Predicting the Annual Salary of a National Hockey League (NHL) Player Based on His Performance

Data Science II

Harrison Rubin

Contents

Introduction

From 2019 to 2021, the National Hockey League (NHL) has compiled performance statistics for its 938 players. As with any other professional sports league, a player's salary is primarily determined by performance, meaning that we expect the best-performing players to receive the highest salaries.

For my final project, I decided to create regression models to predict an NHL player's salary based on his performance statistics. Some of these performance statistics include: goals, assists, points, plus-minus, time on ice, as well as several advanced metrics, such as Corsi, Fenwick, expected plus-minus, and PDO (see codebook for detailed description of each variable). Player salaries range from $700,000 to $12,500,000.

Using data found on the NHL website, Spotrac, and Hockey Reference, I was able to compile each player's performance statistics, salary, and advanced metrics. Below are the links to each of these websites.

http://www.nhl.com/stats/skaters?reportType=season&seasonFrom=20192020&seasonTo=20202021&gameType=2&filter=

https://www.spotrac.com/nhl/

https://www.hockey-reference.com/leagues/NHL_2020_skaters-advanced.html

The three predictive models utilized are boosted tree, random forest, and k-nearest neighbors. Each of these three models was created for forwards and defensemen (therefore 6 models in total), since NHL teams generally award forwards and defensemen salaries based on different performance metrics. Forwards are more highly paid than defensemen. By way of example, the average salary for the five most highly paid forwards is $11.5 million, while the average salary for the five most highly paid defensemen is only $9.8 million.

```
# Load Packages and Set Seed
library(tidyverse)
library(tidymodels)
library(lubridate)
library(readxl)
library(janitor)
library(broom)
```

```
library(naniar)
library(skimr)
library(patchwork)
library(vip)

set.seed(8675309)
```

```
# Read in Data
skaters <- read_excel("data/unprocessed/skaters.xlsx")
salaries <- read_excel("data/unprocessed/skaters_salaries.xlsx")
advanced_stats <- read_csv("data/unprocessed/nhl_advanced.csv")
```

```
## Parsed with column specification:
## cols(
##   player = col_character(),
##   corsi_pct = col_double(),
##   fenwick_pct = col_double(),
##   tk = col_double(),
##   gv = col_double(),
##   exp_plus_minus = col_double(),
##   pdo = col_double()
## )
```

Data Cleaning

Before creating the regression models, I first cleaned the data from the three aforementioned sources. A major step in this process was removal of rookies from the dataset. Rookies are players in their first year in the NHL. Under the Collective Bargaining Agreement entered into by the NHL and its players, rookies can only earn up to a maximum of $925,000 per year, even if rookies score as many points as an NHL veteran who earns millions of dollars each year. In order to prevent skew in the results, I decided to remove rookies altogether. Additionally, I decided to remove players who played fewer than 25 games (out of a possible 82 games in a normal NHL season) in order to prevent further skewed results. Finally, I separated forwards from defensemen in preparation for building separate models for the two types of players.

Below is a summary of any missing entries in the dataset.

```
skaters <- skaters %>%
  clean_names() %>%
  rename(
    plus_minus = x,
    shots = s)

skaters_salaries <- skaters %>%
  left_join(salaries, by = "player") %>%
  filter(!is.na(salary))

salary_dat <- skaters_salaries %>%
  select(player, salary) %>%
  mutate(
    salary = as.numeric(salary)
  )

skaters_salaries <- skaters_salaries %>%
  mutate(
    salary = as.numeric(salary),
```

```
    s_percent = as.numeric(s_percent),
    fow_percent = as.numeric(fow_percent)
  )

# Convert time on ice into minutes
toi_df <- data.frame(skaters_salaries$toi_gp)

toi_min <- toi_df %>%
  separate(skaters_salaries.toi_gp, into = c("pre", "post")) %>%
  pull("pre")

toi_sec <- toi_df %>%
  separate(skaters_salaries.toi_gp, into = c("pre", "post")) %>%
  pull("post")

toi <- (as.numeric(toi_min) + as.numeric(toi_sec) / 60)

skaters_salaries <- skaters_salaries %>%
  mutate(
    toi_gp = toi
  )

skaters_salaries <- skaters_salaries %>%
  left_join(advanced_stats, by = "player")

# Remove rookies and players who played fewer than 25 games
skaters_salaries <- skaters_salaries %>%
  rename(hand = s_c) %>%
  filter(gp >= 25) %>%
  filter(salary >= 925000) %>%
  mutate(tk_gv = tk / gv)

# Remove Duplicates
skaters_salaries <- skaters_salaries[!duplicated(skaters_salaries$player) ,] %>%
  mutate(salary = log10(salary))

# Create distinct data frames for forwards and defensemen
forwards <- skaters_salaries %>%
  filter(pos == "C" | pos == "L" | pos == "R")

defense <- skaters_salaries %>%
  filter(pos == "D") %>%
  filter(!is.na(corsi_pct))

miss_var_summary(skaters)
```

```
## # A tibble: 22 x 3
##    variable   n_miss pct_miss
##    <chr>       <int>    <dbl>
##  1 player          0        0
##  2 s_c             0        0
##  3 pos             0        0
##  4 gp              0        0
##  5 g               0        0
```

```
##  6 a               0        0
##  7 p               0        0
##  8 plus_minus      0        0
##  9 pim             0        0
## 10 p_gp            0        0
## # ... with 12 more rows
```

```
miss_var_summary(salaries)
```

```
## # A tibble: 2 x 3
##   variable n_miss pct_miss
##   <chr>     <int>    <dbl>
## 1 player        2    0.203
## 2 salary        2    0.203
```

```
miss_var_summary(advanced_stats)
```

```
## # A tibble: 7 x 3
##   variable       n_miss pct_miss
##   <chr>           <int>    <dbl>
## 1 pdo                 5    0.566
## 2 player              0    0
## 3 corsi_pct           0    0
## 4 fenwick_pct         0    0
## 5 tk                  0    0
## 6 gv                  0    0
## 7 exp_plus_minus      0    0
```

