Fantasy Football Data Exploration

Connor Gendron, Courtney Zimmer, Rohi Mareddy, Tyler McAfee, David Primrose

12/9/2019

Introduction

If, as Mathematician Clive Humby said in 2006 that "Data is the new oil" then it should surprise no one that "drilling" is occurring everywhere. Certainly no more so than in the world of sports. Whether it's "Moneyball", baseball (Red Sox), or football there is power in numbers. Some, such as Kaggle, are even willing to screen scrape statistics in order to make them commonly available to others. Such is the case with Kaggle's NFL Statistics which were literally screen scraped from www.nfl.com for the common good. If you also include Fantasy Sports then you enlarge that power even more. Let's think of Fantasy football. Fantasy football is a game in which the participants serve as general managers of virtual professional gridiron football teams. Each participant chooses team rosters by participating in a draft in which all players of a real football league are available. Being able to wisely select your team roster can make-or-break your entire season and results. For our project, our team will use the Kaggle NFL Statistics basic statistics, career statistics, and game logs for all NFL player past and present to predict their Fantasy score. The Fantasy score metric is a weighted average calculation developed by Fan Duel (www.fanduel.com). Our prediction of score will be achieved and validated by performing exploratory data analysis on the Kaggle data sets and testing against regression, k-means and HAC clustering, Decision Tree, Association Rule Mining, Naïve Bayes, and SVM algorithms.

The Kaggle NFL Statistics data sets includes 19 individual statistic csv files. We have chose to use 11 by removing fumble files and kicking files which do not impact Fantasy scores. We have then cleansed and combined these files to focus on Quarterback, Running Backs, Tight Ends, and Wide Recievers. Our analysis follows.

Packages

```
library(gtools)
library(dplyr)
library(sqldf)
##install.packages("tm")
library(tm)
##install.packages("stringr")
library(stringr)
##install.packages("quanteda")
library(quanteda)
##install.packages("SnowballC")
library(SnowballC)
##install.packages("wordcloud")
library(wordcloud)
##install.packages("proxy")
library(proxy)
##install.packages("cluster")
library(cluster)
##install.packages("stringi")
library(stringi)
##install.packages("Matrix")
library(Matrix)
library(plyr) ## for adply
library(ggplot2)
##install.packages("factoextra")
library(factoextra) # for fviz
##install.packages("mclust")
library(mclust) # for Mclust EM clustering
```

```
##install.packages("slam")
library(slam)
##install.packages("factoextra")
library(class)
library(e1071)
library(tidytext) # convert DTM to DF
##install.packages("mclust")
#install.packages("dataPreparation")
library(dataPreparation)
#install.packages("fpc")
library(fpc)
library(ggfortify)
library(arules)
library(rpart)
library(rattle)
library(caret)
```

Loading in the Data

```
setwd("/Users/ihrtrobot/Desktop/IST 707/Project/Data")

GameLogsQB<- "Game_Logs_Quarterback.csv"
GameLogsRB<- "Game_Logs_Runningback.csv"
GameLogsWR.TE<- "Game_Logs_Wide_Receiver_and_Tight_End.csv"
GameLogsK<- "Game_Logs_Kickers.csv"
CareerStatsPass<- "Career_Stats_Passing.csv"
CareerStatsRec<- "Career_Stats_Receiving.csv"
CareerStatsRush<- "Career_Stats_Rushing.csv"
BasicStats<- "Basic_Stats.csv"
CareerStatsFumb<- "Career_Stats_Fumbles.csv"
GL.QB<- read.csv(GameLogsQB)
GL.K<- read.csv(GameLogsK)
GL.RB<- read.csv(GameLogsRB)
GL.WR.TE<- read.csv(GameLogsWR.TE)
CS.Pass<- read.csv(CareerStatsPass)
CS.Rec<- read.csv(CareerStatsRec)
CS.Rush<- read.csv(CareerStatsRush)
CS.Fumb<- read.csv(CareerStatsFumb)
Basic<- read.csv(BasicStats)
#head(GL.QB)
#head(GL.RB)
#head(GL.WR.TE)
#head(CS.Pass)
#head(CS.Rec)
#head(CS.Rush)
#head(Basic)
```

Data Cleaning

Basic Data Cleaning

```
#Removing unecessary columns
N.basic<- Basic[,c(-1,-2,-6,-9,-10,-11,-12)]
#Calculating the average height
```

```
avg.height<- round(mean(N.basic$Height..inches., na.rm = TRUE))
#Replacing NA's in the height column with the calculate average height
N.basic$Height..inches.<- na.replace(N.basic$Height..inches., avg.height)
#Calculating the average weight
avg.weight<- round(mean(N.basic$Weight..lbs., na.rm = TRUE))
#Replacing NA's in the weight column with the calculated average weight
N.basic$Weight..lbs.<- na.replace(N.basic$Weight..lbs., avg.weight)
#head(N.basic)
```

Career Stats Passing Data Cleaning

```
#Removing any rows of data in which Games.Played column is equal to zero
CS.Pass<- CS.Pass[which(CS.Pass$Games.Played > 0),]
#Converting the following columns from Factors to Numeric and replacing NA's with zero's
CS.Pass$Passes.Attempted<-  na.replace((as.numeric
                    (as.character(CS.Pass$Passes.Attempted))),0)
CS.Pass$Passes.Completed<- na.replace((as.numeric
                    (as.character(CS.Pass$Passes.Completed))),0)
CS.Pass$Completion.Percentage<-  na.replace((as.numeric
                    (as.character(CS.Pass$Completion.Percentage))),0)
CS.Pass$Passing.Yards<-  na.replace(as.numeric(
                     (as.character(CS.Pass$Passing.Yards))),0)
CS.Pass$Passing.Yards.Per.Attempt<-  na.replace((as.numeric(
                      as.character(CS.Pass$Passing.Yards.Per.Attempt))),0)
CS.Pass$Passing.Yards.Per.Game<-  na.replace((as.numeric(
                       as.character(CS.Pass$Passing.Yards.Per.Game))),0)
CS.Pass$TD.Passes<-  na.replace((as.numeric(
                       as.character(CS.Pass$TD.Passes))),0)
CS.Pass$Percentage.of.TDs.per.Attempts<-  na.replace((as.numeric(
                      as.character(CS.Pass$Percentage.of.TDs.per.Attempts))),0)
CS.Pass$Ints<-  na.replace((as.numeric(as.character(CS.Pass$Ints))),0)
CS.Pass$Int.Rate<-  na.replace((as.numeric(as.character(CS.Pass$Int.Rate))),0)
CS.Pass$Longest.Pass<- na.replace((as.numeric(as.character(CS.Pass$Longest.Pass))),0)
CS.Pass$Passes.Longer.than.20.Yards<- na.replace((as.numeric(
                              as.character(CS.Pass$Passes.Longer.than.20.Yards))),0)
CS.Pass$Passes.Longer.than.40.Yards<- na.replace((as.numeric(
                                as.character(CS.Pass$Passes.Longer.than.40.Yards))),0)
CS.Pass$Sacks<-  na.replace((as.numeric(as.character(CS.Pass$Sacks))),0)
CS.Pass$Sacked.Yards.Lost<-  na.replace((as.numeric(
                      as.character(CS.Pass$Sacked.Yards.Lost))),0)
CS.Pass<- CS.Pass[,c(-2,-3,-5)]
#str(CS.Pass)
#head(CS.Pass)
```

Career Stats Recieving Data Cleaning

```
CS.Rec<- CS.Rec[which(CS.Rec$Games.Played > 0),]
CS.Rec$Longest.Reception<-  na.replace((as.numeric(
                                    as.character(CS.Rec$Longest.Reception))),0)
CS.Rec$Receptions<- na.replace((as.numeric(as.character(CS.Rec$Receptions))),0)
CS.Rec$Receiving.Yards<-  na.replace((as.numeric(
                                    as.character(CS.Rec$Receiving.Yards))),0)
CS.Rec$Yards.Per.Reception<-  na.replace(as.numeric((
                                    as.character(CS.Rec$Yards.Per.Reception))),0)
CS.Rec$Receiving.TDs<-  na.replace((as.numeric(as.character(CS.Rec$Receiving.TDs))),0)
CS.Rec$Receptions.Longer.than.20.Yards<-  na.replace((as.numeric(
```

```
                                as.character(CS.Rec$Receptions.Longer.than.20.Yards))),0)
CS.Rec$Receptions.Longer.than.40.Yards<-  na.replace((as.numeric(
                                as.character(CS.Rec$Receptions.Longer.than.40.Yards))),0)
CS.Rec$First.Down.Receptions<-  na.replace((as.numeric(
                                as.character(CS.Rec$First.Down.Receptions))),0)
CS.Rec$Fumbles<- na.replace((as.numeric(as.character(CS.Rec$Fumbles))),0)
CS.Rec<- CS.Rec[,c(-2,-3,-5)]
#str(CS.Rec)
#head(CS.Rec)
```

Career Stats Rushing Data Cleaning

```
CS.Rush<-  CS.Rush[which(CS.Rush$Games.Played > 0),]
CS.Rush$Fumbles<- na.replace((as.numeric(as.character(CS.Rush$Fumbles))),0)
CS.Rush$Rushing.Attempts<-  na.replace((as.numeric(
                                  as.character(CS.Rush$Rushing.Attempts))),0)
CS.Rush$Rushing.Yards<- na.replace((as.numeric(as.character(CS.Rush$Rushing.Yards))),0)
CS.Rush$Yards.Per.Carry<-  na.replace((as.numeric(
                                  as.character(CS.Rush$Yards.Per.Carry))),0)
CS.Rush$Rushing.Yards.Per.Game<-  na.replace(as.numeric((
                                  as.character(CS.Rush$Rushing.Yards.Per.Game))),0)
CS.Rush$Rushing.TDs<-  na.replace((as.numeric(as.character(CS.Rush$Rushing.TDs))),0)
CS.Rush$Longest.Rushing.Run<-  na.replace((as.numeric(
                                    as.character(CS.Rush$Longest.Rushing.Run))),0)
CS.Rush$Rushing.First.Downs<-  na.replace((as.numeric(
                                    as.character(CS.Rush$Rushing.First.Downs))),0)
CS.Rush$Percentage.of.Rushing.First.Downs<-  na.replace((as.numeric(
                            as.character(CS.Rush$Percentage.of.Rushing.First.Downs))),0)
CS.Rush$Rushing.More.Than.20.Yards<-  na.replace((as.numeric(
                                    as.character(CS.Rush$Rushing.More.Than.20.Yards))),0)
CS.Rush$Rushing.More.Than.40.Yards<-  na.replace((as.numeric(
                                    as.character(CS.Rush$Rushing.More.Than.40.Yards))),0)
CS.Rush$Fumbles<- na.replace((as.numeric(as.character(CS.Rush$Fumbles))),0)
CS.Rush<- CS.Rush[,c(-2,-3,-5)]
#str(CS.Rush)
#head(CS.Rush)
```

Game Log Wide Reciever and Tight End Data Cleaning

```
GL.WR.TE<-  GL.WR.TE[which(GL.WR.TE$Games.Played > 0),]
GL.WR.TE$Games.Started<-  na.replace((as.numeric(
                        as.character(GL.WR.TE$Games.Started))),0)
GL.WR.TE$Receptions<- na.replace((as.numeric(as.character(GL.WR.TE$Receptions))),0)
GL.WR.TE$Receiving.Yards<-  na.replace((as.numeric(
                        as.character(GL.WR.TE$Receiving.Yards))),0)
GL.WR.TE$Yards.Per.Reception<-  na.replace(as.numeric((
                        as.character(GL.WR.TE$Yards.Per.Reception))),0)
GL.WR.TE$Longest.Reception<-  na.replace((as.numeric(
                        as.character(GL.WR.TE$Longest.Reception))),0)
GL.WR.TE$Receiving.TDs<-  na.replace((as.numeric(
                      as.character(GL.WR.TE$Receiving.TDs))),0)
GL.WR.TE$Rushing.Attempts<-  na.replace((as.numeric(
                      as.character(GL.WR.TE$Rushing.Attempts))),0)
GL.WR.TE$Rushing.Yards<-  na.replace((as.numeric(
                      as.character(GL.WR.TE$Rushing.Yards))),0)
GL.WR.TE$Yards.Per.Carry<-  na.replace((as.numeric(
```

```
                      as.character(GL.WR.TE$Yards.Per.Carry))),0)
GL.WR.TE$Longest.Rushing.Run<-  na.replace((as.numeric(
                      as.character(GL.WR.TE$Longest.Rushing.Run))),0)
GL.WR.TE$Rushing.TDs<- na.replace((as.numeric(as.character(GL.WR.TE$Rushing.TDs))),0)
GL.WR.TE<- GL.WR.TE[,c(-2,-3,-7,-11,-24,-25)]
#head(GL.WR.TE)
```

Game Log Running Back Data Cleaning

```
GL.RB<-  GL.RB[which(GL.RB$Games.Played > 0),]
GL.RB$Games.Started<-  na.replace((as.numeric(as.character(GL.RB$Games.Started))),0)
GL.RB$Rushing.Attempts<- na.replace((as.numeric(as.character(GL.RB$Rushing.Attempts))),0)
GL.RB$Rushing.Yards<-  na.replace((as.numeric(as.character(GL.RB$Rushing.Yards))),0)
GL.RB$Yards.Per.Carry<-  na.replace(as.numeric((as.character(GL.RB$Yards.Per.Carry))),0)
GL.RB$Longest.Rushing.Run<-  na.replace((as.numeric(
                                as.character(GL.RB$Longest.Rushing.Run))),0)
GL.RB$Rushing.TDs<-  na.replace((as.numeric(as.character(GL.RB$Rushing.TDs))),0)
GL.RB$Receptions<-  na.replace((as.numeric(as.character(GL.RB$Receptions))),0)
GL.RB$Receiving.Yards<-  na.replace((as.numeric(as.character(GL.RB$Receiving.Yards))),0)
GL.RB$Yards.Per.Reception<-  na.replace((as.numeric(
                                as.character(GL.RB$Yards.Per.Reception))),0)
GL.RB$Longest.Reception<-  na.replace((as.numeric(as.character(GL.RB$Longest.Reception))),0)
GL.RB$Receiving.TDs<- na.replace((as.numeric(as.character(GL.RB$Receiving.TDs))),0)
GL.RB<- GL.RB[,c(-2,-3,-7,-11,-24,-25)]
#head(GL.RB)
```

Game Log Quarterback Data Cleaning

```
GL.QB<- GL.QB[which(GL.QB$Games.Played > 0),]
GL.QB$Games.Started<-  na.replace((as.numeric(as.character(GL.QB$Games.Started))),0)
GL.QB$Passes.Completed<- na.replace((as.numeric(as.character(GL.QB$Passes.Completed))),0)
GL.QB$Passes.Attempted<-  na.replace((as.numeric(as.character(GL.QB$Passes.Attempted))),0)
GL.QB$Completion.Percentage<-  na.replace(as.numeric((
                as.character(GL.QB$Completion.Percentage))),0)
GL.QB$Passing.Yards<-  na.replace((as.numeric(as.character(GL.QB$Passing.Yards))),0)
GL.QB$Passing.Yards.Per.Attempt<-  na.replace((as.numeric(
                as.character(GL.QB$Passing.Yards.Per.Attempt))),0)
GL.QB$TD.Passes<-  na.replace((as.numeric(as.character(GL.QB$TD.Passes))),0)
GL.QB$Ints<-  na.replace((as.numeric(as.character(GL.QB$Ints))),0)
GL.QB$Sacks<-  na.replace((as.numeric(as.character(GL.QB$Sacks))),0)
GL.QB$Sacked.Yards.Lost<-  na.replace((as.numeric(
                as.character(GL.QB$Sacked.Yards.Lost))),0)
GL.QB$Rushing.Attempts<- na.replace((as.numeric(
                as.character(GL.QB$Rushing.Attempts))),0)
GL.QB$Rushing.Yards<-na.replace((as.numeric(as.character(GL.QB$Rushing.Yards))),0)
GL.QB$Yards.Per.Carry<-na.replace((as.numeric(as.character(GL.QB$Yards.Per.Carry))),0)
GL.QB$Rushing.TDs<-na.replace((as.numeric(as.character(GL.QB$Rushing.TDs))),0)
GL.QB<- GL.QB[,c(-2,-3,-7,-11,-28,-29)]
#head(GL.QB)
```

Game Log Kicker Data Cleaning

```
GL.K$Games.Started<-na.replace((as.numeric(as.character(GL.K$Games.Started))),0)
GL.K$Kicks.Blocked<-na.replace((as.numeric(as.character(GL.K$Kicks.Blocked))),0)
GL.K$Longest.FG.Made<-na.replace((as.numeric(as.character(GL.K$Longest.FG.Made))),0)
```

```
GL.K$FGs.Attempted<-na.replace((as.numeric(as.character(GL.K$FGs.Attempted))),0)
GL.K$FGs.Made<-  na.replace((as.numeric(as.character(GL.K$FGs.Made))),0)
GL.K$FG.Percentage<- na.replace((as.numeric(as.character(GL.K$FG.Percentage))),0)
GL.K$Extra.Points.Made<-  na.replace((as.numeric(as.character(GL.K$Extra.Points.Made))),0)
GL.K$Extra.Points.Attempted<-  na.replace(as.numeric((
                  as.character(GL.K$Extra.Points.Attempted))),0)
GL.K$Percentage.of.Extra.Points.Made<-  na.replace((as.numeric(
                  as.character(GL.K$Percentage.of.Extra.Points.Made))),0)
GL.K$Extra.Points.Blocked<-  na.replace((as.numeric(
                  as.character(GL.K$Extra.Points.Blocked))),0)
GL.K$Kickoffs<-  na.replace((as.numeric(as.character(GL.K$Kickoffs))),0)
GL.K$Yards.Per.Kickoff<-  na.replace((as.numeric(as.character(GL.K$Yards.Per.Kickoff))),0)
GL.K$Touchbacks<-  na.replace((as.numeric(as.character(GL.K$Touchbacks))),0)
GL.K$Kickoffs.Returned<-  na.replace((as.numeric(as.character(GL.K$Kickoffs.Returned))),0)
GL.K$Average.Returned.Yards<- na.replace((as.numeric(
                  as.character(GL.K$Average.Returned.Yards))),0)
GL.K<- GL.K[,c(-2,-3,-7,-11)]
#head(GL.K)
```

Formulas and receiving/rushing bind

We are calculting the fantasy points for two of our game log datasets based on the fantasy point formula from Fanduel. We are focusing on the Quarterback dataset and merging our wide receiver/tight end dataset with runningbacks to make our second dataset.

```
#Calculating the fantasy points for the GL.QB data
GL.QB$Fantasy<- (GL.QB$Passing.Yards * .04) + (GL.QB$TD.Passes * 6) +
  (GL.QB$Ints * (-1)) + ((GL.QB$Rushing.Yards - GL.QB$Sacked.Yards.Lost)*.1) +
  (GL.QB$Rushing.TDs * 6)
#Calcualting the cost associated with the fantasy points
GL.QB$Cost<- 3500 + (326.086957*GL.QB$Fantasy)
#Combing the game logs for wide receiver/tight end and runningbacks
GL.RB.WR<- rbind(GL.WR.TE,GL.RB)
#Calculating the fantasy points for the GL.RB.WR data
GL.RB.WR$Fantasy<- (GL.RB.WR$Receptions * .5) + (GL.RB.WR$Receiving.Yards * .1) +
  (GL.RB.WR$Receiving.TDs * 6) + (GL.QB$Rushing.Yards * .1) +
  (GL.RB.WR$Rushing.TDs * 6)
#Calcualting the cost associated with the fantasy points
GL.RB.WR$Cost<- 3500 + (326.086957*GL.RB.WR$Fantasy)
#head(GL.QB)
#head(GL.RB.WR)
```

Further data cleaning for decision tree models. Let's reduce the number of rows by taking out preseason stats and entries for games in which players did not record any stats.

```
GL.QB.Season <- filter(GL.QB, Season == 'Regular Season')
GL.RB.Season <- filter(GL.RB, Season == 'Regular Season')
GL.RB.WR.Season <- filter(GL.RB.WR, Season == 'Regular Season')
GL.WR.TE.Season <- filter(GL.WR.TE, Season == 'Regular Season')
GL.K.Season <- filter(GL.K, Season == 'Regular Season')



GL.QB.Season <- filter(GL.QB.Season, Games.Played ==1)
GL.RB.Season <- filter(GL.RB.Season, Games.Played ==1)
GL.RB.WR.Season <- filter(GL.RB.WR.Season, Games.Played ==1)
GL.WR.TE.Season <- filter(GL.WR.TE.Season, Games.Played ==1)
GL.K.Season <- filter(GL.K.Season, Games.Played ==1)
```

```
GL.QB.Season <- filter(GL.QB.Season, Passes.Completed>0)
GL.RB.WR.Season <- filter(GL.RB.WR.Season, Rushing.Attempts>0, Receptions>0)
```

Exploratory Analysis

To get an idea of our dataset, let's look at some exploratory analysis visualizations. First we can build some dataframes for the average passing, rushing, and receiving yards summarized by decade.

```
#Creating a data frame of average passing yards by decade
gamelogsQBSeason70s <- filter(GL.QB.Season, between(GL.QB.Season$Year, 1969, 1980))
gamelogsQBSeason80s <- filter(GL.QB.Season, between(GL.QB.Season$Year, 1979, 1990))
gamelogsQBSeason90s <- filter(GL.QB.Season, between(GL.QB.Season$Year, 1989, 2000))
gamelogsQBSeason00s <- filter(GL.QB.Season, between(GL.QB.Season$Year, 1999, 2010))
gamelogsQBSeason10s <- filter(GL.QB.Season, between(GL.QB.Season$Year, 2009, 2020))

a <- mean(na.omit(gamelogsQBSeason70s$Passing.Yards))
b <- mean(na.omit(gamelogsQBSeason80s$Passing.Yards))
c <- mean(na.omit(gamelogsQBSeason90s$Passing.Yards))
d <- mean(na.omit(gamelogsQBSeason00s$Passing.Yards))
e <- mean(na.omit(gamelogsQBSeason10s$Passing.Yards))
Decade <- c("1970s", "1980s", "1990s", "2000s", "2010s")
avePassingYards <- c(a,b,c,d,e)
passByDecade <- as.data.frame(cbind(Decade, avePassingYards))
(passByDecade)
```

```
##   Decade  avePassingYards
## 1  1970s 142.769281045752
## 2  1980s  178.74204009434
## 3  1990s 181.977109395795
## 4  2000s 197.961119751166
## 5  2010s 224.845861807137
```

```
#Creating a data frame of average rushing yards by decade
gamelogsRBSeason70s <- filter(GL.RB.Season, between(GL.RB.Season$Year, 1969, 1980))
gamelogsRBSeason80s <- filter(GL.RB.Season, between(GL.RB.Season$Year, 1979, 1990))
gamelogsRBSeason90s <- filter(GL.RB.Season, between(GL.RB.Season$Year, 1989, 2000))
gamelogsRBSeason00s <- filter(GL.RB.Season, between(GL.RB.Season$Year, 1999, 2010))
gamelogsRBSeason10s <- filter(GL.RB.Season, between(GL.RB.Season$Year, 2009, 2020))

a1 <- mean(na.omit(gamelogsRBSeason70s$Rushing.Yards))
b1 <- mean(na.omit(gamelogsRBSeason80s$Rushing.Yards))
c1 <- mean(na.omit(gamelogsRBSeason90s$Rushing.Yards))
d1 <- mean(na.omit(gamelogsRBSeason00s$Rushing.Yards))
e1 <- mean(na.omit(gamelogsRBSeason10s$Rushing.Yards))
aveRushingYards <- c(a1,b1,c1,d1,e1)
rushByDecade <- as.data.frame(cbind(Decade, aveRushingYards))
(rushByDecade)
```

```
##   Decade  aveRushingYards
## 1  1970s 24.6073262581308
## 2  1980s 23.4802538787024
## 3  1990s 24.6479797979798
## 4  2000s 26.3377225246432
## 5  2010s 27.6396983521087
```

```
#Creating a data frame of average receiving yards by decade
gamelogsWRSeason70s <- filter(GL.WR.TE.Season, between(GL.WR.TE.Season$Year, 1969, 1980))
gamelogsWRSeason80s <- filter(GL.WR.TE.Season, between(GL.WR.TE.Season$Year, 1979, 1990))
```

```r
gamelogsWRSeason90s <- filter(GL.WR.TE.Season, between(GL.WR.TE.Season$Year, 1989, 2000))
gamelogsWRSeason00s <- filter(GL.WR.TE.Season, between(GL.WR.TE.Season$Year, 1999, 2010))
gamelogsWRSeason10s <- filter(GL.WR.TE.Season, between(GL.WR.TE.Season$Year, 2009, 2020))

a2 <- mean(na.omit(gamelogsWRSeason70s$Receiving.Yards))
b2 <- mean(na.omit(gamelogsWRSeason80s$Receiving.Yards))
c2 <- mean(na.omit(gamelogsWRSeason90s$Receiving.Yards))
d2 <- mean(na.omit(gamelogsWRSeason00s$Receiving.Yards))
e2 <- mean(na.omit(gamelogsWRSeason10s$Receiving.Yards))
aveReceivingYards <- c(a2,b2,c2,d2,e2)
receivingByDecade <- as.data.frame(cbind(Decade, aveReceivingYards))
(receivingByDecade)
```

```
##   Decade aveReceivingYards
## 1  1970s  21.9064155385521
## 2  1980s  23.5060744115414
## 3  1990s    24.87288450809
## 4  2000s  25.5604390910644
## 5  2010s  29.6703383528537
```

```r
#Creating a data frame of longest FG made by decade
gamelogsKSeason70s <- filter(GL.K.Season, between(GL.K.Season$Year, 1969, 1980))
gamelogsKSeason80s <- filter(GL.K.Season, between(GL.K.Season$Year, 1979, 1990))
gamelogsKSeason90s <- filter(GL.K.Season, between(GL.K.Season$Year, 1989, 2000))
gamelogsKSeason00s <- filter(GL.K.Season, between(GL.K.Season$Year, 1999, 2010))
gamelogsKSeason10s <- filter(GL.K.Season, between(GL.K.Season$Year, 2009, 2020))

a3 <- mean(na.omit(gamelogsKSeason70s$Longest.FG.Made))
b3 <- mean(na.omit(gamelogsKSeason80s$Longest.FG.Made))
c3 <- mean(na.omit(gamelogsKSeason90s$Longest.FG.Made))
d3 <- mean(na.omit(gamelogsKSeason00s$Longest.FG.Made))
e3 <- mean(na.omit(gamelogsKSeason10s$Longest.FG.Made))
aveFGMade <- c(a3,b3,c3,d3,e3)
FGMadeByDecade <- as.data.frame(cbind(Decade, aveFGMade))

a4 <- max(na.omit(gamelogsKSeason70s$Longest.FG.Made))
b4 <- max(na.omit(gamelogsKSeason80s$Longest.FG.Made))
c4 <- max(na.omit(gamelogsKSeason90s$Longest.FG.Made))
d4 <- max(na.omit(gamelogsKSeason00s$Longest.FG.Made))
e4 <- max(na.omit(gamelogsKSeason10s$Longest.FG.Made))
longestFGMade <- c(a4,b4,c4,d4,e4)
longestFGMadeByDecade <- as.data.frame(cbind(Decade, longestFGMade))

(FGMadeByDecade)
```

```
##   Decade        aveFGMade
## 1  1970s 11.7144697294976
## 2  1980s 25.2914603960396
## 3  1990s 27.9826458036984
## 4  2000s 28.5677494199536
## 5  2010s 33.0814963259853
```

```r
(longestFGMadeByDecade)
```

```
##   Decade longestFGMade
## 1  1970s            59
## 2  1980s            59
## 3  1990s            63
```

```
## 4  2000s             62
## 5  2010s             64
```

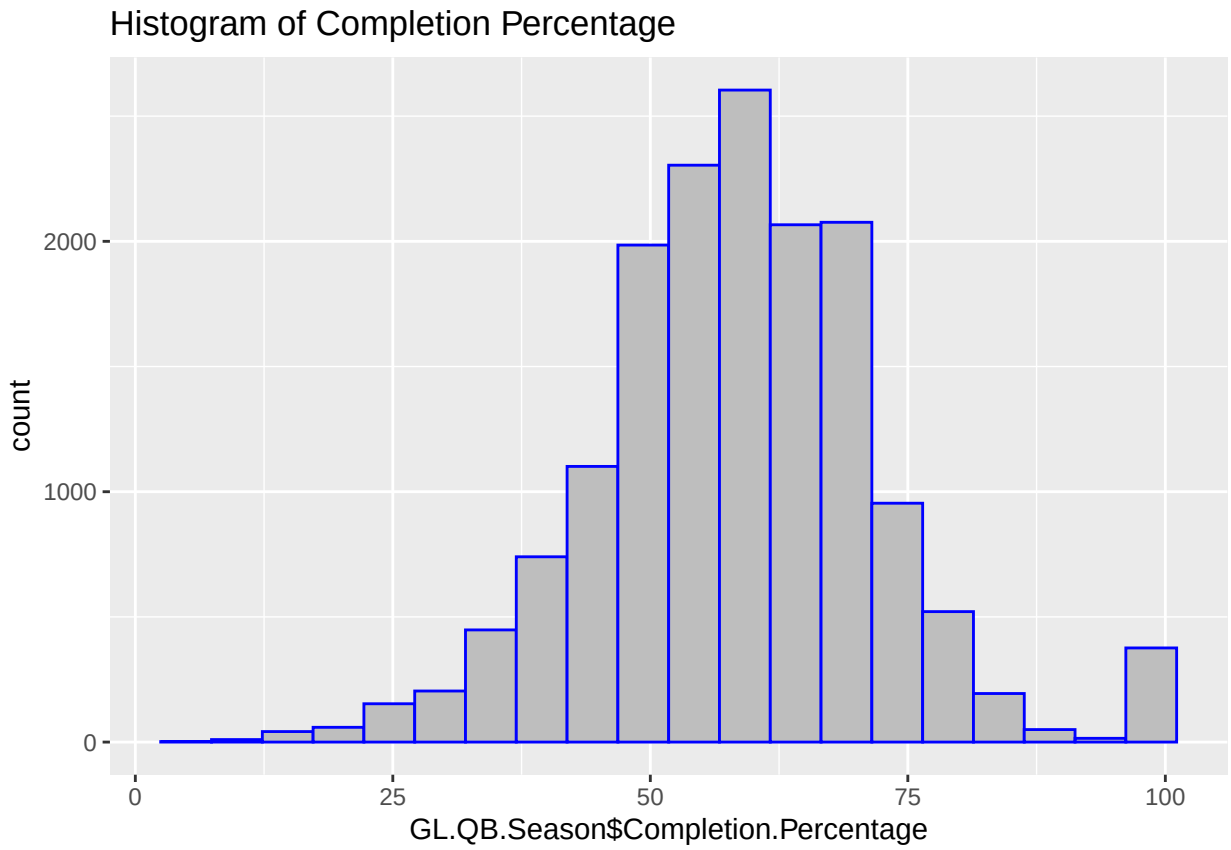
```
#Printing the longest FG made in text
cat("The longest FG made through the 2016 season is",
    max(gamelogsKSeason10s$Longest.FG.Made), "yards")
```

```
## The longest FG made through the 2016 season is 64 yards
```

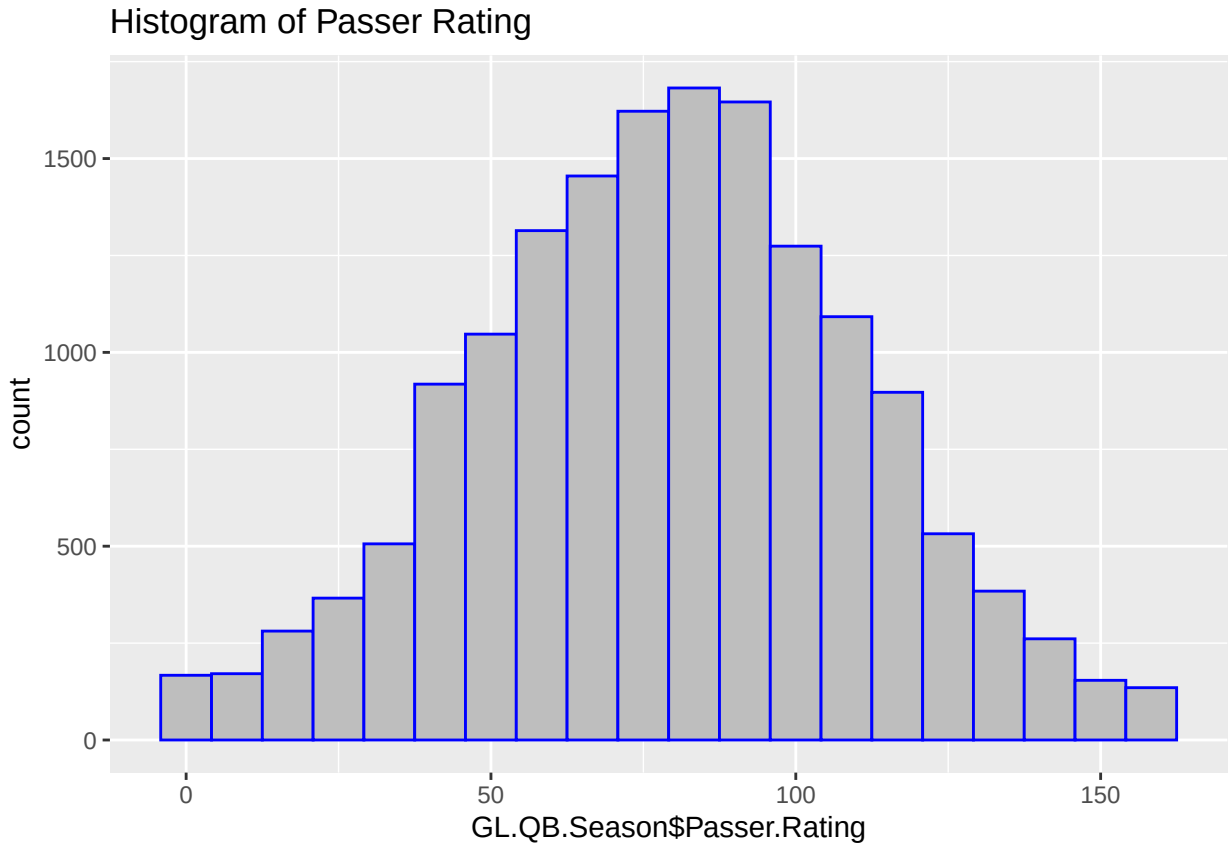
It's interesting that the longest FG made to date is 64 yards, that is really long!

Now let's look at the breakdown of these same stats by the frequency of how these positions perform on a yardage basis.

