Baseball Analytics Project: Predicting Full Season On-Base Percentage

Daniel Jackson

February 22nd, 2024

Libraries Used

```
library(googlesheets4)
library(corrplot)
library(dplyr)
library(glmnet)
library(caret)
library(pls)
library(ggplot2)
library(tree)
library(ipred)
library(randomForest)
library(gbm)
```

Introduction to Analysis

Our goal of this analysis is to predict each player's full season on-base percentage (OBP) at the end of the 2019 Major League Baseball (MLB) regular season given the player's batting statistics in March and April of 2019. We will read in and clean up our data. 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 the player's actual full season on-base percentage to see to see how accurate our predictions were. Let's start by reading in our data from a Google Sheet. The url to that Google Sheet is below for reference:

https://docs.google.com/spreadsheets/d/1U_QsSv6-68VLI0-5YrSPp2Bo-QhzZftiMU10b1Ajk0k/edit#gid=1381900348.

In this data set, there are 320 observations, 28 predictors and the full season OBP response variable.

Read In and Clean Data

```
# Read in and clean data
sheet_url = "https://docs.google.com/spreadsheets/d/1U_QsSv6-68VLI0-5YrSPp2Bo-QhzZftiMU10b1Ajk0k/edit#g
# Use gs4_auth() and sign into Gmail to access Google Sheet
mlb_df = read_sheet(sheet_url)
```

```
## ! Using an auto-discovered, cached token.

##   To suppress this message, modify your code or options to clearly consent to
##   the use of a cached token.

##   See gargle's "Non-interactive auth" vignette for more details:

##   <https://gargle.r-lib.org/articles/non-interactive-auth.html>

## i The googlesheets4 package is using a cached token for
##   'djackson2155@gmail.com'.

## v Reading from "batting".

## v Range 'batting'.
```

```
head(mlb_df)
```

```
## # A tibble: 6 x 29
##   playerid Name            Team  MarApr_PA MarApr_AB MarApr_H MarApr_HR MarApr_R
##      <dbl> <chr>           <chr>     <dbl>     <dbl>    <dbl>     <dbl>    <dbl>
## 1    15998 Cody Bellinger  LAD         132       109       47        14       32
## 2    11477 Christian Yeli~ MIL         124       102       36        14       26
## 3    17975 Scott Kingery   PHI          35        32       13         2        5
## 4     7927 Eric Sogard     TOR          49        43       17         3        8
## 5    14130 Daniel Vogelba~ SEA          92        71       22         8       15
## 6    12861 Anthony Rendon  WSN          86        73       26         6       21
## # i 21 more variables: MarApr_RBI <dbl>, MarApr_SB <dbl>, `MarApr_BB%` <dbl>,
## #   `MarApr_K%` <dbl>, MarApr_ISO <dbl>, MarApr_BABIP <dbl>, MarApr_AVG <dbl>,
## #   MarApr_OBP <dbl>, MarApr_SLG <dbl>, `MarApr_LD%` <dbl>, `MarApr_GB%` <dbl>,
## #   `MarApr_FB%` <dbl>, `MarApr_IFFB%` <dbl>, `MarApr_HR/FB` <dbl>,
## #   `MarApr_O-Swing%` <dbl>, `MarApr_Z-Swing%` <dbl>, `MarApr_Swing%` <dbl>,
## #   `MarApr_O-Contact%` <dbl>, `MarApr_Z-Contact%` <dbl>,
## #   `MarApr_Contact%` <dbl>, FullSeason_OBP <dbl>
```

```
dim(mlb_df)
```

```
## [1] 320  29
```

```
# Look for NAs in any columns
colSums(is.na(mlb_df))
```

```
##          playerid              Name              Team         MarApr_PA
##                 0                 0                 0                 0
##         MarApr_AB          MarApr_H         MarApr_HR          MarApr_R
##                 0                 0                 0                 0
##        MarApr_RBI         MarApr_SB        MarApr_BB%        MarApr_K%
##                 0                 0                 0                 0
##        MarApr_ISO      MarApr_BABIP        MarApr_AVG        MarApr_OBP
##                 0                 0                 0                 0
```

```
##        MarApr_SLG        MarApr_LD%        MarApr_GB%        MarApr_FB%
##                 0                 0                 0                 0
##      MarApr_IFFB%      MarApr_HR/FB   MarApr_O-Swing%   MarApr_Z-Swing%
##                 0                 0                 0                 0
##     MarApr_Swing% MarApr_O-Contact% MarApr_Z-Contact%   MarApr_Contact%
##                 0                 0                 0                 0
##    FullSeason_OBP
##                 0
```

```
# No NAs. That means we have no missing values. Awesome!

# Let's remove playerid column as they will not have impact on predicting player's
# FSOBP (full season obp)
mlb_df = mlb_df[, -which(names(mlb_df) == "playerid")]

# Now since we now all of the data is between March and April, I want to clean up
# our predictor names to make coding easier.
# Let's remove "MarApr_" from each predictor
colnames(mlb_df)
```

```
##  [1] "Name"              "Team"              "MarApr_PA"
##  [4] "MarApr_AB"         "MarApr_H"          "MarApr_HR"
##  [7] "MarApr_R"          "MarApr_RBI"        "MarApr_SB"
## [10] "MarApr_BB%"        "MarApr_K%"         "MarApr_ISO"
## [13] "MarApr_BABIP"      "MarApr_AVG"        "MarApr_OBP"
## [16] "MarApr_SLG"        "MarApr_LD%"        "MarApr_GB%"
## [19] "MarApr_FB%"        "MarApr_IFFB%"      "MarApr_HR/FB"
## [22] "MarApr_O-Swing%"   "MarApr_Z-Swing%"   "MarApr_Swing%"
## [25] "MarApr_O-Contact%" "MarApr_Z-Contact%" "MarApr_Contact%"
## [28] "FullSeason_OBP"
```

```
predictor_names = colnames(mlb_df)
# Remove "MarApr_" from each predictor name
new_names = sub("^MarApr_", "", predictor_names)
colnames(mlb_df) = new_names

# Now let's swap the "%" with "pct"
predictor_names = colnames(mlb_df)
new_names = sub("%", "_pct", predictor_names)
colnames(mlb_df) = new_names

# Now let's swap any "-" with "_"
predictor_names = colnames(mlb_df)
new_names = sub("-", "_", predictor_names)
colnames(mlb_df) = new_names

# Change HR/FB predictor to HR_FB
colnames(mlb_df)[colnames(mlb_df) == "HR/FB"] = "HR_FB"

# Change response variable to FS_OBP to make code easier
colnames(mlb_df)[colnames(mlb_df) == "FullSeason_OBP"] = "FS_OBP"

# Change all predictors to lower case
```

```r
colnames(mlb_df) = tolower(colnames(mlb_df))

# Now that the data is cleaned, we can now start our analysis
```

Exploratory Analysis

As mentioned before, there are 320 observations in this data set. Each observation represents a player in the MLB and their corresponding statistics. There are 28 statistics from that 2019 March-April time frame when these statistics were gathered. Those corresponding statistics will be used as our predictors to predict the full season OBP 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.

pa = player's plate appearances.
ab = player's at bats.
h = player's hits.
hr = player's home runs.
r = player's runs scored.
rbi = player's runs batted in (RBI).
sb = player's stolen bases.
bb_pct = player's walk percentage.
k_pct = player's strikeout percentage.
iso = player's isolated power.
babip = player's BABIP.
avg = player's batting average.
obp = player's on-base percentage.
slg = player's slugging percentage.
ld_pct = player's line drive percentage.
gb_pct = player's ground ball percentage.
fb_pct = player's fly ball percentage.
iffb_pct = player's infield fly ball percentage.
hr_fb = player's home run per fly ball rate.
o_swing_pct = player's out-of-zone swing-per-pitch percentage.
z_swing_pct = player's in-zone swing-per-pitch percentage.
swing_pct = player's total swing-per-pitch percentage.
o_contact_pct = player's out-of-zone contact-per-swing percentage.
z_contact_pct = player's in-zone contact-per-swing percentage.
contact_pct = player's total contact-per-swing percentage.

What is OBP?
Below you will find the official description of what on-base percentage is (via https://www.mlb.com/glossary/standard-stats/on-base-percentage#):
"OBP refers to how frequently a batter reaches base per plate appearance. Times on base include hits, walks and hit-by-pitches, but do not include errors, times reached on a fielder's choice or a dropped third strike."

Here is the formula for OBP:
(Hits + HBP + Walks) / (At-Bats + HBP + Walks + Sacrifice Flies)

Before we dive into our regression models to predict each player's full season OBP, let's take a look at the data and see who in the MLB was performing the best offensively during this time frame.

Let's look at how many players are in this data set per team.

```
table(mlb_df$team)
```

```
##
## - - -   ARI   ATL   BAL   BOS   CHC   CHW   CIN   CLE   COL   DET   HOU   KCR
##     2    10    11    10    10    10    12     9    12    12    11    12     9
##   LAA   LAD   MIA   MIL   MIN   NYM   NYY   OAK   PHI   PIT   SDP   SEA   SFG
##    10    11     8    12    13    12    11    11     8    10    11    12     8
##   STL   TBR   TEX   TOR   WSN
##    10    11    12    10    10
```

There are two players not assigned to a team. I am not too worried about this as we will not be looking at full season OBP by team in our regression analysis.

Now, let's look at the top ten players who had the best OBP at the end of the 2019 season in our data set.

```
top_ten_players_fs_obp = mlb_df[order(-mlb_df$fs_obp), ][1:10, c("name", "fs_obp")]
print(top_ten_players_fs_obp)
```

```
## # A tibble: 10 x 2
##    name             fs_obp
##    <chr>             <dbl>
##  1 Mike Trout        0.438
##  2 Christian Yelich  0.429
##  3 Alex Bregman      0.423
##  4 Anthony Rendon    0.412
##  5 Matt Joyce        0.408
##  6 Cody Bellinger    0.406
##  7 Anthony Rizzo     0.405
##  8 David Freese      0.403
##  9 Juan Soto         0.401
## 10 Carlos Santana    0.397
```

Mike Trout, Christian Yelich and Alex Bregman were the top three players in OBP at the end of the regular season in 2019.

Now, let's look at the top ten players who had the best OBP during the March-April stretch.

```
top_ten_players_obp = mlb_df[order(-mlb_df$obp), ][1:10, c("name", "obp")]
print(top_ten_players_obp)
```

```
## # A tibble: 10 x 2
##    name               obp
##    <chr>            <dbl>
##  1 Cody Bellinger   0.508
##  2 Dominic Smith    0.5
##  3 Mike Trout       0.487
##  4 Christian Yelich 0.46
##  5 Eric Sogard      0.458
##  6 Scott Kingery    0.457
##  7 Daniel Vogelbach 0.457
##  8 Jeff McNeil      0.457
##  9 Hunter Dozier    0.447
## 10 Anthony Rendon   0.442
```

Cody, Bellinger, Dominic Smith and Mike Trout were the top three players in OBP during March-April of the 2019 season.

Let's look at which players were in each top ten.

```
common_names = intersect(top_ten_players_fs_obp$name, top_ten_players_obp$name)
print(common_names)
```

```
## [1] "Mike Trout"       "Christian Yelich" "Anthony Rendon"   "Cody Bellinger"
```

The four players that were in both the top ten in OBP during March-April that finished in the top ten in OBP at the end of the regular season were Mike Trout, Christian Yelich, Anthony Rendon and Cody Bellinger. Looking back on that 2019 season, Mike Trout won the American League Most Valuable Player award and Cody Bellinger won the National League Most Valuable Player Award. Their consistency with OBP definitely played a role in helping them each win the MVP for their respective leagues.

Let's look at top 10 list for hits, homeruns, runs, RBIs, and average.
Hits:

```
top_ten_players_hits = mlb_df[order(-mlb_df$h), ][1:10, c("name", "h")]
print(top_ten_players_hits)
```

```
## # A tibble: 10 x 2
##    name                h
##    <chr>           <dbl>
##  1 Cody Bellinger     47
##  2 Paul DeJong        40
##  3 Elvis Andrus       39
##  4 Trey Mancini       39
##  5 Michael Brantley   39
##  6 David Peralta      39
##  7 Jeff McNeil        37
##  8 Christian Yelich   36
##  9 Tim Anderson       36
## 10 Domingo Santana    36
```

Cody Bellinger, Paul DeJong and Elvis Andurs were the top three players in hits during this stretch.

Homeruns:

```
top_ten_players_homeruns = mlb_df[order(-mlb_df$hr), ][1:10, c("name", "hr")]
print(top_ten_players_homeruns)
```

```
## # A tibble: 10 x 2
##    name               hr
##    <chr>           <dbl>
##  1 Cody Bellinger     14
##  2 Christian Yelich   14
##  3 Eddie Rosario      11
##  4 Joey Gallo         10
##  5 Joc Pederson       10
##  6 Marcell Ozuna      10
##  7 Pete Alonso         9
```

```
##  8 Javier Baez          9
##  9 George Springer      9
## 10 Paul Goldschmidt     9
```

Cody Bellinger, Christian Yelich and Eddie Rosario were top three in homeruns during this stretch.

RBIs:

```
top_ten_players_rbi = mlb_df[order(-mlb_df$rbi), ][1:10, c("name", "rbi")]
print(top_ten_players_rbi)
```

```
## # A tibble: 10 x 2
##    name               rbi
##    <chr>            <dbl>
##  1 Cody Bellinger      37
##  2 Christian Yelich    34
##  3 Marcell Ozuna       28
##  4 Pete Alonso         26
##  5 Joey Gallo          25
##  6 Luke Voit           25
##  7 Domingo Santana     25
##  8 Rhys Hoskins        24
##  9 George Springer     24
## 10 Eddie Rosario       24
```

Cody Bellinger, Christian Yelich and Marcell Ozuna were the top three in RBIs during this stretch.

