Building a High Performing Supervised Learning Model to Predict The Percentage of Voters in County That Voted For Biden In The 2020 US Presidential Election By Demographics and Education Statues.	
Kaggle Team Name	Aplus
Name (SID)	Kyungchae Baek (105548975) Euijun Kim (705788156) Sungwon Lee (405837554) Christopher Apton (105373471)
Model	XGboost

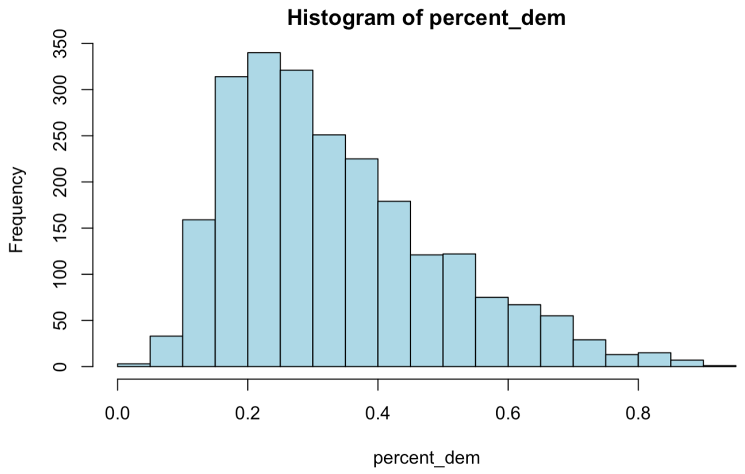
• Introduction: context and background info.

The goal of this paper is to generate a statistical model that predicts the percentage of voters for Biden for each county in the Presidential Election using demographics and education status based on each county. According to exit polls conducted by Edison Research for the National Election Pool, we guessed that race, ethnic heritage, age, and education are important factors which contribute to individuals who are more likely to vote for Biden than Trump because they may be more favorable to Biden policies than Trump's and many people anti-Trump would vote for Biden.

