Salary, Value, and Overall Rating Prediction: Football (Soccer)

Kshitij and Johe

5/17/2020

Introduction

The FIFA games are a series of video games released by EA Sports in which players are able to control (association) football teams and/or players featured in real life football leagues, and play in various modes, single-player and multiplayer, online and offline. Since release, the FIFA games have gained a reputation for being the best and most popular football video game around, and the designers of the game have since chased a perfect reflection of real-life football in their game, whether it be graphics, detail, feel or scale. The games have come a long way from when the first edition was released in 1980, and includes an incredible amount of statistical detail.

We were interested in putting the FIFA statistics to the test to see how consistent they were, and how well they reflect the characters in the game reflect their real-life counterparts. To do this, we have divided our project into two major halves.

The first half of our project deals with a robust exploration of the market value (Value) and weekly wage (Wage) stats of the player characters in the game. We will use the players' ability statistics to model value and wage, both numerically and by manufacturing categories, using a variety of modeling techniques and error analysis.

The second half of our project deals with connecting the FIFA stats of a player character to their real-life counterpart. Particularly, we are interested in seeing how well the performances of a player in real-life predicts their "Overall" rating. Naturally, if a player is considered good, their overall might be high, but the public's conceptions of what makes a "good" player is very subjective, and FIFA is forced to meet the challenge of quantifying it. We want to see how well we can predict a future FIFA overall rating based on a set of performance statistics.

All of the FIFA stats were procured from Kaggle datasets, whereas the seasonal performance datasets were procured from the FBRef database.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.0     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   0.8.5
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts ------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(base)
library(readr)
library(class)
library(glmnet)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 3.0-2
```

```r
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

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

```r
library(gbm)
```

```
## Loaded gbm 2.1.5
```

```r
library(tree)
```

```
## Registered S3 method overwritten by 'tree':
##   method     from
##   print.tree cli
```

```r
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

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

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(FNN)
```

```
##
## Attaching package: 'FNN'

## The following objects are masked from 'package:class':
##
##     knn, knn.cv
```

```r
library(factoextra)
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```r
library(ggrepel)
```

First section

Value and Wage Prediction

Loading the fifa 18 dataset.

```r
fifa18.df <-read.csv("~/Desktop/ADM/Final Project/fifa18.csv")
```

Cleaning the dataset.

```r
#Keep only numeric variables
foot2.df <- fifa18.df%>%
  dplyr::select(-X, -Name, -Photo, - Nationality, -Club, -Flag, -Club.Logo, -ID, -Preferred.Positions)

#Remove +/- stats from the dataset
foot2.df <- as.data.frame(sub("[\\+\\-]\\d+", "\\1", as.matrix(foot2.df)))

# changing vlaue in k to million
foot2.df <- foot2.df%>%
  mutate(isk = ifelse(str_detect(as.character(Value), "K"),1,0))

#Make all values numeric
foot2.df[] <- sapply(foot2.df[], as.character)
foot2.df[] <- sapply(foot2.df[], parse_number)

#Change NAs to 0 (for GK stats)
foot2.df[is.na(foot2.df)] <- 0

# Converting all 'Value' measure in terms of million and filtering 0 Value.
foot2.df <- foot2.df%>%
  mutate(Value = ifelse(isk == 1, Value/1000, Value))%>%
   filter(Value !=0)
```

Building a Stats.df dataset that contains players atrributes like age, acceleration, passing, strength finishing, and so on.

```r
stats.df <- foot2.df %>%
  dplyr::select(1:3, 7:40)
dim(stats.df)
```

```
## [1] 17725    37
```

```r
# 17725    37
names(stats.df)
```

```
##  [1] "Age"                "Overall"            "Potential"
##  [4] "Acceleration"       "Aggression"         "Agility"
##  [7] "Balance"            "Ball.control"       "Composure"
## [10] "Crossing"           "Curve"              "Dribbling"
## [13] "Finishing"          "Free.kick.accuracy" "GK.diving"
## [16] "GK.handling"        "GK.kicking"         "GK.positioning"
## [19] "GK.reflexes"        "Heading.accuracy"   "Interceptions"
## [22] "Jumping"            "Long.passing"       "Long.shots"
## [25] "Marking"            "Penalties"          "Positioning"
## [28] "Reactions"          "Short.passing"      "Shot.power"
## [31] "Sliding.tackle"     "Sprint.speed"       "Stamina"
## [34] "Standing.tackle"    "Strength"           "Vision"
## [37] "Volleys"
```

```r
head(stats.df)[,1:5]
```

```
##   Age Overall Potential Acceleration Aggression
## 1  32      94        94           89         63
## 2  30      93        93           92         48
## 3  25      92        94           94         56
## 4  30      92        92           88         78
## 5  31      92        92           58         29
## 6  28      91        91           79         80
```

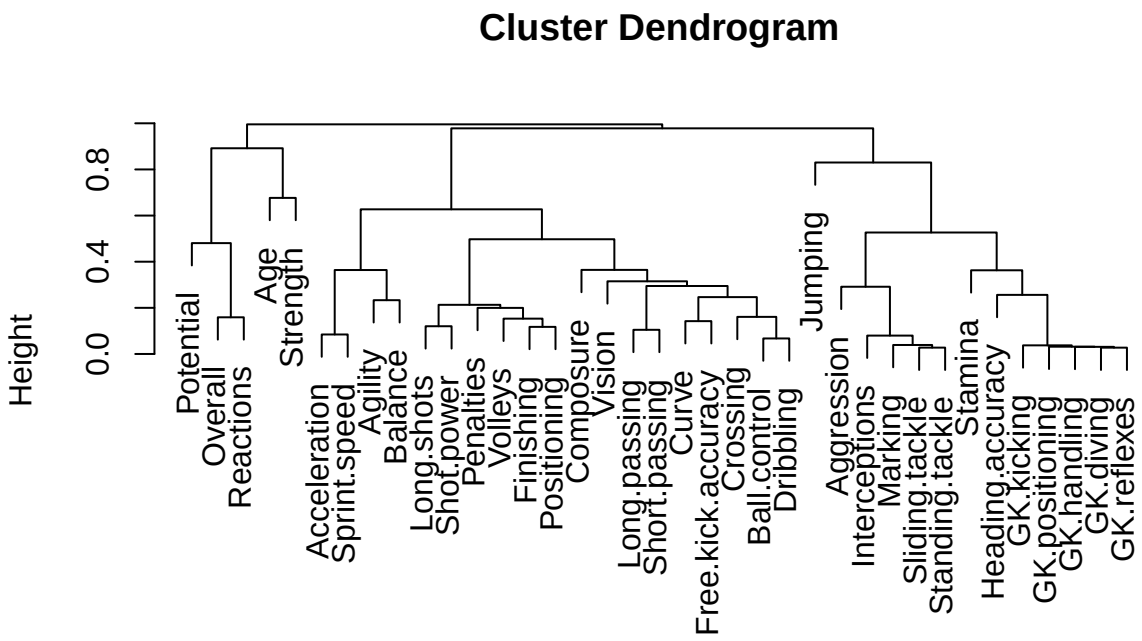
We have 17,725 players with 37 different predictors. For some models we might have to sample half of the observations to make it run faster without compromizing the performace.

Before we go ahead and start making models, let's take a moment to look at the correlation heatmap for the ability statistics of a player. We might find some interesting talking points!

```r
#Correlation Heatmap for Ability Statistics

numPreds <- ncol(stats.df)
feature.mat <- data.matrix(stats.df[,1:numPreds])
feature.mat.scaled <- scale(feature.mat)

feature.cor <- cor(feature.mat)
feature.dist <- 1-as.dist(abs(feature.cor))
feature.cluster <- hclust(feature.dist,method="complete")
plot(feature.cluster)
```




```
feature.names <- names(stats.df[,1:numPreds])
cluster.ord <- feature.cluster$order
feature.cor <- feature.cor[cluster.ord,cluster.ord]
feature.names.ord <- feature.names[cluster.ord]
featureCor.df <- data.frame(feature.cor)
featureCor.df$pred1 <-1:numPreds
featureCor.df <- featureCor.df  %>%
  gather(pred2.name,cor,1:numPreds) %>%
  inner_join(data.frame(pred2.name=feature.names.ord,
                        pred2=1:numPreds)) %>%
  dplyr::select(-pred2.name)
```

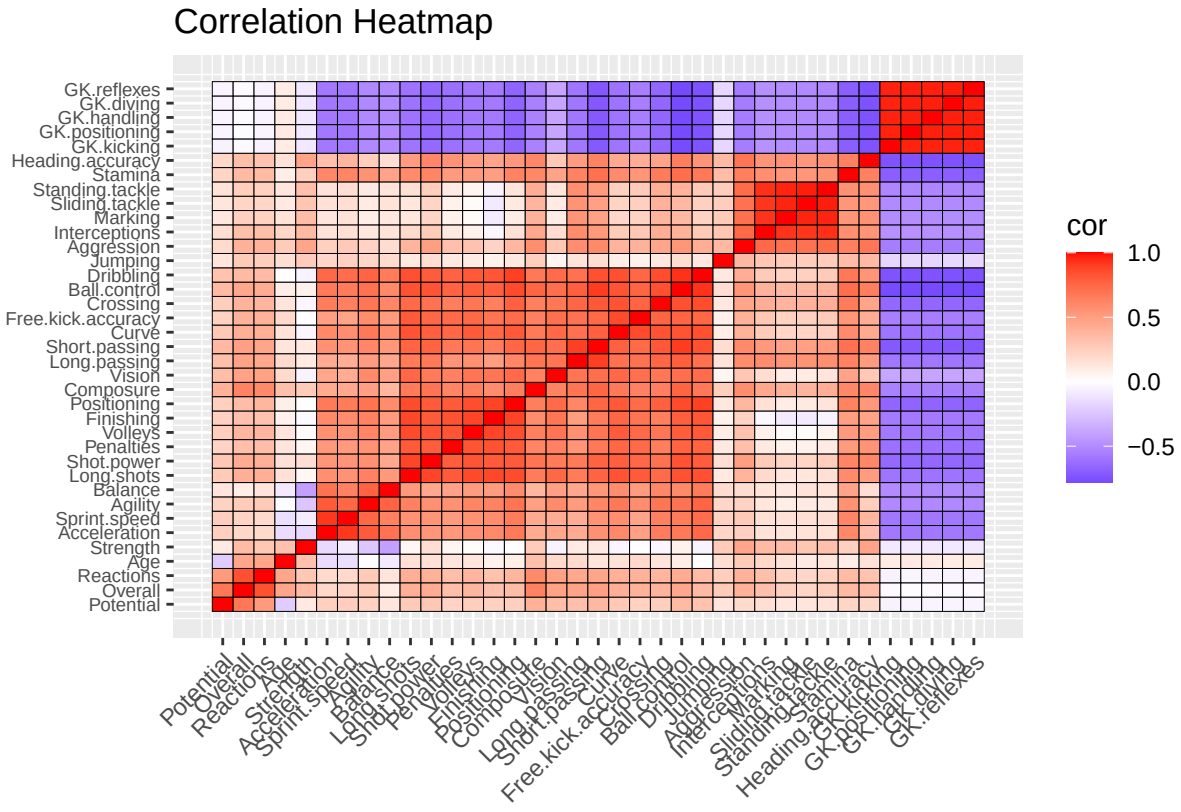
```
## Note: Using an external vector in selections is ambiguous.
## i Use ‘all_of(numPreds)‘ instead of ‘numPreds‘ to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

```
## Joining, by = "pred2.name"
```

```
## Warning: Column ‘pred2.name‘ joining character vector and factor, coercing into
## character vector
```

```
breakLocations=1:numPreds
mpt <- with(featureCor.df,mean(cor))
```

```
featureCor.df %>%
  ggplot()+
  geom_tile(aes(pred1,pred2,fill = cor),color="black")+
  theme(axis.text.x = element_text(angle = 45, hjust = 1,size=9),
        axis.text.y = element_text(size=7))+
  scale_fill_gradient2(low="blue",high="red",midpoint=0)+
  scale_x_continuous(breaks=breakLocations,labels=feature.names.ord)+
  scale_y_continuous(breaks=breakLocations,labels=feature.names.ord)+
  labs(title="Correlation Heatmap",
       x="",y="")
```



As one might intuit, the offensive stats are more correlated with each other, while being less correlated with defensive stats. Meanwhile, goalkeeping stats are correlated opposite with non-goalkeeping stats! That makes sense, as goalkeepers have a unique job, and so they tend to be worse at being outfielders, whereas outfielders tend be worse at goalkeeping. Interrelated stats such as "Ball Control" and "Dribbling", or "Vision" and the passing stats are quite correlated. Again, that makes sense; players who are good dribblers tend to be good controllers of the ball, and players who pass well tend to have great vision, and vice versa in both cases! We see that stats such as "Reactions", "Composure" and "Stamina" are nicely correlated with "Overall", indicating how important they are to overall play.

Our first section is further divided into four sub parts.

- Value Prediction (numerical)
- Value Prediction(catagorical)

- Wage Prediction(numerical)
- Wage Prediction(catagorical)

In order to achieve our goal, we will use the following prediction algorithms for each parts:

- Penalized Lasso regression
- Penalized Ridge regression
- Forward selection
- Backward selection
- KNN regression
- Decision Trees
- Bagging
- RandomForest
- Boosting

Let's start our journey by building models to predict Value of players (numerical).

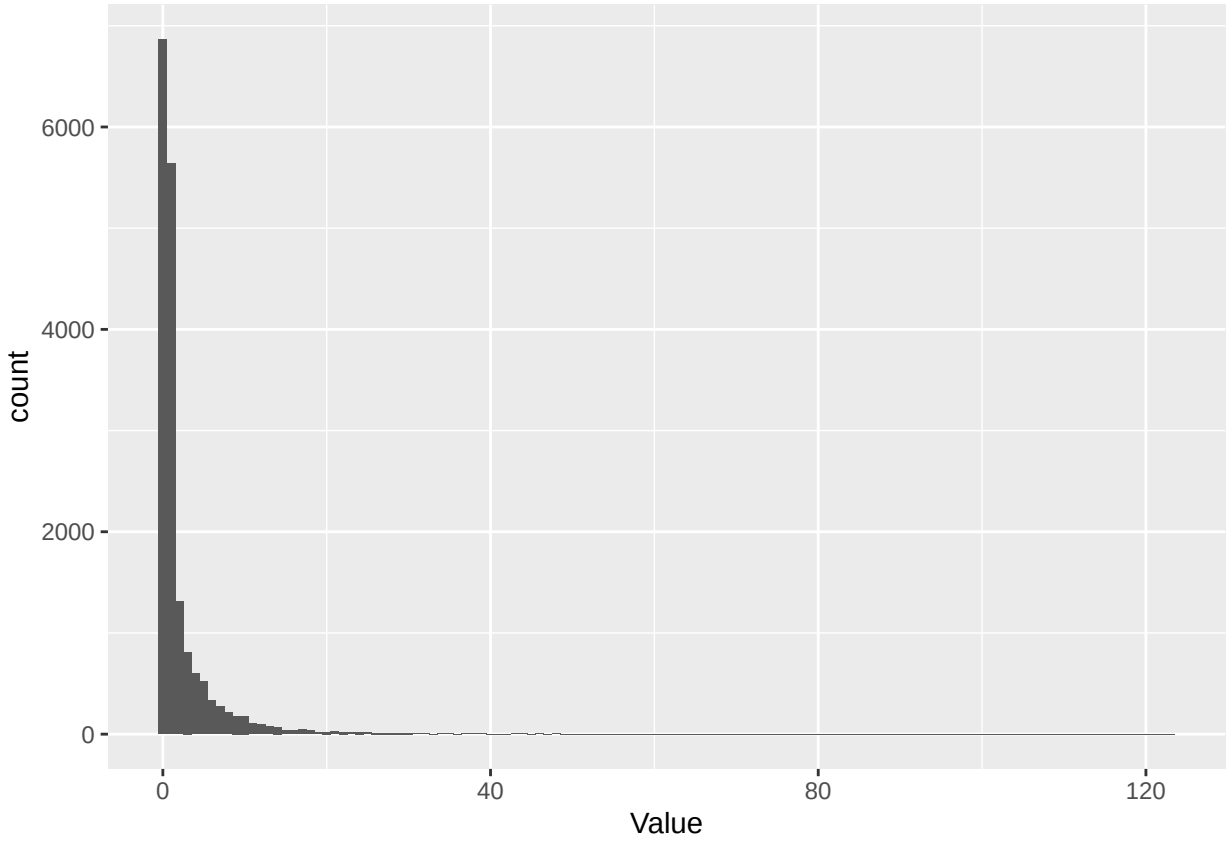
Part (A): Predicting Value of Football (soccer) players.

Value refers to the current market price of players or how much would they cost if they were to be sold to other clubs. Our main goal for this section is to make a powerful predictive model that uses players statistics to predict their market values accurately. In order to achieve that, we will run many algorithms and compared how they perform based on the mean squared error.

In the second half, we will catagorize the Value into four differet catagories (High Value, Meidum High, Medium, Low) and use classificaiton techniques to predict the catagorical Values. Now, Let's get started.

First, we need to create a dataset that contains players statistics and Value.

```
Value <- foot2.df[,c('Value')]
value.df <- (cbind(stats.df, Value)) # combining two datasets
ggplot(value.df, aes(x=Value))+geom_histogram(binwidth = 1) # some odd outliers
```

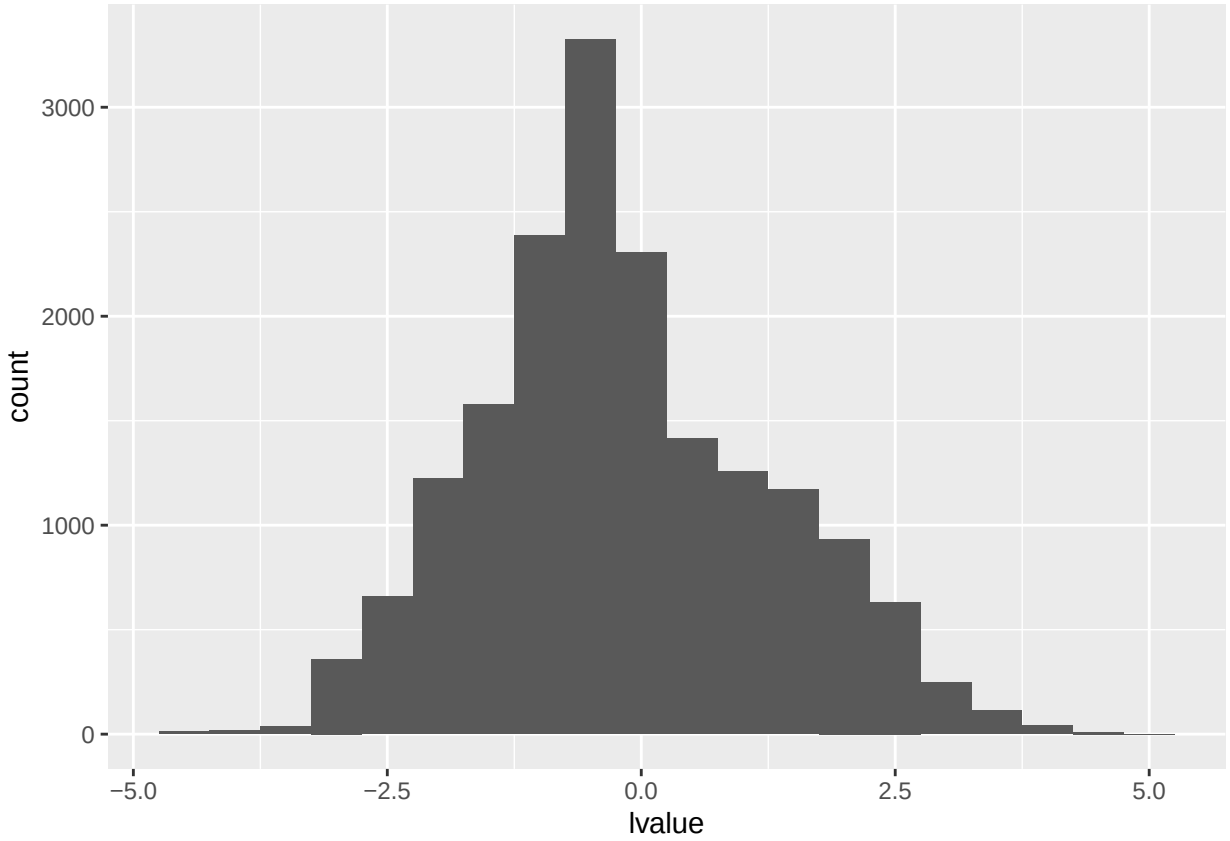


Looks like Value is very right skewed. Some of the most famous football players have extremely high market price, for eg: Messi cost 105 million dollars. Most of the famouse league players like English premier league, Laliga, Chamionscup have high values which explains the skeweness.

However, we decided to log the Value so that it looks more normal and become helpful for our prediction models.

```
value.l.df <- value.df%>%
  mutate(lvalue = log(Value))%>%# logging the value
  dplyr::select(-Value)

ggplot(value.l.df, aes(x=lvalue))+geom_histogram(binwidth = 0.5) # some odd outliers
```



Yes, it looks much more normal.

Let's scale the dataset for our KNN and penalized models.

```
value.ls.df <-scale(as.matrix(value.l.df))
n <- nrow(value.ls.df)
samp <- sample(1:n, n/3, replace = F)
val.x <- value.ls.df[samp,1:ncol(value.ls.df)-1]
val.y <- value.ls.df[samp,ncol(value.ls.df)]
```

Bulding Models

KNN regression

Let's try KNN regression and see how we perform.

```
# Takes in kvla and numfold, performs crossvalidation, and returns mse and se.
knnMSE_SE.val <- function(kVal,numFolds=10){
  numFolds <- 10
  N <- nrow(val.x)
  folds <- sample(1:numFolds,N,rep=T)
  errs <- numeric(numFolds)

  for(fold in 1:numFolds){
    train.x <- val.x[folds != fold,]
```

```
    train.y <- val.y[folds != fold]
    test.x <- val.x[folds == fold,]
    test.y <-  val.y[folds == fold]
    knn.mod <- knn.reg(train.x,test.x,train.y,k=kVal)
    errs[fold] <- with(knn.mod,mean((pred-test.y)^2))
  }
c(mean(errs),sd(errs))
}
# Checking if funtion works
knnMSE_SE.val(17)
```

```
## [1] 0.066169055 0.009913229
```

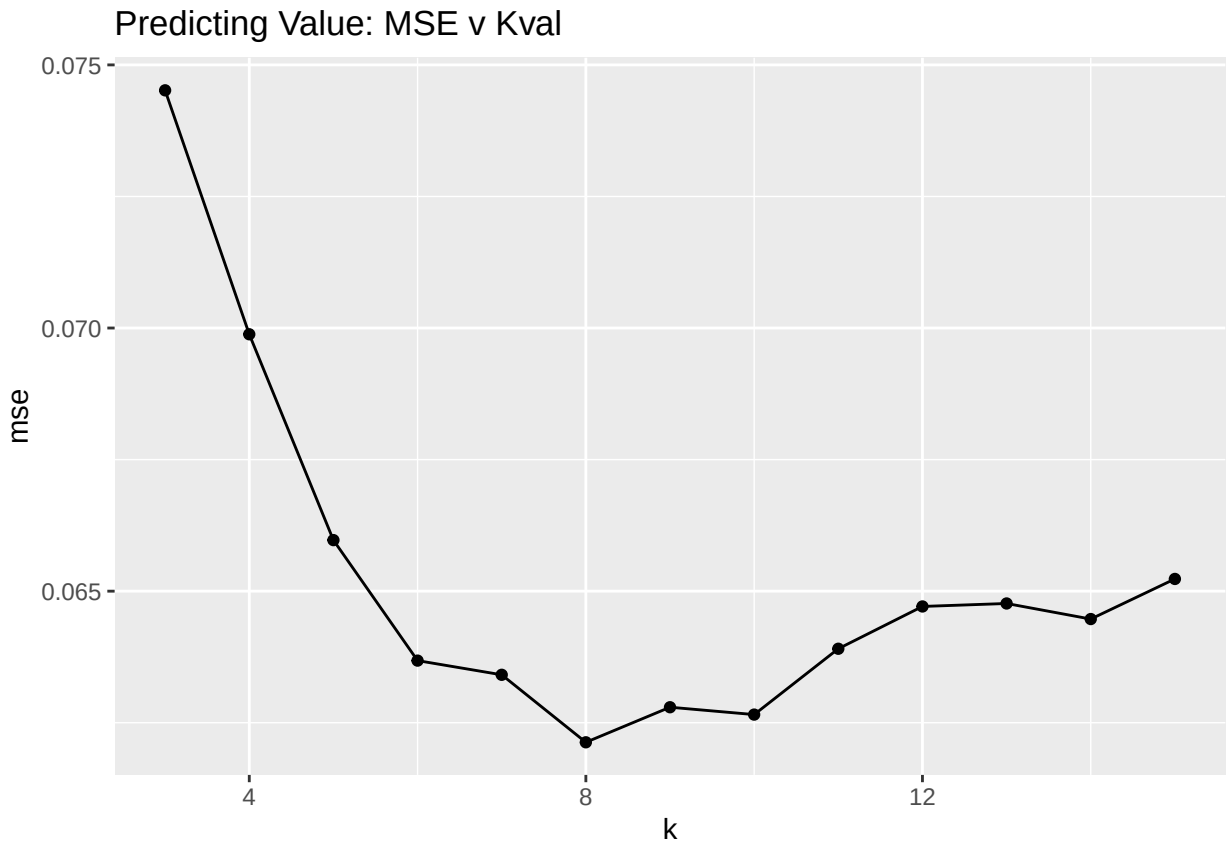
```
# 0.068554581 0.006331594
```

Looks like our KNN cv function works well with mse of 0.068.

Now, we will plug in a range of k values and plot a grap to see which k value has the least associated mse.

```
knnMSE.val <- function(kVal,numFolds=10){ knnMSE_SE.val(kVal,numFolds)[1]}
maxK <- 15
allMSE.val <- map_dbl(3:maxK,knnMSE.val) #

data.frame(k=3:maxK,mse=allMSE.val) %>%
  ggplot(aes(k,mse)) +
  geom_point()+
  geom_line()+
  labs(title="Predicting Value: MSE v Kval")
```



Looks like the best value is around 10. Before running the optimal kNN model, we will use 1 SE rule and pick the least k value that satisfies the minimum threshold of mse cutoff.

