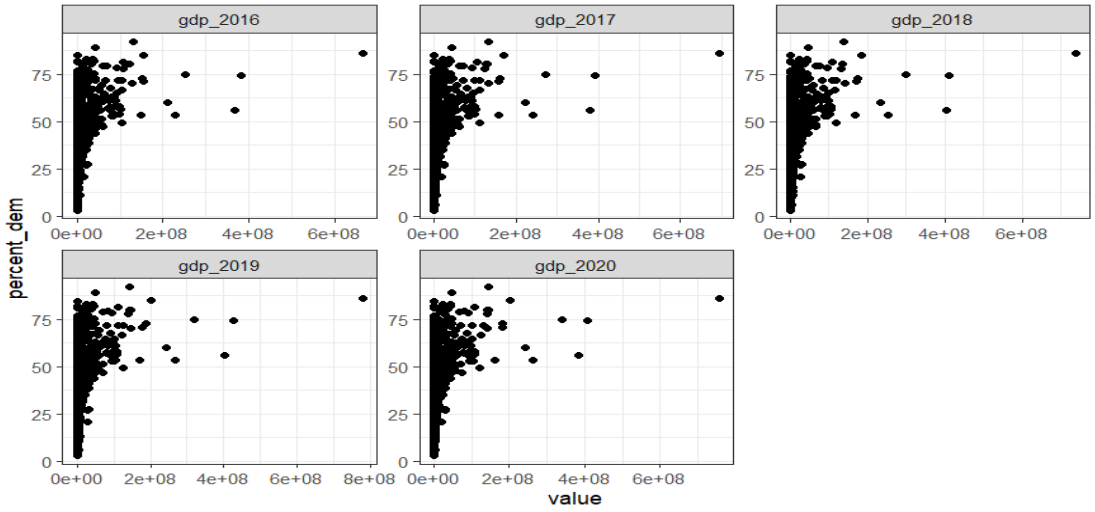


Figure 7: Scatter plot Comparing the percentage of voters who voted for Biden between GDP from 2016 to 2020. From this scatter plot, we can see that the gdp distribution is clustered to the left. Most of the data points are below the 10^8 threshold. Only a few data points pass this threshold. Since there does not exist any mathematical relationship between these variables, we will apply transformation.

The GDP distribution is clustered to the left, so for visibility, we applied log transformation to the variable.

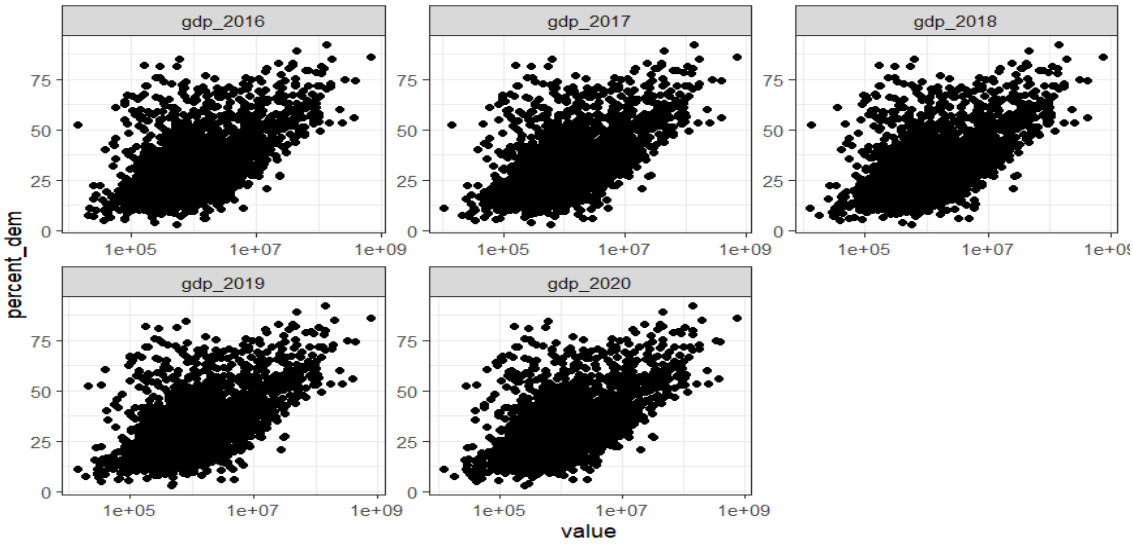


Figure 8: Scatter plot Comparing the percentage of voters who voted for Biden between GDP from 2016 to 2020 (rescale.) After log transformation, we can see that there is a decent positive linear relationship between the gdp variables and percent dem. Thus, we can conclude that gdp variables are rather important in our prediction task.

The relationship between the percentage of voters who voted Biden and the average GDP from 2016 to 2020.

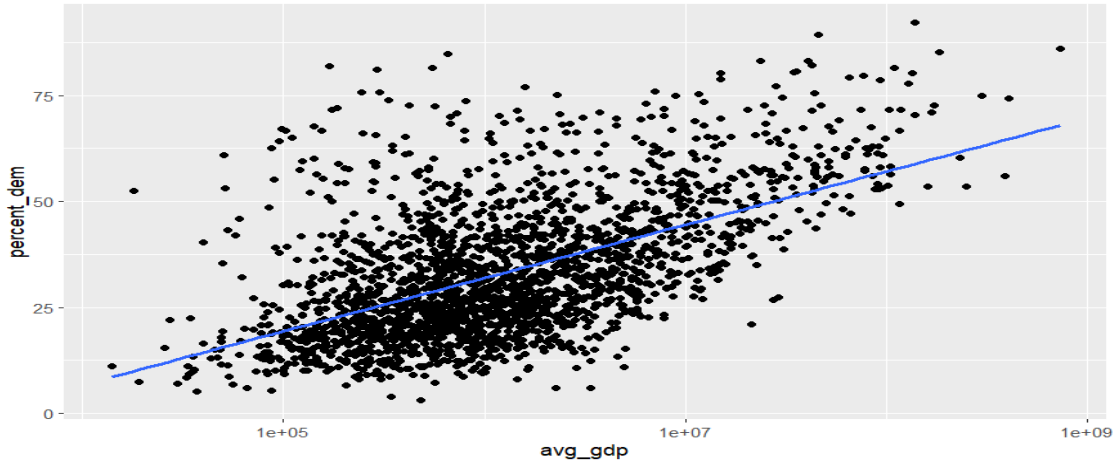


Figure 9: Scatter plot Comparing the percentage of voters who voted for Biden between GDP. From this scatter plot, we can see that the higher the average GDP, the higher the percentage of voters who voted for Biden.

