2020 American National Election Study

Stephen Beecher

Description of the dataset

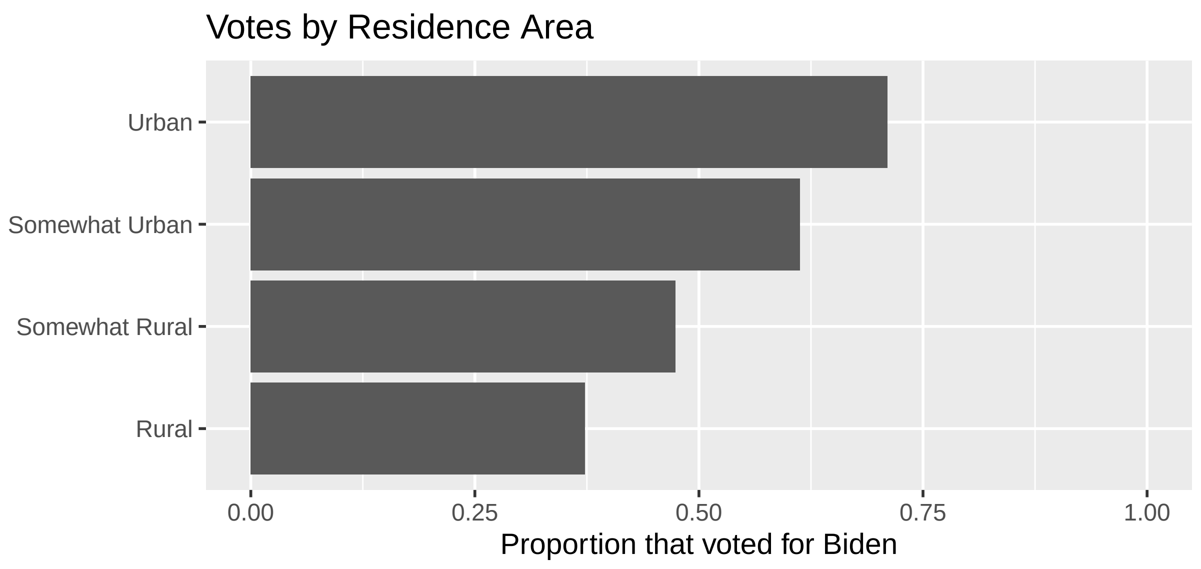
This data comes from the American National Election Study 2020 Time Series Study. There were 8,280 pre-election interviews and 7,449 post-election interviews across U.S. eligible voters. From their website: "Data collection for the ANES 2020 Time Series Study pre-election interviews began in August 2020 and continued until Election Day, Tuesday (November 3). Post-election interviews began soon after the election and continued through the end of December."

I this analysis, I am using just the post-election section of this dataset (some of the visualizations include variables from the pre-election survey). In another file, I processed the data to remove empty columns or columns with no information (see Appendix). I used the post-election data instead of the pre-election data to ensure that the values for some of the variables were true recounts of events (e.g. who did you vote for?) rather than just speculation (e.g. who do you think you want to vote for?).

The main goal of this analysis is to create a model to predict a person's vote based on several other aspects of themselves, such as demographics, evaluations of parties, and opinion on public policy. A secondary goal is to run a similar analysis but to determine whether or not someone will vote. This model will help predict political involvement. Lastly, I used unsupervised learning to reduce the dimensionality of these variables for future analysis.

