Bios 6301: Final Project

Hannah Weeks

12/14/2015

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.

Solution

```
setwd("~/Documents/BIOS 6301/Homework/Final")
##Question 1:

#Read in files
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)))

#Add column for position
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')
cols <- cols[c(1,2,19,6,3:5,7:14,16:18,15)]

# 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
football <- rbind(k[,cols], qb[,cols], rb[,cols], te[,cols], wr[,cols])


##Question 2
noBye <- c("CLE", "NO", "NYG", "PIT")
```

```r
#Separate data frame by whether or not team has had bye week
fbNoBye <- football[football$Team %in% noBye,]
fbBye <- football[!(football$Team %in% noBye),]

#Differentially apply multiplier based on if team has had bye week
for(i in 1:ncol(football)){
  if(class(football[,i]) == "numeric"){
    fbNoBye[,i] <- fbNoBye[,i]*(10/16)
    fbBye[,i] <- fbBye[,i]*(9/16)
  }
}

#Recombine data
football <- rbind(fbBye,fbNoBye)


##Question 3
#Sort decreasingly by fpts
football <- football[order(football$fpts, decreasing = TRUE),]

#Select the top players in each position
bestK <- head(football[football$pos == "k",], 20)
bestQB <- head(football[football$pos == "qb",], 20)
bestRB <- head(football[football$pos == "rb",], 40)
bestTE <- head(football[football$pos == "te",], 20)
bestWR <- head(football[football$pos == "wr",], 60)

#Recombine data
#Make sure everything is still in decreasing order by fpts
football <- rbind(bestK, bestQB, bestRB, bestTE, bestWR)
footballProj <- football[order(football$fpts, decreasing = TRUE),]



##Question 4
#Read in observed data
footballObs <- read.csv('nfl_current15.csv', header=TRUE, stringsAsFactors=FALSE)
names(footballObs)[1] <- "PlayerName"


##Question 5
#Merge projected and observed datasets by player name
footballMerge <- (merge(footballProj, footballObs, by = "PlayerName", all.x = TRUE))
footballMerge[is.na(footballMerge)] <- 0


##Question 6
#Initialize difference matrix
fbDiff <- as.data.frame(matrix(0, ncol = 17, nrow = 160))
colnames(fbDiff) <- c("PlayerName", "Position", "field.goals", "field.goals.attempted", "extra.points",
             "passing.attempts", "passing.completions", "passing.yards", "passing.touchdowns",
             "passing.interceptions", "rushing.attempts", "rushing.yards", "rushing.touchdowns",
             "receiving.attempts", "receiving.yards", "receiving.touchdowns", "fumbles")
```

```r
#Set player names and positions
fbDiff$PlayerName <- footballMerge$PlayerName
fbDiff$Position <- footballMerge$pos

#Obs-Proj
fbDiff$field.goals <- footballMerge$FGM - footballMerge$fg
fbDiff$field.goals.attempted <- footballMerge$FGA - footballMerge$fga
fbDiff$extra.points <- footballMerge$XPM - footballMerge$xpt
fbDiff$passing.attempts <- footballMerge$Att.pass - footballMerge$pass_att
fbDiff$passing.completions <- footballMerge$Cmp.pass - footballMerge$pass_cmp
fbDiff$passing.yards <- footballMerge$Yds.pass - footballMerge$pass_yds
fbDiff$passing.touchdowns <- footballMerge$TD.pass - footballMerge$pass_tds
fbDiff$passing.interceptions <- footballMerge$Int.pass - footballMerge$pass_ints
fbDiff$rushing.attempts <- footballMerge$Att.rush - footballMerge$rush_att
fbDiff$rushing.yards <- footballMerge$Yds.rush - footballMerge$rush_yds
fbDiff$rushing.touchdowns <- footballMerge$TD.rush - footballMerge$rush_tds
fbDiff$receiving.attempts <- footballMerge$Rec.catch - footballMerge$rec_att
fbDiff$receiving.yards <- footballMerge$Yds.catch - footballMerge$rec_yds
fbDiff$receiving.touchdowns <- footballMerge$TD.catch - footballMerge$rec_tds
fbDiff$fumbles <- footballMerge$Fmb - footballMerge$fumbles

#Split data by position
pos <- c("k", "qb", "rb", "te", "wr")
kicker <- fbDiff[fbDiff$Position == "k",]
quarterBack <- fbDiff[fbDiff$Position == "qb",]
runningBack <- fbDiff[fbDiff$Position == "rb",]
tightEnd <- fbDiff[fbDiff$Position == "te",]
wideReceiver <- fbDiff[fbDiff$Position == "wr",]

#Recombine data into list, removing player name and position
fbList <- list(kicker[,-(1:2)], quarterBack[,-(1:2)], runningBack[,-(1:2)], tightEnd[,-(1:2)], wideRece
names(fbList) <- c("k", "qb", "rb", "te", "wr")
```

