IST 707 – Data Analysis
Michael Armesto
David Doman
N'Dea Jackson
Matthew McDonnell

Introduction

The National Basketball Association (NBA) is an American men's professional basketball league. It is composed of 30 teams and is one of the four major professional sports leagues in the United States and Canada. To many, the NBA is considered the premier men's professional basketball league in the world. During the Coronavirus off-season, the league commissioner, Adam Silver, enlisted the help of four of the world's top data scientists to complete detailed analysis of a prior season's player stats and provide visualizations that could be used to forecast player performance in the bubble. Outside of improving league performance, this analysis also stood to help the NBA drive their fan satisfaction ratings during the tumultuous time that is the 2019-2020. With fans and season ticket holders not being able to attend games due to the health and safety protocols surrounding the COVID-19 pandemic, the league wanted to ensure that fans would be able to enjoy the same quality of playoff basketball from their homes.

The available data could also allow General Managers to evaluate player performance and make critical roster decisions. It could also give coaches valuable insight to change their strategy and the way they use each player. Scouts and coaching staff could save hundreds of hours by analyzing the data rather than watching film.

Data Description

The dataset was obtained from Kaggle, where it had been previously scraped from NBA's REST API, which is no longer publicly available. Using a 6-Camera system in each NBA arena, the AutoSTATS Player-Tracking Technology can track 2-dimensional player locations 25 times per second. The dataset used in this report is comprised of each of the 128,069 shot attempts taken in the NBA's 2014-15 season, and various measurables at the moment of the ball's release.

Each shot is described with the following variables:

- Game_ID: an integer identifying the game in which shot was taken
- Matchup: a text field with the date, and teams (ex. MAR 01, 2015 - CHA @ ORL)
- Location: Whether the game was home or away
- W: Whether the game was won or lost
- Final_Margin: the score differential at the game's end
- Shot_Number: an integer describing the order of a player's shots during a game
- Period: Which period the shot was attempted in
- Game_Clock: The time on the game clock when the shot was attempted
- Shot_Clock: The time remaining on the shot clock when the shot was attempted
- Dribbles: The number of times a player dribbled the ball prior to attempting shot
- Touch_Time: Seconds in which a player possessed the ball prior to attempting shot
- Shot_Dist: The distance (feet) from the basket from which the shot was attempted
- Pts_Type: Whether the shot was a 2 or 3-point attempt
- Shot_Result: Whether the shot was "made" or "missed"

- Closest_Defender: The name of the closest opposing player to the ball when shot was attempted
- Closest_Defender_Player_ID: a numeric identifier of the closest opposing player
- Close_Def_Dist: The distance (feet) between closest defender and the ball when shot was attempted
- FGM: A binary variable describing the shot result (made or missed)
- PTS: The number of points scored on the attempt. Could include points from foul shots made even if initial shot was missed
- Player_name: The name of the shooter
- Player_ID: The numeric identifier of the shooter
- LocationX: The horizontal position on the court where the shot was taken
- LocationY : The vertical position on the court where the shot was taken

Data Preparation

The data preparation began with loading the .csv file containing 21 attributes and 128,069 shot logs. A screenshot of the data can be seen below in Figure 1. Before beginning to pre-process the data, it is important to get an understanding of the data that is being analyzed. There were **281** unique player ID's represented in the dataset across **1808** unique games.

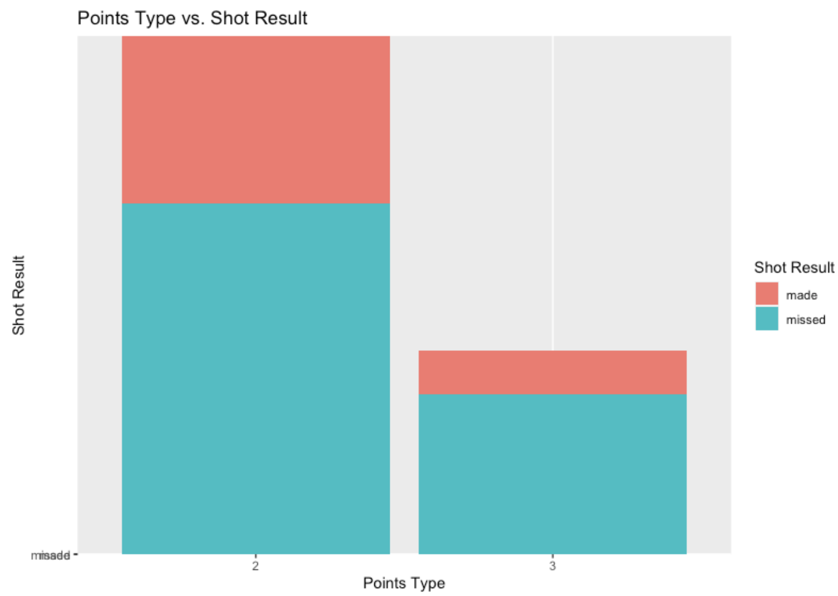
The next step on the data preparation is to perform some data pre-processing. First, the data set was checked for any missing values. After running the *is.na()* query, there were 5567 missing values found. After further analysis, it was determined that all of those values were from the *SHOT_CLOCK* column. *N/A* was determined to not be an error. When the game clock is under 24 seconds, the shot clock, which only counts as high as 24, is turned off. To remedy this issue, whenever an *N/A* was registered in the *SHOT_CLOCK* column, the value from the *GAME_CLOCK* column of that row replaced the *N/A*. The next step in data pre-processing was to separate the values within the *MATCHUP* column. Before being cleaned, the column contained both the date of the game and also the teams that were facing off. The pre-processing of this column resulted in three total columns: *DATE, TEAM,* and *OPPONENT*. Next, the name format for the *CLOSEST_DEFENDER* and *player_name* were standardized across the document as ***Last Name, First Name*.**

Below are summaries of the available variables, and a display of the dataset in table form.

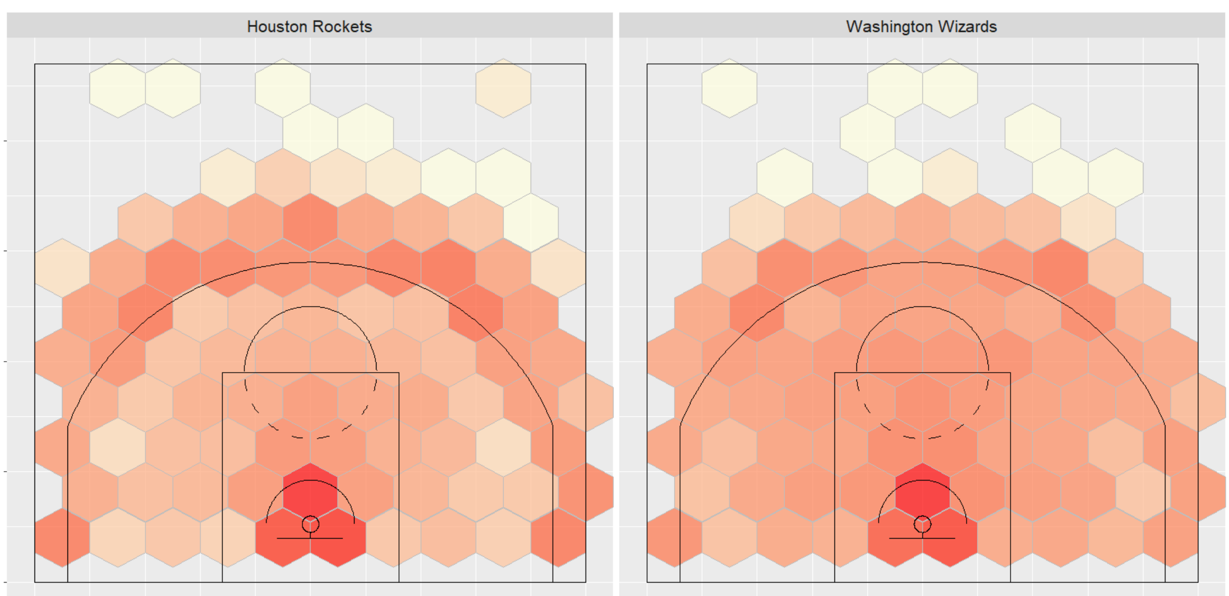
```
> summary(NBA)
    GAME_ID             MATCHUP            LOCATION                W              FINAL_MARGIN
 Min.   :21400001   Length:128069      Length:128069      Length:128069      Min.   :-53.0000
 1st Qu.:21400233   Class :character   Class :character   Class :character   1st Qu.: -8.0000
 Median :21400449   Mode  :character   Mode  :character   Mode  :character   Median :  1.0000
 Mean   :21400452                                                            Mean   :  0.2087
 3rd Qu.:21400673                                                            3rd Qu.:  9.0000
 Max.   :21400908                                                            Max.   : 53.0000

  SHOT_NUMBER         PERIOD       GAME_CLOCK          SHOT_CLOCK        DRIBBLES         TOUCH_TIME
 Min.   : 1.000   Min.   :1.000   Length:128069      Min.   : 0.00   Min.   : 0.000   Min.   :-163.600
 1st Qu.: 3.000   1st Qu.:1.000   Class :character   1st Qu.: 8.20   1st Qu.: 0.000   1st Qu.:   0.900
 Median : 5.000   Median :2.000   Mode  :character   Median :12.30   Median : 1.000   Median :   1.600
 Mean   : 6.507   Mean   :2.469                      Mean   :12.45   Mean   : 2.023   Mean   :   2.766
 3rd Qu.: 9.000   3rd Qu.:3.000                      3rd Qu.:16.68   3rd Qu.: 2.000   3rd Qu.:   3.700
 Max.   :38.000   Max.   :7.000                      Max.   :24.00   Max.   :32.000   Max.   :  24.900
                                                     NA's   :5567
   SHOT_DIST        PTS_TYPE      SHOT_RESULT        CLOSEST_DEFENDER   CLOSEST_DEFENDER_PLAYER_ID
 Min.   : 0.00   Min.   :2.000   Length:128069      Length:128069      Min.   :   708
 1st Qu.: 4.70   1st Qu.:2.000   Class :character   Class :character   1st Qu.:101249
 Median :13.70   Median :2.000   Mode  :character   Mode  :character   Median :201949
 Mean   :13.57   Mean   :2.265                                         Mean   :159038
 3rd Qu.:22.50   3rd Qu.:3.000                                         3rd Qu.:203079
 Max.   :47.20   Max.   :3.000                                         Max.   :530027

 CLOSE_DEF_DIST        FGM              PTS          player_name          player_id
 Min.   : 0.000   Min.   :0.0000   Min.   :0.0000   Length:128069      Min.   :   708
 1st Qu.: 2.300   1st Qu.:0.0000   1st Qu.:0.0000   Class :character   1st Qu.:101162
 Median : 3.700   Median :0.0000   Median :0.0000   Mode  :character   Median :201939
 Mean   : 4.123   Mean   :0.4521   Mean   :0.9973                      Mean   :157238
 3rd Qu.: 5.300   3rd Qu.:1.0000   3rd Qu.:2.0000                      3rd Qu.:202704
 Max.   :53.200   Max.   :1.0000   Max.   :3.0000                      Max.   :204060
```

	GAME_ID	MATCHUP	LOCATION	W	FINAL_MARGIN	SHOT_NUMBER	PERIOD	GAME_CLOCK	SHOT_CLOCK
1104	21400140	NOV 15, 2014 - CHA @ GSW	A	L	-25	6	1	04:14:00	8.3
1105	21400140	NOV 15, 2014 - CHA @ GSW	A	L	-25	7	1	03:49:00	12.4
1106	21400140	NOV 15, 2014 - CHA @ GSW	A	L	-25	8	1	02:50:00	10.3
1107	21400140	NOV 15, 2014 - CHA @ GSW	A	L	-25	9	2	02:22:00	7.6
1108	21400140	NOV 15, 2014 - CHA @ GSW	A	L	-25	10	3	08:16:00	14.6
1109	21400140	NOV 15, 2014 - CHA @ GSW	A	L	-25	11	3	07:28:00	6.0
1110	21400140	NOV 15, 2014 - CHA @ GSW	A	L	-25	12	3	04:44:00	8.1
1111	21400140	NOV 15, 2014 - CHA @ GSW	A	L	-25	13	3	04:12:00	12.7
1112	21400140	NOV 15, 2014 - CHA @ GSW	A	L	-25	14	3	03:11:00	5.8
1113	21400130	NOV 14, 2014 - CHA @ PHX	A	W	8	1	1	11:34:00	6.0
1114	21400130	NOV 14, 2014 - CHA @ PHX	A	W	8	2	1	11:00:00	11.5
1115	21400130	NOV 14, 2014 - CHA @ PHX	A	W	8	3	1	08:05:00	9.4
1116	21400130	NOV 14, 2014 - CHA @ PHX	A	W	8	4	1	07:38:00	8.3

Some visualizations were created and added to the report below to better describe the data including a bar chart of the points type versus the shot result, as well as some heat maps using latitude and longitude location data.



As an example of the way shot location data can help analyze team performance, here are heat maps comparing the shot attempts of the 2014-15 Houston Rockets and Washington Wizards. The rockets clearly employed a focus on shooting 3-point shots, which are obviously worth the most, and extreme close-range shots, which are the most likely to go in. They steered away from mid-range shots, which are less likely to be successful, but worth no more points than a close-range shot. In contrast, the Wizards did not employ such rigorous focus on what is now called 'Moreyball', a play on Rocket's General Manager Daryl Morey. The rockets won 10 more regular season games than the Wizards, and advanced to the conference finals.



Models

Decision Trees

The next analysis utilized to model the dataset was Decision Trees. The *rpart* function was used to complete the decision tree analysis. Many different trees were run, tweaking the variables and parameters each time to determine the tree with the highest prediction accuracy. Pruning was also applied to remove any splits in the tree that did not improve the overall R-squared for the model. Many values were tested for the complexity parameter but .001 was ultimately chosen in the end. This basically means that if any split does not improve the R-squared for the model by at least .001 it will be removed from the tree.

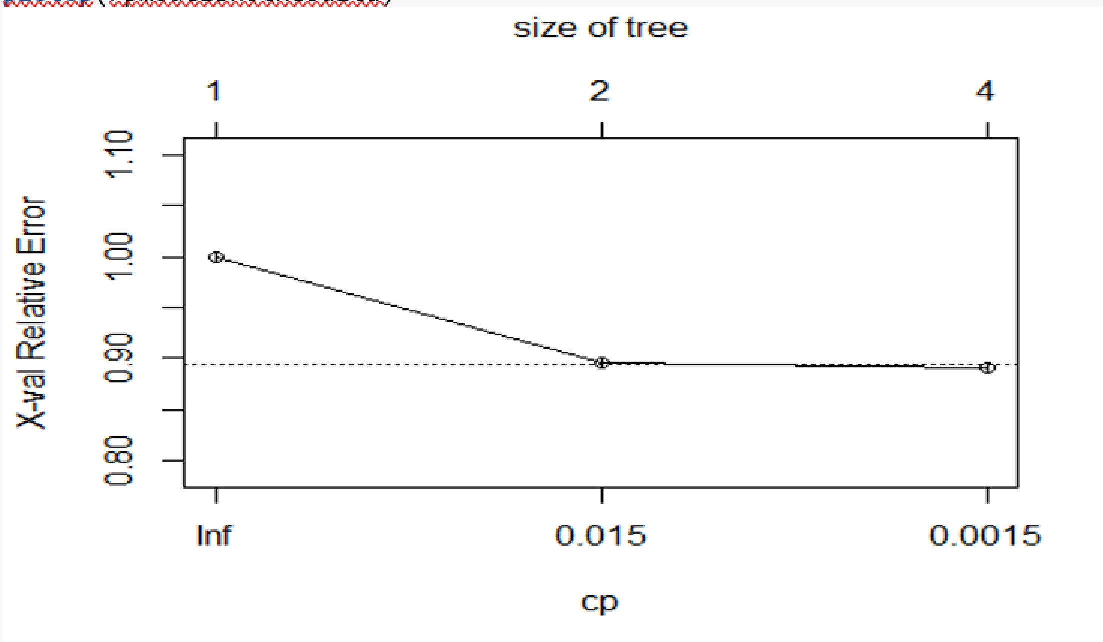
The dataset was split into a training dataset and a testing dataset with 75% of the data being training and the other 25% being testing. This will allow for the model to be built using the training data and the predictions to be made on the testing data. The code below shows how this was done.

```
set.seed(1234)
sl_split<- createDataPartition(NBAshots$SHOT_RESULT, p=.75, list=F)

training<- NBAshots[sl_split, ]
testing<- NBAshots[-sl_split, ]
```

The first decision tree shown below was run with 4 attributes that one would assume to be obvious factors as to whether a shot will be successful or not.

