Final Project

Yuelin Shen

11/22/2022

Contents

We first set up the working environments and load packages.

```r
library(corrplot)
library(discrim)
library(corrr)
library(MASS)
library(tidyverse)
library(tidymodels)
library(ggplot2)
library(janitor)
library(yardstick)
library(dplyr)
library(data.table)
library(glmnet)
```

```r
library(rpart.plot)
library(vip)
library(janitor)
library(randomForest)
library(xgboost)
library(kernlab)
library(ranger)
tidymodels_prefer()
```

Preface

What is MVP in NBA?

The National Basketball Association Most Valuable Player Award (MVP) is an annual National Basketball Association (NBA) award given since the 1955–56 season to the best performing player of the regular season.

Purpose of the model

The purpose of this model is to predict the points per game of a MVP player based on different parameters and MVP samples.

Loading and Exploring Raw Data

```r
mvpstats <- read.csv("mvpsnew.csv")
```

Then we review on over data to know the meanings of each parameter

```r
mvpstats %>% dim()
```

```
## [1] 474  21
```

We have a total of 474 observations with 21 parameters.

Below is the codebook for the dataset. There are a total of 21 variables that could be used to measure players' behaviors and impact on court.

- `Rank`: MVP Rank
- `Player`: Player's Name
- `Age`: Player's aGE
- `Tm`: Team
- `First`: First Place Votes
- `Pts.Won`: Points won
- `Pts.Max`: Maximum Points
- `Share`: Points Share
- `G`: Games Played
- `MP`: Minutes Played per Game

- `PTS`: Points per Game
- `TRB`: Total Rebounds per Game
- `AST`: Assist per Game
- `STL`: Steals per Game
- `BLC`: Blocks per Game
- `FG.`: Field Goal Percentage
- `X3P.`: 3-point Field Goal Percentage
- `FT.`: Free Throw Percentage
- `WS`: Win Shares
- `WS.48`: Win Shares per 48 Minutes
- `Year`: Year of MVP earned

Exploratory Data Analysis

Data Cleaning

We start with exploring our dataset first. Some of the parameters are not numeric, which means that we need to factor them.

```
mvpstats %>% clean_names() %>% head()
```

```
##   rank          player age  tm first pts_won pts_max share  g   mp  pts  trb
## 1    1  Michael Jordan  27 CHI    77     891     960 0.928 82 37.0 31.5  6.0
## 2    2   Magic Johnson  31 LAL    10     497     960 0.518 79 37.1 19.4  7.0
## 3    3  David Robinson  25 SAS     6     476     960 0.496 82 37.7 25.6 13.0
## 4    4 Charles Barkley  27 PHI     2     222     960 0.231 67 37.3 27.6 10.1
## 5    5      Karl Malone  27 UTA     0     142     960 0.148 82 40.3 29.0 11.8
## 6    6   Clyde Drexler  28 POR     1      75     960 0.078 82 34.8 21.5  6.7
##    ast stl blk    fg   x3p    ft   ws ws_48 year
## 1  5.5 2.7 1.0 0.539 0.312 0.851 20.3 0.321 1991
## 2 12.5 1.3 0.2 0.477 0.320 0.906 15.4 0.251 1991
## 3  2.5 1.5 3.9 0.552 0.143 0.762 17.0 0.264 1991
## 4  4.2 1.6 0.5 0.570 0.284 0.722 13.4 0.258 1991
## 5  3.3 1.1 1.0 0.527 0.286 0.770 15.5 0.225 1991
## 6  6.0 1.8 0.7 0.482 0.319 0.794 12.4 0.209 1991
```

```
mvpstats <- mvpstats %>% mutate(Year = factor(Year))
```

Then, we should consider the parameters that could reflect a player's performance and impact on the court. Some of the parameters are irrelevant to our prediction of points per game(PTS), which we would like to exclude them so as to obtain a better prediction.

```
mvpstats <- mvpstats %>% select(-Player, -Rank, -Share, -Tm, -Pts.Won, -Pts.Max, -First, -WS.48)

mvpstats %>% dim()
```

```
## [1] 474  13
```

```
mvpstats %>% head()
```

```
##   Age  G   MP  PTS  TRB  AST STL BLK   FG.  X3P.   FT.   WS Year
## 1  27 82 37.0 31.5  6.0  5.5 2.7 1.0 0.539 0.312 0.851 20.3 1991
## 2  31 79 37.1 19.4  7.0 12.5 1.3 0.2 0.477 0.320 0.906 15.4 1991
## 3  25 82 37.7 25.6 13.0  2.5 1.5 3.9 0.552 0.143 0.762 17.0 1991
## 4  27 67 37.3 27.6 10.1  4.2 1.6 0.5 0.570 0.284 0.722 13.4 1991
## 5  27 82 40.3 29.0 11.8  3.3 1.1 1.0 0.527 0.286 0.770 15.5 1991
## 6  28 82 34.8 21.5  6.7  6.0 1.8 0.7 0.482 0.319 0.794 12.4 1991
```

The dataset now has 13 relevant variables.

We should also check for any NA or missing values in our dataset.

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

```
##      [,1]
## Age  0
## G    0
## MP   0
## PTS  0
## TRB  0
## AST  0
## STL  0
## BLK  0
## FG.  0
## X3P. 0
## FT.  0
## WS   0
## Year 0
```

Clearly, there is no missing or NA value in our dataset now.

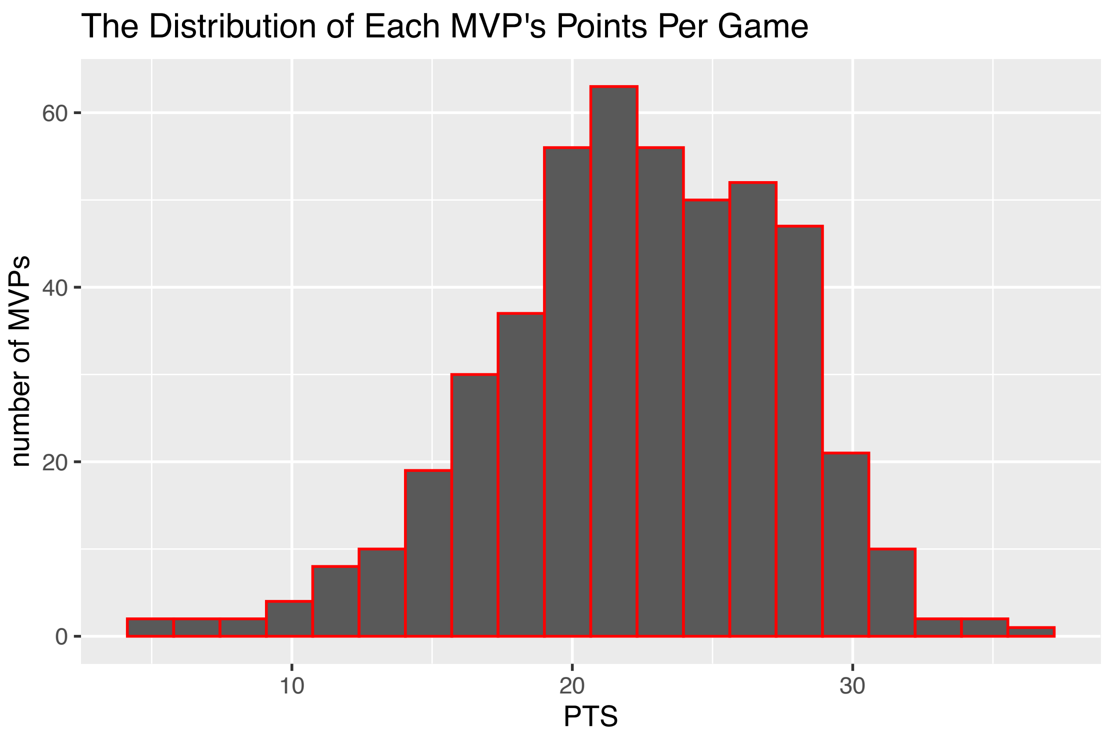
Graphs and Interpretations

After cleaning and modifying our dataset. We then could make our explorative data analysis.

