STAT 697DS Final Report

Connor Kennedy

5/2/2021

```
library(tidyverse)
library(Lahman)
library(GGally)
library(leaps)
library(glmnet)
library(pls)
library(boot)
library(gam)
library(MASS)
library(randomForest)
library(gbm)
library(caret)
```

```
#Load allstar table for a variable later
AS <- AllstarFull %>%
  dplyr::select(-gameID, -teamID, -lgID, -gameNum, -GP, -startingPos) %>%
  mutate(AS = 1)

#Load batting table
#Many years since we will be removing rows later
Bat <- Batting %>%
  filter(yearID > 2013)

#Merge dataframes and binary allstar variable
Combo <- merge(Bat, AS, by = c("playerID", "yearID"), all.x = TRUE) %>%
  mutate(AS = as.factor(ifelse(is.na(AS), 0, AS)),
         AL = ifelse(lgID == 'AL', 1, 0))
```

Part 1

Make sure it should be written in an academic article format, i.e. write the full description and explanation rather than using bullet points and summary. (Try to avoid starting sentences starting with "I . . . ", i. g. "I want to do this and that.", "I think. . . ") The report should include:

i. Title.

A Data Driven Approach to Fantasy Roster Building and MLB All-star Selections

ii. Motivation and interest.

The increase in data availability for sports and legalization of sports betting has prompted many sports fans and statisticians to model many player outcomes. MLB players can receive larger contracts or bonuses if they are selected to the annual all-star team. Players can work on a particular part of their game not only to help their team, but secure more guaranteed money each season that they play. Fantasy sports also have exploded with the availability of these statistics online at a moments notice. There is quite a bit of money to be made if someone is able to predict a player's future outcomes. Teams have incorporated statistics into their own player scouting to find a market inefficiency on undervalued players. Most famously the early 2000s Oakland A's were able to remain competitive while having one of the lowest payrolls in the league.

iii. A sample piece of your data set to show what are included.

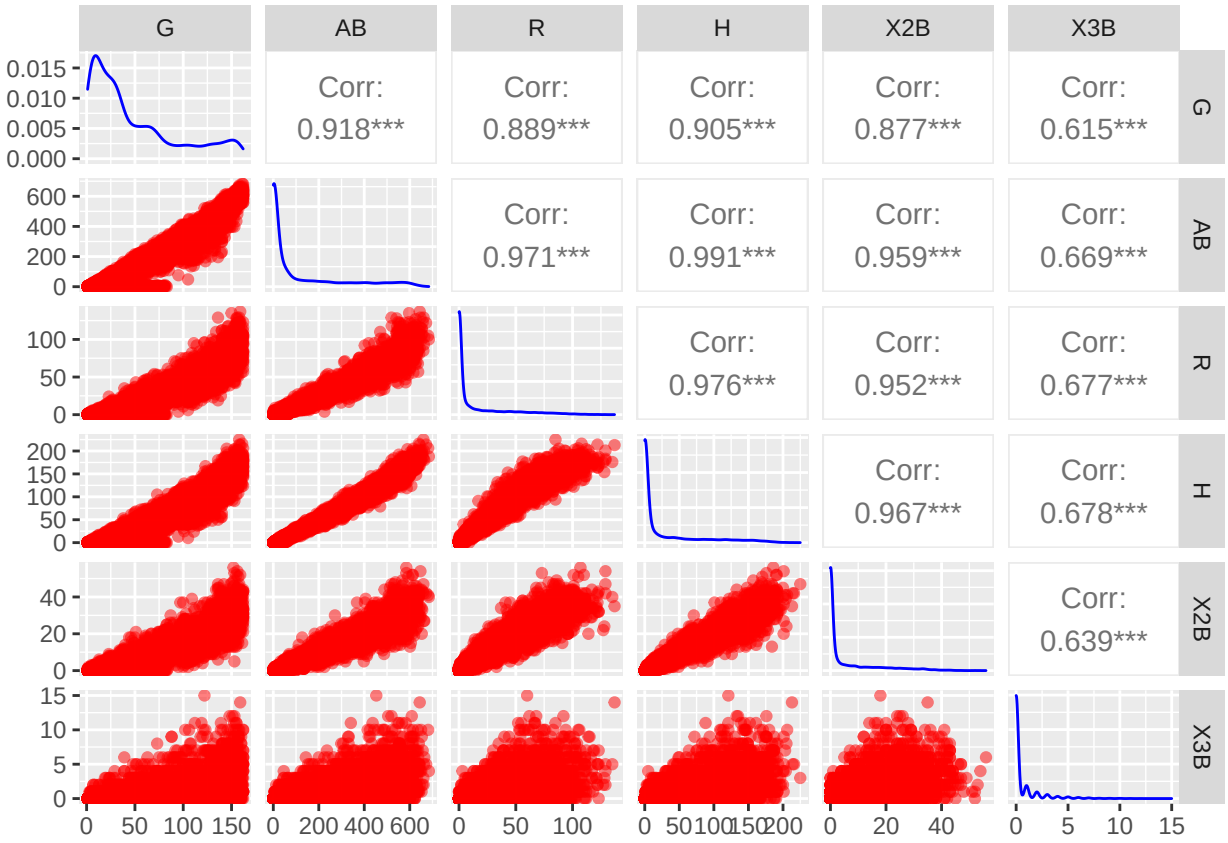
```
head(Combo)
```

```
##    playerID yearID stint teamID lgID  G AB R H X2B X3B HR RBI SB CS BB SO IBB
## 1 aardsda01   2015     1    ATL   NL 33  1 0 0   0   0  0   0  0  0  0  1   0
## 2  abadfe01   2014     1    OAK   AL 69  0 0 0   0   0  0   0  0  0  0  0   0
## 3  abadfe01   2015     1    OAK   AL 62  0 0 0   0   0  0   0  0  0  0  0   0
## 4  abadfe01   2016     1    MIN   AL 39  1 0 0   0   0  0   0  0  0  0  1   0
## 5  abadfe01   2016     2    BOS   AL 18  0 0 0   0   0  0   0  0  0  0  0   0
## 6  abadfe01   2017     1    BOS   AL 48  0 0 0   0   0  0   0  0  0  0  0   0
##   HBP SH SF GIDP AS AL
## 1   0  0  0    0  0  0
## 2   0  0  0    0  0  1
## 3   0  0  0    0  0  1
## 4   0  0  0    0  0  1
## 5   0  0  0    0  0  1
## 6   0  0  0    0  0  1
```

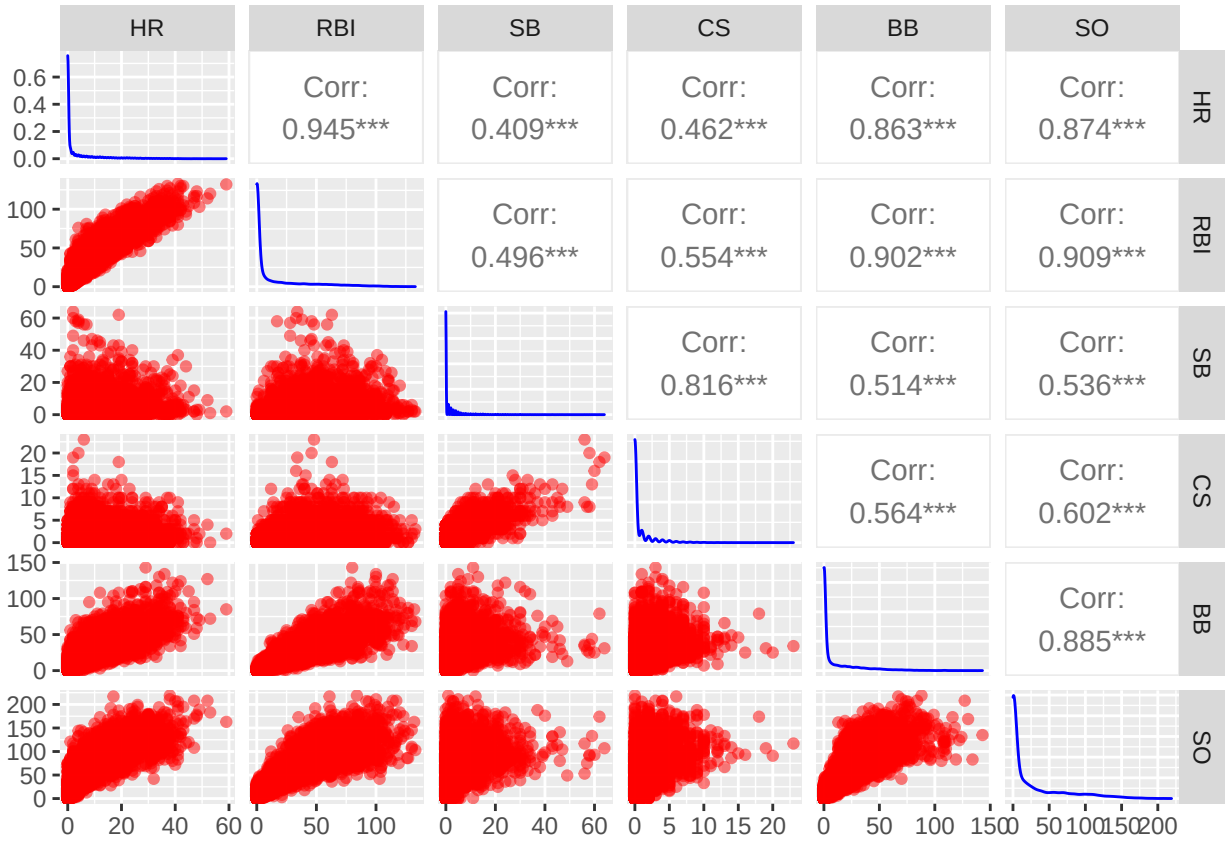
iv. Data summary (variable names and types)

Include the multi-panel scatter plots and correlation table. All graphs should be legible. Describe your findings directly from the data set.

