Bios 6301: Final Project

Molly Olson

December 14, 2015

Due Monday, 14 December, 6:00 PM

200 points total.

Submit a single knitr file (named `final.rmd`), along with a valid PDF output file. Add your name as `author` to the file's metadata section. Raw R code/output or word processor files are not acceptable.

All work should be done by the student, please no collaboration. You may ask the instructor for help or clarification.

Obtain a copy of the football-values lecture – make sure to update this repository if you have previously cloned it. Save the six 2015 CSV files in your working directory (note the new file `nfl_current15.csv`). You may utilize assignment 4, question 3 in your solution.

Task 1: Finding Residuals (80 points)

At the beginning of the course we examined projections for the 2015 NFL season. With the season ~60% completed, let's compare the observed values to the estimated values. Place all code at the end of the instructions.

1. Read and combine the projection data (five files) into one data set, adding a position column.
2. The NFL season is 17 weeks long, and 10 weeks have been completed. Each team plays 16 games and has one week off, called the bye week. Four teams have yet to have their bye week: CLE, NO, NYG, PIT. These four teams have played ten games, and every other team has played nine games. Multiply the numeric columns in the projection data by the percentage of games played (for example, 10/16 if team is PIT).
3. Sort and order the data by the `fpts` column descendingly. Subset the data by keeping the top 20 kickers, top 20 quarterbacks, top 40 running backs, top 60 wide recievers, and top 20 tight ends. Thus the projection data should only have 160 rows.
4. Read in the observed data (`nfl_current15.csv`)
5. Merge the projected data with the observed data by the player's name. Keep all 160 rows from the projection data. If observed data is missing, set it to zero.

   You can directly compare the projected and observed data for each player. There are fifteen columns of interest:

```
##                     Name projected_col observed_col
## 1            field goals            fg          FGM
## 2  field goals attempted           fga          FGA
## 3           extra points           xpt          XPM
## 4       passing attempts      pass_att     Att.pass
## 5    passing completions      pass_cmp     Cmp.pass
## 6          passing yards      pass_yds     Yds.pass
## 7     passing touchdowns      pass_tds      TD.pass
## 8  passing interceptions     pass_ints     Int.pass
## 9       rushing attempts      rush_att     Att.rush
```

```
## 10         rushing yards      rush_yds     Yds.rush
## 11    rushing touchdowns      rush_tds      TD.rush
## 12    receiving attempts       rec_att    Rec.catch
## 13       receiving yards       rec_yds    Yds.catch
## 14  receiving touchdowns       rec_tds     TD.catch
## 15               fumbles       fumbles          Fmb
```

6. Take the difference between the observed data and the projected data for each category. Split the data by position, and keep the columns of interest.

