Stat 844 Final Project Report

Dario Greco

2025-04-01

Stat 844 Final Report

Summary/Introduction

After initial exploratory data analysis steps and basic data cleaning, I try and fit different types of models with increasing complexity. That is, I begin with the simple linear model as a baseline and try to improve it. By the time I started the final project I knew that an error rate below 0.35 was very achievable and was what I was aiming for. Some important feature engineering included coming up with more detailed ways of measuring player production per game and incorporating the cap variable as a proxy for the time a player way playing. Eventually I found that that the tree based methods such as random forest, gbm, and xgboost worked quite well, though I could not find much difference in the cross validation results or public scores even when I changed the feature set. So, for my final model I did the average of my four different best tree based methods (3 GBMS and 1 XGB), all trained on slightly different feature sets and all with different tuning parameters. In the end I was able to achieve a public score of 0.29735.

To begin I do some exploratory data analysis of the predictors vs the response. I will rely on the tidyverse package heavily moving forward.

Exploratory Data Analysis

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
load('final.RData')
summary(dtrain)
```

```
##      team               date               salary              cap
##  Length:600         Length:600         Min.   :  550000   Min.   :69.00
##  Class :character   Class :character   1st Qu.:  650000   1st Qu.:71.40
##  Mode  :character   Mode  :character   Median :  813750   Median :71.40
##                                        Mean   : 1728739   Mean   :72.05
##                                        3rd Qu.: 2162500   3rd Qu.:73.00
##                                        Max.   :10500000   Max.   :75.00
##        gp            injure            toi              pts
##  Min.   : 1.0   Min.   : 0.000   Min.   : 2.717   Min.   : 0.00
##  1st Qu.:13.0   1st Qu.: 0.000   1st Qu.:10.968   1st Qu.: 2.00
##  Median :49.0   Median : 1.000   Median :13.733   Median : 9.00
##  Mean   :44.1   Mean   : 5.875   Mean   :13.869   Mean   :15.79
##  3rd Qu.:73.0   3rd Qu.: 8.000   3rd Qu.:16.753   3rd Qu.:25.00
##  Max.   :82.0   Max.   :68.000   Max.   :25.864   Max.   :81.00
##       goal             pos                nat                 age
##  Min.   : 0.000   Length:600         Length:600         Min.   :20.33
##  1st Qu.: 1.000   Class :character   Class :character   1st Qu.:23.93
##  Median : 3.000   Mode  :character   Mode  :character   Median :25.60
##  Mean   : 5.968                                        Mean   :26.37
##  3rd Qu.: 9.250                                        3rd Qu.:28.29
##  Max.   :33.000                                        Max.   :38.65
##     takeaway        giveaway        relCorsi              pim
##  Min.   : 0.00   Min.   :  0.0   Min.   :-122.85198   Min.   :  0.00
##  1st Qu.: 2.00   1st Qu.:  2.0   1st Qu.:  -6.50454   1st Qu.:  4.00
##  Median :10.00   Median : 12.0   Median :  -0.05425   Median : 18.00
##  Mean   :14.75   Mean   : 17.5   Mean   :  -2.34881   Mean   : 23.37
##  3rd Qu.:24.00   3rd Qu.: 27.0   3rd Qu.:   1.60880   3rd Qu.: 32.00
##  Max.   :98.00   Max.   :102.0   Max.   :  45.91270   Max.   :209.00
##       hits              pm
##  Min.   :  0.00   Min.   :-35.0000
##  1st Qu.: 16.75   1st Qu.: -5.0000
##  Median : 42.00   Median : -1.0000
##  Mean   : 57.26   Mean   : -0.9267
##  3rd Qu.: 83.25   3rd Qu.:  2.0000
##  Max.   :365.00   Max.   : 39.0000
```

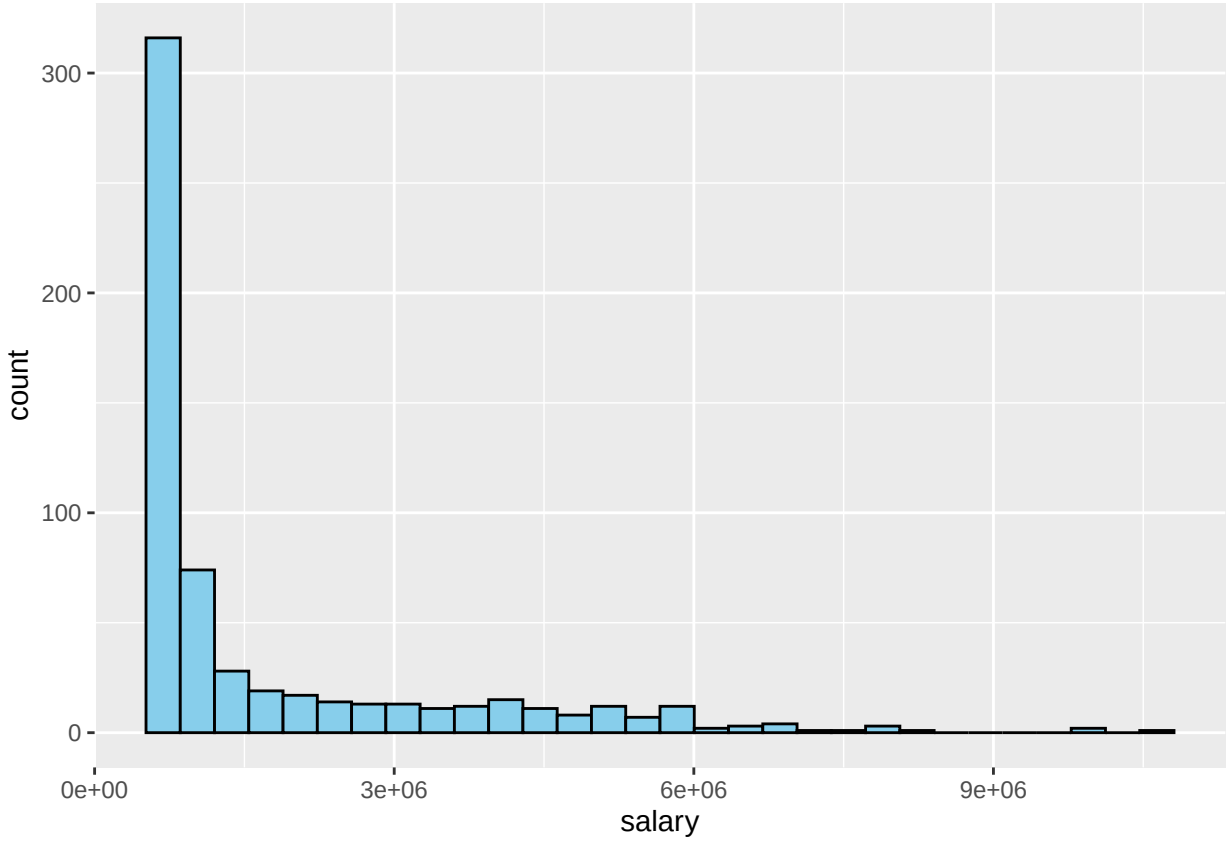
First thing I do is make sure that my categorical variables are indeed factors.

```
dtrain <- dtrain %>%
  mutate(date = as.Date(date),
         team = as.factor(team),
         pos  = as.factor(pos),
         nat  = as.factor(nat))

dtest <- dtest %>%
  mutate(date = as.Date(date),
         team = as.factor(team),
         pos  = as.factor(pos),
         nat  = as.factor(nat))
```

Obviously I should take a look at the response variable here. The tail is very prominent suggesting a log transformation sometime in the future.

