

Group Project

Ben Noon and Rhodes Kirkpatrick

3/8/2022

```
library(pacman)
p_load(tidyverse, tidymodels, glmnet, rpart.plot, baguette, ranger, xgboost, modeldata, skimr, janitor,
final <- read_csv("final_data.csv",
  col_types = cols(Poverty_Percent = col_number(),
    `Preg_birth_rate_for_15-19_year_olds` = col_number(),
    Net_change_in_firms = col_number(),
    Life_Expectancy = col_number(), Diabetes_Rate = col_number(),
    Unemployment_in_June_2012 = col_number(),
    Voted_Republican = col_number(),
    Percent_White = col_number(), Percent_Black = col_number(),
    Percent_Naitive = col_number(), Percent_Asian = col_number(),
    Percent_Pacific_Islander = col_number(),
    Percent_Other = col_number(), Percent_two_or_more_Race = col_number(),
    Percent_Hispanic = col_number(),
    Median_Age = col_number(), Male = col_number(),
    Female = col_number(), Less_than_9th_grade = col_number(),
    `9-12_Grade` = col_number(), Hischool_diploma = col_number(),
    Some_College = col_number(), `Associate's_degree` = col_number(),
    `Bachelor's_degree` = col_number(),
    Graduate_or_professional_degree = col_number(),
    `2011_Poverty_Percent` = col_number()))

#Here the NAs in my dataset are removed. With such a small part of our data having NAs, the NAs are dr
final2 <- na.omit(final)
```

This paper will look at common demographic measurements and some economic variables to try and predict the poverty rate for each US county. This data looks at poverty rates for every US county in the year 2012.

Data set final contains 30 variables. 4 are id variables and 26 numeric variables.

State FIPS Code, County FIPS Code, Postal Code, and Name are all variables to help identify which county is being observed

Poverty Percent: Percent of the population in a county that lives in poverty

Preg birth rate for 15-19 year olds: This is the number of teen pregnancies per 1000 girls

net change in firms: How many firms overall entered or left the county in 2012

life expectancy: the average life expectancy in the county

diabetes rate: The number of people daignosed with diabetes per 1000 people

unemployment in june 2012: The unemployment rate in June 2012

voted republican: This is a binary variable that has 1 if the county voted republican in 2012

percent white - percent other: The percentage of the county that identifies as the one race. (so percent white is the percentage of the county who identifies as only white)

percent two or more races: The percentage of the county that identifies as two or more races

percent Hispanic: The percentage of the county that identifies ethnically as Hispanic

median age: the median age of the county

male: The average age of males in the county

female: The average age of females in the county

less than 9th grade - graduate or professional degree: These variables are the percentage of the county's population that attained the education of less than 9th grade to professional degree

2011 poverty percent: The poverty rate for each county in 2011

```
set.seed(1221891)

#The data is split into 80/20 for training and testing dataframes.

split_data <- final2 %>% initial_split(prop = 8/10, strata = Poverty_Percent)

validation_df <- split_data %>% testing()

training <- split_data %>% training()
```

```
#Recipe creation. The dependent variable is Poverty_Percent. The State, County, Postal Code, and Name are

recipe1 <- recipe(Poverty_Percent ~ ., data = training, importance = TRUE) %>%
  update_role(State_FIPS_Code, County_FIPS_Code, Postal_Code, Name, new_role = "id variable")

#Now that the recipe is made, I will cut my training data into five folds for cross validation.

set.seed(1221891)

fold <- training %>% vfold_cv(v = 5, strata = Poverty_Percent)
```

