# Project 2

*Devin Khosla, Jaejin Kim & Simon Grinsted*

*1/18/2019*

## TASK

At a minimum, your analysis should include:

1. Brief (1 paragraph) description of the data source and measurement scales associated with your feature set.
2. Any additional variables that have been measured, not part of the clustering, to which you wish to make comparisons. Good examples are demographics, such as gender, race, ses, region.
3. A brief (1 paragraph) outline of your analysis plan, which should include:

a. Any pre-processing transformations you will perform, and/or whether you will do the clustering on the original scale, transformed/ standardized scale, or use (a full set of) principle components.
b. Choice of clustering method (and rationale)
c. How you will determine the number of clusters (C(g) is only one approach, there is 1-Wilk's Lambda, BIC, e.g.)

4. The write up should go into more detail about the choice of the number of clusters, and then provide:

a. Graphical display of the clustering (e.g., dendogram, if you used a hierarchical approach) and "best" cluster solution (decide and justify)
b. Comparison to alternative choices for clustering (if any were explored)
c. Description of the clusters with respect to the feature set and the additional variables (demographics)
d. Any substantive conclusions you may be able to draw from the analysis (briefly)

```
library(foreign)
library(cluster)
library(NbClust)
library(klaR)
library(ggdendro)
library(GGally)
library(e1071)
library(caret)
library(knitr)
library(gridExtra)
library(foreign)
library(ggplot2)
library(reshape2)
library(randomForest)
library(NeuralNetTools)
library(dbarts)
```

## DATA SOURCE AND MEASUREMENT SCALES

Data analytics is being used more and more in sports by teams to gain any marginal advantage over their opponents. Teams with limited resources need to be able to indentify key needs and more importantly make sure they get good value for money. We have FIFA 19 data, containing information on over 18,000 real-life

soccer players and ratings of their physical and playing attributes. Using these attribute ratings, we will try to predict player wages. Each of these attributes are on a scale from 0 - 100 with a player receiving a score for each attribute. This should help us answer the question of whether players are over or under paid based solely on their playing ability. With this analysis, teams will be able to make informed decisions.

## DATA PREPARATION AND TRANSFORMATIONS

We start by loading in the data and removing the columns that we will not be using.

```
data <- read.csv("data.csv")
data <- data[,-c(1,2,5,7,11,14,17,18,19,20,21,23,29:54,89)]
```

There are some missing values in the data so we decide to remove these players as there is no data on them.

```
data <- na.omit(data)
```

We must adjust the Wage and Value columns into numerics. Previously they have currency symbols and 'K' to indicate thousands and 'M' to indicate millions. By removing them, we can then treat this data as numbers and not characters. This makes the data easier to work with later on.

```
data$Wage <- gsub("\u20AC","",data$Wage)
data$Wage <- gsub("K","e3",data$Wage)
data$Wage <- as.numeric(data$Wage)

data$Value <- gsub("\u20AC","",data$Value)
data$Value <- gsub("K","e3",data$Value)
data$Value <- gsub("M",'e6',data$Value)
data$Value <- as.numeric(data$Value)

data <- subset(data,data$Wage != 0)
```

We also remove players with wage of 0 since most of these are free agents who do not play for a team. This would likely affect our results if we used them and make it difficult to classify wages so we choose to ignore them.

```
table(data$Position)
```

```
##
##       CAM   CB  CDM   CF   CM   GK  LAM   LB  LCB  LCM  LDM   LF   LM   LS
##    0  948 1754  936   74 1377 1992   21 1305  637  389  239   15 1086  206
##   LW  LWB  RAM   RB  RCB  RCM  RDM   RF   RM   RS   RW  RWB   ST
##  374   78   21 1268  652  387  246   16 1114  201  365   87 2130
```

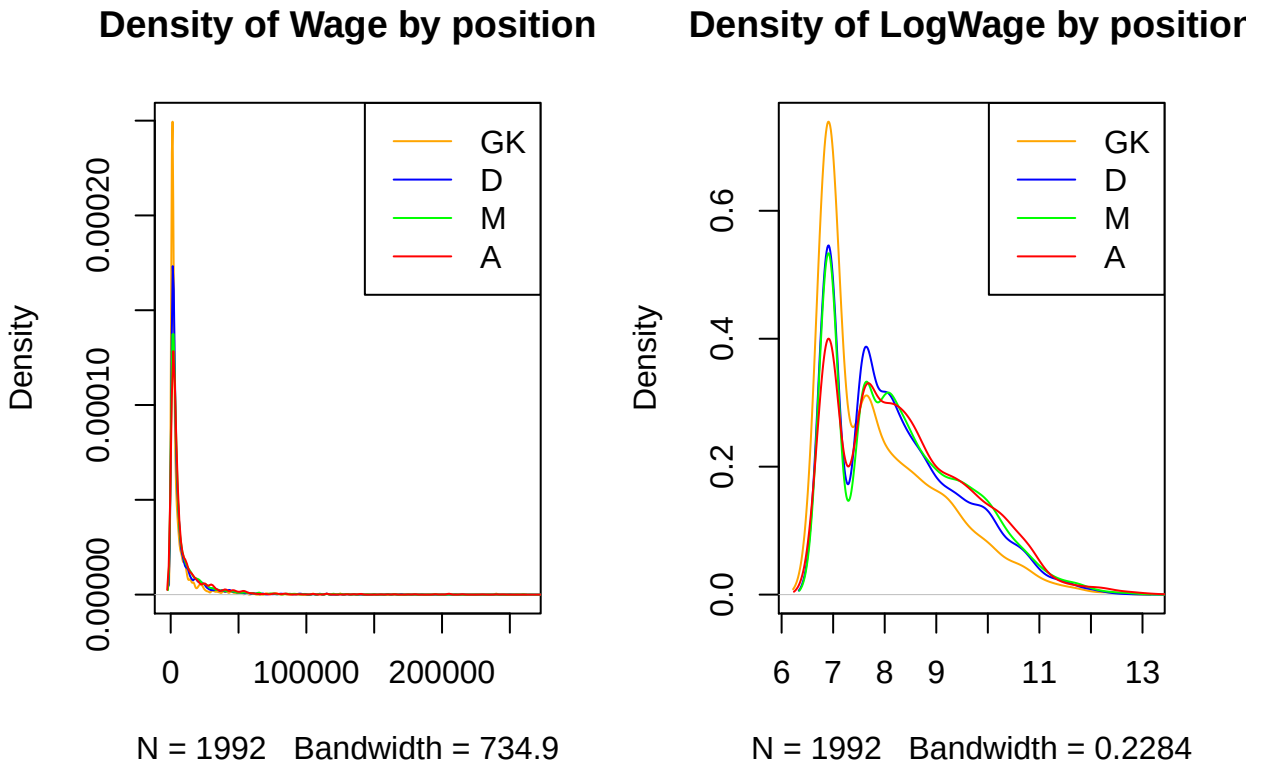
While there are a number of positions that players can play, we want to group these players into broader categories. We create 4 groups of players, Goalkeepers, Defenders, Midfielders and Attackers. We will examine these groups individually as we do not think it makes much sense to compare goalkeepers to attackers for example. Wages and attributes vary dramatically between these groups but hopefully the differences in attributes of players within the same groups will help predict their wages.

```
data$LogWage <- log(data$Wage)
data[17:50] <- sapply(data[17:50],as.numeric)

GK <- subset(data,data$Position == "GK")
D <- subset(data,data$Position == "CB" | data$Position == "LB" | data$Position == "LCB" |
              data$Position == "LWB" | data$Position == "RB" | data$Position == "RCB" |
              data$Position == "RWB")
M <- subset(data,data$Position == "CAM" | data$Position == "CDM" |
```

```
              data$Position == "CM" | data$Position == "LAM" | data$Position == "LCM" |
              data$Position == "LDM" | data$Position == "LM" | data$Position == "RAM" |
              data$Position == "RCB" | data$Position == "RCM" | data$Position == "RDM" |
              data$Position == "RM")
A <- subset(data,data$Position == "CF" | data$Position == "LF" | data$Position == "LS" |
              data$Position == "LW" | data$Position == "RF" | data$Position == "RS" |
              data$Position == "RW" | data$Position == "ST")
```

Additionally, we took the log of wage to help address the skew we see in our sample for each group.

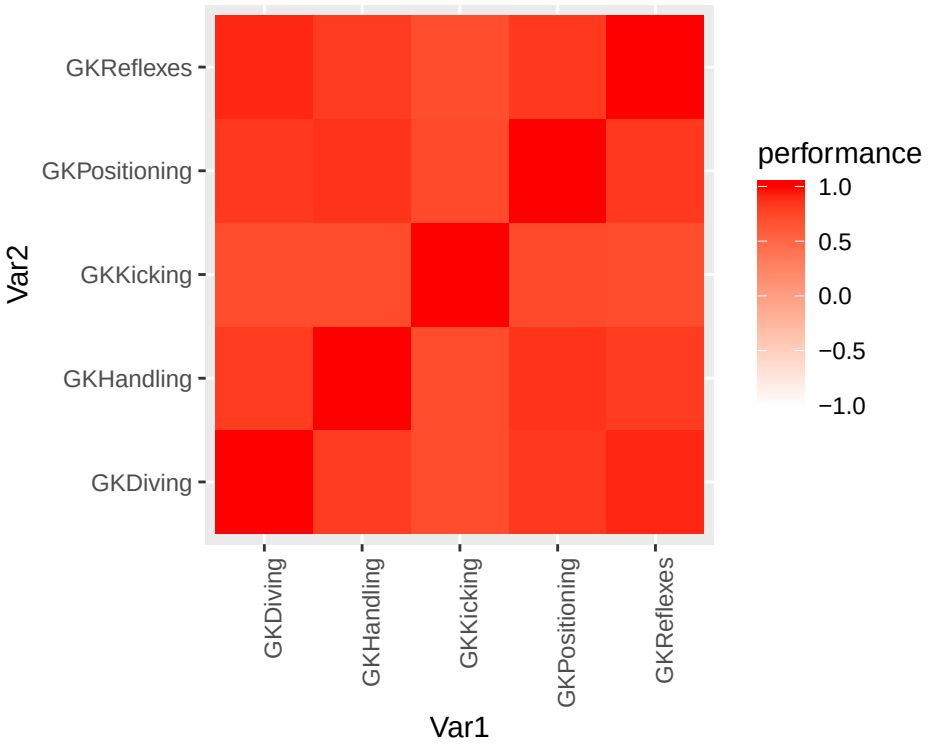
```
par(mfrow=c(1,2))
#Density wages for each group
plot(density(GK$Wage), col="orange", main="Density of Wage by position")
lines(density(D$Wage), col="blue")
lines(density(M$Wage), col="green")
lines(density(A$Wage), col="red")
legend("topright", legend = c("GK","D","M","A"), col = c("orange","blue","green","red"), lty = "solid")

#Density log wages for each group
plot(density(GK$LogWage), col="orange", main="Density of LogWage by position")
lines(density(D$LogWage), col="blue")
lines(density(M$LogWage), col="green")
lines(density(A$LogWage), col="red")
legend("topright", legend = c("GK","D","M","A"), col = c("orange","blue","green","red"), lty = "solid")
```