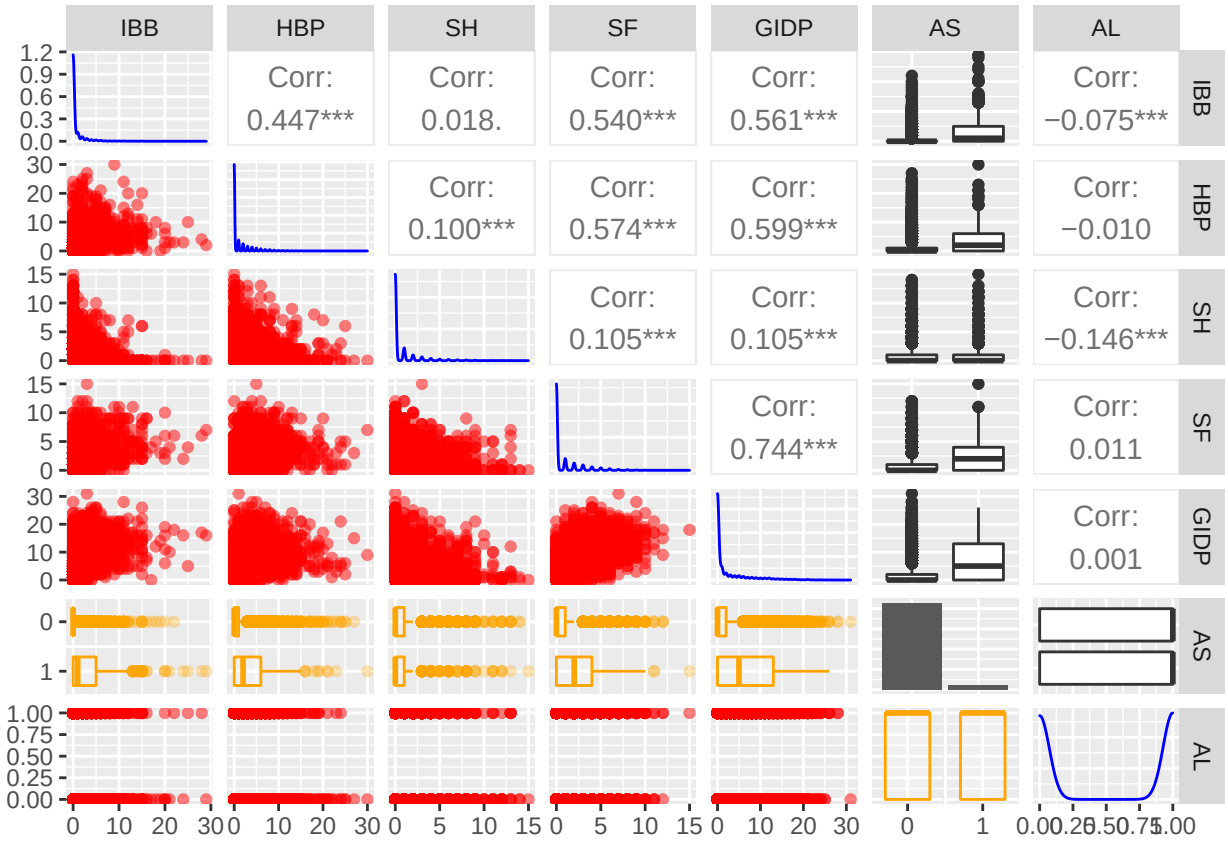
```
ggpairs(Combo, columns = 6:11,
        lower = list(continuous = wrap("points", color="red", alpha=0.5),
                     combo = wrap("box", color="orange", alpha=0.3),
                     discrete = wrap("facetbar", color="yellow", alpha=0.3)),
        diag = list(continuous = wrap("densityDiag", color="blue", alpha=0.5)))
```



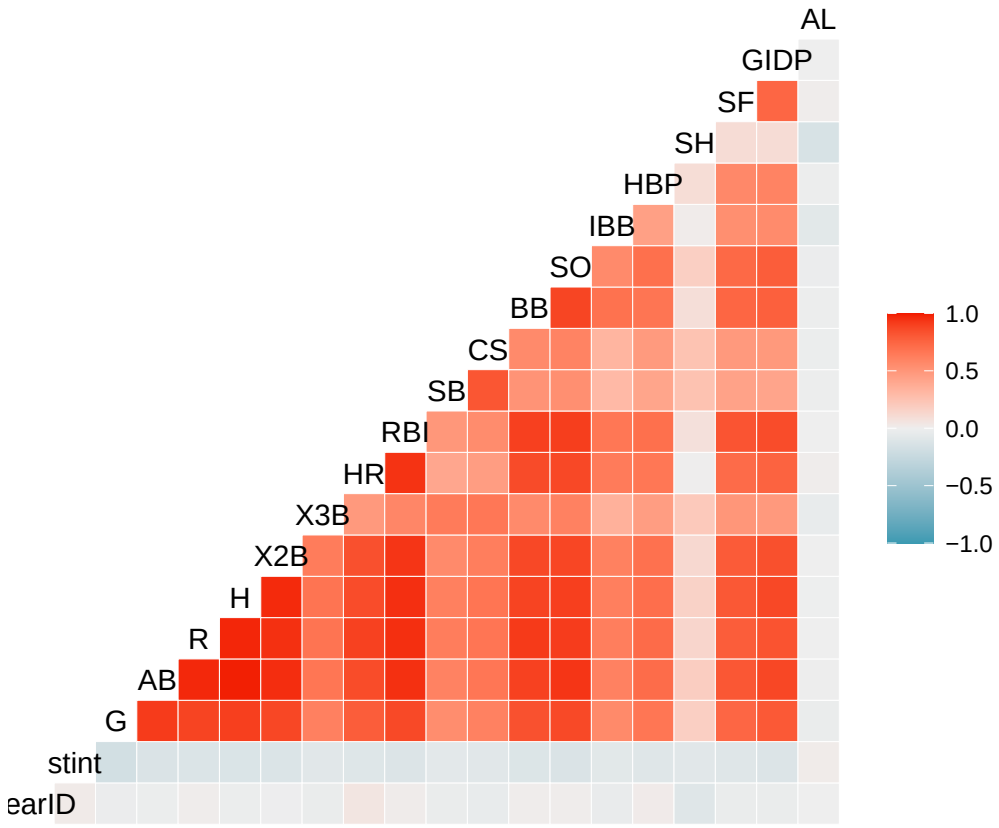

```
ggpairs(Combo, columns = 12:17,
        lower = list(continuous = wrap("points", color="red", alpha=0.5),
                     combo = wrap("box", color="orange", alpha=0.3),
                     discrete = wrap("facetbar", color="yellow", alpha=0.3)),
        diag = list(continuous = wrap("densityDiag", color="blue", alpha=0.5)))
```




```
ggpairs(Combo, columns = 18:24,
        lower = list(continuous = wrap("points", color="red", alpha=0.5),
                     combo = wrap("box", color="orange", alpha=0.3),
                     discrete = wrap("facetbar", color="yellow", alpha=0.3)),
        diag = list(continuous = wrap("densityDiag", color="blue", alpha=0.5)))
```




```
ggcorr(data = Combo, label = F)
```