You will now have a list with five elements. Each element will be a matrix or data.frame with 15 columns.

```
#1. Read and combine the projection data (five files) into one data set, adding a position column.

setwd("~/Documents/Vanderbilt_2015_Fall/Statistical_Computing/Final")

  #read in data
  k <- read.csv('proj_k15.csv', header=TRUE, stringsAsFactors=FALSE)
  qb <- read.csv('proj_qb15.csv', header=TRUE, stringsAsFactors=FALSE)
  rb <- read.csv('proj_rb15.csv', header=TRUE, stringsAsFactors=FALSE)
  te <- read.csv('proj_te15.csv', header=TRUE, stringsAsFactors=FALSE)
  wr <- read.csv('proj_wr15.csv', header=TRUE, stringsAsFactors=FALSE)

  # generate unique list of column names
    cols <- unique(c(names(k), names(qb), names(rb), names(te), names(wr)))

    # create a new column in each data.frame
    # values are recylcled
    # concept: ?Extract
    k[,'pos'] <- 'k'
    qb[,'pos'] <- 'qb'
    rb[,'pos'] <- 'rb'
    te[,'pos'] <- 'te'
    wr[,'pos'] <- 'wr'

    # append 'pos' to unique column list
    cols <- c(cols, 'pos')

    # create common columns in each data.frame
    # initialize values to zero
    k[,setdiff(cols, names(k))] <- 0
    qb[,setdiff(cols, names(qb))] <- 0
    rb[,setdiff(cols, names(rb))] <- 0
    te[,setdiff(cols, names(te))] <- 0
    wr[,setdiff(cols, names(wr))] <- 0

    # combine data.frames by row, using consistent column order
    data <- rbind(k[,cols], qb[,cols], rb[,cols], te[,cols], wr[,cols])
```

```
#head(k)
#head(qb)
#head(rb)
#head(te)
#head(wr)
```

```
    #1. The NFL season is 17 weeks long, and 10 weeks have been completed.  Each team plays 16 games an
    #to have their bye week: CLE, NO, NYG, PIT.  These four teams have played ten games, and every othe
    #the projection data by the percentage of games played (for example, 10/16 if team is PIT).


  #teams that haven't had a bye week
dataNoBye <- data[data$Team == 'CLE' | data$Team == 'NO' | data$Team == 'NYG' | data$Team == 'PIT',]
    #teams that have had a bye week
dataBye <- data[data$Team != 'CLE' & data$Team != 'NO' & data$Team != 'NYG' & data$Team != 'PIT',]

scaleNoBye <- function(x) (10/16)*x
scaleBye <- function(x) (9/16)*x
dataNoBye <- cbind(dataNoBye[c(1,2,19)], lapply(dataNoBye[3:18], scaleNoBye) )
dataBye <- cbind(dataBye[c(1,2,19)], lapply(dataBye[3:18], scaleBye) )

dataScale <- rbind(dataNoBye,dataBye)
```

```
#1. Sort and order the data by the `fpts` column descendingly.  Subset the data by keeping the top 20 k
#wide recievers, and top 20 tight ends.  Thus the projection data should only have 160 rows.


projected <- dataScale[order(dataScale$fpts),]
bestkick <- projected[projected$pos == 'k',][c(1:20),]
bestqb <- projected[projected$pos == 'qb',][c(1:20),]
bestte <- projected[projected$pos == 'te',][c(1:20),]
bestwr <- projected[projected$pos == 'wr',][c(1:60),]
bestrb <- projected[projected$pos == 'rb',][c(1:40),]
projected <- rbind(bestkick, bestqb, bestte, bestwr, bestrb)
```

```
#1. Read in the observed data (`nfl_current15.csv`)


observed <- read.csv('nfl_current15.csv', header=TRUE, stringsAsFactors=FALSE)
```

```
##                                      Name projected_col observed_col
## 1                             field goals            fg          FGM
## 2                    field goals attempted           fga          FGA
## 3                             extra points           xpt          XPM
## 4                         passing attempts      pass_att     Att.pass
## 5                       passing completions      pass_cmp     Cmp.pass
## 6                             passing yards      pass_yds     Yds.pass
## 7                       passing touchdowns      pass_tds      TD.pass
## 8                     passing interceptions     pass_ints     Int.pass
## 9                         rushing attempts      rush_att     Att.rush
## 10                           rushing yards      rush_yds     Yds.rush
## 11 rushing\n                    touchdowns      rush_tds      TD.rush
## 12                      receiving attempts       rec_att    Rec.catch
## 13                         receiving yards       rec_yds    Yds.catch
## 14                   receiving touchdowns       rec_tds     TD.catch
## 15                               fumbles       fumbles          Fmb
```

```r
#1. Merge the projected data with the observed data by the player's name. Keep all 160 rows from the pr
#zero.


projected_col=c('PlayerName','Team','pos','fg','fga','xpt','pass_att','pass_cmp','pass_yds','pass_tds',
                'pass_ints', 'rush_att','rush_yds','rush_tds','rec_att','rec_yds','rec_tds','fumbles')
observed_col=c("Name","Tm","Pos","FGM","FGA","XPM","Att.pass","Cmp.pass","Yds.pass","TD.pass","Int.pass
                            "Att.rush","Yds.rush","TD.rush","Rec.catch","Yds.catch","TD.catch","Fmb")
df1 <- projected[,projected_col]
df2 <- observed[,observed_col]
names(df2) <- c("PlayerName","Tm","Pos","FGM","FGA","XPM","Att.pass","Cmp.pass","Yds.pass","TD.pass","I
               "Yds.catch","TD.catch","Fmb")
```

```r
mergedf <- merge(df1, df2, by = 'PlayerName',all.x=TRUE)
mergedf[is.na(mergedf)] <- 0
```

```r
#1. Take the difference between the observed data and the projected data for each category. Split the d

    mergedf[,'fg.dif']  <- mergedf[,'FGM'] - mergedf[,'fg']
    mergedf[,'fga.dif'] <- mergedf[,'FGA'] - mergedf[,'fga']
    mergedf[,'xpt.dif'] <- mergedf[,'XPM'] -mergedf[,'xpt']
    mergedf[,'pass_att.dif'] <-  mergedf[,'Att.pass'] - mergedf[,'pass_att']
    mergedf[,'pass_cmp.dif'] <-  mergedf[,'Cmp.pass'] - mergedf[,'pass_cmp']
    mergedf[,'pass_yds.dif'] <- mergedf[,'Yds.pass'] - mergedf[,'pass_yds']
    mergedf[,'pass_tds.dif'] <-  mergedf[,'TD.pass'] - mergedf[,'pass_tds']
    mergedf[,'pass_ints.dif'] <-  mergedf[,'Int.pass'] - mergedf[,'pass_ints']
    mergedf[,'rush_att.dif'] <-  mergedf[,'Att.rush'] - mergedf[,'rush_att']
    mergedf[,'rush_yds.dif'] <-  mergedf[,'Yds.rush'] - mergedf[,'rush_yds']
    mergedf[,'rush_tds.dif'] <-  mergedf[,'Yds.rush'] - mergedf[,'rush_tds']
    mergedf[,'rec_att.dif'] <- mergedf[,'Rec.catch'] - mergedf[,'rec_att']
    mergedf[,'rec_yds.dif'] <-  mergedf[,'Yds.catch'] - mergedf[,'rec_yds']
    mergedf[,'rec_tds.dif'] <- mergedf[,'TD.catch'] - mergedf[,'rec_tds']
    mergedf[,'fumbles.dif'] <- mergedf[,'Fmb'] - mergedf[,'fumbles']

diff.df <- mergedf[,c(1,2,3,36:50)]
```

```r
finaldf <- split(diff.df, diff.df$pos)
```

