Kaggle Report Submission - Classification Data

Team 12: Katherine Huynh, Victoria Aye, Oliver Siu, Vivian Yee

Introduction

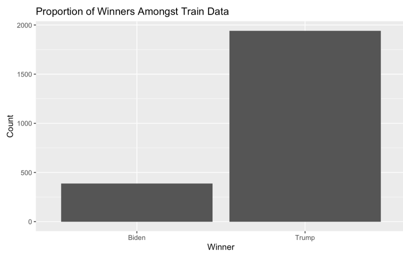
For this project, we referenced a dataset that provides information about the 2020 US Presidential Election (Biden versus Trump). This dataset includes data from 3,111 counties; 75% (2,331 rows) allocated for training and the remaining 25% (780 rows) used for testing. The training set contains 126 variables, such as total population, population by gender, population by age, population by race, and population by education level. The response variable is 'winner,' which is a categorical variable that indicates the winning candidate of the county.

Looking at the voting trends from the article "Behind Biden's 2020 Victory," we see that voting preferences were influenced by demographics and education levels. For instance, Trump had high support from white men without college education, while Biden had more support from women and those with a college degree. Additionally, the ages of individuals was another factor, with younger voters tending to be Democratic and older voters leaning towards Republican. Therefore, we believe that the predictors as mentioned above are significant in predicting the 'winner' for each county.

References:

https://electionlab.mit.edu/
https://data.census.gov/
https://www.pewresearch.org/politics/2021/06/30/behind-bidens-2020-victory/
https://www.ppic.org/wp-content/uploads/jtf-immigrants-political-engagement.pdf
https://www.pewresearch.org/politics/2024/04/09/partisanship-by-gender-sexual-orientation-marital-and-parental-status/

EDA



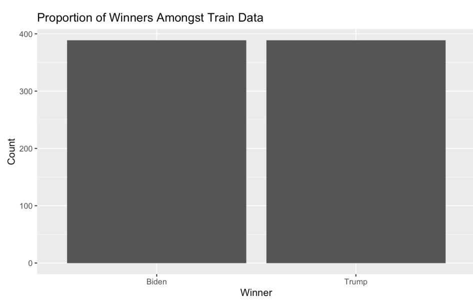
A tibble: 2 × 2

winner <chr>	n <int>
Biden	389
Trump	1942

2 rows

Figure 1

From this model, we can note that the Trump class has a lot more observations than the Biden class. Since there is a large number of observations overall, we should consider downsampling the data to preserve model accuracy.



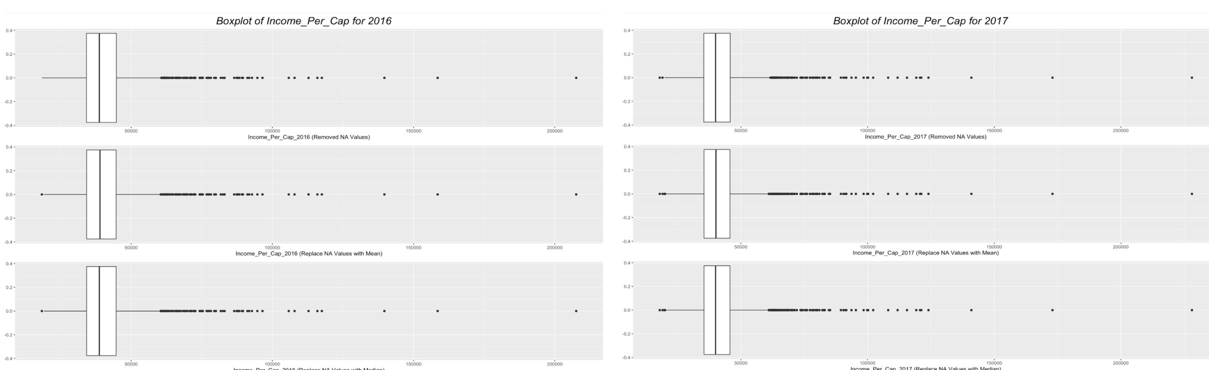
A tibble: 2 × 2

winner <fctr>	n <int>
Biden	389
Trump	389

2 rows

Figure 2

After downsampling the data using the step_downsample() function, we can note that the proportion of observations in each class is much more balanced. Now that the data is more balanced, our models will be able to better learn from each class.



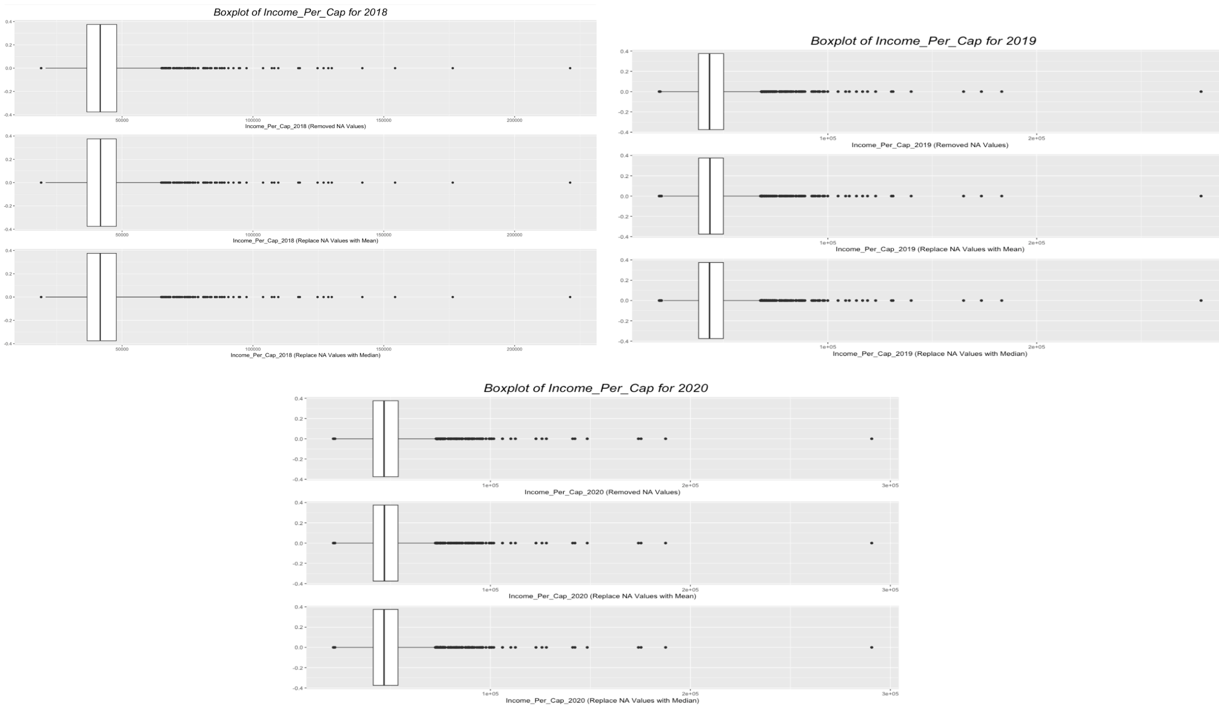


Figure 3

Because income_per_cap data had NAs, we wanted to see if filling the NAs with the median or mean of its column would drastically change the income_per_cap column's distribution. From this, we can see that either option does not change the data drastically.

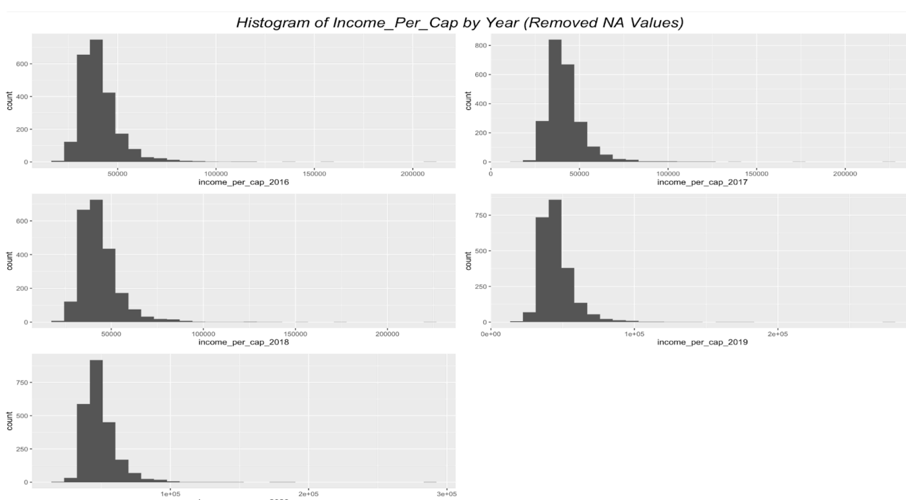


Figure 4

From this model, we can note that the income_per_cap variable tends to be more right-skewed. So, it would be a better choice to use median in order to prevent any outliers from drastically impacting our model accuracy.