```
dtrain %>%
  ggplot(aes(x = salary)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "black")
```




```
  labs(title = "Salary Distribution",
       x = "Salary ",
       y = "Count")
```

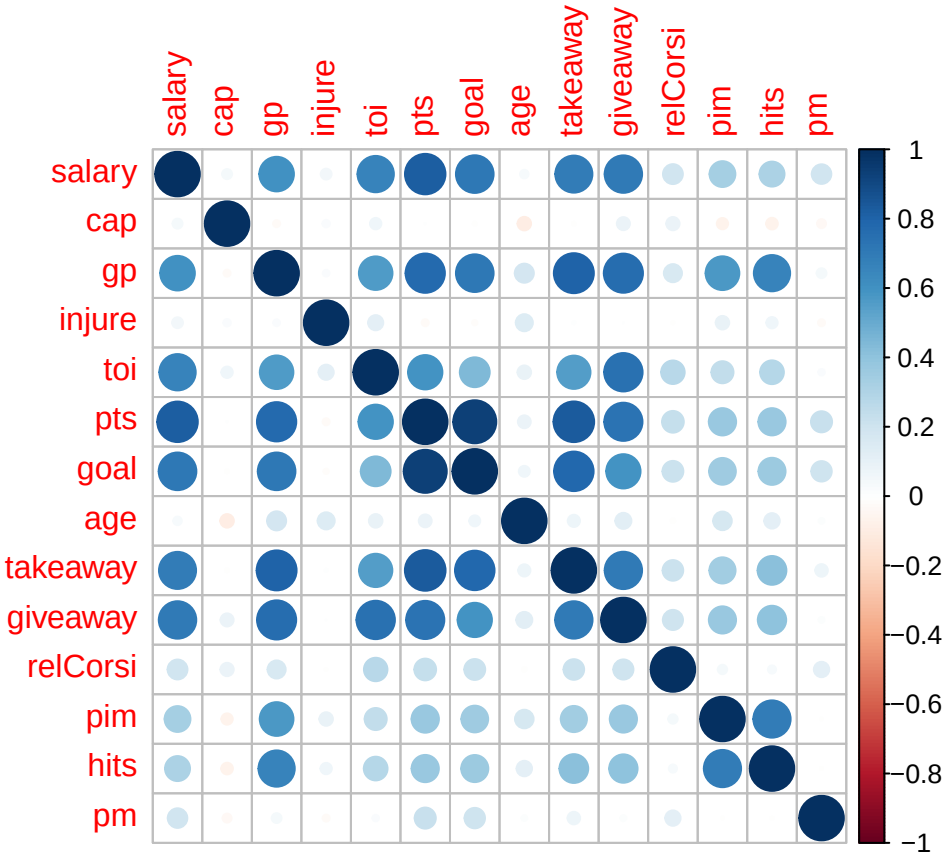
```
## $x
## [1] "Salary "
##
## $y
## [1] "Count"
##
## $title
## [1] "Salary Distribution"
##
## attr(,"class")
## [1] "labels"
```

Next I take a look at the numeric predictors and run a correlation matrix with them against the salary response

```
library(corrplot)
```

```
## corrplot 0.95 loaded
```

```
quant_vars <- dtrain %>%
  select_if(is.numeric)
cor_matrix <- cor(quant_vars, use = "pairwise.complete.obs")
corrplot(cor_matrix, method = "circle")
```



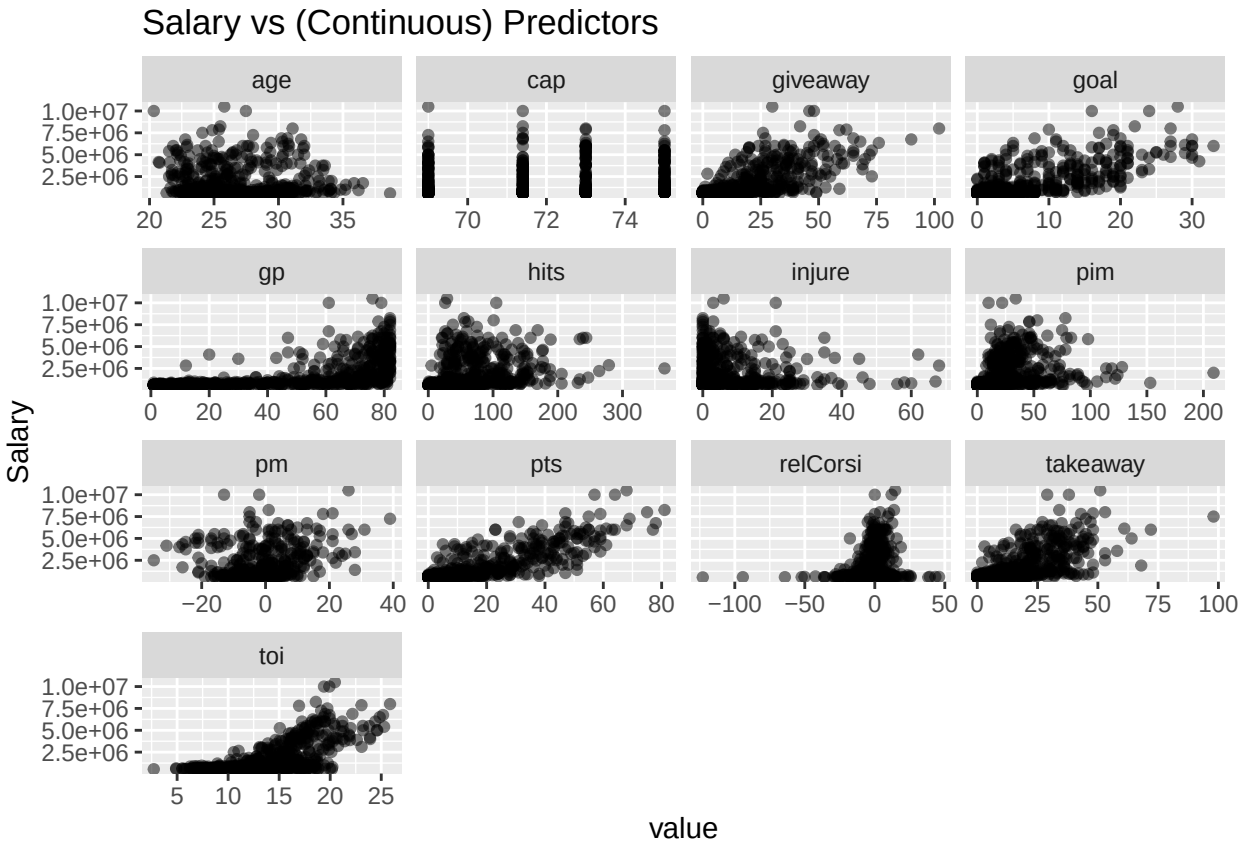
A couple things to note from the correlation matrix (which I make with the helpful corrplot package). One is the relationship between points and predictors like salary, goals, toi, and gp. The other is the strong correlation between goals and points. An expected relationship considering that goals contribute to a players point total. Takeaways, giveaways, and toi also have a strong positive relationship with salary. Perhaps a surprising results is the lack of correlation between age and salary.

We now explore the relationship between salary and the predictors through graphical methods.