Task 2: Creating League S3 Class (80 points)

Create an S3 class called `league`. Place all code at the end of the instructions.

1. Create a function `league` that takes 5 arguments (`stats`, `nTeams`, `cap`, `posReq`, `points`). It should return an object of type `league`. Note that all arguments should remain attributes of the object. They define the league setup and will be needed to calculate points and dollar values.

2. Create a function `calcPoints` that takes 1 argument, a league object. It will modify the league object by calculating the number of points each player earns, based on the league setup.

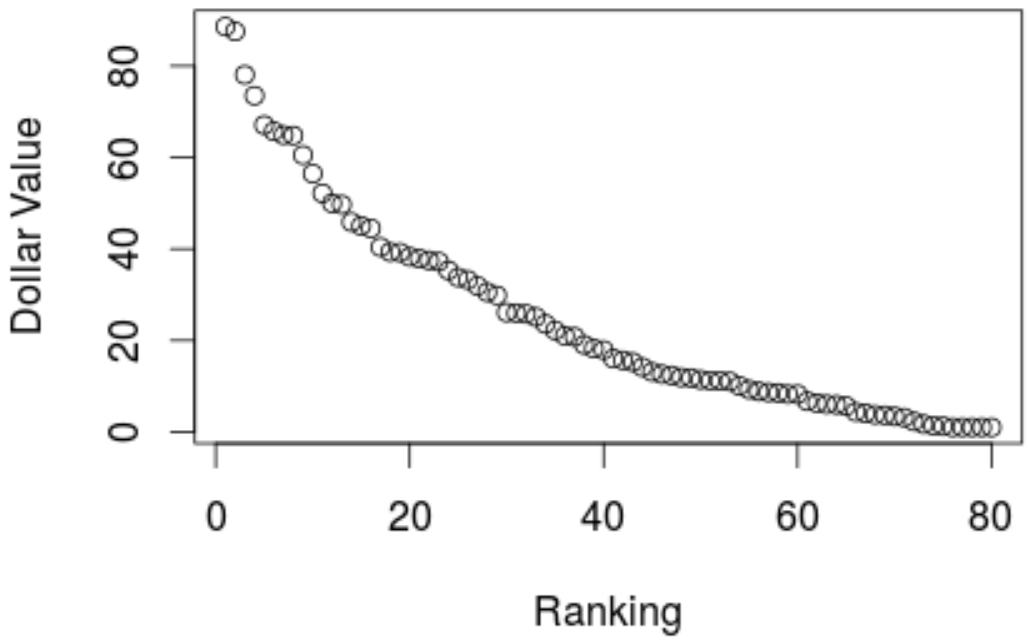
3. Create a function `buildValues` that takes 1 argument, a league object. It will modify the league object by calculating the dollar value of each player.

   As an example if a league has ten teams and requires one kicker, the tenth best kicker should be worth $1. All kickers with points less than the 10th kicker should have dollar values of $0.

4. Create a `print` method for the league class. It should print the players and dollar values (you may choose to only include players with values greater than $0).