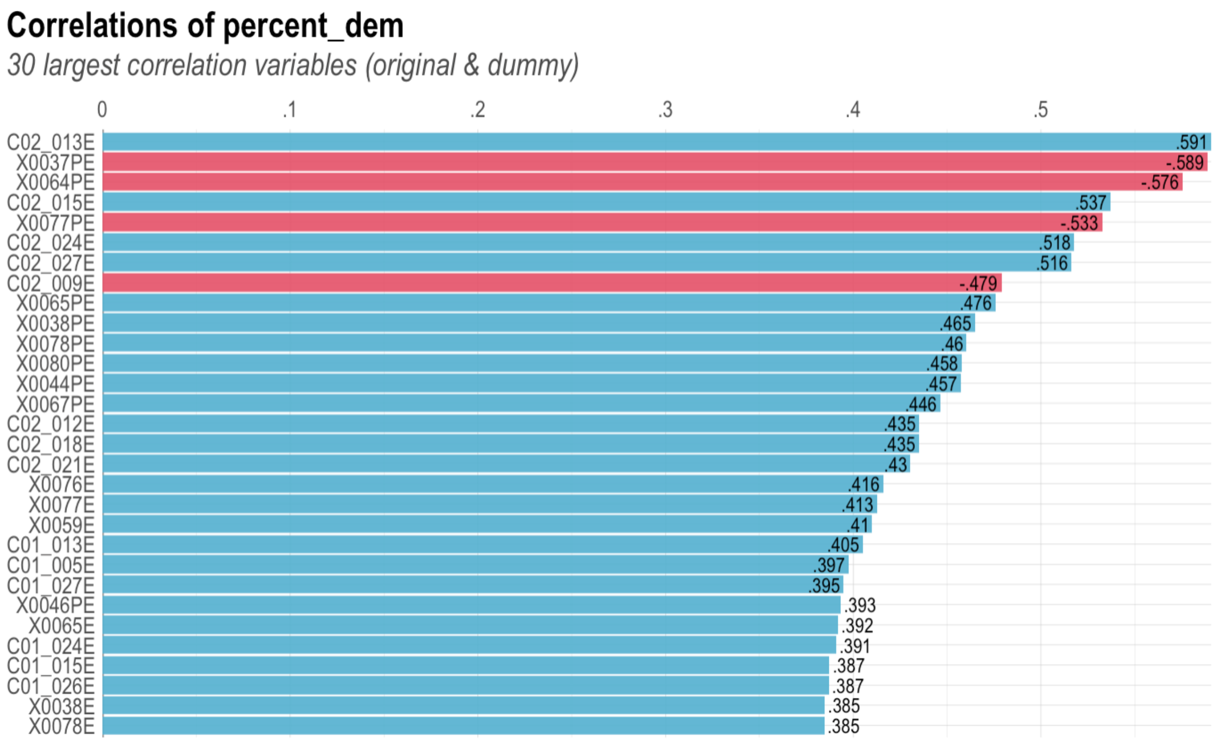
• Exploratory Data Analysis

The Response Variable: percent_dem

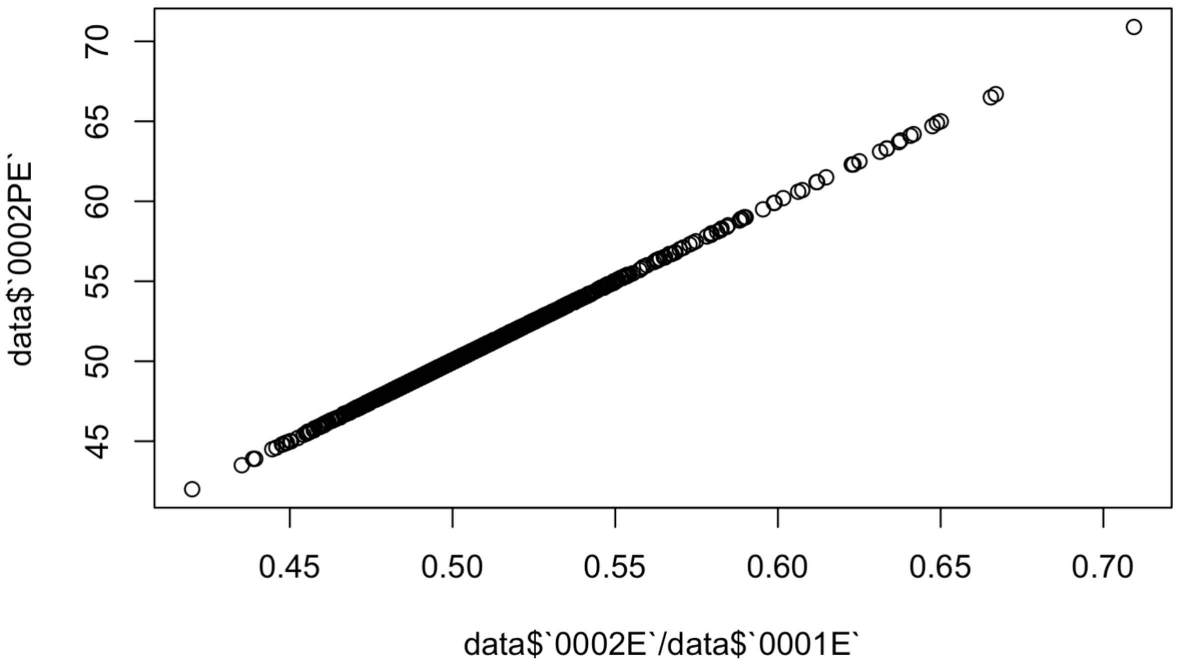
Before attempting a model, we wanted to first look into the normality of the response variable, percent_dem, to see whether we may need to perform a transformation on it. The histogram below indicates that plot is slightly skewed to the right but not close to normal. We decided we could potentially take a transformation for a regression model. However, since we are using the xgboost model, there is no normality assumption since decision trees are completely insensitive to the values and care mostly about the order. So, taking a transformation wouldn't matter in our final results and not taken.



Observing the data, we also looked into the top 30 highly correlated variables with percent_dem which can be shown in the chart below**.** Also, looking at the bottom 30 correlated variables we thought it may be a good idea to attempt PCA to only highlight the important variables and not have noise from the low correlated variables interrupt the results. However, attempting PCA caused a bug that I couldn't fix within the time constraints so the idea had to be scrapped.

