NBA Player Projections: Predicting Win Share Value

Daniel Jackson

March 3rd, 2024

Libraries Used

```
packages = c("corrplot", "dplyr", "glmnet", "caret", "pls","tree", "ipred",
             "randomForest","gbm","readxl", "knitr")
lapply(packages, library, character.only = TRUE)
```

Introduction to Analysis

Our goal of this analysis is to predict NBA player's win share values for the 2023-2024 NBA season. We will be using three NBA seasons during our analysis: 2020-2021, 2021-2022 and 2022-2023. The data for the NBA seasons data can be found using the Basketball Reference URLs below. We only will be analyzing players that played at least 1,000 minutes in each season.
In Part 1 of our analysis, we will read in and clean up our data.
In Part 2 of our analysis, after the data is cleaned up, we will fit various regression models using training data. We will then use those trained models to make predictions on our held-out test data that we created. We will then take the mean squared error rate of those predictions compared to each player's actual win share value to see to see how accurate our predictions were.
In Part 3 of our analysis, after we have chosen a model, we will predict each player's win share value for the 2023-2024 season using their advanced stats from the 2022-2023 season. We will then convert each player's win share value into a dollar value.
In Part 4 of our analysis, acting as the Analytics Director for the Orlando Magic, we will make the same predictions for our own players and NBA free agents to help make off-season decisions on how we will construct our roster going into the 2023-2024 season. We will have four major scenarios to analyze.

Part 1

Read In and Clean Data

Let's start by reading in our data from our CSV files. The data for the NBA seasons data can be found at the URLs below. We converted each seasons advanced data into a CSV file and read those files into R to analyze them.

2020-2021 NBA Season URL:

https://www.basketball-reference.com/leagues/NBA_2021_advanced.html#advanced_stats

2021-2022 NBA Season URL:

https://www.basketball-reference.com/leagues/NBA_2022_advanced.html

2022-2023 NBA Season URL:

https://www.basketball-reference.com/leagues/NBA_2023_advanced.html

Once this data is converted to a CSV file and a file location is created, we can read in each file.

```
# Looking for regular season data from three seasons: 2020-2021, 2021-2022, 2022-2023
# Created three separate CSV files by exporting data from Basketball Reference.
# We will be using advanced NBA metrics to predict win share for each player.

# Link to 2020-2021 season from Basketball Reference:
# https://www.basketball-reference.com/leagues/NBA_2021_advanced.html#advanced_stats
twenty_one_nba_df = read.csv("2021_NBA_Data.csv")
# Remove Player.additional and Rk (rank) in data frame. No use to us in this
# analysis
twenty_one_nba_df = twenty_one_nba_df[, -which(names(twenty_one_nba_df) ==
                                                 "Player.additional")]
twenty_one_nba_df = twenty_one_nba_df[, -which(names(twenty_one_nba_df) ==
                                                 "Rk")]
# Two empty columns in data frame: X and X.1
twenty_one_nba_df = twenty_one_nba_df[, -which(names(twenty_one_nba_df) ==
                                                 "X")]
twenty_one_nba_df = twenty_one_nba_df[, -which(names(twenty_one_nba_df) ==
                                                 "X.1")]

# Let's remove Win Share per 48 minutes variable, as we will only be focusing
# on win shares response variable
twenty_one_nba_df = twenty_one_nba_df[, -which(names(twenty_one_nba_df) ==
                                                 "WS.48")]

dim(twenty_one_nba_df)
```

```
## [1] 705  25
```

```
# 705 observations
# 25 variables

# Link to 2021-2022 season from Basketball Reference:
# https://www.basketball-reference.com/leagues/NBA_2022_advanced.html
twenty_two_nba_df = read.csv("2022_NBA_Data.csv")
# Remove Player.additional and Rk (rank) in data frame. No use to us in this
# analysis
twenty_two_nba_df = twenty_two_nba_df[, -which(names(twenty_two_nba_df) ==
                                                 "Player.additional")]
twenty_two_nba_df = twenty_two_nba_df[, -which(names(twenty_two_nba_df) ==
                                                 "Rk")]
# Two empty columns in data frame: X and X.1
twenty_two_nba_df = twenty_two_nba_df[, -which(names(twenty_two_nba_df) ==
                                                 "X")]
twenty_two_nba_df = twenty_two_nba_df[, -which(names(twenty_two_nba_df) ==
                                                 "X.1")]

# Let's remove Win Share per 48 minutes variable, as we will only be focusing
```

```r
# on win shares response variable
twenty_two_nba_df = twenty_two_nba_df[, -which(names(twenty_two_nba_df) ==
                                                 "WS.48")]
dim(twenty_two_nba_df)
```

```
## [1] 812  25
```

```r
# 812 observations
# 25 variables

# Link to 2022-2023 season from Basketball Reference:
# https://www.basketball-reference.com/leagues/NBA_2023_advanced.html
twenty_three_nba_df = read.csv("2023_NBA_Data.csv")
# Remove Player.additional and Rk (rank) in data frame. No use to us in this
# analysis
twenty_three_nba_df = twenty_three_nba_df[, -which(names(twenty_three_nba_df) ==
                                                 "Player.additional")]
twenty_three_nba_df = twenty_three_nba_df[, -which(names(twenty_three_nba_df) ==
                                                   "Rk")]
# Two empty columns in dataframe: X and X.1
twenty_three_nba_df = twenty_three_nba_df[, -which(names(twenty_three_nba_df) ==
                                                   "X")]
twenty_three_nba_df = twenty_three_nba_df[, -which(names(twenty_three_nba_df) ==
                                                   "X.1")]

# Let's remove Win Share per 48 minutes variable, as we will only be focusing
# on win shares response variable
twenty_three_nba_df = twenty_three_nba_df[, -which(names(twenty_three_nba_df) ==
                                                   "WS.48")]

dim(twenty_three_nba_df)
```

```
## [1] 679  25
```

```r
# 679 observations
# 26 variables

# Let's tidy up variable names to help make our coding easier
colnames(twenty_one_nba_df)
```

```
##  [1] "Player" "Pos"    "Age"    "Tm"     "G"      "MP"     "PER"    "TS."
##  [9] "X3PAr"  "FTr"    "ORB."   "DRB."   "TRB."   "AST."   "STL."   "BLK."
## [17] "TOV."   "USG."   "OWS"    "DWS"    "WS"     "OBPM"   "DBPM"   "BPM"
## [25] "VORP"
```

```r
# Let's make all variables lower case in each data frame.
colnames(twenty_one_nba_df) = tolower(colnames(twenty_one_nba_df))
colnames(twenty_two_nba_df) = tolower(colnames(twenty_two_nba_df))
colnames(twenty_three_nba_df) = tolower(colnames(twenty_three_nba_df))

# Let's make all periods "." into "_per"
```

```r
# 2020-2021
predictor_names = colnames(twenty_one_nba_df)
new_names = gsub("\\.", "_pct", predictor_names)
colnames(twenty_one_nba_df) = new_names

# 2021-2022
predictor_names = colnames(twenty_two_nba_df)
new_names = gsub("\\.", "_pct", predictor_names)
colnames(twenty_two_nba_df) = new_names

# 2022-2023
predictor_names = colnames(twenty_three_nba_df)
new_names = gsub("\\.", "_pct", predictor_names)
colnames(twenty_three_nba_df) = new_names

# Check to see if there are any duplicate names in each data frame. Looking at
# the data, you can see that there are players that played for multiple teams.
# Those players stats are broken up by each team and their totals. We only want
# their totals for their entire year. We only want to look at players that have
# played at least 1000 minutes played. Let's filter out players that played at
# least 1000 minutes. Then we can check if there are still duplicate names in
# each data frame. Once we do that, we can then merge the three data frames
# together.

# 2020-2021
twenty_one_nba_df = twenty_one_nba_df[twenty_one_nba_df$mp >= 1000, ]
# Observations went from 705 to 260
# Check for duplicate player names
# Create a table of player frequencies
player_frequencies = table(twenty_one_nba_df[["player"]])
# Extract player names that have counts greater than 1 (indicating duplicates)
duplicate_players = names(player_frequencies[player_frequencies > 1])
# Print or view the duplicate player names
print(duplicate_players)
```

```
## [1] "Caris LeVert"   "Daniel Theis"   "Delon Wright"   "Gary Trent Jr."
## [5] "James Harden"   "Jarrett Allen"  "Kelly Olynyk"   "Nikola Vučević"
## [9] "Norman Powell"
```

```r
# We have 9 duplicate players in this data frame.
# These 9 players played on multiple teams in the 2020-2021 season and also
# played over 1000 minutes with at least one of those teams.
# Subset data to only keep duplicate players season totals
twenty_one_nba_df = twenty_one_nba_df[!(twenty_one_nba_df[["player"]] %in%
                                          duplicate_players &
                                          twenty_one_nba_df[["tm"]] != "TOT"), ]
# 251 total observations


# 2021-2022
twenty_two_nba_df = twenty_two_nba_df[twenty_two_nba_df$mp >= 1000, ]
# Observations went from 812 to 291
# Check for duplicate player names
```

```r
player_frequencies = table(twenty_two_nba_df[["player"]])
# Extract player names that have counts greater than 1 (indicating duplicates)
duplicate_players = names(player_frequencies[player_frequencies > 1])
# Print or view the duplicate player names
print(duplicate_players)
```

```
##  [1] "Buddy Hield"              "Caris LeVert"
##  [3] "CJ McCollum"              "Dennis Schröder"
##  [5] "Derrick White"            "Domantas Sabonis"
##  [7] "James Harden"             "Josh Hart"
##  [9] "Josh Richardson"          "Justin Holiday"
## [11] "Kristaps Porziņģis"       "Montrezl Harrell"
## [13] "Nickeil Alexander-Walker" "Norman Powell"
## [15] "Robert Covington"         "Seth Curry"
## [17] "Spencer Dinwiddie"        "Torrey Craig"
## [19] "Tyrese Haliburton"
```

```r
# 19 duplicate players in 2021-2022 data.
# Subset data to only keep duplicate players season totals
twenty_two_nba_df = twenty_two_nba_df[!(twenty_two_nba_df[["player"]] %in%
                                        duplicate_players &
                                        twenty_two_nba_df[["tm"]] != "TOT"), ]
# 272 total observations


# 2022-2023
twenty_three_nba_df = twenty_three_nba_df[twenty_three_nba_df$mp >= 1000, ]
# Observations went from 679 to 281
# Check for duplicate player names
player_frequencies = table(twenty_three_nba_df[["player"]])
# Extract player names that have counts greater than 1 (indicating duplicates)
duplicate_players = names(player_frequencies[player_frequencies > 1])
# Print or view the duplicate player names
print(duplicate_players)
```

```
##  [1] "D'Angelo Russell"    "Dorian Finney-Smith" "Eric Gordon"
##  [4] "Jakob Poeltl"        "Jalen McDaniels"     "Jarred Vanderbilt"
##  [7] "Josh Hart"           "Kevin Durant"        "Kyrie Irving"
## [10] "Malik Beasley"       "Mason Plumlee"       "Mikal Bridges"
## [13] "Mike Conley"         "Patrick Beverley"    "Reggie Jackson"
## [16] "Russell Westbrook"   "Saddiq Bey"          "Spencer Dinwiddie"
```

```r
# 18 duplicate players in 2022-2023 data
# Subset data to only keep duplicate players season totals
twenty_three_nba_df = twenty_three_nba_df[!(twenty_three_nba_df[["player"]] %in%
                                        duplicate_players &
                                        twenty_three_nba_df[["tm"]] != "TOT"), ]
# 263 total observations


# Now that we removed duplicate players. We can now combine our three data sets
# into one data set using rbind function.
```

```r
# Before we do so, let's add a season column in each data frame for our exploratory
# analysis
twenty_one_nba_df$season = "2020-2021"
twenty_two_nba_df$season = "2021-2022"
twenty_three_nba_df$season = "2022-2023"
nba_df = rbind(twenty_one_nba_df, twenty_two_nba_df, twenty_three_nba_df)

# Now that we have our cleaned up data, we can now start to explore data!
dim(nba_df)
```

```
## [1] 786  26
```

```r
# 26 variables
# 786 observations

# Check for missing values in data set
colSums(is.na(nba_df))
```

```
##  player     pos     age      tm       g      mp     per  ts_pct   x3par     ftr
##       0       0       0       0       0       0       0       0       0       0
## orb_pct drb_pct trb_pct ast_pct stl_pct blk_pct tov_pct usg_pct     ows     dws
##       0       0       0       0       0       0       0       0       0       0
##      ws    obpm    dbpm     bpm    vorp  season
##       0       0       0       0       0       0
```

```r
# No missing values. Awesome!
```

Exploratory Analysis

After reading in the three NBA seasons into R, cleaning them up, and then combining them together, we see that we have 786 observations and 26 variables in our data set. That means that we have 25 predictors and 1 response variable. Those 25 statistics will be used as our predictors to predict the win share response variable for each player. Below you fill find each predictor in our data set and a brief description of what that statistic represents. Please refer to this list when an abbreviated predictor is used in discussion or in any of the models.

player: Player name.

pos: Position.

age: Age.

tm: Team.

g: Games Played.

mp: Minutes Played.

per: Player Efficiency Rating.

ts_pct: True Shooting Percentage.

x3par: 3-Point Attempt Rate.

ftr: Free Throw Attempt Rate.

orb_pct: Offensive Rebound Percentage.

drb_pct: Defensive Rebound Percentage.

trb_pct: Total Rebound Percentage.

ast_pct: Assist Percentage.

stl_pct: Steal Percentage.

blk_pct: Block Percentage.

tov_pct: Turnover Percentage.

usg_pct: Usage Percentage.

ows: Offensive Win Shares.

dws: Defensive Win Shares.

obpm: Offensive Box Plus/Minus.

dbpm: Defensive Box Plus/Minus.

bpm: Total Box Plus/Minus.

vorp: Wins Over Replacement Player.

season: NBA Season Year Span.

ws: Win Shares (This will be our response variable).

What is a win share?

A "Win Share" reflects the offensive (points added) and defensive (points prevented) contributions of a player that led to one season win while on the floor. Offensive win share and defensive win share are calculated separately. You then add the two win shares together to get total win share for each player. A win share is worth one-third of a team win in the NBA.
During the three year stretch that we are analyzing, let us look at top 10 players in win shares.

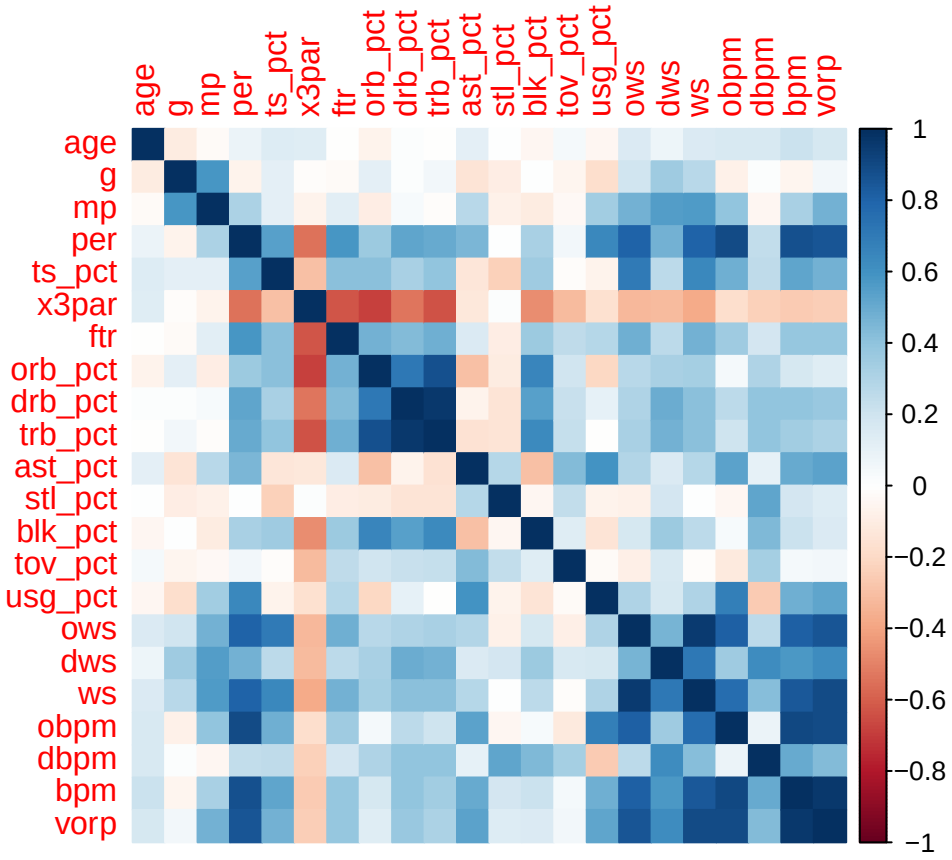
```
top_ten_players_ws = nba_df[order(-nba_df$ws), ][1:10, c("season", "player", "ws")]
kable(top_ten_players_ws)
```

	season	player	ws
345	2020-2021	Nikola Jokić	15.6
392	2021-2022	Nikola Jokić	15.2
319	2022-2023	Nikola Jokić	14.9
16	2021-2022	Giannis Antetokounmpo	12.9
5501	2022-2023	Domantas Sabonis	12.6
1011	2022-2023	Jimmy Butler	12.3
1851	2022-2023	Joel Embiid	12.3
2071	2021-2022	Joel Embiid	12.0
2581	2021-2022	Rudy Gobert	11.7
2101	2022-2023	Shai Gilgeous-Alexander	11.4

We see that Nikola Jokić has led the league in win shares the past three years. The only other player to show up in the top ten twice over that stretch is Joel Embiid in the 2021 and 2022 season. Giannis Antetokounmpo is the next player behind Jokić. It makes sense that these three are in the top ten as Jokić won back to back MVPs (Most Valuable Player awards) in the 2021 and 2020 and 2021 seasons with Embiid winning MVP in the 2022 Season. And Antetokounmpo won back to back MVPs in 2018 and 2019.

Let us look at a correlation plot of all of our predictors.

```
# Look at correlation plot of all predictors
corrplot::corrplot(cor(nba_df[sapply(nba_df, is.numeric)],
                       use = "pairwise.complete.obs"), method = "color")
```



From our correlation plot, we see that only one predictor has a negative correlation with win shares. That is the x3par variable. Let's go ahead and remove that predictor.