Elastinet Model

```
set.seed(1221891)

#I am tuning the penalty for my elastinet to be lambda and my mixture is being tuned through alpha.
lambdas <- 10^seq(from = 10, to = -10, length = 100)
alphas = seq(from = 0, to = 1, by = 0.01)

net <- linear_reg(
  penalty = tune(),
  mixture = tune()
) %>% set_engine("glmnet")

net_wf <- workflow() %>% add_recipe(recipe1) %>% add_model(net)
```

```
net_fit <- net_wf %>% tune_grid(
  fold,
  grid = expand_grid(mixture = alphas, penalty = lambdas),
  metrics = metric_set(rmse)
)

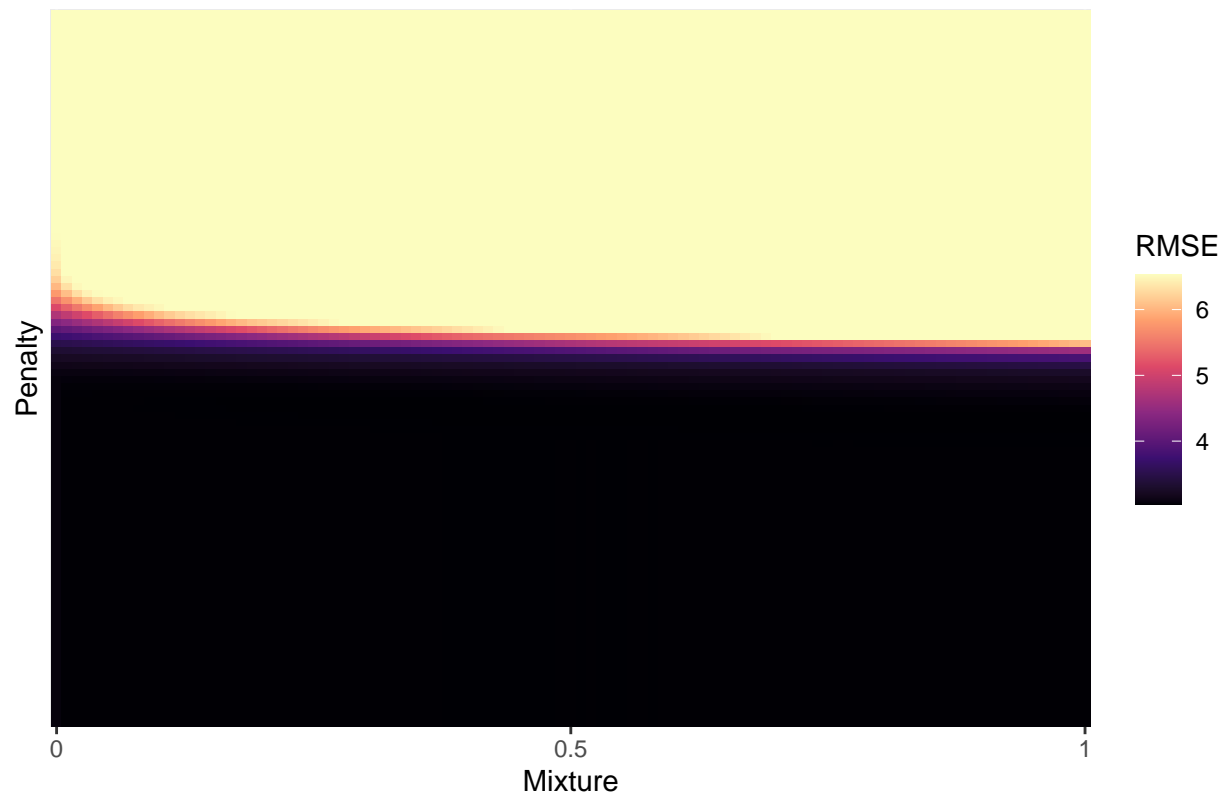
net_fit %>% show_best()
```

```
## # A tibble: 5 x 8
##   penalty mixture .metric .estimator  mean     n std_err .config
##   <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1  0.0486   0.65 rmse    standard    3.04     5  0.0673 Preprocessor1_Model065~
## 2  0.0486   0.64 rmse    standard    3.04     5  0.0672 Preprocessor1_Model064~
## 3  0.0486   0.66 rmse    standard    3.04     5  0.0673 Preprocessor1_Model066~
## 4  0.0486   0.67 rmse    standard    3.04     5  0.0673 Preprocessor1_Model067~
## 5  0.0486   0.63 rmse    standard    3.04     5  0.0672 Preprocessor1_Model063~
```

The best model for our training data has a penalty of 0.049, a mixture of 0.65, and has a rmse of 3.0.

```
net_fit %>% collect_metrics(summarize = TRUE) %>%
  ggplot(aes(x = as.factor(mixture), y = as.factor(penalty), fill = mean)) +
  geom_tile() +
  scale_x_discrete("Mixture", c(0, 0.5, 1)) +
  scale_y_discrete("Penalty", c(0.01, 10^5)) +
  scale_fill_viridis_c("RMSE", option = "magma") +
  ggtitle("RMSE based on tuning grid for elastic net model")
```