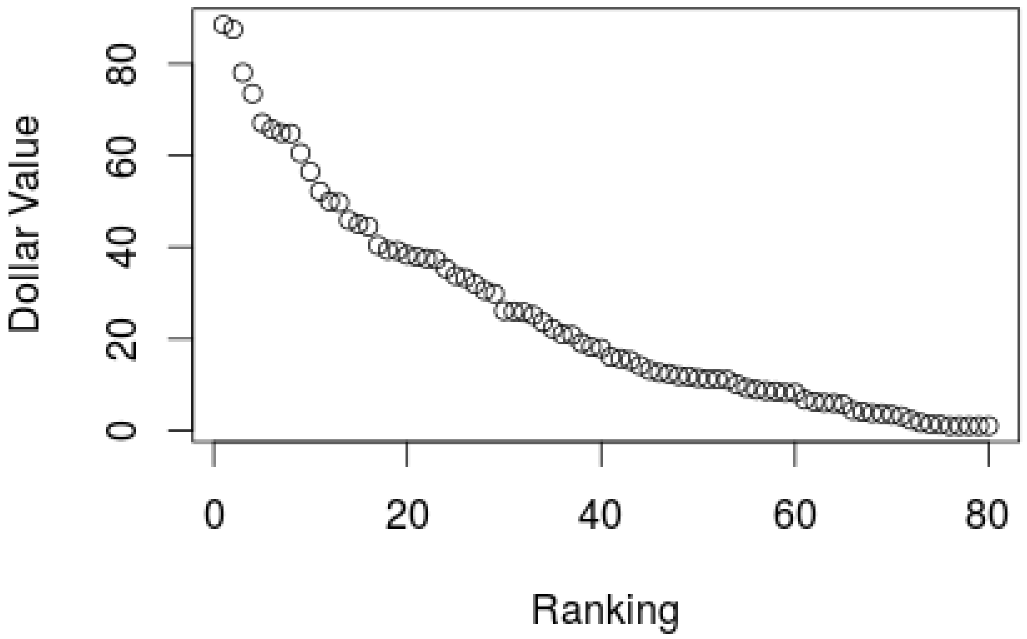
Task 2: Creating League S3 Class (80 points)

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.

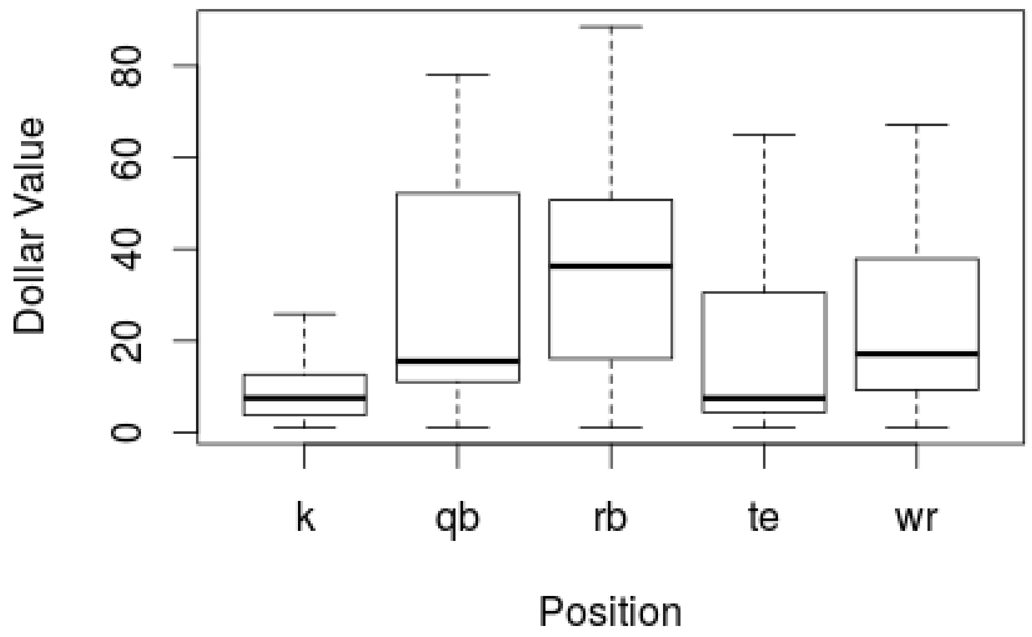
   As an example if a league has ten teams and requires one kicker, the tenth best kicker should be worth \$1. All kickers with points less than the 10th kicker should have dollar values of \$0.
4. Create a `print` method for the league class. It should print the players and dollar values (you may choose 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:



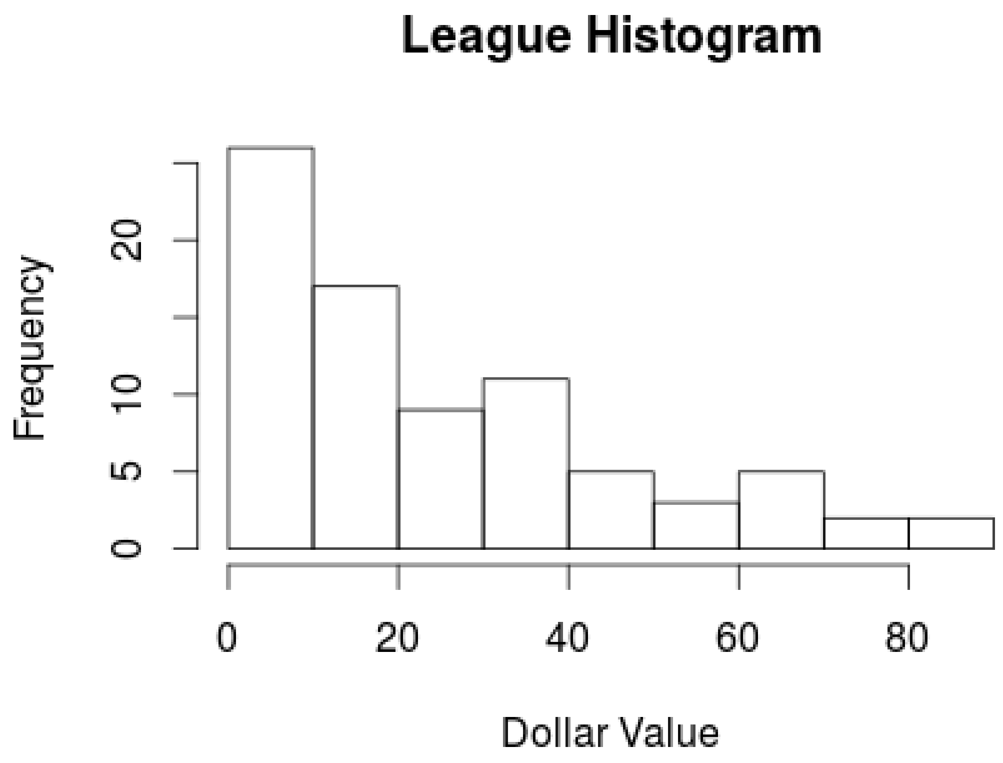
6. Create a `boxplot` method for the league class. Add minimal plotting decorations.

    - 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`.