2.2.4 The Age

We used a scatter plot to investigate the relationship between the percentage of voters who voted Biden and the proportion of age groups.

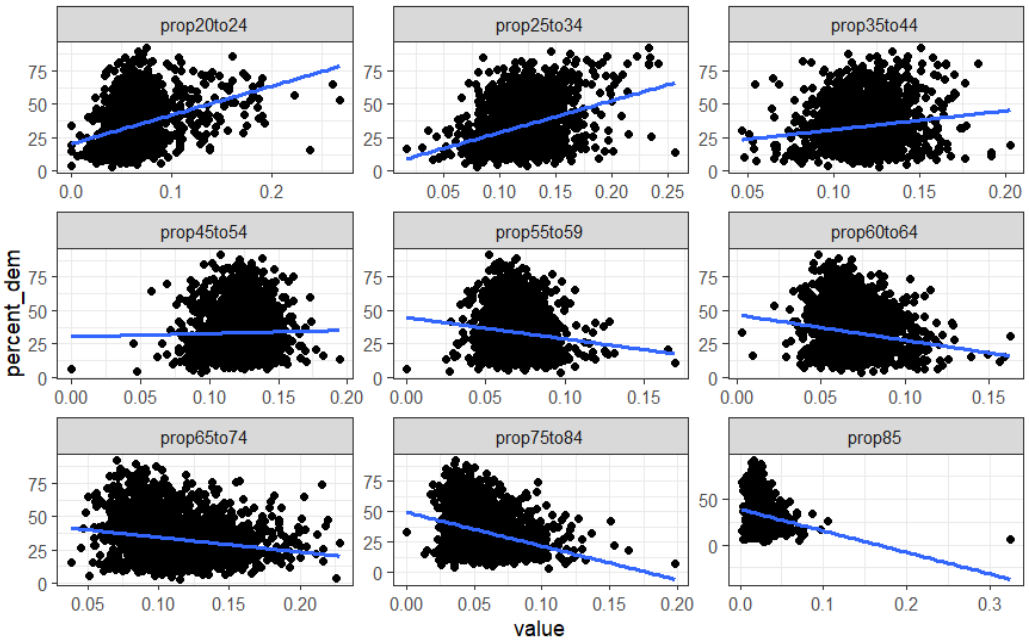


Figure 10: Scatter plot Comparing the percentage of voters who voted for Biden between proportions of age. From this scatter plot, we can see that the higher the proportion of young people, the more voters voted for Biden. We can also identify significant outliers in this plot.

2.2.5 The Education level

We used a scatter plot to investigate the relationship between the percentage of voters who voted Biden and the proportion of education level.

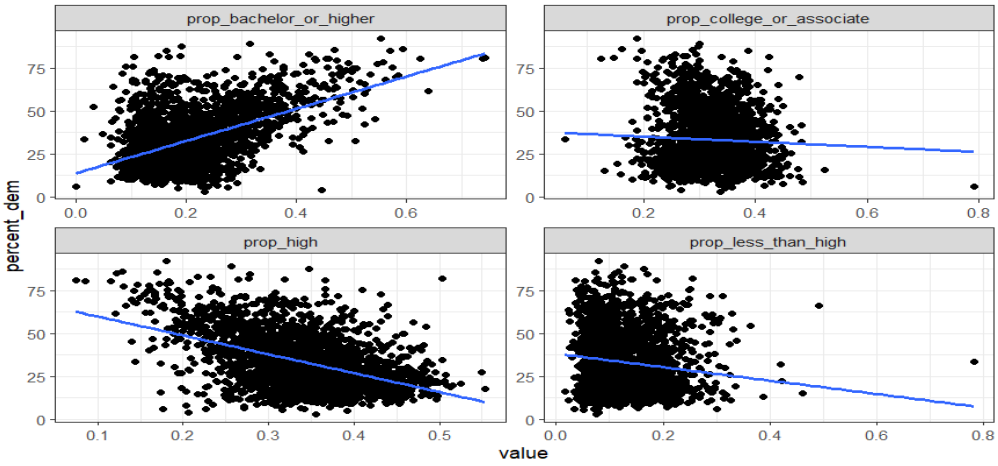


Figure 11: Scatter plot Comparing the percentage of voters who voted for Biden between proportions of education level. From this scatter plot, we can see that the higher the proportion of bachelor's degree and above, the more they voted for Biden, while the higher the proportion of high school graduates, the less they voted for Biden. We can also identify significant outliers in this plot.

2.2.6 The Race

We used a scatter plot to investigate the relationship between the percentage of voters who voted Biden and the proportion of race groups.

