Bios 6301: Final Project

Nick Strayer

25 November, 2015

Task 1: Finding Residuals

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

Task 1: Solution

part 1

```
fileNames = list.files("data/2015/")[2:length(list.files())]
positions = c("k", "qb", "rb", "te", "wr")
df = NULL
for(i in 1:length(fileNames)){
  f = read.csv(paste0("data/2015/",fileNames[i]), stringsAsFactors = F)
  f$position = positions[i]
  df = bind_rows(df, f) # a little dyplr magic.
}
```

part 2

```
noBye = c("CLE", "NO", "NYG", "PIT")
multipliers = ifelse(df$Team %in% noBye, (10/16), (9/16)) #vector for score multiplication

stats = names(df)[3:length(names(df))] #the numeric columns
stats = stats[-which(stats == "position")] #gotta get rid of the position column, too
for(stat in stats) df[,stat] = df[,stat] * multipliers #do the multiplication
```

part 3

```
#sort by fpts
df = df[order(-df$fpts),]

top_df = NULL #initialize holder for this new dataframe
posNums = c(20,20,40,20,60)
#           k  qb rb te wr  This order matches our position vector

for(i in 1:length(positions)){
  #get df with just that position,
  sub_df = df[df$position == positions[i],]

  #take only the first posNum[i] rows
  sub_df = sub_df[1:posNums[i],]

  #rbind that with the other stuff.
  top_df = bind_rows(top_df, sub_df)
}
```

part 4

```
nfl_current = read.csv("data/2015/nfl_current15.csv")
```

part 5

```
# get vector of player names we want.
topPlayers = top_df$PlayerName

# subset the current dataset to only include those rows
```

```
current_sub = nfl_current[nfl_current$Name %in% topPlayers, ]
current_sub$PlayerName = current_sub$Name #rename to merge on.

# Merge with the top_df
df_pred_obs = merge(top_df, current_sub, by = "PlayerName", all = T)
df_pred_obs[df_pred_obs == NA] <- 0 #Turn all my NAs into Os.
```

part 6

```
Name = c('field goals','field goals attempted','extra points','passing attempts','passing completions',
                      'passing yards','passing touchdowns','passing interceptions','rushing attempts','
                      'rushing touchdowns','receiving attempts','receiving yards','receiving touchdowns

projected_col = c('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("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")

#make a vector that is the difference between the proj[i] and obs[i] for the whole dataframe
for(i in 1:length(Name)) df_pred_obs[,Name[i]] = df_pred_obs[,observed_col[i]] - df_pred_obs[,projected_

#break into a list by position.
byPosition = lapply(positions, function(n){ df_pred_obs[df_pred_obs$position == n, Name]  })
names(byPosition) = positions #name the list entries.
```

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

Task 2: Solutions

part 1

```
league = function(stats, nTeams, cap, posReq, points){
  obj = list(stats = stats, nTeams = nTeams, cap = cap, posReq = posReq, points = points)
  class(obj) = 'league' #give it the class
  return(obj)
}
```

part 2

```
calcPoints <- function(d){

  #create a temporary clone of the stats object
  tmp = d$stats
  tmp[is.na(tmp)] = 0

  #grab the names of the stats to be used for calculating points
  pnts_names = names(d$points)

  #take each row present in the pnts list and multiply by given scaler
  for(stat in pnts_names) tmp[,stat] = tmp[,stat] * pnts[stat]

  #take sum of each row and make a new vector called points return this with the stats dataframe
  d$stats[,"points"] = rowSums(tmp[,pnts_names])
  d
}
```

part 3

```
buildValues <- function(d){

  x = d$stats

  # create new data.frame ordered by points descendingly
  df = x[order(-x$points),]

  df[, 'marg'] = 0 #initialize the marginal column

  # calculate marginal points by subtracting "baseline" player's points
  for(pos in names(d$posReq)) {
    ix <- which(df[,'position'] == pos)

    baseline <- as.numeric(d$posReq[pos])*d$nTeams

    if(baseline == 0){
      df[ix, 'marg'] = -1
    } else{
      df[ix, 'marg'] = df[ix,'points'] - as.numeric(df[ix[baseline],'points'])
    }
  }
```

```r
  # create a new data.frame subset by non-negative marginal points
  df_sub <- df[df[,'marg'] >= 0,]

  # calculation for player value
  df_sub[,'value'] <- df_sub[,'marg']*(d$nTeams*d$cap-nrow(df_sub))/sum(df_sub[,'marg']) + 1
  d$stats = df_sub
  d
}
```