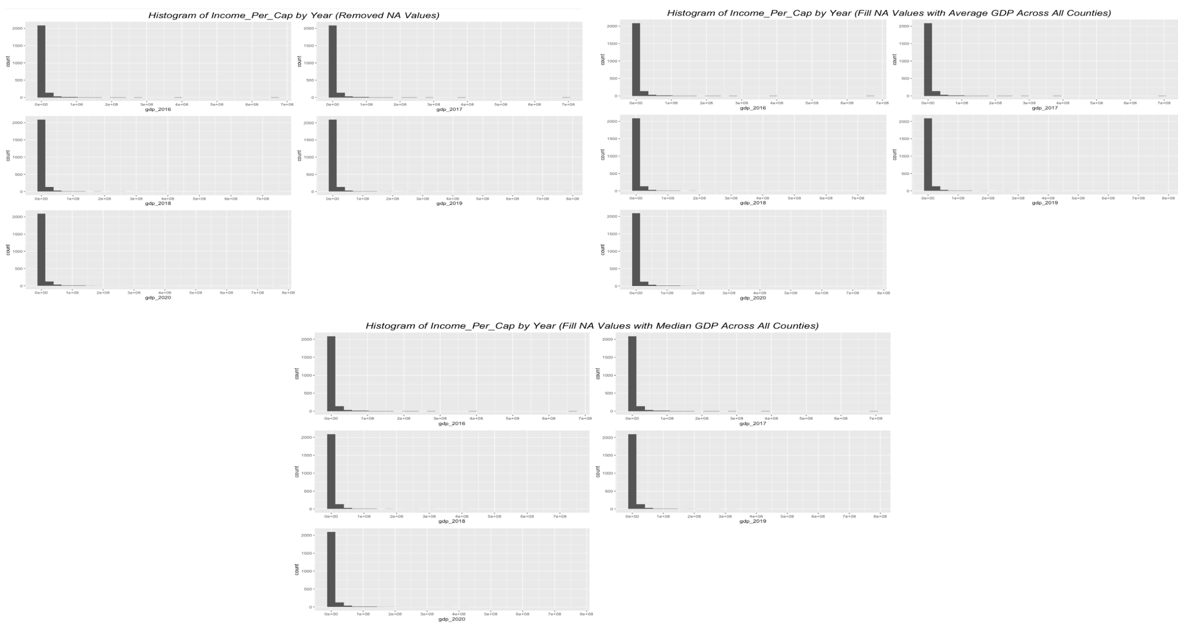


Figure 5

From this model, we noticed that imputing the mean or median for the NA values did not create a large difference in each of the GDP's variable distributions. Since the GDP variable appears to be right skewed, we ended up choosing to impute these values with the median as well.

winner <chr>	Gender <chr>	Proportion <dbl>
Biden	Female	0.5073054
Biden	Male	0.4926946
Trump	Female	0.4975377
Trump	Male	0.5024623

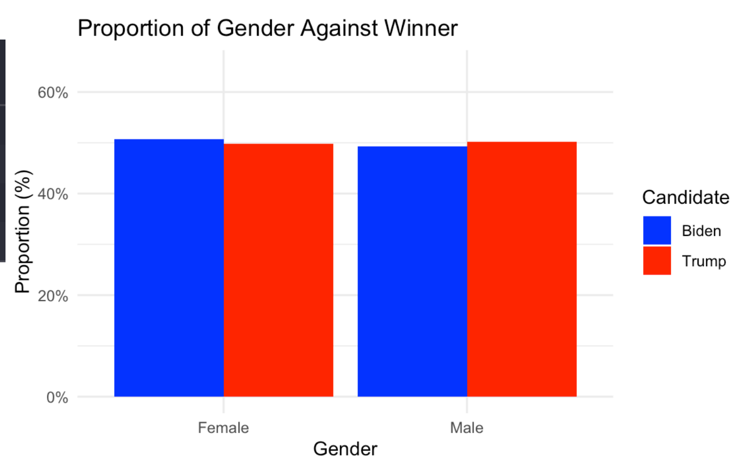


Figure 6

We plotted the proportion of gender against the winner to determine if gender impacts voting preferences. The results show that the counties favoring Biden have slightly more female voters, whereas Trump has more male voters. The difference is marginal, suggesting that gender as a predictor might not be sufficient.

winner <chr>	AgeGroup <chr>	Proportion <dbl>
Biden	under_5	0.05734536
Biden	x10_14	0.06166897
Biden	x15_19	0.06758904
Biden	x20_24	0.07339619
Biden	x25_34	0.13396696
Biden	x35_44	0.12176653
Biden	x45_54	0.12213560
Biden	x55_59	0.06772611
Biden	x5_9	0.05883094
Biden	x60_64	0.06583970
Biden	x65_74	0.10156025
Biden	x75_84	0.04792474
Biden	x85_over	0.02024961
Trump	under_5	0.05747476
Trump	x10_14	0.06469438
Trump	x15_19	0.06303127
Trump	x20_24	0.05749072
Trump	x25_34	0.11533583
Trump	x35_44	0.11571350
Trump	x45_54	0.12327074
Trump	x55_59	0.07216115
Trump	x5_9	0.06058401
Trump	x60_64	0.07191809
Trump	x65_74	0.11356342
Trump	x75_84	0.06023359
Trump	x85_over	0.02452853

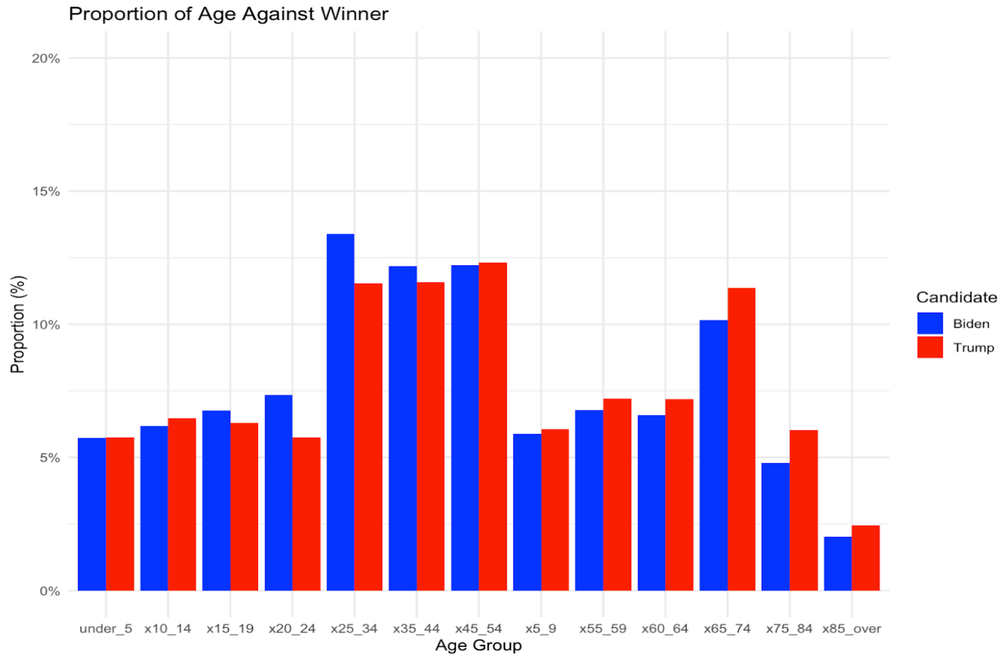


Figure 7

We considered whether age predicts the winner of a county. The bar plot shows that people under 44 tend to vote for Biden, while older populations vote for Trump. Counties with younger age groups favor Biden, indicating that age is a significant predictor.