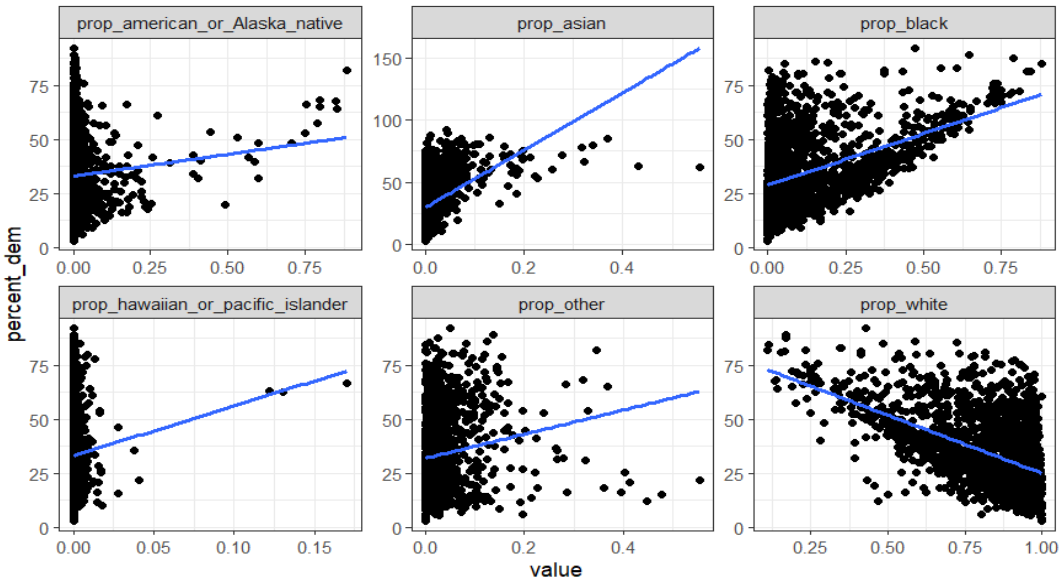


Figure 12: Scatter plot Comparing the percentage of voters who voted for Biden between the races. From these scatter plots, we can see that most of the data points are either clustered to the left and right.

Even though we can plot least squares regression lines and see a decent relationship between the predictor and response variables. The least squares regression lines do not summarize the trends in the data points.

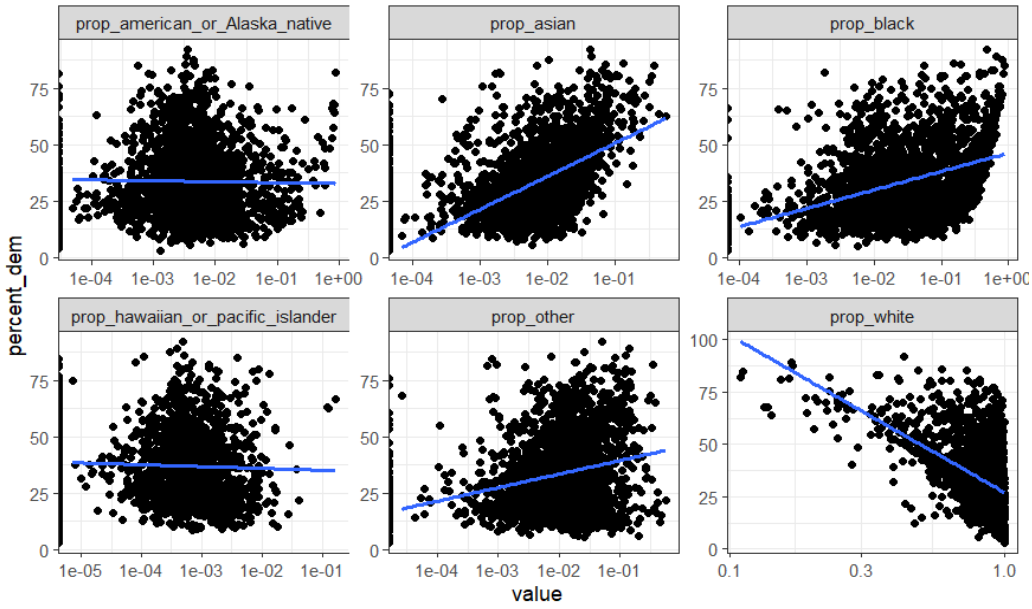


Figure 13: Scatter plot Comparing the percentage of voters who voted for Biden between race groups (rescale.) The higher the proportion of Asians and blacks, the more voters voted for Biden, while the higher the proportion of Whites, the less voters voted for Biden.

3. Preprocessing/Recipes

3.1 Recipe 1

After analyzing the predictor variables, we separated the variables into several groups. We identified most of the predictor variables as related to the total population. Another group of predictor variables are income and gdp variables, which are not directly related to total population. For the group that we considered to be related to total population, we created a tree diagram, further specifying the relationship between each variable and total population variable.

We dropped the variables x0033e, x0036e, x0058e, x0025e, and x0029e because they are duplicated variables. We dropped redundant variables x0005e, x0006e, x0007e, x0008e, x0009e, x0010e, x0011e, x0012e, x0013e, x0014e, x0015e, x0016e, and x0017e (age group variables) because we know that only the proportion of 18 years and older can vote in presidential election and these variables are repeated in x0020e, x0021e, x0022e, x0023e, x0024e, x0025e, x0026e, x0027e, and x0029e (other age group variables). Thus, we simplified the age group variables to x0019e (under 18 years old) and x0021e/x0025e (above 18 years old) and preserved the predictor variables x0020e, x0021e, x0022e, x0023e, x0024e, x0025e, x0026e, x0027e, and x0029e.

Total population is the root, and other predictor variables are parent nodes and leaf nodes in different cases. For example, we further realized that the total population can be separated into

one-race group and two-or-more-races groups. In one race group x0034e, there are races x0037e, x0038e, x0039e, x0044 , x0052e, and x0053e. Then x0039e can be partitioned into x0040e, x0041e, x0042e, and x0043e. This process is repeated for other variables as well until no other leaf nodes can be divided.

In the recipe, we used the bottom to top approach, we divided each leaf node by its parent node. We avoided the top from bottom approach because we might convert a parent node into percentage. This will lead to error as we continue to divide leaf nodes by this parent node, which the leaf node is divided by a percentage. We also did not divide each predictor variable directly by the total population because data varies in each county. If the number of people of a leaf node is small while the total population is extremely large, we are unable to capture the small changes. If the value after division is NaN, then we replace the NaN value by 0. This happens because the denominator is 0 and the numerator is 0. In this case, it makes sense to convert the percentage 0.

For the rest of the predictor variables including income per capita and GDP, we connected them to x0001e (total population) and x2013_code (rural degree). After performing correlation calculation, we realized that Gdp ~ x0001e + x2013_code results in an R-squared value of around 0.7, which is a strong positive correlation. Income ~ x0001e + x2013_code results in an R-squared value of around 0.2, which is weak correlation. However Income most correlated with c01_024e. Nonetheless, we empirically found that regressing Income on x0001e + x2013_code will produce better results in cross validation. We will use the impute linear method with predictor variables x0001e + x2013_code to impute missing values in income and gdp.