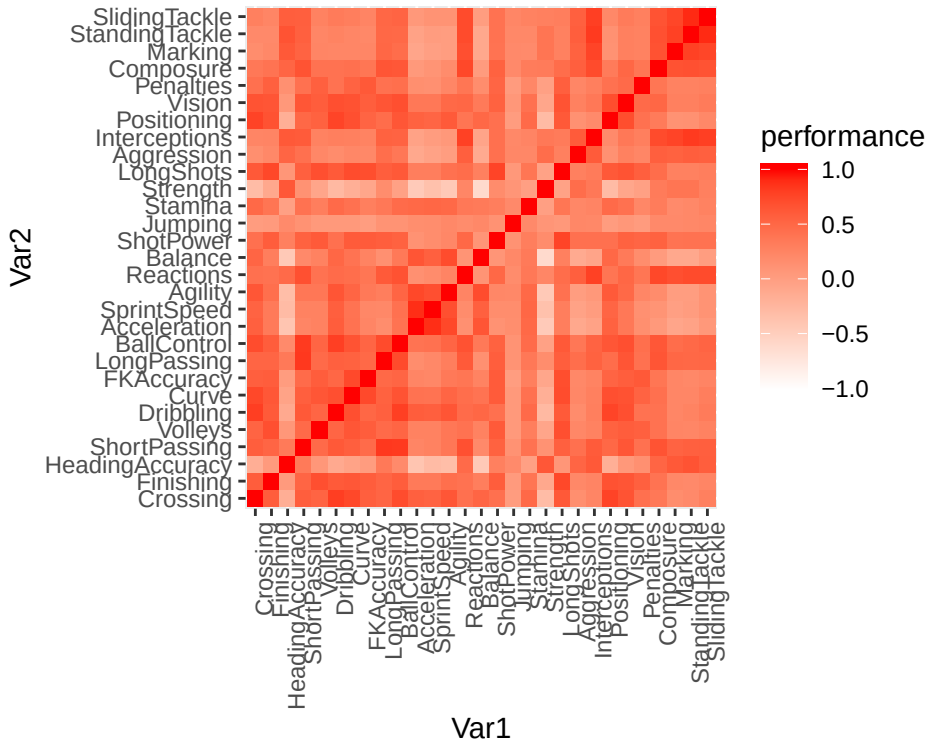

In our initial dataset, we have around 35 attributes which each player is graded on. There are 5 attributes that are designed specifically for goalkeeping attributes. We want to examine if there is any correlation between these attributes.

```
melted_cormat.GK <- melt(round(cor(GK[,46:50]),2))
```

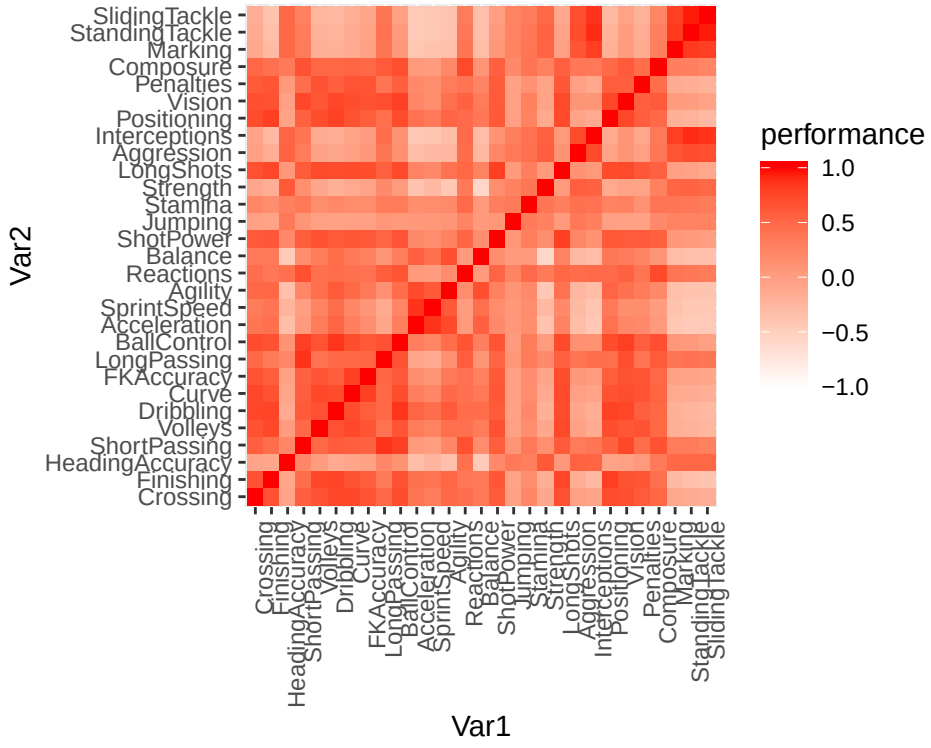
```
melted_cormat.D <- melt(round(cor(D[,17:45]),2))
melted_cormat.M <- melt(round(cor(M[,17:45]),2))
melted_cormat.A <- melt(round(cor(A[,17:45]),2))

ggplot(data = melted_cormat.GK, aes(x=Var1, y=Var2, fill=value))+
  geom_tile()+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  scale_fill_gradient('performance', limits=c(-1, 1),
                      breaks = c(-1, -0.5, 0, 0.5, 1),  low = "white", high = "red")
```



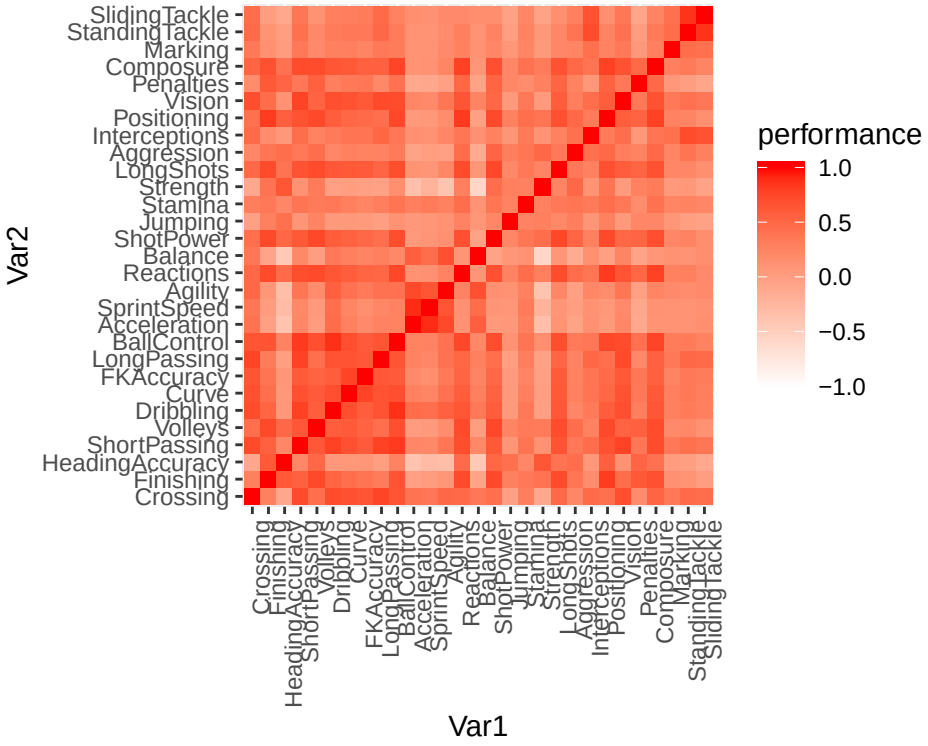

```
ggplot(data = melted_cormat.D, aes(x=Var1, y=Var2, fill=value))+
  geom_tile()+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  scale_fill_gradient('performance', limits=c(-1, 1),
                      breaks = c(-1, -0.5, 0, 0.5, 1),  low = "white", high = "red")
```




```
ggplot(data = melted_cormat.M, aes(x=Var1, y=Var2, fill=value))+
  geom_tile()+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  scale_fill_gradient('performance', limits=c(-1, 1),
                      breaks = c(-1, -0.5, 0, 0.5, 1),  low = "white", high = "red")
```



```
ggplot(data = melted_cormat.A, aes(x=Var1, y=Var2, fill=value))+
  geom_tile()+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  scale_fill_gradient('performance', limits=c(-1, 1),
                      breaks = c(-1, -0.5, 0, 0.5, 1),  low = "white", high = "red")
```




```
rm(melted_cormat.GK)
rm(melted_cormat.D)
rm(melted_cormat.M)
rm(melted_cormat.A)
```

Unsurprisingly there are a number of features that correlate with each other. For example, Ball control and Short passing are highly correlated. Since we also have a fairly large dataset, it makes sense for us to do some engineering on our features. We create 6 new features that will consolidate attributes that are likely to be correlated with each other. These new features are Shooting, Passing, Dribbling, Defending, Physical and Pace.

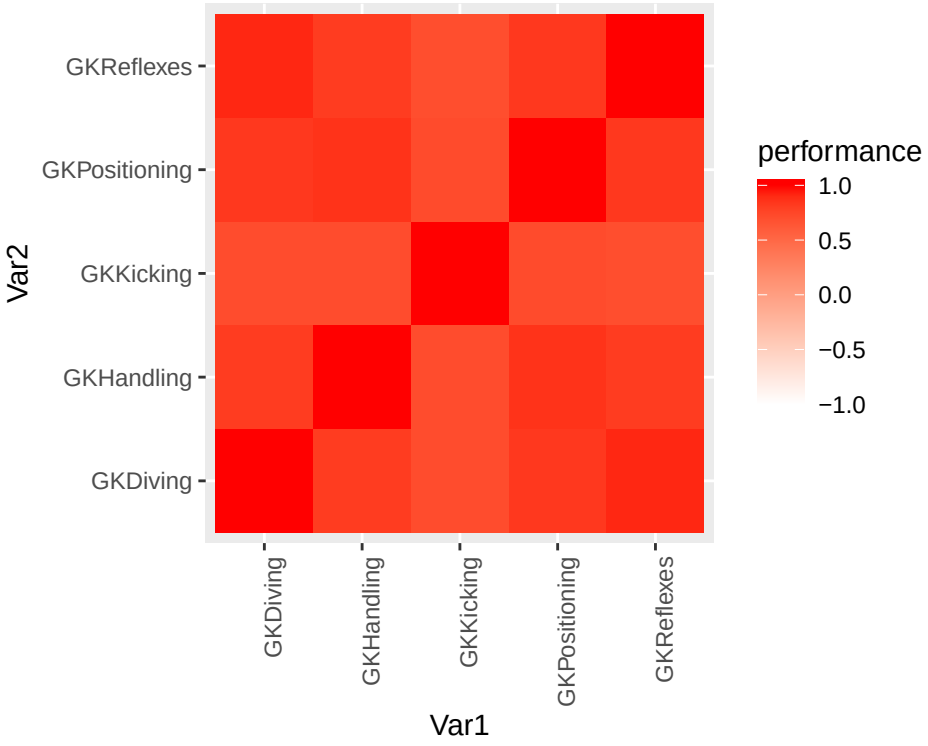
```
data$Shooting <- rowMeans(data[c("Finishing","Volleys","ShotPower","LongShots",
                                 "HeadingAccuracy","Curve","FKAccuracy",
                                 "Penalties","Composure")])
data$Passing <- rowMeans(data[c("Crossing","ShortPassing","LongPassing",
                                "Vision")])
data$Dribbling <- rowMeans(data[c("Dribbling","BallControl")])
data$Defending <- rowMeans(data[c("StandingTackle","SlidingTackle","Marking",
                                  "Interceptions","Positioning")])
data$Physical <- rowMeans(data[c("Reactions","Balance","Jumping","Stamina",
                                 "Strength","Aggression")])
data$Pace <- rowMeans(data[c("Acceleration","SprintSpeed","Agility")])
```

```
GK <- subset(data,data$Position == "GK")
D <- subset(data,data$Position == "CB" | data$Position == "LB" | data$Position == "LCB" |
```