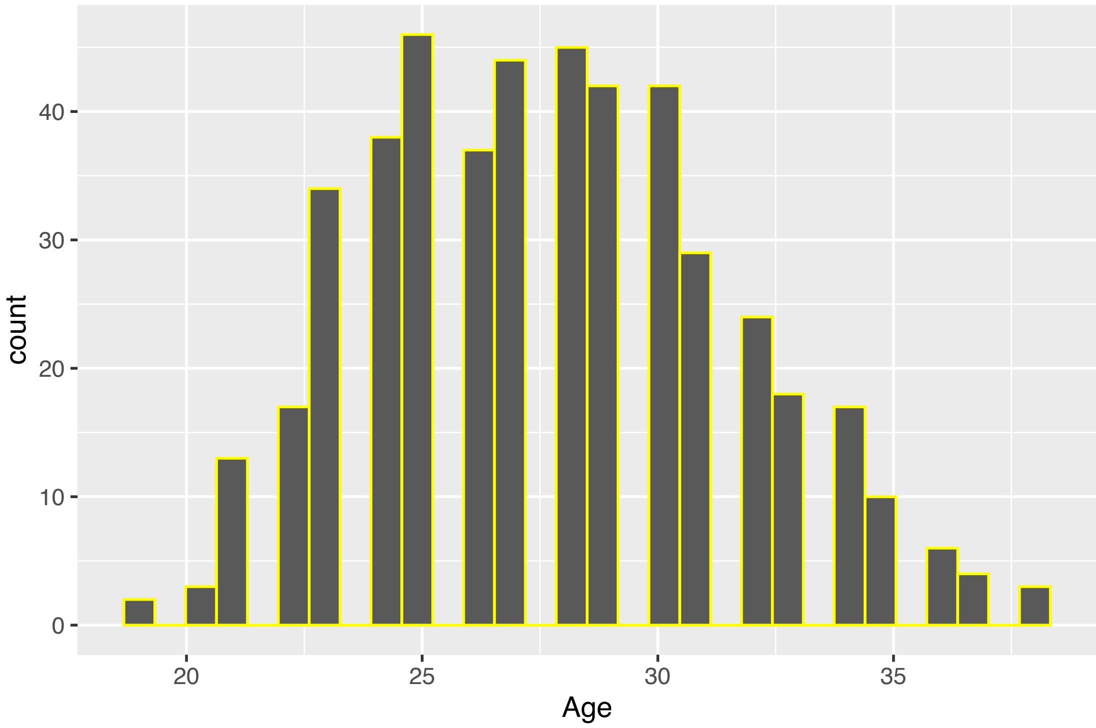
From the histogram, we could observe that each year's number of MVPs are similar. This is because the MPVs of the season are nominated according to the performance of the players during their regular season.

If we then want to access the distribution of every MVP's performance from 1991 to 2021, we would see that the majority of MVPs have PTS around 35 to 40 points. The shape of the distribution is also a little bit left-skewed. There are a few MVPs who might be excellent leaders or made other contributions to become MVPs. But the overall distribution could convey that MVPs of the season are indeed the best player in the league for the year.

```r
ggplot(mvpstats, aes(x = PTS))+
  geom_histogram(bins = 20, color = "Red")+
  ggtitle("The Distribution of Each MVP's Points Per Game")+
  xlab("PTS")+
  ylab("number of MVPs")
```



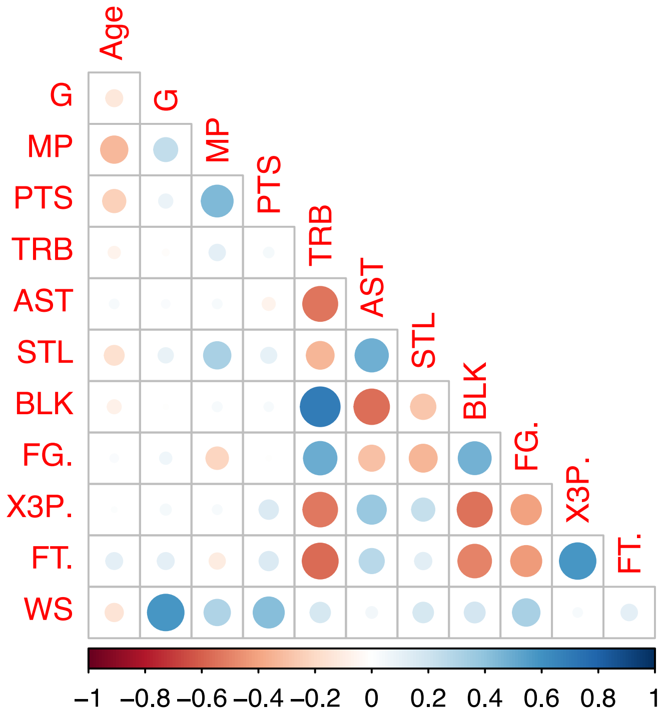

```r
ggplot(mvpstats, aes(x = Age))+
  geom_histogram(color = "Yellow", bins = 30)
```



Although some players might won MVP for several times, for the purpose of predicting MP of the MVP players, we could still consider each time they won as one unique observation. From above plot, we see that the distribution of MVP's age is from 23 to 31 years old.This also makes sense because the peak in a player's career mostly concentrated on this period. This is the time when a player gained some experiences from the games and they are also young and energetic.