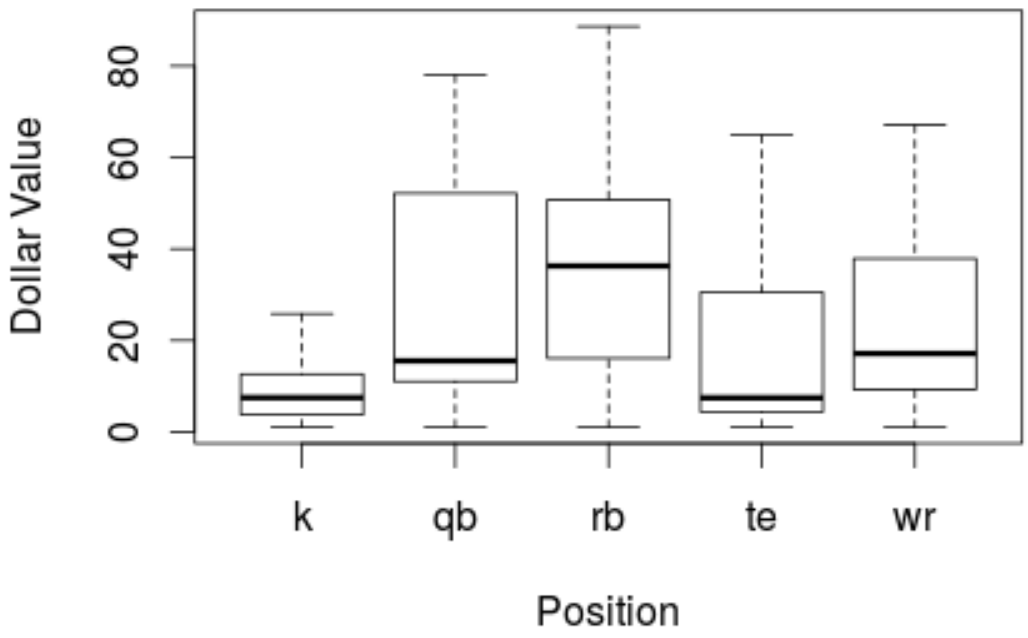
5. Create a `plot` method for the league class. Add minimal plotting decorations (such as axis labels).

    - Here's an example:



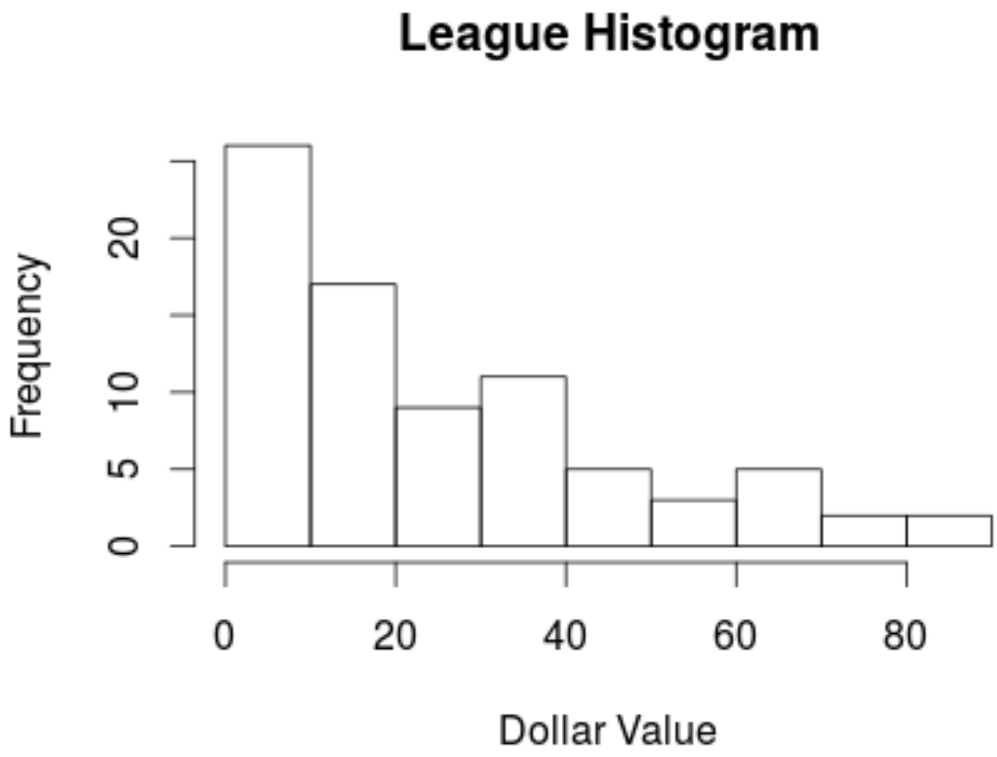
6. Create a `boxplot` method for the league class. Add minimal plotting decorations.