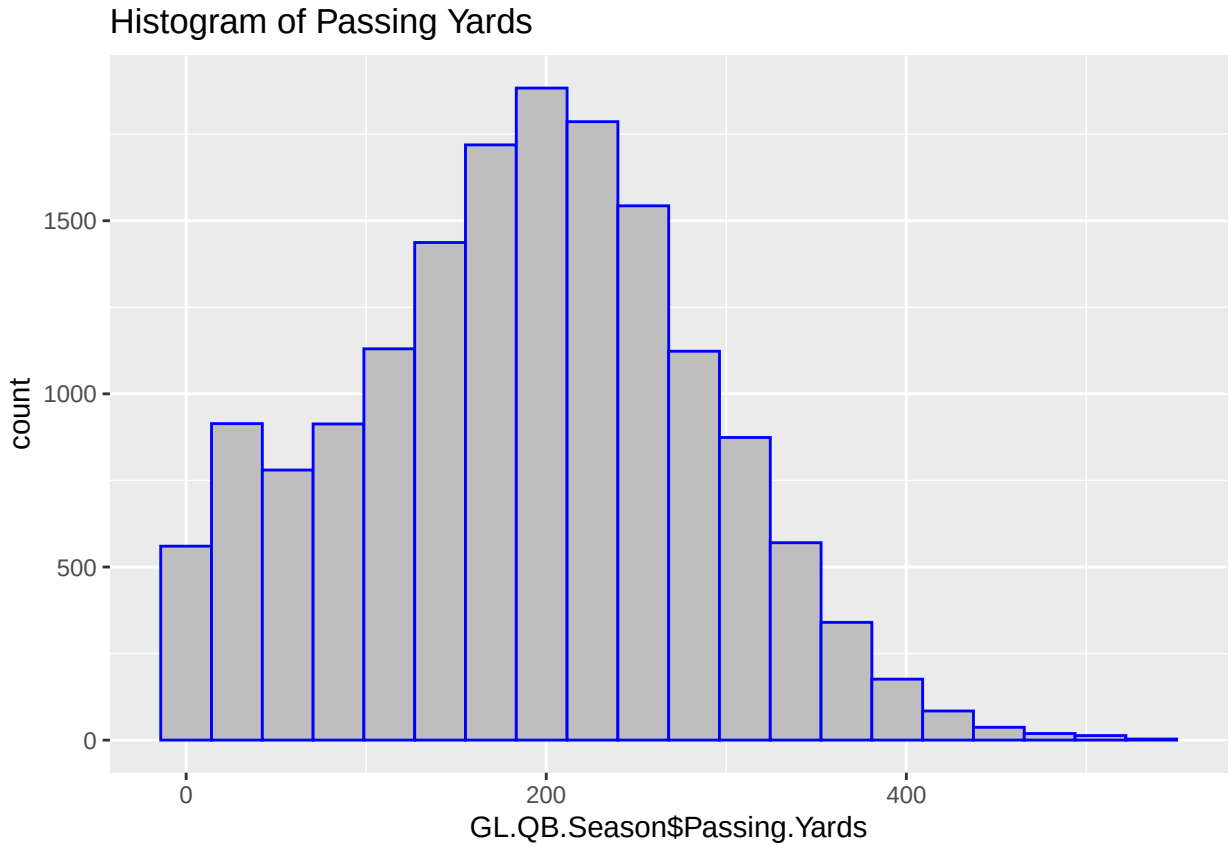
```
ggplot() +geom_histogram(data= GL.QB.Season, aes(x=GL.QB.Season$Completion.Percentage),
    colour= "blue", fill="grey", bins=20) + ggtitle("Histogram of Completion Percentage")
```



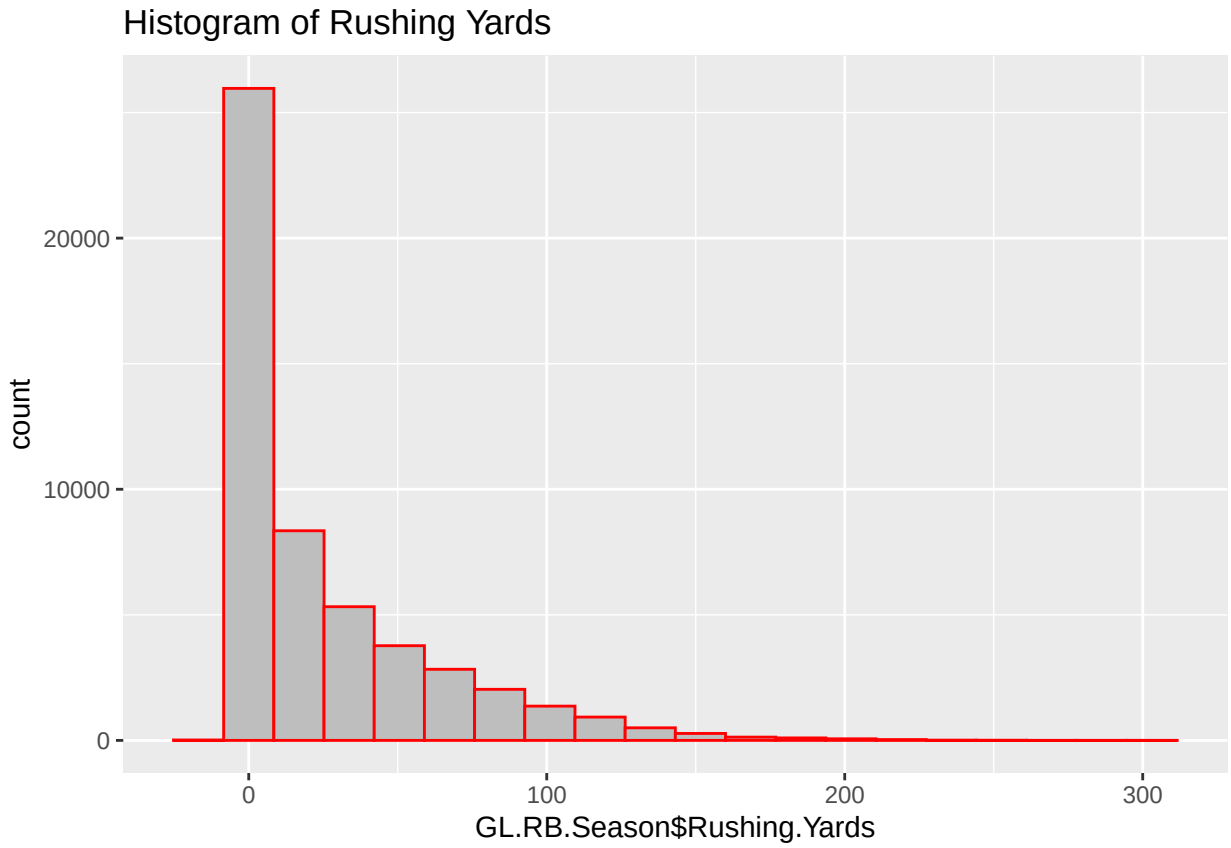

```
ggplot() +geom_histogram(data= GL.QB.Season, aes(x=GL.QB.Season$Passer.Rating),
      colour= "blue", fill="grey", bins=20) + ggtitle("Histogram of Passer Rating")
```



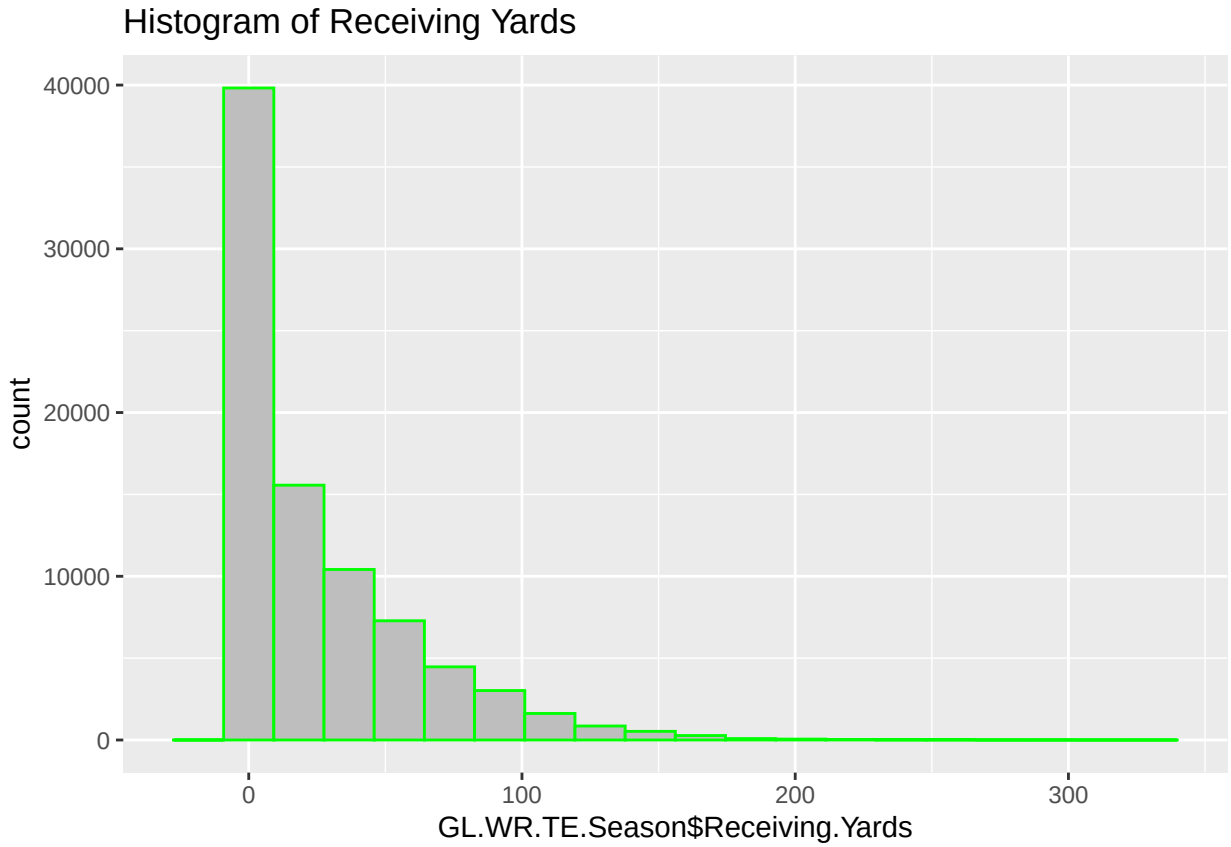

```
ggplot() +geom_histogram(data= GL.QB.Season, aes(x=GL.QB.Season$Passing.Yards),
            colour= "blue", fill="grey", bins=20) + ggtitle("Histogram of Passing Yards")
```



```
ggplot() +geom_histogram(data= GL.RB.Season, aes(x=GL.RB.Season$Rushing.Yards),
          colour= "red", fill="grey", bins=20) + ggtitle("Histogram of Rushing Yards")
```




```
ggplot() +geom_histogram(data= GL.WR.TE.Season, aes(x=GL.WR.TE.Season$Receiving.Yards),
          colour= "green", fill="grey", bins=20) + ggtitle("Histogram of Receiving Yards")
```



It's interesting that the QB stats follow a normalized distribution (more or less), but the rushing and receiving stats are both skewed to the left. This shows that a QB is expected to perform at a certain bar, since they are the "captain" of the team and involved in running the offense. The RBs and WRs split the play percentage amongst other players in the same position. This will be more evident later on when we look into fantasy point production between QBs, RBs, and WRs.

This next bit of code will just be transforming the data a little bit and looking at the overall average performance from each skill position (QB, RB, WR) by decade. The outcome below will be a plot of all the averages across each decade of data.

```
passingYardsSummary <- as.data.frame(tapply(GL.QB.Season$Passing.Yards,
                          list(GL.QB.Season$Year,GL.QB.Season$Player.Id), mean))

QB70s <- passingYardsSummary[(1:10),]
QB80s <- passingYardsSummary[(11:20),]
QB90s <- passingYardsSummary[(21:30),]
QB00s <- passingYardsSummary[(31:40),]
QB10s <- passingYardsSummary[(41:47),]

completeQB10s <- na.omit(t(QB10s))
completeQB00s <- na.omit(t(QB00s))
completeQB90s <- na.omit(t(QB90s))
completeQB80s <- na.omit(t(QB80s))
completeQB70s <- na.omit(t(QB70s))

Names <- rownames(completeQB10s)
rownames(completeQB10s) <- NULL
completeQB10s <- cbind(Names,completeQB10s)

Names <- rownames(completeQB00s)
rownames(completeQB00s) <- NULL
completeQB00s <- cbind(Names,completeQB00s)
```

```
Names <- rownames(completeQB90s)
rownames(completeQB90s) <- NULL
completeQB90s <- cbind(Names,completeQB90s)

Names <- rownames(completeQB80s)
rownames(completeQB80s) <- NULL
completeQB80s <- cbind(Names,completeQB80s)

Names <- rownames(completeQB70s)
rownames(completeQB70s) <- NULL
completeQB70s <- cbind(Names,completeQB70s)

completeQB70s <- t(completeQB70s)
colnames(completeQB70s) <- completeQB70s[1,]
completeQB70s <- completeQB70s[-1, ]

completeQB80s <- t(completeQB80s)
colnames(completeQB80s) <- completeQB80s[1,]
completeQB80s <- completeQB80s[-1, ]

completeQB90s <- t(completeQB90s)
colnames(completeQB90s) <- completeQB90s[1,]
completeQB90s <- completeQB90s[-1, ]

completeQB00s <- t(completeQB00s)
colnames(completeQB00s) <- completeQB00s[1,]
completeQB00s <- completeQB00s[-1, ]

completeQB10s <- t(completeQB10s)
colnames(completeQB10s) <- completeQB10s[1,]
completeQB10s <- completeQB10s[-1, ]

meanQB70s <- mean(as.numeric(completeQB70s))
meanQB80s <- mean(as.numeric(completeQB80s))
meanQB90s <- mean(as.numeric(completeQB90s))
meanQB00s <- mean(as.numeric(completeQB00s))
meanQB10s <- mean(as.numeric(completeQB10s))

meanQBDF <- as.data.frame(cbind(meanQB70s, meanQB80s, meanQB90s, meanQB00s, meanQB10s))

(meanQBDF)
```

```
##   meanQB70s meanQB80s meanQB90s meanQB00s meanQB10s
## 1  122.4493  186.7202  187.4243  220.8641  228.6504
```

```
rownames(meanQBDF) <- ("QBmeans")
colnames(meanQBDF) <- (c("1970s", "1980s", "1990s", "2000s", "2010s"))

meanQBTransDF <- t(meanQBDF)
#plot(meanQBTransDF, ylab = "Ave Passing Yards", xlab = "Decade")
```

```
rushingYardsSummary <- as.data.frame(tapply(GL.RB.Season$Rushing.Yards,
                        list(GL.RB.Season$Year,GL.RB.Season$Player.Id), mean))

RB70s <- rushingYardsSummary[(1:10),]
RB80s <- rushingYardsSummary[(11:20),]
RB90s <- rushingYardsSummary[(21:30),]
```

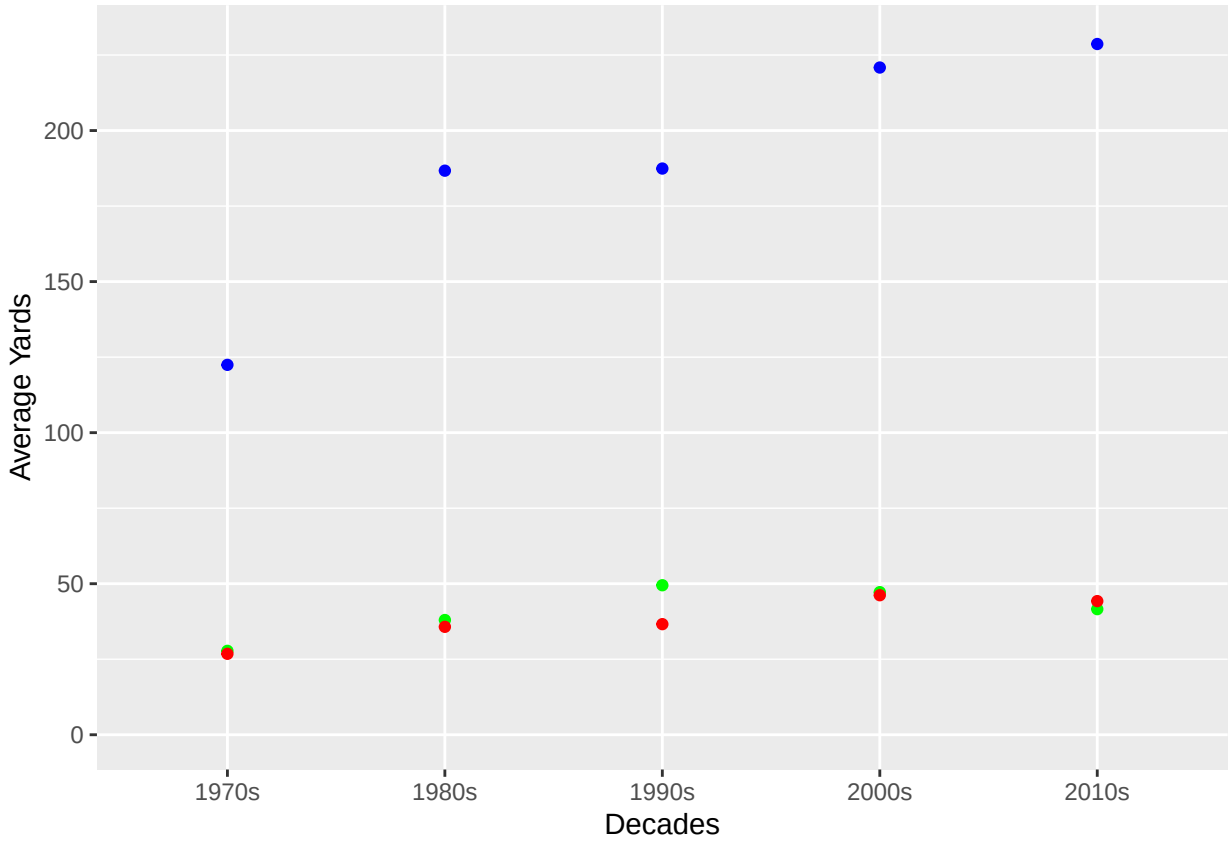
```
RB00s <- rushingYardsSummary[(31:40),]
RB10s <- rushingYardsSummary[(41:47),]

completeRB10s <- na.omit(t(RB10s))
completeRB00s <- na.omit(t(RB00s))
completeRB90s <- na.omit(t(RB90s))
completeRB80s <- na.omit(t(RB80s))
completeRB70s <- na.omit(t(RB70s))

Names <- rownames(completeRB10s)
rownames(completeRB10s) <- NULL
completeRB10s <- cbind(Names,completeRB10s)

Names <- rownames(completeRB00s)
rownames(completeRB00s) <- NULL
completeRB00s <- cbind(Names,completeRB00s)

Names <- rownames(completeRB90s)
rownames(completeRB90s) <- NULL
completeRB90s <- cbind(Names,completeRB90s)

Names <- rownames(completeRB80s)
rownames(completeRB80s) <- NULL
completeRB80s <- cbind(Names,completeRB80s)

Names <- rownames(completeRB70s)
rownames(completeRB70s) <- NULL
completeRB70s <- cbind(Names,completeRB70s)

completeRB70s <- t(completeRB70s)
colnames(completeRB70s) <- completeRB70s[1,]
completeRB70s <- completeRB70s[-1, ]

completeRB80s <- t(completeRB80s)
colnames(completeRB80s) <- completeRB80s[1,]
completeRB80s <- completeRB80s[-1, ]

completeRB90s <- t(completeRB90s)
colnames(completeRB90s) <- completeRB90s[1,]
completeRB90s <- completeRB90s[-1, ]

completeRB00s <- t(completeRB00s)
colnames(completeRB00s) <- completeRB00s[1,]
completeRB00s <- completeRB00s[-1, ]

completeRB10s <- t(completeRB10s)
colnames(completeRB10s) <- completeRB10s[1,]
completeRB10s <- completeRB10s[-1, ]

meanRB70s <- mean(as.numeric(completeRB70s))
meanRB80s <- mean(as.numeric(completeRB80s))
meanRB90s <- mean(as.numeric(completeRB90s))
meanRB00s <- mean(as.numeric(completeRB00s))
meanRB10s <- mean(as.numeric(completeRB10s))

meanRBDF <- as.data.frame(cbind(meanRB70s, meanRB80s, meanRB90s, meanRB00s, meanRB10s))
```

```
rownames(meanRBDF) <- ("RBmeans")
colnames(meanRBDF) <- (c("1970s", "1980s", "1990s", "2000s", "2010s"))

meanRBTransDF <- t(meanRBDF)
#plot(meanRBTransDF, ylab = "Ave Rushing Yards", xlab = "Decade")
```

```
receivingYardsSummary <- as.data.frame(tapply(GL.WR.TE.Season$Receiving.Yards,
                list(GL.WR.TE.Season$Year,GL.WR.TE.Season$Player.Id), mean))

WR70s <- receivingYardsSummary[(1:10),]
WR80s <- receivingYardsSummary[(11:20),]
WR90s <- receivingYardsSummary[(21:30),]
WR00s <- receivingYardsSummary[(31:40),]
WR10s <- receivingYardsSummary[(41:47),]

completeWR10s <- na.omit(t(WR10s))
completeWR00s <- na.omit(t(WR00s))
completeWR90s <- na.omit(t(WR90s))
completeWR80s <- na.omit(t(WR80s))
completeWR70s <- na.omit(t(WR70s))

Names <- rownames(completeWR10s)
rownames(completeWR10s) <- NULL
completeWR10s <- cbind(Names,completeWR10s)

Names <- rownames(completeWR00s)
rownames(completeWR00s) <- NULL
completeWR00s <- cbind(Names,completeWR00s)

Names <- rownames(completeWR90s)
rownames(completeWR90s) <- NULL
completeWR90s <- cbind(Names,completeWR90s)

Names <- rownames(completeWR80s)
rownames(completeWR80s) <- NULL
completeWR80s <- cbind(Names,completeWR80s)

Names <- rownames(completeWR70s)
rownames(completeWR70s) <- NULL
completeWR70s <- cbind(Names,completeWR70s)

completeWR70s <- t(completeWR70s)
colnames(completeWR70s) <- completeWR70s[1,]
completeWR70s <- completeWR70s[-1, ]

completeWR80s <- t(completeWR80s)
colnames(completeWR80s) <- completeWR80s[1,]
completeWR80s <- completeWR80s[-1, ]

completeWR90s <- t(completeWR90s)
colnames(completeWR90s) <- completeWR90s[1,]
completeWR90s <- completeWR90s[-1, ]

completeWR00s <- t(completeWR00s)
colnames(completeWR00s) <- completeWR00s[1,]
completeWR00s <- completeWR00s[-1, ]
```

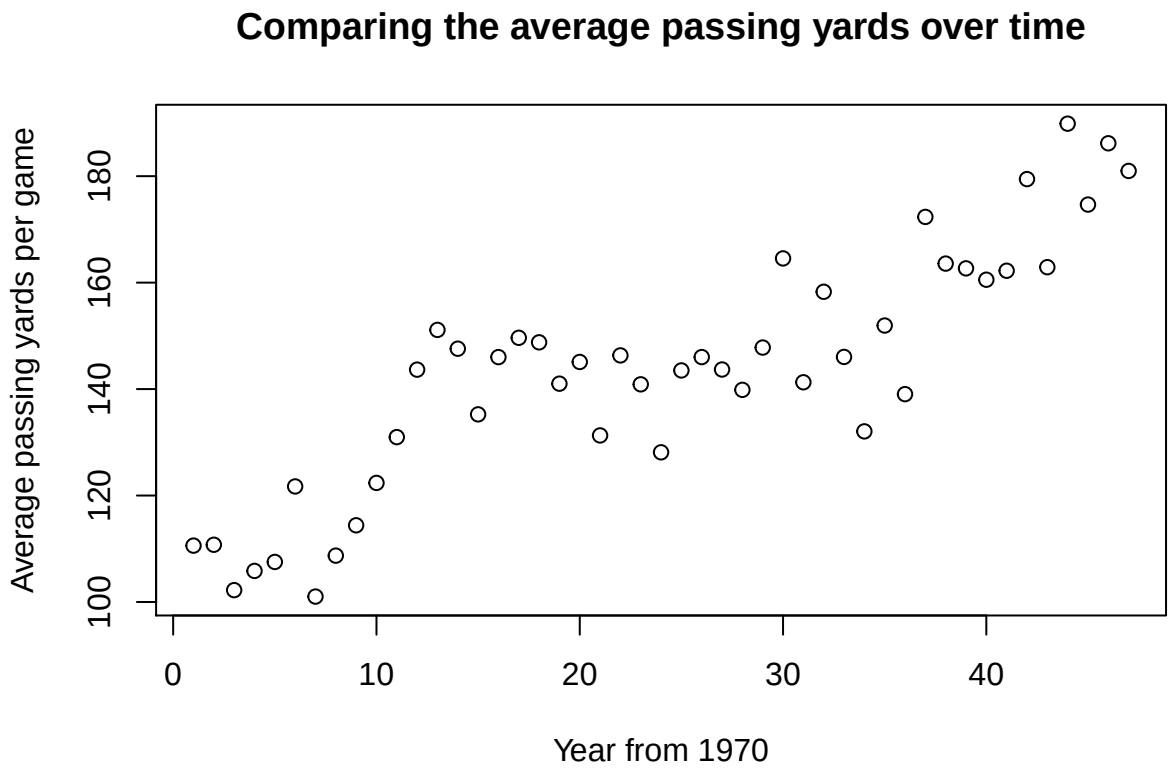
```r
completeWR10s <- t(completeWR10s)
colnames(completeWR10s) <- completeWR10s[1,]
completeWR10s <- completeWR10s[-1, ]

meanWR70s <- mean(as.numeric(completeWR70s))
meanWR80s <- mean(as.numeric(completeWR80s))
meanWR90s <- mean(as.numeric(completeWR90s))
meanWR00s <- mean(as.numeric(completeWR00s))
meanWR10s <- mean(as.numeric(completeWR10s))

meanWRDF <- as.data.frame(cbind(meanWR70s, meanWR80s, meanWR90s, meanWR00s, meanWR10s))

rownames(meanWRDF) <- ("WRmeans")
colnames(meanWRDF) <- (c("1970s", "1980s", "1990s", "2000s", "2010s"))

meanWRTransDF <- t(meanWRDF)
#plot(meanWRTransDF, ylab = "Ave Rushing Yards", xlab = "Decade")
```

```r
meanStats <- as.data.frame(cbind(meanQBTransDF,meanRBTransDF,meanWRTransDF))

decadePlot2 = ggplot() +
  geom_point(data = meanStats, aes(x = Decade, y = meanWRTransDF), color = "green") +
  geom_point(data = meanStats, aes(x = Decade, y = meanQBTransDF), color = "blue") +
  geom_point(data = meanStats, aes(x = Decade, y = meanRBTransDF), color = "red") +
  scale_y_continuous(limit = c(0, 230)) +
  xlab('Decades') +
  ylab('Average Yards')

print(decadePlot2)
```



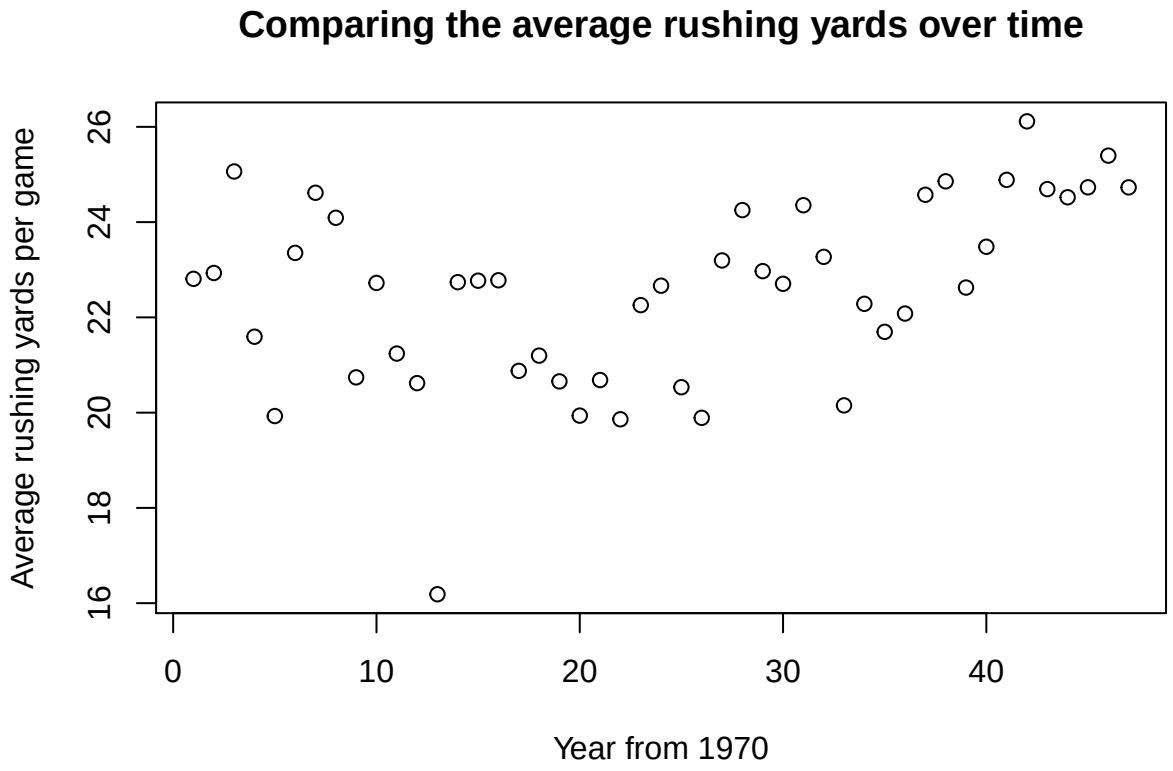
So we have an idea of what the averages are per decade for each position. But how have these transgressed over the years? A more detailed plot will be generated below to show how each position has fluctuated over the years. I will be using the yearly yard summary DFs from above to generate three plots; passing yards, rushing yards, and receiving yards, over the last 46 years or so.

```
#plot(passingYardsSummary[,1])
passingYardsDF <- as.data.frame(rowMeans(passingYardsSummary, na.rm = T))
#passingYardsDF[,1]
colnames(passingYardsDF) <- c("AverageYards")
#passingYardsDF
plot(passingYardsDF[,1], xlab = "Year from 1970", ylab = "Average passing yards per game") +
  title("Comparing the average passing yards over time")
```



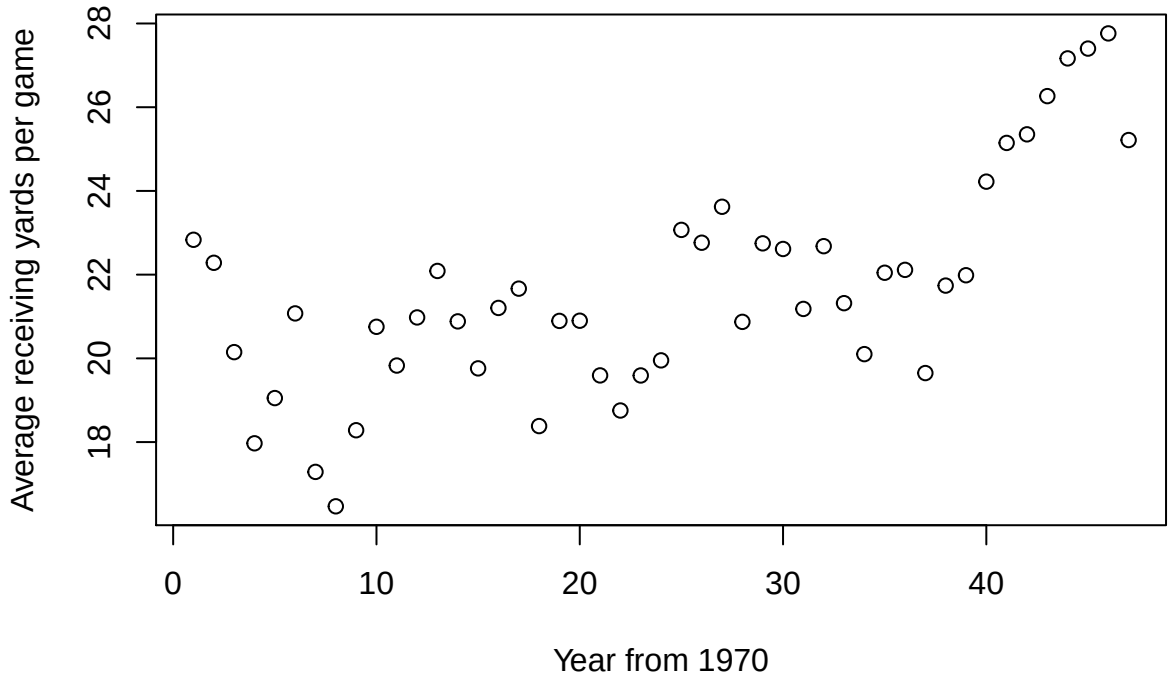

```
## integer(0)
```

```
rushingYardsDF <- as.data.frame(rowMeans(rushingYardsSummary, na.rm = T))
#rushingYardsDF[,1]
colnames(rushingYardsDF) <- c("AverageYards")
#rushingYardsDF
plot(rushingYardsDF[,1], xlab = "Year from 1970", ylab = "Average rushing yards per game") +
      title("Comparing the average rushing yards over time")
```




```
## integer(0)
```

```
receivingYardsDF <- as.data.frame(rowMeans(receivingYardsSummary, na.rm = T))
#receivingYardsDF[,1]
colnames(receivingYardsDF) <- c("AverageYards")
#receivingYardsDF
plot(receivingYardsDF[,1], xlab = "Year from 1970", ylab = "Average receiving yards per game") +
          title("Comparing the average receiving yards over time")
```



```
## integer(0)
```

A couple notes from the plots above. First, the passing plot shows a steady incline in passing yards over the 46 years. Obviously there is some variation, but the average QB performance has always trended upwards. The rushing plot is a lot more surprising. This shows that average rushing yards have more or less stayed the same over the last 5 decades. However, it is important to note that there has been a slight increase in the past couple decades. And finally, the receiving plot is the most intersting (in my opinion), because it shows that average receiving yards was about the same from 1970 until 2008 or so. Then from 2008 until 2016 there was a dramtic increase in average receiving yards. This may be that the skillset of WRs is improving as a collective whole or that some WRs are being targeted more and those that are not targeted as much sometimes have no stats to include in our data set. Let's see if anything in our future analysis might explain this story a little more.

Modeling

Regression

This combination of variables produced the best R-squared and p-vaules that I could find.

```
fan.reg<-lm(GL.QB$Fantasy~ GL.QB$Home.or.Away + GL.QB$Opponent + GL.QB$Outcome
      + GL.QB$Passes.Completed + GL.QB$Passes.Attempted +
      GL.QB$Completion.Percentage + GL.QB$Passing.Yards + GL.QB$TD.Passes
      + GL.QB$Ints + GL.QB$Sacked.Yards.Lost + GL.QB$Rushing.Attempts
      + GL.QB$Rushing.Yards + GL.QB$Rushing.TDs, data=GL.QB)
```

Clustering

Clustering analysis of Career Stats data for Passing , Receving and Rushing Yards using k Means and HAC Algorithms. We will build clusters for each Passing , Receving and Rushing and we will build seperate clusters where we aggregate the yards by Year and view how the clustering is distributed.