RMSE based on tuning grid for elastic net model



This graphs shows that mixture can be between 0 and 1 and still have a small rmse. The important hyperparameter is penalty.

```
#Now that we have our best model, lets take that model to our test data.

final_net <- net_wf %>% finalize_workflow(select_best(net_fit, metric = "rmse"))

validation_net <- final_net %>% last_fit(split_data)

validation_net %>% collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>       <dbl> <chr>
## 1 rmse    standard         3.19 Preprocessor1_Model1
## 2 rsq     standard         0.779 Preprocessor1_Model1
```

The prediction results have the rmse at 3.19 and rsq at 0.779

Decision Tree

```
# Define the decision tree
finaltree = decision_tree(
  mode = "regression",
  cost_complexity = tune(),
  tree_depth = tune(),
```

```

  min_n = 10 # Arbitrarily choosing '10'
) %>% set_engine("rpart")
#tune grid
tree_grid <- grid_regular(cost_complexity(), tree_depth(), levels = 4)

# Define the workflow
finalflow_tree = workflow() %>%
  add_model(finaltree) %>% add_recipe(recipe1)
# Tune!
final_cv_fit = finalflow_tree %>% tune_grid(
  fold,
  grid = expand_grid(
    cost_complexity = seq(0, 0.15, by = 0.01),
    tree_depth = c(1, 2, 5, 10),
  ))

final_cv_fit %>% show_best()

```

```

## # A tibble: 5 x 8
##   cost_complexity tree_depth .metric .estimator mean     n std_err .config
##         <dbl>         <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1             0             5 rmse    standard   3.41     5  0.0792 Preprocesso~
## 2            0.01             5 rmse    standard   3.48     5  0.101  Preprocesso~
## 3            0.01            10 rmse    standard   3.48     5  0.101  Preprocesso~
## 4             0            10 rmse    standard   3.67     5  0.111  Preprocesso~
## 5            0.02             5 rmse    standard   3.68     5  0.0612 Preprocesso~

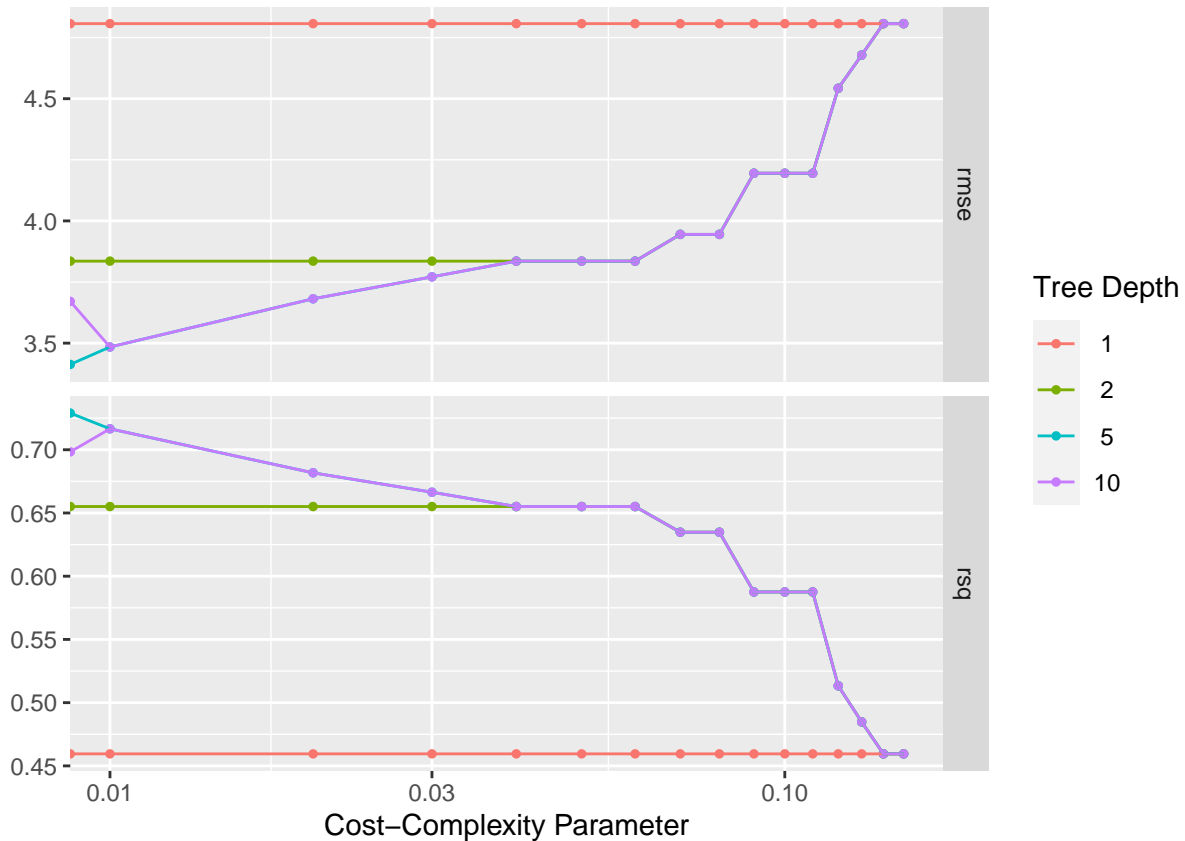
```