Exploratory Data Analysis

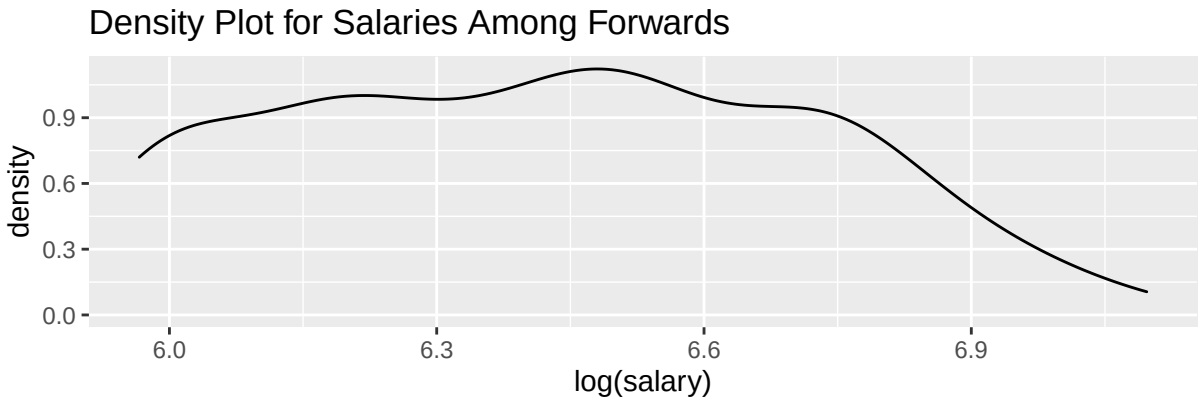
Salaries

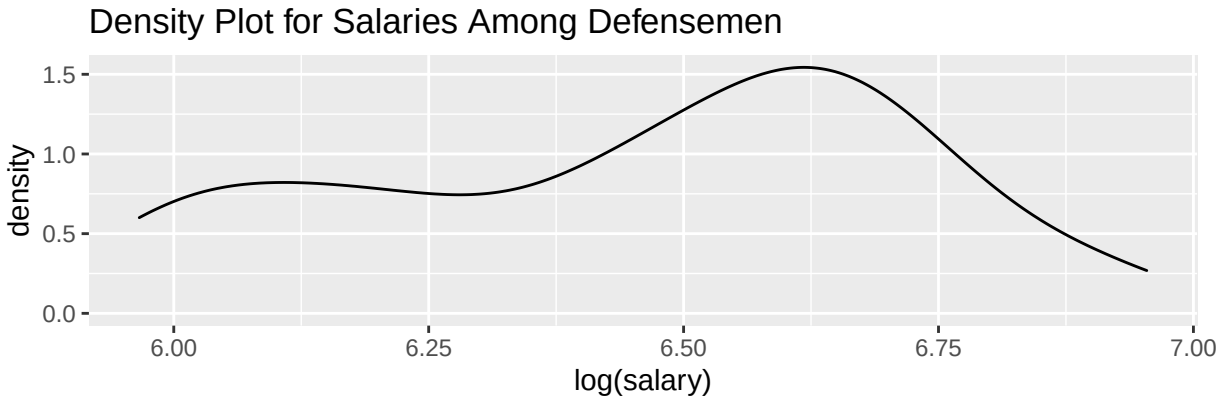
To ensure that the outcome variable, salary, is more normally distributed, and because salaries are very positively skewed, I performed a log transformation with a base of 10 on every player's salary. Below are density plots for the distributions of salaries for both forwards and defensemen.

```
sal_f <- forwards %>%
  ggplot(aes(x = salary)) +
  geom_density() +
  ggtitle("Density Plot for Salaries Among Forwards") +
  xlab("log(salary)")

sal_d <- defense %>%
  ggplot(aes(x = salary)) +
  geom_density() +
  ggtitle("Density Plot for Salaries Among Defensemen") +
  xlab("log(salary)")

sal_f / sal_d
```





Forwards

Below are three scatterplots for the `forwards` dataset. The first plot is expected plus-minus versus points, the second plot is shooting percentage versus number of shots, and the third is PDO (a measure of how lucky a player is) versus Corsi (an overall measure of players' contributions to their teams).

The first plot demonstrates that, in general, players that score more points tend to have a higher expected plus-minus. The second plot shows that there is a smaller positive effect on shooting percentage brought about by the number of shots taken by a player. The third plot describes that there is little to no effect on a player's PDO based on his Corsi.

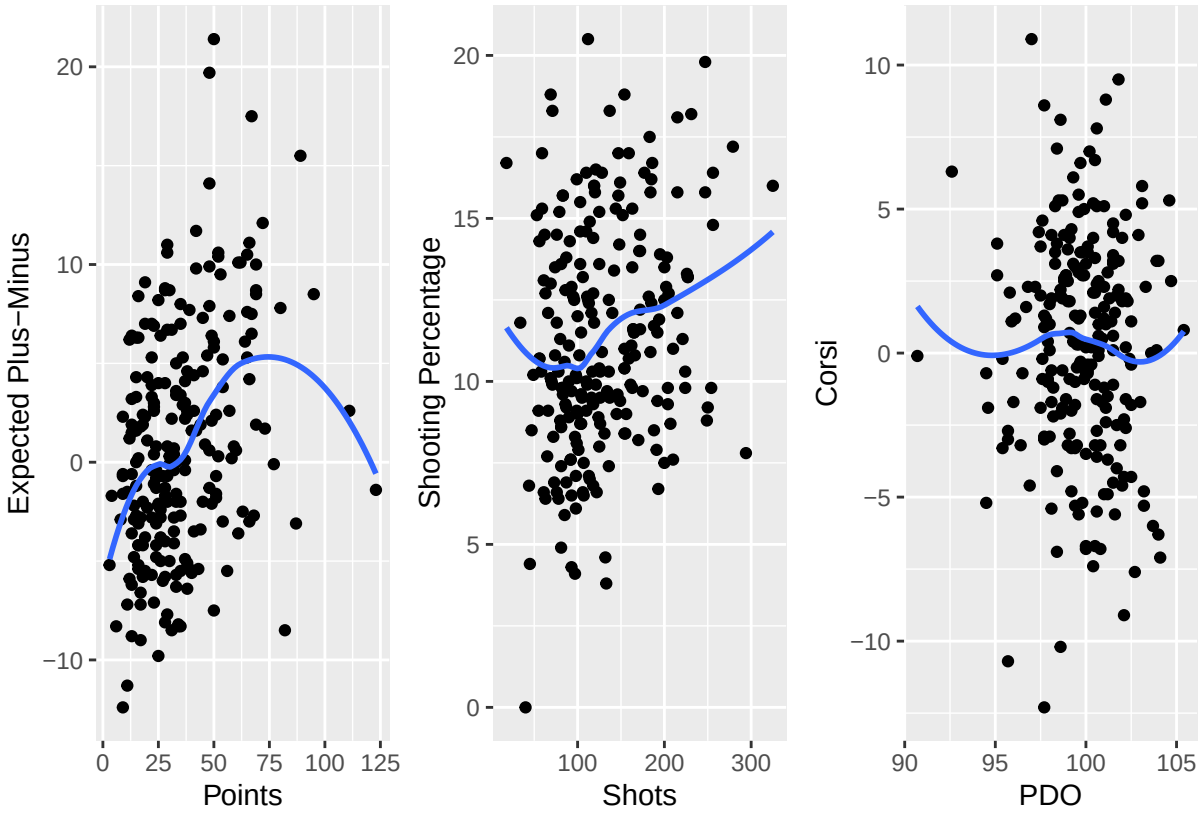
```
p1f <- forwards %>%
  ggplot(aes(x = p, y= exp_plus_minus)) +
  geom_point() +
  xlab("Points") +
  ylab("Expected Plus-Minus") +
  geom_smooth(se = FALSE)

p2f <- forwards %>%
  ggplot(aes(x = shots, y= s_percent)) +
  geom_point() +
  xlab("Shots") +
  ylab("Shooting Percentage") +
  geom_smooth(se = FALSE)

p3f <- forwards %>%
  ggplot(aes(x = pdo, y= corsi_pct)) +
  geom_point() +
  xlab("PDO") +
```

```
  ylab("Corsi") +
  geom_smooth(se = FALSE)

p1f + p2f + p3f + facet_grid()
```



Defensemen

Below are three scatterplots for the `defense` dataset. The first plot is points versus the ratio of takeaways to giveaways (a skilled defensemen typically has more takeaways than giveaways), the second plot is expected plus-minus versus points, and the third plot is PDO versus Corsi.