Career Stats Passing

```
# Compute bins
# bins <- build_bins(CS.Pass, cols = "Year", n_bins = 8, type = "equal_freq")

#CS.Pass1 <- fastDiscretization(dataSet = CS.Pass, bins = list
                  #(Year = c(1940, 1950, 1960, 1970, 1980,1990,2000,2010,2020)))
CS.Pass1 <- (CS.Pass[,-1])
#Building CLusters
CS.PassClust <- t(CS.Pass[,-1])
CS.PassClust <- as.matrix(CS.PassClust)

#Creating a Cluster for aggregation by Year
CS.PassClust1<- (CS.Pass[,-1])
CS.PassClust1<-aggregate(.~Year, data=CS.PassClust1, FUN = sum)
CS.PassClust1<- as.matrix(CS.PassClust1)

#Create Kmeans cluster
set.seed(100)
k <- 3 # number of clusters
CSPass.kmeans <- kmeans(CS.PassClust, k)
CSPass.kmeans$centers<- round(CSPass.kmeans$centers, digits = 3) # cluster centers
(CSPass.kmeans$cluster)
```

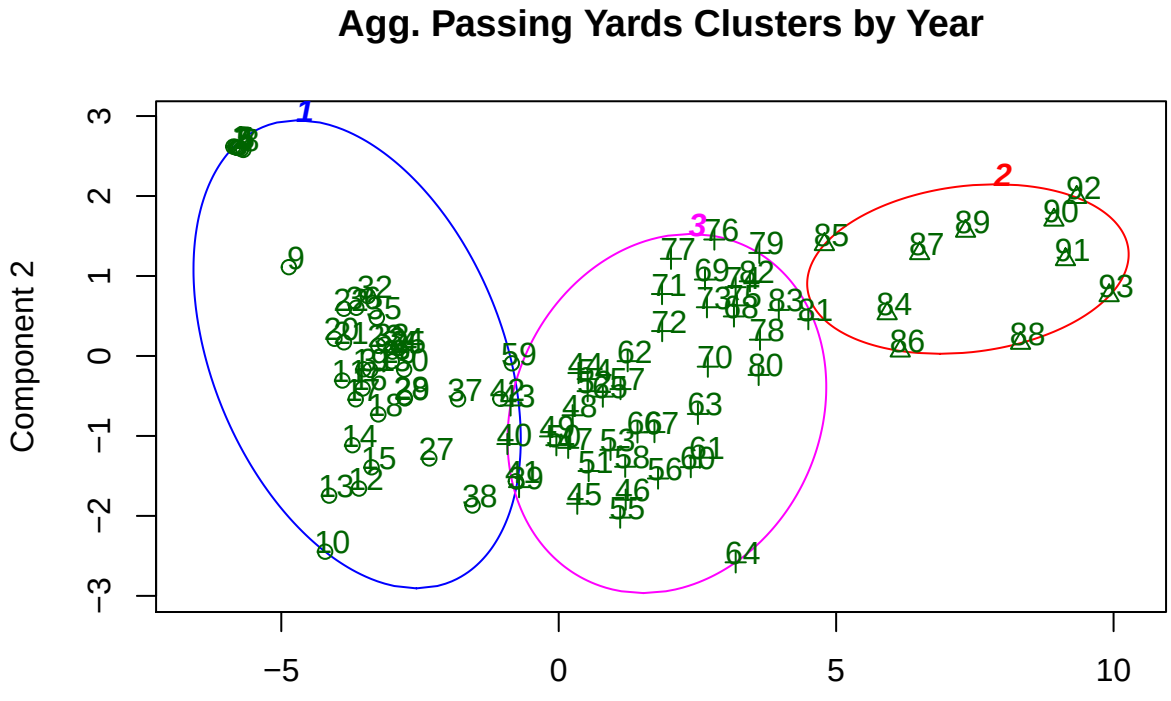
```
##                         Year                 Games.Played
##                            3                            1
##            Passes.Attempted             Passes.Completed
##                            2                            2
```

```
##           Completion.Percentage         Pass.Attempts.Per.Game
##                               1                              1
##                  Passing.Yards      Passing.Yards.Per.Attempt
##                               2                              1
##         Passing.Yards.Per.Game                    TD.Passes
##                               2                              1
## Percentage.of.TDs.per.Attempts                         Ints
##                               1                              1
##                       Int.Rate                 Longest.Pass
##                               1                              1
##    Passes.Longer.than.20.Yards    Passes.Longer.than.40.Yards
##                               1                              1
##                          Sacks             Sacked.Yards.Lost
##                               1                              2
##                  Passer.Rating
##                               1
```

```r
#Creating Kmeans for Agregated CLsuter
set.seed(100)
k <- 3 # number of clusters
CSPass.kmeans1 <- kmeans(CS.PassClust1, k)
CSPass.kmeans1$centers<- round(CSPass.kmeans1$centers, digits = 3) # cluster centers
(CSPass.kmeans1$cluster)
```

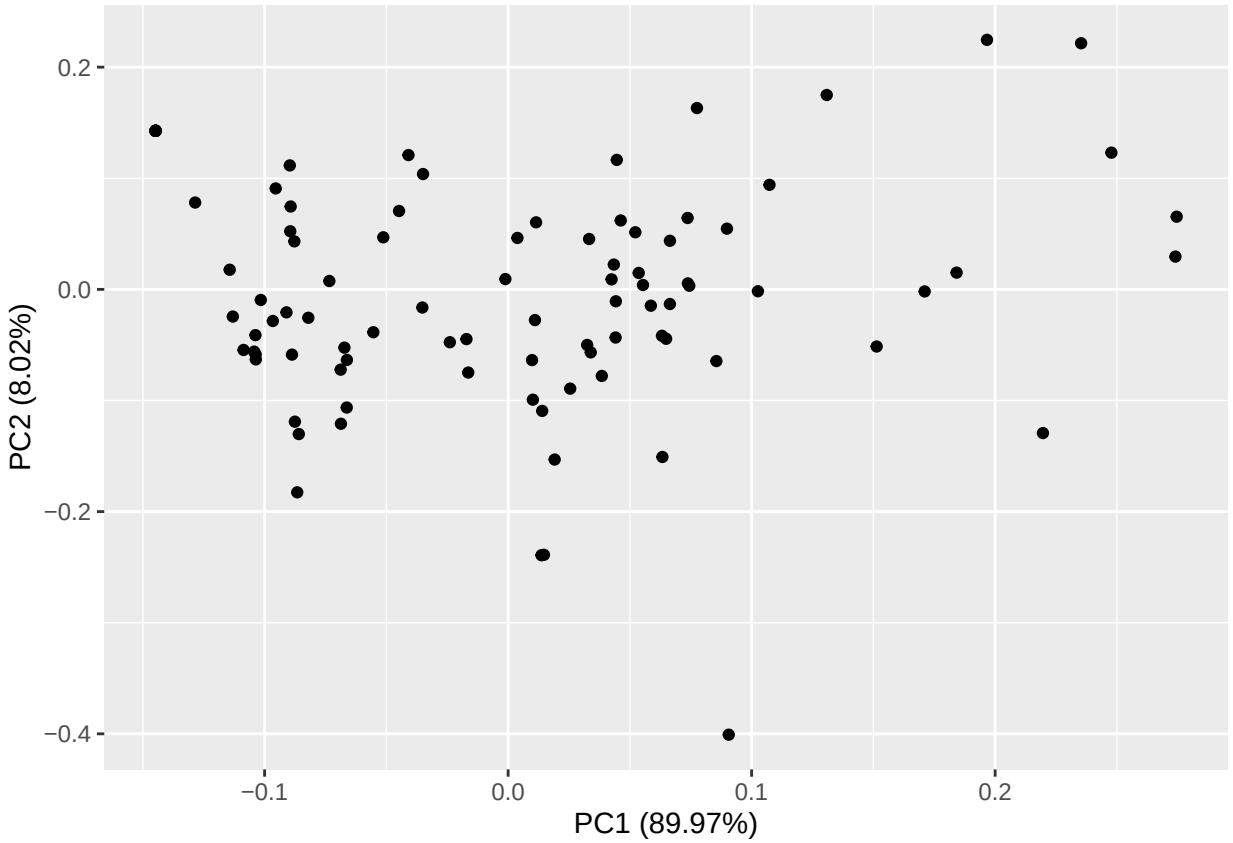
```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [36] 1 1 1 3 3 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3
## [71] 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2
```

```r
#Clusterplot for Kmeans Cluster
clusplot(CS.PassClust1,CSPass.kmeans1$cluster,color =  TRUE ,shade = FALSE,
         labels = 2,lines = 0 , main = 'Agg. Passing Yards Clusters by Year')
```

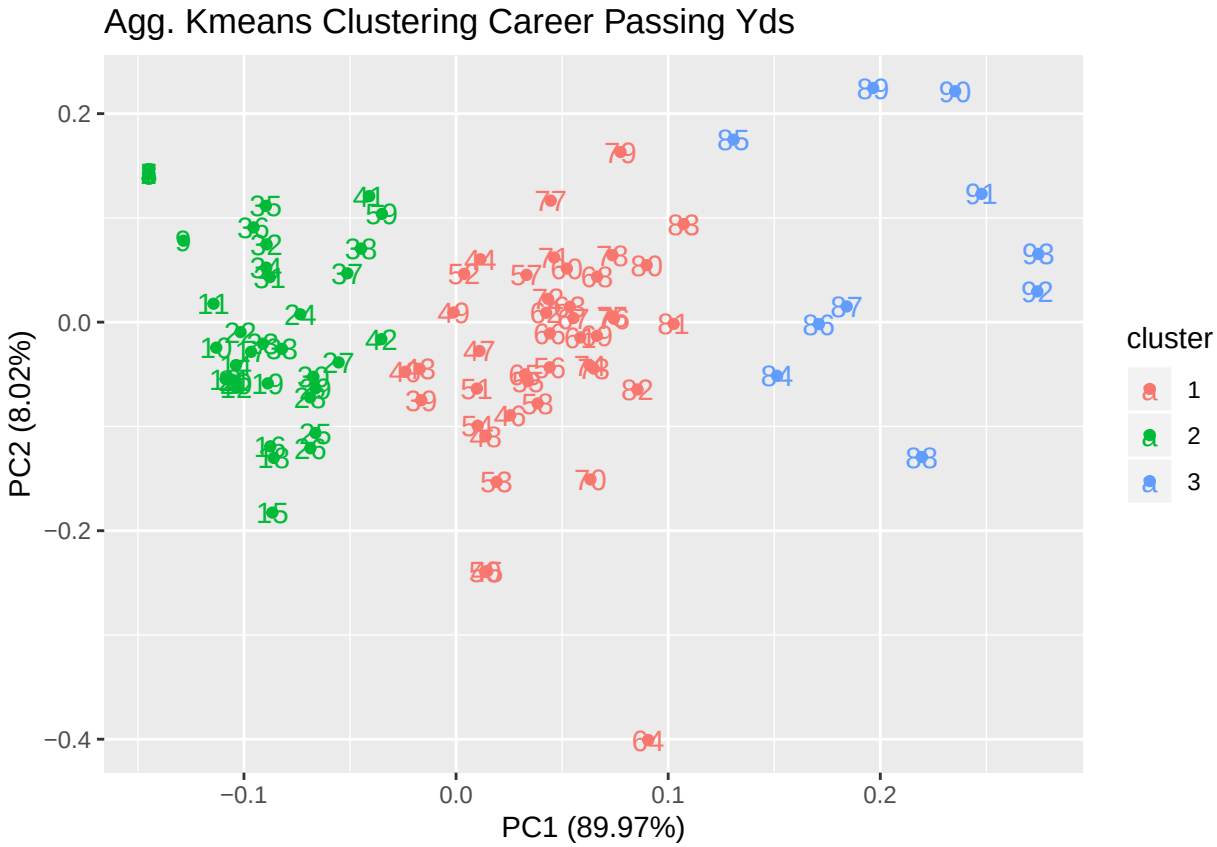


These two components explain 89.23 % of the point variability.

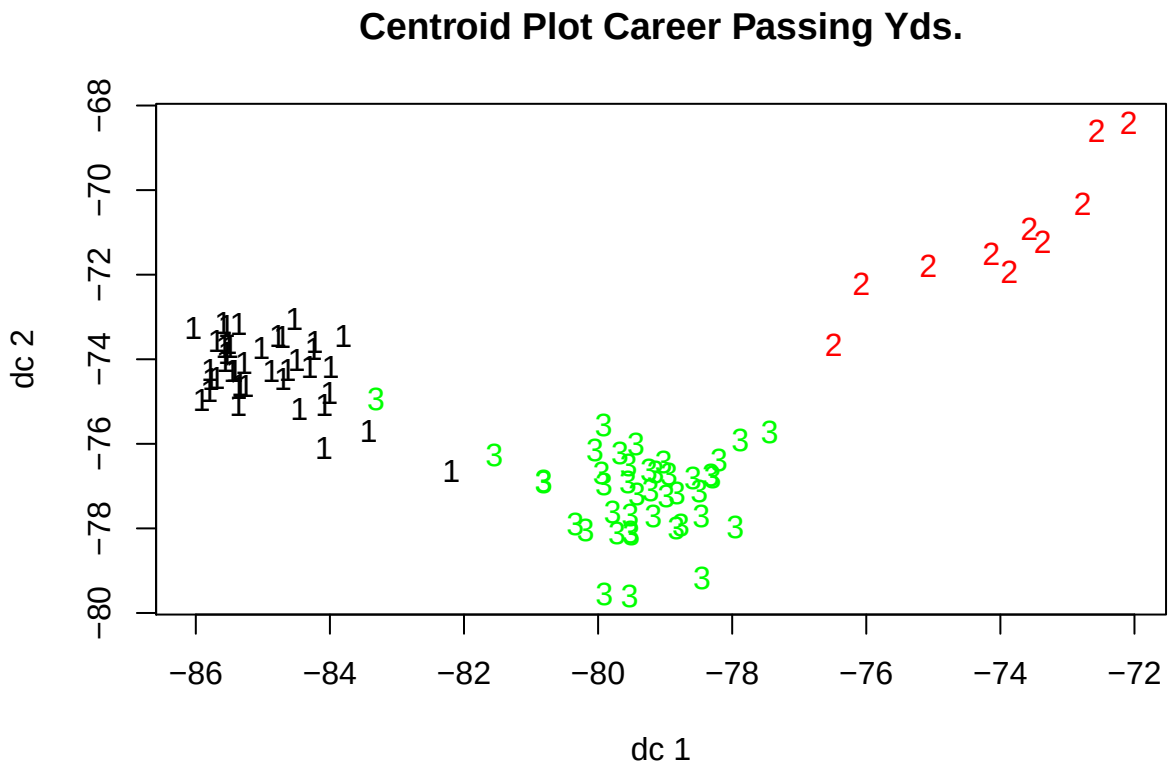
```
#plot(prcomp(CS.PassClust1))
#ggplot(CS.DF)

autoplot(prcomp(CS.PassClust1,CSPass.kmeans$cluster,colour = 'Year'
                ,shade = FALSE,labels = 0,lines = 0))
```




```
set.seed(5)
autoplot(kmeans(CS.PassClust1, 3), data = CS.PassClust1,
         label = TRUE, main = 'Agg. Kmeans Clustering Career Passing Yds')
```

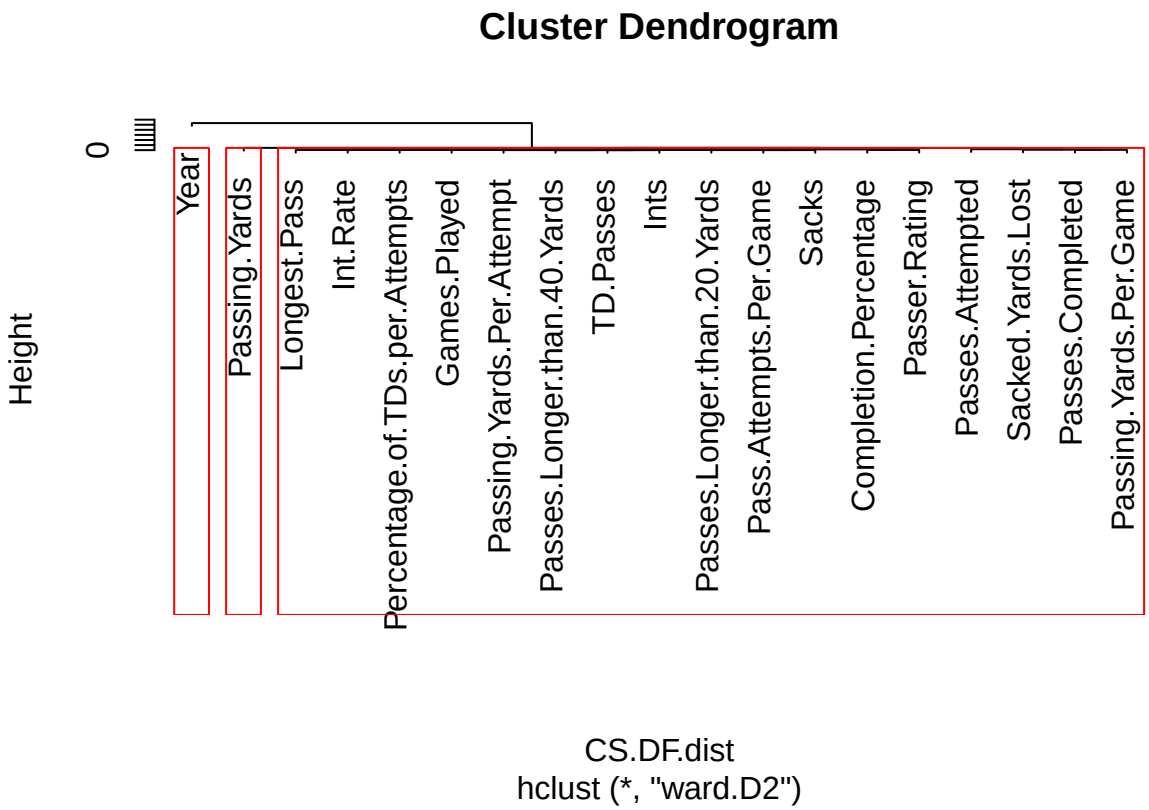



```
#set.seed(100)
#autoplot(kmeans(CS.PassClust,3),data = CS.PassClust)

# Centroid Plot against 1st 2 discriminant functions
plotcluster(CS.PassClust1, CSPass.kmeans1$cluster,main ='Centroid Plot Career Passing Yds.')
```

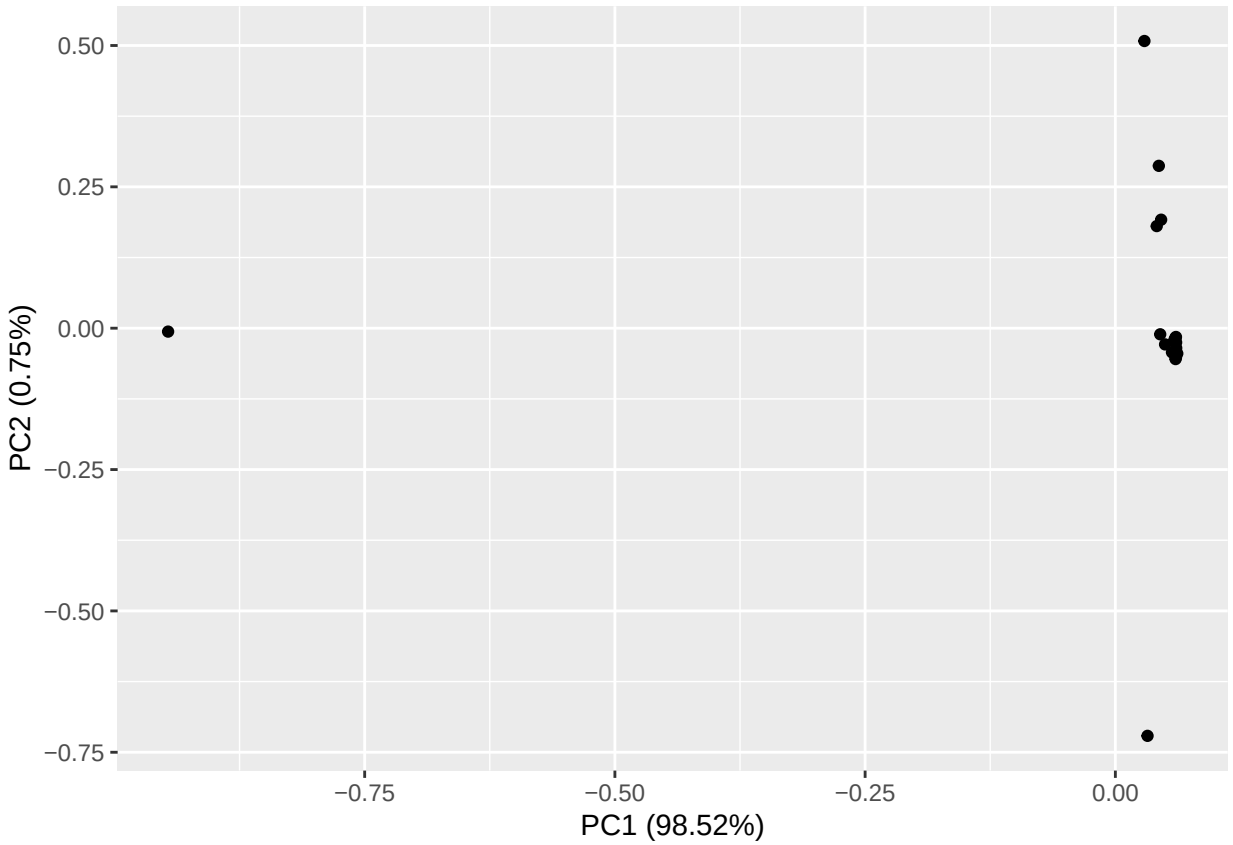


HAC Clustering for CareerPassing

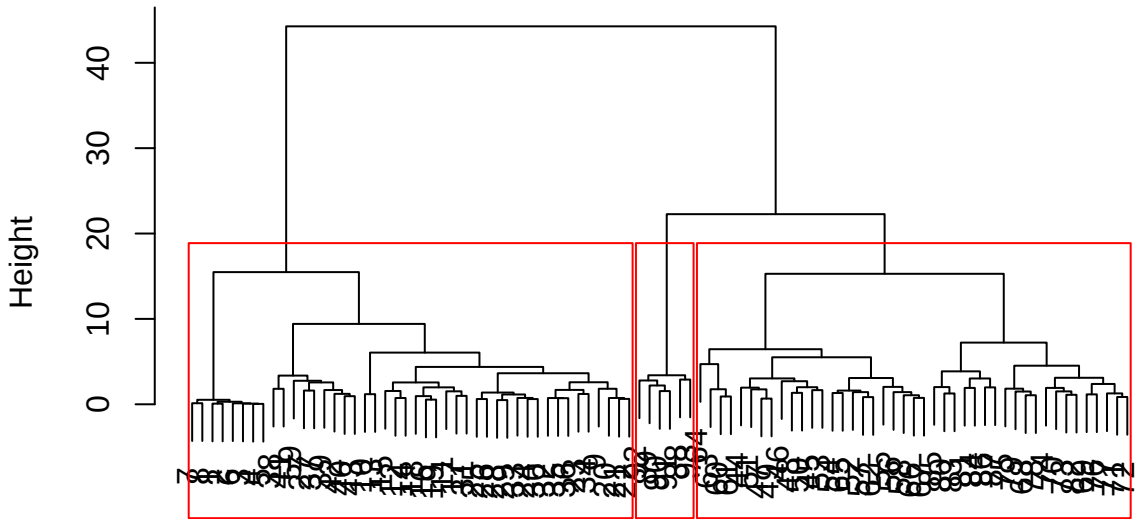
```
CS.DF <- as.data.frame(CS.PassClust)
CS.DF.Scale <- scale(CS.DF)
CS.DF.dist <- dist(CS.DF.Scale,method="euclidean")
CS.DF.fit <- hclust(CS.DF.dist, method="ward.D2")
plot(CS.DF.fit)
rect.hclust(CS.DF.fit, k = 3,)
```



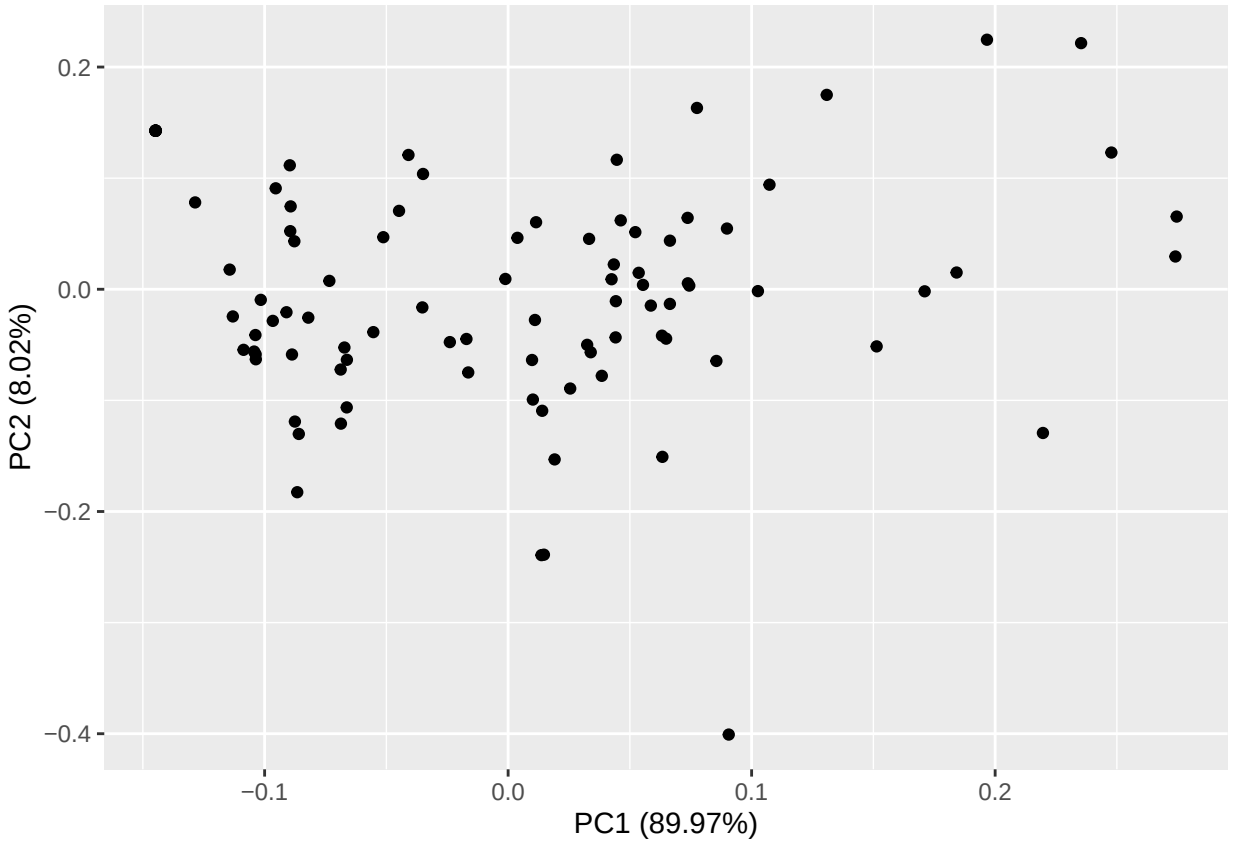

```
autoplot(prcomp(CS.DF))
```




```
CS.DF1 <- as.data.frame(CS.PassClust1)
CS.DF1.Scale <- scale(CS.DF1)
CS.DF1.dist <- dist(CS.DF1.Scale,method="euclidean")
CS.DF1.fit <- hclust(CS.DF1.dist, method="ward.D2")
plot(CS.DF1.fit, main = "Career Passing Dendogram")
rect.hclust(CS.DF1.fit, k = 3)
```




```
autoplot(prcomp(CS.DF1))
```




```
#Looking for completeness and average
CS.DF1.dist.complete <-  hclust(CS.DF1.dist,method="complete")
```

```
CS.DF1.dist.complete
```

```
##
## Call:
## hclust(d = CS.DF1.dist, method = "complete")
##
## Cluster method   : complete
## Distance         : Euclidean
## Number of objects: 93
```

```
CS.DF1.dist.average <-  hclust(CS.DF1.dist,method="average")
CS.DF1.dist.average
```

```
##
## Call:
## hclust(d = CS.DF1.dist, method = "average")
##
## Cluster method   : average
## Distance         : Euclidean
## Number of objects: 93
```

Career Stats Rushing

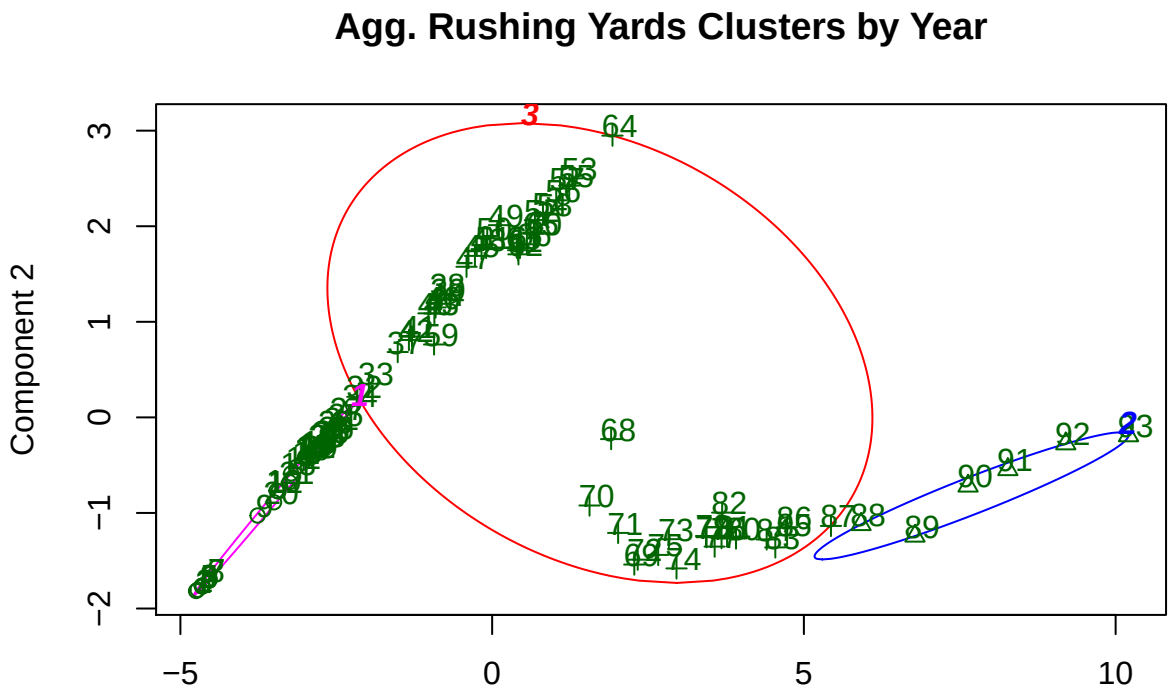
```
#CS.Rush1 <- fastDiscretization(dataSet = CS.Rush,
      #bins = list(Year = c(1940, 1950, 1960, 1970, 1980,1990,2000,2010,2020)))
CS.Rush1 <- (CS.Rush[,-1])
#Building CLusters
CS.RushClust <- t(CS.Rush[,-1])
CS.RushClust <- as.matrix(CS.RushClust)

#Creating a Cluster for aggregation by Year
CS.RushClust1<- (CS.Rush[,-1])
CS.RushClust1<-aggregate(.~Year, data=CS.RushClust1, FUN = sum)
CS.RushClust1<- as.matrix(CS.RushClust1)

#Create Kmeans cluster
set.seed(100)
k <- 3 # number of clusters
CSRush.kmeans <- kmeans(CS.RushClust, k)
CSRush.kmeans$centers<- round(CSRush.kmeans$centers, digits = 3) # cluster centers
(CSRush.kmeans$cluster)
```

```
##                              Year                       Games.Played
##                                 1                                  3
##                  Rushing.Attempts          Rushing.Attempts.Per.Game
##                                 3                                  3
##                     Rushing.Yards                    Yards.Per.Carry
##                                 2                                  3
##            Rushing.Yards.Per.Game                       Rushing.TDs
##                                 3                                  3
##               Longest.Rushing.Run               Rushing.First.Downs
##                                 3                                  3
## Percentage.of.Rushing.First.Downs        Rushing.More.Than.20.Yards
##                                 3                                  3
##        Rushing.More.Than.40.Yards                           Fumbles
##                                 3                                  3
```

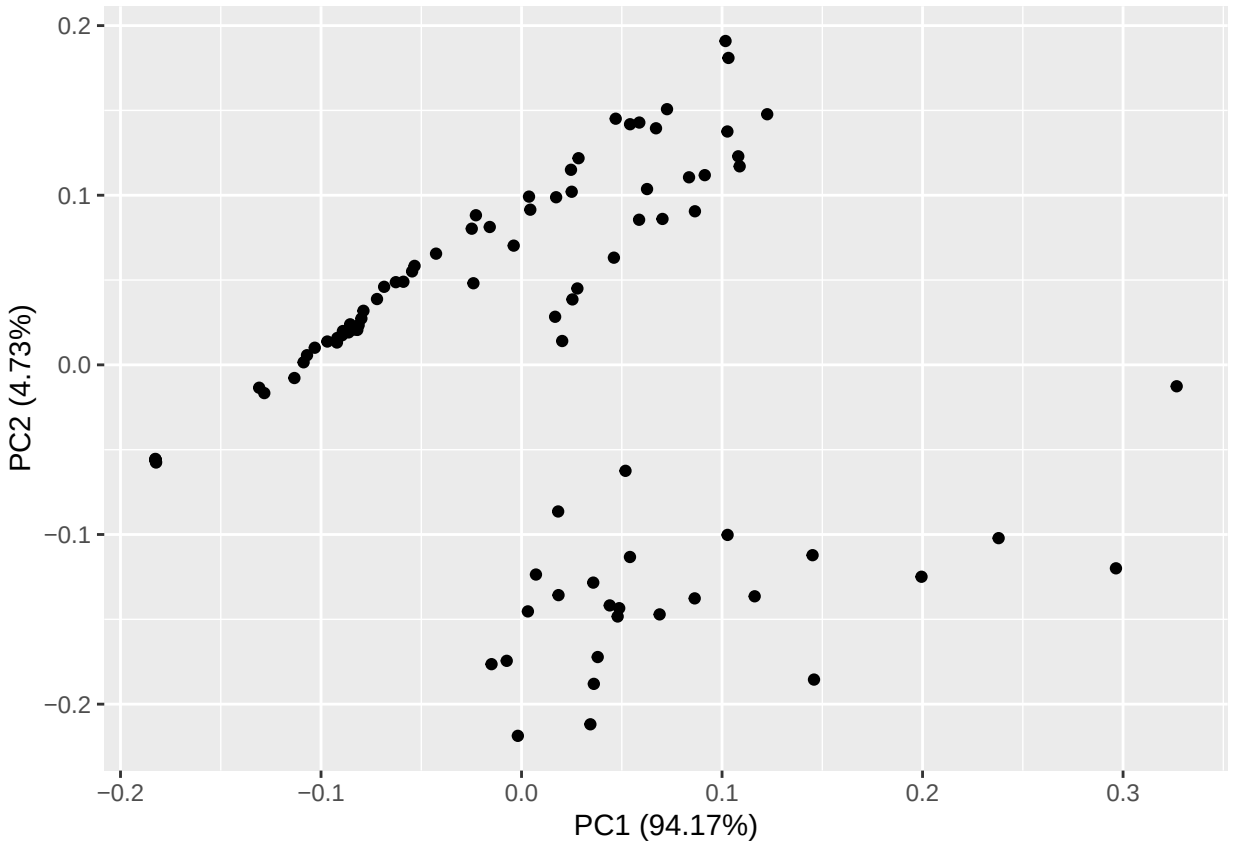
```
#Creating Kmeans for Agregated CLsuter
set.seed(100)
k <- 3 # number of clusters
CSRush.kmeans1 <- kmeans(CS.RushClust1, k)
CSRush.kmeans1$centers<- round(CSRush.kmeans1$centers, digits = 3) # cluster centers
(CSRush.kmeans1$cluster)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 1 1
## [36] 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [71] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2
```

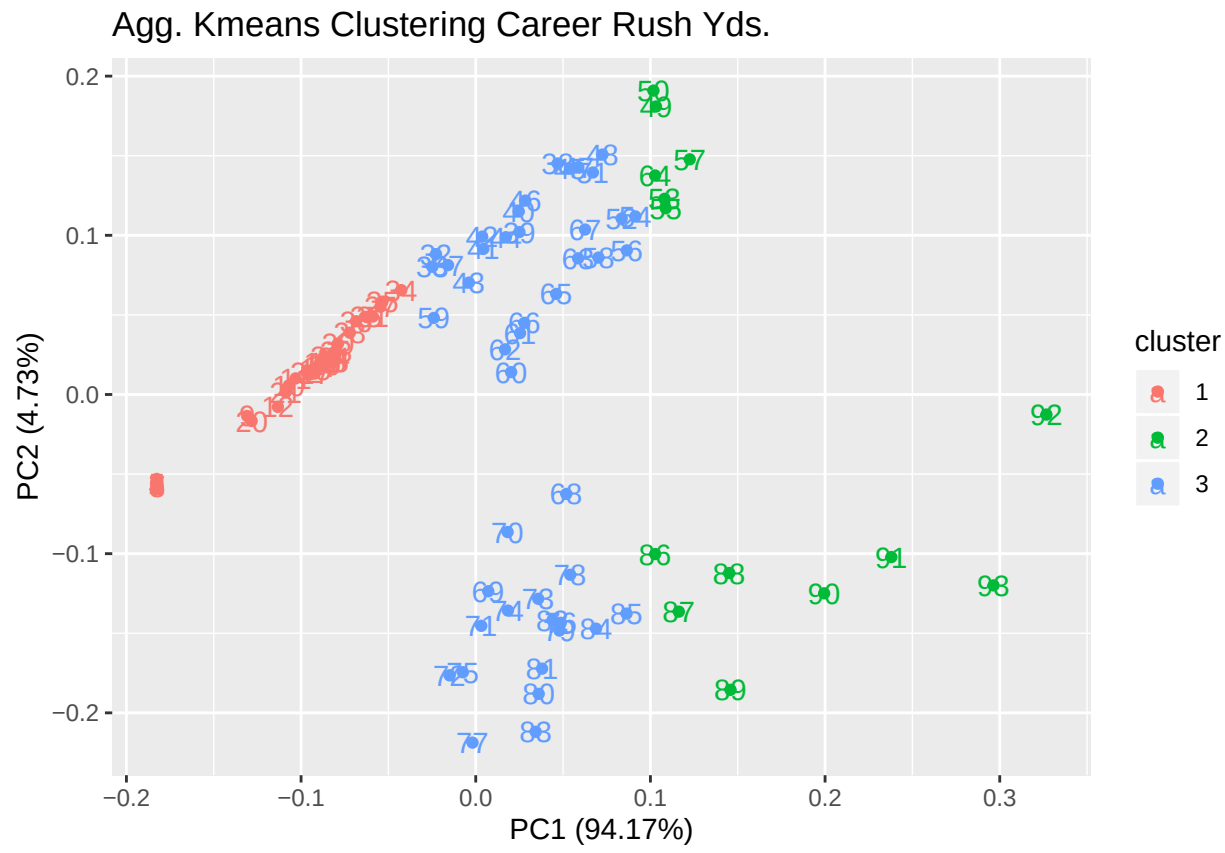
```
#Clusterplot for Kmeans Cluster
clusplot(CS.RushClust1,CSRush.kmeans1$cluster,color =  TRUE ,shade = FALSE,
         labels = 2,lines = 0 , main = 'Agg. Rushing Yards Clusters by Year')
```