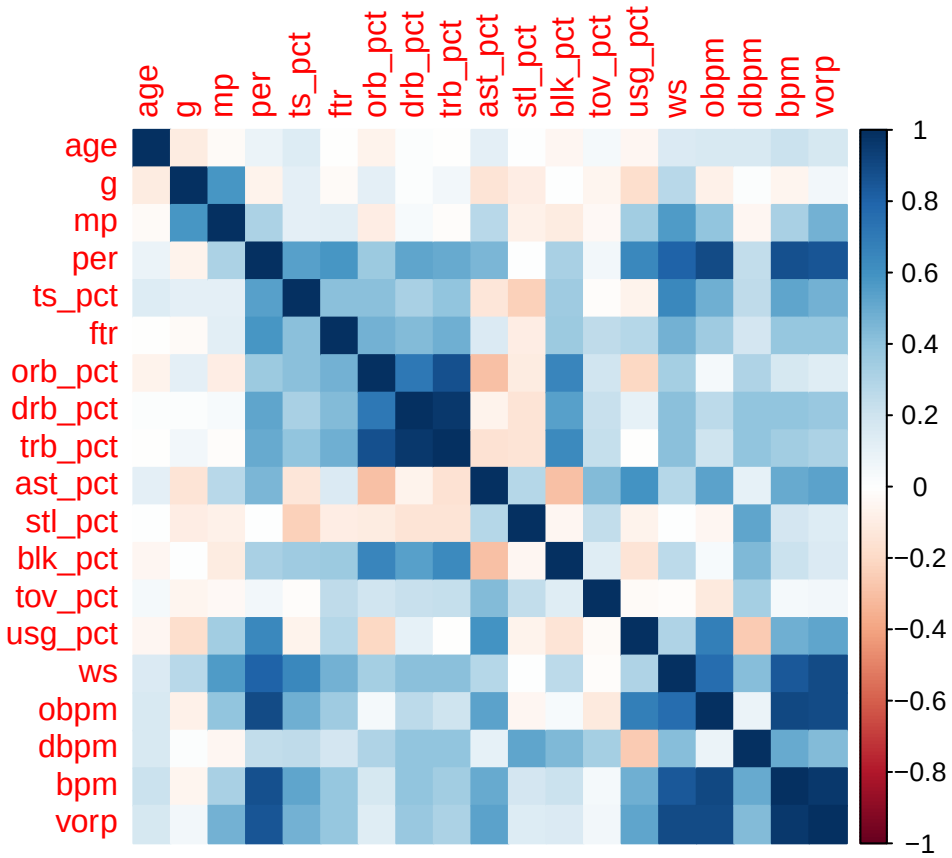
```
nba_df = nba_df[, -which(names(nba_df) == "x3par")]
```

We also know that win shares is the sum of offensive win shares and defensive win shares. For our analysis, we will focus solely on the total win shares as our response variable. We see the multicollinearity between the three variables in our correlation plot. Therefore, we will remove both the offensive and defensive win share variables.

```
nba_df = nba_df[, -which(names(nba_df) == "ows")]
nba_df = nba_df[, -which(names(nba_df) == "dws")]
```

Let us revisit correlation plot again.

```
corrplot::corrplot(cor(nba_df[sapply(nba_df, is.numeric)],
                       use = "pairwise.complete.obs"), method = "color")
```



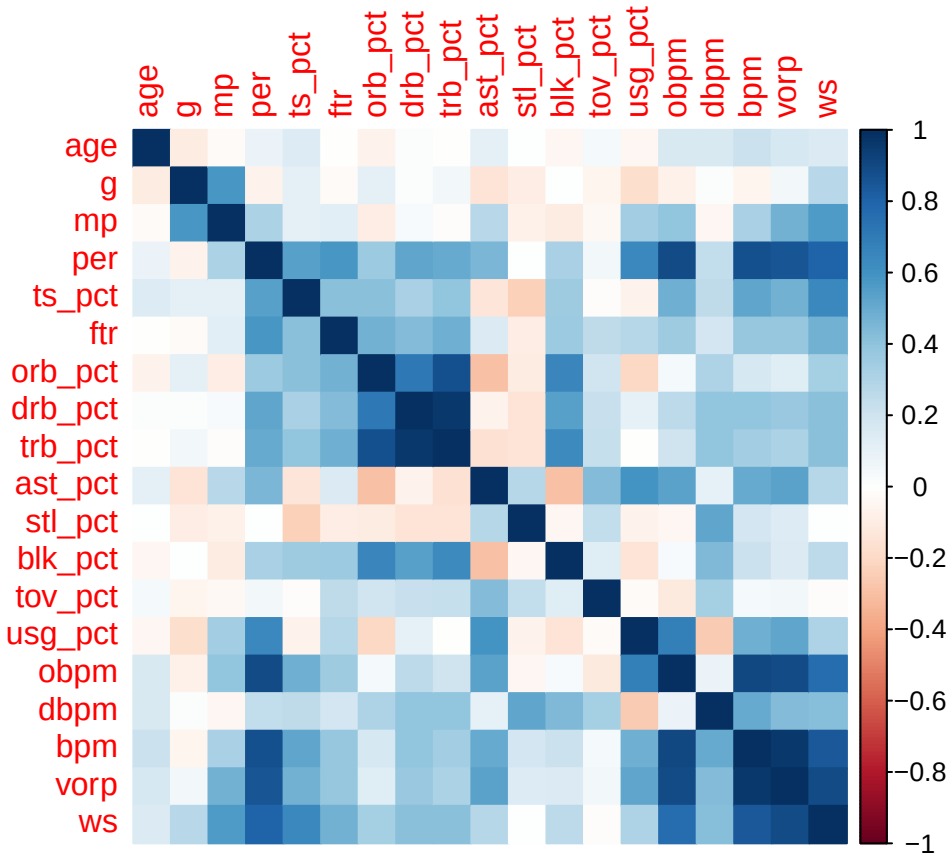
Let us now create a new data frame called ws_df (win shares data frame) which removes the descriptive variables in our nba data set: player, position, season, tm.

```
ws_df = nba_df[, -which(names(nba_df) == "player")]
ws_df = ws_df[, -which(names(ws_df) == "pos")]
ws_df = ws_df[, -which(names(ws_df) == "season")]
ws_df = ws_df[, -which(names(ws_df) == "tm")]
```

Let us move ws response variable to end of data set to help with correlation plot visual using dplyr package and look at correlation plot again.

```
ws_df = ws_df %>%
  select(-"ws", everything(), "ws")

# Let's look at correlation plot of ws_df
corrplot::corrplot(cor(ws_df[sapply(ws_df, is.numeric)],
                       use = "pairwise.complete.obs"), method = "color")
```



Let us look at the correlation of each predictor in regards to ws (win share).

```
correlations = cor(ws_df)
ordered_ws_df_corr = correlations["ws", order(abs(correlations["ws",]),
                                              decreasing = TRUE)]
print(ordered_ws_df_corr)
```

```
##          ws        vorp         bpm         per        obpm      ts_pct
##  1.000000000  0.892298630  0.845270276  0.803064702  0.766251015  0.646603282
##          mp         ftr        dbpm      trb_pct      drb_pct      orb_pct
##  0.562443829  0.477695073  0.421321616  0.418633427  0.413289587  0.331502317
##     usg_pct      ast_pct           g      blk_pct         age      tov_pct
##  0.302883084  0.288977800  0.278518327  0.262283862  0.152748542 -0.011968996
##     stl_pct
##  0.008259969
```

We see that tov_pct and stl_pct are the lowest correlated predictors in regards to ws. Let us remove those predictors.

```
ws_df = ws_df[, -which(names(ws_df) == "tov_pct")]
ws_df = ws_df[, -which(names(ws_df) == "stl_pct")]
```

Looking for multicollinearity issues, we see that orb_pct and drb_pct are highly correlated with trb_pct. Which makes sense as trb_pct is the sum of orb_pct and drb_pct. Let's remove orb_pct and drb_pct.

```
ws_df = ws_df[, -which(names(ws_df) == "drb_pct")]
ws_df = ws_df[, -which(names(ws_df) == "orb_pct")]
```

We also see the same thing with bpm and obpm/dbpm. Since bmp is total box plus/minus, which is obpm plus dbpm, let us remove obpm and dbpm.

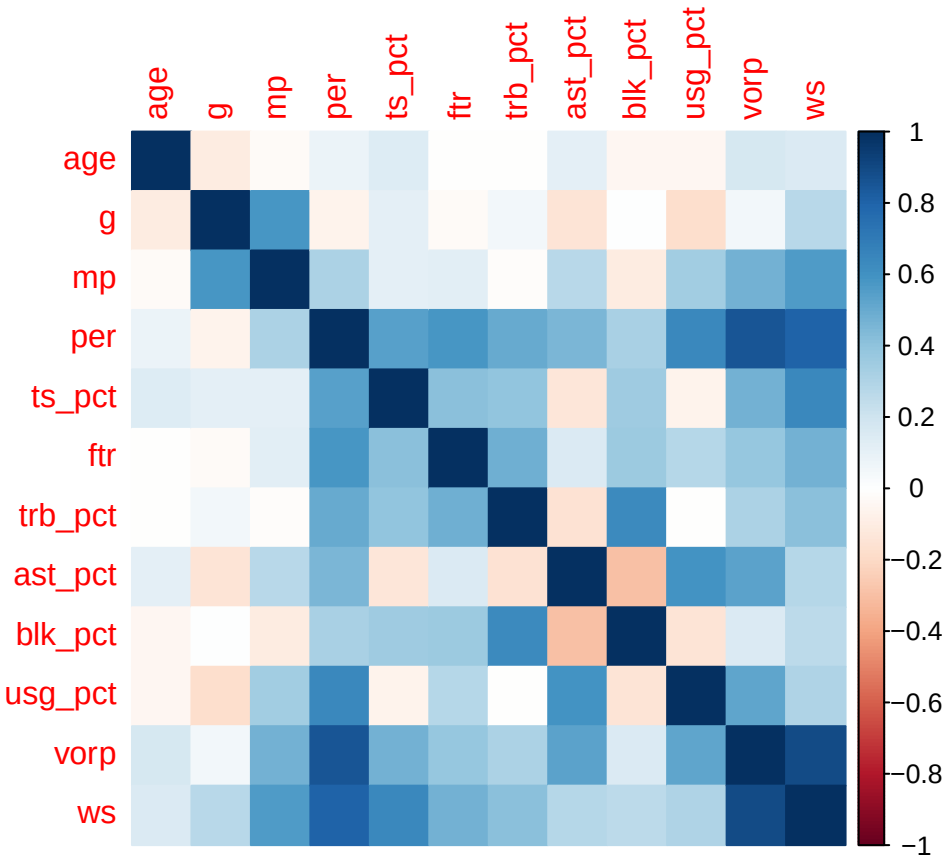
```
ws_df = ws_df[, -which(names(ws_df) == "obpm")]
ws_df = ws_df[, -which(names(ws_df) == "dbpm")]
```

We see that bpm and vorp are highly correlated as well. Between the two, vorp is slightly more correlated with ws than bpm is. To avoid multicollinearity issues let us remove bpm.

```
ws_df = ws_df[, -which(names(ws_df) == "bpm")]
```

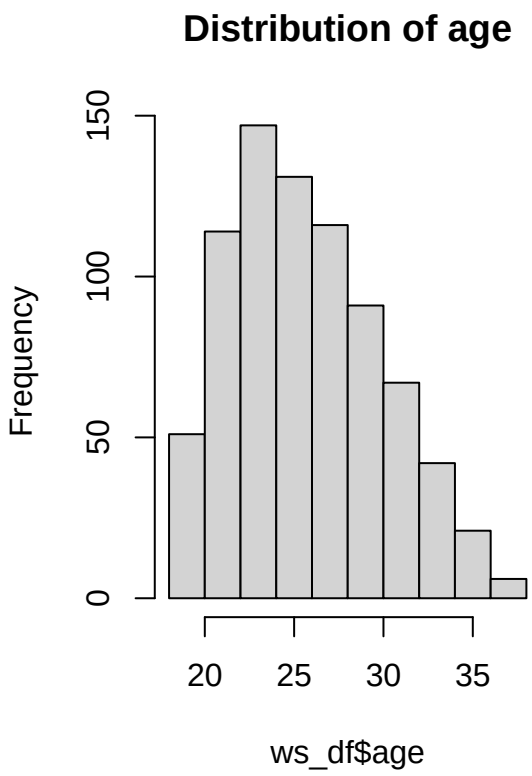
Revisit updated data frame correlation plot.

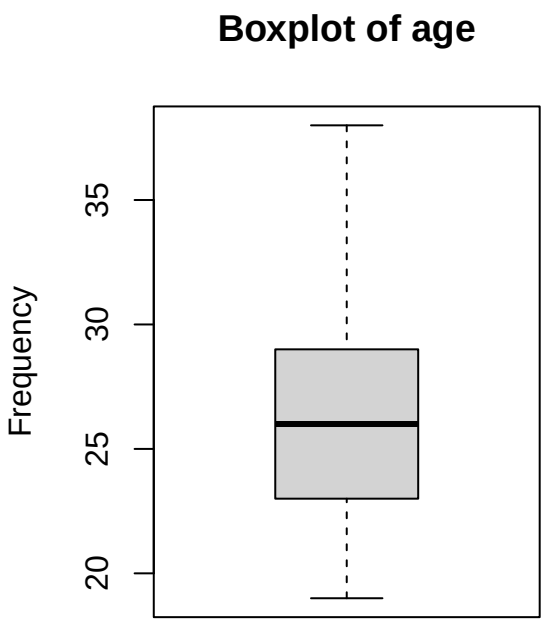
```
corrplot::corrplot(cor(ws_df[sapply(ws_df, is.numeric)],
                       use = "pairwise.complete.obs"), method = "color")
```



We now have 11 predictors to our ws response variable.
Let us now look at the distributions of each predictor and see if we need to do any transformations to help make the distributions more normal. We will try square root, squared and log transformation for each predictor.

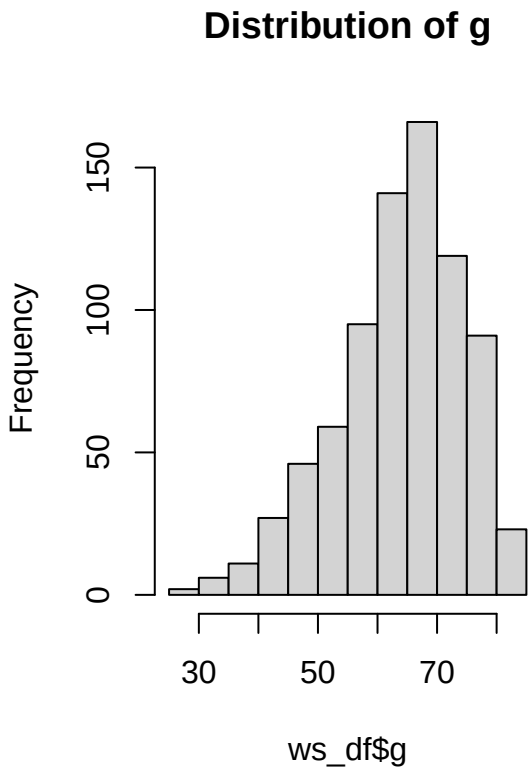
age:

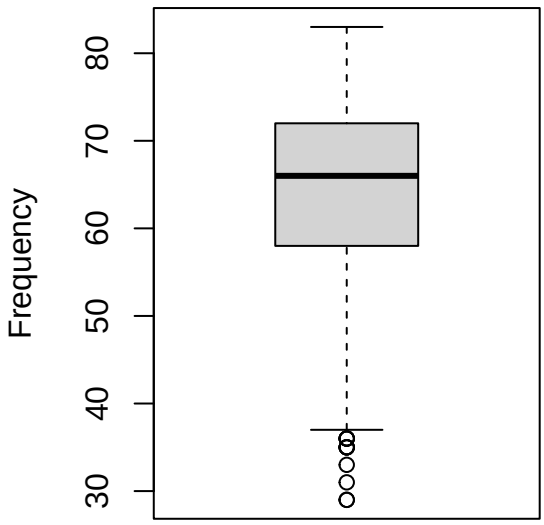




age has relatively normal distribution. It is slightly skewed to the right. The transformations did not make distribution more normal. We chose to not transform this variable.

g:

