final

2022-11-17

Introduction Section:

The purpose of this project is to generate a model to predict the winpercent of a candy brand based on predictors, and find out which predictor is the most influential on the winpercet. Winpercent is the percent that a candy brand received from customer services, higher percent means that candy brand is more popular.

Candy.csv

My data is from the Halloween Candy Power Ranking. It has 12 variables, including chocolate, fruity, caramel, peanutyalmondy, nougat, crispedricewafer, hard, bar, pluribus in form of 0/1, indicate whether the candy has that property or not. It also has sugarpercent and pricepercent. I will use above 11 variables to predict the final winpercent to test the accuracy.The dataset is from FiveThirtyEight Website: https://fivethirtyeight.com/videos/the-ultimate-halloween-candy-power-ranking/ Also there is a github data link: https://github.com/fivethirtyeight/data/tree/master/candy-power-ranking

The key variables are listed below.

`chocolate`: Does it contain chocolate?

`fruity`: Is it fruit flavored?

`caramel`: Is there caramel in the candy?

`peanutalmondy`: Does it contain peanuts, peanut butter or almonds?

`nougat`: Does it contain nougat?

`crispedricewafer`: Does it contain crisped rice, wafers, or a cookie component?

`hard`: Is it a hard candy?

`bar`: Is it a candy bar?

`pluribus`: Is it one of many candies in a bag or box?

`sugarpercent`: The percentile of sugar it falls under within the data set.

`pricepercent`: The unit price percentile compared to the rest of the set.

`winpercent`: The overall win percentage according to 269,000 matchups.

load data:

```
data <- read_csv(file = "candy.csv")
head(data)
```

```
## # A tibble: 6 x 12
##   chocolate fruity caramel peanutya~1 nougat crisp~2  hard   bar pluri~3 sugar~4
##       <dbl>  <dbl>   <dbl>      <dbl>  <dbl>   <dbl> <dbl> <dbl>   <dbl>   <dbl>
## 1         1      0       1          0      0       1     0     1       0   0.732
## 2         1      0       0          0      1       0     0     1       0   0.604
## 3         0      0       0          0      0       0     0     0       0   0.011
## 4         0      0       0          0      0       0     0     0       0   0.011
```

```
## 5         0      1       0          0      0       0     0     0       0   0.906
## 6         1      0       0          1      0       0     0     1       0   0.465
## # ... with 2 more variables: pricepercent <dbl>, winpercent <dbl>, and
## #   abbreviated variable names 1: peanutyalmondy, 2: crispedricewafer,
## #   3: pluribus, 4: sugarpercent
```

Check Missing Values:

```
cbind(
   lapply(
     lapply(data, is.na)
     , sum)
   )
```

```
##                  [,1]
## chocolate        0
## fruity           0
## caramel          0
## peanutyalmondy   0
## nougat           0
## crispedricewafer 0
## hard             0
## bar              0
## pluribus         0
## sugarpercent     0
## pricepercent     0
## winpercent       0
```

Data Split

Training set will be 70 percent of the data and the testing set will be 30 percent of the data. Moreover, the splitting is stratified on the winpercent which is the output variable

```
set.seed(3435)
data_split <- data %>%
  initial_split(prop = 0.7, strata = "winpercent")

data_train <- training(data_split)
data_test <- testing(data_split)

dim(data_train)
```

```
## [1] 57 12
```

```
dim(data_test)
```

```
## [1] 28 12
```

The size of the training set is 57, the size of the testing set size is 28.

