Gambling on the NBA

Jess Kaminsky, Rishi Kowalski, Blain Morin, Derrick Yam

5/18/2018

Contents

Background

In 2015, Americans bet $149 billion illegally on sporting events. After the recent Supreme Court decision on sports gambling, it is no longer a federal crime to gamble on professional sports. Soon, individual states will pass bills legalizing sports gambling in their respective states. Thus, the $149 billion dark market will only grow. Although gambling is often regarded as a degenerate activity, many people have actually taken sports gambling on as their legitimate profession. Picking the outcome of a sporting event involves a lot of random variability. However, with the plethora of data available around sports, win probability models for individual competitions can be remarkably precise. In this paper we describe different methods to build win probability and expected points models for the National Basketball Association (NBA) to aid in gambling decisions.

The NBA is the third most lucrative professional sports league in the world, and is only growing. Due to its popularity, there is an abundance of publicly available data free of charge. As with most data analyses, big data is a curse and a blessing. Although more data means more information, it also means greater computational expense and with that the need for variable selection methods and advanced modeling. These are just some of the issues in this project; the bigger issue, gambling markets are hard to beat. There are publicly available win probability models on almost all major sports websites: ESPN, Fivethirtyeight and basketball-reference are just the beginning of them. Now in order to gain some kind of competitive advantage in the betting market, we have to be better than anything that is available to the public.

In NBA gambling, there are a multitude of ways to bet on the same game. The first and most obvious way is to bet on the money line. The money line means betting on one of the teams to win by any margin of points. Although presented slightly cryptically, a money line of -110 means you need to bet $110 to win $100, the negative money line represents the favorite to win the game. On the other hand a money line of +130 would mean you win $130 if you bet 100. These are essentially the odds the current market is giving for any particular game. A second way to bet on a game is to bet on the spread. The spread is a way of measuring how much better one team is expected to be than their competitor. For instance, if the Boston Celtics are playing the Cleveland Cavaliers and the spread for Boston is (-3), that means Boston is the favorite to win the game by 3 points. In terms of betting, if you bet on Boston with the spread, Boston must win by three or more points in order for you to "cover the spread" and win the bet. The payout on almost all bets with the spread is -110. The last way (that we will talk about) to bet on a game is betting the over-under for a game. The over-under is essentially how many total points both teams are expected to score. If you bet the over and the total points in that game is over the total points set before the game than you win that bet, typically as -110. For instance, if you bet the over when the over-under for Boston vs. Cleveland is 210

points and Boston scores 110 and Cleveland scores 105 points for a total points of 215 you would win that bet. All of these terms may be difficult to memorize at one time. Luckily, there are a ton of online resources for sports gamblers. The rest of this paper will be organized as follows, in the section Complications we will outline some of the complication in data collection, data cleaning, missing information and modeling. In the section Solution we will outline the steps we took to overcome these complications and will show our results. Finally, in the section Conclusion, we will explain some of the implications of our findings, limitations of our results, and future work.

Complications

The most time consuming task of this project was data collection and manipulation. The data was web scraped from sportsbookreviewsonline.com, donbest.com, and teamrankings.com. Each dataset had to be individually cleaned before merging them into the final dataset. There was not an obvious way that we could expedite this process. Our data collection and cleaning process was as follows:

1. Start with a data set of all the games.
2. Add the opening spread variable for that game.
3. Create 5 variables for potential injured players
    - We will use minutes played for each variable because it is most likely the most consistent throughout the season and we cannot use an average plus minus for previous seasons.
4. Join the games with the team stats from teamrankings.com
    - Join by the date
    - Join both the home teams stats and the away team stats.
5. Split the data into a training set and a test set.
    - Our test set will be a random sample of 20% of the games from each season.
    - Our training set will be the remaining 80% of games.
6. Write the data set to a csv and distribute to team members for analysis.

One problem we ran into when merging the data was that each had a unique naming convention for the teams. In order to be able to join our data based on team and season, we first had to create a matrix that mapped out all representations of a given team name based on data source. The task of merging the data was completed mostly using for loops, but could have been sped up using R commands like apply. Our final dataset - cleaned and ready for analysis - contained 15,359 observation of 146 variables.

We were fortunate to find a plethora of team level data to predict our outcomes; however, we were unable to find player level statistics that we were interested in. We hypothesized that it would be useful to include individual player statistics from up until each game - that is, summary statistics of a players performance for the entire season until the start of a given game. This data was not available to download or web scrape.

When building our models, time was not an issue as we were able to parallelize the computations. With 146 potential predictor variables, we needed to perform variable selection techniques and limit the number of predictors used in our final models. If we tried to perform machine learning techniques on all or a large subset of our predictors, our models would have taken a very long time to generate. The first time we generated our models, we accidentally overfit our models by including variables that inherently contained information about the outcome. We also had to retrain our models after realizing that we sampled our training and test data from the full data with replacement, therefore some observations appeared in both out training an test data sets. Some of these original models had logloss of 0.25 and accuracy greater than 0.95, which we thought might be unrealistic. After removing these terms and fixing our training and testing data, we obtained more reasonable estimates of these metrics.

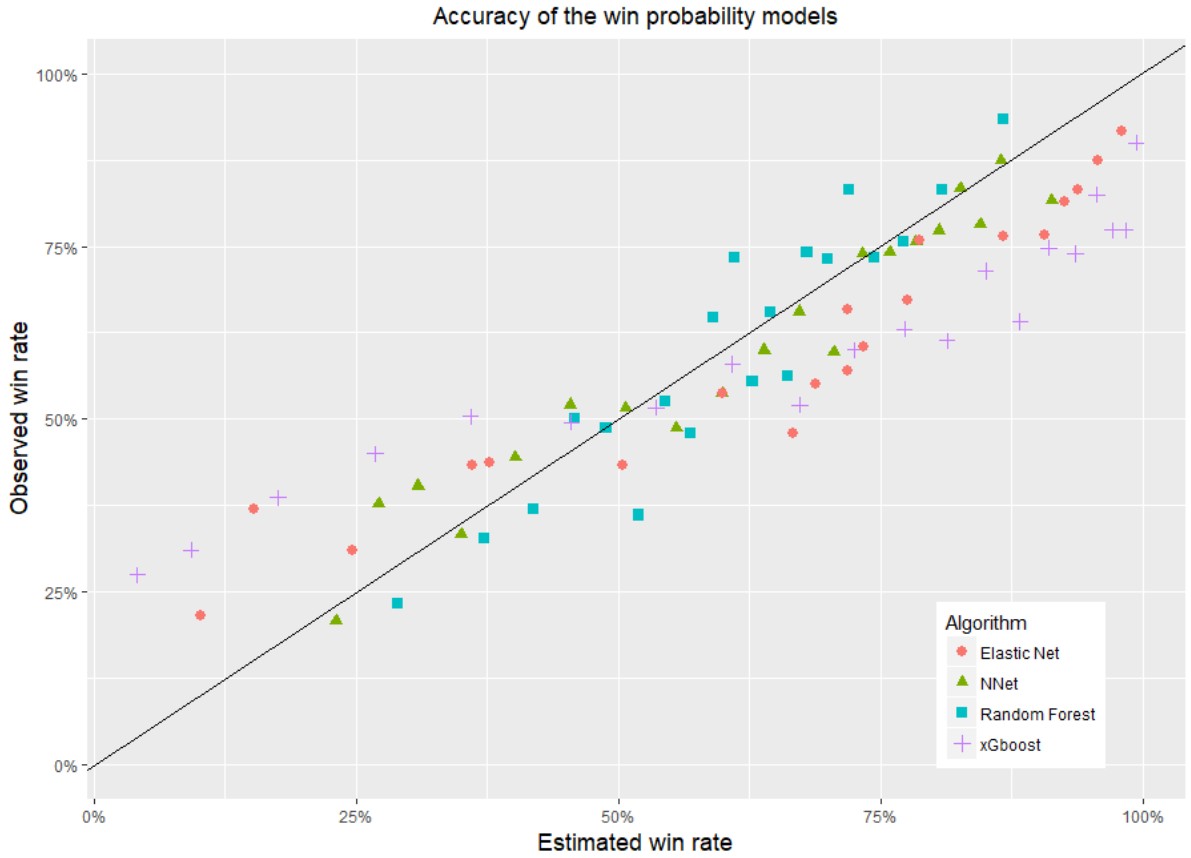
Solution

We used four machine learning methods to generate models that predict win probability for the home team, points scored by the home team, and points scored by the visitor team. The four methods we tried were Elastic Net (EN), Random Forest (RF), Extreme Gradient Boosting (xGB), and Neural Network (NNet. We evaluated the performance of the points models by their mean squared errors (MSE). The performance of the win probability models was evaluated by log loss. For the random forest and neural net we generated two models - one with all the variables and one with a subset of the variables selected by lasso. Based on the results, we chose to present the neural net model generated with all variables and the random forest model with the subset of lasso selected variables.

All 4 win probability models had approximately the same logistic loss value.

Model	Win Probability Log Loss
Elastic Net	0.4541
Extreme Gradient Boost	0.4597
Neural Net	0.4462
Random Forest Lasso	0.4526

To further assess the prediction ability of the models we will look at a calibration plot which shows whether the predicted probabilities agree with the observed probabilities. A model that accurately predictions the game outcome will have points that fall closely on the line which represents the line of perfect fit. From the plot, the neural net and random forest models appear to be the most accurate as their points fall closest to the line. The extreme gradient boost model almost always over or under predicts.