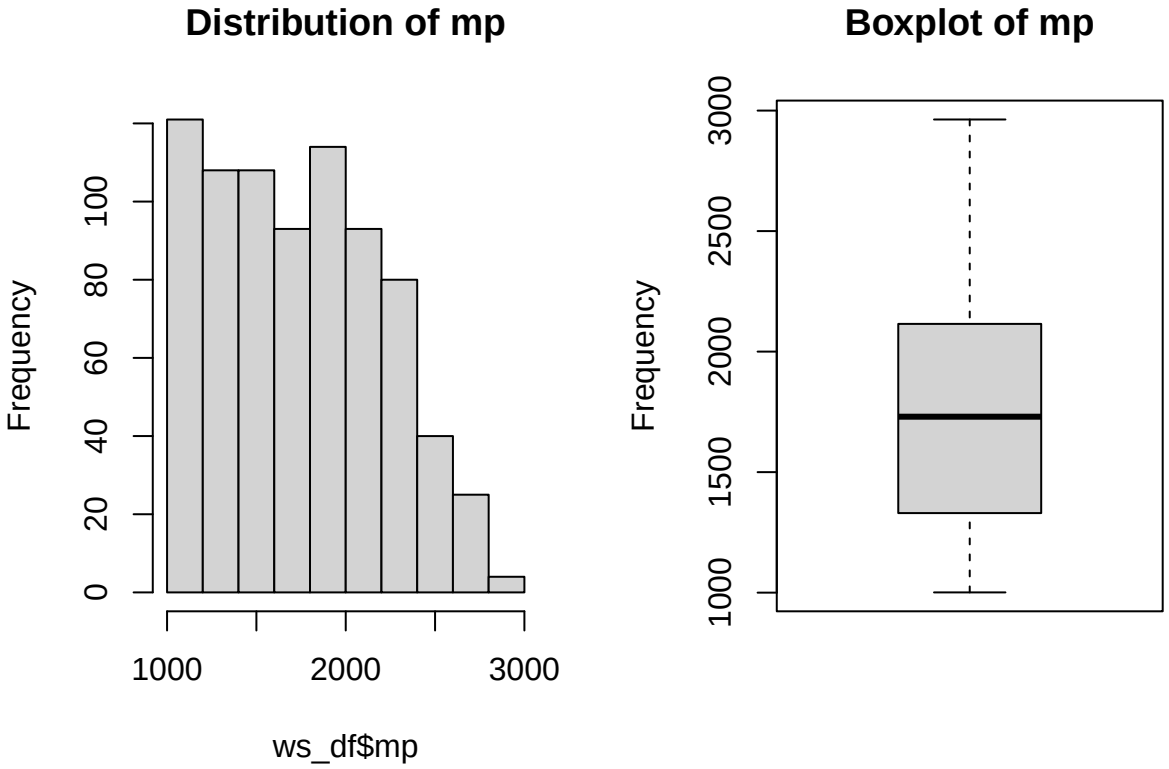




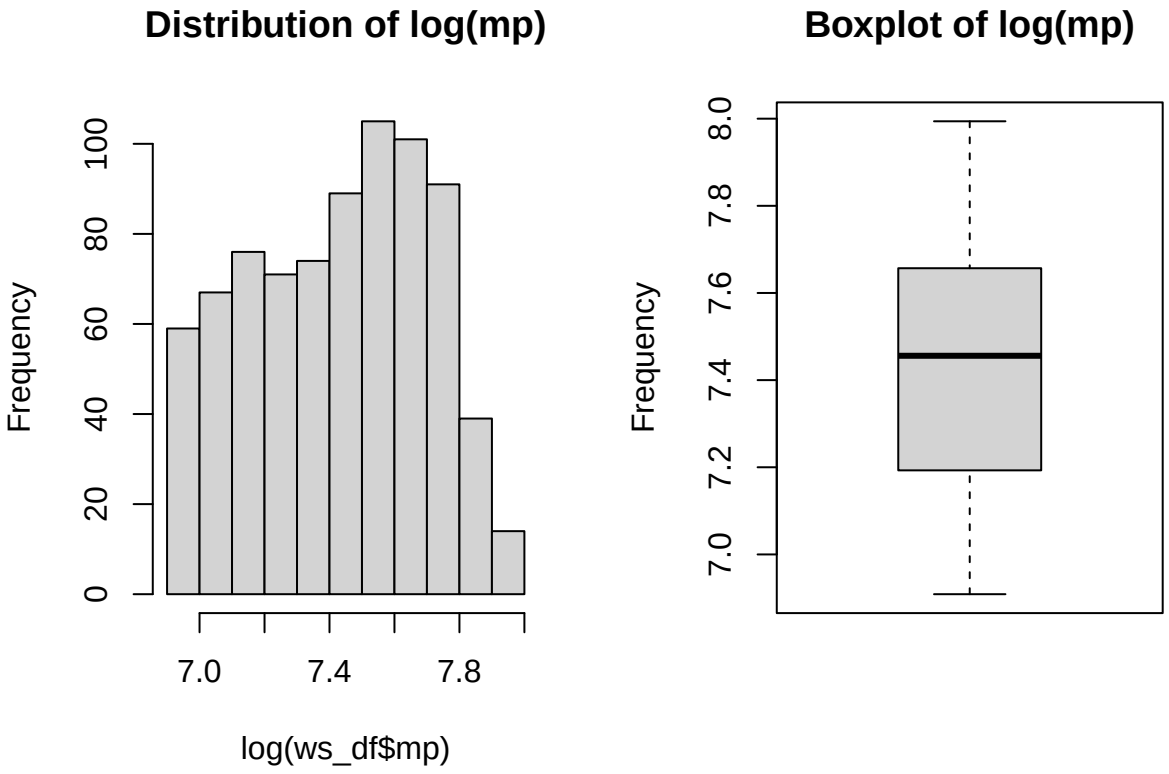
The distribution of g is skewed skightly to the left with about 5 outliers. The transformations did not make

distribution more normal. We chose to not transform this variable.

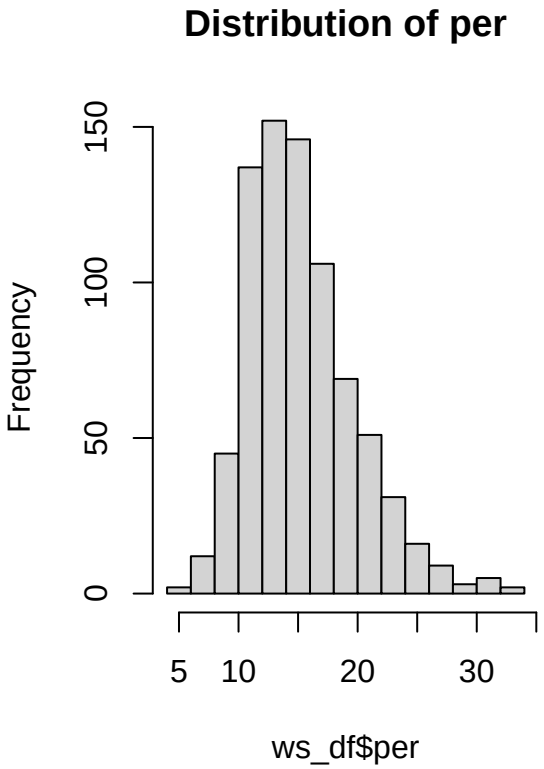
mp:

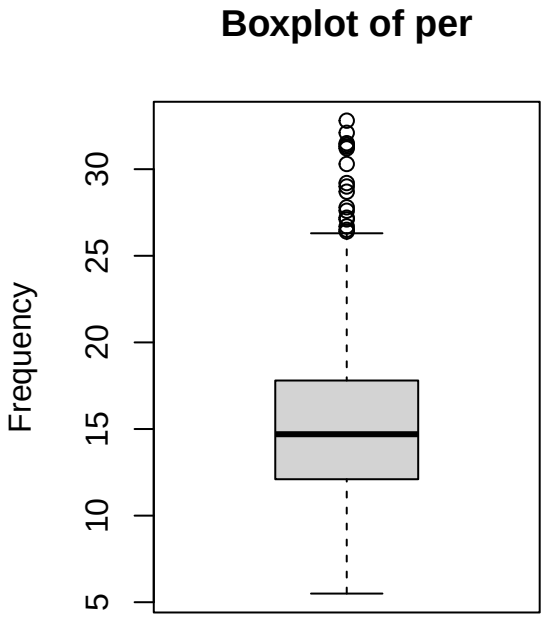


mp does not have much of a normal distribution. It is slightly right skewed. There are no outliers present. The log transformation made distribution slightly more normal.

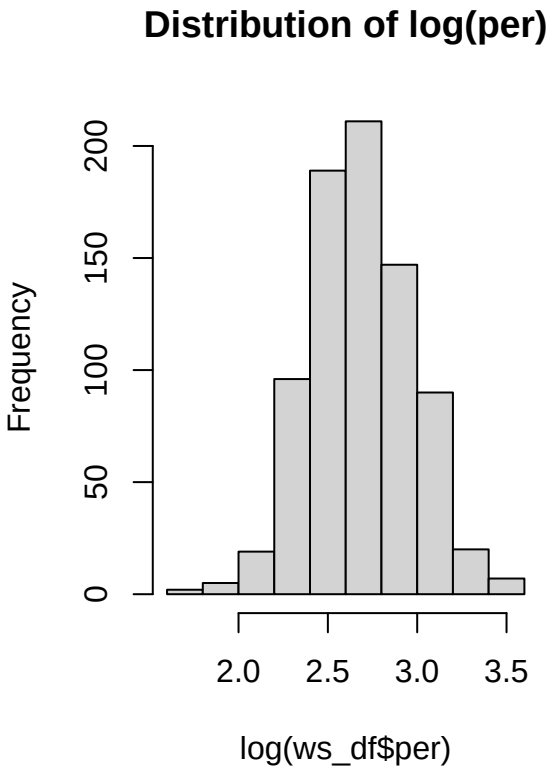


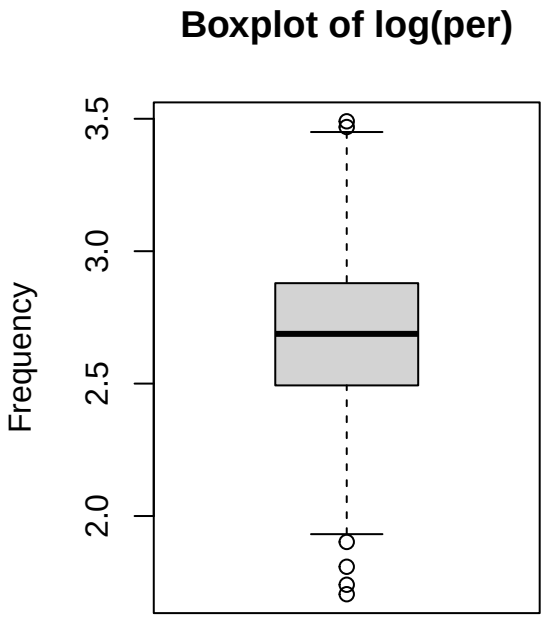
per:



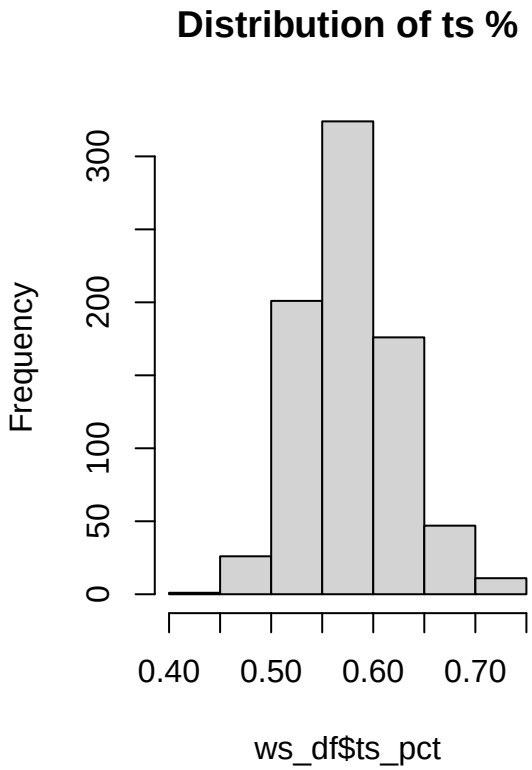


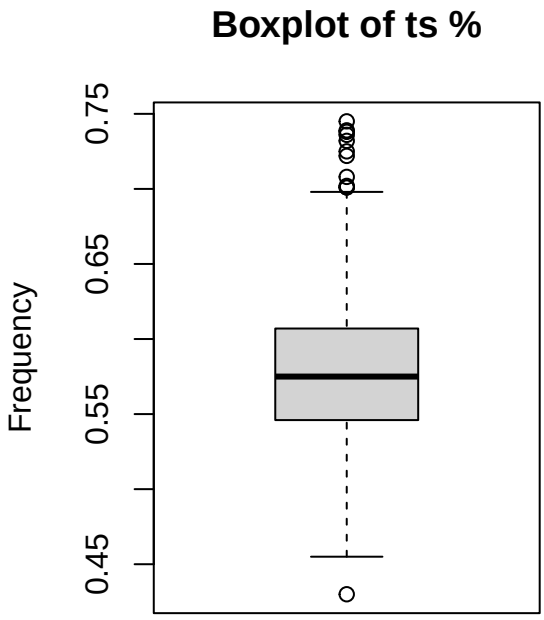
The distribution of per is slightly skewed right. There are several outliers. We found that the log transformation made distribution more normal and reduced outliers.





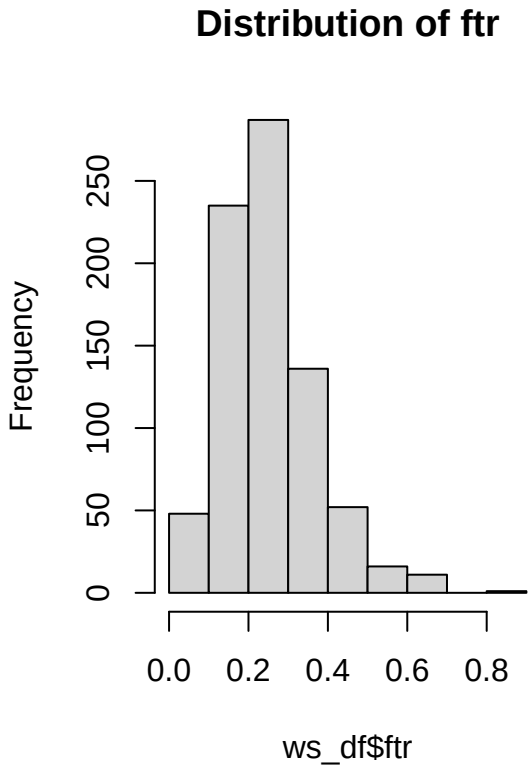
ts_pct:

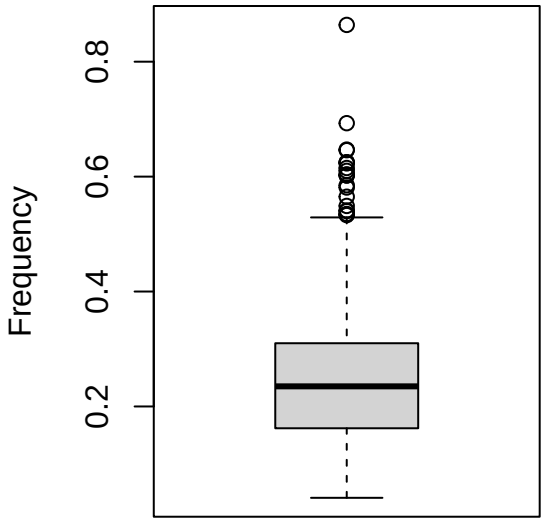




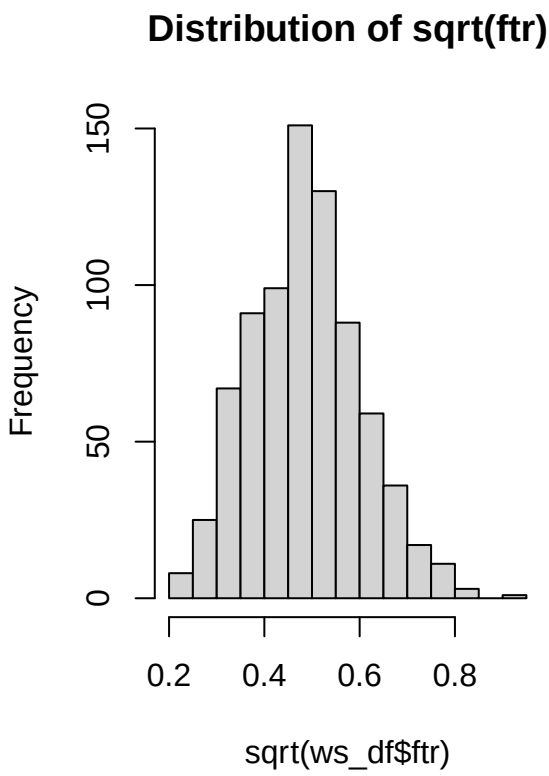
ts_pct has relatively normal distribution. There are several outliers. We chose not to transform this variable.

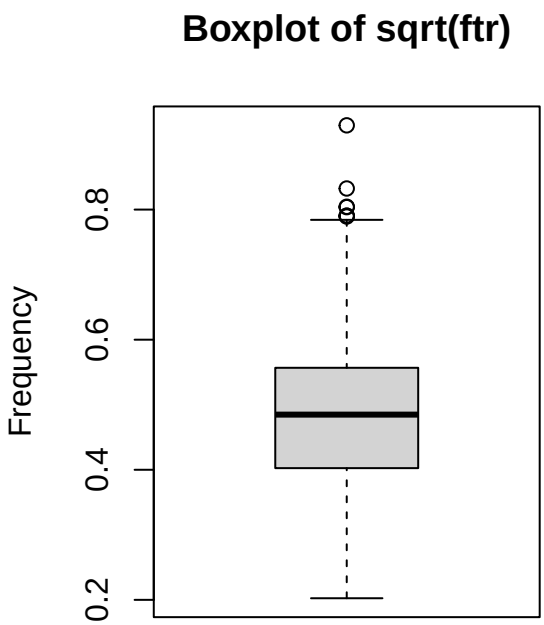
ftr:



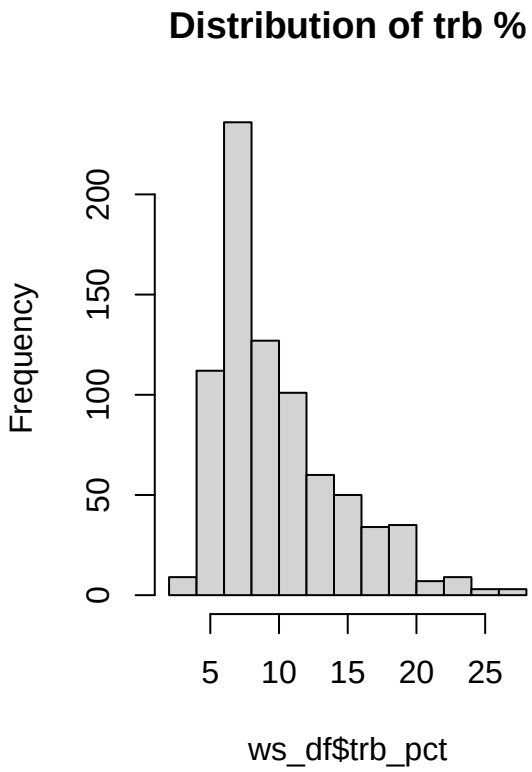


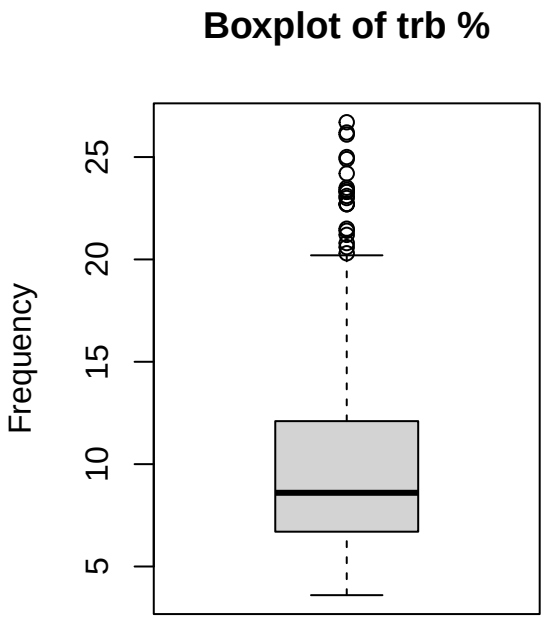
Distribution of ts_pct is skewed right with several outliers. We found that the square root transformation made the distribution more normal and reduced number of outliers.