part 4

```r
print.league = function(d){
  players = d$stats[,c("PlayerName", "position", "value")]
  names(players) = c("Name", "Position", "Value")
  kable(players)
}
```

part 5

```r
plot.league = function(d){
  #grab part of object we want
  df = d$stats
  #sort results
  df = df[order(-df$value), ]
  #add rank column
  df$rank = seq(length(df$value))
  #plot it!
  ggplot(df, aes(x = rank, y = value)) + geom_point(color = "steelblue") + theme_bw() +
    labs(x = "Ranking", y = "Dollar Value", title = "Player Value by Rank")
}
```

part 6

```r
boxplot.league = function(d){
  ggplot(d$stats, aes(x = factor(position), y = value)) +
    theme_bw() + geom_boxplot(fill = "steelblue") +
    labs(x = "Position", y = "Dollar Value", title = "Spread of Values by Position")
}
```

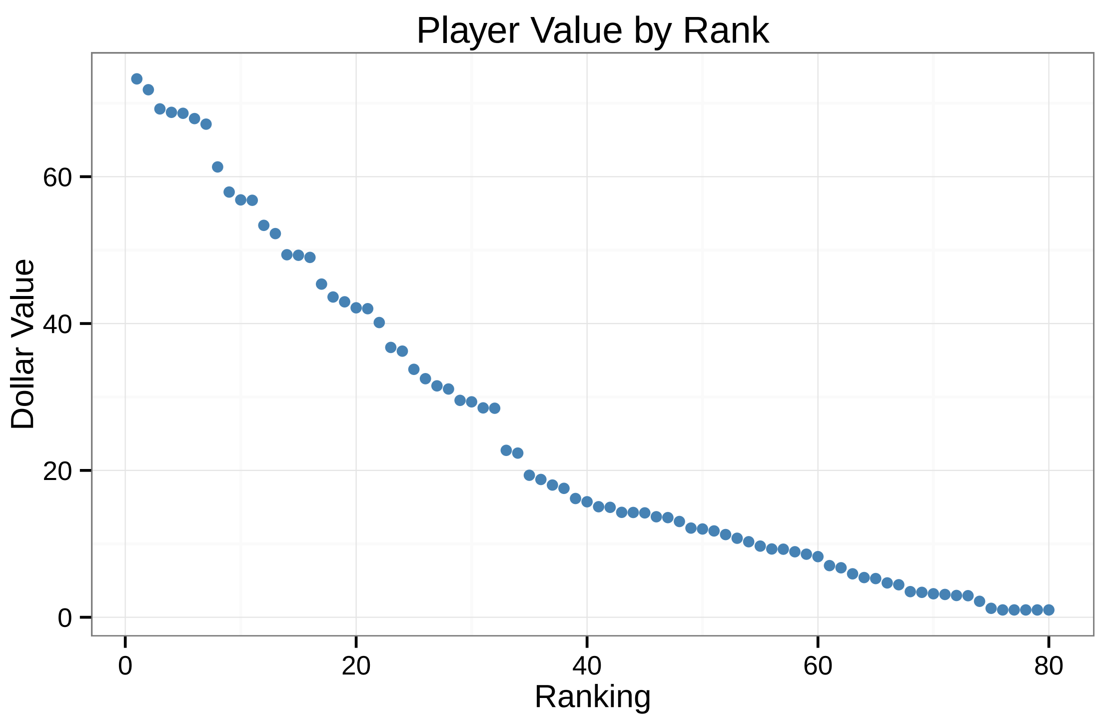
part 7

```r
hist.league = function(d){
  hist(d$stats$value, col = "steelblue", main = "Distribution of Values",
       xlab = "Dollar Value", ylab = "Frequency")
}
```