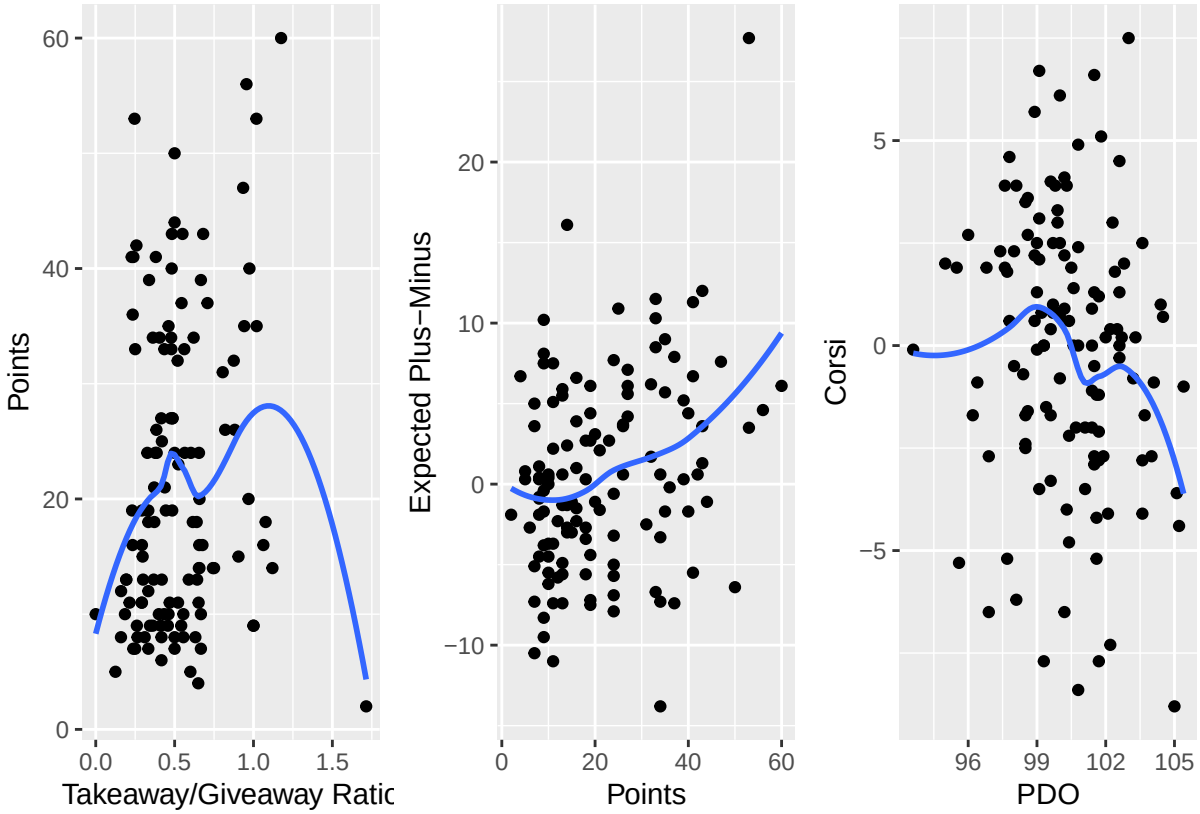
The first plot shows that defensemen who have a higher ratio of takeaways to giveaways tend to score more points. The second plot demonstrates that defensemen that score more points are more likely to have a higher expected plus-minus. The third plot shows that players that have a higher Corsi are slightly more likely to have a lower PDO.

```
p1d <- defense %>%
  ggplot(aes(x = tk_gv, y = p)) +
  geom_point() +
  xlab("Takeaway/Giveaway Ratio") +
  ylab("Points") +
  geom_smooth(se = FALSE)

p2d <- defense %>%
  ggplot(aes(x = p, y= exp_plus_minus)) +
  geom_point() +
  xlab("Points") +
  ylab("Expected Plus-Minus") +
  geom_smooth(se = FALSE)
```

```
p3d <- defense %>%
  ggplot(aes(x = pdo, y= corsi_pct)) +
  geom_point() +
  xlab("PDO") +
  ylab("Corsi") +
  geom_smooth(se = FALSE)

p1d + p2d + p3d + facet_grid()
```



Data Splitting

I decided to create training and testing sets for both forwards and defensemen. As these datasets are relatively small, I used a proportion of 0.6 for splitting into the training and testing sets. Additionally, I chose to stratify by the outcome variable (salary) in order to ensure proper representation of each level of salary (i.e., low, medium, high) within the training and testing sets. I also utilized v-fold cross validation with 5 folds and 5 repeats, therefore creating 25 iterations for forwards and 25 iterations for defensemen.

```
forwards_split <- initial_split(data = forwards, prop = 0.6, strata = salary)
forwards_train <- training(forwards_split)
forwards_test <- testing(forwards_split)
forwards_split
```

```
## <Analysis/Assess/Total>
## <135/93/228>
```

```
defense_split <- initial_split(data = defense, prop = 0.6, strata = salary)
defense_train <- training(defense_split)
defense_test <- testing(defense_split)
```

```
defense_split
```

```
## <Analysis/Assess/Total>
## <73/52/125>
```

```
forwards_fold <- vfold_cv(data = forwards_train, v = 5, repeats = 5, strata = salary)

defense_fold <- vfold_cv(data = defense_train, v = 5, repeats = 5, strata = salary)
```

```
## Warning: The number of observations in each quantile is below the recommended
## threshold of 20. Stratification will be done with 3 breaks instead.

## Warning: The number of observations in each quantile is below the recommended
## threshold of 20. Stratification will be done with 3 breaks instead.

## Warning: The number of observations in each quantile is below the recommended
## threshold of 20. Stratification will be done with 3 breaks instead.

## Warning: The number of observations in each quantile is below the recommended
## threshold of 20. Stratification will be done with 3 breaks instead.

## Warning: The number of observations in each quantile is below the recommended
## threshold of 20. Stratification will be done with 3 breaks instead.
```

Feature Engineering

I created recipes for forwards and defensemen with the following steps: (1) remove all non-numeric predictors, (2) create interactions between select predictors, (3) remove predictors that have linear combinations between them, (4) one-hot dummy code handedness and position, and (5) normalize numeric data to have a standard deviation of one and a mean of zero.

For forwards, I created interactions between points and expected-plus-minus, shots and shooting percentage, and PDO and expected plus-minus. These interactions are meant to highlight forwards that are more well-rounded (i.e., players who play offensive and defensive hockey equally well), those that are purely skilled goal scorers, and those that are "luckier" than others.