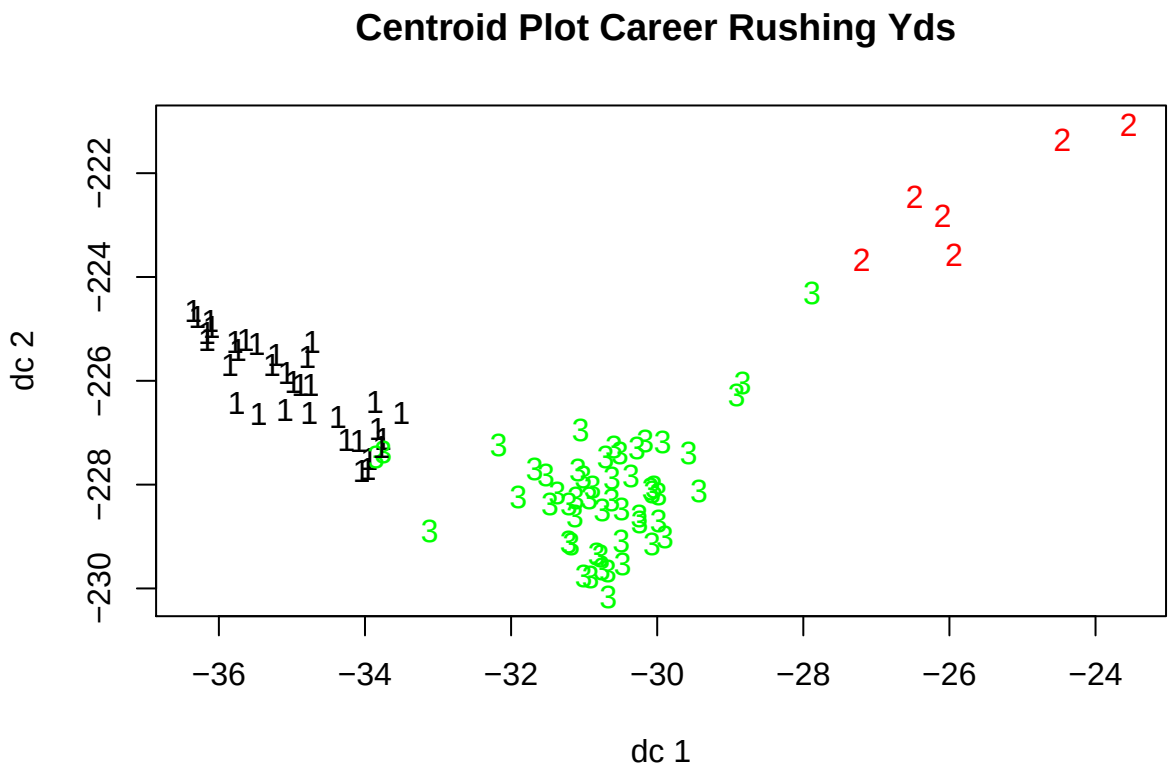

```
#plot(prcomp(CS.RushClust1))
#ggplot(CS.DF)

autoplot(prcomp(CS.RushClust1,CSRush.kmeans$cluster,colour = 'Year',
                shade = FALSE,labels = 0,lines = 0))
```




```
set.seed(5)
autoplot(kmeans(CS.RushClust1, 3), data = CS.RushClust1,
         label = TRUE, main = 'Agg. Kmeans Clustering Career Rush Yds.')
```

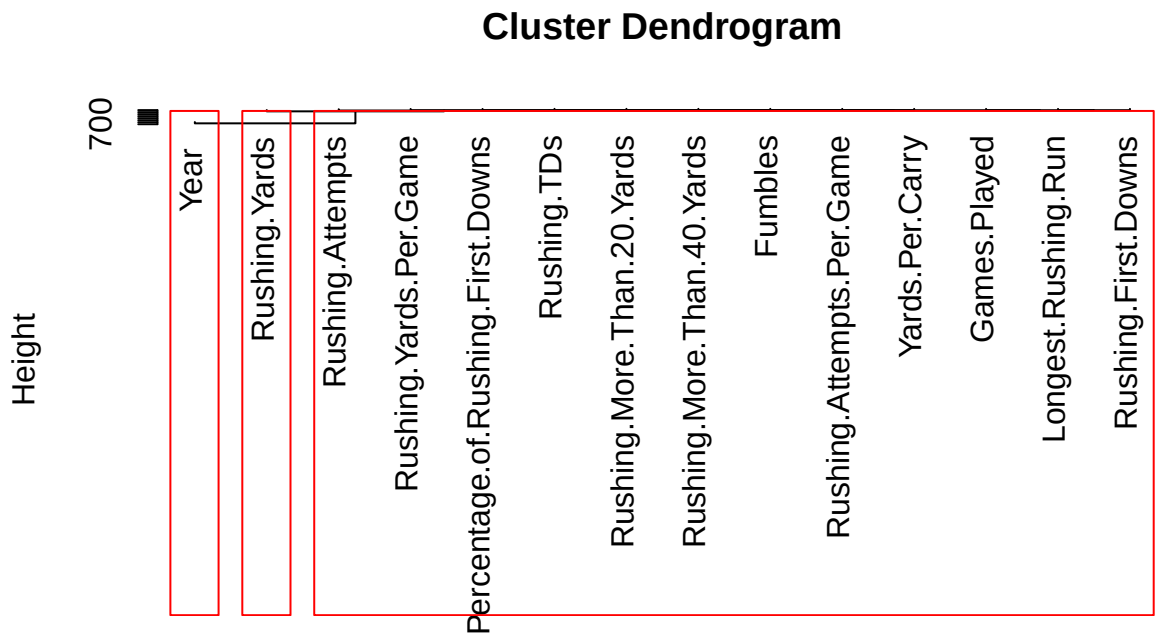


```
#set.seed(100)
#autoplot(kmeans(CS.RushClust,3),data = CS.RushClust)

# Centroid Plot against 1st 2 discriminant functions
plotcluster(CS.RushClust1, CSRush.kmeans1$cluster,main ='Centroid Plot Career Rushing Yds ')
```



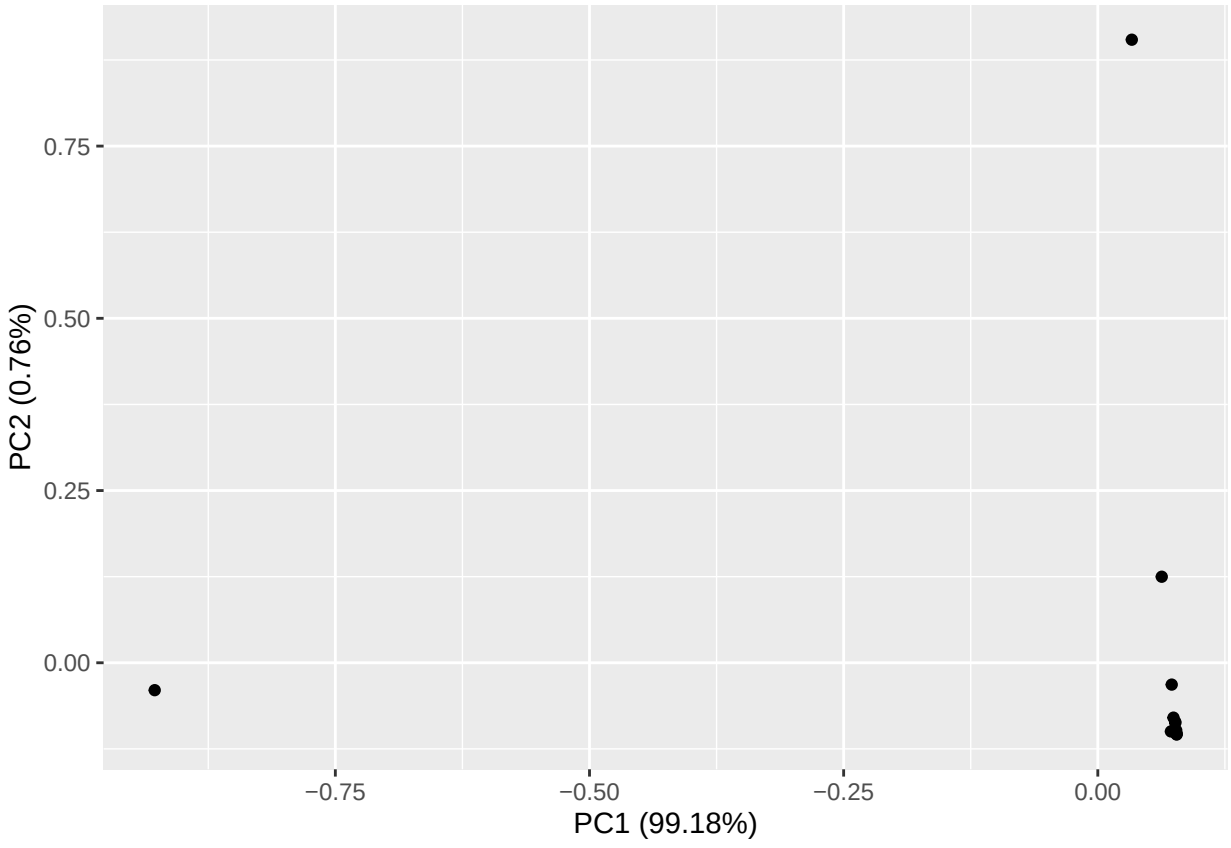
HAC Clustering for Career Rushing

```
CS.DF2 <- as.data.frame(CS.RushClust)
CS.DF2.Scale <- scale(CS.DF2)
CS.DF2.dist <- dist(CS.DF2.Scale,method="euclidean")
CS.DF2.fit <- hclust(CS.DF2.dist, method="ward.D2")
plot(CS.DF2.fit)
rect.hclust(CS.DF2.fit, k = 3,)
```



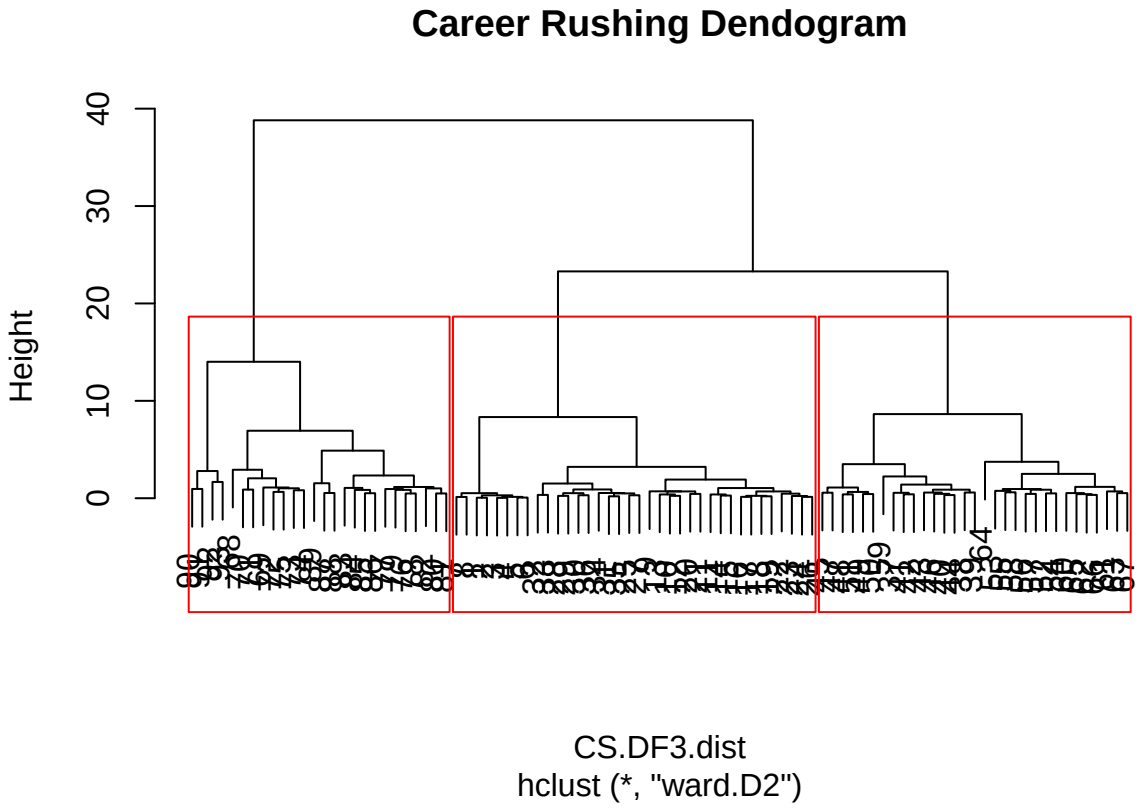
CS.DF2.dist
hclust (*, "ward.D2")

```
autoplot(prcomp(CS.DF2))
```

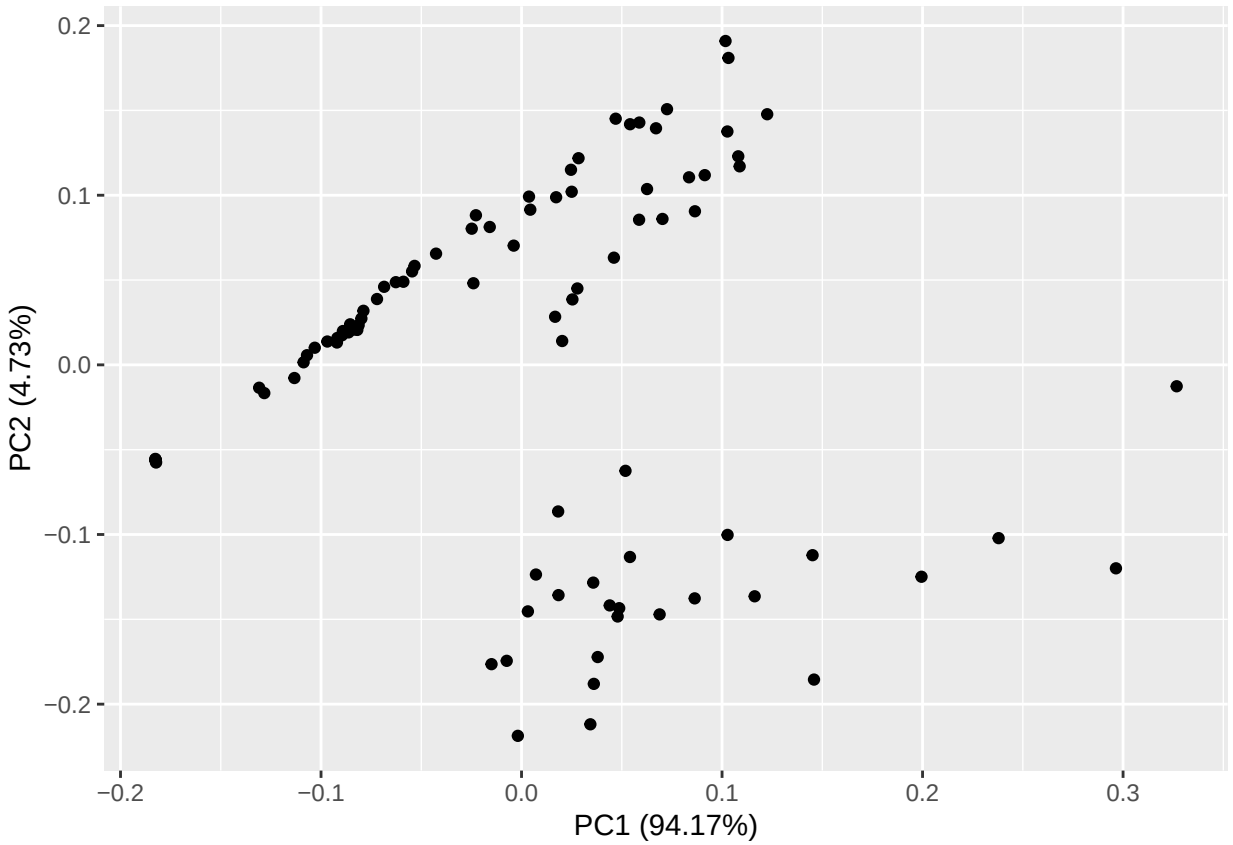



```
CS.DF3 <- as.data.frame(CS.RushClust1)
CS.DF3.Scale <- scale(CS.DF3)
```

```
CS.DF3.dist <- dist(CS.DF3.Scale,method="euclidean")
CS.DF3.fit <- hclust(CS.DF3.dist, method="ward.D2")
plot(CS.DF3.fit , main = "Career Rushing Dendogram")
rect.hclust(CS.DF3.fit, k = 3)
```




```
autoplot(prcomp(CS.DF3))
```




```
#Looking for completeness and average
CS.DF3.dist.complete <-  hclust(CS.DF3.dist,method="complete")
CS.DF3.dist.complete
```

```
##
## Call:
## hclust(d = CS.DF3.dist, method = "complete")
##
## Cluster method   : complete
## Distance         : Euclidean
## Number of objects: 93
```

```
CS.DF3.dist.average <-  hclust(CS.DF3.dist,method="average")
CS.DF3.dist.average
```

```
##
## Call:
## hclust(d = CS.DF3.dist, method = "average")
##
## Cluster method   : average
## Distance         : Euclidean
## Number of objects: 93
```

Career Stats Receving

```
#CS.Rec1 <- fastDiscretization(dataSet = CS.Rec,
          #bins = list(Year = c(1940, 1950, 1960, 1970, 1980,1990,2000,2010,2020)))
CS.Rec1 <- (CS.Rec[,-1])
#Building CLusters
CS.RecClust <- t(CS.Rec[,-1])
CS.RecClust <- as.matrix(CS.RecClust)
```

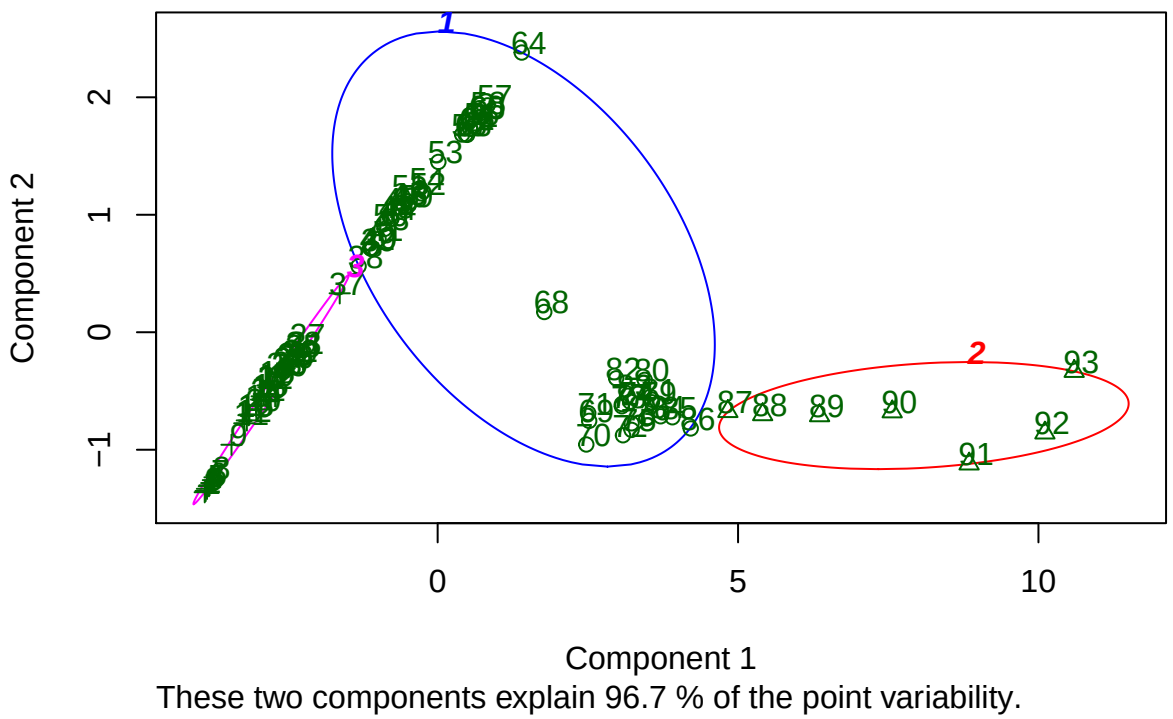
```r
#Creating a Cluster for aggregation by Year
CS.RecClust1<- (CS.Rec[,-1])
CS.RecClust1<-aggregate(.~Year, data=CS.RecClust1, FUN = sum)
CS.RecClust1<- as.matrix(CS.RecClust1)

#Create Kmeans cluster
set.seed(100)
k <- 3 # number of clusters
CSRec.kmeans <- kmeans(CS.RecClust, k)
CSRec.kmeans$centers<- round(CSRec.kmeans$centers, digits = 3) # cluster centers
(CSRec.kmeans$cluster)
```

```
##                              Year                     Games.Played
##                                 2                                1
##                        Receptions                   Receiving.Yards
##                                 1                                3
##              Yards.Per.Reception                   Yards.Per.Game
##                                 1                                1
##                Longest.Reception                     Receiving.TDs
##                                 1                                1
## Receptions.Longer.than.20.Yards Receptions.Longer.than.40.Yards
##                                 1                                1
##            First.Down.Receptions                          Fumbles
##                                 1                                1
```

```r
#Creating Kmeans for Agregated CLsuter
set.seed(100)
k <- 3 # number of clusters
CSRec.kmeans1 <- kmeans(CS.RecClust1, k)
CSRec.kmeans1$centers <- round(CSRec.kmeans1$centers, digits = 3) # cluster centers
(CSRec.kmeans1$cluster)
```

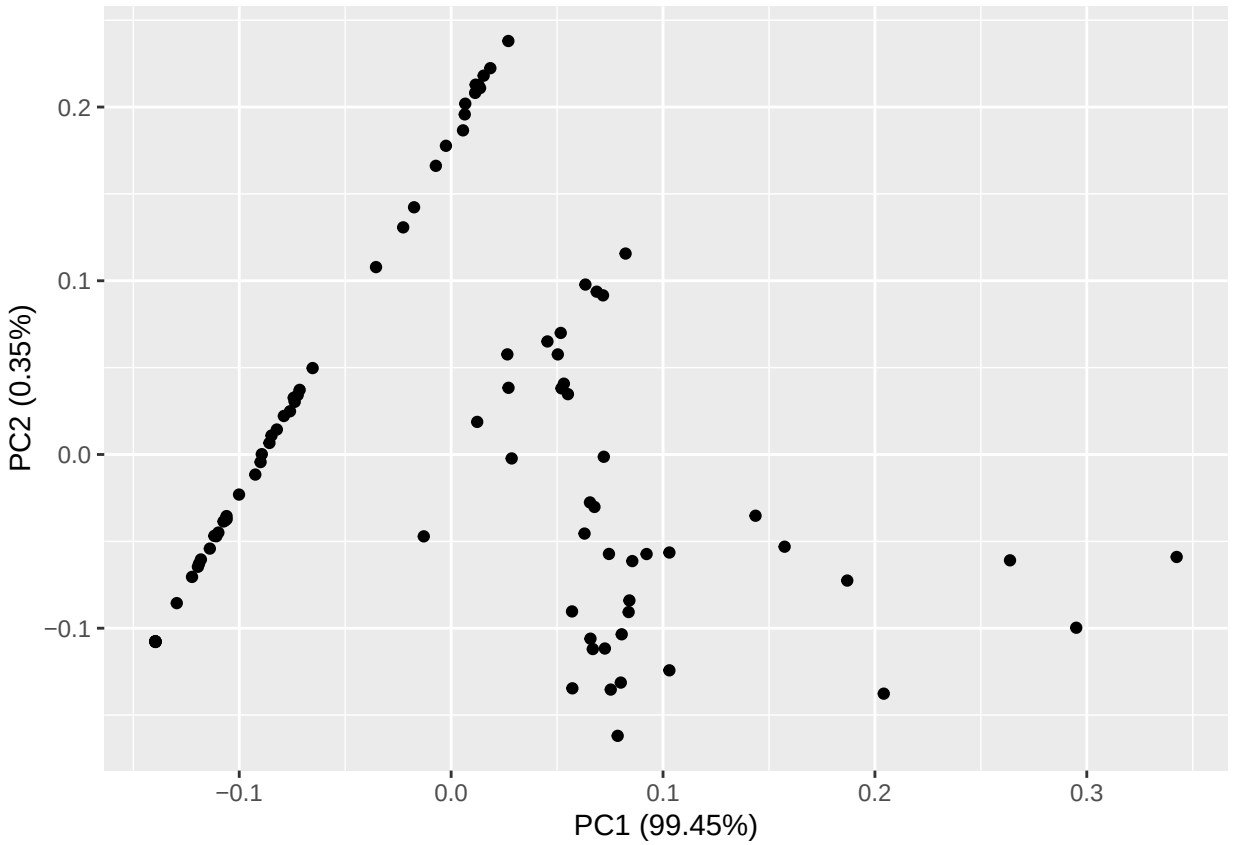
```
##  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [36] 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2
```

```r
#Clusterplot for Kmeans Cluster
clusplot(CS.RecClust1,CSRec.kmeans1$cluster,color =  TRUE ,shade = FALSE,
         labels = 2,lines = 0 , main = 'Agg. Rec. Yards Clusters by Year')
```

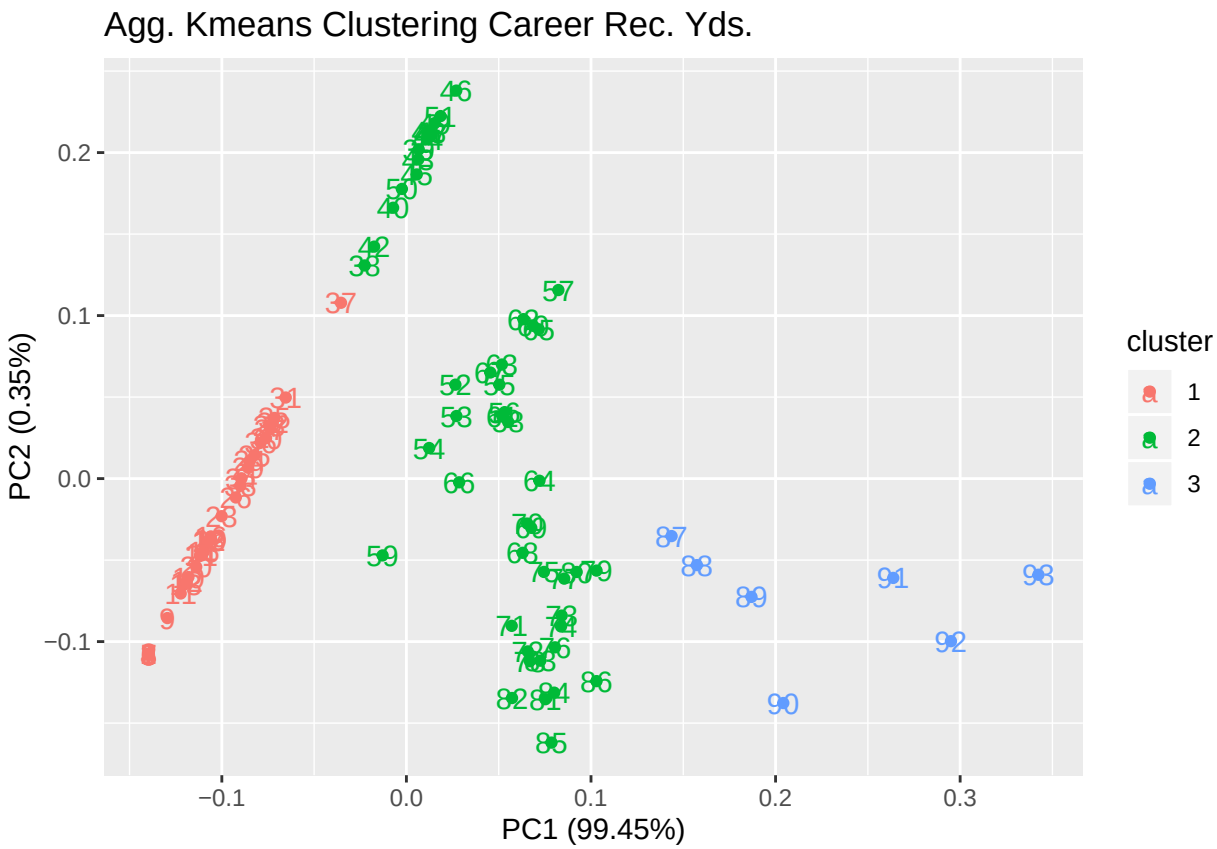


These two components explain 96.7 % of the point variability.

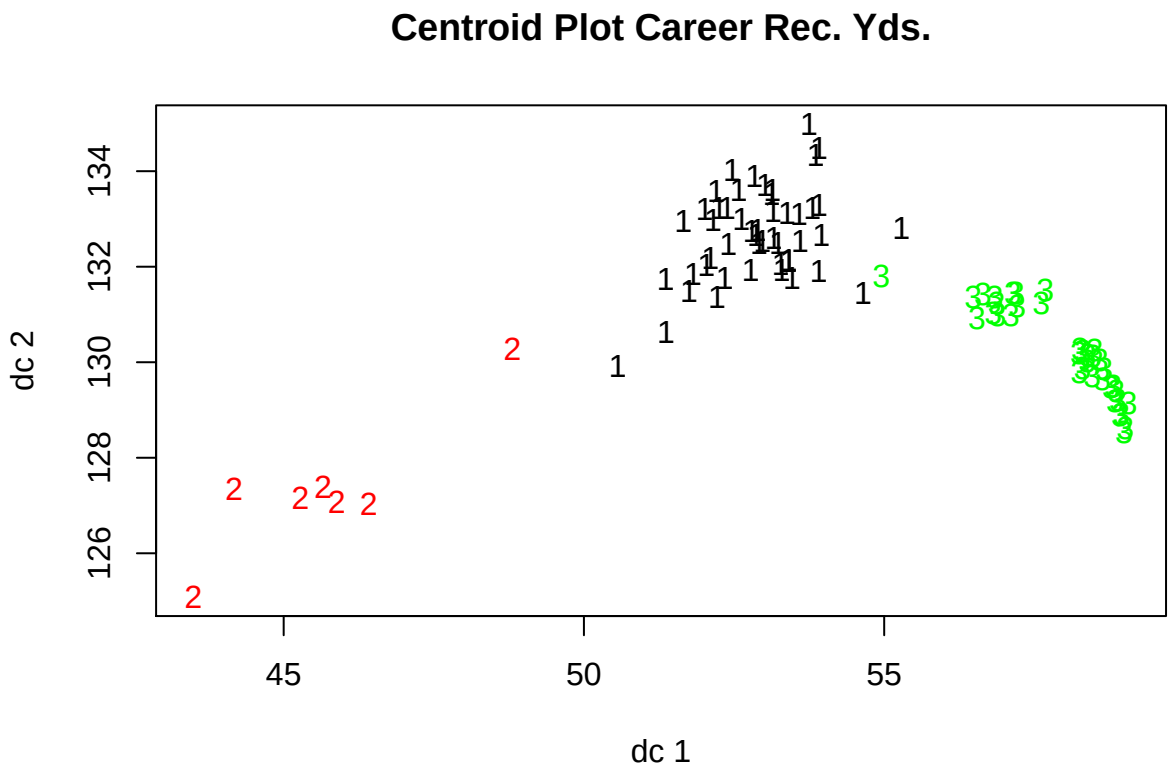
```
#plot(prcomp(CS.RecClust1))
#ggplot(CS.DF)

autoplot(prcomp(CS.RecClust1,CSRec.kmeans$cluster,colour = 'Year',
                shade = FALSE,labels = 0,lines = 0))
```




```
set.seed(5)
autoplot(kmeans(CS.RecClust1, 3), data = CS.RecClust1,
         label = TRUE, main = 'Agg. Kmeans Clustering Career Rec. Yds.')
```

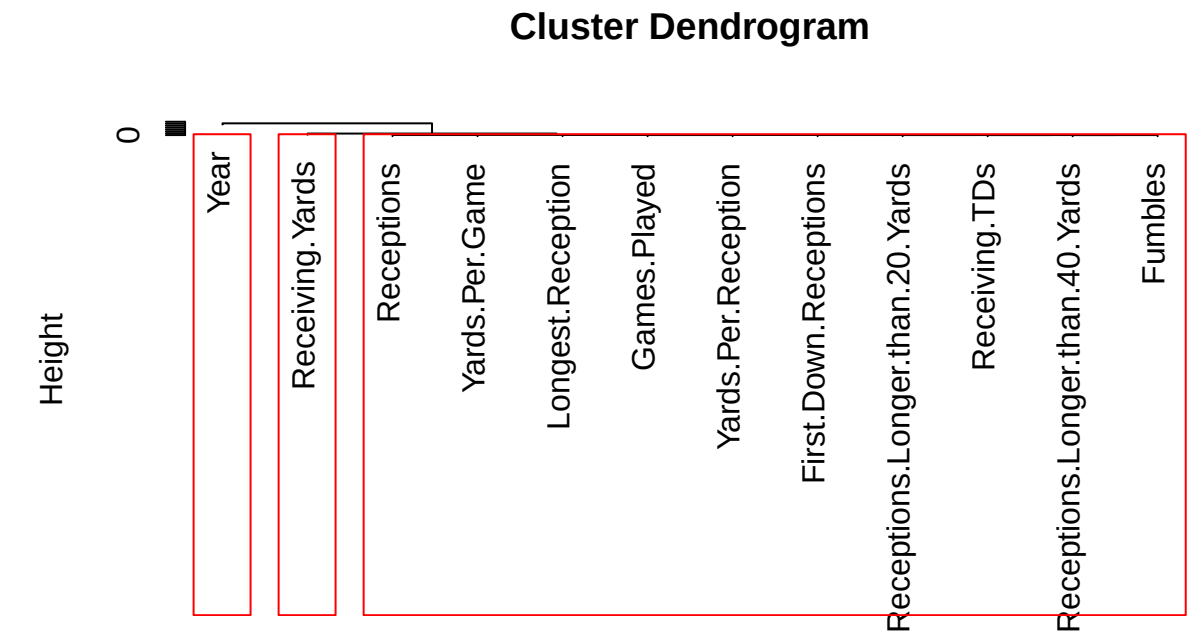


```r
#set.seed(100)
#autoplot(kmeans(CS.RecClust,3),data = CS.RecClust)

# Centroid Plot against 1st 2 discriminant functions
plotcluster(CS.RecClust1, CSRec.kmeans1$cluster,main ='Centroid Plot Career Rec. Yds.')
```