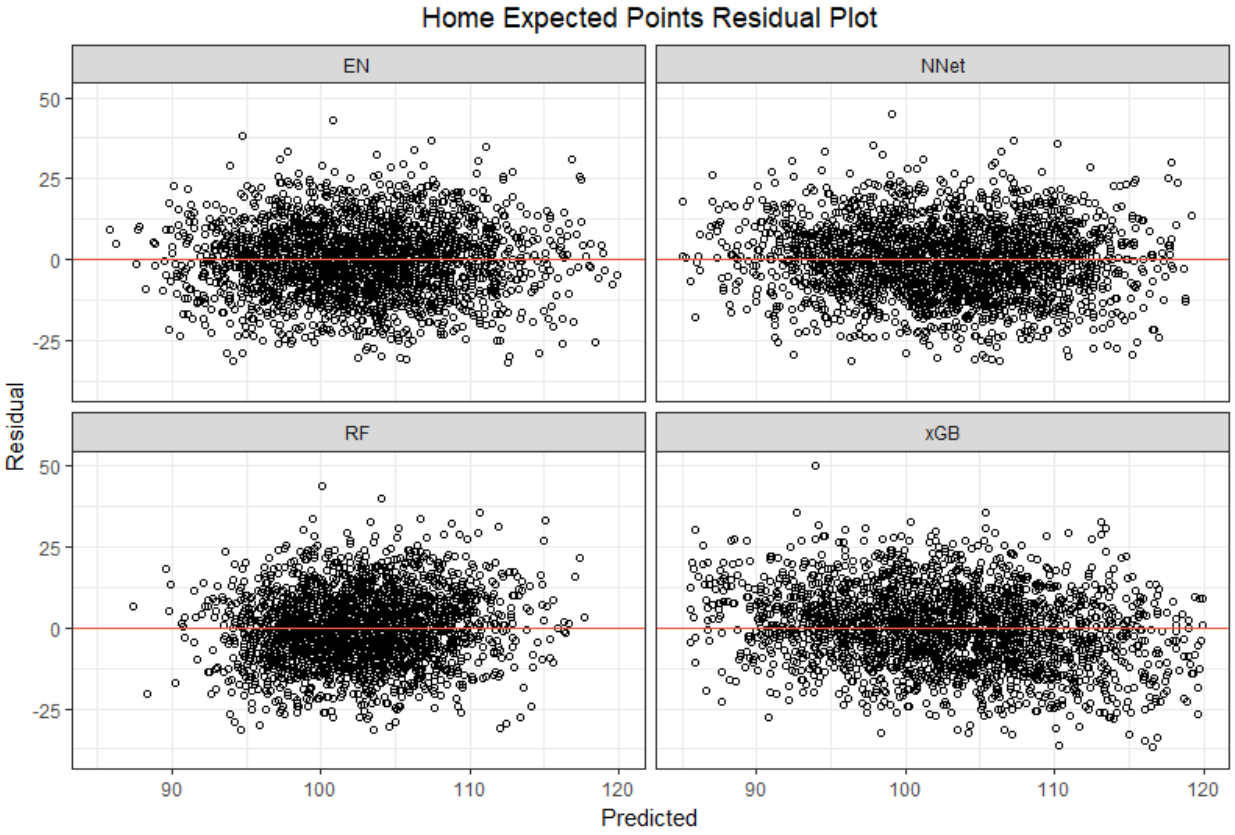


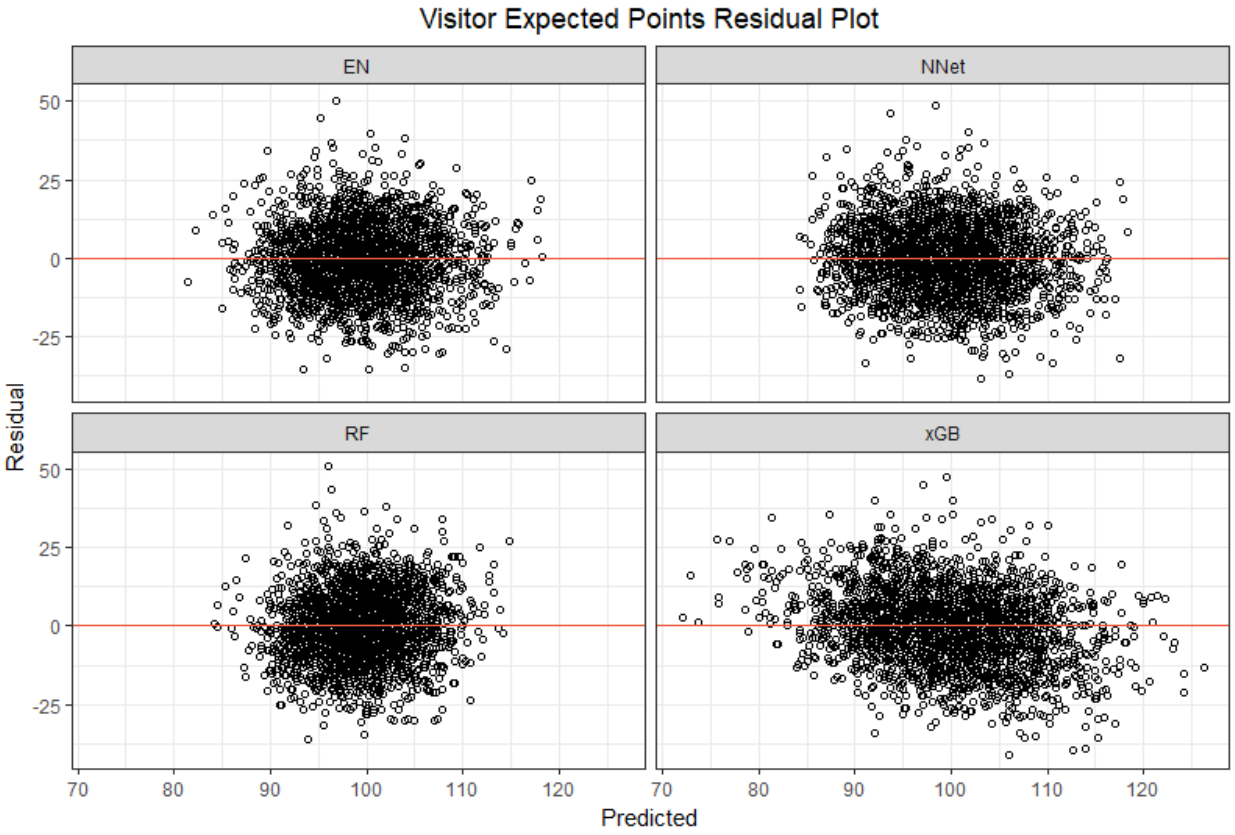
The elastic net performed best in terms of MSE in predicting the number of points scored by the home and

visiting teams; however was closely followed by neural net and random forest. Extreme gradient boosting was the worst model for predicting the continuous outcome, points scored.

Model	Expected Home Points MSE	Expected Visitor Points MSE
Elastic Net	118.62	124.81
Extreme Gradient Boost	142.14	146.07
Neural Net	120.1	127.8
Random Forest Lasso	120.41	156.89

To further illustrate the quality of these models we present their residual plots. The poor predictive ability of the extreme gradient boost model is emphasized in the plots below as it has the widest spread of points around the horizontal, y = 0.





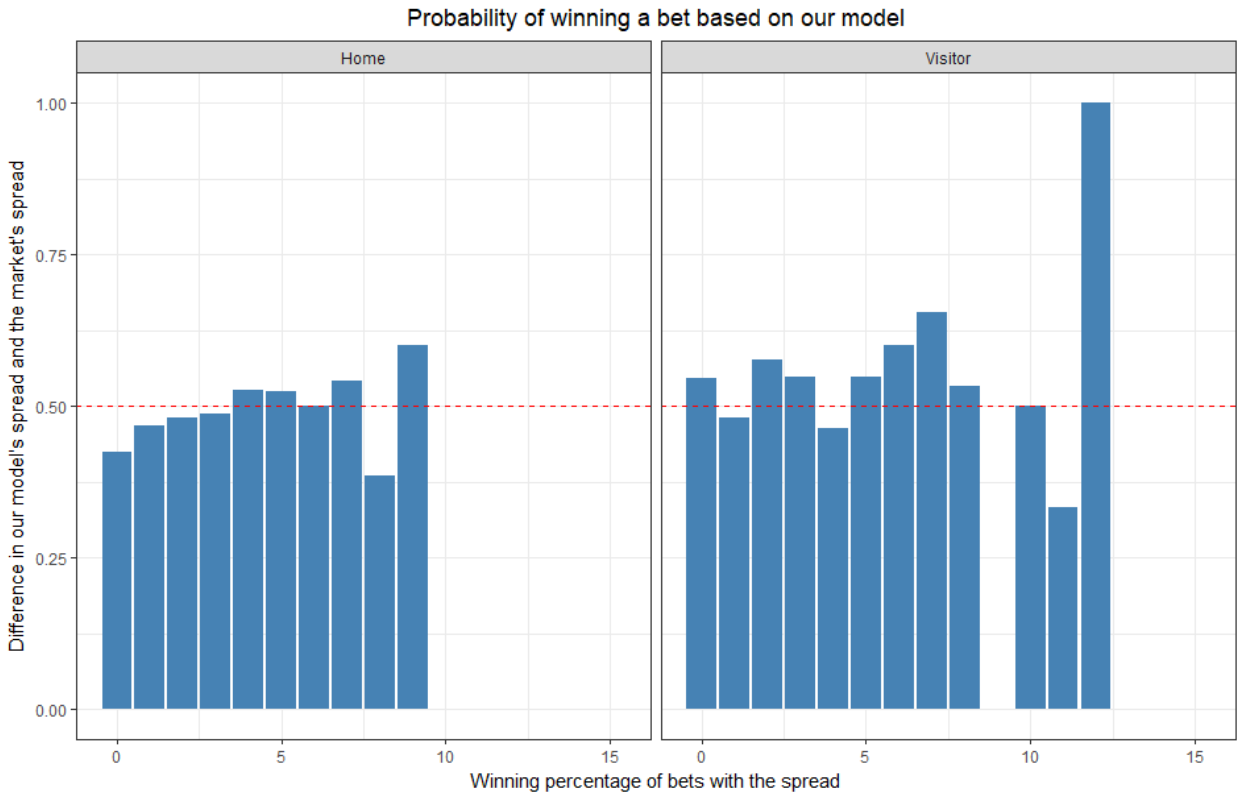
Conclusion

One potential use of our model is the development of a probabilistic betting strategy for NBA games. Given that our models assess teams' probabilities of winning games, comparisons between our model's outputs and betting lines is fairly straightforward, allowing us to develop a betting strategy such that we use our model to find games where the model's probability of a team winning is higher than the probability of that team winning based on the betting lines given to them, allowing us to place smarter bets on NBA outcomes.