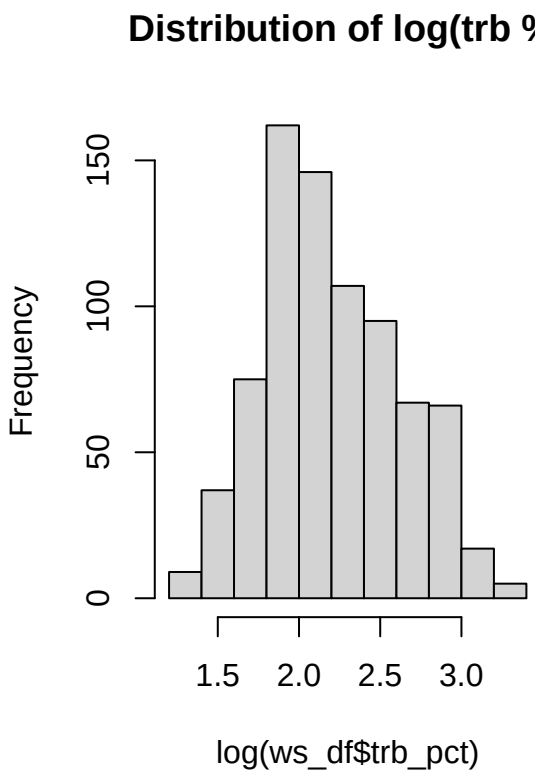


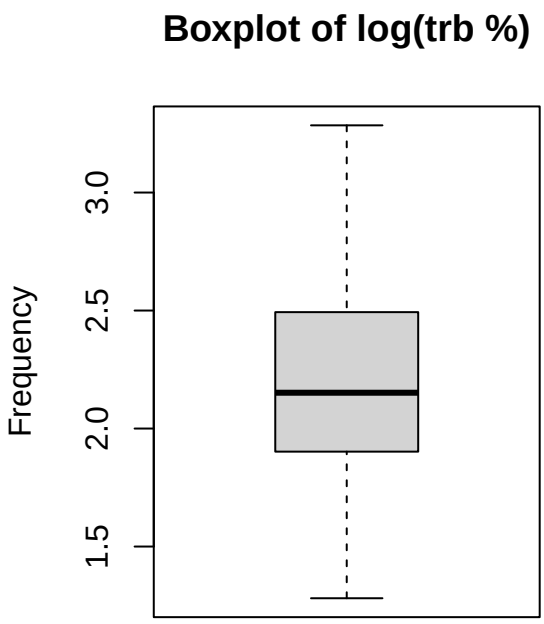
trb_pct:



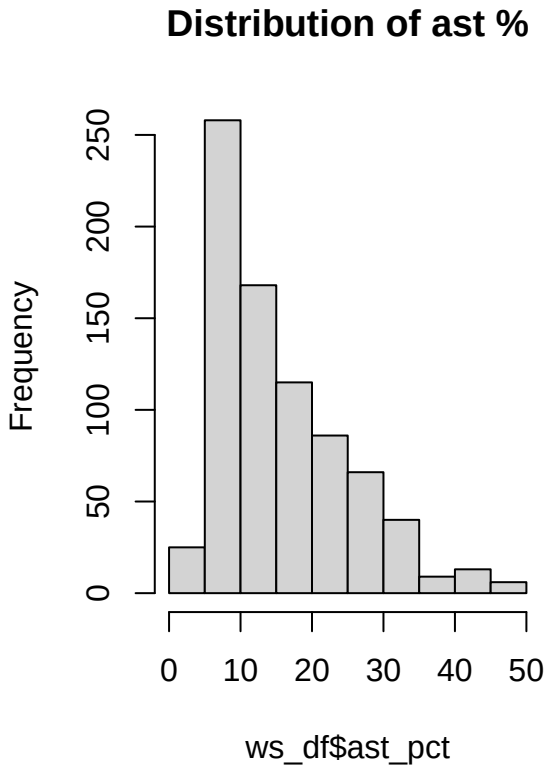


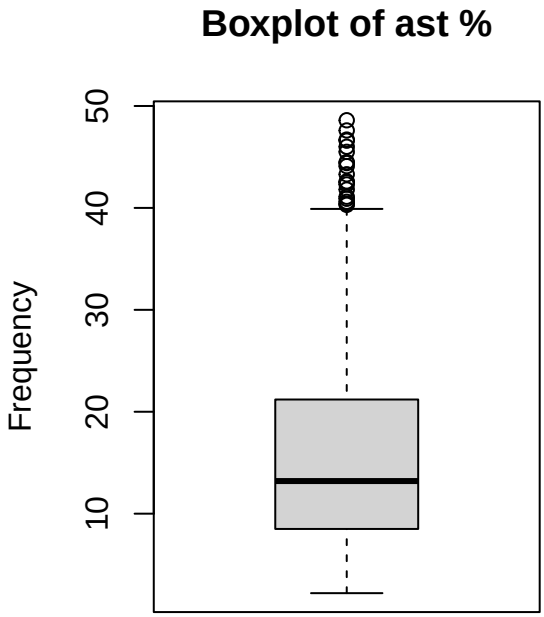
The distribution of trb_pct is skewed right with a ton of outliers. We found that a log transformation made distribution more normal and produced zero outliers.



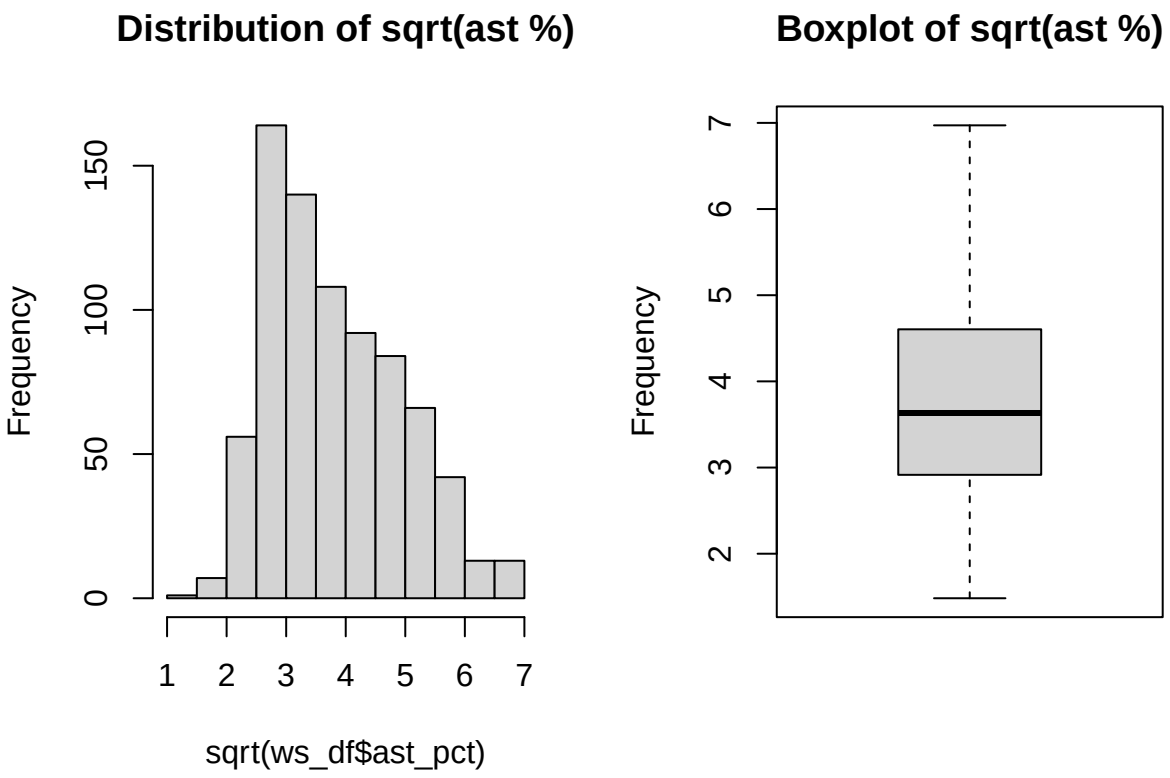


ast_pct:

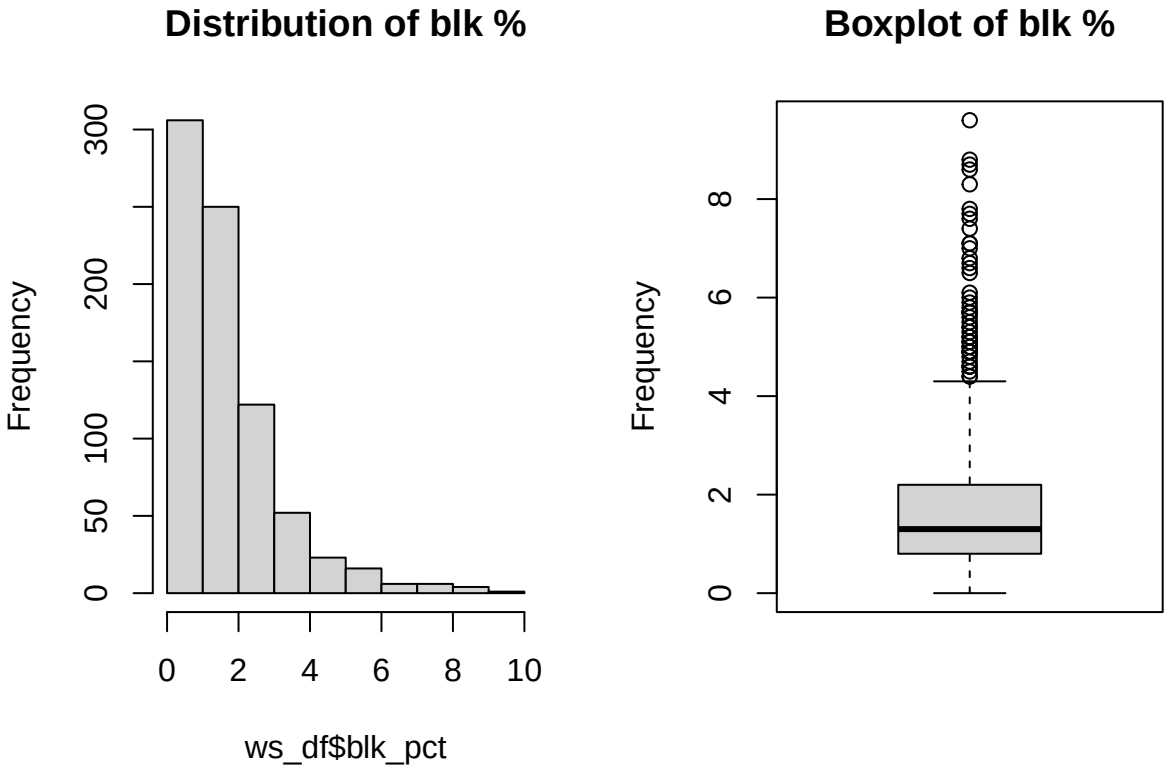




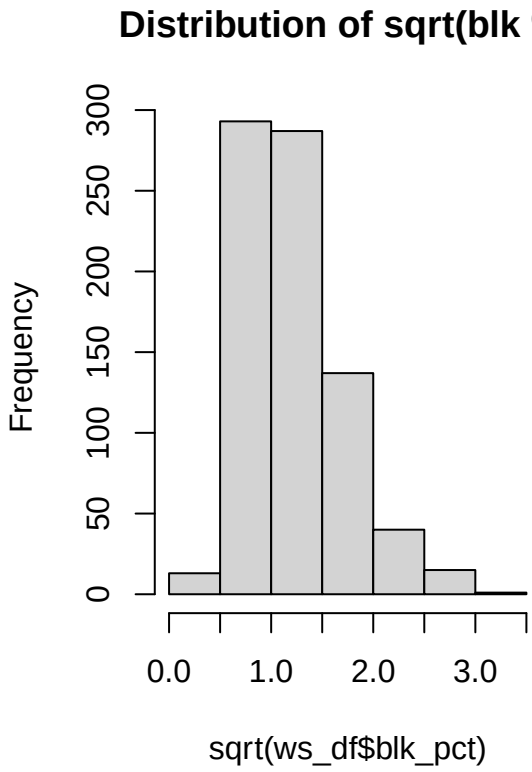
The distribution of ast_pct is skewed to the right with a bunch of outliers. We found that a square root transformation made distribution more normal and produced zero outliers.

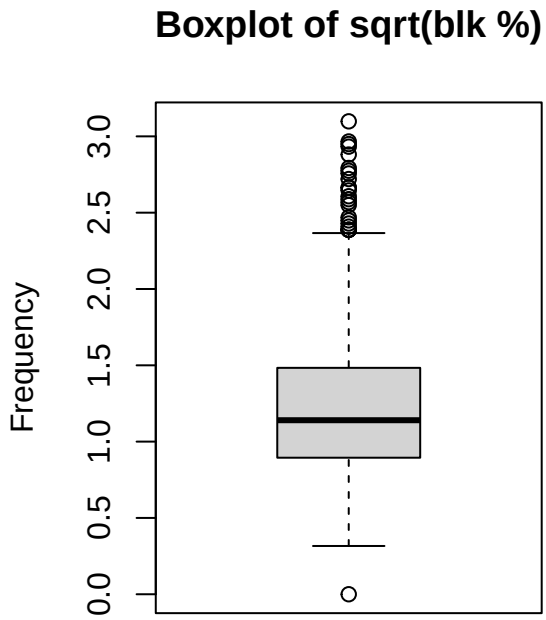


blk_pct:

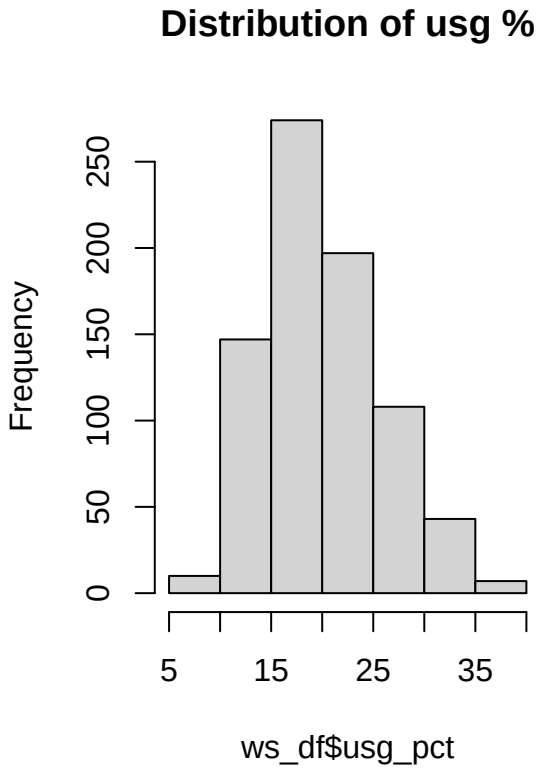


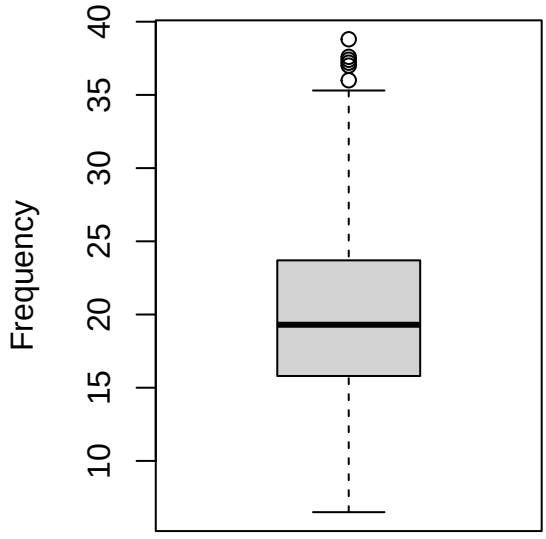
The distribution of blk_pct is heavily skewed right with many outliers. We found that a square root transformation makes distribution more normal. Still a lot of outliers.



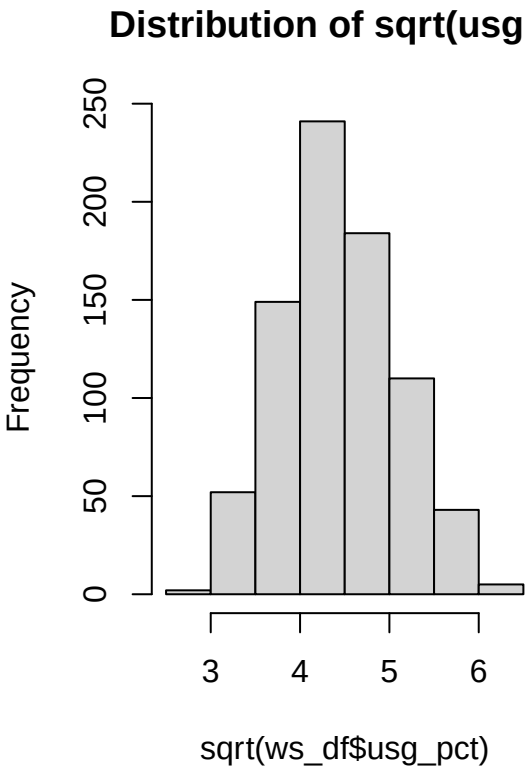


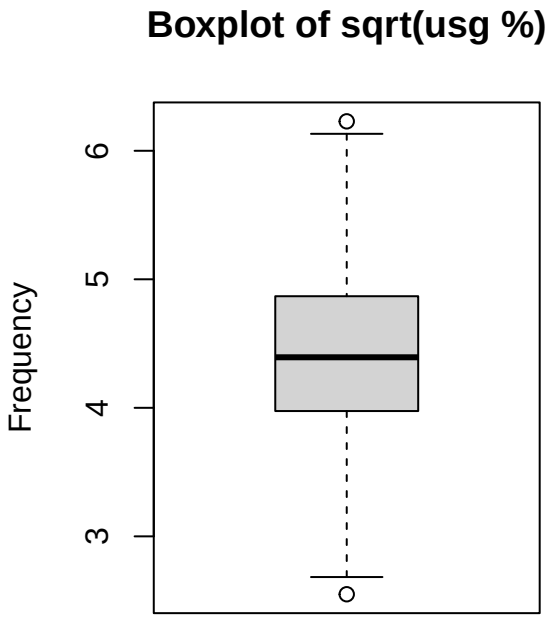
usg_pct:



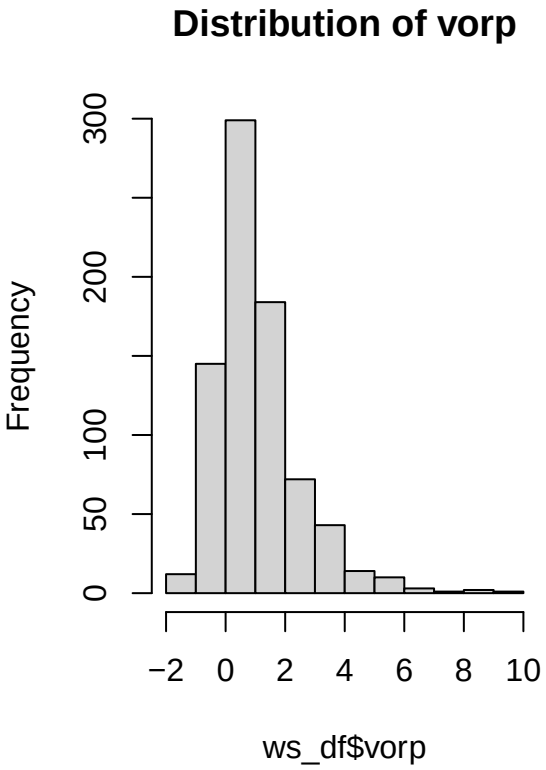


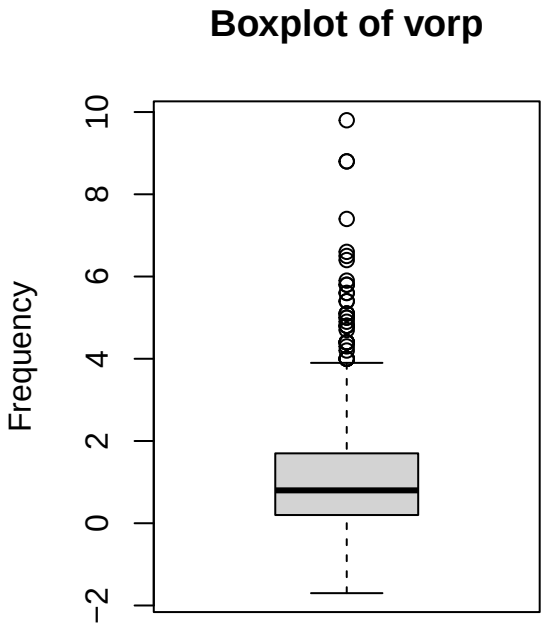
The disribution of usg_pct is relatively normal with a few outliers. We found that a square root transformation makes distribution more normal and only produces two outliers.





vorp:

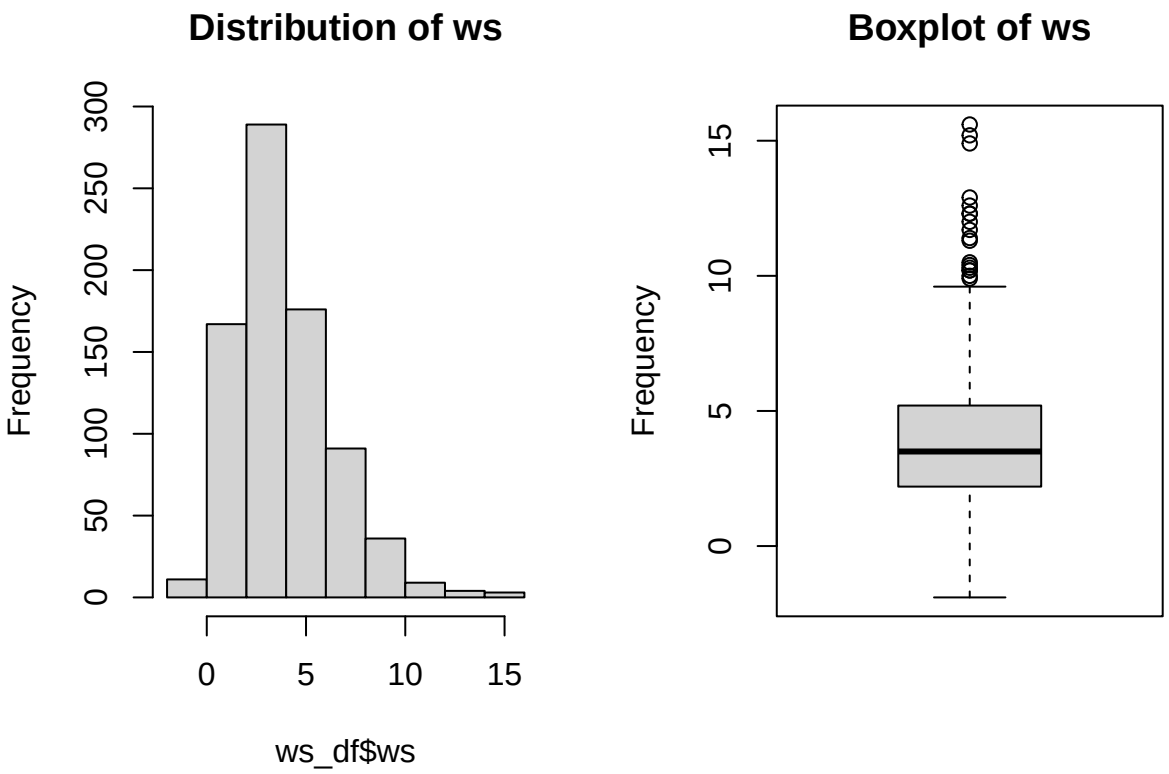




The distribution of vorp is skewed heavily to the right with many outliers. We found that no transformations made the distribution more normal.

Let us look at distribution of our ws response variable.

ws:



The distribution of ws is slightly skewed to the right with many outliers. We decided not to transform our response variable.
Now, that we have explored our data and removed/transformed some predictors, we can now turn our focus to our quantitative regression model analysis.

Part 2

Quantitative Regression Analysis

In this section of our analysis, we will run several regression models. We will start with a linear regression model on the entire data itself to get an idea of how the data is fitting to the model. After we explore how a linear model performs on the entire data set, we will then create a training and test data set to see how well a trained model can predict the ws response variable of our test data. Let us start with looking at a linear model on the entire data set.

Linear Regression Model

Let us run a linear regression model on entire data set.

```
ws_lm = lm(ws ~., ws_df)
summary(ws_lm)
```

```
##
## Call:
```

```
## lm(formula = ws ~ ., data = ws_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.92773 -0.36568  0.00095  0.36946  1.74209
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.109e+00  4.756e-01  -4.434 1.06e-05 ***
## age          9.960e-03  4.967e-03   2.005   0.0453 *
## g            1.814e-02  2.709e-03   6.696 4.11e-11 ***
## mp           1.206e-03  7.018e-05  17.181  < 2e-16 ***
## per          3.061e-01  1.837e-02  16.660  < 2e-16 ***
## ts_pct       1.021e+00  7.750e-01   1.318   0.1879
## ftr          1.955e+00  2.252e-01   8.680  < 2e-16 ***
## trb_pct     -3.706e-02  7.437e-03  -4.983 7.73e-07 ***
## ast_pct     -3.635e-02  3.543e-03 -10.259  < 2e-16 ***
## blk_pct     -1.246e-01  1.965e-02  -6.340 3.91e-10 ***
## usg_pct     -1.589e-01  7.890e-03 -20.139  < 2e-16 ***
## vorp         9.758e-01  3.475e-02  28.077  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5491 on 774 degrees of freedom
## Multiple R-squared:  0.9523, Adjusted R-squared:  0.9516
## F-statistic:  1405 on 11 and 774 DF,  p-value: < 2.2e-16
```

We are looking to see which predictors are statistically significant when predicting ws. To do this, we we will use the hypothesis test with the null hypothesis being that none of the predictors are statistically significant. If the p-value for a predictor is less than 0.05, we reject the null hypothesis and say that the predictor is statistically significant.
We see that all of the p-values of our predictors are less than 0.05 besides ts_pct. age is barely less than 0.05, so we will keep it. Let us remove ts_pct.

```
ws_lm = lm(ws ~. - ts_pct, ws_df)
summary(ws_lm)
```

```
##
## Call:
## lm(formula = ws ~ . - ts_pct, data = ws_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.90960 -0.36052 -0.00001  0.35806  1.75707
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.5822096  0.2578008  -6.137 1.34e-09 ***
## age          0.0104842  0.0049533   2.117   0.0346 *
## g            0.0183306  0.0027064   6.773 2.49e-11 ***
## mp           0.0012074  0.0000702  17.198  < 2e-16 ***
## per          0.3209740  0.0144804  22.166  < 2e-16 ***
## ftr          2.0017572  0.2224906   8.997  < 2e-16 ***
```

```
## trb_pct     -0.0406494  0.0069242  -5.871 6.44e-09 ***
## ast_pct     -0.0385697  0.0031189 -12.367  < 2e-16 ***
## blk_pct     -0.1295778  0.0192821  -6.720 3.52e-11 ***
## usg_pct     -0.1651351  0.0063196 -26.131  < 2e-16 ***
## vorp         0.9746335  0.0347587  28.040  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5493 on 775 degrees of freedom
## Multiple R-squared:  0.9522, Adjusted R-squared:  0.9516
## F-statistic:  1544 on 10 and 775 DF,  p-value: < 2.2e-16
```

All of the p-values of each predictor are less than 0.05 in this model. Our fitted model explains 95% of the variability in the data. Which is good!
Let's use this model to predict the ws values within the model.

```
pred_ws_lm = predict(ws_lm, newdata = ws_df)
mean((pred_ws_lm - ws_df$ws)^2)
```

```
## [1] 0.2975244
```

This model produced a mean squared error rate of 0.30 when we predicted ws using the fitted model.
Let us add the transformations we did earlier to see if we can reduce that mean squared error rate.

```
transform_ws_lm = lm(ws ~ age + g + log(mp) + log(per) + log(ts_pct) + sqrt(ftr) +
             log(trb_pct) + sqrt(ast_pct) + sqrt(blk_pct) + sqrt(usg_pct) +
             vorp, ws_df)
summary(transform_ws_lm)
```

```
##
## Call:
## lm(formula = ws ~ age + g + log(mp) + log(per) + log(ts_pct) +
##     sqrt(ftr) + log(trb_pct) + sqrt(ast_pct) + sqrt(blk_pct) +
##     sqrt(usg_pct) + vorp, data = ws_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.31614 -0.37758 -0.00181  0.38805  1.97576
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -13.015995   0.966703 -13.464  < 2e-16 ***
## age             0.008728   0.005330   1.638  0.10191
## g               0.020415   0.002866   7.124 2.40e-12 ***
## log(mp)         1.660859   0.123092  13.493  < 2e-16 ***
## log(per)        3.081229   0.239345  12.874  < 2e-16 ***
## log(ts_pct)     1.035495   0.494092   2.096  0.03643 *
## sqrt(ftr)       2.565850   0.236851  10.833  < 2e-16 ***
## log(trb_pct)   -0.264581   0.080823  -3.274  0.00111 **
## sqrt(ast_pct)  -0.341330   0.032237 -10.588  < 2e-16 ***
## sqrt(blk_pct)  -0.350068   0.064861  -5.397 9.01e-08 ***
## sqrt(usg_pct)  -1.113505   0.068364 -16.288  < 2e-16 ***
```

```
## vorp            1.218487   0.030285  40.234  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5875 on 774 degrees of freedom
## Multiple R-squared:  0.9454, Adjusted R-squared:  0.9446
## F-statistic:  1218 on 11 and 774 DF,  p-value: < 2.2e-16
```

We see that age has p-value greater than 0.05. Let us remove age predictor.

```
transform_ws_lm = lm(ws ~ g + log(mp) + log(per) + log(ts_pct) + sqrt(ftr) +
             log(trb_pct) + sqrt(ast_pct) + sqrt(blk_pct) + sqrt(usg_pct) +
             vorp, ws_df)
summary(transform_ws_lm)
```

```
##
## Call:
## lm(formula = ws ~ g + log(mp) + log(per) + log(ts_pct) + sqrt(ftr) +
##     log(trb_pct) + sqrt(ast_pct) + sqrt(blk_pct) + sqrt(usg_pct) +
##     vorp, data = ws_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.26909 -0.37433 -0.01316  0.38476  1.94205
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -12.615431   0.936261 -13.474  < 2e-16 ***
## g               0.019843   0.002847   6.969 6.84e-12 ***
## log(mp)         1.659716   0.123224  13.469  < 2e-16 ***
## log(per)        3.064425   0.239384  12.801  < 2e-16 ***
## log(ts_pct)     1.100563   0.493026   2.232  0.02588 *
## sqrt(ftr)       2.552475   0.236967  10.771  < 2e-16 ***
## log(trb_pct)   -0.265952   0.080906  -3.287  0.00106 **
## sqrt(ast_pct)  -0.337775   0.032199 -10.490  < 2e-16 ***
## sqrt(blk_pct)  -0.356302   0.064819  -5.497 5.25e-08 ***
## sqrt(usg_pct)  -1.124887   0.068084 -16.522  < 2e-16 ***
## vorp            1.226286   0.029940  40.958  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5881 on 775 degrees of freedom
## Multiple R-squared:  0.9452, Adjusted R-squared:  0.9445
## F-statistic:  1337 on 10 and 775 DF,  p-value: < 2.2e-16
```

All of the predictors in this model have p-values that are less than 0.05. This model explains about 94.5% of the variability in the data. Slightly less than our linear model with no transformations.
Let us predict ws using transformed linear model.

```
pred_transform_ws_lm = predict(transform_ws_lm, newdata = ws_df)
mean((pred_transform_ws_lm - ws_df$ws)^2)
```

```
## [1] 0.3410566
```

The mean squared error was 0.34. This is higher than the 0.30 that our original linear model produced.

Training and Test Data

Now, let us create training and test data from ws_df. We can then fit our models using training data to then predict ws variable in test data.

```
dim(ws_df)
```

```
## [1] 786  12
```

As we know, there are 786 observations.
Since we only have 786 observations, our parameter estimates will have higher variance. Since we have a relatively smaller number of observations, let us do a 70:30 split on our data where 70% of our data will be our training data,and 30% of our data will be our test data.

```
set.seed(1)
index = createDataPartition(ws_df$ws, p = 0.7, list = FALSE)
train_df = ws_df[index,]
test_df = ws_df[-index,]
```

Let us look at the dimensions for each set to see how many observations are each.
Training set:

```
dim(train_df)
```

```
## [1] 552  12
```

There are 552 observations in training set.

Test set:

```
dim(test_df)
```

```
## [1] 234  12
```

There are 234 observations in test set.

Linear Regression Model Continued

Let us now fit our non-transformed linear model from earlier using the training data.

```
ws_lm = lm(ws ~. - ts_pct, train_df)
summary(ws_lm)
```

```
##
## Call:
## lm(formula = ws ~ . - ts_pct, data = train_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.8979 -0.3589  0.0053  0.3836  1.7387
```

```
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.341e+00  3.181e-01  -4.215 2.93e-05 ***
## age          3.160e-03  6.074e-03   0.520    0.603
## g            1.752e-02  3.259e-03   5.376 1.13e-07 ***
## mp           1.203e-03  8.426e-05  14.283  < 2e-16 ***
## per          3.250e-01  1.837e-02  17.693  < 2e-16 ***
## ftr          1.956e+00  2.659e-01   7.358 7.00e-13 ***
## trb_pct     -4.790e-02  8.654e-03  -5.535 4.87e-08 ***
## ast_pct     -3.817e-02  3.813e-03 -10.008  < 2e-16 ***
## blk_pct     -1.099e-01  2.311e-02  -4.758 2.51e-06 ***
## usg_pct     -1.664e-01  7.698e-03 -21.615  < 2e-16 ***
## vorp         9.902e-01  4.440e-02  22.300  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5596 on 541 degrees of freedom
## Multiple R-squared:  0.9481, Adjusted R-squared:  0.9471
## F-statistic: 988.1 on 10 and 541 DF,  p-value: < 2.2e-16
```

We see our model on training data explains 94.8% of variability. We see that age is not statistically significant in training data as the p-value is greater than 0.05. Let us remove it.

```
ws_lm = lm(ws ~. - age - ts_pct, train_df)
summary(ws_lm)
```

```
##
## Call:
## lm(formula = ws ~ . - age - ts_pct, data = train_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.89427 -0.35442  0.00275  0.38418  1.72577
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.2347230  0.2443788  -5.052 5.97e-07 ***
## g            0.0173210  0.0032341   5.356 1.26e-07 ***
## mp           0.0012038  0.0000842  14.297  < 2e-16 ***
## per          0.3251438  0.0183538  17.715  < 2e-16 ***
## ftr          1.9526267  0.2656254   7.351 7.29e-13 ***
## trb_pct     -0.0480439  0.0086435  -5.558 4.28e-08 ***
## ast_pct     -0.0381645  0.0038108 -10.015  < 2e-16 ***
## blk_pct     -0.1108623  0.0230216  -4.816 1.91e-06 ***
## usg_pct     -0.1670307  0.0075958 -21.990  < 2e-16 ***
## vorp         0.9932161  0.0439889  22.579  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5592 on 542 degrees of freedom
## Multiple R-squared:  0.9481, Adjusted R-squared:  0.9472
## F-statistic:  1099 on 9 and 542 DF,  p-value: < 2.2e-16
```

The R^2 value is still 94.81%. Let us predict test data ws value using our trained linear model.

```
pred_ws_lm = predict(ws_lm, newdata = test_df)
mean((pred_ws_lm - test_df$ws)^2)
```

```
## [1] 0.2851207
```

Our fitted model produced a mean squared error rate of 0.29 when predicting test data ws value.