Average:

```
top_ten_players_avg = mlb_df[order(-mlb_df$avg), ][1:10, c("name", "avg")]
print(top_ten_players_avg)
```

```
## # A tibble: 10 x 2
##    name             avg
##    <chr>          <dbl>
##  1 Cody Bellinger 0.431
##  2 Scott Kingery  0.406
##  3 Eric Sogard    0.395
##  4 Tim Anderson   0.375
##  5 Tom Murphy     0.37
##  6 Dominic Smith  0.37
##  7 Jeff McNeil    0.37
##  8 Jose Martinez  0.364
##  9 Elvis Andrus   0.361
## 10 James McCann   0.357
```

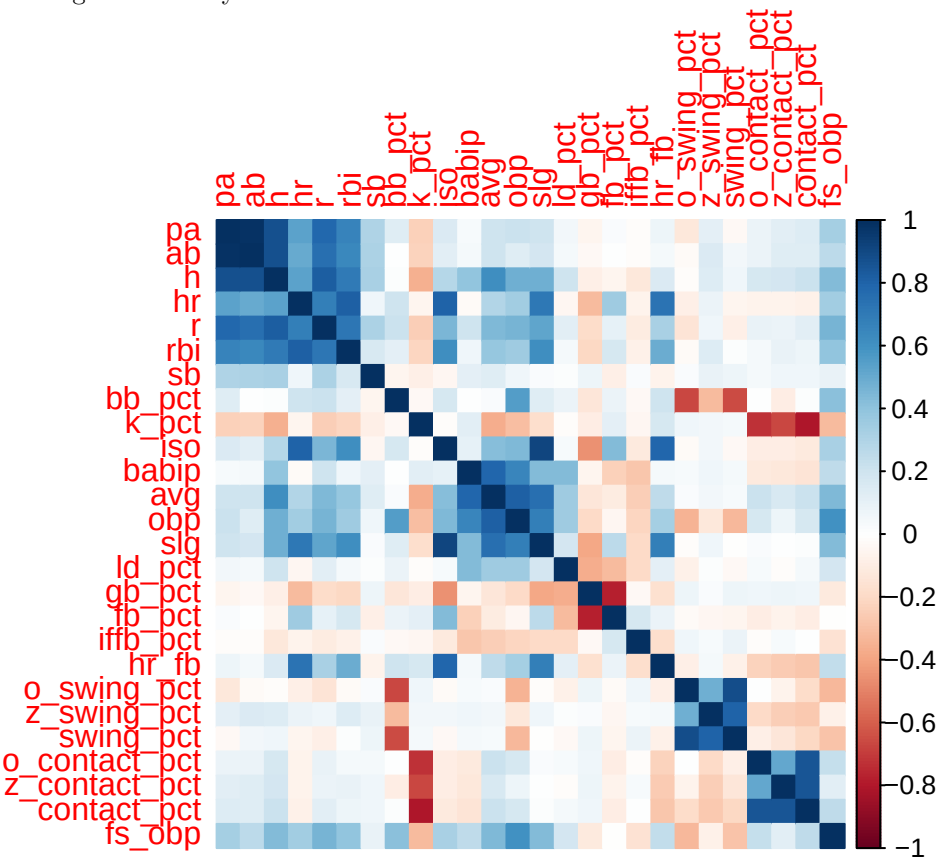
Cody Bellinger, Scott Kingery and Eric Sogard were top three in batting average during this stretch.

As we can see, Cody Bellinger led the entire league in hits, homeruns, RBIs, batting average and OBP during this stretch. What is interesting is that we do not see Mike Trout anywere in the top ten for any of the stats that we looked at. Without analyzing the entire 2019 season's data, we cannot say for sure what happened. But I think that it is safe to say that he had a slow start and finished incredibly strong to win his 5th MVP award that year.

Now, we want to start looking at the numeric values and see how they are correlated to each other and our response variable of fs_obp. We can remove name and team for this analysis.

```
mlb_df = mlb_df[, -which(names(mlb_df) == "name")]
mlb_df = mlb_df[, -which(names(mlb_df) == "team")]
```

Let's look at correlation plot of our variables to see if we can remove any of our predictors.We are looking for instances of multicollinearity. We want to try and remove some variables to help simplify model during our regression analysis.



We see that fb_pct (flyball percentage) has no correlation to fs_obp so we can remove this predictor. We also see that sb also has little correlation with fs_obp. Let's remove sb as well.

```
mlb_df = mlb_df[, -which(names(mlb_df) == "fb_pct")]
mlb_df = mlb_df[, -which(names(mlb_df) == "sb")]
```

Let's look at the highly positively correlated predictors: We see that pa and ab are highly correlated. Knowing what we know about our formula to predict OBP, let's keep at-bats, and remove pa.

```
mlb_df = mlb_df[, -which(names(mlb_df) == "pa")]
```

We see that homeruns and hits are not that correlated, but we know that when you hit a homerun, it is considered a hit and based on our calculation for OBP, we only care about hits, not homeruns. Let's remove homeruns. Since we are removing hr, let's also remove hr_fb.

```
mlb_df = mlb_df[, -which(names(mlb_df) == "hr")]
mlb_df = mlb_df[, -which(names(mlb_df) == "hr_fb")]
```

We see that avg and babip highly correlated. Since we know BABIP is batting average on balls put in play, we want to see average for balls also not put in play, so let's remove babip predictor.

```
mlb_df = mlb_df[, -which(names(mlb_df) == "babip")]
```

We see that o_swing_pct and z_swing_pct is highly correlated with total swing_pct. Since both o_swing_pct and z_swing_pct are used to calculate swing_pct, let's remove o_swing_pct and z_swing_pct.

```
mlb_df = mlb_df[, -which(names(mlb_df) == "o_swing_pct")]
mlb_df = mlb_df[, -which(names(mlb_df) == "z_swing_pct")]
```

We see the same thing with o_contact_pct and z_contact_pct being highly correlated with contact_pct. Since o_contact_pct and z_contact_pct are used to calculate contact_pct, let's remove o_contact_pct and z_contact_pct.

```
mlb_df = mlb_df[, -which(names(mlb_df) == "o_contact_pct")]
mlb_df = mlb_df[, -which(names(mlb_df) == "z_contact_pct")]
```

We see that slg and iso, are highly correlated. The iso stat is difference between slugging and average to get isolated power. In this case, let's just remove iso because we already have slg and avg predictors in data set.

```
mlb_df = mlb_df[, -which(names(mlb_df) == "iso")]
```

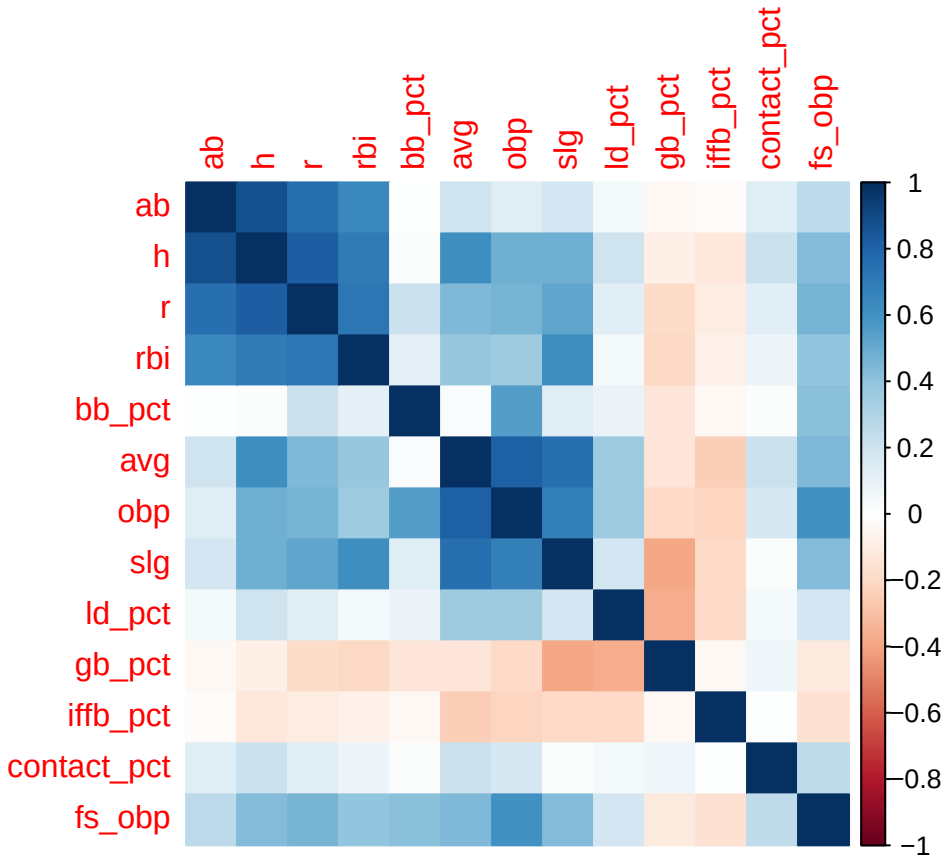
Let's look at the highly negatively correlated predictors: We see that contact_pct and k_pct are negatively correlated and could create multicollinearity issues. Based on our knowledge of what we are looking for in full season OBP, let's keep contact_pct and remove k_pct.

```
mlb_df = mlb_df[, -which(names(mlb_df) == "k_pct")]
```

We also see that swing_pct and bb_pct are highly negatively correlated. Let's remove swing_pct.

```
mlb_df = mlb_df[, -which(names(mlb_df) == "swing_pct")]
```

Now that we have removed some highly(positively and negativly) correlated predictors to avoid multicollinearity issues, let's look at how many predictors that we are left with.



We are now left with 12 predictors to help us predict our full season OBP statistic.

Let's look at correlation of each predictor in regards to full season OBP.

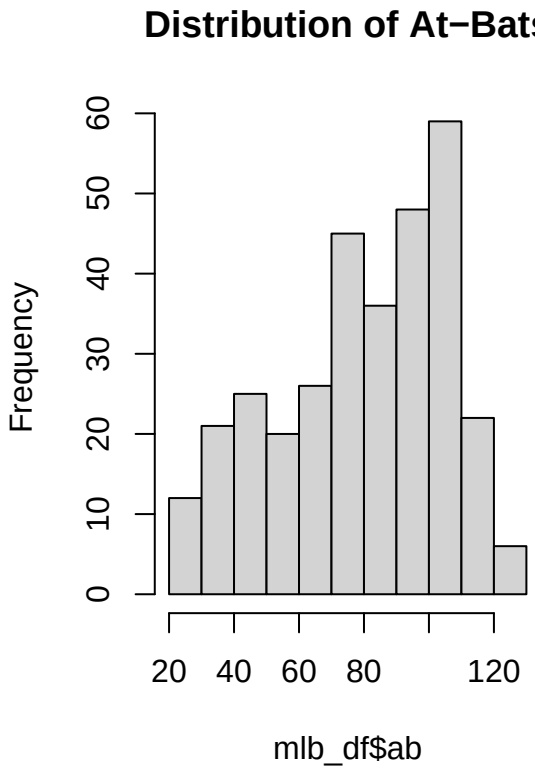
```
correlations = cor(mlb_df)
ordered_fs_obp_correlations = correlations["fs_obp", order(abs(correlations["fs_obp",]), decreasing = TR
print(ordered_fs_obp_correlations)
```

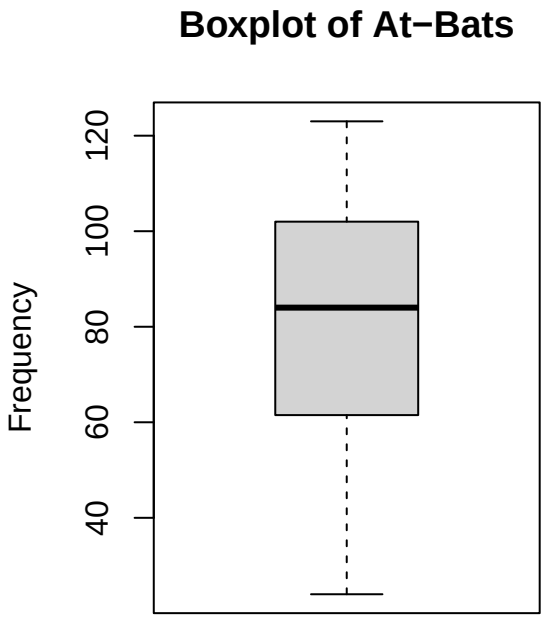
```
##      fs_obp         obp           r         avg           h         slg
##   1.0000000   0.6013535   0.4621977   0.4465277   0.4349363   0.4348545
##      bb_pct         rbi          ab contact_pct      ld_pct    iffb_pct
##   0.4156883   0.3967608   0.2622212   0.2557964   0.1831951  -0.1567309
##      gb_pct
##  -0.1177898
```

We see that obp, r, avg, h, slg, and bb_pct are most correlated variables with fs_obp.
We then see that gb_pct, iffb_pct, ld_pct, and contact_pct are least correlated variables with fs_obp.
We will most likely not be using some of these predictors that are not heavily correlated with fs_obp. We will see this when we run regression models.

Let's look at histogram and boxplot of each of the 12 predictor to see how each predictor is distributed. We can also see if we want to transform any of the predictors in our linear regression analysis. We tried doing a log and a square root transformation on each predictor.

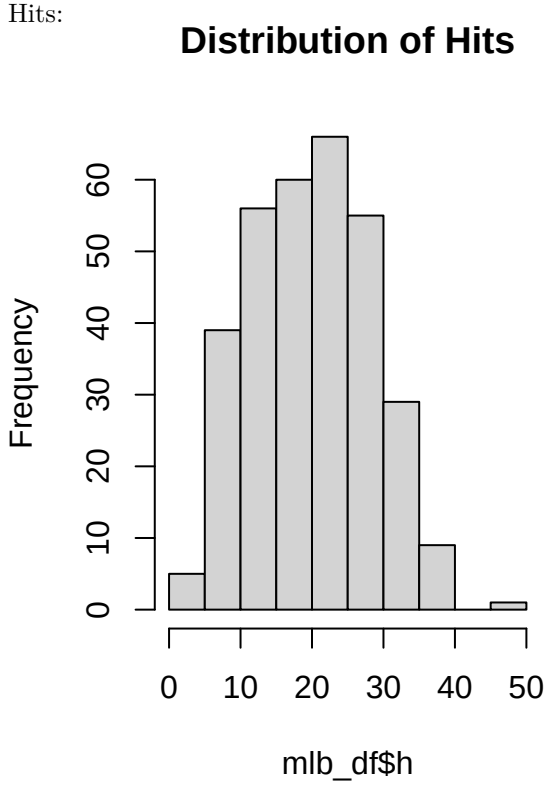
At bats:

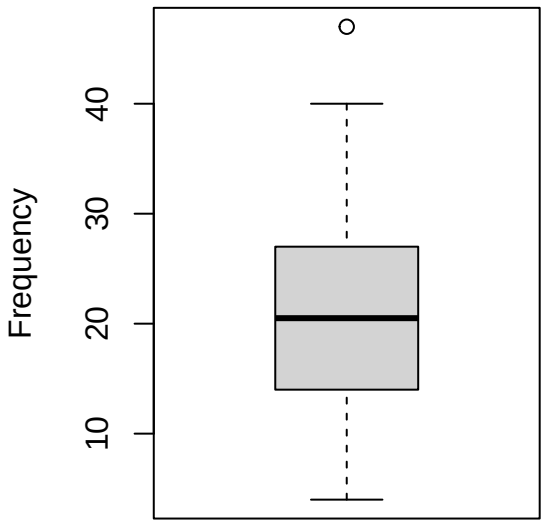




The at-bats predictor is slightly skewed left. There are no outliers either. We tried both a log and square root transformation to make distribution more normal, but were unable to do so. We decided against transforming this predictor.