Another method would be to compare the expected point differential from our expected home and visitor point models and test them against the observed point differential and market spread. We would bet on the visitor team if the expected point differential minus the market spread is greater than 0, otherwise we would bet on the home team. We first simply look at our results overall to see which proportion of these bets would be in the spread, as shown in the table below. From the table, we see that the proportion of winning bets is much higher for the visiting team.

Bet	Win	n
Home	0.4815	1082
Visitor	0.531	1305

To investigate this futher, we look at the distribution of winning bet probabilities based on difference in our expected spread and the market spread for home and visiting teams. This is shown in the plot below. From the model, we see substantially more winning bets when betting on the visiting team, confirming the numbers shown above. A conceptual explanation for this phenomenon is betting markets overly favor the home team unjustifiably. We think that our expected points models capture the true more accurately.



Some future directions for this project include testing a wider variety of feature selection and dimension reduction methods, potentially including the application of techniques such as principal components analysis (PCA) to reduce the dimension of our data or the development of a genetic algorithm to pick the most optimal subset of features with which we can predict game outcome. The incorporation of spatial tracking data into our model would provide us with a large amount of game-specific data with which we could use to further improve our model's performance.

Modeling sports outcomes without access to spatial data is often a challenging task, as tracking data for many sports is often proprietary and expensive to access. Without using tracking data, our best model performed well, and could indeed serve as the basis for a smart betting strategy.