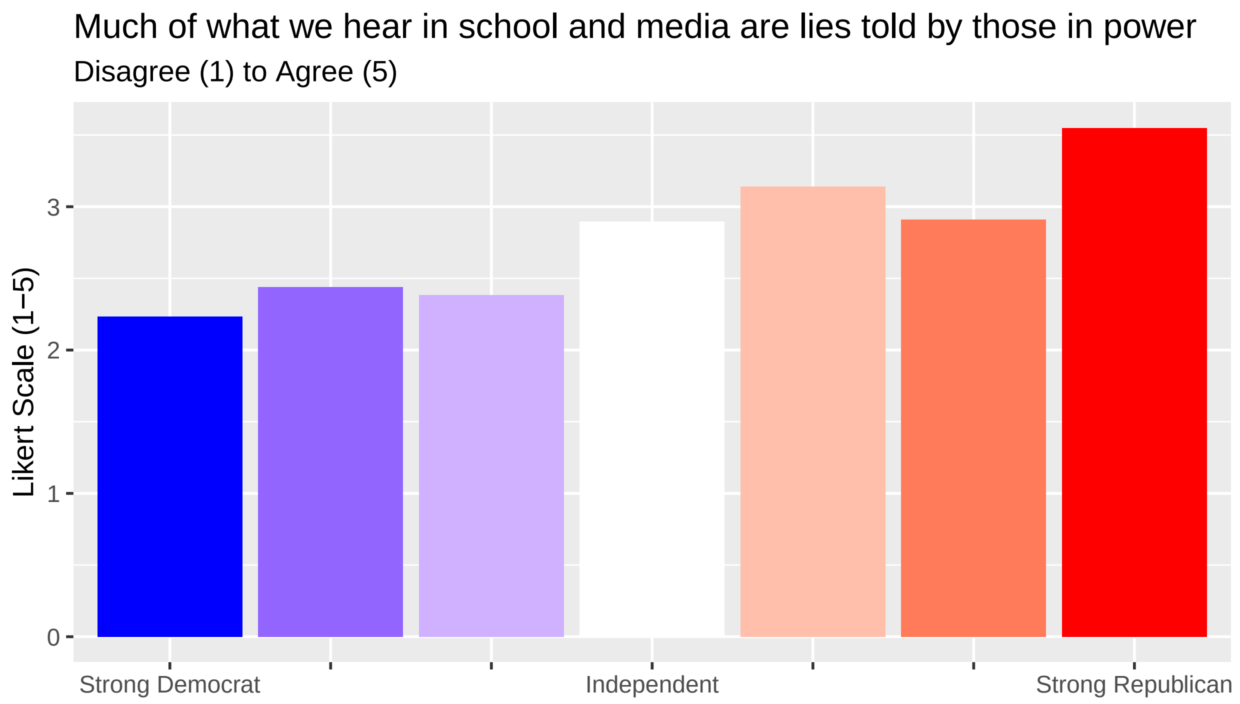
Visualizations

To illustrate some of these examples, I created three visualizations.