    - Here's an example:



7. Create a `hist` method for the league class. Add minimal plotting decorations.

    - Here's an example:



I will test your code with the following:

```
# x is combined projection data
pos <- list(qb=1, rb=2, wr=3, te=1, k=1)
pnts <- list(fg=4, xpt=1, pass_yds=1/25, pass_tds=4, pass_ints=-2,
             rush_yds=1/10, rush_tds=6, fumbles=-2, rec_yds=1/20, rec_tds=6)
l <- league(stats=x, nTeams=10, cap=200, posReq=pos, points=pnts)
l
hist(l)
boxplot(l)
plot(l)
```

I will test your code with additional league settings (using the same projection data). I will try some things that should work and some things that should break. Don't be too concerned, but here's some things I might try:

- Not including all positions
- Including new positions that don't exist
- Requiring no players at a position
- Requiring too many players at a position (ie - there aren't 100 kickers)

Note that at this point it should be easy to change a league setting (such as `nTeams`) and re-run `calcPoints` and `buildValues`.

```r
#1. Create a function `league` that takes 5 arguments (`stats`, `nTeams`, `cap`, `posReq`, `points`).
#all arguments should remain attributes of the object.  They define the league setup and will be needed


league <- function(stats, nTeams, cap, posReq, points){
  outputObj = list(stats = stats,nTeams = nTeams, cap = cap, posReq = posReq, points = points)
  class(outputObj) = 'league'
  return(outputObj)
}
```

```r
#1. Create a function `calcPoints` that takes 1 argument, a league object.  It will modify the league o
#earns, based on the league setup.

calcPoints <- function(league){
  x <- league$stats
  x[,'p_fg'] <- x[,'fg']*4
  x[,'p_xpt'] <- x[,'xpt']*1
  x[,'p_pass_yds'] <- x[,'pass_yds']/25
  x[,'p_pass_tds'] <- x[,'pass_tds']*4
  x[,'p_pass_ints'] <- x[,'pass_ints']*-2
  x[,'p_rush_yds'] <- x[,'rush_yds']/10
  x[,'p_rush_tds'] <- x[,'rush_tds']*6
  x[,'p_fumbles'] <- x[,'fumbles']*-2
  x[,'p_rec_yds'] <- x[,'rec_yds']/20
  x[,'p_rec_tds'] <- x[,'rec_tds']*6

  x[,'points'] <- rowSums(x[,grep("^p_", names(x))])
  return(x)
}
```

```r
#1. Create a function `buildValues` that takes 1 argument, a league object.  It will modify the league
```

```r
buildValues <- function(league){
    x <- calcPoints(league)
      # create new data.frame ordered by points descendingly
    x2 <- x[order(x[,'points'], decreasing=TRUE),]


    # determine the row indeces for each position
    k.ix <- which(x2[,'pos']=='k')
    qb.ix <- which(x2[,'pos']=='qb')
    rb.ix <- which(x2[,'pos']=='rb')
    te.ix <- which(x2[,'pos']=='te')
    wr.ix <- which(x2[,'pos']=='wr')

    x2[qb.ix, 'marg'] <- x2[qb.ix,'points'] - x2[qb.ix[max(1,as.numeric(league$posReq['qb']))*league$nT
    x2[rb.ix, 'marg'] <- x2[rb.ix,'points'] - x2[rb.ix[max(1,as.numeric(league$posReq['rb']))*league$nT
    x2[wr.ix, 'marg'] <- x2[wr.ix,'points'] - x2[wr.ix[max(1,as.numeric(league$posReq['wr']))*league$nT
    x2[te.ix, 'marg'] <- x2[te.ix,'points'] - x2[te.ix[max(1,as.numeric(league$posReq['te']))*league$nT
    x2[k.ix,  'marg'] <- x2[k.ix, 'points'] -  x2[k.ix[max(1,as.numeric(league$posReq['k']))*league$nTea

    x3 <- x2[x2[,'marg'] >= 0,]
```

```
    x3 <- x3[order(x3[,'marg'], decreasing=TRUE),]
    rownames(x3) <- NULL
    x3[,'value'] <- x3[,'marg']*(league$nTeams*league$cap-nrow(x3))/sum(x3[,'marg']) + 1
    x4 <- x3[,c('PlayerName','pos','points','marg','value')]
    return(x4)
}
```

```
#1. Create a `print` method for the league class. It should print the players and dollar values (you ma
#$0).

#**Note: I changed the print, plot, hist, and boxplot function names to print.league, plot.league, hist
#plotting tools in the function. So when I called the function name, it didn't like it because it thoug
#**I interpreted that you would test the data using "projected data" meaning a data frame with the top
#example is named `projected`**

print.league <- function(league){
  subset <- buildValues(league)[buildValues(league)$value > 0,]
  for(i in 1:nrow(subset))
  cat(i,"Name:", subset[i,1],"      ", "Value: $", subset$value[i], "\n")
}
#print.league(l)
```

```
#1. Create a `plot` method for the league class. Add minimal plotting decorations (such as axis labels)


plot.league <- function(league){
  subset <- buildValues(league)
  subset[,'rank'] <- seq(1:length(subset[,'value']))
  plot(subset$rank,subset$value, xlab = "Ranking", ylab = "Dollar Value")
}
```

```
#1. Create a `boxplot` method for the league class. Add minimal plotting decorations.

boxplot.league <- function(league) {
  data <- buildValues(league)
  boxplot(data$value ~ data$pos, xlab="Position", ylab="Dollar Value")
}
```

```
#1. Create a `hist` method for the league class. Add minimal plotting decorations.

hist.league <- function(league){
  data <- buildValues(league)
  hist(data$value, xlab = "Dollar Value",col="red")
}
```

```
#TESTING
# x is combined projection data
# pos <- list(qb=1, rb=2, wr=3, te=1, k=1)
# pnts <- list(fg=4, xpt=1, pass_yds=1/25, pass_tds=4, pass_ints=-2,
#              rush_yds=1/10, rush_tds=6, fumbles=-2, rec_yds=1/20, rec_tds=6)
# l <- league(stats=projected, nTeams=10, cap=200, posReq=pos, points=pnts)
# l
```

```
# calcPoints(l)
# buildValues(l)
# hist.league(l)
# boxplot.league(l)
# plot.league(l)
```