Code Appendix

Data Cleaning and Web Scraping

```
##Steps:

1.) Start with a data set of all the games.
2.) Add the opening spread variable for that game.
3.) Create 5 variables for potential injured players
    - We will use minutes played for each variable because
    it is most likely the most consistent throughout the
    season and we cannot use an average plus minus for previous seasons.
4.) Join the games with the team stats from teamrankings.com
    - Join by the date
    - Join both the home teams stats and the away team stats.
5.) Split the data into a training set and a test set.
```

```
    - Our test set will be a random sample of 20% of the games
      from each season.
    - Our training set will be the remaining 80% of games.
6.) Write the data set to a csv.
7.) Train a randomForest model with the binary outcome being won.Home
8.) Check the variable importance
9.) Check the MSE and variance explained
10.) Check the calibration of the model by splitting into 20 bins.
11.) If the calibration is correct retrain the model on all of the data.

##Load Libraries

library(htmltab)
library(tidyr)
library(lubridate)
library(dplyr)
library(stringr)
library(XML)
library(tibble)

##1.) Start with a data set of all the games.

- Start with all of the games from
http://www.sportsbookreviewsonline.com/scoresoddsarchives/nba/nbaoddsarchives.htm

##2.) Add the opening spread variable for that game.

- Data cleaned in the Previous Results Cleaning RMD file.

- Use the schedule function from the NBA package in order to predict future games.

- We are going to pull the new data from DonBest.Com


setwd("C:/Users/dyam/Dropbox (Brown)/NBA Win Probability/Previous Results 20072018")
Games <- read.csv("NBAOdds20072018.csv")

Games <- Games %>%
  select(Date, Home, HFinal, Away, VFinal, OSpreadH, CSpreadH, Covered)

Games <- Games %>%
  mutate(Date = ymd(Date)) %>%
  rename(Visitor = Away)

##Join the teams with their abbreviations
Names <- read.csv("Different Franchise Names.csv")
head(Names)

Names <- Names %>% rename(Season = Year)
hNames <- Names %>% rename(Home = Franchise2018,
                           FranchiseCSH = FranchiseCS,
                           Abbreviation2018H = Abbreviation2018,
                           AbbreviationCSH = AbbreviationCS)
```

```
vNames <- Names %>% rename(Visitor = Franchise2018,
                           FranchiseCSV = FranchiseCS,
                           Abbreviation2018V = Abbreviation2018,
                           AbbreviationCSV = AbbreviationCS)

Games <- Games %>%
  mutate(Season = ifelse(Date < "2018-07-01" & Date > "2017-09-01", 2018,
    ifelse(Date < "2017-07-01" & Date > "2016-09-01", 2017,
           ifelse(Date < "2016-07-01" & Date > "2015-09-01", 2016,
                  ifelse(Date < "2015-07-01" & Date > "2014-09-01", 2015,
                         ifelse(Date < "2014-07-01" & Date > "2013-09-01", 2014,
    ifelse(Date < "2013-07-01" & Date > "2012-09-01", 2013,
           ifelse(Date < "2012-07-01" & Date > "2011-09-01", 2012,
                  ifelse(Date < "2018-07-01" & Date > "2010-09-01", 2011,
                         ifelse(Date < "2010-07-01" & Date > "2009-09-01", 2010,
    ifelse(Date < "2009-07-01" & Date > "2008-09-01", 2009,
           ifelse(Date < "2008-07-01" & Date > "2007-09-01", 2008,
                                NA)))))))))))
Games %>% filter(is.na(Season)) %>% count

Games <- left_join(Games, hNames)

Games <- left_join(Games, vNames)

##3.) Create 5 variables for potential injured players
    - We must use some kind of metric that weights the important of certain players on a certain team.
    - Since there is no data about the players performance up until a certain date in a season we canno
    - Therefore, we will use the players salary within that year weighted by the time past beginning of

#Games <- Games %>%
#  mutate(Injury1H = NA,
#         Injury2H = NA,
#         Injury3H = NA,
#         Injury4H = NA,
#         Injury5H = NA,
#         Injury1V = NA,
#         Injury2V = NA,
#         Injury3V = NA,
#         Injury4V = NA,
#         Injury5V = NA)

##Better source of injury data
#http://www.prosportstransactions.com/basketball/Search/SearchResults.
#php?Player=&Team=&BeginDate=2008-09-01&EndDate=2018-02-20&InjuriesChkBx=yes&Submit=Search

##From the server
Injuries <- read.csv("InjuryDataProSportsTrans.csv")

###Ran on the server

##Test One
#url <- "http://www.prosportstransactions.com/basketball/Search/SearchResults.
#php?Player=&Team=&BeginDate=2008-09-01&EndDate=2018-02-20&ILChkBx=yes&Submit=Search"
```

```
#
#Lines <- readLines(url)
#Lines.tib <- as_tibble(Lines)
#Lines.text <- Lines.tib[1:dim(Lines.tib)[1],]
#schedule.team <- htmltab(Lines.text[[1]], rm_nodata_cols = F)
#
###This website only allows data tables of length 25
###We need to create a vector of the page numbers
#TableLength <- vector(length = 590)
#TableLength[1] <- 0
#
#for(i in 2:length(TableLength)){
#  TableLength[i] <- TableLength[i-1] + 25
#}
#
#InjuryData <- tibble()
#
#
#for(i in 1:length(TableLength)){
#  url <- paste0("http://www.prosportstransactions.com/basketball/Search/SearchResults
#.php?Player=&Team=&BeginDate=2008-09-01&EndDate=2018-02-20&ILChkBx=yes&Submit=Search&start="
#, as.character(TableLength[i]))
#  tryCatch({
#      Lines <- readLines(url)
#      Lines.tib <- as_tibble(Lines)
#      Lines.text <- Lines.tib[1:dim(Lines.tib)[1],]
#      InjuryDatePage <- htmltab(Lines.text[[1]], rm_nodata_cols = F)
#      InjuryData <- bind_rows(InjuryData, InjuryDatePage)
#    }, error = function(e){
#      InjuryData <- InjuryData
#    })
#}
#
#write.csv(InjuryData, "InjuryDataProSportsTrans.csv")

##I think I want to find the inactives using ProSportsTrans
##This website only allows data tables of length 25
##We need to create a vector of the page numbers
#TableLength <- vector(length = 600)
#TableLength[1] <- 0
#
#for(i in 2:length(TableLength)){
#  TableLength[i] <- TableLength[i-1] + 25
#}
#
#InjuryMissed <- tibble()
#for(i in 1:length(TableLength)){
#  url <- paste0("http://www.prosportstransactions.com/basketball/Search/SearchResults
#.ph#p?Player=&Team=&BeginDate=2007-10-01&EndDate=2018-02-21&InjuriesChkBx=yes&PersonalChkBx
#=#yes&Submit=Search&start=",
#                as.character(TableLength[i]))
#  tryCatch({
#    Lines <- readLines(url)
```

```
#    Lines.tib <- as_tibble(Lines)
#    Lines.text <- Lines.tib[1:dim(Lines.tib)[1],]
#    InjuryDatePage <- htmltab(Lines.text[[1]], rm_nodata_cols = F)
#    InjuryMissed <- bind_rows(InjuryMissed, InjuryDatePage)
#    }, error = function(e){
#    InjuryMissed <- InjuryMissed
#    })
#}
#
#write.csv(InjuryMissed, file = "InjuryMissedProSports.csv")

##Clean the Injury Data

##I dont think we need the injuries data files, just the inactives
Inactives <- read.csv("InjuryMissedProSports.csv") ##Players Missing Games

head(Inactives)
Inactives <- Inactives %>%
  select(-X)
head(Inactives)
names(Inactives) <- c("Date", "Team", "Acquired", "Relinquished", "Notes")
Inactives$Acquired <- str_extract(Inactives$Acquired,
                    "[:upper:][:alpha:]+[:blank:][:upper:][:alpha:]+")
Inactives$Relinquished <- str_extract(Inactives$Relinquished,
                    "[:upper:][:alpha:]+[:blank:][:upper:][:alpha:]+")

Inactives$Date <- ymd(Inactives$Date)
Inactives <- Inactives %>%
mutate(Season = ifelse(Date < "2018-07-01" & Date > "2017-09-01", 2018,
      ifelse(Date < "2017-07-01" & Date > "2016-09-01", 2017,
             ifelse(Date < "2016-07-01" & Date > "2015-09-01", 2016,
                    ifelse(Date < "2015-07-01" & Date > "2014-09-01", 2015,
                           ifelse(Date < "2014-07-01" & Date > "2013-09-01", 2014,
      ifelse(Date < "2013-07-01" & Date > "2012-09-01", 2013,
             ifelse(Date < "2012-07-01" & Date > "2011-09-01", 2012,
                    ifelse(Date < "2018-07-01" & Date > "2010-09-01", 2011,
                           ifelse(Date < "2010-07-01" & Date > "2009-09-01", 2010,
      ifelse(Date < "2009-07-01" & Date > "2008-09-01", 2009,
             ifelse(Date < "2008-07-01" & Date > "2007-09-01", 2008,
                                  NA))))))))))))
unique(Inactives$Team)

head(Names)
NamesInj <- Names %>%
  rename(Team = NickNameCS)

NamesInj$Team <- as.character(NamesInj$Team)
NamesInj$NickNameCS <- as.character(NamesInj$FranchiseCS)

Inactives$Team <- as.character(Inactives$Team)

Inactives <- left_join(Inactives, NamesInj)
```

```
Inactives <- Inactives %>%
  dplyr::mutate(Returning = ifelse(is.na(Relinquished), 1, 0))

Inactives <- Inactives %>%
  dplyr::mutate(Player = ifelse(Returning == 1, Acquired, Relinquished))

head(Inactives)

##I dont think we need the injuries data files, just the inactives
Inactives <- Inactives %>%
  select(Date, Player, everything()) %>%
  select(-Relinquished, -Acquired) %>%
  arrange(Player)

###Load Salary Data
Salaries <- read.csv("SalaryDataESPN.csv")
head(Salaries)
Salaries <- Salaries %>%
  select(-X) %>%
  rename(Player = NAME,
         FranchiseCS = TEAM,
         Salary = SALARY)

head(Salaries)
Salaries$Player <- gsub("'", "", Salaries$Player)
Salaries$Player <- str_extract(Salaries$Player,
                 "[:upper:][:alpha:]+[:blank:][:upper:][:alpha:]+")
Salaries$Salary <- gsub("\\$", "", Salaries$Salary)
Salaries$Salary <- gsub(",", "", Salaries$Salary)
head(Salaries)

##Create a team importance metric = your salary as a proportion of the teams average
Salaries$Salary <- as.numeric(as.character(Salaries$Salary))
Salaries <- Salaries %>% filter(!is.na(Salary))

Salaries <- Salaries %>%
  group_by(FranchiseCS, Season) %>%
  mutate(TeamAverage = mean(Salary)) %>%
  mutate(PlayerImportance = Salary/TeamAverage) %>%
  mutate(StdImp = (PlayerImportance -
  mean(PlayerImportance))/sd(PlayerImportance)) ##Standardized Throughout the league.

head(Salaries)

##Join with the inactives data

head(Inactives)
head(Salaries)

Inactives <-left_join(Inactives, Salaries)
write.csv(Inactives, file = "InactivesCleanedToJoin.csv")

###Dates
```

```
#Date2008 <- as.character(seq(as.Date("2007-10-01"), as.Date("2008-07-01"), by="days"))
#Date2009 <- as.character(seq(as.Date("2008-10-01"), as.Date("2009-07-01"), by="days"))
#Date2010 <- as.character(seq(as.Date("2009-10-01"), as.Date("2010-07-01"), by="days"))
#Date2011 <- as.character(seq(as.Date("2010-10-01"), as.Date("2011-07-01"), by="days"))
#Date2012 <- as.character(seq(as.Date("2011-10-01"), as.Date("2012-07-01"), by="days"))
#Date2013 <- as.character(seq(as.Date("2012-10-01"), as.Date("2013-07-01"), by="days"))
#Date2014 <- as.character(seq(as.Date("2013-10-01"), as.Date("2014-07-01"), by="days"))
#Date2015 <- as.character(seq(as.Date("2014-10-01"), as.Date("2015-07-01"), by="days"))
#Date2016 <- as.character(seq(as.Date("2015-10-01"), as.Date("2016-07-01"), by="days"))
#Date2017 <- as.character(seq(as.Date("2016-10-01"), as.Date("2017-07-01"), by="days"))
#Date2018 <- as.character(seq(as.Date("2017-10-01"), as.Date("2018-07-01"), by="days"))
#
#DatesData <- c(Date2008, Date2009, Date2010, Date2011, Date2012, Date2013,
#               Date2014, Date2015, Date2016, Date2017, Date2018)
#
#URLData <- read.csv("Links to stats to scrape.csv")
#head(URLData)
#URLData$URL <- paste0(URLData$URL, "?date=")
#
###Test One
#    url2 <- "https://www.teamrankings.com/nba/stat/offensive-efficiency?date=2018-02-06"
#    Lines <- readLines(url2)
#    Lines.tib <- as_tibble(Lines)
#    Lines.text2 <- Lines.tib[1:dim(Lines.tib)[1],]
#    schedule.team <- htmltab(Lines.text2[[1]], rm_nodata_cols = F)
#    names(schedule.team) <- c("Rank", "Team", paste0("YTD.", URLData$Stat[1]),
#                              paste0("L3.", URLData$Stat[1]),
```

```
#                               paste0("L1.", URLData$Stat[1]),
#                               paste0("Home.", URLData$Stat[1]),
#                               paste0("Away.", URLData$Stat[1]),
#                               paste0("LY.", URLData$Stat[1]))
#schedule.team <- schedule.team %>% mutate(Date = "2018-02-06")
#
#schedule.team <- schedule.team %>% select(Date, Team)
#df.tib <- tibble()
##DatesData <- DatesData[1:3]
##URLData <- URLData[1:3,]
#datetibble <- schedule.team
#
#for(i in 1:length(DatesData)){
#  Date1 <- as.character(DatesData[i])
#  datetibble <- as_tibble(schedule.team) %>%
#    mutate(Date = Date1)
#  for(j in 1:length(URLData$URL)){
#    url <- paste0(URLData$URL[j], Date1)
#    Lines <- readLines(url)
#    Lines.tib <- as_tibble(Lines)
#    Lines.text2 <- Lines.tib[1:dim(Lines.tib)[1],]
#    VariableTable <- htmltab(Lines.text2[[1]], rm_nodata_cols = F)
#    names(VariableTable) <- c("Rank", "Team",
#                               paste0("YTD.", URLData$Stat[j]),
#                               paste0("L3.", URLData$Stat[j]),
#                               paste0("L1.", URLData$Stat[j]),
#                               paste0("Home.", URLData$Stat[j]),
```

```
#                               paste0("Away.", URLData$Stat[j]),
#                               paste0("LY.", URLData$Stat[j]))
#
#    VariableTable <- VariableTable %>%
#      mutate(Date = Date1)
#    VariableTable <- VariableTable %>%
#      dplyr::select(-Rank)
#    datetibble <- full_join(datetibble, VariableTable,
#                            by = c("Team", "Date"))
#  }
#  df.tib <- bind_rows(df.tib, datetibble)
#}
#
#write.csv(df.tib, "Statistics20082018.csv")
#
####Find the player salary from espn
###Test One
#url <- "http://www.espn.com/nba/salaries/_/year/2011/page/1"
#
#Lines <- readLines(url)
#Lines.tib <- as_tibble(Lines)
#Lines.text <- Lines.tib[1:dim(Lines.tib)[1],]
#schedule.team <- htmltab(Lines.text[[1]], rm_nodata_cols = F)
#
###This website only allows data tables of length 25
###We need to create a vector of the page numbers
#TableLength <- vector(length = 15)
#TableLength[1] <- 1
#
#for(i in 2:length(TableLength)){
#  TableLength[i] <- TableLength[i-1] + 1
#}
#
#Years <- as.character(seq(2008, 2018, 1))
#SalaryData <- tibble()
#
#for(i in 1:length(Years)){
#  year <- paste(Years[i])
```

```
#  urlyear <- paste0("http://www.espn.com/nba/salaries/_/year/", year, "/page/")
#  for(j in 1:length(TableLength)){
#    url <- paste0(urlyear, as.character(TableLength[j]))
#  tryCatch({
#    Lines <- readLines(url)
#    Lines.tib <- as_tibble(Lines)
#    Lines.text <- Lines.tib[1:dim(Lines.tib)[1],]
#    SalaryDatePage <- htmltab(Lines.text[[1]], rm_nodata_cols = F)
#    SalaryDatePage <- SalaryDatePage %>%
#      mutate(Season = paste(year))
#    SalaryData <- bind_rows(SalaryData, SalaryDatePage)
#    }, error = function(e){
#    SalaryData <- SalaryData
#    })
#  }
#
#}
#
#
#write.csv(SalaryData, "SalaryDataESPN.csv")
#
#
#4.) Join the games with the team stats from teamrankings.com
    - Join by the date
    - Join both the home teams stats and the away team stats.

library(readr)
TeamStats <- read_csv("Statistics20082018.csv")
head(TeamStats)

s1 <- "Date"
s2 <- "Team"
s3 <- "L3"
s4 <- "Home"
s5 <- "Away"
s6 <- "LY"
TeamStat <- TeamStats %>%
  select(contains(s1), contains(s2), contains(s3),
       contains(s4), contains(s5), contains(s6))

TeamStat <- TeamStat %>%
  mutate_all(funs(gsub("%", "", .)))

summary(TeamStat)

TeamStat <- TeamStat[, 3:dim(TeamStat)[2]]
TeamStat <- TeamStat %>%
  mutate_all(funs(as.numeric(as.character(.))))
TeamStat <- bind_cols(TeamStats[,2:3], TeamStat)

##Missing variables in the beginning fof the season
URLData <- read.csv("Links to stats to scrape.csv")
head(URLData)
```

```
Missing <- filter(TeamStat, is.na(Away.Blockspg))

##Impute the missing data as the previous years information
for (i in 1:length(URLData$Stat)){
  l <- paste(URLData$Stat[i])
  Missing[, paste0("Home.", l)] <- Missing[,paste0("LY.", l)]
  Missing[, paste0("Away.", l)] <- Missing[,paste0("LY.", l)]
  Missing[, paste0("L3.", l)] <- Missing[,paste0("LY.", l)]
}

NonMissing <- filter(TeamStat, !is.na(Away.Blockspg))

TeamStat <- bind_rows(Missing, NonMissing)

##Remove the Fbeff variables as they are missing
TeamStat <- TeamStat %>%
  select(-contains("Fbeff"))

summary(TeamStat)
dim(TeamStat)

##Drop the observations missing values
TeamStat <- drop_na(TeamStat)
dim(TeamStat)

##Add the season variable
TeamStat <- TeamStat %>%
  mutate(Season = ifelse(Date < "2018-07-01" & Date > "2017-09-01", 2018,
  ifelse(Date < "2017-07-01" & Date > "2016-09-01", 2017,
         ifelse(Date < "2016-07-01" & Date > "2015-09-01", 2016,
                ifelse(Date < "2015-07-01" & Date > "2014-09-01", 2015,
                       ifelse(Date < "2014-07-01" & Date > "2013-09-01", 2014,
  ifelse(Date < "2013-07-01" & Date > "2012-09-01", 2013,
         ifelse(Date < "2012-07-01" & Date > "2011-09-01", 2012,
                ifelse(Date < "2018-07-01" & Date > "2010-09-01", 2011,
                       ifelse(Date < "2010-07-01" & Date > "2009-09-01", 2010,
  ifelse(Date < "2009-07-01" & Date > "2008-09-01", 2009,
         ifelse(Date < "2008-07-01" & Date > "2007-09-01", 2008,
                                NA)))))))))))

##Need to change the Team Stat names to be the standard names
head(Names)
unique(TeamStat$Team)
TeamStat <- TeamStat %>% rename(TeamRankingsName = Team)
##We need the same variable for the home and the away team.
HomeStats <- TeamStat %>%
  select(-contains("LY."), -contains("Away."))
dim(HomeStats)

VisitorStats <- TeamStat %>%
  select(-contains("LY."), -contains("Home."))
dim(VisitorStats)
```

```
##Add a suffix for each so we know what variable it represents.
for(i in 1:length(names(HomeStats))){
  names(HomeStats)[i] <- paste0(names(HomeStats)[i], ".H")
}

HomeStats <- HomeStats %>%
  rename(TeamRankingsName = TeamRankingsName.H) %>%
  rename(Season = Season.H) %>%
  rename(Date = Date.H)

HomeStats <- left_join(HomeStats, Names, by = c("Season", "TeamRankingsName"))

HomeStats <- HomeStats %>%
  select(-Franchise2018, -TeamRankingsName,
         -NickNameCS, - Abbreviation2018, -AbbreviationCS)

HomeStats <- HomeStats %>%
  rename(Home = FranchiseCS)

for(i in 1:length(names(VisitorStats))){
  names(VisitorStats)[i] <- paste0(names(VisitorStats)[i], ".V")
}
VisitorStats <- VisitorStats %>%
  rename(TeamRankingsName = TeamRankingsName.V) %>%
  rename(Season = Season.V) %>%
  rename(Date = Date.V)

VisitorStats <- left_join(VisitorStats, Names)

VisitorStats <- VisitorStats %>%
  select(-Franchise2018, -TeamRankingsName,
         -NickNameCS, - Abbreviation2018, -AbbreviationCS)

VisitorStats <- VisitorStats %>%
  rename(Visitor = FranchiseCS)
VisitorStats <- VisitorStats %>%
  group_by(Date, Visitor) %>%
  slice(1)
HomeStats <- HomeStats %>%
  group_by(Date, Home) %>%
  slice(1)

dim(Games)
Games$Home <- as.character(Games$Home)
Games$Visitor <- as.character(Games$Visitor)

Games <- Games %>%
  select(-FranchiseCSH, -TeamRankingsName,
         -NickNameCS, - Abbreviation2018H, -AbbreviationCSH,
         -FranchiseCSV,  - Abbreviation2018V, -AbbreviationCSV)
Games <- left_join(Games, HomeStats) %>%
  left_join(VisitorStats)
```

```
dim(Games)
Games <- drop_na(Games)
dim(Games)
##We lose <200 observations by dropping the NAs

##Create a days from the start of the season variable

Games <- Games %>%
  group_by(Season) %>%
  mutate(Day = Date - min(Date) +1)

dim(Games)

##Create a won.home variable
Games <- Games %>%
  ungroup() %>%
  mutate(Won.Home = ifelse(HFinal > VFinal, 1, 0))
#write.csv(Games, file = "GamesStatsJoined.csv")

##Still need to join the injury data
setwd("C:/Users/dyam/Dropbox (Brown)/NBA Win Probability/Previous Results 20072018")
Games <- read.csv("GamesStatsJoined.csv")
Inactives <- read.csv("InactivesCleanedToJoin.csv")
head(Inactives)
Inactives <- Inactives %>%
  select(Date, Player, FranchiseCS, Notes, Season, Returning,
         Salary, TeamAverage, PlayerImportance, StdImp)
head(Inactives)

Games <- Games %>%
  mutate(Injury1H = NA,
         Injury2H = NA,
         Injury3H = NA,
         Injury4H = NA,
         Injury5H = NA,
         Injury1V = NA,
         Injury2V = NA,
         Injury3V = NA,
         Injury4V = NA,
         Injury5V = NA)

Inactives$Date <- ymd(Inactives$Date)
Inactives$FranchiseCS <- as.character(Inactives$FranchiseCS)
Games$Date <- ymd(Games$Date)

Games$Home <- as.character(Games$Home)
Games$Visitor <- as.character(Games$Visitor)

Games <- Games %>%
  ungroup()
Inactives <- Inactives %>%
  ungroup
```

```
Inactives <- Inactives %>%
  mutate(PID = seq(1, dim(Inactives)[1], 1))

##Home Variables
for(i in 1:nrow(Games)){
  tempin <- Inactives %>%
    rename(Home = FranchiseCS) %>%
    filter(Date > Games$Date[i] - 15 & Date <= Games$Date[i]) %>%
    filter(Home == Games$Home[i])
  tempPID <- tempin %>%
    filter(Returning == 1)
  tempin <- filter(tempin, !Player %in% tempPID$Player)
  tempin <- tempin %>%
    group_by(Player) %>%
    arrange(Date) %>%
    mutate(PlayerImportance = max(PlayerImportance, na.rm = T)) %>%
    mutate(PlayerImportance = ifelse(PlayerImportance == -Inf, 0,
                                     PlayerImportance)) %>%
    slice(1)
  if(dim(tempin)[1] > 5){
    tempin <- tempin %>%
      arrange(PlayerImportance) %>%
      slice(5)
  } else {
    tempin <- tempin
  }
  if(dim(tempin)[1] == 1){
    Games$Injury1H[i] <- tempin$PlayerImportance[1]
    Games$Injury3H[i] <- 0
    Games$Injury4H[i] <- 0
    Games$Injury5H[i] <- 0
    Games$Injury2H[i] <- 0
  } else if(dim(tempin)[1] == 2){
    Games$Injury1H[i] <- tempin$PlayerImportance[1]
    Games$Injury2H[i] <- tempin$PlayerImportance[2]
    Games$Injury3H[i] <- 0
    Games$Injury4H[i] <- 0
    Games$Injury5H[i] <- 0
  } else if(dim(tempin)[1] == 3){
    Games$Injury3H[i] <- tempin$PlayerImportance[3]
    Games$Injury1H[i] <- tempin$PlayerImportance[1]
    Games$Injury2H[i] <- tempin$PlayerImportance[2]
    Games$Injury4H[i] <- 0
    Games$Injury5H[i] <- 0
  } else if(dim(tempin)[1] == 4){
    Games$Injury4H[i] <- tempin$PlayerImportance[4]
    Games$Injury3H[i] <- tempin$PlayerImportance[3]
    Games$Injury1H[i] <- tempin$PlayerImportance[1]
    Games$Injury2H[i] <- tempin$PlayerImportance[2]
    Games$Injury5H[i] <- 0
  } else if(dim(tempin)[1] ==5){
    Games$Injury5H[i] <- tempin$PlayerImportance[5]
    Games$Injury4H[i] <- tempin$PlayerImportance[4]
```

```
    Games$Injury3H[i] <- tempin$PlayerImportance[3]
    Games$Injury1H[i] <- tempin$PlayerImportance[1]
    Games$Injury2H[i] <- tempin$PlayerImportance[2]

  } else {
    Games$Injury5H[i] <- 0
    Games$Injury4H[i] <- 0
    Games$Injury3H[i] <- 0
    Games$Injury1H[i] <- 0
    Games$Injury2H[i] <- 0
  }

}


#Visitor Variables
for(i in 1:nrow(Games)){
  tempin <- Inactives %>%
    rename(Visitor = FranchiseCS) %>%
    filter(Date > Games$Date[i] - 15 & Date <= Games$Date[i]) %>%
    filter(Visitor == Games$Visitor[i])
  tempPID <- tempin %>%
    filter(Returning == 1)
  tempin <- filter(tempin, !Player %in% tempPID$Player)
  tempin <- tempin %>%
    group_by(Player) %>%
    arrange(Date) %>%
    mutate(PlayerImportance = max(PlayerImportance, na.rm = T)) %>%
    mutate(PlayerImportance = ifelse(PlayerImportance == -Inf, 0,
                                     PlayerImportance)) %>%
    slice(1)
  if(dim(tempin)[1] > 5){
    tempin <- tempin %>%
      arrange(PlayerImportance) %>%
      slice(5)
  } else {
    tempin <- tempin
  }
  if(dim(tempin)[1] == 1){
    Games$Injury1V[i] <- tempin$PlayerImportance[1]
    Games$Injury3V[i] <- 0
    Games$Injury4V[i] <- 0
    Games$Injury5V[i] <- 0
    Games$Injury2V[i] <- 0
  } else if(dim(tempin)[1] == 2){
    Games$Injury1V[i] <- tempin$PlayerImportance[1]
    Games$Injury2V[i] <- tempin$PlayerImportance[2]
    Games$Injury3V[i] <- 0
    Games$Injury4V[i] <- 0
    Games$Injury5V[i] <- 0
  } else if(dim(tempin)[1] == 3){
    Games$Injury3V[i] <- tempin$PlayerImportance[3]
    Games$Injury1V[i] <- tempin$PlayerImportance[1]
```

```
    Games$Injury2V[i] <- tempin$PlayerImportance[2]
    Games$Injury4V[i] <- 0
    Games$Injury5V[i] <- 0
  } else if(dim(tempin)[1] == 4){
    Games$Injury4V[i] <- tempin$PlayerImportance[4]
    Games$Injury3V[i] <- tempin$PlayerImportance[3]
    Games$Injury1V[i] <- tempin$PlayerImportance[1]
    Games$Injury2V[i] <- tempin$PlayerImportance[2]
    Games$Injury5V[i] <- 0
  } else if(dim(tempin)[1] ==5){
    Games$Injury5V[i] <- tempin$PlayerImportance[5]
    Games$Injury4V[i] <- tempin$PlayerImportance[4]
    Games$Injury3V[i] <- tempin$PlayerImportance[3]
    Games$Injury1V[i] <- tempin$PlayerImportance[1]
    Games$Injury2V[i] <- tempin$PlayerImportance[2]

  } else {
    Games$Injury5V[i] <- 0
    Games$Injury4V[i] <- 0
    Games$Injury3V[i] <- 0
    Games$Injury1V[i] <- 0
    Games$Injury2V[i] <- 0
  }
}

Games <- Games %>%
  select(-X)

#write.csv(Games, "GamesStatsInj.csv")
Games2 <- Games %>%
  mutate(Inj.H = Injury1H + Injury2H + Injury3H +
           Injury4H + Injury5H,
         Inj.V = Injury1V + Injury2V + Injury3V +
           Injury4V + Injury5V)
Games2 <- Games2 %>%
  select(-Injury1H, -Injury2H, -Injury3H, -Injury4H, -Injury5H,
         -Injury1V, -Injury2V, -Injury3V, -Injury4V, -Injury5V)

##Create a game id variable
Games2 <- Games2 %>%
  mutate(GID = seq(1:dim(Games2)[1]))

##Add in the distance traveled, time zone changes, time between games,

setwd()
head(Names)
Names <- Names %>%
  mutate(City = as.character(TeamRankingsName))

Names <- Names %>%
  mutate(City = ifelse(City == "LA Clippers", "Los Angeles", City),
         City = ifelse(City == "LA Lakers", "Los Angeles", City),
         City = ifelse(City == "Indiana", "Indianapolis", City),
```

```
         City = ifelse(City == "Golden State", "San Francisco", City),
         City = ifelse(City == "Okla City", "Oklahoma City", City),
         City = ifelse(City == "Utah", "Salt Lake City", City),
         City = ifelse(City == "Minnesota", "Minneapolis", City))

#read in geographic data
latlong <- read.csv("sports.csv", stringsAsFactors = FALSE)

#subset to nba
latlong <- latlong[latlong$sport == "NBA", ]

#subset down to teammame
nbageo <- data.frame(latlong$team_name, latlong$lat, latlong$lon)
names(nbageo) <- c("team_name", "lat", "lon")

#read city information
cityinfo <- read.csv("CitiesEnriched.csv", stringsAsFactors = FALSE)
head(cityinfo)
citinfo <- cityinfo %>%
  select(City, Lat, Lng, ElevationMeters, UTCOffsetHours) %>%
  mutate(UTCOffsetHours = UTCOffsetHours - 1)

TorontoInfo <- latlong %>%
  filter(team_name == "Toronto Raptors") %>%
  select(Lat = lat, Lng = lon, City = team_name) %>%
  mutate(ElevationMeters = 75.89,
         UTCOffsetHours = -5,
         City = "Toronto")

citinfo <- bind_rows(citinfo, TorontoInfo)

NamesCity <- left_join(Names, citinfo)

##Add in the stadium long and lat
nbageo <- nbageo %>% rename(Franchise2018 =
                team_name, latStadium = lat, lonStadium = lon)

NamesCity <- left_join(NamesCity, nbageo)

##Using the fields package
#install.packages("fields")
library(fields)

teams <- unique(Games2$Home)

dist.df <- temp.dist

for(i in teams){
  l <- i

temp.dist <- Games2 %>%
  filter(Home == l | Visitor == l) %>%
  arrange(Date) %>%
```

```
  select(GID, Date, Season, Home, Visitor)

temp.city <- NamesCity %>%
  select(Home = Franchise2018,
         Lat, Lng, ElevationMeters, Season, UTCOffsetHours)

temp.dist <- left_join(temp.dist, temp.city)

temp.city <- NamesCity %>%
  select(Visitor = Franchise2018,
         Lat.V = Lat, Lng.V = Lng, Elevation.V = ElevationMeters, Season)

temp.dist <- left_join(temp.dist, temp.city)

temp.dist <- temp.dist %>%
  arrange(Date) %>%
  mutate(PrevLat = lag(Lat, 1),
         PrevLng = lag(Lng, 1))

mat.1 <- as.matrix(cbind(temp.dist$Lng, temp.dist$Lat))
mat.2 <- as.matrix(cbind(temp.dist$PrevLng, temp.dist$PrevLat))

dis.mat <- rdist.earth.vec(mat.1, mat.2)

temp.dist$DistTraveled <- dis.mat

temp.dist <- temp.dist %>%
  mutate(TimeBetweenGames = Date - lag(Date, 1)) %>%
  mutate(Team = l) %>%
  mutate(TimeChange = UTCOffsetHours - lag(UTCOffsetHours, 1)) %>%
  select(GID, Team, DistTraveled, TimeBetweenGames, TimeChange)

  print(paste(l))

  print(dim(temp.dist))
  dist.df <- bind_rows(dist.df, temp.dist)
}

dim(dist.df)

##Join with the Games data

Home.dist.df <- dist.df %>%
  rename(Home = Team,
         DistTraveled.H = DistTraveled,
         TimeBetweenGames.H = TimeBetweenGames,
         TimeChange.H = TimeChange)

Games.total <- left_join(Games2, Home.dist.df)

Visitor.dist.df <- dist.df %>%
  rename(Visitor = Team,
         DistTraveled.V = DistTraveled,
```

```
         TimeBetweenGames.V = TimeBetweenGames,
         TimeChange.V = TimeChange)

Games.total <- left_join(Games.total, Visitor.dist.df)

dim(Games.total)
Games.total <- na.omit(Games.total)
dim(Games.total)

## days from beginning of season,

Games.total <- Games.total %>%
  group_by(Season) %>%
  mutate(DaysFromStart = Date - min(Date))

##5.) Split the data into a training set and a test set.

- Our test set will be a random sample of 20% of the games from each season.
- Our training set will be the remaining 80% of games.
set.seed(0)
##Calculate 20% of the data
test <- dim(Games.total)[1]*0.2
test <- test/11
test

##Test set
NBATest <- Games.total %>%
  group_by(Season) %>%
  sample_n(round(test))
dim(NBATest)

NBATrain <- anti_join(Games.total, NBATest)

dim(NBATrain)

##6.) Write the data set to a csv.
write.csv(Games.total, "NBASLBD.csv")
write.csv(NBATrain, "NBATrainSLBD.csv")
write.csv(NBATest, "NBATestSLBD.csv")
```