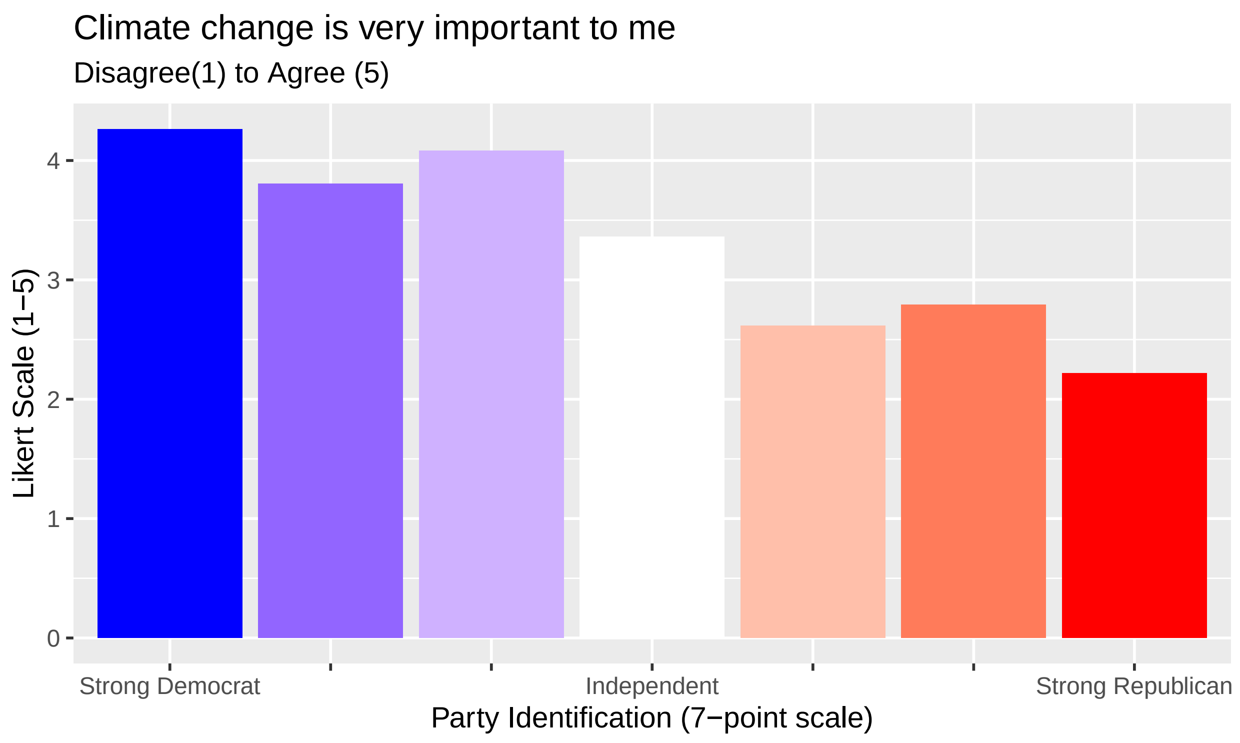


This first one illustrates one feature's relationship to our target variable, vote. In this bar graph, I limited the response to a binary between Joe Biden and Donald Trump, but in the full analysis I include every candidate. We can see in this figure that people in Urban areas voted more for Biden, and follows a decreasing pattern as the areas become more rural.

These next two bar graphs are what this study calls "Thermometer Ratings," or levels of agreement and disagreement with a given statement. With the statement in the title, these figures show the mean response grouped by party identification.



:his result is whether the participant voted in the 2020 election, which is the response variable for



These graphs were made more out of curiosity than use in data analysis, but we can see some interesting patterns.

Data Analysis

To answer the questions outlined above, I used various machine learning techniques to create models and compare their error rates and efficiency.

Vote Choice

First, and the main goal of this analysis, I made several models using the categorical variable "Vote Choice" as the response, and the other variables in the post survey as responses. I removed a couple variables for one of two reasons: they were not aspects of the participants but rather methods of survey, or they had duplicate information.

I used three techniques to create this classification model, Linear Discriminant Analysis (LDA), Random Forest, and k-Nearest Neighbors. The goal was to minimize the test error rate (estimated in different ways).

LDA

LDA is a classification method that assumes each class density is multivariate Gaussian and has equal covariance. This dataset is very large, and it would be very difficult to check these assumptions on each of the predictors, but since the sample size is large, this method is more robust to deviations from these assumptions.

To estimate the test error, I used 5-fold cross-validation and computed the mean test errors from each of the 5 train/test cases.

```r
lda.cv <- function(formula, data, k, seed = 11000){
  data1 <- data
  data1$V202073 <- factor(data1$V202073)

  set.seed(seed)
  n <- nrow(data1)
  sample_i <- sample(1:n, n)
  count = 0

  error <- 0

  for(i in 1:k){
    n_i <- round((n-count)/(k-i+1))
    test_i <- data1[sample_i[c((count+1):(count+n_i))],]
    train_i <- data1[sample_i[-c((count+1):(count+n_i))],]
    count <- count + n_i

    lda_model1 <- lda(formula, data=train_i)
    lda_test_preds <- predict(lda_model1, newdata = test_i)
    error[i] <- mean(lda_test_preds$class != test_i$V202073)
  }
  (cat("Predicted Test Error by ", k, "-fold CV: ", mean(error), sep=""))
  return(error)
}

ldaError <- lda.cv(V202073~.-V200010b-V202110x-V202105x, post, 5)
```

```
## Predicted Test Error by 5-fold CV: 0.0678744
```

This model ran relatively quickly given the large amount of data, and has a very high predictive power. LDA is somewhat sensitive to multicollinearity, and I did not check the assumptions, but this did not seem to have a negative effect on the resulting model.