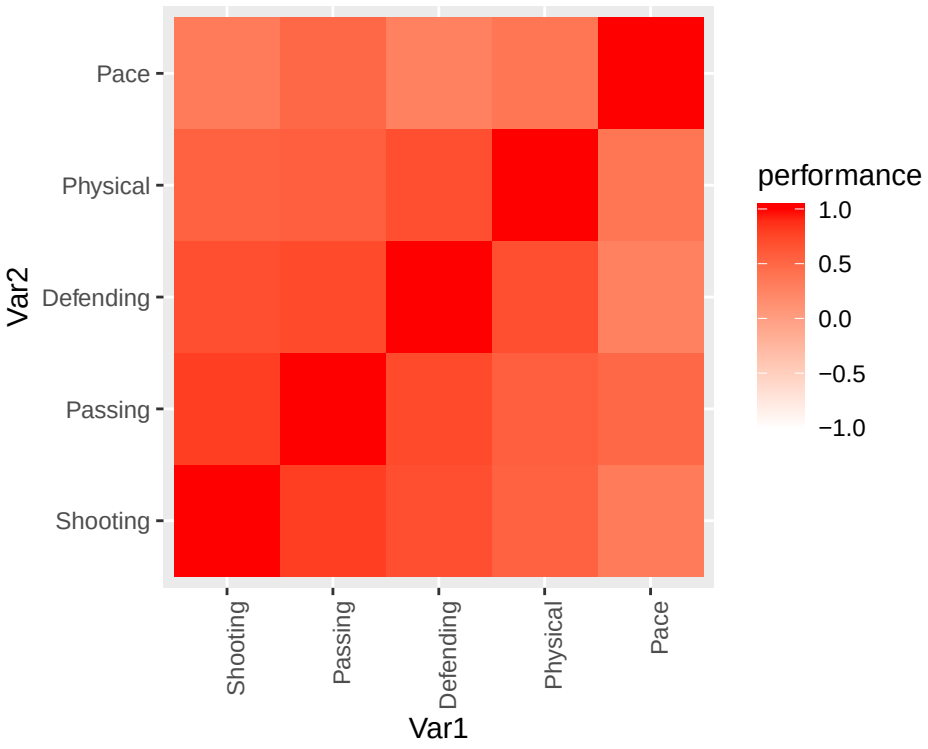
```
              data$Position == "LWB" | data$Position == "RB" | data$Position == "RCB" |
              data$Position == "RWB")
M <- subset(data,data$Position == "CAM" | data$Position == "CDM" |
              data$Position == "CM" | data$Position == "LAM" | data$Position == "LCM" |
              data$Position == "LDM" | data$Position == "LM" | data$Position == "RAM" |
              data$Position == "RCB" | data$Position == "RCM" | data$Position == "RDM" |
              data$Position == "RM")
A <- subset(data,data$Position == "CF" | data$Position == "LF" | data$Position == "LS" |
              data$Position == "LW" | data$Position == "RF" | data$Position == "RS" |
              data$Position == "RW" | data$Position == "ST")
```

We also want to see if there are still high correlations between these new features for each group of player.

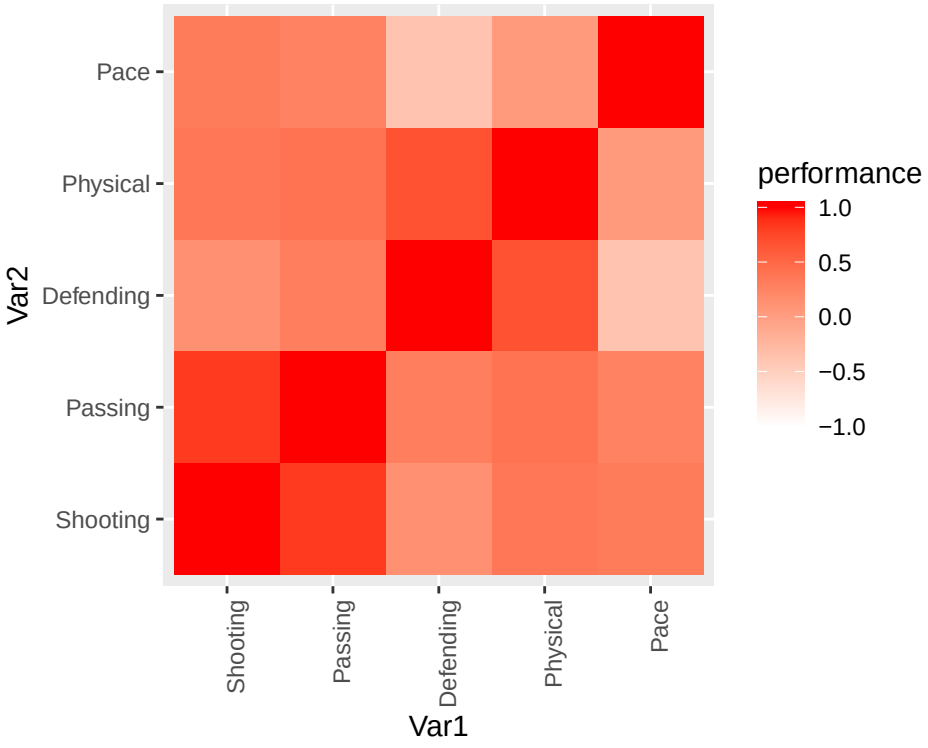
```
melted_cormat.GK <- melt(round(cor(GK[,46:50]),2))
melted_cormat.D <- melt(round(cor(D[,52:56]),2))
melted_cormat.M <- melt(round(cor(M[,52:56]),2))
melted_cormat.A <- melt(round(cor(A[,52:56]),2))

ggplot(data = melted_cormat.GK, aes(x=Var1, y=Var2, fill=value))+
  geom_tile()+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  scale_fill_gradient('performance', limits=c(-1, 1),
                      breaks = c(-1, -0.5, 0, 0.5, 1),  low = "white", high = "red")
```



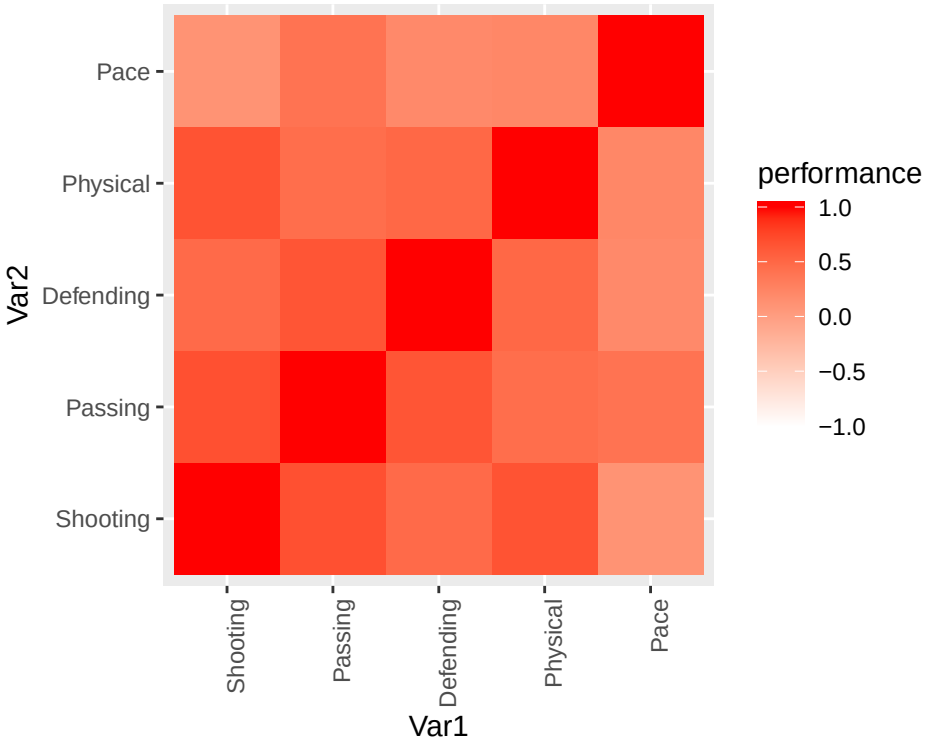

```
ggplot(data = melted_cormat.D, aes(x=Var1, y=Var2, fill=value))+
  geom_tile()+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  scale_fill_gradient('performance', limits=c(-1, 1),
                      breaks = c(-1, -0.5, 0, 0.5, 1),  low = "white", high = "red")
```




```
ggplot(data = melted_cormat.M, aes(x=Var1, y=Var2, fill=value))+
  geom_tile()+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  scale_fill_gradient('performance', limits=c(-1, 1),
                      breaks = c(-1, -0.5, 0, 0.5, 1),  low = "white", high = "red")
```



```
ggplot(data = melted_cormat.A, aes(x=Var1, y=Var2, fill=value))+
  geom_tile()+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  scale_fill_gradient('performance', limits=c(-1, 1),
                      breaks = c(-1, -0.5, 0, 0.5, 1),  low = "white", high = "red")
```




```
rm(melted_cormat.GK)
rm(melted_cormat.D)
rm(melted_cormat.M)
rm(melted_cormat.A)
```

We find that while there is still some correlation, it is greatly reduced. Since Goalkeepers only have 5 attributes, we cannot do much feature engineering on them.

```
cutGK2 <- round(c(999, 2000, 260001), 1)
cutD2 <- round(c(999, 3000, 380001), 1)
cutM2 <- round(c(999, 3000, 420001), 1)
cutA2 <- round(c(999, 4000, 565001), 1)

GK$Wagecode2 <- cut(GK$Wage, breaks=cutGK2, labels=c("Low", "High"))
D$Wagecode2 <- cut(D$Wage, breaks=cutD2, labels=c("Low", "High"))
M$Wagecode2 <- cut(M$Wage, breaks=cutM2, labels=c("Low", "High"))
A$Wagecode2 <- cut(A$Wage, breaks=cutA2, labels=c("Low", "High"))

rm(cutGK2)
rm(cutD2)
rm(cutM2)
rm(cutA2)
```

We also want to create two groups of players and assign them as low and high wage players. This will be helpful later on for splitting the data into train and test splits.

# ANALYSIS PLAN

We will compare a number of different approaches in our analysis. Our initial plan is to create baseline linear models and compare these results to SVM and Random Forest. We expect that since this is a regression problem, Random Forests will not perform as well as the linear models. SVM may be able to perform well assuming we use the kernel trick appropriately.