```
id <- which.min(allMSE.val)
(kMin <- (3:maxK)[id])
```

```
## [1] 8
```

```
(mseInfo <- knnMSE_SE.val(kMin))
```

```
## [1] 0.063003609 0.003290033
```

```
# 0.065747730 0.005831267
mseCut <- mseInfo[1]+mseInfo[2]
(theVals <- (3:maxK)[allMSE.val < mseCut])
```

```
##  [1]  5  6  7  8  9 10 11 12 13 14 15
```

```
(k1SE <- min(theVals))
```

```
## [1] 5
```

```
allMSE.val[k1SE]
```

```
## [1] 0.06341059
```

```
mse <- knnMSE_SE.val(k1SE)
(mse.knn.val <- mse[1])
```

```
## [1] 0.06645651
```

Our KNN model produced a mean squared error (mse) of 0.071, which is not bad.

Lasso penalized regression

Well, that was just one algorithm. Let's see if Lasso performs better. Since we have 37 predictors, I believe that peanalized regression should perform better as it takes into consideration multicollinearity between predictors.

First, we need a grid of lambda values. It is important to remember that there is no natural scale for the lambda values. After some experimentation, I settled on this grid.

```
lambda.grid <- 10^seq(-5,2,length=100)
```

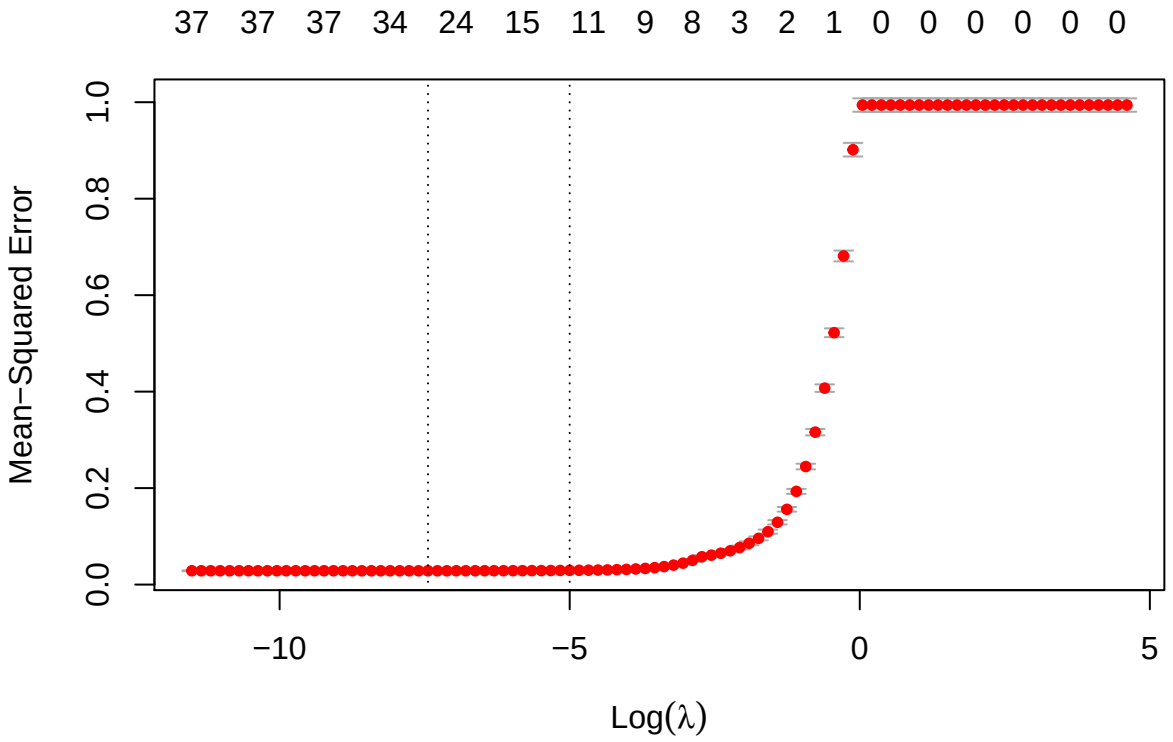
Cross-validation and lambda selection

Now we can run cross-validated (cv) lasso and look at the results. This is basically going to use cv on all the lambga values, store associated errors, and pick the lambda with least error, which we want.

```
samp <- sample(1:n, n/3)
val.x <- value.ls.df[samp,1:ncol(value.ls.df)-1]
val.y <- value.ls.df[samp,ncol(value.ls.df)]
```

```
mod.lasso.cv.val <- cv.glmnet(val.x,
                              val.y,
                              alpha=1,
                              lambda=lambda.grid)

plot(mod.lasso.cv.val)
```




```
# picking lambda val within 1 se with minimum mse.
(lambda.opt.lasso <- mod.lasso.cv.val$lambda.1se)
```

```
## [1] 0.006734151
```

```
# using a built in function, we can directly access the mse associated with best lambda within 1 se.
(mse.lasso.val <-with(mod.lasso.cv.val,cvm[lambda==lambda.1se]))
```

```
## [1] 0.02941026
```

```
mod.lasso.opt <- glmnet(val.x, val.y, alpha=1, lambda=lambda.opt.lasso)

coefs.opt <- data.matrix(coefficients(mod.lasso.opt)!=0)
(lassoVariables <- rownames(coefs.opt)[coefs.opt==TRUE])
```

```
##  [1] "(Intercept)"       "Age"               "Overall"
##  [4] "Potential"         "Finishing"         "Free.kick.accuracy"
##  [7] "Heading.accuracy"  "Marking"           "Penalties"
## [10] "Positioning"       "Reactions"         "Short.passing"
## [13] "Shot.power"        "Stamina"           "Vision"
## [16] "Volleys"
```

Wow, lasso with mse of 0.029 performs better than KNN.

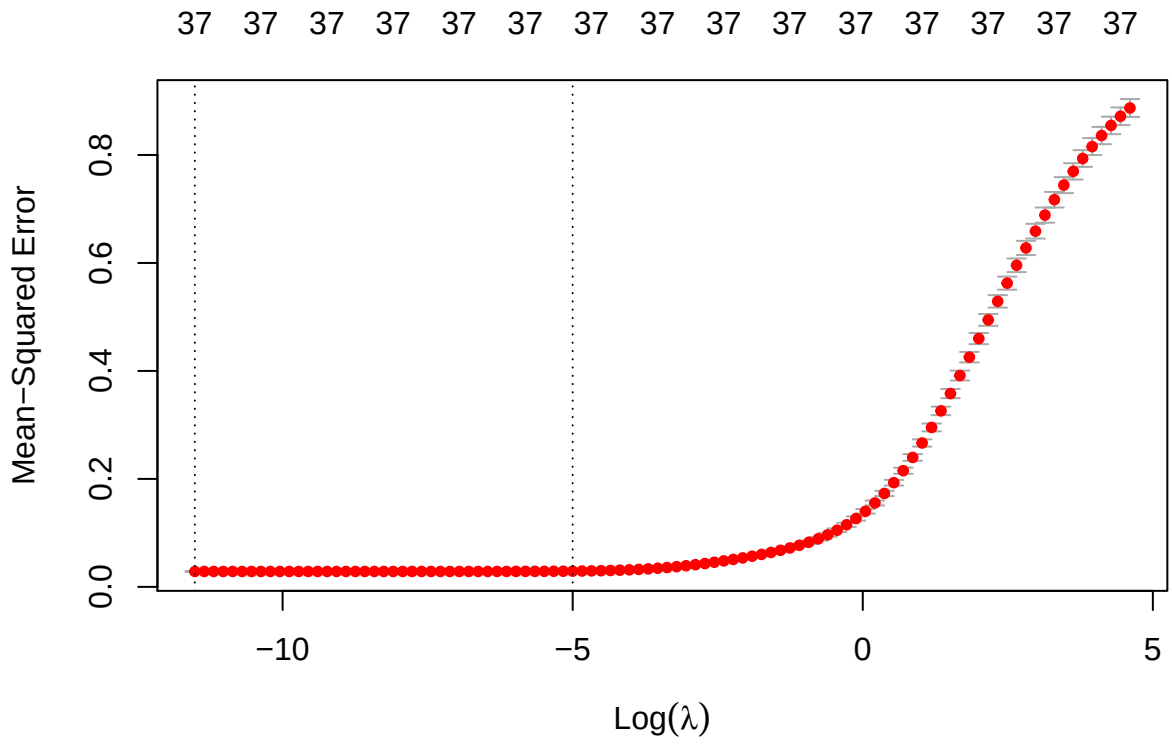
Ridge regression.

Ridge regression shrinks the unnnecessary variables to zero and also takes care of multicollinearity.

Let's run crossvalidated lasso and look at the results.

```
mod.ridge.cv.val <- cv.glmnet(val.x,
                              val.y,
                              alpha=0,
                              lambda=lambda.grid)

plot(mod.ridge.cv.val)
```




```
(mse.ridge.val <-with(mod.ridge.cv.val,cvm[lambda==lambda.1se]))
```

```
## [1] 0.02917389
```

```
# picking lambda val within 1 se with minimum mse.
(lambda.opt.ridge <- mod.ridge.cv.val$lambda.1se)
```

```
## [1] 0.006734151
```

```
c(mse.lasso.val, mse.ridge.val, mse.knn.val)
```

```
## [1] 0.02941026 0.02917389 0.06645651
```

Ridge and Lasso are neck to neck. Wonderful results so far.

Forward Selection

Now, we will perform forward seleciton on our dataset. The idea here is to slowly build the model by selecting the predictors that explains the most variabiliy in reponse variable.

First, let's set up soome vectors like available and model predictors which we will use to keep track of our models as we build them.

```
numPreds <- ncol(value.l.df)-1
availPreds <- 1:numPreds
modelPreds.for.val <- c()
```

The main loop for Forward selection

```
## keep track of the R2 for reference
maxR2 <- c()
##Keep going as long as there are available predictors left
while(length(availPreds) > 0){
##add predictor which increases R^2 the most
##keep track of the R^2 values for reference
  allR2 <- c()
  for(id in availPreds){
    ##the augmented predictors
    augPreds <- c(modelPreds.for.val,id)
    ## Build the data frame with the augmented predictors
    data.df <- value.l.df[,c(augPreds,numPreds+1)]
    ##the model and its summary
    mod.curr <- lm(lvalue ~ ., data=data.df)
    mod.sum <- summary(mod.curr)
    ##grab the R^2
    allR2 <- c(allR2,mod.sum$r.squared)
  }
  ##Find the index of the min R^2
  max.id <- which.max(allR2)
  ##get the best predictor and R^2
  bestPred <- availPreds[max.id]
  bestR2 <- max(allR2)
  ##Add these into the collection
  modelPreds.for.val <- c(modelPreds.for.val,bestPred)
  ## remove the bestPred from the availPreds
  availPreds <- setdiff(availPreds,bestPred)
  maxR2 <- c(maxR2,bestR2)
  ##remove bestsPred from avail
  ## Print stuff out for debugging and attention-grabbing
  print(sprintf("Pred Added: %s R^2 Value: %s",bestPred,round(bestR2,3)))
  ##print(modelPreds)
}
```

```
## [1] "Pred Added: 2 R^2 Value: 0.888"
## [1] "Pred Added: 1 R^2 Value: 0.965"
## [1] "Pred Added: 13 R^2 Value: 0.97"
## [1] "Pred Added: 37 R^2 Value: 0.97"
## [1] "Pred Added: 25 R^2 Value: 0.97"
## [1] "Pred Added: 16 R^2 Value: 0.971"
```

```
## [1] "Pred Added: 33 R^2 Value: 0.971"
## [1] "Pred Added: 28 R^2 Value: 0.971"
## [1] "Pred Added: 7 R^2 Value: 0.971"
## [1] "Pred Added: 36 R^2 Value: 0.971"
## [1] "Pred Added: 10 R^2 Value: 0.971"
## [1] "Pred Added: 14 R^2 Value: 0.971"
## [1] "Pred Added: 23 R^2 Value: 0.971"
## [1] "Pred Added: 8 R^2 Value: 0.971"
## [1] "Pred Added: 27 R^2 Value: 0.971"
## [1] "Pred Added: 18 R^2 Value: 0.971"
## [1] "Pred Added: 24 R^2 Value: 0.971"
## [1] "Pred Added: 20 R^2 Value: 0.971"
## [1] "Pred Added: 32 R^2 Value: 0.971"
## [1] "Pred Added: 34 R^2 Value: 0.971"
## [1] "Pred Added: 6 R^2 Value: 0.971"
## [1] "Pred Added: 35 R^2 Value: 0.971"
## [1] "Pred Added: 3 R^2 Value: 0.971"
## [1] "Pred Added: 12 R^2 Value: 0.971"
## [1] "Pred Added: 9 R^2 Value: 0.971"
## [1] "Pred Added: 22 R^2 Value: 0.971"
## [1] "Pred Added: 30 R^2 Value: 0.971"
## [1] "Pred Added: 17 R^2 Value: 0.971"
## [1] "Pred Added: 26 R^2 Value: 0.971"
## [1] "Pred Added: 31 R^2 Value: 0.971"
## [1] "Pred Added: 15 R^2 Value: 0.971"
## [1] "Pred Added: 29 R^2 Value: 0.971"
## [1] "Pred Added: 19 R^2 Value: 0.971"
## [1] "Pred Added: 11 R^2 Value: 0.971"
## [1] "Pred Added: 21 R^2 Value: 0.971"
## [1] "Pred Added: 4 R^2 Value: 0.971"
## [1] "Pred Added: 5 R^2 Value: 0.971"
```

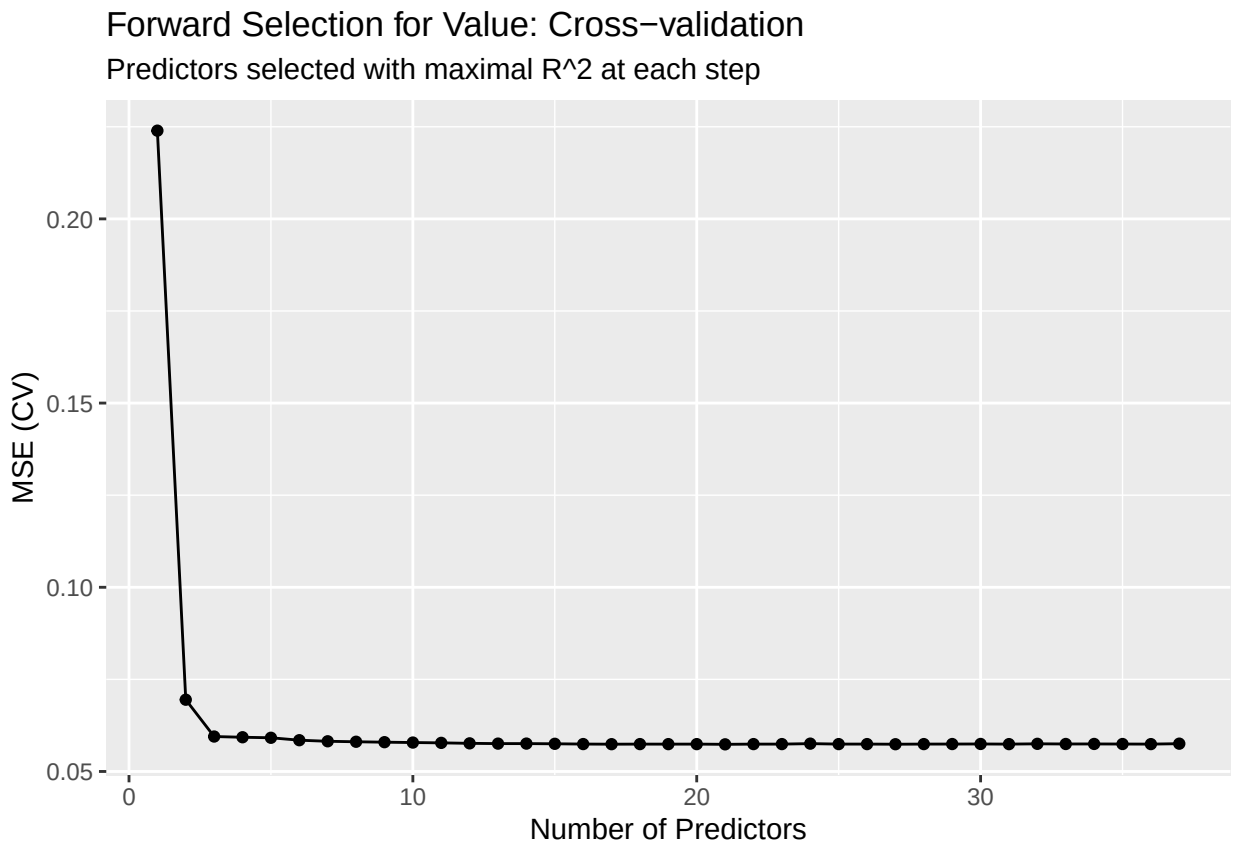
Our R^2 values of aorund 97% looks amazing! Now, we will use map_dbl to simplify the work.

```
## args: a data frame and a number of folds (default to 10).
## ret: k-fold cross-validated MSE
mseCV_SE.val <- function(data.df,numFolds=10){
dataSize <- nrow(data.df)
folds <- sample(1:numFolds,dataSize,rep=T)
mse <- numeric(numFolds)
# fold <- numFolds
for(fold in 1:numFolds){
train.df <- data.df[folds != fold,]
test.df <- data.df[folds == fold,]
mod <- lm(lvalue ~ .,data=train.df)
vals <- predict(mod,newdata=test.df)
mse[fold] <- with(test.df,mean((lvalue-vals)^2))
}
c(mean(mse),sqrt(var(mse)/numFolds))}
```

Now, let's cross validate allthe models and see which model with least predictor reduces mse the most.

```
allMSE.for.val <- map_dbl(1:numPreds,
  function(totPred)
  mseCV_SE.val(value.l.df[,c(modelPreds.for.val[1:totPred],numPreds+1)])[1])

data.frame(numPred=1:numPreds,
mse=allMSE.for.val) %>%
ggplot()+
geom_point(aes(numPred,mse))+
geom_line(aes(numPred,mse))+
labs(title="Forward Selection for Value: Cross-validation",
  subtitle="Predictors selected with maximal R^2 at each step",
  x = "Number of Predictors",
  y = "MSE (CV)")
```



Looks like first 5-10 predictors can reduce a lot of variability.

```
(predMin <- which.min(allMSE.for.val))
```

```
## [1] 21
```

```
# 24
##build this model
temp.df <- value.l.df[,c(modelPreds.for.val[1:predMin],numPreds+1)]
## get both the MSE estimate and the SE
(mseInfo <- mseCV_SE.val(temp.df))
```

```
## [1] 0.057427014 0.001384161
```

```
# 0.057424923 0.001134351
```

Now we need to identify the smallest predictor set within one SE of the min MSE.

```
## add the MSE and the SE.
(mseCut <- mseInfo[1]+mseInfo[2])
```

```
## [1] 0.05881117
```

```
(thePreds <- (1:numPreds)[allMSE.for.val < mseCut])
```

```
##  [1]  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
## [26] 31 32 33 34 35 36 37
```

```
# 6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
# 33 34 35 36 37
```

We can use the predictors starting at the minimum index.

```
(optNumPreds <- min(thePreds))
```

```
## [1] 6
```

```
# 6
(preds.forward.val <- modelPreds.for.val[1:optNumPreds])
```

```
## [1]  2  1 13 37 25 16
```

```
# 2  1 13 37 25 16
```

The cross-validated estimated MSE has already been computed so let's hand pick them.

```
(mse.forward.val <- allMSE.for.val[optNumPreds])
```

```
## [1] 0.05848733
```

```
# 0.05847
names.df <- names(stats.df)
(forwardVars <- names.df[preds.forward.val])
```

```
## [1] "Overall"     "Age"         "Finishing"   "Volleys"     "Marking"
## [6] "GK.handling"
```

```
best.df <- value.l.df[,c(preds.forward.val, ncol(value.l.df))]
mod <- lm(lvalue~., data=best.df)
summary(mod)
```

```
##
## Call:
## lm(formula = lvalue ~ ., data = best.df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.75938 -0.13118  0.00387  0.14998  1.33857
##
## Coefficients:
##               Estimate Std. Error  t value Pr(>|t|)
## (Intercept) -1.223e+01  1.763e-02 -693.707  < 2e-16 ***
## Overall      2.169e-01  3.792e-04  571.968  < 2e-16 ***
## Age         -9.313e-02  4.568e-04 -203.878  < 2e-16 ***
## Finishing    2.076e-03  2.290e-04    9.067  < 2e-16 ***
## Volleys      1.483e-03  2.297e-04    6.455 1.11e-10 ***
## Marking     -2.224e-03  1.459e-04  -15.248  < 2e-16 ***
## GK.handling -3.191e-03  2.284e-04  -13.969  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2418 on 17718 degrees of freedom
## Multiple R-squared:  0.9708, Adjusted R-squared:  0.9707
## F-statistic: 9.801e+04 on 6 and 17718 DF,  p-value: < 2.2e-16
```

```
mseCV_SE.val(best.df)
```

```
## [1] 0.05843780 0.00107564
```

We got mse: 0.058 for forward selection.

Based on the best model for forward selection, Overall, Age, Finishing, Volleys, GK.handeling seems to be some good predictors. Overall variables has the most positive impact because a player with higher overall would have higher value. Similary, an older player would have less value because they would have less career time to play for the club as oppose to a younge player. Similarly, goalkeeper handeling has negative association. GL.handling would be pretty bad for all other players who are not a goal keeper. And usually, goak keepers have less value in general, so it makes sense.

Comparing the mse results so far.

```
c( mse.lasso.val, mse.ridge.val, mse.knn.val, mse.forward.val)
```

```
## [1] 0.02941026 0.02917389 0.06645651 0.05848733
```

- So far, penalized regression seems to be the best ones with mse of 0.0292.

Backward selection

The concept is pretty similar to forward selection but we will start building model backwards.

Main loop for backward selection.

```
availPreds <- 1:numPreds
modelPreds.back <- c()
while(length(availPreds) >1){
allR2 <- c()
pred <- 1
for(pred in availPreds){
testPreds <- setdiff(availPreds,pred)
data.df <- value.l.df[,c(testPreds,numPreds+1)]
mod <- lm(lvalue ~ ., data=data.df)
mod.sum <- summary(mod)
allR2 <- c(allR2,mod.sum$r.squared)
}
max.id <- which.max(allR2)
bestPred <- availPreds[max.id]
##tack the best predictor on the front
modelPreds.back <- c(bestPred,modelPreds.back)
availPreds <- setdiff(availPreds,bestPred)
print(sprintf("Pred Added: %s R^2 Value: %s",bestPred,round(allR2[max.id],3)))
}
```

```
## [1] "Pred Added: 5 R^2 Value: 0.971"
## [1] "Pred Added: 4 R^2 Value: 0.971"
## [1] "Pred Added: 21 R^2 Value: 0.971"
## [1] "Pred Added: 11 R^2 Value: 0.971"
## [1] "Pred Added: 19 R^2 Value: 0.971"
## [1] "Pred Added: 29 R^2 Value: 0.971"
## [1] "Pred Added: 15 R^2 Value: 0.971"
## [1] "Pred Added: 31 R^2 Value: 0.971"
## [1] "Pred Added: 26 R^2 Value: 0.971"
## [1] "Pred Added: 17 R^2 Value: 0.971"
## [1] "Pred Added: 30 R^2 Value: 0.971"
## [1] "Pred Added: 22 R^2 Value: 0.971"
## [1] "Pred Added: 9 R^2 Value: 0.971"
## [1] "Pred Added: 12 R^2 Value: 0.971"
## [1] "Pred Added: 3 R^2 Value: 0.971"
## [1] "Pred Added: 35 R^2 Value: 0.971"
## [1] "Pred Added: 6 R^2 Value: 0.971"
## [1] "Pred Added: 34 R^2 Value: 0.971"
## [1] "Pred Added: 32 R^2 Value: 0.971"
## [1] "Pred Added: 20 R^2 Value: 0.971"
## [1] "Pred Added: 24 R^2 Value: 0.971"
## [1] "Pred Added: 18 R^2 Value: 0.971"
## [1] "Pred Added: 36 R^2 Value: 0.971"
## [1] "Pred Added: 8 R^2 Value: 0.971"
## [1] "Pred Added: 37 R^2 Value: 0.971"
## [1] "Pred Added: 7 R^2 Value: 0.971"
## [1] "Pred Added: 27 R^2 Value: 0.971"
## [1] "Pred Added: 23 R^2 Value: 0.971"
## [1] "Pred Added: 28 R^2 Value: 0.971"
## [1] "Pred Added: 10 R^2 Value: 0.971"
## [1] "Pred Added: 14 R^2 Value: 0.971"
## [1] "Pred Added: 33 R^2 Value: 0.971"
## [1] "Pred Added: 16 R^2 Value: 0.97"
```

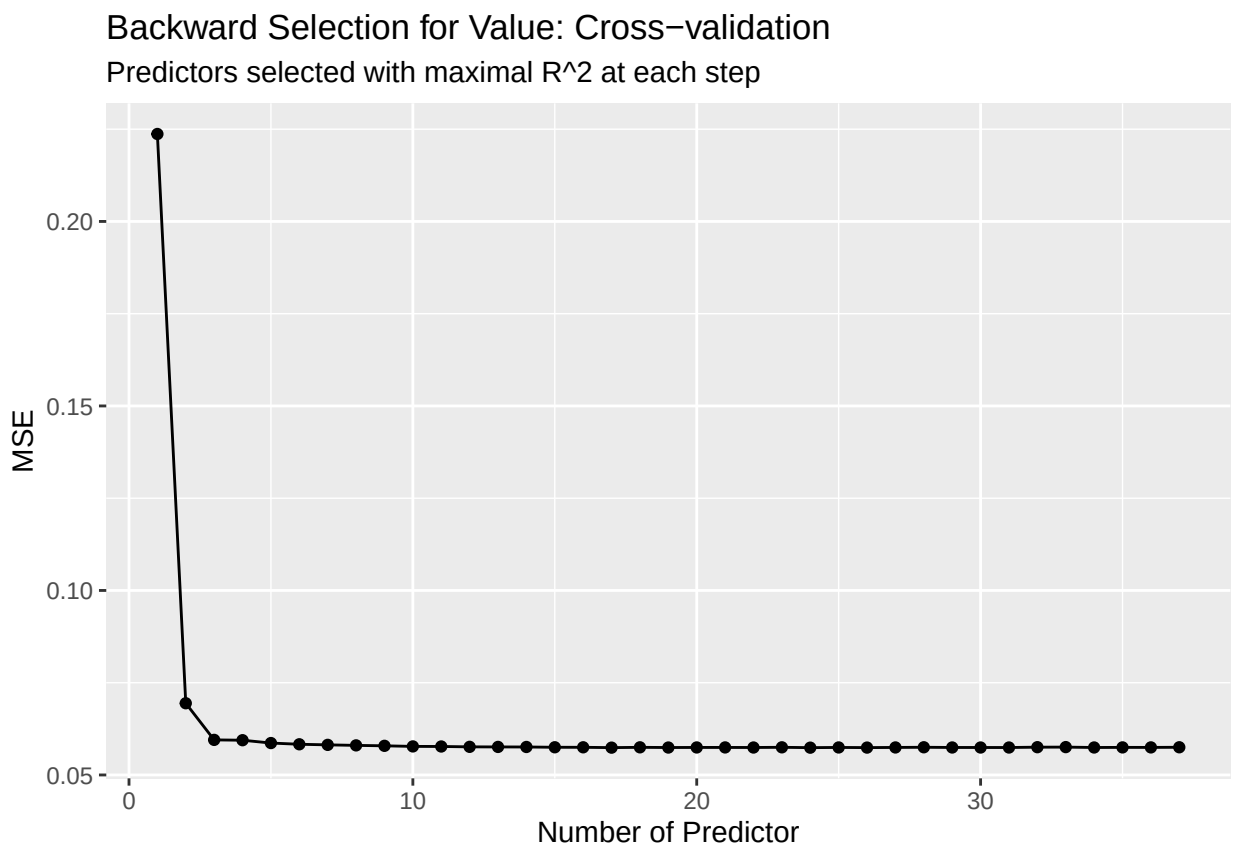
```
## [1] "Pred Added: 25 R^2 Value: 0.97"
## [1] "Pred Added: 13 R^2 Value: 0.965"
## [1] "Pred Added: 1 R^2 Value: 0.888"
```

```
modelPreds.back <- c(availPreds,modelPreds.back)
```

Again, crossvalidating the models to get the mse associated with each number of predictors from best backward seleciton model. We then create a plot.