Random Forest

First proposed by Breiman (2001) and Cutler, Random forest is an ensemble classifier that compiles the results from many decision trees to create a model with higher predictive power than a single tree. This model does not have any mathematical assumptions about the distributions of the data, but it cannot handle missing data. This dataset has about 500 missing points, so I used a random forest imputation method to fill them. I was going to also use cross-validation to estimate the test error, but this method is extremely computationally intense (especially the imputation), so running it multiple times would have taken an extremely long time. So, I used a single sample of 5,000 training observations and 3,280 testing observations to estimate the test error.

```r
set.seed(100)
i <- sample(nrow(post), 5000)

train <- post[i,]
test <- post[-i,]

train_imputed_rf <- rfImpute(V202073~.-V200010b-V202110x-V202105x, data=train)
```

```
## ntree      OOB      1      2      3      4      5      6      7      8      9     10     11     12
##   300:   2.36% 46.43%  0.00%  0.00%  0.00%  0.15%  0.33% 93.62%100.00%100.00%100.00%100.00%100.00%10
## ntree      OOB      1      2      3      4      5      6      7      8      9     10     11     12
##   300:   2.22% 39.29%  0.00%  0.00%  0.00%  0.05%  0.07% 95.74%100.00%100.00%100.00%100.00%100.00%10
## ntree      OOB      1      2      3      4      5      6      7      8      9     10     11     12
##   300:   2.20% 39.29%  0.00%  0.00%  0.00%  0.15%  0.13% 87.23%100.00%100.00%100.00%100.00%100.00%10
## ntree      OOB      1      2      3      4      5      6      7      8      9     10     11     12
##   300:   2.24% 39.29%  0.00%  0.00%  0.00%  0.15%  0.13% 91.49%100.00%100.00%100.00%100.00%100.00%10
## ntree      OOB      1      2      3      4      5      6      7      8      9     10     11     12
##   300:   2.34% 39.29%  0.00%  0.00%  0.00%  0.20%  0.20% 97.87%100.00%100.00%100.00%100.00%100.00%10
```

```r
rf_model1 <- randomForest(V202073~.-V200010b-V202110x-V202105x, data=train_imputed_rf)
rf1_test_preds <- predict(rf_model1, newdata = test)

(rf1Error <- mean(rf1_test_preds != test$V202073, na.rm = T))
```

```
## [1] 0.04296346
```

This value is less than that of the LDA model, which is to be expected given that the random forest has no predictor distribution assumptions. However, since this model took multiple times longer to run, and the error difference is not enormous, for further analysis of this data, I chose to use LDA to save time.