For defensemen, I created interactions between the takeaway/giveaway ratio and points, points and expected-plus-minus, and PDO and expected-plus-minus. The purpose of these interactions is to highlight defensemen that are "shutdown defensemen" (i.e., players that emphasize playing defense rather than scoring goals and are effective at preventing top players on the opposing team from scoring), those that are skilled offensively (scoring more goals), and those that are "luckier" than others, hence the incorporation of PDO.

```
# Forwards Recipe
forwards_recipe <- recipe(salary ~ ., data = forwards_train) %>%
  step_rm(player) %>%
  step_rm(fow_percent) %>%
  step_dummy(all_nominal(), one_hot = TRUE) %>%
  step_interact(~ p:exp_plus_minus + shots:s_percent + pdo:exp_plus_minus) %>%
  step_lincomb(all_predictors()) %>%
  step_normalize(all_predictors())

forwards_recipe %>%
  prep() %>%
  bake(new_data = NULL)
```

```
## # A tibble: 135 x 30
```

```
##         gp       g       a plus_minus     pim    p_gp     evg     evp      ppg
##      <dbl>   <dbl>   <dbl>      <dbl>   <dbl>   <dbl>   <dbl>   <dbl>    <dbl>
##  1  1.07    0.845   0.467     -1.48    4.26    0.420   1.47    0.984  -0.255
##  2  0.614  -0.459   0.467      0.0633 -0.714  -0.0502 -0.582  -0.0873  0.00982
##  3  0.246  -0.241   0.228      1.35   -0.339  -0.0110  0.0491  0.448  -0.520
##  4 -0.213  -0.350  -0.0106     0.918  -1.09   -0.0894 -0.109   0.219  -0.520
##  5 -0.213   0.0845 -0.408      1.00   -0.0570 -0.129   0.523   0.219  -0.520
##  6 -0.489  -0.133  -0.249     -3.87   -0.0570 -0.0502 -0.109  -0.317   0.00982
##  7  0.890  -0.459  -0.249     -2.24    2.43   -0.599  -1.21   -0.776   1.07
##  8  0.798  -0.459  -0.249      0.491  -0.292  -0.560  -0.425  -0.164  -0.785
##  9  0.0626 -0.567  -0.170     -2.07   -0.996  -0.403  -1.06   -0.776   0.540
## 10 -0.305  -0.0241 -0.647      0.234   0.131  -0.364  -0.267  -0.547  -0.255
## # ... with 125 more rows, and 21 more variables: ppp <dbl>, otg <dbl>,
## #   gwg <dbl>, shots <dbl>, s_percent <dbl>, toi_gp <dbl>, corsi_pct <dbl>,
## #   fenwick_pct <dbl>, tk <dbl>, gv <dbl>, exp_plus_minus <dbl>, pdo <dbl>,
## #   tk_gv <dbl>, salary <dbl>, hand_L <dbl>, hand_R <dbl>, pos_C <dbl>,
## #   pos_L <dbl>, p_x_exp_plus_minus <dbl>, shots_x_s_percent <dbl>,
## #   exp_plus_minus_x_pdo <dbl>
```

```
save(forwards_fold, forwards_recipe, forwards_split, file = "data/forwards_setup.rda")

# Defense Recipe
defense_recipe <- recipe(salary ~ ., data = defense_train) %>%
  step_rm(player) %>%
  step_rm(fow_percent) %>%
  step_rm(pos) %>%
  step_dummy(all_nominal(), one_hot = TRUE) %>%
  step_interact(~ tk_gv:p + p:exp_plus_minus + pdo:exp_plus_minus) %>%
  step_lincomb(all_predictors()) %>%
  step_normalize(all_predictors())

defense_recipe %>%
  prep() %>%
  bake(new_data = NULL)
```

```
## # A tibble: 73 x 28
##         gp       g       a plus_minus    pim   p_gp     evg    evp     ppg    ppp
##      <dbl>   <dbl>   <dbl>      <dbl>  <dbl>  <dbl>   <dbl>  <dbl>   <dbl>  <dbl>
##  1  0.264 -0.273   1.83       -1.24   0.450  1.49  -0.729   0.500  0.743   2.08
##  2 -1.03  -1.05    0.0285     -1.24  -0.338  0.130 -1.43   -1.11   0.0895  0.667
##  3  0.264 -0.273  -0.540       0.358 -1.01  -0.604 -0.0334 -0.121 -0.564  -0.748
##  4  0.183 -0.273  -0.634      -0.861 -0.901 -0.661 -0.381  -0.618  0.0895 -0.324
##  5 -1.20  -0.273  -0.729      -0.404 -1.35  -0.322 -0.0334 -0.493 -0.564  -0.748
##  6  0.427 -0.792  -0.540      -1.39   0     -0.830 -0.729  -0.618 -0.564  -0.606
##  7  0.264 -1.05   -0.445       1.04   1.18  -0.774 -1.08   -0.369 -0.564  -0.748
##  8  0.589 -0.0142 -0.919      -0.404  0.450 -0.943 -0.0334 -0.618  0.0895 -0.606
##  9 -0.628 -0.533  -0.729       0.815  0     -0.661 -0.729  -0.618  0.0895 -0.606
## 10  0.345 -0.792  -0.634      -0.709 -1.01  -0.887 -0.729  -0.493 -0.564  -0.748
## # ... with 63 more rows, and 18 more variables: otg <dbl>, gwg <dbl>,
## #   shots <dbl>, s_percent <dbl>, toi_gp <dbl>, corsi_pct <dbl>,
## #   fenwick_pct <dbl>, tk <dbl>, gv <dbl>, exp_plus_minus <dbl>, pdo <dbl>,
## #   tk_gv <dbl>, salary <dbl>, hand_L <dbl>, hand_R <dbl>, tk_gv_x_p <dbl>,
## #   p_x_exp_plus_minus <dbl>, exp_plus_minus_x_pdo <dbl>
```

```
save(defense_fold, defense_recipe, defense_split, file = "data/defense_setup.rda")
```

Model Tuning

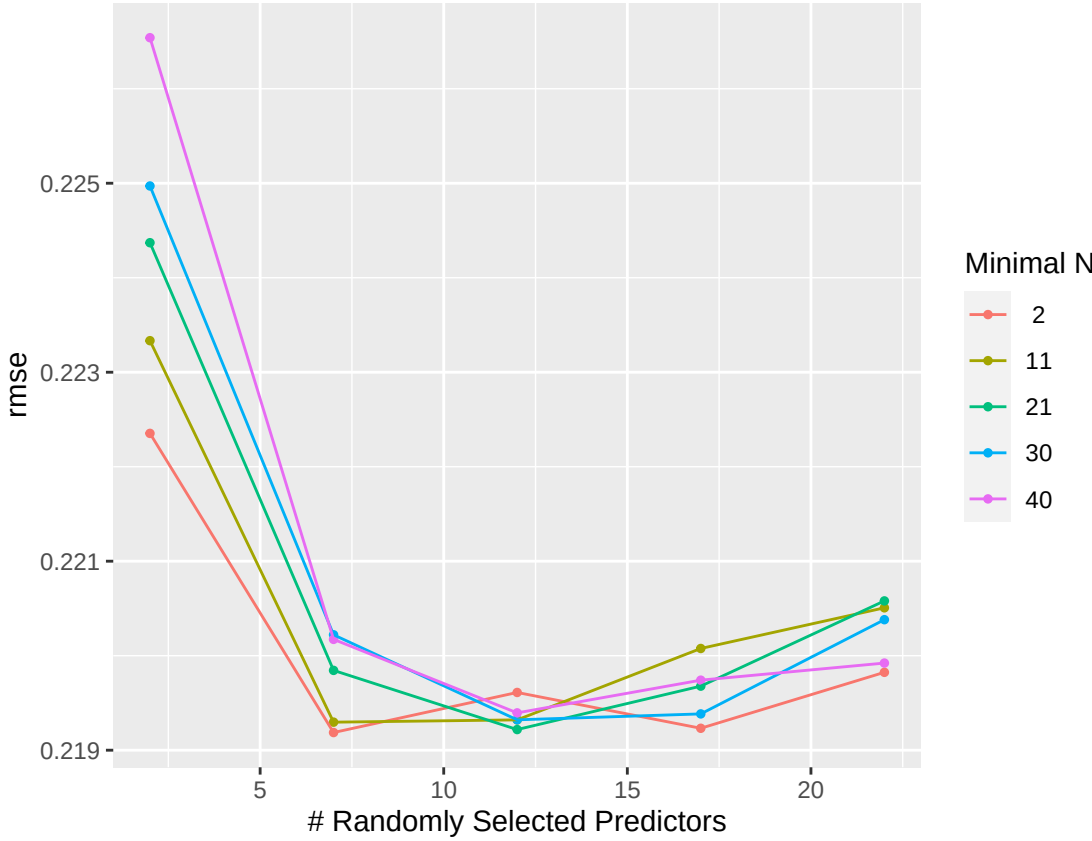
For the random forest models (for both forwards and defensemen), I set the tuning parameters as follows: the mtry was set as a range from 2 to 22 (the number of predictors), and a regular grid set with 5 levels. mtry is defined as the number of predictors that will be randomly sampled at each split when creating tree models. For boosted tree models (for both forwards and defensemen), I used the same mtry range as with random forest models, although I set the learning rate as a range between 10^{-2} and 10^{-5}. The learning rate is a parameter typically used in boosting methods for tree models (hence we get a "boosted tree model"). For k-nearest neighbors models (for both forwards and defensemen), mtry and learning rate are not used.

```
load("data/forwards_rf_tune.rda")
load("data/forwards_bt_tune.rda")
load("data/forwards_knn_tune.rda")

load("data/defense_rf_tune.rda")
load("data/defense_bt_tune.rda")
load("data/defense_knn_tune.rda")
```