```
allMSE.val.back <- map_dbl(1:(numPreds),
  function(totPred) mseCV_SE.val(value.l.df[,c(modelPreds.back[1:totPred],numPreds+1)])[1])

data.frame(numPreds=1:(numPreds),MSE=allMSE.val.back) %>%
ggplot()+
geom_point(aes(numPreds,MSE))+
geom_line(aes(numPreds,MSE))+
labs(title="Backward Selection for Value: Cross-validation",
subtitle="Predictors selected with maximal R^2 at each step",
x="Number of Predictor")
```



Finding out the the smallest model.

```
predMin <- which.min(allMSE.val.back)
# build this model
temp.df <- value.l.df[,c(modelPreds.back[1:predMin],numPreds+1)]
```

```r
# get both the MSE estimate and the SE
mseInfo <- mseCV_SE.val(temp.df)
# 0.0574136096 0.0008218357
## add the MSE and the SE.
mseCut <- mseInfo[1]+mseInfo[2]
thePreds.val.back <- (1:numPreds)[allMSE.val.back < mseCut]
#We can use the predictors starting at the minimum index.
optNumPreds <- min(thePreds.val.back)
# 6
(preds.val.back <- modelPreds.back[1:optNumPreds])
```

```
## [1]  2  1 13 25 16 33
```

The cross-validated estimated MSE.

```r
(mse.backward.val <- allMSE.val.back[optNumPreds])
```

```
## [1] 0.05831545
```

```r
# 0.05819308
names.df <- names(stats.df)
(backVars <- names.df[preds.val.back])
```

```
## [1] "Overall"     "Age"         "Finishing"   "Marking"     "GK.handling"
## [6] "Stamina"
```

Comparing mse results.

```r
c( mse.lasso.val, mse.ridge.val, mse.knn.val, mse.forward.val, mse.backward.val)
```

```
## [1] 0.02941026 0.02917389 0.06645651 0.05848733 0.05831545
```

MSE for backward selection is not that good. So far, penalized regression (both lasso and ridge) seem to be the best bet with mse of 0.029.

```r
print(lassoVariables)
```

```
##  [1] "(Intercept)"       "Age"               "Overall"
##  [4] "Potential"         "Finishing"         "Free.kick.accuracy"
##  [7] "Heading.accuracy"  "Marking"           "Penalties"
## [10] "Positioning"       "Reactions"         "Short.passing"
## [13] "Shot.power"        "Stamina"           "Vision"
## [16] "Volleys"
```

```r
print(forwardVars)
```

```
## [1] "Overall"     "Age"         "Finishing"   "Volleys"     "Marking"
## [6] "GK.handling"
```

```
print(backVars)
```

```
## [1] "Overall"     "Age"         "Finishing"   "Marking"     "GK.handling"
## [6] "Stamina"
```

We see that variables like Overall, Age, Finishing, Marking, and stamina seems to be common among our models so far.

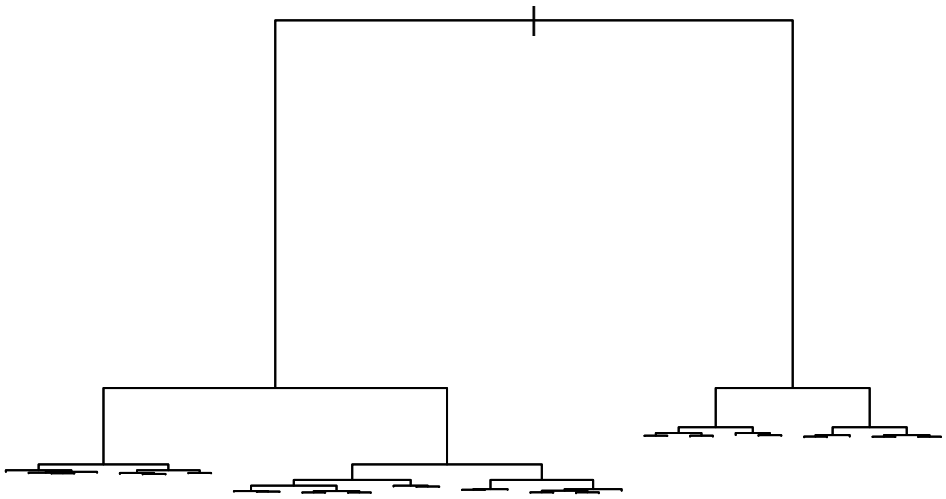
Decision tree

To start, let's build some test and training data by subsetting the large

```
N <- nrow(value.l.df)
n <- 1000
train <- sort(sample(1:N,n,rep=F))
test <- sample(1:N,n,rep=F)
##get rid of overlaps
both <- intersect(train,test)
train <- setdiff(train,both)
test <- setdiff(test,both)
##ok...
train.df <- value.l.df[train,]
test.df <- value.l.df[test,]
```

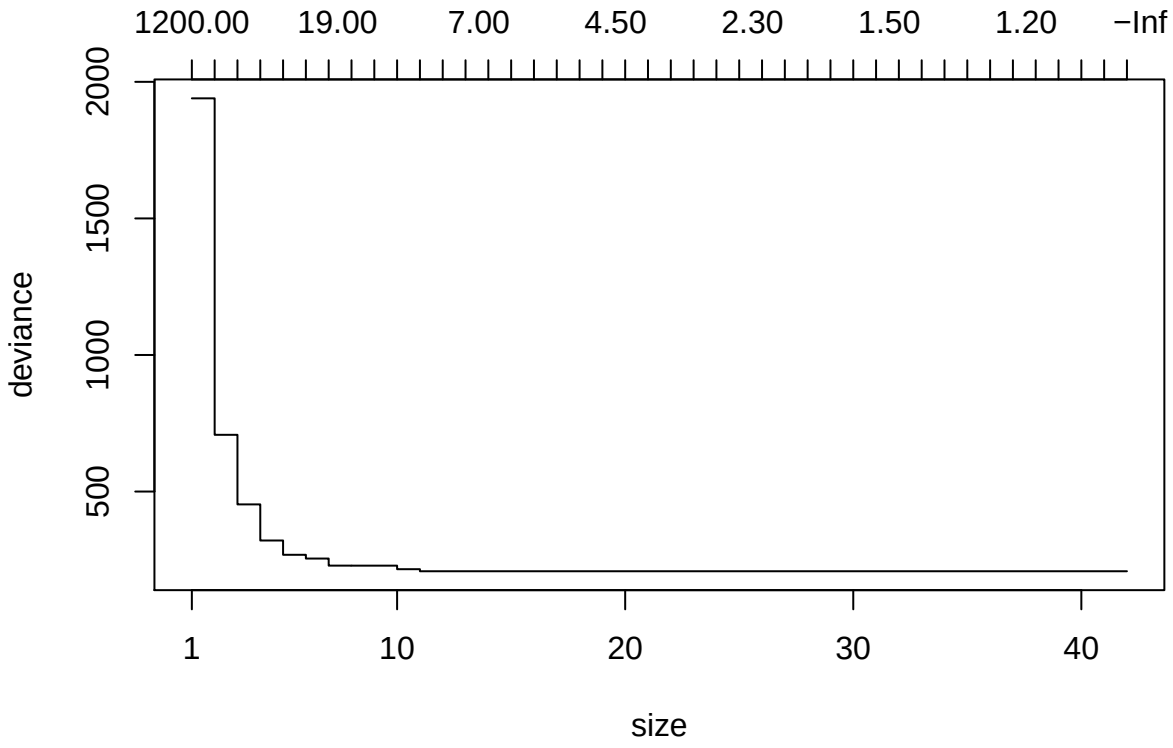
Here's a deep tree built with a very small mindev.

```
mod.tree.val <- tree(lvalue ~ .,
                 data=train.df,
                 control=tree.control(nrow(train.df),
                                      ## small mindev, large tree
                                      mindev=0.0005,
                                      ## keep this small too
                                      minsize=5))
plot(mod.tree.val)
```



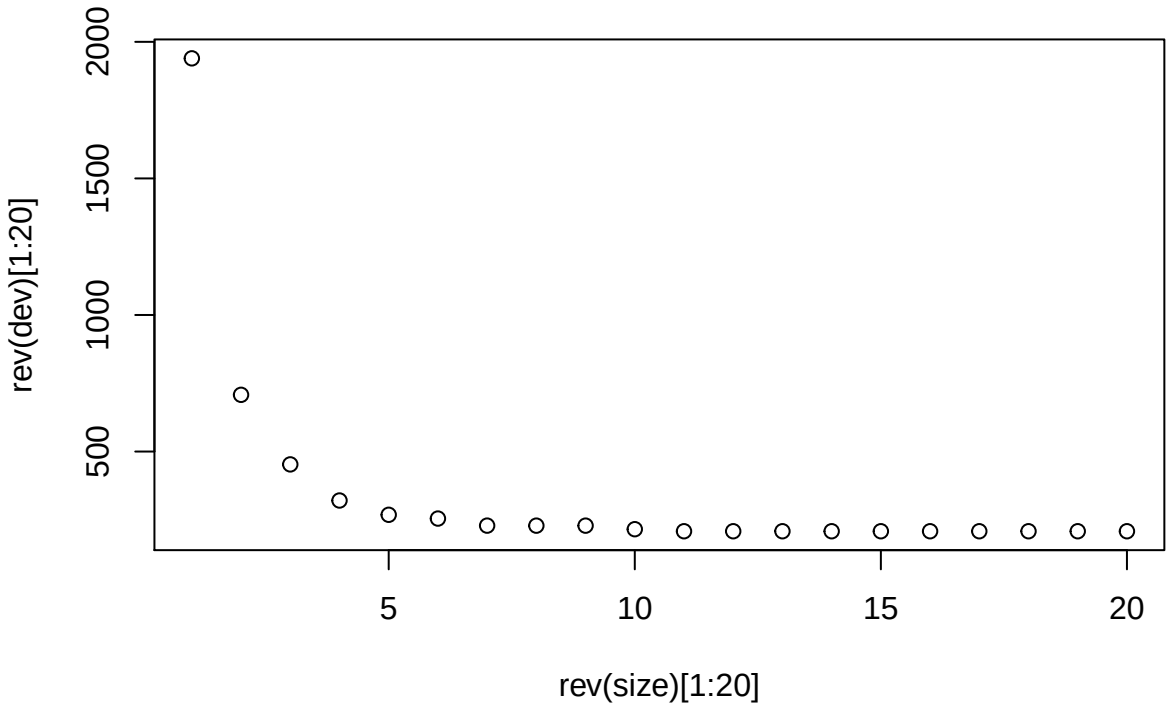
Cross validating the tree.

```
tree.cv.val <- cv.tree(mod.tree.val)
plot(tree.cv.val)
```



Let's take a closer look at the start of this plot.

```
with(tree.cv.val,plot(rev(size)[1:20],rev(dev)[1:20]))
```



Looks like something around 8-9 will work

```
mod.prune.val <- prune.tree(mod.tree.val,best=9)
```

And the predictions.

```
preds.val <- predict(mod.prune.val,newdata=test.df)
(mse.tree.val <- with(test.df,mean((lvalue-preds.val)^2)))
```

```
## [1] 0.216749
```

Comparing mse results.

```
c(mse.lasso.val, mse.ridge.val, mse.knn.val, mse.forward.val,mse.backward.val,mse.tree.val)
```

```
## [1] 0.02941026 0.02917389 0.06645651 0.05848733 0.05831545 0.21674895
```

- Still, no one is close to penalized regression.

Bagging

Test/validation data.

We need to build train and test datasets to train and test our models.

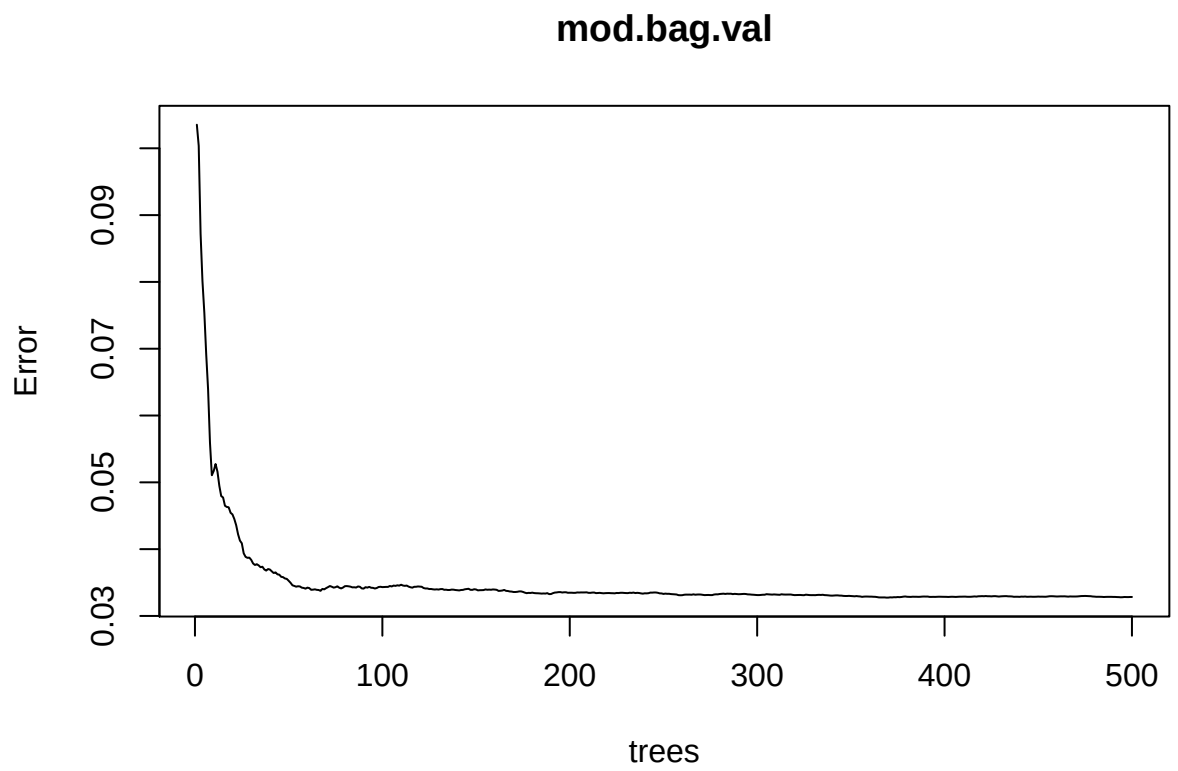
```
N <- nrow(value.l.df)
numPred <- ncol(value.l.df)-1
n <- 1000
build <- sample(1:N,n,rep=F)
data.df <- value.l.df[build,]

##Data to test/validate the model
n <- 1000
test <- sample(setdiff(1:N,build),n,rep=F)
test.df <- value.l.df[test,]
```

Now we will build a Bagged model with 500 tees using the R randomForest function.

Again, in randomForest that is a argument (mtry) which indicates the number of predictors to use at each branch. For now, we'll use all the predictors. Later, we'll limit the number of predictors.

```
numTree <- 500
# data.df <- solubil.df
## The response is the last predictor.
mod.bag.val <- randomForest(lvalue ~ .,
                            data=data.df,
                            ntree=numTree,
                            mtry=numPred)  ## Use all the predictors
plot(mod.bag.val)
```



```
(mse.bag.val <- min(mod.bag.val$mse))
```

```
## [1] 0.03274402
```

The "Mean of squared residuals" is the Out of Bag estimate of the MSE (as opposed to the error rate)

```
c( mse.lasso.val, mse.ridge.val, mse.knn.val, mse.forward.val, mse.backward.val, mse.bag.val)
```

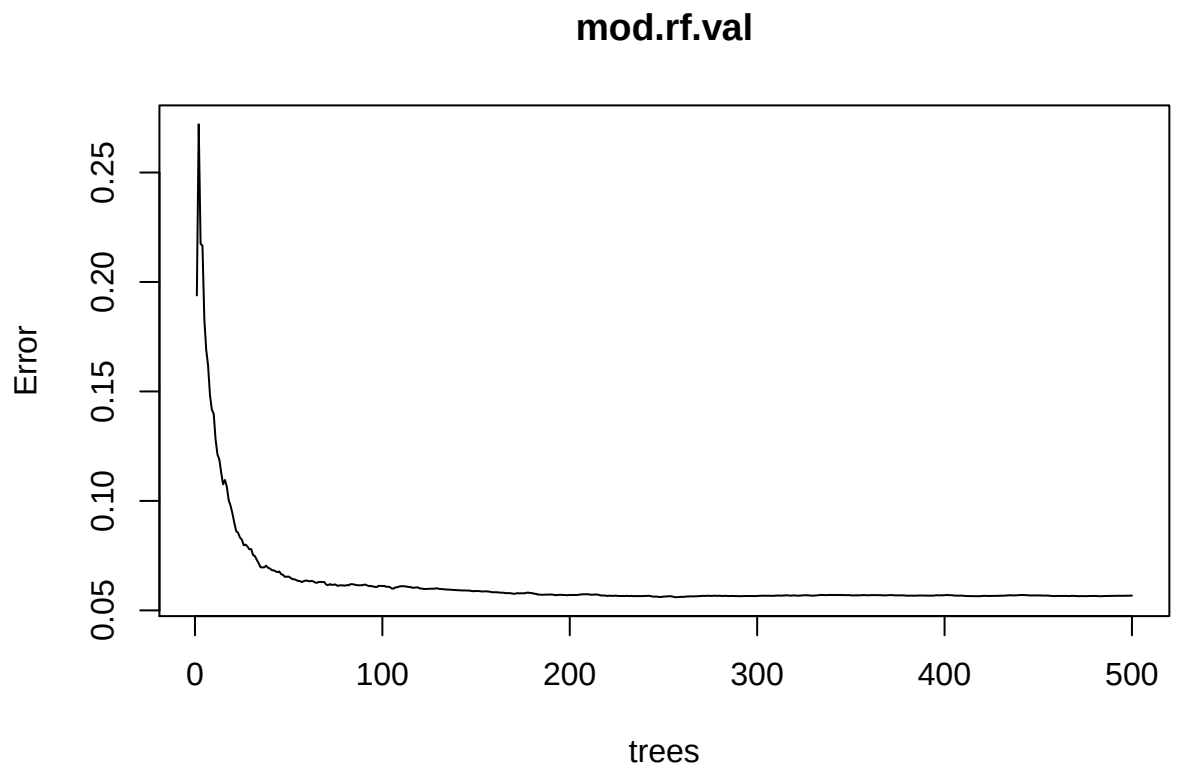
```
## [1] 0.02941026 0.02917389 0.06645651 0.05848733 0.05831545 0.03274402
```

- Baggin seems to do pretty good with mse of 0.030, which is very close to the mse of penalized regressions.

RandomForest

Running Random Forest model with quarter of predictors.

```
mod.rf.val <- randomForest(lvalue ~ .,
                           data=data.df,
                           ntree=numTree,
                           mtry=numPred/3)  ## Use 1/3 predictors
plot(mod.rf.val)
```



```
(mse.rf.val <- min(mod.rf.val$mse))
```

```
## [1] 0.05601985
```

```
c( mse.lasso.val, mse.ridge.val, mse.knn.val, mse.forward.val, mse.backward.val, mse.bag.val, mse.rf.val
```

```
## [1] 0.02941026 0.02917389 0.06645651 0.05848733 0.05831545 0.03274402 0.05601985
```

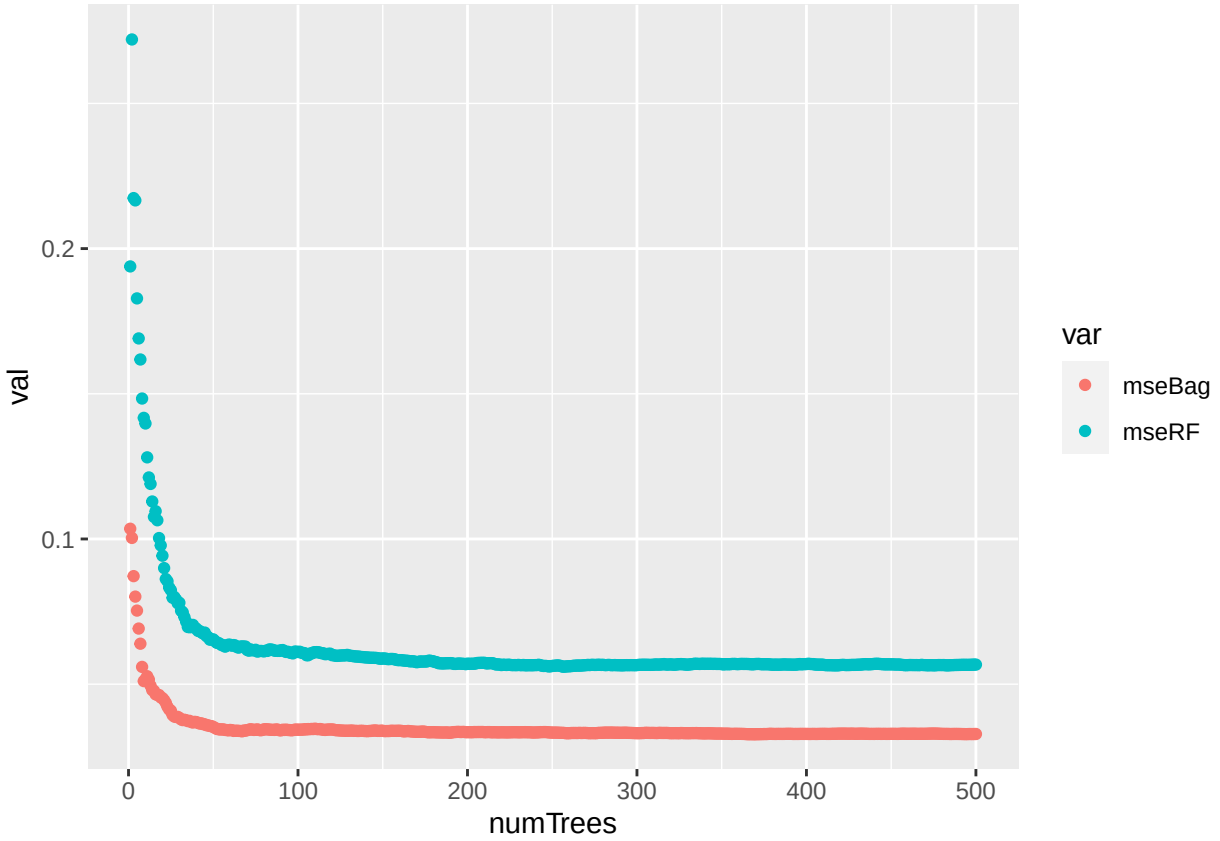
- Apparently, randomforest with 1/3 predictors was just okay with mse of 0.059.

Let's try to visualize how randomforest and bagging perform.

```
results.df <- data.frame(numTrees = 1:numTree,
                         mseBag = mod.bag.val$mse,
                         mseRF = mod.rf.val$mse)
```

We can produce a starter plot for this. We fix up the finer details later.

```
results.df %>%
  gather(var,val,mseBag:mseRF) %>%
  ggplot()+
  geom_point(aes(numTrees,val,color=var))
```



Bagging is doing better thatn RandomForest.

Boosting

This is going to be our last model that we will use for predicting Value. Build some smaller data sets to work with.

```
samp <- sample(1:N,N/3,rep=F)
data.v.df <- value.l.df[samp,]
n <- nrow(data.v.df)
train <- sample(1:n,n/2,rep=F)
train.df <- data.v.df[train,]
test.df <- data.v.df[-train,]
```

Making a very shallow tree, which we will later use for cv.

```
numTrees <- 2000
theShrinkage <- 0.1
theDepth <- 2
mod.gbm.val <- gbm(lvalue~.,
                   data=train.df,
                   distribution="gaussian", ## for regression
                   n.trees=numTrees,
                   shrinkage=theShrinkage,
                   interaction.depth = theDepth)
```

Now let's make predictions and compute the MSE on train and test data. Like a random forest, you can specify how many learners to use (up to the total from the gbm model).

```
pred.bval <- predict(mod.gbm.val,
                     n.trees=numTrees, ## use them all
                     newdata=train.df)
train.df$pred.gbm <- pred.bval
(with(train.df,mean((lvalue-pred.gbm)^2)))
```

```
## [1] 0.002075578
```

A good fit on the training data. Now the testing data.