```
# Make a long dataframe for easier plotting
continuous_vars <- c("cap", "gp", "injure", "toi", "pts", "goal",
                     "age", "takeaway", "giveaway", "relCorsi",
                     "pim", "hits", "pm")

# This will put all the (continous) predictors on top of each other so GGplot can visualize it easier
dtrain_long <- dtrain %>%
  pivot_longer(cols = all_of(continuous_vars), names_to = "predictor", values_to = "value")
```

```r
# Plot salary vs pred
ggplot(dtrain_long, aes(x = value, y = salary)) +
  geom_point(alpha = 0.5) +
  facet_wrap(~ predictor, scales = "free_x")+
  labs(title = "Salary vs (Continuous) Predictors",
       y = "Salary ")
```



Points, and by extension goals, have an obvious positive relationship with salary. Total on ice time and games played have a clear positive relationship with salary. You would expect the good players to be both on the ice and in the lineup more, and consequently get paid more. Takeaways and giveaways also have a positive relationship with salary, but this is likely due to the previous fact which is that good players, and, hopefully, higher paid players, will be on the ice more, giving them more chances to both takeaway the puck and giveaway the puck.

Preprocessing

I obverse the following discrepancies in the data, seen in the R chunk below. Moving forward I would like my data set to be cleaned such that I have a reasonable amount of positions. For example having a category for both RW/C and C/RW seems to be unreasonable. My intuition is that there are three main types of a players in hockey (not including goalies). They are the position of centre, winger, and defense. In general, I have observed, through years of playing ice hockey myself, that a defense is more capable of playing forward than a forward is of playing defense. This means that if a player is put down as both F/D where F is some forward position, I have assumed that their natural position is that of defense.

In a similar manner, I have found that it is easier for a centre to be decent at winger than it is for a winger

to be good at centre. So if a player is even remotely capable of playing centre they are more valuable (think of situations where the centre gets kicked out of the faceoff and a winger needs to jump in as a replacement).

So I have established some hierarchy in how the position strings should be modified. As a result I have filtered the positions in the following manner. If a given position string has "D" they become a "Defense", out of the remaining players any that have a "C" become "Centre" and the remaining are "Wingers".

```
table(dtrain$pos)
```

```
##
##       C     C/D    C/LW C/LW/RW    C/RW C/RW/LW       D    D/RW      LW    LW/C
##     122       1      23       3      22       3     201       1      65      21
## LW/C/RW    LW/D   LW/RW LW/RW/C      RW    RW/C   RW/LW RW/LW/C
##       4       1      22       2      62      16      29       2
```

```
table(dtest$pos)
```

```
##
##       C    C/LW    C/RW C/RW/LW       D      LW    LW/C LW/C/RW   LW/RW      RW
##      79      26      14       1     143      45      10       2      13      42
##    RW/C RW/C/LW   RW/LW RW/LW/C
##       8       1      15       1
```

```
# This is My helper function to implement the intuition laid out above
generalize_position <- function(pos) {
  pos <- toupper(pos)
  if (str_detect(pos, "D")) {
    return("Defense")
  } else if (str_detect(pos, "C")) {
    return("Centre")
  } else if (str_detect(pos, "LW|RW")) {
    return("Winger")
  } else {
    return("Other")
  }
}

# apply to both the training and test set to avoid confusion moving forward
dtrain <- dtrain %>%
  mutate(pos_general = map_chr(pos, generalize_position)) %>%
  mutate(pos_general = as.factor(pos_general)) # make sure we have factors

summary(dtrain$pos_general)
```

```
##  Centre Defense  Winger
##     218     204     178
```

```
# apply to the test set in real time
dtest <- dtest %>%
  mutate(pos_general = map_chr(pos, generalize_position)) %>%
  mutate(pos_general = as.factor(pos_general))

summary(dtest$pos_general)
```

```
##  Centre Defense  Winger
##     142     143     115
```

Another benefit of such a generalization is the fact that the training and testing data have similar relative amounts of each player position. This is an improvement from above when some position types in the training set were not present in the testing set.

Though I suspect that nationality will not be too informative, after all a good player is a good player and should get paid accordingly, I make a similar edit to the dataset where a player is either from "Canada", "USA", or "other",

```
table(dtrain$nat)
```

```
##
## AUT CAN CHE CZE DEU DNK FIN FRA HRV ITA NOR RUS SVK SVN SWE USA
##   5 322  12  14   5   2  14   3   1   1   1  13   5   1  41 160
```

```
table(dtest$nat)
```

```
##
## AUT CAN CHE CZE DEU DNK FIN LVA NOR RUS SVK SWE USA
##   1 214   3  13   4   2   8   2   1  12   4  21 115
```

```
# applt to training
dtrain <- dtrain %>%
  mutate(nat_group = if_else(nat == "CAN", "Canada",
                             if_else(nat == "USA", "USA", "Other"))) %>%
  mutate(nat_group = as.factor(nat_group)) # needs factors

# apply to test
dtest <- dtest %>%
  mutate(nat_group = if_else(nat == "CAN", "Canada",
                             if_else(nat == "USA", "USA", "Other"))) %>%
  mutate(nat_group = as.factor(nat_group))

table(dtrain$nat_group)
```

```
##
## Canada  Other    USA
##    322    118    160
```

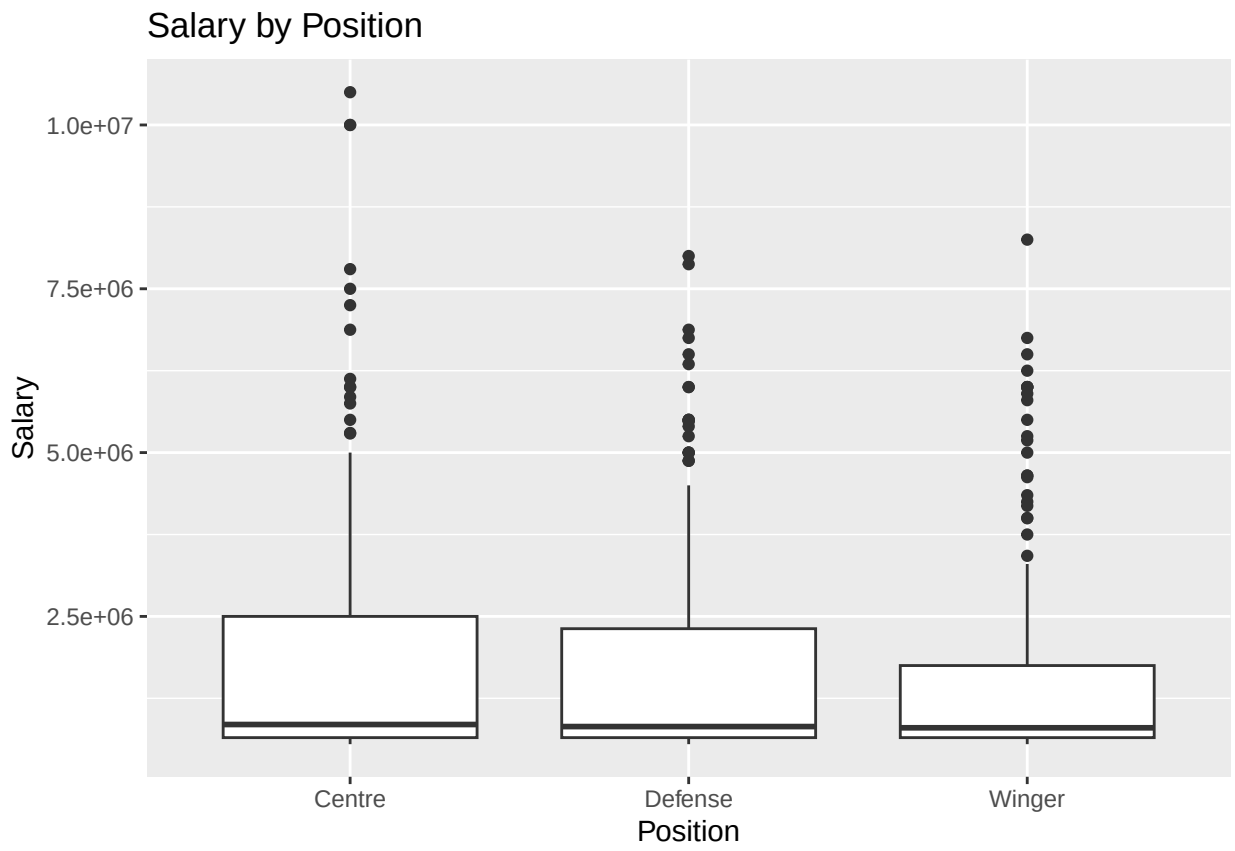
```
table(dtest$nat_group)
```