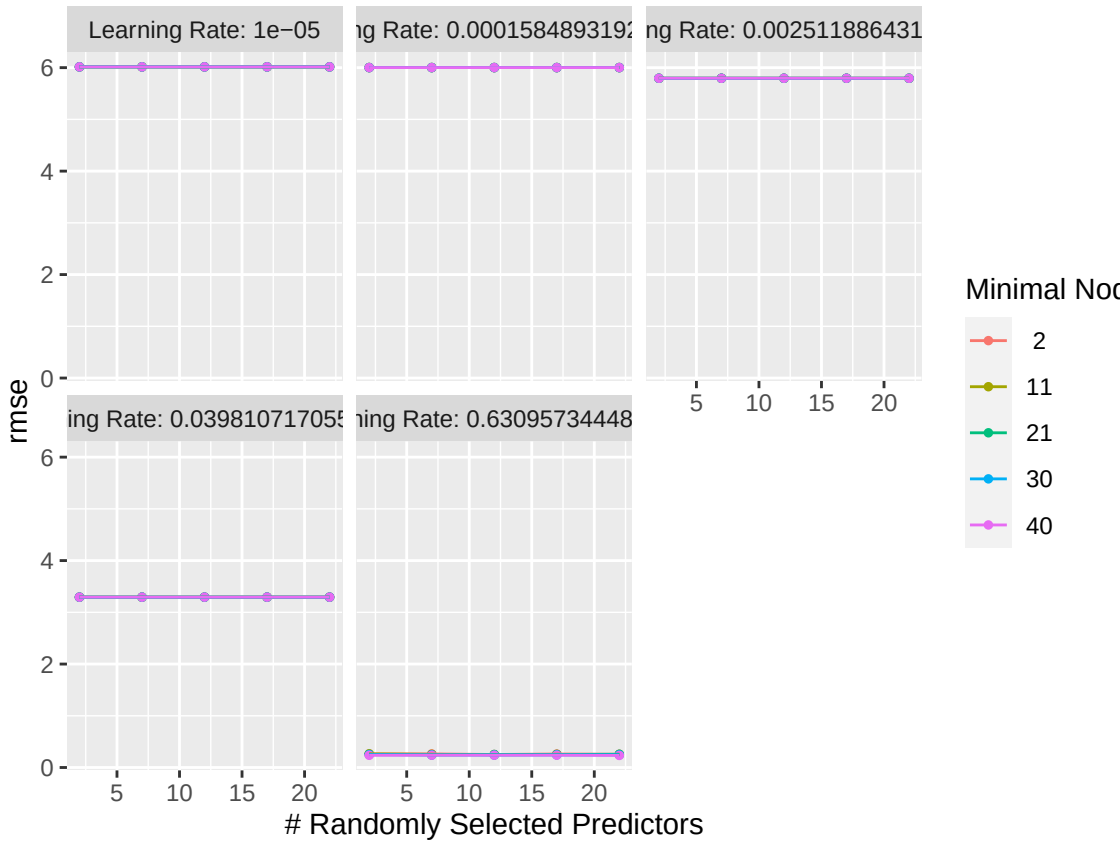
Below are the autoplots for random forest, boosted tree, and k-nearest neighbors models for forwards. The autoplots show the performance profiles across tuning parameters for each of the three models.

```
forwards_rf_tune %>%
  autoplot(metric = "rmse")
```



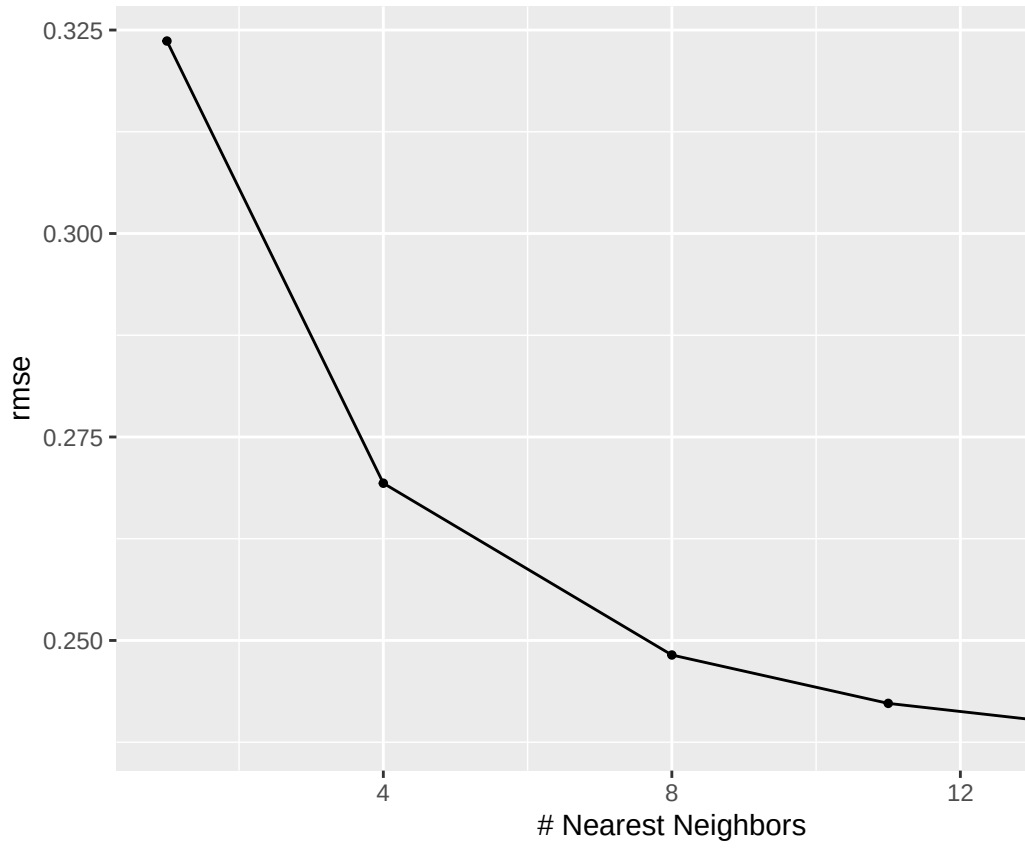
Forwards Random Forest

```
forwards_bt_tune %>%
  autoplot(metric = "rmse")
```