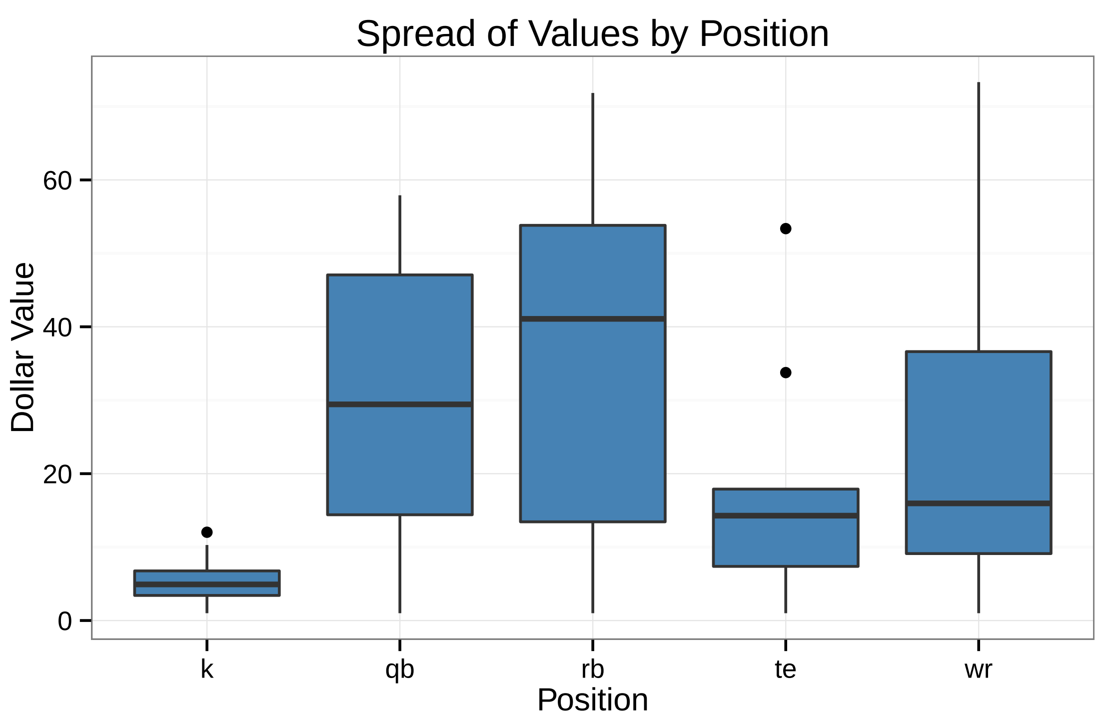
test code

```r
x = top_df #my x is the dataframe top_df
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)
```

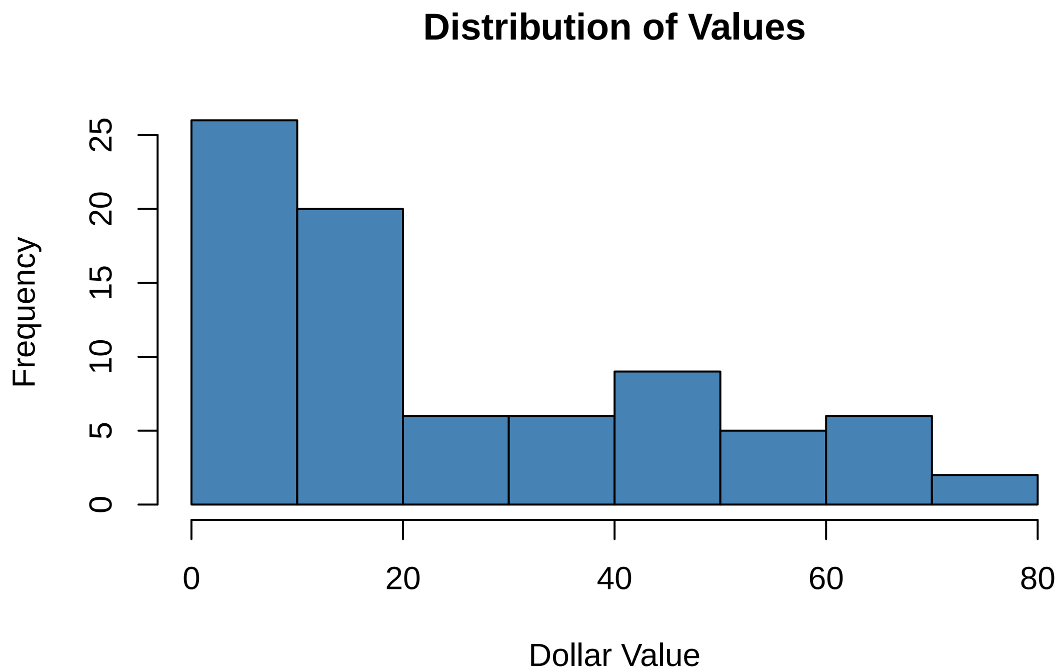
```r
l <- league(stats=x, nTeams=10, cap=200, posReq=pos, points=pnts)
l <- calcPoints(l)
l <- buildValues(l)
plot(l)
```




```r
boxplot(l)
```



```r
hist(l)
```