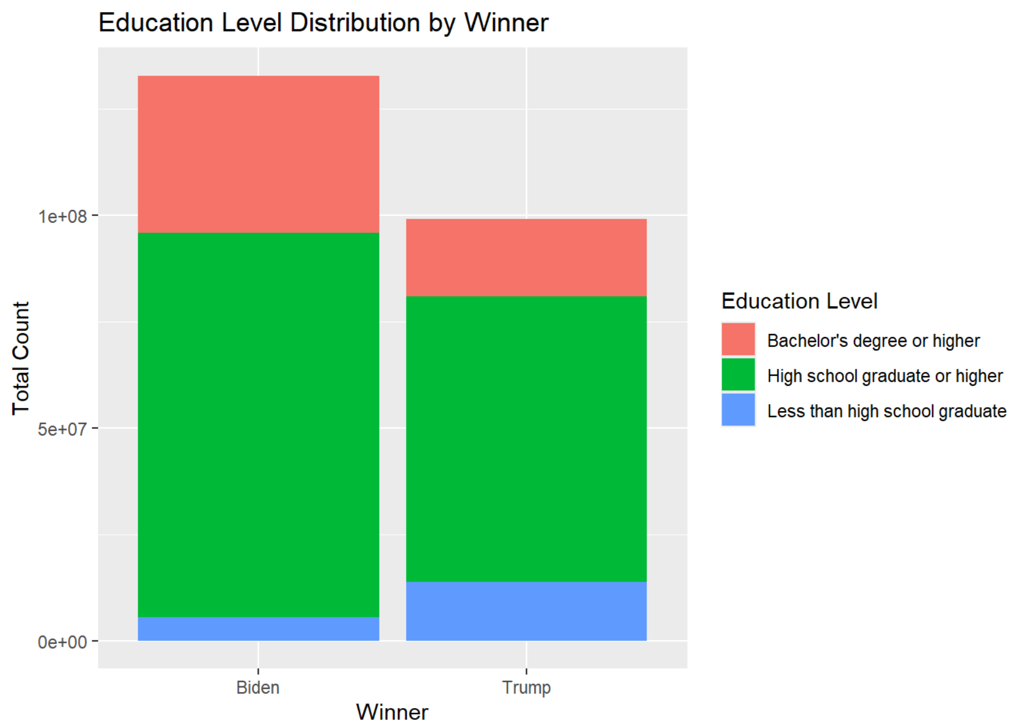


Figure 8

Another consideration was whether voters' education levels would affect their voting patterns. This stacked bar chart shows that districts in favor of Biden had more voters with Bachelor's degrees or higher whereas districts in favor of Trump had more voters whose education levels were less than high school graduates.

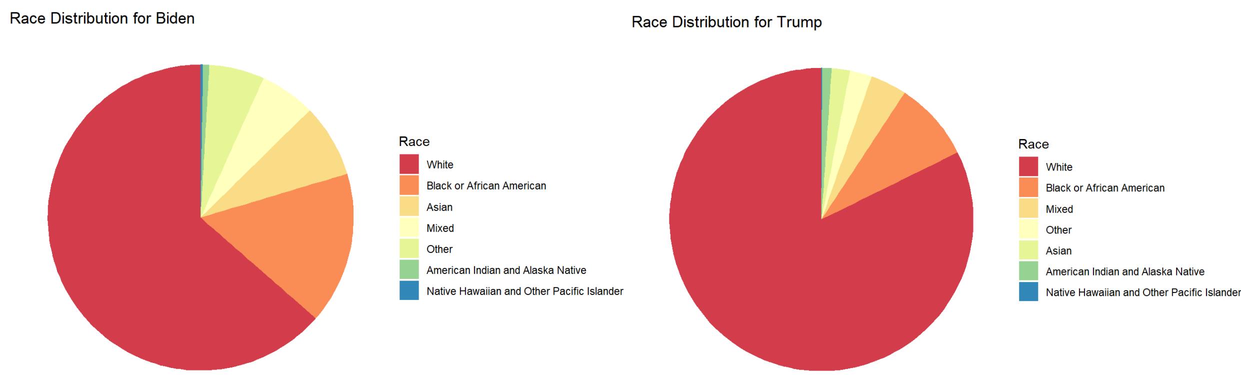


Figure 9

Amongst Biden and Trump voters, white voters make up the largest demographic, then African Americans, Asians, Mixed, Other, American Indian and Alaskan Native, and then Native Hawaiian and Other Pacific Islanders. However, Biden had more votes among minority populations than white voters.

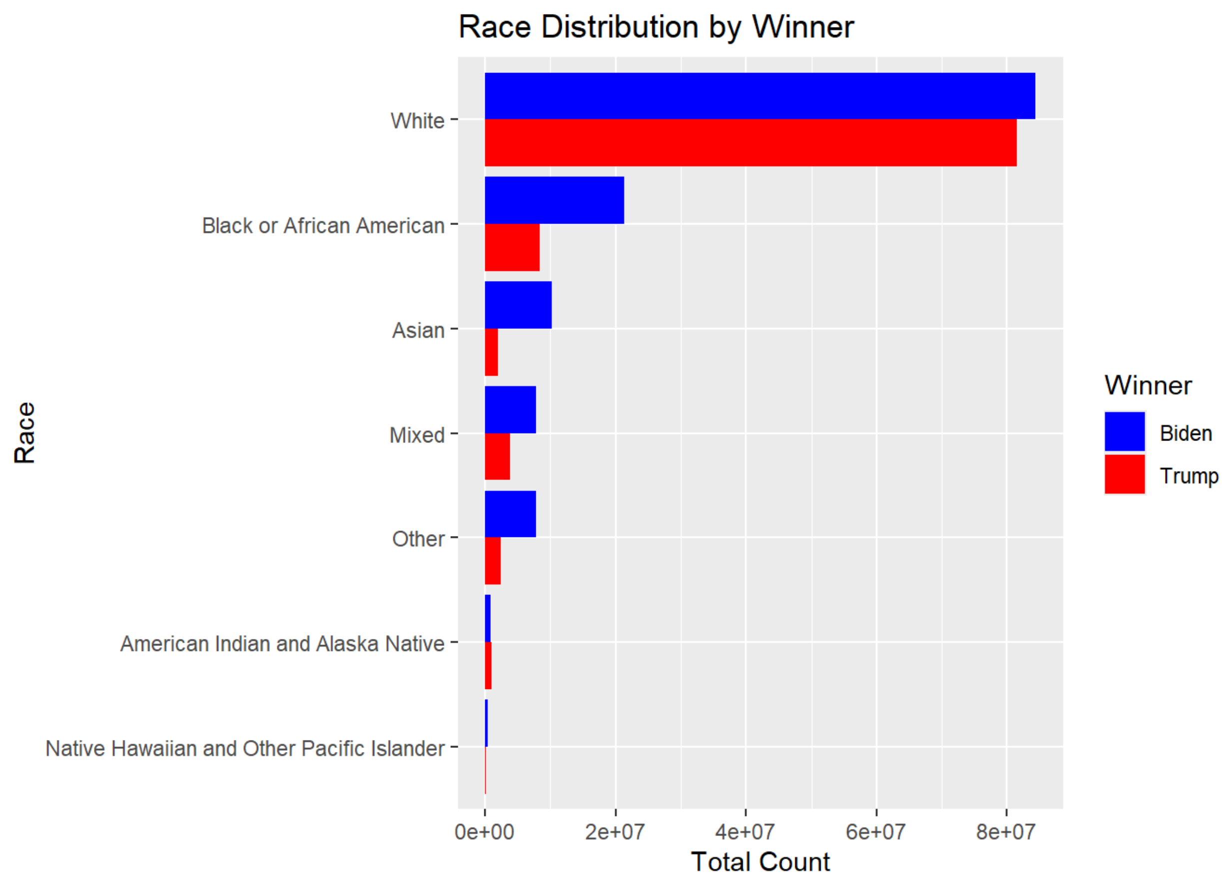


Figure 10

Expanding on figure 9, we wanted to visualize how racial demographics compare between districts. Trump and Biden had a similar amount of white voters. However, Biden had 3-4 times more voters amongst minority races than Trump. Thus, the proportion of minority races amongst voters could be an important predictor.