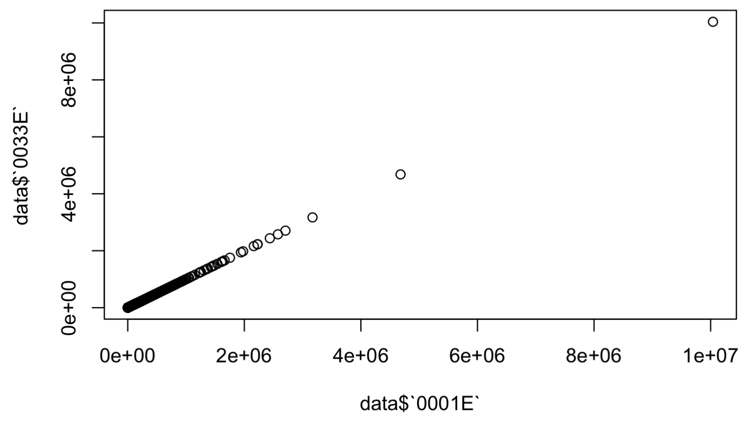


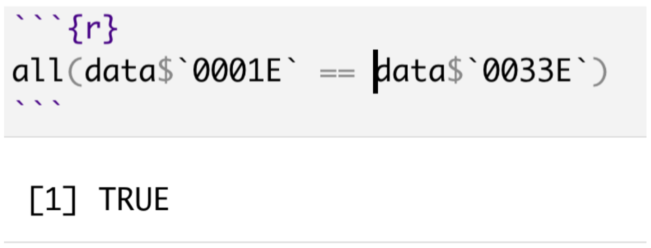
Looking into the variable names, I observed that there's a relationship between PE and P variables which can be shown in the example below.



So, we thought that removing the PE variables would make the model preform better since there are less redundant features. However, upon testing removing the PE variables or the E variables would actually hurt the model performance. So, we ended up just keeping all the features for the final model.

Next, we started looking into variables which could be categorical. Using a XGboost model may not need one hot encoding. Since, the splits would hopefully be good enough to split the numeric categorical variables anyway. It appears looking at the data, there are not any categorical variables. However, looking closely at the data we can see that 2 columns have the same values. Here's the code below to make sure.



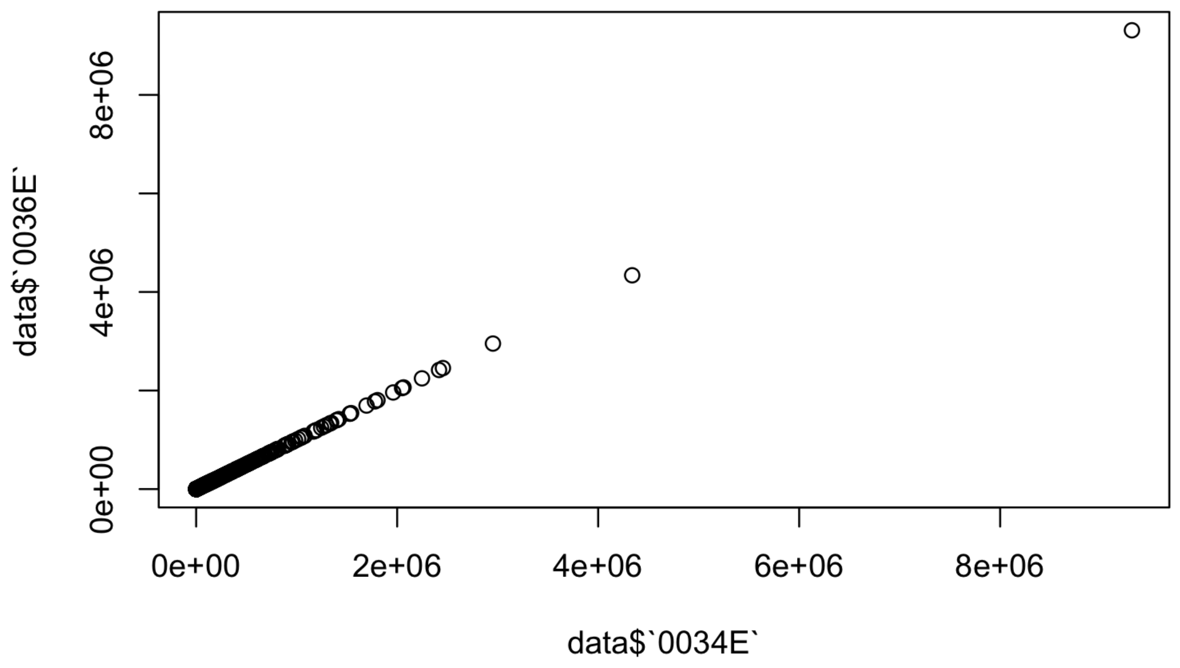


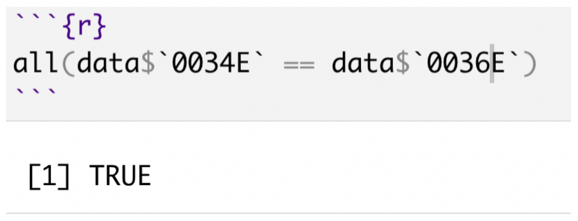
~~~
```{r}
all(data$`0001E` == data$`0033E`)
```

[1] TRUE
~~~

So, we can remove 0033E to have a little less features to deal with. 0001E is the estimate of the total population in each county. We could remove 0033E and 0033PE since 0033PE is also the same thing as 0033E and 0001E.