The training results with a cost complexity of 0, three depth of 5, and a rmse of 3.41.

```

final_cv_fit %>% autoplot()

```



The graph shows a better picture of how the models did over all. Its clear that increasing cost complexity of decrease the performance.

```
final_final_cv_fit =
  finalflow_tree %>%
  finalize_workflow(select_best(final_cv_fit, metric = "rmse"))

veryfinal_tree_cv = final_final_cv_fit %>% last_fit(split_data)
veryfinal_tree_cv%>% collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>       <dbl> <chr>
## 1 rmse    standard       3.38 Preprocessor1_Model11
## 2 rsq     standard       0.753 Preprocessor1_Model11
```

Single tree produced a rmse of 3.38 and a rsq of 0.753

```
#random forest model

set.seed(1221891)

tune_random <- rand_forest(
  mtry = tune(),
  trees = 500,
  min_n = tune())
```

```
) %>% set_engine("ranger") %>% set_mode("regression")
```

```
random_wf <- workflow() %>%  
  add_recipe(recipe1) %>%  
  add_model(tune_random)
```

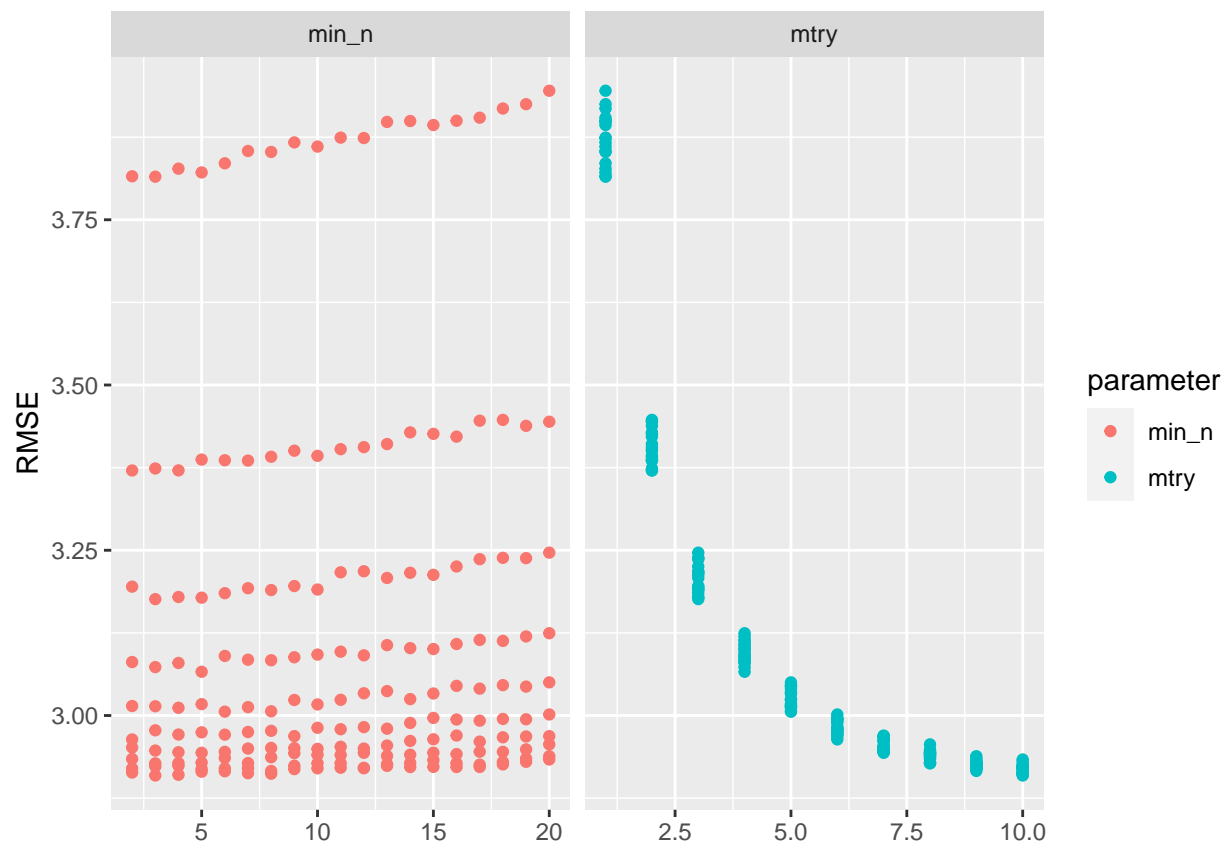
```
fit_random <- random_wf %>%  
  tune_grid(  
    fold,  
    grid = expand_grid(mtry = c(1:10),  
                      min_n = c(2:20)),  
    metrics = metric_set(rmse)  
  )
```

```
fit_random %>% select_best()
```