Exploratory Data Analysis

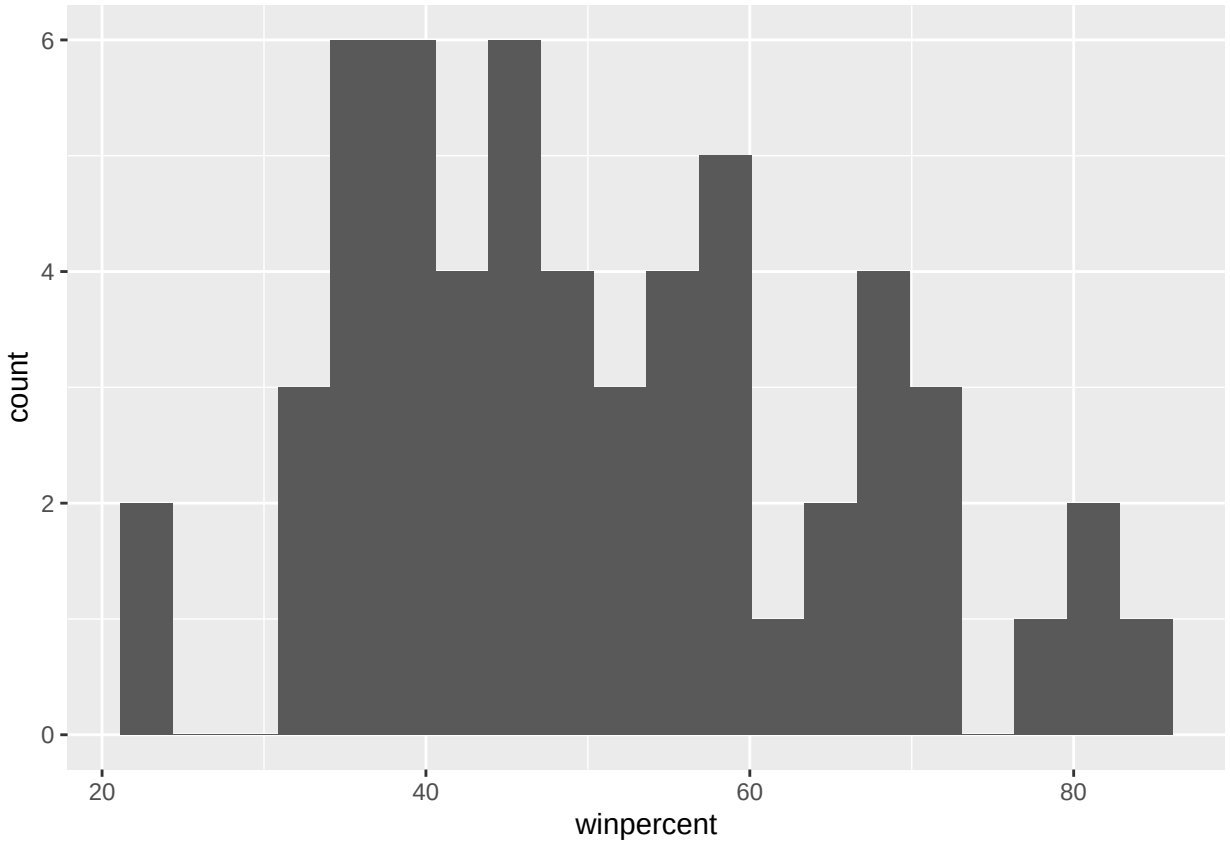
```
data %>%
  head()
```

```
## # A tibble: 6 x 12
##   chocolate fruity caramel peanutya~1 nougat crisp~2  hard   bar pluri~3 sugar~4
```

```
##       <dbl>  <dbl>   <dbl>      <dbl>  <dbl>   <dbl> <dbl> <dbl>   <dbl>   <dbl>
## 1         1      0       1          0      0       1     0     1       0   0.732
## 2         1      0       0          0      1       0     0     1       0   0.604
## 3         0      0       0          0      0       0     0     0       0   0.011
## 4         0      0       0          0      0       0     0     0       0   0.011
## 5         0      1       0          0      0       0     0     0       0   0.906
## 6         1      0       0          1      0       0     0     1       0   0.465
## # ... with 2 more variables: pricepercent <dbl>, winpercent <dbl>, and
## #   abbreviated variable names 1: peanutyalmondy, 2: crispedricewafer,
## #   3: pluribus, 4: sugarpercent
```

First we need to see how the data looks like, the first 9 variables are all in the forms of 0 or 1, means it contains that variables or not. It also has three percent value variables.