# LINEAR REGRESSION

As our baseline, we will try to run a linear regression to predict Log wages based on the physical attributes of the players. We include the specific position they play as a factor variable. We also include age and international reputation, as we expect these variables to be relevant.

```
LMD <- lm(LogWage~factor(Position)+Age+Crossing+Finishing+HeadingAccuracy+ShortPassing
          +Volleys+Dribbling+Curve+FKAccuracy+LongPassing+BallControl+Acceleration
          +SprintSpeed+Agility+Reactions+Balance+ShotPower+Jumping+Stamina+Strength
          +LongShots+Aggression+Interceptions+Positioning+Vision+Penalties+Composure
          +Marking+StandingTackle+SlidingTackle+International.Reputation,data=D)
summaryLMD <- summary(LMD)
summaryLMD$r.squared
```

```
## [1] 0.6605942
```

```
summaryLMD$adj.r.squared
```

```
## [1] 0.6584076
```

```
rm(LMD)
rm(summaryLMD)
```

```
LMM <- lm(LogWage~factor(Position)+Age+Crossing+Finishing+HeadingAccuracy+ShortPassing
          +Volleys+Dribbling+Curve+FKAccuracy+LongPassing+BallControl+Acceleration
          +SprintSpeed+Agility+Reactions+Balance+ShotPower+Jumping+Stamina+Strength
          +LongShots+Aggression+Interceptions+Positioning+Vision+Penalties+Composure
          +Marking+StandingTackle+SlidingTackle+International.Reputation,data=M)
summaryLMM <- summary(LMM)
summaryLMM$r.squared
```

```
## [1] 0.6266567
```

```
summaryLMM$adj.r.squared
```

```
## [1] 0.62453
```

```
rm(LMM)
rm(summaryLMM)
```

```
LMA <- lm(LogWage~factor(Position)+Age+Crossing+Finishing+HeadingAccuracy+ShortPassing
          +Volleys+Dribbling+Curve+FKAccuracy+LongPassing+BallControl+Acceleration
          +SprintSpeed+Agility+Reactions+Balance+ShotPower+Jumping+Stamina+Strength
          +LongShots+Aggression+Interceptions+Positioning+Vision+Penalties+Composure
          +Marking+StandingTackle+SlidingTackle+International.Reputation,data=A)
summaryLMA <- summary(LMA)
summaryLMA$r.squared
```

```
## [1] 0.6892528
```

```
summaryLMA$adj.r.squared
```

```
## [1] 0.6857195
```

```
rm(LMA)
rm(summaryLMA)
```

For the Goalkeepers, we look at a much smaller group of attributes to determine their wages. This is because the attributes that pertain to goalkeepers is a smaller set.

```
LMGK <- lm(LogWage~Age+GKDiving+GKHandling+GKKicking+GKPositioning+GKReflexes
           +International.Reputation,data=GK)
summaryLMGK <- summary(LMGK)
summaryLMGK$r.squared
```

```
## [1] 0.6829777
```

```
summaryLMGK$adj.r.squared
```

```
## [1] 0.6818592
```

```
rm(LMGK)
rm(summaryLMGK)
```

We choose to look at the R-squared and adjusted R-squared in order to evaluate these regression models. The R-squared for each group is somewhat similar but we find that the Goalkeepers models perform the best (0.683, meaning that 68.3% of the variation we see in LogWage is explained by the selected physical attributes) and the Midfielders model worst (0.627). One concern we had that with such a large number of features, this may negatively effect the adjusted R-squared for each model. However we find that the adjusted R-squared is only slightly lower than the R-squared.

Since we cannot use such a high number of for our other methods, we also choose to look at our generated features to see how a linear model would perform with fewer features.

```
LMGK <- lm(LogWage~Age+GKDiving+GKHandling+GKKicking+GKPositioning+GKReflexes,data=GK)
summaryLMGK <- summary(LMGK)
summaryLMGK$r.squared
```

```
## [1] 0.6617039
```

```
summaryLMGK$adj.r.squared
```

```
## [1] 0.6606813
```

```
rm(LMGK)
rm(summaryLMGK)
```

```
LMD <- lm(LogWage~Age+Shooting+Passing+Defending+Physical+Pace+Dribbling
          +factor(Position),data=D)
summaryLMD <- summary(LMD)
summaryLMD$r.squared
```

```
## [1] 0.6124551
```

```
summaryLMD$adj.r.squared
```

```
## [1] 0.6115815
```

```
rm(LMD)
rm(summaryLMD)
```

```
LMM <- lm(LogWage~Age+Shooting+Passing+Defending+Physical+Pace+Dribbling
          +factor(Position),data=M)
summaryLMM <- summary(LMM)
summaryLMM$r.squared
```

```
## [1] 0.574176
```

```
summaryLMM$adj.r.squared
```

```
## [1] 0.5731398
```

```
rm(LMM)
rm(summaryLMM)
```

```
LMA <- lm(LogWage~Age+Shooting+Passing+Defending+Physical+Pace+Dribbling
          +factor(Position),data=A)
summaryLMA <- summary(LMA)
summaryLMA$r.squared
```

```
## [1] 0.6465265
```

```
summaryLMA$adj.r.squared
```

```
## [1] 0.6450564
```

```
rm(LMA)
rm(summaryLMA)
```

We find that the R-squared and adjusted R-Squared actually fell when using our features. While the change is not dramatic, we would still recommend using the model with the entire set of features.

# SVM

Now, we split our data into 70% training and 30% test data sets to prepare for our next approaches. The first of these is SVM with cross validation.

```
TC <- trainControl(method = "repeatedcv",number = 10,repeats = 20)

set.seed(2011)
GK.inTrain <- createDataPartition(y = GK$Wagecode2, p = 0.7, list = FALSE)
D.inTrain <- createDataPartition(y = D$Wagecode2, p = 0.7, list = FALSE)
M.inTrain <- createDataPartition(y = M$Wagecode2, p = 0.7, list = FALSE)
A.inTrain <- createDataPartition(y = A$Wagecode2, p = 0.7, list = FALSE)

GK.training <- GK[GK.inTrain,]
D.training <- D[D.inTrain,]
M.training <- M[M.inTrain,]
A.training <- A[A.inTrain,]

GK.testing <- GK[-GK.inTrain,]
D.testing <- D[-D.inTrain,]
M.testing <- M[-M.inTrain,]
A.testing <- A[-A.inTrain,]

rm(GK.inTrain)
rm(D.inTrain)
```

```
rm(M.inTrain)
rm(A.inTrain)
```

```
GK.svm <- train(LogWage~Age+GKDiving+GKHandling+GKKicking+GKPositioning+GKReflexes,
                data = GK.training, method = "svmRadial",trControl = TC)
svm.pred <- predict(GK.svm,GK.testing)
print(GK.svm,showSD = T)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 1395 samples
##    6 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 20 times)
## Summary of sample sizes: 1256, 1255, 1257, 1254, 1255, 1256, ...
## Resampling results across tuning parameters (values below are 'mean (sd)'):
##
##   C     RMSE                    Rsquared
##   0.25  0.6470310 (0.05916900)  0.6869261 (0.06041631)
##   0.50  0.6413464 (0.05799140)  0.6924843 (0.05986453)
##   1.00  0.6384377 (0.05640299)  0.6958803 (0.05904400)
##   MAE
##   0.4534222 (0.03692062)
##   0.4503479 (0.03673393)
##   0.4486915 (0.03635083)
##
## Tuning parameter 'sigma' was held constant at a value of 0.265762
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.265762 and C = 1.
```

```
#xtabs(~svm.pred+GK.testing$Wagecode2)
cor(svm.pred,GK.testing$LogWage)
```

```
## [1] 0.8558792
```

```
rm(GK.svm)
rm(svm.pred)
```

```
D.svm <- train(LogWage~Age+Shooting+Passing+Defending+Physical+Pace+Dribbling,
               data = D.training, method = "svmRadial",trControl = TC)
svm.pred <- predict(D.svm,D.testing)
print(D.svm,showSD = T)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 4048 samples
##    7 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 20 times)
## Summary of sample sizes: 3643, 3643, 3644, 3643, 3643, 3642, ...
## Resampling results across tuning parameters (values below are 'mean (sd)'):
##
##   C     RMSE                    Rsquared
##   0.25  0.7300568 (0.03902976)  0.6351894 (0.04031282)
```

```
##   0.50  0.7283779 (0.03943715)  0.6377839 (0.04037142)
##   1.00  0.7301597 (0.03980662)  0.6372107 (0.04042501)
##   MAE
##   0.5545872 (0.02333652)
##   0.5525148 (0.02327916)
##   0.5530109 (0.02353189)
##
## Tuning parameter 'sigma' was held constant at a value of 0.1646301
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.1646301 and C = 0.5.
```

```
#xtabs(~svm.pred+D.testing$Wagecode2)
cor(svm.pred,D.testing$LogWage)
```

```
## [1] 0.8265273
```

```
rm(D.svm)
rm(svm.pred)
```

```
M.svm <- train(LogWage~Age+Shooting+Passing+Defending+Physical+Pace+Dribbling,
               data = M.training, method = "svmRadial",trControl = TC)
svm.pred <- predict(M.svm,M.testing)
print(M.svm,showSD = T)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 5192 samples
##    7 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 20 times)
## Summary of sample sizes: 4673, 4673, 4673, 4673, 4673, 4673, ...
## Resampling results across tuning parameters (values below are 'mean (sd)'):
##
##   C     RMSE                    Rsquared
##   0.25  0.7328295 (0.03675572)  0.6570909 (0.03654774)
##   0.50  0.7302628 (0.03788199)  0.6606787 (0.03686425)
##   1.00  0.7310867 (0.03843393)  0.6610244 (0.03694631)
##   MAE
##   0.5423806 (0.01998442)
##   0.5391555 (0.02067401)
##   0.5396711 (0.02121733)
##
## Tuning parameter 'sigma' was held constant at a value of 0.1500653
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.1500653 and C = 0.5.
```

```
#xtabs(~svm.pred+M.testing$Wagecode2)
cor(svm.pred,M.testing$LogWage)
```

```
## [1] 0.8163914
```

```
rm(M.svm)
rm(svm.pred)
```

```
A.svm <- train(LogWage~Age+Shooting+Passing+Defending+Physical+Pace+Dribbling,
               data = A.training, method = "svmRadial",trControl = TC)
```

```
svm.pred <- predict(A.svm,A.testing)
print(A.svm,showSD = T)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 2367 samples
##    7 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 20 times)
## Summary of sample sizes: 2131, 2129, 2129, 2131, 2130, 2130, ...
## Resampling results across tuning parameters (values below are 'mean (sd)'):
##
##   C     RMSE                    Rsquared
##   0.25  0.7261699 (0.04185857)  0.6677454 (0.03606708)
##   0.50  0.7180732 (0.04080231)  0.6756229 (0.03629773)
##   1.00  0.7185391 (0.04028091)  0.6757356 (0.03631353)
##   MAE
##   0.5620763 (0.02978583)
##   0.5551022 (0.02925622)
##   0.5548305 (0.02902604)
##
## Tuning parameter 'sigma' was held constant at a value of 0.1529924
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.1529924 and C = 0.5.
```

```
#xtabs(~svm.pred+A.testing$Wagecode2)
cor(svm.pred,A.testing$LogWage)
```

```
## [1] 0.816086
```

```
rm(A.svm)
rm(svm.pred)
```

We observe that the Defenders, Midfielders and Attackers return an RMSE in the range 0.7-0.74. Goalkeepers are lower at approximately 0.64. Goalkeepers therefore have a higher R-squared than the other positions, at 0.71 vs. approximately 0.66.