Part 2

Specify what your goal is. In other words, what and why do you want to predict based on the statistical learning techniques. Choose one qualitative response and one quantitative response for regression and classification analysis in Problems 6 and 7.

```
#Use Lahman function to grab 'total bases' for fantasy scoring
ESPN <- battingStats(data = Combo, cbind = TRUE,
                     idvars = c("playerID", "yearID", "stint", "teamID",
                                "lgID")) %>%
  dplyr::select(-c(BA, PA, SlugPct, OBP, OPS, BABIP, stint, teamID, lgID)) %>%
  #Combine stints since players changing teams does not impact fantasy scoring
  group_by(playerID, yearID) %>%
  summarise(G = sum(G), AB = sum(AB), R = sum(R), H = sum(H), X2B = sum(X2B),
            X3B = sum(X3B), HR = sum(HR), RBI = sum(RBI), SB = sum(SB),
            CS = sum(CS), BB = sum(BB), SO = sum(SO), IBB = sum(IBB),
            HBP = sum(HBP), SH = sum(SH), SF = sum(SF), GIDP = sum(GIDP),
            AS = AS, TB = sum(TB)) %>%
  #Calculate fantasy scoring
  mutate(FB = R + TB + RBI + BB - SO + SB)

#Remove duplicate rows
ESPN <- distinct(ESPN)

#Use lead() to grab fantasy scoring from their next season
```

```
ESPN_lag <- ESPN %>%
  group_by(playerID) %>%
  mutate(FB_lag = ifelse(yearID < 2019, lead(FB), NA))

#Remove unnecessary rows
ESPN_lag <- na.omit(ESPN_lag)
```

My quantitative variable goal is to predict fantasy baseball scoring using data from the player's previous season to help people draft their teams.

Statistic	Variable	ESPN Scoring
Runs Scored	R	1
Total Bases	TB	1
Runs Batted In	RBI	1
Walks	BB	1
Strikeouts	SO	-1
Stolen Base	SB	1

My qualitative variable goal is to help predict baseball all-star selections using only their batting statistics.

Part 3

Identify the high leverage points? (Remove them unless those are meaningful points.)

There are a few outliers but none are particularly high leverage. Plots seen in linear regression under part 6.

Part 4

Do you need to use log scale or log-log scale? Specify the features if any. (When you see your data are highly clustered in one (smaller) side of your axis and widely scattered in the other side, consider using a log scale for the variable.

```
ESPN2_lag <- ESPN_lag %>%
  mutate(G = log(G + 1), AB = log(AB + 1), R = log(R + 1), H = log(H + 1),
         RBI = log(RBI + 1), SO = log(SO + 1), TB = log(TB + 1),
         FB = ifelse(FB > 0, log(FB),0),
         FB_lag = ifelse(FB_lag > 0, log(FB_lag),0))
```

Many "counting" statistics needed a log transformation to help reduce skew.

Part 5

Missing data points? For missing data points, you can do (you should state what you used):

```
#Original data
anyNA(ESPN)
```

```
## [1] FALSE
```

```
#Lagged data
anyNA(ESPN2_lag)
```

```
## [1] FALSE
```

There are not any missing data points in the selected data set. I introduced NA values into the lagged data set to ensure that players without data for the following season are not given fantasy points from another player. Those rows with NA values are then removed since they are not necessary for regression. None of the following missing data methods are applicable or necessary.

i. If there are only very few out of many data points, you may just remove them.

ii. You can use unsupervised learning to fix them. See Reference #1 at the end.

iii. Using random forest to estimate the missing data points. See Reference #2.

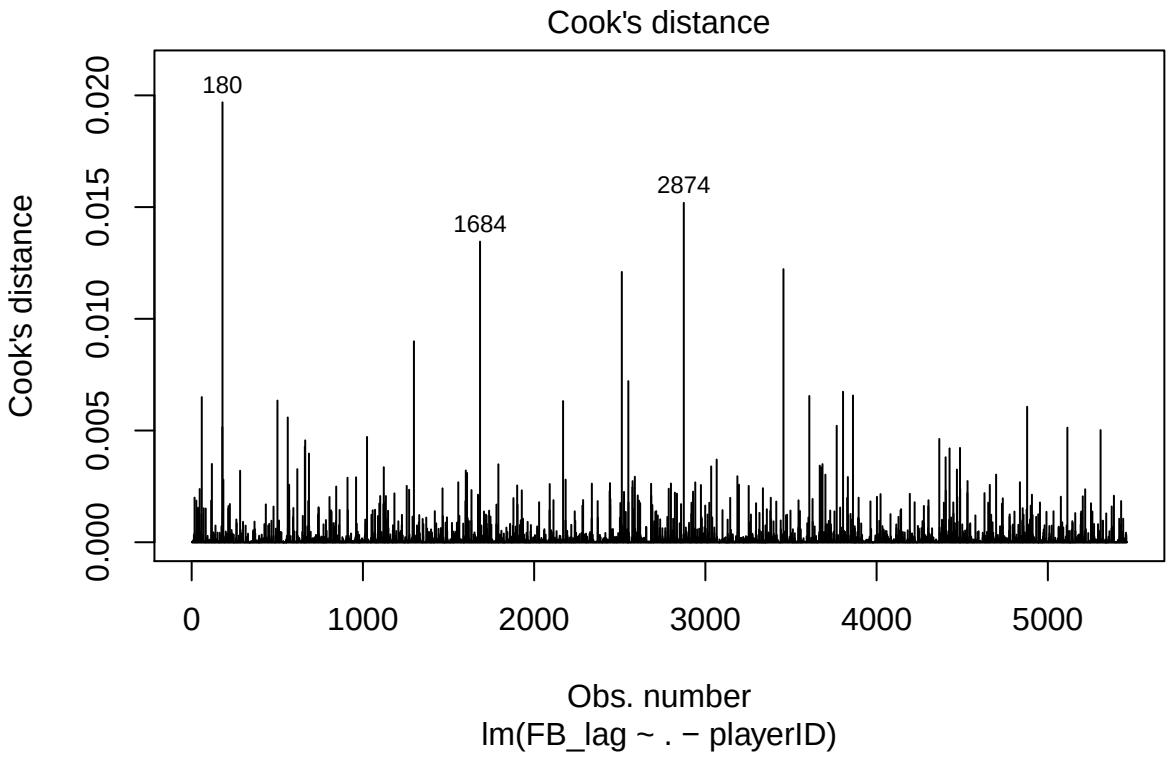
Part 6

For a quantitative response variable, find the following regression models (coefficients) and compare them:

i. Standard linear model. (Chapter 3)

```
set.seed(1)

#Fit a full model (except playerID)
SLM <- lm(FB_lag ~ . - playerID, data = ESPN2_lag)
plot(SLM, which = 4)
```




```
train.control <- trainControl(method = "cv", number = 10)

SLM.CV <- train(FB_lag ~ . - playerID, data = ESPN2_lag, method = "lm",
                trControl = train.control)

print(SLM.CV)
```

```
## Linear Regression
##
## 5462 samples
##   22 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4916, 4916, 4916, 4916, 4915, 4917, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   1.035814  0.8179021  0.6596936
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
summary(SLM.CV)
```

```
##
```

```
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.8664 -0.3764  0.0056  0.3211  4.9028
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 24.5656296 20.2349445   1.214  0.22479
## yearID      -0.0118317  0.0100376  -1.179  0.23855
## G           -0.2219273  0.0201100 -11.036  < 2e-16 ***
## AB           0.3759048  0.0469691   8.003 1.47e-15 ***
## R            0.1644014  0.0616923   2.665  0.00772 **
## H            0.0077069  0.1404785   0.055  0.95625
## X2B          0.0098512  0.0039606   2.487  0.01290 *
## X3B          0.0122611  0.0127128   0.964  0.33485
## HR           0.0026852  0.0042497   0.632  0.52751
## RBI         -0.0304013  0.0564464  -0.539  0.59019
## SB           0.0024395  0.0044751   0.545  0.58568
## CS           0.0283492  0.0141666   2.001  0.04543 *
## BB          -0.0003856  0.0017703  -0.218  0.82756
## SO          -0.3026806  0.0548188  -5.521 3.52e-08 ***
## IBB          0.0280420  0.0091585   3.062  0.00221 **
## HBP          0.0048667  0.0068459   0.711  0.47718
## SH          -0.0868643  0.0098169  -8.848  < 2e-16 ***
## SF           0.0068088  0.0130507   0.522  0.60188
## GIDP        -0.0054366  0.0057355  -0.948  0.34323
## AS1         -0.0589594  0.0610318  -0.966  0.33407
## TB           0.1344693  0.1320540   1.018  0.30859
## FB           0.5413432  0.0443277  12.212  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.034 on 5440 degrees of freedom
## Multiple R-squared:  0.8191, Adjusted R-squared:  0.8184
## F-statistic:  1173 on 21 and 5440 DF,  p-value: < 2.2e-16
```