Hits:

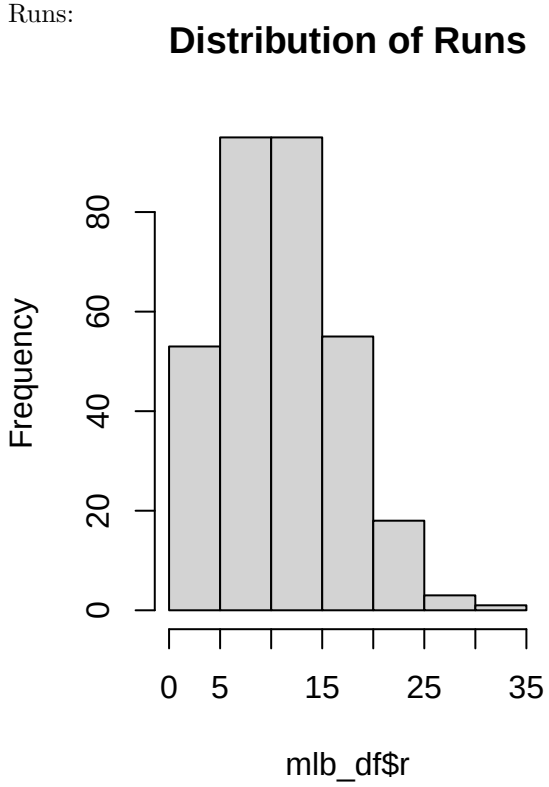


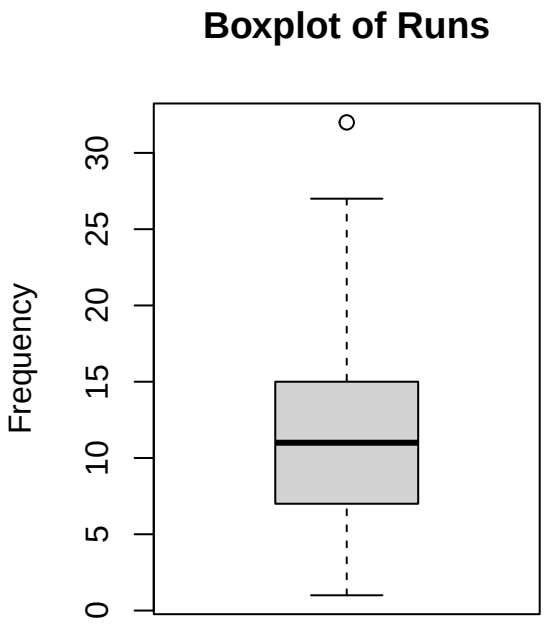


The hits predictor has relativey normal distribution and one outlier. We decided not to transform this

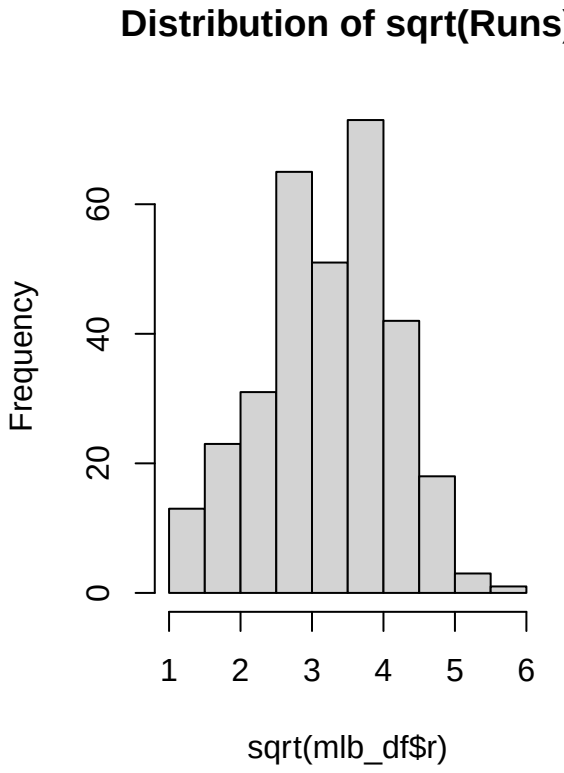
variable.

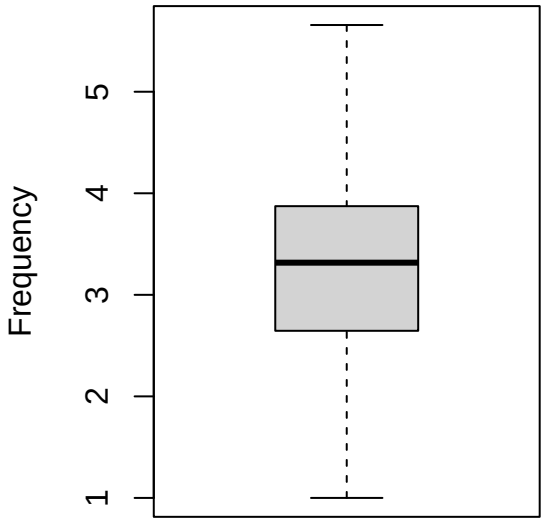
Runs:

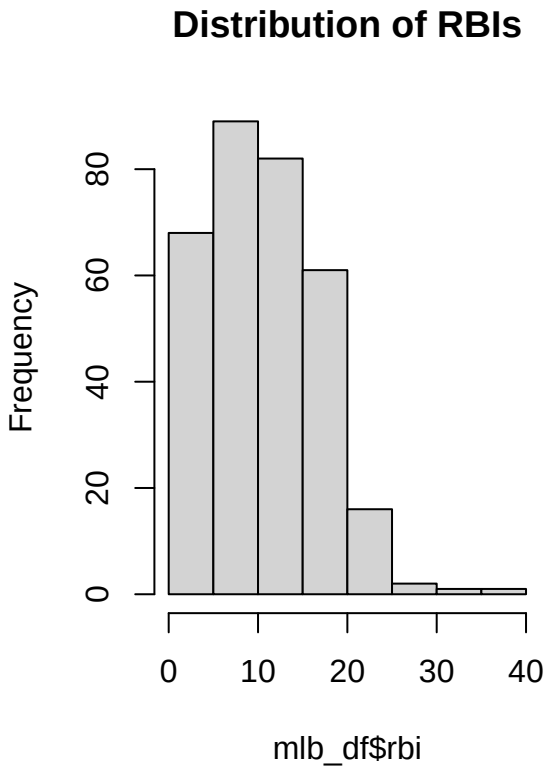


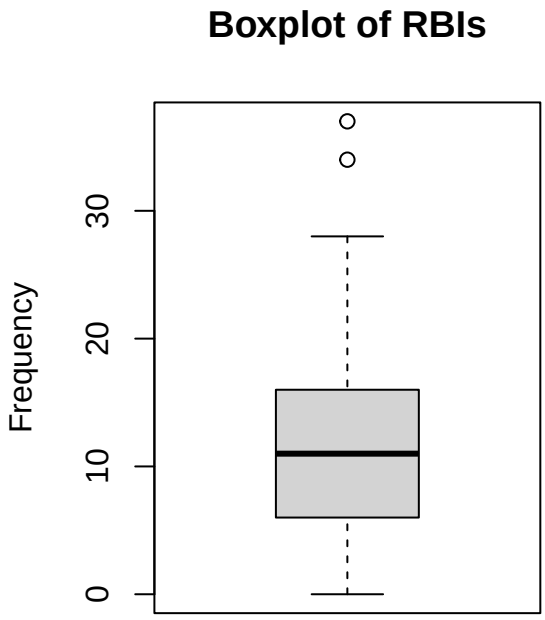


The distributions of runs is slightly skewed right and there is one outlier. We found that a square root transformation made the distribution more normal and created no outliers.



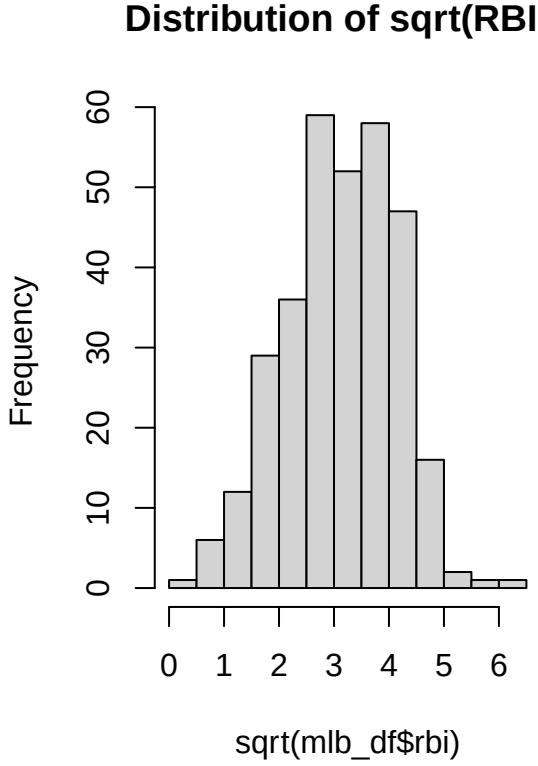


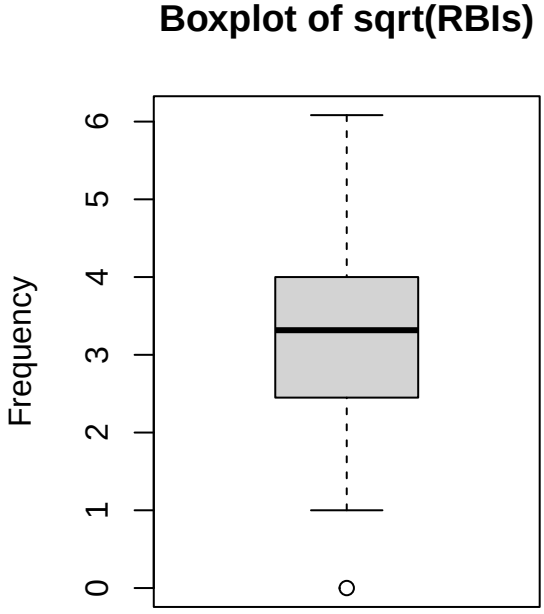




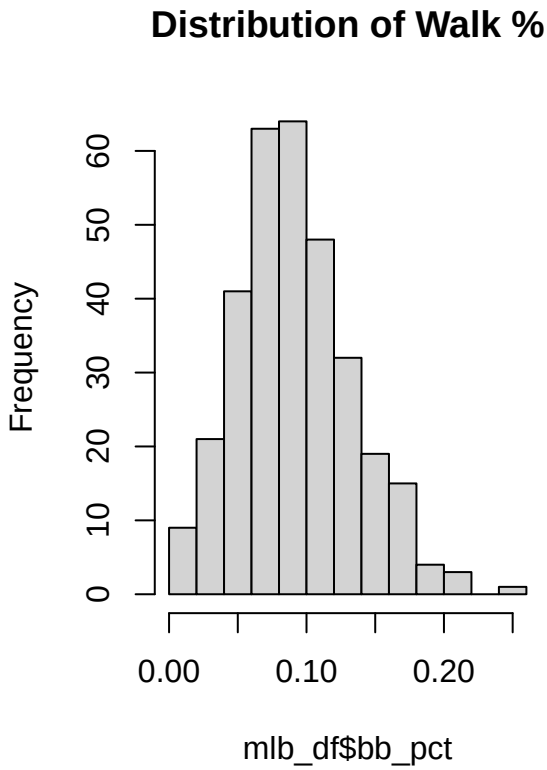
RBIs:

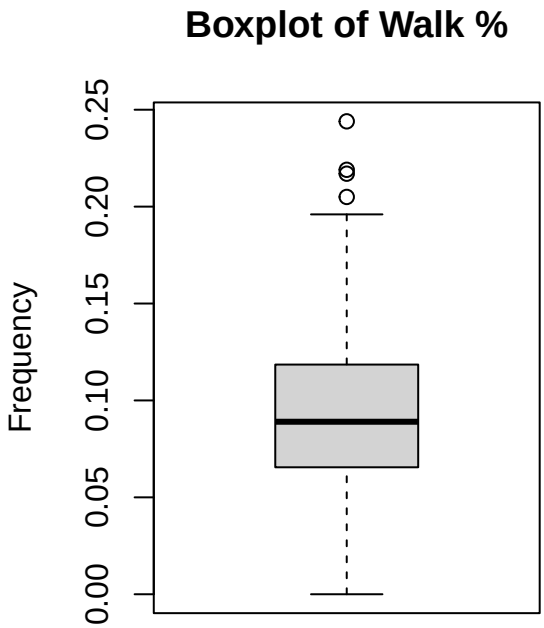
The distribution of RBIs is slightly skewed right and has two outliers. We found that a square root transformation created a more normal distribution. There was still one outlier when we transformed the predictor.