After some histograms, we would then examine the internal relationships among parameters.

```
mvpstats %>%
  select(where(is.numeric)) %>%
  cor(use = "complete.obs") %>%
  corrplot(type = "lower", diag = FALSE)
```



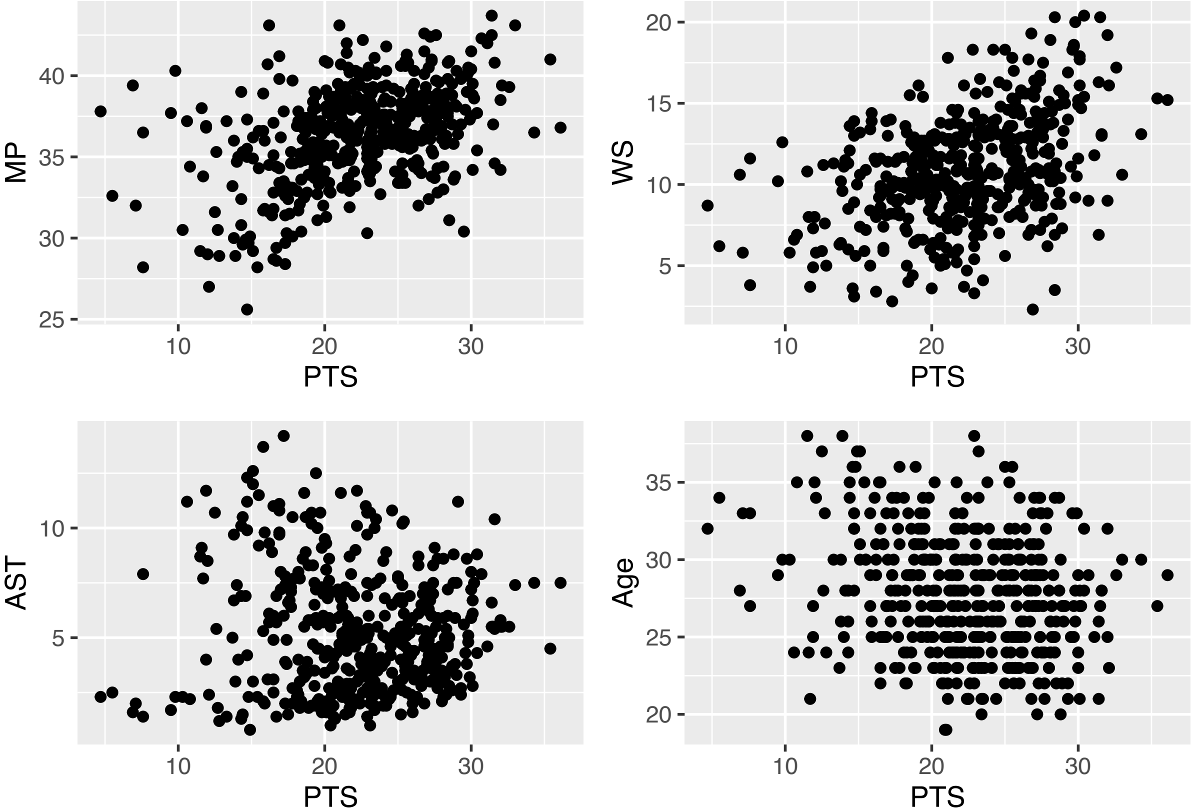
From the corr plot, we found that PTS have strong, positive correlaiton with MP and WS, and PTS have negative correlation with AST and Age. We could also observe many other correlations among parameters.

In fact, TRB has little positive impact on PTS from the corr plot, which is surprising because more rebounds usually means more chances to get points.

Also, the correlation between PTS and BLK is trivial. This is acceptable because blocks usually correlated with defense performances rather than offense performances.

Then, we choose scatter plots to plot PTS and MP, PTS and WS, and PTS and AST, PTS and Age.

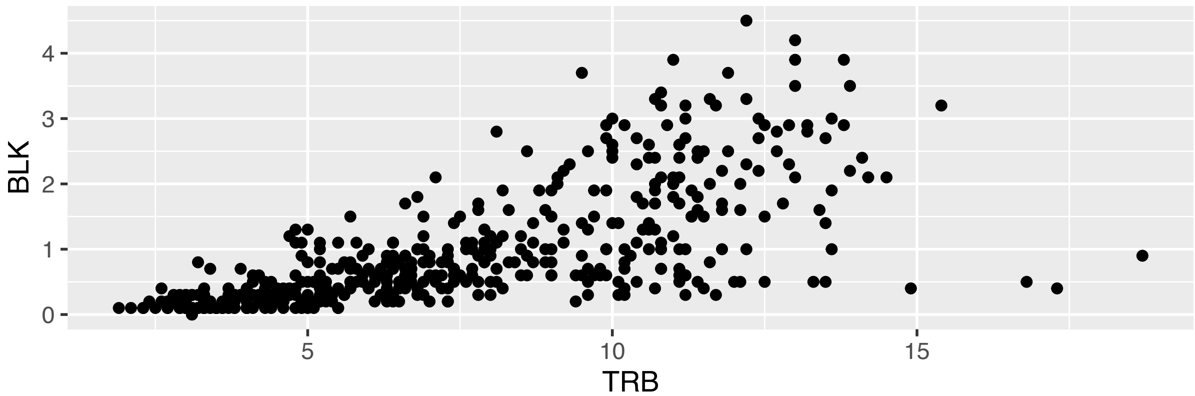
```
s1 <- mvpstats %>% ggplot(aes(x = PTS, y = MP))+ geom_point()
s2 <- mvpstats %>% ggplot(aes(x = PTS, y = WS))+ geom_point()
s3 <- mvpstats %>% ggplot(aes(x = PTS, y = AST))+ geom_point()
s4 <- mvpstats %>% ggplot(aes(x = PTS, y = Age))+ geom_point()
grid.arrange(s1, s2, s3, s4)
```

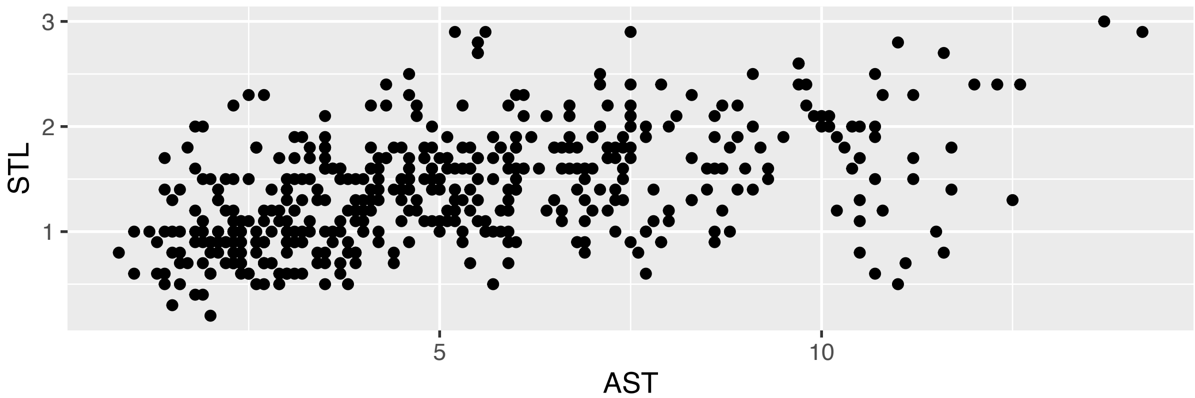


We saw some roughly linear correlation between PTS and MP, and PTS and WS. We did not observe strong correlation between PST and Age, which also reflected on the correlation matrix.

Beside the correlation between PTS and other predictors, I also observe two other pairs of parameters: BLK and TRB and AST and STL has strong correlations. Then, we could examine their correlation by plotting scatter plots again.

```
s4 <- mvpstats %>% ggplot(aes(x = TRB, y = BLK))+ geom_point()
s5 <- mvpstats %>% ggplot(aes(x = AST, y = STL))+ geom_point()
grid.arrange(s4, s5)
```





Based on the plots, it is reasonable to say that these two pairs are strongly correlated, which means that I could establish interactions between them.