Task 3: Simulations with Residuals (40 points)

Using residuals from task 1, create a list of league simulations. The simulations will be used to generate confidence intervals for player values. Place all code at the end of the instructions.

1. Create a function `addNoise` that takes 4 arguments: a league object, a list of residuals, number of simulations to generate, and a RNG seed. It will modify the league object by adding a new element `sims`, a matrix of simulated dollar values.

   The original league object contains a `stats` attribute. Each simulation will modify this by adding residual values. This modified `stats` data.frame will then be used to create a new league object (one for each simulation). Calculate dollar values for each simulation. Thus if 1000 simulations are requested, each player will have 1000 dollar values. Create a matrix of these simulated dollar values and attach it to the original league object.

   As an example assume you want to simulate new projections for quarterbacks. The residuals for quarterbacks is a 20x15 matrix. Each row from this matrix is no longer identified with a particular player, but rather it's potential error. Given the original projection for the first quarterback, sample one value between 1 and 20. Add the 15 columns from the sampled row to the 15 columns for the first quarterback. Repeat the process for every quarterback. Note that stats can't be negative so replace any negative values with 0.

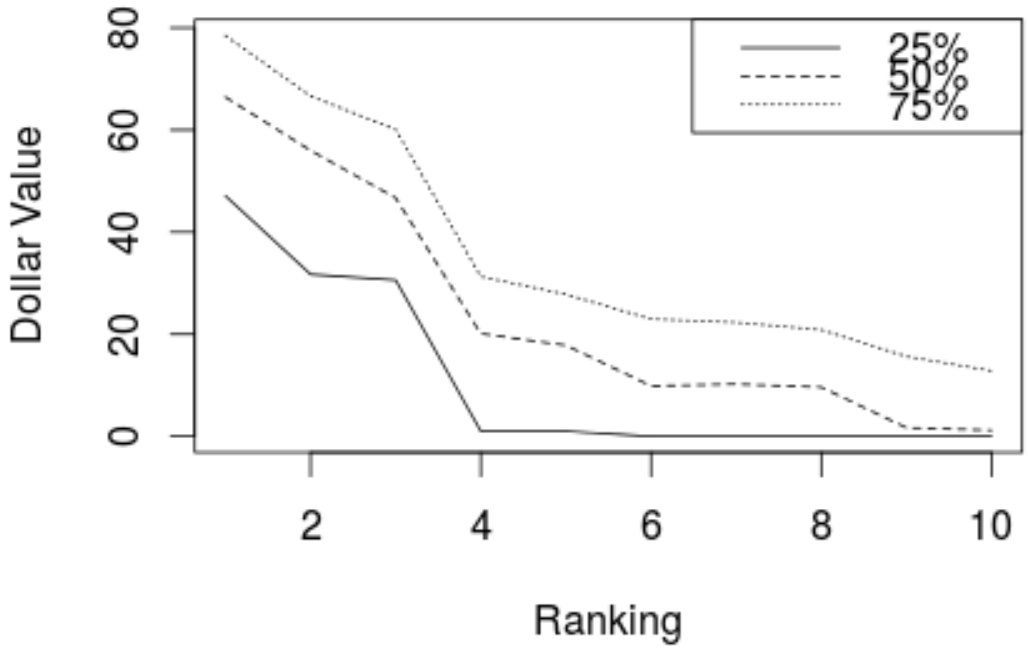
2. Create a `quantile` method for the league class; it takes at least two arguments, a league object and a probs vector. This method requires the `sims` element; it should fail if `sims` is not found. The `probs` vector should default to `c(0.25, 0.5, 0.75)`. It should run `quantile` on the dollar values for each player.