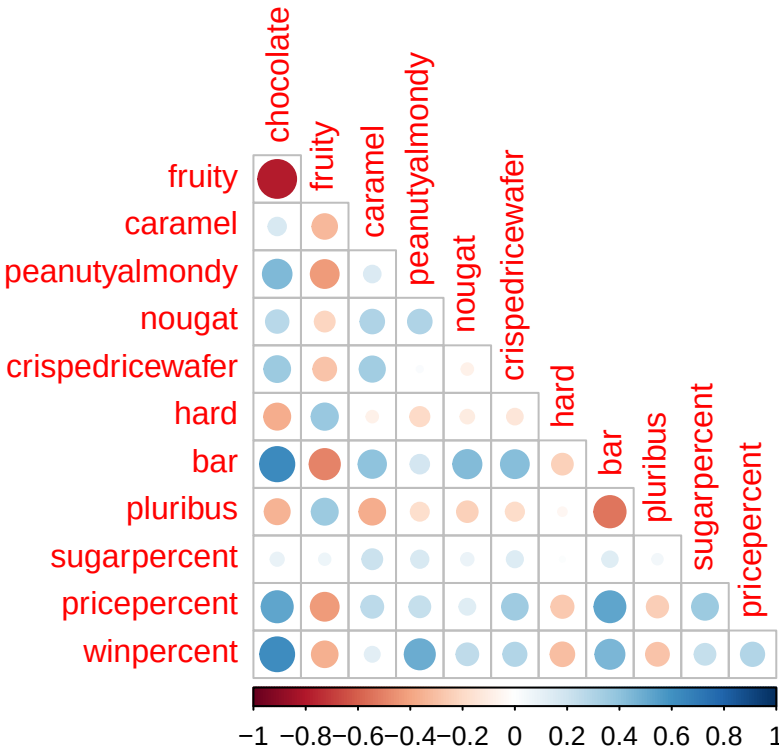
```
ggplot(data_train, aes(x=winpercent)) +
  geom_histogram(bins = 20)
```



Since there are only 2 obs near 20 winpercent, I cannot use strata in later data split part since the original data set is too small.

Correlation Matrix:

```
data_train %>%
  select(is.numeric) %>%
  cor() %>%
  corrplot(type = 'lower', diag = FALSE)
```



We can see the winpercent has strong positive relation with `chocolate`, `peanutyalmondy`, `bar`, moderate positive relation with `pricepercent`, `sugarpercent`, moderate negative relation with `fruity`, `hard`, and `pluribus`. I decide to remove `caramel`, `nougat`,and `sugarpercent` because theyt are too low relate to winpercent.

Data Split:

Split into 5 folds: Use Cross validation to tune the model and select the best model using `rmse` as standard

```
data_folds <- vfold_cv(data_train, v = 5)
```

Model Building

```
data_recipe <- recipe(winpercent ~ chocolate + fruity + peanutyalmondy + crispedricewafer + hard + bar +


data_recipe <- data_recipe %>%
  step_scale(chocolate, fruity, peanutyalmondy,crispedricewafer, hard,bar, pluribus, pricepercent)

data_recipe <- data_recipe %>%
  step_center(chocolate, fruity, peanutyalmondy, crispedricewafer, hard,bar, pluribus, pricepercent)
```

Linear Regression: Because the data are composed by numerical variables, I use the linear regression model first:

```
lm_model <- linear_reg() %>%
  set_engine("lm")

#set up workflow for the model
linear_wkf <- workflow() %>%
  add_recipe(data_recipe) %>%
  add_model(lm_model)
```

```
#fit the model
linear_fit <- fit_resamples(linear_wkf, data_folds)

#evaluate performance
collect_metrics(linear_fit)
```

```
## # A tibble: 2 x 6
##   .metric .estimator   mean     n std_err .config
##   <chr>   <chr>       <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   13.2       5   1.28  Preprocessor1_Model1
## 2 rsq     standard    0.348     5   0.138 Preprocessor1_Model1
```

The mean rmse acorss fold is 13.188

```
sd(data_train$winpercent)
```

```
## [1] 14.88768
```

Compare with the standard deviation with the rmse, standard deviation is bigger, so the linear regression model is valid.