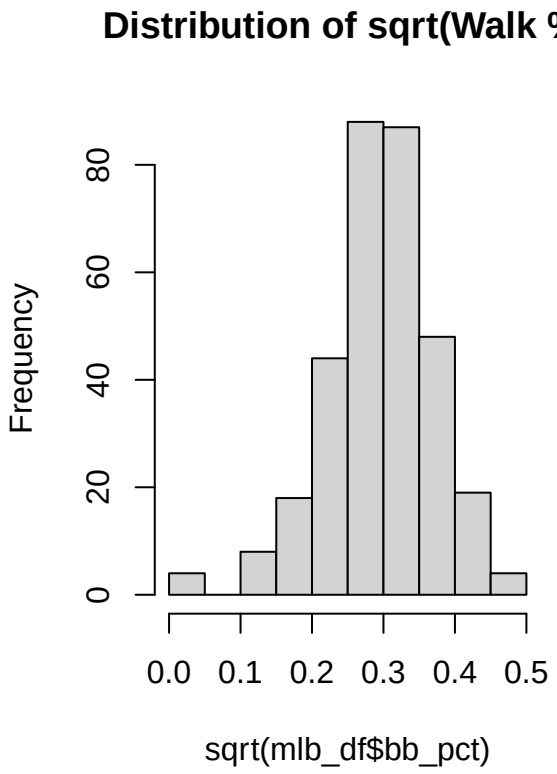


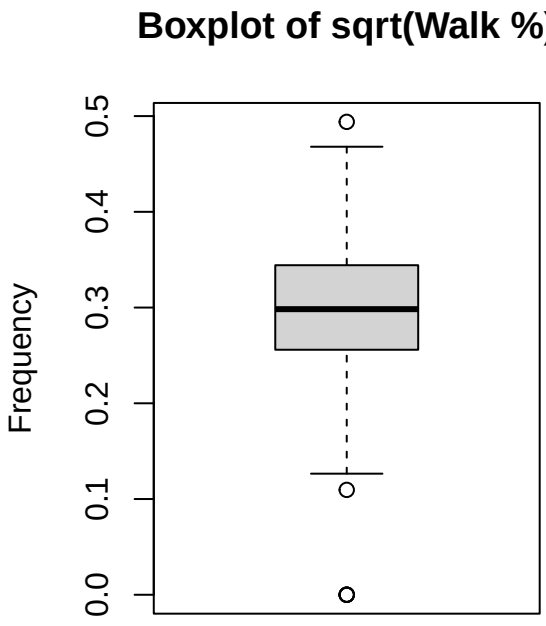
Walk percentage:



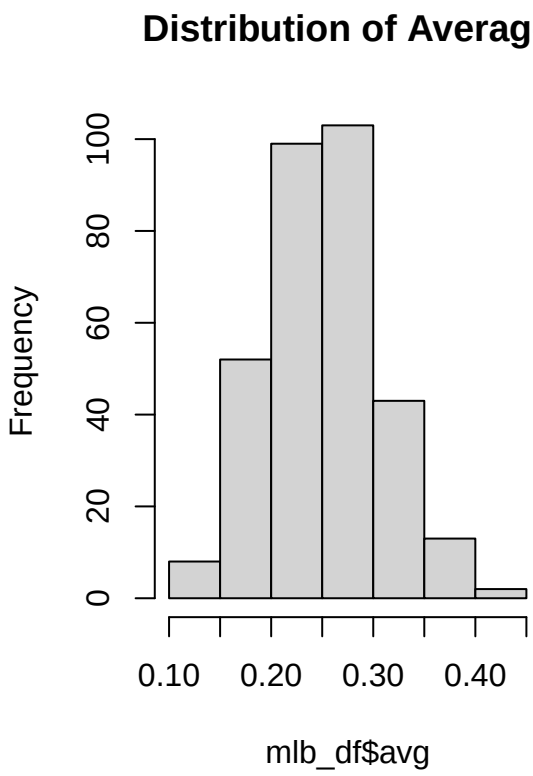


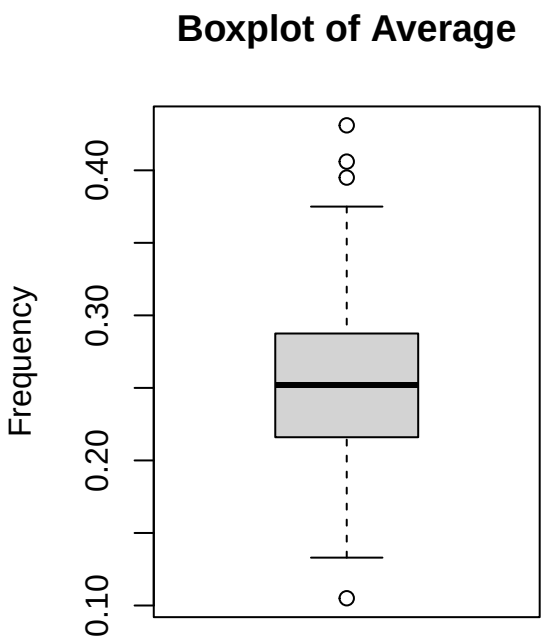
The distribution of walk percentage is relatively normal. There are a few outliers that can be seen. We found that a square root transformation kept the distribution normal and reduced the outliers down to only three.



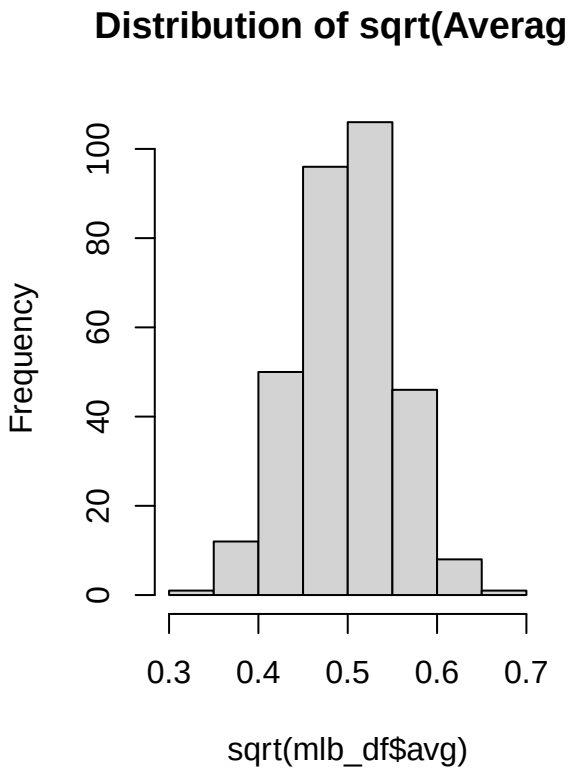


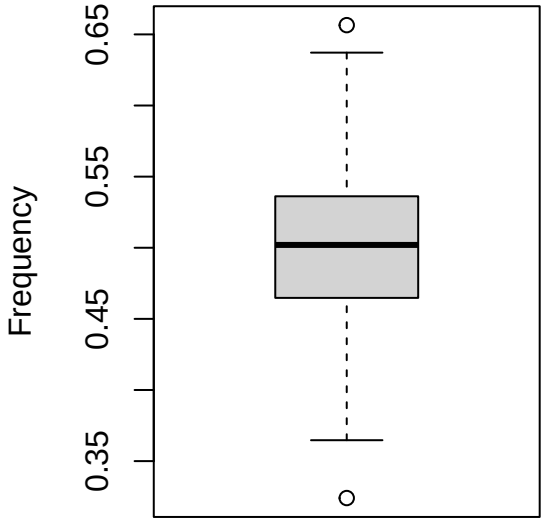
Average:



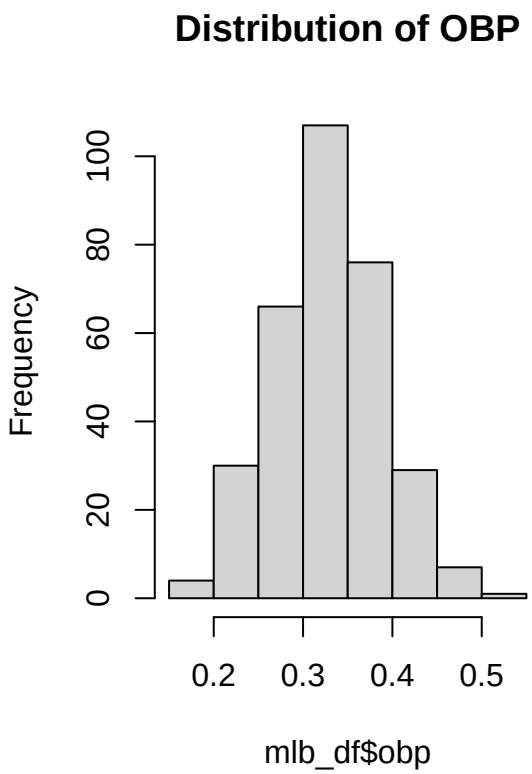


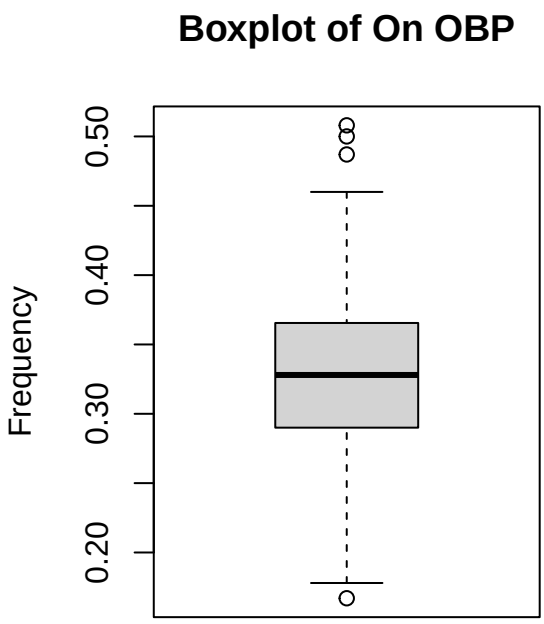
The distribution of average is relatively normal. There are four outliers that can be seen. We found that a square root transformation kept the distribution normal, and reduced the outliers down to two.



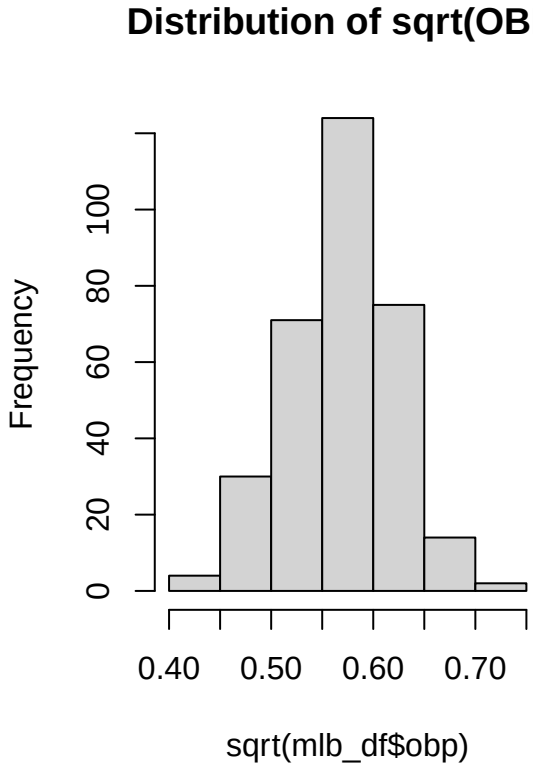


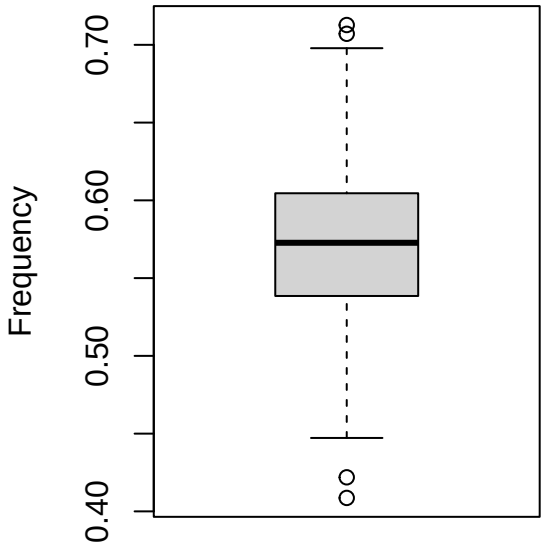
OBP:



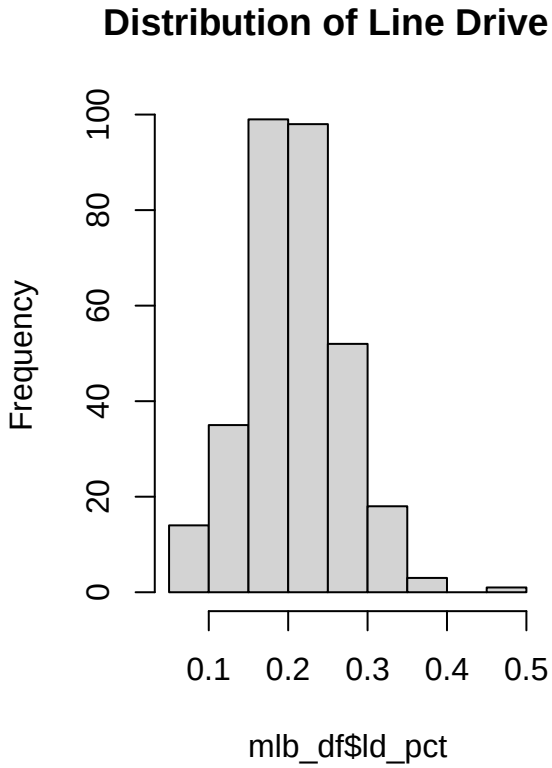


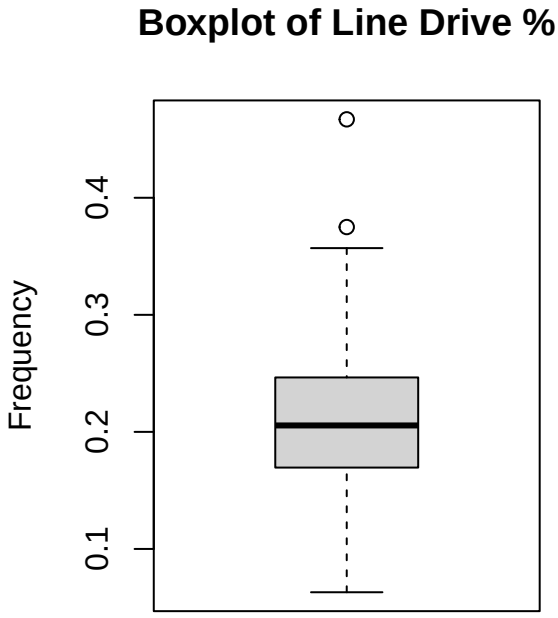
The distribution of OBP is relatively normal. There are four outliers that can be seen. We found that a square root transformation kept the distribution normal. It did not remove the four outliers but brought them closer to our uper and lower whiskers.





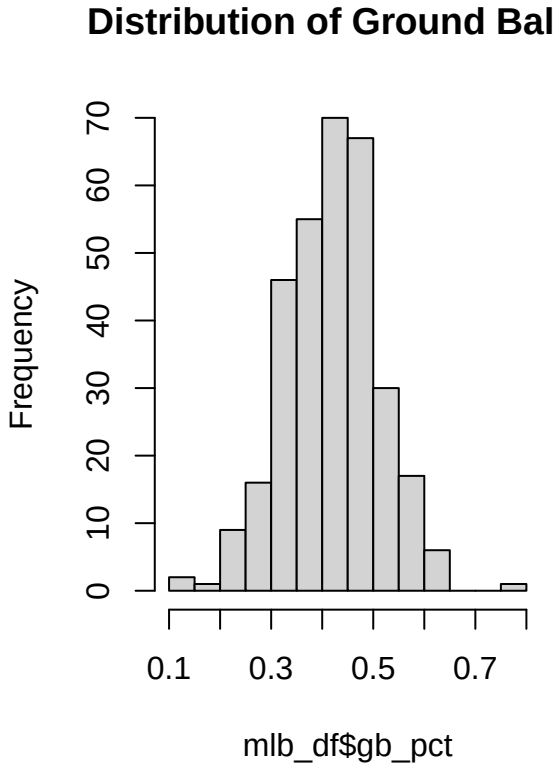
Line drive percentage:

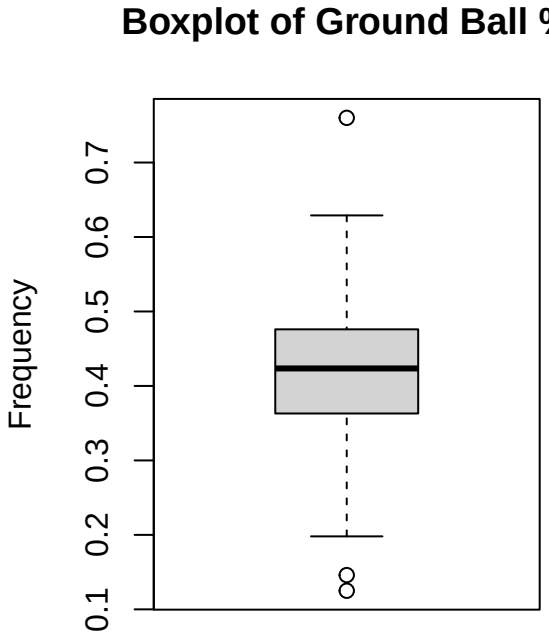




The distribution of line drive percentage predictor is relatively normal. We can see that we have two outlies. We tried both a log and square root transformation and found that neither made the distribution normal or handled those outliers.