```
pred.bval <- predict(mod.gbm.val,
                     n.trees=numTrees, ## use them all
                     newdata=test.df)

test.df$pred.gbm <- pred.bval
(mse.gbm.val <- with(test.df,mean((lvalue-pred.gbm)^2)))
```

```
## [1] 0.006660317
```

- Wow, Boosting has the smallest mse of 0.007 so far.

Let's perform crossvalidation with 5 folds to get the best numTree

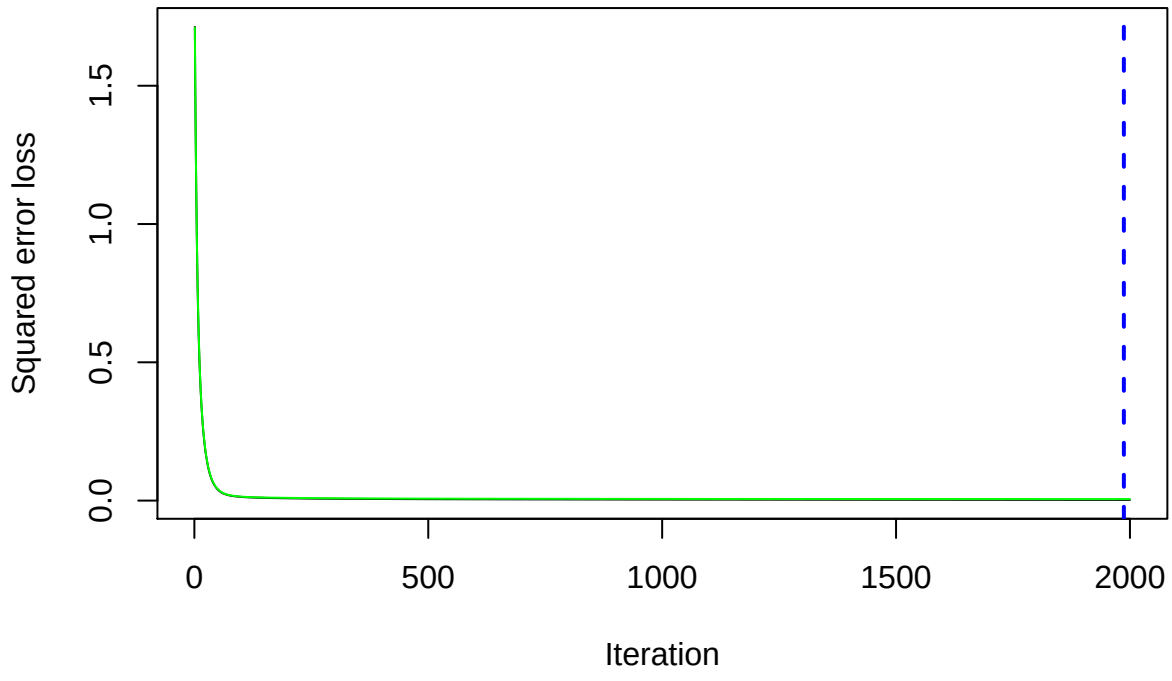
Since it takes a lot of time to run all 17000 observation, we will jsut take a smaple of around 5000 for our purpose.

```
N <- nrow(value.l.df)
samp <- sample(1:N,N/3,rep=F)
data.v.df <- value.l.df[samp,]
```

```
mod.gbm.cv <- gbm(lvalue~.,
                  data = data.v.df,
                  distribution="gaussian", ## for regression
                  n.trees=numTrees,
                  shrinkage=theShrinkage,
                  interaction.depth = theDepth,
                  cv.folds = 5,
                  n.minobsinnode=10,
                  n.cores = 5)  ## <-- use the cores on  your computer
```

Here's what we see.

```
gbm.best <- gbm.perf(mod.gbm.cv,method="cv")
```




```
#We can extract the optimal number of trees.
numTreesOpt <- gbm.best
```

Let's rebuild with the optimal number of trees.

```
mod.gbm.ot.val <- gbm(lvalue  ~ .,
                data=train.df,
                distribution="gaussian", ## for regression
                n.trees=numTreesOpt,
                shrinkage=theShrinkage,
                interaction.depth = theDepth
                )
```

And the predictions.

```
pred.gbm.val <- predict(mod.gbm.ot.val,
                     newdata=test.df,
                     shrinkage = theShrinkage,
                n.trees=numTreesOpt,
                interaction.depth = theDepth)
(mse.gbm.ot.val <- with(test.df,mean((lvalue-pred.gbm.val)^2)))
```

```
## [1] 0.008175525
```

Now let's cross-validate to get a better estimate of other parameters like shrinkage and tree depth.

```
n<- nrow(value.l.df)
samp <- sample(1:n,n/3,rep=F)
data.v.df <- value.l.df[samp,]
N<- nrow(data.v.df)
cvGBM.val <- function(theShrinkage,theDepth,numTrees,numFolds=10){
  print(sprintf("Shrinkage: %s   Depth: %s: numTrees: %s",theShrinkage,theDepth,numTrees))
  folds <-  sample(1:numFolds,N,rep=T)
  mse <- numeric(numFolds)
  fold <- 1
  for(fold in 1:numFolds){
    train.df <- data.v.df[folds != fold,]
    test.df <- data.v.df[folds == fold,]
    mod.gbm <- gbm(lvalue  ~  .,
                   data=train.df,
                   distribution="gaussian",
                   shrinkage=theShrinkage,
                   n.trees=numTrees,
                   interaction.depth = theDepth)
    pred.gbm <- predict(mod.gbm,
                        newdata=test.df,
                        shrinkage=theShrinkage,
                        n.trees=numTrees,
                        interaction.depth = theDepth)
    mse[fold] <- with(test.df,mean((lvalue - pred.gbm)^2))
  }
  mean(mse)
}
```

Testing to see if it works.

```
suppressWarnings(cvGBM.val(theShrinkage,theDepth,numTreesOpt,numFolds=5))
```

```
## [1] "Shrinkage: 0.1   Depth: 2: numTrees: 1987"
```

```
## [1] 0.005580478
```

Now we are ready for a grid. Let's create a grid of values.

```
shrinks <- c(0.5,.1)
depths <- c(1,2,3)
cv.vals <- expand.grid(shrinks,depths)
cv.vals
```

```
##   Var1 Var2
## 1  0.5    1
## 2  0.1    1
## 3  0.5    2
## 4  0.1    2
## 5  0.5    3
## 6  0.1    3
```

Run the cvGBM against all of these. There are 12 total so it will take few minutes.

```
suppressWarnings(err.vals.Value <- apply(cv.vals,1,function(row) cvGBM.val(row[1],row[2],numTreesOpt)))
```

```
## [1] "Shrinkage: 0.5   Depth: 1: numTrees: 1987"
## [1] "Shrinkage: 0.1   Depth: 1: numTrees: 1987"
## [1] "Shrinkage: 0.5   Depth: 2: numTrees: 1987"
## [1] "Shrinkage: 0.1   Depth: 2: numTrees: 1987"
## [1] "Shrinkage: 0.5   Depth: 3: numTrees: 1987"
## [1] "Shrinkage: 0.1   Depth: 3: numTrees: 1987"
```

Let's pick the best mse associated with optimal parameters.

```
id <- which.min(err.vals.Value)
(best.params <- cv.vals[id,])
```

```
##   Var1 Var2
## 6  0.1    3
```

```
(mse.gbm.opm.val <- err.vals.Value[id])
```

```
## [1] 0.004942624
```

```
c(mse.lasso.val, mse.ridge.val, mse.knn.val, mse.forward.val, mse.bag.val, mse.rf.val, mse.gbm.opm.val)
```

```
## [1] 0.029410258 0.029173892 0.066456505 0.058487334 0.032744020 0.056019853
## [7] 0.004942624
```

Clearly, Boosting is the best predictive model with mse of just 0.0056. Bagging and penalized regression (both lasso and ridge) were also very good with low mse of 0.030.

Conclusion for Value prediction (numeric):

- Boosting is clearly the best prediction model for predicting Value of fifa 18 football players.
- Bagging and penalized regression (Ridge and Lasso) also did a decent job in our case with mse around 0.30.
- Attributes like Overall, Age, Finishing, Volleys, GK seem to be some of the best predictors of Value, which are frequently seem amongst different models.

Part (B): Value Classification

In this part we will categorizing the Value into 4 catagories: Very High, Medium High, Medium, Low. We will then try to predict the catagorical values based on different algorithms.

First, let's catagorise the Value into 4 different catagories. Players with more than 5 million market value are catagorised as Very High, between 1.5 to 5 million are Medium High, between 0.5 to 1.5 millioin are Medium, and below 0.5 million euro are Low value.

```
catval.df <- value.df%>%
  mutate(CataVal =ifelse(Value>5,'V.High',ifelse(between(Value,1.5, 5),'Med.High',ifelse(between(Value,
  dplyr::select(-Value)
(with(catval.df,table(CataVal)))
```

```
## CataVal
##      low Med.High   Medium   V.High
##     6569     3166     5806     2184
```

Based on our catagorisation, there are larger numner of Low Value players (6569), and small number of Very High Value players (2184).

Penalized Regression

We will begin with rigde regression on this classification problem.

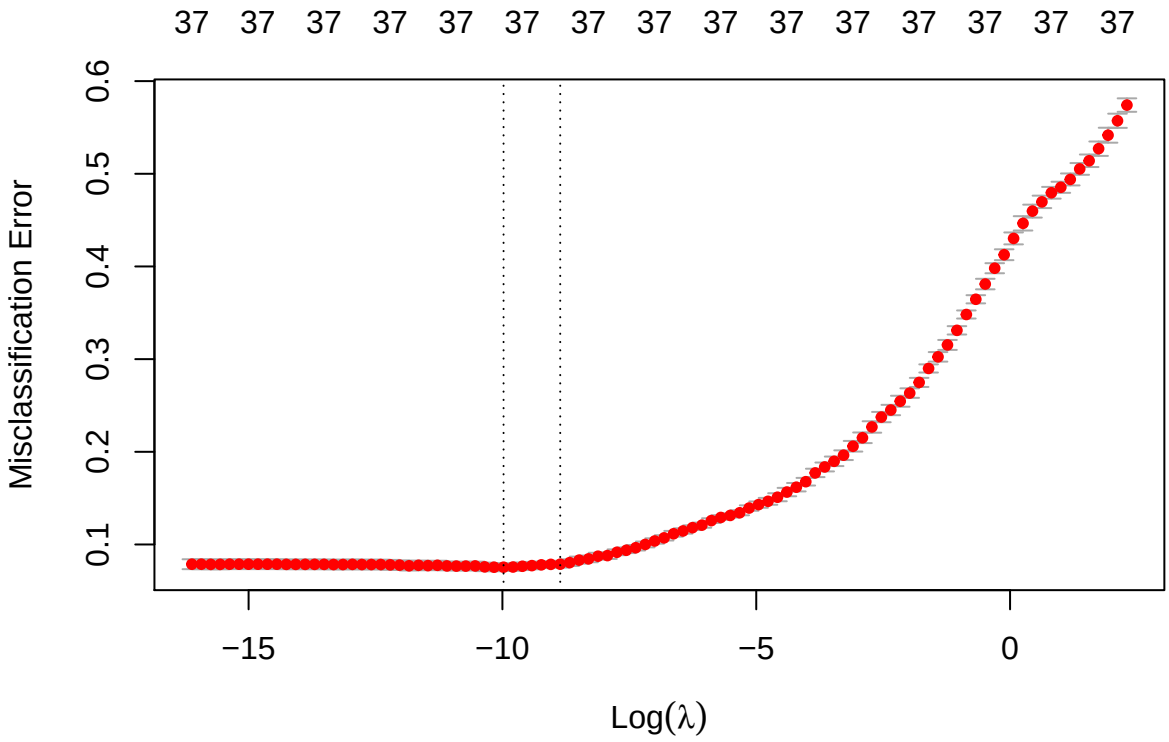
Let's build the dataset.

```
numPreds <- ncol(catval.df)-1
n<-nrow(catval.df)
samp <- sample(1:n, n/3, rep=F)
cata.x <- data.matrix(catval.df[samp,1:numPreds])
cata.y <- data.matrix(catval.df[samp,-(1:numPreds)])
lambda.grid <- seqlambda.grid <- 10^seq(-7,1,length=100)
```

We will first run the crossvalidation on multinomial ridge regresison to get the optimal lambda value.

```
ridge.cv.cval <- cv.glmnet(cata.x,cata.y,
                      lambda=lambda.grid,
                      family="multinomial",
                      type.measure="class",
                      alpha=0)

plot(ridge.cv.cval)
```



Let's find out the optimal lambda within 1se and identify the associated error.

```
lambda.opt <- ridge.cv.cval$lambda.1se
id <- with(ridge.cv.cval,which(lambda==lambda.opt))
(err.ridge.cval <- with(ridge.cv.cval,cvm[id]))
```

```
## [1] 0.07870684
```

Let's use optimal lambda for our model, predict, and compare the results with actual values using confusion matrix.

```
mod <- glmnet(cata.x,cata.y,
                      lambda=lambda.opt,
                      family="multinomial",
                      alpha=0)
pred = predict(mod, newx = cata.x, s=lambda.opt, type="class")
(err.ridg.optmod<-mean(pred != cata.y))
```

```
## [1] 0.07109005
```

```
#Confusion Matrix and err
(table(pred, cata.y))
```

```
##           cata.y
## pred        low Med.High Medium V.High
```

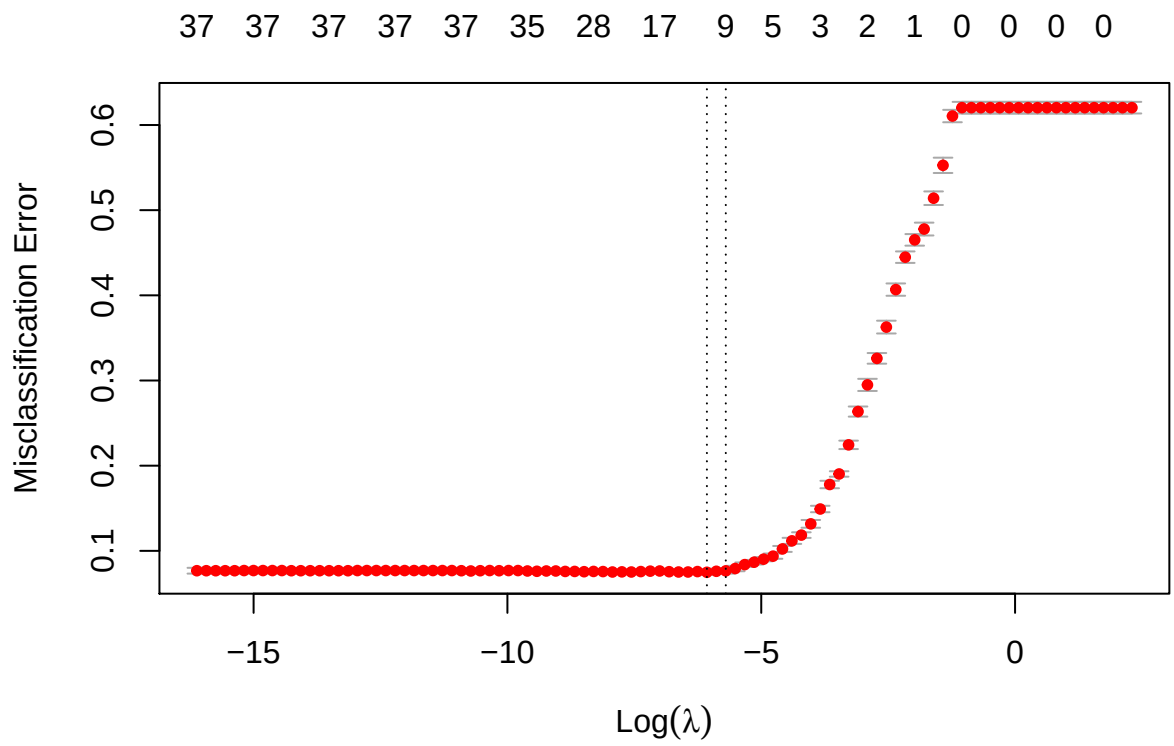
```
##    low       2134        0     96      0
##    Med.High     3      990     57     56
##    Medium     106       67   1727      0
##    V.High       0       35      0    637
```

- A 7.3% error rate in not that bad. How about in case of lasso? Let's find out.

Lasso regression

We will repeat a similar process with lasso.

```
lasso.cv.cval <- cv.glmnet(cata.x,cata.y,
                       lambda=lambda.grid,
                       family="multinomial",
                       type.measure="class",
                       alpha=1)
plot(lasso.cv.cval)
```




```
lambda.opt <- lasso.cv.cval$lambda.1se
id <- with(lasso.cv.cval,which(lambda==lambda.opt))
(err.lasso.cval <- with(lasso.cv.cval,cvm[id]))
```

```
## [1] 0.07650643
```

```
c(err.ridge.cval, err.lasso.cval)
```

```
## [1] 0.07870684 0.07650643
```

Both seem to have similar performaces with error around 7.4%.

Decision Tree

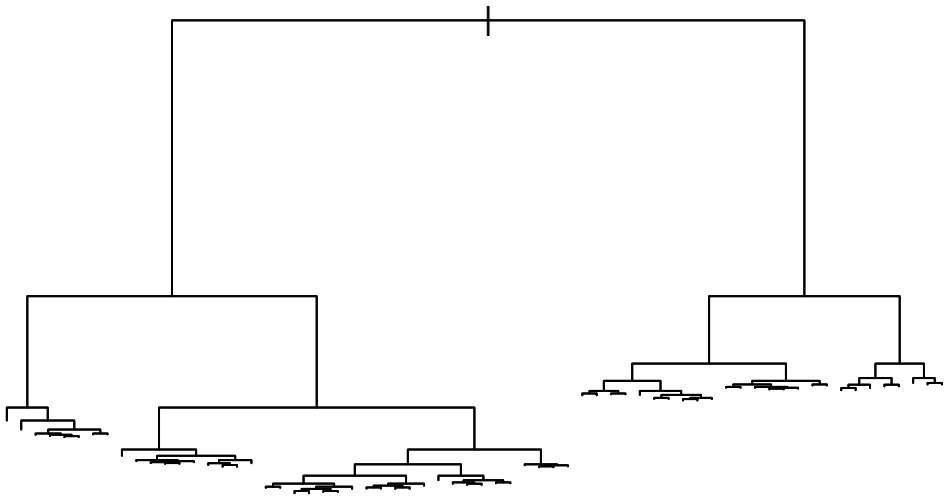
We will use decision tree to predict our catagorical Value. First, let's make some training and testin sets.

```
catval.df$CataVal <- as.factor(catval.df$CataVal)
N <- nrow(catval.df)
train <- sample(1:N,N/2,rep=F)
train.df <- catval.df[train,]
test.df <-  catval.df[-train,]
```

Building a deep tree model.

```
mod.tree.cval <- tree(CataVal ~ .,
                 data=train.df,
                 control=tree.control(nrow(train.df),
                                      mindev=0.001,
                                      minsize = 10))
```

```
{plot(mod.tree.cval)}
```

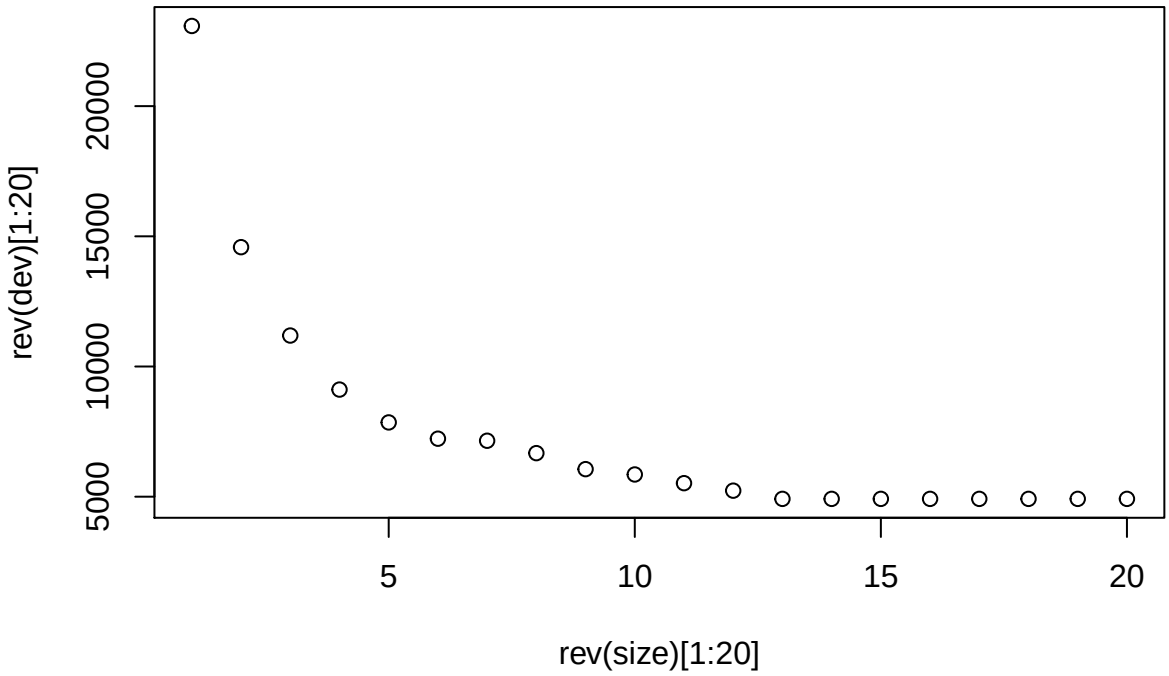


Cross-validating on the deep tree.

```
mod.cv <- cv.tree(mod.tree.cval)
```

Take a closer look at the start of this plot.

```
with(mod.cv,plot(rev(size)[1:20],rev(dev)[1:20]))
```



Looks like something around 12-14 will work

```
mod.prune <- prune.misclass(mod.tree.cval,best=13)
preds <- predict(mod.tree.cval,newdata=test.df,type="class")
#confusion matrix
with(test.df,table(CataVal, preds))
```

```
##           preds
## CataVal     low Med.High Medium V.High
##   low      3207        0    122      0
##   Med.High    1     1488     46     34
##   Medium     63       41   2779      0
##   V.High      0       63      0   1019
```

```
#error
(err.tree.cval <- with(test.df,mean(CataVal != preds)))
```

```
## [1] 0.04174659
```

Looks like a 4.94% error rate, which is amazing.

```
c(err.ridge.cval, err.lasso.cval,err.tree.cval)
```

```
## [1] 0.07870684 0.07650643 0.04174659
```

- Compared to penalized regressions, seem to do better with 4.95% error.

Bagging and Random Forests

While we are hear, let's predict using bagging and random forests.

```
## Regression: CataVal as response
numPreds <- ncol(catval.df)-1
numTree <- 250
mod.bag.cval <- randomForest(CataVal ~ .,
                         data=catval.df,
                         mtry=numPreds,
                         ntree=numTree)
```

Note that since we are bootstrapping, we get an Out of Bag estimate of the error rate.

```
mod.bag.cval
```

```
##
## Call:
##  randomForest(formula = CataVal ~ ., data = catval.df, mtry = numPreds,      ntree = numTree)
##                Type of random forest: classification
##                      Number of trees: 250
## No. of variables tried at each split: 37
##
##         OOB estimate of  error rate: 2.05%
## Confusion matrix:
##           low Med.High Medium V.High class.error
## low      6471        0     98      0  0.01491856
## Med.High    0     3083     51     32  0.02621605
## Medium     84       52   5670      0  0.02342404
## V.High      0       46      0   2138  0.02106227
```

```
(err.bag.cval <- with(mod.bag.cval,err.rate[numTree]))
```

```
## [1] 0.02047955
```

- Wow, 2% error rate is very good compared to other methods.

RandomForest

Running the same model with 1/3 predictors.

```
mod.rf.cval <- randomForest(CataVal ~ .,
                        data=catval.df,
                        mtry=numPreds/3,
                        ntree=numTree)
```

Finding the error rate.

```
mod.rf.cval
```

```
##
## Call:
##  randomForest(formula = CataVal ~ ., data = catval.df, mtry = numPreds/3,      ntree = numTree)
##                Type of random forest: classification
##                      Number of trees: 250
## No. of variables tried at each split: 12
##
##         OOB estimate of  error rate: 2.27%
## Confusion matrix:
##           low Med.High Medium V.High class.error
## low      6454        0    115      0  0.01750647
## Med.High    0     3071     63     32  0.03000632
## Medium     77       55   5673      1  0.02290734
## V.High      0       60      0   2124  0.02747253
```

```
(err.rf.cval <- with(mod.rf.cval,err.rate[numTree]))
```

```
## [1] 0.02273625
```

Bagging and Random Forest are pretty competitive with each other with error of around 2.2% They both easily outperform the other methods.