Model Construction

Data Spliting

```
set.seed(3435)

mvpstats_split <- initial_split(mvpstats, strata = "PTS", prop = 0.7)
mvpstats_train <- training(mvpstats_split)
mvpstats_test <- testing(mvpstats_split)

mvpstats_folds <- vfold_cv(mvpstats_train, v = 5, strata = "PTS")
```

We should examine both training and testing set to make sure the number of observations is enough.

```
dim(mvpstats_train)
```

```
## [1] 329  13
```

```
dim(mvpstats_test)
```

```
## [1] 145  13
```

The training set contains 329 observations and the testing set contains 145 observations, which are sufficient here.

Setting up Model

Cretate Recipe

After splitting our dataset, we are going to create the recipe for our following predictions.

```
mvpstats_recipe <- recipe(PTS ~ Age + G + MP +TRB + AST + STL + BLK + FG. + X3P.+ FT.+ WS + Year, data =
  step_interact(terms = ~ starts_with("TRB"):BLK + AST:STL) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_predictors())
```

Linear Regression Model

The first model we use is a linear regression model.

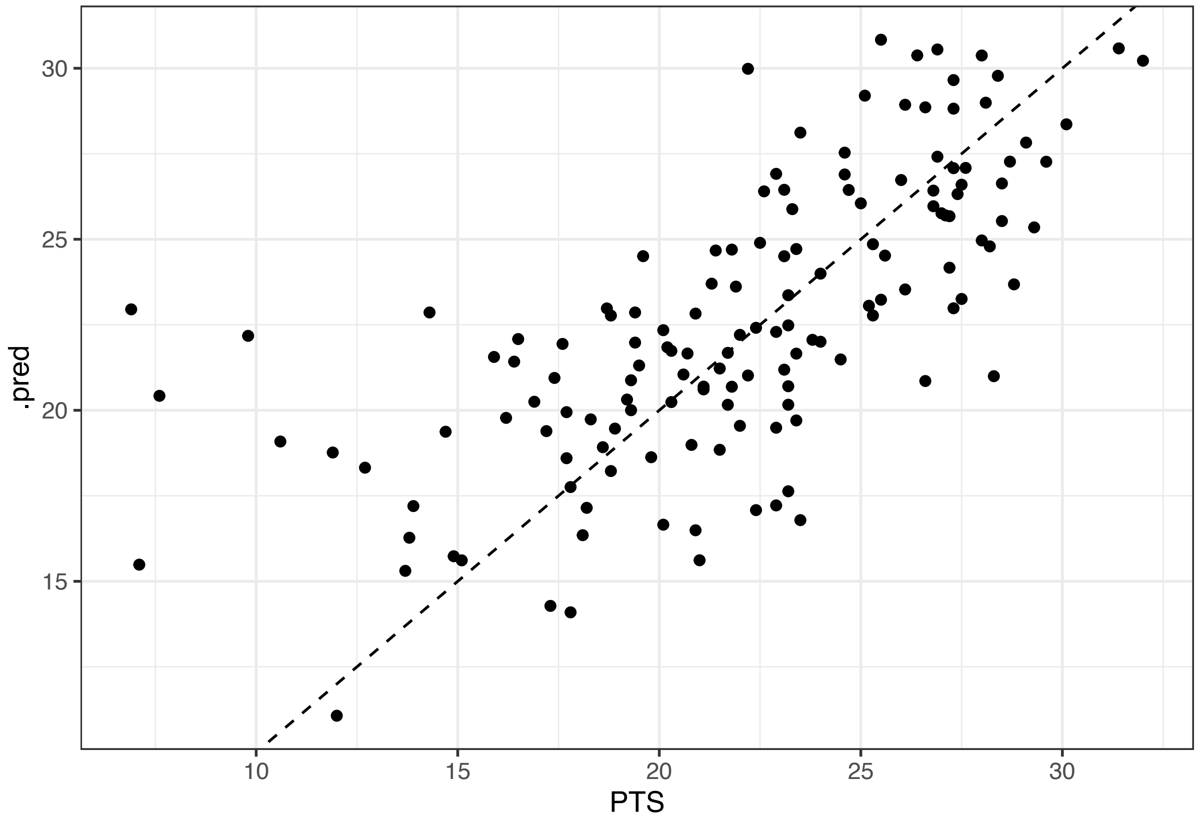
```
lm_model <- linear_reg() %>%
  set_engine("lm")

linear_wk <- workflow() %>%
  add_recipe(mvpstats_recipe) %>%
  add_model(lm_model)

linear_fit <- fit_resamples(linear_wk, mvpstats_folds)

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   3.55      5  0.226  Preprocessor1_Model1
## 2 rsq     standard   0.545     5  0.0503 Preprocessor1_Model1
```

We select the best model across different folds.

```
best_linear <- select_best(linear_fit, metric = "rmse")
linear_final <- finalize_workflow(linear_wk, best_linear)
linear_final_fit <- fit(linear_final, data = mvpstats_train)
```

```
linear_predict <- augment(linear_final_fit, new_data = mvpstats_test) %>% select(PTS, starts_with(".pre
linear_predict %>% ggplot(aes(x = PTS, y = .pred)) +
  geom_point(alpha = 1) +
  geom_abline(lty = 2) +
  theme_bw()
```



Finally, we examine the performance of the linear regression model by examing its `rmse` and `rsq`.

```
linear_accuracy_rmse <- augment(linear_final_fit, new_data = mvpstats_test) %>%
  rmse(truth = PTS, estimate = .pred)
linear_accuracy_rsq <- augment(linear_final_fit, new_data = mvpstats_test) %>%
  rsq(truth = PTS, estimate = .pred)

linear_performance <- rbind(linear_accuracy_rmse, linear_accuracy_rsq)
linear_performance
```

```
## # A tibble: 2 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       3.74
## 2 rsq     standard       0.475
```