```
##
## Canada  Other    USA
##    214     71    115
```

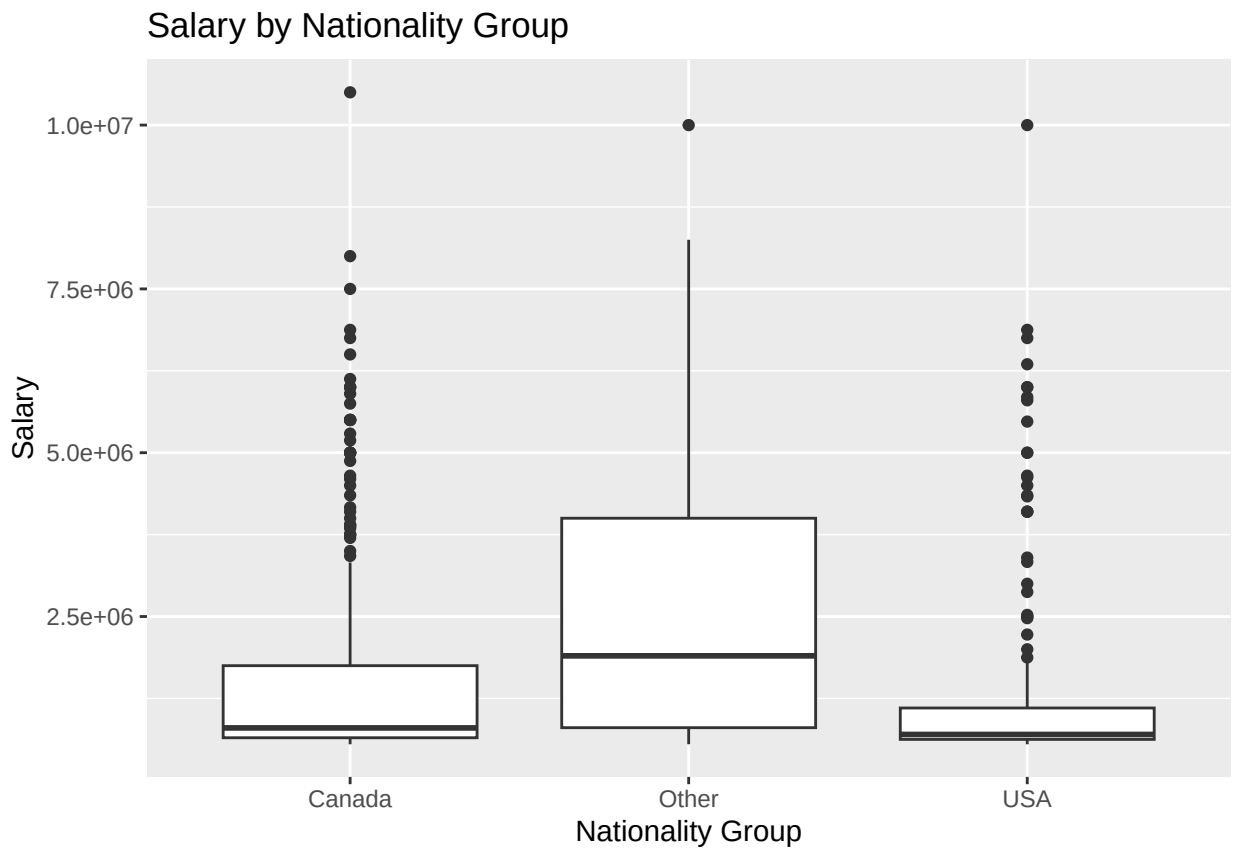
Once again I see the benefits of such an aggregation. Countries such as Italy and France, which are present in the training set, are not present in the test set. By doing this cleaning now I am, hopefully, saving any future issues for running my models on the test set.

It is now reasonable to look the cleaned up categorical variables vs the salary response

```r
# Salary vs Position
ggplot(dtrain, aes(x = pos_general, y = salary)) +
  geom_boxplot() +
  labs(title = "Salary by Position", x = "Position", y = "Salary")
```




```r
# Salary vs Nationality
ggplot(dtrain, aes(x = nat_group, y = salary)) +
  geom_boxplot() +
  labs(title = "Salary by Nationality Group", x = "Nationality Group", y = "Salary")
```

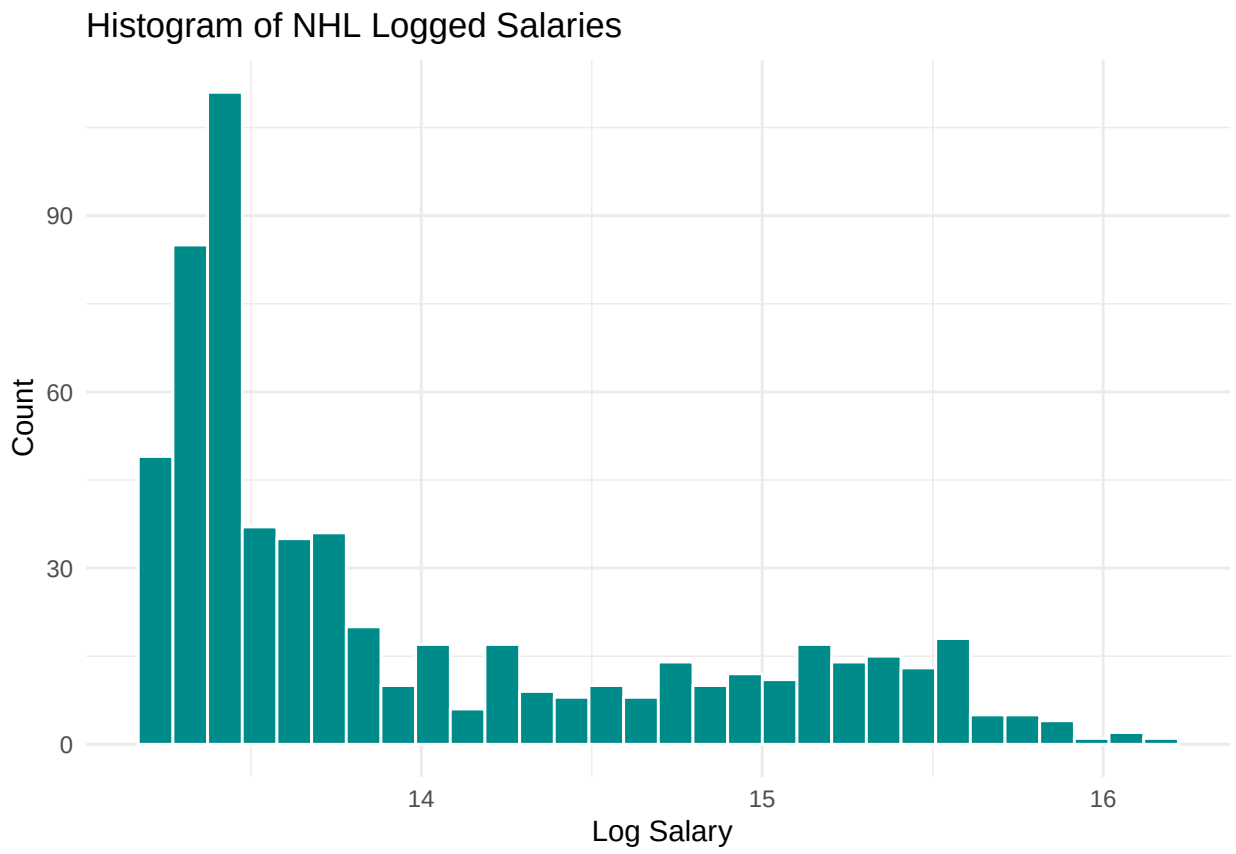


All the position variables have similar distributions in the training set. I see an interesting relationship in the national group boxplots where players not from Canada or the USA seem to get paid more on average. A working hypothesis for this could be the fact that an international player who is only capable of making below league average money in the NHL may prefer to make less money in Europe. For example, a Russian player may prefer to be a star in the KHL (Russian hockey league), than a lower ranked player in the NHL, while still being able to make (relatively) similar money. Not to mention any work, tax, or lifestyle reason an international player may prefer to not play in the NHL.

It is clear a transformation should be applied to salary. I apply a log1p (log (1 + x)) transformation as I have assumed that is the same one that will be used in the RMLSE calculation on Kaggle based off of this article: https://www.kaggle.com/code/carlolepelaars/understanding-the-metric-rmsle.

NOTE: I realized later in the "evaluation.R" that we are just taking the normal log. By then I had already done most of my analysis so I thought I would keep my log1p transformation to be consistent with my Kaggle public scores. The difference should be marginal and non-consequential anyways.