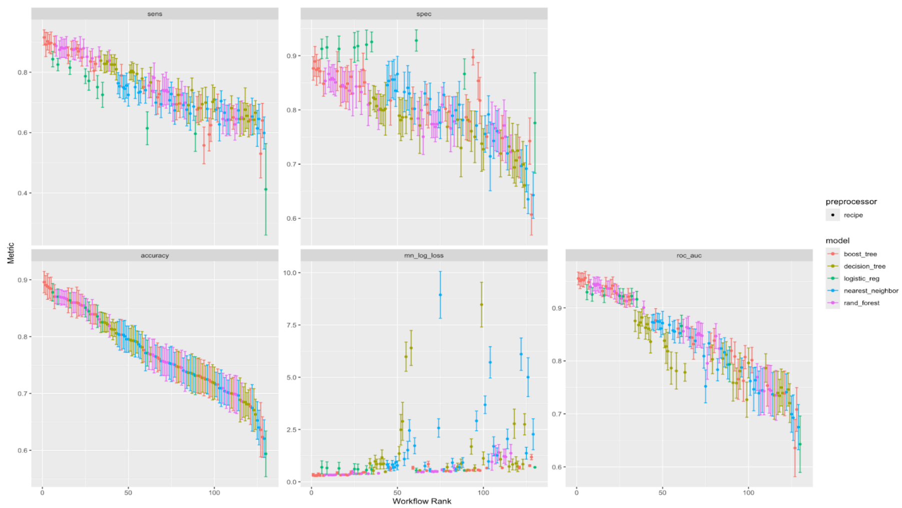
Preprocessing + Recipes

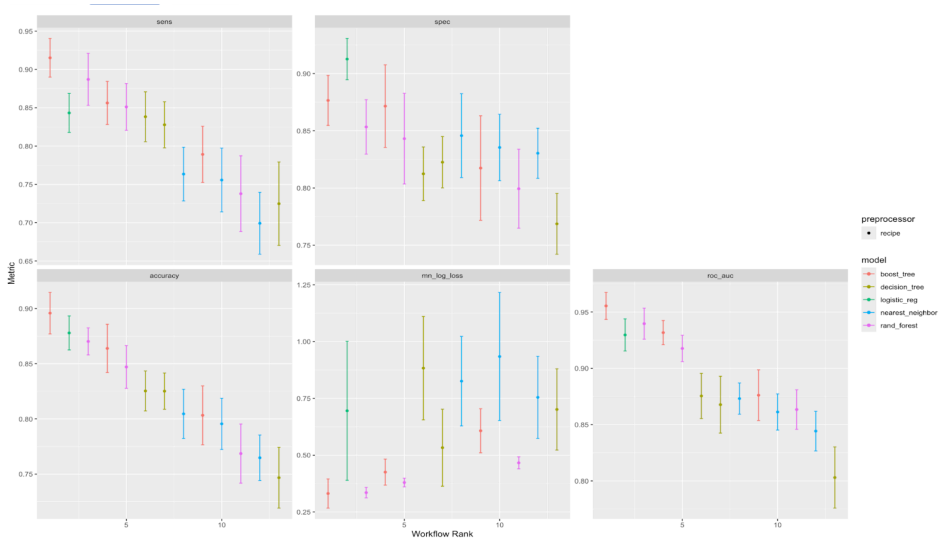
From figures 3-5, we decided to replace the NA values of the income_per_cap and gdp columns with the median of that column. Then, we removed the “x0033e”, “x0036e”, “x0058e” variables because they were a repeat of the “x0001e”, “x0034e”, and “x0035e” columns. Additionally, the “id” and “name” columns were removed since they are not predictor variables. Lastly, we noticed in figure 1 that there was an imbalance in the winner class. So, we used the step_downsample() function to create a more balanced dataset. Then, we applied the prep() and juice() recipe to prepare and apply the recipe to the training data and return the downsampled dataset, which would be used in the following recipes.

For recipes, we created a base recipe, filter recipe, and a pca recipe. For the base recipe, we set the winner variable as the response variable and the remaining variables as the predictor. We also created 7 additional variables: prop_white (proportion of people who are White), prop_minority (proportion of those who are Asian, Black or African American, American Indian and Alaska Native, Hispanic, Native Hawaiian, and/or some other race), prop_college (proportion of those who graduated with a Bachelor’s degree or higher), prop_old (proportion of those who are 65+), prop_ men (proportion of men), prop_female (proportion of female), and prop_minor (proportion of those that are younger than 18). To avoid duplicates, we removed variables that correspond to the original counts of these percentages; for instance, because we have prop_white, we should not keep the variable that has the count of people who are white. Lastly, we factored the last x2013_code variable and applied step_dummy() since this variable should be treated as a categorical variable, not a numerical variable; we had to apply step_dummy() because some models would not allow for a factor predictor.

For the filter variable, we added the step_corr() function with a tunable threshold to the base recipe. For the pca recipe, we started with the base recipe. Then, we had R filter out any variables with near zero variance using the step_nzv() function to get rid of any unnecessary predictors. We also added the step_corr() filter on all numeric predictors (minus the outcomes variable) to eliminate any highly correlated predictors. We also added the step_lincomb() function to remove any linear combinations of other predictors. We then added the step_normalize() function on all numeric predictors (minus the outcome variable) because PCA assumes that the data is already normally distributed. Lastly, we applied the step_pca() function with a tunable threshold and num_comp = 5 so that R can try to reduce the number of features in this dataset.

We then created a workflow set using these 3 recipes and 5 models (boosted tree, decision trees, logistic regression, k nearest neighbor, and random forests). After removing the pca_glmnet and filter_glmnet, we then called the workflow map. After evaluating this workflow set on our 10-fold cross validation with a grid of 10, we called autoplot to visualize all the different types of models and ranked the overall results.


sens
spec
accuracy
mn_log_loss
roc_auc
Metric
Workflow Rank
preprocessor
recipe
model
boost_tree
decision_tree
logistic_reg
nearest_neighbor
rand_forest


sens
spec
accuracy
mn_log_loss
roc_auc
Metric
Workflow Rank
preprocessor
recipe
model
boost_tree
decision_tree
logistic_reg
nearest_neighbor
rand_forest

A tibble: 65 × 9

wflow_id <chr>	.config <chr>	.metric <chr>	mean <dbl>	std_err <dbl>	n <int>	preprocessor <chr>	model <chr>	rank <int>
simple_xgboost	Preprocessor1_Model10	accuracy	0.8958502	0.011417281	10	recipe	boost_tree	1
simple_xgboost	Preprocessor1_Model10	mn_log_loss	0.3311619	0.039128739	10	recipe	boost_tree	1
simple_xgboost	Preprocessor1_Model10	roc_auc	0.9554513	0.007278601	10	recipe	boost_tree	1
simple_xgboost	Preprocessor1_Model10	sens	0.9151822	0.015303261	10	recipe	boost_tree	1
simple_xgboost	Preprocessor1_Model10	spec	0.8765182	0.013241351	10	recipe	boost_tree	1
simple_glmnet	Preprocessor1_Model10	accuracy	0.8779352	0.009353832	10	recipe	logistic_reg	2
simple_glmnet	Preprocessor1_Model10	mn_log_loss	0.6953067	0.185857868	10	recipe	logistic_reg	2
simple_glmnet	Preprocessor1_Model10	roc_auc	0.9296696	0.008640755	10	recipe	logistic_reg	2
simple_glmnet	Preprocessor1_Model10	sens	0.8432524	0.015438204	10	recipe	logistic_reg	2
simple_glmnet	Preprocessor1_Model10	spec	0.9126181	0.010926290	10	recipe	logistic_reg	2
simple_rf	Preprocessor1_Model10	accuracy	0.8702092	0.007454715	10	recipe	rand_forest	3
simple_rf	Preprocessor1_Model10	mn_log_loss	0.3349293	0.014029821	10	recipe	rand_forest	3
simple_rf	Preprocessor1_Model10	roc_auc	0.9396941	0.008376165	10	recipe	rand_forest	3
simple_rf	Preprocessor1_Model10	sens	0.8870445	0.020610498	10	recipe	rand_forest	3
simple_rf	Preprocessor1_Model10	spec	0.8533738	0.014456700	10	recipe	rand_forest	3
filter_xgboost	Preprocessor07_Model1	accuracy	0.8639001	0.013307682	10	recipe	boost_tree	4
filter_xgboost	Preprocessor07_Model1	mn_log_loss	0.4252523	0.034766703	10	recipe	boost_tree	4
filter_xgboost	Preprocessor07_Model1	roc_auc	0.9317441	0.006517027	10	recipe	boost_tree	4
filter_xgboost	Preprocessor07_Model1	sens	0.8562753	0.017099047	10	recipe	boost_tree	4
filter_xgboost	Preprocessor07_Model1	spec	0.8715250	0.021924276	10	recipe	boost_tree	4
filter_rf	Preprocessor06_Model1	accuracy	0.8470985	0.011778318	10	recipe	rand_forest	5
filter_rf	Preprocessor06_Model1	mn_log_loss	0.3794004	0.011527807	10	recipe	rand_forest	5
filter_rf	Preprocessor06_Model1	roc_auc	0.9176648	0.007097596	10	recipe	rand_forest	5
filter_rf	Preprocessor06_Model1	sens	0.8510796	0.018561675	10	recipe	rand_forest	5
filter_rf	Preprocessor06_Model1	spec	0.8431174	0.024062845	10	recipe	rand_forest	5
simple_cart	Preprocessor1_Model03	accuracy	0.8253374	0.011001393	10	recipe	decision_tree	6
simple_cart	Preprocessor1_Model03	mn_log_loss	0.8830060	0.138372832	10	recipe	decision_tree	6
simple_cart	Preprocessor1_Model03	roc_auc	0.8754500	0.012209310	10	recipe	decision_tree	6
simple_cart	Preprocessor1_Model03	sens	0.8382591	0.019784905	10	recipe	decision_tree	6
simple_cart	Preprocessor1_Model03	spec	0.8124157	0.014228115	10	recipe	decision_tree	6
filter_cart	Preprocessor05_Model1	accuracy	0.8251687	0.010005559	10	recipe	decision_tree	7
filter_cart	Preprocessor05_Model1	mn_log_loss	0.5327904	0.102990992	10	recipe	decision_tree	7
filter_cart	Preprocessor05_Model1	roc_auc	0.8677301	0.015342529	10	recipe	decision_tree	7
filter_cart	Preprocessor05_Model1	sens	0.8278003	0.018311631	10	recipe	decision_tree	7
filter_cart	Preprocessor05_Model1	spec	0.8225371	0.013622507	10	recipe	decision_tree	7
simple_knn	Preprocessor1_Model03	accuracy	0.8045884	0.013543250	10	recipe	nearest_neighbor	8
simple_knn	Preprocessor1_Model03	mn_log_loss	0.8258039	0.119853599	10	recipe	nearest_neighbor	8
simple_knn	Preprocessor1_Model03	roc_auc	0.8731248	0.008428719	10	recipe	nearest_neighbor	8
simple_knn	Preprocessor1_Model03	sens	0.7634278	0.021296185	10	recipe	nearest_neighbor	8
simple_knn	Preprocessor1_Model03	spec	0.8457490	0.022291548	10	recipe	nearest_neighbor	8
pca_xgboost	Preprocessor03_Model1	accuracy	0.8032726	0.016205609	10	recipe	boost_tree	9