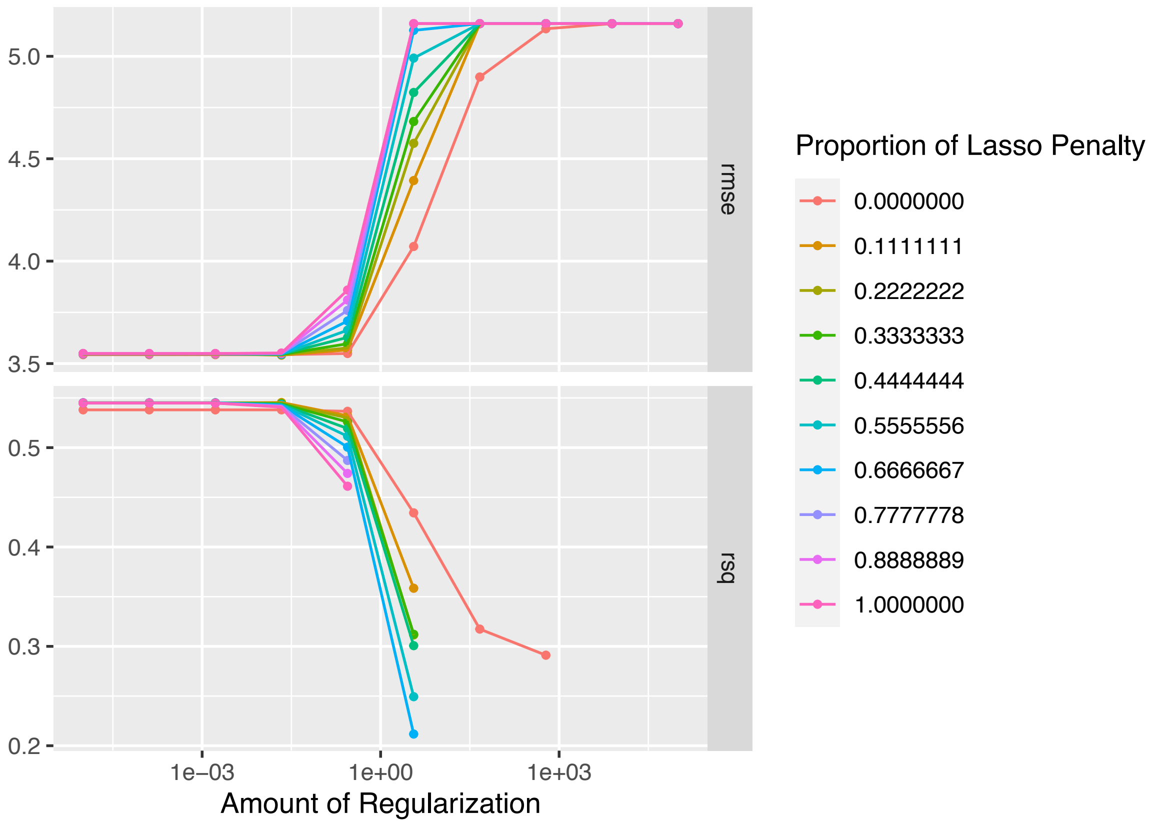
Elastic Net Regressioin Model

Then, the secon model we choose is an Elastic Net Regression Model, specifically a ridge regressoin model.

For this model, we choose L1 regularization and L2 penalty, which is a Ridge Regression Model. The range of regularization is from 0 to 1, and penalty range is from -5 to 5.

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

```r
#set up workflow for the model
ridge_wk <- workflow() %>%
  add_recipe(mvpstats_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_wk,
  resamples = mvpstats_folds,
  grid = ridge_grid
  )

ridge_res %>% autoplot()
```



Next, we set metric to `rmse` and select the best model.

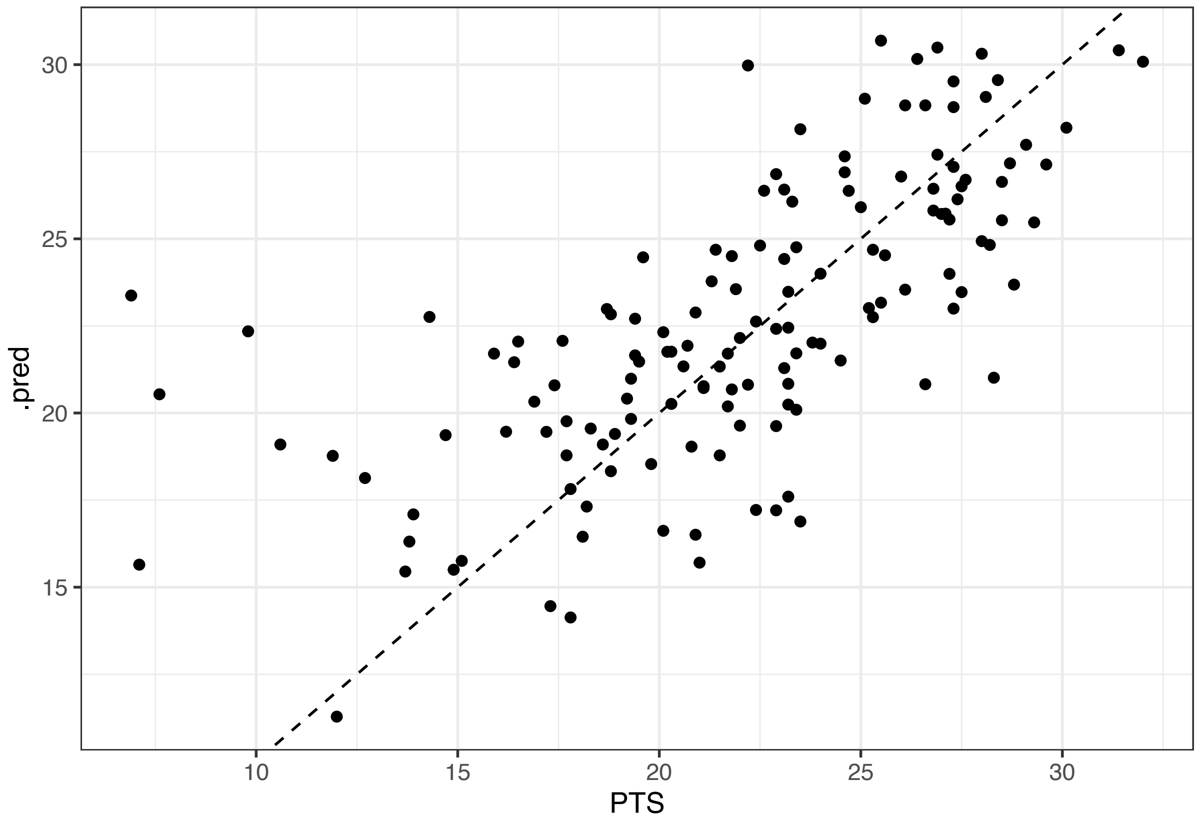
```r
ridge_res %>% collect_metrics() %>% head()
```

```
## # 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    3.54      5   0.210  Preprocessor1_Model001
```

```
## 2 0.00001          0 rsq     standard   0.538     5  0.0454 Preprocessor1_Model001
## 3 0.000129         0 rmse    standard   3.54      5  0.210  Preprocessor1_Model002
## 4 0.000129         0 rsq     standard   0.538     5  0.0454 Preprocessor1_Model002
## 5 0.00167          0 rmse    standard   3.54      5  0.210  Preprocessor1_Model003
## 6 0.00167          0 rsq     standard   0.538     5  0.0454 Preprocessor1_Model003
```

```
best_penalty <- select_best(ridge_res, metric = "rmse")
ridge_final <- finalize_workflow(ridge_wk ,best_penalty)
ridge_final_fit <- fit(ridge_final, data = mvpstats_train)
```

```
ridge_predict <- augment(ridge_final_fit, new_data = mvpstats_test) %>% select(PTS, starts_with(".pred")
ridge_predict %>% ggplot(aes(x = PTS, y = .pred)) +
  geom_point(alpha = 1) +
  geom_abline(lty = 2) +
  theme_bw()
```



We could also exmaine the accuracy level of this model.

```
ridge_accuracy_rmse <- augment(ridge_final_fit, new_data = mvpstats_test) %>%
  rmse(truth = PTS, estimate = .pred)
ridge_accuracy_rsq <- augment(ridge_final_fit, new_data = mvpstats_test) %>%
  rsq(truth = PTS, estimate = .pred)

elastic_net_performance <- rbind(ridge_accuracy_rmse, ridge_accuracy_rsq)
elastic_net_performance
```

```
## # A tibble: 2 x 3
```

```
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       3.74
## 2 rsq     standard       0.472
```