Task 3: Simulations with Residuals

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.

Task 3: Solution

part 1

```
addNoise = function(obj, resids, numOfSims = 100, seed = 8){
  set.seed(seed)
  sims = NULL #holder for the simulation data.

  for(j in 1:numOfSims){
    newObj = obj #clone the league object for this simulation
    stats = newObj$stats #grab the stats section.

    #Going down players in stats df
    for(i in 1:length(stats$PlayerName)){

      #Given the original projection for the position of given row
      currentPos = as.character(stats[i, "position"])

      #sample one value between 1 and the number of that position
      sampledVal = sample(1:dim(resids[[currentPos]])[1], 1)

      #grab that row from the resids dataframe
      sampledRow = resids[[currentPos]][sampledVal,]
      rowNames   = names(sampledRow)

      #Add the 15 columns from the sampled row to the 15 columns for the first position pick
      stats[i,projected_col] = stats[i,projected_col] + sampledRow
    }

    #Note that stats can't be negative so replace any negative values with 0.
    stats[stats < 0] = 0

    #take dollar values from this run and add it to a growing sims matrix
    newObj$stats = stats
    newObj <- buildValues(calcPoints(newObj))
    #grab values
    vals = newObj$stats$value

    #add it to sims.
    sims = cbind(sims, vals)
  }
```

```
  obj$sims = sims #attach the simulation matrix
  obj #return the object with it's shiny new simulations
}
```

part 2

```
quantile.league = function(obj, probs = c(0.25, 0.5, 0.75)){
  if(is.null(obj$sims)){
    print("You forgot to run the simulations!")
  } else{ #the simulations have already been run, let's do stuff.

    #grab the list of player names from the stats object
    names = obj$stats$PlayerName

    #set up a matrix or df to hold the results
    res_mat = matrix(length(names),length(probs) + 1)

    #take the quantile for each row of the sims
    res_list = lapply(names, function(d){quantile(obj$sims[which(names == d), ], probs = probs)})

    #package them
    res_df = do.call(rbind.data.frame, res_list)
    res_df$names = names
    names(res_df) = c(probs,"names")
    res_df[,c("names", probs)] #reorder on return
  }
}
```

part 3

```
conf.interval = function(obj, probs = c(0.25, 0.5, 0.75)){
  #test for the sims element
  if(is.null(obj$sims)){
    print("You forgot to run the simulations!")
  } else{
    #run the quantile function:
    quants = quantile(obj, probs)

    #grab the player positions and append to the quants df
    quants[,"position"] = obj$stats[,"position"]

    #sort by last column
    lastCol = as.character(probs[length(probs)])
    quants = quants[order(-quants[lastCol]),]

    #split the df by position
    splitDf = split(quants, quants$position)

    #only take the required number of players per position
    returnDf = lapply(names(obj$posReq), function(pos){
      splitDf[[pos]][seq(obj$posReq[[pos]] * obj$nTeams),] #only grab the first n of the position
    })
    #return list of dataframes
```

```
    names(returnDf) = names(obj$posReq)
    class(returnDf) = "league.conf.interval"
    returnDf
  }
}
```