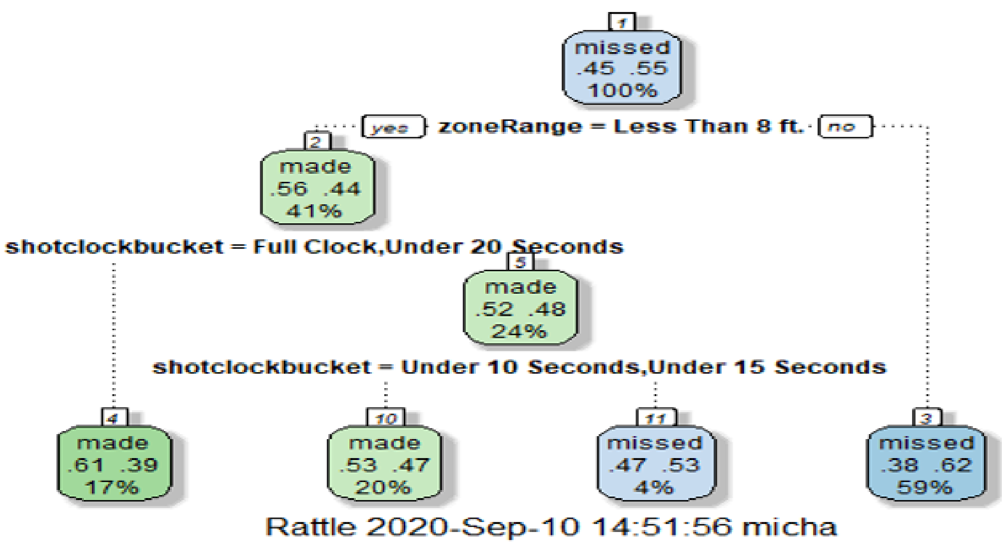
```
set.seed(1234)

rpmodelShotResulta <- rpart(SHOT_RESULT ~ shotclockbucket + SHOT_DISTkbucket
+  CLOSE_DEF_DISTkbucket + zoneRange, data=training,
        control=rpart.control(minsplit=1, minbucket=1, cp=0.001, xval = 10),
parms=list(split="gini"))

printcp(rpmodelShotResulta)
## Variables actually used in tree construction:
## [1] shotclockbucket zoneRange
## Root node error: 41437/91580 = 0.45247
## n= 91580
##           CP nsplit rel error  xerror      xstd
## 1 0.1046408      0   1.00000 1.00000 0.0036351
## 2 0.0022685      1   0.89536 0.89536 0.0035852
## 3 0.0010000      3   0.89082 0.89082 0.0035823
```

```
plotcp(rpmodelShotResulta)
```



From the plot above it shows that between 2 and 4 nodes is a good size for the decision tree. Below is the tree produced from the training dataset:

```
fancyRpartPlot(rpmodelShotResulta)
```

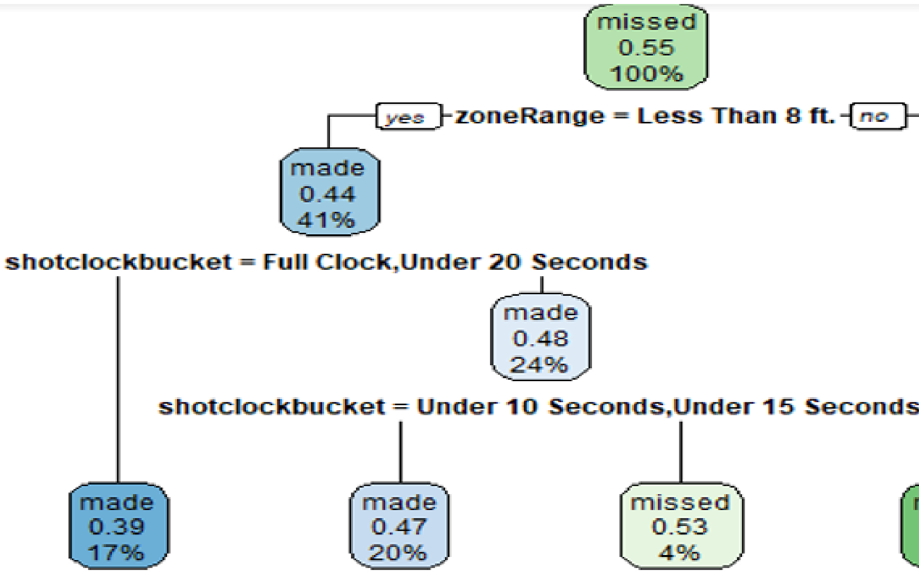


The next step was to view a confusion matrix based on the predictions made using the testing dataset, look at variable importance and to determine the model's accuracy. As shown below in the confusion matrix, this model predicted missed shots better than it did made shots. As for variable importance, zoneRange was the most important variable in this tree. Finally, this decision tree resulted in having a 59.53% accuracy rate, which is not a very precise model. The results and decision tree are shown below:

```
rpresultsShotResultb <- rpart.predict(rpmodelShotResulta, newdata=testing, ty
pe=c("class"))

# Confusion Matrix
rpconfMat= table(rpresultsShotResultb, testing$SHOT_RESULT)
addmargins(rpconfMat)
```

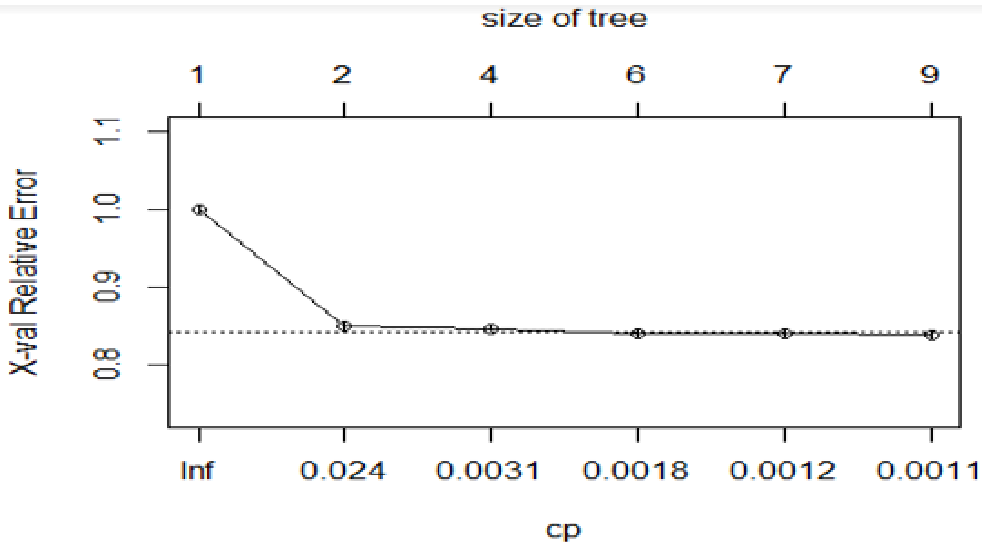
rpresultsShotResultb	made	missed	Sum
made	6334	4876	11210
missed	7478	11838	19316
Sum	13812	16714	30526

```
rpaccuracy <- sum(diag(rpconfMat))/sum(rpconfMat)
rpaccuracy

## [1] 0.5952958

summary(rpresultsShotResultb)

##   made missed
##  11210  19316

summary(rpmodelShotResulta)
##             CP nsplit rel error    xerror        xstd
## 1 0.104640780      0 1.0000000 1.0000000 0.003635052
## 2 0.002268504      1 0.8953592 0.8953592 0.003585244
## 3 0.001000000      3 0.8908222 0.8908222 0.003582314
## Variable importance
##        zoneRange SHOT_DISTkbucket  shotclockbucket
##               50               37               13

# Plots
rpart.plot::rpart.plot(rpmodelShotResulta)
```



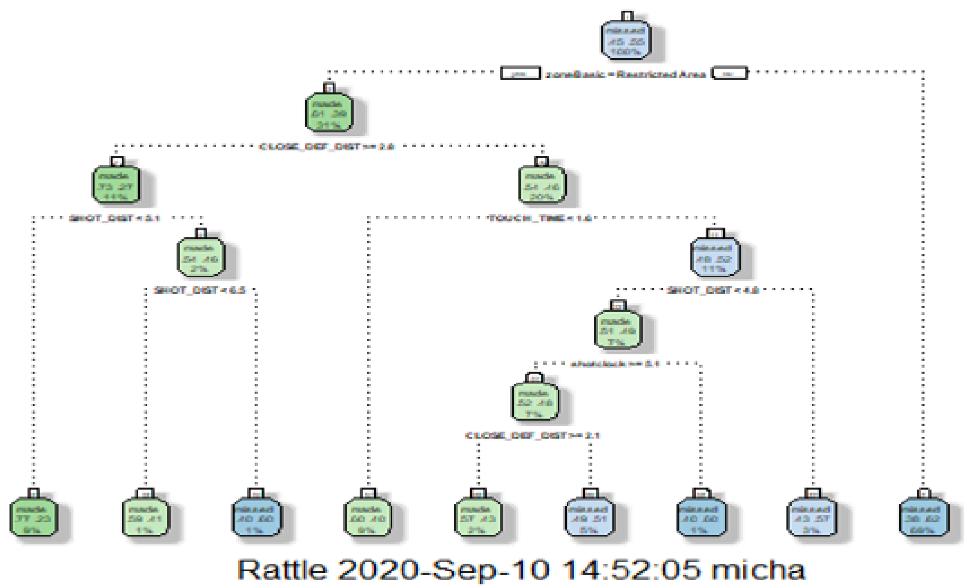
A second decision tree was run using additional variables that may not be so obvious when determining if a shot will be successful or not. For example, the number of dribbles, the period in the game, the amount of time a player touches the ball before the shot is attempted, etc. were included into the 2nd decision tree. The code for this tree is shown below:

```
rpmodelShotResult <- rpart(SHOT_RESULT ~ LOCATION + PERIOD + shotclock + DRIB
BLES + TOUCH_TIME + SHOT_DIST + CLOSE_DEF_DIST + zoneBasic +  zoneRange, data
=training,
       control=rpart.control(minsplit=1, minbucket=1, cp=0.001, xval = 10) ,p
arms=list(split="gini"))

printcp(rpmodelShotResult)
## Variables actually used in tree construction:
## [1] CLOSE_DEF_DIST SHOT_DIST      shotclock      TOUCH_TIME
## [5] zoneBasic
## Root node error: 41437/91580 = 0.45247
## n= 91580
##          CP nsplit rel error  xerror      xstd
## 1 0.1500833      0   1.00000 1.00000 0.0036351
## 2 0.0037286      1   0.84992 0.84992 0.0035529
## 3 0.0026064      3   0.84246 0.84721 0.0035508
## 4 0.0012308      5   0.83725 0.84012 0.0035451
## 5 0.0011584      6   0.83602 0.84024 0.0035452
## 6 0.0010000      8   0.83370 0.83896 0.0035441

plotcp(rpmodelShotResult)
```



The plot above shows that anywhere between 4 and 7 nodes seems to be the sweet spot as it relates to the size of the tree. Below is the tree produced based on the training data.

```
fancyRpartPlot(rpmodelShotResult)
```



As was the case with the 1st decision tree, the confusion matrix, prediction accuracy and variable importance were analyzed. Again, this model predicted missed shots better than it did made shots. Important variables showed to be zoneBasic, Shot_Dist, zoneRange and

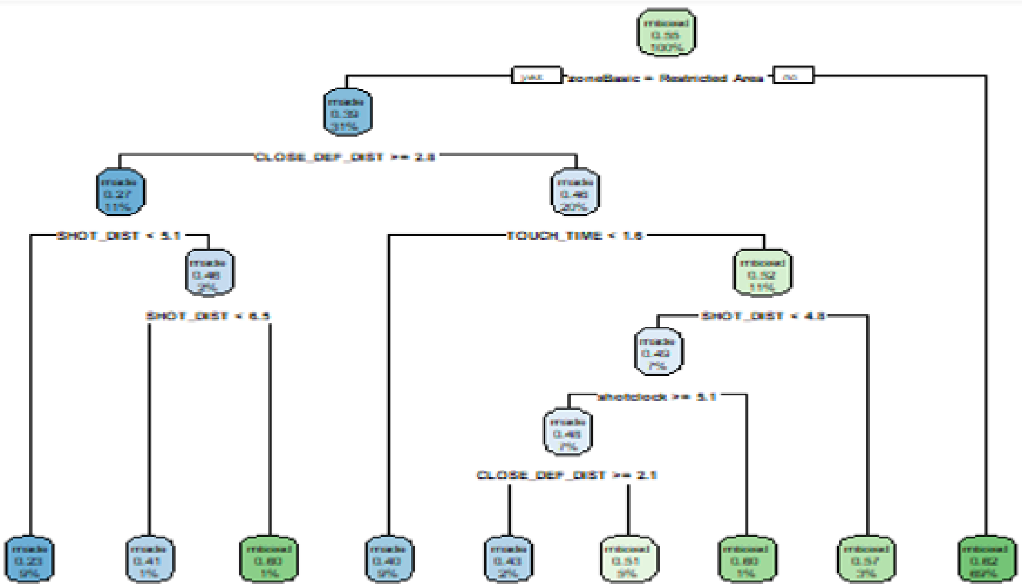
CLOSE_DEF_DIST. The results for the second tree was a 62.38% accuracy rate, which is an improvement from the original tree.

```
rpresultsShotResult2 <- rpart.predict(rpmodelShotResult, newdata=testing, typ
e=c("class"))
# Confusion Matrix
rpconfMat= table(rpresultsShotResult2, testing$SHOT_RESULT)
addmargins(rpconfMat)
```

rpresultsShotResult2	made	missed	Sum
made	4535	2207	6742
missed	9277	14507	23784
Sum	13812	16714	30526

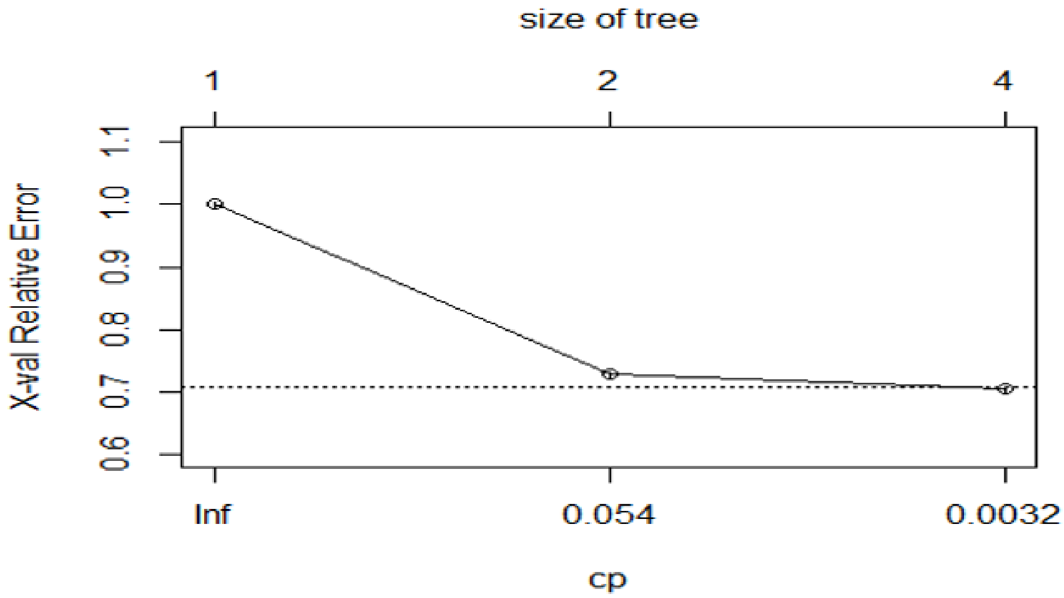
```
rpaccuracy <- sum(diag(rpconfMat))/sum(rpconfMat)
rpaccuracy

## [1] 0.6237961

summary(rpresultsShotResult2)

##   made missed
##   6742  23784

summary(rpmodelShotResult)
##             CP nsplit rel error    xerror        xstd
## 1 0.150083259      0 1.0000000 1.0000000 0.003635052
## 2 0.003728552      1 0.8499167 0.8499167 0.003552932
## 3 0.002606366      3 0.8424596 0.8472138 0.003550801
```

```
## 4 0.001230784      5 0.8372469 0.8401187 0.003545093
## 5 0.001158385      6 0.8360161 0.8402394 0.003545192
## 6 0.001000000      8 0.8336994 0.8389603 0.003544146
##
## Variable importance
##      zoneBasic       SHOT_DIST       zoneRange CLOSE_DEF_DIST      TOUCH_TIME
##             29              27              20             13               5
##      shotclock        DRIBBLES
##              5               1

# Plots
rpart.plot::rpart.plot(rpmodelShotResult)
```