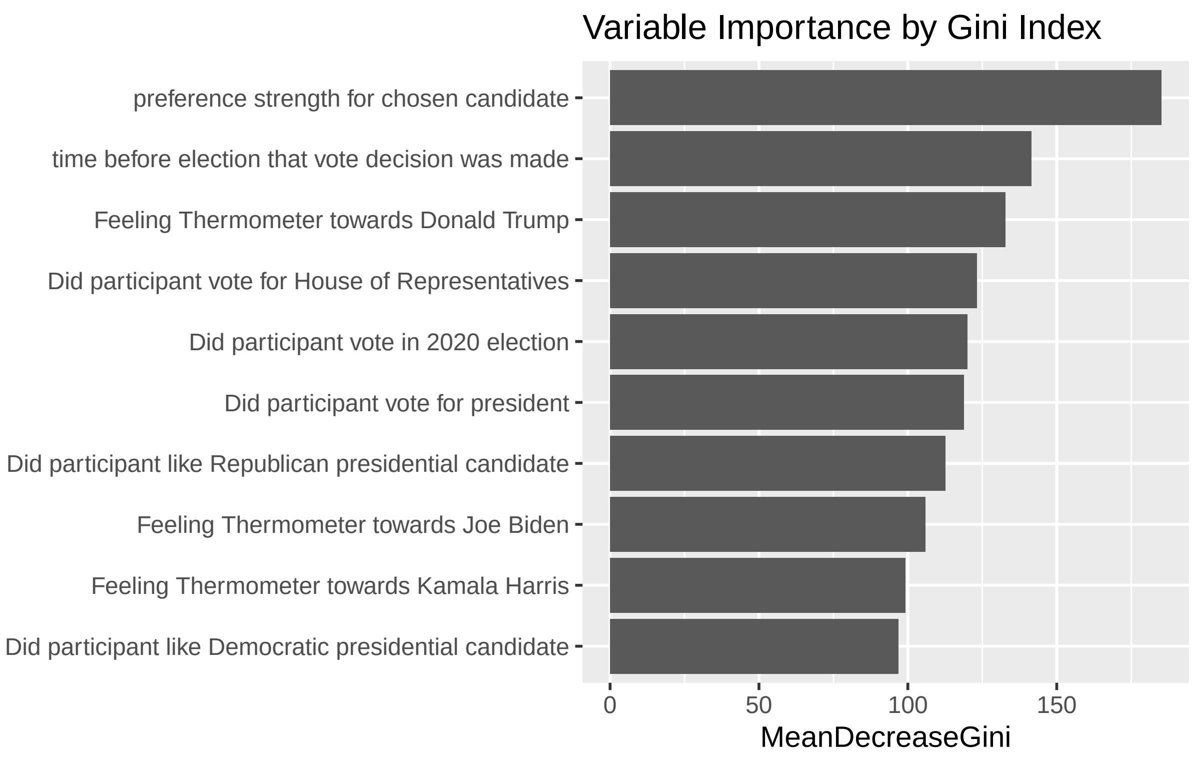
As a method of interpretation, I created a graph which shows the variables in the random forest model that are the most “important,” measured by Gini Index, which is a measure of the decrease in node impurities after splitting on that variable.

```r
i <- rownames_to_column(data.frame(randomForest::importance(rf_model1)), 'variable')

i$variable <- i$variable %>%
  recode(V202074="preference strength for chosen candidate",
         V202075y = "time before election that vote decision was made",
         V202144 = "Feeling Thermometer towards Donald Trump",
         V202080 = "Did participant vote for House of Representatives",
         V202066 = "Did participant vote in 2020 election",
         V202072 = "Did participant vote for president",
         V202436 = "Did participant like Republican presidential candidate",
         V202143 = "Feeling Thermometer towards Joe Biden",
         V202156 = "Feeling Thermometer towards Kamala Harris",
         V202435 = "Did participant like Democratic presidential candidate"
```

```
        )

slice_max(i, order_by = MeanDecreaseGini, n = 10) %>%
  ggplot(aes(x = MeanDecreaseGini, y = reorder(variable, MeanDecreaseGini))) +
  geom_col() +
  labs(y='', title = "Variable Importance by Gini Index")
```



I found it interesting in this result that included is whether the participant voted in the 2020 election, which is the response variable for my secondary analysis. Also, the most important variable had nothing to do with which person someone voted for, only the strength of their preference. This variable was the first in the list in several repetitions of this model (almost all of these top 10 keep the same order).

Logically, most of these relate to current feelings about the presidential candidates, which are likely related to vote choice. In further analysis, I would like to limit the predictors to specific information, such as demographics, to limit these highly correlated answers, but time did not permit in this report. The correlated nature of many of these predictors might be leading to overestimates of the predictive powers of my models, and accounting for this in the models or reducing the dimensions might help this issue.