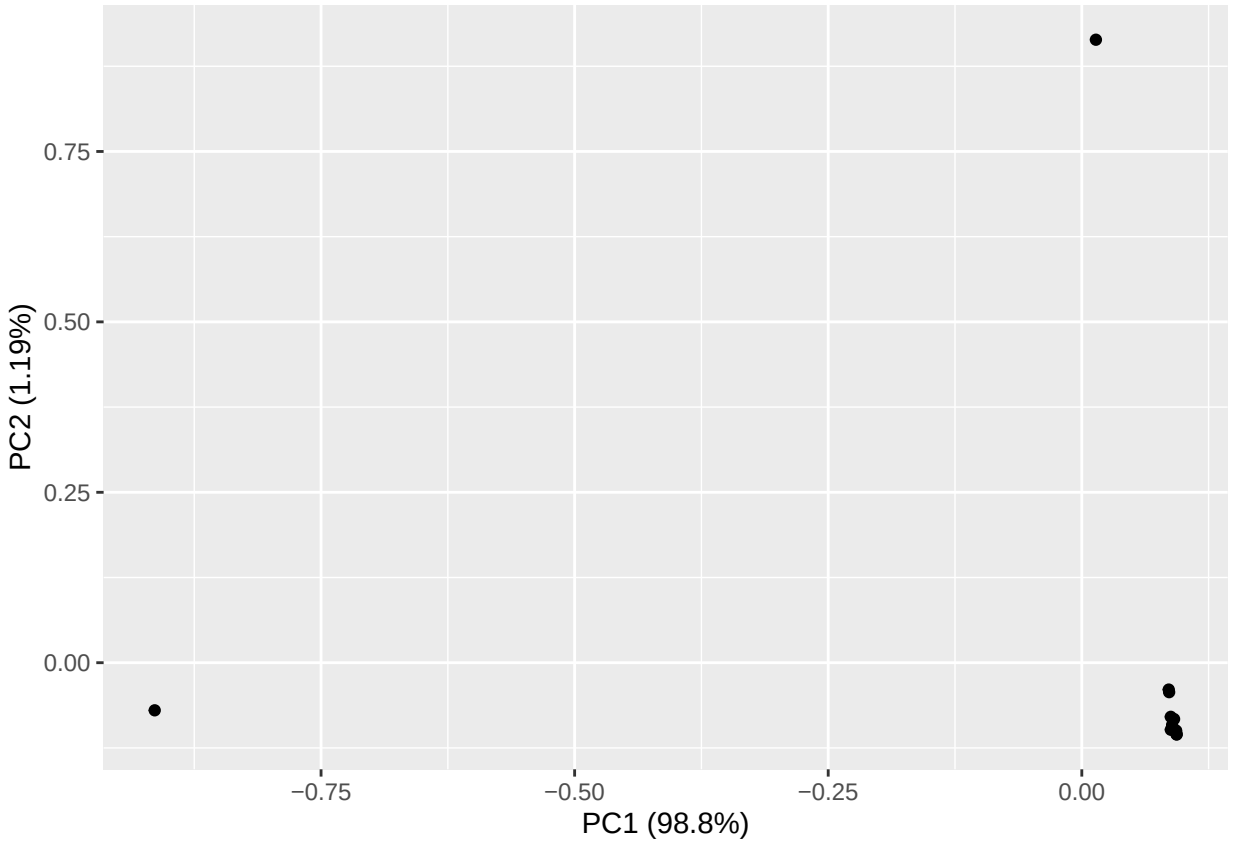
HAC Clustering for Career Receving

```r
CS.DF4 <- as.data.frame(CS.RecClust)
CS.DF4.Scale <- scale(CS.DF4)
CS.DF4.dist <- dist(CS.DF4.Scale,method="euclidean")
CS.DF4.fit <- hclust(CS.DF4.dist, method="ward.D2")
plot(CS.DF4.fit)
rect.hclust(CS.DF4.fit, k = 3,)
```



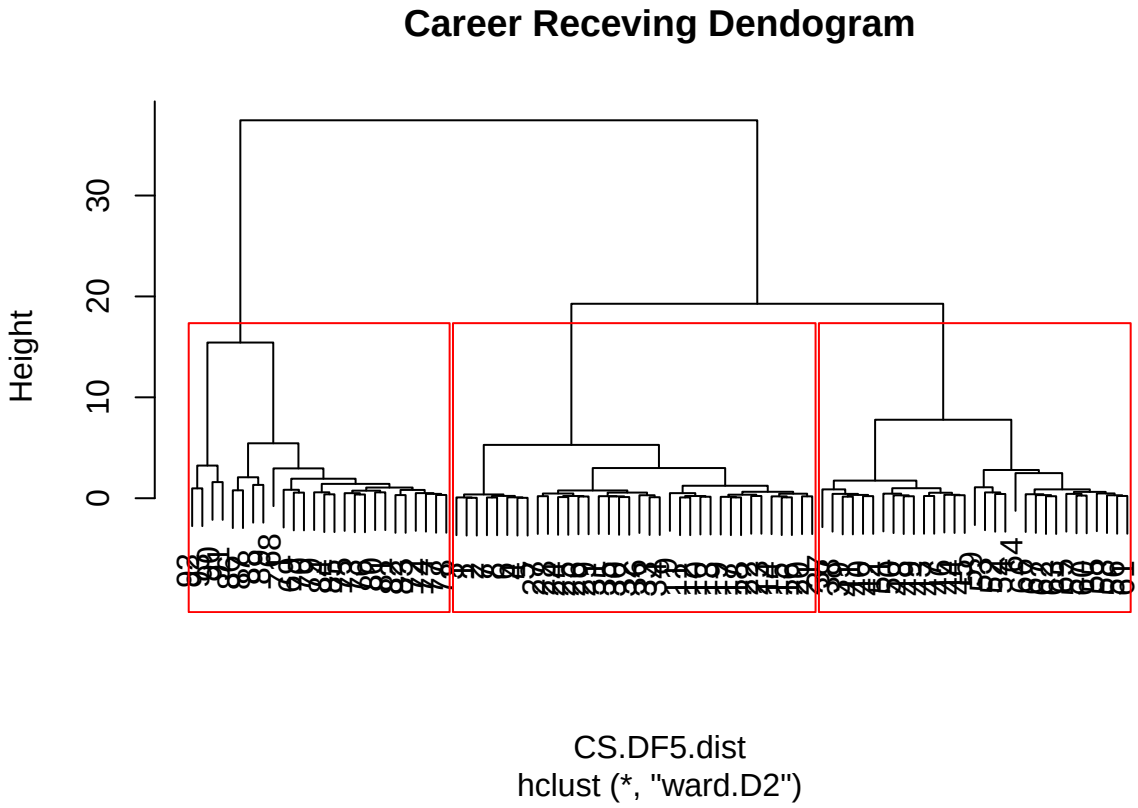
CS.DF4.dist
hclust (*, "ward.D2")

```
autoplot(prcomp(CS.DF4))
```

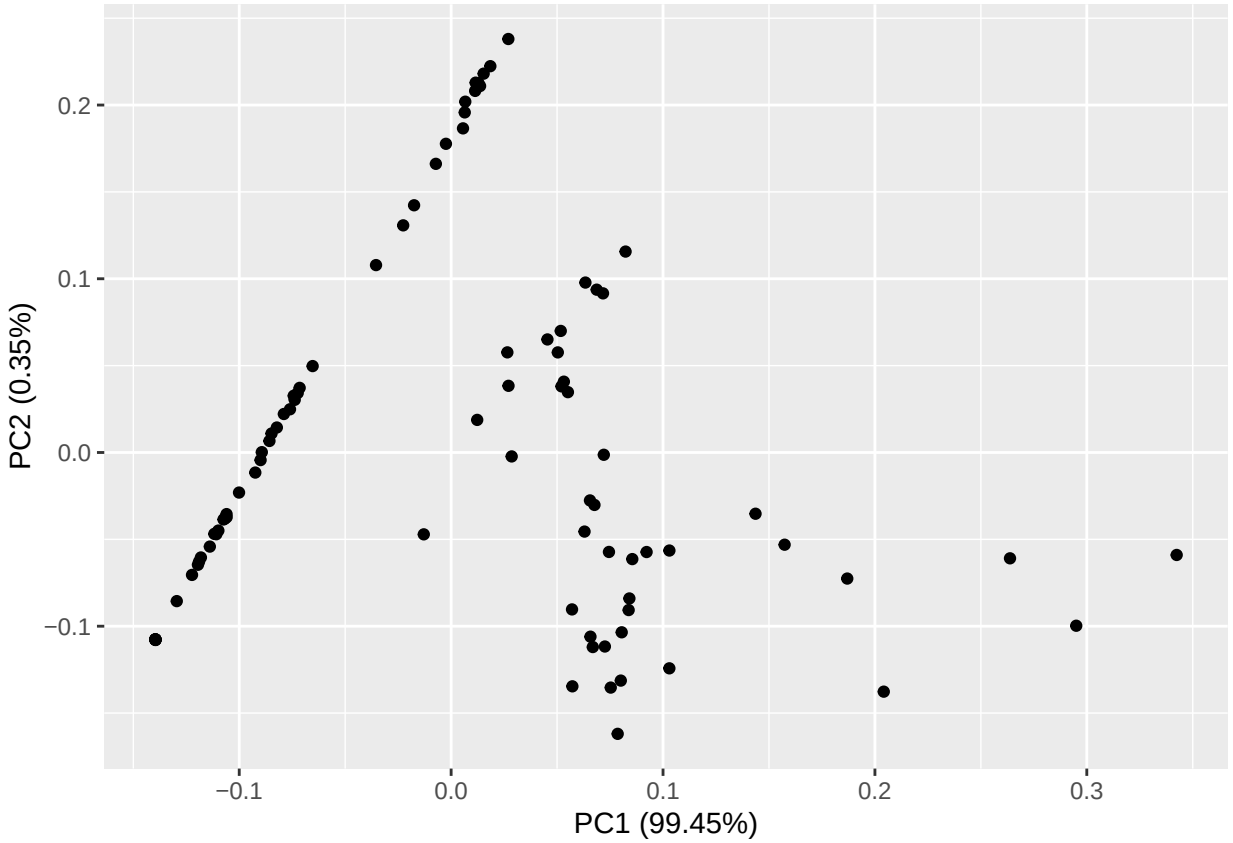



```
CS.DF5 <- as.data.frame(CS.RecClust1)
CS.DF5.Scale <- scale(CS.DF5)
```

```
CS.DF5.dist <- dist(CS.DF5.Scale,method="euclidean")
CS.DF5.fit <- hclust(CS.DF5.dist, method="ward.D2")
plot(CS.DF5.fit, main = "Career Receving Dendogram")
rect.hclust(CS.DF5.fit, k = 3)
```




```
autoplot(prcomp(CS.DF5))
```




```
#Looking for completeness and average
CS.DF5.dist.complete <-  hclust(CS.DF1.dist,method="complete")
CS.DF5.dist.complete
```

```
##
## Call:
## hclust(d = CS.DF1.dist, method = "complete")
##
## Cluster method   : complete
## Distance         : Euclidean
## Number of objects: 93
```

```
CS.DF5.dist.average <-  hclust(CS.DF1.dist,method="average")
CS.DF5.dist.average
```

```
##
## Call:
## hclust(d = CS.DF1.dist, method = "average")
##
## Cluster method   : average
## Distance         : Euclidean
## Number of objects: 93
```

Looking at the distribution of the clusters , we do notice that there are certain overlaps when plotted before the aggregation of the yards, but after aggregation the overlap reduces. This clustering analysis doesnt provide any promising insight on how the data is distributed in the dataset, but it gives the observation as how the number passing,receving,rushing yards increased with year and dropped in the recent years.

Decision Tree

```
# In order to help with cleaner decision trees later on, we will bin the fantasy
#data that we generated in the beginning of the code. We will apply this binning method
#to QBs and RB/WRs, since these are the positions that we will be analyzing at a deeper level.

GL.QB.Season$OverUnder18 <- ifelse(GL.QB.Season$Fantasy >= 18, "Over18", "Under18")
GL.RB.WR.Season$OverUnder10 <- ifelse(GL.RB.WR.Season$Fantasy >= 10, "Over10", "Under10")

#Labeling the RB/WRs with respective position based on how the player played in the game
GL.RB.WR.Season$Position <- ifelse(GL.RB.WR.Season$Receptions >
                                   GL.RB.WR.Season$Rushing.Attempts, "Reciever", "RunningBack")
```

Sampling

The sport of football has evolved much over the decades. To get an accurate prediction model, let's create two subsets of our data. The first subset will be for the most recent season of data; 2016. The second subset of data will be for the 2010 years up until 2016. This should give us similar data that we can analyze and make decisions based off of. Let's also sample our data into a training and testing subset. We will create testing and training subsets of the QB and RB/WR dataframes.

```
QB.2016 <- filter(GL.QB.Season, Year == 2016)
RB.WR.2016 <- filter(GL.RB.WR.Season, Year == 2016)
QB.10s <- filter(GL.QB.Season, Year > 2009)
RB.WR.10s <- filter(GL.RB.WR.Season, Year > 2009)

#Set the training ratio for 60% of our data
trainRatio <- 0.6
set.seed(11)

#Sampling
sampleQB <- sample.int(n = nrow(QB.2016), size =
                         floor(trainRatio*nrow(QB.2016)), replace = FALSE)
sampleRBWR <- sample.int(n = nrow(RB.WR.2016), size =
                           floor(trainRatio*nrow(RB.WR.2016)), replace = FALSE)
sampleQB10s <- sample.int(n = nrow(QB.10s), size =
                            floor(trainRatio*nrow(QB.10s)), replace = FALSE)
sampleRBWR10s <- sample.int(n = nrow(RB.WR.10s), size =
                              floor(trainRatio*nrow(RB.WR.10s)), replace = FALSE)

#Creating QB test and train subsets
trainQB <- QB.2016[sampleQB,]
testQB <- QB.2016[-sampleQB,]
trainQB10s <- QB.10s[sampleQB10s,]
testQB10s <- QB.10s[-sampleQB10s,]

#Creating RW.WR test and train subsets
trainRBWR <- RB.WR.2016[sampleRBWR,]
testRBWR <- RB.WR.2016[-sampleRBWR,]
trainRBWR10s <- RB.WR.10s[sampleRBWR10s,]
testRBWR10s <- RB.WR.10s[-sampleRBWR10s,]
```

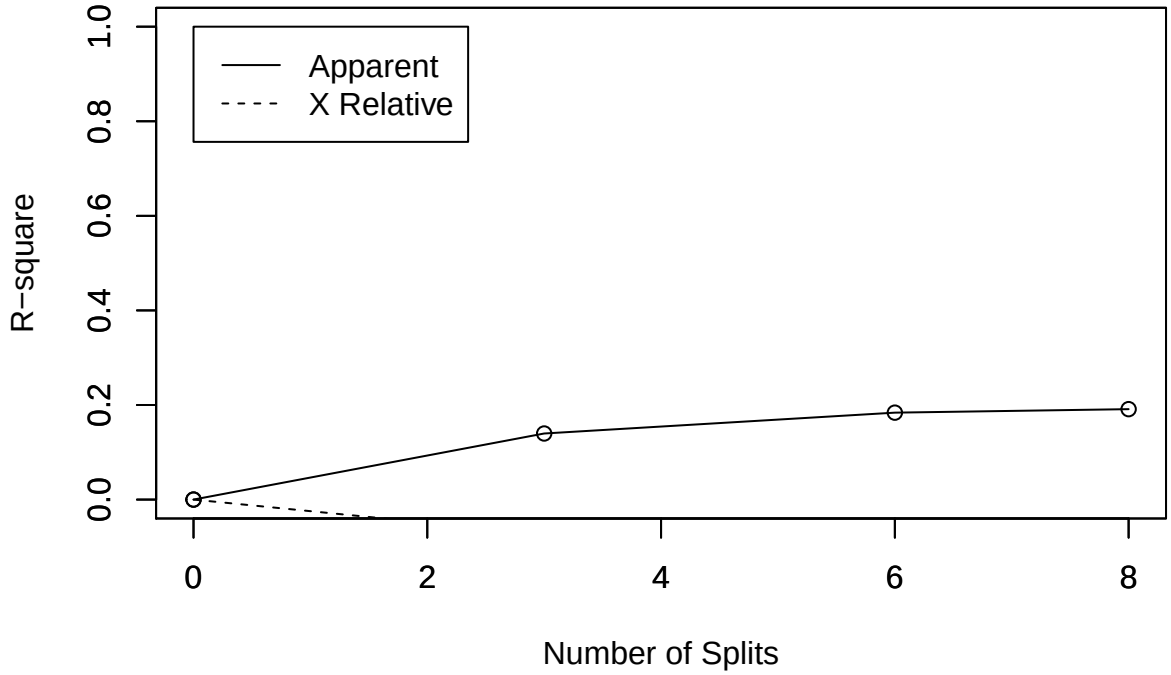
If we use all variables to predict the outcome for fantasy score bin (over/under), then we incur a lot of data processing time with no concise resutls to show for. So we will trim our training data sets to only include a few of the prediction variables. This will dramatically reduce processing time as well as allow us to focus on just a few variables of interest. If you recall from before, the scoring for fantasy points is primarily based on yards gained (passing, rushing, receiving), interceptions (negative), and touchdowns (passed, rushed, recieved). We can easily look at statistics and compare averages for players to predict who will perform better. So our focus will be on some of the underlying factors that aren't explicit from statistics. For this prediction model, we will not focus on just a few variables which do not have a direct relation to

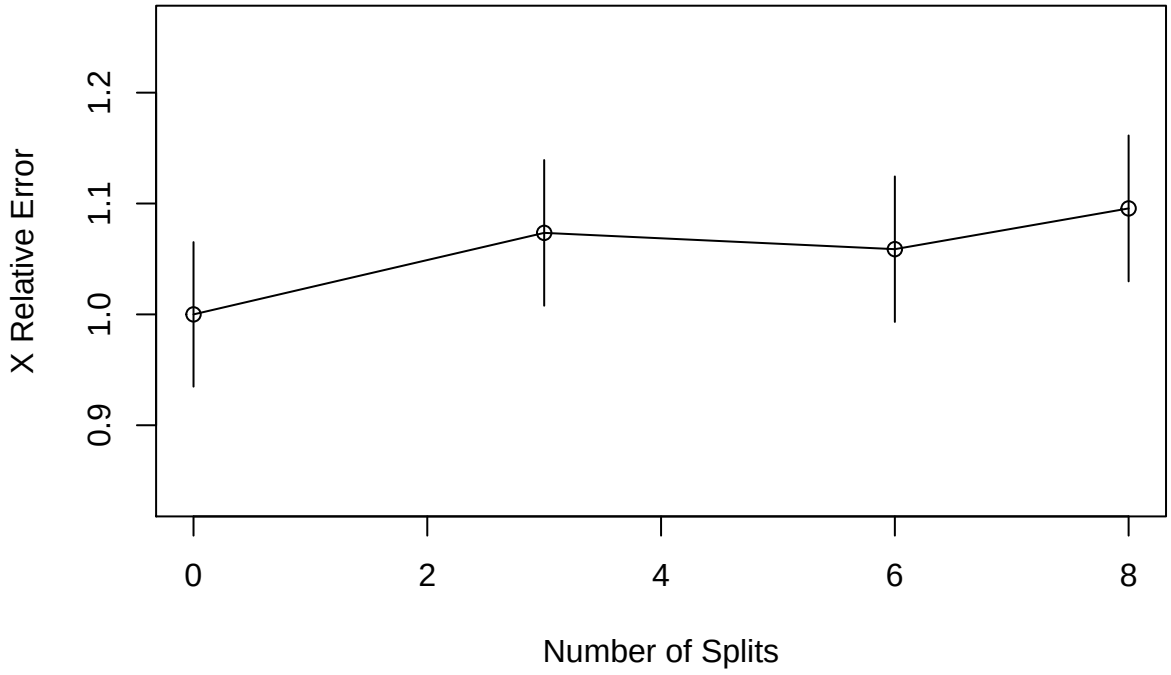
fantasy points to see if they potentially play a role in how a player performs. For QBs, we will take a subset of trainQB which will include Week, Home.or.Away, Passes.Attempted, Sacks, OverUnder18 variables. And for RB/WRs we will take a subset of trainRBWR which will include, Week, Home.or.Away, Games.Started, OverUnder10, Position.

Note, the reason that RBs and WRs will be compared to each other instead of split is because in many fantasy leagues the owner can pick flex players to fill positions. A flex position is any RB, WR, or TE.

```
#First train the four subsets of data. Set minsplit and max depth to reasonable values in order to be reader f

train_treeQB2016 <- rpart(OverUnder18 ~ ., data = trainQB[,c(4,5,11,17,26)],
                      method = "class", control=rpart.control(cp = 0, minsplit = 3, maxdepth = 4))
train_treeRBWR2016 <- rpart(OverUnder10 ~ ., data = trainRBWR[,c(4,5,9,22,23)],
                      method = "class", control=rpart.control(cp = 0, minsplit = 3, maxdepth = 4))
train_treeQB10s <- rpart(OverUnder18 ~ ., data = trainQB10s[,c(4,5,11,17,26)],
                      method = "class", control=rpart.control(cp = 0, minsplit = 3, maxdepth = 4))
train_treeRBWR10s <- rpart(OverUnder10 ~ ., data = trainRBWR10s[,c(4,5,9,22,23)],
                      method = "class", control=rpart.control(cp = 0, minsplit = 3, maxdepth = 4))

rsq.rpart(train_treeQB2016)
```

```
##
## Classification tree:
## rpart(formula = OverUnder18 ~ ., data = trainQB[, c(4, 5, 11,
##     17, 26)], method = "class", control = rpart.control(cp = 0,
##     minsplit = 3, maxdepth = 4))
##
## Variables actually used in tree construction:
## [1] Passes.Attempted Sacks            Week
##
## Root node error: 136/322 = 0.42236
##
## n= 322
##
##          CP nsplit rel error xerror     xstd
## 1 0.0441176      0   1.00000 1.0000 0.065172
## 2 0.0147059      3   0.86029 1.0735 0.065685
## 3 0.0036765      6   0.81618 1.0588 0.065603
## 4 0.0000000      8   0.80882 1.0956 0.065788
```

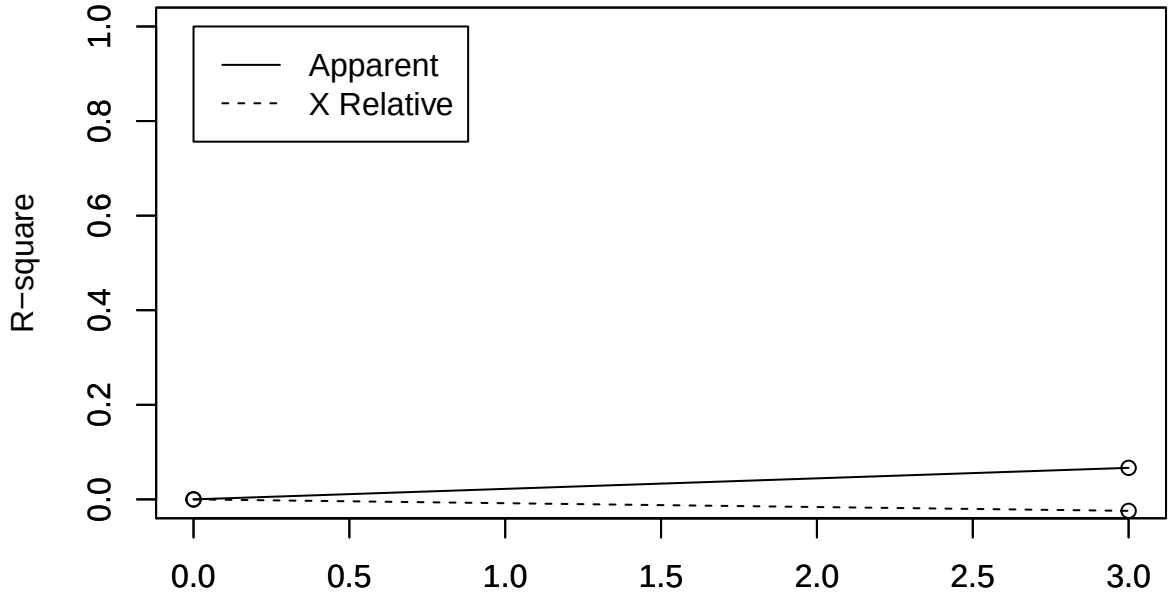


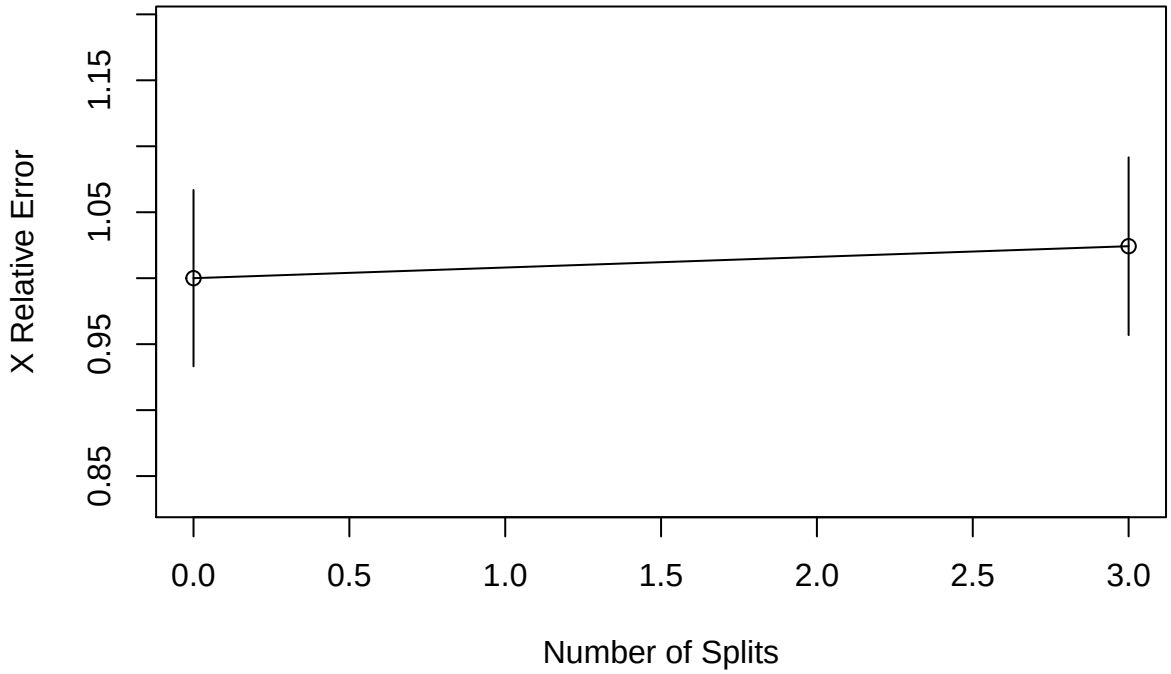



```
rsq.rpart(train_treeRBWR2016)
```

```
##
## Classification tree:
## rpart(formula = OverUnder10 ~ ., data = trainRBWR[, c(4, 5, 9,
##     22, 23)], method = "class", control = rpart.control(cp = 0,
##     minsplit = 3, maxdepth = 4))
##
## Variables actually used in tree construction:
## [1] Games.Started Position      Week
##
## Root node error: 165/628 = 0.26274
##
## n= 628
```

```
##
##         CP nsplit rel error xerror     xstd
## 1 0.022222      0   1.00000 1.0000 0.066845
## 2 0.000000      3   0.93333 1.0242 0.067357
```

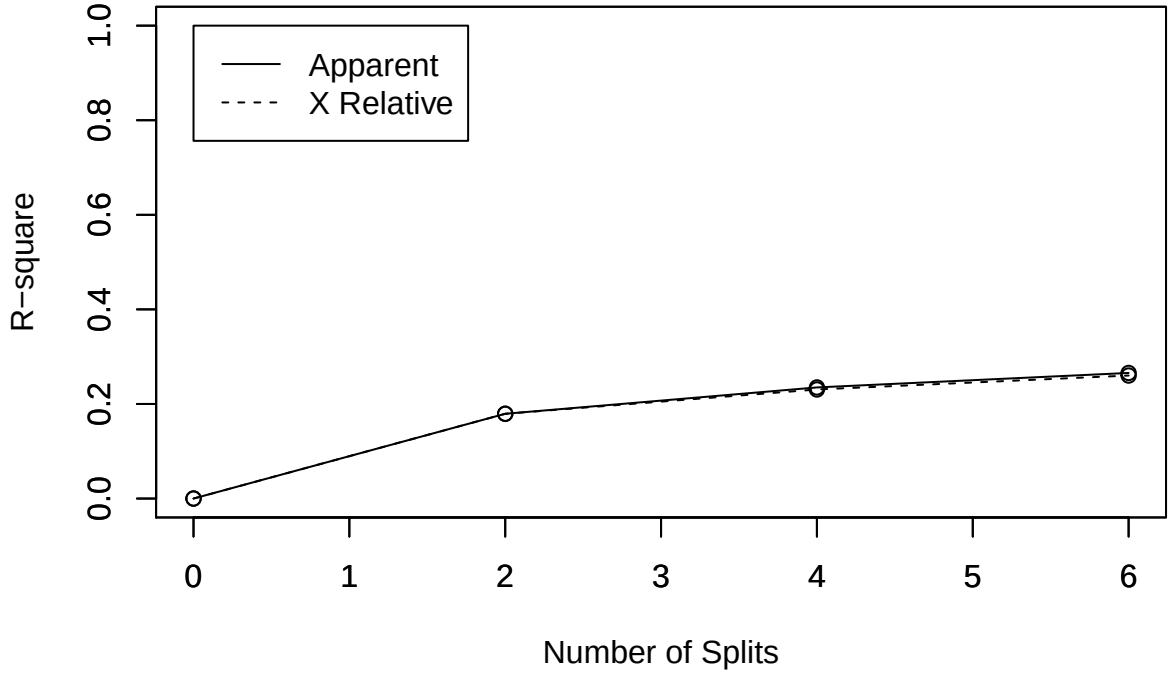


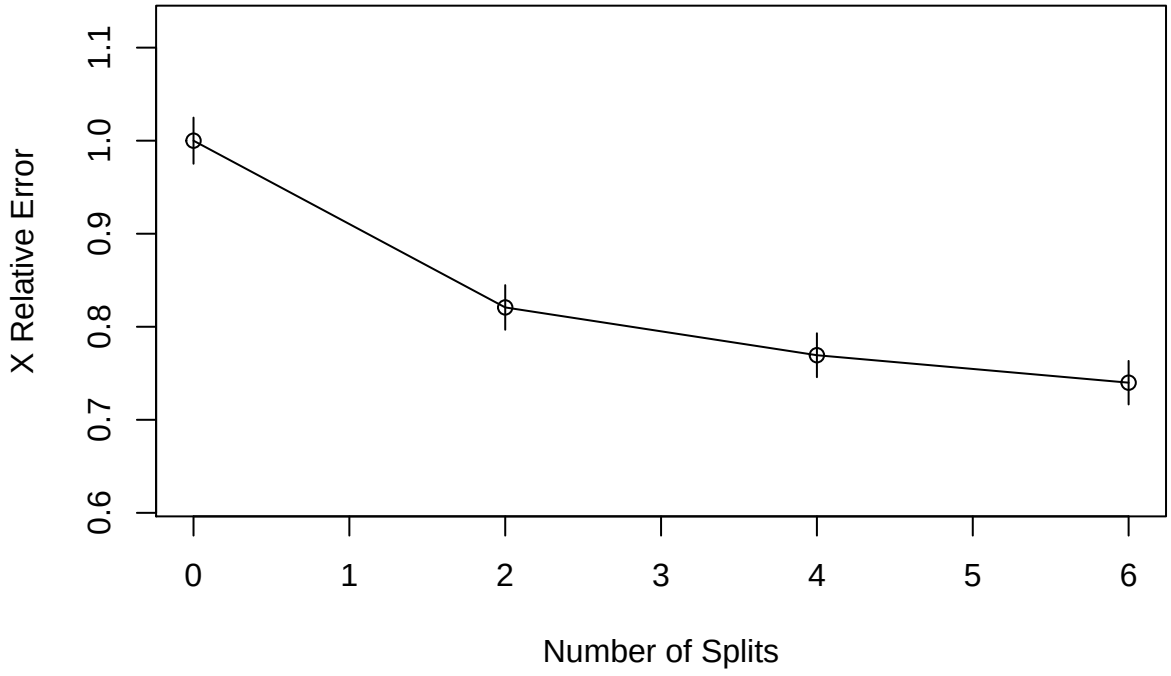



```
rsq.rpart(train_treeQB10s)
```

```
##
## Classification tree:
## rpart(formula = OverUnder18 ~ ., data = trainQB10s[, c(4, 5,
##     11, 17, 26)], method = "class", control = rpart.control(cp = 0,
##     minsplit = 3, maxdepth = 4))
##
## Variables actually used in tree construction:
```

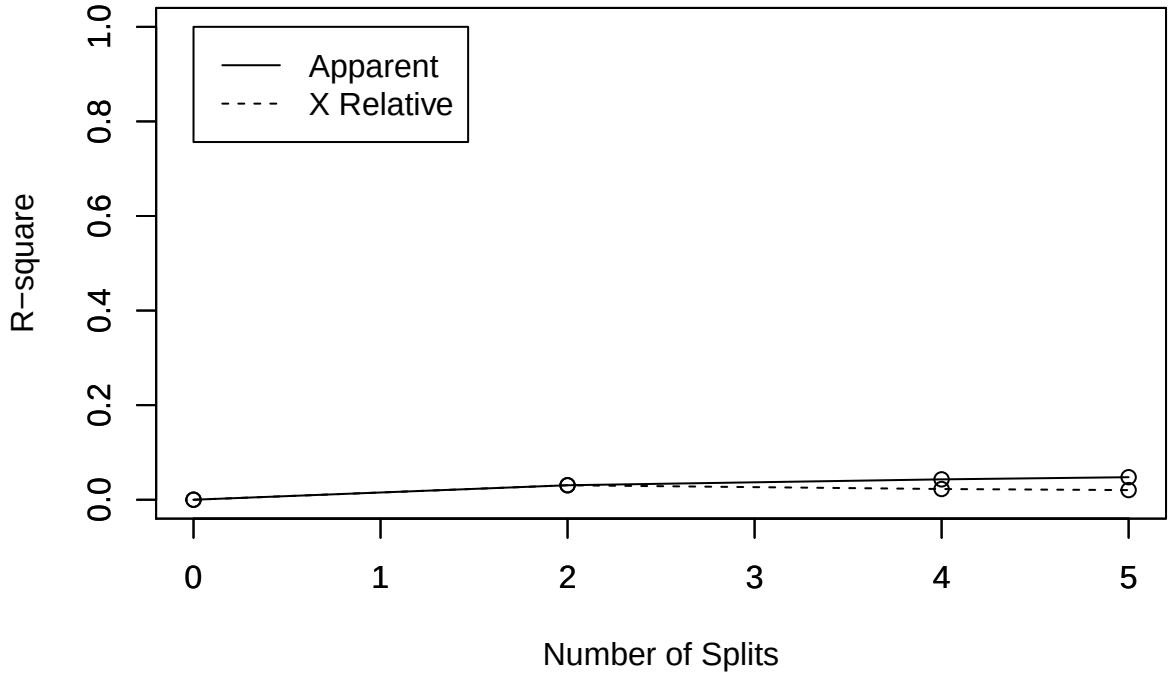
```
## [1] Passes.Attempted Sacks
##
## Root node error: 915/2090 = 0.4378
##
## n= 2090
##
##         CP nsplit rel error  xerror     xstd
## 1 0.089617      0   1.00000 1.00000 0.024788
## 2 0.027869      2   0.82077 0.82077 0.023973
## 3 0.015301      4   0.76503 0.76940 0.023614
## 4 0.000000      6   0.73443 0.73989 0.023381
```

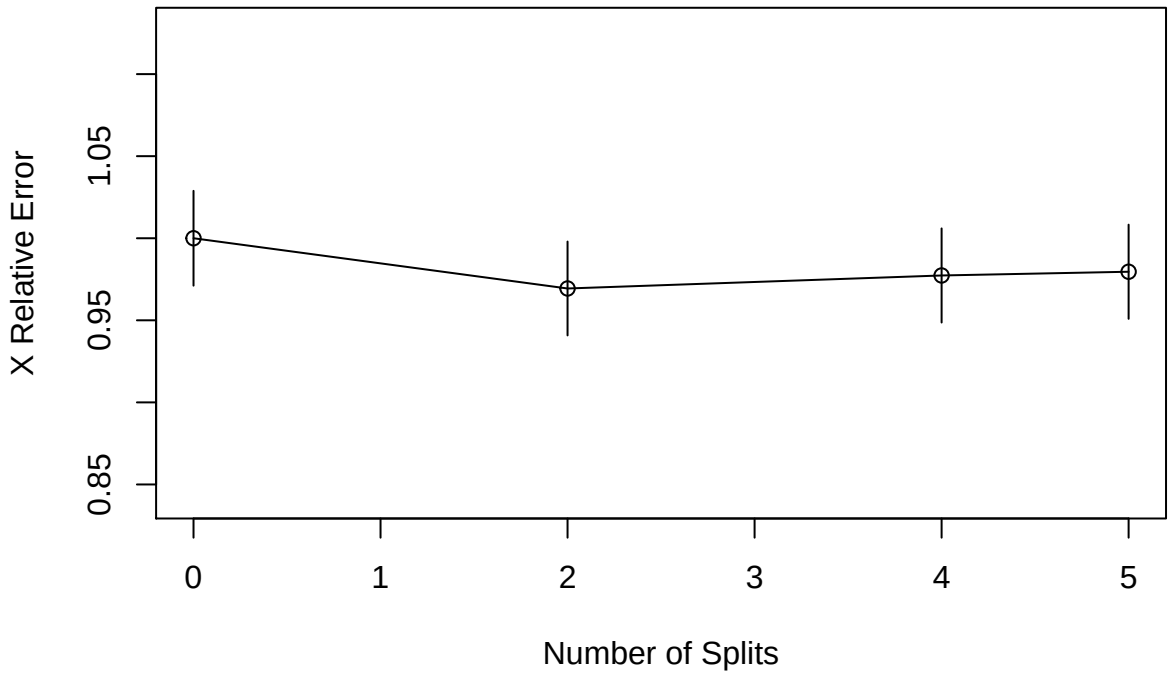




```
rsq.rpart(train_treeRBWR10s)
```