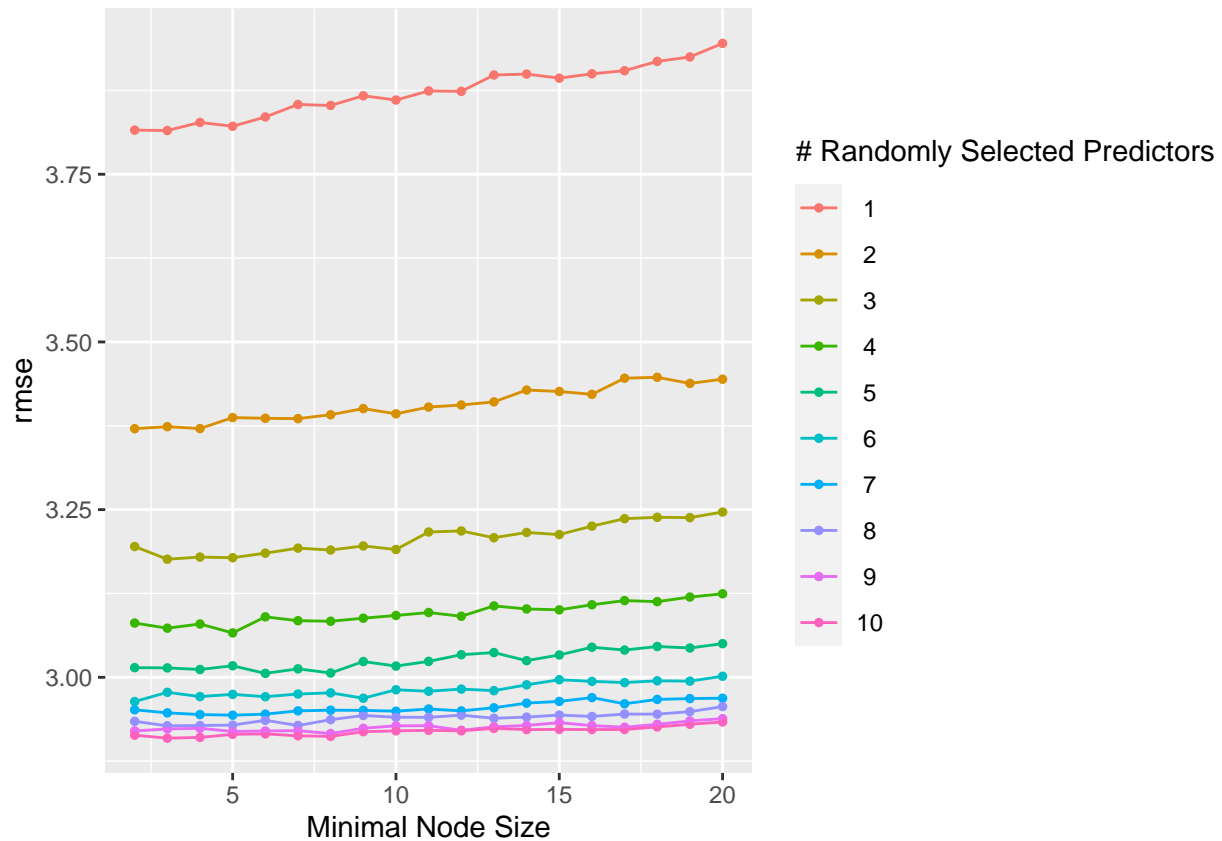
```
## # A tibble: 1 x 3  
##   mtry min_n .config  
##   <int> <int> <chr>  
## 1     10     3 Preprocessor1_Model173
```