```
c(err.ridge.cval, err.lasso.cval, err.tree.cval, err.bag.cval,err.rf.cval)
```

```
## [1] 0.07870684 0.07650643 0.04174659 0.02047955 0.02273625
```

Conclusion:

- Bagging and randomforest model performed great at predicting the catagorical Value with an error rate of 2.2%.

Part (C): Wage prediction for Football (soccer) players (numerical)

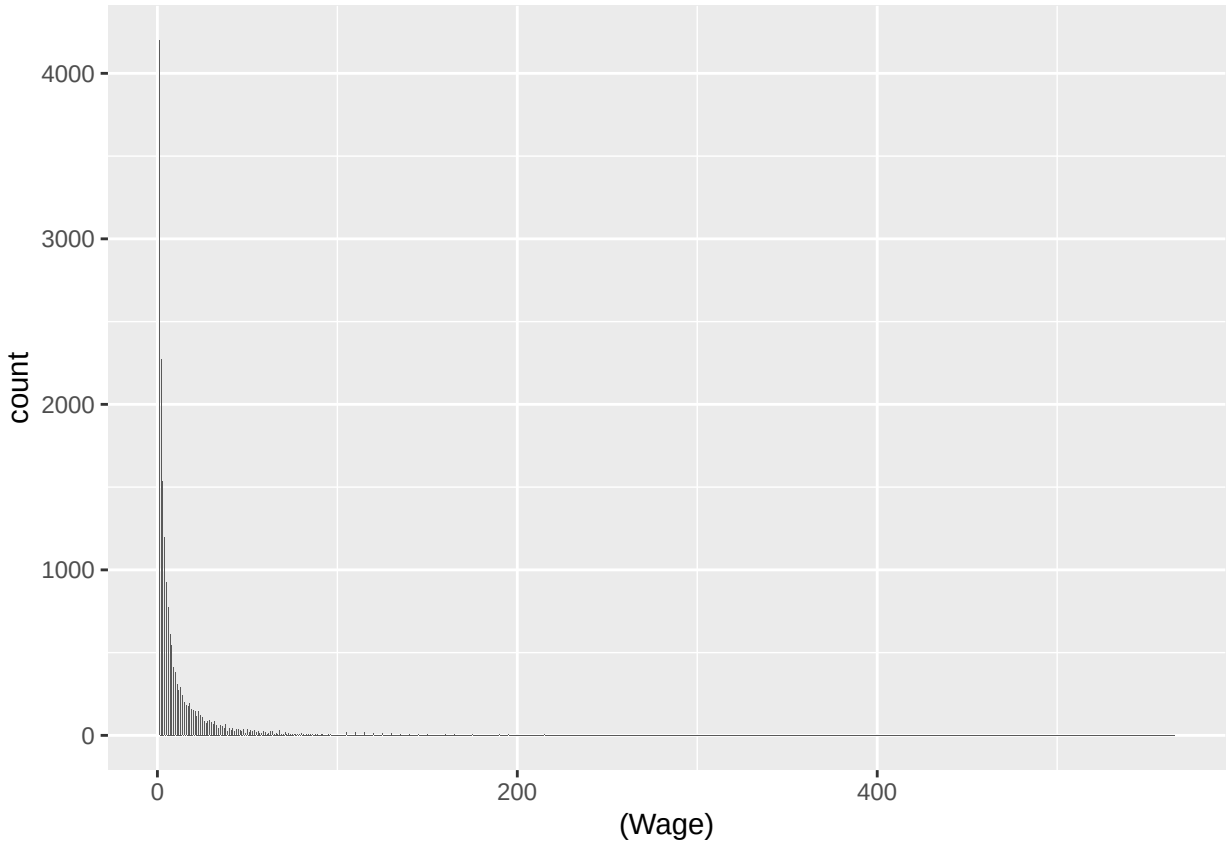
In this section, we plan to build models that accurately predict wages of football players based on their footballing skills and other attributes like age, strength, etc. Since, it might not be very efficient to use all the algorithms model like in previous case, we will only hand pick and implement few best models.

Building a wage.df dataset.

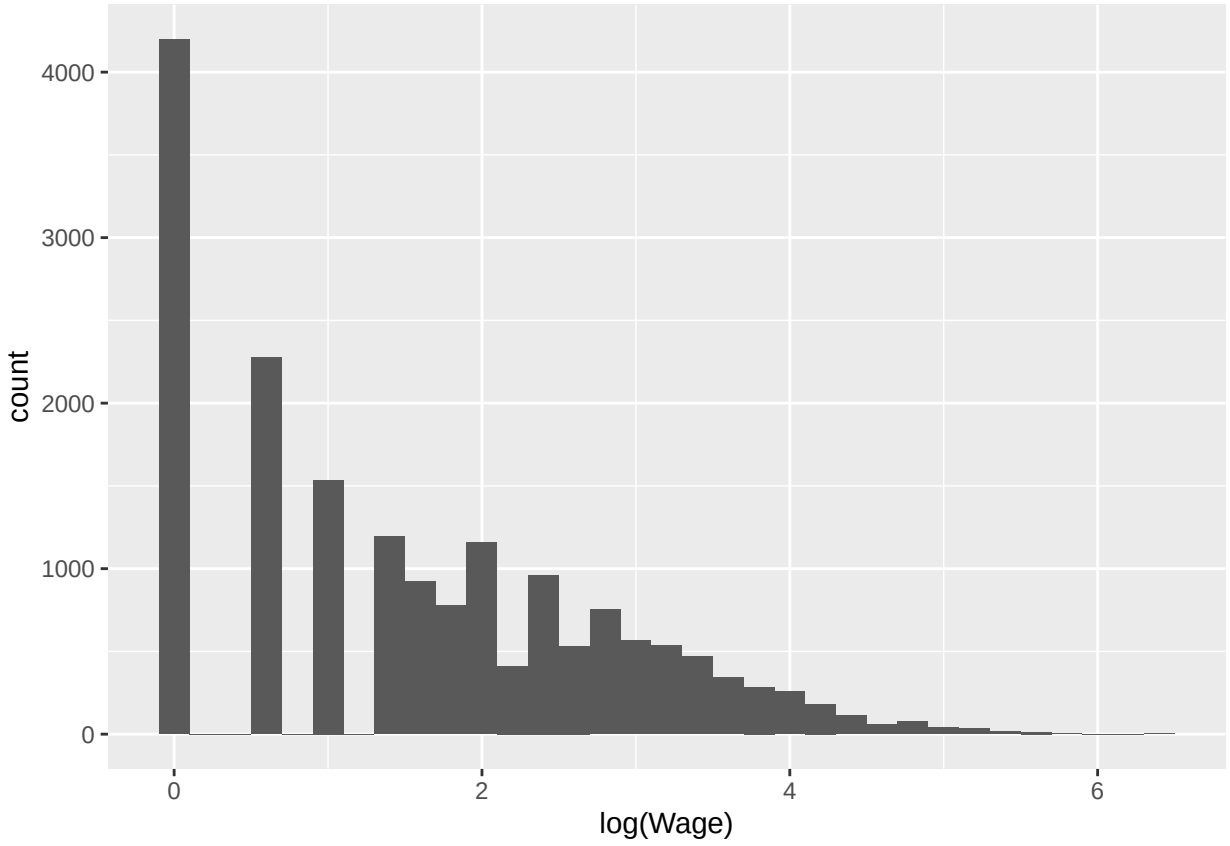
```
wage <- foot2.df%>%dplyr::select(Wage)
wage.df <- cbind(stats.df, wage)
```

Let's see the distribution of wage among players.

```
ggplot(wage.df, aes((Wage)))+geom_histogram(binwidth = .5)
```




```
ggplot(wage.df, aes(log(Wage)))+geom_histogram(binwidth = .2) # data is very right skewed
```




```
length(which(wage.df$Wage<5)) # no.players with salaray less than 5k
```

```
## [1] 9205
```

Wow, wage seems to be extremely right skewed with some player earning way higher than rest of the players. Majority of the players in real life play in smaller leagues which explains why Fifa dataset has such skewed data. Handful of the top league players who play in English premier league and Laliga have very high salary due to thier demand around the world.
We plan to log the wage in order to normalize the wage distribution to some extent.

```
#logging wage
wage.df <-wage.df%>%mutate(lwage=log(Wage))%>%dplyr::select(-Wage)
wagel.df <- scale(as.matrix(wage.df))
wage.x <- wagel.df[,1:ncol(wagel.df)-1]
wage.y <- wagel.df[,ncol(wagel.df)]
```

Lasso penalized regression

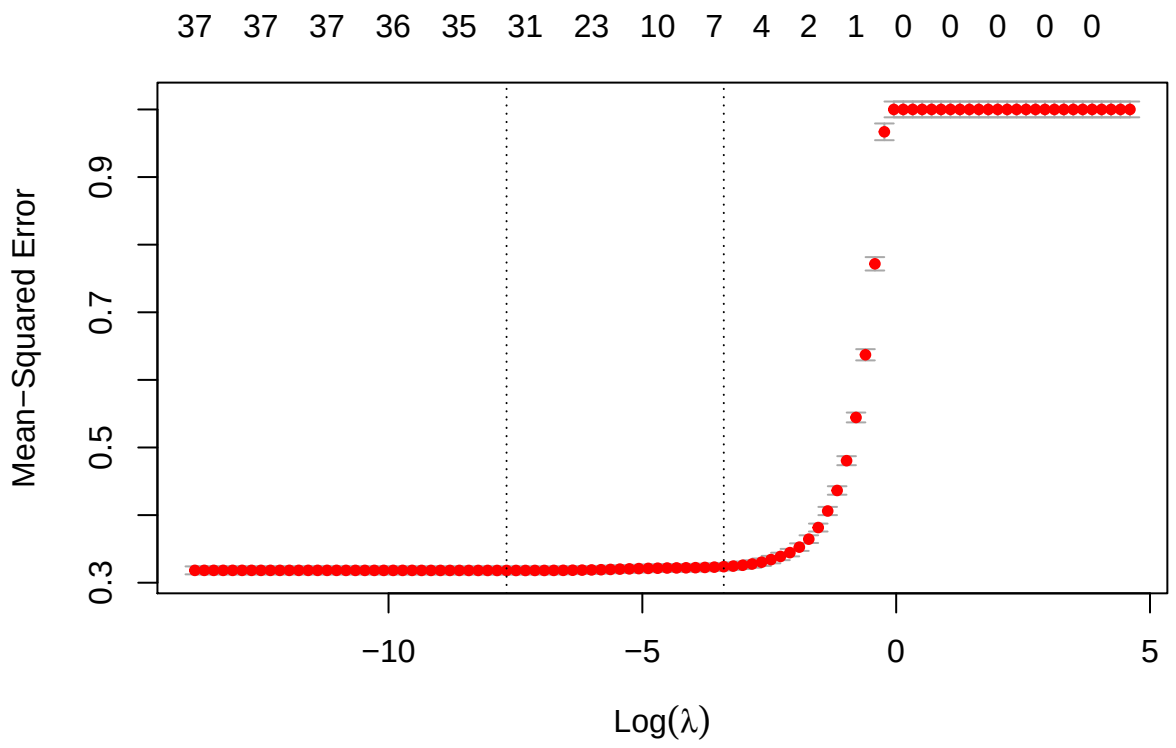
Let's start by fitting lasso regression. We need a grid of lambda values.

```
lambda.grid <- 10^seq(-6,2,length=100)
```

Now we can run cross-validated lasso, select lambda, and look at the results.

```
mod.lasso.cv.wage <- cv.glmnet(wage.x,
                           wage.y,
                           alpha=1,
                           lambda=lambda.grid)

plot(mod.lasso.cv.wage)
```




```
(mse.lasso.wage <-with(mod.lasso.cv.wage,cvm[lambda==lambda.1se])) # mse
```

```
## [1] 0.3237012
```

```
(lambda.opt <- mod.lasso.cv.wage$lambda.1se) # optimal lamnda val within 1 se
```

```
## [1] 0.03351603
```

```
mod.lasso.opt <- glmnet(wage.x, wage.y, alpha=1, lambda=lambda.opt)
```

We got a mse of 0.3215 for lasso regression. Let's see what are the non zero coefficients in our lasso model.

```
(coefs.opt <- data.matrix(coefficients(mod.lasso.opt)))
```

```
##                                s0
```

```
## (Intercept)        -5.959784e-14
## Age                 0.000000e+00
## Overall             6.716712e-01
## Potential           1.322101e-01
## Acceleration        0.000000e+00
## Aggression          0.000000e+00
## Agility             0.000000e+00
## Balance             0.000000e+00
## Ball.control        0.000000e+00
## Composure           0.000000e+00
## Crossing            0.000000e+00
## Curve               0.000000e+00
## Dribbling           0.000000e+00
## Finishing           2.473793e-03
## Free.kick.accuracy  0.000000e+00
## GK.diving           0.000000e+00
## GK.handling         0.000000e+00
## GK.kicking          0.000000e+00
## GK.positioning      0.000000e+00
## GK.reflexes         0.000000e+00
## Heading.accuracy    3.154993e-04
## Interceptions       0.000000e+00
## Jumping             0.000000e+00
## Long.passing        0.000000e+00
## Long.shots          0.000000e+00
## Marking             0.000000e+00
## Penalties           2.675806e-02
## Positioning         5.789612e-03
## Reactions           0.000000e+00
## Short.passing       0.000000e+00
## Shot.power          0.000000e+00
## Sliding.tackle      0.000000e+00
## Sprint.speed        0.000000e+00
## Stamina             0.000000e+00
## Standing.tackle     0.000000e+00
## Strength            0.000000e+00
## Vision              0.000000e+00
## Volleys             1.550130e-02
```

```
coefs.lasso <- coefs.opt[-1] ##drop constant term
(lassoPreds <- which(coefs.lasso != 0))
```

```
## [1]  2  3 13 20 26 27 37
```

```
(w.lasso.var <- names(wage.df)[lassoPreds]) # Non zero predictors
```

```
## [1] "Overall"          "Potential"        "Finishing"        "Heading.accuracy"
## [5] "Penalties"        "Positioning"      "Volleys"
```

These variables make sense. A player with higher potential, acceleration, finishing, positioning, and penalty takes would have higher wage.

KNN regression.

Let's try KNN and see what happens. Building datasets for knn.

```
n <- nrow(wagel.df)
samp <- sample(1:n, n/3)
wage.x <- wagel.df[samp,1:ncol(wagel.df)-1]
wage.y <- wagel.df[samp,ncol(wagel.df)]
```

Let's do some cros-validate

```
knnMSE_SE.wage <- function(kVal,numFolds=10){
  numFolds <- 10
  N <- nrow(wage.x)
  folds <- sample(1:numFolds,N,rep=T)
  errs <- numeric(numFolds)

  for(fold in 1:numFolds){
    train.x <- wage.x[folds != fold,]
    train.y <- wage.y[folds != fold]
    test.x <- wage.x[folds == fold,]
    test.y <-  wage.y[folds == fold]
    knn.mod <- knn.reg(train.x,test.x,train.y,k=kVal)
    errs[fold] <- with(knn.mod,mean((pred-test.y)^2))
  }
c(mean(errs),sd(errs))
}
```

Testing the function.

```
knnMSE_SE.wage(15)
```

```
## [1] 0.3252592 0.0285563
```

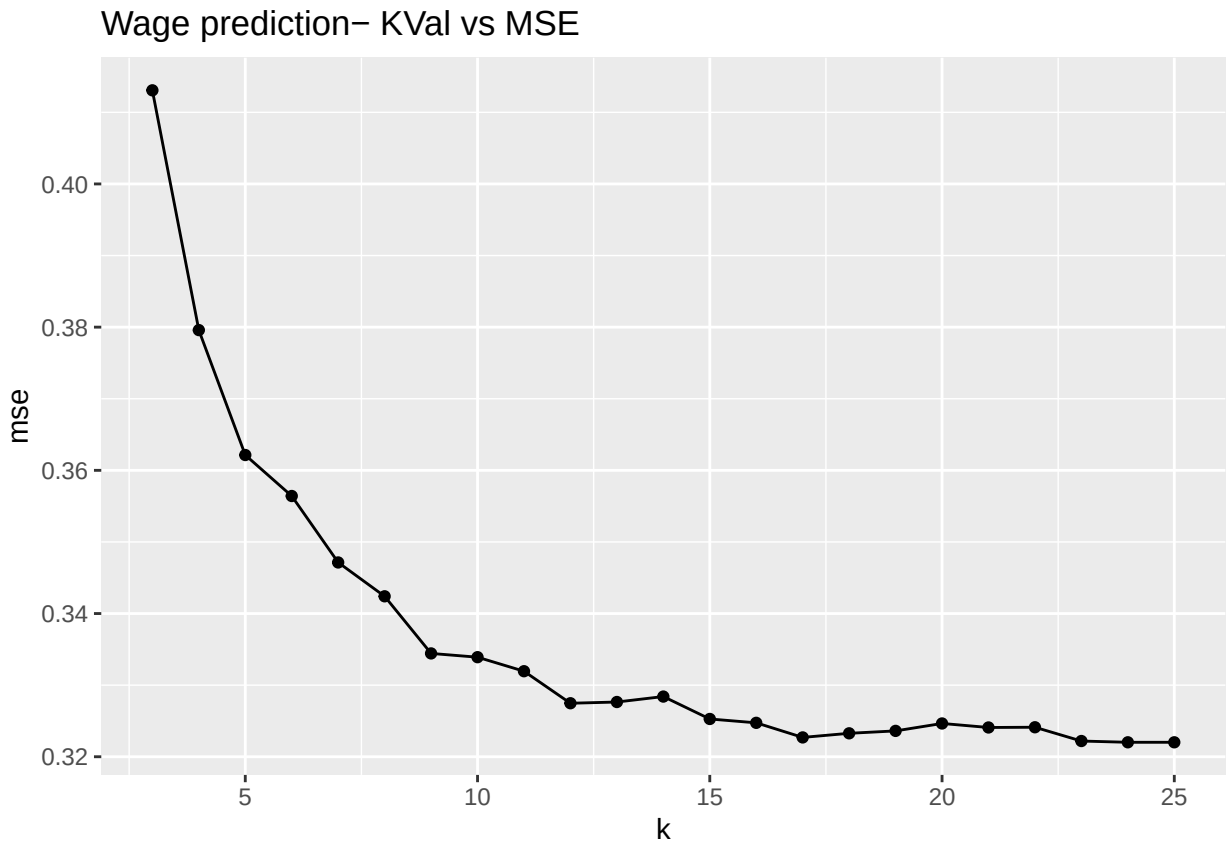
```
# 0.3151299 0.0161007
```

Now, let's fit in range of k val and plot it with mse to pick the best kval.

```
knnMSE <- function(kVal,numFolds=10){ knnMSE_SE.wage(kVal,numFolds)[1]}

maxK <- 25
allMSE.wage <- map_dbl(3:maxK,knnMSE)

data.frame(k=3:maxK,mse=allMSE.wage) %>%
  ggplot(aes(k,mse)) +
  geom_point()+
  geom_line()+
  labs(title="Wage prediction- KVal vs MSE")
```



Looks like the best value is around 15

```
id <- which.min(allMSE.wage)
(kMin <- (3:maxK)[id])
```

```
## [1] 24
```

```
(mseInfo <- knnMSE_SE.wage(kMin))
```

```
## [1] 0.32102357 0.01903571
```

```
## very large SE
mseCut <- mseInfo[1]+mseInfo[2]
##
(theVals <- (3:maxK)[allMSE.wage < mseCut])
```

```
##  [1]  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

According this, we will use kVal=12

```
(k1SE <- min(theVals))
```

```
## [1] 9
```

```
allMSE.wage[k1SE]
```

```
## [1] 0.3319449
```

```
mse <- knnMSE_SE.wage(k1SE)
(mse.knn.wage <- mse[1])
```

```
## [1] 0.3359333
```

Let's compare the mse valued for models so far.

```
c(mse.lasso.wage, mse.knn.wage)
```

```
## [1] 0.3237012 0.3359333
```

So far, all the models seem to be consistent with knn having the lowest mse of 0.319.

Forward Selection

Setting up some vectors to keep track of our models as we build them.

```
numPreds <- ncol(wage.df)-1
availPreds <- 1:numPreds
modelPreds.for.w <- c()
```

The main loop for Forward selection.

```
## keep track of the R2 for reference
maxR2 <- c()
##Keep going as long as there are available predictors left
while(length(availPreds) > 0){
##add predictor which increases R^2 the most
##keep track of the R^2 values for reference

  allR2 <- c()
  for(id in availPreds){
    ##the augmented predictors
    augPreds <- c(modelPreds.for.w,id)
    ## Build the data frame with the augmented predictors
    data.df <- wage.df[,c(augPreds,numPreds+1)]
    ##the model and its summary
    mod.curr <- lm(lwage ~ ., data=data.df)
    mod.sum <- summary(mod.curr)
    ##grab the R^2
    allR2 <- c(allR2,mod.sum$r.squared)
  }
  ##Find the index of the min R^2
  max.id <- which.max(allR2)
  ##get the best predictor and R^2
  bestPred <- availPreds[max.id]
```

```
  bestR2 <- max(allR2)
  ##Add these into the collection
  modelPreds.for.w <- c(modelPreds.for.w,bestPred)
  ## remove the bestPred from the availPreds
  availPreds <- setdiff(availPreds,bestPred)
  maxR2 <- c(maxR2,bestR2)
  ##remove bestsPred from avail
  ## Print stuff out for debugging and attention-grabbing
  print(sprintf("Pred Added: %s R^2 Value: %s",bestPred,round(bestR2,3)))
  ##print(modelPreds)
}
```

```
## [1] "Pred Added: 2 R^2 Value: 0.661"
## [1] "Pred Added: 3 R^2 Value: 0.674"
## [1] "Pred Added: 26 R^2 Value: 0.678"
## [1] "Pred Added: 37 R^2 Value: 0.678"
## [1] "Pred Added: 4 R^2 Value: 0.679"
## [1] "Pred Added: 12 R^2 Value: 0.679"
## [1] "Pred Added: 8 R^2 Value: 0.68"
## [1] "Pred Added: 20 R^2 Value: 0.68"
## [1] "Pred Added: 15 R^2 Value: 0.681"
## [1] "Pred Added: 10 R^2 Value: 0.681"
## [1] "Pred Added: 1 R^2 Value: 0.681"
## [1] "Pred Added: 33 R^2 Value: 0.681"
## [1] "Pred Added: 22 R^2 Value: 0.681"
## [1] "Pred Added: 31 R^2 Value: 0.681"
## [1] "Pred Added: 25 R^2 Value: 0.682"
## [1] "Pred Added: 27 R^2 Value: 0.682"
## [1] "Pred Added: 29 R^2 Value: 0.683"
## [1] "Pred Added: 23 R^2 Value: 0.683"
## [1] "Pred Added: 13 R^2 Value: 0.683"
## [1] "Pred Added: 14 R^2 Value: 0.683"
## [1] "Pred Added: 34 R^2 Value: 0.683"
## [1] "Pred Added: 32 R^2 Value: 0.683"
## [1] "Pred Added: 6 R^2 Value: 0.683"
## [1] "Pred Added: 11 R^2 Value: 0.683"
## [1] "Pred Added: 35 R^2 Value: 0.683"
## [1] "Pred Added: 21 R^2 Value: 0.683"
## [1] "Pred Added: 19 R^2 Value: 0.683"
## [1] "Pred Added: 36 R^2 Value: 0.683"
## [1] "Pred Added: 28 R^2 Value: 0.683"
## [1] "Pred Added: 17 R^2 Value: 0.683"
## [1] "Pred Added: 9 R^2 Value: 0.683"
## [1] "Pred Added: 24 R^2 Value: 0.683"
## [1] "Pred Added: 30 R^2 Value: 0.683"
## [1] "Pred Added: 5 R^2 Value: 0.683"
## [1] "Pred Added: 16 R^2 Value: 0.683"
## [1] "Pred Added: 7 R^2 Value: 0.683"
## [1] "Pred Added: 18 R^2 Value: 0.683"
```

A R^2 value of 0.68 is not that bad.

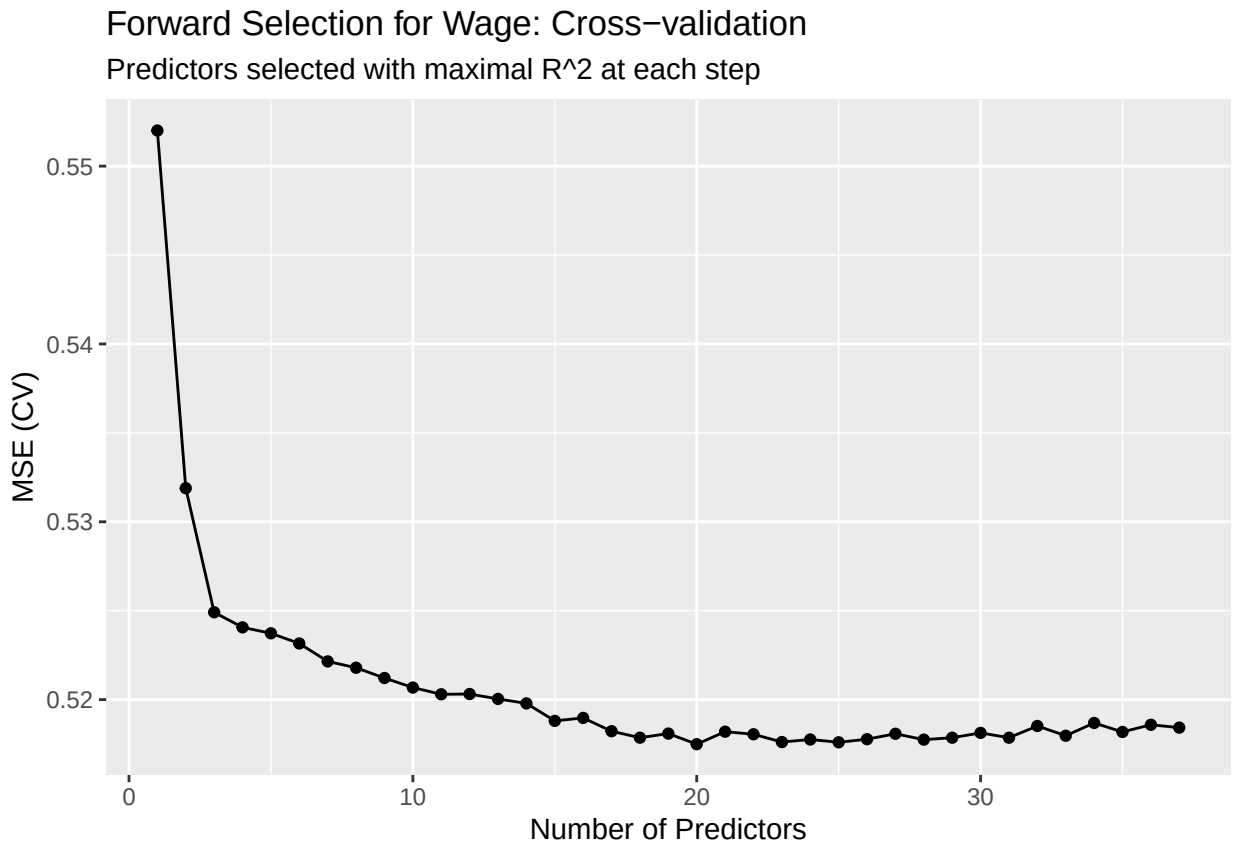
Function to calculate the mse and se based on 10 fold cross validation.

```
## args: a data frame and a number of folds (default to 10).
## ret: k-fold cross-validated MSE
mseCV_SE.w <- function(data.df,numFolds=10){
dataSize <- nrow(data.df)
folds <- sample(1:numFolds,dataSize,rep=T)
mse <- numeric(numFolds)
# fold <- numFolds
for(fold in 1:numFolds){
train.df <- data.df[folds != fold,]
test.df <- data.df[folds == fold,]
mod <- lm(lwage ~ .,data=train.df)
vals <- predict(mod,newdata=test.df)
mse[fold] <- with(test.df,mean((lwage-vals)^2))
}
c(mean(mse),sqrt(var(mse)/numFolds))}
```

We will now use cv to calculate the mse associate with forward model selection. We will use map_dbl to simplify the work. This means we need to build an "anonymous function" inside of map_dbl.

```
allMSE.w <- map_dbl(1:numPreds,
  function(totPred)
  mseCV_SE.w(wage.df[,c(modelPreds.for.w[1:totPred],numPreds+1)])[1])

data.frame(numPred=1:numPreds,
mse=allMSE.w) %>%
ggplot()+
geom_point(aes(numPred,mse))+
geom_line(aes(numPred,mse))+
labs(title="Forward Selection for Wage: Cross-validation",
  subtitle="Predictors selected with maximal R^2 at each step",
  x = "Number of Predictors",
  y = "MSE (CV)")
```




```
(predMin <- which.min(allMSE.w))
```

```
## [1] 20
```

```
# 26
# build this model
temp.df <- wage.df[,c(modelPreds.for.w[1:predMin],numPreds+1)]
# get both the MSE estimate and the SE
(mseInfo <- mseCV_SE.w(temp.df))
```

```
## [1] 0.517400765 0.008599289
```

```
# 0.517632465 0.003794512
```

Now we need to identify the smallest predictor set within one SE of the min MSE.

```
## add the MSE and the SE.
mseCut <- mseInfo[1]+mseInfo[2]
##
(thePreds.for.w <- (1:numPreds)[allMSE.w < mseCut])
```

```
##  [1]  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
## [26] 28 29 30 31 32 33 34 35 36 37
```

```
#8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
```

We can use the predictors starting at the minimum index.

```
(optNumPreds <- min(thePreds.for.w))
```

```
## [1] 3
```

```
## [1] 8
(preds.forward.w <- modelPreds.for.w[1:optNumPreds])
```

```
## [1]  2  3 26
```

```
# 2  3 26 37  4 12  8 20 15
```

The cross-validated estimated MSE has already been computed.

```
(mse.forward.wage <- allMSE.w[optNumPreds])
```

```
## [1] 0.5249096
```

```
##  0.5214088
numPred <- ncol(wage.df)-1
names.df <- names(wage.df)
(w.for.var <- names.df[preds.forward.w])
```

```
## [1] "Overall"   "Potential" "Penalties"
```

```
mod.f <-lm(lwage ~ ., data=wage.df[,c(preds.forward.w,numPred+1)])
summary(mod.f)
```

```
##
## Call:
## lm(formula = lwage ~ ., data = wage.df[, c(preds.forward.w, numPred +
##     1)])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.0204 -0.4673  0.0109  0.4991  2.4797
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -9.2711933  0.0646946 -143.31   <2e-16 ***
## Overall      0.1249670  0.0011006  113.55   <2e-16 ***
## Potential    0.0319539  0.0012202   26.19   <2e-16 ***
## Penalties    0.0057308  0.0003658   15.67   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7242 on 17721 degrees of freedom
## Multiple R-squared:  0.6781, Adjusted R-squared:  0.6781
## F-statistic: 1.245e+04 on 3 and 17721 DF,  p-value: < 2.2e-16
```

Comparing results.

```
c(mse.lasso.wage, mse.knn.wage, mse.forward.wage)
```

```
## [1] 0.3237012 0.3359333 0.5249096
```

Forward selection has the biggest mse so far. Lasso and KNN still perform better.

```
print(w.lasso.var)
```

```
## [1] "Overall"          "Potential"        "Finishing"        "Heading.accuracy"
## [5] "Penalties"        "Positioning"      "Volleys"
```

```
print( w.for.var)
```

```
## [1] "Overall"   "Potential" "Penalties"
```

Comparing the selection of important variables, we see Overall, Potential, and Penalties to be occuring across the models.