```
ggplot(dtrain, aes(x = log1p(salary))) +
  geom_histogram(bins = 30, fill = "cyan4", color = "white") +
  labs(title = "Histogram of NHL Logged Salaries", x = " Log Salary", y = "Count") +
  theme_minimal()
```




```
dtrain$log_salary <- log1p(dtrain$salary)
```

New Variables

I came up with some potentially useful variables that I will be using in some combination with each of the tree methods below. The first few are pretty obvious, points per game, points per time on ice, and goals per time on ice, are all measures of a players efficiency. For example there could be some player that does not play as much but gets a lot of points in their role. I hope the contribution of such players is reflected in these variables.

Interactions are included despite the fact that the tree methods are meant to find them. I include an interaction between time on ice and takeaways. I suspect that this will measure a players defensive contributions well. Furthermore, we have seen from the correlation matrix above that takeaway and toi are important for predicting salary. An interaction between points and age is also added. The reason for this is that younger players may be given a big salary based of their "potential production", while an older player with a similar current production will likely not be paid in the same way.

Finally, since we are interested in predicting salary, I thought I would try and use the only other monetary predictor in the dataset, that is the cap variable, more intentionally. The cap also implicitly encodes the time the player got the contract, which may be its greatest utility. That is, the cap has tended to increase with time, and all else equal, similar players get paid more now in absolute terms than they did say two decades ago. A point per game player got paid a lot less in the 80s than they did now.
I include a pts and cap interaction. I suspect that on either the higher or lower end of the salary spectrum, players can be effected by the cap quite a bit. So given that pts is a pretty good proxy for a players salary the interaction seemed natural. If this intention was not captured by points, hopefully it is captured by the

interaction between cap and time on ice, another good proxy for the quality of a player. Lastly I include the ratio of points to cap. This can potentially capture how much each point a player makes "is worth", or the money spent by the team, given the cap, has been "efficient". This could also capture what I mentioned above about players getting paid differently depending on what "era" they are in. Perhaps players get paid a "ratio" of the cap depending on their production. I suspect that if our dataset was much larger and include players from the 90s or earlier 2000s, these cap variables would become extremely important in normalizing everyone.

```
data_cleaner_feat <- function(df) {
    df <- df %>% mutate(
        pts_per_gp = pts/gp,
        pts_per_min = pts/toi,
        goals_per_min =  goal/toi,
        cap_pts_int = cap*pts,
        toi_takeaway_int = toi*takeaway,
        age_pts_int = age*pts,
        cap_toi_int = cap*toi,
        cap_ratio = pts/(cap)
      )
    return(df)
}

# This function is usefor for applying to the dtest set when it comes time for predictions.
dtrain = data_cleaner_feat(dtrain)
```

Inital Model Fitting

I first do the most obvious thing which is to a fit a linear model with all the predictors. I make use of the caret library here which allows for easy cross validation checking. Some documentation I found useful can be found here: https://topepo.github.io/caret/model-training-and-tuning.html

Given the disproportionate distribution of salary, I make the obvious logarithm transformation. Also, since I have already taken the log transform of my data, the built in RMSE evaluation should be a good proxy for the RMLSE of the raw salary.

```
library(caret)
control <- trainControl(method = "cv", number = 5)

model_lm_1<- train(log_salary ~ cap + gp + injure + toi + goal + pos + nat_group + age +
                     takeaway + giveaway + relCorsi + pim + hits + pm + pos_general,
                   data = dtrain,
                   method = "lm",
                   trControl = control,
                    metric = "RMSE")

print(model_lm_1)
```

```
## Linear Regression
##
## 600 samples
##  15 predictor
##
```

```
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 480, 480, 480, 480, 480
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.3829977  0.7712496  0.2988694
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Next I try some basic models that seem reasonable given my above predictor vs response plots. I test out some simple feature engineering with a non-linear term for total ice time, points per game, and an interaction between points and position.

```
model_lm_2<- train(log_salary ~ pts + toi + toi^2 + takeaway + nat_group + pts*pos_general + pts/gp,
                   data = dtrain,
                   method = "lm",
                   trControl = control,
                   metric = "RMSE")

print(model_lm_2)
```

```
## Linear Regression
##
## 600 samples
##   6 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 480, 480, 480, 480, 480
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.3707525  0.7809884  0.2709167
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Next I try some GAM models, also using caret.

```
set.seed(1)
library(caret)
library(mgcv)
model_gam_spline <- train(
  log_salary ~ cap + gp + injure + toi + goal + pos + nat_group + age +
    takeaway + giveaway + relCorsi + pim + hits + pm + pos_general,
  data = dtrain,
  method = "gamSpline",
  trControl = control,
  metric = "RMSE"
)

print(model_gam_spline)
```

```
## Generalized Additive Model using Splines
##
## 600 samples
##  15 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 480, 480, 480, 480, 480
## Resampling results across tuning parameters:
##
##   df  RMSE       Rsquared   MAE
##   1   0.3722196  0.7726635  0.2916149
##   2   0.3437694  0.8048567  0.2591042
##   3   0.3375277  0.8116535  0.2497425
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was df = 3.
```

And I get a pretty reasonable CV-error with the df = 3 model.

Tree Based methods

The tree based methods seem like a good candidates for this dataset. For one they will be able to find some non-obvious interactions terms in the data that I may be overlooking. Second, they will be able to account for interactions terms implicitly, saving me the effort of trying to find some clever feature engineering (beyond what I have already done). I consider Random Forest, Gradient Boosting Methods and XGBoost below, once again all within the caret framework.

Random Forest

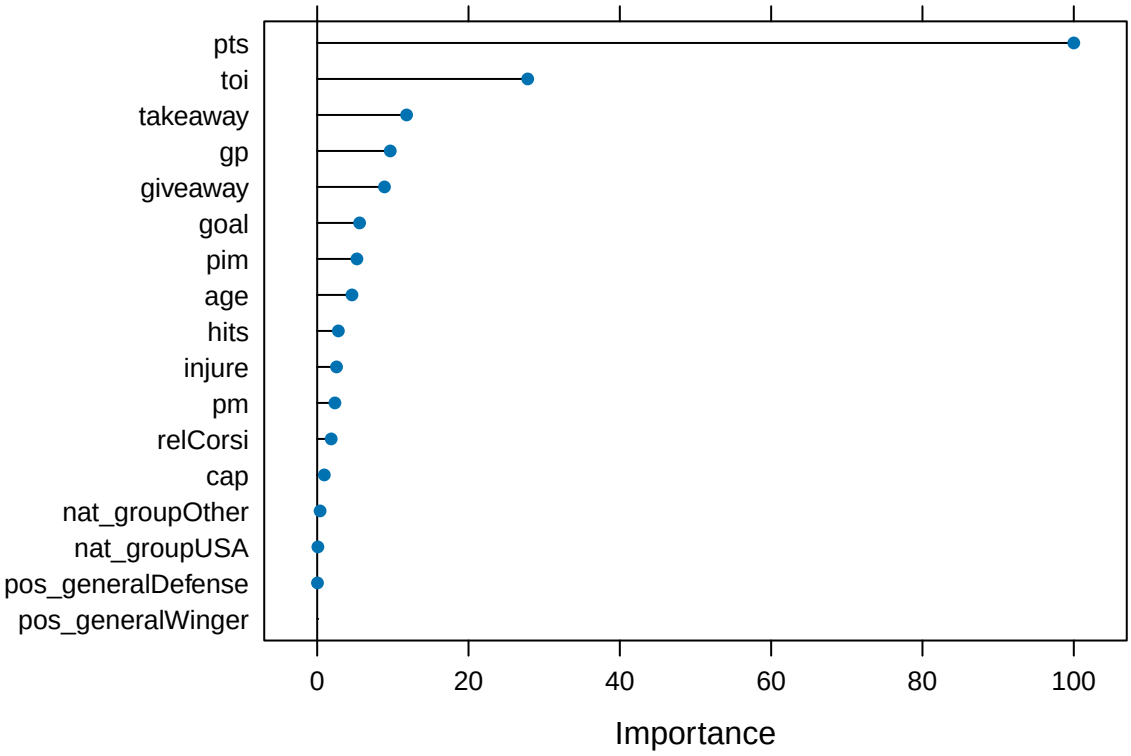
From the model summary below I can see that the random forest did a pretty reasonable job of getting a good RMLSE quickly.