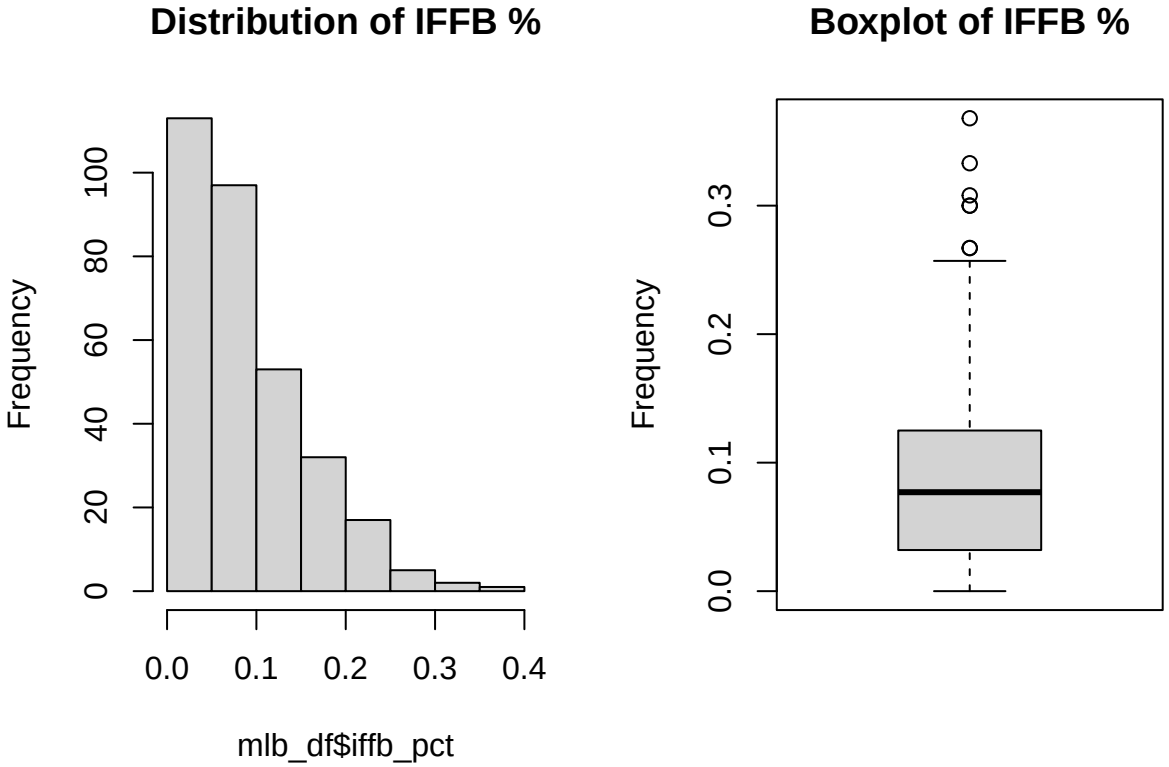
Ground ball percentage:



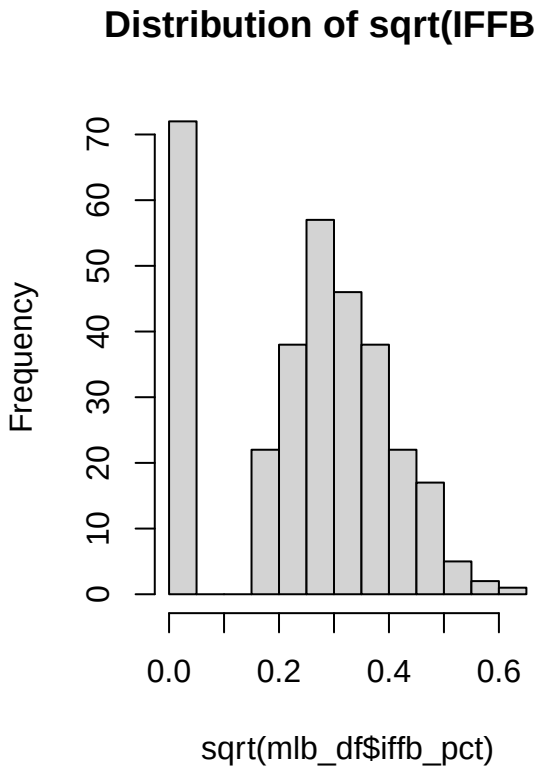


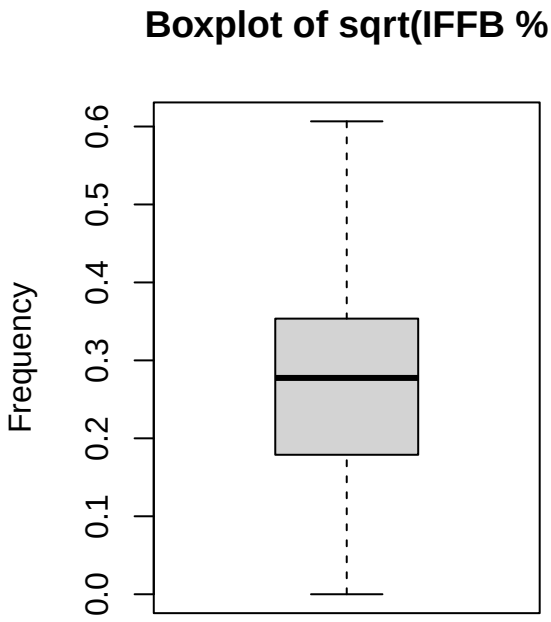
Distribution of ground ball percentage is relatively normal. We see three outliers. After seeing that the log and square root transformation did not help with distribution or outliers, we decided not to transform this predictor.

Infield fly ball percentage:

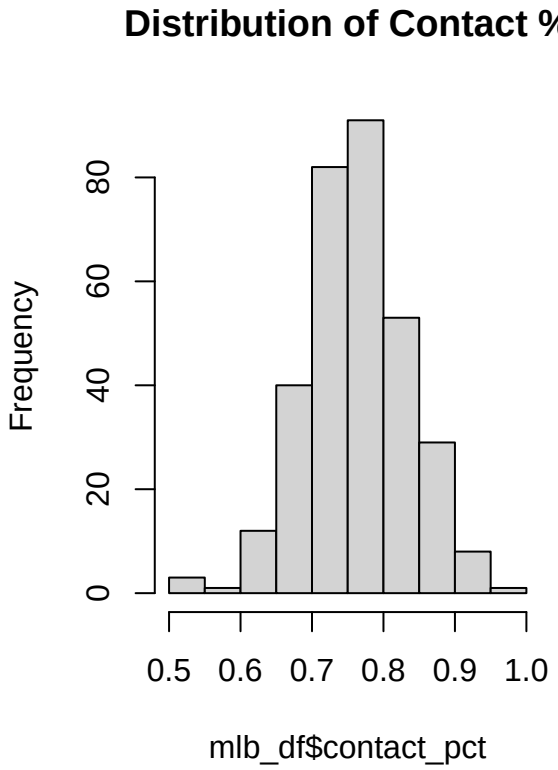


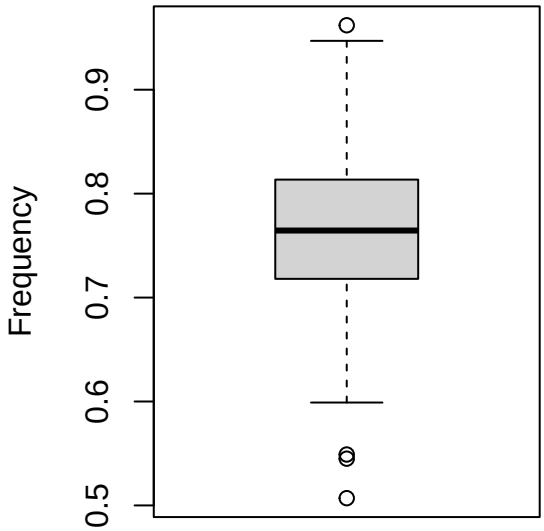
We can see that the distribution of infield fly ball percentage is skewed heavily to the right with five outliers. We found that the square root transformation had the best affect on the normality of the distribution, compared to how left skewed it was.





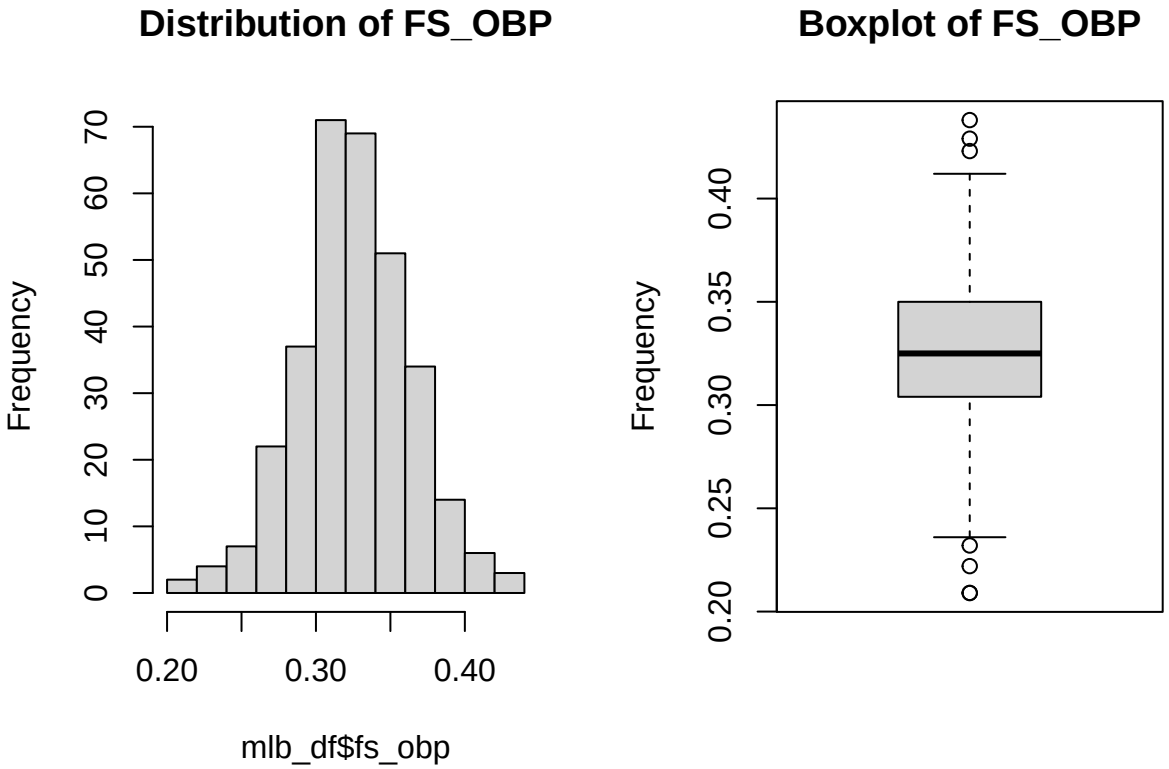
Contact percentage:





The contact percentage predictor has relatively normal distribution with four outliers. We found that the log and square root transformation did not help make the distribution more normal nor did it help with the outliers. We decided not to transform this variable.

Let's look at the distribution of our full season OBP response variable:



Our response variable follows a relatively normal distribution. We do see that there are six outliers. Since we transformed our predictors, we decided against predicting 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.

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 traning and test data set to see how well a trained model can predict the full season OBP of our test data. Let us start with looking at a linear model on the entire data set.

Linear Regression Model

Let us run linear regression model on entire data set.

```
fs_obp_lm = lm(fs_obp ~., mlb_df)
summary(fs_obp_lm)
```

```
##
## Call:
## lm(formula = fs_obp ~ ., data = mlb_df)
##
## Residuals:
```

```
##       Min        1Q    Median        3Q       Max
## -0.075292 -0.021070  0.000946  0.019394  0.094191
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.1752275  0.0327906   5.344 1.78e-07 ***
## ab          -0.0004781  0.0002914  -1.641 0.101797
## h            0.0023234  0.0011799   1.969 0.049823 *
## r            0.0004653  0.0005887   0.790 0.429898
## rbi          0.0005311  0.0004905   1.083 0.279708
## bb_pct       0.0557555  0.1171349   0.476 0.634417
## avg         -0.3246093  0.1607088  -2.020 0.044266 *
## obp          0.3568579  0.1496640   2.384 0.017714 *
## slg          0.0334318  0.0327951   1.019 0.308810
## ld_pct       0.0099081  0.0320109   0.310 0.757133
## gb_pct       0.0143149  0.0221508   0.646 0.518603
## iffb_pct    -0.0272898  0.0226609  -1.204 0.229414
## contact_pct  0.0898009  0.0236749   3.793 0.000179 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02818 on 307 degrees of freedom
## Multiple R-squared:  0.4602, Adjusted R-squared:  0.4391
## F-statistic: 21.81 on 12 and 307 DF,  p-value: < 2.2e-16
```

We are looking to see which predictors are statistically significant when predicting full season OBP. 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 can see that many of the variables have p-values that are greater than 0.05: ab, r, rbi, bb_pct, slg, ld_pct, gb_pct, iffb_pct. We see variables with p-values less than 0.05: h, avg, obp, contact_pct.
Let's run model using h, avg, obp, contact_pct.

```
fs_obp_lm = lm(fs_obp ~ h + avg + obp + contact_pct, mlb_df)
summary(fs_obp_lm)
```

```
##
## Call:
## lm(formula = fs_obp ~ h + avg + obp + contact_pct, data = mlb_df)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.082792 -0.020303 -0.000462  0.020199  0.100979
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.1501187  0.0184007   8.158 8.19e-15 ***
## h            0.0011562  0.0002402   4.813 2.32e-06 ***
## avg         -0.2395798  0.0567897  -4.219 3.22e-05 ***
## obp          0.4660259  0.0469992   9.916  < 2e-16 ***
## contact_pct  0.0770035  0.0229925   3.349 0.000909 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 0.02843 on 315 degrees of freedom
## Multiple R-squared:  0.4365, Adjusted R-squared:  0.4293
## F-statistic:    61 on 4 and 315 DF,  p-value: < 2.2e-16
```

We see that all of the predictors in this model have p-values less than 0.05. WE see that there is a R^2 value of approximately 0.44 meaning that this model explains about 44% of the variance in the data set.
Of the predictors that we have in this model, let us add our transformations that we went with in our exploratory analysis to help make distributions of predictors more normal.
Earlier, we decided to take square root of both avg and obp to help make predictors more normal.

```
fs_obp_lm = lm(fs_obp ~ h + sqrt(avg) + sqrt(obp) + contact_pct, mlb_df)
summary(fs_obp_lm)
```

```
##
## Call:
## lm(formula = fs_obp ~ h + sqrt(avg) + sqrt(obp) + contact_pct,
##     data = mlb_df)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.083719 -0.020670 -0.000386  0.019980  0.100027
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.0611401  0.0241432   2.532 0.011814 *
## h            0.0011291  0.0002446   4.616 5.71e-06 ***
## sqrt(avg)   -0.2230389  0.0573112  -3.892 0.000122 ***
## sqrt(obp)    0.5139255  0.0539040   9.534  < 2e-16 ***
## contact_pct  0.0772777  0.0231759   3.334 0.000957 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02866 on 315 degrees of freedom
## Multiple R-squared:  0.4272, Adjusted R-squared:  0.4199
## F-statistic: 58.72 on 4 and 315 DF,  p-value: < 2.2e-16
```

These transformations did not create any major improvements on our data. Since there was no major statistical impact on our model with the transformations, let us not transform avg and obp predictors to maintain simplicity.