part 4

```
plot.league.conf.interval = function(obj, position){
  #grab the position we are looking at out of the list
  posDf      = obj[[position]]
  #generate the rankings
  posDf$rank = seq(length(posDf$names))
  #remove all the columns but rank and the probs for the intervals.
  posDf      = posDf[c(-1, -(dim(posDf)[2]-1))]
  plotDf     = melt(posDf, id = "rank")
  names(plotDf) = c("Rank", "Probability", "Dollar Value")

  ggplot(plotDf, aes(x = Rank, y = `Dollar Value`, color = Probability)) +
    geom_line() + theme_bw()
}
```

Test code:

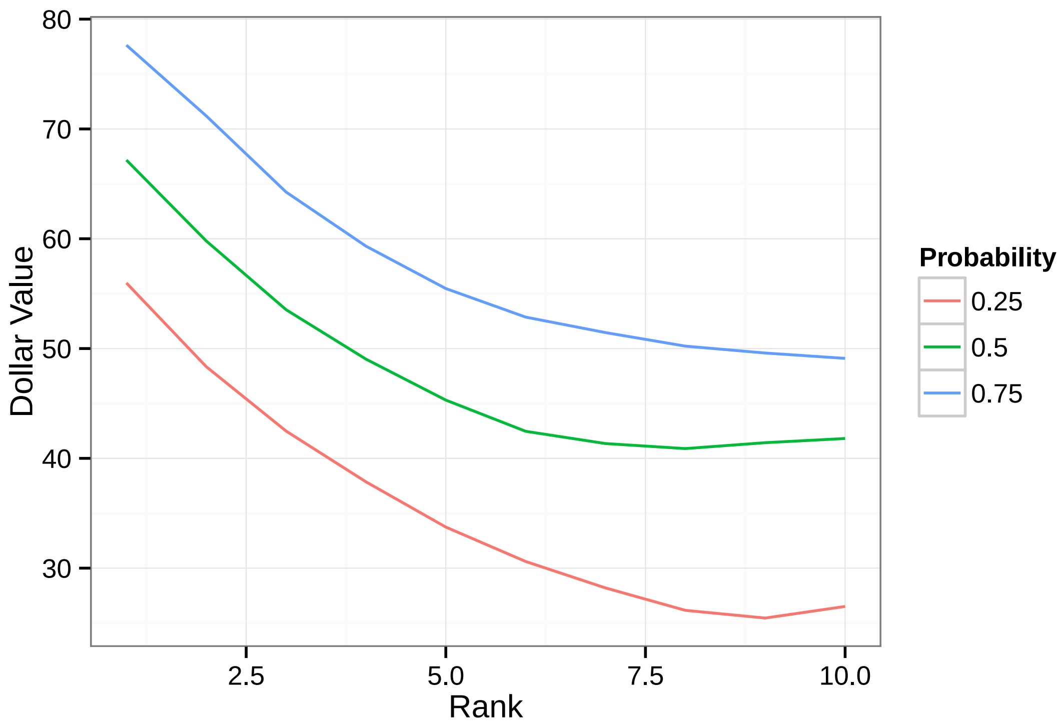
```
#the noise vector
noise = byPosition

l1 <- addNoise(l, noise, 10000)
quantile(l1)
```

```
##                  names      0.25       0.5      0.75
## 1           Drew Brees 55.974622 67.167563 77.631638
## 2         Andrew Luck 48.351400 59.798486 71.185841
## 3       Aaron Rodgers 42.481777 53.536074 64.249524
## 4      Russell Wilson 37.861038 49.035018 59.325506
## 5      Peyton Manning 33.739730 45.304678 55.462868
## 6  Ben Roethlisberger 30.606473 42.463205 52.861998
## 7            Matt Ryan 28.198449 41.346614 51.454972
## 8          Eli Manning 26.157738 40.890133 50.220310
## 9           Cam Newton 25.457131 41.426915 49.592167
## 10           Tony Romo 26.512551 41.807851 49.104697
## 11      Marshawn Lynch 30.602324 42.221588 49.041855
## 12        Le'Veon Bell 32.330153 42.036665 48.892167
## 13     Adrian Peterson 32.939384 41.724515 48.877422
## 14          Eddie Lacy 32.186592 41.020250 48.471218
## 15      Jamaal Charles 31.671294 39.878636 47.828377
## 16       C.J. Anderson 30.887841 38.925382 46.998654
## 17          Matt Forte 30.085605 37.728982 45.862392
## 18        LeSean McCoy 29.362231 36.675045 44.451841
## 19      DeMarco Murray 28.836226 35.959547 43.511722
## 20         Jeremy Hill 28.283847 35.392666 42.772991
## 21         Mark Ingram 27.370096 34.485118 41.720974
```