As mentioned in the Exploratory Data Analysis section, the income and gdp variables are clustered and not ideal for machine learning algorithms to learn. Thus, we apply normalization, which converts the data points to a normal distribution with mean 0 and standard deviation 1. The transformed data points are comparatively more spread out, allowing our machine learning algorithms to learn from a wide range of data. If we feed original income and gdp data into our machine learning algorithms, our algorithms may only be able to learn the trend when income and gdp are small and fail when predicting data when income and gdp data are large.

Finally, we convert the variable x2013_code to a categorical variable. However, the Xgboost algorithm does not support categorical variables as input. In this case, we do not change x2013_code.

3.2 Recipe 2

Recipe 2 is similar to Recipe 1 except that we used impute means to replace missing values in income and gdp variables.

3.3 Recipe 3

In Recipe 3, we impute missing values using mean and convert variables to percentage similar to Recipe 1. We also divide total votes by total population. We propose that the NaN values resulting from division by zero when converting variables to percentages should not be converted to 0. Instead, we keep the NaN values as it is, since the Xgboost algorithm accepts NA values as input. We are uncertain if this approach is better, but using ensemble methods, explained in the next section, we can add penalty terms and find the best combination of these three recipes.

3.4 Recipe for Ensemble

We empirically found out that by combining two Xgboost models with different hyperparameters and Recipe 1, Recipe 2, and Recipe 3 to produce better results. This will be our recipe for the ensemble approach.

4. Candidate models/Model evaluation/Tuning

4.1 Candidate models with default hyperparameters

To predict the percentage of voters who voted for Biden, we built six models: Multiple linear regression, Polynomial support vector machine, Decision tree, Knn, Random forest, and Xgboost. We first trained the models with default hyperparameters to compare performance of each model and later compare the performance with tuned models.

4.1.1 Candidate Model Description

- Decision tree
  Decision tree is a type of model that splits the data into subsets based on asking several questions. Data will be further splitted as more questions are asked. As we move down the tree, the data are splitted into more precise subsets. This process continues until we reach leaf nodes with predicted outcomes. Decision trees can be used for both regression and classification.

- K-Nearest Neighbors (KNN)
  KNN finds closest K examples in the training data set. Then for regression, it predicts the mean of the closet K examples. For classification, it predicts the most common output value among these closest K examples.

- Multiple linear regression
  Multiple linear regression is a statistical method that models the relationship between predictor variables and response variables. It fits a linear relationship equation to the data.

The process is similar to a simple linear regression model, but the only difference is that multiple linear regression considers more predictor variables.

- Polynomial support vector machine
  Polynomial support vector machine is a variation of a support vector machine. It uses a polynomial kernel to map the original data into a higher dimensional space. A linear separator is used to classify the data. Compared to a linear support vector machine, a polynomial support vector machine is able to capture more complex relationships of the data.

- Random forest
  Random forest method constructs multiple decision trees. It is able to handle high dimensional and large data sets with high accuracy. It is also to handle missing values.

- Xgboost
  Xgboost boosts weak learners such as decision trees to produce a strong predictive model. It builds decision trees sequentially and each tree tries to correct errors made by previous trees. In practice, we can tune hyperparameters such as shrinkage, max_depth and number of trees to build a more predictive model.

- Multiple linear regression regularization
  In addition to multiple linear regression, multiple linear regression regularization adds penalty terms to the parameters of the model in order to reduce the degree of freedom and prevent the issue of overfitting. There are two types of regularization: lasso and ridge regularization. Mixture argument of 0 specifies ridge regularization. Mixture argument of 1 specifies lasso regularization.

4.1.2 Summary of baseline models (without tuning)

	Type of model	Engine	Recipe	Hyperparameters (default)
Candidate model 1	Decision tree	rpart	Recipe 1 (with x2013_code converted to categorical)	Cost complexity Min_n Tree depth
Candidate model 2	Knn	kknn	Recipe 1 (with x2013_code converted to categorical)	Neighbors Weight_func Dist_power
Candidate model 3	Multiple linear regression	lm	Recipe 1 (with x2013_code converted to categorical)	None
Candidate model 4	Polynomial support vector machine	kernlab	Recipe 1 (with x2013_code converted to categorical)	Cost Degree Scale_factor Margin
Candidate model 5	Random forest	ranger	Recipe 1 (with x2013_code converted to categorical)	Trees Min_n Mtry
Candidate model 6	Xgboost	xgboost	Recipe 1 (without x2013_code converted to categorical)	Trees Tree_depth Learn rate Loss_reduction Sample_size Mtry Min_n Stop iter

Table1: Summary of baseline models. The same recipe and default hyperparameters were applied to all models. Candidate Model 3 is also the baseline for multiple linear regression with regularization.

4.1.3 Summary of performance of baseline models using 10-fold cross-validation

	Metric	Score	Std_Error
Candidate Model 1: Decision tree	rmse	9.459781	0.1185024
Candidate Model 2: Knn	rmse	8.621471	0.1695749
Candidate Model 3: Multiple linear regression	rmse	7.935698	0.1506193
Candidate Model 4: Polynomial support vector machine	rmse	7.987706	0.1421599
Candidate Model 5: Random forest	rmse	7.263876	0.105131
Candidate Model 6: Xgboost	rmse	7.039952	0.1507456

Table2: Summary of performance of baseline models. The rmse score was measured using a 10-fold cross-validation. From this table, we can see that the Xgboost model shows the best performance, while the decision tree model shows the worst performance. These results are our baselines. We attempt to search for better hyperparameters that outperforms their respective baselines.