Solution

I used the following code to test my functions. For the projected data, I used my `footballProj` data frame from Task 1.

```
posVec <-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,
l <-league(stats=footballProj, nTeams=10, cap=200, posReq=pos, points=pnts)
```

```
##Question 1
#Function that creates a league object
league <- function(stats, nTeams, cap, posReq, points){
  #Make league object
  x <- list(stats, nTeams, cap, posReq, points)

  #Assign attributes
  attr(x, "Stats") <- stats
  attr(x, "NumberOfTeams") <- nTeams
  attr(x, "MoneyAvailablePerTeam") <- cap
  attr(x, "PositionsRequired") <- posReq
  attr(x, "Points") <- points

  #Create S3 class
  class(x) <- "league"
  return(x)
}


##Question 2
#Function to calculate points
calcPoints <- function(league){
  x <- attr(league, "Stats")
  points <- unlist(attr(league, "Points"))

  # convert NFL stat to fantasy points
  x[,'p_fg'] <- x[,'fg']*points["fg"]
  x[,'p_xpt'] <- x[,'xpt']*points["xpt"]
  x[,'p_pass_yds'] <- x[,'pass_yds']*points["pass_yds"]
  x[,'p_pass_tds'] <- x[,'pass_tds']*points["pass_tds"]
  x[,'p_pass_ints'] <- x[,'pass_ints']*points["pass_ints"]
  x[,'p_rush_yds'] <- x[,'rush_yds']*points["rush_yds"]
  x[,'p_rush_tds'] <- x[,'rush_tds']*points["rush_tds"]
  x[,'p_fumbles'] <- x[,'fumbles']*points["fumbles"]
  x[,'p_rec_yds'] <- x[,'rec_yds']*points["rec_yds"]
  x[,'p_rec_tds'] <- x[,'rec_tds']*points["rec_tds"]

  # sum selected column values for every row
  # this is total fantasy points for each player
  x[,'points'] <- rowSums(x[,grep("^p_", names(x))])

  #Update stats attribute with points data
  attr(l, "Stats") <- x
  return(l)
}
```

```r
##Question 3
#Function to make dollar values
buildValues <- function(league){
  x <- attr(league, "Stats")

  #Attribute values to be used within function
  posReq <- unlist(attr(league, "PositionsRequired"))
  nTeams <- attr(league, "NumberOfTeams")
  cap <- attr(league, "MoneyAvailablePerTeam")

  #Make sure fantasy points have been calculated
  if(!("points" %in% names(x))){
    stop("Invalid league object. Calculate points first.")
  } else{

    # 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')

    # calculate marginal points by subtracting "baseline" player's points
    x2[k.ix, 'marg'] <- x2[k.ix,'points'] - x2[k.ix[nTeams*posReq["k"]],'points']
    x2[qb.ix, 'marg'] <- x2[qb.ix,'points'] - x2[qb.ix[nTeams*posReq["qb"]],'points']
    x2[rb.ix, 'marg'] <- x2[rb.ix,'points'] - x2[rb.ix[nTeams*posReq["rb"]],'points']
    x2[te.ix, 'marg'] <- x2[te.ix,'points'] - x2[te.ix[nTeams*posReq["te"]],'points']
    x2[wr.ix, 'marg'] <- x2[wr.ix,'points'] - x2[wr.ix[nTeams*posReq["wr"]],'points']

    # create a new data.frame subset by non-negative marginal points
    x3 <- x2
    # re-order by marginal points
    x3 <- x3[order(x3[,'marg'], decreasing=TRUE),]

    # reset the row names
    rownames(x3) <- NULL

    # calculation for player value
    x3[,'value'] <- x3[,'marg']*(nTeams*cap-nrow(x3))/sum(x3[,'marg']) + 1

    # create a data.frame with more interesting columns
    x4 <- x3[,c('PlayerName','pos','points','marg','value')]

    #Update stats attribute
    attr(l, "Stats") <- x4
    return(l)
  }
}
```

```r
##Question 4
#Print method for league
print.league <- function(league){
  stats <- attr(league, "Stats")

  #Nothing to print if points not calculated
  if(!("points" %in% names(stats))){
    print("Points not calculated")
  } else if(!("value" %in% names(stats))){
    #If points but no dollar values, print points
    print(stats[c("PlayerName", "points")])
  } else{

    #If dollar values calculated, only print players with positive values
    stats <- stats[stats$value > 0,]
    dollarVals <- stats[c("PlayerName", "value")]
    print(dollarVals)
  }
}

##Question 5
#Plot method for league
plot.league <- function(league){
  stats <- attr(league, "Stats")

  #Need dollar values for plotting
  if(!("points" %in% names(stats))){
    stop("Invalid league object. Calculate points first.")
  } else if(!("value" %in% names(stats))){
    stop("Invalid league object. Build values first.")
  } else{

    #Plot players with positive dollar values
    stats <- stats[stats$value > 0,]
    plot(1:nrow(stats), stats$value, main = "League Values by Ranking", xlab = "Ranking", ylab = "Dolla
  }
}


#Question 6
#Boxplot method for league
boxplot.league <- function(league){
  stats <- attr(league, "Stats")

  #Need dollar values for plotting
  if(!("points" %in% names(stats))){
    stop("Invalid league object. Calculate points first.")
  } else if(!("value" %in% names(stats))){
    stop("Invalid league object. Build values first.")
  } else{
  stats <- stats[stats$value > 0,]

  #Splot stats by position
  k <- stats[stats$pos == "k",]
  qb <- stats[stats$pos == "qb",]
  rb <- stats[stats$pos == "rb",]
  te <- stats[stats$pos == "te",]
  wr <- stats[stats$pos == "wr",]

  #boxplot for each position
  boxplot(k$value, qb$value, rb$value, te$value, wr$value, names = c("k", "qb", "rb", "te", "wr") ,main
  }
}

##Question 7
#Histogram method for league
hist.league <- function(league){
  stats <- attr(league, "Stats")

  #Need dollar values for plotting
  if(!("points" %in% names(stats))){
    stop("Invalid league object. Calculate points first.")
  } else if(!("value" %in% names(stats))){
    stop("Invalid league object. Build values first.")
  } else{

    #histogram for positive dollar values
    stats <- stats[stats$value > 0,]
    hist(stats$value, main = "League Values", xlab = "Dollar Values")
  }
}
```