Elastic Net

```
library(readr)
library(caret)
library(doParallel)
library(dplyr)
library(glmnet)
library(e1071)

NBATrainSLBD <- read_csv("NBATrainSLBD.csv")
NBATestSLBD <- read_csv("NBATestSLBD.csv")
```

```
### Grid of parameter values to check
### alpha = 0 is ridge
### alpha = 1 is lasso
### alpha in between is elastic net
### lambda is our penalty
parameter.values = expand.grid(alpha = seq(0,1, by = .1), lambda = 10^seq(3, -3, length.out = 300))

### Three outcomes to check
outcomes.wins = NBATrainSLBD %>%
  select(Won.Home)

outcomes.points = NBATrainSLBD %>%
  select(HFinal, VFinal)

outcomes.wins$Won.Home = factor(outcomes.wins$Won.Home)
outcomes.points = as.matrix(outcomes.points)

### Specify Factors
factors = c("Home",
            "Visitor",
            "Season")

### Change columns to factors
NBATrainSLBD[factors] = as.data.frame(lapply(NBATrainSLBD[factors], factor))
NBATestSLBD[factors] = as.data.frame(lapply(NBATestSLBD[factors], factor))

### Input variables
x = NBATrainSLBD %>%
  select(-X1, -Date, -HFinal, -VFinal, -Won.Home, -GID, -CSpreadH, -Covered)

### Change x to model matrix
x = model.matrix(~ . -1, data = x)
x = x[,-1] ### Removes intercept

### Change test data to model matrix
NBATestSLBD = model.matrix(~., data = NBATestSLBD)


### Run win model

cl = makeCluster(detectCores())
registerDoParallel(cl)

wins.model = train(x = x, y = outcomes.wins$Won.Home,
                   method = "glmnet", family = "binomial", tuneGrid = parameter.values)

stopCluster(cl)

predictions.wins = predict.train(wins.model, newdata = NBATestSLBD, type = "prob")


### Run points model
### result.1 is home points
```

```
### result.2 is visit points
cl = makeCluster(detectCores())
registerDoParallel(cl)

predictions.points = foreach(i = 1:ncol(outcomes.points), .combine = cbind, .packages = "caret") %dopar%

  lasso = train(x = x, y = outcomes.points[ , i],
                method = "glmnet", tuneGrid = parameter.values)
  lasso.predictions = predict.train(lasso, newdata = NBATestSLBD)


}

stopCluster(cl)



### Logloss function
logLoss = function(pred, actual){

  -1*mean(log(pred[model.matrix(~ actual + 0) - pred > 0]))

}


### Return test data back to df
NBATestSLBD = as.data.frame(NBATestSLBD)

### Logloss of our win prediction
logLoss(predictions.wins$`1`, NBATestSLBD$Won.Home)

### Make point predictions into df
predictions.points = as.data.frame(predictions.points)

### Rename cols
predictions.points = predictions.points %>%
  rename(Homepoints = result.1, Visitpoints = result.2)

### MSE for our predictions

mean((predictions.points$Homepoints - NBATestSLBD$HFinal)^2)
mean((predictions.points$Visitpoints - NBATestSLBD$VFinal)^2)

### Home points model

cl = makeCluster(detectCores())
registerDoParallel(cl)

home.point.elastic = train(x = x, y = outcomes.points[, 1],
              method = "glmnet", tuneGrid = parameter.values)


### Visit Points model
```

```
visit.point.elastic = train(x = x, y = outcomes.points[, 2],
                            method = "glmnet", tuneGrid = parameter.values)



stopCluster(cl)

save(wins.model, file = "wins.model.elastic.Rdata")
save(home.point.elastic, file = "home.point.elastic.Rdata")
save(visit.point.elastic, file = "visit.point.elastic.Rdada")
```