We see that our SVM predictions have a correlation of approximately 0.8 with LogWage across all groups. In line with our earlier results, this is strongest for GKs at 0.832. It is weakest for Defenders at 0.801.

# RANDOM FOREST

We now run random forest predictions on each grouping. We run this with 1000 trees on our training data.

```
TC <- trainControl(method = "cv", number = 10, search = "grid")
```

```
GK.rf <- caret::train(LogWage~Age+GKDiving+GKHandling+GKKicking+GKPositioning+GKReflexes,
                      data = GK.training, method = "rf",trControl = TC, ntree = 1000,
                      importance=TRUE)
rf.pred <- predict(GK.rf,GK.testing)
print(GK.rf)
```

```
## Random Forest
##
```

```
## 1395 samples
##    6 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1256, 1256, 1255, 1255, 1255, 1255, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE       Rsquared   MAE
##   2     0.6330627  0.6948289  0.4583829
##   4     0.6409615  0.6874203  0.4633508
##   6     0.6440176  0.6844769  0.4653365
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
```

```
rm(GK.rf)
rm(rf.pred)
```

```
D.rf <- caret::train(LogWage~Age+Shooting+Passing+Defending+Physical+Pace+Dribbling,
                     data = D.training, method = "rf",trControl = TC, ntree = 1000,
                     importance=TRUE)
rf.pred <- predict(D.rf,D.testing)
print(D.rf)
```

```
## Random Forest
##
## 4048 samples
##    7 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3644, 3643, 3643, 3643, 3644, 3643, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE       Rsquared   MAE
##   2     0.7352799  0.6283087  0.5691326
##   4     0.7392350  0.6237805  0.5696028
##   7     0.7463143  0.6168580  0.5740098
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
```

```
rm(D.rf)
rm(rf.pred)
```

```
M.rf <- caret::train(LogWage~Age+Shooting+Passing+Defending+Physical+Pace+Dribbling,
                     data = M.training, method = "rf",trControl = TC, ntree = 1000,
                     importance=TRUE)
rf.pred <- predict(M.rf,M.testing)
print(M.rf)
```

```
## Random Forest
##
## 5192 samples
##    7 predictor
```

```
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4671, 4673, 4673, 4674, 4673, 4672, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE       Rsquared   MAE
##   2     0.7331997  0.6549472  0.5544088
##   4     0.7377848  0.6501275  0.5567624
##   7     0.7437617  0.6445139  0.5607377
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
```

```
rm(M.rf)
rm(rf.pred)
```

```
A.rf <- caret::train(LogWage~Age+Shooting+Passing+Defending+Physical+Pace+Dribbling,
                     data = A.training, method = "rf",trControl = TC, ntree = 1000,
                     importance=TRUE)
rf.pred <- predict(A.rf,A.testing)
print(A.rf)
```

```
## Random Forest
##
## 2367 samples
##    7 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2130, 2130, 2129, 2130, 2130, 2130, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE       Rsquared   MAE
##   2     0.7194132  0.6730374  0.5677311
##   4     0.7250697  0.6668667  0.5697009
##   7     0.7306916  0.6614921  0.5740812
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
```

```
rm(A.rf)
rm(rf.pred)
rm(TC)
```

Our optimal models for each group return similar RMSE and R-squared results for random forest as they did for SVM. We see that the Defenders, Midfielders and Attackers return RMSEs in the range 0.71-0.74 with R-squareds between 0.62-0.68. Again, our peformance for GKs is slightly better, with a RMSE of 0.63 and an R-squared of 0.71 in the optimal model.

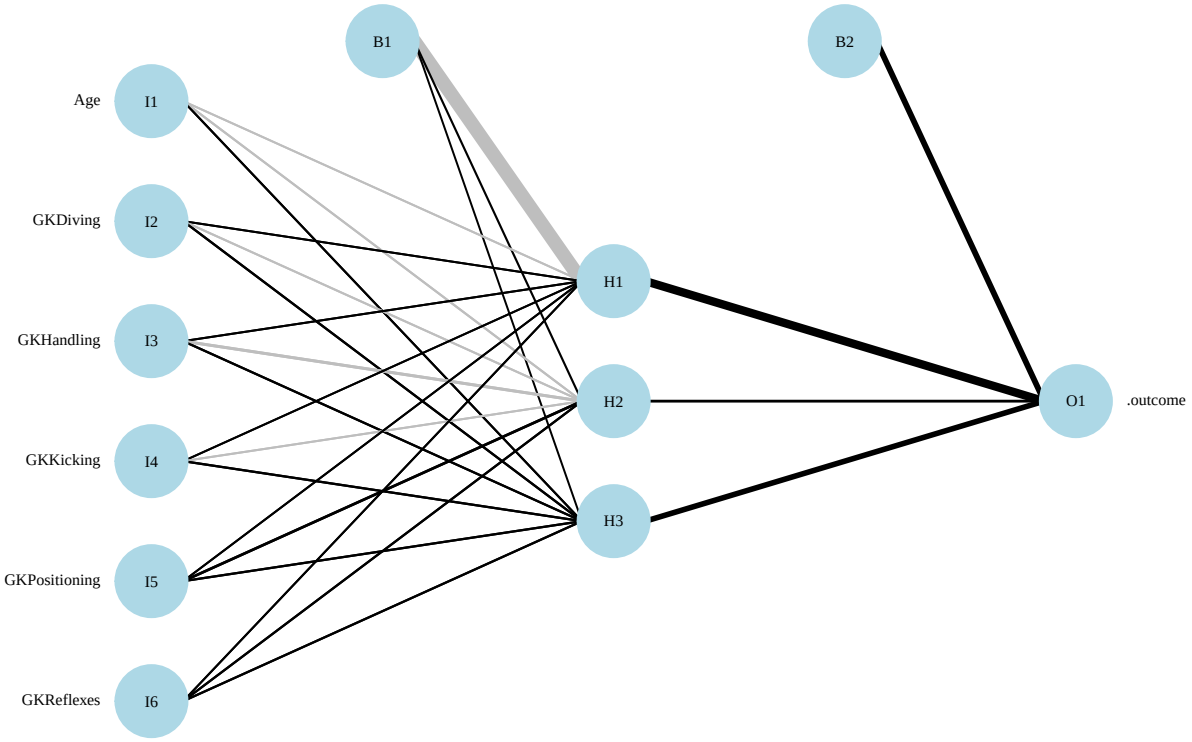
# NEURAL NETS

```
TC <- trainControl(method = "cv", number = 10, search = "grid")
```

```
GK.nnet <- train(LogWage ~ Age + GKDiving + GKHandling + GKKicking + GKPositioning
                 + GKReflexes, data = GK.training, method = "nnet", metric = "RMSE",
                 trControl = TC, importance = TRUE, linout = 1, trace = FALSE)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
## trainInfo, : There were missing values in resampled performance measures.
```

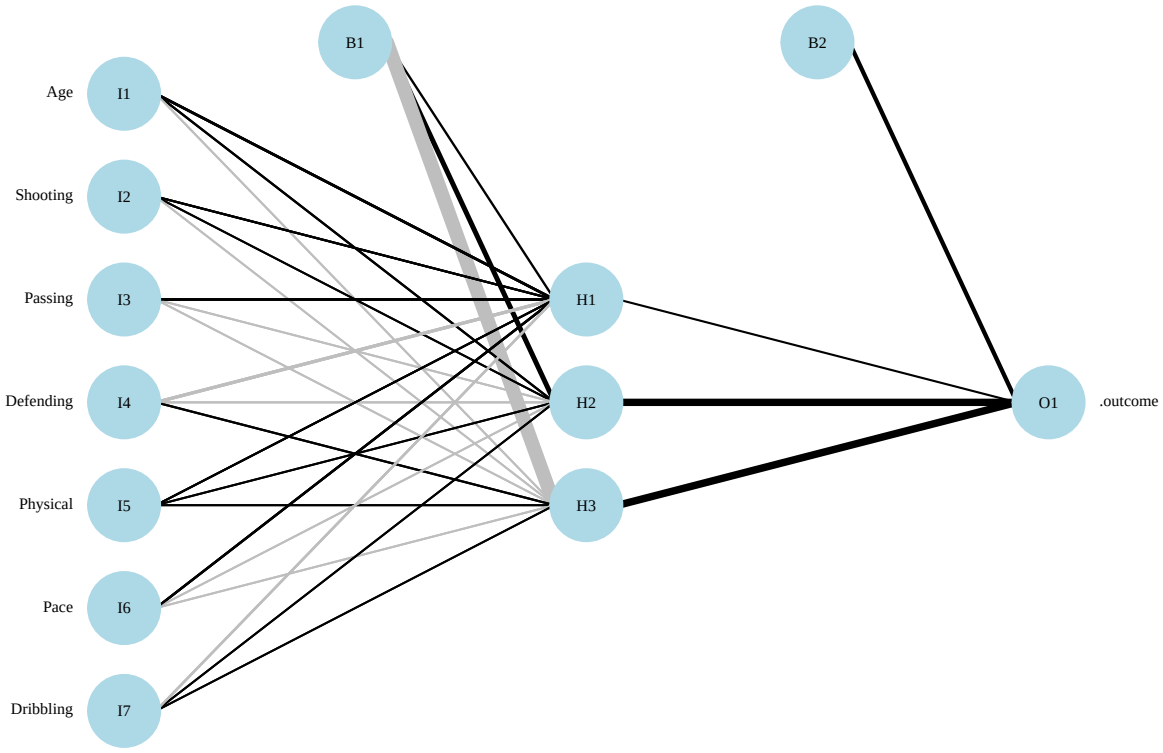
```
pred.GK.nnet <- predict(GK.nnet, GK.testing)
par(mar=numeric(4),family='serif')
plotnet(GK.nnet$finalModel, cex_val=.5)
```




```
D.nnet <- train(LogWage ~ Age+Shooting+Passing+Defending+Physical+Pace+Dribbling,
                data = D.training, method = "nnet", metric = "RMSE", trControl = TC,
                importance=TRUE, linout = 1, trace = FALSE)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
## trainInfo, : There were missing values in resampled performance measures.
```