4.1.4 Bar chart: Performance comparison of models

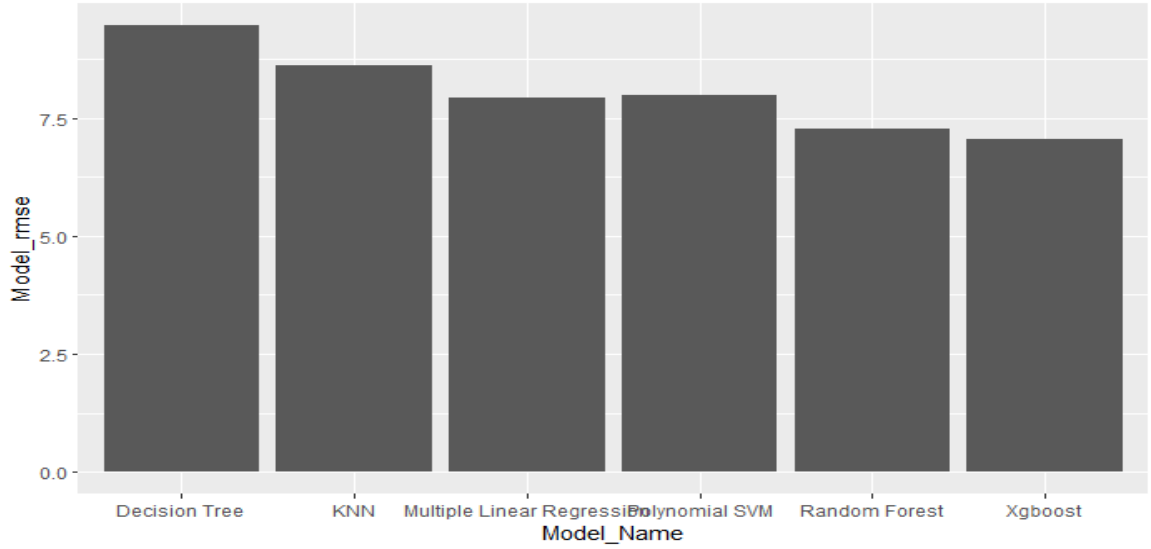


Figure 1: Bar chart Comparing the performance of baseline models using rmse. We can see that the Xgboost model has the lowest rmse score(highest performance) and the Decision tree model has the highest rmse score(lowest performance.)

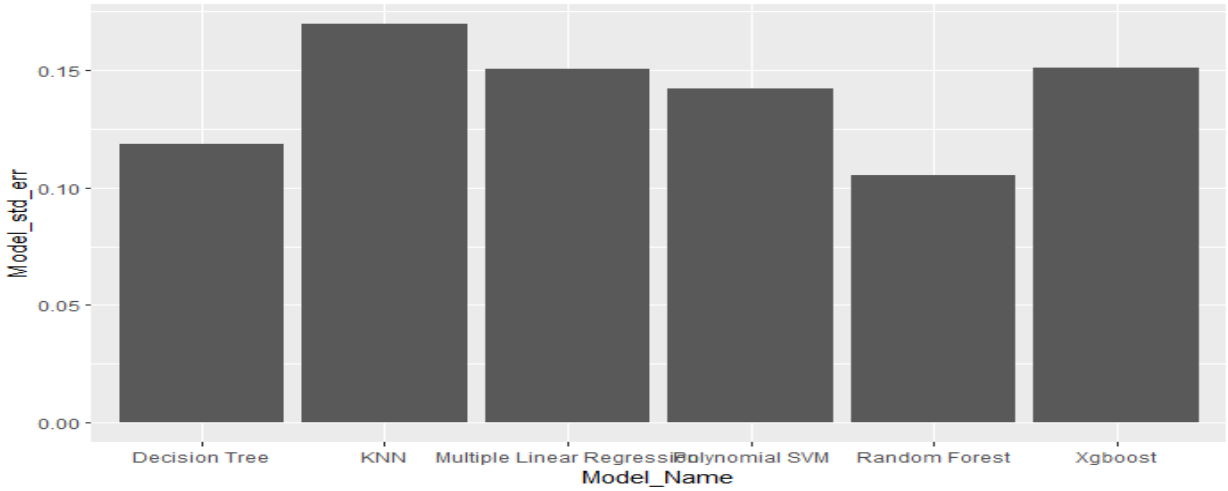


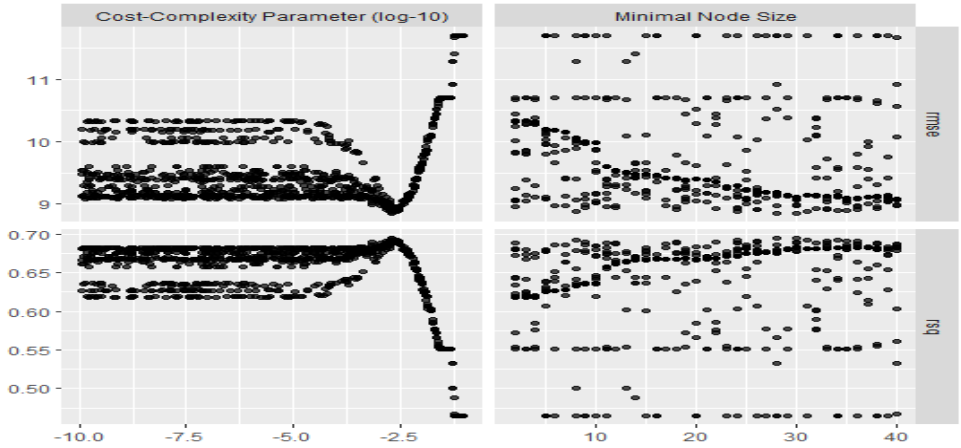
Figure 2: Bar chart Comparing the performance of baseline models using standard error. We can see that the Random forest model has the lowest standard error score and the Knn model has the highest standard error score.