3. Create a function `conf.interval`; it takes at least two arguments, a league object and a probs vector. This method requires the `sims` element; it should fail if `sims` is not found. It should return a new object of type `league.conf.interval`.

   The new object will contain the output of `quantile`. However, results should be split by position and ordered by the last column (which should be the highest probability) descendingly. Restrict the number of rows to the number of required players at each position.

4. Create a `plot` method for the league.conf.interval class; it takes at least two arguments, a league.conf.interval object and a position. Plot lines for each probability; using the defaults, you would have three lines (0.25, 0.5, 0.75). Add minimal plotting decorations and a legend to distinguish each line.

   - Here's an example:



I will test your code with the following:

```
# l1 <- addNoise(l, noise, 10000)
# quantile(l1)
# ci <- conf.interval(l1)
# plot(ci, 'qb')
# plot(ci, 'rb')
# plot(ci, 'wr')
# plot(ci, 'te')
# plot(ci, 'k')
```

I am interpreting that the stats attribute will be in the format of the projected data, the data frame of the top 160 players.

```
#1. Create a function `addNoise` that takes 4 arguments: a league object, a list of residuals, number o
#the league object by adding a new element `sims`, a matrix of simulated dollar values.

#helper function, changed the x value from calPoints
calcPointsTask3 <- function(df){
  x <- df
  x['p_fg'] <- x['fg']*4
  x['p_xpt'] <- x['xpt']*1
  x['p_pass_yds'] <- x['pass_yds']/25
  x['p_pass_tds'] <- x['pass_tds']*4
  x['p_pass_ints'] <- x['pass_ints']*-2
```

```
  x[,'p_rush_yds'] <- x[,'rush_yds']/10
  x[,'p_rush_tds'] <- x[,'rush_tds']*6
  x[,'p_fumbles'] <- x[,'fumbles']*-2
  x[,'p_rec_yds'] <- x[,'rec_yds']/20
  x[,'p_rec_tds'] <- x[,'rec_tds']*6

  x[,'points'] <- rowSums(x[,grep("^p_", names(x))])
  return(x)
}
```

```
#helper function. uses calcPointsTask3.
buildValuesTask3 <- function(df,league){
    x <- calcPointsTask3(df)
      # create new data.frame ordered by points descendingly
    x2 <- x[order(x[,'points'], decreasing=TRUE),]


    # determine the row indeces for each position
    k.ix <- which(x2[,'pos']=='k')
    qb.ix <- which(x2[,'pos']=='qb')
    rb.ix <- which(x2[,'pos']=='rb')
    te.ix <- which(x2[,'pos']=='te')
    wr.ix <- which(x2[,'pos']=='wr')

    x2[qb.ix, 'marg'] <- x2[qb.ix,'points'] - x2[qb.ix[max(1,as.numeric(league$posReq['qb']))*league$nT
    x2[rb.ix, 'marg'] <- x2[rb.ix,'points'] - x2[rb.ix[max(1,as.numeric(league$posReq['rb']))*league$nT
    x2[wr.ix, 'marg'] <- x2[wr.ix,'points'] - x2[wr.ix[max(1,as.numeric(league$posReq['wr']))*league$nT
    x2[te.ix, 'marg'] <- x2[te.ix,'points'] - x2[te.ix[max(1,as.numeric(league$posReq['te']))*league$nT
    x2[k.ix,  'marg'] <- x2[k.ix, 'points'] -  x2[k.ix[max(1,as.numeric(league$posReq['k']))*league$nTea

    #x3 <- x2[x2[,'marg'] >= 0,]

    x3 <- x2[order(x2[,'marg'], decreasing=TRUE),]
    rownames(x3) <- NULL
    x3[,'value'] <- x3[,'marg']*(league$nTeams*league$cap-nrow(x3))/sum(x3[,'marg']) + 1
    x4 <- x3[,c('pos','points','marg','value')]
    return(x4)
}
```

```
#helper function for addNoise, will return the vector of values for one iteration.
returnValueVector <- function(league, res){
  data <- league$stats
  pos <- league$posReq
  #matrix for residuals
  matx <- matrix(NA, nrow = nrow(data), ncol = 15)

      for(i in 1:nrow(data)){
        #if ith row in the stats is a kicker's stats
        if(data[i,]$pos == 'k'){
          #sample a row in the kicker section of finaldf list
          r <- as.matrix(res$k[sample(nrow(res$k),1, replace=T),c(4:18)])
          #assign it to the ith row of the matrix
          matx[i,] <- r
```

```
        }
        if(data[i,]$pos == 'qb'){
          r<- as.matrix(res$qb[sample(nrow(res$qb),1, replace=T),c(4:18)])
          matx[i,] <- r
        }
        if(data[i,]$pos == 'rb'){
          r <- as.matrix(res$rb[sample(nrow(res$rb),1, replace=T),c(4:18)])
          matx[i,] <- r
        }
        if(data[i,]$pos == 'wr'){
          r <- as.matrix(res$wr[sample(nrow(res$wr),1, replace=T),c(4:18)])
          matx[i,] <- r