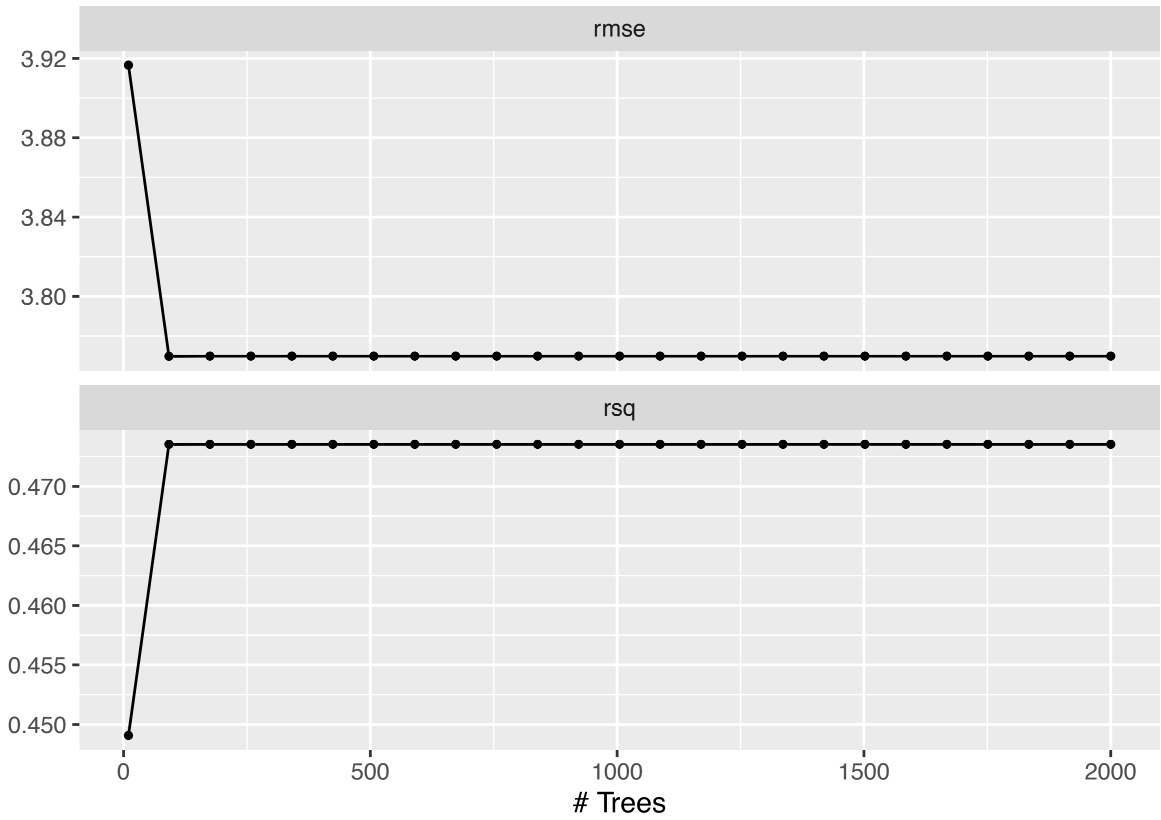
Clearly, the rsq of our ridge regression model has a `rsq` around 0.47, which does not yield a convincing prediction.

Tree-based Methods

Boosted Tree Model

The third model I choose is the boosted tree model.

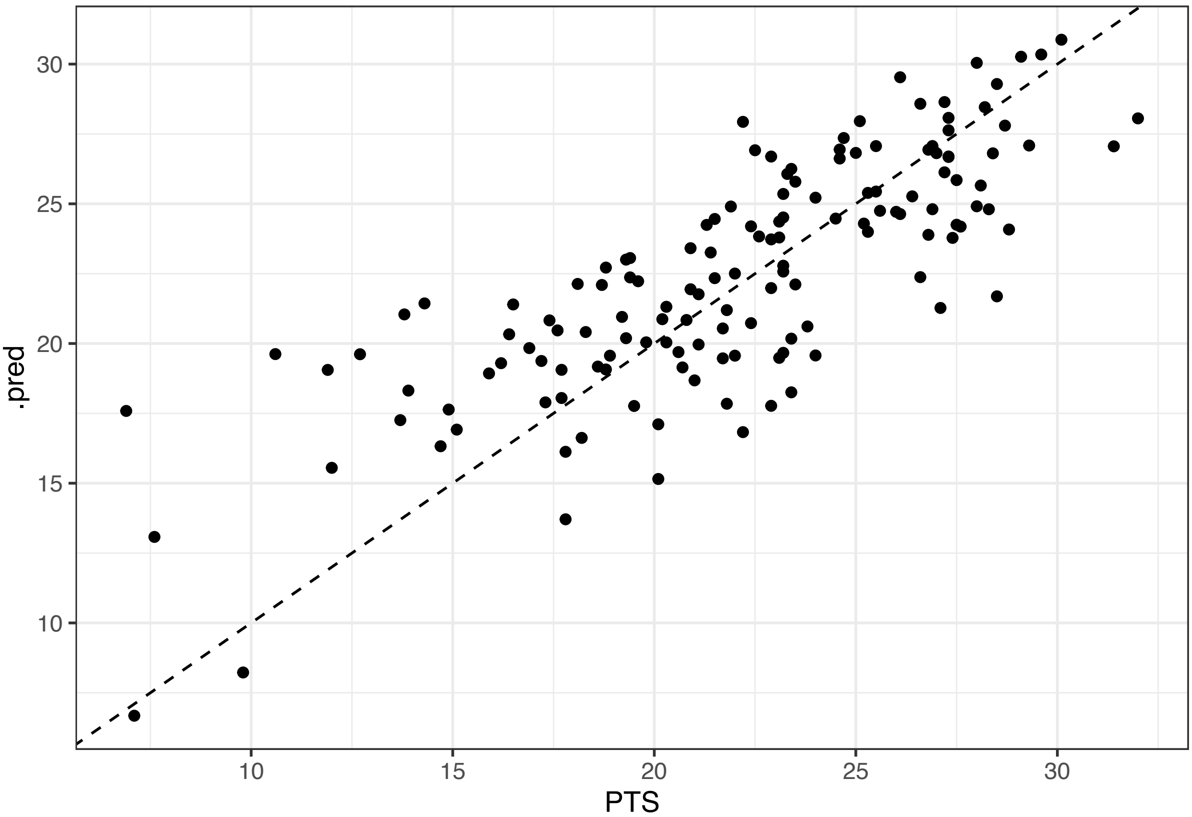
```r
boost_spec <- boost_tree() %>%
  set_engine("xgboost") %>%
  set_mode("regression")

boost_wf <- workflow() %>%
  add_model(boost_spec %>%
  set_args(trees = tune())) %>%
  add_recipe(mvpstats_recipe)

boost_grid <- grid_regular(trees(range = c(10, 2000)), levels = 25)

boost_tune_res <- tune_grid(
  boost_wf,
  resamples = mvpstats_folds,
  grid = boost_grid,
)

boost_tune_res %>% autoplot()
```