The second model I use is ridge regression:

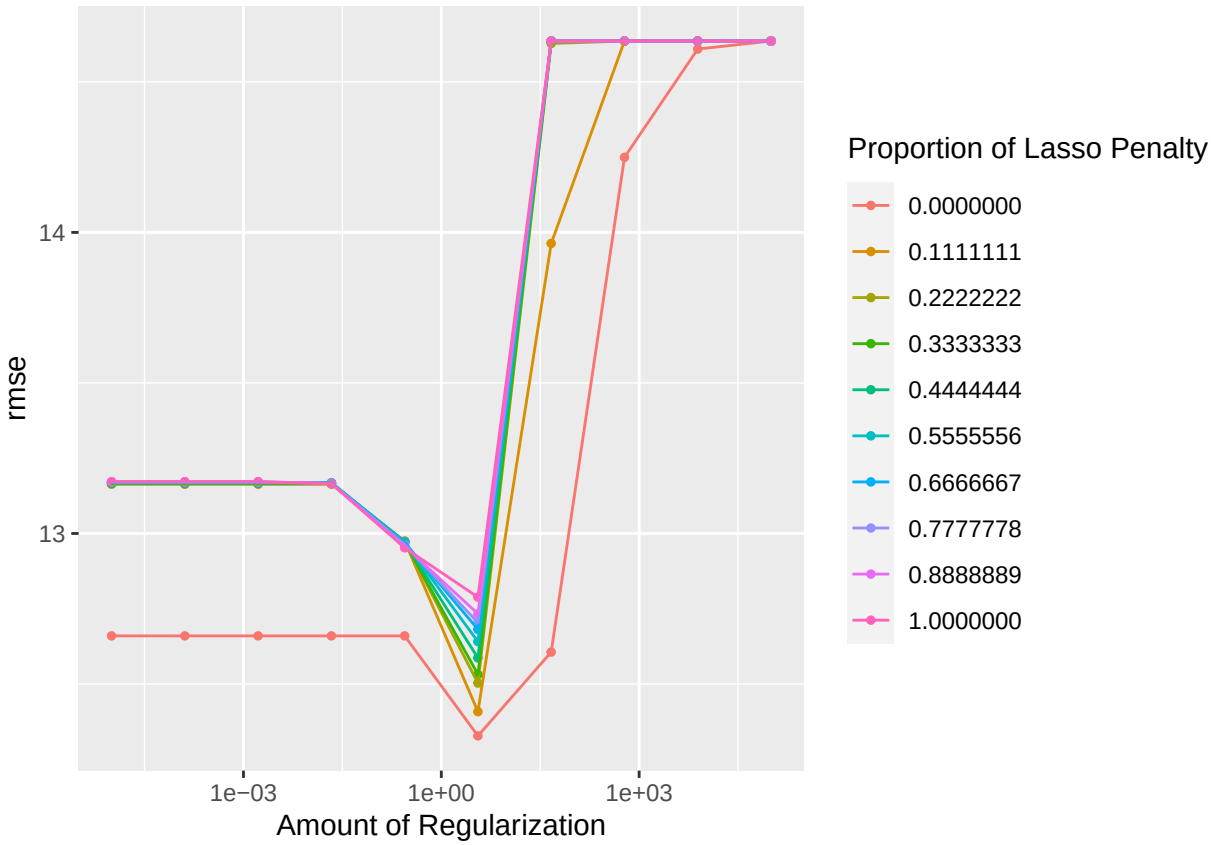
```
ridge_spec <- linear_reg(mixture = tune(), penalty = tune()) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

#set up workflow for the model
ridge_wkf <- workflow() %>%
  add_recipe(data_recipe) %>%
  add_model(ridge_spec)

#set up grid for tuning
ridge_grid <- grid_regular(penalty(range = c(-5, 5)),
                           mixture(range = c(0, 1)), levels = 10)

#tune the model
ridge_res <- tune_grid(
  ridge_wkf,
  resamples = data_folds,
  grid = ridge_grid,
  metrics = metric_set(rmse)
)

autoplot(ridge_res)
```