Also looking at 0034E, we can see it’s contains the same values as 0036E which can be shown in the plot below. So, we should remove 0036E and 0036PE to simplify the data and not have redundant columns.

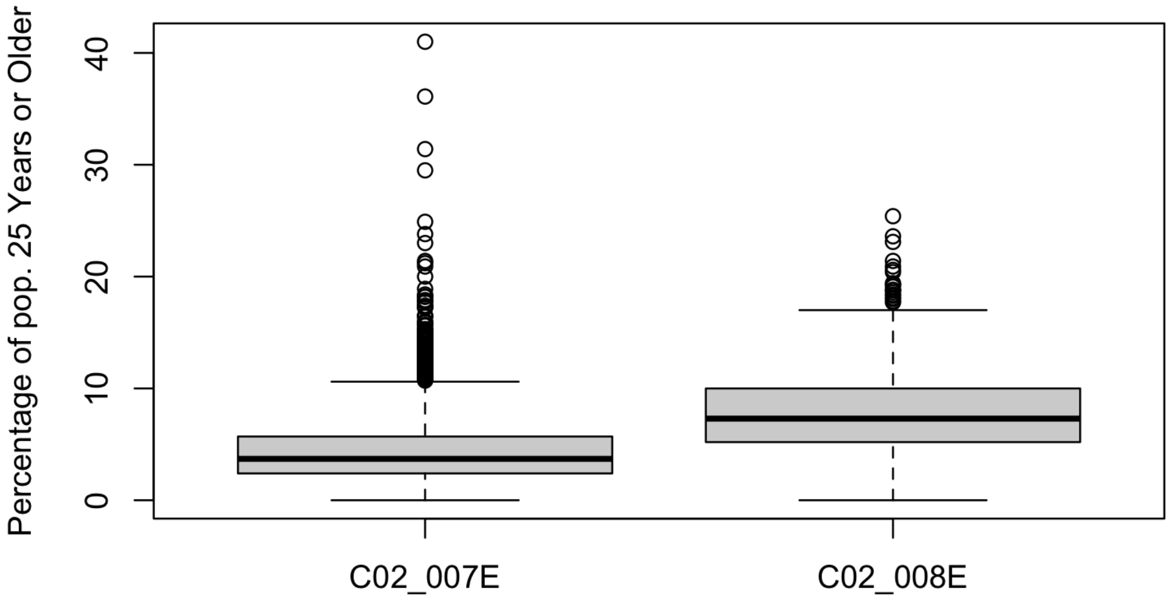




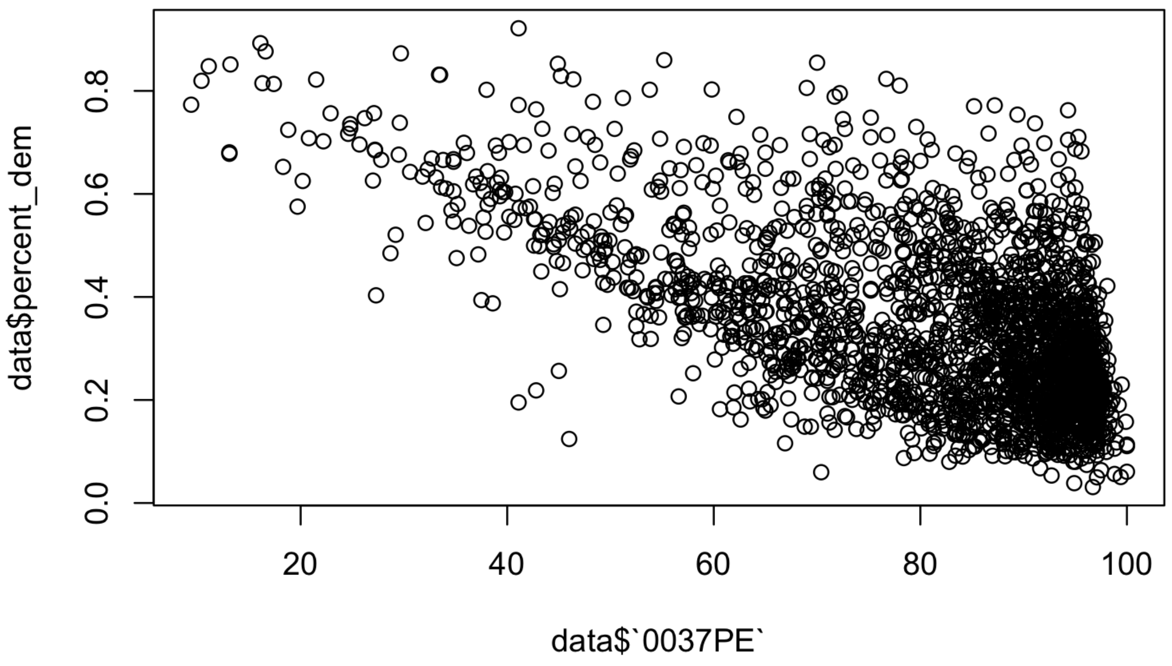
~~~
```{r}
all(data$`0034E` == data$`0036E`)
```

[1] TRUE
~~~