Bagging

Test/validation data.

Creating training and testing sets to train and validate our results.

```
N <- nrow(wage.df)
numPred <- ncol(wage.df)-1
n <- 1000
build <- sample(1:N,n,rep=F)
data.df <- wage.df[build,]
##Data to test/validate the model
n <- 1000
test <- sample(setdiff(1:N,build),n,rep=F)
test.df <- wage.df[test,]
```

Now we can buld a build a Bagged model with 200 tees using the R randomForest function.

```
numTree <- 200
mod.bag.wage <- randomForest(lwage ~ .,
                         data=data.df,
                         ntree=numTree,
                         mtry=numPred)  ## Use all the predictors
mod.bag.wage
```

```
##
## Call:
##  randomForest(formula = lwage ~ ., data = data.df, ntree = numTree,      mtry = numPred)
##                Type of random forest: regression
##                      Number of trees: 200
```

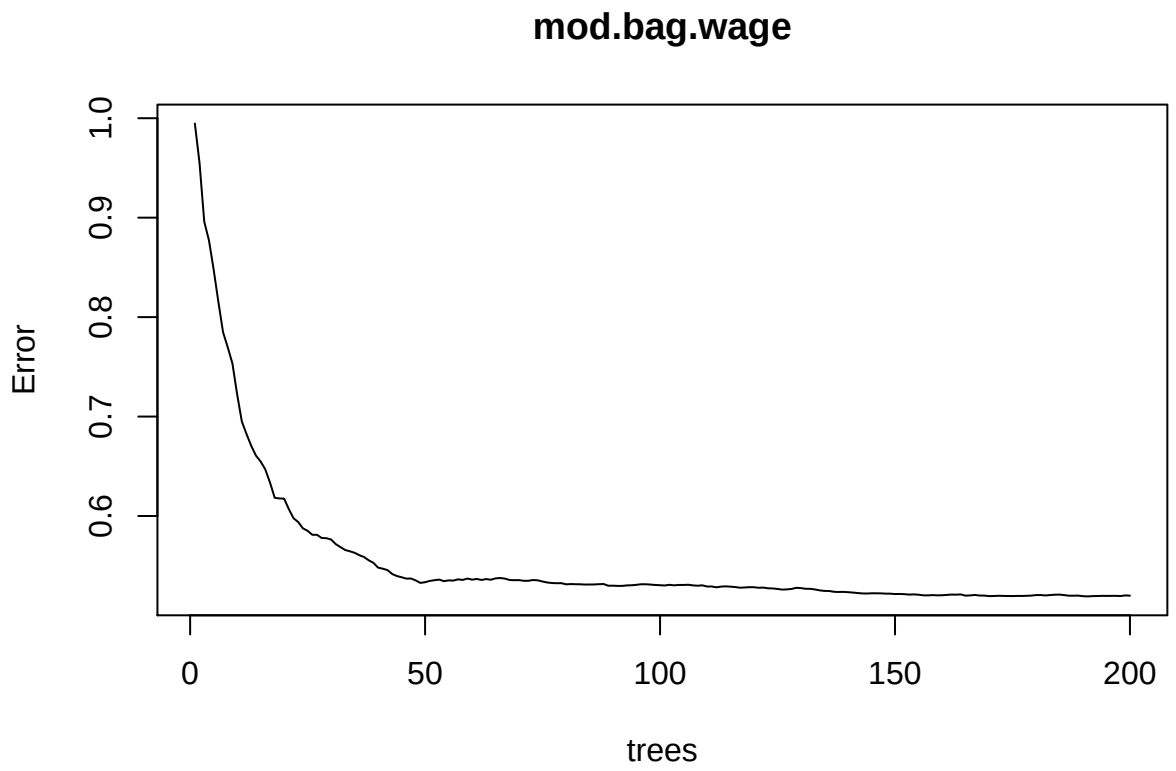
```
## No. of variables tried at each split: 37
##
##           Mean of squared residuals: 0.5199105
##                     % Var explained: 70.28
```

```
(mse.bag.wage <- min(mod.bag.wage$mse)) # mse
```

```
## [1] 0.5191662
```

The "Mean of squared residuals" is the Out of Bag estimate of the MSE (as opposed to the error rate)

```
plot(mod.bag.wage)
```



RandomForest

Similar idea but we will use 1/3 of predictors.

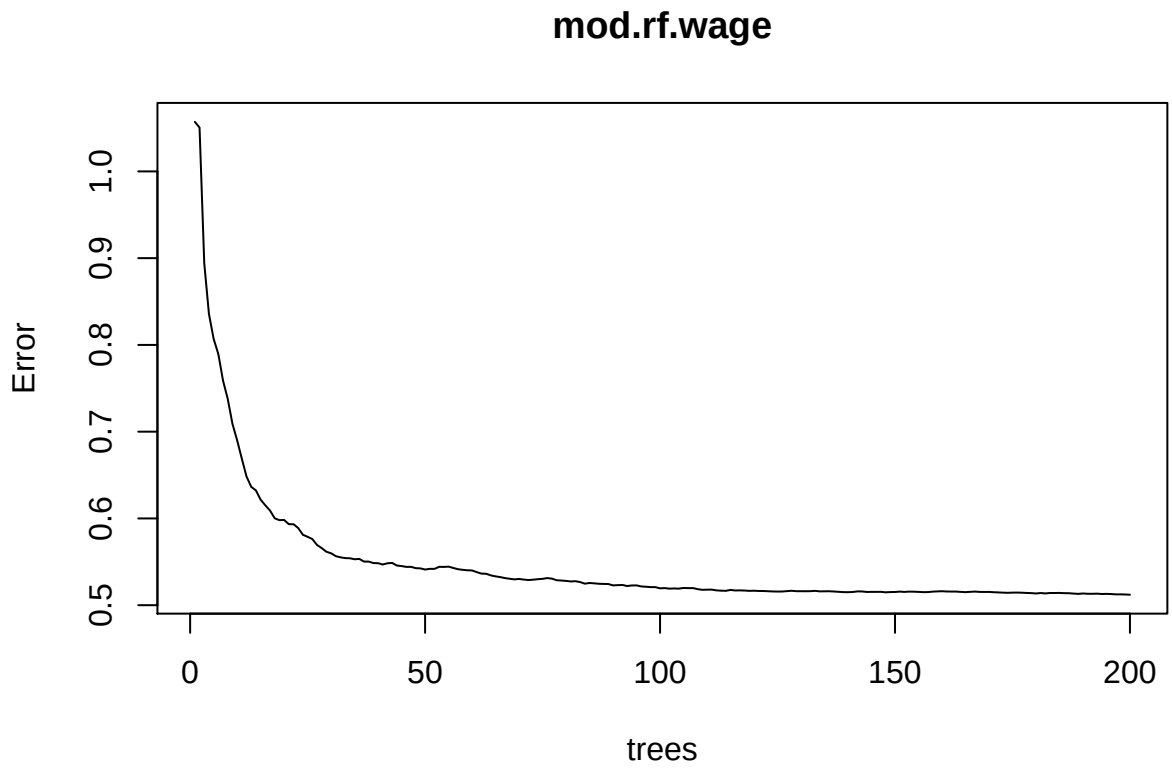
```
mod.rf.wage <- randomForest(lwage ~ .,
                        data=data.df,
                        ntree=numTree,
                        mtry=numPred/3)  ## Use all the predictors
(mse.rf.wage <- min(mod.rf.wage$mse)) # mse
```

```
## [1] 0.5120702
```

```
mod.rf.wage
```

```
##
## Call:
##  randomForest(formula = lwage ~ ., data = data.df, ntree = numTree,      mtry = numPred/3)
##                Type of random forest: regression
##                      Number of trees: 200
## No. of variables tried at each split: 12
##
##           Mean of squared residuals: 0.5120702
##                     % Var explained: 70.73
```

```
plot(mod.rf.wage)
```



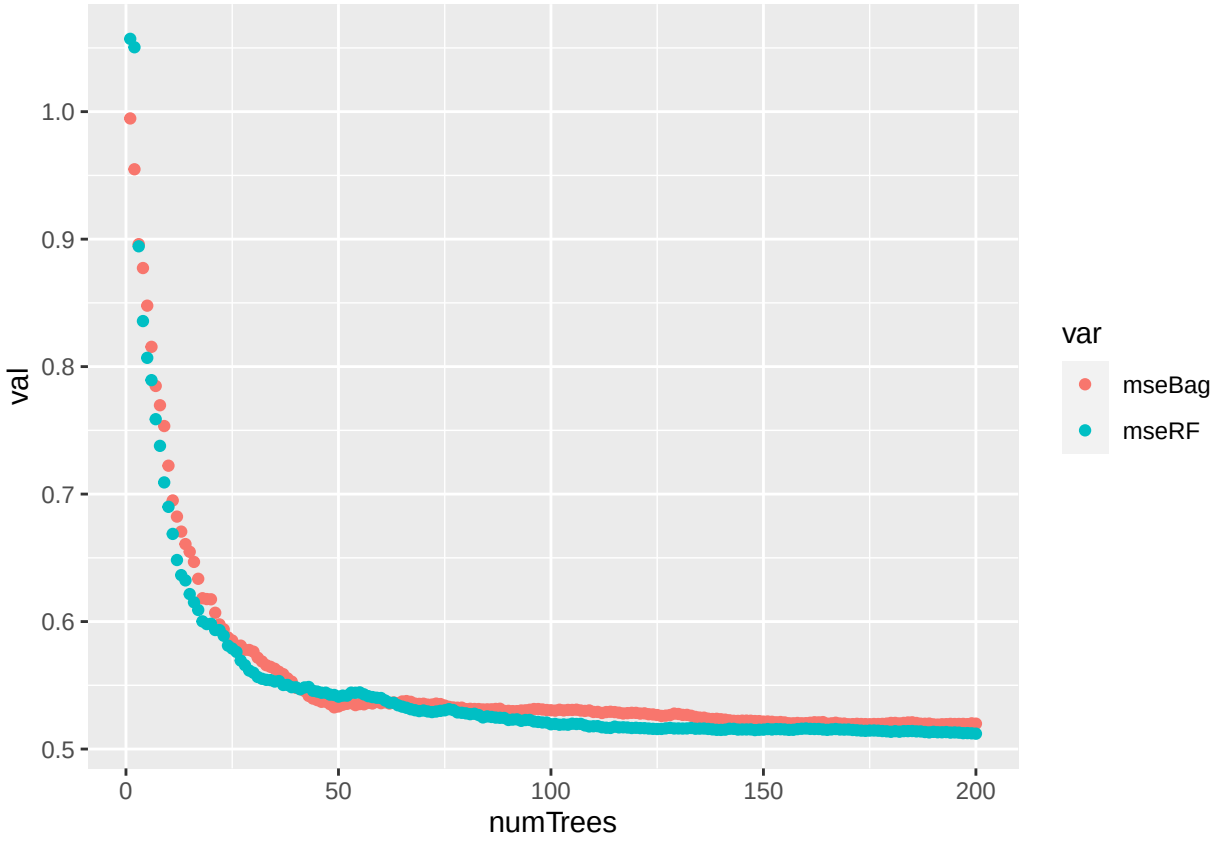
Let's see the visual comparision on the performance of bagging and random forest.

```
results.df <- data.frame(numTrees = 1:numTree,
                         mseBag = mod.bag.wage$mse,
                         mseRF = mod.rf.wage$mse)
```

We can produce a starter plot for this.

```
results.df %>%
  gather(var,val,mseBag:mseRF) %>%
```

```
ggplot()+
geom_point(aes(numTrees,val,color=var))
```



Random forest seem to be slightly better. Comparing results across all models so far.

```
c(mse.lasso.wage, mse.knn.wage, mse.forward.wage, mse.bag.wage, mse.rf.wage)
```

```
## [1] 0.3237012 0.3359333 0.5249096 0.5191662 0.5120702
```

- KNN and penalized regression clearly seem to do a good job.
- Variables like Overall, Potential, Penalties, and Finishing seem to be a good predictors of wage and they all make sense.
- A player with higher potential, scores penalties, has lethal finishing, and has a higher overall ranking obviously deserved a high wage.

Part (D): Wage Classification:

Categorizing wage into three catagories: High, Medium, Low

Players earning more than 25,000 euro a week are High wage, between 5,000 and 25,000 are Mediumm, and below 5,000 euro are Low.

Let's build the dataset called catawage.df.

```
catawage.df <- cbind(stats.df, wage)%>%mutate(catawage = as.factor(ifelse(Wage>25, 'High',ifelse(betwee
with(catawage.df,table(catawage))
```

```
## catawage
##   High    Low Medium
##   2060   9205   6460
```

RandomForest

Running the same model with 1/3 predictors.

```
numTree <- 500
mod.rf.cwage <- randomForest(catawage ~ .,
                         data=catawage.df,
                         mtry=numPreds/3,
                         ntree=numTree)
```

```
(mod.rf.cwage)
```

```
##
## Call:
##  randomForest(formula = catawage ~ ., data = catawage.df, mtry = numPreds/3,      ntree = numTree)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 12
##
##         OOB estimate of  error rate: 0%
## Confusion matrix:
##        High  Low Medium class.error
## High   2060    0      0           0
## Low       0 9205      0           0
## Medium    0    0   6460           0
```

```
(err.rf.cwage <- with(mod.rf.cwage,err.rate[numTree]))
```

```
## [1] 0
```

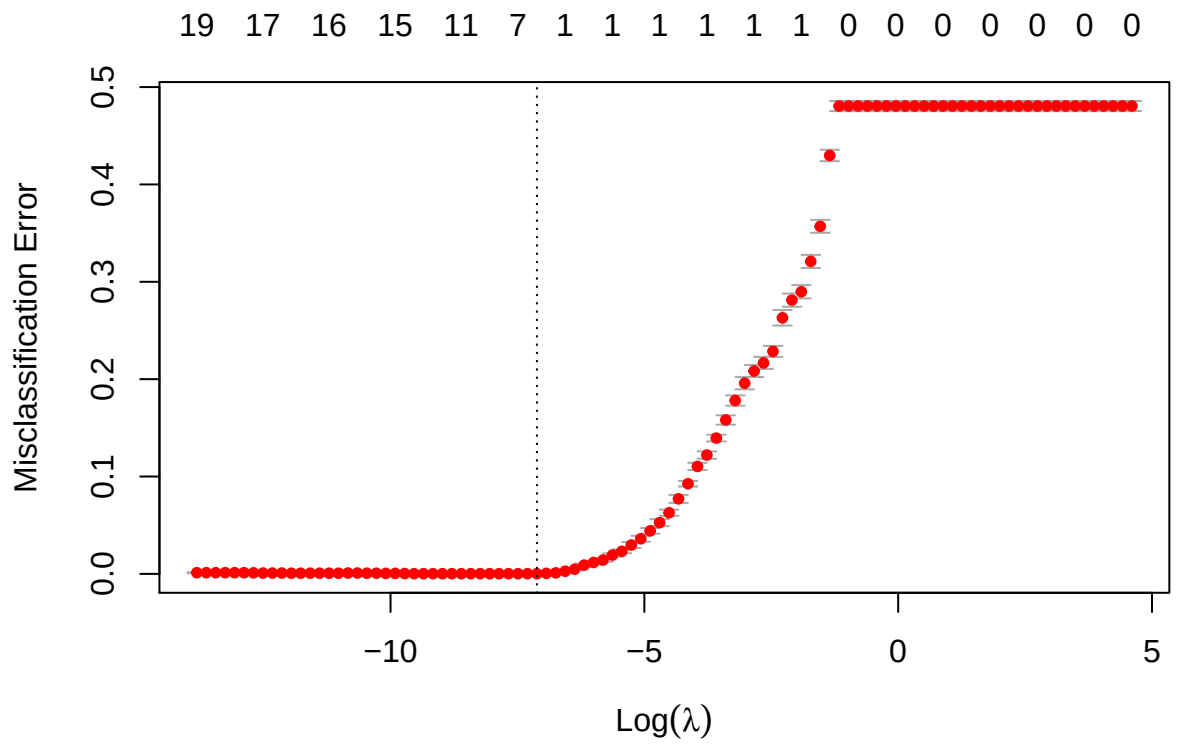
An error rate of 0%. This is amazing.

Lasso regression

```
numPreds <- ncol(catawage.df)-1
cata.w.x <- data.matrix(catawage.df[,1:numPreds])
cata.w.y <- data.matrix(catawage.df[,-(1:numPreds)])
```

We will repeat a similar process with lasso.

```
N <- nrow(cata.w.x)
samp <- sample(1:N, N/3)
cata.w.x <- cata.w.x[samp,]
cata.w.y <- cata.w.y[samp]
lasso.cv.cwage <- cv.glmnet(cata.w.x,cata.w.y,
                        lambda=lambda.grid,
                        family="multinomial",
                        type.measure="class",
                        alpha=1)
plot(lasso.cv.cwage)
```




```
lambda.opt <- lasso.cv.cwage$lambda.1se
id <- with(lasso.cv.cwage,which(lambda==lambda.opt))
(err.lasso.cwage <- with(lasso.cv.cwage,cvm[id]))
```

```
## [1] 0.000169262
```

We again got an error rate of 0.003%, which is grea. Let's create a confusion matrix.

```
mod <- glmnet(cata.w.x,cata.w.y,
                        lambda=lambda.opt,
                        family="multinomial",
                        alpha=1)
pred = predict(mod, newx = cata.w.x, s=lambda.opt, type="class")
(err.ridge.cval <-mean(pred != cata.w.y))
```

```
## [1] 0
```

```
#Confusion Matrix and err
(table(pred, cata.w.y))
```

```
##         cata.w.y
## pred     High  Low Medium
##   High    666    0      0
##   Low       0 3069      0
##   Medium    0    0   2173
```

Amaing result with 0% error.

Conclusion: Our randomforest and lasso regresison did a fantantic job of predicting the catagorical wage of players from fifa 18 dataset. (0% error)!

Second Section

Overall Stat Prediction

So far, although we have made quite a few models and made error analyses, we have not predicted anything. That's not because we can't predict, but we found that predicting the market value of players is not all that interesting. For one, it's all encapsulated in the game (i.e FIFA 19, let's say), and even if you wanted to predict it for the next game (FIFA 19), you would need the next game (FIFA 20) to come out so you would have the player statistics to be able to make the prediction. But if you have the game (FIFA 20), you already know the market value! A wise man once said never predict something you already know.

So we thought we'd predict something we don't know, which is the overall stat of the next game before it even arrives. The FIFA 19 game came out in September of 2018, so they were using the real-life performance statistics from the 2017-18 season (which ends in May 2018) to make the players in the game. Thus, we trained our models to predict the OVerall stat from the FIFA 19 game using the 2017-18 performance statistics. Then, we will use the 2018-19 performance statistics of players and try to predict their Overall stats for FIFA 20. FIFA 20 came out last year, so we also have Overall stats for those players for us to check how well we did!

Since we are dealing with three different datasets in this part of the project, a major challenge was to link the three. They are all recorded with different conventions, the players' names, club names, ages etc are all recorded differently, at different times, and were at times at discord. Quite a bit of effort was put into fixing these issues, and we were still only able to make the predictions for one league, so we chose the most popular one, the English Premier League. The dataset still has about 500 players, so it is not too lacking to make interesting predictions with!

First, we read in the data, and clean it up a bit, fixing any weird disconnects between the datasets.

```
allPrem1718 <- read_csv("~/Desktop/ADM/Final Project/prem1718.csv") #Real-life performance statistics f
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Player = col_character(),
##   Squad = col_character(),
##   Pos = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
allPrem1718 <- allPrem1718 %>%
  rename(Name = Player, Club = Squad) %>%
  dplyr::select(Name, Club, Age:`SoTp90`) #34 predictors

fifa19.df <- read_csv("~/Desktop/ADM/Final Project/fifa19.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   X1 = col_double(),
##   ID = col_double(),
##   Age = col_double(),
##   Overall = col_double(),
##   Potential = col_double(),
##   Special = col_double(),
##   `International Reputation` = col_double(),
##   `Weak Foot` = col_double(),
##   `Skill Moves` = col_double(),
##   `Jersey Number` = col_double(),
##   Crossing = col_double(),
##   Finishing = col_double(),
##   HeadingAccuracy = col_double(),
##   ShortPassing = col_double(),
##   Volleys = col_double(),
##   Dribbling = col_double(),
##   Curve = col_double(),
##   FKAccuracy = col_double(),
##   LongPassing = col_double(),
##   BallControl = col_double()
##   # ... with 24 more columns
## )
## See spec(...) for full column specifications.
```

```
fifa20.df <- read.csv("~/Desktop/ADM/Final Project/fifa20.csv")
fifa20.df$short_name <- as.character(fifa20.df$short_name)

#Fix some names

fifa19.df$Name[[4]] <- "David de Gea"
fifa19.df$Name[[33]] <- "Philippe Coutinho"
fifa19.df$Name[[70]] <- "Cesar Azpilicueta"
fifa19.df$Name[[266]] <- "Alvaro Morata"
fifa19.df$Name[[470]] <- "Gerard Deulofeu"
fifa19.df$Name[[646]] <- "Silva Danilo"
fifa19.df$Name[[665]] <- "J. Gudmundsson"
fifa19.df$Name[[671]] <- "Vicente Iborra"
fifa19.df$Name[[1084]] <- "Cedric Soares"
fifa19.df$Name[[913]] <- "Eric Maxim Choupo-Moting"
fifa19.df$Name[[927]] <- "Heurelho Gomes"
```

```
fifa19.df$Name[[1257]] <- "Ayoze Perez"
fifa19.df$Name[[1296]] <- "Ki Sung-yueng"
fifa19.df$Name[[1744]] <- "Georges-Kevin N'Koudou"
fifa19.df$Name[[1777]] <- "Bojan Krkic"
fifa19.df$Name[[1914]] <- "Mikel Merino"
fifa19.df$Name[[1976]] <- "C. Musonda Jr."
fifa19.df$Name[[2115]] <- "Javier Manquillo"
fifa19.df$Name[[2509]] <- "M. Biram Diouf"
fifa19.df$Name[[5520]] <- "Lee Chung-yong"
allPrem1718$Name[[205]] <- "Heung-min Son"
allPrem1718$Name[[172]] <- "Idrissa Gueye"
allPrem1718$Name[[66]] <- "Bruno Saltor"
allPrem1718$Name[[50]] <- "J. Gudmundsson"
allPrem1718$Name[[110]] <- "Silva Danilo"
fifa20.df$short_name[[145]] <- "Cesar Azpilicueta"
fifa20.df$short_name[[939]] <- "J. Gudmundsson"
fifa20.df$short_name[[1352]] <- "Victor Camarasa"
fifa20.df$short_name[[722]] <- "Silva Danilo"
fifa20.df$short_name[[455]] <- "Gerard Deulofeu"
fifa20.df$short_name[[227]] <- "Idrissa Gana Gueye"
fifa20.df$short_name[[44]] <- "Heung-min Son"
fifa20.df$short_name[[929]] <- "Vicente Iborra"
fifa20.df$short_name[[2261]] <- "Javier Manquillo"
fifa20.df$short_name[[1542]] <- "Martin Montoya"
fifa20.df$short_name[[188]] <- "Alvaro Morata"
fifa20.df$short_name[[2236]] <- "Geroges-Kevin N'Koudou"
fifa20.df$short_name[[14043]] <- "Kayne Ramsey"
fifa20.df$short_name[[1407]] <- "Cedric Soares"
fifa20.df$short_name[[1486]] <- "Ki Sung-yueng"


fifa19Vars.df <- fifa19.df %>%
  dplyr::select(Name, Age, Overall) #Select only three variables from the FIFA dataset

#Remove accents, join the dataframes together

allPrem1718$Name <- iconv(allPrem1718$Name,from="UTF-8",to="ASCII//TRANSLIT")
fifa19Vars.df$Name <- iconv(fifa19Vars.df$Name,from="UTF-8",to="ASCII//TRANSLIT")

fifa19Vars.df <- fifa19Vars.df %>%
  separate(Name, c("FirstName", "LastName"), "(\\s)", extra = "merge") %>%
  mutate(LastName = ifelse(is.na(LastName), FirstName, LastName), FirstName = ifelse(FirstName==LastNam
  mutate(FirstName = substr(FirstName, 1, 1))
```

```
## Warning: Expected 2 pieces. Missing pieces filled with `NA` in 869 rows [28,
## 31, 36, 55, 58, 68, 76, 77, 82, 84, 87, 91, 92, 103, 105, 109, 119, 120, 122,
## 124, ...].
```

```
allPrem1718 <- allPrem1718 %>%
  separate(Name, c("FirstName", "LastName"), "(\\s)", extra = "merge") %>%
  mutate(LastName = ifelse(is.na(LastName), FirstName, LastName), FirstName = ifelse(FirstName==LastNam
  dplyr::select(FirstName, LastName, Age:`SoTp90`) %>%
  mutate(FirstName = substr(FirstName, 1, 1))
```

```
## Warning: Expected 2 pieces. Missing pieces filled with `NA` in 9 rows [5, 142,
## 156, 231, 235, 248, 382, 404, 511].
```

```
data1718 <- allPrem1718 %>%
  mutate(Age = Age+1) %>%  #Fixed the age!
  left_join(fifa19Vars.df, by = c("FirstName", "LastName", "Age"))

data1718[is.na(data1718$Overall),] #not included in the game for whatever reason
```

```
## # A tibble: 37 x 38
##    FirstName LastName   Age    MP   Gls   Ast    PK PKatt  CrdY  CrdR Glsp90
##    <chr>     <chr>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
##  1 R         Aarons      22     4     0     0     0     0     0     0   0
##  2 <NA>      Adri'an     31    19     0     0     0     0     2     0   0
##  3 N         Amrabat     31     3     0     0     0     0     0     0   0
##  4 L         Britton     35     5     0     0     0     0     1     0   0
##  5 Y         Cabaye      32    31     0     1     0     0     7     0   0
##  6 '         Capoue      29    23     1     0     0     0     3     0   0.06
##  7 M         Carrick     36     2     0     0     0     0     0     0   0
##  8 J         Collins     34    13     1     0     0     0     3     0   0.09
##  9 T         Deeney      30    29     5     2     3     5     1     1   0.24
## 10 D         Delaney     36     2     0     0     0     0     0     0   0
## # ... with 27 more rows, and 27 more variables: Astp90 <dbl>, GnPKp90 <dbl>,
## #   xG <dbl>, npxG <dbl>, xA <dbl>, xGp90 <dbl>, xAp90 <dbl>, npxGp90 <dbl>,
## #   CmpPct <dbl>, TotDist <dbl>, PrgDist <dbl>, SCmpPct <dbl>, MCmpPct <dbl>,
## #   LCmpPct <dbl>, KP <dbl>, DrbSuccPct <dbl>, Tkl <dbl>, TklW <dbl>,
## #   Blocks <dbl>, Int <dbl>, Clr <dbl>, Err <dbl>, Pos <chr>, Sh <dbl>,
## #   SoT <dbl>, SoTp90 <dbl>, Overall <dbl>
```