4.2 Candidate models with tuning

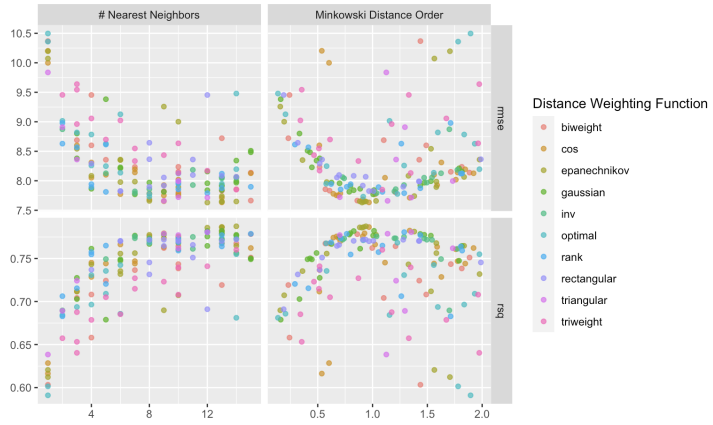
We tuned each model to find the best performing hyperparameters in predicting the percentage of voters who voted for Biden. The same recipe was used for all models, and the performance of each tuned model was measured using 10 fold cross validation.

4.2.1 Tuning of the hyperparameters for candidate models

- Decision tree
  We tuned the cost complexity and the min_n hyperparameters with grid size 1000 and used the default value for the tree depth hyperparameter. After tuning, this model showed best performance (rmse score: 8.84) when cost complexity is 0.00197 and min n is 28.

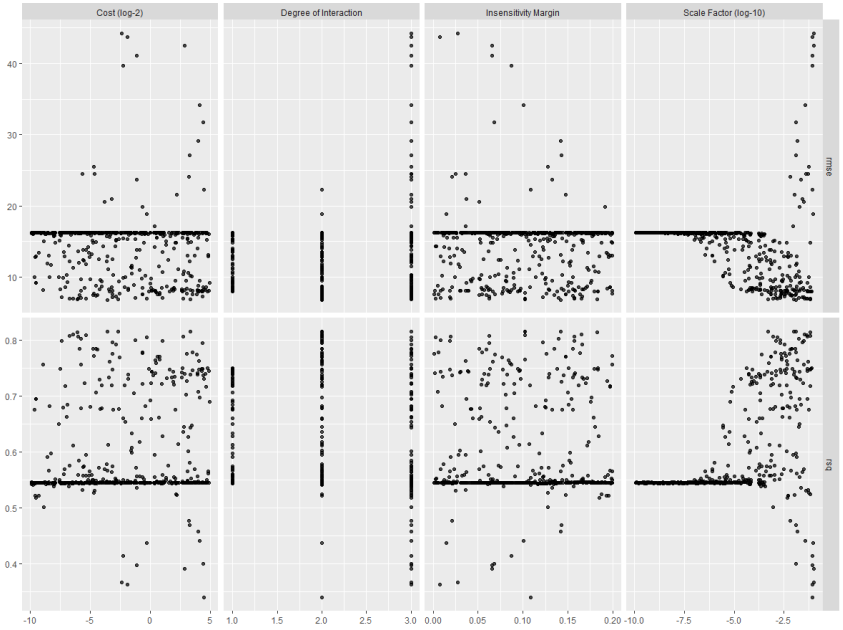


- Knn
  We tuned all hyperparameters of the K-nearest neighbors model: neighbors, weight_func, dist_power. After tuning, this model showed best performance (rmse score: 7.87) when neighbors is 6, weight_func is optimal, and dist_power is 0.886.

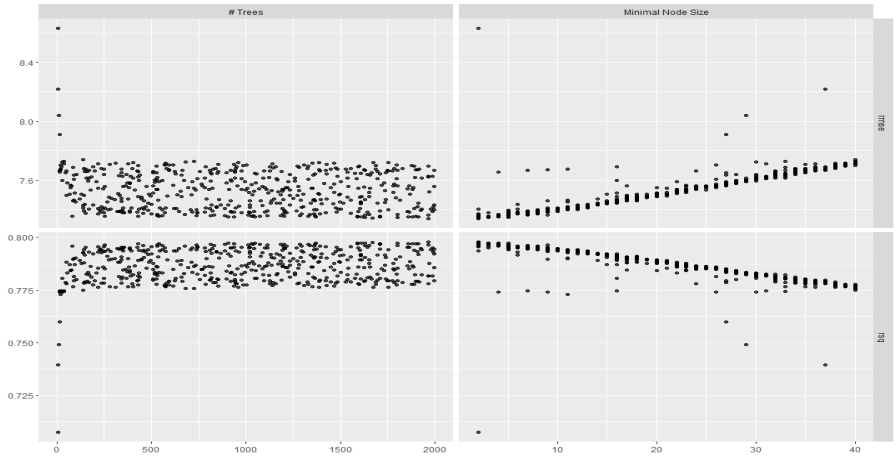


- Multiple linear regression
  Since the linear regression model does not have hyperparameters that can be tuned, we did not tune this model.

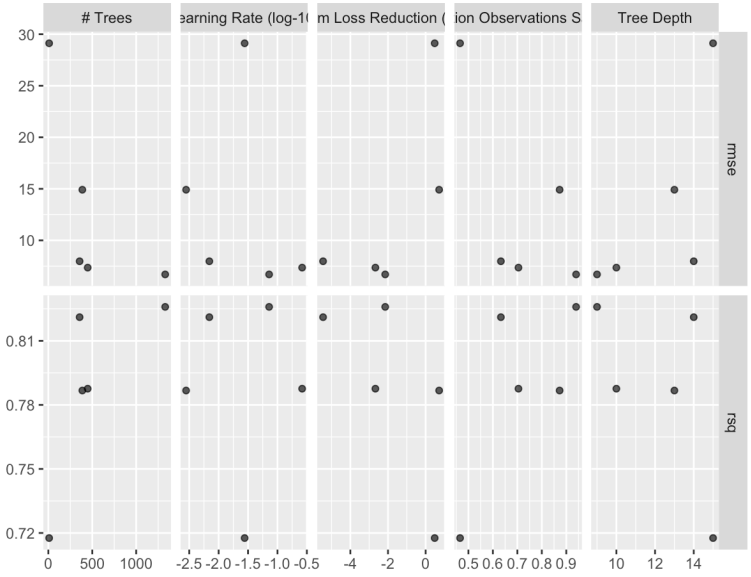
- Polynomial support vector machine
  We tuned all hyperparameters of the Polynomial support vector machine model: cost, degree, scale_factor and margin with grid size 500. After tuning, this model showed best performance (rmse score: 6.84) when cost is 0.0863, degree is 2 ,scale_factor is 0.0127, and margin is 0.142.