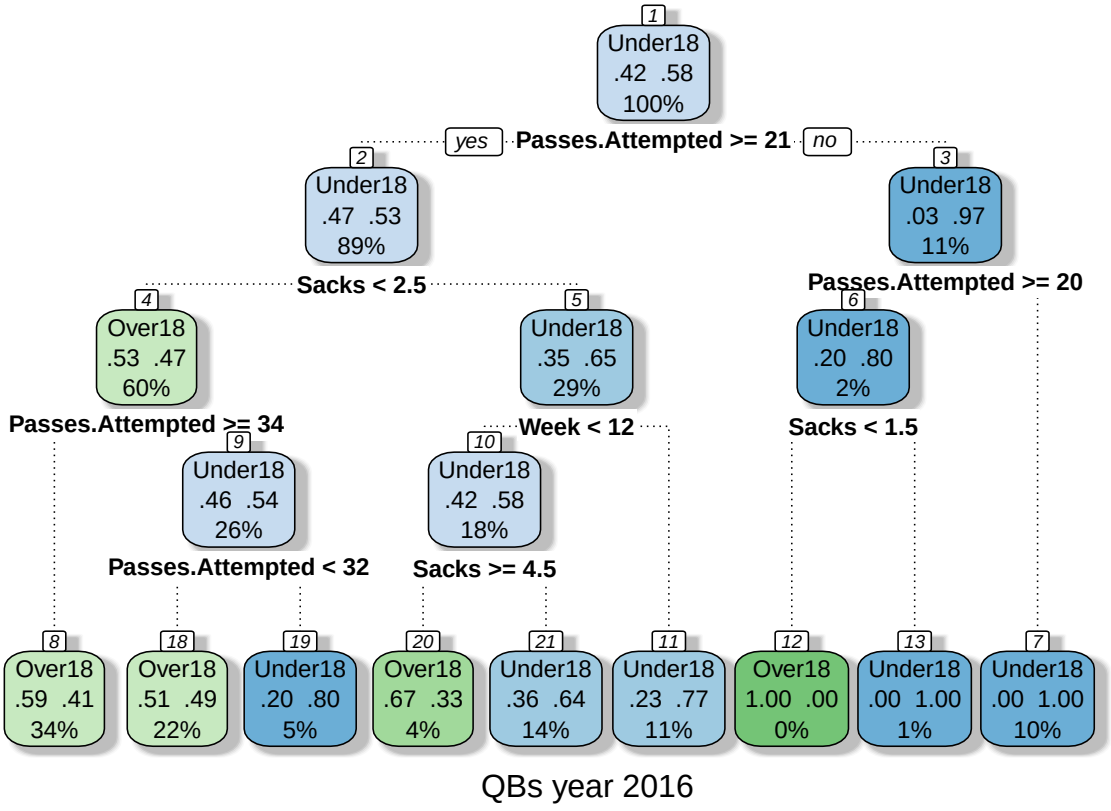
```
##
## Classification tree:
## rpart(formula = OverUnder10 ~ ., data = trainRBWR10s[, c(4, 5,
##     9, 22, 23)], method = "class", control = rpart.control(cp = 0,
##     minsplit = 3, maxdepth = 4))
##
## Variables actually used in tree construction:
## [1] Games.Started Home.or.Away  Position      Week
##
## Root node error: 882/3342 = 0.26391
##
## n= 3342
##
##          CP nsplit rel error  xerror     xstd
## 1 0.0153061      0   1.00000 1.00000 0.028889
## 2 0.0062358      2   0.96939 0.96939 0.028599
## 3 0.0045351      4   0.95692 0.97732 0.028675
## 4 0.0000000      5   0.95238 0.97959 0.028697
```



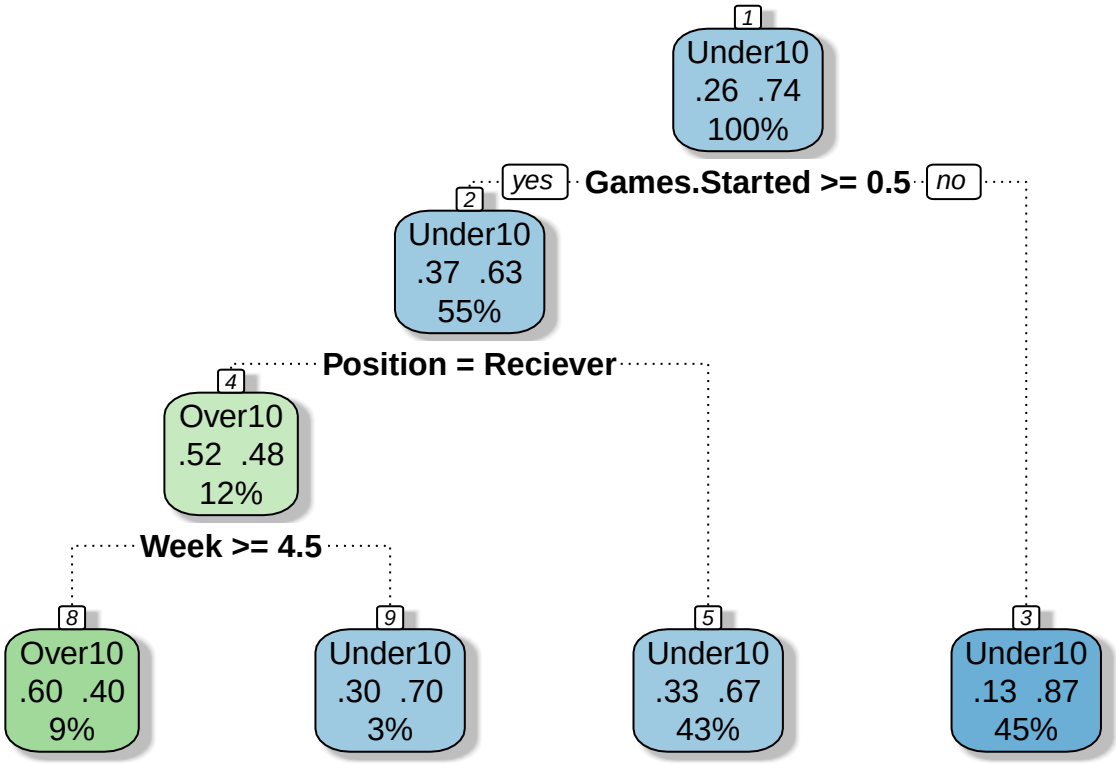


From the predicition models, we can see that the QB models are not very accurate. For both subsets of QB data, we had a root node error of about 45%. However, the RB/WR prediction models were fairly more accurate. Both of those models had about a 75% accuracy. Although not all models were super accurate, let's look at a visual representation to understand the role played by the variables mentioned above.

```
# Plotting decision trees
fancyRpartPlot(train_treeQB2016, caption = "QBs year 2016")
```

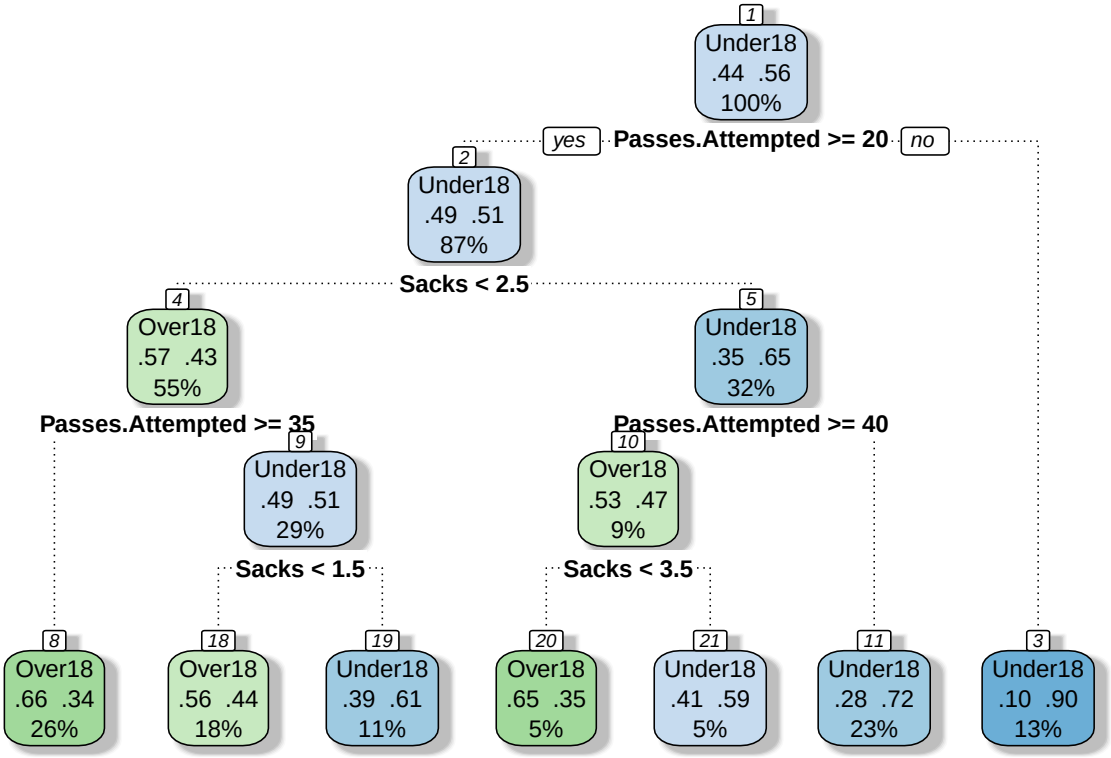


```
fancyRpartPlot(train_treeRBWR2016, caption = "RBs and WRs year 2016")
```



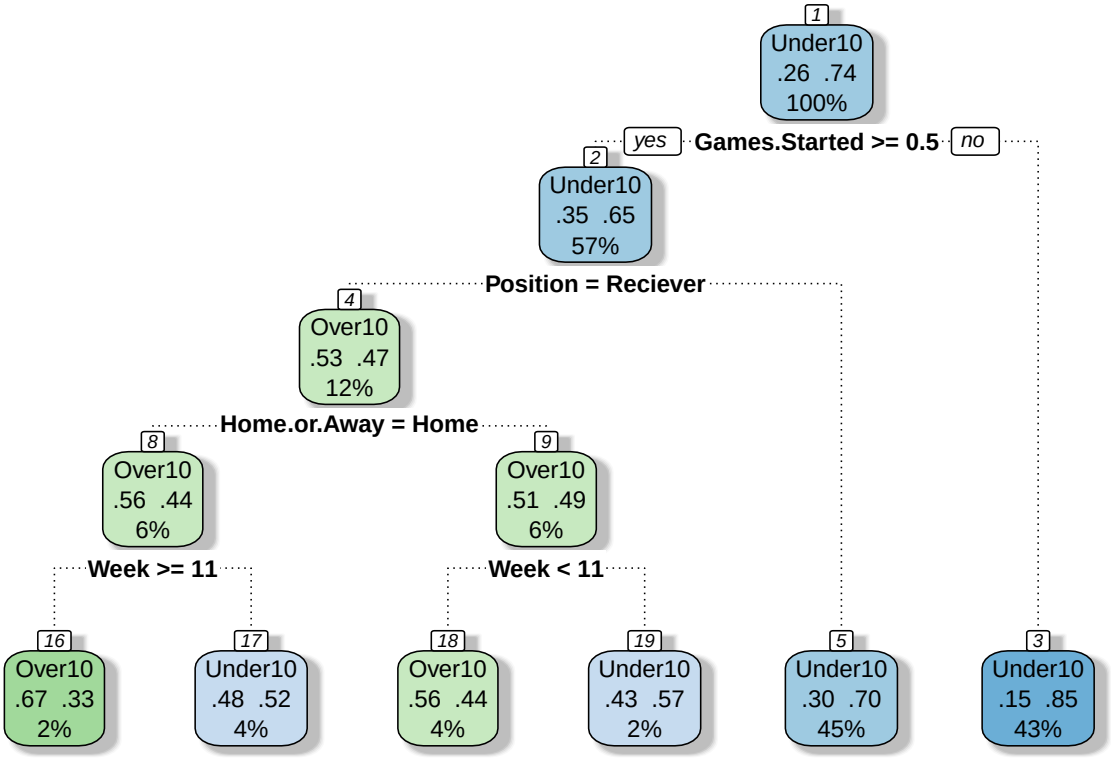
RBs and WRs year 2016

```
fancyRpartPlot(train_treeQB10s, caption = "QBs 2010s decade")
```



QBs 2010s decade

```
fancyRpartPlot(train_treeRBWR10s, caption = "RBs and WRs 2010s decade")
```



RBs and WRs 2010s decade

Looking at the 2016 plots first, we can see that there were more root nodes than in the decade plots. One hypothesis for this is due to the fact that there was not as much data, so it was harder for the algorithm to clean up the results in an accurate manner.

As noted before, the QB models were not very accurate, but there are a couple trivial things to note. In 2016 for a given game, if passes attempted was over 21, sacks was 2 or less, and it was week 8 or further in the season, then there is a 68% chance that a QB will have 18 or more fantasy points. 34% of the original data supported this. This makes sense due to the fact that more pass attempts will result in potential for more completions and TDs by a QB. And being sacked a minimal amount also supports higher fantasy performances because it means that the QB is not having as much pressure when he is trying to make a pass. The 2010 decade results are similar and thus not very interesting. So let's look at the more interesting results; RB/WR decision trees. These models were fairly accurate, as previously mentioned, and they uncover a couple interesting pieces of information. In 2016, for a given game, if a player started the game and was not a receiver (ie was a running back), then there is a 71% percent chance that the running back will score 10 or more fantasy points. 43% of the original data supported this. Also, if the player did not start, but did have at least one play on the ball and was a running back, then there is a 90% chance that the running back will score 10 or more fantasy points. 35% of the original data supported this. Finally, looking at the last plot, in the 2010 decade, for a given game, if a player was a running back (and had more than one rushing attempt), then nothing else is needed to be able to make a decision about fantasy points. There is an 80% chance that the running back will score 10 or more fantasy points. This highlights the fact that running backs are better performers for fantasy teams throughout the years 2010-2016. This is presumably because there will typically only be one star running back on a team. However, there may be multiple WR targets for the QB to disperse passes too (running back may even be included in these). Thus making it more common for RBs to individually influence a football game when compared to WRs.

Association Rule Mining

Association rule mining was applied to two of our game log datasets. Within the game log data, there is a game outcome that list if the game was won or lost. Therfore the righ-hand side of our rules have been set to Outcome = W in the hopes of seeing what other variables are most closely associated with winning games.

Association Rule Mining on QB

```
detach(package:tm, unload=TRUE)
#Subsetting the GL.QB dataset to only have the variables we want used in our rules
NominalQB <- subset(GL.QB, select = c(3,5,7,10:11,13,15:17,20,21,23))

#str(NominalQB)
#summary(NominalQB)

#Binning or factorizing the variables
NominalQB$Passes.Completed <- cut(NominalQB$Passes.Completed, breaks = c(-Inf,0,10,20,30,Inf),
                 labels=c("Zero_CompletePasses","1-10_CompletePasses","11-20_CompletePasses",
                                     "21-30_CompletePasse", "31+_CompletePasses"))

NominalQB$Passes.Attempted <- cut(NominalQB$Passes.Attempted, breaks = c(-Inf,0,10,20,40,60,Inf),
              labels=c("Zero_AttemptedPasses","1-10_AttemptedPasses","11-20_AttemptedPasse",
              "21-40_AttemptedPasses","41-60_AttemptedPasses","61+_AttemptedPasses"))

NominalQB$Passing.Yards <- cut(NominalQB$Passing.Yards, breaks = c(-Inf,0,100,200,300,400,500,Inf),
             labels=c("Negative-Zero_PassingYards","1-100_PassingYards","101-200_PassingYards",
                "201-300_PassingYards","301-400_PassingYards",
                     "401-500_PassingYards","500+_PassingYards"))

NominalQB$Rushing.Attempts <- cut(NominalQB$Rushing.Attempts, breaks = c(-Inf,0,10,20,Inf),
                                  labels=c("Zero_RushingAttempts","1-10_RushingAttempts",
                                           "11-20_RushingAttempts","21+_RushingAttempts"))

NominalQB$Rushing.Yards <- cut(NominalQB$Rushing.Yards, breaks = c(-Inf,-1,0,20,40,60,80,100,150,Inf),
                labels=c("Negative_RushingYards","Zero_RushingYards","1-20_RushingYards",
                 "21-40_RushingYards","31-60_RushingYards","61-80_RushingYards",
                 "81-100_RushingYards","101-150_RushingYards","150+_RushingYards"))

NominalQB$TD.Passes <- as.factor(NominalQB$TD.Passes)
NominalQB$Ints <- as.factor(NominalQB$Ints)
NominalQB$Sacks <- as.factor(NominalQB$Sacks)
NominalQB$Rushing.TDs <- as.factor(NominalQB$Rushing.TDs)

#Running the apriori algorithm on our factorized dataset. The righ-hand side was set to Outcome=W
#or outcome of the game is equal to a winning. Minimum support and confidence were set and
#the minimum length of variables was set to 3.
WinningGames.QB <- apriori(data=NominalQB,parameter = list(supp=.09, conf=.75, minlen=3),
                      appearance = list(default="lhs", rhs=c("Outcome=W")),
                     control=list(verbose=FALSE))

#Inspecting our rules by sorting them by either confidence, life and support.
QB.Rules.Conf<-sort(WinningGames.QB, by="confidence", decreasing=TRUE)
inspect(QB.Rules.Conf[1:5])
```

```
##     lhs                                       rhs            support confidence     lift count
## [1] {Rushing.Attempts=1-10_RushingAttempts,
##      Rushing.Yards=Negative_RushingYards}   => {Outcome=W} 0.09744638  0.8022923 1.606611  2240
## [2] {Rushing.Yards=Negative_RushingYards,
##      Rushing.TDs=0}                         => {Outcome=W} 0.09579327  0.8007273 1.603477  2202
## [3] {Rushing.Attempts=1-10_RushingAttempts,
##      Rushing.Yards=Negative_RushingYards,
##      Rushing.TDs=0}                         => {Outcome=W} 0.09579327  0.8007273 1.603477  2202
## [4] {Season=Regular Season,
```

```
##      Home.or.Away=Home,
##      Ints=0,
##      Rushing.Attempts=1-10_RushingAttempts} => {Outcome=W} 0.09844695  0.7671186 1.536175  2263
## [5] {Home.or.Away=Home,
##      Ints=0,
##      Rushing.Attempts=1-10_RushingAttempts} => {Outcome=W} 0.12093792  0.7525717 1.507045  2780
```

```
QB.Rules.lift<-sort(WinningGames.QB, by="lift", decreasing=TRUE)
inspect(QB.Rules.lift[1:5])
```

```
##     lhs                                      rhs            support confidence     lift count
## [1] {Rushing.Attempts=1-10_RushingAttempts,
##      Rushing.Yards=Negative_RushingYards}   => {Outcome=W} 0.09744638  0.8022923 1.606611  2240
## [2] {Rushing.Yards=Negative_RushingYards,
##      Rushing.TDs=0}                         => {Outcome=W} 0.09579327  0.8007273 1.603477  2202
## [3] {Rushing.Attempts=1-10_RushingAttempts,
##      Rushing.Yards=Negative_RushingYards,
##      Rushing.TDs=0}                         => {Outcome=W} 0.09579327  0.8007273 1.603477  2202
## [4] {Season=Regular Season,
##      Home.or.Away=Home,
##      Ints=0,
##      Rushing.Attempts=1-10_RushingAttempts} => {Outcome=W} 0.09844695  0.7671186 1.536175  2263
## [5] {Home.or.Away=Home,
##      Ints=0,
##      Rushing.Attempts=1-10_RushingAttempts} => {Outcome=W} 0.12093792  0.7525717 1.507045  2780
```

```
QB.Rules.support<-sort(WinningGames.QB, by="support", decreasing=TRUE)
inspect(QB.Rules.support[1:5])
```

```
##     lhs                                      rhs            support confidence     lift count
## [1] {Home.or.Away=Home,
##      Ints=0,
##      Rushing.Attempts=1-10_RushingAttempts} => {Outcome=W} 0.12093792  0.7525717 1.507045  2780
## [2] {Season=Regular Season,
##      Home.or.Away=Home,
##      Ints=0,
##      Rushing.Attempts=1-10_RushingAttempts} => {Outcome=W} 0.09844695  0.7671186 1.536175  2263
## [3] {Rushing.Attempts=1-10_RushingAttempts,
##      Rushing.Yards=Negative_RushingYards}   => {Outcome=W} 0.09744638  0.8022923 1.606611  2240
## [4] {Rushing.Yards=Negative_RushingYards,
##      Rushing.TDs=0}                         => {Outcome=W} 0.09579327  0.8007273 1.603477  2202
## [5] {Rushing.Attempts=1-10_RushingAttempts,
##      Rushing.Yards=Negative_RushingYards,
##      Rushing.TDs=0}                         => {Outcome=W} 0.09579327  0.8007273 1.603477  2202
```

Above is the top 5 results of our association rule mining on the QB data and it resulted in really good confidence and lift well over 1. What was interesting in the results is the lack of variability in the left-hand side rules. Rushing.Attmepts between 1-10 appear in 4 out of the 5 top rules. Home games are also a strong variable as well as Negative Rushing yards.

Association Rule Mining on RB.WR

```
Nominal_RB.WR <- subset(GL.RB.WR, select = c(3,5,7,10:11,13:16,18:19))

#str(NominalQB)
#summary(NominalQB)

Nominal_RB.WR$Receptions <- cut(Nominal_RB.WR$Receptions, breaks = c(-Inf,0,10,20,Inf),
           labels=c("Zero_Receptions","1-10_Receptions","11-20_Receptions","20+_Receptions"))
```

```
Nominal_RB.WR$Receiving.Yards <- cut(Nominal_RB.WR$Receiving.Yards,
                                     breaks = c(-Inf,-1,50,100,150,200,250,300,Inf),
           labels=c("Negative_RecievingYards", "Zero-50_RecievingYards",
                "51-100_RecievingYards","101-150_RecievingYards", "151-200_RecievingYards",
                 "201-250_RecievingYards","251-300_RecievingYards","300+_RecievingYards"))

Nominal_RB.WR$Longest.Reception <- cut(Nominal_RB.WR$Longest.Reception, breaks = c(-Inf,-1,30,60,Inf),
                 labels=c("Negative_LongestReception","Zero-30_LongestReceptions",
                  "31-60_LongestReceptions","61+_LongestReceptions"))

Nominal_RB.WR$Receiving.TDs <- as.factor(Nominal_RB.WR$Receiving.TDs)

Nominal_RB.WR$Rushing.Attempts <- cut(Nominal_RB.WR$Rushing.Attempts, breaks = c(-1,25,45),
                                labels=c("Zero-25_RushingAttempts","21-45_RushingAttempts"))

Nominal_RB.WR$Rushing.Yards <- cut(Nominal_RB.WR$Rushing.Yards, breaks = c(-Inf,-1,50,100,150,200,250,Inf),
                labels=c("Negative_RushingYards","Zero-50_RushingYards","51-100_RushingYards",
                 "101-150_RushingYards","151-200_RushingYards","201-250_RushingYards"
                 ,"251+_RushingYards"))

Nominal_RB.WR$Longest.Rushing.Run <- cut(Nominal_RB.WR$Longest.Rushing.Run,
                                         breaks = c(-Inf,-1,20,40,60,80,Inf),
                labels=c("Negative_LongestRushingRun","Zero-20_LongestRushingRun",
                    "21-40_LongestRushingRun","41-60_LongestRushingRun",
                     "61-80_LongestRushingRun","81+_LongestRushingRun"))

Nominal_RB.WR$Rushing.TDs <- as.factor(Nominal_RB.WR$Rushing.TDs)

WinningGames.RBWR <- apriori(data=Nominal_RB.WR,parameter = list(supp=.01, conf=.65, minlen=3),
                      appearance = list(default="lhs", rhs=c("Outcome=W")),
                     control=list(verbose=FALSE))

RB.WR.Rules.Conf<-sort(WinningGames.RBWR, by="confidence", decreasing=TRUE)
inspect(RB.WR.Rules.Conf[1:5])
```

```
##     lhs                                              rhs            support confidence     lift count
## [1] {Longest.Reception=Zero-30_LongestReceptions,
##      Rushing.Yards=101-150_RushingYards}          => {Outcome=W} 0.01039412  0.7268314 1.442023  1647
## [2] {Season=Regular Season,
##      Rushing.Yards=101-150_RushingYards}          => {Outcome=W} 0.01046354  0.7227550 1.433936  1658
## [3] {Home.or.Away=Home,
##      Receiving.Yards=Zero-50_RecievingYards,
##      Rushing.TDs=1}                               => {Outcome=W} 0.01545549  0.7205060 1.429474  2449
## [4] {Receiving.TDs=0,
##      Rushing.Yards=101-150_RushingYards}          => {Outcome=W} 0.01012906  0.7194083 1.427296  1605
## [5] {Home.or.Away=Home,
##      Receiving.Yards=Zero-50_RecievingYards,
##      Longest.Reception=Zero-30_LongestReceptions,
##      Rushing.TDs=1}                               => {Outcome=W} 0.01516519  0.7190305 1.426546  2403
```

```
RB.WR.Rules.lift<-sort(WinningGames.RBWR, by="lift", decreasing=TRUE)
inspect(RB.WR.Rules.lift[1:5])
```

```
##     lhs                                              rhs            support confidence     lift count
## [1] {Longest.Reception=Zero-30_LongestReceptions,
##      Rushing.Yards=101-150_RushingYards}          => {Outcome=W} 0.01039412  0.7268314 1.442023  1647
## [2] {Season=Regular Season,
```

```
##      Rushing.Yards=101-150_RushingYards}          => {Outcome=W} 0.01046354  0.7227550 1.433936  1658
## [3] {Home.or.Away=Home,
##      Receiving.Yards=Zero-50_RecievingYards,
##      Rushing.TDs=1}                               => {Outcome=W} 0.01545549  0.7205060 1.429474  2449
## [4] {Receiving.TDs=0,
##      Rushing.Yards=101-150_RushingYards}          => {Outcome=W} 0.01012906  0.7194083 1.427296  1605
## [5] {Home.or.Away=Home,
##      Receiving.Yards=Zero-50_RecievingYards,
##      Longest.Reception=Zero-30_LongestReceptions,
##      Rushing.TDs=1}                               => {Outcome=W} 0.01516519  0.7190305 1.426546  2403
```

```
RB.WR.Rules.support<-sort(WinningGames.RBWR, by="support", decreasing=TRUE)
inspect(RB.WR.Rules.support[1:5])
```

```
##     lhs                                                rhs            support confidence     lift count
## [1] {Home.or.Away=Home,
##      Receiving.TDs=1}                                => {Outcome=W} 0.03021047  0.6556636 1.300827  4787
## [2] {Home.or.Away=Home,
##      Receiving.TDs=1,
##      Rushing.Attempts=Zero-25_RushingAttempts}       => {Outcome=W} 0.02993279  0.6540265 1.297579  4743
## [3] {Home.or.Away=Home,
##      Receptions=1-10_Receptions,
##      Receiving.TDs=1}                                => {Outcome=W} 0.02992648  0.6567867 1.303056  4742
## [4] {Home.or.Away=Home,
##      Receptions=1-10_Receptions,
##      Receiving.TDs=1,
##      Rushing.Attempts=Zero-25_RushingAttempts}       => {Outcome=W} 0.02964880  0.6551388 1.299786  4698
## [5] {Home.or.Away=Home,
##      Receiving.TDs=1,
##      Longest.Rushing.Run=Zero-20_LongestRushingRun} => {Outcome=W} 0.02889779  0.6512587 1.292088  4579
```

Above is the top 5 results of our association rule mining on the RB.WR data and it resulted in really good confidence and lift well over 1. Unlike with the QB dataset, we got much more variability of variables on the left-hand side. Home games still are a popular variable, as they appeared in the top rules of both of our datasets. When the rules are sorted by "Support", Receiving TDs = 1 appear in all of the top 5 rules, this leads us to believe that Receiving TDs = 1 is a very common variable and has a good association with winning games.

Naive Bayes

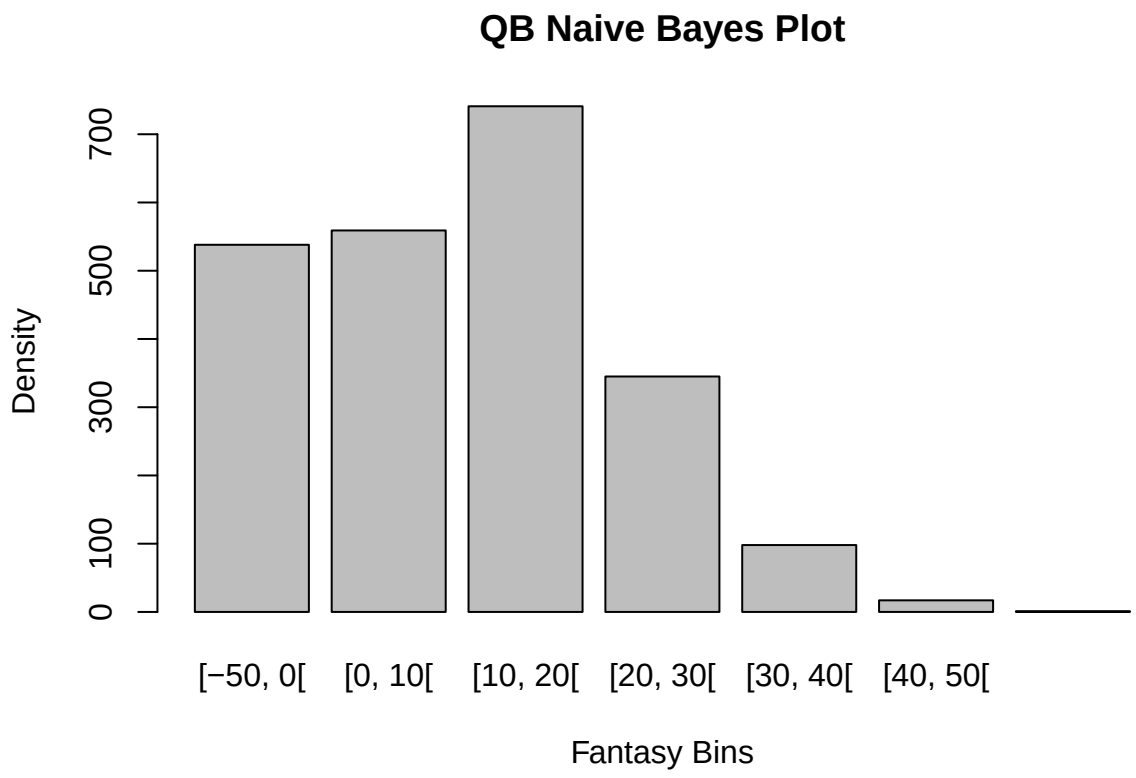
QB Naive Bayes

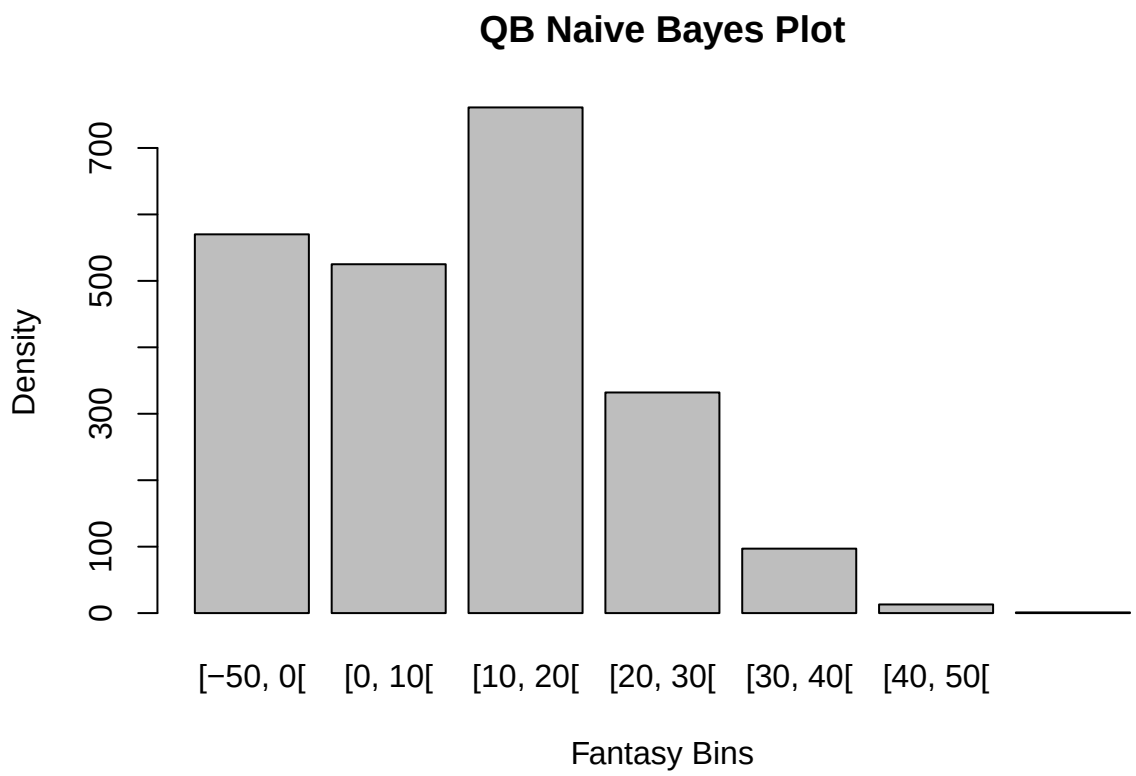
```
# Compute bins
GL.QB_Bin <- fastDiscretization(dataSet = GL.QB, list(Fantasy = c(-50, 0, 10, 20,30,40,50,Inf)))
```

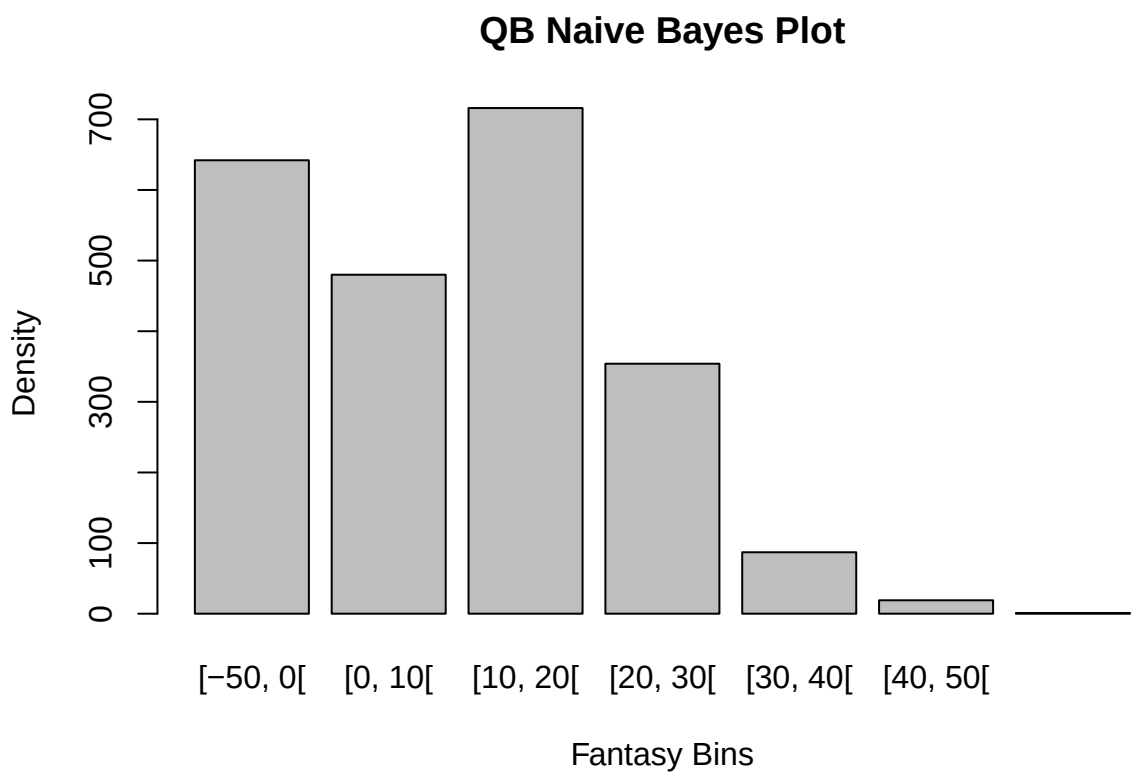
```
## [1] "fastDiscretization: I will discretize 1 numeric columns using, bins."
## [1] "fastDiscretization: it took me: 0.07s to transform 1 numeric columns into, binarised columns."
```