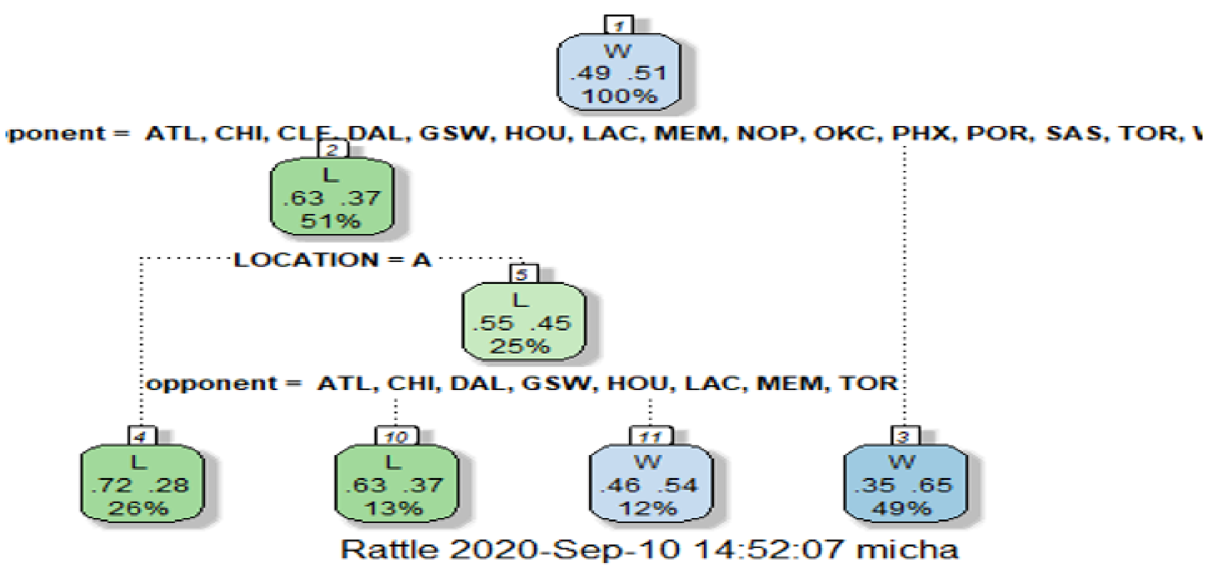
A third tree was run to determine whether a win or loss could be predicted. Variables included in this tree were the location of the game and the opponent.

```
rpmodelWinLoss <- rpart(W ~ LOCATION + opponent, data=training,
         control=rpart.control(minsplit=1, minbucket=1, cp=0.001, xval = 10),
parms=list(split="gini"))
printcp(rpmodelWinLoss)
## Variables actually used in tree construction:
## [1] LOCATION opponent
## Root node error: 45270/91580 = 0.49432
## n= 91580
##         CP nsplit rel error  xerror      xstd
## 1 0.274619      0   1.00000 1.00000 0.0033422
## 2 0.010459      1   0.72538 0.72774 0.0032082
## 3 0.001000      3   0.70446 0.70393 0.0031842

plotcp(rpmodelWinLoss)
```



As shown in the plot above, tree size between 2 and 4 nodes seems to be the best option. The decision tree produced from the training data is shown below:

```
fancyRpartPlot(rpmodelWinLoss)
```



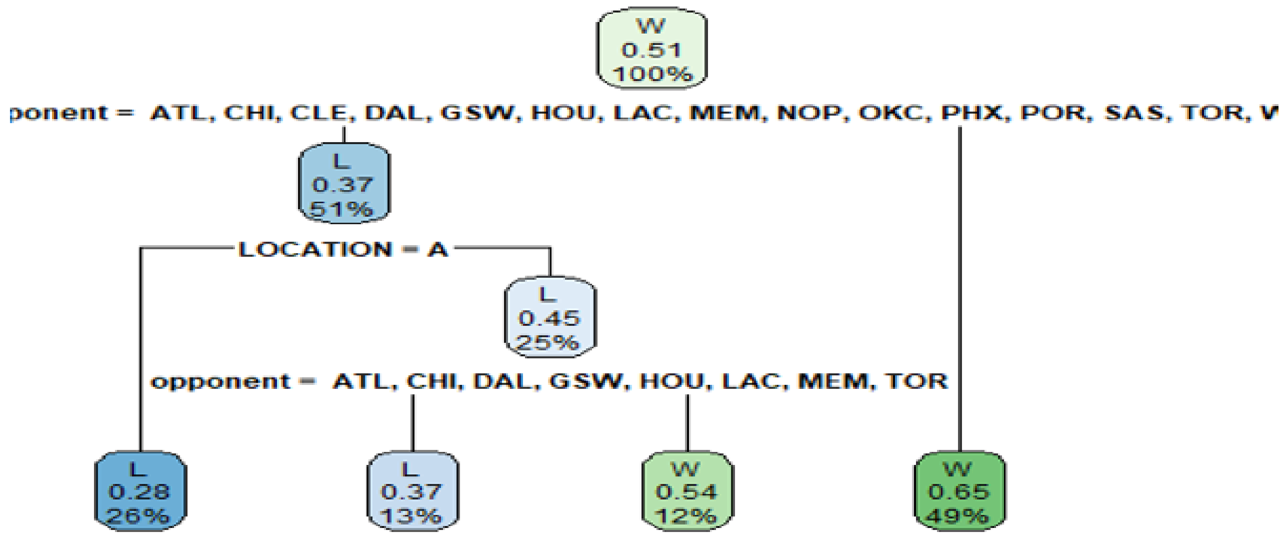
This win/loss tree resulted in having a prediction accuracy of 64.54%, which is higher than either of the successful shot decision trees were able to accomplish.

```
rpresultsWinLoss <- rpart.predict(rpmodelWinLoss, newdata=testing, type=c("cl
ass"))

# Confusion Matrix
```

```
rpconfMat= table(rpresultsWinLoss, testing$W)
addmargins(rpconfMat)
```

rpresultsWinLoss	L	W	Sum
L	8102	3734	11836
W	7093	11597	18690
Sum	15195	15331	30526

```
rpaccuracy <- sum(diag(rpconfMat))/sum(rpconfMat)
rpaccuracy

## [1] 0.6453187

summary(rpresultsWinLoss)

##     L     W
## 11836 18690

summary(rpmodelWinLoss)
##           CP nsplit rel error    xerror        xstd
## 1 0.27461895      0 1.0000000 1.0000000 0.003342195
## 2 0.01045947      1 0.7253810 0.7277446 0.003208207
## 3 0.00100000      3 0.7044621 0.7039320 0.003184155
## Variable importance
## opponent LOCATION
##       86       14

# Plots
rpart.plot::rpart.plot(rpmodelWinLoss)
```



As shown in the analysis above, the highest accuracy for predicting a successful shot came from the 2nd decision tree at 62.38%. Important variables in both shot success decision trees included zoneRange, which makes a lot of sense in that the further away from the basket the shot is taken from the less likely it would be to go in. The win/loss decision tree was able to achieve a 64.54% prediction accuracy, which was slightly higher than the shot success prediction accuracy.

SVM Analysis

All packages used in SVM analysis.

```
library(arules)
library(arulesViz)
library(ggplot2)
library(kernlab)
library(dplyr)
library(jsonlite)
library(kernlab)
library(e1071)
library(readr)
library(FactoMineR)
library(dplyr)
library(e1071)
library(caret)
## Loading required package: lattice
library(rpart)
library(rpart.plot)
library(knitr)
library(caret)
getwd()
## [1] "/Users/mattmcdonnell"
NBAdata <- read.csv("/Users/mattmcdonnell/Downloads/dataset_20200829.csv")
dim(NBAdata)
## [1] 122106     30
```

Support Vector Machine Analysis taking a sample size of 1,000 rows to reduce run time from what was very long and even impossible to run without crashing.

```
NBAdata <- NBAdata[sample(nrow(NBAdata), 1000), ]
```

Predicting a win using SVM First create a dataframe with the desired variables.

```
svmWin <- data.frame(Location = NBAdata$LOCATION, Win = NBAdata$W,
                     Opponent = NBAdata$opponent)
```

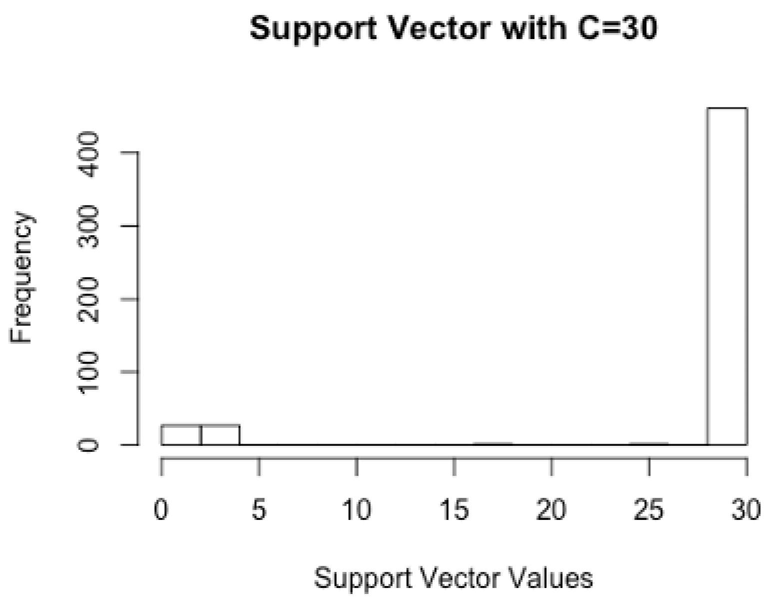
Create the train and test datasets using a data partition ratio of 70:30.

```
trainList <- createDataPartition(y= svmWin$Win, p=.7, list=FALSE)
trainData <- svmWin[trainList,]
testData <- svmWin[-trainList,]
```

Run the model and print the histogram showing how the cost coefficient performs/affects the model.

```
svmOutput <- ksvm(Win ~., data=trainData, kernel="rbfdot", kpar="automatic",
                     C=30, cross=3, prob.model=TRUE) # Usinig a larger C to
mitigate for classification
svmOutput
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 30
##
```

```
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.375
##
## Number of Support Vectors : 517
##
## Objective Function Value : -13918.52
## Training error : 0.33
## Cross validation error : 0.41003
## Probability model included.
hist(alpha(svmOutput)[[1]],
     main = "Support Vector with C=30",
     xlab = "Support Vector Values")
```



The many numbers to the right side of the histogram mean it's hard for the model to predict the zone properly. The numbers on the left had side of the histogram are too simple and easy to predict so don't offer any use in modeling. Alter this with a different 'C=' helps create the most efficiently accurate model.

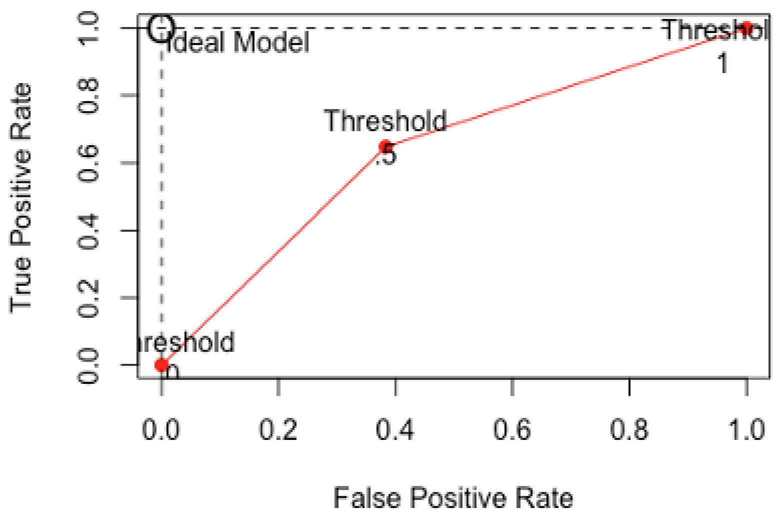
```
svmWin <- data.frame(Location = NBAdata$LOCATION, Win = NBAdata$W,
                     Opponent = NBAdata$opponent)
svmPred <- predict(svmOutput, testData)
svmPred
##   [1] W W L W W W W W W L W W W W L L L L W W W W L W W W L L L L L L W L
W L W
##  [38] W W L L W L L W L W W L L W W W L W W L W L W W W W W L W L L L W W
W L W
##  [75] L W W L W L L L L L W W L W W W L W L W L L W W W W L W W W W W L L
W L L
## [112] L W W L L W W L W L L L L L W W W W W L W L L L L L W W L W W W L W
L W W
## [149] L W L W W W W W L W W W L L W W L L W W L L L W W W L W W W W L W L
L L L
```

```
## [186] W W L W L W W L L L L L L W L W W L W L W W W W L W W W W W W L L W
L L W
## [223] L L L W W L L W L L L W W W W W W L W L W W L L W W W L L W L L L L
W W L
## [260] W L L L L W W W W W W L L L L L L L W L L L L W L L L W L W L W L L
W L W
## [297] W L L W
## Levels: L W
confusionMatrix(testData$Win, svmPred)
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   L   W
##          L  89  55
##          W  52 104
##
##                Accuracy : 0.6433
##                  95% CI : (0.5863, 0.6975)
##     No Information Rate : 0.53
##     P-Value [Acc > NIR] : 4.664e-05
##
##                   Kappa : 0.285
##
##  Mcnemar's Test P-Value : 0.8467
##
##             Sensitivity : 0.6312
##             Specificity : 0.6541
##          Pos Pred Value : 0.6181
##          Neg Pred Value : 0.6667
##              Prevalence : 0.4700
##          Detection Rate : 0.2967
##    Detection Prevalence : 0.4800
##       Balanced Accuracy : 0.6426
##
##        'Positive' Class : L
##
```

Visualizing a .5 Threshold of the best predictive svm analysis. Note that each sample of the 1000 is different and the SVM results are due to be slightly variable with each run so the sensitivity (true positive rate), specificity (false negative rate), and accuracy in the threshold plot are slightly off, but still illustrate the same image approximately.

```
tp <- c(0, 0.6471, 1 ) # The Sensitivity of the model acts as the true
positive rate
1-0.6167 # 0.3833
## [1] 0.3833
fp <- c(0 ,0.3833, 1) # The Specificity shows the false negative rate so
subtracting by 1 gives is the false positive rate
plot(fp,tp, pch = 19, col = "red", xlab = "False Positive Rate",
     ylab =  "True Positive Rate", main = )
lines(fp,tp, col = "red")
```

```
xadj <- c(.02, 0, -.04)
yadj <- c(0.02,.03, -.05)
text(x = fp + xadj, y = tp + yadj,
     labels = c("Threshold\n0", "Threshold\n.5", "Threshold\n1"))
abline(v = 0, lty = 2)
abline(h = 1, lty = 2)
text(.13, .96, labels = "Ideal Model")
points(0,1, pch = "O", cex = 1.5)
```



This plot is decent, but not too predictive. The slope is not as sharp as it should be to safely fall under classification as a strong model, but still shows higher that 50% predictive power (random guessing at if the team will win). This is a relatively difficult outcome to predict though, which is why it may actually have more practical use than it statistically seems.