From this graph we can see the rmse value increase as penalty increase. Model with low mixture value, rmse increases slower.

```
collect_metrics(ridge_res) %>%
  head()
```

```
## # A tibble: 6 x 8
##    penalty mixture .metric .estimator  mean     n std_err .config
##      <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 0.00001        0 rmse    standard    12.7     5    1.15 Preprocessor1_Model001
## 2 0.000129       0 rmse    standard    12.7     5    1.15 Preprocessor1_Model002
## 3 0.00167        0 rmse    standard    12.7     5    1.15 Preprocessor1_Model003
## 4 0.0215         0 rmse    standard    12.7     5    1.15 Preprocessor1_Model004
## 5 0.278          0 rmse    standard    12.7     5    1.15 Preprocessor1_Model005
## 6 3.59           0 rmse    standard    12.3     5    1.23 Preprocessor1_Model006
```

```
best_penalty <- select_best(ridge_res, metric = "rmse")
ridge_final <- finalize_workflow(ridge_wkf, best_penalty)
ridge_final_fit <- fit(ridge_final, data = data_train)
augment(ridge_final_fit, new_data = data_test) %>%
  rmse(truth = winpercent, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        11.5
```

Since we choose the best penalty, the lowest rmse value is 11.545.

The third method I use is boost tree model.

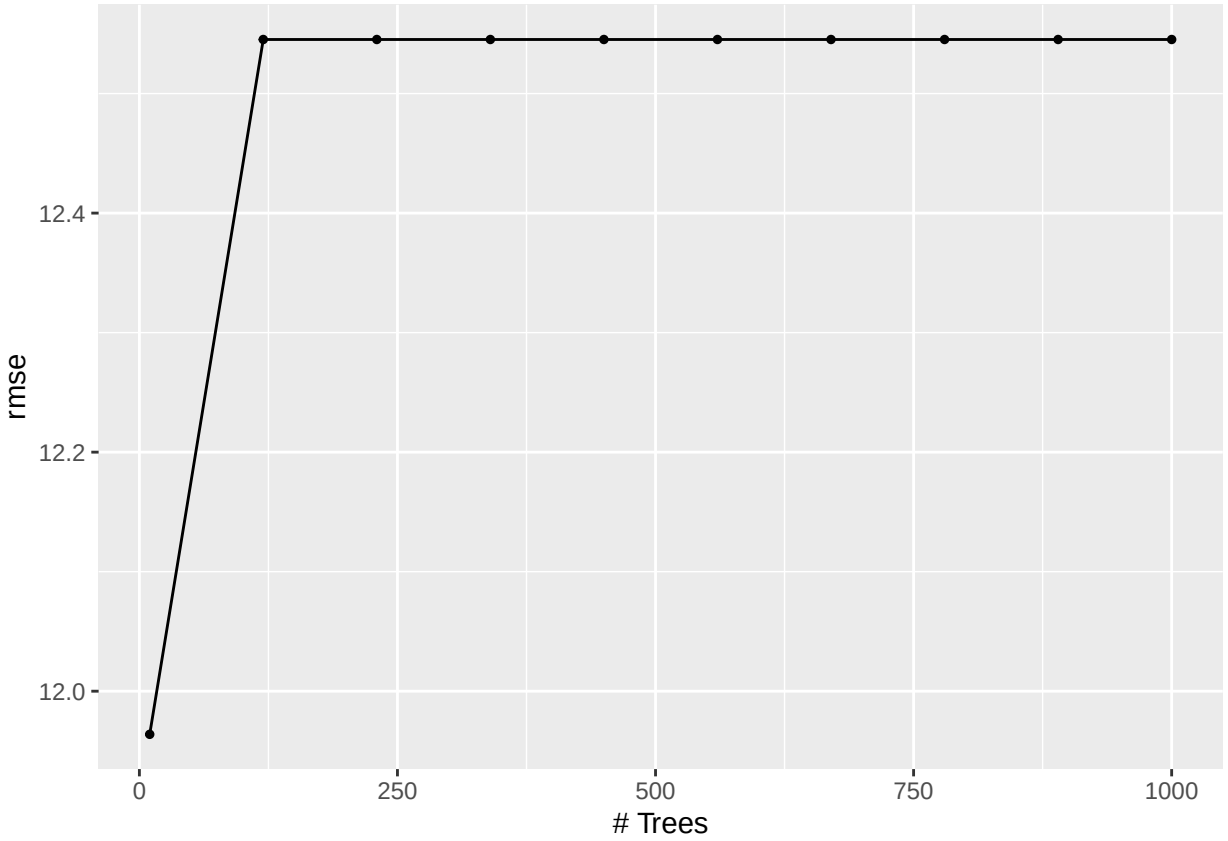
```r
#set up boost tree model
boost_model <- boost_tree(trees = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("regression")

#set up workflow for the model
boost_wkf <- workflow() %>%
  add_recipe(data_recipe) %>%
  add_model(boost_model)

#set up the grid for tuning
boost_grid <- grid_regular(trees(range = c(10, 1000)), levels = 10)

#tune the model
tune_res_boost <- tune_grid(
  boost_wkf,
  resamples = data_folds,
  grid = boost_grid,
  metrics = metric_set(rmse)
)

autoplot(tune_res_boost)
```




```r
boost_matrix <- collect_metrics(tune_res_boost) %>% arrange(mean)
boost_matrix[1,]
```

```
## # A tibble: 1 x 7
```

```
##   trees .metric .estimator  mean     n std_err .config
##   <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    10 rmse    standard    12.0     5   0.964 Preprocessor1_Model01
```