Extreme Gradient Boost

```
#load the required libraries
library(xgboost)
library(readr)
library(stringr)
library(caret)
library(car)
library(dplyr)
library(data.table)
library(drat)

#read in test and train data
train <- read.csv("NBATrainSLBD.csv")
test <- read.csv("NBATestSLBD.csv")

#subset out first column
test <- test[,-1]
train <- train[,-1]

#make names for binary factor variable
train$Won.Home <- as.factor(train$Won.Home)
train$Won.Home <- ifelse(train$Won.Home == 0, "lose", "win")

#convert data into form required by xgboost
setDT(train)
setDT(test)

train_dat=train[,-c(1, 3, 5, 7, 8, 135, 138)]
test_dat=test[,-c(1, 3, 5, 7, 8, 135, 138)]

##################################
### WIN PROBABILITY HOME TEAM ###
##################################

xgb_grid_1 = expand.grid(
  nrounds = 500,
  eta = c(0.1, 0.2, 0.25, 0.3),
  max_depth = c(4, 10, 12, 20),
  gamma = c(0.5,1),
  colsample_bytree = c(0.5, 0.75),
```

```
  min_child_weight = 1,
  subsample = c(0.65, 0.75)
)

xgb_trcontrol_1 = trainControl(
  method = "cv",
  number = 5,
  verboseIter = TRUE,
  returnData = FALSE,
  returnResamp = "all",
  classProbs = TRUE,
  summaryFunction = twoClassSummary,
  allowParallel = TRUE
)

xgb_train_1 = train(
  x = data.matrix(train_dat),
  y = as.factor(train$Won.Home),
  trControl = xgb_trcontrol_1,
  tuneGrid = xgb_grid_1,
  method = "xgbTree",
  nthread = 8
)

## Assess Model Fit - LogLoss
logLoss = function(pred, actual){
  -1*mean(log(pred[model.matrix(~ actual + 0) - pred > 0]))
  }

preds <- predict(xgb_train_1, data.matrix(test_dat))

preds_prob <- predict(xgb_train_1, data.matrix(test_dat), type = "prob")[2]
logLoss(preds_prob, test$Won.Home)


preds_numeric <- ifelse(preds == "win", 1, 0)
gbm.ROC <- roc(predictor=preds_numeric,
               response=test$Won.Home,
               levels= c(0,1))
gbm.ROC$auc

png('roc_xgb.png')
plot(gbm.ROC,main="ROC - Extreme Gradient Boost Model")
dev.off()




save(xgb_train_1, file ="xgb_train_1.RData")

##################################
### EXPECTED POINTS HOME TEAM ###
##################################
```

```r
# xgb_grid_2 = expand.grid(
#   nrounds = 250,
#   lambda = c(.001, .01, .1),
#   alpha = c(.001, .01, .1),
#   eta = c(0.1, 0.2, 0.3)
# )

xgb_grid_2b = expand.grid(
  nrounds = 500,
  lambda = c(.001, .005, .01, .015),
  alpha = c(.01, .015, .1, .15),
  eta = c(.05, .1, .15)
)

xgb_trcontrol_2 = trainControl(
  method = "cv",
  number = 3,
  verboseIter = TRUE,
  returnData = FALSE,
  returnResamp = "all",
  allowParallel = TRUE
)

xgb_train_2 = train(
  x = data.matrix(train_dat),
  y = train$HFinal,
  trControl = xgb_trcontrol_2,
  tuneGrid = xgb_grid_2b,
  method = "xgbLinear",
  nthread = 8
)

save(xgb_train_2b, file ="xgb_train_2b.RData")

## Assess Model Fit - MSE and Plots
pred_hfinal <- predict(xgb_train_2b, data.matrix(test_dat))

plot(pred_hfinal, (test$HFinal - pred_hfinal), xlab = "Predicted Values",
     ylab = "Observed Values", main = "Expected Points Home Team")
abline(h = 0)

mean((test$HFinal - pred_hfinal)^2)

###################################
### EXPECTED POINTS AWAY TEAM ###
###################################
xgb_grid_3 = expand.grid(
   nrounds = 250,
  lambda = c(.01, .1),
   alpha = c(.01, .1),
   eta = c(0.1, 0.2, 0.3)
)
```

```
# xgb_grid_3b = expand.grid(
#   nrounds = 500,
#   lambda = c(.001, .005, .01, .015),
#   alpha = c(.001, .005, .01, .015),
#   eta = c(.05, .1, .15)
# )


xgb_trcontrol_3 = trainControl(
  method = "cv",
  number = 3,
  verboseIter = TRUE,
  returnData = FALSE,
  returnResamp = "all",
  allowParallel = TRUE
)

xgb_train_3 = train(
  x = data.matrix(train_dat),
  y = train$VFinal,
  trControl = xgb_trcontrol_3,
  tuneGrid = xgb_grid_3,
  method = "xgbLinear",
  nthread = 8
)

save(xgb_train_3, file ="xgb_train_3.RData")

## Assess Model Fit - MSE and Plots
pred_vfinal <- predict(xgb_train_3, data.matrix(test_dat))

plot(pred_vfinal, (test$VFinal - pred_vfinal), xlab = "Predicted Values",
     ylab = "Observed Values", main = "Expected Points Visitor Team")
abline(h = 0)

mean((test$VFinal - pred_vfinal)^2)

### TRY MORE ###
##this one improves MSE
xgb_train_2b = train(
  x = data.matrix(train_dat),
  y = train$HFinal,
  trControl = xgb_trcontrol_2,
  tuneGrid = xgb_grid_2b,
  method = "xgbLinear",
  nthread = 8
)

##this one does not improve MSE

xgb_train_3b = train(
  x = data.matrix(train_dat),
  y = train$VFinal,
```

```
  trControl = xgb_trcontrol_3,
  tuneGrid = xgb_grid_3b,
  method = "xgbLinear",
  nthread = 8
)

save(xgb_train_3b, file ="xgb_train_3b.RData")
save(xgb_train_2b, file ="xgb_train_2b.RData")
```