1–41 of 65 rows Previous 1 2 Next

A tibble: 65 × 9

wflow_id <chr>	.config <chr>	.metric <chr>	mean <dbl>	std_err <dbl>	n <int>	preprocessor <chr>	model <chr>	rank <int>
pca_xgboost	Preprocessor03_Model1	mn_log_loss	0.6073111	0.058997799	10	recipe	boost_tree	9
pca_xgboost	Preprocessor03_Model1	roc_auc	0.8761211	0.013696577	10	recipe	boost_tree	9
pca_xgboost	Preprocessor03_Model1	sens	0.7891363	0.022292694	10	recipe	boost_tree	9
pca_xgboost	Preprocessor03_Model1	spec	0.8174089	0.027737490	10	recipe	boost_tree	9
filter_knn	Preprocessor05_Model1	accuracy	0.7955466	0.014114953	10	recipe	nearest_neighbor	10
filter_knn	Preprocessor05_Model1	mn_log_loss	0.9343030	0.171455762	10	recipe	nearest_neighbor	10
filter_knn	Preprocessor05_Model1	roc_auc	0.8612550	0.009769647	10	recipe	nearest_neighbor	10
filter_knn	Preprocessor05_Model1	sens	0.7556680	0.025274277	10	recipe	nearest_neighbor	10
filter_knn	Preprocessor05_Model1	spec	0.8354251	0.017651506	10	recipe	nearest_neighbor	10
pca_rf	Preprocessor06_Model1	accuracy	0.7685897	0.016285640	10	recipe	rand_forest	11
pca_rf	Preprocessor06_Model1	mn_log_loss	0.4661071	0.016033001	10	recipe	rand_forest	11
pca_rf	Preprocessor06_Model1	roc_auc	0.8634168	0.010679135	10	recipe	rand_forest	11
pca_rf	Preprocessor06_Model1	sens	0.7377868	0.030043320	10	recipe	rand_forest	11
pca_rf	Preprocessor06_Model1	spec	0.7993927	0.020973966	10	recipe	rand_forest	11
pca_knn	Preprocessor05_Model1	accuracy	0.7647773	0.012533666	10	recipe	nearest_neighbor	12
pca_knn	Preprocessor05_Model1	mn_log_loss	0.7543708	0.109720780	10	recipe	nearest_neighbor	12
pca_knn	Preprocessor05_Model1	roc_auc	0.8442662	0.010704575	10	recipe	nearest_neighbor	12
pca_knn	Preprocessor05_Model1	sens	0.6991903	0.024531437	10	recipe	nearest_neighbor	12
pca_knn	Preprocessor05_Model1	spec	0.8303644	0.013305077	10	recipe	nearest_neighbor	12
pca_cart	Preprocessor05_Model1	accuracy	0.7467274	0.016760186	10	recipe	decision_tree	13
pca_cart	Preprocessor05_Model1	mn_log_loss	0.7011295	0.108730527	10	recipe	decision_tree	13
pca_cart	Preprocessor05_Model1	roc_auc	0.8029987	0.016501983	10	recipe	decision_tree	13
pca_cart	Preprocessor05_Model1	sens	0.7247638	0.033078029	10	recipe	decision_tree	13
pca_cart	Preprocessor05_Model1	spec	0.7686910	0.016130729	10	recipe	decision_tree	13

Figure 11-14

Candidate Models/Model Evaluation/Tuning

Candidate Models:

For the first candidate model, we noticed in the workflow set (figure above) that the boosted tree model with the base recipe seemed to perform the best. So, we decided to tune the hyperparameters of the boosted tree model and see how it performs individually. However, we found that the cross validation score for this model was lower than the lightgbm engine (the fourth candidate model).

For the second candidate model, we noticed that the logistic regression, random forest, and boosted tree models with the base recipe (mentioned above) were the top performing workflows in the figure above. So, we decided to try and create an ensemble model for this as well. Ultimately, this ensemble model had the highest performing accuracy score from R amongst all the ensemble models.

Similar to the previous model, the third model is also an ensemble model. After trying out different combinations of ensemble models, we found that the ensemble model combining random forest and the boosted tree model performed had the highest accuracy rate from cross validation. So, we decided to use this model as another potential candidate. We found that this ensemble model had a lower accuracy score from R than the second candidate model despite having the same Kaggle score. So, we decided to use the previous candidate model.

The fourth model was designed following the completion of boosted tree engine testing. When no tuning was applied, lightgbm returned higher accuracy than xgboost during cross-validation comparisons. The lightgbm engine also scored well on the public leaderboard and was a candidate for as a simple model. With this model, we found that it was less prone to overfitting. Additionally, it had the highest accuracy score from cross validation amongst all the non-ensemble models. So, we decided to use this as our second final model.

For the fifth model, we also decided to try an mlp engine. Although we did not include it in the workflow set, we were curious about its overall performance. So, we applied the same base recipe that we applied previously. Because the other mlp engines required other packages (such as tensorflow), we decided to use the “nnet” engine, which allows for a single hidden layer. This model did not perform as well as the boosted tree models mentioned above, which is shown in both its lower accuracy and Kaggle score.

For the sixth model, we tried logistic regression using glmnet engine. We chose to test this model as it was a simple and quick model to run. However, the results were average compared to our other models. Thus, we decided not to choose this as our final model. Like the fifth model, this model also had a lower accuracy and Kaggle score. So, we decided to use the lightgbm boosted tree model instead.

Recipes Used in the Candidate Models Below

Recipe Number	**Recipe**

Recipe 1	train_removed_downsample <- recipe(winner ~., data = train_removed) %>%
	  step_downsample(winner) %>%
	  prep() %>%
	  juice()
	
	base_recipe <- recipe(winner ~., data = train_removed_downsample) %>%
	  step_mutate(x2013_code = factor(x2013_code)) %>%
	  step_dummy(x2013_code) %>%
	step_impute_median(all_numeric()) %>%
	  step_mutate(prop_white = x0064e / x0001e,
	        # black + american Indian + asian + native Hawaiian + some other race + hispanic
	        prop_minor = (x0064e + x0065e + x0066e + x0067e + x0068e + x0069e + x0071e)/
	x0001e,
	        prop_women = x0003e / x0001e,
	        prop_men = x0002e / x0001e,
	        prop_minor = x0019e / x0001e,
	        prop_college = (c01_005e + c01_013e + c01_018e + c01_021e + c01_024e + c01_027e) /
	x0001e,
	        prop_old = x0024e / x0001e) %>%

<table>
<tr><td></td><td>

```
  step_rm(x0064e) %>%
  step_rm(x0065e) %>%
  step_rm(x0066e) %>%
  step_rm(x0067e) %>%
  step_rm(x0068e) %>%
  step_rm(x0069e) %>%
  step_rm(x0071e) %>%
  step_rm(x0003e) %>%
  step_rm(x0002e) %>%
  step_rm(x0019e) %>%
  step_rm(x0020e) %>%
  step_rm(x0021e) %>%
  step_rm(x0022e) %>%
  step_rm(x0023e) %>%
  step_rm(x0024e) %>%
  step_rm(x0005e) %>%
  step_rm(x0006e) %>%
  step_rm(x0007e)
```

</td></tr>
<tr><td>Recipe 2</td><td>

```
oprop_rec <-
  recipe(winner ~ ., data = train) %>%
  update_role(id, new_role = "id") %>%
  step_naomit(all_predictors()) %>%
  step_mutate(
    prop_male = x0002e / x0001e,
    prop_female = x0003e / x0001e,
    prop_adult_male = x0088e / x0087e,
    prop_adult_female = x0089e / x0087e,
    prop_minor = x0019e / x0001e,
    prop_senior = x0024e / x0001e,
    prop_white = x0037e / x0001e,
    prop_mix_combo = (x0064e + x0065e + x0066e + x0067e + x0068e + x0069e)/ x0001e,
    prop_hispanic = x0071e / x0001e,
    prop_hs = (c01_003e + c01_004e + c01_017e + c01_020e + c01_023e + c01_026e) / x0001e,
    prop_b = (c01_005e + c01_018e + c01_021e + c01_024e + c01_027e) / x0001e
  ) %>%
  step_rm(
    x0002e, x0003e, x0088e, x0089e,
    x0024e,
    x0064e:x0071e,
    c01_003e:c01_005e, c01_017e, c01_018e, c01_020e, c01_021e,
    c01_023e, c01_024e, c01_026e, c01_027e
  ) %>%
  step_mutate(x2013_code = factor(x2013_code)) %>%
  step_rm(
    x0019e:x0031e, # cumulative
    x0033e, # dupe x0001e
    x0036e, # dupe x0034e
    x0058e, # dupe x0035e
    c01_006e:c01_016e, # cumulative & dupe x0010e
```