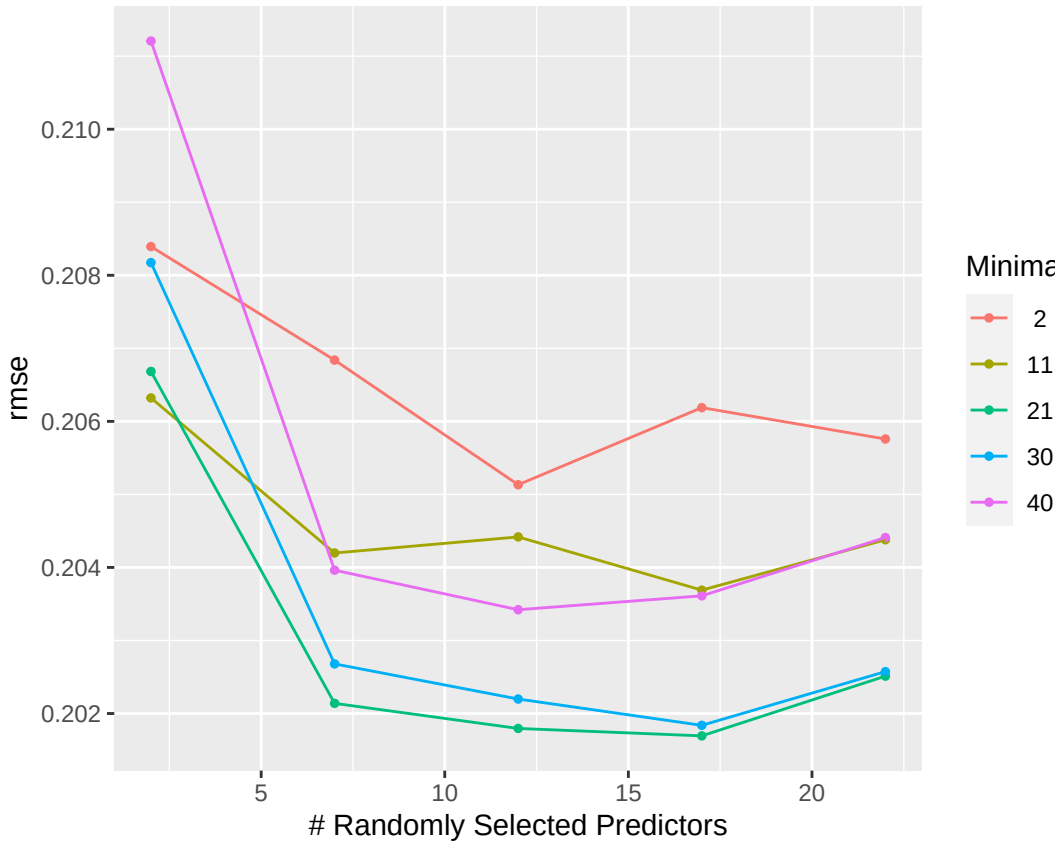
Forwards Boosted Tree

```
forwards_knn_tune %>%
  autoplot(metric = "rmse")
```



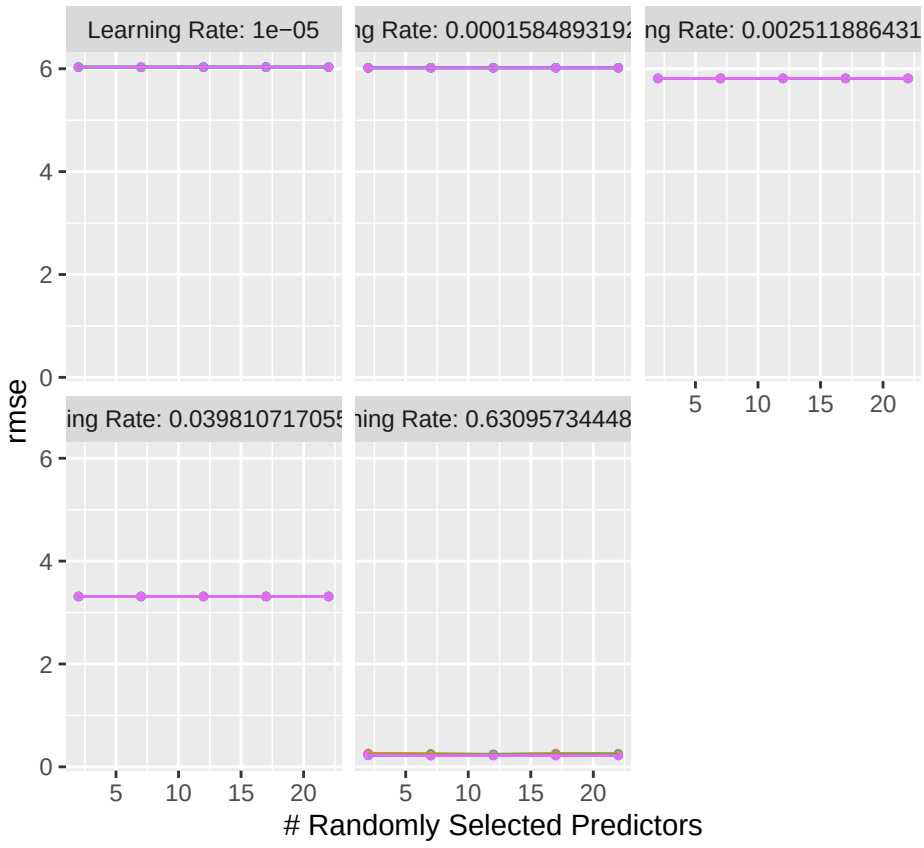
Forwards k-Nearest Neighbors

```
defense_rf_tune %>%
  autoplot(metric = "rmse")
```



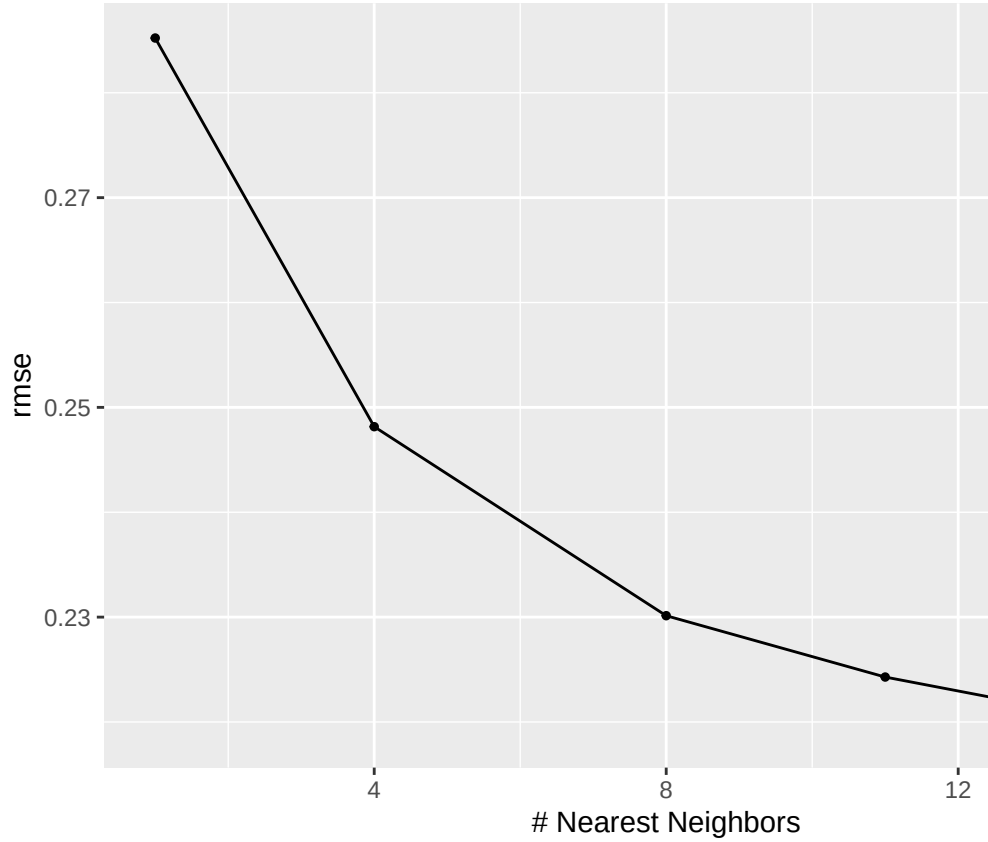
Defensemen Random Forest

```
defense_bt_tune %>%
  autoplot(metric = "rmse")
```



Defensemen Boosted Tree

```
defense_knn_tune %>%
  autoplot(metric = "rmse")
```