ii. Best subset selection. (Chapter 6)

```
set.seed(1)

BSS.fits <- regsubsets(FB_lag ~ . - playerID, data = ESPN2_lag, nvmax = 18,
                       really.big = T)

summary.table <- data.frame(p=apply(summary(BSS.fits)$which,1,sum),
                            rsq=summary(BSS.fits)$rsq,
                            adjr2=summary(BSS.fits)$adjr2,
                            cp=summary(BSS.fits)$cp,
                            bic=summary(BSS.fits)$bic)
#Check best fitted models & associated criteria
cbind(summary(BSS.fits)$which, summary.table)
```

```
##    (Intercept) yearID     G    AB     R     H   X2B   X3B    HR   RBI    SB
## 1         TRUE  FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 2         TRUE  FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 3         TRUE  FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 4         TRUE  FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 5         TRUE  FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## 6         TRUE  FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## 7         TRUE  FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE
## 8         TRUE  FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE
## 9         TRUE  FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE
## 10        TRUE  FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE
## 11        TRUE  FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE
## 12        TRUE   TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE
## 13        TRUE   TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE
## 14        TRUE   TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE
## 15        TRUE   TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE
## 16        TRUE   TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE
## 17        TRUE   TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE
## 18        TRUE   TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##       CS    BB    SO   IBB   HBP    SH    SF  GIDP   AS1    TB   FB  p
## 1  FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE  2
## 2  FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE  3
## 3  FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE  4
## 4  FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE TRUE  5
## 5  FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE TRUE  6
## 6  FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE TRUE  7
## 7  FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE TRUE  8
## 8   TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE TRUE  9
## 9   TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE TRUE 10
## 10  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE TRUE 11
## 11  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE TRUE 12
## 12  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE TRUE 13
## 13  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE TRUE 14
## 14  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE TRUE 15
## 15  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE TRUE 16
## 16  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE TRUE 17
## 17  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE TRUE 18
## 18  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE TRUE 19
##          rsq     adjr2         cp       bic
## 1  0.8049028 0.8048671 407.918631 -8909.103
## 2  0.8098299 0.8097602 261.778109 -9040.209
## 3  0.8129333 0.8128305 170.469057 -9121.474
## 4  0.8151415 0.8150060 106.077304 -9177.726
## 5  0.8166014 0.8164333  64.182165 -9212.428
## 6  0.8173771 0.8171763  42.858333 -9226.974
## 7  0.8179261 0.8176924  28.351743 -9234.813
## 8  0.8183280 0.8180615  18.268927 -9238.277
## 9  0.8186973 0.8183981   9.163457 -9240.787
## 10 0.8188714 0.8185391   5.931508 -9237.427
## 11 0.8189169 0.8185514   6.563784 -9230.193
## 12 0.8189578 0.8185591   7.331741 -9222.824
## 13 0.8189886 0.8185567   8.405026 -9215.148
## 14 0.8190109 0.8185457   9.736759 -9207.213
## 15 0.8190301 0.8185317  11.158341 -9199.188
```

```
## 16 0.8190406 0.8185089  12.841896 -9190.900
## 17 0.8190515 0.8184865  14.513995 -9182.624
## 18 0.8190583 0.8184600  16.309597 -9174.224
```

```
BSS <- train(FB_lag ~ G + AB + R + X2B + CS + SO + IBB + SH + FB,
             data=ESPN2_lag, method = "lm", trControl = train.control)
print(BSS)
```

```
## Linear Regression
##
## 5462 samples
##    9 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4916, 4916, 4916, 4916, 4915, 4917, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   1.034625  0.8183227  0.6597853
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

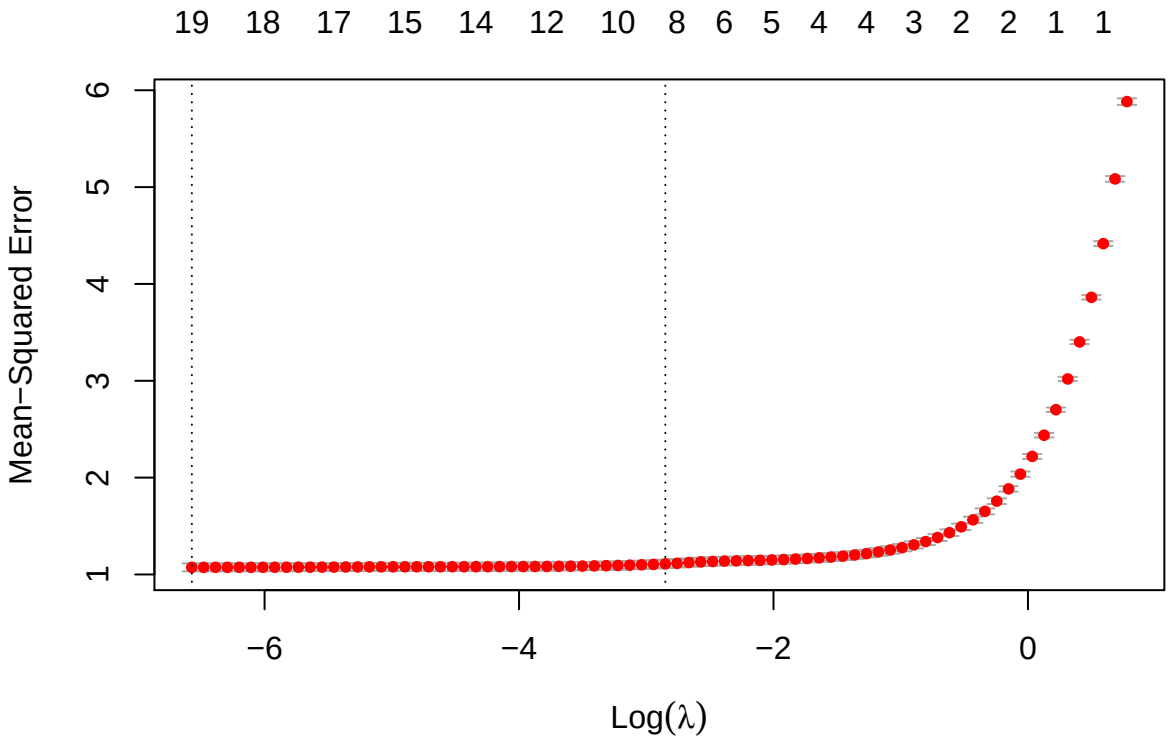
iii. The Lasso. (Chapter 6)

```
set.seed(1)

lasso.x <- model.matrix(FB_lag ~ . - playerID, ESPN2_lag)[, -1]
lasso.y <- ESPN2_lag$FB_lag

lasso.cv.out <- cv.glmnet(lasso.x, lasso.y, alpha = 1, type.measure = "mse")

plot(lasso.cv.out)
```




```
lasso.cv.out
```

```
##
## Call:  cv.glmnet(x = lasso.x, y = lasso.y, type.measure = "mse", alpha = 1)
##
## Measure: Mean-Squared Error
##
##      Lambda Index Measure      SE Nonzero
## min 0.00140    80   1.074 0.04046      19
## 1se 0.05783    40   1.110 0.04331       9
```

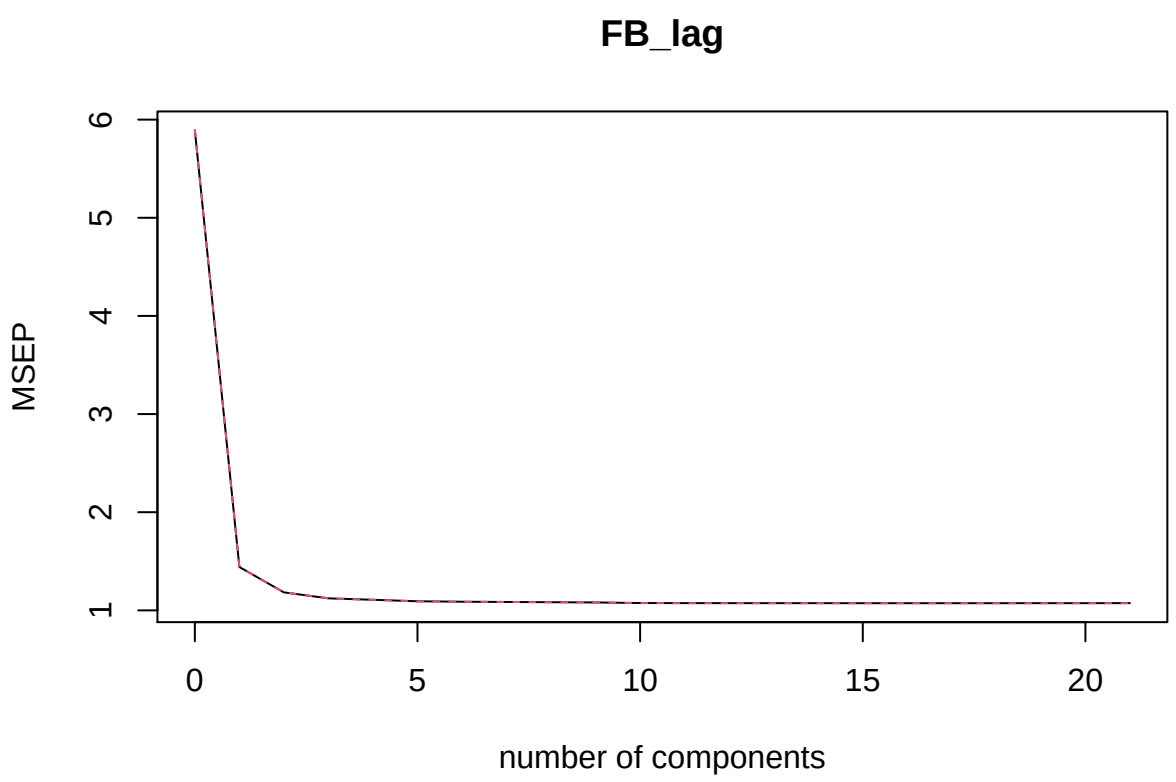
Lasso with 10-fold CV selects $\lambda = 0.0014$ with an estimated test MSE of 1.074.

iv. Partial Least Squares. (Chapter 6)