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.

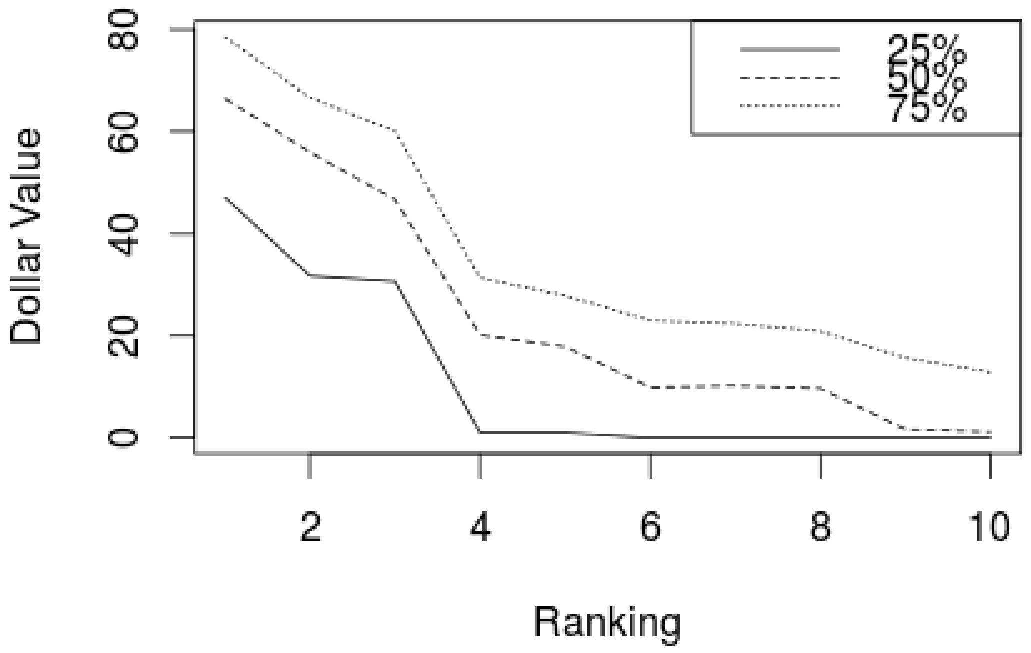
2. Create a `quantile` method for the league class; it takes at least two arguments, a league object and a probs vector. This method requires the `sims` element; it should fail if `sims` is not found. The `probs` vector should default to `c(0.25, 0.5, 0.75)`. It should run `quantile` on the dollar values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