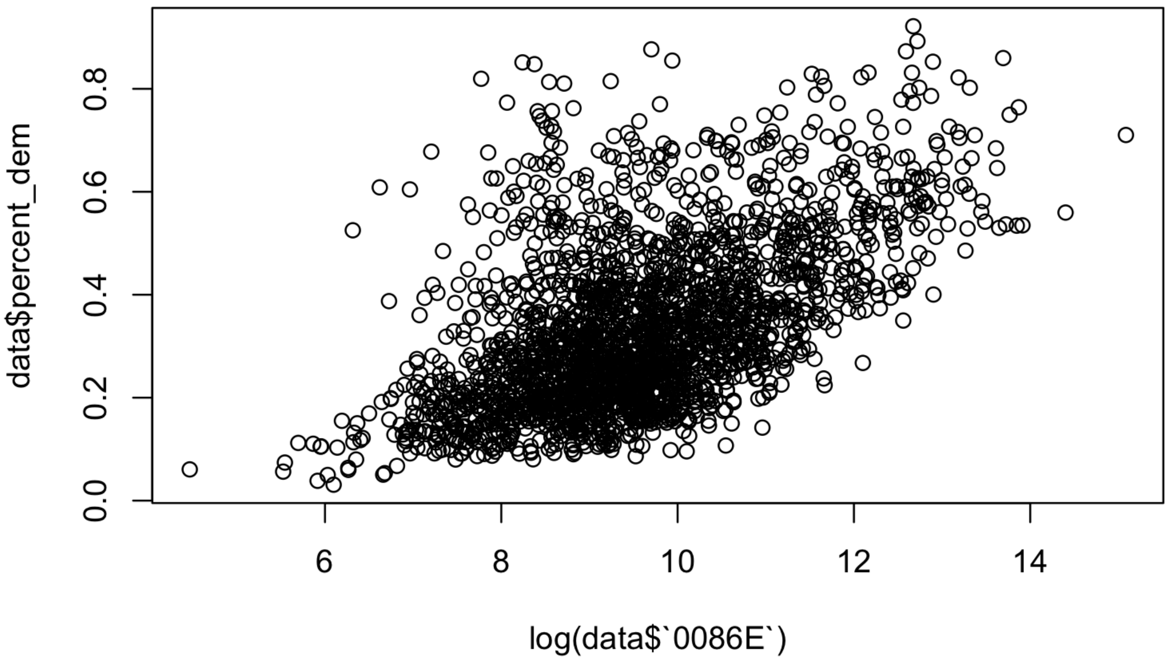
Looking at C02_007E and C02_008E, these are individuals who are 25 and older without a college education. Maybe we could add a categorical variable called abnorm where if there’s a percentage higher than either threshold of 0.1 for C02_007E or 0.15 for C02_008E could be deemed as abnormal and set to 1 or 0 otherwise. The plot below helps us determine what significance to use for the new categorical variable.



Lets look at some variables that may be highly correlated with percent_dem.



With this information, we can manually assign weights to the XGboost model and potentially put more weight on 0037PE. 0037PE is the total percentage of white individuals in each county. Next lets look at 0086E which is total housing units. This can help us determine if the area is Urban, suburban, or rural and their political beliefs. We needed to take the log transformation since population and housing units are exponential growths in order to see the information.



We can see that the more housing units in each county, the higher likelihood of voting for Biden over Trump. This makes sense since Trump's policies are geared towards farmers and rural areas compared to Biden. We could also assign more weight to the XGboost model and take the log transformation of 0086E to see the relationship clearer.