```
rf_model_1$results
```

```
##   mtry      RMSE  Rsquared       MAE     RMSESD RsquaredSD      MAESD
## 1    2 0.3464113 0.8152794 0.2463354 0.02627527 0.03504695 0.02313788
## 2    5 0.3352967 0.8208920 0.2315881 0.02943241 0.03947503 0.02289472
## 3    9 0.3314713 0.8238094 0.2280458 0.03090761 0.03969394 0.02197118
## 4   13 0.3313180 0.8234617 0.2262527 0.03130512 0.03989817 0.02165752
## 5   17 0.3346080 0.8199904 0.2279958 0.02764762 0.03613298 0.01904377
```

I can also take a look at a variable importance plot. I see quite an expected result. Points are the most important with total ice time quite far behind in second. This is similar to the results of the correlation matrix above. Surprisingly position is not mattering much.

```
plot(varImp(rf_model_1))
```



Some other random forests were tried with more expansive tuning grids but I had mostly the same results above and have omitted them. The next method I tried was the gradient boosting methods.

Gradient boosting methods gave me an opportunity to really try some serious model tuning. The number of trees, interaction depth, learning rate, and minimum observations per leaf all had to be tuned. This is also when I began using the feature engineered variables used above.

#Gradient Boosting Methods

I return to fitting models with some of my new engineered variables and most of the original predictors.

```
library(gbm)
set.seed(123)

gbm_grid <- expand.grid(
  n.trees = c(50, 75, 100, 150,  200, 300, 600),
  interaction.depth = c(3, 4, 5, 6),
  shrinkage = c(0.01, 0.05, 0.075, 0.1),
  n.minobsinnode = c(5, 10, 15)
)

gbm_model_1 <- train(
  log_salary ~ toi_takeaway_int + cap_pts_int + toi + pts + age_pts_int +
    pim + age + giveaway + injure + pts_per_gp + hits + pm + gp +
    pos_general + nat_group, data = dtrain,
  method = "gbm",
  trControl = control,
  tuneGrid = gbm_grid,
```

```
  metric = "RMSE",
  maximize = FALSE,
  verbose = FALSE
)
```

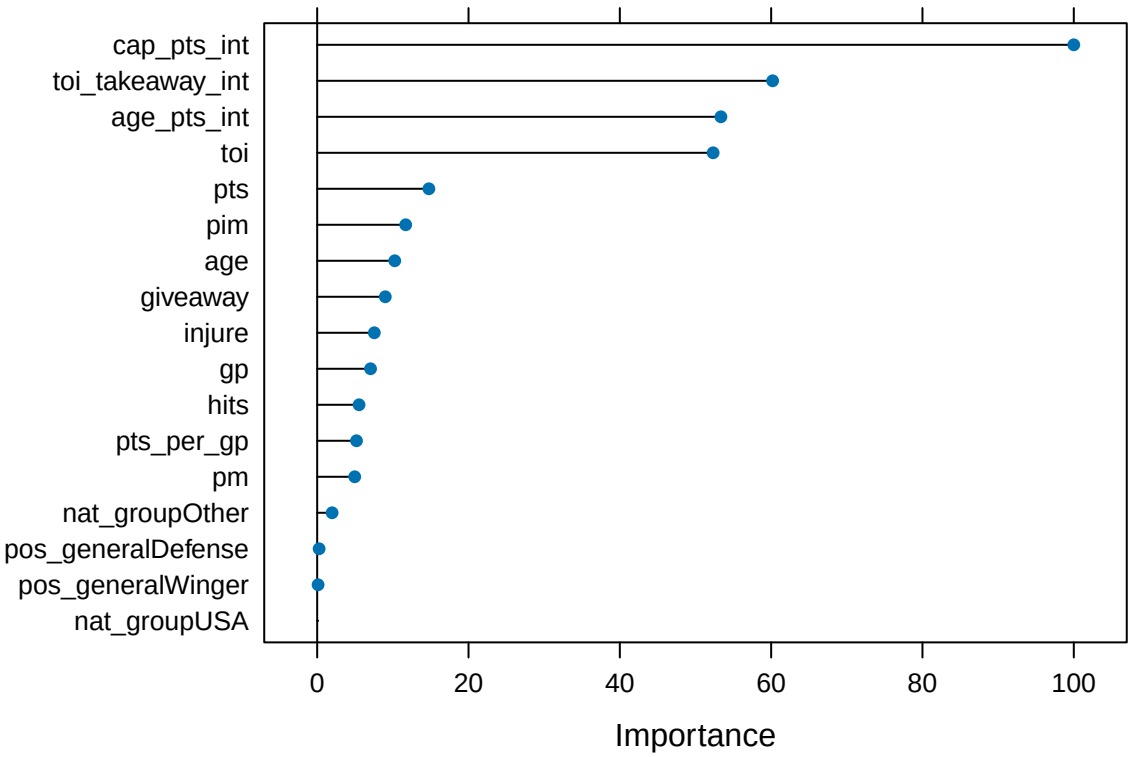
I see a very good improvement in my 5-fold cross validation RMSE with a measured RMSE of 0.3247. Furthermore I see that my engineering variables are making a difference. There is large decreases in importance of raw points, as I suspect much of the effect has been absorbed by some of my interaction terms. The best tuned model has parameters of n.trees = 600, interaction.depth = 5, shrinkage = 0.01, and n.minobsinnode = 10. The was my first submission on the Kaggle test set and it got a score of 0.30140.

```
library(gbm)
```

```
## Loaded gbm 2.2.2
```

```
## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.co
```

```
plot(varImp(gbm_model_1))
```




```
cat("The lowest RMSE, achieved by the best tune, is: ", min(gbm_model_1$results$RMSE))
```

```
## The lowest RMSE, achieved by the best tune, is:  0.3247107
```

Here is the best tune:

```
gbm_model_1$bestTune
```

```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 56     600                 5      0.01             10
```

Now I try to manually input some other interaction. Up to now position has not been very relevant and I found that strange. I am worried the models use points too much and are not accounting for the fact that defense will naturally get less points, but can still be as valuable, if not more, than forwards with similar numbers. I try and make a slightly more simple search grid in order to save on computation time.