The best model had a mtry of 10 and a min_n of 4

```
fit_random %>%  
  collect_metrics(summarize = T) %>%  
  filter(.metric == "rmse") %>%  
  select(mean, min_n, mtry) %>%  
  pivot_longer(min_n:mtry,  
               values_to = "value",  
               names_to = "parameter") %>%  
  ggplot(aes(value, mean, color = parameter)) +  
  geom_point() + facet_wrap(~parameter, scales = "free_x") + labs(x = NULL, y = "RMSE")
```



```
autoplot(fit_random)
```

The second graph shows that `min_n` has much less variation than `mtry`. `mtry` seems to converge to/around 10 to minimize rmse.

```
final_random <- random_wf %>% finalize_workflow(select_best(fit_random, metric = "rmse"))
final_fit_raondom <- final_random %>% last_fit(split_data)
final_fit_raondom %>% collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>       <dbl> <chr>
## 1 rmse    standard         2.92 Preprocessor1_Model11
## 2 rsq     standard         0.816 Preprocessor1_Model11
```

The rmse for the random forest is 2.92 and the rsq is 0.816.

Random forest was able to really minimize error on the testing data. Next we will see which variables were important to best model.

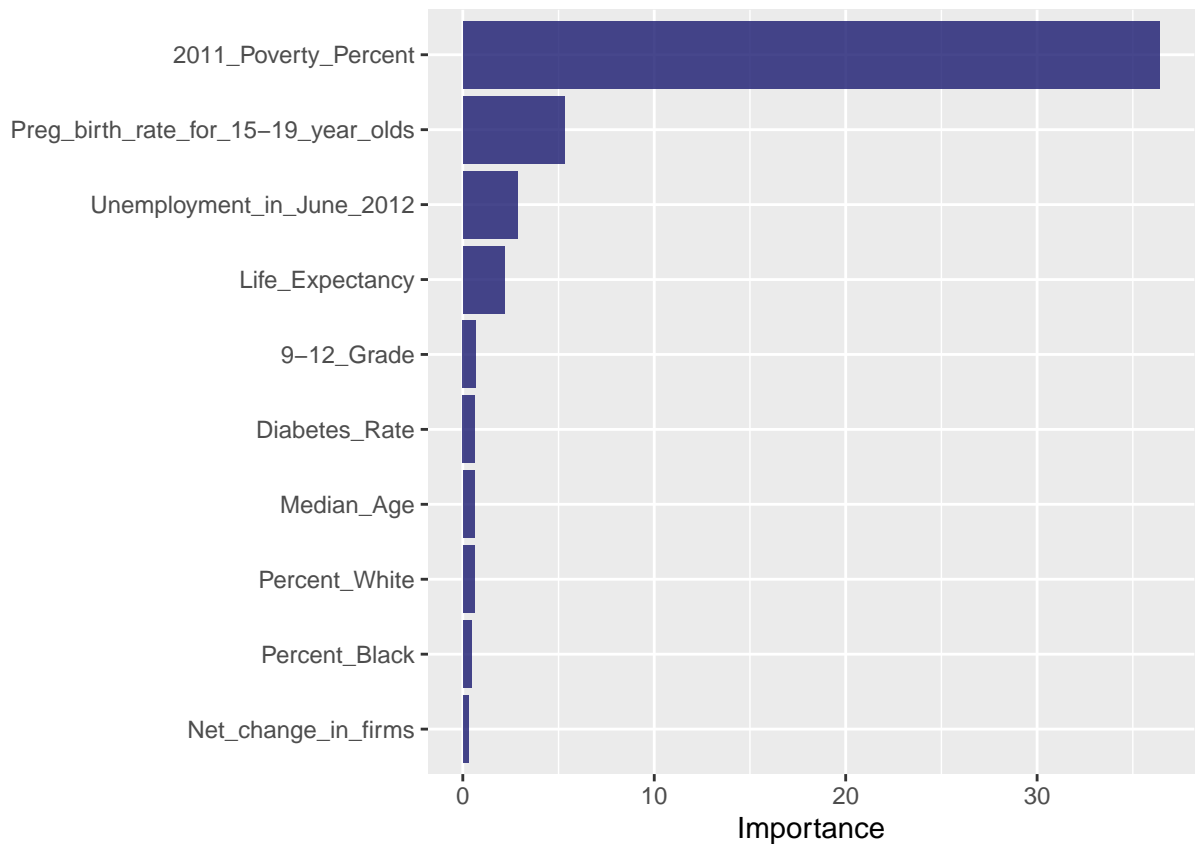
```
set.seed(1221891)

imp_spec <- tune_random %>%
  finalize_model(select_best(fit_random)) %>%
  set_engine("ranger", importance = "permutation")
```

```

workflow() %>%
  add_recipe(recipe1) %>%
  add_model(imp_spec) %>%
  fit(training) %>%
  extract_fit_parsnip() %>%
  vip(aesthetics = list(alpha = 0.8, fill = "midnightblue"))