• Preprocessing / Recipes

First of all, we found 1 row with missing values in the data, so we removed the row containing the NA value in the training data. Later, we used a normal recipe in tidymodels to get our data ready for modeling. We also tried a recipe for PCA, however, it had errors and didn't work out. Maybe PCA would've worked better done outside of the tidymodels framework. The normal recipe just contained the y variable we are looking for percent_dem and the other variables as the x variables which are named from the training data. The model we used had 1 outcome and 213 predictors.

• Candidate models

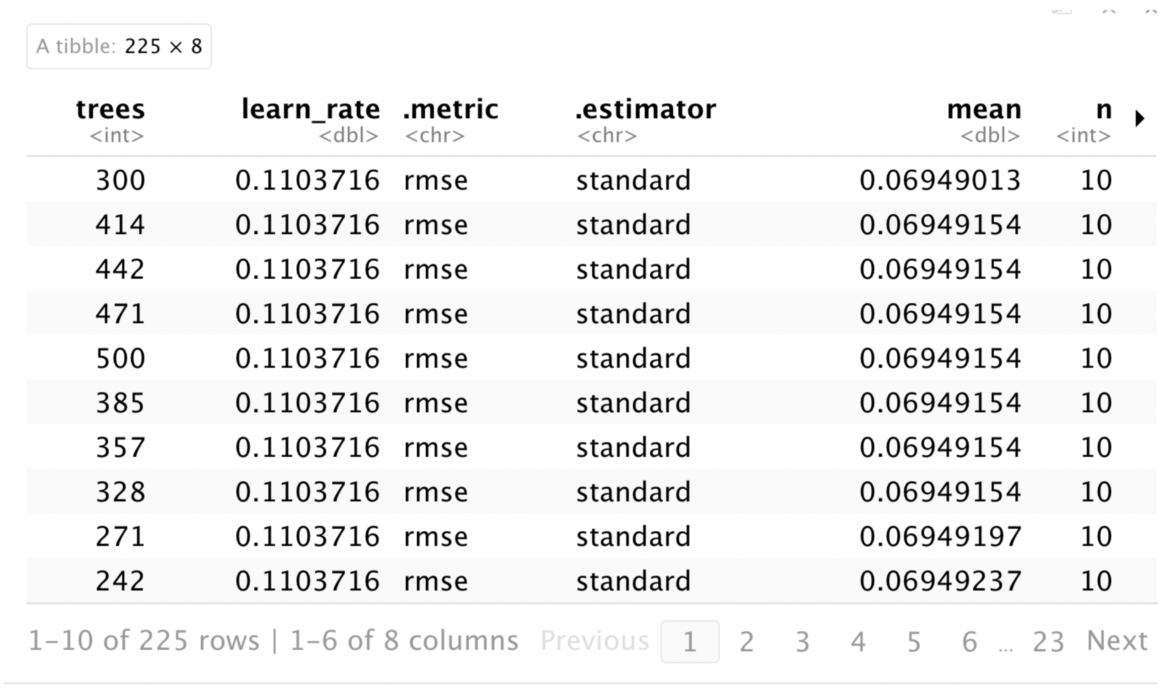
We looked into three final candidate models after tuning, linear model, xboost engine with 500 trees and 0.1 learn rate, and xboost engine with 200 trees and 0.1 learn rate. However, it was difficult tuning the XGmodel with high numbers of trees and we ended up using these 2 xgboost models as the final contenders.

1. lm_model: The linear model from the beginning was just quickly outperformed by any XGboost model.
2. boost_model: This model performed the best in the private score, however, I was worried about it any other issues using so many trees when the default parameter was 15 and we were using 500. It could take a lot longer to train for very small benefits comparing than 200 trees, but I expected it would be better performance with diminishing returns.
3. boost_model2: This model was created in case the boost model with 500 trees would be causing some other issue and testing if there are difference by the number of trees.

	Model Identifier	**Type of Model**	**Engine**	**Recipe used**	**Hyperparameters**
1	lm_model	Linear Regression	lm	norm_recipe	
2	boost_model	XGboost	xgboost	norm_recipe	trees = 500, learn_rate = 0.1
3	boost_model2	XGboost	xgboost	norm_recipe	trees = 200, learn_rate = 0.1