Neural Net

```
require(nnet)
require(dplyr)
require(stats)
require(caret)
require(doParallel)

set.seed(1)


#assume data is already read in


#expected points model code
#subset out "Won.Home," "Date," "Row," "Covered.Spread"
train <- train[, -c(1, 2, 8, 9, 136)]

#xval for nnet parameters
ngrid <- expand.grid(decay = c(.1, .5, .75), size = c(2, 3, 4, 5))

#register parallel
cl <- makeCluster(24)
registerDoParallel(cl)

#nnets for all vars
h.all.nn.train <- train(HFinal/152 ~ ., data = train[, -4], method = "nnet", maxit = 1000, tuneGrid = ng
v.all.nn.train <- train(VFinal/151 ~ ., data = train[, -2], method = "nnet", maxit = 1000, tuneGrid = ng

save(h.all.nn.train, v.all.nn.train, file = "nn_all.RData")
stopCluster(cl)

#winprob model code
#subset out "Date," "Row," "Covered.Spread," final scores
train <- train[, -c(1, 2, 8, 9)]

cl <- makeCluster(24)
registerDoParallel(cl)

win.all.nn <- train(Won.Home ~., data = train[, -c(2, 4)], method = "nnet",
    maxit = 1000,
    tuneGrid = ngrid)
```

```
save(win.all.nn, file = "win.nnet.RData")
stopCluster(cl)
```