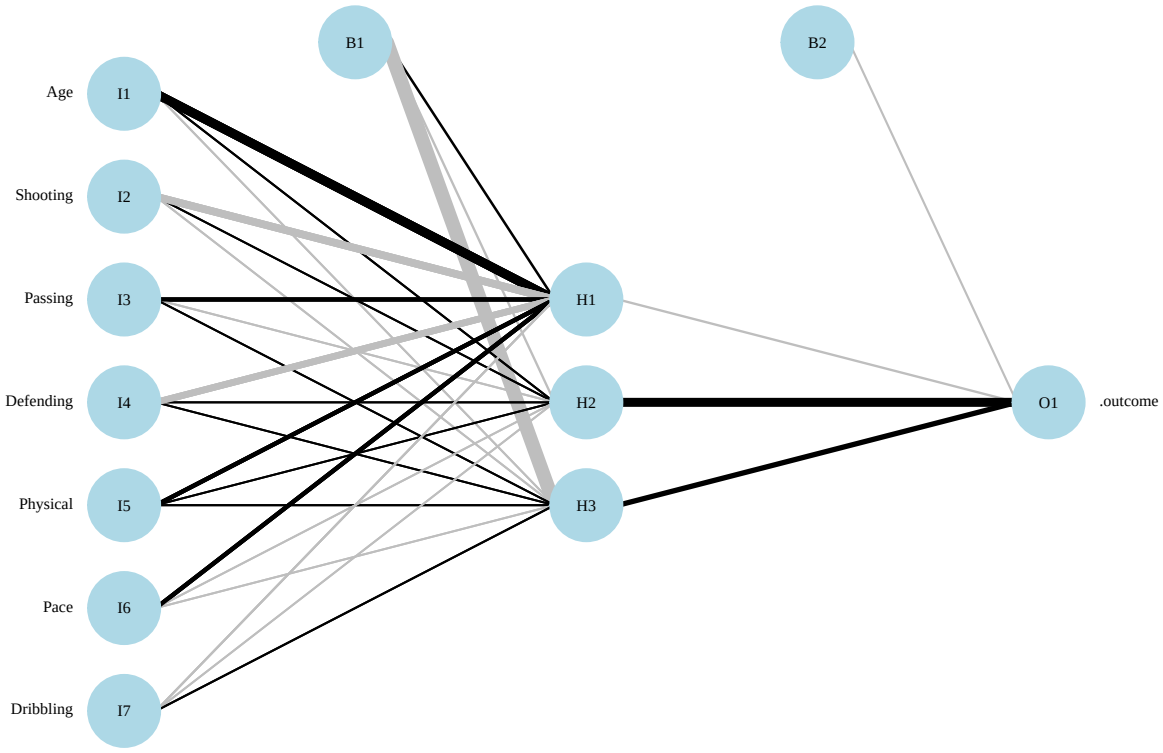
```
pred.D.nnet <- predict(D.nnet, D.testing)
par(mar=numeric(4),family='serif')
plotnet(D.nnet$finalModel, cex_val=.5)
```




```
M.nnet <- train(LogWage ~ Age+Shooting+Passing+Defending+Physical+Pace+Dribbling,
                data = M.training, method = "nnet", metric = "RMSE", trControl = TC,
                importance=TRUE, linout = 1, trace = FALSE)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
## trainInfo, : There were missing values in resampled performance measures.
```

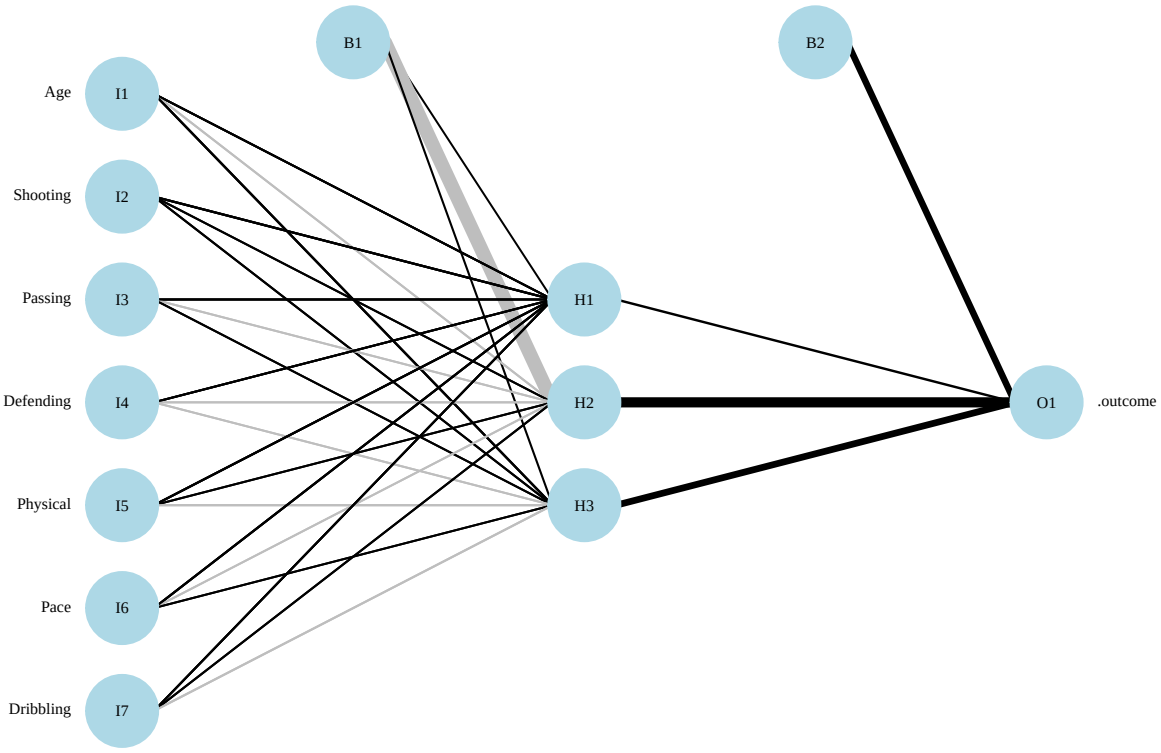
```
pred.M.nnet <- predict(M.nnet, M.testing)
par(mar=numeric(4),family='serif')
plotnet(M.nnet$finalModel, cex_val=.5)
```




```
A.nnet <- train(LogWage ~ Age+Shooting+Passing+Defending+Physical+Pace+Dribbling,
                data = A.training, method = "nnet", metric = "RMSE", trControl = TC,
                importance=TRUE, linout = 1, trace = FALSE)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
## trainInfo, : There were missing values in resampled performance measures.
```

```
pred.A.nnet <- predict(A.nnet, A.testing)
par(mar=numeric(4),family='serif')
plotnet(A.nnet$finalModel, cex_val=.5)
```




```
rm(GK.nnet)
rm(pred.GK.nnet)
rm(D.nnet)
rm(pred.D.nnet)
rm(M.nnet)
rm(pred.M.nnet)
rm(A.nnet)
rm(pred.A.nnet)
```

For GKs, this graph shows that neural nets analysis is not appropriate, giving us only a single connecting node. For the other 3 groups, we see that each of our inputs are approximately balanced, with none weighing much more heavily than the others. However these plots still indicate that a linear regression model may be most appropriate.

# BART

Finally, we run Bayesian Additive Regression Trees (BART) on our data.

```
pdbart.wrapper <- function(bartfit,data,...){
  form <- as.formula(paste0("~-1+",paste0(bartfit$call$formula[c(2,3)],collapse="+")))
  mm <- model.matrix(form,data)
  pdbart(x.train=mm[,-1],y.train=mm[,1],...)
}
```

```
GK.bart <- bart2(LogWage ~ Age + GKDiving + GKHandling + GKKicking + GKPositioning
                 + GKReflexes,data = GK.training,keepTrees = TRUE)
GK.bart.pred <- predict(GK.bart,GK.testing)
GK.bart.pred2 <- GK.bart.pred[,2,]
```

```r
cor(GK.bart.pred2[1,],GK.testing$LogWage)
```

```
## [1] 0.8486012
```

```r
cor(GK.bart.pred2[2,],GK.testing$LogWage)
```

```
## [1] 0.8526702
```

```r
cor(GK.bart.pred2[3,],GK.testing$LogWage)
```

```
## [1] 0.8490358
```

```r
cor(GK.bart.pred2[4,],GK.testing$LogWage)
```

```
## [1] 0.849553
```

```r
#RMSE
error <- GK.bart.pred2[1,] - GK.testing$LogWage
sqrt(mean(error^2))
```

```
## [1] 0.6244468
```

```r
error <- GK.bart.pred2[2,] - GK.testing$LogWage
sqrt(mean(error^2))
```

```
## [1] 0.6165488
```

```r
error <- GK.bart.pred2[3,] - GK.testing$LogWage
sqrt(mean(error^2))
```

```
## [1] 0.6252536
```

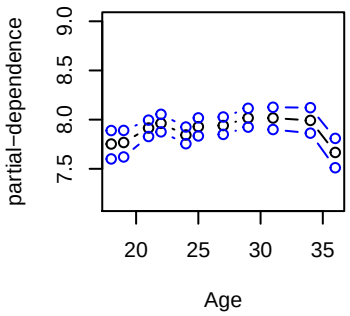
```r
error <- GK.bart.pred2[4,] - GK.testing$LogWage
sqrt(mean(error^2))
```

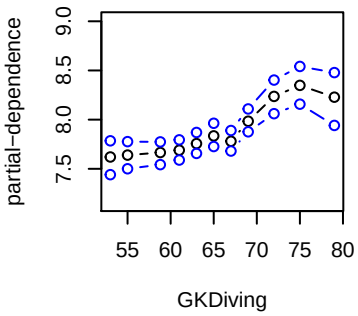
```
## [1] 0.6235481
```

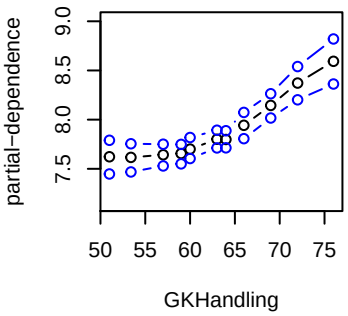
```r
par(mfrow=c(2,3),oma=c(0,1,0,1))
pdbart.run <- pdbart.wrapper(GK.bart,data=GK.training)
```

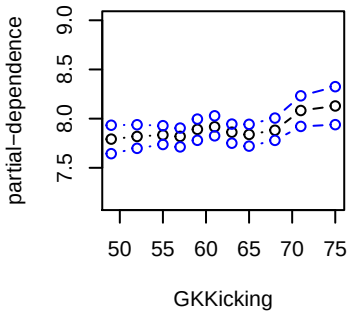
```
##
## Running BART with numeric y
##
## number of trees: 200
## Prior:
##  k: 2.000000
##  degrees of freedom in sigma prior: 3.000000
##  quantile in sigma prior: 0.900000
##  scale in sigma prior: 0.002934
##  power and base for tree prior: 2.000000 0.950000
##  use quantiles for rule cut points: false
## data:
##  number of training observations: 1395
##  number of test observations: 92070
##  number of explanatory variables: 6
##  init sigma: 0.682497, curr sigma: 0.682497
##
## Cutoff rules c in x<=c vs x>c
## Number of cutoffs: (var: number of possible c):
## (1: 100) (2: 100) (3: 100) (4: 100) (5: 100)
```