</td></tr>
</table>

	c01_019e, # dupe x0011e
	c01_022e, # dupe x0012e:x0014e
	c01_025e # dupe x0015e:x0017e
	) %>%
	step_rm(
	x0040e:x0043e, # native subcategory
	x0045e:x0051e, # asian subcategory
	x0053e:x0056e, # hawaii subcategory
	x0072e:x0075e, # hispanic subcategory
	x0076e:x0085e, # non-hispanic category
	) %>%
	step_impute_median(all_numeric_predictors()) %>%
	step_normalize(all_numeric_predictors()) %>%
	step_zv(all_predictors()) %>% # zero variance predictors
	step_nzv(all_predictors()) %>% # near-zero variance predictors
	step_dummy(all_nominal_predictors())

Candidate Model Number	**Model Identifier**	**Type of Model**	**Engine**	**Recipes or Listing of Other Variables in Model**:	**Hyperparameters**
1	xgboost2024	Boosted Tree	xgboost	Recipe 1	learn_rate = 0.00425, trees = 1857, tree_depth = 12, mtry =73, min_n = 2, loss_reduction = 0.00386, sample_size = 0.987, stop_iter = 4
2	stacks	Logistic Regression Random Forest Boosted Tree	glmnet randomForest xgboost	Recipe 1	Logistic Regression: penalty = 0.000287, mixture = 0.719 Random Forest: min_n = 4, trees = 104 Boosted Tree: learn_rate = 0.0664, trees = 1309, tree_depth = 4, mtry = 253, min_n = 7, loss_reduction = 0.000105, sample_size = 0.994, stop_iter = 19

					Blend Predictions: penalty = 10^-2
3	stacks_rf_xgboost	Random Forest Boosted Tree	randomForest xgboost	Recipe 1	Random Forest: min_n = 12, trees = 242, mtry = 15 Boosted Tree: learn_rate = 0.0102, trees = 1500, tree_depth = 14, mtry = 43, min_n = 4, loss_reduction = 0.00000479, sample_size = 0.527, stop_iter = 20 Blend Predictions: penalty = 10^-3
4	29_lbg	Boosted Tree	lightgbm	Recipe 2	N/A
5	mlp_nnet	MLP Model	nnet	Recipe 1	hidden_units = 7, penalty = 0, epochs = 600
6	log_reg_res	Logistic Regression	glmnet	Recipe 1	Penalty = 0.00000000127, Mixture = 0.0839

Evaluations and Tuning

Our candidate models mainly included logistic, random forest and boosted tree models. Models that include xgboost and lightgbm perform the best according to Kaggle leaderboard. We also tried MLP, however, that did not work as well with the workflow, most likely due to overfitting. In our top candidate models, we use latin hypercube sampling (LHS) to give us the most efficiency when it comes to finding the best parameters without overfitting. Some interesting things we noticed about LHS is that when comparing using LHS versus not for logistic regression is that there were many differences in accuracy or Kaggle score. LHS introduces randomness to the hyperparameter search which is more efficient when it comes to finding the best parameter combinations. One major challenge we came across with LHS is that it never produces the same exact parameters which causes our models to have different outcomes. Thus, we decided to manually input the best parameters from LHS instead of R selecting it.

To compare and evaluate the candidate models, we mainly used v-fold cross validation with 10 folds and stratified on the winner response variable. We assessed model performance based on the cross validation accuracy score, the standard error (if applicable), and the Kaggle score. From these results, we determined that the two best models were the stacks and 29_lbg models due to their high accuracy scores, relatively low SEs, and high scores when submitted to Kaggle.

We also implemented the stacks autoplot function to gain insights into which model generally produced the most accurate predictions. As shown in the plots, the ensemble models that combine logistic regression, random forest, and xgboost achieved the highest accuracy scores. The autoplot of the stack model revealed that logistic regression had the highest stacking coefficient, indicating its importance in the ensemble.

Results

Model Identifier	Accuracy Score from Cross Validation	SE (if applicable)	Kaggle Public Leaderboard Score (if submitted)
stacks	0.939	N/A	0.96652
29_lbg	0.939	0.0040529	0.94560
stacks_rf_xgboost	0.932	N/A	0.94560
xgboost2024	0.930	0.00594	0.94560
log_reg_res	0.935	0.00356	0.93305
mlp_nnet	0.834	0.000599	N/A

<u>Model Performance Evaluation Plots</u>

Autoplot Comparison of stacks_rf_xgboost, log_reg_res, mlp_nnet, xgboost2024, & 29_lbg

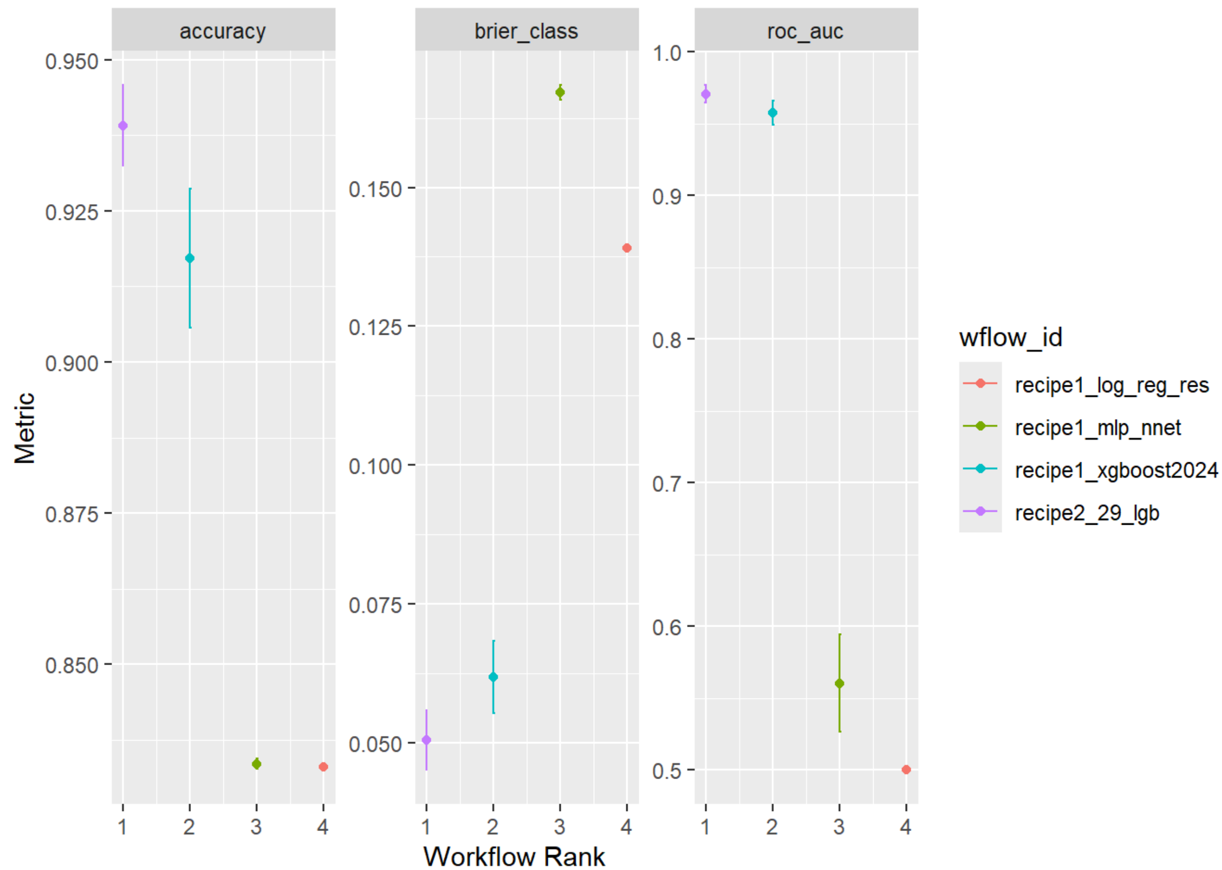
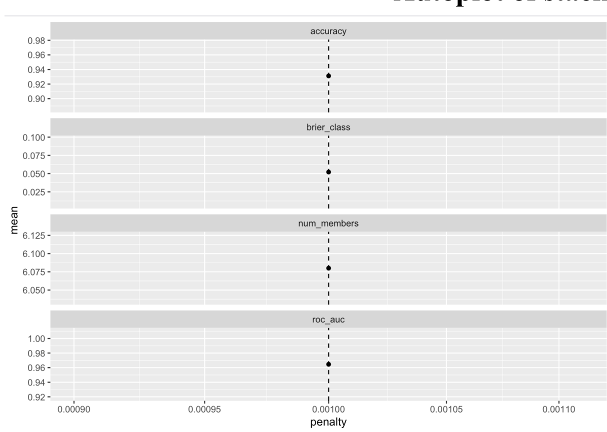
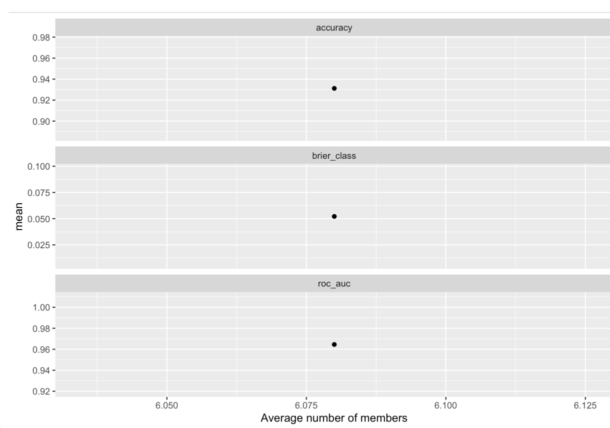