• Model evaluation and tuning

Tuning: We decided to tune the hyper parameters for the xgboost model. The linear model would probably not reach the same potential as XGboost even with tuning so it wasn't tuned. I created a tuning grid with learning rates from 0.01 to 0.5 and also trees from 20 trees to 500 trees.



```
A tibble: 225 × 8

 trees   learn_rate  .metric  .estimator        mean   n
 <int>        <dbl>  <chr>    <chr>            <dbl> <int>
   300    0.1103716  rmse     standard    0.06949013  10
   414    0.1103716  rmse     standard    0.06949154  10
   442    0.1103716  rmse     standard    0.06949154  10
   471    0.1103716  rmse     standard    0.06949154  10
   500    0.1103716  rmse     standard    0.06949154  10
   385    0.1103716  rmse     standard    0.06949154  10
   357    0.1103716  rmse     standard    0.06949154  10
   328    0.1103716  rmse     standard    0.06949154  10
   271    0.1103716  rmse     standard    0.06949197  10
   242    0.1103716  rmse     standard    0.06949237  10

1–10 of 225 rows | 1–6 of 8 columns   Previous  1  2  3  4  5  6 … 23  Next
```

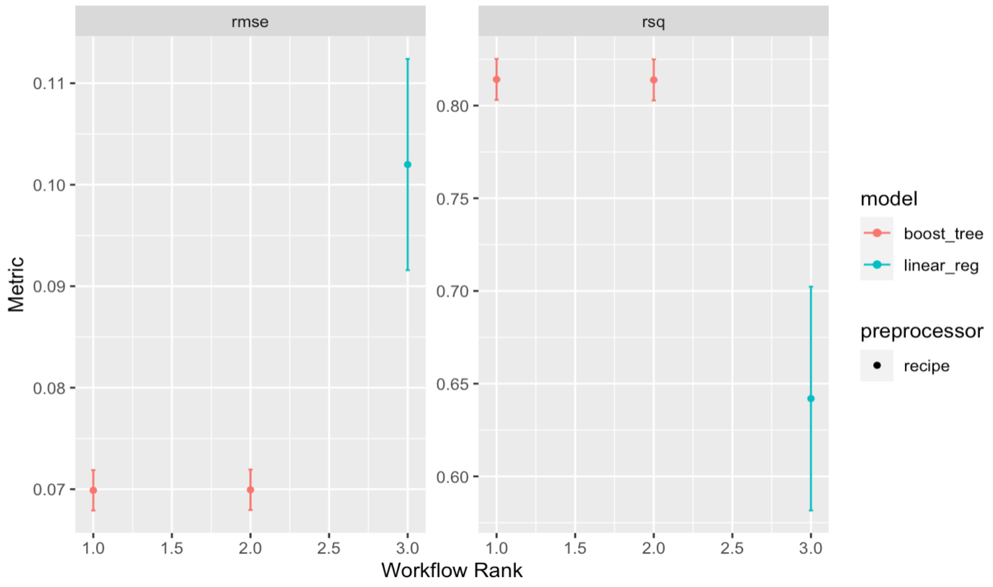
We can observe that between 300 and 500 trees that the results are roughly the same with the lowest mean RMSE. The tuning grid used a learning rate of 0.11, however, I just rounded it to 0.1 for simplicity.

Model Evaluation: We compared the models using the RMSE of the validation data and also using the public score on kaggle. Using v-fold cross validation we can observe in the code output below the mean RMSE of each model with 10 folds.

	Model Identifier	Metric Score	SE of Metric

1	lm_model	0.10198945	0.006326405
2	boost_model	0.06989105	0.001208451
3	boost_model2	0.06994251	0.001210428

From these results, we can say the xgboost model with 500 trees and 0.1 learn rate in boost_model provides the best results slightly over the xgboost model with 200 trees from boost_model2. We can also look at the plot below for a visual comparison of the 3 models.



• **Discussion of final model**

The selection of the final model was a little difficult between the two XGboost models. I thought that having more trees could lead to overfitting. However, it only leads to diminishing returns which we could observe from the RMSE scores. So, a model with 500 trees would be the best, however, the long training time doesn’t make up for the slight increase in performance compared to the 200 tree model.

Strengths and weaknesses: Our model was pretty weak compared to other models on kaggle. However, it could be mostly from a lack of additional tuning and/or cleaning the data. Another weakness of the model, was not using a stack/ensemble. Having just a singular model may not capture all the relationships that using a variety of models may capture and improve performance. One strength of the model was the reliability of the model. There wasn't too much variance in the model.

Some potential improvements would be to spend more time tuning the model. Also, stacking the boost model with other regression models to improve performance. Another improvement would be to clean the data more and observe the relationships between variables closely to make sure we are only using variables we need when fitting the model.