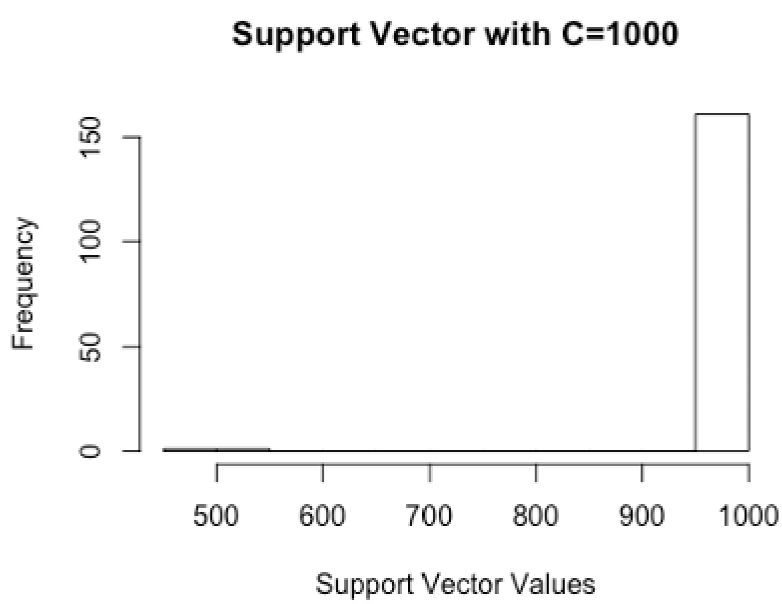
The second SVM analysis performed is to predict a successful shot attempt given a number of real in game situational variables, which can be seen below in the newly created data frame.
First create a data frame of all variables needed for this SVM analysis since it looks into a different question.

```
svmScore <- data.frame(SHOT_RESULT = NBAdata$SHOT_RESULT, PTS =
as.factor(NBAdata$PTS),
                       PTS_TYPE = as.factor(NBAdata$PTS_TYPE), SHOT_NUMBER =
as.factor(NBAdata$SHOT_NUMBER), DRIBBLES = as.factor(NBAdata$DRIBBLES),
                       TOUCH_TIME = as.factor(NBAdata$TOUCH_TIME),locationX =
as.factor(NBAdata$locationX), locationY = as.factor(NBAdata$locationY),
                       SHOT_DIST = as.factor(NBAdata$SHOT_DIST),zoneBasic =
as.factor(NBAdata$zoneBasic), PERIOD = as.factor(NBAdata$PERIOD),
                       shotclock = as.factor(NBAdata$shotclock),CLOSE_DEF_DIST =
as.factor(NBAdata$CLOSE_DEF_DIST), zoneRange = as.factor(NBAdata$zoneRange),
```

```
                    nameZone = as.factor(NBAdata$nameZone), gameclock =
NBAdata$gameclock)
```

Create new train and test datasets using a data partition ratio of 70:30 for this question.

```
trainList2 <- createDataPartition(y= svmScore$SHOT_RESULT, p=.7, list=FALSE)
trainData2 <- svmScore[trainList2,]
testData2 <- svmScore[-trainList2,]
svmOutput2 <- ksvm(SHOT_RESULT ~., data=trainData2, kernel="tanhdot",
kpar="automatic",
                   C=1000, cross=3, prob.model=TRUE) # Usinig a larger C to
##  Setting default kernel parameters
svmOutput2
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1000
##
## Hyperbolic Tangent kernel function.
##  Hyperparameters : scale =  1  offset =  1
##
## Number of Support Vectors : 163
##
## Objective Function Value : -180462053
## Training error : 0.229672
## Cross validation error : 0.158431
## Probability model included.
hist(alpha(svmOutput2)[[1]],
     main = "Support Vector with C=1000",
     xlab = "Support Vector Values")
```



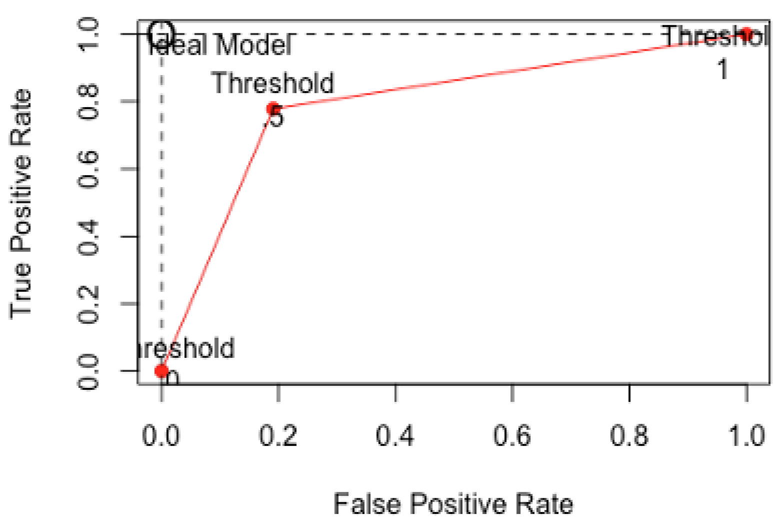
The many numbers to the right side of the histogram mean it's hard for the model to predict the zone properly. The numbers on the left had side of the histogram are too simple and easy to predict so don't offer any use in modeling. Altering this with a different 'C=' can help create the most efficiently accurate model. Ideally it would be more balanced, but R shuts down when increasing C too much so C=100 is a good compromise here. The histogram was not a great indicator in this particular question though. Changing the cost efficient did little to balance it out.

```
svmPred2 <- predict(svmOutput2, testData2)
svmPred2
##   [1] made   missed made   missed made   missed made   missed made   made
##  [11] missed made   made   made   missed made   made   made   made
missed
##  [21] made   made   missed made   made   made   missed made   made   made
##  [31] missed made   made   missed made   missed made   missed missed
missed
##  [41] made   made   missed missed made   missed missed missed made   made
##  [51] made   made   missed made   missed missed made   made   made
missed
##  [61] made   missed missed missed missed missed made   missed missed made
##  [71] missed missed missed missed made   missed made   missed missed
missed
##  [81] made   missed missed made   missed missed missed missed missed
missed
##  [91] missed made   missed made   missed made   missed made   missed made
## [101] made   made   made   missed made   made   made   made   missed made
## Levels: made missed
confusionMatrix(testData2$SHOT_RESULT, svmPred2)
## Confusion Matrix and Statistics
##
##           Reference
## Prediction made missed
##     made    110     31
##     missed   38    120
##
##                Accuracy : 0.7692
##                  95% CI : (0.7173, 0.8158)
##     No Information Rate : 0.505
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5382
##
##  Mcnemar's Test P-Value : 0.4701
##
##             Sensitivity : 0.7432
##             Specificity : 0.7947
##          Pos Pred Value : 0.7801
##          Neg Pred Value : 0.7595
##              Prevalence : 0.4950
##          Detection Rate : 0.3679
##    Detection Prevalence : 0.4716
##       Balanced Accuracy : 0.7690
```

```
##
##        'Positive' Class : made
##
```

Visualizing a .5 Threshold of the best predictive SVM analysis. Note that each sample of the 1000 is different and the SVM results are due to be slightly variable with each run so the sensitivity (true positive rate), specificity (false negative rate), and accuracy in the threshold plot are slightly off, but still illustrate the same image approximately.

```
tp <- c(0, 0.7778, 1 ) # The Sensitivity of the model acts as the true positive rate
1-0.8092 # 0.1908
## [1] 0.1908
fp <- c(0 ,0.1908, 1) # The Specificity shows the false negative rate so subtracting by 1 gives is the false positive rate
plot(fp,tp, pch = 19, col = "red", xlab = "False Positive Rate",
     ylab =  "True Positive Rate", main = )
lines(fp,tp, col = "red")
xadj <- c(.02, 0, -.04)
yadj <- c(0.02,.03, -.05)
text(x = fp + xadj, y = tp + yadj,
     labels = c("Threshold\n0", "Threshold\n.5", "Threshold\n1"))
abline(v = 0, lty = 2)
abline(h = 1, lty = 2)
text(.1, .97, labels = "Ideal Model")
points(0,1, pch = "O", cex = 1.5)
```



The threshold plot supports the accuracy, which was the highest of the models used on this dataset to predict a successful shot attempt. The sharp increase in the true positive rate coinciding with a slow increase in false positive rate is great when looking for strong predictive power in an SVM model. In threshold plotting, a sharp positive slope is ideal.

Rule Mining

To begin the association rule mining section of the analysis, some necessary packages needed to be downloaded and their libraries loaded including readr, dplyr, tidyr, hms, devtools, sqldf, arules, and arulesViz.

After loading the libraries, the attributes were analyzed to see which could stand to undergo discretization. Data discretization is defined as a process of converting continuous data attribute values into a finite set of intervals and associating with each some specific data value. Both discretization and numeric-to-nominal transformations are necessary in order for the Apriori algorithm to properly function. The "shotclock", "SHOT_DIST", "CLOSE_DEF_DIST", and "secondsRemaining" columns were discretized. The "shotclock" column was broken down into the folliwing bins: **Buzzer Beater** (0.0-4.9s), **Under 10 Seconds** (5.0-9.9s), **Under 15 Seconds** (10.0-14.9s), **Under 20 Seconds** (15.0-19.9s), and **Full Clock** (20.0-24.0s). The "SHOT_DIST" column was broken down into the following bins: **Short Range** (0-14.9ft), **Mid Range** (15.0-29.9ft), **Long Range** (30.0-44.9ft), and **Mid-Court** (50.0ft+). The "CLOSE_DEF_DIST" column was broken down into the following bins: **Very Close** (0.0-14.99ft), **Close** (15.0-29.9ft), **Farther Out** (30.0-44.9ft), and **Not Closely Guarded** (50.0ft+). Finally, the "secondsRemaining" column was broken down into the following bins: **End of Quarter** (0.0-9.9s), **Under 20 Seconds** (10.0-19.9s), **Under 30 Seconds** (20.0-29.9s), **Under 40 Seconds** (30.0-39.9s), **Under 50 Seconds** (40.0-49.9s), and **Under 1 Minute** (50.0-60.0s).

```
#Discretization
 range(NBA$SHOT_DIST)
## [1]  0.0 47.2
range(NBA$shotclock)
## [1]  0 24
range(NBA$TOUCH_TIME)
## [1] -100.5   24.9
range(NBA$SHOT_DIST)
## [1]  0.0 47.2
range(NBA$CLOSE_DEF_DIST)
## [1]  0.0 53.2
NBA$shotclock <- cut(NBA$shotclock, breaks = c(-Inf, 5.0, 10.0, 15.0, 20.0,
Inf),
                    labels = c("Buzzer Beater", "Under 10 Seconds", "Under 15
Seconds", "Under 20 Seconds", "Full Clock"))

NBA$SHOT_DIST <- cut(NBA$SHOT_DIST, breaks = c(-Inf, 15.0, 30.0, 45, Inf),
                      labels = c("Short Range", "Mid Range", "Long Range",
"Mid-Court"))

NBA$CLOSE_DEF_DIST <- cut(NBA$CLOSE_DEF_DIST, breaks = c(-Inf, 15, 30, 45,
Inf),
                       labels = c("Very Close", "Close", "Farther Out", "Not
Closely Guarded"))
```

```
NBA$secondsRemaining <- cut(NBA$secondsRemaining, breaks = c(-Inf, 10, 20,
30, 40, 50, Inf),
                        labels = c("End Quarter", "Under 20 Seconds", "Under
30 Seconds", "Under 40 Seconds", "Under 50 Seconds", "Under 1 Minute"))
```

Once the above steps were complete, the Apriori algorithm could be applied to the data set. Apriori is an algorithm for frequent set mining and association rule learning over relational databases. Given a set of transactions, T, the goal of association rule mining is to find all of the rules having: support >= minsup threshold and confidence >= minconf threshold. Some terms often associated with Apriori and their definitions can be found below. These terms were heavily weighed when formulating recommendations for the National Basketball Association.

- **Itemset:** a collection of one of more items.
- **Support:** gives an idea of how frequent an itemset is in all the transactions.
- **Confidence:** an indication of how often the rule has been found to be true.
    - **P(X|Y) = P(X,Y)/P(X)**
- **Lift:** the ratio of the observed support to that expected.
  Support {A,B}/(Support {A} * Support {B})

After reviewing the initial set of rules that were created, it was determined that all of the attributes that were being factored into the rules weren't particularly relevant. Some of these rows included: ——————. Instead of modifying the original NBA data set, the NBA data set was put into an NBA Test data set.

Lastly, the remaining columns that hadn't been discretionized were converted to factors.

```
NBATest <- NBA
 NBATest <- NBATest[, -35]
 NBATest$distanceShot <- as.factor(NBATest$distanceShot)
 NBATest$zoneRange <- as.factor(NBATest$zoneRange)
 NBATest$slugZone <- as.factor(NBATest$slugZone)
 NBATest$nameZone <- as.factor(NBATest$nameZone)
 #NBATest$zoneBasic <- as.factor(NBATest$zoneBasic)
 NBATest$minutesRemaining <- as.factor(NBATest$minutesRemaining)
 NBATest$idEvent <- as.factor(NBATest$idEvent)
 NBATest$slugTeamAway <- as.factor(NBATest$slugTeamAway)
 NBATest$slugTeamHome <- as.factor(NBATest$slugTeamHome)
 NBATest$gameclock <- as.factor(NBATest$gameclock)
 NBATest$TOUCH_TIME <- as.factor(NBATest$TOUCH_TIME)
 NBATest <- NBATest[, -18]
 NBATest <- NBATest[, -27]
 NBATest <- NBATest[, -35]
 NBATest <- NBATest[, -35]
 NBATest <- NBATest[, -7]
 NBATest <- NBATest[,-2]

#Association Rule Mining
 NBArules <- apriori(NBA, parameter = list(supp = 0.25, conf=0.7))
```

```
## Warning: Column(s) 11, 22, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 42 not
 ## logical or factor. Applying default discretization (see '?
discretizeDF').
## Apriori
 ##
## Parameter specification:
 ##  confidence minval smax arem  aval originalSupport maxtime support minlen
 ##         0.7    0.1    1 none FALSE            TRUE       5    0.25      1
 ##  maxlen target  ext
 ##      10  rules TRUE
 ##
## Algorithmic control:
 ##  filter tree heap memopt load sort verbose
 ##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
 ##
## Absolute minimum support count: 31513
##
## set item appearances ...[0 item(s)] done [0.00s].
 ## set transactions ...[4009 item(s), 126052 transaction(s)] done [1.60s].
 ## sorting and recoding items ... [51 item(s)] done [0.06s].
 ## creating transaction tree ... done [0.18s].
 ## checking subsets of size 1 2 3 4 5 6 7 8 9 10
## Warning in apriori(NBA, parameter = list(supp = 0.25, conf = 0.7)): Mining
 ## stopped (maxlen reached). Only patterns up to a length of 10 returned!
##  done [3.33s].
 ## writing ... [88775 rule(s)] done [1.63s].
 ## creating S4 object  ... done [0.12s].
summary(NBArules)
## set of 88775 rules
 ##
## rule length distribution (lhs + rhs):sizes
 ##     1     2     3     4     5     6     7     8     9    10
##     5   327  2378  7984 16060 21506 20006 13009  5802  1698
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   5.000   6.000   6.335   7.000  10.000
##
## summary of quality measures:
 ##     support         confidence        coverage           lift
 ##  Min.   :0.2502   Min.   :0.7104   Min.   :0.2502   Min.   :0.987
##  1st Qu.:0.2615   1st Qu.:0.9952   1st Qu.:0.2630   1st Qu.:1.000
##  Median :0.2682   Median :1.0000   Median :0.2745   Median :1.816
##  Mean   :0.2947   Mean   :0.9786   Mean   :0.3024   Mean   :1.596
##  3rd Qu.:0.3157   3rd Qu.:1.0000   3rd Qu.:0.3184   3rd Qu.:1.831
##  Max.   :0.9998   Max.   :1.0000   Max.   :1.0000   Max.   :3.803
##      count
##  Min.   : 31540
##  1st Qu.: 32964
##  Median : 33811
##  Mean   : 37152
```

```
##   3rd Qu.: 39790
##   Max.   :126026
##
## mining info:
 ##   data ntransactions support confidence
 ##    NBA        126052    0.25        0.7
inspect(NBArules[1:10])
##      lhs                       rhs                         support
confidence
 ## [1]  {}                      => {PTS_TYPE=2}               0.7352918
0.7352918
## [2]  {}                      => {CLOSE_DEF_DIST=Very Close} 0.9925031
0.9925031
## [3]  {}                      => {isShotAttempted}          0.9997937
0.9997937
## [4]  {}                      => {slugSeason=2014-15}       0.9997937
0.9997937
## [5]  {}                      => {yearSeason=2015}          0.9997937
0.9997937
## [6]  {PERIOD=3}              => {CLOSE_DEF_DIST=Very Close} 0.2508171
0.9916567
## [7]  {PERIOD=3}              => {isShotAttempted}          0.2528877
0.9998432
## [8]  {PERIOD=3}              => {slugSeason=2014-15}       0.2528877
0.9998432
## [9]  {PERIOD=3}              => {yearSeason=2015}          0.2528877
0.9998432
## [10] {zoneRange=24+ ft.} => {PTS_TYPE=3}                   0.2611065
0.9953428
##      coverage  lift      count
## [1]  1.0000000 1.0000000  92685
 ## [2]  1.0000000 1.0000000 125107
 ## [3]  1.0000000 1.0000000 126026
 ## [4]  1.0000000 1.0000000 126026
 ## [5]  1.0000000 1.0000000 126026
 ## [6]  0.2529274 0.9991472  31616
 ## [7]  0.2529274 1.0000494  31877
 ## [8]  0.2529274 1.0000494  31877
 ## [9]  0.2529274 1.0000494  31877
 ## [10] 0.2623282 3.7601507  32913
NBARM <- apriori(NBATest, parameter = list(supp = 0.25, conf=0.7, minlen =
1))
## Warning: Column(s) 33 not logical or factor. Applying default
discretization
 ## (see '? discretizeDF').
## Apriori
 ##
## Parameter specification:
 ##   confidence minval smax arem  aval originalSupport maxtime support minlen
 ##          0.7    0.1    1 none FALSE            TRUE       5    0.25      1
```

```
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 31513
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[4846 item(s), 126052 transaction(s)] done [1.29s].
## sorting and recoding items ... [30 item(s)] done [0.04s].
## creating transaction tree ... done [0.15s].
## checking subsets of size 1 2 3 4 5 6 7 8 9 10
## Warning in apriori(NBATest, parameter = list(supp = 0.25, conf = 0.7,
minlen
## = 1)): Mining stopped (maxlen reached). Only patterns up to a length of
10
## returned!
##  done [0.14s].
## writing ... [29821 rule(s)] done [0.54s].
## creating S4 object  ... done [0.07s].
summary(NBARM)
## set of 29821 rules
##
## rule length distribution (lhs + rhs):sizes
##    1    2    3    4    5    6    7    8    9   10
##    5  175 1290 4080 7162 7840 5640 2681  808  140
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   5.000   6.000   5.801   7.000  10.000
##
## summary of quality measures:
##     support          confidence        coverage            lift
##  Min.   :0.2506   Min.   :0.7106   Min.   :0.2506   Min.   :0.987
##  1st Qu.:0.2630   1st Qu.:0.9952   1st Qu.:0.2637   1st Qu.:1.000
##  Median :0.2682   Median :1.0000   Median :0.2732   Median :1.347
##  Mean   :0.3003   Mean   :0.9800   Mean   :0.3077   Mean   :1.466
##  3rd Qu.:0.2897   3rd Qu.:1.0000   3rd Qu.:0.3476   3rd Qu.:1.827
##  Max.   :0.9998   Max.   :1.0000   Max.   :1.0000   Max.   :2.783
##     count
##  Min.   : 31583
##  1st Qu.: 33155
##  Median : 33811
##  Mean   : 37848
##  3rd Qu.: 36518
##  Max.   :126026
##
## mining info:
```

```
##      data ntransactions support confidence
##   NBATest        126052    0.25        0.7
inspect(NBARM[1:20])
##      lhs                             rhs                           support
confidence  coverage      lift  count
## [1]  {}                            => {PTS_TYPE=2}
0.7352918  0.7352918 1.0000000 1.0000000  92685
## [2]  {}                            => {CLOSE_DEF_DIST=Very Close}
0.9925031  0.9925031 1.0000000 1.0000000 125107
## [3]  {}                            => {isShotAttempted}
0.9997937  0.9997937 1.0000000 1.0000000 126026
## [4]  {}                            => {slugSeason=2014-15}
0.9997937  0.9997937 1.0000000 1.0000000 126026
## [5]  {}                            => {yearSeason=2015}
0.9997937  0.9997937 1.0000000 1.0000000 126026
## [6]  {PERIOD=3}                    => {CLOSE_DEF_DIST=Very Close}
0.2508171  0.9916567 0.2529274 0.9991472  31616
## [7]  {PERIOD=3}                    => {isShotAttempted}
0.2528877  0.9998432 0.2529274 1.0000494  31877
## [8]  {PERIOD=3}                    => {slugSeason=2014-15}
0.2528877  0.9998432 0.2529274 1.0000494  31877
## [9]  {PERIOD=3}                    => {yearSeason=2015}
0.2528877  0.9998432 0.2529274 1.0000494  31877
## [10] {PERIOD=1}                    => {CLOSE_DEF_DIST=Very Close}
0.2615191  0.9925330 0.2634865 1.0000302  32965

NBARM <- sort(NBARM, decreasing = TRUE, by = "lift")
```