```
set.seed(1)

PLS <- plsr(FB_lag ~ . - playerID, data = ESPN2_lag, scale = T,
            validation = "CV")

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




```
MSEP(PLS)$val[1,1,]
```

```
## (Intercept)     1 comps     2 comps     3 comps     4 comps     5 comps
##    5.890718    1.441704    1.183318    1.122638    1.107812    1.092583
##     6 comps     7 comps     8 comps     9 comps    10 comps    11 comps
##    1.088005    1.085688    1.083167    1.080697    1.074587    1.073914
##    12 comps    13 comps    14 comps    15 comps    16 comps    17 comps
##    1.073645    1.073479    1.073341    1.072993    1.073143    1.073133
##    18 comps    19 comps    20 comps    21 comps
##    1.073174    1.073324    1.073387    1.073794
```

```
which.min(MSEP(PLS)$val[1,1,])
```

```
## 15 comps
##       16
```

```
min(MSEP(PLS)$val[1,1,])
```

```
## [1] 1.072993
```

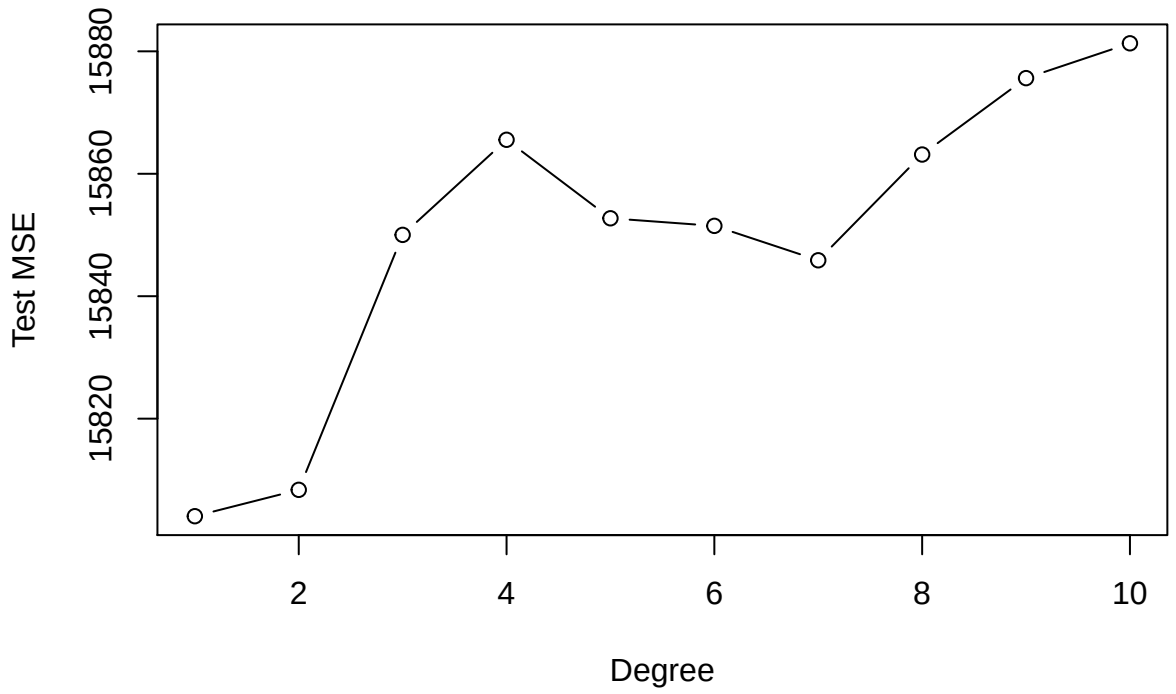
Partial least squares regression with 10-fold CV selects 15 components with an estimated test MSE of 1.072993. There is dimension reduction since there are 21 possible predictors.

v. Polynomial (find the best polynomial degree.) (Chapter 7)

```
set.seed(1)

Batting.deltas <- rep(NA, 10)
for (i in 1:10) {
Batting.fit <- glm(FB_lag ~ poly(AB, i), data = ESPN2_lag)
Batting.deltas[i] <- cv.glm(ESPN_lag, Batting.fit, K = 10)$delta[1]
}

plot(1:10, Batting.deltas, xlab = "Degree", ylab = "Test MSE", type = "b")
```




```
min(Batting.deltas)
```

```
## [1] 15804.07
```

```
which.min(Batting.deltas)
```

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

```
Batting.fit1 <- lm(FB_lag ~ AB, data = ESPN2_lag)
Batting.fit2 <- lm(FB_lag ~ poly(AB, 2), data = ESPN2_lag)
Batting.fit3 <- lm(FB_lag ~ poly(AB, 3), data = ESPN2_lag)
```

```
Batting.fit4 <- lm(FB_lag ~ poly(AB, 4), data = ESPN2_lag)
Batting.fit5 <- lm(FB_lag ~ poly(AB, 5), data = ESPN2_lag)
anova(Batting.fit1, Batting.fit2, Batting.fit3, Batting.fit4, Batting.fit5)
```

```
## Analysis of Variance Table
##
## Model 1: FB_lag ~ AB
## Model 2: FB_lag ~ poly(AB, 2)
## Model 3: FB_lag ~ poly(AB, 3)
## Model 4: FB_lag ~ poly(AB, 4)
## Model 5: FB_lag ~ poly(AB, 5)
##   Res.Df    RSS Df Sum of Sq         F    Pr(>F)
## 1   5460 9239.5
## 2   5459 7606.2  1   1633.32 1177.3707 < 2.2e-16 ***
## 3   5458 7603.8  1      2.34    1.6869 0.1940693
## 4   5457 7586.4  1     17.39   12.5377 0.0004021 ***
## 5   5456 7568.9  1     17.53   12.6377 0.0003812 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Poly <- train(FB_lag ~ AB + AB^2, data=ESPN2_lag, method = "lm",
              trControl = train.control)
print(Poly)
```

```
## Linear Regression
##
## 5462 samples
##    1 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4916, 4916, 4917, 4916, 4915, 4917, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   1.300038  0.7130392  0.9807413
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

For 10-fold CV, d = 1 is the optimal degree for the polynomial since it has the smallest test MSE. This is not supported when using ANOVA as a quadratic polynomial seems most appropriate. These calculations were done when using at-bats (AB) as the sole predictor of lagged fantasy points (FB_lag).

vi. Natural cubic spline (either state the degree of freedom or knots.) (Chapter 7)

```
set.seed(1)

NCS <- lm(ESPN2_lag$FB_lag ~ ns(ESPN_lag$AB, df = 5))
summary(NCS)
```

```
##
## Call:
## lm(formula = ESPN2_lag$FB_lag ~ ns(ESPN_lag$AB, df = 5))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.7006 -0.3163 -0.0900  0.3691  5.5178
##
## Coefficients: (1 not defined because of singularities)
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)                23.1194     0.4138   55.88   <2e-16 ***
## ns(ESPN_lag$AB, df = 5)1 -20.9661     0.4156  -50.44   <2e-16 ***
## ns(ESPN_lag$AB, df = 5)2 -19.0649     0.3826  -49.83   <2e-16 ***
## ns(ESPN_lag$AB, df = 5)3  -8.8069     0.3311  -26.60   <2e-16 ***
## ns(ESPN_lag$AB, df = 5)4 -44.1030     0.8605  -51.26   <2e-16 ***
## ns(ESPN_lag$AB, df = 5)5       NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.18 on 5457 degrees of freedom
## Multiple R-squared:  0.7638, Adjusted R-squared:  0.7636
## F-statistic:  4412 on 4 and 5457 DF,  p-value: < 2.2e-16
```