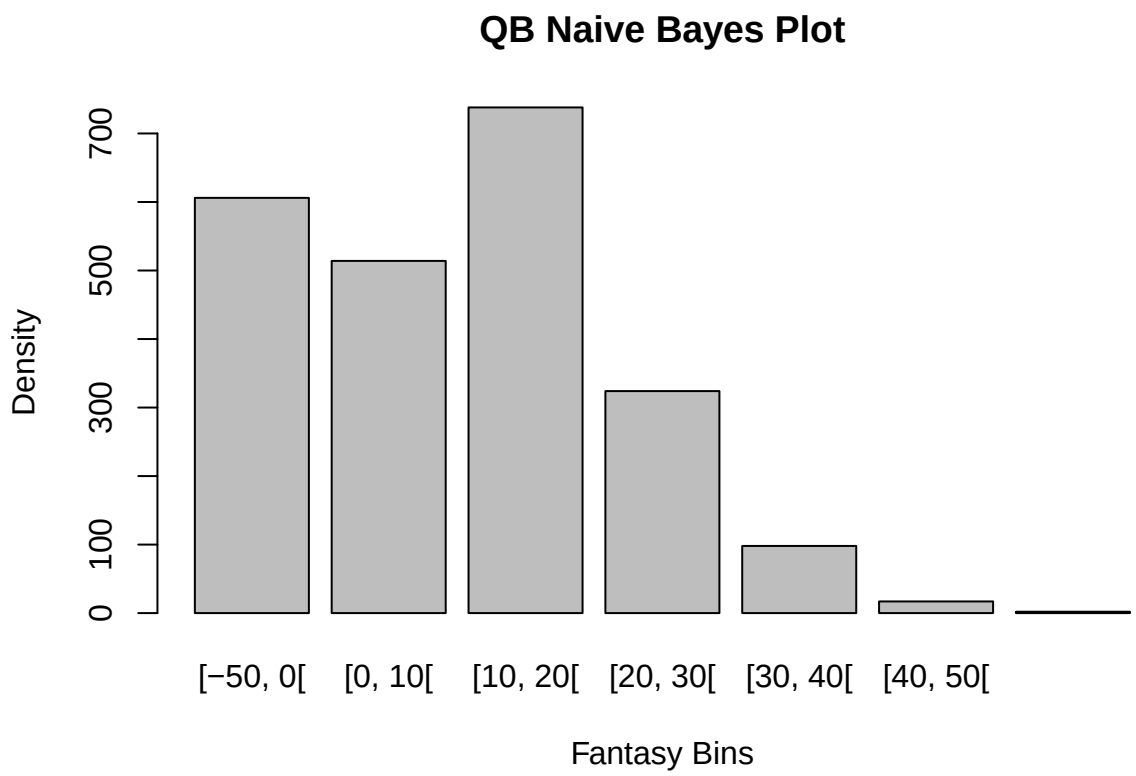
```
#Segmenting data into Test/Train for Classifier use
N <- nrow(GL.QB_Bin)
kfolds <- 10
holdout <- split(sample(1:N), 1:kfolds)
#head(holdout)

AllResults <- list()
AllLabels <- list()

for (k in 1:kfolds){

  GL.QB_Test <- GL.QB_Bin[holdout[[k]], ]
```

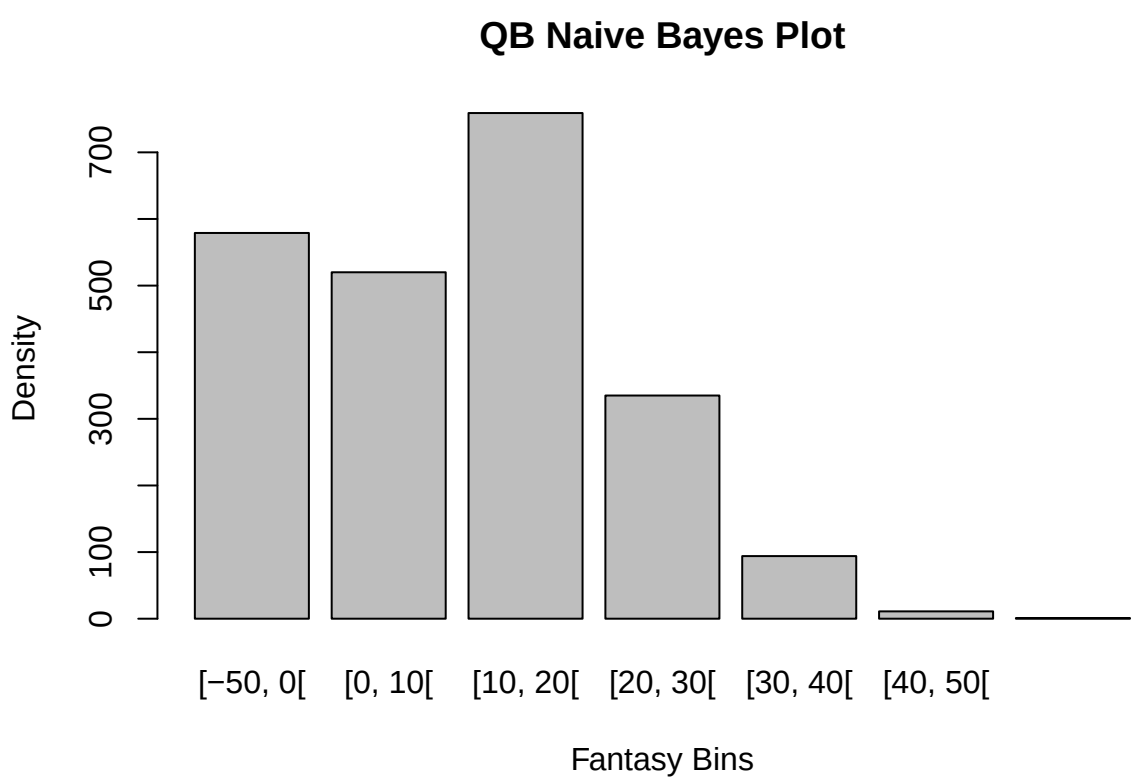
```
  GL.QB_Train = GL.QB_Bin[-holdout[[k]], ]

  (head(GL.QB_Train))
  GL.QB_Test$Fantasy
  (table(GL.QB_Test$Fantasy))

  (head(GL.QB_Test))
  GL.QB_Test_noFantasy<-GL.QB_Test[-c(1)]
  GL.QB_Test_justFantasy<-GL.QB_Test$Fantasy
  (head(GL.QB_Test_justFantasy))

  #Build naive Bayes model
  train_naibayes<-naiveBayes(Fantasy~., data=GL.QB_Train, na.action = na.pass)
  summary(train_naibayes)

  #Apply the model to predicting test data
  nb_Pred <- predict(train_naibayes, GL.QB_Test)

  #Creating the confusion matrix
  NaiveBayes1 <- (confusionMatrix(nb_Pred, GL.QB_Test$Fantasy))

  AllResults<- c(AllResults,nb_Pred)
  AllLabels<- c(AllLabels, GL.QB_Test_justFantasy)

  plot(nb_Pred, ylab = "Density", xlab = "Fantasy Bins" ,main = "QB Naive Bayes Plot")
}
```

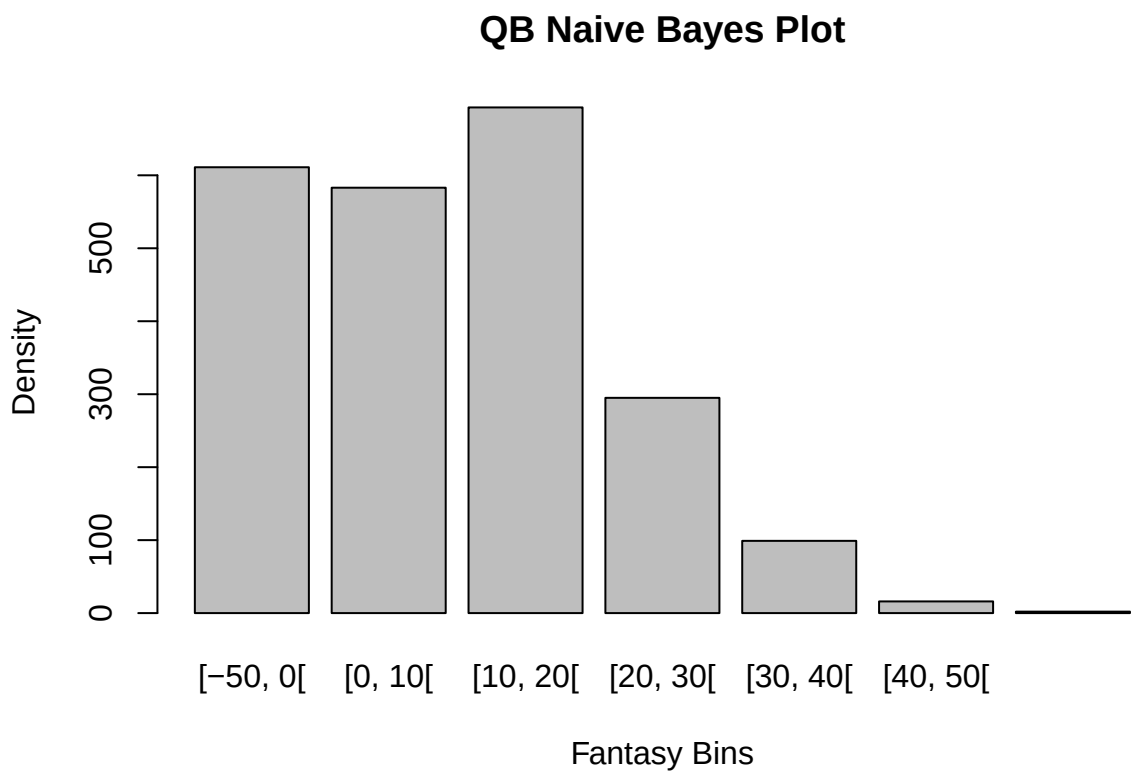


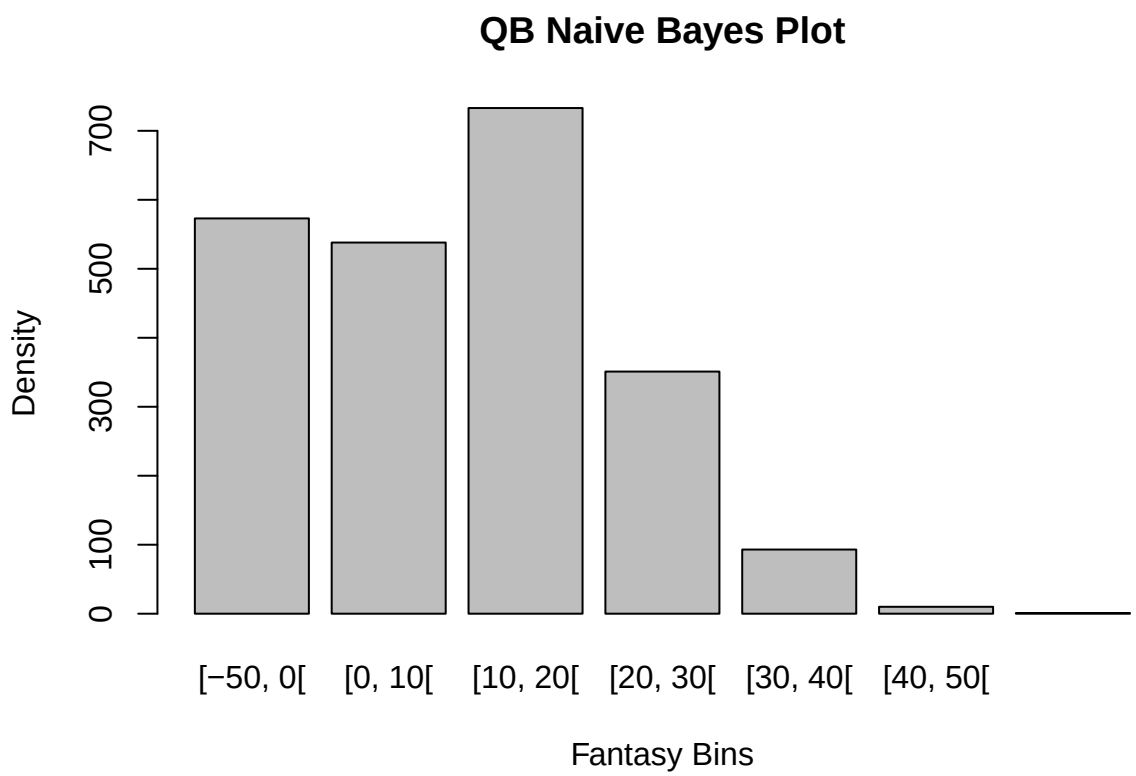

QB Naive Bayes Plot
Density
0
100
300
500
700
[−50, 0[
[0, 10[
[10, 20[
[20, 30[
[30, 40[
[40, 50[
Fantasy Bins

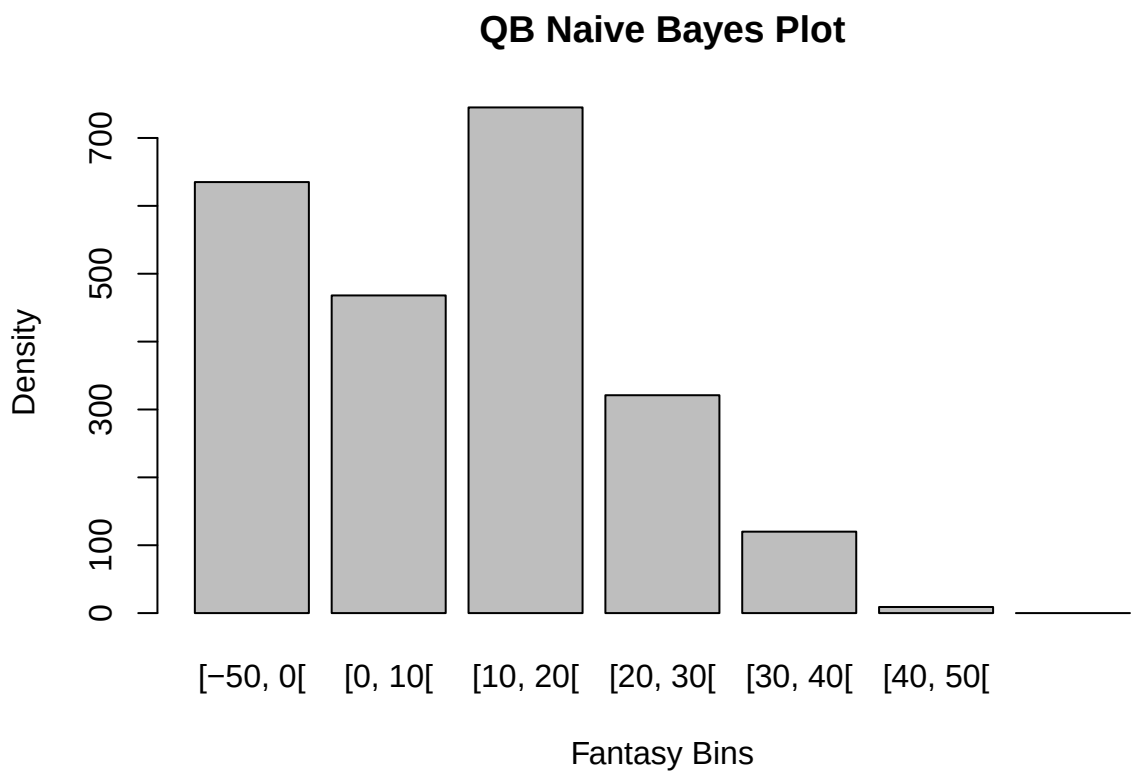

QB Naive Bayes Plot
Density
0
100
300
500
700
[−50, 0[
[0, 10[
[10, 20[
[20, 30[
[30, 40[
[40, 50[
Fantasy Bins

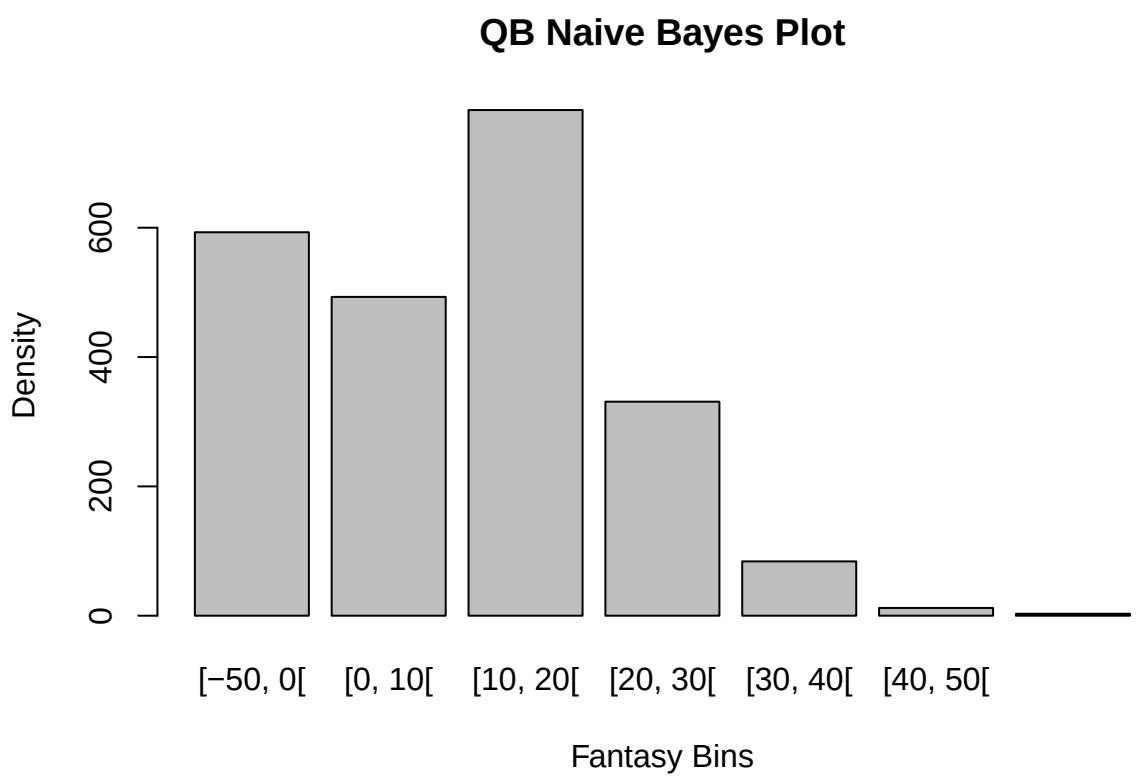

QB Naive Bayes Plot
Density
0
100
300
500
700
[−50, 0[
[0, 10[
[10, 20[
[20, 30[
[30, 40[
[40, 50[
Fantasy Bins

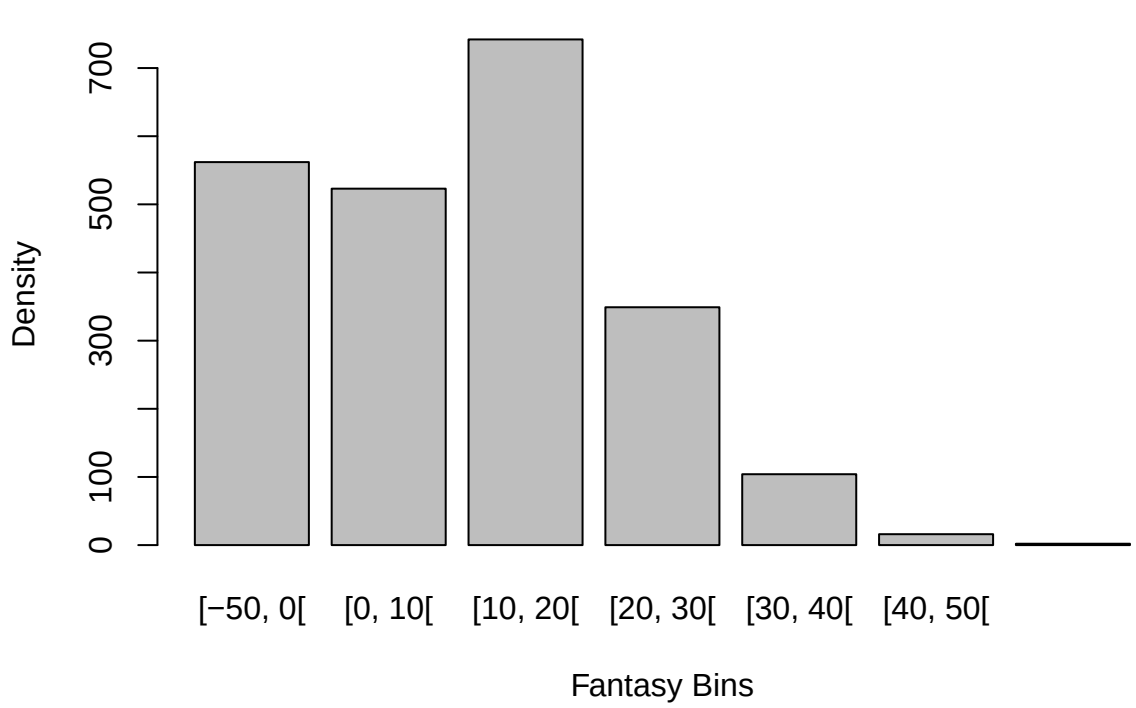

QB Naive Bayes Plot
Density
0
100
300
500
700
[−50, 0[
[0, 10[
[10, 20[
[20, 30[
[30, 40[
[40, 50[
Fantasy Bins


QB Naive Bayes Plot
Density
0
100
300
500
[−50, 0[
[0, 10[
[10, 20[
[20, 30[
[30, 40[
[40, 50[
Fantasy Bins


QB Naive Bayes Plot
Density
0
100
300
500
700
[−50, 0[
[0, 10[
[10, 20[
[20, 30[
[30, 40[
[40, 50[
Fantasy Bins


QB Naive Bayes Plot
Density
0
100
300
500
700
[−50, 0[
[0, 10[
[10, 20[
[20, 30[
[30, 40[
[40, 50[
Fantasy Bins


QB Naive Bayes Plot
Density
0
200
400
600
[−50, 0[
[0, 10[
[10, 20[
[20, 30[
[30, 40[
[40, 50[
Fantasy Bins




```
NaiveBayes1
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   [-50, 0[ [0, 10[ [10, 20[ [20, 30[ [30, 40[ [40, 50[
##   [-50, 0[        154     408        0        0        0        0
##   [0, 10[          12     503        8        0        0        0
##   [10, 20[          0     136      592       14        0        0
##   [20, 30[          0       0       44      301        4        0
##   [30, 40[          0       0        0       14       89        1
##   [40, 50[          0       0        0        0        3       13
##   [50, +Inf[        0       0        0        0        0        0
##             Reference
## Prediction   [50, +Inf[
##   [-50, 0[            0
##   [0, 10[             0
##   [10, 20[            0
##   [20, 30[            0
##   [30, 40[            0
##   [40, 50[            0
##   [50, +Inf[          2
##
## Overall Statistics
##
##                Accuracy : 0.7198
##                  95% CI : (0.7009, 0.738)
##     No Information Rate : 0.4556
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6334
##
##  Mcnemar's Test P-Value : NA
```

```
##
## Statistics by Class:
##
##                      Class: [-50, 0[ Class: [0, 10[ Class: [10, 20[
## Sensitivity                  0.92771         0.4804          0.9193
## Specificity                  0.80863         0.9840          0.9093
## Pos Pred Value               0.27402         0.9618          0.7978
## Neg Pred Value               0.99309         0.6935          0.9666
## Prevalence                   0.07224         0.4556          0.2802
## Detection Rate               0.06701         0.2189          0.2576
## Detection Prevalence         0.24456         0.2276          0.3229
## Balanced Accuracy            0.86817         0.7322          0.9143
##                      Class: [20, 30[ Class: [30, 40[ Class: [40, 50[
## Sensitivity                   0.9149         0.92708        0.928571
## Specificity                   0.9756         0.99319        0.998687
## Pos Pred Value                0.8625         0.85577        0.812500
## Neg Pred Value                0.9856         0.99681        0.999562
## Prevalence                    0.1432         0.04178        0.006092
## Detection Rate                0.1310         0.03873        0.005657
## Detection Prevalence          0.1519         0.04526        0.006963
## Balanced Accuracy             0.9453         0.96014        0.963629
##                      Class: [50, +Inf[
## Sensitivity                  1.0000000
## Specificity                  1.0000000
## Pos Pred Value               1.0000000
## Neg Pred Value               1.0000000
## Prevalence                   0.0008703
## Detection Rate               0.0008703
## Detection Prevalence         0.0008703
## Balanced Accuracy            1.0000000
```

Accuracy is 66.4% certainly not super high but decent. The p-value is significant. Across all 10 folds (and subsequent graphs) there were significant variations in the location of bin Fantasy Points. Ultimately the bin for Fantasy 10 points remained the highest and most dense of there ranges. Bin -50 was the second highest. Clearly the prediction that given the QB_Test set most QBs will be in the Fantasy point range of 0-10.

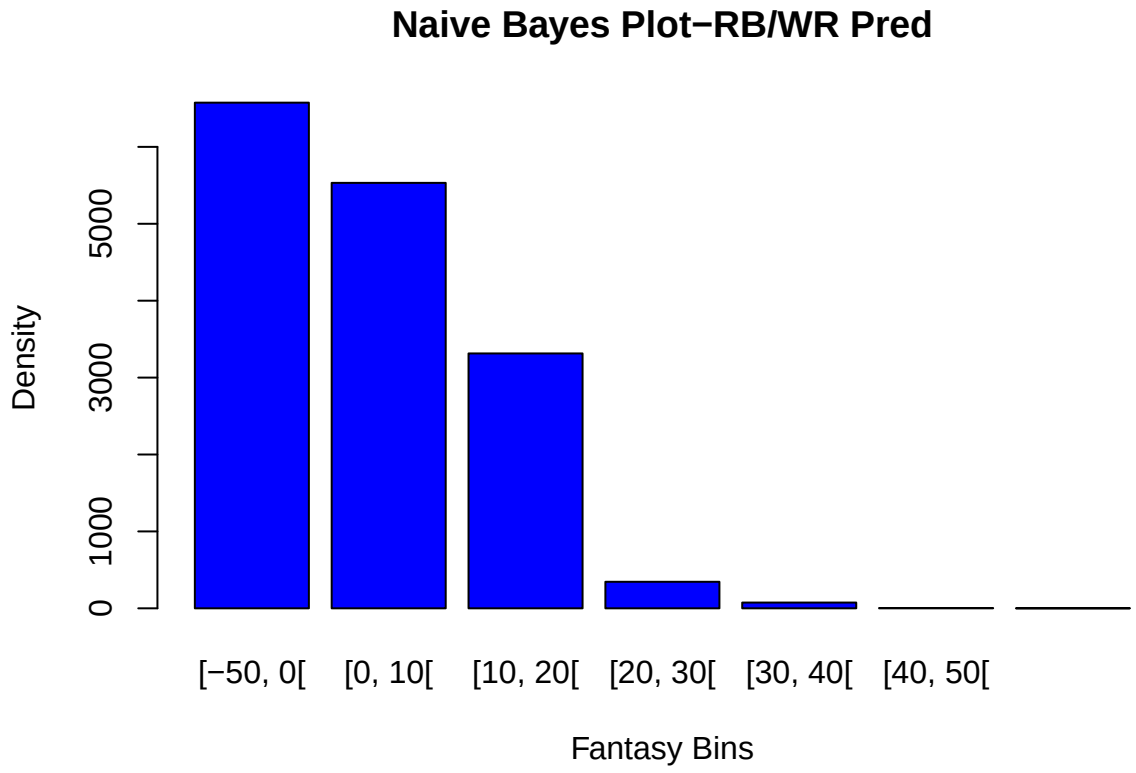
RB-WR Naive Bayes

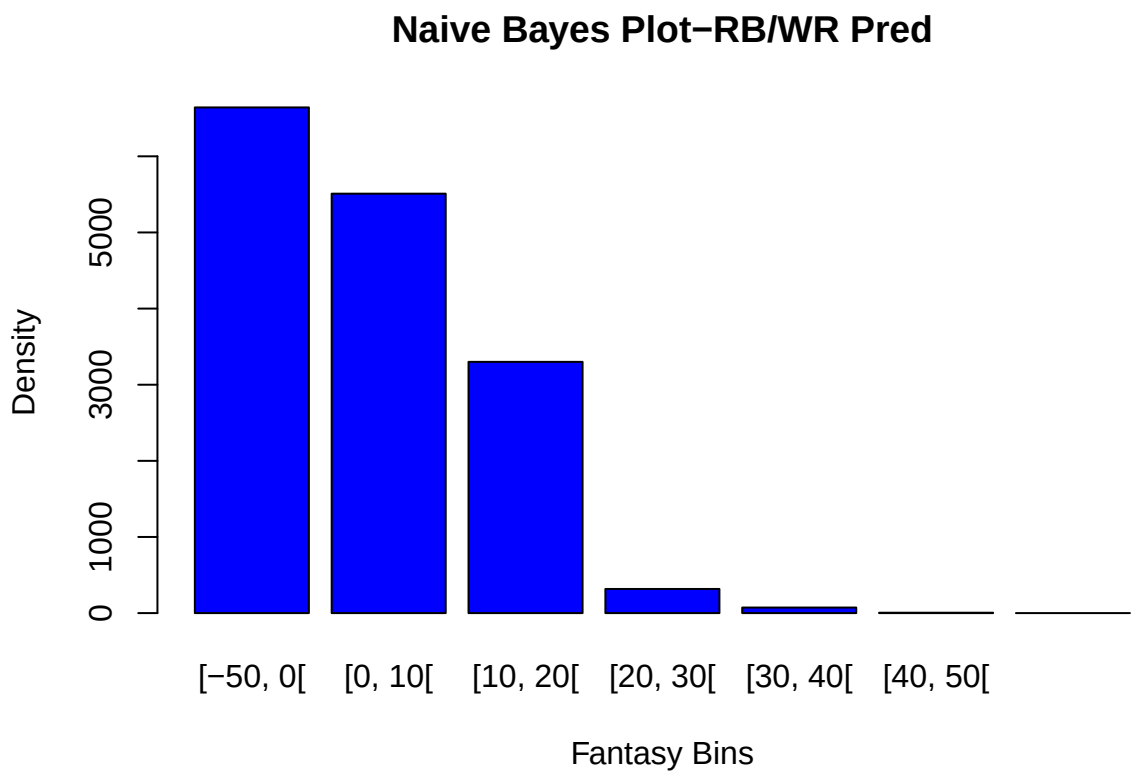
```
#Using our cleaned data from above
GL.RB.WR<- rbind(GL.WR.TE,GL.RB)
GL.RB.WR$Fantasy<- (GL.RB.WR$Receptions * .5) + (GL.RB.WR$Receiving.Yards * .1) +
  (GL.RB.WR$Receiving.TDs * 6) + (GL.QB$Rushing.Yards * .1) + (GL.RB.WR$Rushing.TDs * 6)
# Compute RB.WR bins - putting the numeric Fantasy calc field as factored bins
GL.RB.WR_Bin <- fastDiscretization(dataSet = GL.RB.WR, list(Fantasy = c(-50, 0, 10, 20,30,40,50,Inf)))
```

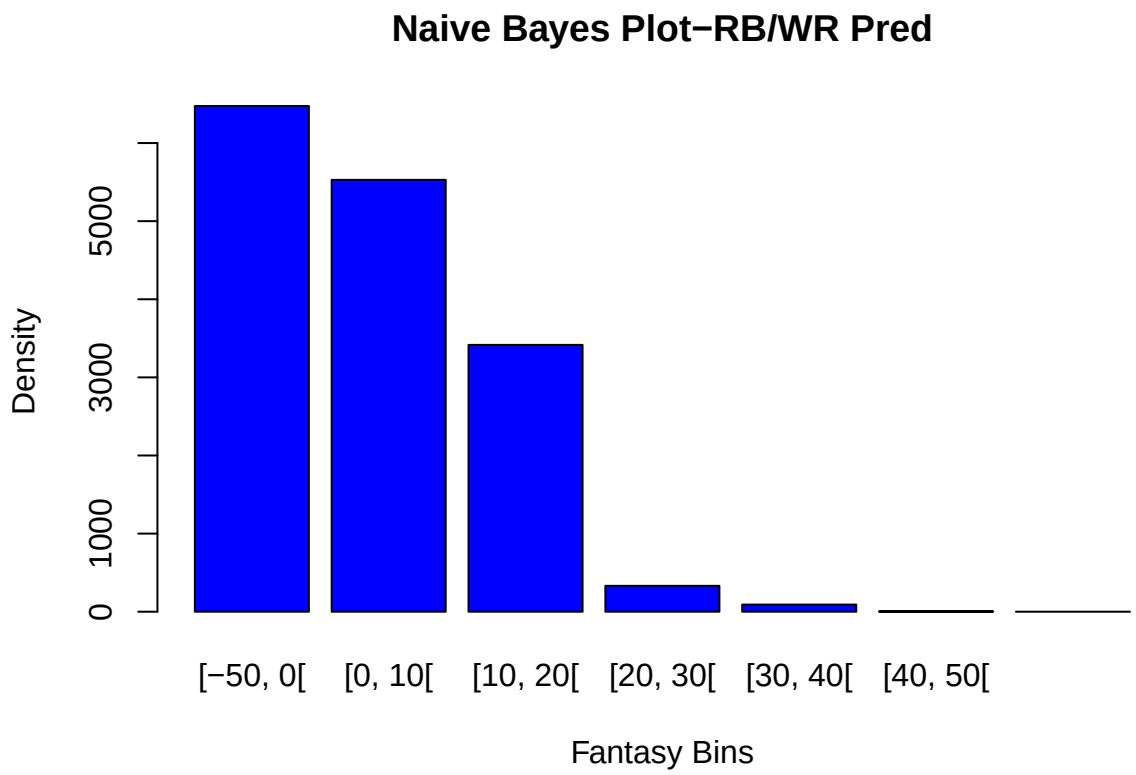
```
## [1] "fastDiscretization: I will discretize 1 numeric columns using, bins."
## [1] "fastDiscretization: it took me: 0.51s to transform 1 numeric columns into, binarised columns."
```

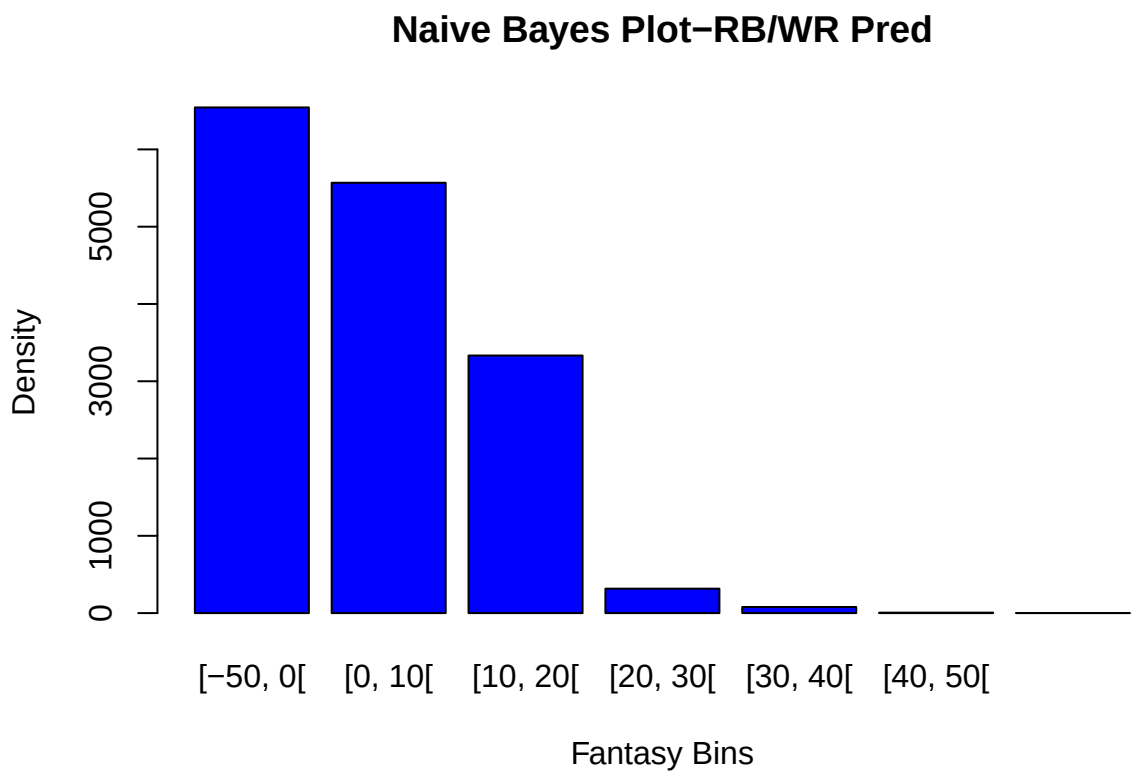
```
#Segmenting QB data into kfolds for Classifier use
N <- nrow(GL.RB.WR_Bin)
kfolds <- 10
holdout <- split(sample(1:N), 1:kfolds)
#head(holdout) #verifying the result

for (k in 1:kfolds){

  GL.RB.WR_Test <- GL.RB.WR_Bin[holdout[[k]], ]
  GL.RB.WR_Train = GL.RB.WR_Bin[-holdout[[k]], ]
```