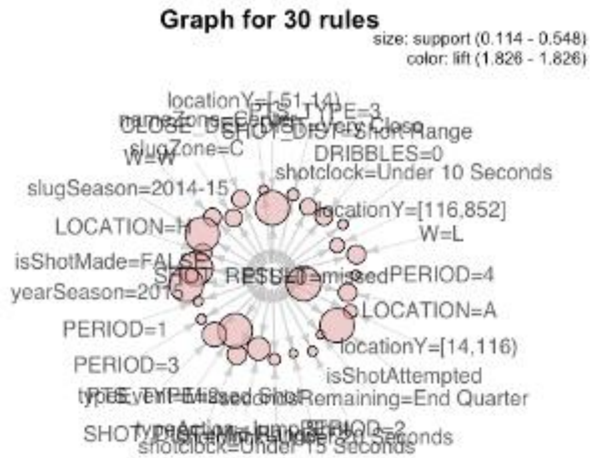
After the columns had been properly formatted, a second rule mining algorithm was run with the right-hand side of the rule specified as “W=W”. The left-hand side of these rules would be the factors that lead to won games. The results from this test and a plot of the top 20 rules, sorted by confidence, can be seen below.

```
NBArulesTest <- apriori(NBATest, parameter = list(conf = 0.08, supp = 0.01,
maxlen=15), appearance = list(rhs = c("W=W")))
## Warning: Column(s) 33 not logical or factor. Applying default
discretization
## (see '? discretizeDF').
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##        0.08    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target  ext
##      15  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
```

```
##
## Absolute minimum support count: 1260
##
## set item appearances ...[1 item(s)] done [0.00s].
## set transactions ...[4846 item(s), 126052 transaction(s)] done [1.37s].
## sorting and recoding items ... [305 item(s)] done [0.08s].
## creating transaction tree ... done [0.14s].
## checking subsets of size 1 2 3 4
## Warning in apriori(NBATest, parameter = list(conf = 0.08, supp = 0.01,
maxlen
## = 15), : Mining stopped (time limit reached). Only patterns up to a
length of 4
## returned!
##  done [5.26s].
## writing ... [25056 rule(s)] done [0.46s].
## creating S4 object  ... done [0.11s].
summary(NBArulesTest)
## set of 25056 rules
##
## rule length distribution (lhs + rhs):sizes
##     1     2     3     4
##     1   252  3527 21276
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   4.000   4.000   3.839   4.000   4.000
##
## summary of quality measures:
##      support          confidence        coverage            lift
##  Min.   :0.01000   Min.   :0.3062   Min.   :0.01210   Min.   :0.6071
##  1st Qu.:0.01227   1st Qu.:0.4909   1st Qu.:0.02264   1st Qu.:0.9732
##  Median :0.01678   Median :0.5107   Median :0.03324   Median :1.0125
##  Mean   :0.02917   Mean   :0.5336   Mean   :0.05685   Mean   :1.0580
##  3rd Qu.:0.02903   3rd Qu.:0.5430   3rd Qu.:0.05758   3rd Qu.:1.0766
##  Max.   :0.50439   Max.   :0.9215   Max.   :1.00000   Max.   :1.8269
##      count
##  Min.   : 1261
##  1st Qu.: 1547
##  Median : 2115
##  Mean   : 3677
##  3rd Qu.: 3660
##  Max.   :63579
##
## mining info:
##     data ntransactions support confidence
##  NBATest        126052    0.01       0.08
NBArulesTest <- sort(NBArulesTest, decreasing = TRUE, by = "confidence")
inspect(NBArulesTest[1:30])
##      lhs                           rhs      support confidence   coverage
lift count
## [1]  {idTeam=1610612744,
```

```
##       slugTeamHome=GSW}            => {W=W} 0.01508108  0.9214736 0.01636626
1.826918  1901
 ## [2]  {team= GSW ,
 ##       slugTeamHome=GSW}            => {W=W} 0.01508108  0.9214736
0.01636626 1.826918  1901
 ## [3]  {LOCATION=H,
##       slugTeamHome=GSW}            => {W=W} 0.01508108  0.9214736 0.01636626
1.826918  1901
 ## [4]  {LOCATION=H,
##       idTeam=1610612744}           => {W=W} 0.01508108  0.9214736 0.01636626
1.826918  1901
 ## [5]  {team= GSW ,
##       LOCATION=H}                  => {W=W} 0.01508108  0.9214736 0.01636626
1.826918  1901
 ## [6]  {team= GSW ,
##       idTeam=1610612744,
##       slugTeamHome=GSW}            => {W=W} 0.01508108  0.9214736 0.01636626
1.826918  1901
 ## [7]  {LOCATION=H,
##       idTeam=1610612744,
##       slugTeamHome=GSW}            => {W=W} 0.01508108  0.9214736 0.01636626
1.826918  1901
 ## [8]  {yearSeason=2015,
##       idTeam=1610612744,
##       slugTeamHome=GSW}            => {W=W} 0.01508108  0.9214736 0.01636626
1.826918  1901
 ## [9]  {idTeam=1610612744,
##       slugTeamHome=GSW,
 ##       isShotAttempted}             => {W=W} 0.01508108  0.9214736
0.01636626 1.826918  1901
 ## [10] {slugSeason=2014-15,
##       idTeam=1610612744,
 ##       slugTeamHome=GSW}            => {W=W} 0.01508108  0.9214736
0.01636626 1.826918  1901
```

```
plot(NBArulesTest[1:30], method = "graph")
```



A third apriori test was run, this time with the right-hand side of the rule specified as "SHOT_RESULT=missed". The left hand side of these rules would be the factors that lead to missed shots. The results from this test and a plot of the top 20 rules, sorted by confidence, can be seen below.

```
NBArulesTest1 <- apriori(NBATest, parameter = list(conf = 0.5, supp = 0.1,
maxlen=15), appearance = list(rhs = c("SHOT_RESULT=missed")))
## Warning: Column(s) 33 not logical or factor. Applying default
discretization
 ## (see '? discretizeDF').
## Apriori
 ##
## Parameter specification:
 ##  confidence minval smax arem  aval originalSupport maxtime support minlen
 ##         0.5    0.1    1 none FALSE            TRUE       5     0.1      1
 ##  maxlen target  ext
 ##      15  rules TRUE
 ##
## Algorithmic control:
 ##  filter tree heap memopt load sort verbose
 ##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
 ##
## Absolute minimum support count: 12605
##
## set item appearances ...[1 item(s)] done [0.00s].
 ## set transactions ...[4846 item(s), 126052 transaction(s)] done [1.29s].
 ## sorting and recoding items ... [59 item(s)] done [0.05s].
 ## creating transaction tree ... done [0.17s].
 ## checking subsets of size 1 2 3 4 5 6
## Warning in apriori(NBATest, parameter = list(conf = 0.5, supp = 0.1,
maxlen =
 ## 15), : Mining stopped (time limit reached). Only patterns up to a length
of 6
 ## returned!
```

```
##  done [5.97s].
 ## writing ... [7937 rule(s)] done [0.15s].
 ## creating S4 object  ... done [0.11s].
summary(NBArulesTest1)
## set of 7937 rules
 ##
## rule length distribution (lhs + rhs):sizes
 ##    1    2    3    4    5    6
##    1   26  215  931 2466 4298
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    5.00    6.00    5.36    6.00    6.00
##
## summary of quality measures:
 ##     support          confidence        coverage           lift
##  Min.   :0.1001   Min.   :0.5024   Min.   :0.1002   Min.   :0.9173
##  1st Qu.:0.1169   1st Qu.:0.9947   1st Qu.:0.1237   1st Qu.:1.8162
##  Median :0.1362   Median :0.9966   Median :0.1557   Median :1.8195
##  Mean   :0.1673   Mean   :0.9283   Mean   :0.1867   Mean   :1.6949
##  3rd Qu.:0.1928   3rd Qu.:1.0000   3rd Qu.:0.2336   3rd Qu.:1.8258
##  Max.   :0.5477   Max.   :1.0000   Max.   :1.0000   Max.   :1.8258
##      count
##  Min.   :12613
##  1st Qu.:14732
##  Median :17167
##  Mean   :21090
##  3rd Qu.:24306
##  Max.   :69039
##
## mining info:
 ##     data ntransactions support confidence
 ##  NBATest        126052     0.1        0.5
NBArulesTest1 <- sort(NBArulesTest1, decreasing = TRUE, by = "lift")
 inspect(NBArulesTest1[1:30])
##     lhs                               rhs                     support
confidence  coverage     lift count
 ## [1]  {PTS=0}                        => {SHOT_RESULT=missed} 0.5477025
1 0.5477025 1.825809 69039
 ## [2]  {PTS=0,
##       secondsRemaining=End Quarter} => {SHOT_RESULT=missed} 0.1140720
1 0.1140720 1.825809 14379
 ## [3]  {PTS=0,
##       shotclock=Under 20 Seconds}   => {SHOT_RESULT=missed} 0.1137705
1 0.1137705 1.825809 14341
 ## [4]  {PERIOD=4,
 ##       PTS=0}                        => {SHOT_RESULT=missed} 0.1280186
1 0.1280186 1.825809 16137
 ## [5]  {PTS=0,
 ##       shotclock=Under 10 Seconds}   => {SHOT_RESULT=missed} 0.1367293
1 0.1367293 1.825809 17235
```

```
## [6]  {PERIOD=2,
##       PTS=0}                          => {SHOT_RESULT=missed} 0.1351506
1 0.1351506 1.825809 17036
## [7]  {PERIOD=3,
##       PTS=0}                          => {SHOT_RESULT=missed} 0.1372449
1 0.1372449 1.825809 17300
## [8]  {PERIOD=1,
##       PTS=0}                          => {SHOT_RESULT=missed} 0.1419573
1 0.1419573 1.825809 17894
## [9]  {PTS_TYPE=3,
##       PTS=0}                          => {SHOT_RESULT=missed} 0.1716752
1 0.1716752 1.825809 21640
## [10] {PTS=0,
##       shotclock=Under 15 Seconds}     => {SHOT_RESULT=missed} 0.1684146
1 0.1684146 1.825809 21229
```

```
plot(NBArulesTest1[1:30], method = "graph")
```

A fourth apriori test was run, this time with the right-hand side of the rule specified as "SHOT_RESULT=made". The left hand side of these rules would be the factors that lead to lost games. The results from this test and a plot of the top 20 rules, sorted by confidence, can be seen below.

```
NBArulesTest2 <- apriori(NBATest, parameter = list(conf = 0.8, supp = 0.001,
maxlen=15), appearance = list(rhs = c("SHOT_RESULT=made")))
## Warning: Column(s) 33 not logical or factor. Applying default
discretization
## (see '? discretizeDF').
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5   0.001      1
##  maxlen target  ext
##      15  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 126
##
## set item appearances ...[1 item(s)] done [0.00s].
## set transactions ...[4846 item(s), 126052 transaction(s)] done [1.27s].
## sorting and recoding items ... [3806 item(s)] done [0.09s].
## creating transaction tree ... done [0.12s].
## checking subsets of size 1 2 3 4
## Warning in apriori(NBATest, parameter = list(conf = 0.8, supp = 0.001,
maxlen
## = 15), : Mining stopped (time limit reached). Only patterns up to a
```

```
length of 4
 ## returned!
##  done [22.45s].
 ## writing ... [98925 rule(s)] done [2.03s].
 ## creating S4 object  ... done [0.24s].
summary(NBArulesTest2)
## set of 98925 rules
 ##
## rule length distribution (lhs + rhs):sizes
 ##     2     3     4
##    12  4692 94221
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   4.000   4.000   3.952   4.000   4.000
##
## summary of quality measures:
 ##     support          confidence        coverage             lift
##  Min.   :0.001007   Min.   :0.8000   Min.   :0.001007   Min.   :1.769
##  1st Qu.:0.001285   1st Qu.:0.9938   1st Qu.:0.001293   1st Qu.:2.197
##  Median :0.001817   Median :1.0000   Median :0.001833   Median :2.211
##  Mean   :0.005118   Mean   :0.9928   Mean   :0.005146   Mean   :2.195
##  3rd Qu.:0.003332   3rd Qu.:1.0000   3rd Qu.:0.003364   3rd Qu.:2.211
##  Max.   :0.449894   Max.   :1.0000   Max.   :0.452313   Max.   :2.211
##      count
##  Min.   :  127.0
##  1st Qu.:  162.0
##  Median :  229.0
##  Mean   :  645.1
##  3rd Qu.:  420.0
##  Max.   :56710.0
##
## mining info:
 ##     data ntransactions support confidence
 ##  NBATest        126052   0.001        0.8
NBArulesTest2 <- sort(NBArulesTest2, decreasing = TRUE, by = "lift")
 inspect(NBArulesTest2[1:30])
##      lhs                                    rhs                     support
confidence    coverage     lift count
 ## [1]  {PTS=3}                             => {SHOT_RESULT=made}
0.093033034          1 0.093033034 2.210934 11727
 ## [2]  {PTS=2}                             => {SHOT_RESULT=made}
0.359264431          1 0.359264431 2.210934 45286
 ## [3]  {PTS=2,
##        typeAction=Running Slam Dunk Shot} => {SHOT_RESULT=made} 0.001047187
1 0.001047187 2.210934   132
 ## [4]  {PTS=2,
##        typeAction=Running Dunk Shot}      => {SHOT_RESULT=made} 0.001118586
1 0.001118586 2.210934   141
 ## [5]  {PTS=2,
##        typeAction=Turnaround Bank shot}   => {SHOT_RESULT=made} 0.001023387
```

```
1 0.001023387 2.210934   129
 ## [6]  {PTS=2,
##       player_name=john henson}           => {SHOT_RESULT=made} 0.001039254
1 0.001039254 2.210934   131
 ## [7]  {player_name=john henson,
##       isShotMade=TRUE}                   => {SHOT_RESULT=made} 0.001039254
1 0.001039254 2.210934   131
 ## [8]  {player_name=john henson,
##       typeEvent=Made Shot}               => {SHOT_RESULT=made} 0.001039254
1 0.001039254 2.210934   131
 ## [9]  {PTS=2,
 ##       player_id=203089}                  => {SHOT_RESULT=made}
0.001039254          1 0.001039254 2.210934   131
 ## [10] {player_id=203089,
 ##       isShotMade=TRUE}                   => {SHOT_RESULT=made}
0.001039254          1 0.001039254 2.210934   131
```

```
plot(NBArulesTest2[1:30], method = "graph")
```

Results

When performing the initial Apriori analysis, several combinations of support and confidence were used to filter through the possible rules for the National Basketball Association dataset.