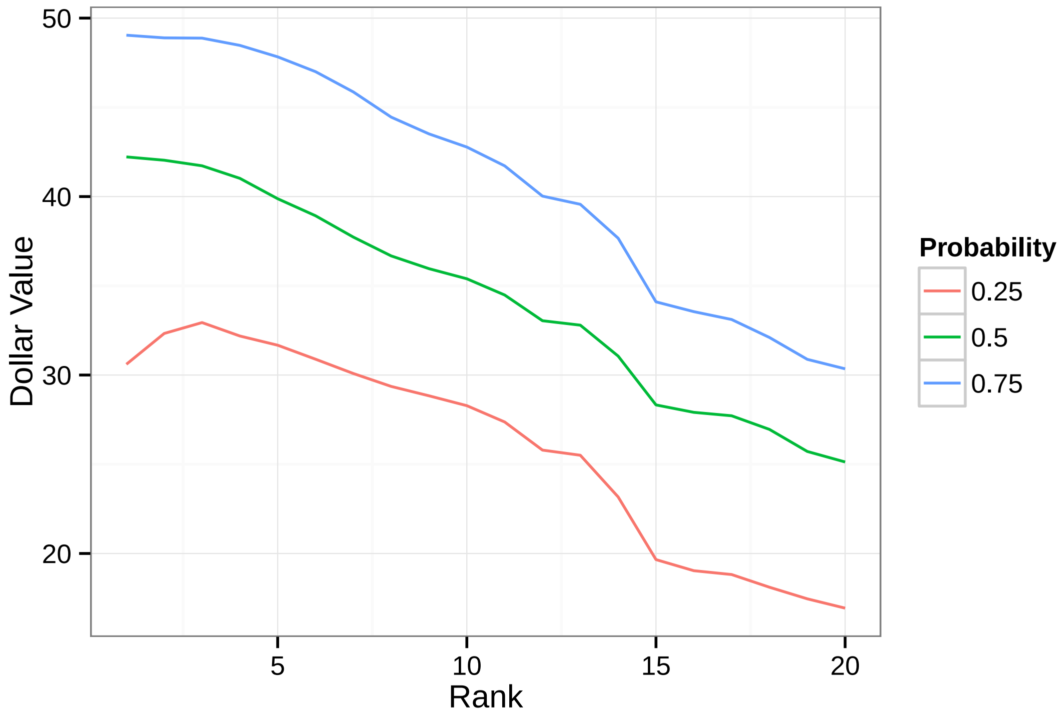
```
## 22 Stephen Gostkowski 26.804025 34.023467 41.237781
## 23    Garrett Hartley 26.653816 33.687618 40.721731
## 24       Lamar Miller 25.793496 33.042987 40.024926
## 25     Justin Forsett 25.505925 32.794725 39.567009
## 26      Justin Tucker 24.993039 32.614811 39.389384
## 27         Josh Brown 24.372394 32.384723 38.802943
## 28    Steven Hauschka 23.756733 31.926715 38.287582
## 29      Alfred Morris 23.169777 31.060599 37.668098
## 30     Dustin Hopkins 22.375052 30.571164 37.045291
## 31        Cody Parkey 21.825853 30.199601 36.482770
## 32       Connor Barth 21.467048 29.615198 35.909222
## 33       Mason Crosby 20.813675 29.247376 35.309424
## 34     Adam Vinatieri 20.117565 28.893463 34.749859
## 35      Melvin Gordon 19.657549 28.325103 34.100473
## 36        Carlos Hyde 19.036648 27.910173 33.551727
## 37         Frank Gore 18.821602 27.716025 33.111591
## 38      Antonio Brown 18.464766 27.541156 32.673877
## 39    Latavius Murray 18.110091 26.953340 32.104766
## 40  Odell Beckham Jr. 17.672078 26.248890 31.474719
## 41      Joseph Randle 17.460197 25.717547 30.879449
## 42    Rashad Jennings 16.938936 25.128528 30.348351
## 43   Demaryius Thomas 16.677827 24.798719 29.820502
## 44         Dez Bryant 16.273583 24.229799 29.111757
## 45     Calvin Johnson 15.981273 23.655380 28.553180
## 46       Randall Cobb 15.804315 23.453798 27.976221
## 47        Julio Jones 15.939420 23.222482 27.572828
## 48     Rob Gronkowski 15.738775 23.060334 27.124974
## 49     Alshon Jeffery 15.261141 22.556274 26.433283
## 50         A.J. Green 15.472672 22.295332 26.014113
## 51         Mike Evans 15.462247 21.970948 25.643755
## 52      Brandin Cooks 15.426688 21.809200 25.191321
## 53   Emmanuel Sanders 15.065997 21.275589 24.690515
## 54        T.Y. Hilton 14.536248 20.779025 24.131744
## 55    Jordan Matthews 14.159322 20.319649 23.597555
## 56       Jimmy Graham 13.383380 19.580256 22.986521
## 57    Martavis Bryant 13.100949 19.042311 22.307708
## 58    DeAndre Hopkins 12.572470 18.416790 21.791149
## 59     Julian Edelman 11.886456 17.666496 20.964738
## 60      Andre Johnson 11.202344 16.889527 20.292907
## 61     DeSean Jackson 10.511040 16.152266 19.597645
## 62      Davante Adams  9.936893 15.544593 18.888166
## 63      Sammy Watkins  9.537744 14.898058 18.176512
## 64        Golden Tate  8.849038 14.060383 17.477135
## 65      Jeremy Maclin  8.302995 13.397485 16.741907
## 66       Keenan Allen  7.783168 12.640240 16.031196
## 67   Brandon Marshall  7.339363 12.204224 15.445989
## 68    Marques Colston  6.854730 11.628188 14.856275
## 69       Mike Wallace  6.313410 11.004606 14.155460
## 70    Vincent Jackson  5.847096 10.431987 13.544934
## 71       Amari Cooper  5.430084  9.947573 12.997397
## 72         Greg Olsen  4.596678  9.197359 12.360377
## 73       Travis Kelce  3.982988  8.466829 11.645865
## 74        Eric Decker  2.658761  7.453001 10.766421
## 75  Martellus Bennett  1.959822  6.642343  9.659627
```

```
## 76        Jason Witten  1.000000  5.379553  8.549904
## 77       Julius Thomas  1.000000  3.670212  7.244094
## 78        Dwayne Allen  1.000000  1.331379  5.048344
## 79           Zach Ertz  1.000000  1.000000  1.377775
## 80        Coby Fleener  1.000000  1.000000  1.000000
```