Now that we have a nice dataset, let's subset just the outfield players. We don't want to predict the Overall stat of goalkeepers based on how many goals they scored or how many key passes they made! We could make a separate model for the goalkeepers instead. One could take this further and try to predict different positions with separate models, but that might be more difficult, as there is a lot of variance even within one position. Two players may have the same position on paper and yet have wildly differing playstyles and influences, and two playeers with different positions may be similar. For this project, let's stick with modelling all the outfield players together.

```
#Remove the goalkeepers
outData1718 <- data1718 %>%
  filter(!str_detect(Pos, "GK"))

#More cleaning
outData1718.df <- outData1718 %>%
  dplyr::select(-FirstName, -LastName, -Pos) %>%
  filter(Overall != 0)

#Change NAs to 0
outData1718.df[is.na(outData1718.df)] <- 0

outData1718.df
```

```
## # A tibble: 460 x 35
##      Age    MP   Gls   Ast    PK PKatt  CrdY  CrdR Glsp90 Astp90 GnPKp90    xG
```

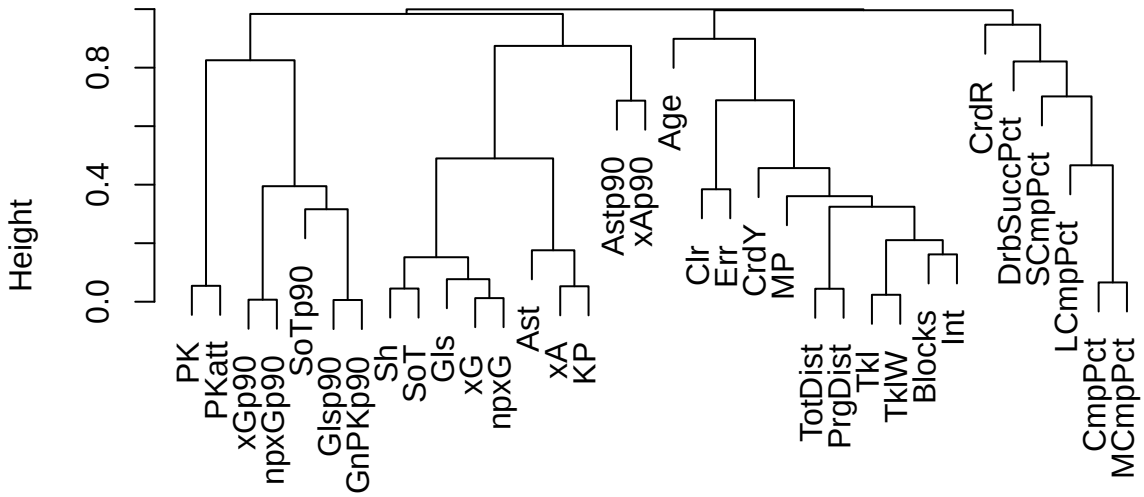
```
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl>   <dbl> <dbl>
##  1    27    28     5     1     0     0     7     0   0.21   0.04    0.21   3.4
##  2    20    31     5     1     0     0     0     0   0.26   0.05    0.26   6.4
##  3    32    11     0     0     0     1     2     1   0      0       0      1.4
##  4    32     6     0     0     0     0     1     0   0      0       0      0
##  5    25    17     0     0     0     0     0     0   0      0       0      1.5
##  6    30    25    21     6     4     4     2     0   0.96   0.28    0.78  16.8
##  7    23    38     2     3     0     0     5     0   0.05   0.08    0.05   2.4
##  8    28    34     2     7     0     0     5     1   0.07   0.25    0.07   1.6
##  9    29    14     0     0     0     0     3     0   0      0       0      0.5
## 10    19    19     1     1     0     0     3     0   0.06   0.06    0.06   1.2
## # ... with 450 more rows, and 23 more variables: npxG <dbl>, xA <dbl>,
## #   xGp90 <dbl>, xAp90 <dbl>, npxGp90 <dbl>, CmpPct <dbl>, TotDist <dbl>,
## #   PrgDist <dbl>, SCmpPct <dbl>, MCmpPct <dbl>, LCmpPct <dbl>, KP <dbl>,
## #   DrbSuccPct <dbl>, Tkl <dbl>, TklW <dbl>, Blocks <dbl>, Int <dbl>,
## #   Clr <dbl>, Err <dbl>, Sh <dbl>, SoT <dbl>, SoTp90 <dbl>, Overall <dbl>
```

Nice! Now we have a nice dataset with just numbers. Let's quickly take a look at the correlation heatmap to see if we spot anything interesting.

```
#Correlation Heatmap for Player Statistics

numPreds <- ncol(outData1718.df)-1
feature.mat <- data.matrix(outData1718.df[,1:numPreds])
feature.mat.scaled <- scale(feature.mat)

feature.cor <- cor(feature.mat)
feature.dist <- 1-as.dist(abs(feature.cor))
feature.cluster <- hclust(feature.dist,method="complete")
plot(feature.cluster)
```



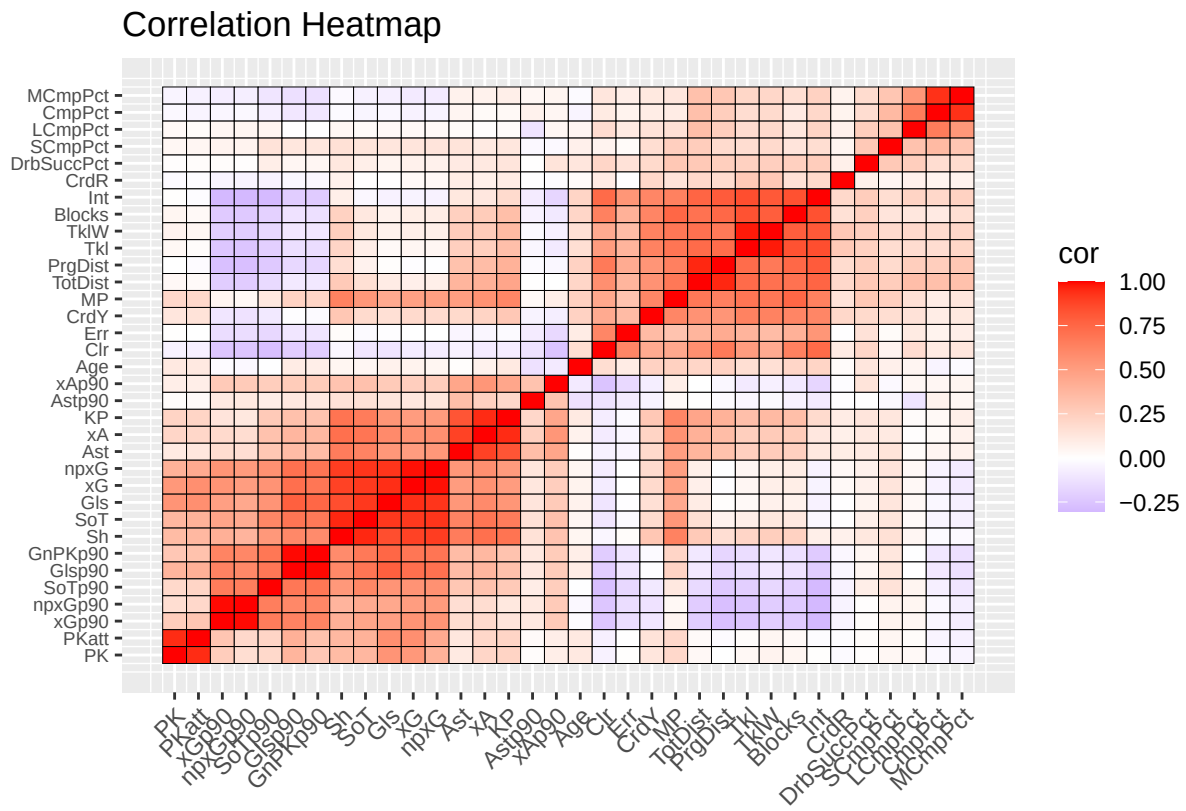

```
feature.names <- names(outData1718.df[,1:numPreds])
cluster.ord <- feature.cluster$order
feature.cor <- feature.cor[cluster.ord,cluster.ord]
feature.names.ord <- feature.names[cluster.ord]
featureCor.df <- data.frame(feature.cor)
featureCor.df$pred1 <-1:numPreds
featureCor.df <- featureCor.df  %>%
  gather(pred2.name,cor,1:numPreds) %>%
  inner_join(data.frame(pred2.name=feature.names.ord,
                        pred2=1:numPreds)) %>%
  dplyr::select(-pred2.name)
```

```
## Joining, by = "pred2.name"

## Warning: Column ‘pred2.name‘ joining character vector and factor, coercing into
## character vector
```

```
breakLocations=1:numPreds
mpt <- with(featureCor.df,mean(cor))
featureCor.df %>%
  ggplot()+
  geom_tile(aes(pred1,pred2,fill = cor),color="black")+
  theme(axis.text.x = element_text(angle = 45, hjust = 1,size=9),
        axis.text.y = element_text(size=7))+
  scale_fill_gradient2(low="blue",high="red",midpoint=0)+
```

```
  scale_x_continuous(breaks=breakLocations,labels=feature.names.ord)+
  scale_y_continuous(breaks=breakLocations,labels=feature.names.ord)+
  labs(title="Correlation Heatmap",
       x="",y="")
```



That's pretty cool. We can see how the offensive stats, like goals scored or shots taken, are more correlated with each other, while being negatively correlated with defensive stats, like tackles won or interceptions. The xStats are a new, sophisticated addition to football analytics, and they involve using freeze-frames to see how "easy" it is to score a particular goal, or make an assist, etc. The "expectedness" of a goal, for instance, is called xG, and is quickly becoming a staple stat.

Ridge Regression

We made the choice to not do many models; we already did quite a few for the market value calculations, so we are familiar with them. Let's do the best five. Kicking off with Ridge Regression.

```
numPreds <- ncol(outData1718.df)-1
ovr.x <- data.matrix(outData1718.df[,1:numPreds])
ovr.y <- data.matrix(outData1718.df[,-(1:numPreds)])
lambda.grid <- 10^seq(-5,1,length=50)

ridge.cv <- cv.glmnet(ovr.x,ovr.y,lambda=lambda.grid,alpha=0)
lambda.opt <- ridge.cv$lambda.1se
id <- with(ridge.cv,which(lambda==lambda.opt))
```

```
ridge.mse <- with(ridge.cv,cvm[id])
ridge.mse #mean squared error
```

```
## [1] 13.36309
```

```
ridge.mod <- ridge.cv$glmnet.fit
opt_lambda <- ridge.cv$lambda.min

predOvrRidge <-  predict(ridge.mod, s = opt_lambda, newx = ovr.x)

sst <- sum((ovr.y - mean(ovr.y))^2)
sse <- sum((predOvrRidge - ovr.y)^2)

# R squared
rsqRidge <- 1 - sse / sst
rsqRidge #R^2 on the training data (2017-18)
```

```
## [1] 0.5891586
```

The R^2 isn't half bad! Looks like we're off to a good start. Let's use this model to make predictions with the 2018/19 performance statistics.

```
allPrem1819 <- read_csv("~/Desktop/ADM/Final Project/prem1819.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Player = col_character(),
##   Squad = col_character(),
##   Pos = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
allPrem1819 <- allPrem1819 %>%
  mutate(Name = Player, Club = Squad) %>%
  dplyr::select(-Player,-Squad) %>%
  dplyr::select(Name, Club, Age:`SoTp90`)

outData1819 <- allPrem1819 %>%
  filter(!(str_detect(Pos, "GK")))

outData1819.df <- outData1819 %>%
  dplyr::select(-Name, -Club, -Pos)

outData1819.df[is.na(outData1819.df)] <- 0

new.x <- data.matrix(outData1819.df[,1:numPreds])

predOvrRidge <-  predict(ridge.mod, s = opt_lambda, newx = new.x)
```

```
predOvrRidge <- as.vector(predOvrRidge)

outData1819$predOvrRidge <- predOvrRidge

predictions <- outData1819 %>%
  dplyr::select(Name, Club, Age, predOvrRidge)

predictions %>%
  arrange(desc(predOvrRidge))
```

```
## # A tibble: 470 x 4
##    Name               Club              Age predOvrRidge
##    <chr>              <chr>           <dbl>        <dbl>
##  1 Eden Hazard        Chelsea            27         89.8
##  2 Mohamed Salah      Liverpool          26         89.3
##  3 Sergio Agüero      Manchester City    30         88.8
##  4 Virgil van Dijk    Liverpool          27         87.6
##  5 David Luiz         Chelsea            31         87.4
##  6 Sadio Mané         Liverpool          26         86.8
##  7 Paul Pogba         Manchester Utd     25         86.6
##  8 David Silva        Manchester City    32         86.6
##  9 Aymeric Laporte    Manchester City    24         86.5
## 10 Christian Eriksen  Tottenham          26         85.7
## # ... with 460 more rows
```

Alright, we've got the first few predictions on board! That top 10 list of players has some very familiar names; if you'd asked me to name the ten best players in the Premier League, those would all be in the conversation, so that's a good sign that we're on the right track.

Lasso Regression

Let's try that again, this time with Lasso.

```
lasso.cv <- cv.glmnet(ovr.x,ovr.y,lambda=lambda.grid,alpha=1)
lambda.opt <- lasso.cv$lambda.1se
id <- with(lasso.cv,which(lambda==lambda.opt))
lasso.mse <- with(lasso.cv,cvm[id])
lasso.mse #mean squared error
```

```
## [1] 13.59675
```

```
lasso.mod <- lasso.cv$glmnet.fit
opt_lambda <- lasso.cv$lambda.min

predOvrLasso <-  predict(lasso.mod, s = opt_lambda, newx = ovr.x)

sst <- sum((ovr.y - mean(ovr.y))^2)
sse <- sum((predOvrLasso - ovr.y)^2)

# R squared
rsqLasso <- 1 - sse / sst
```

```
new.x <- data.matrix(outData1819.df[,1:numPreds])

predOvrLasso <-  predict(lasso.mod, s = opt_lambda, newx = new.x)

predOvrLasso <- as.vector(predOvrLasso)

predictions$predOvrLasso <- predOvrLasso

predictions %>%
  arrange(desc(predOvrLasso))
```

```
## # A tibble: 470 x 5
##    Name              Club              Age predOvrRidge predOvrLasso
##    <chr>             <chr>           <dbl>        <dbl>        <dbl>
##  1 Eden Hazard       Chelsea            27         89.8         90.3
##  2 Mohamed Salah     Liverpool          26         89.3         88.7
##  3 Sergio Agüero     Manchester City    30         88.8         88.6
##  4 Virgil van Dijk   Liverpool          27         87.6         88.4
##  5 Paul Pogba        Manchester Utd     25         86.6         87.5
##  6 David Luiz        Chelsea            31         87.4         87.3
##  7 Aymeric Laporte   Manchester City    24         86.5         86.7
##  8 Sadio Mané        Liverpool          26         86.8         86.6
##  9 David Silva       Manchester City    32         86.6         86.5
## 10 Christian Eriksen Tottenham          26         85.7         86.2
## # ... with 460 more rows
```

Quite similar, actually. Again, the top ten names are all of world class footballers, so there seems to be nothing out of the ordinary here.

Boosted Regression

Let's try something that's not a penalized regression for a change, how about some boosting? Let's make some test/train combos and predict away.

```
N <- nrow(outData1718.df)
train <- sample(1:N,N/2,rep=F)
train.df <- outData1718.df[train,]
test.df <- outData1718.df[-train,]

GBMfunc <- function(shrink,depth,numTrees,numFolds=10){
  folds <-  sample(1:numFolds,N,rep=T)
  mse <- numeric(numFolds)
  fold <- 1
  for(fold in 1:numFolds){
    train.df <- outData1718.df[folds != fold,]
    test.df <- outData1718.df[folds == fold,]
    mod.gbm <- gbm(Overall  ~  .,
                   data=train.df,
                   distribution="gaussian",
                   shrinkage=shrink,
                   n.trees=numTrees,
```

```
                    interaction.depth = depth)
    pred.gbm <- predict(mod.gbm,
                        newdata=test.df,
                        shrinkage=shrink,
                        n.trees=numTrees,
                        interaction.depth = depth)
    mse[fold] <- with(test.df,mean((Overall - pred.gbm)^2))
  }
  mean(mse)
}

GBMfunc(0.1, 2, 100) #Just checking, already a lower MSE!
```

```
## [1] 11.22855
```

Just used a few standard shrinkage, depth and number of trees values for the boosting, and we already get a lower MSE! Let's find the best values for these parameters.

```
shrinks <- c(1,.1,.05,.01)
depths <- c(1,2,3, 4)
cv.vals <- expand.grid(shrinks,depths)
mse.vals <- apply(cv.vals,1,function(row) GBMfunc(row[1],row[2],250, 5))
minID <- which.min(mse.vals)
boost.mse <- mse.vals[minID]
boost.mse #lowest mse
```

```
## [1] 10.66484
```

```
cv.vals[minID,] #the best shrink and depth respectively
```

```
##   Var1 Var2
## 7 0.05    2
```

And now we use these to find the best number of trees.

```
theShrink <- cv.vals[minID, 1]
theDepth <- cv.vals[minID, 2]
tree.vals <- seq(10,250, by = 10)
mse.vals <- c()
for(i in 1:length(tree.vals)){
  mse.vals[i] <- GBMfunc(theShrink, theDepth, tree.vals[i], 5)
}
min(mse.vals)
```

```
## [1] 10.66144
```

```
minID <- which.min(mse.vals)
tree.vals[minID] #best tree size
```

```
## [1] 240
```

Great! Now let's use the model to make some predictions.

```
numTrees <- tree.vals[minID]
mod.gbm <- gbm(Overall  ~  .,
                   data=outData1718.df,
                   distribution="gaussian",
                   shrinkage=theShrink,
                   n.trees=numTrees,
                   interaction.depth = theDepth)
predOvrBoost <- predict(mod.gbm,
                        newdata=outData1819.df,
                        shrinkage=shrink,
                        n.trees=numTrees,
                        interaction.depth = depth)
predOvrBoost <- as.vector(predOvrBoost)

predictions$predOvrBoost <- predOvrBoost

predictions %>%
  arrange(desc(predOvrBoost))
```

```
## # A tibble: 470 x 6
##    Name              Club             Age predOvrRidge predOvrLasso predOvrBoost
##    <chr>             <chr>          <dbl>        <dbl>        <dbl>        <dbl>
##  1 Eden Hazard       Chelsea           27         89.8         90.3         88.9
##  2 Sergio Agüero     Manchester Ci~    30         88.8         88.6         87.7
##  3 Mohamed Salah     Liverpool         26         89.3         88.7         87.2
##  4 Harry Kane        Tottenham         25         83.3         83.5         86.0
##  5 Paul Pogba        Manchester Utd    25         86.6         87.5         85.9
##  6 Christian Eriksen Tottenham         26         85.7         86.2         85.8
##  7 Gerard Deulofeu   Watford           24         80.9         80.6         85.6
##  8 Roberto Firmino   Liverpool         26         84.4         84.0         85.6
##  9 Raheem Sterling   Manchester Ci~    23         85.5         85.1         85.6
## 10 David Silva       Manchester Ci~    32         86.6         86.5         85.4
## # ... with 460 more rows
```

Fantastic. The MSE for this model is smaller, so it could be even better, and it looks like we moved around a few players. Interestingly, we're seeing that defenders are falling down the pecking order. It could be that the Boosting algorithm is noticing a stronger correlation between higher overall stats and good offensive contribution. I wouldn't be surprised; people have complained in the past that the game tends to underrate defenders and overrate forwards.

Random Forest

Let's try using a random forest now. It's usually quite competitive with boosted regression. They're quite related after all!

```
mod.rf <- randomForest(Overall ~ .,
                       data = outData1718.df,
                       mtry = (ncol(outData1718.df)-1)/3,
                       ntree = 1000)
minID <- which.min(mod.rf$mse)
```

```
rf.mse <- with(mod.rf, mse[minID])
rf.mse #Pretty competitive with boosting
```

```
## [1] 10.67638
```

And now predict with this model.

```
predOvrRF <- predict(mod.rf, newdata = outData1819.df)

predictions$predOvrRF <- predOvrRF

predictions %>%
  arrange(desc(predOvrRF))
```

```
## # A tibble: 470 x 7
##    Name           Club       Age predOvrRidge predOvrLasso predOvrBoost predOvrRF
##    <chr>          <chr>    <dbl>        <dbl>        <dbl>        <dbl>     <dbl>
##  1 Eden Hazard    Chelsea     27         89.8         90.3         88.9      88.7
##  2 Mohamed Salah  Liverpo~    26         89.3         88.7         87.2      86.8
##  3 Sergio Agüero  Manches~    30         88.8         88.6         87.7      86.6
##  4 Raheem Sterl~  Manches~    23         85.5         85.1         85.6      86.2
##  5 Paul Pogba     Manches~    25         86.6         87.5         85.9      85.4
##  6 Christian Er~  Tottenh~    26         85.7         86.2         85.8      85.1
##  7 Roberto Firm~  Liverpo~    26         84.4         84.0         85.6      85.1
##  8 Harry Kane     Tottenh~    25         83.3         83.5         86.0      85.1
##  9 Raúl Jiménez   Wolves      27         84.6         84.4         84.4      85.0
## 10 Pierre-Emeri~  Arsenal     29         85.2         85.3         84.0      84.9
## # ... with 460 more rows
```

So far, we have predicted with 4 models. Let's do one more, and then we can check in with the real results.