Solution

```r
#These are how I tested my functions
# l <-league(stats=footballProj, nTeams=10, cap=200, posReq=posVec, points=pnts)
# l <- addNoise(league=l, nsims=100)

##Question 1
#Function to add residuals to stats
addNoise <- function(league, res = fbList, nsims = 1, seed = NULL){
  set.seed(seed)

  #Initialize objects to be used in function
  pos <- c("k", "qb", "rb", "te", "wr")
  stats <- attr(league, "Stats")
  s <- split(stats, stats$pos)
  dollarMatrix <- matrix(nrow = nrow(stats),ncol = nsims)

  #Create dollar values for each simulation
  for(j in 1:nsims){
    test <- league
    noiseList <- as.list(rep(NA, length(pos)))
    names(noiseList) <- pos

    #For each player, sample a row from that position's residuals
    #Add sampled residuals to player's values
    for(i in 1:5){
      sNum <- s[[i]][,5:19]
      noised <- apply(sNum, 1,
        function(x) x + res[[i]][sample(nrow(res[[i]]),1),])

      noiseList[[i]] <- do.call("rbind", noised)
      noiseList[[i]][,'pos'] <- pos[i]
    }

    #Set negative stats to zero
    dfNoise <- do.call("rbind", noiseList)
    dfNoise[dfNoise < 0] <- 0
    row.names(dfNoise) <- NULL

    #Update stats with noised data
    attr(test, "Stats")[,5:19] <- dfNoise[-16]

    #Get points and dollar values
    test <- calcPoints(test)
    test <- buildValues(test)
    vals <- attr(test, "Stats")$value

    #Add dollar values to matrix
    dollarMatrix[,j] <- vals
  }
  #Add sims attribute to league object
  attr(league, "sims") <- dollarMatrix
}
```

```r
##Question 2
#Quantile method for league
quantile.league <- function(league, probs = c(0.25, 0.5, 0.75)){
  #Require sims attribute
  if(!("sims" %in% names(attributes(league)))){
    stop("Invalid league object. Add noise first.")
  } else{
    sims <- attr(league, "sims")

    #Run quantile on dollar values for each player
    quantileMatrix <- apply(sims, 1, function(x) quantile(x, probs = probs))
    quantileMatrix <- t(quantileMatrix)
    return(quantileMatrix)
  }
}


##Question 3
#Function to create league.conf.interval object
conf.interval <- function(league, probs = c(0.25, 0.5, 0.75), ...){
  #Require sims attribute
  if(!("sims" %in% names(attributes(league)))){
    stop("Invalid league object. Add noise first.")
  } else{
    #Values to be used in function
    positions <- attr(league, "Stats")$pos
    sims <- attr(league, "Sims")
    q <- quantile(league)
    qPos <- data.frame(positions, q)

    #Order by highest percentile column
    qPos <- qPos[order(qPos[,length(probs)], decreasing=TRUE),]

    #Restrict to number of positions required times number of teams
    posReq <- unlist(attr(league, "PositionsRequired"))
    nTeams <- attr(league, "NumberOfTeams")

    numK <-head(qPos[qPos$pos == "k",], posReq["k"]*nTeams)
    numQB <-head(qPos[qPos$pos == "qb",], posReq["qb"]*nTeams)
    numRB <-head(qPos[qPos$pos == "rb",], posReq["rb"]*nTeams)
    numTE <-head(qPos[qPos$pos == "te",], posReq["te"]*nTeams)
    numWR <-head(qPos[qPos$pos == "wr",], posReq["wr"]*nTeams)

    #Recombine data
    lci <- list(numK, numQB, numRB, numTE, numWR)
    names(lci) <- pos

    class(lci) <- "league.conf.interval"
    return(lci)
  }
}
```

```r
##Question 4
#Plot method for league.conf.interval
plot.league.conf.interval <- function(leagueCI, position, ...){
  pos <- c("k", "qb", "rb", "te", "wr")
  posNames <- c("Kicker", "Quarterback", "Runningback", "Tight.End", "Wide.Receiver")


  for(i in 1:5){
    #Run plot for specified position
    if(position == pos[i]){
      toPlot <- lci[[i]]

      #Get quantile names for legend
      qNames <- names(toPlot)[-1]
      qNames <- sub("^X", "", qNames)
      posTitle <- c("Dollar Values for", posNames[i])

      #Find maximum dollar value for axis limits
      ymax <- max(toPlot[,2:ncol(toPlot)])

      #Plot lines for each quantile
      plot(toPlot[,2], type='l', xlab = "Ranking", ylab = "Dollar Value", main = posTitle, ylim = c(1,y
      for(i in 3:ncol(toPlot)){
        lines(toPlot[,i], lty = i-1)
      }
      #Add legend
      legend('topright', title = "Percentiles", legend = qNames, lty=1:ncol(toPlot), cex = .7)
    }
  }
}
```

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.