To start, the second Apriori analysis, with the right-hand side (RHS) of the rule specified as “W=W”, will be broken down. To read these rules, you read from left to right just like you would a book. The items on the left-hand side (LHS) of the rule are attributes that lead to a game being won (RHS). ***25,056*** rules were created by this analysis. The top 10 will be discussed below.

```
##      lhs                              rhs      support confidence   coverage
lift count
## [1]  {idTeam=1610612744,
##       slugTeamHome=GSW}         => {W=W} 0.01508108  0.9214736 0.01636626
```

```
1.826918  1901
## [2]  {team= GSW ,
##       slugTeamHome=GSW}         => {W=W} 0.01508108  0.9214736 0.01636626
1.826918  1901
## [3]  {LOCATION=H,
##       slugTeamHome=GSW}         => {W=W} 0.01508108  0.9214736 0.01636626
1.826918  1901
## [4]  {LOCATION=H,
##       idTeam=1610612744}        => {W=W} 0.01508108  0.9214736 0.01636626
1.826918  1901
## [5]  {team= GSW ,
##       LOCATION=H}               => {W=W} 0.01508108  0.9214736 0.01636626
1.826918  1901
## [6]  {team= GSW ,
##       idTeam=1610612744,
##       slugTeamHome=GSW}         => {W=W} 0.01508108  0.9214736 0.01636626
1.826918  1901
## [7]  {LOCATION=H,
##       idTeam=1610612744,
##       slugTeamHome=GSW}         => {W=W} 0.01508108  0.9214736 0.01636626
1.826918  1901
## [8]  {yearSeason=2015,
##       idTeam=1610612744,
##       slugTeamHome=GSW}         => {W=W} 0.01508108  0.9214736 0.01636626
1.826918  1901
## [9]  {idTeam=1610612744,
##       slugTeamHome=GSW,
##       isShotAttempted}          => {W=W} 0.01508108  0.9214736 0.01636626
1.826918  1901
## [10] {slugSeason=2014-15,
##       idTeam=1610612744,
##       slugTeamHome=GSW}         => {W=W} 0.01508108  0.9214736 0.01636626
```

The rules were sorted by lift. After scrolling through the rules, a great number of them had a confidence level above **0.90**. Of the 10 rules being discussed above, **4 of the 10** rules contain the date and location of the game. Golden State appears as the home team in **8 of the 10** rules.

The next Apriori analysis to be analyzed was when the RHS was specified as "SHOT_RESULT = missed". **7,937** rules were created by this analysis. The top 10 will be discussed below.

```
##      lhs                                  rhs                   support
confidence  coverage     lift count
## [1]  {PTS=0}                           => {SHOT_RESULT=missed} 0.5477025
1 0.5477025 1.825809 69039
## [2]  {PTS=0,
##       secondsRemaining=End Quarter} => {SHOT_RESULT=missed} 0.1140720
1 0.1140720 1.825809 14379
## [3]  {PTS=0,
##       shotclock=Under 20 Seconds}   => {SHOT_RESULT=missed} 0.1137705
1 0.1137705 1.825809 14341
## [4]  {PERIOD=4,
##       PTS=0}                        => {SHOT_RESULT=missed} 0.1280186
1 0.1280186 1.825809 16137
## [5]  {PTS=0,
##       shotclock=Under 10 Seconds}   => {SHOT_RESULT=missed} 0.1367293
1 0.1367293 1.825809 17235
## [6]  {PERIOD=2,
##       PTS=0}                        => {SHOT_RESULT=missed} 0.1351506
1 0.1351506 1.825809 17036
## [7]  {PERIOD=3,
##       PTS=0}                        => {SHOT_RESULT=missed} 0.1372449
1 0.1372449 1.825809 17300
## [8]  {PERIOD=1,
##       PTS=0}                        => {SHOT_RESULT=missed} 0.1419573
1 0.1419573 1.825809 17894
## [9]  {PTS_TYPE=3,
##       PTS=0}                        => {SHOT_RESULT=missed} 0.1716752
1 0.1716752 1.825809 21640
## [10] {PTS=0,
##       shotclock=Under 15 Seconds}   => {SHOT_RESULT=missed} 0.1684146
1 0.1684146 1.825809 21229
```

After scrolling through the rules, a great number of them had a confidence level of 1. As expected, one of the contributing attributes to a missed shot was that "PTS = 0". Some of the other contributing factors to a missed shot included the shot clock being under 20 seconds, the period in which the shot was taken, and the seconds remaining in the quarter. Some of the other rules, not depicted above, included the closeness of the defender, the type of shot that was taken, and the zone from which the shot was taken.

The next Apriori analysis to be analyzed was when the RHS was specified as "SHOT_RESULT = made". **98,925** rules were created by this analysis. The top 10 will be discussed below.

```
##      lhs                                    rhs                 support
confidence    coverage     lift count
## [1]  {PTS=3}                            => {SHOT_RESULT=made} 0.093033034
```

```
1 0.093033034 2.210934 11727
## [2]  {PTS=2}                               => {SHOT_RESULT=made} 0.359264431
1 0.359264431 2.210934 45286
## [3]  {PTS=2,
##       typeAction=Running Slam Dunk Shot} => {SHOT_RESULT=made} 0.001047187
1 0.001047187 2.210934   132
## [4]  {PTS=2,
##       typeAction=Running Dunk Shot}      => {SHOT_RESULT=made} 0.001118586
1 0.001118586 2.210934   141
## [5]  {PTS=2,
##       typeAction=Turnaround Bank shot}   => {SHOT_RESULT=made} 0.001023387
1 0.001023387 2.210934   129
## [6]  {PTS=2,
##       player_name=john henson}           => {SHOT_RESULT=made} 0.001039254
1 0.001039254 2.210934   131
## [7]  {player_name=john henson,
##       isShotMade=TRUE}                   => {SHOT_RESULT=made} 0.001039254
1 0.001039254 2.210934   131
## [8]  {player_name=john henson,
##       typeEvent=Made Shot}               => {SHOT_RESULT=made} 0.001039254
1 0.001039254 2.210934   131
## [9]  {PTS=2,
##       player_id=203089}                  => {SHOT_RESULT=made} 0.001039254
1 0.001039254 2.210934   131
## [10] {player_id=203089,
##       isShotMade=TRUE}                   => {SHOT_RESULT=made} 0.001039254
1 0.001039254 2.210934   131
```

The rules were sorted by lift. After scrolling through the rules, a great number of them had a confidence level of 1. Of the 10 rules being discussed above, **6 of the 10** rules contain two-point shots and **1** contained three-point shots. The type of shot also played a role in whether or not a shot would be made, whether it was a **running slam dunk or a turn-around bank shot**. Lastly, of the rules being examined above, **John Henson** was the player likely to make the shot.

Clustering Analysis

This analysis attempts to categorize NBA players into types. Professional basketball is often said to be an increasingly position-less game. While the six-foot-eleven Kevin Durant may be tall enough to play center, he is listed as a power forward, but often plays like a guard. To cluster players based on more tangible gameplay factors, the analysis was based upon player shot selection, and relative accuracy within each of 5 zones on the court, along with other characteristics such as height and weight. Then, a decision tree model is used to predict season win totals for teams given their combinations of player type. To increase accuracy, 8 seasons of shooting data were taken from the nbastatR package, which scrapes the NBA's advanced stats from their website. The 2019-20 season was excluded because of complications from the COVID-19 pandemic.

Pre-processing

```
## load in shots data from nbastatR package
shots_2011 <- teams_shots(all_active_teams = TRUE, seasons = 2011,
                          season_types = 'Regular Season')

shots_2012 <- teams_shots(all_active_teams = TRUE, seasons = 2012,
                          season_types = 'Regular Season')

shots_2013 <- teams_shots(all_active_teams = TRUE, seasons = 2013,
                          season_types = 'Regular Season')

shots_2014 <- teams_shots(all_active_teams = TRUE, seasons = 2014,
                          season_types = 'Regular Season')

shots_2016 <- teams_shots(all_active_teams = TRUE, seasons = 2016,
                          season_types = 'Regular Season')

shots_2017 <- teams_shots(all_active_teams = TRUE, seasons = 2017,
                          season_types = 'Regular Season')

shots_2018 <- teams_shots(all_active_teams = TRUE, seasons = 2018,
                     season_types = 'Regular Season')
```

The rows from each dataset are bound together.

```
## bind shots together for 1,601,750 total shots
shots <- rbind(shots_2011, shots_2012, shots_2013,
               shots_2014, shots_2015, shots_2016,
               shots_2017, shots_2018)
```

All the individual shot datasets by season are removed from memory to improve memory constraints in later calculations.

```
## drop all the individual tables from memory
remove(list = c('shots_2011', 'shots_2012', 'shots_2013',
```

```
                'shots_2014', 'shots_2015', 'shots_2016',
                'shots_2017', 'shots_2018'))
```

The rows from each dataset are bound together and modified using dplyr's piping style. Points gained from each shot attempt are obtained by looking at whether a shot attempt was from 2/3 point range, and whether it went in. This will be used later to calculate Expected Points per Attempt from each zone on the court. Next, zones are modified by combining shots from the left corner and right corner. Appropriate variables are then converted to factors using mutate_at() and mutate_if().

```
shots <- shots %>%
  ## convert shot result column to num
  mutate(made = as.numeric(isShotMade)) %>%
  ## variable for points actually scored on attempt
  mutate(pts = case_when(
    typeShot == '3PT Field Goal' & isShotMade == TRUE ~ 3,
    typeShot == '2PT Field Goal' & isShotMade == TRUE ~ 2,
    TRUE ~ 0)) %>%
  ## merge left and right 3s to one zone
  mutate(zoneBasic = case_when(
    zoneBasic == 'Left Corner 3' ~ 'Corner 3',
    zoneBasic == 'Right Corner 3' ~ 'Corner 3',
    TRUE ~ zoneBasic)) %>%
  ## all characters to factors
  mutate_if(is.character, as.factor) %>%
  ## some numbers to factors
  mutate_at(c('idTeam', 'idPlayer', 'yearSeason', 'numberPeriod',
              'idGame'), as.factor)
```

To strengthen the clustering of player types, both height and weight are brought in from a separate Kaggle dataset. Next, a third dataset with roster positions is incorporated, simply for visualization later on. Unfortunately, this data could not be found in one single package. Both of these datasets are merged with the shots dataset using left_join().

```
## https://www.kaggle.com/justinas/nba-height-and-weight-analysis/?select=all
_seasons.csv
heights <- read_csv('C:/Users/marmesto/Documents/Cuse/Data Analytics/Project/
all_seasons.csv')

heights <- heights %>%
  select(player_name, player_height, player_weight, 15:22) %>%
  mutate_at(c('player_name', 'season'), as.factor)

shots <- left_join(shots, heights, by = c('namePlayer' = 'player_name',
                                          'slugSeason' = 'season'))

## read in another dataset with player roster positions
## https://www.kaggle.com/drgilermo/nba-players-stats?select=player_data.csv
positions <- read_csv('C:/Users/marmesto/Documents/Cuse/Data Analytics/Projec
t/datasets_1358_30676_player_data.csv')
```

```r
positions <- positions %>%
  select(name, position) %>%
  mutate_at(c('name', 'position'), as.factor)

shots <- left_join(shots, positions, by = c('namePlayer' = 'name'))
```

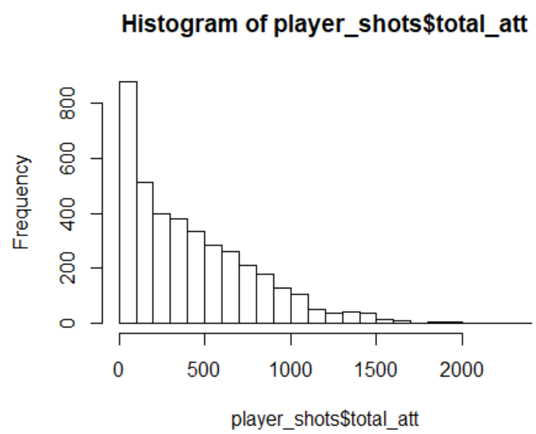
Next, a new data frame is created from shots. In player_shots, each row will consist of a single player and their stats for the season, including stats within each zone. Note that players with multiple seasons in the dataset will essentially be treated as different players for each season. Every season, a player may change their style of play, and thus fit into a different role or cluster.

To start, all backcourt shots are removed. Though fun to watch, these shots are often made in desperation as the final seconds of game or half-time count down. Then, rows are grouped by player and year, and total shot attempts for that year are calculated. The rows are then grouped by player, year, and zone. Accuracy, expected points per attempt, and percentage of total attempts within each zone are determined.

Using pivot_wider(), a wide dataframe is created with separate columns for each zone's relevant stats, along with other player stats. To filter out players with less data, a histogram is created to look at each player's total attempts. The minimum was set at 300 shots in a season.

```r
## new dataframe where each player/season's shots are
## broken out by stats in each zone
## just one row per player/season
player_shots <- shots %>%
  ## no deep shots, they're usually desperate
  filter(zoneBasic != 'Backcourt') %>%
  ## group by player and season
  group_by(idPlayer, yearSeason) %>%
  ## create a total attempts stat to divide by
  mutate(total_att = length(idPlayer)) %>%
  ## ungroup now that we have total attempts
  ungroup() %>%
  ## lets work per player, per zone
  group_by(idPlayer, zoneBasic, yearSeason) %>%
  ## accuracy in each zone
  mutate(zone_acc = mean(made)*100) %>%
  ## expected points per attempt, in each zone
  mutate(zone_epa = mean(pts)) %>%
  ## attempts in each zone
  mutate(zone_att = length(made)) %>%
  ## percent of attempts taken in each zone
  mutate(zone_attpct = (zone_att/total_att)*100) %>%
  distinct(idPlayer, zoneBasic, yearSeason,.keep_all = TRUE) %>%
  select(namePlayer, idPlayer, yearSeason, position, zoneBasic, 30:44) %>%
  ## create wide datset, only one row for each player
  ## column for each zone's stats
  pivot_wider(names_from = zoneBasic,
```

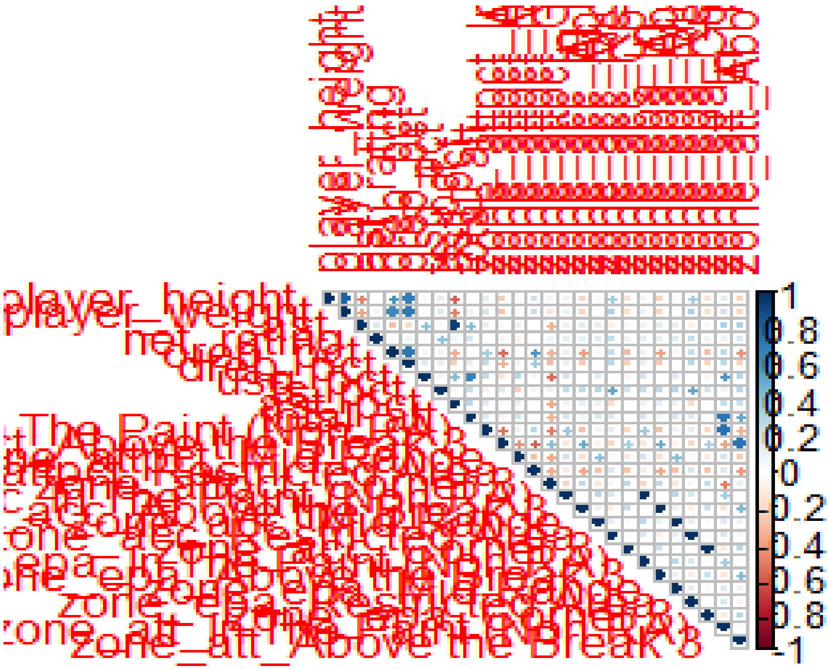
```
                values_from = c(zone_attpct, zone_acc,
                                zone_epa, zone_att))

## investigate average shots taken
hist(player_shots$total_att, breaks = 20)
```




```
## keep only players with 300+ shot seasons
player_shots <- player_shots %>%
  filter(total_att > 300) %>%
  ungroup() %>%
  drop_na()

## ensure no incomplete cases
nrow(player_shots[!complete.cases(player_shots),])

## [1] 0
```

Since there are 30 variables to choose from (excluding name, ID, year, and roster position), a correlation matrix is created to assess for potential collinearity.

```
shots_cor <- cor(player_shots[5:31])
corrplot::corrplot(shots_cor, type = 'upper', )
```



Principal Component Analysis

Since many variables show high correlation with each other, principal component analysis (PCA) is performed to reduce dimensionality.