Random Forest

```
##Load libraries
library(dplyr)
library(randomForest)
library(ggplot2)
library(readr)
library(doParallel)
library(randomForest)


##Load Data

setwd("C:/Users/dyam/Google Drive/NBA Final Project SLBD")
NBA <- read_csv("NBASLBD.csv")
NBATrain <- read_csv("NBATrainSLBD.csv")
NBATest <- read_csv("NBATestSLBD.csv")
set.seed(1)

##Organize the data

##Select only the number of variables that we need to run the RF
##Use the matches function to get all of the stats
NBATrain <- NBATrain %>%
  select(Won.Home, everything())

NBATrain <- NBATrain %>%
  select(- HFinal, - VFinal, -CSpreadH, -Covered, -X1, - Date)

NBATrain <- NBATrain %>%
  mutate(Home = as.factor(Home),
         Visitor = as.factor(Visitor))

NBATest <- NBATest %>%
  select(Won.Home, everything())

NBATest <- NBATest %>%
  select(- HFinal,  - VFinal, -CSpreadH, -Covered, -X1, - Date)

NBATest <- NBATest %>%
  mutate(Home = as.factor(Home),
         Visitor = as.factor(Visitor))

##Train an RF model on all of the data.

Parallel Random Forest

cl <- makeCluster(7)
registerDoParallel(cl)
```

```
All.Forest <- foreach(y=seq(10), .combine=combine , .packages='randomForest') %dopar% {
   rf <- randomForest(Won.Home ~ ., NBATrain,
                      mtry = 10, nodesize = 10, ntree=70, norm.votes=FALSE, do.trace = T,
                      importance = T, keep.forest=TRUE)
}

stopCluster(cl)

save(All.Forest, file = "allWPTeam.RData")
##Run a variable selection using best subset regression.

##Use Lasso to select variables

library(glmnet)

lasso.nba <- cv.glmnet(as.matrix(NBATrain[,c(-1, -2, -3)]), as.matrix(NBATrain[,1]),
                       alpha = 1, family = "binomial")

tmp_coeffs <- coef(lasso.nba, s = "lambda.min")
Coefficients <- data.frame(name = tmp_coeffs@Dimnames[[1]][tmp_coeffs@i + 1], coefficient = tmp_coeffs@
head(Coefficients)

Coefficients <- Coefficients[-1,]
head(Coefficients)

Coef.vec <- as.character(Coefficients$name)
head(Coef.vec)

Trainsubset <- NBATrain %>% select(Won.Home, Home, Visitor, one_of(Coef.vec))
head(Trainsubset)

write.csv(Trainsubset, "TrainingSubsetTeam.csv")

cl <- makeCluster(7)
registerDoParallel(cl)

subForest <- foreach(y=seq(10), .combine=combine , .packages='randomForest') %dopar% {
   rf.sub <- randomForest(Won.Home ~ ., Trainsubset,
                      mtry = 10, nodesize = 10, ntree=70, norm.votes=FALSE, do.trace = T,
                      importance = T, keep.forest=TRUE)
}


stopCluster(cl)

save(subForest, file = "subsetWPTeam.RData")

##Compare the two models

## Predict WP on test data
NBATest$wp.all <- predict(All.Forest, NBATest)
NBATest$wp.subset<- predict(subForest, NBATest)
```

```
##Create a data frame for the WP accuracy plot
plot.all <- NBATest %>%
  dplyr::select(GID, wp.all, wp.subset, Won.Home) %>%
  mutate(all.cat = cut_number(wp.all, 20),
         subset.cat = cut_number(wp.subset, 20))

all.probs <- plot.all %>%
  group_by(all.cat) %>%
  summarise(estimated = mean(wp.all), observed = mean(Won.Home), type = "All")
subset.probs <- plot.all %>%
  group_by(subset.cat) %>%
  summarise(estimated = mean(wp.subset), observed = mean(Won.Home), type = "Subset")

wp.all <- bind_rows(all.probs, subset.probs)

p.wpacc <- ggplot(wp.all, aes(x = estimated, y = observed,
                              shape = type, colour = type))

wpacc <- p.wpacc +
  geom_point(size = 2.5) +
  geom_abline(intercept = 0, slope = 1) +
  scale_x_continuous(labels = scales::percent, "Estimated win rate") +
  scale_y_continuous(labels = scales::percent, "Observed win rate") +
  scale_colour_manual(values = c("black", "red"),
                                  name  ="Win Probability Model",
                             breaks=c("All", "Subset"),
                             labels=c("All", "Subset")) +
  scale_shape_manual(values = c(19, 3), name  ="Win Probability Model",
                            breaks=c("All", "Subset"),
                             labels=c("All", "Subset")) +
  ylab("Proportion of Games Won") +
  ggtitle("Accuracy of the win probability models") +
  theme(plot.title = element_text(hjust = 0.5, size = rel(1.2)),
        legend.position = c(.85, .15),
        axis.title.y = element_text(size = rel(1.2)),
        axis.title.x = element_text(size = rel(1.2)))
wpacc
#ggsave(wpacc, file = "Figures/WP_accuracy_plot.png", height = 6, width = 8)
##Compare the log loss

logLoss = function(pred, actual){
  -1*mean(log(pred[model.matrix(~ actual + 0) - pred > 0]))
}

logLoss(NBATest$wp.all, NBATest$Won.Home)
logLoss(NBATest$wp.subset, NBATest$Won.Home)

##Select only the number of variables that we need to run the RF
##Use the matches function to get all of the stats
NBATrain <- NBATrain %>%
  select(HFinal, everything())

NBATrain <- NBATrain %>%
```

```
  select(-Won.Home, - VFinal, -CSpreadH, -Covered, -X1, - Date)

NBATrain <- NBATrain %>%
  mutate(Home = as.factor(Home),
         Visitor = as.factor(Visitor))

set.seed(1)

NBATest <- NBATest %>%
  select(HFinal, everything())

NBATest <- NBATest %>%
  select(- Won.Home, - VFinal, -CSpreadH, -Covered, -X1, - Date)

NBATest <- NBATest %>%
  mutate(Home = as.factor(Home),
         Visitor = as.factor(Visitor))

##Train an RF model on all of the data.

library(doParallel)
library(randomForest)
cl <- makeCluster(7)
registerDoParallel(cl)

All.Forest <- foreach(y=seq(10), .combine=combine , .packages='randomForest') %dopar% {
   rf <- randomForest(HFinal ~ ., NBATrain,
                      mtry = 10, nodesize = 10, ntree=50, norm.votes=FALSE, do.trace = T,
                      importance = T, keep.forest=TRUE)
}

stopCluster(cl)

save(All.Forest, file = "allEPTeam.RData")

##Use Lasso to select variables

library(glmnet)

lasso.nba <- cv.glmnet(as.matrix(NBATrain[,c(-1, -2, -3)]), as.matrix(NBATrain[,1]), alpha = 1)

tmp_coeffs <- coef(lasso.nba, s = "lambda.min")
Coefficients <- data.frame(name = tmp_coeffs@Dimnames[[1]][tmp_coeffs@i + 1], coefficient = tmp_coeffs@
Coefficients <- Coefficients[-1,]
head(Coefficients)

Coef.vec <- as.character(Coefficients$name)
head(Coef.vec)

Coefficients <- Coefficients[-1,]
head(Coefficients)

Coef.vec <- as.character(Coefficients$name)
```

```
head(Coef.vec)

Trainsubset <- NBATrain %>% select(HFinal, Home, Visitor, one_of(Coef.vec))
head(Trainsubset)
#Trainsubset <- cbind(NBATrain$HFinal, Trainsubset) %>%
#  rename(HFinal = `NBATrain$HFinal`)
#Trainsubset <- as_tibble(Trainsubset)

cl <- makeCluster(7)
registerDoParallel(cl)

subForest <- foreach(y=seq(10), .combine=combine , .packages='randomForest') %dopar% {
   rf.sub <- randomForest(HFinal ~ ., Trainsubset,
                       mtry = 10, nodesize = 10, ntree=70, norm.votes=FALSE, do.trace = T,
                       importance = T, keep.forest=TRUE)
}

stopCluster(cl)

save(subForest, file = "subsetEPTeam.RData")

##Test the Results

## Predict WP on test data
NBATest$ep.all <- predict(All.Forest, NBATest)
NBATest$ep.subset<- predict(subForest, NBATest)

plot(NBATest$ep.all, (NBATest$HFinal - NBATest$ep.all))
plot(NBATest$ep.subset, (NBATest$HFinal - NBATest$ep.subset))

mean((NBATest$HFinal-NBATest$ep.all)^2)
mean((NBATest$HFinal-NBATest$ep.subset)^2)

##Select only the number of variables that we need to run the RF
##Use the matches function to get all of the stats
NBATrain <- NBATrain %>%
  select(VFinal, everything())

NBATrain <- NBATrain %>%
  select(-Won.Home, - HFinal, -CSpreadH, -Covered, -X1, - Date)

NBATrain <- NBATrain %>%
  mutate(Home = as.factor(Home),
         Visitor = as.factor(Visitor))

set.seed(1)

NBATest <- NBATest %>%
  select(VFinal, everything())

NBATest <- NBATest %>%
  select(- Won.Home, - HFinal, -CSpreadH, -Covered, -X1, - Date)
```

```r
NBATest <- NBATest %>%
  mutate(Home = as.factor(Home),
         Visitor = as.factor(Visitor))


#Xtest <- as.data.frame(NBATest[,-1])
#Ytest <- NBATest$Won.Home

##Train an RF model on all of the data.

library(doParallel)
library(randomForest)
cl <- makeCluster(7)
registerDoParallel(cl)

All.Forest <- foreach(y=seq(10), .combine=combine , .packages='randomForest') %dopar% {
   rf <- randomForest(VFinal ~ ., NBATrain,
                      mtry = 10, nodesize = 10, ntree=70, norm.votes=FALSE, do.trace = T,
                      importance = T, keep.forest=TRUE)
}

stopCluster(cl)

save(All.Forest, file = "VallEP.RData")

##Use Lasso to select variables

library(glmnet)

lasso.nba <- cv.glmnet(as.matrix(NBATrain[,c(-1, -2, -3)]), as.matrix(NBATrain[,1]), alpha = 1)

tmp_coeffs <- coef(lasso.nba, s = "lambda.min")
Coefficients <- data.frame(name = tmp_coeffs@Dimnames[[1]][tmp_coeffs@i + 1], coefficient = tmp_coeffs@
Coefficients <- Coefficients[-1,]
head(Coefficients)

Coef.vec <- as.character(Coefficients$name)
head(Coef.vec)

Trainsubset <- NBATrain %>% select(VFinal, Home, Visitor, one_of(Coef.vec))

cl <- makeCluster(7)
registerDoParallel(cl)

subForest <- foreach(y=seq(10), .combine=combine , .packages='randomForest') %dopar% {
   rf.sub <- randomForest(VFinal ~ ., Trainsubset,
                      mtry = 10, nodesize = 10, ntree=50, norm.votes=FALSE, do.trace = T,
                      importance = T, keep.forest=TRUE)
}


stopCluster(cl)
```

```
save(subForest, file = "VsubsetEP.RData")

##Test the Results

## Predict WP on test data
NBATest$ep.all <- predict(All.Forest, NBATest)
NBATest$ep.subset<- predict(subForest, NBATest)

plot(NBATest$ep.all, (NBATest$VFinal - NBATest$ep.all))
plot(NBATest$ep.subset, (NBATest$VFinal - NBATest$ep.subset))

mean((NBATest$VFinal-NBATest$ep.all)^2)
mean((NBATest$VFinal-NBATest$ep.subset)^2)
```