```
NCS.CV <- train(FB_lag ~ ns(AB, df = 5), data = ESPN2_lag,
                method = "lm", trControl = train.control)
print(NCS.CV)
```

```
## Linear Regression
##
## 5462 samples
##    1 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4916, 4916, 4916, 4916, 4915, 4917, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   1.176992  0.7646826  0.7281157
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

The natural spline is fit using 5 degrees of freedom.

vii. Smoothing spline. (Chapter 7)

```
set.seed(1)

Smooth <- smooth.spline(x = ESPN2_lag$AB, y = ESPN2_lag$FB_lag)

mean(resid(Smooth)^2)
```

```
## [1] 1.348502
```

viii. Local Regression. (Chapter 7)

```
set.seed(1)

Local <- loess(ESPN2_lag$FB_lag ~ ESPN2_lag$AB)

mean(resid(Local)^2)
```

```
## [1] 1.390995
```

ix. Generalized Additive Model. (Chapter 7)

```
set.seed(1)

split = sample(seq_len(nrow(ESPN2_lag)), size = floor(0.8*nrow(ESPN2_lag)))

#80% train
quan_train = ESPN2_lag[split,]

#20% test
quan_test = ESPN2_lag[-split,]

GAM0 <- gam::gam(FB_lag ~ lo(G, span = 0.7) + s(FB, df = 5), data = ESPN2_lag)

GAM1 <- gam::gam(FB_lag ~ lo(G, span = 0.7) + s(FB, df = 5) + AS,
                 data = ESPN2_lag)

GAM2 <- gam::gam(FB_lag ~ lo(G, span = 0.7) + s(FB, df = 5) + factor(yearID),
                 data = ESPN2_lag)

GAM3 <- gam::gam(FB_lag ~ lo(G, span = 0.7) + s(FB, df = 5) + AS +
                   factor(yearID), data = ESPN2_lag)

anova(GAM0, GAM1, GAM2, GAM3)
```

```
## Analysis of Deviance Table
##
## Model 1: FB_lag ~ lo(G, span = 0.7) + s(FB, df = 5)
## Model 2: FB_lag ~ lo(G, span = 0.7) + s(FB, df = 5) + AS
## Model 3: FB_lag ~ lo(G, span = 0.7) + s(FB, df = 5) + factor(yearID)
## Model 4: FB_lag ~ lo(G, span = 0.7) + s(FB, df = 5) + AS + factor(yearID)
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1      5453     6035.6
## 2      5452     6035.5  1   0.0325  0.86398
## 3      5449     6025.7  3   9.8620  0.03042 *
## 4      5448     6025.6  1   0.0638  0.81026
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
GAM.train <- gam::gam(FB_lag ~ lo(G, span = 0.7) + s(FB, df = 5) +
                        factor(yearID), data = quan_train)

pred.GAM <- predict(GAM.train, newdata = quan_test)

mse <- mean((quan_test$FB_lag - pred.GAM)^2)
mse
```

```
## [1] 0.9918142
```

Use Cross Validation or Validation Set Approach to compare the estimated test errors for the above regression. (Chapter 5)

Approach	10-fold CV MSE
Standard Linear Model	1.07291
Best Subset Selection	1.07045
The Lasso	1.074
Partial Least Squares	1.072993
Polynomial	1.690098
Natural cubic spline	1.38531
Smoothing spline	1.348502
Local Regression	1.390995
Generalized Additive Model	0.9918142*

*GAM was estimated using validation set approach

Overall the simpler models appeared to outperform many more complicated models. I believe this is because they are more robust and less prone to overfitting. GAM had a promising outcome, but I was unable to use 10-fold CV so this is likely too optimistic. I would prefer to use Best Subset Selection to find a linear model overall.

Part 7

For a qualitative response variable, compare the following classification models:

i. Logistic Regression. (Chapter 4)

```
set.seed(1)

ESPN2 <- ESPN_lag %>% ungroup() %>% dplyr::select(-c(playerID, FB, FB_lag))

split = sample(seq_len(nrow(ESPN2)), size = floor(0.8*nrow(ESPN2)))

#80% train
qual_train = ESPN2[split,]

#20% test
qual_test = ESPN2[-split,]
AS_test = ESPN2$AS[-split]
```

```
LR <- glm(AS ~ ., data = ESPN2, family = binomial, subset = split)

prob.LR <- predict(LR, qual_test, type = "response")

pred.LR <- rep(0, length(prob.LR))
pred.LR[prob.LR > 0.5] <- 1

#Confusion table
table(pred.LR, AS_test)
```

```
##         AS_test
## pred.LR    0    1
##       0 1009   58
##       1   13   13
```

```
#Test error rate
mean(pred.LR != AS_test)
```

```
## [1] 0.06495883
```

ii. Linear Discriminant Analysis. (Chapter 4)

```
set.seed(1)

LDA <- lda(AS ~ ., data = ESPN2, subset = split)

pred.LDA <- predict(LDA, qual_test)

#Confusion table
table(pred.LDA$class, AS_test)
```

```
##    AS_test
##       0   1
##   0 993  52
##   1  29  19
```

```
#Test error rate
mean(pred.LDA$class != AS_test)
```

```
## [1] 0.07410796
```

iii. Quadratic Discriminant Analysis. (Chapter 4)

```
set.seed(1)

QDA <- qda(AS ~ G + AB + R + H + X2B + X3B + HR + RBI + SB + CS + BB + SO +
             IBB + HBP + SH + SF + GIDP + yearID, data = ESPN2, subset = split)
```

```
pred.QDA <- predict(QDA, qual_test)

#Confusion table
table(pred.QDA$class, AS_test)
```

```
##    AS_test
##       0   1
##   0 847  37
##   1 175  34
```

```
#Test error rate
mean(pred.QDA$class != AS_test)
```

```
## [1] 0.1939616
```

iv. Bagging. Visualize the best tree and the importance plot. (Chapter 8)

```
set.seed(1)

BAG.50 <- randomForest(AS ~ ., data = ESPN2, mtry = 19, ntree = 50)
print(BAG.50)
```

```
##
## Call:
##  randomForest(formula = AS ~ ., data = ESPN2, mtry = 19, ntree = 50)
##                Type of random forest: classification
##                      Number of trees: 50
## No. of variables tried at each split: 19
##
##         OOB estimate of  error rate: 6.7%
## Confusion matrix:
##      0   1 class.error
## 0 4993  95  0.01867138
## 1  271 103  0.72459893
```

```
BAG.100 <- randomForest(AS ~ ., data = ESPN2, mtry = 19, ntree = 100)
print(BAG.100)
```

```
##
## Call:
##  randomForest(formula = AS ~ ., data = ESPN2, mtry = 19, ntree = 100)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 19
##
##         OOB estimate of  error rate: 6.76%
## Confusion matrix:
##      0  1 class.error
## 0 4998 90  0.01768868
## 1  279 95  0.74598930
```

```r
BAG.500 <- randomForest(AS ~ ., data = ESPN2, mtry = 19, ntree = 500)
print(BAG.500)
```

```
##
## Call:
##  randomForest(formula = AS ~ ., data = ESPN2, mtry = 19, ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 19
##
##         OOB estimate of  error rate: 6.46%
## Confusion matrix:
##      0  1 class.error
## 0 5011 77  0.01513365
## 1  276 98  0.73796791
```