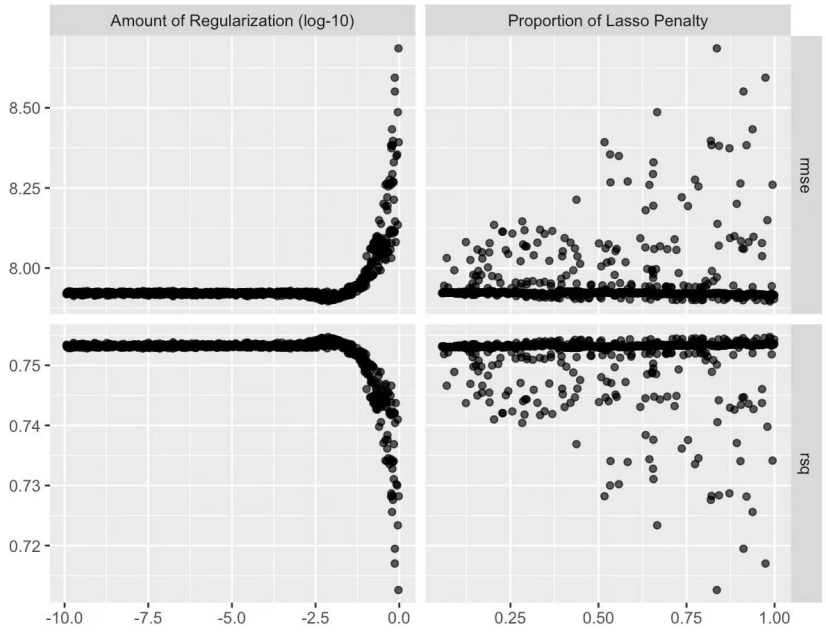
- Random forest
  We tuned the trees and the min_n hyperparameters with grid size 500 and used the default value for the mtry hyperparameter. After tuning, this model showed best performance (rmse score: 7.34) when trees is 1965 and min_n is 2.



- Xgboost
  We tuned 5 hyperparameters of the Xgboost model: trees, tree_depth, learn rate, loss_reduction, and sample_size with grid size 300 and used default value for 3 remaining hyperparameters: mtry, min_n and stop iter. After tuning, this model showed best performance (rmse score: 6.16) when trees is 1891, tree_depth is 7, learn rate is 0.01, loss_reduction is 2.78e-9, and sample_size is 0.264. Instead of 300 tuning points, we only displayed 5 parameters on our plot since a large amount of parameters caused memory overflow in the R studio.



- Multiple linear regression regularization
  We tuned the penalty and the mixture hyperparameters with grid size 10000. After tuning, this model showed best performance (rmse score: 7.9) when penalty is 0.00744 and mixture is 0.987.



4.2.2 Summary of Tuned Models

	Type of model	Engine	Recipe	Hyperparameters
Candidate model 1	Decision tree	rpart	Recipe 1 (with x2013_code converted to categorical)	Cost complexity:0.00197 Min_n: 28 Tree depth: default
Candidate model 2	Knn	kknn	Recipe 1 (with x2013_code converted to categorical)	Neighbors: 6 Weight_func: default Dist_power: 0.886.
Candidate model 3	Multiple linear regression	lm	Recipe 1 (with x2013_code converted to categorical)	None
Candidate model 4	Polynomial support vector machine	kernlab	Recipe 1 (with x2013_code converted to categorical)	Cost: 0.0863 Degree: 2 Scale_factor: 0.0127 Margin: 0.142
Candidate model 5	Random forest	ranger	Recipe 1 (with x2013_code converted to categorical)	Trees: 1965 Min_n: 2 Mtry: default
Candidate model 6	Xgboost	xgboost	Recipe 1 (without x2013_code converted to categorical)	Trees: 1891 Tree_depth: 7 Learn rate: 0.01 Loss_reduction: 2.78e-9 Sample_size: 0.264 Mtry: default Min_n: default Stop iter: default
Candidate model 7	Multiple linear regression regularization	glmnet	Recipe 1 (with x2013_code converted to categorical)	Penalty: 0.00744 Mixture: 0.987

Table3: Summary of tuned models. The same recipe and tuned hyperparameters were applied to all models.

4.2.3 Summary of best performance of tuned models using 10-fold cross-validation

	Metric	Score	Std_Error
Candidate Model 1: Decision tree	rmse	8.84	0.176
Candidate Model 2: Knn	rmse	7.87	0.127
Candidate Model 3: Multiple linear regression	rmse	.	.
Candidate Model 4: Polynomial support vector machine	rmse	6.84	0.115
Candidate Model 5: Random forest	rmse	7.34	0.177
Candidate Model 6: Xgboost	rmse	6.16	0.108
Candidate Model 7: Multiple linear regression with regularization	rmse	7.90	0.167

Table4: Summary of performance of tuned models.