Defensemen k-Nearest Neighbors

Model Evaluation

Forwards

The tibbles below demonstrate the models with the smallest root mean squared error for the random forest model, boosted tree model, and k-nearest neighbors model for forwards, respectively.

```
forwards_rf_tune %>%
  select_best(metric = "rmse")
```

```
## # A tibble: 1 x 3
##    mtry min_n .config
##   <int> <int> <chr>
## 1     7     2 Preprocessor1_Model02
```

```
forwards_bt_tune %>%
  select_best(metric = "rmse")
```

```
## # A tibble: 1 x 4
##    mtry min_n learn_rate .config
##   <int> <int>      <dbl> <chr>
## 1    22    40      0.631 Preprocessor1_Model125
```

```
forwards_knn_tune %>%
  select_best(metric = "rmse")
```

```
## # A tibble: 1 x 2
##   neighbors .config
##       <int> <chr>
```

```
## 1        15 Preprocessor1_Model5
```

The tibbles below show the best models for forwards.

```
## # A tibble: 1 x 8
##    mtry min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     7     2 rmse    standard   0.219    25 0.00663 Preprocessor1_Model02
```

Using the training data set for forwards, the root mean squared error for the best-performing model is 0.219, meaning that the predicted salaries are, on average, $10^{0.219}$ times different than the players' true salaries. As shown below, the best performing model for forwards is a random forest model with mtry of 7 and min_n of 2. min_n is defined as the number of data points required to execute a split in a tree-based model, which includes random forest models).

```
tune_results %>%
  select(model_type, assessment_info) %>%
  unnest(assessment_info) %>%
  filter(.metric == "rmse") %>%
  arrange(mean)
```

```
## # A tibble: 155 x 11
##    model_type  mtry min_n .metric .estimator  mean     n std_err .config
##    <chr>      <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1 rf             7     2 rmse    standard   0.219    25 0.00663 Preprocessor1_~
##  2 rf            12    21 rmse    standard   0.219    25 0.00679 Preprocessor1_~
##  3 rf            17     2 rmse    standard   0.219    25 0.00660 Preprocessor1_~
##  4 rf             7    11 rmse    standard   0.219    25 0.00656 Preprocessor1_~
##  5 rf            12    11 rmse    standard   0.219    25 0.00671 Preprocessor1_~
##  6 rf            12    30 rmse    standard   0.219    25 0.00679 Preprocessor1_~
##  7 rf            17    30 rmse    standard   0.219    25 0.00681 Preprocessor1_~
##  8 rf            12    40 rmse    standard   0.219    25 0.00673 Preprocessor1_~
##  9 rf            12     2 rmse    standard   0.220    25 0.00667 Preprocessor1_~
## 10 rf            17    21 rmse    standard   0.220    25 0.00672 Preprocessor1_~
## # ... with 145 more rows, and 2 more variables: learn_rate <dbl>,
## #   neighbors <int>
```

```
forwards_rf_workflow_tuned <- forwards_rf_workflow %>%
  finalize_workflow(select_best(forwards_rf_tune, metric = "rmse"))

forwards_rf_results <- fit(forwards_rf_workflow_tuned, forwards_train)
```

```
## Warning: 'keep_original_cols' was added to `step_dummy()` after this recipe was created.
## Regenerate your recipe to avoid this warning.
```

```
forwards_metric <- metric_set(rmse)
```

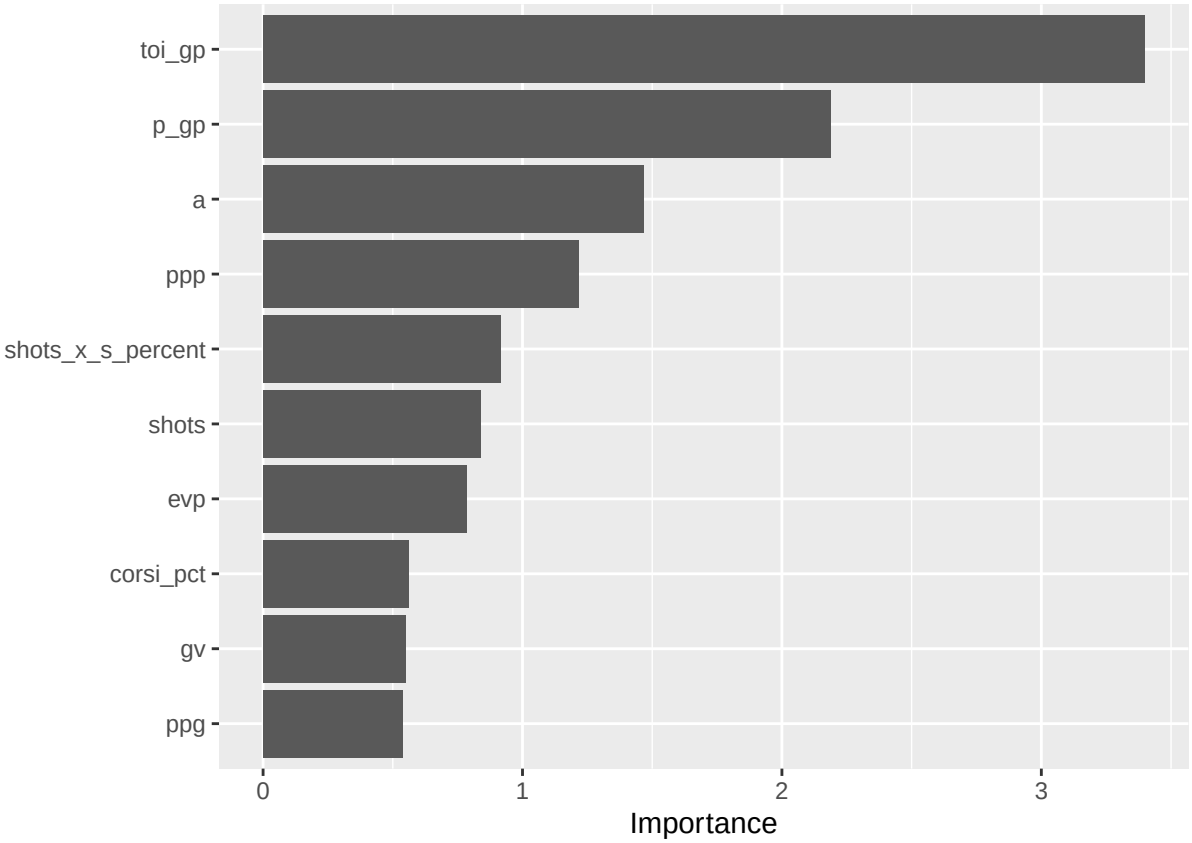
Using the testing data set for forwards, the root mean squared error for the best-performing model is 0.209, which is 0.01 lower than the root mean squared error for the best-performing model using the training set.

```
predict(forwards_rf_results, new_data = forwards_test) %>%
  bind_cols(forwards_test %>% select(salary)) %>%
  forwards_metric(truth = salary, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.225
```

The variable importance plot demonstrates which predictors influence the model the most for predicting a forward's salary. For forwards, the most important predictors are time on ice per game, points per game, assists, power play points, and the interaction term between shots and shooting percentage.

```
load("data/forwards_vip.rda")
forwards_vip
```



Defensemen

The tibbles below demonstrate the models with the smallest root mean squared error for the random forest model, boosted tree model, and k-nearest neighbors model for forwards, respectively.

```
defense_rf_tune %>%
  select_best(metric = "rmse")
```

```
## # A tibble: 1 x 3
##    mtry min_n .config
##   <int> <int> <chr>
## 1    17    21 Preprocessor1_Model14
```

```
defense_bt_tune %>%
  select_best(metric = "rmse")
```

```
## # A tibble: 1 x 4
##    mtry min_n learn_rate .config
##   <int> <int>      <dbl> <chr>
## 1     7    40      0.631 Preprocessor1_Model122
```

```
defense_knn_tune %>%
  select_best(metric = "rmse")
```

```
## # A tibble: 1 x 2
##   neighbors .config
##       <int> <chr>
## 1        15 Preprocessor1_Model5
```