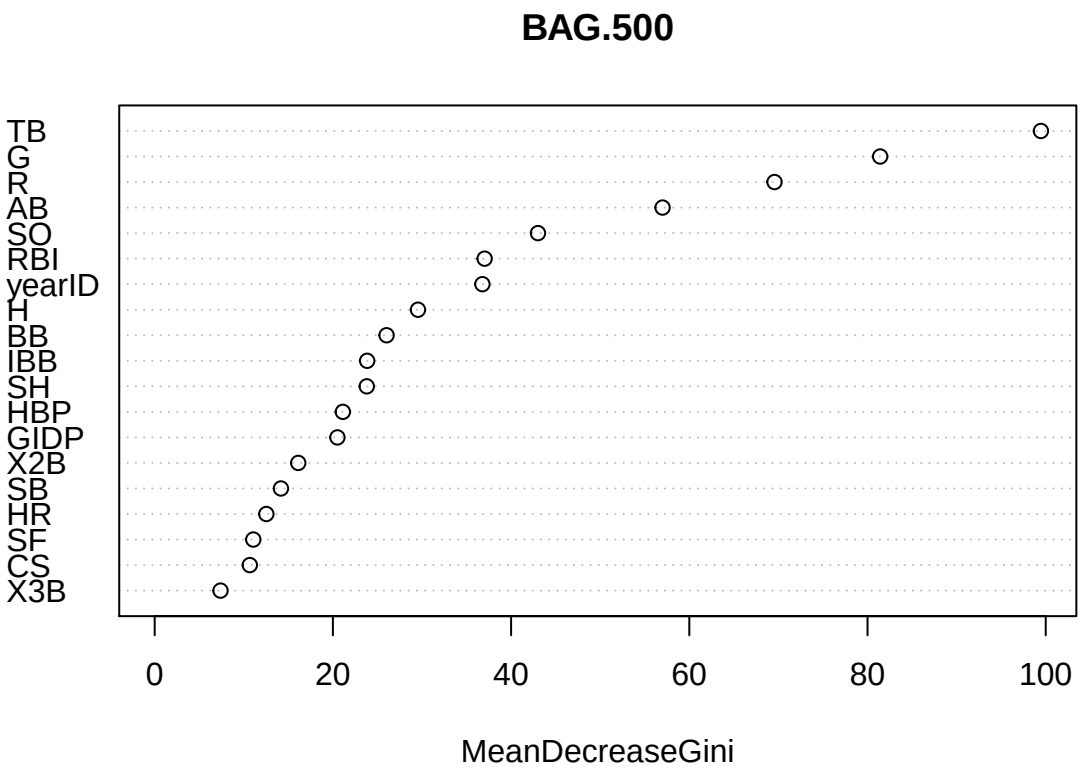
```r
BAG.1000 <- randomForest(AS ~ ., data = ESPN2, mtry = 19, ntree = 1000)
print(BAG.1000)
```

```
##
## Call:
##  randomForest(formula = AS ~ ., data = ESPN2, mtry = 19, ntree = 1000)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 19
##
##         OOB estimate of  error rate: 6.48%
## Confusion matrix:
##      0  1 class.error
## 0 5018 70  0.01375786
## 1  284 90  0.75935829
```

```r
#Best tree
print(BAG.500)
```

```
##
## Call:
##  randomForest(formula = AS ~ ., data = ESPN2, mtry = 19, ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 19
##
##         OOB estimate of  error rate: 6.46%
## Confusion matrix:
##      0  1 class.error
## 0 5011 77  0.01513365
## 1  276 98  0.73796791
```

```r
#Importance plot
varImpPlot(BAG.500)
```



v. Random Forest. Visualize the best tree and the importance plot. (Chapter 8)

```
RF.50 <- randomForest(AS ~ ., data = ESPN2, ntree = 50, importance = T)
print(RF.50)
```

```
##
## Call:
##  randomForest(formula = AS ~ ., data = ESPN2, ntree = 50, importance = T)
##                Type of random forest: classification
##                      Number of trees: 50
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 6.24%
## Confusion matrix:
##      0  1 class.error
## 0 5036 52  0.01022013
## 1  289 85  0.77272727
```

```
RF.100 <- randomForest(AS ~ ., data = ESPN2, ntree = 100, importance = T)
print(RF.100)
```

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

```
##  randomForest(formula = AS ~ ., data = ESPN2, ntree = 100, importance = T)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 6.19%
## Confusion matrix:
##      0  1 class.error
## 0 5044 44 0.008647799
## 1  294 80 0.786096257
```

```
RF.500 <- randomForest(AS ~ ., data = ESPN2, ntree = 500, importance = T)
print(RF.500)
```

```
##
## Call:
##  randomForest(formula = AS ~ ., data = ESPN2, ntree = 500, importance = T)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 6.15%
## Confusion matrix:
##      0  1 class.error
## 0 5045 43 0.008451258
## 1  293 81 0.783422460
```

```
RF.1000 <- randomForest(AS ~ ., data = ESPN2, ntree = 1000, importance = T)
print(RF.1000)
```

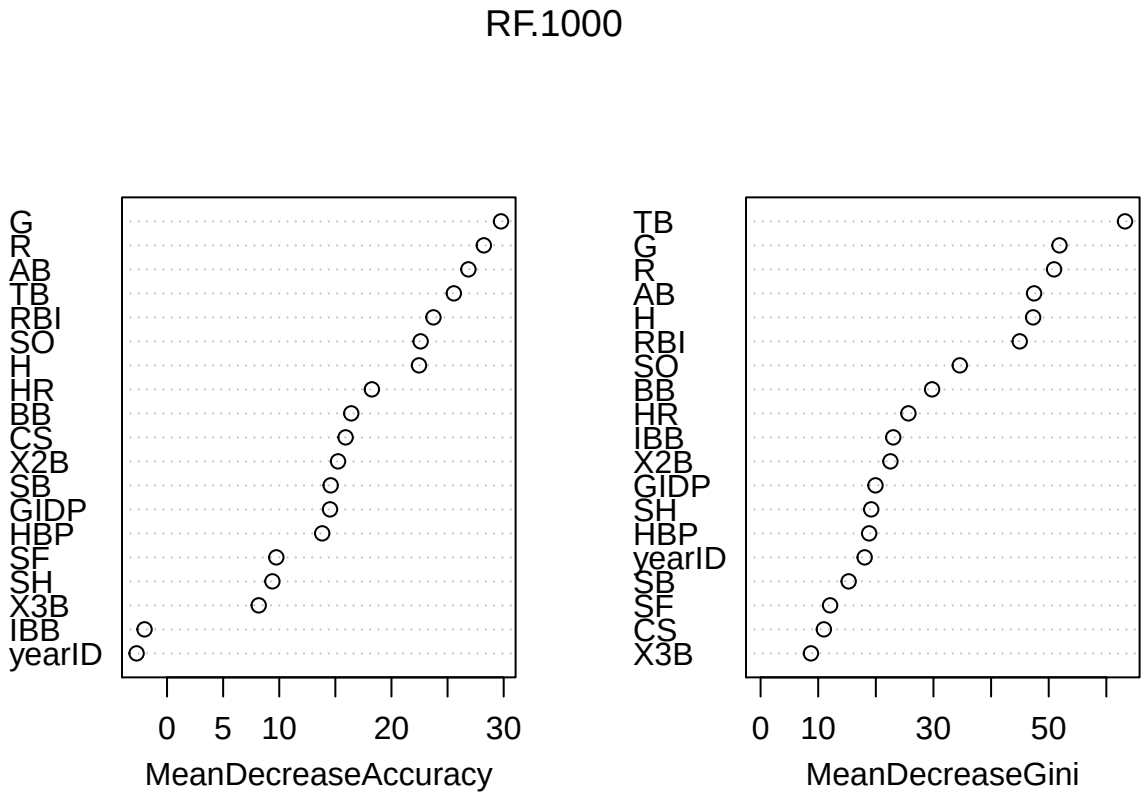
```
##
## Call:
##  randomForest(formula = AS ~ ., data = ESPN2, ntree = 1000, importance = T)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 6.08%
## Confusion matrix:
##      0  1 class.error
## 0 5049 39 0.007665094
## 1  293 81 0.783422460
```

```
#Best tree
print(RF.1000)
```

```
##
## Call:
##  randomForest(formula = AS ~ ., data = ESPN2, ntree = 1000, importance = T)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 4
```

```
##
##         OOB estimate of  error rate: 6.08%
## Confusion matrix:
##      0  1 class.error
## 0 5049 39 0.007665094
## 1  293 81 0.783422460
```

```
#Importance plot
varImpPlot(RF.1000)
```