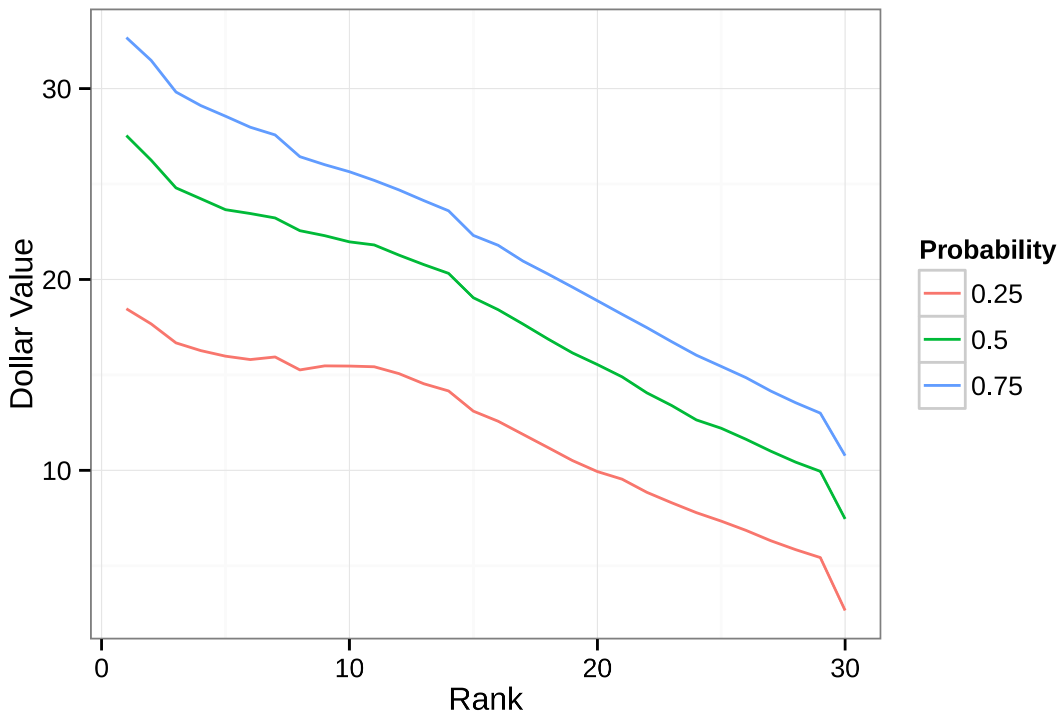
```
ci <- conf.interval(l1)
plot(ci, 'qb')
```