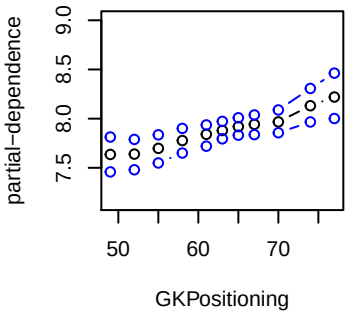
```
## (6: 100)
## Running mcmc loop:
## iteration: 100 (of 1000)
## iteration: 200 (of 1000)
## iteration: 300 (of 1000)
## iteration: 400 (of 1000)
## iteration: 500 (of 1000)
## iteration: 600 (of 1000)
## iteration: 700 (of 1000)
## iteration: 800 (of 1000)
## iteration: 900 (of 1000)
## iteration: 1000 (of 1000)
## total seconds in loop: 97.972408
##
## Tree sizes, last iteration:
## [1] 2 3 2 2 1 2 2 3 2 3 2 8 2 2 3 1 3 2
## 2 2 1 2 4 4 1 2 2 3 1 2 2 2 2 2 3 2 2 2
## 2 2 2 4 3 3 2 2 2 1 2 3 2 2 3 2 3 3 3 3
## 2 2 2 2 2 3 2 3 4 2 3 2 2 2 2 3 3 2 3 2
## 2 4 4 2 4 2 2 2 5 2 2 2 2 2 4 3 2 2 2 2
## 4 2 2 2 2 3 2 2 2 3 2 2 2 2 2 3 2 2 2 2
## 2 2 2 5 3 2 2 2 2 3 2 2 2 2 2 2 3 2 4 3
## 2 1 3 3 2 2 2 2 2 3 2 4 3 2 2 2 2 3 4 1
## 2 2 2 2 3 2 3 3 2 2 2 2 4 2 2 2 2 2 3 2
## 2 3 2 2 3 3 2 3 1 3 2 3 2 3 2 2 1 2 2 2
## 2 2
##
## Variable Usage, last iteration (var:count):
## (1: 47) (2: 53) (3: 53) (4: 35) (5: 43)
## (6: 42)
## DONE BART
```

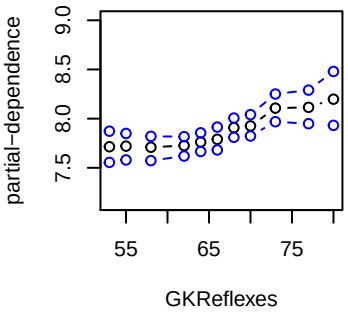



```
rm(GK.bart)
rm(GK.bart.pred)
rm(GK.bart.pred2)
rm(error)
rm(pdbart.run)
```

```
D.bart <- bart2(LogWage ~ Age+Shooting+Passing+Defending+Physical+Pace+Dribbling,
                data = D.training,keepTrees = TRUE)
D.bart.pred <- predict(D.bart,D.testing)
D.bart.pred2 <- D.bart.pred[,2,]

cor(D.bart.pred2[1,],D.testing$LogWage)
```

```
## [1] 0.8144094
```

```
cor(D.bart.pred2[2,],D.testing$LogWage)
```

```
## [1] 0.8093633
```

```
cor(D.bart.pred2[3,],D.testing$LogWage)
```

```
## [1] 0.81627
```

```
cor(D.bart.pred2[4,],D.testing$LogWage)
```

```
## [1] 0.8099966
```

```
#RMSE
error <- D.bart.pred2[1,] - D.testing$LogWage
sqrt(mean(error^2))
```

```
## [1] 0.6995214
```

```
error <- D.bart.pred2[2,] - D.testing$LogWage
sqrt(mean(error^2))
```

```
## [1] 0.7076214
```

```
error <- D.bart.pred2[3,] - D.testing$LogWage
sqrt(mean(error^2))
```

```
## [1] 0.6967549
```

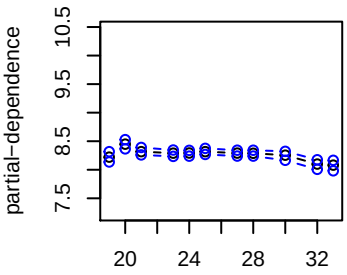
```
error <- D.bart.pred2[4,] - D.testing$LogWage
sqrt(mean(error^2))
```

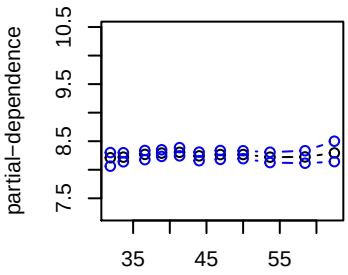
```
## [1] 0.7075563
```

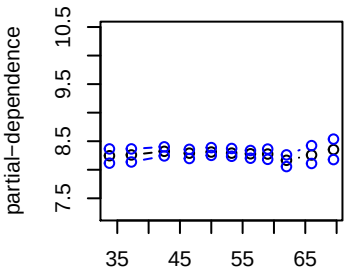
```
par(mfrow=c(2,3),oma=c(0,1,0,1))
pdbart.run <- pdbart.wrapper(D.bart,data=D.training)
```

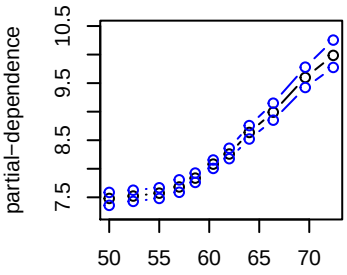
```
##
## Running BART with numeric y
##
## number of trees: 200
## Prior:
##  k: 2.000000
##  degrees of freedom in sigma prior: 3.000000
##  quantile in sigma prior: 0.900000
##  scale in sigma prior: 0.003291
##  power and base for tree prior: 2.000000 0.950000
##  use quantiles for rule cut points: false
## data:
##  number of training observations: 4048
##  number of test observations: 311696
##  number of explanatory variables: 7
##  init sigma: 0.772063, curr sigma: 0.772063
##
## Cutoff rules c in x<=c vs x>c
## Number of cutoffs: (var: number of possible c):
## (1: 100) (2: 100) (3: 100) (4: 100) (5: 100)
## (6: 100) (7: 100)
## Running mcmc loop:
## iteration: 100 (of 1000)
## iteration: 200 (of 1000)
## iteration: 300 (of 1000)
## iteration: 400 (of 1000)
## iteration: 500 (of 1000)
## iteration: 600 (of 1000)
## iteration: 700 (of 1000)
## iteration: 800 (of 1000)
## iteration: 900 (of 1000)
## iteration: 1000 (of 1000)
## total seconds in loop: 390.193329
##
## Tree sizes, last iteration:
## [1] 2 2 3 2 2 2 3 2 2 2 2 3 3 3 3 2 6 2
## 2 2 3 2 2 1 2 2 2 2 2 3 2 2 1 2 2 3 1 4
## 2 1 2 2 2 2 2 2 3 3 2 2 2 3 4 2 2 2 3 4
```

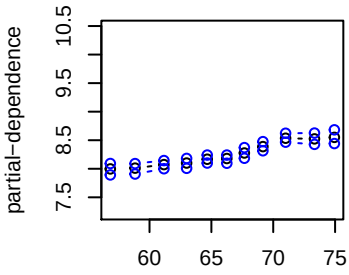
```
## 2 2 4 2 2 2 1 2 3 2 2 2 3 2 3 4 2 3 2 3
## 2 2 2 2 2 2 3 4 2 4 2 2 2 1 2 2 2 3 2 5
## 3 2 2 2 2 3 2 2 2 3 3 2 3 3 3 2 4 3 2 2
## 3 2 3 2 3 2 3 1 3 3 2 6 1 2 2 2 2 2 5 1
## 2 2 4 3 3 2 4 3 3 4 3 2 2 1 2 2 2 1 2 2
## 2 2 2 2 3 3 3 3 2 2 4 2 2 2 2 2 3 2 2 2
## 3 4 2 2 2 2 2 2 2 2 2 2 2 2 2 3 1 2 2 3
## 3 2
##
## Variable Usage, last iteration (var:count):
## (1: 38) (2: 41) (3: 30) (4: 67) (5: 27)
## (6: 40) (7: 33)
## DONE BART
```

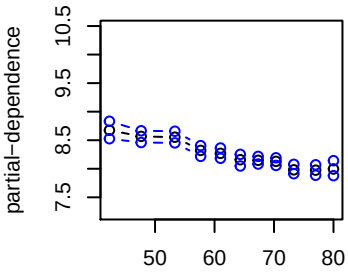


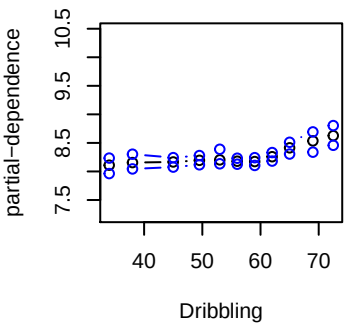




```
rm(D.bart)
rm(D.bart.pred)
rm(D.bart.pred2)
rm(error)
rm(pdbart.run)
```



```
M.bart <- bart2(LogWage ~ Age+Shooting+Passing+Defending+Physical+Pace+Dribbling,
                data = M.training,keepTrees = TRUE)
M.bart.pred <- predict(M.bart,M.testing)
M.bart.pred2 <- M.bart.pred[,2,]

cor(M.bart.pred2[1,],M.testing$LogWage)
```

```
## [1] 0.8104362
```

```
cor(M.bart.pred2[2,],M.testing$LogWage)
```

```
## [1] 0.812402
```

```
cor(M.bart.pred2[3,],M.testing$LogWage)
```

```
## [1] 0.8106905
```

```
cor(M.bart.pred2[4,],M.testing$LogWage)
```

```
## [1] 0.8107393
```

```
#RMSE
error <- M.bart.pred2[1,] - M.testing$LogWage
sqrt(mean(error^2))
```

```
## [1] 0.7183622
```

```
error <- M.bart.pred2[2,] - M.testing$LogWage
sqrt(mean(error^2))
```

```
## [1] 0.7152212
```

```
error <- M.bart.pred2[3,] - M.testing$LogWage
sqrt(mean(error^2))
```

```
## [1] 0.717555
```

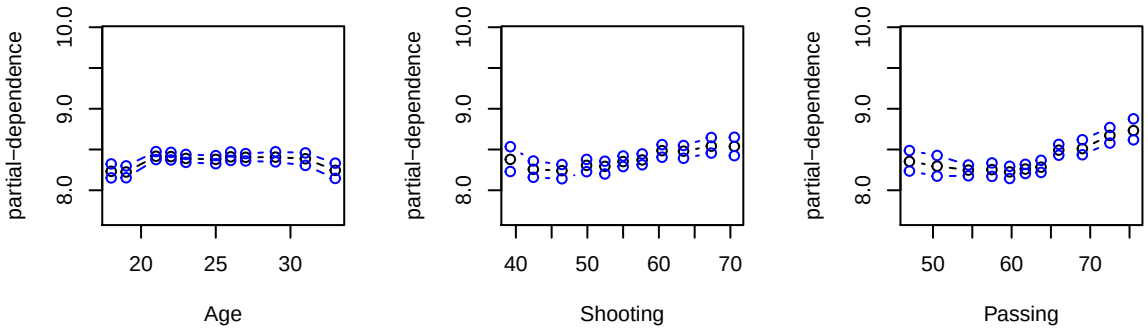
```
error <- M.bart.pred2[4,] - M.testing$LogWage
sqrt(mean(error^2))
```

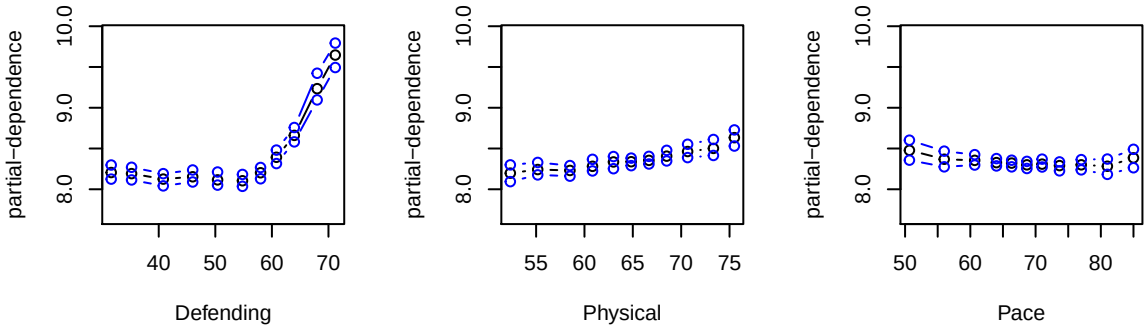
```
## [1] 0.7174777
```

```
par(mfrow=c(2,3),oma=c(0,1,0,1))
pdbart.run <- pdbart.wrapper(M.bart,data=M.training)
```