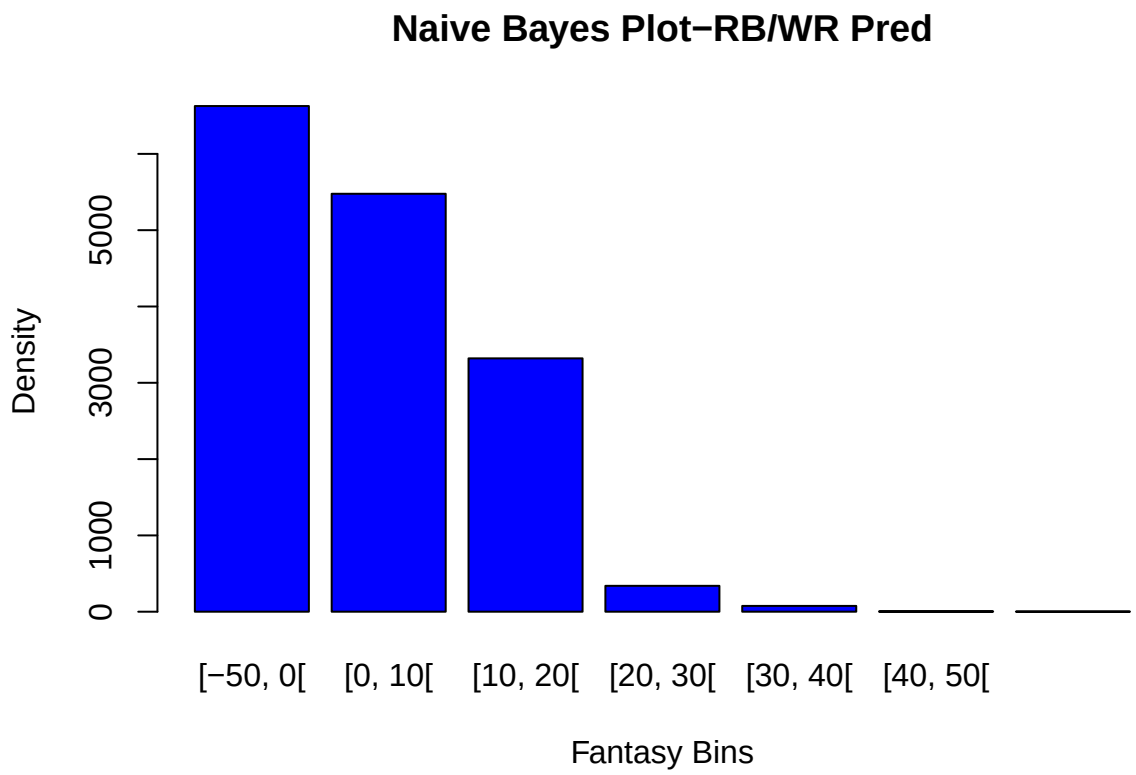
```
  (head(GL.RB.WR_Train))
  GL.RB.WR_Test$Fantasy
  (table(GL.RB.WR_Test$Fantasy))

  #Build Naive Bayes model
  train_naibayes1 <- naiveBayes(Fantasy~., data=GL.RB.WR_Train, na.action = na.pass)
  summary(train_naibayes1)

  #Apply the model to predicting test data
  nb_Pred1 <- predict(train_naibayes1, GL.RB.WR_Test)

  #Creating the confusion matrix
  NaiveBayes2 <- (confusionMatrix(nb_Pred1, GL.RB.WR_Test$Fantasy))

  #Plotting Naive Bayes
  par("mar")
  plot(nb_Pred1, ylab = "Density", xlab = "Fantasy Bins", main = "Naive Bayes Plot-RB/WR Pred", col="blue")
}
```

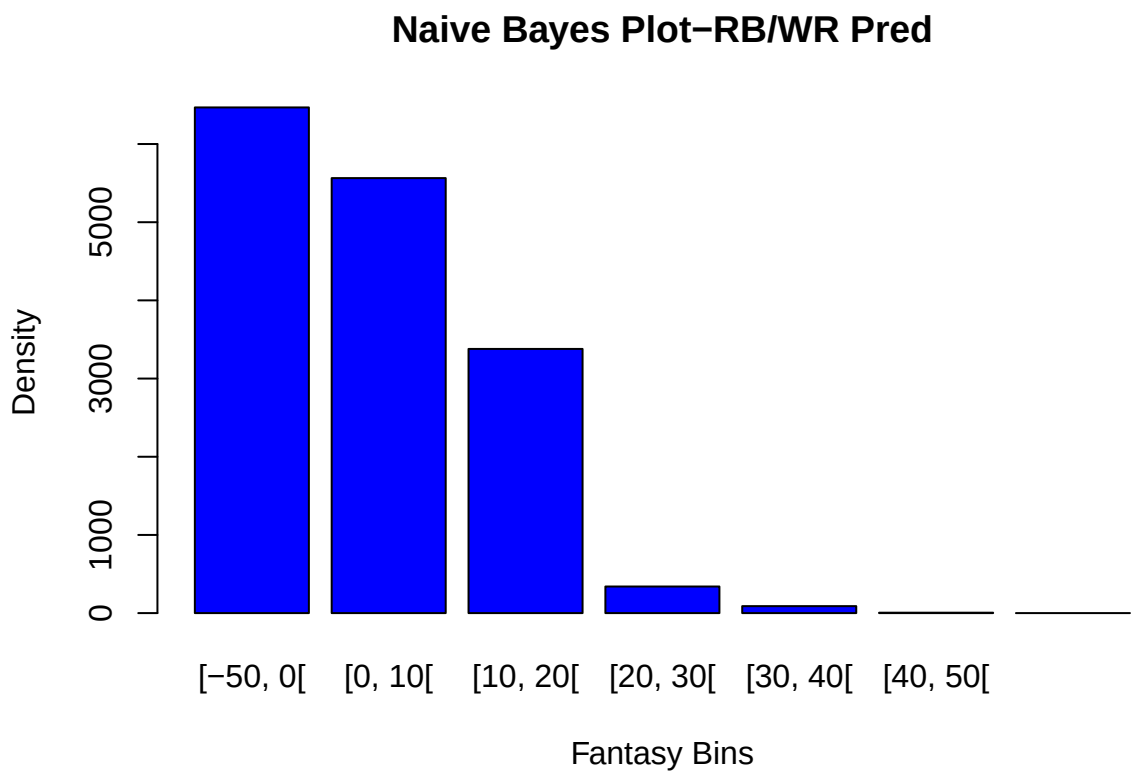


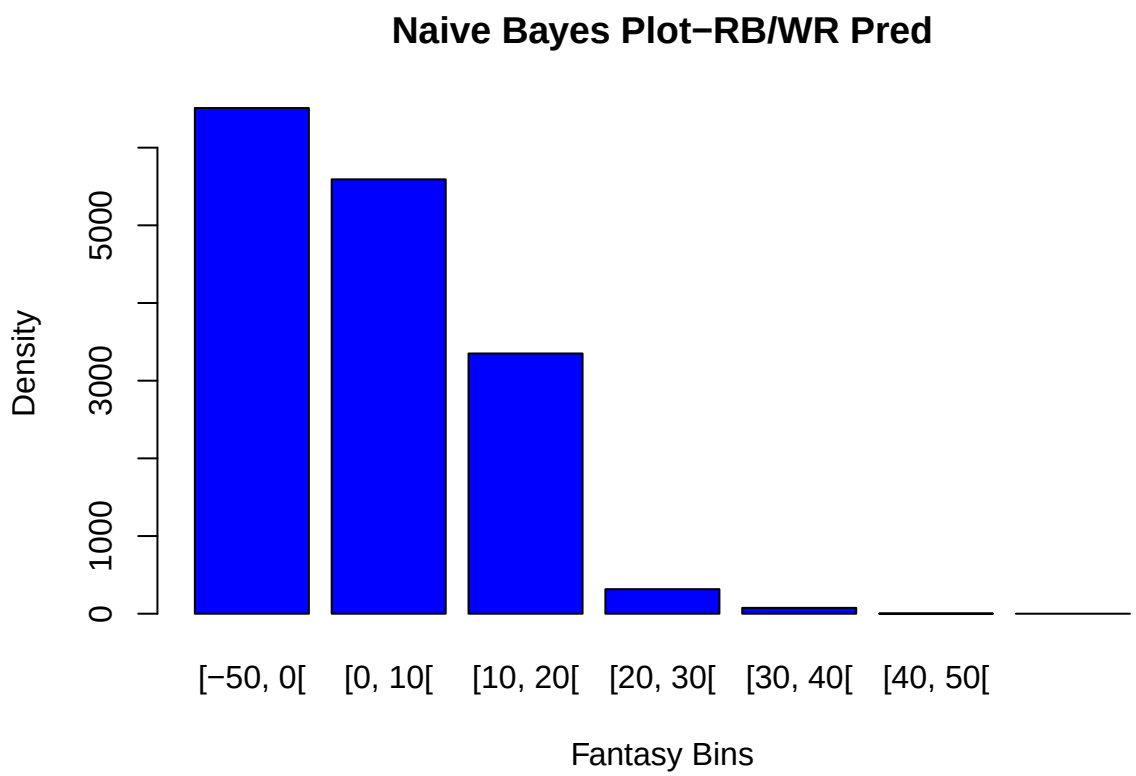

Naive Bayes Plot−RB/WR Pred
Density
0
1000
3000
5000
[−50, 0[
[0, 10[
[10, 20[
[20, 30[
[30, 40[
[40, 50[
Fantasy Bins

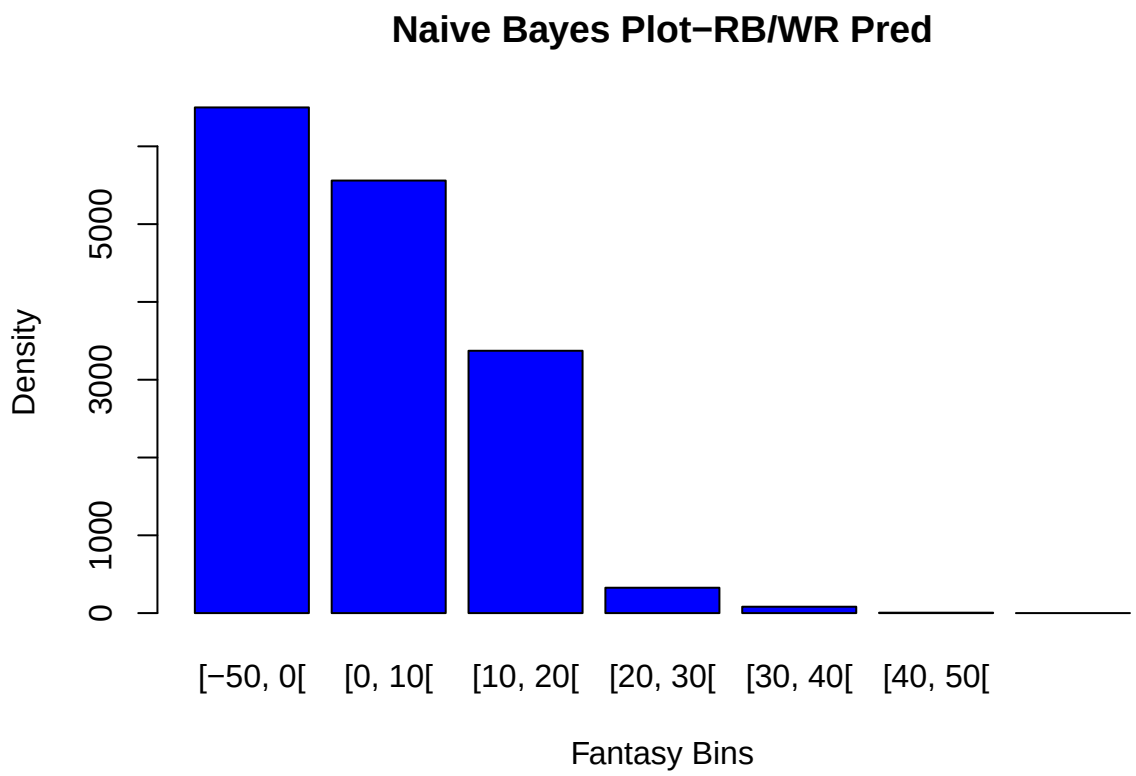

Naive Bayes Plot−RB/WR Pred
Density
0
1000
3000
5000
[−50, 0[
[0, 10[
[10, 20[
[20, 30[
[30, 40[
[40, 50[
Fantasy Bins

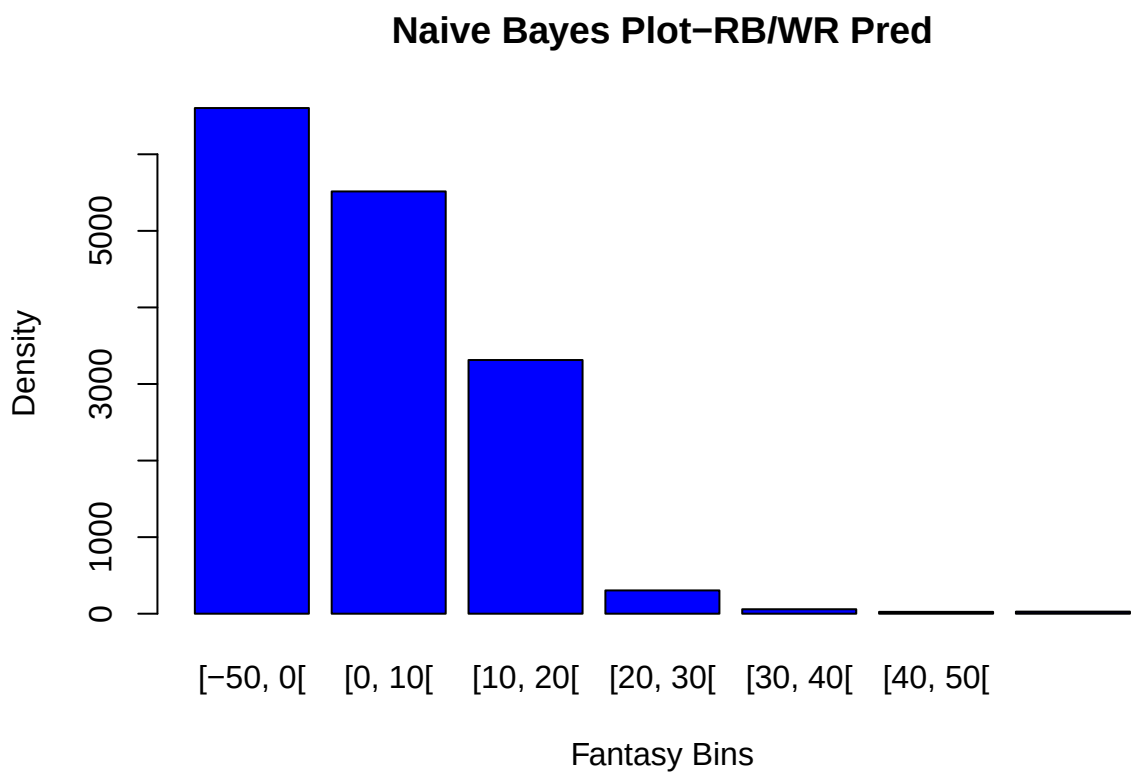

Naive Bayes Plot−RB/WR Pred
Density
0
1000
3000
5000
[−50, 0[
[0, 10[
[10, 20[
[20, 30[
[30, 40[
[40, 50[
Fantasy Bins

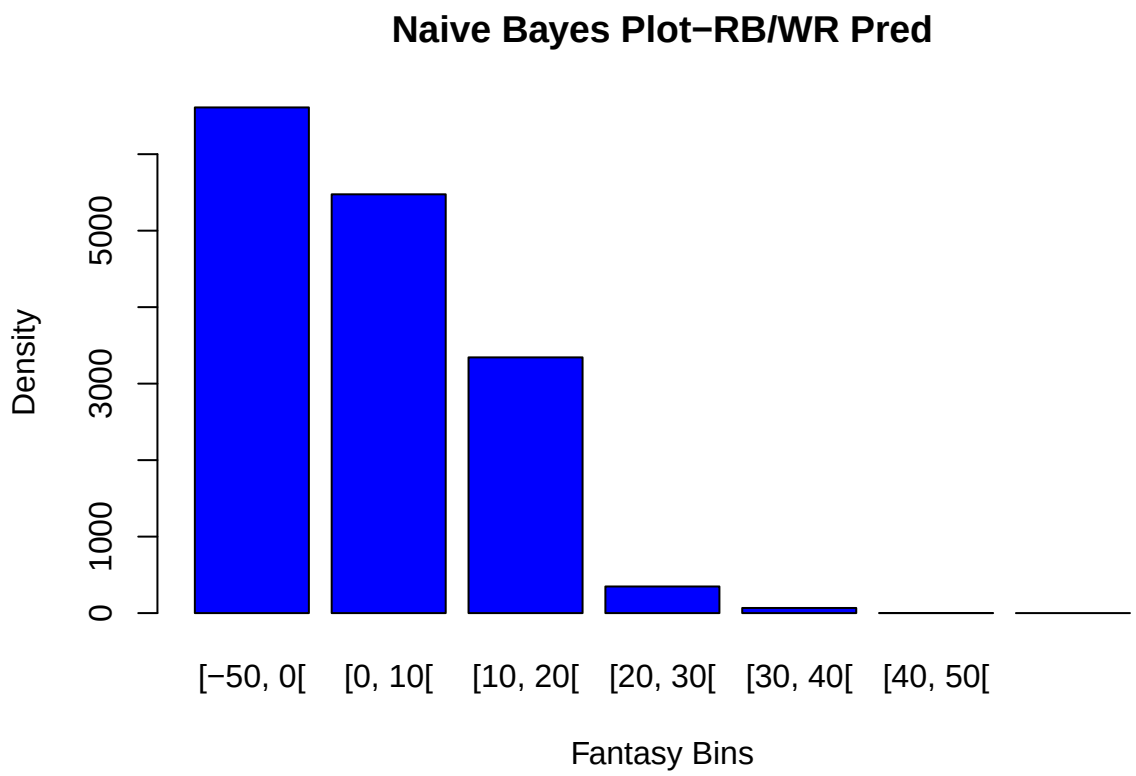

Naive Bayes Plot−RB/WR Pred
Density
0
1000
3000
5000
[−50, 0[
[0, 10[
[10, 20[
[20, 30[
[30, 40[
[40, 50[
Fantasy Bins


Naive Bayes Plot−RB/WR Pred
Density
0
1000
3000
5000
[−50, 0[
[0, 10[
[10, 20[
[20, 30[
[30, 40[
[40, 50[
Fantasy Bins


Naive Bayes Plot−RB/WR Pred
Density
0
1000
3000
5000
[−50, 0[
[0, 10[
[10, 20[
[20, 30[
[30, 40[
[40, 50[
Fantasy Bins


Naive Bayes Plot−RB/WR Pred
Density
0
1000
3000
5000
[−50, 0[
[0, 10[
[10, 20[
[20, 30[
[30, 40[
[40, 50[
Fantasy Bins


Naive Bayes Plot−RB/WR Pred
Density
0
1000
3000
5000
[−50, 0[
[0, 10[
[10, 20[
[20, 30[
[30, 40[
[40, 50[
Fantasy Bins




```
#Display the confusion matrix
NaiveBayes2
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   [-50, 0[ [0, 10[ [10, 20[ [20, 30[ [30, 40[ [40, 50[
##   [-50, 0[        759    5821       31        0        0        0
##   [0, 10[          12    5410       53        0        0        0
##   [10, 20[          7    1511     1743       81        1        0
##   [20, 30[          0       0      113      227        8        0
##   [30, 40[          0       3        5       23       32        4
##   [40, 50[          0       0        0        0        1        0
##   [50, +Inf[        0       0        0        0        0        0
##             Reference
## Prediction   [50, +Inf[
##   [-50, 0[            0
##   [0, 10[             0
##   [10, 20[            0
##   [20, 30[            0
##   [30, 40[            0
##   [40, 50[            0
##   [50, +Inf[          0
##
## Overall Statistics
##
##                Accuracy : 0.5157
##                  95% CI : (0.5079, 0.5235)
##     No Information Rate : 0.8044
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2827
##
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: [-50, 0[ Class: [0, 10[ Class: [10, 20[
## Sensitivity                   0.9756         0.4245          0.8961
## Specificity                   0.6116         0.9790          0.8849
## Pos Pred Value                0.1148         0.9881          0.5214
## Neg Pred Value                0.9979         0.2927          0.9838
## Prevalence                    0.0491         0.8044          0.1228
## Detection Rate                0.0479         0.3414          0.1100
## Detection Prevalence          0.4172         0.3455          0.2110
## Balanced Accuracy             0.7936         0.7018          0.8905
##                      Class: [20, 30[ Class: [30, 40[ Class: [40, 50[
## Sensitivity                 0.68580        0.761905       0.000e+00
## Specificity                 0.99220        0.997785       9.999e-01
## Pos Pred Value              0.65230        0.477612       0.000e+00
## Neg Pred Value              0.99329        0.999366       9.997e-01
## Prevalence                  0.02089        0.002651       2.524e-04
## Detection Rate              0.01433        0.002020       0.000e+00
## Detection Prevalence        0.02196        0.004228       6.311e-05
## Balanced Accuracy           0.83900        0.879845       5.000e-01
##                      Class: [50, +Inf[
## Sensitivity                         NA
## Specificity                          1
## Pos Pred Value                      NA
## Neg Pred Value                      NA
## Prevalence                           0
## Detection Rate                       0
## Detection Prevalence                 0
## Balanced Accuracy                   NA
```

Accuracy is a low 51.9% and remained constant over the 10 folds. This would need refinement. The p-value is NOT significant. Therefore we need to recalibrate the mashup of the RB, TE, WR stats. Across all 10 folds (and subsequent graphs) there minor variations in the location of bin Fantasy Points. Ultimately the bin for Fantasy -50 points remained the highest and most dense of there ranges. Bin 0 was the second highest. Clearly an accurate prediction here would be unreliable. Nevertheless I doubt model refinement would change the outcome.

Support Vector Machine

GL.QB SVM test and train split

In order to get a more manageable data set size we will first randomly sample 20% of the original GL.QB table, next we wil split our subset so that we will train with 67% of the subset and test on 33% of the subset. Finally we will create a new column that bins fantasy point results into "1" for players that score more than 15 and "0" for those that score 15 or fewer points.

```
N <- floor(nrow(GL.QB)*.20)
kfolds <- 5
set.seed(10)
holdout<- split(sample(1:N), 1:kfolds)
k<- .33*(N)

GL.QB$FantasyBIN<- ifelse(GL.QB$Fantasy > 15, 1, 0)
```

GL.QB SVM Fantasy Bin.

Now that the data is prepped for the test we will holdout our test data and run our SVM against the remaining training set. We will tell the machine to predict our Fantasy Bin given the selected variables. After trials with different kernels

and costs this final combination yielded the strongest results. Finally to grade the results we took the total number of matches between prediction and actual bin divided by the total number of results.

```
#QB1_resultsSVM <- data.frame(orig=c(), pred=c())
#for (k in 1:kfolds) {
# QB1_SVMtest <- GL.QB[holdout[[k]], ]
# QB1_SVMtrain <- GL.QB[-holdout[[k]], ]

# QB1.SVMtest_model <- svm(FantasyBIN ~ Home.or.Away + Passing.Yards +
#Completion.Percentage + TD.Passes + Ints + Rushing.Yards + Rushing.TDs + Passer.Rating +
#Outcome + Passes.Attempted, kernel= "polynomial", cost=.5 , QB1_SVMtrain, na.action=na.pass)

# QB1.SVMpred <- predict(QB1.SVMtest_model,type=c("class"))

#id_col=QB1_SVMtest$Player.Id

#QB1.SVMpred<- ifelse(QB1.SVMpred > 0.5, 1, 0)

#newpred=cbind(id_col, QB1_SVMtest$FantasyBIN,QB1.SVMpred)
#colnames(newpred)=c("Player.Id", "Fantasy", "FantasyPred")
#}
#QB.SVM<-as.data.frame(newpred)
# Below is a function to run a two sample Z-score test for grading fantasy point predictions.
#Score<- function(a,b, m1, m2){
# length.a<- length(a)
#  length.b<- length(b)
#  z<- (mean(a)-mean(b)) / (sqrt(m1/length.a + m2/length.b))

#  return(z)
#}

#pre_grade<-length(which(QB.SVM$Fantasy==QB.SVM$FantasyPred))
#grade<- pre_grade/length(QB.SVM$Fantasy)
#grade
#QB.SVM
```

GL.RB.WR SVM Fantasy Bin

Similar to how we prepped the GL.QB dataset we must first subset the GL.RB.WR data so that it is a more manageable size. Because this table is so large we can only successfuly run the machine against a relatively small proportion. We still use the 67-33 split for training and test data and also create another new BIN column using the same parameters as earlier. After some experimentation we found a polynomial kernel with cost .5 and our selected variables below yielded the best results. We graded the results the same way as we graded the QB bin results by dividing the correct matches by the total results.

```
#N1 <- floor((nrow(GL.RB.WR))*.015)
#kfolds1 <- 10
#set.seed(10)
#holdout1<- split(sample(1:N1), 1:kfolds1)
#k1<- .33*(N1)

#GL.RB.WR$FantasyBIN<- ifelse(GL.RB.WR$Fantasy > 15, 1, 0)

#RB.WR_resultsSVM <- data.frame(orig=c(), pred=c())
#for (k1 in 1:kfolds1) {
# RB.WR_SVMtest <- GL.RB.WR[holdout1[[k1]], ]
# RB.WR_SVMtrain <- GL.RB.WR[-holdout1[[k1]], ]
```

```
# RB.WR.SVMtest_model <- svm(FantasyBIN ~ Home.or.Away+ Outcome + Receptions +
#Rushing.Yards+ Longest.Rushing.Run + Rushing.TDs + Receiving.TDs + Receiving.Yards,
#kernel= "polynomial", cost=.5 , RB.WR_SVMtrain, na.action=na.pass)

# RB.WR.SVMpred <- predict(RB.WR.SVMtest_model,type=c("class"))

#id_col2=RB.WR_SVMtest$Player.Id

#RB.WR.SVMpred<- ifelse(RB.WR.SVMpred > 0.5, 1, 0)

#newpred2=cbind(id_col, RB.WR_SVMtest$FantasyBIN,RB.WR.SVMpred)
#colnames(newpred2)=c("Player.Id", "Fantasy", "FantasyPred")
#}
#RB.WR.SVM<- as.data.frame(newpred2)
#pre_grade1<-length(which(RB.WR.SVM$Fantasy==RB.WR.SVM$FantasyPred))
#grade1<- pre_grade1/length(RB.WR.SVM$Fantasy)
#grade1
#RB.WR.SVM
```

GL.QB SVM Fantasy point prediction

Next we tried to predict the actual fantasy point production without binning the results. By using a very similar formula to the same SVM we made earlier but replacing "FantasyBIN" with "Fantasy" for the variable we want to predict we were able to generate a "Fantasy" point prediction SVM. The results were scored by looking at the Z-score(using our "Score" function from earlier) results between the true output and prediction as well as measuring error on the individual rows as the absolute difference between prediction and actual output. We also calculated the average error between prediction and actual output.

```
#QB2_resultsSVM <- data.frame(orig=c(), pred=c())
#for (k in 1:kfolds) {
# QB1_SVM1test <- GL.QB[holdout[[k]], ]
# QB1_SVM1train <- GL.QB[-holdout[[k]], ]

# QB1.SVM1test_model <- svm(Fantasy ~ Home.or.Away + Passing.Yards +
#Completion.Percentage + TD.Passes + Ints + Rushing.Yards + Rushing.TDs +
#Passer.Rating + Outcome + Passes.Attempted, kernel= "polynomial", cost=1 ,
#QB1_SVM1train, na.action=na.pass)

# QB1.SVM1pred <- predict(QB1.SVM1test_model,type=c("class"))

#id_col1=QB1_SVM1test$Player.Id

#newpred2=cbind(id_col, QB1_SVMtest$Fantasy,QB1.SVM1pred)
#colnames(newpred2)=c("Player.Id", "Fantasy", "FantasyPred")

#}
#QB1.SVM<- as.data.frame(newpred2)

#Score(QB1.SVM$Fantasy, QB1.SVM$FantasyPred, var(QB1.SVM$Fantasy), var(QB1.SVM$FantasyPred))
#QB1.SVM$error<- abs(QB1.SVM$Fantasy-QB1.SVM$FantasyPred)
#avg.error<- mean(QB1.SVM$error)
#avg.error
#QB1.SVM
```

GL.RB.WR SVM Fantasy Point Prediction

The transformation from the BIN prediction to the actual points prediction was the same as when we were testing the GL.QB table, simply replace "FantasyBIN" with "Fantasy" as our predicted variable and adjusting independent variables, kernel and cost to optimize result. These results were also graded by examining the Z-score, row errors and average row error.

```
#RB.WR2_resultsSVM <- data.frame(orig=c(), pred=c())
#for (k1 in 1:kfolds1) {
# RB.WR2_SVMtest <- GL.RB.WR[holdout1[[k1]], ]
# RB.WR2_SVMtrain <- GL.RB.WR[-holdout1[[k1]], ]

# RB.WR.test_model2 <- svm(Fantasy ~ Home.or.Away+ Outcome + Receptions +
#Rushing.Yards+ Longest.Rushing.Run + Rushing.TDs + Receiving.TDs + Receiving.Yards,
#kernel= "polynomial", cost=1 , RB.WR2_SVMtrain, na.action=na.pass)

# RB.WR.SVMpred2 <- predict(RB.WR.test_model2,type=c("class"))

#id_col3=RB.WR2_SVMtest$Player.Id

#newpred3=cbind(id_col, RB.WR2_SVMtest$Fantasy,RB.WR.SVMpred2)
#colnames(newpred3)=c("Player.Id", "Fantasy", "FantasyPred")

#}
#RB.WR.SVM2<- as.data.frame(newpred3)

#Score(RB.WR.SVM2$Fantasy, RB.WR.SVM2$FantasyPred, var(RB.WR.SVM2$Fantasy), var(RB.WR.SVM2$FantasyPred))
#RB.WR.SVM2$error<- abs(RB.WR.SVM2$Fantasy-RB.WR.SVM2$FantasyPred)
#avg.error2<- mean(RB.WR.SVM2$error)
#avg.error2
#RB.WR.SVM2
```

CS.Pass Passer Rating Bin Prediction

Rather than subsetting randomly for the career stats table we thought it best to subset using given data. Taking this approach we felt that QB's who played in more than half(8) of the games would yield more interesting results. After subsetting and splitting the data we trained the machine using all available variables in the CS.Pass table to try to predict the QB's passer rating bin, "1" being a passer rating greater than 35 and "2" being a passer rating 35 or less. Trials led us to choose a polynomial kernel with a cost of 10 and the results were graded using the same method as the other bin predictions, matches divided by total results.

```
#CS.Pass1<- CS.Pass[which(CS.Pass$Games.Played > 8),]
#N2 <- nrow(CS.Pass1)
#kfolds2 <- 15
#set.seed(10)
#holdout2<- split(sample(1:N2), 1:kfolds2)
#k2<- .33*(N2)

#CS.Pass1$RatingBIN<- ifelse(CS.Pass1$Passer.Rating > 35, 1, 0)

#CS.Pass_resultsSVM <- data.frame(orig=c(), pred=c())
#for (k2 in 1:kfolds2) {
# CS.Pass_SVMtest <- CS.Pass1[holdout2[[k2]], ]
# CS.Pass_SVMtrain <- CS.Pass1[-holdout2[[k2]], ]

# CS.Pass.SVMtest_model <- svm(Passer.Rating ~ ., kernel= "polynomial",
      #cost= 10 , CS.Pass_SVMtrain, na.action=na.pass)

 #CS.Pass.SVMpred <- predict(CS.Pass.SVMtest_model,type=c("class"))
```

```
#id_col4<- CS.Pass_SVMtest$Player.Id

#CS.Pass.SVMpred<- ifelse(CS.Pass.SVMpred > 0.5, 1, 0)

#newpred4<- cbind(id_col4, CS.Pass1$RatingBIN,CS.Pass.SVMpred)
#colnames(newpred4)=c("Player.Id", "Passer.Rating", "P.Rating.Pred")
#}

#CS.Pass.SVM<- as.data.frame(newpred4)
#pre_grade2<-length(which(CS.Pass.SVM$Passer.Rating==CS.Pass.SVM$P.Rating.Pred))
#grade2<- pre_grade2/length(CS.Pass.SVM$Passer.Rating)
#grade2
#CS.Pass.SVM

#CS.Pass1$BINpred<- CS.Pass.SVM$P.Rating.Pred
```

Conclusion

Clive Humby was right. As you can see there is a "new-wealth" in data. It might be in the beauty of visualizations or uncovering new exploratory analysis (who knew that the longest football field goal is 64 yards?), or in seeing the Quarterback stats really do follow a normal distribution. For some it will be the beauty of evaluating overall average performance at the football skill level of running backs and wide receivers. For others it is the dramatic increase in receiving yards beginning in 2008. We can probably score a lot of beer on that argument alone!

On the Fantasy Scores the emergence of Week 8 as a driver of 68% of QBs obtaining 18 fantasy points is a key in team selection and if a player started the game and was not a receiver (i.e. was a running back), then there is a 71% percent chance that the running back will score 10 or more fantasy points. 43% of the original data supported this. Also, if the player did not start, but did have at least one play on the ball and was a running back, then there is a 90% chance that the running back will score 10 or more fantasy points. This is clearly important to team selection (and betting)!

Association Rules Mining connected us to the confidence and lift of Home games. Home games appeared in the top rules of both of our datasets. When the rules are sorted by "Support", Receiving TDs = 1 appear in all the top 5 rules, this leads us to believe that Receiving TDs = 1 is also a very common variable and has a good association with winning games.

Our Naïve Bayes accuracy for our QB data set is 66.4% certainly not super high but decent. The p-value is significant. Across all 10 folds (and subsequent graphs) there were significant variations in the location of bin Fantasy Points. Ultimately the bin for Fantasy 10 points remained the highest and most dense of these ranges. Bin -50 was the second highest. These make our earlier views on fantasy points by QBs and Receivers even more interesting. Clearly the prediction that given the QB_Test set most QBs will be in the Fantasy point range of 0-10.

For our Support Vector Machine (SVM) algorithm runs we found that after some experimentation a polynomial kernel with cost .5 and selected variables yielded the best results. We graded the results the same way as we graded the QB bin results by dividing the correct matches by the total results. After sub setting and splitting the data we trained the machine using all available variables in the CS.Pass table to try to predict the QB's passer rating bin, "1" being a passer rating greater than 35 and "2" being a passer rating 35 or less. Trials led us to choose a polynomial kernel with a cost of 10 and the results were graded using the same method as the other bin predictions, matches divided by total results.

In summary, we feel we have both learned a "wealth of new information" by both exploring and wrangling with the data sets. Our outcomes provide us with some clear "bets" to place on our next team roster selections.