The tibbles below show the best models for defensemen.

```
## # A tibble: 1 x 8
##    mtry min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    17    21 rmse    standard   0.202    25 0.00608 Preprocessor1_Model14
```

Using the training data set for defensemen, the root mean squared error for the best-performing model is 0.202, meaning that the predicted salaries are, on average, $10^{0.202}$ times different than the players' true salaries. As shown below, the best performing model for forwards is a random forest model with mtry of 17 and min_n of 21.

```
tune_results %>%
  select(model_type, assessment_info) %>%
  unnest(assessment_info) %>%
  filter(.metric == "rmse") %>%
  arrange(mean)
```

```
## # A tibble: 155 x 11
##    model_type  mtry min_n .metric .estimator  mean     n std_err .config
##    <chr>      <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1 rf            17    21 rmse    standard   0.202    25 0.00608 Preprocessor1_~
##  2 rf            12    21 rmse    standard   0.202    25 0.00616 Preprocessor1_~
##  3 rf            17    30 rmse    standard   0.202    25 0.00589 Preprocessor1_~
##  4 rf             7    21 rmse    standard   0.202    25 0.00630 Preprocessor1_~
##  5 rf            12    30 rmse    standard   0.202    25 0.00606 Preprocessor1_~
##  6 rf            22    21 rmse    standard   0.203    25 0.00605 Preprocessor1_~
##  7 rf            22    30 rmse    standard   0.203    25 0.00598 Preprocessor1_~
##  8 rf             7    30 rmse    standard   0.203    25 0.00608 Preprocessor1_~
##  9 rf            12    40 rmse    standard   0.203    25 0.00578 Preprocessor1_~
## 10 rf            17    40 rmse    standard   0.204    25 0.00567 Preprocessor1_~
## # ... with 145 more rows, and 2 more variables: learn_rate <dbl>,
## #   neighbors <int>
```

```
defense_rf_workflow_tuned <- defense_rf_workflow %>%
  finalize_workflow(select_best(defense_rf_tune, metric = "rmse"))

defense_rf_results <- fit(defense_rf_workflow_tuned, defense_train)
```

```
## Warning: 'keep_original_cols' was added to `step_dummy()` after this recipe was created.
## Regenerate your recipe to avoid this warning.
```

```
defense_metric <- metric_set(rmse)
```

Using the testing data set for defensemen, the root mean squared error for the best-performing model is 0.243, which is 0.041 higher than the root mean squared error for the best-performing model using the training set.

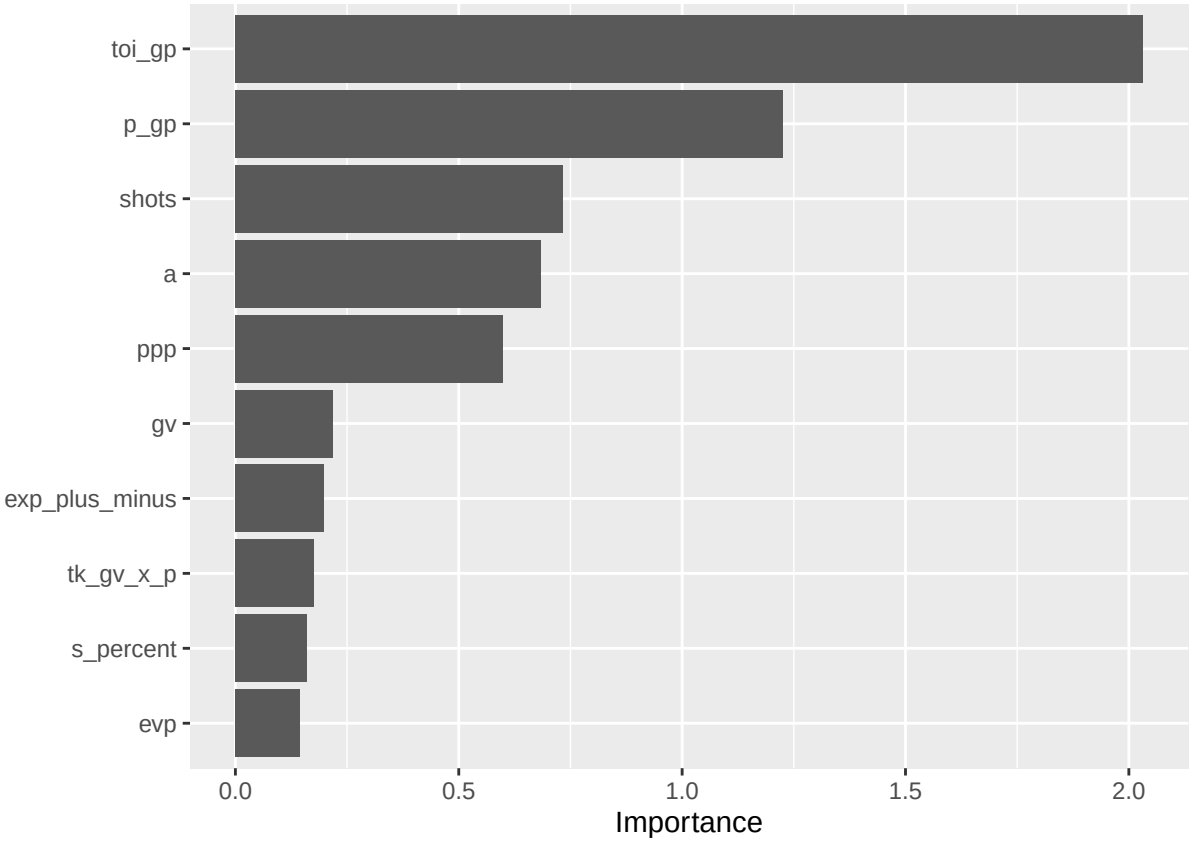
```
predict(defense_rf_results, new_data = defense_test) %>%
  bind_cols(defense_test %>% select(salary)) %>%
  defense_metric(truth = salary, estimate = .pred)
```

```
## # A tibble: 1 x 3
```

```
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.207
```

The variable importance plot demonstrates which predictors influence the model the most for predicting a defenseman's salary. For defensemen, the most important predictors are time on ice per game, points per game, shots, assists, and power play points.

```
load("data/defense_vip.rda")
defense_vip
```




```
# Write out objects for executive summary
save(sal_d, sal_f, forwards_best, def_best, forwards_vip, defense_vip, file = "data/exec_summ.rda")
```

Conclusion

The model results for forwards and defensemen demonstrate that the predicted salaries are, on average, at least $10^{0.2}$ times different from the actual salaries earned by players. As a result, we can conclude that there are other factors besides performance that affect a player's salary. For example, players "in their prime" (i.e., between the ages of 26 and 32) earn more money than players younger than 26 and older than 32. Players in their prime typically earn more money than others for a variety of reasons, including that as players enter the middle of their careers, they tend to score more points and have more of an impact on their teams' success. Additionally, a common trait among such experienced players is good leadership, which cannot be statistically measured. Another factor in determining a player's salary is negotiating leverage, which can be attained for various reasons, such as being a "fan-favorite", or being a home-grown player (meaning that the player was drafted by the team with which he is negotiating). In order to improve the models, it would make sense to attempt to include these other factors as predictors. However, we are guaranteed to have many exceptions because it will be quite difficult to assign numeric values to such intangible traits.