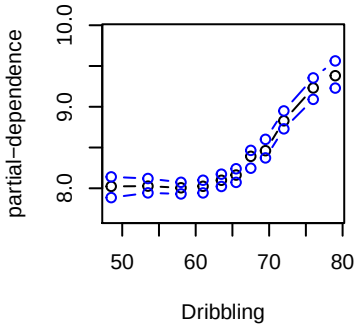
```
##
## Running BART with numeric y
##
## number of trees: 200
## Prior:
##  k: 2.000000
##  degrees of freedom in sigma prior: 3.000000
##  quantile in sigma prior: 0.900000
##  scale in sigma prior: 0.004182
##  power and base for tree prior: 2.000000 0.950000
##  use quantiles for rule cut points: false
## data:
##  number of training observations: 5192
##  number of test observations: 399784
##  number of explanatory variables: 7
```

```
##  init sigma: 0.885039, curr sigma: 0.885039
##
## Cutoff rules c in x<=c vs x>c
## Number of cutoffs: (var: number of possible c):
## (1: 100) (2: 100) (3: 100) (4: 100) (5: 100)
## (6: 100) (7: 100)
## Running mcmc loop:
## iteration: 100 (of 1000)
## iteration: 200 (of 1000)
## iteration: 300 (of 1000)
## iteration: 400 (of 1000)
## iteration: 500 (of 1000)
## iteration: 600 (of 1000)
## iteration: 700 (of 1000)
## iteration: 800 (of 1000)
## iteration: 900 (of 1000)
## iteration: 1000 (of 1000)
## total seconds in loop: 499.897235
##
## Tree sizes, last iteration:
## [1] 2 2 1 2 3 2 2 2 2 2 4 2 3 2 2 2 3 3
## 2 2 3 3 2 3 2 3 2 2 1 2 1 2 2 2 2 2 2 2
## 2 2 3 1 4 3 1 3 4 3 2 1 2 2 2 2 2 2 3 2
## 2 2 2 2 3 2 2 3 2 3 3 2 2 3 2 3 2 2 3 2
## 3 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 3
## 5 2 5 2 1 2 2 2 3 1 2 3 3 1 2 2 2 2 2 2
## 2 2 1 2 2 2 2 2 3 4 2 2 3 3 2 3 2 2 2 2
## 3 2 2 2 2 2 2 2 2 3 3 2 3 2 2 2 1 2 1 4
## 2 3 4 2 3 3 4 1 2 2 3 2 6 2 4 1 1 3 2 2
## 2 2 2 2 3 2 2 2 2 1 2 1 3 2 2 3 2 3 2 2
## 2 3
##
## Variable Usage, last iteration (var:count):
## (1: 37) (2: 29) (3: 39) (4: 45) (5: 39)
## (6: 29) (7: 35)
## DONE BART
```




```
rm(M.bart)
rm(M.bart.pred)
rm(M.bart.pred2)
rm(error)
rm(pdbart.run)
```




```
A.bart <- bart2(LogWage ~ Age+Shooting+Passing+Defending+Physical+Pace+Dribbling,
                data = A.training,keepTrees = TRUE)
A.bart.pred <- predict(A.bart,A.testing)
A.bart.pred2 <- A.bart.pred[,2,]

cor(A.bart.pred2[1,],A.testing$LogWage)
```

```
## [1] 0.8107127
```

```
cor(A.bart.pred2[2,],A.testing$LogWage)
```

```
## [1] 0.8092063
```

```r
cor(A.bart.pred2[3,],A.testing$LogWage)
```

```
## [1] 0.808278
```

```r
cor(A.bart.pred2[4,],A.testing$LogWage)
```

```
## [1] 0.8020632
```

```r
#RMSE
error <- A.bart.pred2[1,] - A.testing$LogWage
sqrt(mean(error^2))
```

```
## [1] 0.739602
```

```r
error <- A.bart.pred2[2,] - A.testing$LogWage
sqrt(mean(error^2))
```

```
## [1] 0.7433817
```

```r
error <- A.bart.pred2[3,] - A.testing$LogWage
sqrt(mean(error^2))
```

```
## [1] 0.7440452
```

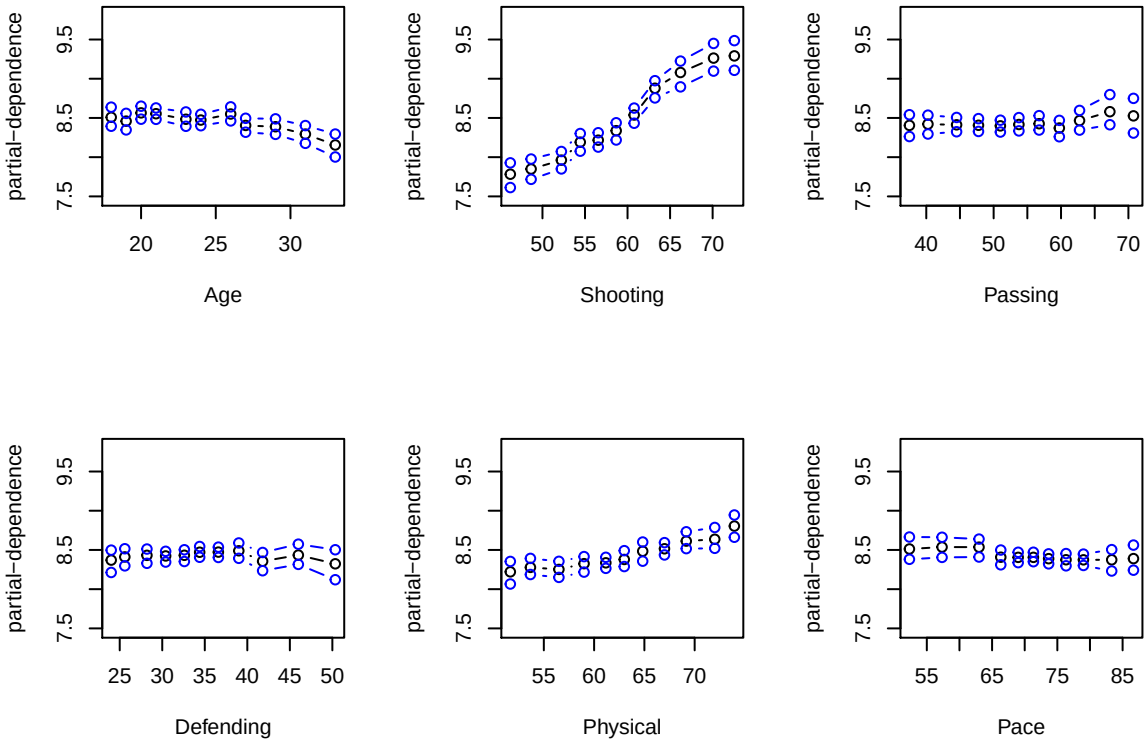
```r
error <- A.bart.pred2[4,] - A.testing$LogWage
sqrt(mean(error^2))
```

```
## [1] 0.7551755
```

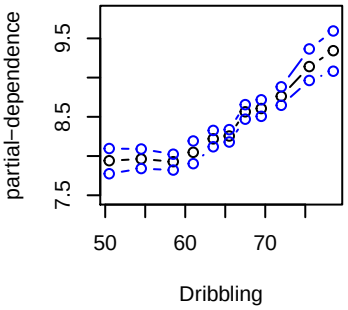
```r
par(mfrow=c(2,3),oma=c(0,1,0,1))
pdbart.run <- pdbart.wrapper(A.bart,data=A.training)
```

```
##
## Running BART with numeric y
##
## number of trees: 200
## Prior:
##  k: 2.000000
##  degrees of freedom in sigma prior: 3.000000
##  quantile in sigma prior: 0.900000
##  scale in sigma prior: 0.002763
##  power and base for tree prior: 2.000000 0.950000
##  use quantiles for rule cut points: false
## data:
##  number of training observations: 2367
##  number of test observations: 182259
##  number of explanatory variables: 7
##  init sigma: 0.754639, curr sigma: 0.754639
##
## Cutoff rules c in x<=c vs x>c
## Number of cutoffs: (var: number of possible c):
## (1: 100) (2: 100) (3: 100) (4: 100) (5: 100)
## (6: 100) (7: 100)
## Running mcmc loop:
## iteration: 100 (of 1000)
## iteration: 200 (of 1000)
## iteration: 300 (of 1000)
## iteration: 400 (of 1000)
```

```
## iteration: 500 (of 1000)
## iteration: 600 (of 1000)
## iteration: 700 (of 1000)
## iteration: 800 (of 1000)
## iteration: 900 (of 1000)
## iteration: 1000 (of 1000)
## total seconds in loop: 220.103842
##
## Tree sizes, last iteration:
## [1] 3 5 2 3 2 2 2 2 3 3 2 2 4 2 2 2 3 2
## 2 2 2 4 2 2 2 3 2 2 3 1 3 2 2 2 2 3 2 2
## 3 2 2 2 2 2 2 3 3 2 2 2 2 2 3 2 2 1 2 2
## 3 2 2 2 2 4 2 3 2 2 2 3 3 4 1 3 3 2 3 2
## 2 2 3 2 1 2 2 2 2 2 3 3 2 2 2 2 2 2 3 2
## 3 2 3 2 3 3 2 2 2 3 2 2 2 2 4 2 2 2 2 2
## 2 2 2 2 2 4 5 2 2 2 2 2 3 2 2 2 3 2 2 2
## 3 3 2 1 3 2 3 2 1 2 2 2 3 2 3 2 2 3 2 2
## 2 3 2 2 2 2 2 2 4 3 1 2 3 2 4 2 3 2 2 2 3
## 2 1 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 1 2 5 2 2
## 2 2
##
## Variable Usage, last iteration (var:count):
## (1: 39) (2: 32) (3: 42) (4: 35) (5: 41)
## (6: 35) (7: 34)
## DONE BART
```




```
rm(A.bart)
rm(A.bart.pred)
rm(A.bart.pred2)
rm(error)
rm(pdbart.run)
```




In these outputs, we find further evidence that that this problem should be solved using linear regression. Using the BART output, we plot marginal plots for each position and attribute. The more these look like straight lines, the more it's simply a linear model. While there are some exceptions, such as age for goalkeepers, most of these features do look linear. This also might help explain why each of our methods yeilded similar results.

# Conclusion

In trying to find the best model to predict player wages, we find that this is truly a linear regression problem. A simple additive linear model seems to do as well as any other method in explaining how wages change with changes in player skill level. Using this model, we will be able to predict a new players wage given their attributes. If we also take existing players and find their wage using the model, we may be able to show whether that player is underpaid or overpaid for their skill level.