```
set.seed(123)

gbm_grid2 <- expand.grid(
  n.trees = c(550, 600, 700),
  interaction.depth = c(6,8),
  shrinkage = c(0.01, 0.05, 0.075, 0.1),
  n.minobsinnode = c(3,4,5)
)
gbm_model_2 <- train(
  log_salary ~ cap_pts_int + age_pts_int + toi_takeaway_int + pts_per_gp +
    toi + pim + age + giveaway + injure + hits + pm + gp +
    goal + pts:toi + pts:pos_general + toi:pos_general + goal:pos_general, data = dtrain,
  method = "gbm",
  trControl = control,
  tuneGrid = gbm_grid2,
  metric = "RMSE",
  maximize = FALSE,
  verbose = FALSE
)
```

The following was the best tune. The model got a 0.30529 score on kaggle which was slightly worse than the previous model but still shows signs of potential

```
gbm_model_2$bestTune
```

```
##   n.trees interaction.depth shrinkage n.minobsinnode
## 1     550                 6      0.01              3
```

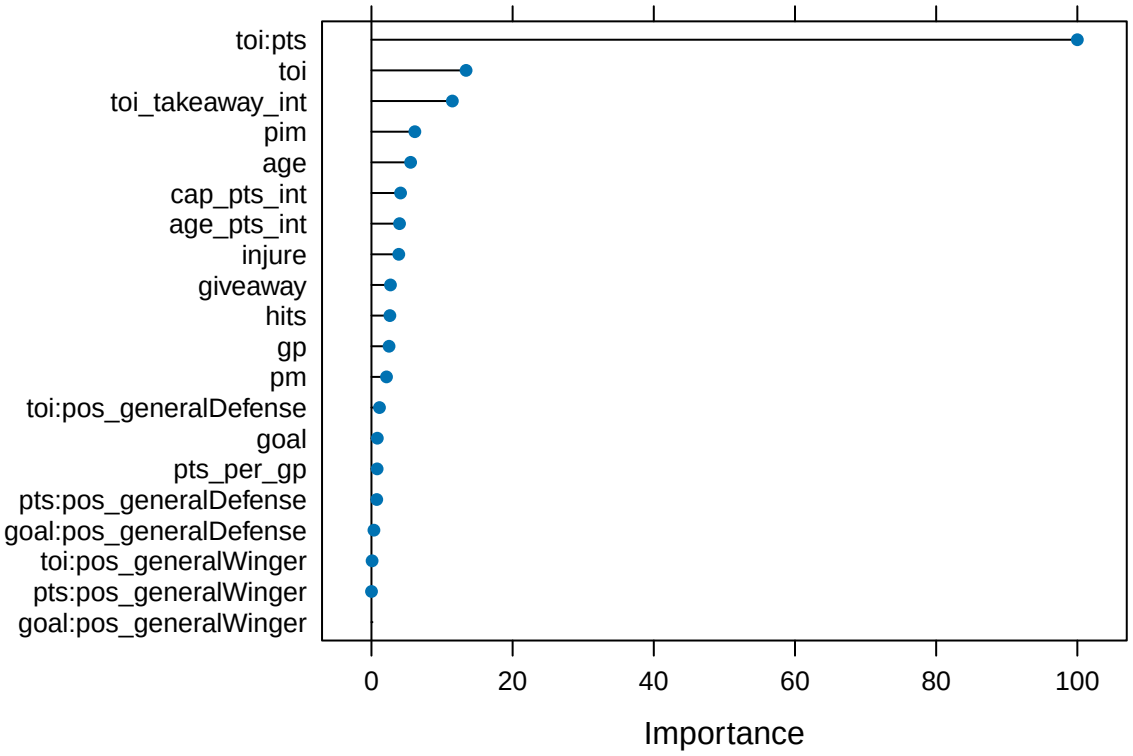
Here is the cross validation results

```
min(gbm_model_2$results$RMSE)
```

```
## [1] 0.3267436
```

The variable importance plot

```
plot(varImp(gbm_model_2))
```



We did not see much improvement. Next I look at adding in the cap related variables instead to see if there is an improvement.

```
set.seed(123)
gbm_grid3 <- expand.grid(
  n.trees = c(400, 600),
  interaction.depth = c(6, 7),
  shrinkage = c(0.005, 0.015, 0.2),
  n.minobsinnode = c(3,5)
)

gbm_model_3<- train(
  log_salary ~ toi_takeaway_int + cap_pts_int + toi + pts + age_pts_int +
    pim + age + giveaway + injure + pts_per_gp + hits + pm + gp
  + goal + pts_per_min+ goals_per_min + cap_toi_int + cap_ratio, data = dtrain,
  method = "gbm",
  trControl = control,
  tuneGrid = gbm_grid3,
  metric = "RMSE",
  maximize = FALSE,
  verbose = FALSE
)
```

Here is the best tune and CV RMSE from this model. It achieved a 0.30953 test score on Kaggle. The cap related engineering variables seem to being making some difference. Overall, the above three models all work in a similar manner.

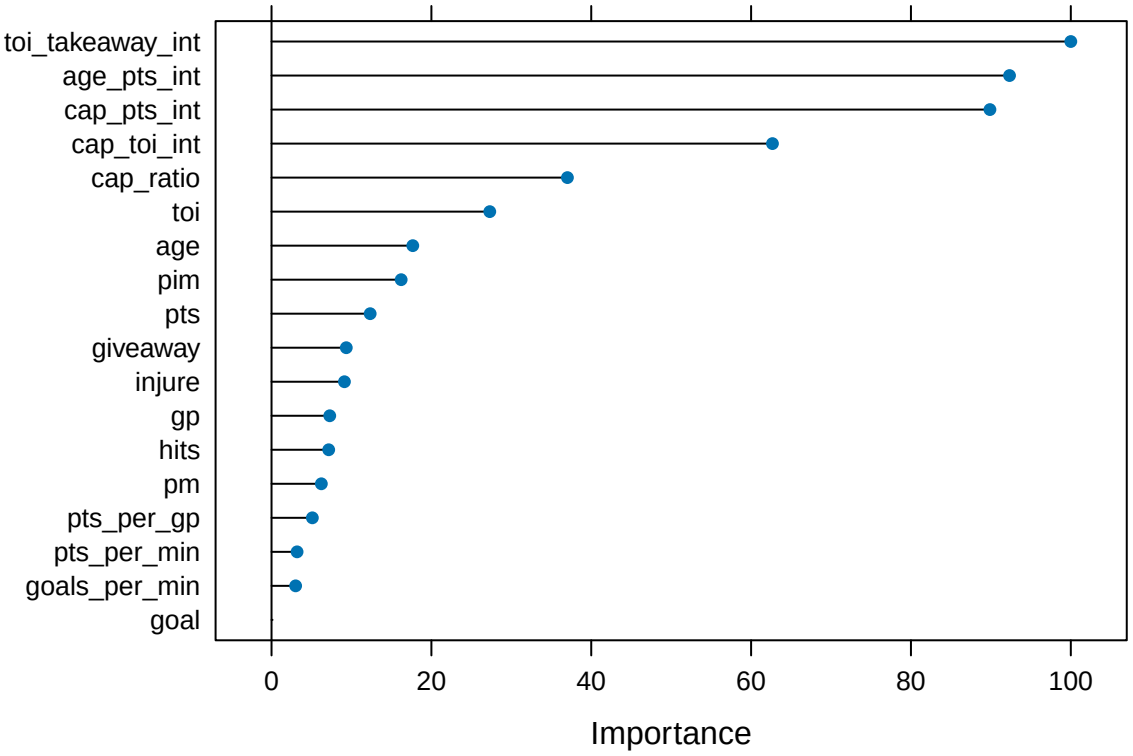
```
gbm_model_3$bestTune
```

```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 15     400                 7     0.015              5
```

```
min(gbm_model_3$results$RMSE)
```

```
## [1] 0.3239964
```

```
plot(varImp(gbm_model_3))
```



Finally I try xgboost, the natural next step after the gradient boosting methods, and a model I knew well from my STAT 841 class. Rather than trying a bunch of small grids like I did above until I get a good result, I decided to fit a massive grid and take advantage of the fact the xgboost packages allow for parallel processing. I have included the grid I used below, but have not ran it in this notebook to save time when knitting to a pdf (see the commented out function). The model that was selected from the grid search is below. A link to xgboost documentation is here: https://xgboost.readthedocs.io/en/stable/R-package/xgboost_introduction.html

Despite the massive grid and the featured engineering predictors, there was not a big upgrade in the model RMSE. It achieved a 0.30969 RMLSE on Kaggle.