Let us add in the variables that we initially removed, one by one to see if their respective p-values are less than 0.05: ab, r, rbi, bb_pct, slg, ld_pct, gb_pct, iffb_pct.
If we transformed the predictors we are adding in earlier in our analysis, do the same here.
ab:

```
fs_obp_lm = lm(fs_obp ~ h + avg + obp + contact_pct + ab, mlb_df)
summary(fs_obp_lm)
```

```
##
## Call:
## lm(formula = fs_obp ~ h + avg + obp + contact_pct + ab, data = mlb_df)
##
```

```
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.081423 -0.020171 -0.000285  0.019775  0.100422
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.1815279  0.0280523   6.471 3.74e-10 ***
## h            0.0027506  0.0011027   2.494 0.013134 *
## avg         -0.3469717  0.0920282  -3.770 0.000195 ***
## obp          0.4607716  0.0470442   9.794  < 2e-16 ***
## contact_pct  0.0757938  0.0229635   3.301 0.001076 **
## ab          -0.0004287  0.0002894  -1.481 0.139540
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02837 on 314 degrees of freedom
## Multiple R-squared:  0.4404, Adjusted R-squared:  0.4315
## F-statistic: 49.42 on 5 and 314 DF,  p-value: < 2.2e-16
```

The p-value of ab is greater than 0.05. Let's not add this predictor.

sqrt(r):

```
fs_obp_lm = lm(fs_obp ~ h + avg + obp + contact_pct + sqrt(r), mlb_df)
summary(fs_obp_lm)
```

```
##
## Call:
## lm(formula = fs_obp ~ h + avg + obp + contact_pct + sqrt(r),
##     data = mlb_df)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.077677 -0.020666  0.000067  0.019964  0.102028
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.1434860  0.0191333   7.499 6.62e-13 ***
## h            0.0007509  0.0004032   1.862 0.063489 .
## avg         -0.2110383  0.0611529  -3.451 0.000635 ***
## obp          0.4436598  0.0502446   8.830  < 2e-16 ***
## contact_pct  0.0785622  0.0230056   3.415 0.000722 ***
## sqrt(r)      0.0042432  0.0033914   1.251 0.211806
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0284 on 314 degrees of freedom
## Multiple R-squared:  0.4393, Adjusted R-squared:  0.4304
## F-statistic:  49.2 on 5 and 314 DF,  p-value: < 2.2e-16
```

The p-value of sqrt(r) is greater than 0.05. Let's not add this predictor.

sqrt(rbi):

```r
fs_obp_lm = lm(fs_obp ~ h + avg + obp + contact_pct + sqrt(rbi), mlb_df)
summary(fs_obp_lm)
```

```
##
## Call:
## lm(formula = fs_obp ~ h + avg + obp + contact_pct + sqrt(rbi),
##     data = mlb_df)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.078734 -0.020246  0.000959  0.020265  0.099615
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.1367093  0.0189314   7.221 3.91e-12 ***
## h            0.0006049  0.0003173   1.907 0.057493 .
## avg         -0.2221664  0.0566537  -3.921 0.000108 ***
## obp          0.4550097  0.0467531   9.732  < 2e-16 ***
## contact_pct  0.0824586  0.0228744   3.605 0.000363 ***
## sqrt(rbi)    0.0061924  0.0023567   2.628 0.009021 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02816 on 314 degrees of freedom
## Multiple R-squared:  0.4486, Adjusted R-squared:  0.4398
## F-statistic:  51.1 on 5 and 314 DF,  p-value: < 2.2e-16
```

Although the p-value of sqrt(rbi) is less than 0.05, it made our h predictor no longer statistically significant as the p-value of h is now greater than 0.05. Let us try and add just the rbi predictor without the transformation.

```r
fs_obp_lm = lm(fs_obp ~ h + avg + obp + contact_pct + rbi, mlb_df)
summary(fs_obp_lm)
```

```
##
## Call:
## lm(formula = fs_obp ~ h + avg + obp + contact_pct + rbi, data = mlb_df)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.079237 -0.020423  0.001169  0.020063  0.099874
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.1453850  0.0183325   7.930 3.87e-14 ***
## h            0.0006337  0.0003133   2.023 0.043955 *
## avg         -0.2179495  0.0569194  -3.829 0.000155 ***
## obp          0.4497381  0.0470176   9.565  < 2e-16 ***
## contact_pct  0.0833983  0.0229269   3.638 0.000322 ***
## rbi          0.0009389  0.0003658   2.567 0.010717 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.02818 on 314 degrees of freedom
## Multiple R-squared:  0.4481, Adjusted R-squared:  0.4393
## F-statistic: 50.98 on 5 and 314 DF,  p-value: < 2.2e-16
```

In this model, both rbi and h are statistically significant as each of their p-values are less than 0.05.

sqrt(bb_pct):

```
fs_obp_lm = lm(fs_obp ~ h + avg + obp + contact_pct + rbi + sqrt(bb_pct), mlb_df)
summary(fs_obp_lm)
```

```
##
## Call:
## lm(formula = fs_obp ~ h + avg + obp + contact_pct + rbi + sqrt(bb_pct),
##     data = mlb_df)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.076969 -0.019915  0.001002  0.019293  0.104088
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.1539332  0.0188164   8.181 7.15e-15 ***
## h             0.0007562  0.0003188   2.372 0.018284 *
## avg          -0.4130081  0.1182211  -3.494 0.000545 ***
## obp           0.6592448  0.1208665   5.454 1.00e-07 ***
## contact_pct   0.0819002  0.0228487   3.584 0.000392 ***
## rbi           0.0009403  0.0003643   2.581 0.010301 *
## sqrt(bb_pct) -0.1005271  0.0534653  -1.880 0.061005 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02806 on 313 degrees of freedom
## Multiple R-squared:  0.4542, Adjusted R-squared:  0.4438
## F-statistic: 43.42 on 6 and 313 DF,  p-value: < 2.2e-16
```

The p-value of sqrt(bb_pct) is greater than 0.05. Let's not add this predictor.

slg:

```
fs_obp_lm = lm(fs_obp ~ h + avg + obp + contact_pct + rbi + slg, mlb_df)
summary(fs_obp_lm)
```

```
##
## Call:
## lm(formula = fs_obp ~ h + avg + obp + contact_pct + rbi + slg,
##     data = mlb_df)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.078747 -0.019853  0.001048  0.020611  0.096137
##
## Coefficients:
```

```
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.1425002  0.0185439   7.684 1.99e-13 ***
## h            0.0007935  0.0003496   2.269 0.023921 *
## avg         -0.2625162  0.0715170  -3.671 0.000284 ***
## obp          0.4433099  0.0474264   9.347  < 2e-16 ***
## contact_pct  0.0887972  0.0235173   3.776 0.000191 ***
## rbi          0.0006110  0.0004850   1.260 0.208696
## slg          0.0285366  0.0277298   1.029 0.304229
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02817 on 313 degrees of freedom
## Multiple R-squared:  0.4499, Adjusted R-squared:  0.4394
## F-statistic: 42.67 on 6 and 313 DF,  p-value: < 2.2e-16
```

The p-value of slg is greater than 0.05. Adding slg also made the p-value of rbi greater than 0.05. Let's not add this predictor.

ld_pct:

```
fs_obp_lm = lm(fs_obp ~ h + avg + obp + contact_pct + rbi + slg, mlb_df)
summary(fs_obp_lm)
```

```
##
## Call:
## lm(formula = fs_obp ~ h + avg + obp + contact_pct + rbi + slg,
##     data = mlb_df)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.078747 -0.019853  0.001048  0.020611  0.096137
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.1425002  0.0185439   7.684 1.99e-13 ***
## h            0.0007935  0.0003496   2.269 0.023921 *
## avg         -0.2625162  0.0715170  -3.671 0.000284 ***
## obp          0.4433099  0.0474264   9.347  < 2e-16 ***
## contact_pct  0.0887972  0.0235173   3.776 0.000191 ***
## rbi          0.0006110  0.0004850   1.260 0.208696
## slg          0.0285366  0.0277298   1.029 0.304229
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02817 on 313 degrees of freedom
## Multiple R-squared:  0.4499, Adjusted R-squared:  0.4394
## F-statistic: 42.67 on 6 and 313 DF,  p-value: < 2.2e-16
```

The p-value of ld_pct is greater than 0.05. Let's not add this predictor.

gb_pct:

```r
fs_obp_lm = lm(fs_obp ~ h + avg + obp + contact_pct + rbi + gb_pct, mlb_df)
summary(fs_obp_lm)
```

```
##
## Call:
## lm(formula = fs_obp ~ h + avg + obp + contact_pct + rbi + gb_pct,
##     data = mlb_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.07921 -0.02049  0.00115  0.02034  0.10035
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.1431819  0.0200893   7.127  7.1e-12 ***
## h            0.0006241  0.0003157   1.977 0.048925 *
## avg         -0.2181263  0.0570074  -3.826 0.000157 ***
## obp          0.4513971  0.0474859   9.506  < 2e-16 ***
## contact_pct  0.0829094  0.0230320   3.600 0.000370 ***
## rbi          0.0009580  0.0003730   2.568 0.010685 *
## gb_pct       0.0048834  0.0180758   0.270 0.787215
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02822 on 313 degrees of freedom
## Multiple R-squared:  0.4482, Adjusted R-squared:  0.4376
## F-statistic: 42.37 on 6 and 313 DF,  p-value: < 2.2e-16
```

The p-value of gb_pct is greater than 0.05. Let's not add this predictor.

sqrt(iffb_pct):

```r
fs_obp_lm = lm(fs_obp ~ h + avg + obp + contact_pct + rbi + sqrt(iffb_pct), mlb_df)
summary(fs_obp_lm)
```

```
##
## Call:
## lm(formula = fs_obp ~ h + avg + obp + contact_pct + rbi + sqrt(iffb_pct),
##     data = mlb_df)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.077551 -0.020698  0.000222  0.019660  0.096487
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)     0.1506136  0.0185449   8.122 1.07e-14 ***
## h               0.0006720  0.0003132   2.145 0.032685 *
## avg            -0.2322418  0.0573949  -4.046 6.56e-05 ***
## obp             0.4478900  0.0468961   9.551  < 2e-16 ***
## contact_pct     0.0862354  0.0229240   3.762 0.000201 ***
## rbi             0.0009701  0.0003652   2.656 0.008301 **
## sqrt(iffb_pct) -0.0174038  0.0103936  -1.674 0.095038 .
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0281 on 313 degrees of freedom
## Multiple R-squared:  0.453,  Adjusted R-squared:  0.4425
## F-statistic:  43.2 on 6 and 313 DF,  p-value: < 2.2e-16
```

The p-value of sqrt(iffb_pct) is greater than 0.05. Let's not add this predictor.

Of the predictors that we removed from our initial linear model, we only added back the rbi predictor. Now, let us have the fitted model on entire data set predict fs_obp and see how well it performs by checking mean squared error.

```
fs_obp_lm = lm(fs_obp ~ h + avg + obp + contact_pct + rbi, mlb_df)
pred_fs_obp_lm = predict(fs_obp_lm, newdata = mlb_df)
mean((pred_fs_obp_lm - mlb_df$fs_obp)^2)
```

```
## [1] 0.0007790778
```

The mean squared error is 0.0008. However, this was using the model fitted on the entire data set to predict fs_obp. So there is probably a good chance that our model is overfitting the data.

Training and Test Data

Now, let's create training and test data from mlb_df. We can then use fitted model on training data to predict test data.

```
dim(mlb_df)
```

```
## [1] 320  13
```

As we know, there are 320 observations.
Since we only have 320 observations, our parameter estimates will have higher variance. Since we have a 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(mlb_df$fs_obp, p = 0.7, list = FALSE)
mlb_train = mlb_df[index,]
mlb_test = mlb_df[-index,]
```

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

```
dim(mlb_train)
```

```
## [1] 226  13
```

There are 226 observations in our training set.

Test set:

```
dim(mlb_test)
```

```
## [1] 94 13
```

There are 94 observations in our test set. Now let us go back to our linear regression analysis and use the training data to refit our model.

Linear Regression Model Continued

Fit our model from earlier using the training data.

```
fs_obp_lm = lm(fs_obp ~ h + avg + obp + contact_pct + rbi, mlb_train)
summary(fs_obp_lm)
```

```
##
## Call:
## lm(formula = fs_obp ~ h + avg + obp + contact_pct + rbi, data = mlb_train)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.074260 -0.020787  0.000591  0.017599  0.098345
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.1567509  0.0211988   7.394 2.95e-12 ***
## h            0.0007039  0.0003517   2.002   0.0466 *
## avg         -0.2371433  0.0671193  -3.533   0.0005 ***
## obp          0.4911045  0.0552512   8.889 2.27e-16 ***
## contact_pct  0.0581611  0.0265330   2.192   0.0294 *
## rbi          0.0007617  0.0004082   1.866   0.0633 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02697 on 220 degrees of freedom
## Multiple R-squared:  0.499,  Adjusted R-squared:  0.4877
## F-statistic: 43.83 on 5 and 220 DF,  p-value: < 2.2e-16
```

All of the predictors have p-values less than 0.05, besides rbi. Since we initially removed rbi, then added it back in, let us remove rbi again.