A new dataframe is created where each variable is scaled. Identifying variables, like names are dropped. Variables directly related to player quality, like usage percent, and net rating are also dropped. This is to keep the analysis focused on play style rather than player quality, even though it may later impede the ability to predict wins. Not every team can have Steph Curry, but another player who also spreads the perimeter and takes deep shots may still contribute to success.

PCA is performed, and summary results are obtained. Here, only the first 3 dimensions are fully displayed shown for conciseness.

```
## create new scaled data frame
## drop some obvious skill measures like net_rating
player_shots2 <- data.frame(scale(select(player_shots,
                                  -c(namePlayer, idPlayer,
                                     total_att, yearSeason, position,
                                     net_rating, usg_pct, ts_pct, ast,
                                     starts_with('zone_epa'),
                                     starts_with('zone_att_')))))

## perform PCA
pca <- prcomp(player_shots2)
summary(pca)
```

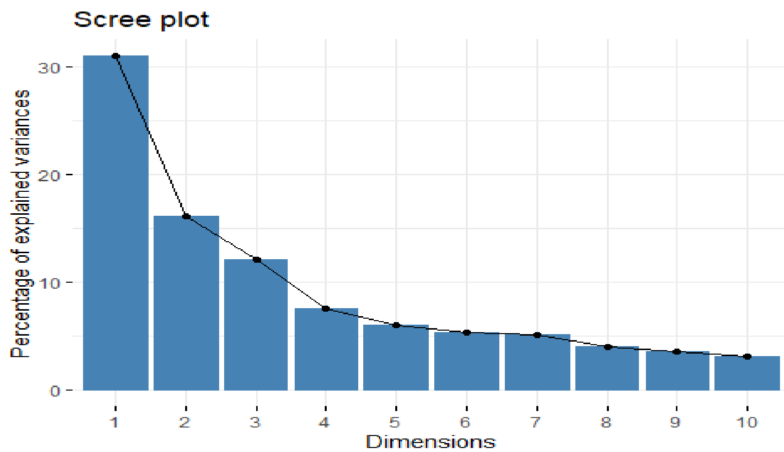
```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6
## Standard deviation     2.1557 1.5564 1.3483 1.06133 0.94816 0.89192
## Proportion of Variance 0.3098 0.1615 0.1212 0.07509 0.05993 0.05303
## Cumulative Proportion  0.3098 0.4713 0.5925 0.66757 0.72751 0.78054
##                            PC7     PC8     PC9    PC10    PC11    PC12
## Standard deviation     0.87883 0.76958 0.72785 0.68110 0.57924 0.50851
## Proportion of Variance 0.05149 0.03948 0.03532 0.03093 0.02237 0.01724
## Cumulative Proportion  0.83203 0.87152 0.90683 0.93776 0.96013 0.97737
##                           PC13    PC14      PC15
## Standard deviation     0.44037 0.38155 8.521e-16
## Proportion of Variance 0.01293 0.00971 0.000e+00
## Cumulative Proportion  0.99029 1.00000 1.000e+00
```

```
## observe contributions of all variables
pca_var <- get_pca_var(pca)
pca_var$contrib[,1:5]
```

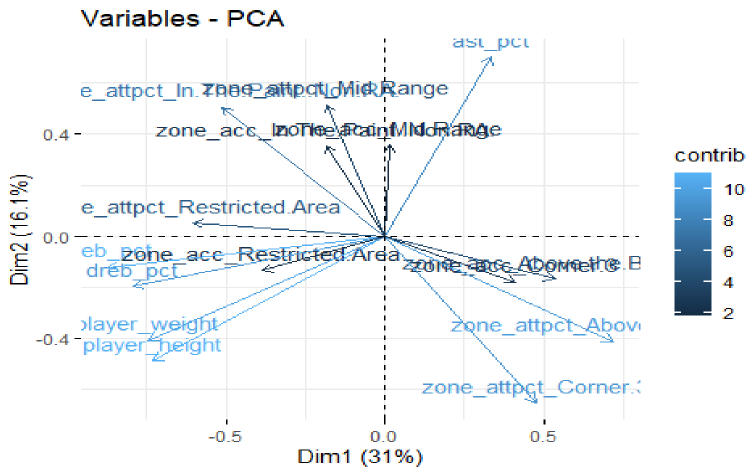
```
##                                          Dim.1      Dim.2      Dim.3
## player_height                     11.505400146  9.9667773  4.0033627
## player_weight                     11.993377117  6.9791674  3.7527832
## oreb_pct                          16.490596307  0.6161065  0.4256865
## dreb_pct                          13.577519053  1.6132111  0.7097490
## ast_pct                            2.444030609 20.4851669  2.6616744
## zone_attpct_In.The.Paint..Non.RA.  5.618995267 10.6152147  0.1587528
## zone_attpct_Above.the.Break.3     11.159372167  7.1390938  0.8282484
## zone_attpct_Mid.Range              0.708098366 11.0073961 15.2366179
## zone_attpct_Restricted.Area        7.857548590  0.1077322 22.8330288
## zone_attpct_Corner.3               4.848617048 17.7478313  0.2228929
## zone_acc_In.The.Paint..Non.RA.     0.728032560  5.1721060 12.8450839
## zone_acc_Above.the.Break.3         6.188948574  1.1910763  3.8370613
## zone_acc_Mid.Range                 0.005306691  5.2438084 25.1247825
## zone_acc_Restricted.Area           3.200840931  0.7581777  4.7605562
## zone_acc_Corner.3                  3.673316574  1.3571344  2.5997195
```

Here the dimensions are visualized. Using the first plot, one can see there is a significant drop off in explained variance after the 7th component. Keeping the first 7 components explains 83.2% of the variance, which was deemed adequate for the model.

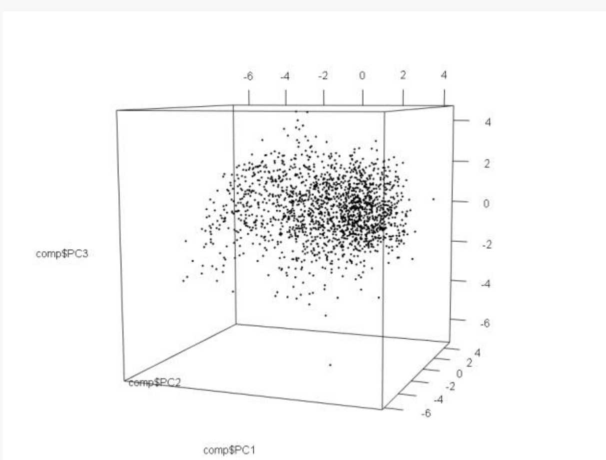
```
## plot effect of dimensions
fviz_eig(pca)
```




```r
fviz_pca_var(pca, col.var = 'contrib')
```




```r
comp <- data.frame(pca$x[,1:7])
plot3d(comp$PC1, comp$PC2, comp$PC3)
```

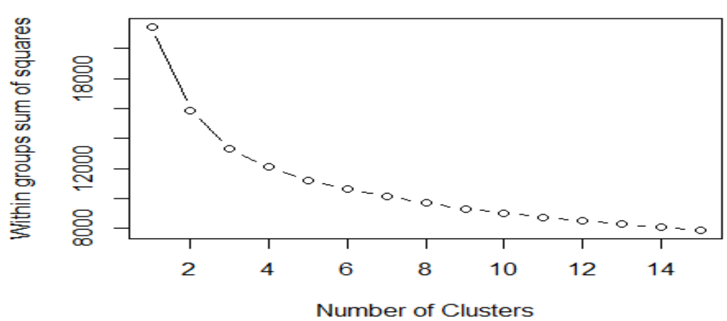


In order to perform K-means clustering, a number of clusters must be selected. Here, the weighted sum of square is calculated and plotted to look for an elbow in the graph. No obvious elbow is found, but one can see that there are very diminishing returns after 4 clusters.

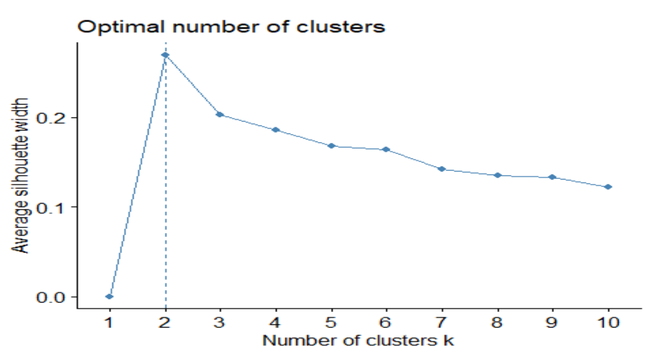
The Silhouette method was also used, but this method hinted that 2 clusters were ideal. Unfortunately, this did not seem like enough clusters for an interesting analysis, so it was decided that 4 clusters would be used.

K-Means Clustering

```
## kmeans
## determine number of clusters via weighted sum of squares
## similar to this example:
## https://github.com/mylesmharrison/delta_PCA_kmeans/blob/master/delta.R

## try elbow to determine number of clusters
wss <- (nrow(comp)-1)*sum(apply(comp,2,var))
for (i in 2:15) wss[i] <- sum(kmeans(comp, centers=i,
                                     nstart=100,
                                     iter.max=1000)$withinss)
plot(1:15, wss, type="b", xlab="Number of Clusters",
     ylab="Within groups sum of squares")
```



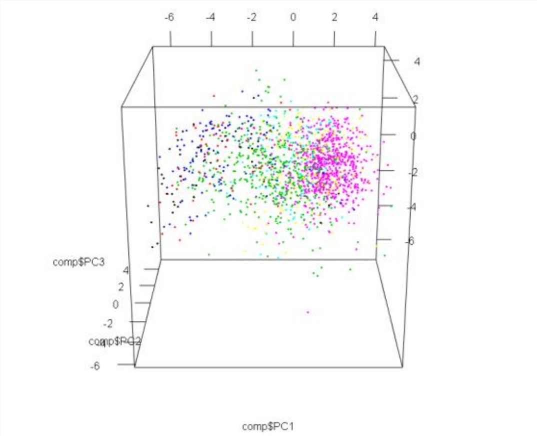

```
## silhouette method to determine number of clusters
fviz_nbclust(comp, kmeans, method = 'silhouette')
```



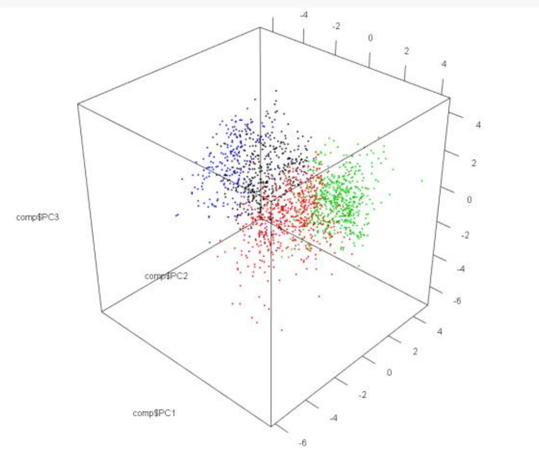
```
clusters <- 4
k <- kmeans(comp, clusters)
k$tot.withinss

## [1] 12100.54

k$size

## [1] 393 556 556 215
```

For a quick visual, the data is plotted with colors from the actual roster positions, and then plotted again by the k-means clusters.

```
## visualize data grouped by rosters positions
player_shots$position <- as.numeric(player_shots$position)
plot3d(comp$PC1, comp$PC2, comp$PC3, col = player_shots$position)
```




```
## visualize data grouped by our clusters
plot3d(comp$PC1, comp$PC2, comp$PC3,col = k$cluster)
```

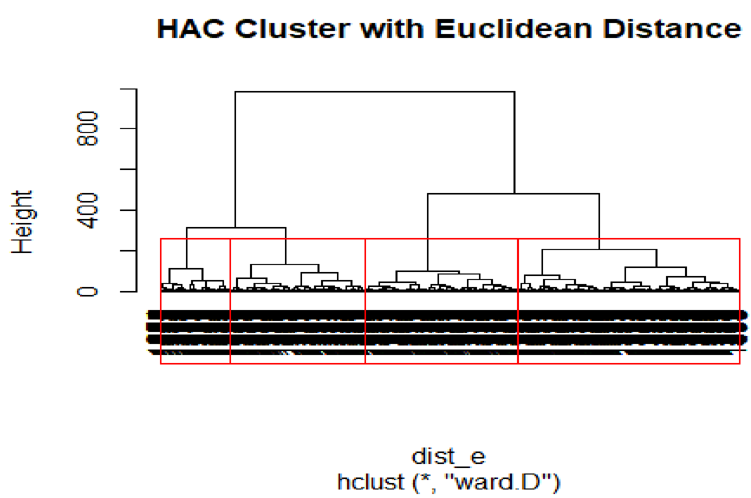


With these visuals, though somewhat hard to see in a 2-dimensional space, one can see that the traditional roster positions are somewhat arbitrary, and not defined by actual player performance. The clustering technique creates much more distinctive player types.

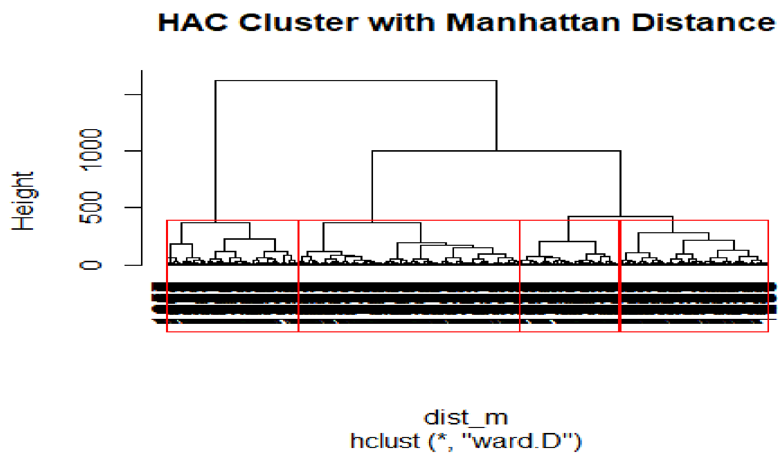
Hierarchical Clustering

Next, hierarchical clustering is performed. Euclidean distance, manhattan distance, and cosine are all evaluated. Here, the 'single' and 'complete' methods are not included, as they were not found to work well.