```
library(xgboost)
```

```
# Grid I used. Takes 2 hours + to run. Would turn on parallel for this
xgb_grid <- expand.grid(
  nrounds = c(100, 150, 200, 250, 300, 400),
  max_depth = c(3, 5, 7, 9),
  eta = c(0.01, 0.025, 0.05, 0.075),
  gamma = c(0, 1, 5),
  colsample_bytree = c(0.5, 0.7, 0.9),
  min_child_weight = c(1, 3, 5),
  subsample = c(0.5, 0.7, 0.9)
)

# Takes way to long to run this for the purpose of this report
# xgb_model_search <- train(
#   log_salary ~ toi_takeaway_int + cap_pts_int + toi + pts + age_pts_int +
#     pim + age + giveaway + injure + pts_per_gp + hits + pm + gp +
#     goal + pts_per_min + goals_per_min + cap_toi_int + cap_ratio,
#   data = dtrain,
#   method = "xgbTree",
#   trControl = control,
#   tuneGrid = xgb_big,
#   metric = "RMSE",
#   maximize = FALSE,
#   verbose = TRUE
# )


# This was the model that was chosen.
xgb_model <- xgboost::xgboost(
    data = xgboost::xgb.DMatrix(data.matrix(dtrain[, c(
      "toi_takeaway_int", "cap_pts_int", "toi", "pts", "age_pts_int", "pim", "age", "giveaway",
      "injure", "pts_per_gp", "hits", "pm", "gp", "goal", "pts_per_min",
      "goals_per_min", "cap_toi_int", "cap_ratio"
    )]), label = dtrain$log_salary),
    nrounds = 200, max_depth = 3, eta = 0.05,gamma = 0,
    colsample_bytree = 0.5,min_child_weight = 1, subsample = 0.5,
    objective = "reg:squarederror",
    verbose = 0
  )
```

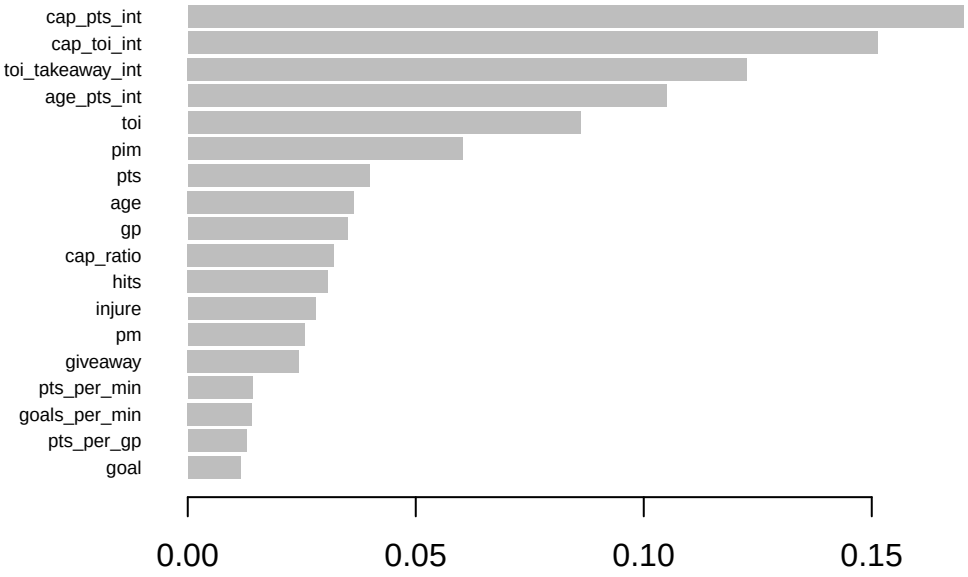
And it appears the xgb model likes the cap engineered variables as a predictor for salary.

```
library(xgboost)
```

```
##
## Attaching package: 'xgboost'
```

```
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
importance_matrix <- xgb.importance(model=xgb_model)
xgb.plot.importance(importance_matrix)
```



Final Model

All along the way I had been testing what happens when I take the average of my individual model predictions. Usually I would get a better test score than any of the individual models. The four models above were my 4 best, and I when I did the average of all 4 of them I got my best test score of 0.29735 on Kaggle. This is the final model that you see in my model source file. Below I have extracted the best model from the grid searches I conducted above. This is what is in my source file following the desired output style with the libraries.

```
# My first gbm I did. This is submission 1 on Kaggle.
  gbm1 <- gbm::gbm(
    formula = log_salary ~ toi_takeaway_int + cap_pts_int + toi + pts + age_pts_int +
      pim + age + giveaway + injure + pts_per_gp + hits + pm + gp +
      pos_general + nat_group,
    data = dtrain,
    distribution = "gaussian",
    n.trees = 600, interaction.depth = 5,
    shrinkage = 0.01, n.minobsinnode = 10,
    verbose = FALSE
  )
  # My gbm with slightly different training set. I make some room for interactions. This is submission
  gbm2 <- gbm::gbm(
    formula = log_salary ~ cap_pts_int + age_pts_int + toi_takeaway_int + pts_per_gp +
      toi + pim + age + giveaway + injure + hits + pm + gp +
      goal + pts:toi + pts:pos_general + toi:pos_general + goal:pos_general,
    data = dtrain,
    distribution = "gaussian",n.trees = 550, interaction.depth = 6,
```

```
    shrinkage = 0.01,n.minobsinnode = 3,
    verbose = FALSE
  )

  # This is using some other featured engineered variables
   gbm3 <- gbm::gbm(
    formula = log_salary ~ toi_takeaway_int + cap_pts_int+ toi + pts + age_pts_int +
      pim + age +giveaway +injure + pts_per_gp + hits +pm + gp +
      goal + pts_per_min +goals_per_min + cap_toi_int + cap_ratio,
    data = dtrain,
    distribution = "gaussian", n.trees = 400, interaction.depth = 7,
    shrinkage = 0.015, n.minobsinnode = 5,
    verbose = FALSE
  )
  # This was my xgb boost on the featured engineered set
  xgb <- xgboost::xgboost(
    data = xgboost::xgb.DMatrix(data.matrix(dtrain[, c(
      "toi_takeaway_int", "cap_pts_int", "toi", "pts", "age_pts_int", "pim", "age", "giveaway",
      "injure", "pts_per_gp", "hits", "pm", "gp", "goal", "pts_per_min",
      "goals_per_min", "cap_toi_int", "cap_ratio"
    )]), label = dtrain$log_salary),
    nrounds = 200, max_depth = 3, eta = 0.05,gamma = 0,
    colsample_bytree = 0.5,min_child_weight = 1, subsample = 0.5,
    objective = "reg:squarederror",
    verbose = 0
  )
```

And then I take the predictions of all the individual models to get the final prediction:

$$(Pred_{gbm1} + Pred_{gbm2} + Pred_{gbm3} + Pred_{xgb})/4 = Final_{pred}$$

Evaluation

```
rm(list=ls())
# first set working directory to the data location
load("final.Rdata")
# final.Rdata contains dtrain and dtest
# load user-defined FinalModel function
source("21154733.R")
# call the function
tmp <- system.time(res <- FinalModel(dtrain, dtest))
# time used, this is the "Running_time" you report in yMy R sMyce file
tmp[3]
```

```
## elapsed
##   0.606
```

```
dim(res)
```

```
## [1] 400   2
```

```r
head(res)
```

```
##   Id    salary
## 1  1  636077.1
## 2  2 1130045.1
## 3  3  630754.8
## 4  4 4380351.1
## 5  5 5984602.2
## 6  6  805159.6
```

Thank you for reading my report!