Figure 15

Autoplot of stacks_rf_xgboost Model





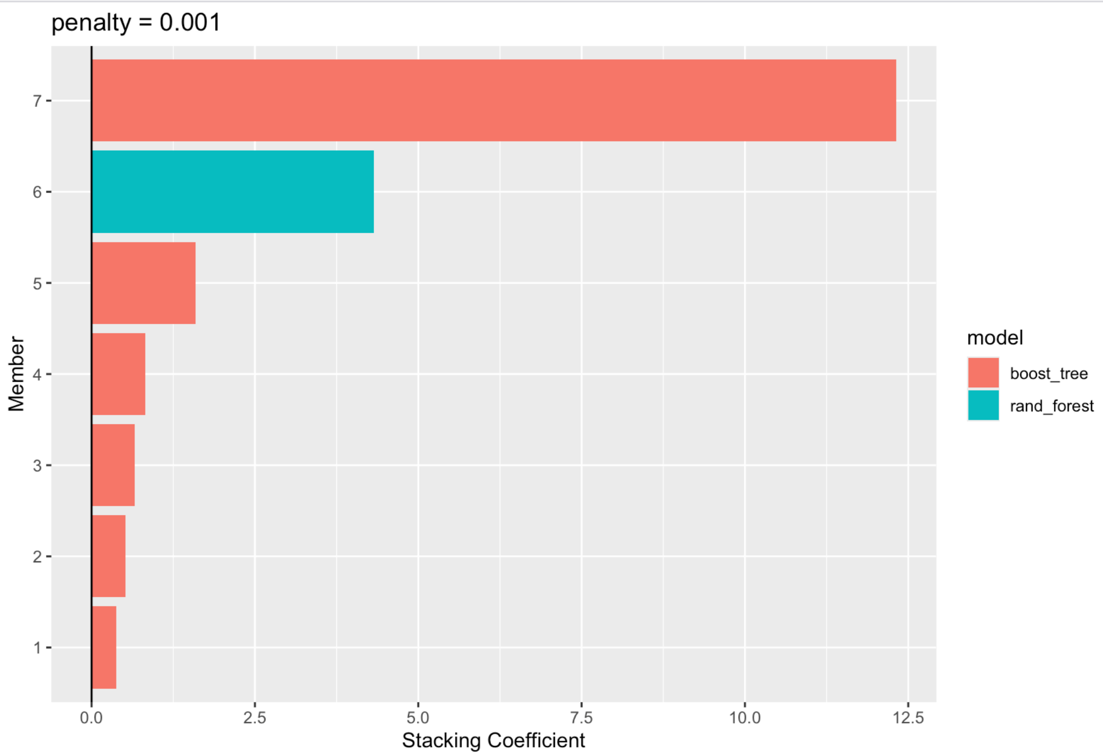
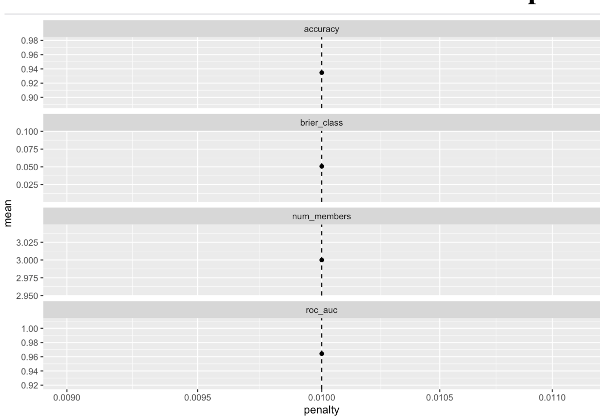
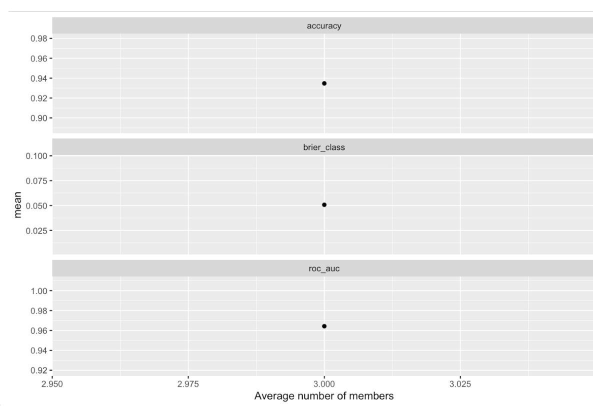


Figure 16 - 18

Autoplot of Stacks Model





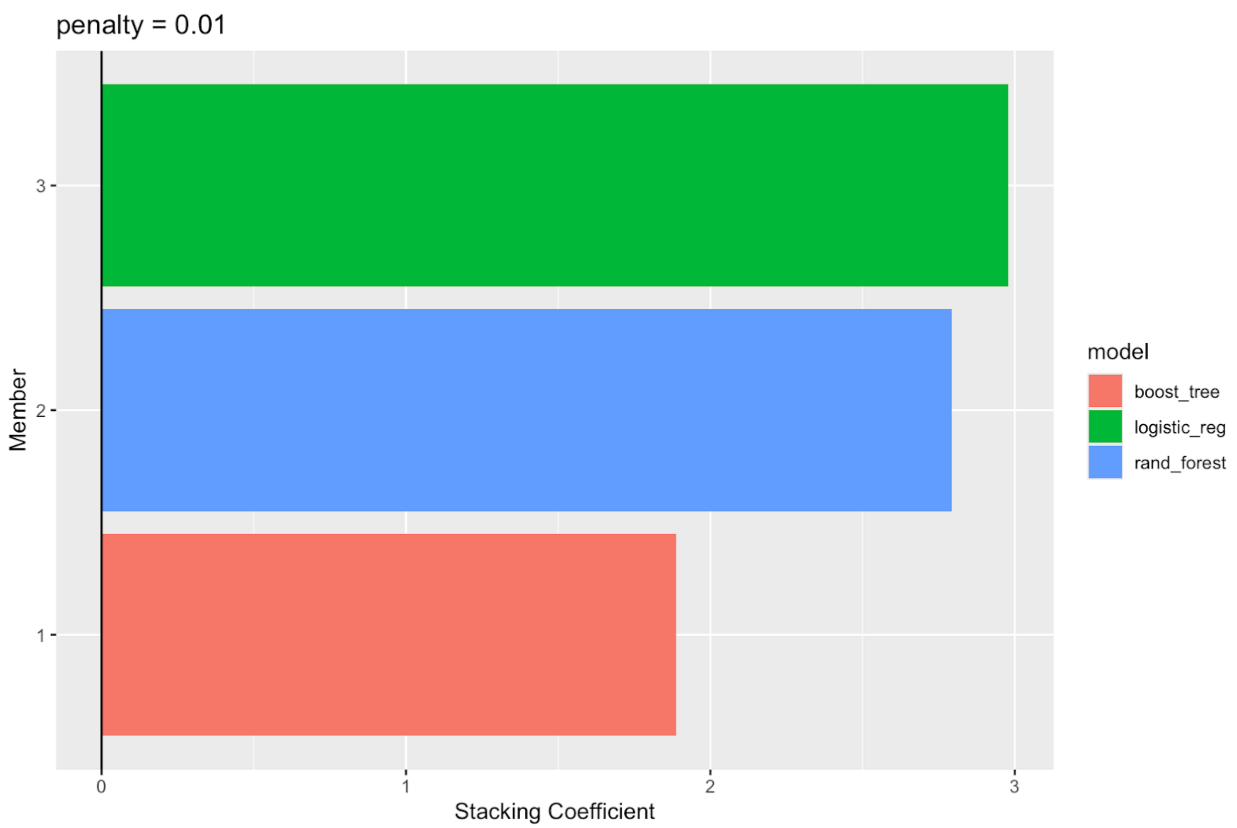


Figure 19-21

Discussion of Final Model

The two models selected for submission were 29_lbg and stacks, with 29_lbg responsible for our final private leaderboard score. These two models were selected because they produced the highest accuracy score based on cross-validation metrics. Additionally, we thought that the 29_lbg model itself would be able to compensate for the hypertuning we used in the stacks model. So, any overfitting the stacks model may have would be able to be balanced out by the 29_lbg model. After submitting both models as our final candidates, we found that our hypothesis about the 29_lbg model being able to avoid overfitting was actually correct. 29_lbg ended up being our better model according to the private leaderboard score.