Let us try fitting transformed model using the training data set.

```
transform_ws_lm = lm(ws ~ age + g + log(mp) + log(per) + log(ts_pct) + sqrt(ftr) + log(trb_pct) + sqrt(a
summary(transform_ws_lm)
```

```
##
## Call:
## lm(formula = ws ~ age + g + log(mp) + log(per) + log(ts_pct) +
##     sqrt(ftr) + log(trb_pct) + sqrt(ast_pct) + sqrt(blk_pct) +
##     sqrt(usg_pct) + vorp, data = train_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.20674 -0.38523 -0.01749  0.40449  1.89543
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -1.338e+01  1.154e+00 -11.592  < 2e-16 ***
## age            9.450e-05  6.397e-03   0.015 0.988220
## g              1.848e-02  3.418e-03   5.406 9.68e-08 ***
## log(mp)        1.722e+00  1.459e-01  11.807  < 2e-16 ***
## log(per)       3.364e+00  3.047e-01  11.042  < 2e-16 ***
## log(ts_pct)    1.031e+00  5.921e-01   1.741 0.082308 .
## sqrt(ftr)      2.400e+00  2.801e-01   8.567  < 2e-16 ***
## log(trb_pct)  -3.397e-01  9.861e-02  -3.445 0.000615 ***
## sqrt(ast_pct) -3.370e-01  3.880e-02  -8.685  < 2e-16 ***
## sqrt(blk_pct) -3.248e-01  7.683e-02  -4.228 2.77e-05 ***
## sqrt(usg_pct) -1.181e+00  8.253e-02 -14.313  < 2e-16 ***
## vorp           1.208e+00  3.862e-02  31.275  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5867 on 540 degrees of freedom
## Multiple R-squared:  0.943,  Adjusted R-squared:  0.9419
## F-statistic: 812.8 on 11 and 540 DF,  p-value: < 2.2e-16
```

We see that age and log(ts_pct) are not statistically significant predictors in this model. Let us remove each predictor.

```
transform_ws_lm = lm(ws ~ g + log(mp) + log(per) + sqrt(ftr) +log(trb_pct) +
                       sqrt(ast_pct) + sqrt(blk_pct) + sqrt(usg_pct) +
                       vorp, train_df)
summary(transform_ws_lm)
```

```
##
## Call:
## lm(formula = ws ~ g + log(mp) + log(per) + sqrt(ftr) + log(trb_pct) +
##     sqrt(ast_pct) + sqrt(blk_pct) + sqrt(usg_pct) + vorp, data = train_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.22803 -0.39764 -0.01061  0.39390  1.91876
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -14.147858   1.040062 -13.603  < 2e-16 ***
## g               0.018746   0.003388   5.533 4.91e-08 ***
## log(mp)         1.710810   0.145817  11.733  < 2e-16 ***
## log(per)        3.698590   0.236979  15.607  < 2e-16 ***
## sqrt(ftr)       2.496601   0.274222   9.104  < 2e-16 ***
## log(trb_pct)   -0.399044   0.092613  -4.309 1.95e-05 ***
## sqrt(ast_pct)  -0.373073   0.032813 -11.370  < 2e-16 ***
## sqrt(blk_pct)  -0.358483   0.074312  -4.824 1.83e-06 ***
## sqrt(usg_pct)  -1.267067   0.065753 -19.270  < 2e-16 ***
## vorp            1.219194   0.037794  32.259  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5873 on 542 degrees of freedom
## Multiple R-squared:  0.9427, Adjusted R-squared:  0.9418
## F-statistic: 991.2 on 9 and 542 DF,  p-value: < 2.2e-16
```

All of the predictors are statistically significant in this model. Let us now predict the ws value in test data.

```
pred_transform_ws_lm = predict(transform_ws_lm, newdata = test_df)
mean((pred_transform_ws_lm - test_df$ws)^2)
```

```
## [1] 0.360101
```

The mean squared error rate that our transformed model produced when predicting test data ws value was 0.36. Our model with no transformations produced a lower mean squared error. Since both models produced a similar R^2 of approximately 94%, we will choose our linear model that produced a mean squared error rate of 0.29. We will compare every other model's mean squared error rate to this mean squared error rate to see what our final model selection will be.

Ridge Regression Model

Using ridge regression, our goal here is to see if we can produce a test mean squared error less than the the mean squared error that we found in the linear regression model section above. What we will first do is create a training matrix and a test matrix for our ridge regression. Using cross-validation, I found that the best shrinking parameter, lambda, to use for this model is when lambda = 0.2152. Using lambda = 0.2152 in my ridge regression model and using the training matrix to predict the response ws values in the test data, I found a test mean squared error of 0.34. This model did not produced a lower test mean squared error than our linear regression model.

```
train_matrix = model.matrix(ws ~., train_df)
test_matrix = model.matrix(ws ~., test_df)
# Now we need to select lambda using cross-validation
cv_out = cv.glmnet(train_matrix, train_df$ws, alpha = 0)
best_lam = cv_out$lambda.min
best_lam
```

```
## [1] 0.2152225
```

Lambda chosen by cross-validation is 0.2152 Now we fit ridge regression model and make predictions.

```
ws_ridge = glmnet(train_matrix, train_df$ws, alpha = 0)
pred_ws_ridge = predict(ws_ridge, s = best_lam, newx = test_matrix)
# Find mean squared error:
mean((pred_ws_ridge - test_df$ws)^2)
```

```
## [1] 0.3376394
```

Lasso Regression

Now we will fit a lasso regression model to the data. We will use the same training and test matrices created in the ridge regression model from the previous section to fit this lasso regression model. The best shrinking parameter, lambda, to use in the lasso regression model is when lambda = 0.0027. Using that lambda in the regression model, we get a test mean squared error value of 0.28. Our lasso regression model, although not by much, did produce a lower test mean squared error than our linear model. We will keep this model in mind moving forward. However, our linear model is more interpretable than our lasso regression model.

```
cv_out = cv.glmnet(train_matrix, train_df$ws, alpha = 1)
best_lam = cv_out$lambda.min
best_lam
```

```
## [1] 0.002653364
```

Lambda chosen by cross-validation is 0.0027.
Now we fit lasso regression model and make predictions.

```
ws_lasso = glmnet(train_matrix, train_df$ws, alpha = 1)
pred_ws_lasso = predict(ws_lasso, s = best_lam, newx = test_matrix)
# Find mean squared error:
mean((pred_ws_lasso - test_df$ws)^2)
```

```
## [1] 0.2842548
```

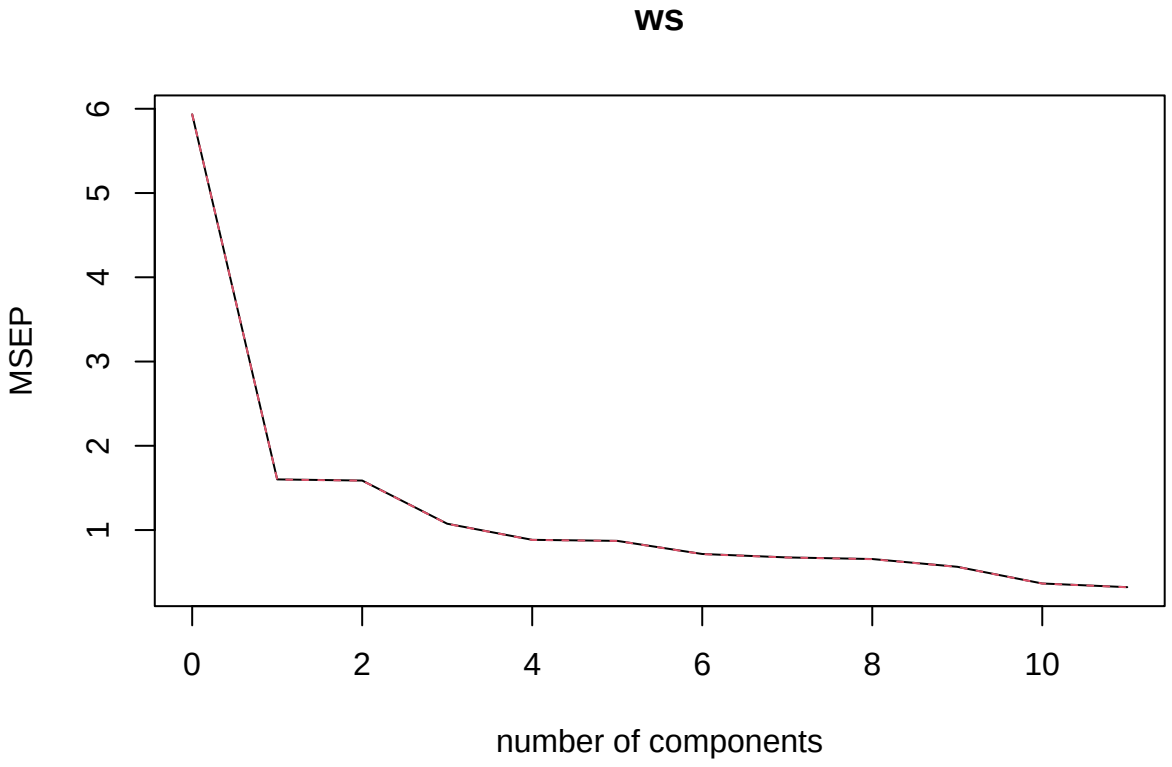
Principal Components Regression

The main goal of principal components regression is to reduce the dimensions of the model by removing predictors to simplify the model. Let us fit a principal components regression model using the training data to then predict the response variable ws in the test data.

```r
ws_pcr = pcr(ws ~., data = train_df,
                 scale = TRUE, validation = "CV")
summary(ws_pcr)
```

```
## Data:    X dimension: 552 11
##  Y dimension: 552 1
## Fit method: svdpc
## Number of components considered: 11
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           2.436    1.265    1.260    1.037   0.9402   0.9335   0.8460
## adjCV        2.436    1.265    1.259    1.036   0.9394   0.9331   0.8451
##        7 comps  8 comps  9 comps  10 comps  11 comps
## CV      0.8215   0.8099   0.7510    0.6048    0.5676
## adjCV   0.8210   0.8090   0.7501    0.6040    0.5668
##
## TRAINING: % variance explained
##     1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X     32.25    53.40    67.54    77.55    83.70    89.11    92.82    96.17
## ws    73.13    73.48    82.14    85.43    85.76    88.39    89.10    89.41
##     9 comps  10 comps  11 comps
## X     98.36     99.65    100.00
## ws    90.91     94.14     94.83
```

```r
validationplot(ws_pcr, val.type = "MSEP")
```



We see that M = 10 produces lowest mean squared error of prediction on training data The PCR model when M = 11, explains 99% of the variance of the training data.

```
pred_ws_pcr = predict(ws_pcr, test_df, ncomp = 11)
mean((pred_ws_pcr - test_df$ws)^2)
```

```
## [1] 0.2833607
```

Using all 11 predictors, we see that we get a mean squared error rate of 0.28. We want to try and reduce our dimensions of our model using principal component regression. Our linear model has 9 predictors. We see that M = 7 produces a relatively low mean squared error on the training data. Let us try M = 7 in our prediction model.

```
pred_ws_pcr = predict(ws_pcr, test_df, ncomp = 7)
mean((pred_ws_pcr - test_df$ws)^2)
```

```
## [1] 0.6098959
```

This produced a test mean squared error rate of 0.61. This is much higher than what we have produced so far.

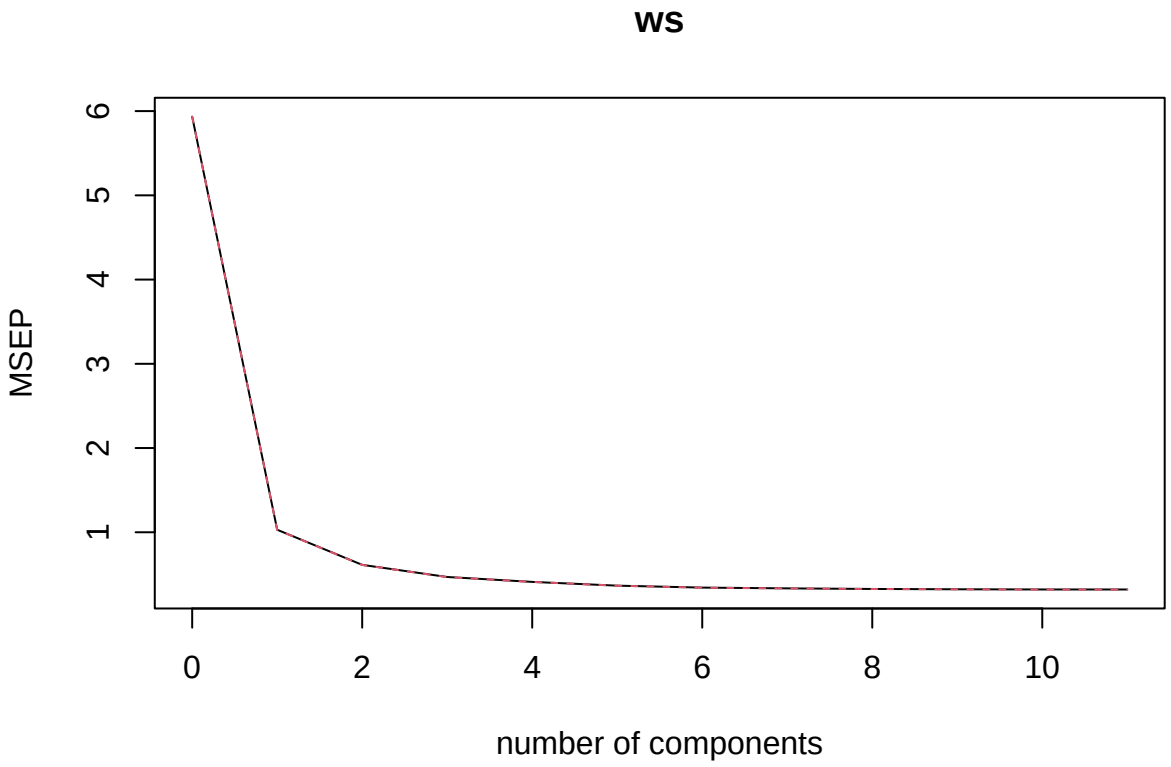
Partial Least Squares

We will now try and reduce the dimensions by fitting a partial least squares model. Our goal, once again, is to create a more simple model.

```
ws_pls = plsr(ws ~., data = train_df,
                    scale = TRUE, validation = "CV")
summary(ws_pls)
```

```
## Data:    X dimension: 552 11
##  Y dimension: 552 1
## Fit method: kernelpls
## Number of components considered: 11
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           2.436    1.015   0.7831   0.6849   0.6410   0.6052   0.5859
## adjCV        2.436    1.014   0.7824   0.6839   0.6408   0.6046   0.5850
##        7 comps  8 comps  9 comps  10 comps  11 comps
## CV      0.5780   0.5719   0.5686    0.5665    0.5663
## adjCV   0.5771   0.5714   0.5682    0.5658    0.5656
##
## TRAINING: % variance explained
##     1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X     31.69    44.76    53.83    68.35    78.28    84.28    87.49    90.77
## ws    82.88    89.91    92.41    93.34    94.07    94.47    94.63    94.72
##     9 comps  10 comps  11 comps
## X     94.42     96.62    100.00
## ws    94.79     94.83     94.83
```

```
validationplot(ws_pls, val.type = "MSEP")
```



We see that M = 11 produces lowest mean squared error of prediction on training data. However, we are trying to reduce variables in the model, so let's look at when M = 6 as that has almost same MSEP as when M = 11. The partial least squares model when M = 6 explains about 94.5% of the variance of training data.

```
pred_ws_pls = predict(ws_pls, test_df, ncomp = 6)
mean((pred_ws_pls - test_df$ws)^2)
```

```
## [1] 0.3086326
```

This partial least squares model produced a test mean squared error rate of 0.31. There are 7 predictors being used: age, g, ts_pct, ftr, trb_pct, ast_pct, usg_pct. This is only two less predictors than our linear model.
Let us find the mean square error rate when M = 5 to reduce dimensions of model.

```
pred_ws_pls = predict(ws_pls, test_df, ncomp = 5)
mean((pred_ws_pls - test_df$ws)^2)
```

```
## [1] 0.3566948
```

This produced a mean squared error rate of 0.36. This is no lower than what we have already produced.

Regression Trees

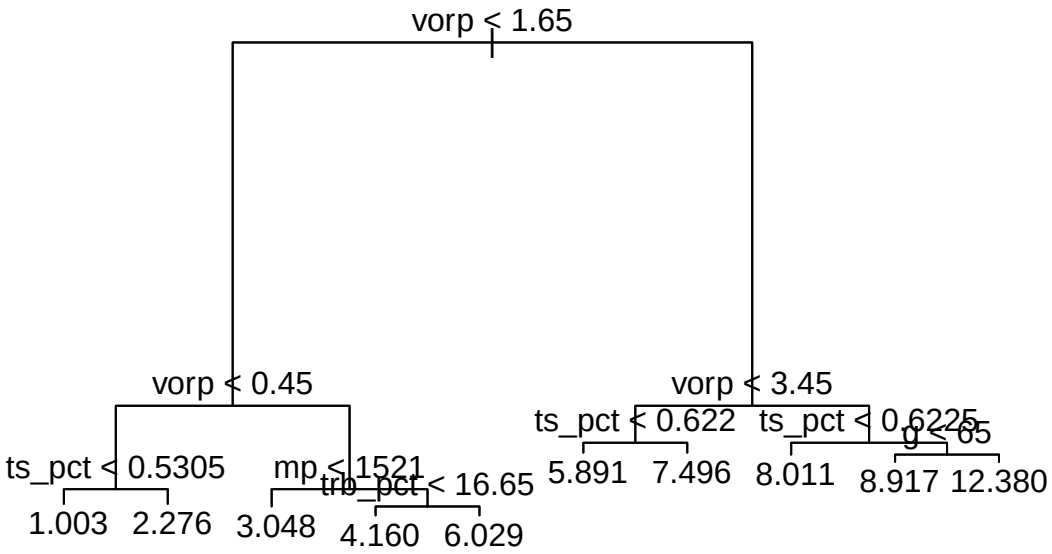
We will now fit a regression tree using our training data set and analyze how many terminal nodes we will have.

```
ws_tree = tree(ws ~ ., train_df)
summary(ws_tree)
```

```
##
## Regression tree:
## tree(formula = ws ~ ., data = train_df)
## Variables actually used in tree construction:
## [1] "vorp"    "ts_pct"  "mp"      "trb_pct" "g"
## Number of terminal nodes:  10
## Residual mean deviance:  1.027 = 556.8 / 542
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -3.760000 -0.690800  0.009211  0.000000  0.640400  3.217000
```

This tree has 10 terminal nodes. There were five predictors used in construction of tree: vorp, ts_pct, mp, trb_pct, and g.

```
plot(ws_tree)
text(ws_tree, pretty = 0)
```



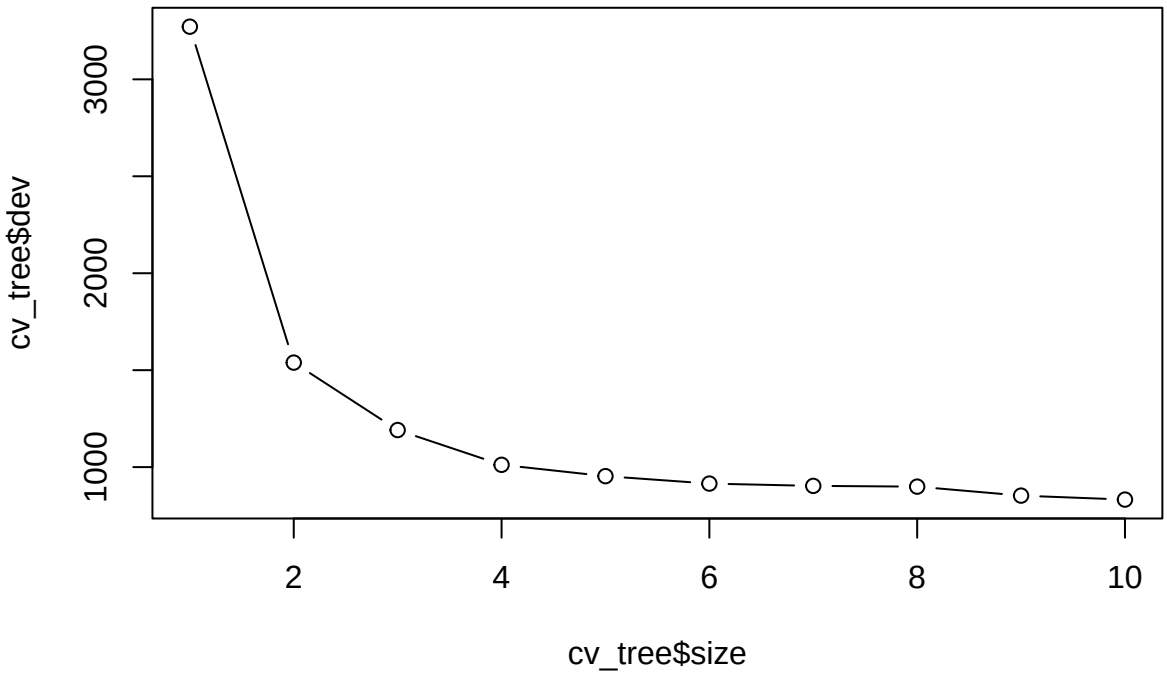
Let us use our tree model to predict test data ws.

```
pred_ws_tree = predict(ws_tree, newdata = test_df)
mean((pred_ws_tree - test_df$ws)^2)
```

```
## [1] 1.432647
```