KNN

Ending it with good ol' KNN.

```
#KNN

ovr.x <- scale(ovr.x)

N <- nrow(ovr.x)

KNNfunc <- function(kVal,numFolds=10){
  folds <- sample(1:numFolds,N,rep=T)
  errs <- numeric(numFolds)
  for(fold in 1:numFolds){
    train.x <- ovr.x[folds != fold,]
    train.y <- ovr.y[folds != fold]
    test.x <- ovr.x[folds == fold,]
    test.y <-  ovr.y[folds == fold]
    knn.mod <- knn.reg(train.x,test.x,train.y,k=kVal)
    errs[fold] <- with(knn.mod,mean((pred-test.y)^2))
```

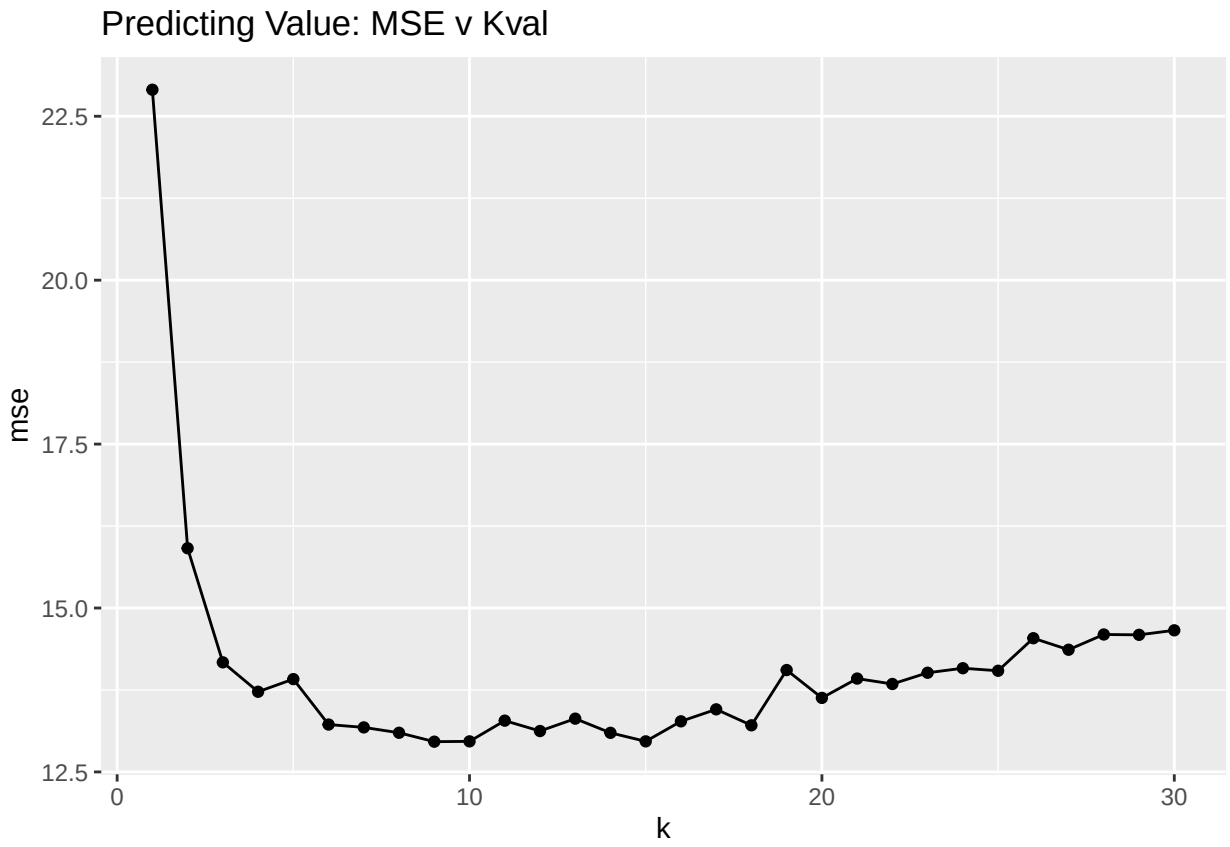
```
  }
c(mean(errs),sd(errs))
}

KNNfuncMean <- function(kVal, numFolds=10){
  KNNfunc(kVal, numFolds)[1]
} #just mse

KNNfuncSE <- function(kVal, numFolds=10){
  KNNfunc(kVal, numFolds)[2]
} #se

kVals <- 1:30
allErrs <- map_dbl(kVals, KNNfuncMean)
allSEs <- map_dbl(kVals, KNNfuncSE)

data.frame(k=1:30,mse=allErrs) %>%
  ggplot(aes(k,mse)) +
  geom_point()+
  geom_line()+
  labs(title="Predicting Value: MSE v Kval")
```




```
kOpt <- which.min(allErrs)
knn.mse <- allErrs[kOpt]
knn.mse
```

```
## [1] 12.96127
```

That's a nice mean-squared error, actually, it's better than the penalized regressions! It's still worse than the decision tree based models, though. Let's make some predictions.

```
train.x <- scale(ovr.x)
test.x <- scale(as.matrix(outData1819.df))
train.y <- outData1718.df$Overall

knn.mod <- knn.reg(train.x,test.x,train.y,k=kOpt)

predictions$predOvrKNN <- knn.mod$pred

predictions %>%
  arrange(desc(predOvrKNN))
```

```
## # A tibble: 470 x 8
##    Name  Club    Age predOvrRidge predOvrLasso predOvrBoost predOvrRF predOvrKNN
##    <chr> <chr> <dbl>        <dbl>        <dbl>        <dbl>     <dbl>      <dbl>
##  1 Moha~ Live~    26         89.3         88.7         87.2      86.8       87.2
##  2 Pier~ Arse~    29         85.2         85.3         84.0      84.9       86.1
##  3 Serg~ Manc~    30         88.8         88.6         87.7      86.6       86
##  4 Harr~ Tott~    25         83.3         83.5         86.0      85.1       85.4
##  5 Eden~ Chel~    27         89.8         90.3         88.9      88.7       85.2
##  6 Raúl~ Wolv~    27         84.6         84.4         84.4      85.0       85
##  7 Rahe~ Manc~    23         85.5         85.1         85.6      86.2       84.4
##  8 Lero~ Manc~    22         81.4         81.1         84.1      84.7       84.2
##  9 Jami~ Leic~    31         82.2         82.2         81.6      81.3       84.2
## 10 Ryan~ Bour~    24         83.4         82.6         81.1      81.4       84.1
## # ... with 460 more rows
```

Some other names are starting to rise up as well. But along the way, a few players such as Eden Hazard, Mohamed Salah and Sergio Aguero have been mainstays. At the very least, we can be confident they'll be high up on the actual list as well.

Comparing MSEs

Before we go to the actual FIFA 20 Overalls, let's compare the MSEs one last time.

```
mods <- c("Ridge", "Lasso", "Boosting", "Random Forest", "KNN")
mses <- c(ridge.mse, lasso.mse, boost.mse, rf.mse, knn.mse)
rbind(mods, mses)
```

```
##      [,1]               [,2]               [,3]
## mods "Ridge"            "Lasso"            "Boosting"
## mses "13.3630857480971" "13.5967518675405" "10.6648406068345"
##      [,4]               [,5]
## mods "Random Forest"    "KNN"
## mses "10.6763761219534" "12.9612734047596"
```

Looks like boosted regression narrowly edges random forest, KNN comes in third and then finally the penalized regressions!

Comparing with real FIFA 20 Overalls

Let's get the data in and join it with our predictions dataset

```
#Compare with real FIFA 20 values

fifa20Vars.df <- fifa20.df%>%
  dplyr::select(Name = short_name, Age = age, Overall = overall)

fifa20Vars.df$Name <- iconv(fifa20Vars.df$Name,from="UTF-8",to="ASCII//TRANSLIT")

fifa20Vars.df <- fifa20Vars.df %>%
  separate(Name, c("FirstName", "LastName"), "(\\s)", extra = "merge") %>%
  mutate(LastName = ifelse(is.na(LastName), FirstName, LastName), FirstName = ifelse(FirstName==LastNam
  mutate(FirstName = substr(FirstName, 1, 1))
```

```
## Warning: Expected 2 pieces. Missing pieces filled with `NA` in 813 rows [14,
## 26, 30, 43, 45, 53, 57, 63, 69, 70, 80, 81, 82, 92, 95, 102, 106, 108, 116,
## 129, ...].
```

```
predictions <- predictions %>%
  separate(Name, c("FirstName", "LastName"), "(\\s)", extra = "merge", remove = FALSE) %>%
  mutate(LastName = ifelse(is.na(LastName), FirstName, LastName), FirstName = ifelse(FirstName==LastNam
  mutate(FirstName = substr(FirstName, 1, 1))
```

```
## Warning: Expected 2 pieces. Missing pieces filled with `NA` in 13 rows [45, 46,
## 56, 93, 130, 135, 142, 204, 205, 218, 332, 356, 452].
```

```
predictions$FirstName <- iconv(predictions$FirstName,from="UTF-8",to="ASCII//TRANSLIT")
predictions$LastName <- iconv(predictions$LastName,from="UTF-8",to="ASCII//TRANSLIT")

predictions <- predictions %>%
  mutate(Age = Age+1) %>%
  left_join(fifa20Vars.df, by = c("FirstName", "LastName", "Age")) %>%
  dplyr::select(-FirstName, -LastName, actualOverall = Overall)

#predictions[is.na(predictions$actualOverall),] #Not in the game for whatever reason

finalPredictions <- predictions %>%
  filter(!is.na(actualOverall))

finalPredictions %>%
  arrange(desc(actualOverall))
```

```
## # A tibble: 440 x 9
##    Name  Club    Age predOvrRidge predOvrLasso predOvrBoost predOvrRF predOvrKNN
##    <chr> <chr> <dbl>        <dbl>        <dbl>        <dbl>     <dbl>      <dbl>
##  1 Kevi~ Manc~    28         79.0         78.9         80.7      80.4       78.6
##  2 Eden~ Chel~    28         89.8         90.3         88.9      88.7       85.2
##  3 Virg~ Live~    28         87.6         88.4         84.0      83.5       82.3
##  4 Moha~ Live~    27         89.3         88.7         87.2      86.8       87.2
##  5 Serg~ Manc~    31         88.8         88.6         87.7      86.6       86
```

```
##  6 Harr~ Tott~    26         83.3         83.5         86.0      85.1       85.4
##  7 N'Go~ Chel~    28         82.0         82.4         83.8      82.1       77.7
##  8 Pier~ Arse~    30         85.2         85.3         84.0      84.9       86.1
##  9 Chri~ Tott~    27         85.7         86.2         85.8      85.1       83.6
## 10 Sadi~ Live~    27         86.8         86.6         84.5      84.6       84.1
## # ... with 430 more rows, and 1 more variable: actualOverall <int>
```

Not bad! Hazard, Van Dijk, Salah, Aguero, Eriksen... these are all names we saw quite a few times in the top ten. As you may have noticed, we're missed the mark on the first player by quite a bit. Why might this be? Well, Kevin De Bruyne, although one of the best in the league for certain, missed more than half of the 2018/19 season due to injuries. Thus, his performance stats didn't do justice to his capability, which the actual FIFA 20 Overall stat reflects. Kante is another player who seems to have gone missing. Although his performance stats are okay, sometimes, there are just qualities in a player that are simply too hard to quantify and put in a statistical box. Kante is a very important player for Chelsea and deserves his high rating in the game. Perhaps more stats geared towards his defensive/pressuring style of football would help?

Let's calculate a mean squared error based on this actual value and our predicted values, and then declare a final winner.

```
mse2020ridge <- mean((finalPredictions$predOvrRidge - finalPredictions$actualOverall)^2)
mse2020lasso <- mean((finalPredictions$predOvrLasso - finalPredictions$actualOverall)^2)
mse2020boost <- mean((finalPredictions$predOvrBoost - finalPredictions$actualOverall)^2)
mse2020rf <- mean((finalPredictions$predOvrRF - finalPredictions$actualOverall)^2)
mse2020KNN <- mean((finalPredictions$predOvrKNN - finalPredictions$actualOverall)^2)

mses2020 <- c(mse2020ridge, mse2020lasso, mse2020boost, mse2020rf, mse2020KNN)
rbind(mods, mses2020) #Looks like Boosted Wins!
```

```
##          [,1]               [,2]               [,3]
## mods     "Ridge"            "Lasso"            "Boosting"
## mses2020 "12.5900127055412" "12.6110956971148" "9.22358041101357"
##          [,4]               [,5]
## mods     "Random Forest"    "KNN"
## mses2020 "9.85200828197555" "13.7881593714927"
```

Looks like boosted regression wins! Random forest comes in second, then the penalized regressions, and KNN is actually fifth, despite being third in the cross validation MSE calculations.

Goalkeeper Predictions

We have been ignoring goalkeepers for a while, so let's give them some love. Let's use specific goalkeeping stats to try to predict their overall score. There are far fewer goalkeepers than outfield players, so our dataset is much more limited. We might have to be more creative with our modeling. We could do every model again, but let's just go with boosting, since it performed best on the outfield players.

```
gkPrem1718 <- read_csv("~/Desktop/ADM/Final Project/prem1718gk.csv")
```

```
## Parsed with column specification:
## cols(
##   Player = col_character(),
##   Squad = col_character(),
##   Age = col_double(),
```

```
##   MP = col_double(),
##   GA = col_double(),
##   GA90 = col_double(),
##   SoTA = col_double(),
##   Saves = col_double(),
##   W = col_double(),
##   D = col_double(),
##   L = col_double(),
##   CS = col_double(),
##   `CS%` = col_double(),
##   PKatt = col_double(),
##   PKA = col_double(),
##   PKsv = col_double(),
##   PKm = col_double()
## )
```

```
gkPrem1819 <- read_csv("~/Desktop/ADM/Final Project/prem1819gk.csv")
```

```
## Parsed with column specification:
## cols(
##   Player = col_character(),
##   Squad = col_character(),
##   Age = col_double(),
##   MP = col_double(),
##   GA = col_double(),
##   GA90 = col_double(),
##   SoTA = col_double(),
##   Saves = col_double(),
##   W = col_double(),
##   D = col_double(),
##   L = col_double(),
##   CS = col_double(),
##   `CS%` = col_double(),
##   PKatt = col_double(),
##   PKA = col_double(),
##   PKsv = col_double(),
##   PKm = col_double()
## )
```

```
gkPrem1718$Player <- iconv(gkPrem1718$Player,from="UTF-8",to="ASCII//TRANSLIT")
gkPrem1819$Player <- iconv(gkPrem1819$Player,from="UTF-8",to="ASCII//TRANSLIT")

gkPrem1718 <- gkPrem1718 %>%
  rename(Name = Player, Club = Squad) %>%
  separate(Name, c("FirstName", "LastName"), "(\\s)", extra = "merge") %>%
  mutate(LastName = ifelse(is.na(LastName), FirstName, LastName), FirstName = ifelse(FirstName==LastNam
  dplyr::select(FirstName, LastName, Age:PKm) %>%
  mutate(FirstName = substr(FirstName, 1, 1))
```

```
## Warning: Expected 2 pieces. Missing pieces filled with `NA` in 2 rows [1, 10].
```

```
gkData1718 <- gkPrem1718 %>%
  mutate(Age = Age+1) %>%
  left_join(fifa19Vars.df, by = c("FirstName", "LastName", "Age"))

gkData1718[is.na(gkData1718$Overall),] #Not in the game for whatever reason
```

```
## # A tibble: 3 x 18
##   FirstName LastName   Age    MP    GA  GA90  SoTA Saves     W     D     L    CS
##   <chr>     <chr>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 <NA>      Adri'an     31    19    29  1.53    95    68     7     6     6     6
## 2 B         Foster      35    37    55  1.49   152    99     6    12    19    10
## 3 O         Karnezis    33    15    24  1.67    52    31     4     3     7     4
## # ... with 6 more variables: `CS%` <dbl>, PKatt <dbl>, PKA <dbl>, PKsv <dbl>,
## #   PKm <dbl>, Overall <dbl>
```

A couple of players not in the dataset. That's not ideal, but we gotta make do.

```
gkData1718.df <- gkData1718 %>%
  dplyr::select(-FirstName, -LastName) %>%
  filter(Overall != 0)

gkData1718.df[is.na(gkData1718.df)] <- 0

gkPrem1819 <- gkPrem1819 %>%
  rename(Name = Player, Club = Squad) %>%
  separate(Name, c("FirstName", "LastName"), "(\\s)", extra = "merge", remove = FALSE) %>%
  mutate(LastName = ifelse(is.na(LastName), FirstName, LastName), FirstName = ifelse(FirstName==LastNam
  dplyr::select(Name, Club, FirstName, LastName, Age:PKm) %>%
  mutate(FirstName = substr(FirstName, 1, 1))
```

```
## Warning: Expected 2 pieces. Missing pieces filled with `NA` in 3 rows [1, 11,
## 14].
```

```
gkData1819.df <- gkPrem1819 %>%
  dplyr::select(-Name, -Club, -FirstName, -LastName)

N <- nrow(gkData1718.df)
boot <- sample(1:N,2*N,rep=T)
boot.df <- gkData1718.df[boot,] #boot the data so we have enough to do boosted regression


mod.gbm <- gbm(Overall  ~  .,
                   data=boot.df,
                   distribution="gaussian",
                   shrinkage=0.01,
                   n.trees=10,
                   interaction.depth = 10)

predOvr <- predict(mod.gbm,
                        newdata=gkData1819.df,
                        shrinkage=0.01,
                        n.trees=10,
```

```
                        interaction.depth = 10)

predOvr <- as.vector(predOvr)

gkPrem1819$predOvr <- predOvr

gkPreds1819 <- gkPrem1819 %>%
  mutate(Age = Age+1) %>%
  left_join(fifa20Vars.df, by = c("FirstName", "LastName", "Age")) %>%
  dplyr::select(-FirstName, -LastName, actualOverall = Overall) %>%
  filter(!is.na(actualOverall))


mean((gkPreds1819$actualOverall-gkPreds1819$predOvr)^2) #MSE is much worse
```

```
## [1] 50.07427
```

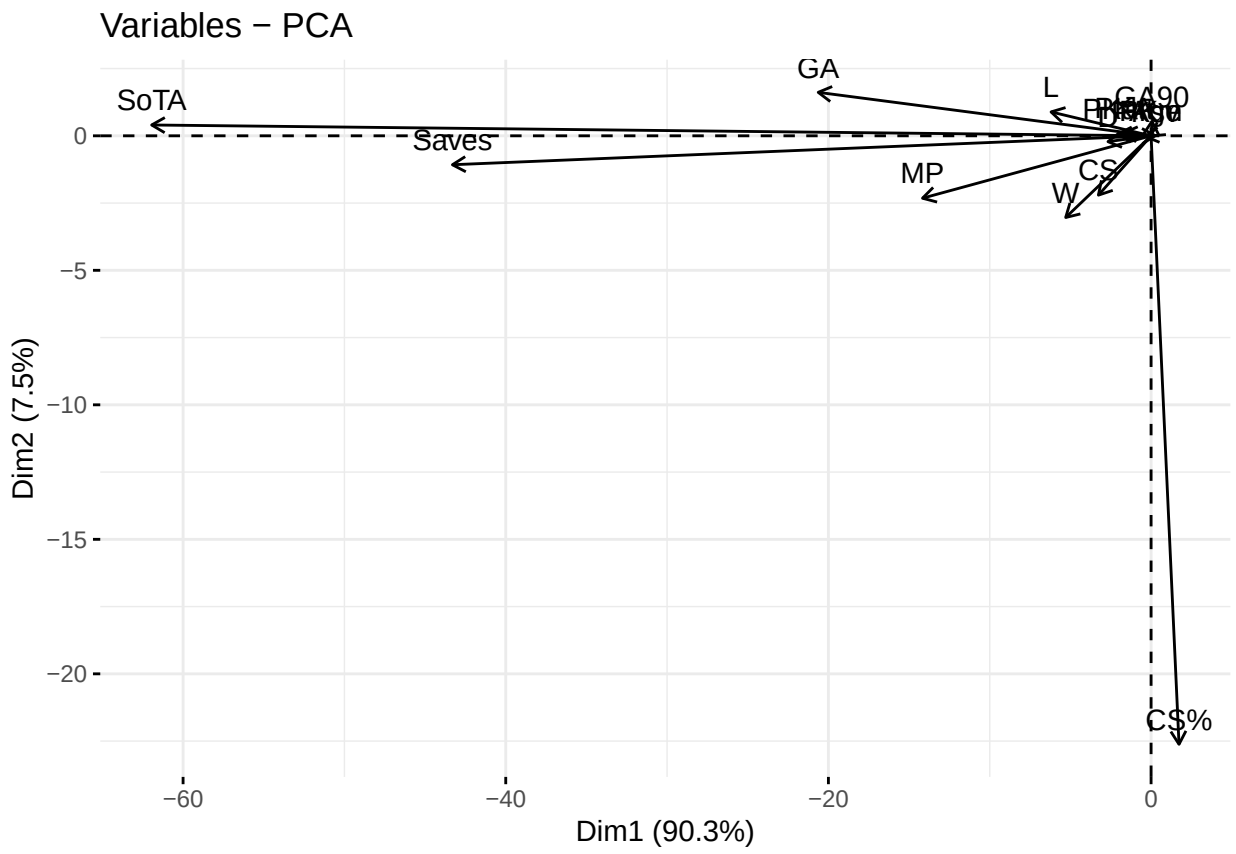
Unfortunately, our data was too small for us to do boosting regression freely. So, instead, we did a bootstrap to get a larger dataset, and used that as our training data. Compared to our outfielder models, our MSE is quite bad. But it's understandable, with such a small sampling size and far fewer predictors, we expected a worse model.

PCA

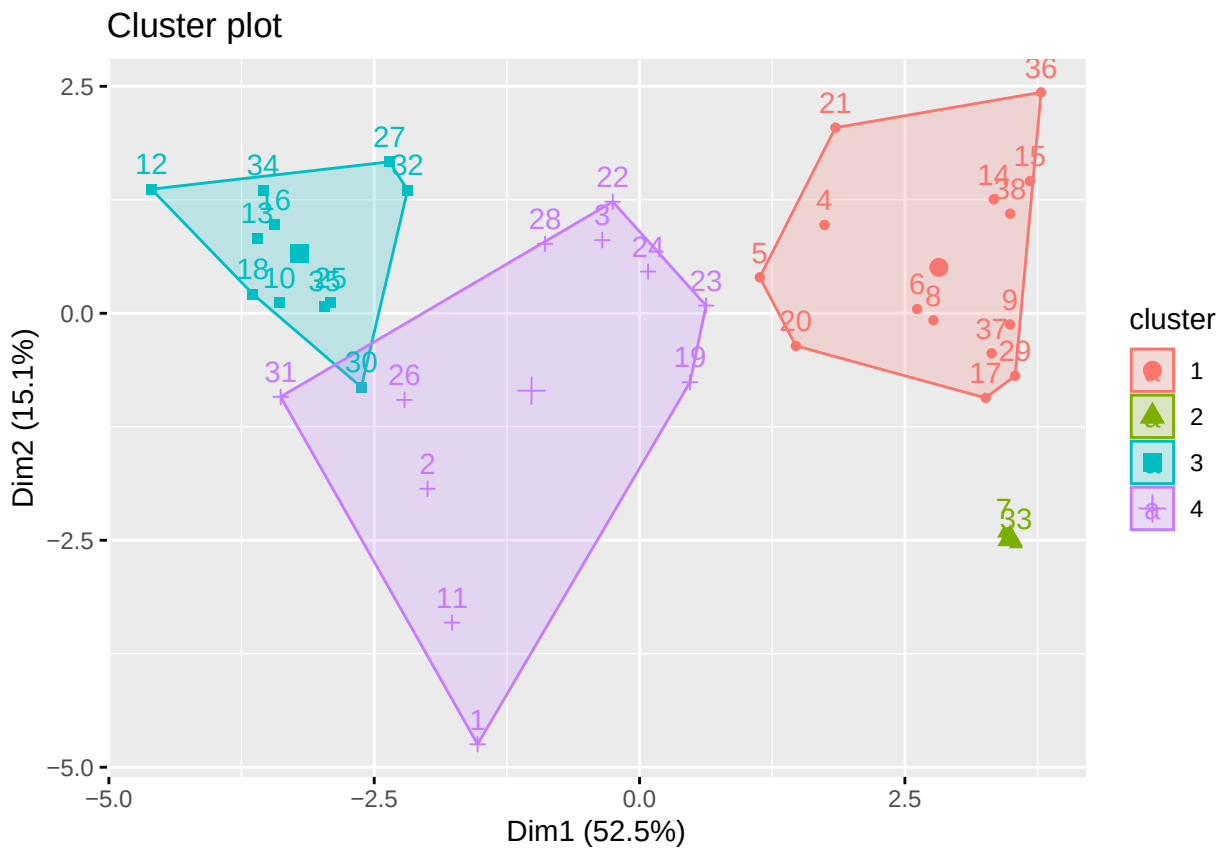
However, we decided to take advantage of the uncluttered environment given by a small sample size to do some unsupervised learning. Let's try to cluster the goalkeepers of the 2018-19 season using some principal component analysis!

```
gkData1819.df[is.na(gkData1819.df)] <- 0

K <- 4
mod.km <- kmeans(gkData1819.df,K,nstart=25)
mod.pc <- prcomp(gkData1819.df)
fviz_pca_var(mod.pc)
```




```
fviz_cluster(mod.km,data=gkData1819.df)
```



Roughy four clusters seem to show up well pretty easily. Let's add their names to their points and see who ends up where.

```
rot.mat <- mod.pc$rotation
gkData1819.mat <- as.matrix(gkData1819.df)
gkData1819.rot <-  gkData1819.mat%*% rot.mat
gkData1819Rot.df <- data.frame(gkData1819.rot)

mod.km2 <- kmeans(gkData1819Rot.df,K,nstart=25)
gkData1819Rot.df$cluster <- factor(mod.km$cluster)
names <- as.matrix(gkPrem1819)[,1]
gkData1819Rot.df$Name <- names

gkData1819Rot.df%>%
    ggplot()+
    geom_point(aes(PC1,PC2,color=cluster))+
    geom_text_repel(aes(PC1,PC2,color=cluster,label=Name),size=2.5)
```

That is an interesting image, and we can make some sense out of it The bottom right ones are two substitute keepers; they didn't play very many games at all, so it makes sense they're away from the rest. The red ones in the middle are some of the better keepers. Allison and Ederson, standing out from the rest, and often considered the two best keepers in the league, and they are quite similar. LLoris and Kepa are close as well, and are of the same ilk; consistently great, all rounded and playing with some of the best defenders as well. The purple keepers on the left, like De Gea, Ryan, Fabianski, are considered great shot-stoppers. Looking at the PCA breakdown, we see that they are aligned more with stats like shots against and saves, so it makes sense again. And finally, the top right corner are goalkeepers playing for worse teams, which means they have to go against really good forwards and are behind some of the less good defenders.

Future Work

There is a lot of room for future work in this project, especially in the second half with the Overall predictions. For starters, we are still missing many of the best players in the world! So one might (I might) begin by adding the other leagues in the future. Of course, the big challenge is to integrate the three datasets properly, with no incorrect information, but once the data cleaning part is one, everything else follows as we did earlier. Secondly, we could expand more into looking at goalkeepers. We could also make multiple models for other positions as well, and see if it improves our predictive power. Finally, we could do some sort of unsupervised learning methods for the outfielders as well, and perhaps all players overall. It would be interesting to see what different types and styles of players emerge, and if there are some truly remarkable outliers among even the best players (spoilers: Messi and Ronaldo are, quite frankly, the biggest possible outliers).

Conclusion

All in all, we believe we acheived what we set out to do. We did a thorough investigation of the market value in the FIFA 2018 game. We applied our full knowledge of all the models, did a lot of error analysis, and did everything for both categorical and numerical variables. And in the second part, we predicted the Overall stat in the FIFA 2020 game by linking it to the real-life performance statistics from 2018-19 (after training the FIFA 19 stats on 2017-18), and we did much better than we thought we would when we set out to do it. Naturally, there will be some things missing from the stats, especially if the training data is a single season, but as we add more stats and players and increase our scope, our predictive power would only get better and better. It would be interesting to see how well the model would perform if trained on the FIFA 20 dataset as well, and tested on the next game, when FIFA 21 comes out in September 2020!