• Appendix: Final annotated script

~~~
---
title: "Stats 101C - Kaggle Competition"
author: "Instructions"
date: "Summer 2022"
output:
  pdf_document: default
  html_document: default
---


```{r setup, include=FALSE}
library(knitr)
library(tidyverse)
library(tidymodels)
library(lubridate)
library(glmnet)
library(xgboost)
~~~

```
knitr::opts_chunk$set(echo = TRUE)
```

Data

~~~
```{r}
# Reading data
data <- read_csv("/Users/chrisapton/Desktop/Summer 2022/Stats
101C/Homework/Kaggle
Competition/ucla-stat-101c-22summer/train.csv")
 data <- data[-1334, ] # remove the row with NA value
 head(data)
```
~~~

Cleaning Data

~~~
```{r}
# Cleaning the data
id_data <- data[, "id"] # column id indicates nothing
data <- select(data, !"id")
head(data)
```
~~~

Splitting Data

~~~
```{r}
# Splitting the data into train and test
set.seed(502)
~~~

~~~~
data_split <- initial_split(data, prop = 0.80, strata =
percent_dem)
train <- training(data_split)
test <- testing(data_split)
```

```{r}
# Creating the recipe
norm_recipe <-
  recipe(percent_dem ~ . , data = train) %>%
  # estimate the means and standard deviations
  prep(training = train, retain = TRUE)
norm_recipe
```

```{r}
# creating the models and organizing them to a list
lm_model <- linear_reg() %>% set_engine("lm")

boost_model <- boost_tree(mode = "regression", trees = 500,
learn_rate = 0.1) %>%
  set_engine("xgboost")

boost_model2 <- boost_tree(mode = "regression", trees = 200,
learn_rate = 0.1) %>%
  set_engine("xgboost")
~~~~

~~~~
model_list = list(lm = lm_model, boost_model_1 = boost_model,
boost_model_2 = boost_model2)
```

```{r}
# creating a workflow_set to be fitted
preproc = list(norm = norm_recipe)
glmnet_models <- workflow_set(preproc = preproc, models =
model_list)
glmnet_models
```

```{r}
# created a variable for cross fold validation with 10 folds
and # fit the model
ames_folds <- vfold_cv(train, v = 10)
keep_pred <- control_resamples(save_pred = TRUE, save_workflow
= TRUE)

glmnet_models <-
  glmnet_models %>%
  workflow_map("fit_resamples",
               seed = 1101, verbose = TRUE,
               resamples = ames_folds, control = keep_pred)
```
~~~~

~~~
```{r}
# metrics of the fitted models
collect_metrics(glmnet_models) %>%
  filter(.metric == "rmse")
```


```{r}
# importing test data for kaggle
kaggle_data <- read_csv("/Users/chrisapton/Desktop/Summer
2022/Stats 101C/Homework/Kaggle
Competition/ucla-stat-101c-22summer/test.csv")
head(kaggle_data)
```


Cleaning Data
```{r}
# Cleaning the data
id_data <- kaggle_data[, "id"]
kaggle_data <- select(kaggle_data, !"id")
head(kaggle_data)
```


```{r}
# fitting the final models for output
glmnet_wflow <-
~~~

```
  workflow() %>%
  add_model(boost_model) %>%
  add_recipe(norm_recipe)

glmnet_fit <- fit(glmnet_wflow, train)

glmnet_wflow_2 <-
  workflow() %>%
  add_model(boost_model2) %>%
  add_recipe(norm_recipe)

glmnet_fit_2 <- fit(glmnet_wflow_2, train)
```

```{r}
# output contained in results.csv and results_2.csv
kaggle_test_results <- bind_cols(id_data, predict(glmnet_fit,
new_data = kaggle_data))
kaggle_test_results_2 <- bind_cols(id_data,
predict(glmnet_fit_2, new_data = kaggle_data))

names(kaggle_test_results) <- c("Id", "Predicted")
names(kaggle_test_results_2) <- c("Id", "Predicted")
write.csv(kaggle_test_results,"results.csv", row.names = FALSE)
write.csv(kaggle_test_results_2,"results_2.csv", row.names =
FALSE)
```

~~~
kaggle_test_results

kaggle_test_results_2

```
~~~

• Appendix: Team member contributions

Kyungchae Baek (105548975): Support other members and writing report

Euijun Kim: (705788156): Report, visualization, and support others

Sungwon Lee (405837554): Script verification and supported others on writing the report

Christopher Apton (105373471): Model, report, and script verification

• References

Brittany Mayes., et al.. “Exit poll results and analysis for the 2020 presidential election” The Washington Post. 14 Dec 2020, www.washingtonpost.com/elections/interactive/2020/exit-polls/presidential-election-exit-polls/