```



Looking at the graph above, knowing poverty from the year before is very important. A surprise important measure is teen pregnancy being the second most common variable. Unemployment and life expectancy coming up after teen pregnancy. It is not surprising only white and black race groups were important. The other demographics had very small if any presence in a majority of the counties. Diabetes rate had a much smaller impact than expected, and college didn't even matter at all.

Boosted Trees

```

#Define Boosted Tree Model
finalboost <- boost_tree(
  mode = "regression",
  trees = tune(),
  tree_depth = tune(),
  min_n = tune(),
  learn_rate = tune(),
  mtry = 4.9
) %>% set_engine("xgboost")

```

```

#Define Workflow
boost_wf <- workflow() %>% add_recipe(recipe1) %>% add_model(finalboost)

#Fit the workflow via tuning
finalboost_fit <-boost_wf %>% tune_grid(
  fold,
  grid = expand_grid(trees = c(10,50,100),
                    min_n = c(1,5,10),
                    tree_depth = c(3:10),
                    learn_rate = c(0.1,0.01)),
  metrics = metric_set(rmse),
  verbosity = 0
)

```

```

## Warning: The '...' are not used in this function but one or more objects were
## passed: 'verbosity'

```

```

## [00:41:29] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:29] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:30] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:30] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:31] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:31] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:31] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:31] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:32] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:32] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:33] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:33] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:34] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:34] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:35] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:35] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:36] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:36] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:37] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:37] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:37] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:37] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:38] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:38] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:39] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:39] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:40] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:40] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:41] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:41] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:43] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:43] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:43] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:43] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:44] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:41:44] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat

```


[illegible]

[illegible]

[illegible]

[illegible]


```
## [00:44:50] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:44:50] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:44:51] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:44:51] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:44:52] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:44:52] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:44:53] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:44:53] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:44:54] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:44:54] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:44:55] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
## [00:44:55] WARNING: amalgamation/./src/c_api/c_api.cc:718: 'ntree_limit' is deprecated, use 'iterat
```

```
finalboost_fit %>% show_best()
```

```
## # A tibble: 5 x 10
##   trees min_n tree_depth learn_rate .metric .estimator mean      n std_err
##   <dbl> <dbl>      <int>      <dbl> <chr>    <chr>      <dbl> <int>  <dbl>
## 1   100     5         8        0.1 rmse    standard    2.96     5  0.0838
## 2   100    10         8        0.1 rmse    standard    2.96     5  0.0782
## 3   100    10        10        0.1 rmse    standard    2.97     5  0.0750
## 4   100     1         7        0.1 rmse    standard    2.97     5  0.0778
## 5   100     5         7        0.1 rmse    standard    2.98     5  0.102
## # ... with 1 more variable: .config <chr>
```