The mean `rmse` value across folds is 11.96381.

Random forest :

```
library(discrim)
library(poissonreg)
library(corrr)
```

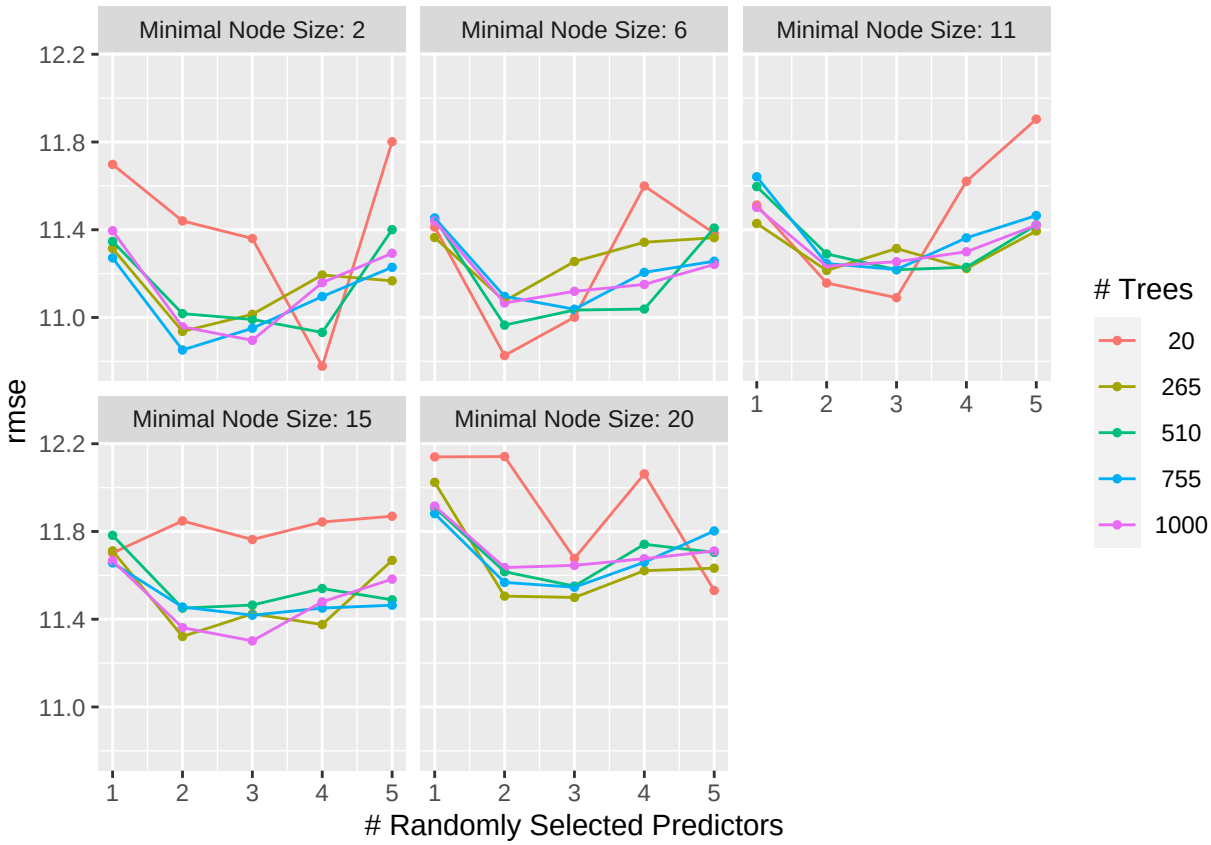
```
random_model <-
  rand_forest(
              min_n = tune(),
              mtry = tune(),
              trees = tune(),
              mode = "regression") %>%
  set_engine("ranger", importance = 'impurity')

#set up the workflow for the model
random_wkf <- workflow() %>%
  add_recipe(data_recipe) %>%
  add_model(random_model)

#build the grid for tuning
random_grid <- grid_regular(mtry(range = c(1, 5)),
                            trees(range = c(20, 1000)),
                            min_n(range = c(2, 20)),
                            levels = 5)


#Tune the model
tune_res_random <- tune_grid(
  random_wkf,
  resamples = data_folds,
  grid = random_grid,
  metrics = metric_set(rmse)
)

#Show the result with lowest rmse
autoplot(tune_res_random)
```




```
random_matrix <- collect_metrics(tune_res_random) %>% arrange(mean)
random_matrix[1,]
```

```
## # A tibble: 1 x 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     4    20     2 rmse    standard    10.8     5    1.04 Preprocessor1_Model0~
```