        }
        if(data[i,]$pos == 'te'){
          r <- as.matrix(res$te[sample(nrow(res$te),1, replace=T),c(4:18)])
          matx[i,] <- r
        }
      }
      matx <- as.data.frame(matx)
      matx[,'pos'] <- data$pos
      matx[matx < 0] <- 0
      names(matx) <- c('fg','fga','xpt','pass_att','pass_cmp','pass_yds','pass_tds','pass_ints','rush_a
                       'fumbles','pos')



      val.i <- buildValuesTask3(matx,league)[,'value']
      return(val.i)

}
```

```
#Create a function `addNoise` that takes 4 arguments: a league object, a list of residuals, number of s
addNoise <- function(league, res, sim, seed){
  set.seed(seed)
  data <- league$stats
  pos <- league$posReq

  length <- length(returnValueVector(league,res))
  val <- matrix(NA, nrow = length, ncol=sim)

  for(j in 1:sim){
     val[,j] <- returnValueVector(league, res)
  }

  #val is the sims matrix
  attr(league, 'sims') <- val
  return(league)
}
```

```
#tests
#addNoise(l,finaldf,10,1)
```

```r
#1. Create a `quantile` method for the league class; it takes at least two arguments, a league object a
#it should fail if `sims` is not found. The `probs` vector should default to `c(0.25, 0.5, 0.75)`.  It

#**Note: I named the function quantile.league, since it uses the funciton quantile and I got confused w

quantile.league <- function(league, res, sim, probs = c(0.25,0.5,0.75)){
  data <- addNoise(league,res,sim,seed=1)
  sims.dat <- attributes(data)$sims
  returnMtx <- matrix(NA, nrow = nrow(sims.dat), ncol=length(probs))
  if(is.null(attributes(data)$sims) == TRUE){
    return("Requires `sims` element")
  }
  for(i in 1:nrow(sims.dat)){
    returnMtx[i,] <- quantile(attributes(data)$sims[i,],probs)
  }
  returnMtx
  colnames(returnMtx) <- probs
  returnMtx
  return(returnMtx)
}
```

```r
#test
#quantile.league(l,finaldf,10)
```

```r
#1. Create a function `conf.interval`; it takes at least two arguments, a league object and a probs vec
#fail if `sims` is not found. It should return a new object of type `league.conf.interval`.

# The new object will contain the output of `quantile`. However, results should be split by position an
#probability) descendingly.  Restrict the number of rows to the number of required players at each posi

league.conf.interval <- function(quantile){
  outputObj = list(stats = quantile)
  class(outputObj) = 'league.conf.interval'
  return(outputObj)
}
```

```r
conf.interval <- function(league, res, sim, probs=c(0.25,0.5,0.75)){
  data <- addNoise(league,res,sim,seed=1)

  if(is.null(attributes(data)$sims) == TRUE){
    return("Requires `sims` element")
  }

  x <- quantile.league(league, res, sim, probs)
  #I don't know how to get back the position column in the correct order since I didn't carry it throug
  #sims data
  position <- data$stats$pos

  returndata <- cbind(x,position)
  projected <- dataScale[order(dataScale$fpts),]
```

```
  returndata <- returndata[order(returndata[,ncol(returndata)-1]),]
  returndata <- as.data.frame(returndata)

  returndata2 <- split(returndata, returndata$position)

  #returndata
  class(returndata2) = 'league.conf.interval'
  returndata2
}
```

```
#test
#conf.interval(l,finaldf,10)
#class(conf.interval(l,finaldf,10))
```

```
#1. Create a `plot` method for the league.conf.interval class; it takes at least two arguments, a leagu
#each probability; using the defaults, you would have three lines (0.25, 0.5, 0.75). Add minimal plotti

#I think this would turn out the way it is supposed to if my coding ordered everything correctly

plot.league.conf.interval <- function(league.conf.interval, pos){
  data <- league.conf.interval

    plot(as.numeric(as.character(data[[pos]][,1])), type="l", col="red",xlab="Ranking",ylab="Value",mai
    lines(as.numeric(as.character(data[[pos]][,2])), lty=2, col="blue")
    lines(as.numeric(as.character(data[[pos]][,3])), lty=3, col="darkgreen")

}
```

Additional Tips

Use your best judgement in interpreting my instructions, and please do not hesitate to ask for clarification.

You have most of the code for tasks 1 and 2, it's a matter of restructuring it.

If you're stuck, explain your algorithm, why it fails, and move on. Attempt everything.