The boost's best model in training has trees at 100, min_n at 10, learn rate of 0.1, and a rmse of 2.96

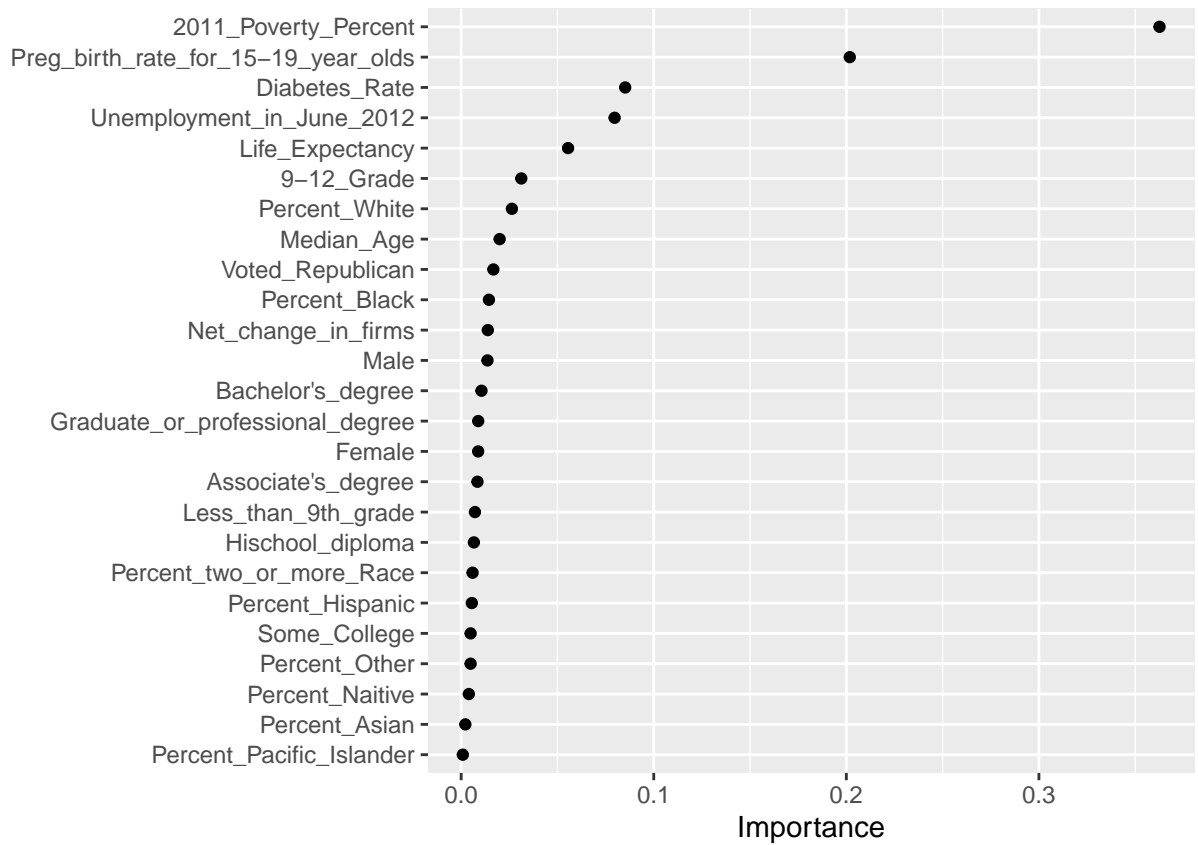
```
final_finalboost_fit =
  boost_wf %>%
  finalize_workflow(select_best(finalboost_fit, metric = "rmse"))
```

```
veryfinalboost_cv = final_finalboost_fit %>% last_fit(split_data)
veryfinalboost_cv%>% collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>    <chr>      <dbl> <chr>
## 1 rmse    standard      3.07 Preprocessor1_Model1
## 2 rsq     standard      0.795 Preprocessor1_Model1
```

Boosted trees resulted in a rmse of 3.01 and a rsq of 0.802.

```
extract_workflow(veryfinalboost_cv) %>%
  extract_fit_parsnip() %>%
  vip(geom = "point", num_features = 25)
```



The boosted model had a very similar set of important variables. Interestingly, whether a county voted republican jumped in importance.