Our tree model produced a mean squared error greater than 1. This may mean that our tree overfitted our training data. Let us try pruning our tree model using cross-validation to see if that reduces the mean squared error rate.

```
cv_tree = cv.tree(ws_tree)
plot(cv_tree$size, cv_tree$dev, type = "b")
```



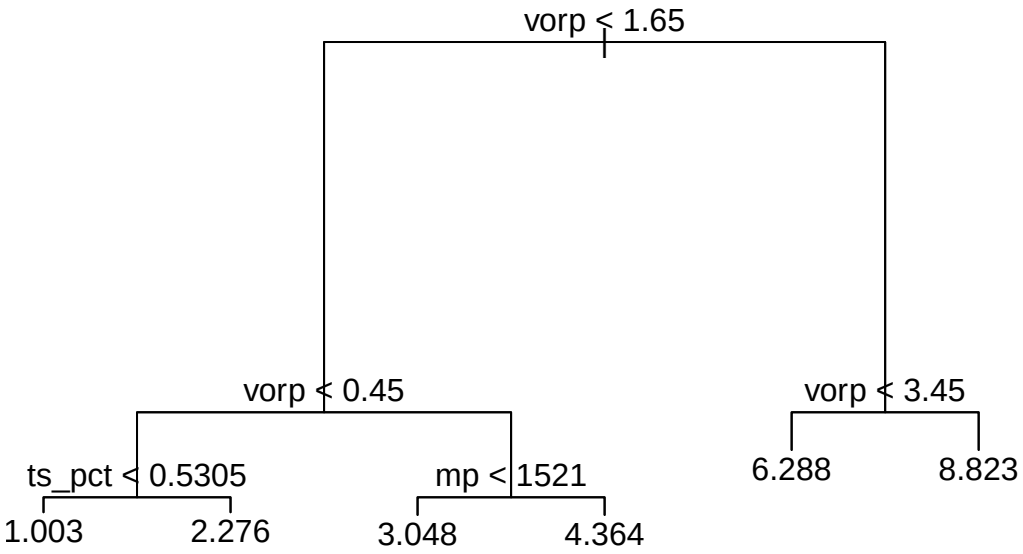
We see that tree size 10 minimizes cross-validation error. However, we already used 10 terminal nodes in our first tree model. Based on the plot, we see that 6 terminal nodes relatively minimizes cross-validation error. Let us prune our tree to 6 terminal nodes and see if it has any effect on our mean squared error.

```
ws_prune = prune.tree(ws_tree, best = 6)
summary(ws_prune)
```

```
##
## Regression tree:
## snip.tree(tree = ws_tree, nodes = c(11L, 6L, 7L))
## Variables actually used in tree construction:
## [1] "vorp"   "ts_pct" "mp"
## Number of terminal nodes:  6
## Residual mean deviance:  1.36 = 742.7 / 546
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -3.9640 -0.7757 -0.0481  0.0000  0.6359  6.7770
```

This tree has 6 terminal nodes. There were three predictors used in construction of tree: vorp, ts_pct and mp.

```
plot(ws_prune)
text(ws_prune, pretty = 0)
```



Let us now predict test data ws value using pruned tree model.

```
pred_ws_prune = predict(ws_prune, newdata = test_df)
mean((pred_ws_prune - test_df$ws)^2)
```

```
## [1] 1.786669
```

Like our first tree model, this pruned tree model produced a mean squared error greater than 1. We will not be using regression trees to predict ws value.

Bagging Model

Now we will fit a bagging model with 25 boostrap replications and find the test mean squared error rate to see how they compare to previous models.

```
ws_bag = bagging(ws ~., data = train_df)
ws_bag
```

```
##
## Bagging regression trees with 25 bootstrap replications
##
## Call: bagging.data.frame(formula = ws ~ ., data = train_df)
```

Let us predict test data ws value using bagging model.

```
pred_ws_bag = predict(ws_bag, newdata = test_df)
mean((pred_ws_bag - test_df$ws)^2)
```

```
## [1] 1.124551
```

Similar to the regression tree models we fit, our bagging model did not perform well as it produced a test MSE over 1. We will not be using this model.

Random Forest

Now we will fit a random forest model with 100 trees and using all predictors in the data set and then use that model to predict our test data.

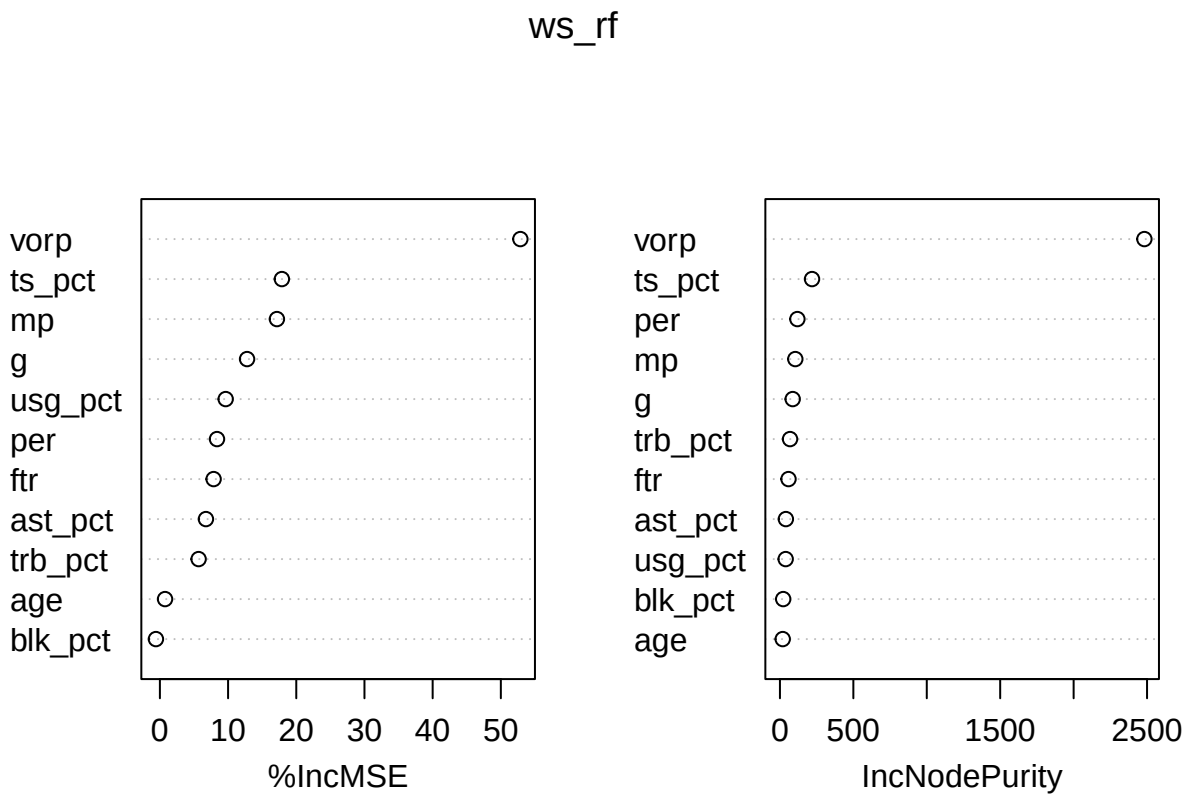
```
set.seed(1)
ws_rf = randomForest(ws ~., ntree = 100,
                          mtry = 12, importance = TRUE, data = train_df)
```

```
## Warning in randomForest.default(m, y, ...): invalid mtry: reset to within valid
## range
```

```
ws_rf
```

```
##
## Call:
##  randomForest(formula = ws ~ ., data = train_df, ntree = 100,      mtry = 12, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 100
## No. of variables tried at each split: 11
##
##           Mean of squared residuals: 0.6496693
##                     % Var explained: 89.01
```

```
varImpPlot(ws_rf)
```



Using 11 variables at each split, this random forest model explains approximately 89% of the data. We see that vorp, ts_pct and mp are the three most important predictors in this model.
Let us now predict the test data ws value.

```
pred_ws_rf = predict(ws_rf, newdata = test_df)
mean((pred_ws_rf - test_df$ws)^2)
```

```
## [1] 0.5160904
```

Our random forest model produced a test mean squared error rate of 0.52. This is no better than what we have already produced with previous models.

Boosted Model

In this section, we will fit a boosted model using the training data and will make predictions on that model using the test data to compare. We will try a few different shrinking parameter lambda values to compare mean squared test error rates of models. Our boosted models that we will fit will use lambda values of 0.001, 0.01. 0.1, and 0.2 and 1000 trees.

Lambda = 0.001:

```
ws_boost = gbm(ws ~., data = train_df, distribution = "gaussian",
                   n.trees = 1000)
pred_ws_boost = predict(ws_boost, newdata = test_df,
                        n.trees = 1000)
mean((pred_ws_boost - test_df$ws)^2)
```

```
## [1] 0.4480716
```

This boosted model produced a test mean squared error rate of 0.45.

Lambda = 0.01:

```
ws_boost = gbm(ws ~., data = train_df, distribution = "gaussian",
               n.trees = 1000, shrinkage = 0.01)
pred_ws_boost = predict(ws_boost, newdata = test_df,
                        n.trees = 1000)
mean((pred_ws_boost - test_df$ws)^2)
```

```
## [1] 0.6491903
```

This boosted model produced a test mean squared error rate of 0.65.

Lambda = 0.1

```
ws_boost = gbm(ws ~., data = train_df, distribution = "gaussian",
               n.trees = 1000, shrinkage = 0.1)
pred_ws_boost = predict(ws_boost, newdata = test_df,
                        n.trees = 1000)
mean((pred_ws_boost - test_df$ws)^2)
```

```
## [1] 0.4808009
```

This boosted model produced a test mean squared error rate of 0.48.

Lambda = 0.2

```
ws_boost = gbm(ws ~., data = train_df, distribution = "gaussian",
               n.trees = 1000, shrinkage = 0.2)
pred_ws_boost = predict(ws_boost, newdata = test_df,
                        n.trees = 1000)
mean((pred_ws_boost - test_df$ws)^2)
```

```
## [1] 0.4882996
```

This boosted model produced a test mean squared error rate of 0.49.
None of the boosted models performed better than our linear model.

Analysis Conclusion

During our quantitative regression analysis, we fit nine models in our analysis: linear regression model, ridge regression mode, lasso regression model, principal component regression model, partial least squares model, regression tree model, bagging model, random forest model and a boosted model.
When using the NBA training data to train these models and predict the ws response variable for each player in our test data set, we were only able to produce various test mean squared error rates. Below you will find each model we fit along with the corresponding mean squared error rate that the model produced.

Linear Regression: 0.29.

Ridge Regression: 0.34.

Lasso Regression: 0.28.

Principal Component Regression: 0.34.

Partial Least Squares: 0.31.

Regression Tree: 1.43.

Bagging: 1.

Random Forest: 0.52.

Boosted: 0.49.

Of all the models, our lasso regression model produced the lowest test mean squared error rate of 0.28. The next model with the lowest test mean squared error rate is our linear model that we fit. Compared to our lasso regression model, our linear regression model is more interpretable and easier to convey to non-technical stakeholders. Since the difference in test mean squared error rates between the two models is only 0.01, we will choose the fitted linear regression to predict win share value for NBA players.
Our chosen linear model model uses the following variables: g, mp, per, ftr, vorp, trb_pct, ast_pct, blk_pct, and usg_pct to predict ws.

Here is the estimated linear model equation below with each of the predictor's respective coefficient:

Win Shares Estimate = -1.2347 + 0.0173(*Games Played*) + 0.0012(*Minutes Played*) + 0.3251(*Player Efficiency Rating*) + 1.9526(*Free Throw Attempt Rate*) + 0.9932(*Wins Over Replacement*) - 0.0480(*Total Rebound %*) - 0.0382(*Assist %*) - 0.1109(*Block %*) - 0.1670(*Usage %*)

Part 3

We have now chosen our linear regression model to predict NBA player's win share values. Now, we will predict each player's win share value for the 2023-2024 season using their advanced stats from the 2022-2023 season.
Using the 2022-2023 season data, let us create 2023-2024 ws_est (win share estimate) column to estimate win shares for each player for next year.

```
twenty_three_nba_df$ws_est = predict(ws_lm, newdata = twenty_three_nba_df)
```

This code took in the 2022-2023 data and predicted our 2023-2024 win share estimate for each player that played more than 1,000 minutes during the 2022-2023 season.
Let us look at the top ten players in estimated win share value for the 2023-2024 season.

```
top_ten_players_ws_est = twenty_three_nba_df[order(-twenty_three_nba_df$ws_est), ][1:10, c("player", "ws
kable(top_ten_players_ws_est)
```

	player	ws_est
319	Nikola Jokić	15.040304
101	Jimmy Butler	12.486686
185	Joel Embiid	11.963378
550	Domantas Sabonis	11.517184
210	Shai Gilgeous-Alexander	11.214848
161	Luka Dončić	10.918253
591	Jayson Tatum	9.824837
13	Giannis Antetokounmpo	9.366693
374	Damian Lillard	9.325284
142	Anthony Davis	9.189665

Now, let us translate our estimated win share values to a dollar value using Seth Partnow's formula from the Midrange Theory:

Production value = Win Share * League Value Per Win

Seth Partnow used the league value per win of $2.8 million/win from the 2018-2019 season in his book. From the Athletic article written by Mike Vorkunov and Seth Partnow, "NBA analytics: Exactly how much does a win cost?", we found that the league value per win in 2022-2023 was $3.44 million. You can find the link to this Athletic article below and in the sources section:
https://theathletic.com/3517502/2022/08/23/nba-analytics-win-cost/#

We can now use that league value per win to predict a player's estimated production value (in million USD) based on our estimated win share value.

```
twenty_three_nba_df$ws_dollar_val = ((twenty_three_nba_df$ws_est) * 3.44)
```

Let us now look at the top ten players in estimated win share dollar value.

```
top_ten_players_ws_dollar_val = twenty_three_nba_df[order(-twenty_three_nba_df$ws_dollar_val), ][1:10, 
kable(top_ten_players_ws_dollar_val)
```

	player	ws_est	ws_dollar_val
319	Nikola Jokić	15.040304	51.73865
101	Jimmy Butler	12.486686	42.95420
185	Joel Embiid	11.963378	41.15402
550	Domantas Sabonis	11.517184	39.61911
210	Shai Gilgeous-Alexander	11.214848	38.57908
161	Luka Dončić	10.918253	37.55879
591	Jayson Tatum	9.824837	33.79744
13	Giannis Antetokounmpo	9.366693	32.22142
374	Damian Lillard	9.325284	32.07898
142	Anthony Davis	9.189665	31.61245

These are the same players who were top ten in win share estimate for the 2023-2024 season.

Part 4

Acting as the Analytics Director for the Orlando Magic, we will estimate the win share value and win share dollar value for our own players and the available NBA free agents to help make off-season decisions on how we will construct our roster going into the 2023-2024 season. Below you fill find a synopsis of the state of the Orlando Magic and decisions that need to be made following the 2022-2023 season.

Cap Space Projection: $26-63.8 million.

The Magic need to make decisions on Jonathan Isaac, Markelle Fultz, and Gary Harris:

- Isaac has a partial guarantee for $7.6 million next season and a potential salary of $17.4 million.
- Fultz has a partial guarantee of $2 million next season and a potential salary of $17 million.
- Harris has a potential salary of $13 million, which is not guaranteed.
- Assume the Magic will waive Bol Bol, Goga Bitazde and any players with a club option.

Decision 1

Goal: Analyze the players whose options were declined:

Bol Bol: $2.2 million

Goga Bitazde: $2.1 million

Since both Bol Bol and Bitazde declined their options, we now have their combined $2.2 million and $2.1 million added to our cap space. Using our predicted win share values, let us take a look and see what we estimated each player's win share dollar value to be.
Before we do that, let us re-read our 2022-2023 NBA Data in because Goga Bitazde did not play over 1,000 minutes in the 2022-2023 season.

```
twenty_three_nba_df = read.csv("2023_NBA_Data.csv")
twenty_three_nba_df = twenty_three_nba_df[, -which(names(twenty_three_nba_df) ==
                                                   "Player.additional")]
twenty_three_nba_df = twenty_three_nba_df[, -which(names(twenty_three_nba_df) == "Rk")]
twenty_three_nba_df = twenty_three_nba_df[, -which(names(twenty_three_nba_df) == "X")]
twenty_three_nba_df = twenty_three_nba_df[, -which(names(twenty_three_nba_df) == "X.1")]
```

```
twenty_three_nba_df = twenty_three_nba_df[, -which(names(twenty_three_nba_df) == "WS.48")]
colnames(twenty_three_nba_df) = tolower(colnames(twenty_three_nba_df))
predictor_names = colnames(twenty_three_nba_df)
new_names = gsub("\\.", "_pct", predictor_names)
colnames(twenty_three_nba_df) = new_names

player_frequencies = table(twenty_three_nba_df[["player"]])
# Extract player names that have counts greater than 1 (indicating duplicates)
duplicate_players = names(player_frequencies[player_frequencies > 1])

twenty_three_nba_df = twenty_three_nba_df[!(twenty_three_nba_df[["player"]] %in%
                                            duplicate_players &
                                            twenty_three_nba_df[["tm"]] != "TOT"), ]
```

Now let us add our estimated win share and win share dollar value columns into data frame.

```
twenty_three_nba_df$ws_est = predict(ws_lm, newdata = twenty_three_nba_df)
twenty_three_nba_df$ws_dollar_val = ((twenty_three_nba_df$ws_est) * 3.44)
```

Let us pull Bol Bol's and Bitazde's statistics.

```
players_to_search = c("Bol Bol", "Goga Bitadze")
selected_players = twenty_three_nba_df[twenty_three_nba_df$player %in%
                                       players_to_search, ]
kable(selected_players[c("player", "pos", "ws_est", "ws_dollar_val")])
```

	player	pos	ws_est	ws_dollar_val
58	Goga Bitadze	C	2.058863	7.082489
65	Bol Bol	PF	2.666870	9.174034

We see that we were anticipating Bitadze to be worth approx $7.1 million in the 2023-2024 season. We were anticipating Bol Bol to be worth approx $9.2 million in the 2023-2024 season.
Based on our predicted win share dollar values for each player for next year, we would have liked to sign them both at their player option values of $2.1 million and $2.2 million respectfully. However, they declined their options and will be testing the free agency market.
With both players declining their option, we know that our cap space currently sits at $26 million.