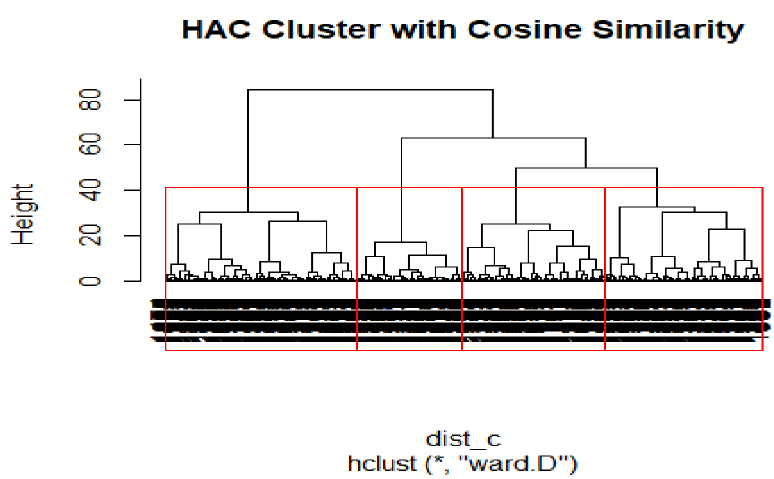
```
## calculate distances
dist_e <- dist(comp, method = 'euclidean')
dist_m <- dist(comp, method = 'manhattan')
dist_c <- dist(comp, method = 'cosine')

## for report conciseness, only including 'ward.D' method here
## single and complete methods didn't work well

## euclidean
hclust_e3<- hclust(dist_e, method = 'ward.D')
plot(hclust_e3, hang = -1, main = 'HAC Cluster with Euclidean Distance')
rect.hclust(hclust_e3, k = clusters)
```



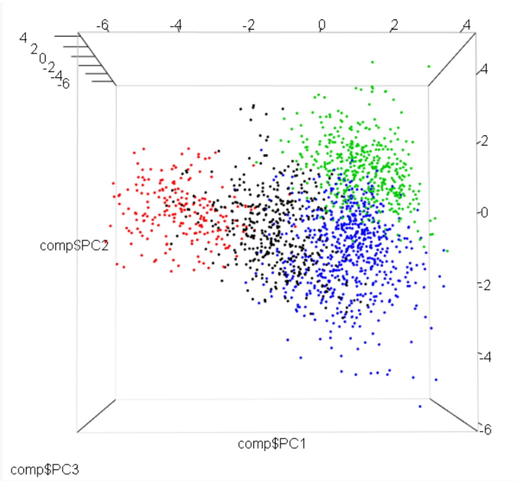

```
group_e <- cutree(hclust_e3, k = clusters)
table(group_e)

## group_e
##   1   2   3   4
## 400 210 452 658

## manhattan
hclust_m3<- hclust(dist_m, method = 'ward.D')
plot(hclust_m3, hang = -1, main = 'HAC Cluster with Manhattan Distance')
rect.hclust(hclust_m3, k = clusters)
```



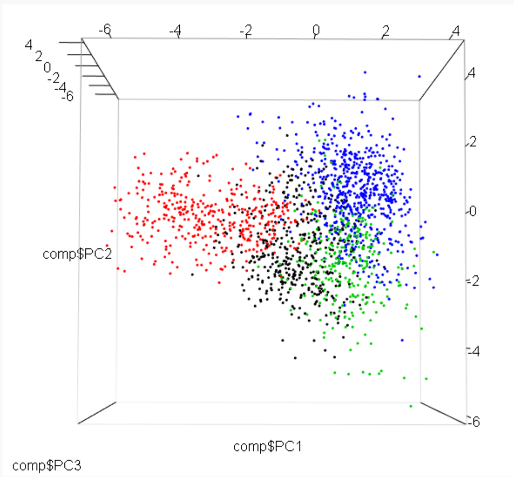

```
group_m <- cutree(hclust_m3, k = clusters)
table(group_m)

## group_m
##   1   2   3   4
## 424 376 286 634

## cosine
hclust_c3<- hclust(dist_c, method = 'ward.D')
plot(hclust_c3, hang = -1, main = 'HAC Cluster with Cosine Similarity')
rect.hclust(hclust_c3, k = clusters)
```



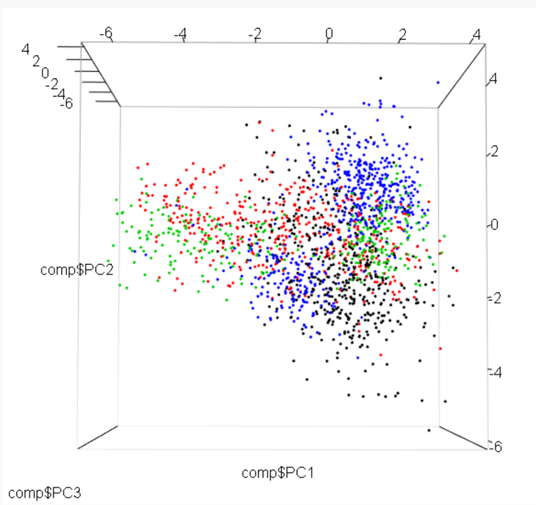

```
group_c <- cutree(hclust_c3, k = clusters)
table(group_c)

## group_c
##   1   2   3   4
## 451 412 302 555

## assess which gives most distinct clusters
plot3d(comp$PC1, comp$PC2, comp$PC3,col = group_e)
```



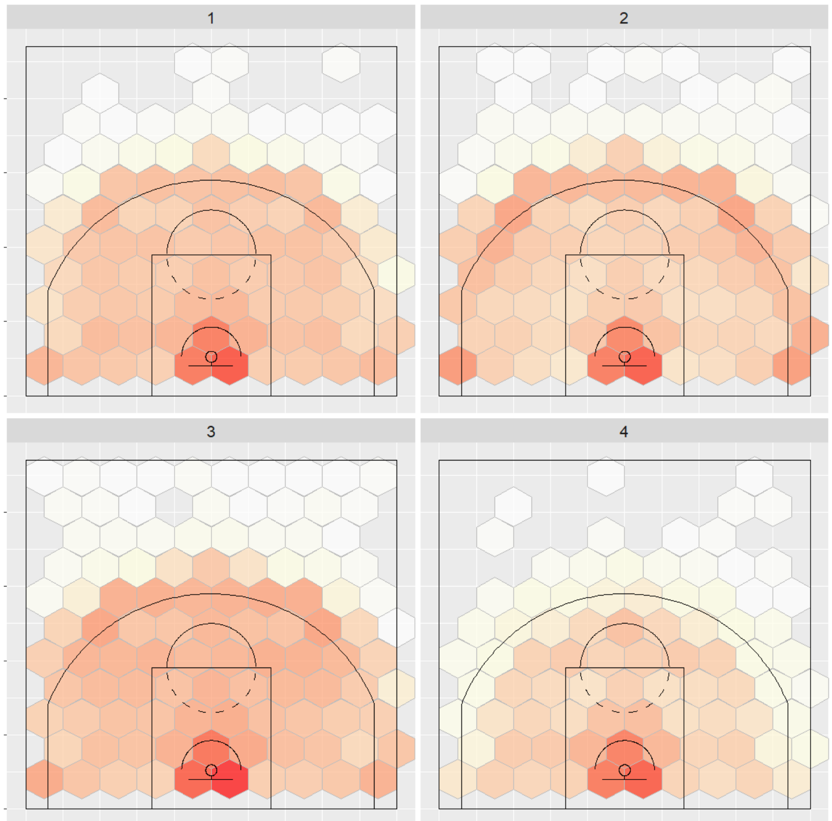

```
plot3d(comp$PC1, comp$PC2, comp$PC3,col = group_m)
```




```
plot3d(comp$PC1, comp$PC2, comp$PC3,col = group_c)
```



Results



Above is a heat map representing where players of each k-means cluster attempted their successful shots, and were most accurate. Players of cluster 4 are classic centers, staying near the paint and basket. Surprisingly, Blake Griffin is in this cluster in all but his last 2 seasons, when he began shooting more mid-range shots and entered Cluster 1. Cluster 2 are more perimeter-type players like Klay Thompson and Kyle Korver, focusing on layups and 3-point shots.

Clusters 1 and 3 are the most similar in shot selection. Both frequently take shots in every zone, but cluster 3 players, like Kyle Lowry and Kobe Bryant, are typically shorter, and take fewer mid-range shots, favoring the perimeter and extreme close range. Cluster 1 players are taller, and distribute their shots more evenly around the court, frequently taking mid-ranged jumpers. These players include LeBron James, Kevin Durant, Kawhi Leonard, and many of the classic big (but not too big) men that teams often lean on.

Validation and Win Prediction

Now that all of the clustering methods have been performed, they can be analyzed. First a dataframe is created with all of the player stats from player_shots, as well as each player's cluster.

Because there is not official metric for NBA player type, analyzing accuracy will be tricky. Here, players with less than 3 seasons of data are excluded. Then, the variance of a player's cluster over his career is analyzed. By this metric, k-means clustering creates the most consistent results, given that 64% of players have low variance in their clustering. Cosine similarity is the least useful method, as only 28% of players are clustered consistently.

```
## dataframe with all cluster types and player stats
clus_playershots <- data.frame(k$cluster,group_e, group_m, group_c, player_sh
ots) %>%
  arrange(namePlayer, yearSeason)

## assess variation among players with 3+ seasons
clus_playervariation <- clus_playershots %>%
  group_by(idPlayer, namePlayer) %>%
  filter(n()>3) %>%
  summarise(variance_k = var(k.cluster),
            variance_hac_e = var(group_e),
            variance_hac_m = var(group_m),
            variance_hac_c = var(group_c))

## `summarise()` regrouping output by 'idPlayer' (override with `.groups` arg
ument)

## assess how many players are clustered into the same cluster every time
nrow(clus_playervariation[clus_playervariation$variance_k < 0.25,])/nrow(clus
_playervariation)*100

## [1] 64.21569

nrow(clus_playervariation[clus_playervariation$variance_hac_e < 0.25,])/nrow(
clus_playervariation)*100

## [1] 60.78431

nrow(clus_playervariation[clus_playervariation$variance_hac_m < 0.25,])/nrow(
clus_playervariation)*100

## [1] 61.76471

nrow(clus_playervariation[clus_playervariation$variance_hac_c < 0.25,])/nrow(
clus_playervariation)*100

## [1] 28.43137
```

Based on this metric, it was found that the k-means clustering and HAC clustering with Euclidean and Manhattan distancing methods were both fairly consistent in their evaluation of the same players. It is intuitive that a player may not be clustered into the same type every season. For instance, a young player may come into the league and develop skills, like 3-point shooting, over time. This would effect his clustering result.

Another way to assess the usefulness of the clustering analysis, is to predict a teams season wins based on their roster, and what player types it contains.

Team data is brought in from nbastatR to find total regular season wins. The win_cut variable is created to split the team win totals into quartiles.

```
### predicting wins
seasons <- game_logs(seasons = 2011:2018, league = 'NBA',
                     result_types = 'team',
                     season_types = 'Regular Season') %>%
  filter(outcomeGame == 'W') %>%
  count(idTeam, yearSeason, name = 'Wins') %>%
  mutate_at(c(1:2), as.factor) %>%
  mutate(win_cut = quantcut(Wins, q = 3,
                            na.rm=TRUE)) %>%
  mutate(win_cut = as.factor(win_cut))
```

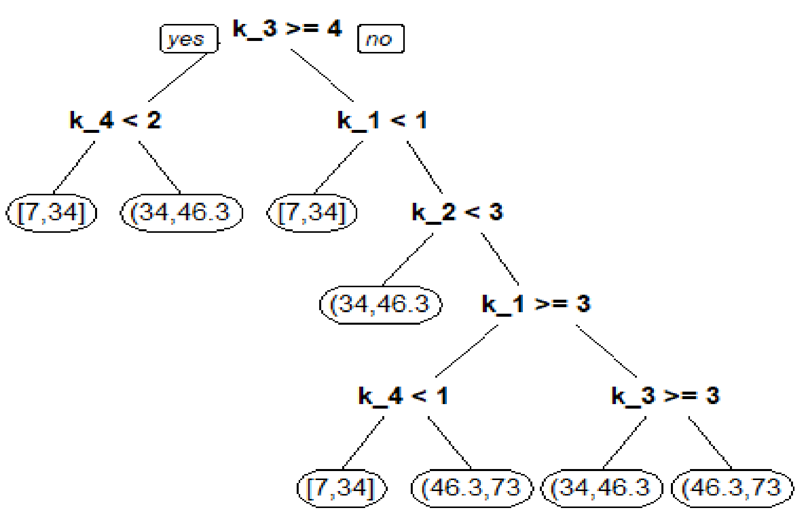
Then, players on each team are grouped from the earlier shots dataset, and clusters are brought in with left_join(). In the end, a new dataset with a team name, year, total wins, and number of each player type is created. This is then split into a training, and sample dataset.

```
rosters <- shots %>%
  distinct(idPlayer, idTeam, yearSeason,.keep_all = TRUE) %>%
  select(idPlayer, namePlayer, yearSeason, nameTeam,
         idTeam)

rosters_clus <- left_join(clus_playershots, rosters) %>%
  select(c('k.cluster','group_e', 'namePlayer', 'idPlayer',
           'yearSeason','nameTeam', 'idTeam')) %>%
  mutate_at(c('k.cluster', 'group_e'), as.factor)

## Joining, by = c("namePlayer", "idPlayer", "yearSeason")

win_pred_k <- rosters_clus %>%
  count(k.cluster, nameTeam, idTeam, yearSeason) %>%
  pivot_wider(names_from = k.cluster,
              names_prefix = 'k_',
              values_from = n) %>%
  mutate_if(is.numeric, funs(replace_na(., 0)))

win_pred_hac <- rosters_clus %>%
  count(group_e, nameTeam, idTeam, yearSeason) %>%
  pivot_wider(names_from = group_e,
              names_prefix = 'hac_',
              values_from = n) %>%
```

```
  mutate_if(is.numeric, funs(replace_na(., 0)))

win_pred <- left_join(win_pred_k, win_pred_hac)

## Joining, by = c("nameTeam", "idTeam", "yearSeason")

win_pred <- left_join(win_pred, seasons) %>%
  select(c(nameTeam, idTeam, yearSeason, Wins,
           win_cut, everything()))

## Joining, by = c("idTeam", "yearSeason")

set.seed(199)
sample_ind <- sample(nrow(win_pred),nrow(win_pred)*0.75)
train <- win_pred[sample_ind,]
test <- win_pred[-sample_ind,]
```

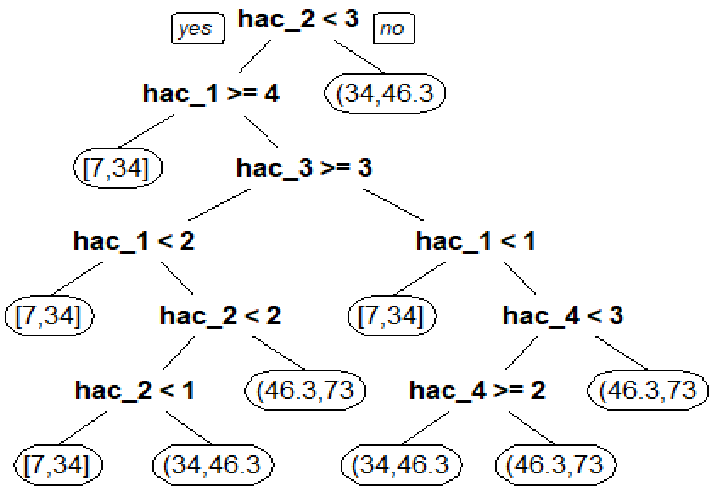
Next, decision tree models are created for both the k-means and HAC cluster data as predictors for team wins.

```
model_k <- rpart(win_cut ~ k_1 + k_2 + k_3 + k_4,
               data = train, method = 'class')
model_hac <- rpart(win_cut ~ hac_1 + hac_2 + hac_3 + hac_4,
               data = train, method = 'class')

## plot
prp(model_k)
```




```
## test accuracy
test$pred_k <- predict(model_k, test, type = 'class')
k_acc <- mean(test$pred_k == test$win_cut)
k_acc

## [1] 0.55
```

```r
prp(model_hac)
```




```r
test$pred_hac <- predict(model_hac, test, type = 'class')
hac_acc <- mean(test$pred_hac == test$win_cut)
hac_acc

## [1] 0.4166667
```

This form of modeling did not lead to incredibly accurate season wins predictions. The k-means clusters were found to put a team in the correct third of teams 51% of the time, while the HAC clustering gave the correct result only 41% of the time. Decision tree pruning was performed, but left out of this report as it only created minor improvements in the accuracy.

While the clustering analysis could be useful for roster decisions and player comparisons, another method must be found to use it as a guideline for team success. Similarly, linear regression modeling was found to be very inaccurate.

Conclusion

As previously said, the goal of this analysis is to help NBA teams improve/predict performance. Predicting whether a shot is successful or not, predicting a win vs. a loss, and categorizing players were the main objectives in this report. Although many different factors contribute to determining the outcomes to these objectives, the dataset utilized in this report relied on situational variables (dribbles, shot location, and distance from hoop) and scenario variables (opponent, venue location). Having access to additional information such as a team's record, travel time for the away team, etc., more accurate predictions may have been possible.

As for the findings in this report, it was clear that the SVM analysis was the most effective in predicting whether a shot will be made or not with a prediction accuracy of 79.3%. This was substantially higher than any of the other methods. When predicting a win vs. a loss, Decision Trees were the most effective with an accuracy of 64.54% although SVM was just slightly lower with a prediction accuracy of 62.9%.

With the clustering analysis, a team General Managers could evaluate the player types on their own rosters and others, check for mismatches, and evaluate holes in the players they have available. Rather than simply looking at the position on a player's roster position, the clustering accounts for actual player performance and decision-making. This could save scouts from watching hours of game tape to decide on a player's role within their team.

Association rule mining, also sometimes called frequent pattern analysis, are if-then statements that help to show the probability of relationship between data items within data sets. It has a number of applications and real-world examples, however, in this instance, it was used to aid the NBA in improving their player/team performance. Upon completion of the Apriori association rule mining analysis, tens of thousands of rules were created. While there is no list rule that could correctly determine whether a shot would be made or not due to environmental and player factors, data scientists were able to come up with some of the top factors that influence games. When determining whether or not a game would be won, the major factors were ***home team, location,*** and ***teamID***. When determining whether or not a shot would be missed, the major factors were ***seconds remaining, period of the game, the type of shot, time left on the shot clock, shot distance,*** and ***defender distance.*** Lastly, when determining whether or not a shot would be made, the major contributing factors were ***type of shot, player, game clock,*** and ***defender distance***. These are some of the attributes that the team recommends the NBA consider when aiming to improve league performance.