The greatest strength of 29_lbg is the simplicity of the model. 29_lbg is a gradient boosting model (boosted trees) utilizing the lightgbm engine with all model parameters left at default. Ease of model application was appealing for this project since the data required complex preprocessing and recipes. Therefore, the simplicity of this setup leaves minimal points of failure for reproducibility since pseudorandomization only occurs during k-fold cross-validation. Another strength of this model is that it is able to capture nonlinear relationships, which allows it to better predict accurate results.

29_lbg has some weaknesses. As with many gradient-boosting models, lightgbm requires a sufficiently large dataset to make accurate predictions. The training set provided was on a national scale, which provided an adequate amount of observations, but the model would likely perform worse if the scope of observations was scaled down to the state level. Another weakness was that lightgbm required a specific complex recipe to make good predictions. During engine testing, the lightgbm engine's performance differed depending on the recipe applied, so applications or alterations to the model for different data may be time-consuming. Lastly, this lightgbm model, like other boosted tree models, is more difficult to interpret than other models such as logistic regression.

There are possible improvements that may benefit the 29_lbg model. First, testing and tuning an ensemble model using lightgbm may improve performance. Engine testing was very time-consuming for this project, so the performance of the lightgbm engine was discovered late into the project. However, even with a basic implementation of the lightgbm engine, 29_lbg and an ensemble model utilizing lightgbm were our 2nd and 3rd best models on the private leaderboard respectively. Another improvement we can make with our model is experimenting more with removing correlated predictors. While we did not have enough time to extensively play around with different recipes, we believe it would be worthwhile to consider converting a lot of the population predictors into proportions because a lot of the predictors overlapped; for instance, a lot of the people in the "Race alone or in combination with one or mother other races: Total population: White" (x0064e) variable would also be in the "Total population:One race:White" (x0037e) variable. Also, we would have liked to experiment more with removing the income_per_cap and gdp variables since a lot of them would be correlated with one another; for instance, gdp_2018 should be some type of multiple of gdp_2017.

Some additional data that could help us improve the model is data on immigration status. Although there is data on the proportion of citizens that are each race, we were curious about whether being a naturalized citizen or a U.S. born citizen impacts voting results. From the Public Policy Institute of California's article "Immigrants and Political Engagement," the article writes that naturalized citizens typically vote for Democrats. However, this is already assuming that naturalized citizens will go and vote. But, what if naturalized citizens tend to abstain from voting? This is a factor that we believe the current data failed to encapsulate. We were also curious about whether married status impacts voter turnout. In the article "Changing Partisan Coalitions in a Politically Divided Nation," Pew Research Center notes that married men and women are more likely to vote for the Republican Party. With a correlation as strong as this, we felt that our model could potentially be more accurate if we were able to have metrics that quantified this factor.

Appendix

Final annotated script

Final Model Submission
29_lgb.R by Oliver Siu

Screen recording of script running
https://drive.google.com/file/d/1YSZj_FL_N-6FIEMeKh28CMYizZYfj35B/view?usp=sharing

`29_lgb.R`

```
############################## 29_lgb.R ##############################

# load libraries and set preferences
library(tidyverse)
library(tidymodels)
tidymodels_prefer()
library(bonsai)
library(lightgbm)

# load data
train <- read_csv("train_class.csv")
test <- read_csv("test_class.csv")

# preprocessing
train <-
  train %>%
  select(!name) %>%
  mutate(winner = as.factor(winner))

# set seed
set.seed(2024)

# recipe setup
oprop_rec <-
  recipe(winner ~ ., data = train) %>%
  update_role(id, new_role = "id") %>%
  step_naomit(all_predictors()) %>%
  step_mutate(
    prop_male = x0002e / x0001e,
    prop_female = x0003e / x0001e,
    prop_adult_male = x0088e / x0087e,
    prop_adult_female = x0089e / x0087e,
```

```
    prop_minor = x0019e / x0001e,
    prop_senior = x0024e / x0001e,
    prop_white = x0037e / x0001e,
    prop_mix_combo = (x0064e + x0065e + x0066e + x0067e + x0068e +
x0069e)/ x0001e,
    prop_hispanic = x0071e / x0001e,
    prop_hs = (c01_003e + c01_004e + c01_017e + c01_020e + c01_023e +
c01_026e) / x0001e,
    prop_b = (c01_005e + c01_018e + c01_021e + c01_024e + c01_027e) /
x0001e
  ) %>%
  step_rm(
    x0002e, x0003e, x0088e, x0089e,
    x0024e,
    x0064e:x0071e,
    c01_003e:c01_005e, c01_017e, c01_018e, c01_020e, c01_021e,
    c01_023e, c01_024e, c01_026e, c01_027e
  ) %>%
  step_mutate(x2013_code = factor(x2013_code)) %>%
  step_rm(
    x0019e:x0031e, # cumulative
    x0033e, # dupe x0001e
    x0036e, # dupe x0034e
    x0058e, # dupe x0035e
    c01_006e:c01_016e, # cumulative & dupe x0010e
    c01_019e, # dupe x0011e
    c01_022e, # dupe x0012e:x0014e
    c01_025e # dupe x0015e:x0017e
  ) %>%
  step_rm(
    x0040e:x0043e, # native subcategory
    x0045e:x0051e, # asian subcategory
    x0053e:x0056e, # hawaii subcategory
    x0072e:x0075e, # hispanic subcategory
    x0076e:x0085e, # non-hispanic category
  ) %>%
  step_impute_median(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_zv(all_predictors()) %>% # zero variance predictors
  step_nzv(all_predictors()) %>% # near-zero variance predictors
  step_dummy(all_nominal_predictors())
```

```r
# model setup
lgb_spec <-
  boost_tree() %>%
  set_engine("lightgbm") %>%
  set_mode("classification")

# workflow setup
lgb_wflow <-
  workflow() %>%
  add_recipe(oprop_rec) %>%
  add_model(lgb_spec)

# folds for k-fold (v-fold) cross-validation and options setup
folds <- train %>% vfold_cv(v = 10, strata = winner)
keep_pred <- control_resamples(save_pred = TRUE, save_workflow = TRUE)

# k-fold (v-fold) cross-validation
lgb_res <-
  lgb_wflow %>%
  fit_resamples(resamples = folds, control = keep_pred)

# fit model to training data
lgb_fit <- lgb_wflow %>% fit(data = train)

# making predictions on test data
predictions <- lgb_fit %>% predict(new_data = test)

# format final output
solution <-
  test %>%
  select(id) %>%
  bind_cols(predictions) %>%
  rename(winner = .pred_class)

# write out final output as csv
write_csv(solution, "solution.csv")
```

Team member contributions

A. Katherine Huynh
   a. Also created stacks model which we ended up submitting as one of our candidate models
   b. Created xgboost2024 and stacks_rf_xgboost model
      i. Both of which tied for third place on the public leaderboard for 3 points extra credit
   c. Created Figure 1-5
   d. Created base recipe of figuring out which columns were duplicates
   e. Created workflow set of 3 recipes and 5 different models and ran a workflow map to see the models with the best predictive capability
   f. Typed the preprocessing section and the additional data section in the discussion of final model
B. Victoria Aye
   a. Tested and improved stack model, which resulted in being the highest public score out of all our submission and 2nd in overall
   b. Tested logistic regression model and wrote about it
   c. Cowrote evaluation, model tuning, and discussion of Final Model
C. Oliver Siu
   a. Designed 2 of the team's top 5 private scoring models
   b. Created Figures 8-10
   c. Created 29_lbg model
   d. Assisted with recipe research
   e. Responsible for engine research for rand_forest, logist_reg, & boost_tree
   f. Generated comparison plots for non-ensemble models
   g. Cowrote Discussion of Final Model
D. Vivian Yee
   a. Created Figures 6 and 7
   b. Tested models to improve public score on Kaggle (logistic regression, decision tree, boosted tree)
   c. Wrote introduction
   d. Cowrote evaluation and model tuning