As always, we then select the best model.

```
best_boost <- select_best(boost_tune_res, metric = "rmse")
boost_final <- finalize_workflow(boost_wf, best_boost)
boost_final_fit <- fit(boost_final, data = mvpstats_train)
```

```
boost_predict <- augment(boost_final_fit, new_data = mvpstats_test) %>% select(PTS, starts_with(".pred"
boost_predict %>% ggplot(aes(x = PTS, y = .pred)) +
  geom_point(alpha = 1) +
  geom_abline(lty = 2) +
  theme_bw()
```




```
boost_accuracy_rmse <- augment(boost_final_fit, new_data = mvpstats_test) %>%
  rmse(truth = PTS, estimate = .pred)
boost_accuracy_rsq <- augment(boost_final_fit, new_data = mvpstats_test) %>%
  rsq(truth = PTS, estimate = .pred)

boost_performance <- rbind(boost_accuracy_rmse, boost_accuracy_rsq)
boost_performance
```

```
## # A tibble: 2 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       3.08
## 2 rsq     standard       0.631
```

For the boosted tree model, we observe a higher `rsq` of 0.63, which adds more credibility to this model.

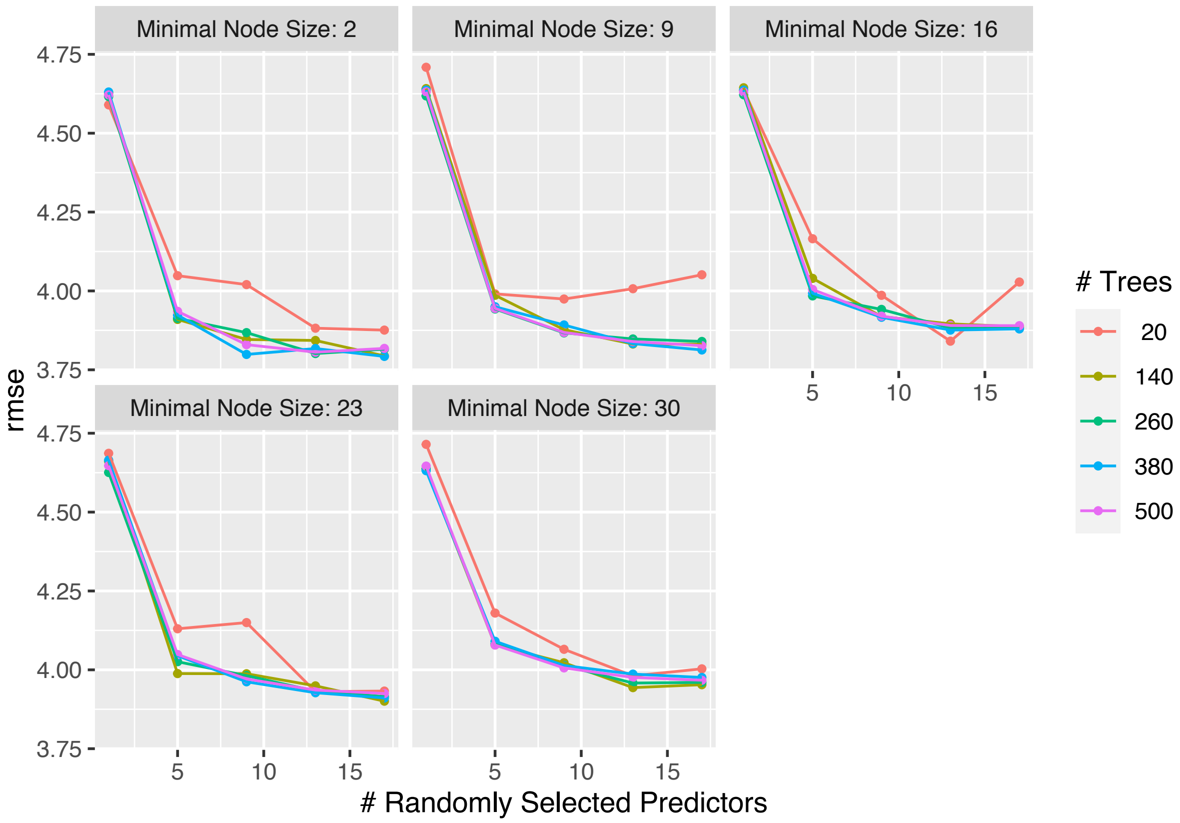
Random Forest Model

The fourth model I select is random forest.

```
bagging_spec <- rand_forest(mtry = .cols()) %>%
  set_engine("ranger", importance = 'impurity') %>%
  set_mode("regression") %>%
  set_args(min_n = tune(), mtry = tune(), trees = tune())
```

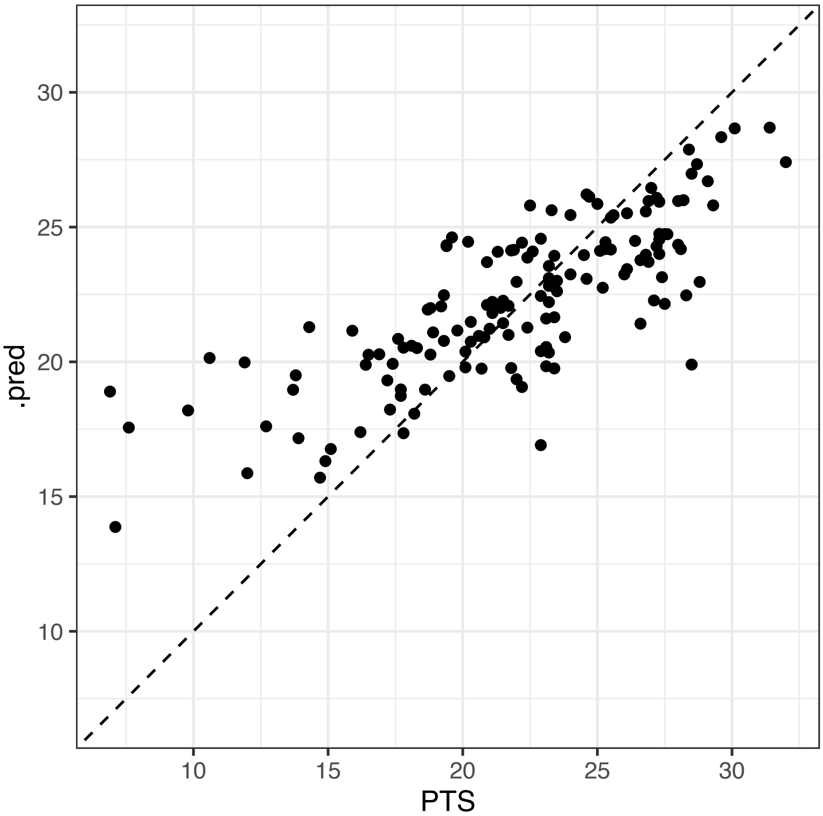
```r
rand_tree_wk <- workflow() %>%
  add_recipe(mvpstats_recipe) %>%
  add_model(bagging_spec)

forest_grid <- grid_regular(mtry(range = c(1, 17)),
                            trees(range = c(20, 500)),
                            min_n(range = c(2, 30)),
                            levels = 5)

tune_res_rf <- tune_grid(
  object = rand_tree_wk,
  resamples = mvpstats_folds,
  grid = forest_grid,
  metrics = metric_set(rmse)
)

best_rf <- select_best(tune_res_rf, metric = "rmse")
rf_final <- finalize_workflow(rand_tree_wk, best_rf)
rf_final_fit <- fit(rf_final, data = mvpstats_train)
```

```r
autoplot(tune_res_rf)
```




```r
rf_predict <- augment(rf_final_fit, new_data = mvpstats_test) %>% select(PTS, starts_with(".pred"))
rf_predict %>% ggplot(aes(x = PTS, y = .pred)) +
  geom_point(alpha = 1) +
```

```r
  geom_abline(lty = 2) +
  coord_obs_pred() +
  theme_bw()
```