```
fs_obp_lm = lm(fs_obp ~ h + avg + obp + contact_pct, mlb_train)
summary(fs_obp_lm)
```

```
##
## Call:
## lm(formula = fs_obp ~ h + avg + obp + contact_pct, data = mlb_train)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.073101 -0.019640  0.001924  0.015876  0.099042
```

```
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.1598489  0.0212521   7.522 1.35e-12 ***
## h            0.0011069  0.0002792   3.965 9.93e-05 ***
## avg         -0.2537051  0.0669027  -3.792 0.000193 ***
## obp          0.5083690  0.0547764   9.281  < 2e-16 ***
## contact_pct  0.0522602  0.0264914   1.973 0.049774 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02712 on 221 degrees of freedom
## Multiple R-squared:  0.4911, Adjusted R-squared:  0.4819
## F-statistic: 53.32 on 4 and 221 DF,  p-value: < 2.2e-16
```

Now, all of the predictors have p-values less than 0.05. The contact_pct predictor's p-value just just barely came in less than 0.05. So we will keep it.
The R^2 value is 0.4911 meaning our trained model explains 49% of the variability in our training data.

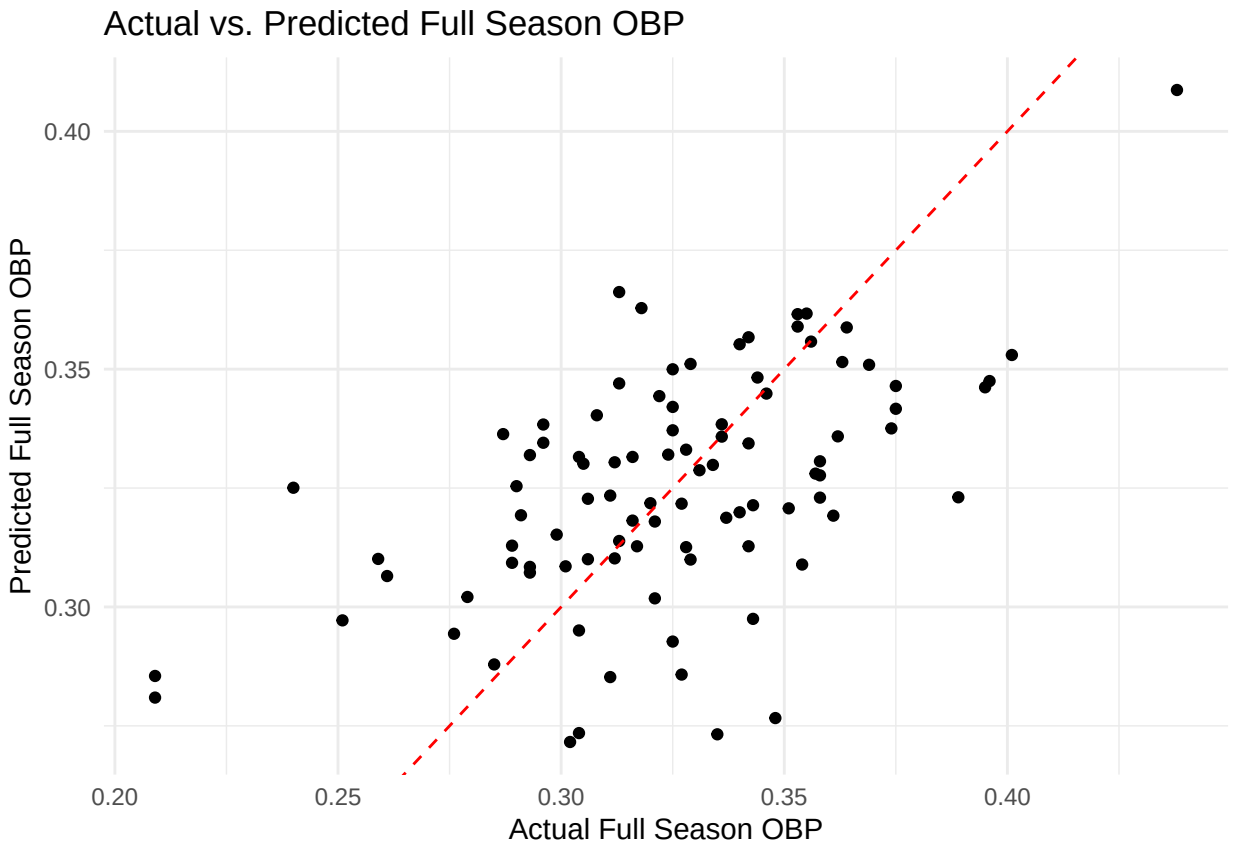
Let us use this model to predict the fs_obp value in our test data and check the mean squared error.

```
pred_fs_obp_lm = predict(fs_obp_lm, newdata = mlb_test)
mean((pred_fs_obp_lm - mlb_test$fs_obp)^2)
```

```
## [1] 0.0009979314
```

The mean squared error value is 0.00099, which is very close to 0. Which is what we want. We want our test mean squared error rate to be as close to 0 as possible.
This is our first fitted model on the training data that was used to predict test data. We will use this test mean squared error rate to compare other regression models that we fit. This is how we will see if other models have more accurate predictions.
Below you will see a plot of our predicted versus actual values of fs_obp.

```
par(mfrow = c(1, 1))
lm_plot = data.frame(Actual = mlb_test$fs_obp, Predicted = pred_fs_obp_lm)
ggplot(lm_plot, aes(x = Actual, y = Predicted)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red", linetype = "dashed") +  # Adds a line of perfect
  labs(x = "Actual Full Season OBP",
       y = "Predicted Full Season OBP",
       title = "Actual vs. Predicted Full Season OBP") +
  theme_minimal()
```



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 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 = 0.00247. Using lambda = 0.00247 in my ridge regression model using the training matrix to predict the response fs_obp values in the test data, I found a test mean squared error of 0.001. This model produced the same tese mean squared error as our linear regression model.

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

```
## [1] 0.002474503
```

Lambda chosen by cross-validation is 0.00247.
Now we fit ridge regression model and make predictions.

```
fs_obp_ridge = glmnet(train_matrix, mlb_train$fs_obp, alpha = 0)
pred_fs_obp_ridge = predict(fs_obp_ridge, s = best_lam, newx = test_matrix)
# Find mean squared error:
mean((pred_fs_obp_ridge - mlb_test$fs_obp)^2)
```

```
## [1] 0.001041706
```

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 = 0.0000091. Using that lambda in the regression model, we get a test mean squared error value of 0.001. Similar to our ridge regresison model, our lasso regression model produced the same test mean squared error of our linear regression model.

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

```
## [1] 9.102162e-06
```

Lambda chosen by cross-validation is 0.0000091.
Now we fit lasso regression model and make predictions.

```
fs_obp_lasso = glmnet(train_matrix, mlb_train$fs_obp, alpha = 1)
pred_fs_obp_lasso = predict(fs_obp_lasso, s = best_lam, newx = test_matrix)
# Find mean squared error:
mean((pred_fs_obp_lasso - mlb_test$fs_obp)^2)
```

```
## [1] 0.001091332
```

Principal Components Regression

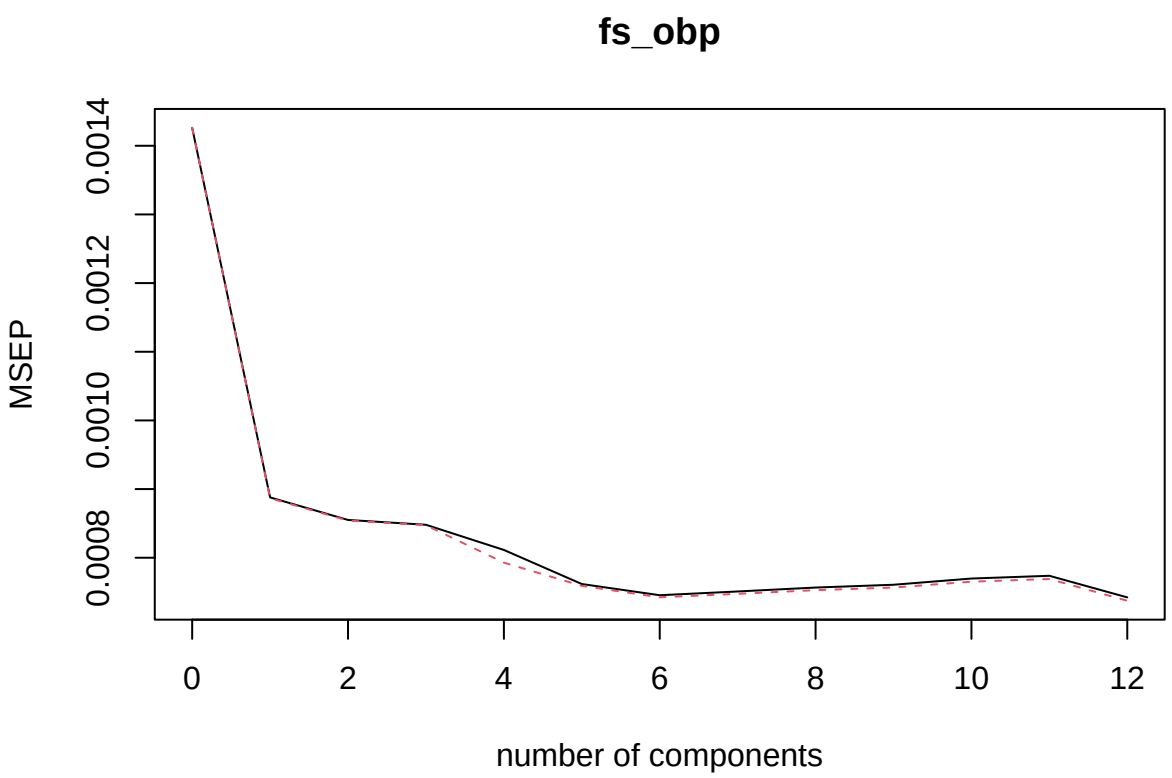
The main goal of principal components regression is to reduce the dimensions of the model by removing predictors to simplify the model. Let us fit a principal components regression model using the training data to then predict the response variable fs_obp in the test data.

```
fs_obp_pcr = pcr(fs_obp ~., data = mlb_train,
              scale = TRUE, validation = "CV")
summary(fs_obp_pcr)
```

```
## Data:    X dimension: 226 12
##  Y dimension: 226 1
## Fit method: svdpc
## Number of components considered: 12
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV         0.03776  0.02980  0.02924  0.02912  0.02848  0.02760  0.02730
## adjCV      0.03776  0.02978  0.02923  0.02911  0.02816  0.02755  0.02725
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps
## CV     0.02740  0.02751  0.02758   0.02774   0.02782   0.02724
## adjCV  0.02734  0.02744  0.02751   0.02766   0.02773   0.02715
##
```

```
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X         40.27    53.98    64.97    74.08    82.58    89.59    94.77    97.47
## fs_obp    38.81    41.92    42.90    49.49    49.50    51.04    51.31    51.47
##         9 comps  10 comps  11 comps  12 comps
## X         99.01     99.74     99.90    100.00
## fs_obp    51.49     51.52     51.52     53.83
```

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



We see that M = 6 produces lowest mean squared error of prediction on training data and explains 51% of the variance of the training data. Now, let us predict our test data.

```
pred_fs_obp_pcr = predict(fs_obp_pcr, mlb_test, ncomp = 6)
mean((pred_fs_obp_pcr - mlb_test$fs_obp)^2)
```

```
## [1] 0.001055973
```

From our principal component regression model, we get a test mean squared error value of 0.001. Similar to our ridge regression model and our lasso regression model, the produced the same test mean squared error of our linear regression model.

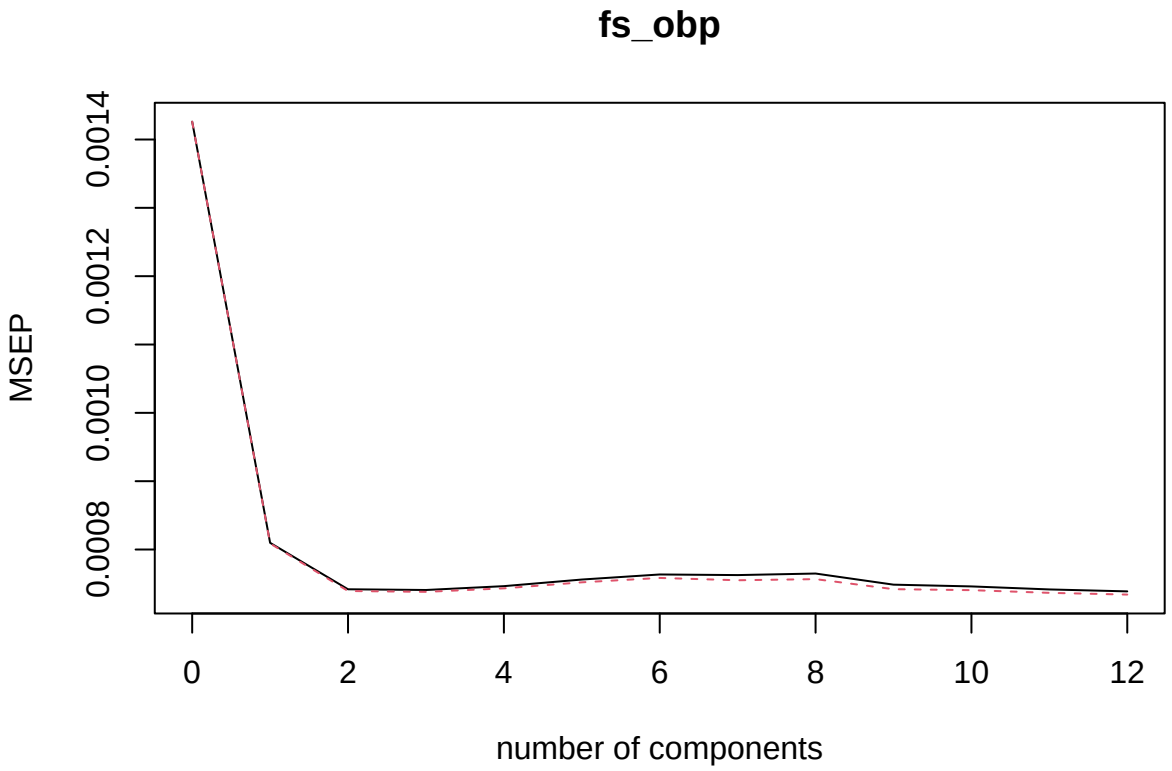
Partial Least Squares

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.