4.2.4 Bar chart: Performance comparison of Tuned models

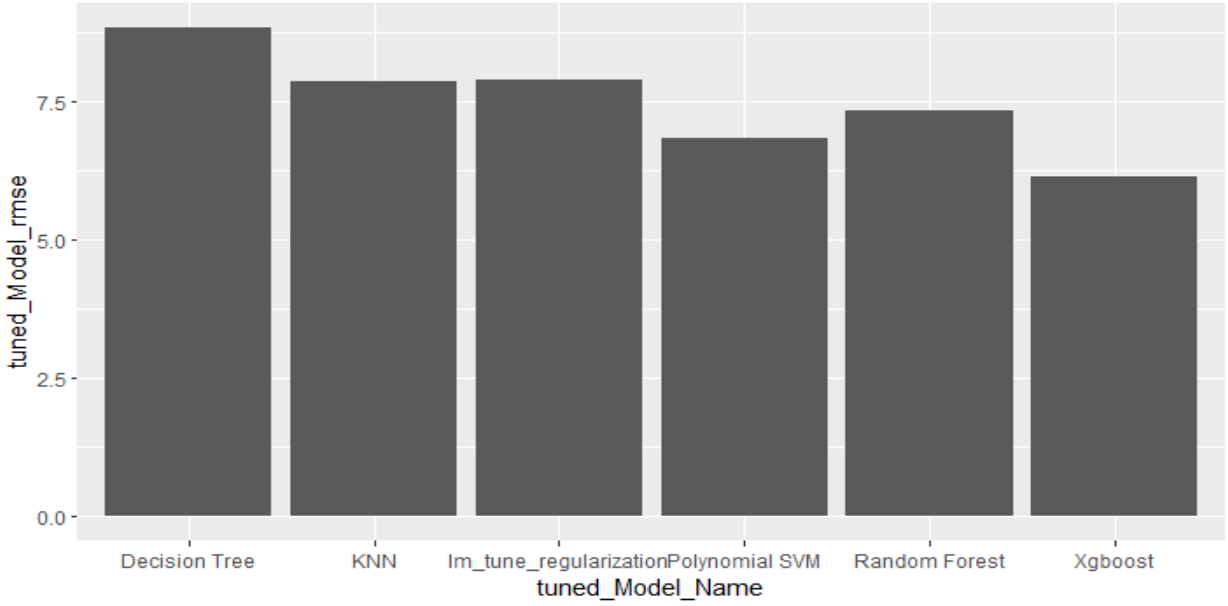


Figure 3: Bar chart Comparing the performance of tuned models using rmse. We can see that the Xgboost model has the lowest rmse score(highest performance) and the Decision tree model has the highest rmse score(lowest performance.)

4.2.5 Performance Comparison of Baseline and Tuned Models

	Type	Metric	Score	Standard error
Candidate model 1 Decision Tree	Baseline	rmse	9.459781	0.1185024
	Tuned	rmse	8.84	0.176
Candidate model 2 Knn	Baseline	rmse	8.621471	0.1695749
	Tuned	rmse	7.87	0.127
Candidate model 3 Multiple Linear Regression	Baseline	rmse	7.935698	0.1506193
	Tuned	rmse	.	.
Candidate model 4 Polynomial Support Vector Machine	Baseline	rmse	7.987706	0.1421599
	Tuned	rmse	6.84	0.115
Candidate model 5 Random Forest	Baseline	rmse	7.263876	0.105131
	Tuned	rmse	7.34	0.177
Candidate model 6 Xgboost	Baseline	rmse	7.039952	0.1507456
	Tuned	rmse	6.16	0.108
Candidate model 7 Multiple Linear Regression with Regularization	Baseline	rmse	.	.
	Tuned	rmse	7.90	0.167

Table 5: Performance Comparison of Baseline and Tuned Models

5. Discussion of Final Model

5.1 Conclusion

Based on the performance comparison of baseline and tuned models table, we can see that candidate model 6, which is Xgboost model after tuning, produces the lowest score of 6.16 and standard error of 0.108.

Strengths:

1. High performance: Xgboost achieves higher accuracy with lower standard error compared to all other candidate models.
2. High flexibility: Xgboost can handle missing data and supports regression, classification, and ranking tasks.

3. Tuning parameters: Xgboost provides numerous parameters for tuning in order to achieve a more optimized model

Weaknesses:

1. Problem of overfitting: because of the high flexibility and complexity of the model, Xgboost might have problems with overfitting, especially when the variables are not tuned correctly.
2. Long training time: training an Xgboost model can take a long time, especially with large datasets and complex models.

To achieve a better score with our model, we will use the Ensemble method to combine two Xgboost models with two sets of hyperparameters and three recipes. Combination of models and recipes are crossed.

Model	Recipe
Xgboost_1 trees = 1507L, min_n = 14L, tree_depth = 15L, learn_rate = 0.020866287, loss_reduction = 5.443530e-5, sample_size = 0.4569762, stop_iter = 15L	Recipe 1 2 3
Xgboost_2 trees = 1891, tree_depth = 7, learn_rate = 0.01, loss_reduction = 2.78e-9, sample_size = 0.264	Recipe 1 2 3

One of our team members stumbled upon the set of hyperparameters in model Xgboost_1 when playing around with hyperparameter tuning. We found out that this set of hyperparameters produces really good results. We set the penalty term to 0.01 and get a rmse of around 6.025. This Ensemble model outperforms all of our previously tuned models, and we will use this model as our final submission

5.2 Improvements

Originally, we believed that normal transformation analyzed the relationship of variables well. However when we were developing the relationship between variables, we realized that log transformation produced a clearer visualization than normal transformation. In the future, we would consider comparing more transformation methods to better fit the relationship of the variables.

6. Script Verification Video Link

https://drive.google.com/file/d/14tZg6gSjuRHW7Kbp0_jDlM1ACmBWGXZZ/view?usp=drive_link

7. Appendix (Team Member Contribution)

Team Member Name	Contribution
Phillips (Ruoyan) Li	Designed code for the baseline model and tuned model. Designed all the recipes. Proposed the idea to convert variables to percentage. Contributed in writing the report Ran the hyperparameter tuning code for algorithms.
Vincent (Haojie) Liu	Designed code for the ensemble model. Ran the hyperparameter tuning code for algorithms.
Cyril (Yizhuo) Chang	Contributed in running hyperparameter tuning code. Mascot of the team
Jinyoung Hwang	Contributed in writing the report. Ran the hyperparameter tuning code for algorithms. Generated data visualization plots in the report.
Kristy (Yingyue) Lai	Contributed in writing the report. Ran the hyperparameter tuning code for many algorithms. Drafted the tree diagram.

	Coordinated with team members to combine scripts and results into a systematic report.

We will only present the code of the report in the Appendix. Hyperparameter tuning takes a lot of time, and each team member is responsible for running one or two of all the models tuning. It will take a lot of time to generate a pdf file with output.