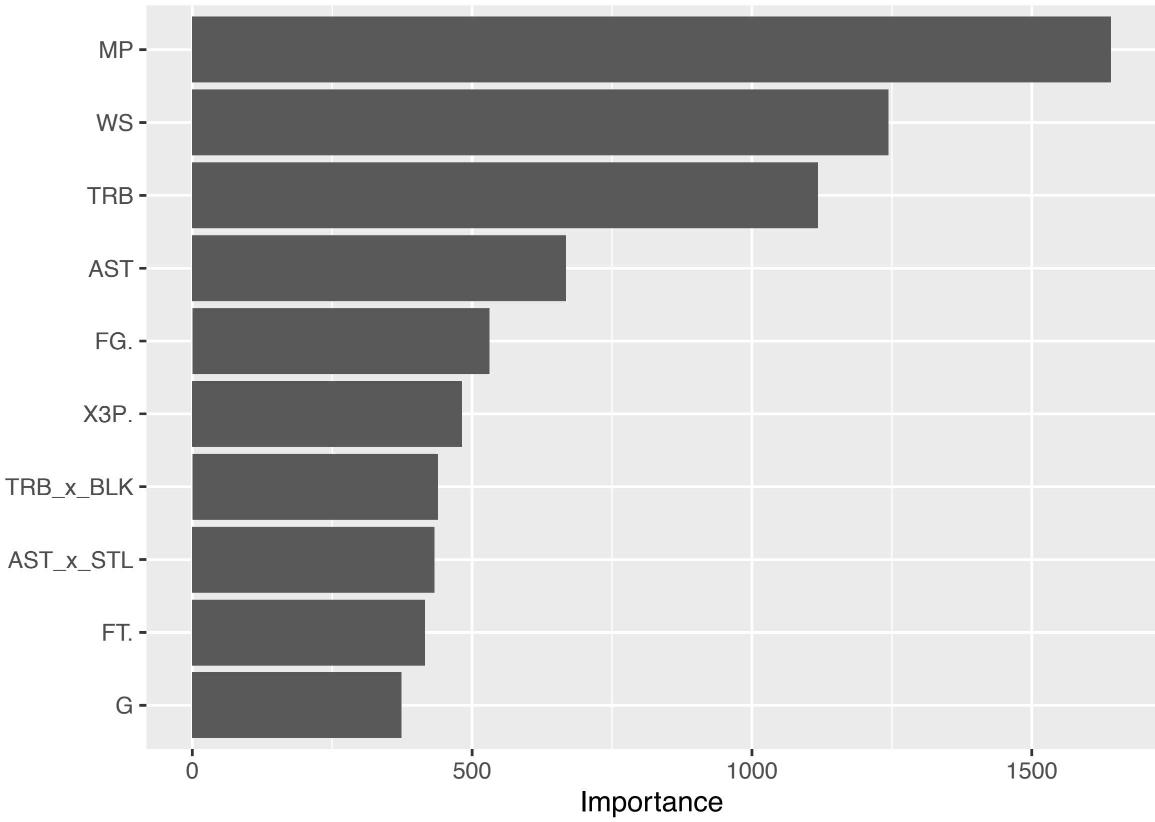

```r
rf_accuracy_rmse <- augment(rf_final_fit, new_data = mvpstats_test) %>%
  rmse(truth = PTS, estimate = .pred)
rf_accuracy_rsq <- augment(rf_final_fit, new_data = mvpstats_test) %>%
  rsq(truth = PTS, estimate = .pred)

rf_performance <- rbind(rf_accuracy_rmse, rf_accuracy_rsq)
rf_performance
```

```
## # A tibble: 2 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       3.29
## 2 rsq     standard       0.615
```

The result shows that random forest has an estimation of rsq around 0.6, which is in the middle among four models.

We could also use `vip()` to find out that MP has the highest variable importance.

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



Conclusion

Performance Comparison

After computing four different models for our dataset `mvpstats`, we could combine the results together to see which model performs the best.

We first compare the rmse index.

```
rmse_comparisons <- bind_rows(linear_accuracy_rmse, ridge_accuracy_rmse , boost_accuracy_rmse, rf_accura
  tibble() %>% mutate(model = c("Linear", "Ridge", "Boosted Tree", "Random Forest")) %>%
  select(model, .estimate) %>%
  arrange(.estimate)

rmse_comparisons
```

```
## # A tibble: 4 x 2
##   model         .estimate
##   <chr>             <dbl>
## 1 Boosted Tree       3.08
## 2 Random Forest      3.29
## 3 Linear             3.74
## 4 Ridge              3.74
```

After filtering them with ascending order, we find out that the boosted tree model has the lowest rmse.

In addition, we could also extract the `rsq`.

```
rsq_comparisons <- bind_rows(linear_accuracy_rsq, ridge_accuracy_rsq , boost_accuracy_rsq, rf_accuracy_r
  tibble() %>% mutate(model = c("Linear", "Ridge", "Boosted Tree", "Random Forest")) %>%
  select(model, .estimate) %>%
  arrange(desc(.estimate))

rsq_comparisons
```

```
## # A tibble: 4 x 2
##   model         .estimate
##   <chr>             <dbl>
## 1 Boosted Tree      0.631
## 2 Random Forest     0.615
## 3 Linear            0.475
## 4 Ridge             0.472
```

By sorting them in descending order, it is also clear to see that the boosted tree model also has the highest rsq.

In conclusion, based on our predictiom and selection of models, it is reasonable to see that the boosted tree model works the best with our dataset.

The report has three components: data gathering, exploratory data analysis, and model fitting. Indeed, I get the dataset from `Kaggle` and conduct some data cleaning procedures so as to make this dataset workable. In particular, I examine the relationships among variables to see if there are some pairs of variables that may have interactions. Fortunately, I was able to find two pairs: `TRB` and `BLC`, `AST` and `STL`.Also, I modify the dataset by selecting the relevant variables. For example, `First` does not contribute to PTS prediction because there are many comfounding variables in voting processes.

After the data analysis, I started to build up the models. I selected linear regression, elastic net, boosted tree, and random forest. These four models could give me a better interpretation because they cover from linear to tuning. Clearly, after comparisions among models, the result shows that boosted tree has the best performance. Also, I could interpret from the result that there are many factors that could determine the impact of a player on the court. We could not simply look at one parameter, for example PTS, to conclude that "xxx is the best player in the league because he has the highest PTS".