vi. Boosting. Visualize the best tree and the importance plot. (Chapter 8)

```
set.seed(1)

BOOST.50 <- gbm(AS ~ ., data = qual_train, distribution = "multinomial",
                n.trees = 50)
BOOST.est.50 <- predict.gbm(object = BOOST.50, newdata = qual_test,
                            type = "response", n.trees = 50)
labels.50 <- colnames(BOOST.est.50)[apply(BOOST.est.50, 1, which.max)]
BOOST.cm.50 <- confusionMatrix(qual_test$AS, as.factor(labels.50))
print(BOOST.cm.50)
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction    0    1
##          0 1008   14
##          1   63    8
##
##                Accuracy : 0.9296
##                  95% CI : (0.9127, 0.944)
##     No Information Rate : 0.9799
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1458
##
##  Mcnemar's Test P-Value : 4.498e-08
##
##             Sensitivity : 0.9412
##             Specificity : 0.3636
##          Pos Pred Value : 0.9863
##          Neg Pred Value : 0.1127
##              Prevalence : 0.9799
##          Detection Rate : 0.9222
##    Detection Prevalence : 0.9350
##       Balanced Accuracy : 0.6524
##
##        'Positive' Class : 0
##
```

```
BOOST.100 <- gbm(AS ~ ., data = qual_train, distribution = "multinomial",
                 n.trees = 100)
BOOST.est.100 <- predict.gbm(object = BOOST.100, newdata = qual_test,
                             type = "response", n.trees = 100)
labels.100 <- colnames(BOOST.est.100)[apply(BOOST.est.100, 1, which.max)]
BOOST.cm.100 <- confusionMatrix(qual_test$AS, as.factor(labels.100))
print(BOOST.cm.100)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1009   13
##          1   60   11
##
##                Accuracy : 0.9332
##                  95% CI : (0.9168, 0.9473)
##     No Information Rate : 0.978
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2055
##
##  Mcnemar's Test P-Value : 7.289e-08
##
##             Sensitivity : 0.9439
##             Specificity : 0.4583
##          Pos Pred Value : 0.9873
```

```
##          Neg Pred Value : 0.1549
##              Prevalence : 0.9780
##          Detection Rate : 0.9231
##    Detection Prevalence : 0.9350
##       Balanced Accuracy : 0.7011
##
##        'Positive' Class : 0
##
```

```
BOOST.500 <- gbm(AS ~ ., data = qual_train, distribution = "multinomial",
                 n.trees = 500)
BOOST.est.500 <- predict.gbm(object = BOOST.500, newdata = qual_test,
                             type = "response", n.trees = 500)
labels.500 <- colnames(BOOST.est.500)[apply(BOOST.est.500, 1, which.max)]
BOOST.cm.500 <- confusionMatrix(qual_test$AS, as.factor(labels.500))
print(BOOST.cm.500)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1004   18
##          1   59   12
##
##                Accuracy : 0.9296
##                  95% CI : (0.9127, 0.944)
##     No Information Rate : 0.9726
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.207
##
##  Mcnemar's Test P-Value : 5.154e-06
##
##             Sensitivity : 0.9445
##             Specificity : 0.4000
##          Pos Pred Value : 0.9824
##          Neg Pred Value : 0.1690
##              Prevalence : 0.9726
##          Detection Rate : 0.9186
##    Detection Prevalence : 0.9350
##       Balanced Accuracy : 0.6722
##
##        'Positive' Class : 0
##
```

```
BOOST.1000 <- gbm(AS ~ ., data = qual_train, distribution = "multinomial",
                  n.trees = 1000)
BOOST.est.1000 <- predict.gbm(object = BOOST.1000, newdata = qual_test,
                              type = "response", n.trees = 1000)
labels.1000 <- colnames(BOOST.est.1000)[apply(BOOST.est.100, 1, which.max)]
BOOST.cm.1000 <- confusionMatrix(qual_test$AS, as.factor(labels.1000))
print(BOOST.cm.1000)
```

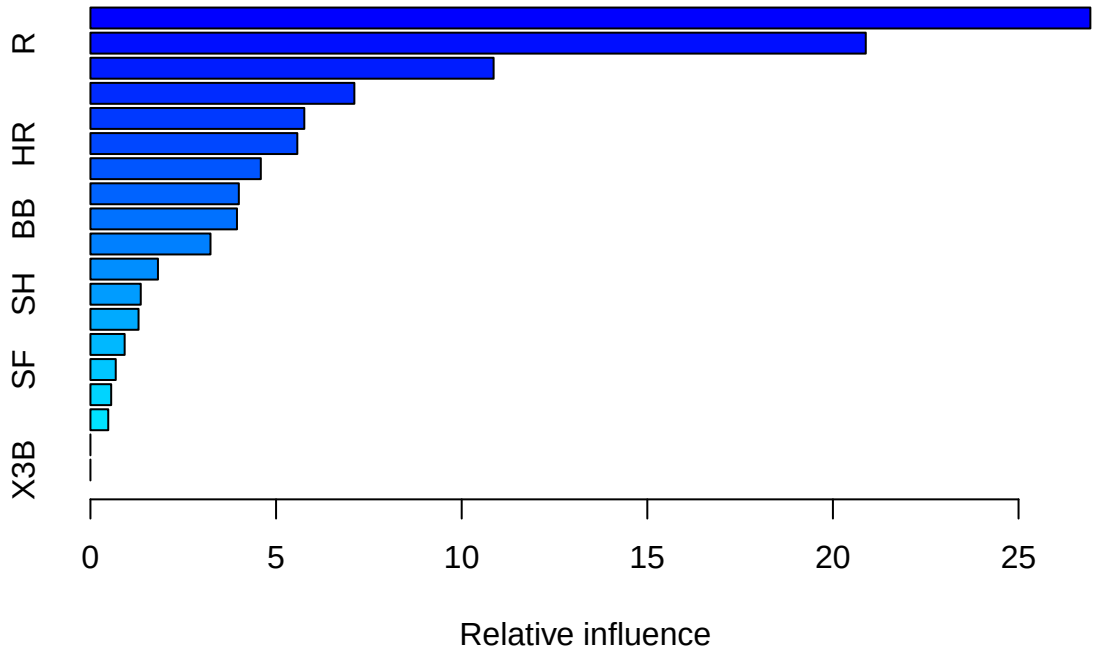
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1009   13
##          1   60   11
##
##                Accuracy : 0.9332
##                  95% CI : (0.9168, 0.9473)
##     No Information Rate : 0.978
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2055
##
##  Mcnemar's Test P-Value : 7.289e-08
##
##             Sensitivity : 0.9439
##             Specificity : 0.4583
##          Pos Pred Value : 0.9873
##          Neg Pred Value : 0.1549
##              Prevalence : 0.9780
##          Detection Rate : 0.9231
##    Detection Prevalence : 0.9350
##       Balanced Accuracy : 0.7011
##
##        'Positive' Class : 0
##
```

```
#Best tree
print(BOOST.cm.100)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1009   13
##          1   60   11
##
##                Accuracy : 0.9332
##                  95% CI : (0.9168, 0.9473)
##     No Information Rate : 0.978
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2055
##
##  Mcnemar's Test P-Value : 7.289e-08
##
##             Sensitivity : 0.9439
##             Specificity : 0.4583
##          Pos Pred Value : 0.9873
##          Neg Pred Value : 0.1549
##              Prevalence : 0.9780
##          Detection Rate : 0.9231
##    Detection Prevalence : 0.9350
```

```
##       Balanced Accuracy : 0.7011
##
##        'Positive' Class : 0
##
```

```
#Importance plot
summary(BOOST.100)
```




```
##           var    rel.inf
## TB         TB 26.9341741
## R           R 20.8847110
## RBI       RBI 10.8600746
## IBB       IBB  7.1092190
## HBP       HBP  5.7610728
## HR         HR  5.5724612
## G           G  4.5890247
## H           H  3.9975052
## BB         BB  3.9472366
## AB         AB  3.2336849
## GIDP     GIDP  1.8187428
## SH         SH  1.3551402
## SO         SO  1.2959177
## X2B       X2B  0.9213099
## SF         SF  0.6813222
## SB         SB  0.5584836
```

```
## CS         CS  0.4799196
## yearID yearID  0.0000000
## X3B       X3B  0.0000000
```

For iv, v and vi, calculate the Gini index on each leaf of the final tree to examine the purity of the node. Add an importance plot for each classifier. (Chapter 8)

Use Cross Validation or Validation Set Approach to compare the estimated test errors for the above classification. (Chapter 5) Find the confusion matrix (true/false positive/negative table) for each classification for comparison. (Chapter 4)

Approach	Estimated Test Error Rate
Logistic Regression	0.06495883
Linear Discriminant Analysis	0.07410796
Quadratic Discriminant Analysis	0.1939616
Bagging	0.0646
Random Forest	0.0608
Boosting	0.0668

Random forest with 1000 trees performed best among all of the classification models. This model is generally robust enough to deal with outliers in the data so I agree with the test error rate results. I would use random forest when predicting MLB all stars.

Part 8

Summary of finding and conclusion. Add references if there is any.

When predicting future fantasy baseball scoring the simpler models appeared to outperform many more complicated models. I believe this is because they are more robust and less prone to overfitting. I would prefer to use Best Subset Selection or Lasso to find useful covariates. I believe that domain knowledge is key so I would like to run the important covariates by people with more experience than myself.

When predicting all-star selections using only batting statistics random forest with 1000 trees performed best among all of the classification models. This model is generally robust enough to deal with outliers in the data so I agree with the test error rate results. Tree based methods tended to outperform all other methods so I would continue to use bagging or boosting since they had similar estimated test error rates.

A potential issue with both of my predictions is the national league's use of the pitcher in the batting order. Half of baseball requires the pitcher to hit so a they may have poor batting numbers yet still be selected as an all-star. Their fantasy scoring also only uses their pitching numbers. Overall pitchers do not play every day or bat very often so the impact on my models will be minimized, but a dataset that is able to classify players would be useful.

REFERENCES

Lewis, M. (2003). Moneyball: The art of winning an unfair game. New York: W.W. Norton.