We can see the minimum value of rmse occur when trees=510 and node=2, the value is 10.86772, it has the minimum rmse value compare with the other three models.

```
Model <- c('Linear Regression', 'Boost Tree', 'Lasso', 'Random Forest')
RMSE <- c(13.188, 11.54502  , 11.96391, 10.86772)
Model_Comparison <- data.frame(Model, RMSE)
Model_Comparison
```

```
##               Model     RMSE
## 1 Linear Regression 13.18800
## 2        Boost Tree 11.54502
## 3             Lasso 11.96391
## 4     Random Forest 10.86772
```

Based on the comparison, random forest model did the best because of the smallest RMSE number.

```
#select the best model
final_model <- select_best(tune_res_random)

#set up workflow for the last model
final_wkf <- finalize_workflow(random_wkf, final_model)
```

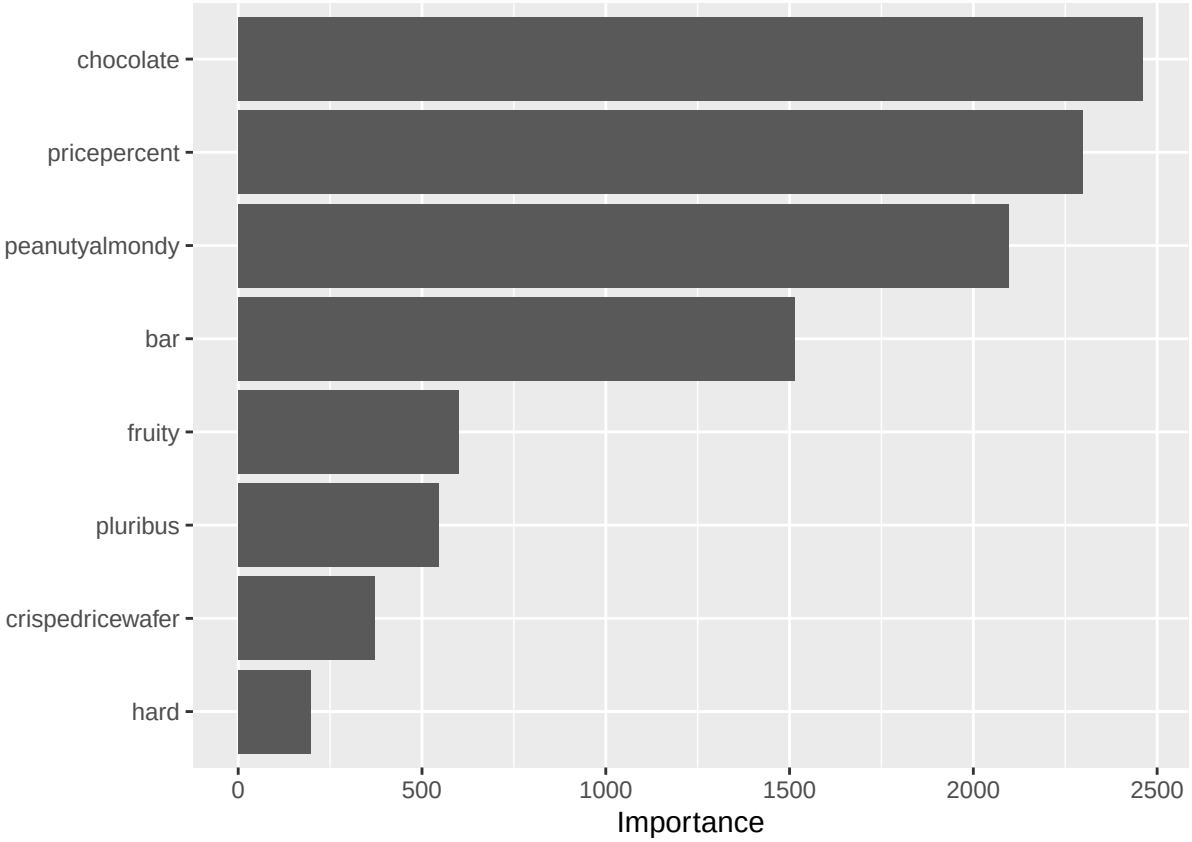
```r
#fit the model to the training set
final_fit <- fit(final_wkf, data = data_train)

#Evaluate the model performance on the testing set
multi_metric <- metric_set(rmse, rsq, mae)
final_predict <- predict(final_fit, data_test) %>%
  bind_cols(data_test %>% select(winpercent))
multi_metric(final_predict, truth = winpercent, estimate = .pred)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      10.6
## 2 rsq     standard       0.452
## 3 mae     standard       8.16
```

By examining the prediction result, we can find out that the random forest have a relatively good prediction performance on predicting the winpercent of candy brand. The Root mean square error is 10.5 which is relatively small but still large. The R-square is 0.46 which means that less than 50 percent of variance can be explained by the model which is relatively low. It may because of the winpercent of candy is hard to predict due to customer's different preference, cannot simply predict by its components or properties.

```r
final_fit %>% extract_fit_engine() %>% vip()
```



We can see chocolate is the most important one, and pricepercent, peanutyalmondy, and bar are also very important

Conclusion

The random forest model performed the best between the Boost Tree, Lasso regression, and the Linear Regression perform poorly. I'm not surprised by the model performance.

The first step is to do a data cleaning. I organize the variable names and exclude the variables that are not important: `caramel`, `nougat`,and `sugarpercent`. Then I check if there are any missing values, and the result turns out that there are no missing values. After that, I split data into training set and testing set stratified by happiness score which is the output variable. The second step is to do an exploratory data analysis. I build a correlation matrix to see the correlation between variables.The third step is to build models and select the best model between them and the standard is rmse. I split data into five folds. I build linear regression, lasso regression, boost trees, and random forest. I use the cross validation to tune the models and select the model with lowest average rmse across all folds. At last, I use the variable importance plots to determine the most significant predictor.

The general conclusion is that the winpercent of candy is hard to predict. In real life, due to customers' different preference, like area difference, history difference, and so on, customers' likeness of candy cannot simply predict by the candy's components or properties, so my model's rsq value are not high enough. If there is more data or contains more variables outside of the candy's own properties, the model may perform better.