```r
fs_obp_plsr = plsr(fs_obp ~., data = mlb_train,
                   scale = TRUE, validation = "CV")
summary(fs_obp_plsr)
```

```
## Data:    X dimension: 226 12
##  Y dimension: 226 1
## Fit method: kernelpls
## Number of components considered: 12
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV         0.03776  0.02846  0.02724  0.02722  0.02732  0.02750  0.02763
## adjCV      0.03776  0.02844  0.02719  0.02717  0.02726  0.02742  0.02754
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps
## CV     0.02762  0.02766  0.02736   0.02731   0.02723   0.02718
## adjCV  0.02748  0.02751  0.02724   0.02721   0.02714   0.02709
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X         39.66    51.01    62.00    68.76    74.59    81.37    85.00    88.98
## fs_obp    44.41    51.10    51.53    51.71    52.01    52.42    53.21    53.57
##         9 comps  10 comps  11 comps  12 comps
## X         90.64     98.15     99.84    100.00
## fs_obp    53.78     53.80     53.83     53.83
```

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



We see that M = 10 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 = 3 as that has almost same MSEP as when M = 10.
The partial least squares model when M = 3 explains about 51% of the variance of training data.
Let us predict our test data.

```
pred_fs_obp_plsr = predict(fs_obp_plsr, mlb_test, ncomp = 3)
mean((pred_fs_obp_plsr - mlb_test$fs_obp)^2)
```

```
## [1] 0.001043898
```

From our partial least squares model, we get a test mean squared error value of 0.001. This is the same mean squared error value from our previous model that we fit.

Regression Trees

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

```
fs_obp_tree = tree(fs_obp ~ ., mlb_train)
summary(fs_obp_tree)
```

```
##
## Regression tree:
## tree(formula = fs_obp ~ ., data = mlb_train)
## Variables actually used in tree construction:
##  [1] "obp"         "contact_pct" "h"           "bb_pct"      "ab"
##  [6] "ld_pct"      "slg"         "avg"         "rbi"         "gb_pct"
## [11] "r"
## Number of terminal nodes:  16
## Residual mean deviance:  0.0004346 = 0.09127 / 210
## Distribution of residuals:
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -0.0603400 -0.0123500 -0.0008542  0.0000000  0.0126500  0.0624200
```

There are 16 terminal nodes in this tree model and there are 11 variables being used as well.

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

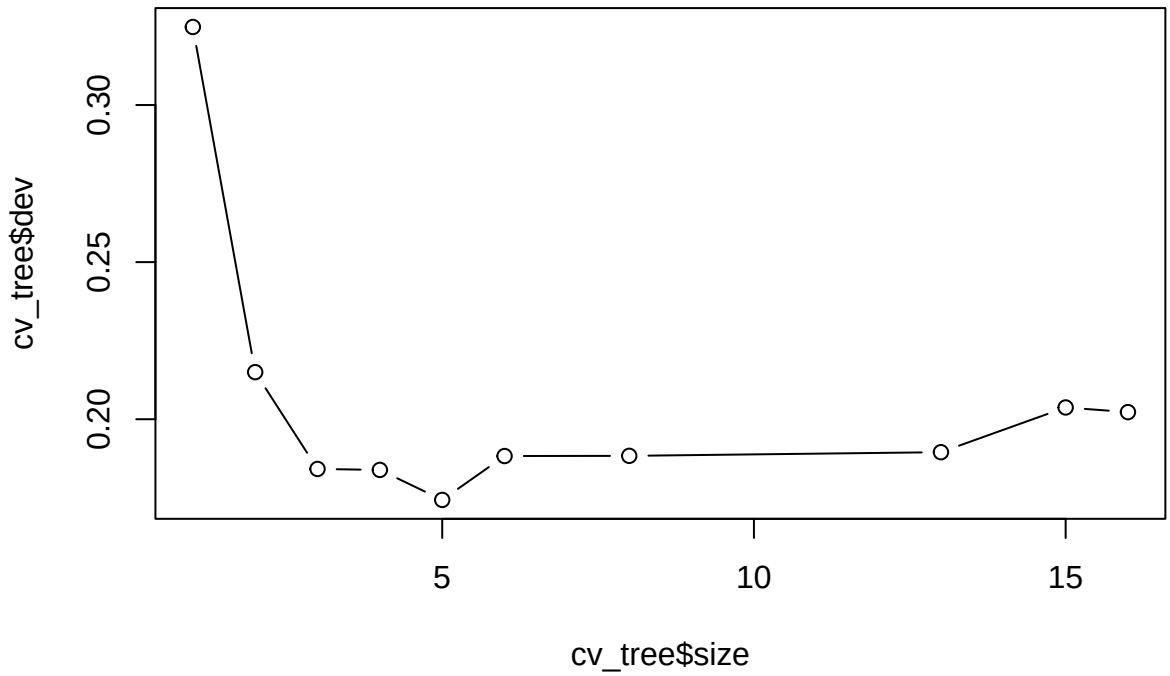
Not the prettiest tree to visually look at. Let's use the trained tree model to predict our test data.

```
yhat = predict(fs_obp_tree, newdata = mlb_test)
mean((yhat - mlb_test$fs_obp)^2)
```

```
## [1] 0.001247102
```

Once again, our model produced a test mean squared error value of 0.001. So far, we have not produced a different mean squared error rate.
Since this tree is quite confusing to understand, we want to try and prune the tree using cross-validation, to make it more simple.

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



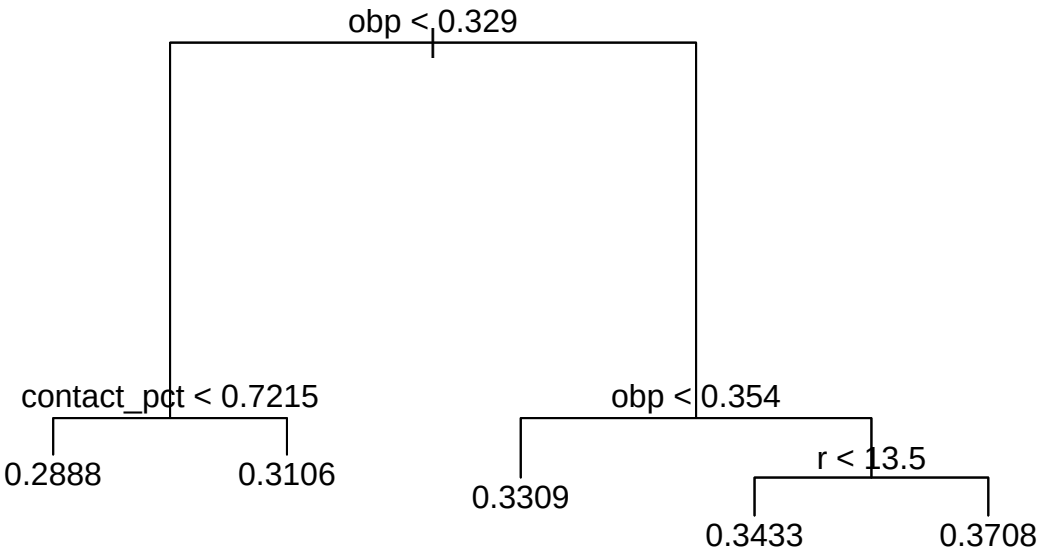
We see that tree size 5 minimizes cross-validation error. Let us prune our tree to make it more interpretable and see if it impacts our test mean squared error rate.

```
fs_obp_prune = prune.tree(fs_obp_tree, best = 5)
summary(fs_obp_prune)
```

```
##
## Regression tree:
## snip.tree(tree = fs_obp_tree, nodes = c(5L, 15L, 6L, 4L))
## Variables actually used in tree construction:
## [1] "obp"         "contact_pct" "r"
## Number of terminal nodes:  5
## Residual mean deviance:  0.0006473 = 0.143 / 221
## Distribution of residuals:
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -0.0675500 -0.0175500 -0.0002477  0.0000000  0.0151800  0.0770600
```

3 variables were used in this pruned tree that now has 5 terminal nodes.
Let us plot this pruned tree.

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



This pruned tree is much more easy to interpret compared to the previous tree that had 16 terminal nodes.
Let us predict the test fs_obp using this pruned tree model.

```
yhat = predict(fs_obp_prune, newdata = mlb_test)
mean((yhat - mlb_test$fs_obp^2))
```

```
## [1] 0.2185648
```

Although this pruned tree was much more interpretable, it did produce a test mean squared error rate of 0.219 compared to the 0.001 of our un-pruned tree.

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.

```
fs_obp_bag = bagging(fs_obp ~., data = mlb_train)
fs_obp_bag
```

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

Now let us predict our test data.

```
yhat_bag = predict(fs_obp_bag, newdata = mlb_test)
mean((yhat_bag - mlb_test$fs_obp)^2)
```

```
## [1] 0.00115233
```

The bagging model fitted by the training data produced a test mean squared error rate of 0.001. This mean squared error rate seems to be consistent with every model that we fit to the trained data.

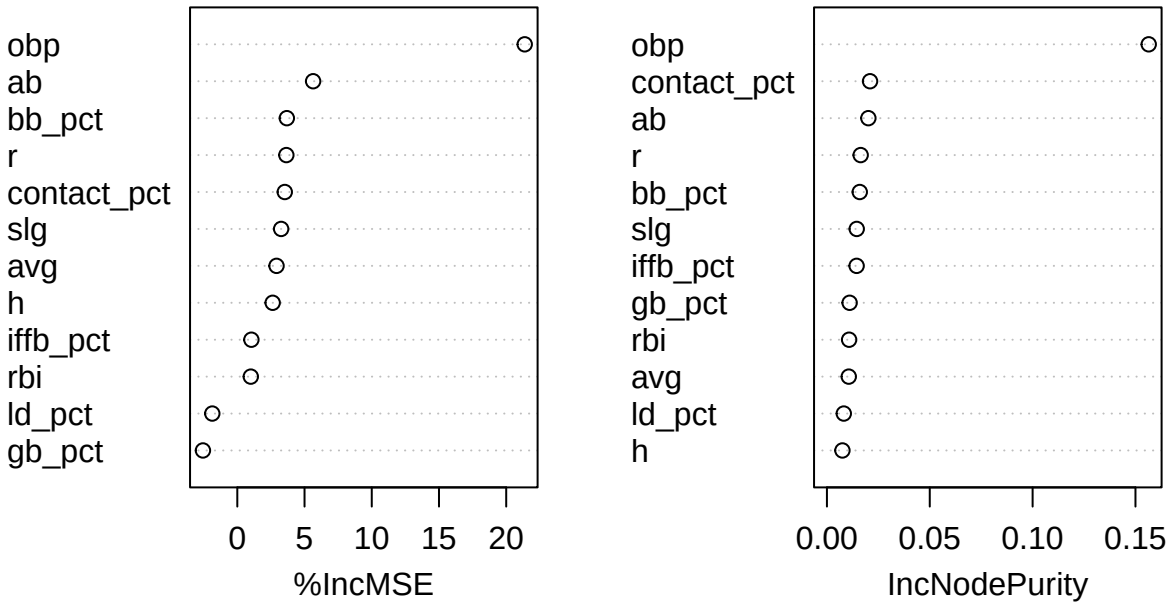
Random Forest

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

```
set.seed(1)
fs_obp_rf = randomForest(fs_obp ~., ntree = 100,
                         mtry = 12, importance = TRUE, data = mlb_train)
fs_obp_rf
```

```
##
## Call:
##  randomForest(formula = fs_obp ~ ., data = mlb_train, ntree = 100,      mtry = 12, importance = TRUE
##                Type of random forest: regression
##                      Number of trees: 100
## No. of variables tried at each split: 12
##
##           Mean of squared residuals: 0.0007837985
##                     % Var explained: 44.55
```

```
varImpPlot(fs_obp_rf)
```



The random forest used all 12 predictors at each split and explains approximately 45% of the variance in the training data. We see that the most important predictors in this model are obp, contact_pct and ab. Now let us now predict our test data.

```
yhat_rf = predict(fs_obp_rf, newdata = mlb_test)
mean((yhat_rf - mlb_test$fs_obp)^2)
```

```
## [1] 0.001110244
```

The random forest model also produced a test mean squared error rate of 0.001.

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:

```
fs_obp_boost = gbm(fs_obp ~., data = mlb_train, distribution = "gaussian",
                   n.trees = 1000)
yhat_boost = predict(fs_obp_boost, newdata = mlb_test,
                     n.trees = 1000)
mean((yhat_boost - mlb_test$fs_obp)^2)
```

```
## [1] 0.001260719
```

This boosted model produced a test mean squared error ate of 0.001.

Lambda = 0.01:

```
fs_obp_boost = gbm(fs_obp ~., data = mlb_train, distribution = "gaussian",
                   n.trees = 1000, shrinkage = 0.01)
yhat_boost = predict(fs_obp_boost, newdata = mlb_test,
                     n.trees = 1000)
mean((yhat_boost - mlb_test$fs_obp)^2)
```

```
## [1] 0.001176503
```

This boosted model also produced a test mean squared error rate of 0.001.

Lambda = 0.1:

```
fs_obp_boost = gbm(fs_obp ~., data = mlb_train, distribution = "gaussian",
                   n.trees = 1000, shrinkage = 0.1)
yhat_boost = predict(fs_obp_boost, newdata = mlb_test,
                     n.trees = 1000)
mean((yhat_boost - mlb_test$fs_obp)^2)
```

```
## [1] 0.001248302
```

This boosted model also produced a test mean squared error rate of 0.001.

Lambda = 0.2

```
fs_obp_boost = gbm(fs_obp ~., data = mlb_train, distribution = "gaussian",
                   n.trees = 1000, shrinkage = 0.2)
yhat_boost = predict(fs_obp_boost, newdata = mlb_test,
                     n.trees = 1000)
mean((yhat_boost - mlb_test$fs_obp)^2)
```

```
## [1] 0.001375415
```

This boosted model also produced a test mean squared error rate of 0.001.

Conclusion

During our quantitative regression analysis, we fit 9 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 MLB training data to train these models and predict the full season OBP for each player, we were only able to produce a test mean squared error of 0.001 for all of the models. Since all of our models produced the same test mean squared error, we want to choose the most simple and interpretable model.
Of all the models, the most interpretable is our simple linear model R code below:

```
fs_obp_lm = lm(fs_obp ~ h + avg + obp + contact_pct, mlb_train)
```

This linear model takes hits, average, obp and contact percentage to predict the full season obp of a player.

Here is the model below with each predicgtor's respective coefficient:
Full Season OBP estimate = 0.1598 + 0.0011(*Hits*) - 0.2537(*Average*) + 0.5084(*On-Base Percentage*) + 0.0523(*Contact Percentage*)

Based on our analysis, given a set of MLB data, we can use the linear model formula above to predict a player's full season OBP.

Sources

https://docs.google.com/spreadsheets/d/1U_QsSv6-68VLI0-5YrSPp2Bo-QhzZftiMU10b1Ajk0k/edit#gid=1381900348.

https://www.mlb.com/glossary/standard-stats/on-base-percentage#