KNN

As a final model to compare, I used k-nearest neighbors. This model also took longer to compute than LDA, because of the large amount of predictors. Also, I was unsure what value of k to use. To solve both of these issues, I used a single train/test set to save time (same size as in the random forest model), and I repeated the model over several values of k and reported the test error. kNN also cannot have missing values, so in a similar manner to the last method, I used a kNN imputation method to replace them.

```
set.seed(100)
i <- sample(nrow(post), 5000)

train <- post[i,]
```

```r
test <- post[-i,]

train_imputed_knn <- kNN(train)[,1:527]
test_imputed_knn <- kNN(test)[,1:527]

knn_classify <- function(kk, train, test){
  test_preds <- class::knn(train[,which(!names(train)%in%c("V202073", "V200010b", "V202110x", "V202105x
  test_error <- mean(test_preds != test$V202073)
  return(test_error)
}

k <- c(1 , 4 , 7 , 10 , 13 , 16 , 30 , 45 , 60 , 80 , 100 , 150 , 200)
errors <- matrix(nrow=length(k), ncol=2)

for(i in 1:length(k)){
  testerr <- knn_classify(k[i], train_imputed_knn, test_imputed_knn)
  errors[i,]=c(k[i], testerr)
}

errors
```

```
##       [,1]      [,2]
##  [1,]    1 0.3957317
##  [2,]    4 0.3603659
##  [3,]    7 0.3435976
##  [4,]   10 0.3399390
##  [5,]   13 0.3265244
##  [6,]   16 0.3243902
##  [7,]   30 0.3137195
##  [8,]   45 0.3143293
##  [9,]   60 0.3146341
## [10,]   80 0.3222561
## [11,]  100 0.3356707
## [12,]  150 0.4079268
## [13,]  200 0.4704268
```

Here we can see that this model performs significantly worse than the other two by a very large margin. Even the model with the best value of $k = 30$ still has a test error of 31%. To see if the imputation method was the issue, I tried it on the random forest imputation with $k = 30$:

```r
knn_classify(30, train_imputed_rf, test)
```

```
## [1] 0.3140244
```

This is not any better than the model using the kNN imputation. This method is much worse than the other two likely because of the large amount of predictors. The more predictors there are in a kNN model, the "farther apart" the points are (in terms of the distance measure used to find nearby points) and the model loses its ability to classify points well. This data has 524 variables, causing the kNN algorithm to lose a large amount of predictive power.

Vote Choice Conclusion

After using all three of these models, LDA and random forest performed very well, with random forest slightly outperforming LDA. Because of the extremely long run time of the random forest (including imputation), I would recommend LDA, and I use this method in my secondary analysis, which follows.

Political Involvement

As a supplement to the main analysis above, I created a similar model, but I replaced the response variable with the binary predictor of whether or not someone voted. I used LDA, which is shown above to have high predictive power with this dataset. Once again I used 5-fold cross-validation.

```r
lda.cv <- function(formula, data, k, seed = 11000){
  data1 <- data
  data1$V202073 <- as.numeric(data1$V202073)

  set.seed(seed)
  n <- nrow(data1)
  sample_i <- sample(1:n, n)
  count = 0

  error <- 0

  for(i in 1:k){
    n_i <- round((n-count)/(k-i+1))
    test_i <- data1[sample_i[c((count+1):(count+n_i))],]
    train_i <- data1[sample_i[-c((count+1):(count+n_i))],]
    count <- count + n_i

    lda_model1 <- lda(formula, data=train_i)
    lda_test_preds <- predict(lda_model1, newdata = test_i)
    error[i] <- mean(lda_test_preds$class != test_i$V202072)
  }
  (cat("Predicted Test Error by ", k, "-fold CV: ", mean(error), sep=""))
  return(error)
}

ldaError <- lda.cv(V202072~.-V200010b-V202110x-V202105x, post, 5)
```

```
## Predicted Test Error by 5-fold CV: 0.01533816
```

This model has extremely low test error, even less than the random forest above. This has by far the best predictability of any model I tried. The issue of variables that are extremely correlated with the response still looms, so this might be an underestimate of test error rate.