Decision 2

Goal: Decide whether to retain Gary Harris.
Let us pull Harris' estimated win share statistics.

```
selected_players = twenty_three_nba_df[twenty_three_nba_df$player == "Gary Harris", ]
kable(selected_players[c("player", "pos", "ws_est", "ws_dollar_val")])
```

	player	pos	ws_est	ws_dollar_val
250	Gary Harris	SG	2.320093	7.981121

Gary Harris' potential salary will be $13 million. Based on our predictions, we see that is estimated win share dollar value for the 2023-2024 season will be approx $8 million. Harris does not have a contract buyout if we waive him. Therefore, I do not think that we should resign Gary Harris.
The $13 million that we will pay him next year does not match the $8 million that we are expecting him to be worth. Since his $13 million his not guaranteed, we should waive Gary Harris. This brings our cap space number from $26 million to $39 million.

Decision 3

Goal: Do we want to guarantee Jonathan Isaac's and/or Markelle Fultz's contract?
Let us pull Isaac's and Fultz's win share stats.

```
players_to_search = c("Markelle Fultz", "Jonathan Isaac")
selected_players = twenty_three_nba_df[twenty_three_nba_df$player %in%
                                       players_to_search, ]
kable(selected_players[c("player", "pos", "ws_est", "ws_dollar_val")])
```

	player	pos	ws_est	ws_dollar_val
200	Markelle Fultz	PG	3.740426	12.867067
297	Jonathan Isaac	PF	1.123787	3.865828

Markelle Fultz:
We are expecting Fultz to be worth approx $12.9 million in the 2023-2024 season. We are planning to pay Fultz $17 million next year. If we waive him, we will have to pay a $2 million buyout. If we add that $2 million dollar buyout to what we think he will be worth next year, that puts his adjusted worth at $14.9 million. We would still be netting out $2.1 million by waiving Fultz.
Therefore, we will not re-sign Fultz. This brings our salary cap from $39 million to $54 million after we buy Fultz out of his contract.

Jonathan Isaac:
We are expecting Isaac to be worth approx $3.9 million in the 2023-2024 season. We are planning to pay Isaac $7.4 million next year. If we waive him, we will have to pay a $7.6 million buyout. Let's add that $7.6 million to what we think he will be worth next year. That puts his adjusted worth at $11.5 million. We would still be netting out $5.9 million by not resigning Isaac.
Therefore, we will also not re-sign Isaac.This brings our salary cap from $54 million to $63.8 million.

Decision 4

Create a Free agency Plan A and Plan B.

We decided to waive Harris, Fultz and Isaac going into the 2023-2024 season. That means we are down a Shooting Guard (SG), Point Guard (PG), and a Power Forward(PF). Bol Bol and Bitadze also declined their player options, which means we are down a Center (C) and another PF. In this analysis, we assume that we will not be making any trades. We also assume that we will be signing any free agent for one year and that they are worth their market value right now.

Let us read in and clean up our NBA free agent data from an Excel spreadsheet.

```
fa_df = readxl::read_xlsx("NBA_Free_Agency_Data.xlsx")

# Linear model we will be using:
```

```
ws_lm = lm(ws ~ g + mp + per + ftr + trb_pct + ast_pct + blk_pct +
             usg_pct + vorp, train_df)

# Let's clean up data frame so our predictors match previous data frames
fa_df = fa_df[, -which(names(fa_df) == "Player-additional")]
colnames(fa_df) = tolower(colnames(fa_df))

# Remove age variable. There are two in data frame
fa_df = fa_df[, -which(names(fa_df) == "age")]

# Remove pos column
fa_df = fa_df[, -which(names(fa_df) == "pos")]

# Rename pos. column
colnames(fa_df)[colnames(fa_df) == "pos."] = "pos"

# Change % to _pct
predictor_names = colnames(fa_df)
new_names = gsub("\\%", "_pct", predictor_names)
colnames(fa_df) = new_names

# change " " to "_"
predictor_names = colnames(fa_df)
new_names = gsub(" ", "_", predictor_names)
colnames(fa_df) = new_names
```

Let us now use our trained linear model to predict each players win share value and win share dollar value.

```
fa_df$ws_est = predict(ws_lm, newdata = fa_df)
fa_df$ws_dollar_val = ((fa_df$ws_est) * 3.44)
```

Let us also convert the market_value column into market value in millions USD.

```
fa_df$market_value = (fa_df$market_value) / 1000000
```

Let us check for NA values by column

```
colSums(is.na(fa_df))
```

```
##         player            pos            exp           team           type
##              0              0              0              0              0
## 2022-23_salary             tm              g             mp            per
##             37              0              0              0              0
##         ts_pct           3par            ftr       orb_pct       drb_pct
##              0              0              0              0              0
##        trb_pct        ast_pct        stl_pct        blk_pct        tov_pct
##              0              0              0              0              0
##        usg_pct            ows            dws             ws         ws/48
##              0              0              0              0              0
##           obpm           dbpm            bpm           vorp   market_value
##              0              0              0              0            102
##         ws_est  ws_dollar_val
##              0              0
```

We see that there are 102 NA values in the market_value column.
Let us look at what the min market_value is in the data set.

```
min(fa_df$market_value, na.rm = TRUE)
```

```
## [1] 1.5
```

We see that $1.5 million is the minimum market value. Let us assume that all players with no market value right now are at least worth the minimum $1.5 million.

```
fa_df$market_value = ifelse(is.na(fa_df$market_value), 1.5, fa_df$market_value)
# Check for NA values by column
colSums(is.na(fa_df))
```

```
##         player            pos            exp           team           type
##              0              0              0              0              0
## 2022-23_salary             tm              g             mp            per
##             37              0              0              0              0
##         ts_pct          3par            ftr       orb_pct       drb_pct
##              0              0              0              0              0
##        trb_pct        ast_pct        stl_pct        blk_pct        tov_pct
##              0              0              0              0              0
##        usg_pct            ows            dws             ws          ws/48
##              0              0              0              0              0
##           obpm           dbpm            bpm           vorp   market_value
##              0              0              0              0              0
##         ws_est  ws_dollar_val
##              0              0
```

Now, we do not have any NA values in the market_value column. We see that we have missing values in the 2022-2023_salary column. However, we will not be using that predictor in our analysis so we will disregard that.
As mentioned earlier, we know that we will not be making any trades. Therefore, we need to replace the positions of the five players that we just lost.
We need:
1 PG
1 SG
2 PFs
1 C

We see that we do not need a SF based on what we just waived. Let us filter out all of the Small Forwards in our data set.

```
no_sf_fa_df = subset(fa_df, pos != "SF")
```

Since we need five players and our cap space is $63.8 million, that puts our average salary per player at $12.76 million. That means we need to be strategic in maximizing our free agency plan.

For Plan A, we will play it more on the safe side and maximize the players we get based on what we think they are worth and what we can sign them for. Meaning we will not overspend to get any player based on our win share dollar value model. We will look at what players, by position, that have the biggest delta between what we think they are worth and what their market value is.

For Plan B, we will be a little more aggressive by overpaying on one player to make a big free agent splash, and then sign the remaining four players based on what we have left in our cap space.

Plan A

Let's filter out any players whose market values are worth more than what we think their win share dollar value will be and create a column that measures the delta between ws_dollar_val and market_value

```
plan_a_df = no_sf_fa_df[!no_sf_fa_df$market_value > no_sf_fa_df$ws_dollar_val, ]
plan_a_df$delta = (plan_a_df$ws_dollar_val - plan_a_df$market_value)
```

Now, let us write a code that maximizes both the market_value of a player and maximizes the delta that we created. This will code will recommend what players to sign for the best value based on what we think they are worth and what their market value is. This code will also choose two PFs.

```
cap_space = 63.8
selected_players = plan_a_df %>%
  filter(pos %in% c("PF", "PG", "SG", "C")) %>% # filter by positions
  arrange(desc(market_value), desc(delta)) %>% # arrange by market_value and delta
  group_by(pos) %>% # Group by positions
  top_n(ifelse(first(pos) == "PF", 2, 1), wt = delta + market_value) %>% # Returns 2 PFs
  ungroup() %>%
  filter(sum(market_value) <= cap_space) %>%  # Filter to ensure sum of total
  # market_value is within cap_space
  select(player, pos, market_value, ws_dollar_val, delta)
kable(selected_players)
```

player	pos	market_value	ws_dollar_val	delta
Josh Hart	SG	20.0	22.71182	2.711819
Harrison Barnes	PF	15.0	21.25852	6.258521
Mason Plumlee	C	12.5	27.94216	15.442156
Jevon Carter	PG	5.0	10.61694	5.616936
Keita Bates-Diop	PF	1.5	13.46253	11.962525

```
sum(selected_players$market_value)
```

```
## [1] 54
```

```
rem_cap_space = cap_space - sum(selected_players$market_value)
print(rem_cap_space)
```

```
## [1] 9.8
```

This code produced these five players. These five players maximize the market value and delta (what we think they are worth next year and what the market says they are worth) that we created. The sum of these players market values add up to $54 million. This leaves us with $9.8 million in cap space.
Let us assume we can sign one more player with the remaining cap space that we have. Since we initially did not sign an SF, let us look at top five SFs by delta to see who we can sign for the best value.

```
sf_plan_a_df = subset(fa_df, pos == "SF")
sf_plan_a_df = sf_plan_a_df[!sf_plan_a_df$market_value > sf_plan_a_df$ws_dollar_val, ]
sf_plan_a_df$delta = (sf_plan_a_df$ws_dollar_val - sf_plan_a_df$market_value)
```

```
top_5_SFs = sf_plan_a_df %>%
  filter(pos == "SF" & market_value < rem_cap_space) %>%
  arrange(desc(delta)) %>%
  slice_head(n = 5) %>%
  select(player, pos, market_value, ws_dollar_val, delta)
kable(top_5_SFs)
```

player	pos	market_value	ws_dollar_val	delta
Yuta Watanabe	SF	1.5	8.512753	7.012753
Troy Brown Jr.	SF	5.0	11.436868	6.436868
Torrey Craig	SF	7.0	12.884999	5.884999
Anthony Lamb	SF	1.5	7.379043	5.879043
Otto Porter Jr.	SF	1.5	6.139176	4.639176

Let us sign Yuta Watanabe at the $1.5 million market value, since he is the free agent with the highest delta of remaining SF free agents we can sign. This does leave us with surplus cap space of $8.3 million, but we are operating under the impression that we can only sign one more player and he is the best value based on our approach. This means we we can sign him for a good price based on what we expect of him.

```
rem_cap_space - 1.5
```

```
## [1] 8.3
```

Players signed in Plan A:

Josh Hart SG

Harrison Barnes PF

Mason Plumlee C

Jevon Carter PG

Keita Bates-Diop PF

Yuta Watanabe SF

Remaining Cap Space: $8.3 million.

Plan B

For Plan B, let us be a little more aggressive and go after one major player without ensuring that we are getting them at an expected value price. Let us add our delta column to fa_df set.

```
fa_df$delta = (fa_df$ws_dollar_val - fa_df$market_value)
```

Let us take a bigger chance in this plan and use a majority of our salary cap to sign one player. Let's look at top five players in each position by market value:

```
cap_space = 63.8
selected_players = fa_df %>%
  filter(market_value <= cap_space) %>%  # Filter by market value
  arrange(desc(market_value)) %>%  # Arrange by market value
  group_by(pos) %>%  # Group by positions
  slice_head(n = 5) %>%  # Select top 5 players within each position
  ungroup() %>%  # Ungroup the data
  select(player, pos, market_value, ws_dollar_val, delta)
kable(selected_players)
```

player	pos	market_value	ws_dollar_val	delta
Jakob Poeltl	C	25.0	22.980223	-2.0197769
Brook Lopez	C	20.0	24.436871	4.4368713
Christian Wood	C	17.5	17.074876	-0.4251241
Nikola Vučević	C	12.5	26.896766	14.3967662
Mason Plumlee	C	12.5	27.942156	15.4421557
Draymond Green	PF	25.0	16.064504	-8.9354961
Jerami Grant	PF	20.0	16.195362	-3.8046377
PJ Washington	PF	17.5	11.240416	-6.2595836
Harrison Barnes	PF	15.0	21.258521	6.2585207
Grant Williams	PF	15.0	13.148084	-1.8519159
Kyrie Irving	PG	46.5	25.284331	-21.2156689
Fred VanVleet	PG	40.0	22.357915	-17.6420849
D'Angelo Russell	PG	20.0	18.201414	-1.7985859
Tre Jones	PG	20.0	12.691451	-7.3085494
Coby White	PG	15.0	10.128376	-4.8716242
Cameron Johnson	SF	25.0	13.929889	-11.0701106
Khris Middleton	SF	25.0	2.764950	-22.2350503
Kyle Kuzma	SF	15.0	6.395784	-8.6042165
Kelly Oubre Jr.	SF	12.5	6.543614	-5.9563856
Jae Crowder	SF	12.5	5.185933	-7.3140670
James Harden	SG	50.0	26.554782	-23.4452182
Josh Hart	SG	20.0	22.711819	2.7118189
Austin Reaves	SG	20.0	18.959450	-1.0405503
Jordan Clarkson	SG	20.0	6.631948	-13.3680521
Bruce Brown	SG	15.0	14.872635	-0.1273646

Since this is our more aggressive plan, we are taking a chance and going after a player who may be worth way more than what we think he will be in 2023-2024 season. Looking at the available players in free agency by position, let us go ahead and sign Fred VanFleet at market value of $40 million. This fills our PG spot.

```
rem_cap_space = cap_space - 40
print(rem_cap_space)
```

```
## [1] 23.8
```

This leaves us with $23.8 million in cap space to fill our other four roster spots.
Since we are taking a chance on Fred VanFleet. Let us be slightly more conservative selecting our remaining four players by selecting them from the Plan A data set to ensure we are maximizing value of remaining four free agent spots.

```
selected_players = plan_a_df %>%
  filter(pos %in% c("PF", "SG", "C")) %>% # filter by positions
  arrange(desc(delta)) %>% # arrange by market_value and delta
  group_by(pos) %>% # Group by positions
  top_n(ifelse(first(pos) == "PF", 2, 1), wt = delta) %>% # Returns 2 PFs
  ungroup() %>%
  filter(sum(market_value) <= rem_cap_space) %>%  # Filter to ensure
  # total market_value is within cap_space
  select(player, pos, market_value, ws_dollar_val, delta)
print(selected_players)
```

```
## # A tibble: 4 x 5
##   player            pos   market_value ws_dollar_val delta
##   <chr>             <chr>        <dbl>         <dbl> <dbl>
## 1 Mason Plumlee     C             12.5          27.9 15.4
## 2 Keita Bates-Diop  PF             1.5          13.5 12.0
## 3 Derrick Jones Jr. PF             1.5          10.0  8.55
## 4 Josh Okogie       SG             5            11.6  6.57
```

```
kable(selected_players)
```

player	pos	market_value	ws_dollar_val	delta
Mason Plumlee	C	12.5	27.94216	15.442156
Keita Bates-Diop	PF	1.5	13.46253	11.962525
Derrick Jones Jr.	PF	1.5	10.04633	8.546334
Josh Okogie	SG	5.0	11.57408	6.574080

This code filtered through the free agent data set where the players estimated win share dollar value is greater than their market value. We will sign these four free agents.

```
rem_cap_space = rem_cap_space - sum(selected_players$market_value)
print(rem_cap_space)
```

```
## [1] 3.3
```

We are now left with $3.3 million in our cap space. Let us take the same approach that we took in Plan A and sign a SF (assuming we can sign one more player).

```
top_5_SFs = sf_plan_a_df %>%
  filter(pos == "SF" & market_value < rem_cap_space) %>%
  arrange(desc(delta)) %>%
  slice_head(n = 5) %>%
  select(player, pos, market_value, ws_dollar_val, delta)
kable(top_5_SFs)
```

player	pos	market_value	ws_dollar_val	delta
Yuta Watanabe	SF	1.5	8.512753	7.012753

player	pos	market_value	ws_dollar_val	delta
Anthony Lamb	SF	1.5	7.379043	5.879043
Otto Porter Jr.	SF	1.5	6.139176	4.639176
Joe Ingles	SF	1.5	5.476831	3.976831
Lamar Stevens	SF	1.5	4.606113	3.106113

Once again, Yuta Watanabe is the player that we can sign for the best value with the remaining salary cap that we have. We are left with a small surplus of $1.8 million in our cap.

```
rem_cap_space - 1.5
```

```
## [1] 1.8
```

Players signed in Plan B:

Fred VanFleet PG

Mason Plumlee C

Keita Bates-Diop PF

Derrick Jones Jr. PF

Josh Okogie SG

Yuta Watanabe SF

Remaining Cap Space: $1.8 million.

Part 4 Conclusion

Taking the more safe approach and making sure we signed every free agent for a good value in Part A, we were able to fill our necessary roster with a left over cap space of $8.3 million.
In our more aggressive approach in Plan B, we signed Kyrie Irving, who is projected to be worth less than what his market value is based on our model. We are taking a chance on him as he is a saavy veteran who has won an NBA Finals during his career. We were able to fill our remaining spots with free agents whose market values are less than what we predicted their worth to be next year. We were left with $1.8 million in our cap space in this plan.

Sources

https://www.basketball-reference.com/leagues/NBA_2021_advanced.html#advanced_stats

https://www.basketball-reference.com/leagues/NBA_2022_advanced.html

https://www.basketball-reference.com/leagues/NBA_2023_advanced.html

https://theathletic.com/3517502/2022/08/23/nba-analytics-win-cost/#

Partnow, S. (2022). *The Midrange Theory: Baseketball's Evolution In The Age Of Analytics*