Unsupervised Learning

A large issue with this data is that there are a large number of variables, and many of them are related. This makes models hard to interpret, and any algorithm that is sensitive to multicollinearity is likely a poor choice for this dataset. Here I used Principal Components Analysis to try to reduce the dimensionality of this data, and used the resulting matrix to find some patterns.

```r
post_imputed <- kNN(post)[,1:527]

post_imputed$V202073 <- as.numeric(post_imputed$V202073)

results <- prcomp(post_imputed, scale = TRUE)

vars <- results$sdev^2 / sum(results$sdev^2)
vars[which(cumsum(vars)<.75)]
```

```
##  [1] 0.650029839 0.024797831 0.013093287 0.009964902 0.007306540 0.007131141
##  [7] 0.005942683 0.005365256 0.004544015 0.004350337 0.004038980 0.003761638
## [13] 0.003742000 0.003401510
```

From this PCA, we can see that 75% of the total variance is accounted for in the first 14 principal components, most of which is in the first one (65% of the variance). This means that the data is mostly spread out along one line.

In the following R chunk, I found the coefficients of the linear combination of predictors in the first principal component and sorted them in decreasing order. Since most of the variation is found in this principal component, the predictors with the highest coefficients are responsible for the most variance.

```
pc1 <- abs((-1*results$rotation)[,1])
sort(pc1, decreasing = T)[1:10]
```

```
##    V202001    V202016   V202098y    V202060    V202014    V202097    V202021
## 0.05307443 0.05298627 0.05296150 0.05285655 0.05276508 0.05272918 0.05269086
##    V202036    V202040    V202013
## 0.05268882 0.05266100 0.05265024
```

None of them seemed excessively large here, but I thought this would be an interesting method to check if there are any predictors that were responsible for a large part of the variance.

Not many conclusions can be made from this PCA analysis, but I think using the first few principal components in further analysis will help reduce the large number of predictors.

Further Analysis

These models are very good at predicting certain variables, but there is almost no interpretability. Moreover, many of the variables give similar information, and if organized, this data could lead to more helpful conclusions past black-box prediction. Like I have mentioned previously, sorting the variables on certain topics could lead to interesting insights and reduce the dimensionality to help certain models perform better. This would also reduce multicollinearity, which might be overinflating the estimated predictive powers of models.

Appendix

Data processing

Data from the American National Election Study 2020 Time Series Study

```
full <- read_csv('data/anes_timeseries_2020_csv_20210719.csv')

post <- full[,754:1494]  # All entries in codebook starting with POST: and the weights for the post stu
pre <- full[,40:753]     # All entries in codebook starting with PRE: and the weights for the pre study

# Removing columns with all missing values

pre <- pre[,(colSums(pre >= 0) > 0|is.na(colSums(pre >= 0)))]
post <- post[,(colSums(post >= 0) > 0|is.na(colSums(post >= 0)))]

# Removing columns that are all the same

uniquePre <- apply(pre, 2, function(x){sum(unique(x)>0)})
uniquePost <- apply(post, 2, function(x){sum(unique(x)>0)})

pre <- pre[,which(uniquePre>1)]
post <- post[,which(uniquePost>1)]

# Adding weight variable

pre <- cbind(full[,c('V200010a')], pre)
post <- cbind(full[,c('V200010b')], post)

post$V202073 <- factor(post$V202073)
full_pruned <- cbind(pre, post)

write.csv(post, "data/anes_post.csv")
write.csv(full_pruned, "data/anes_full.csv")
```