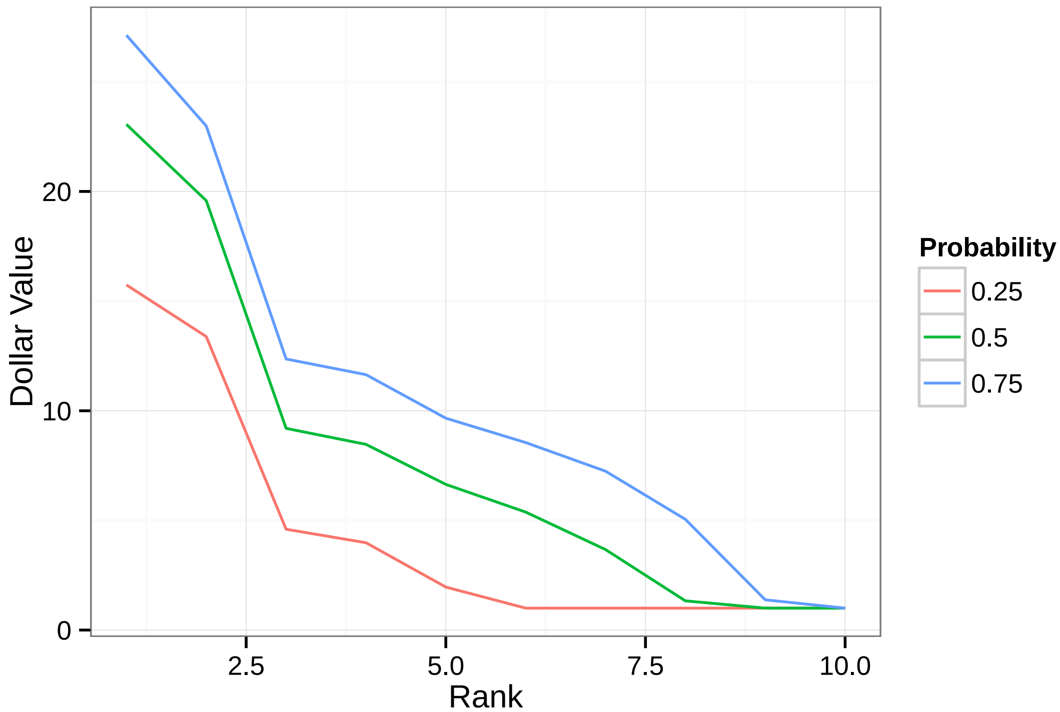

```
plot(ci, 'rb')
```



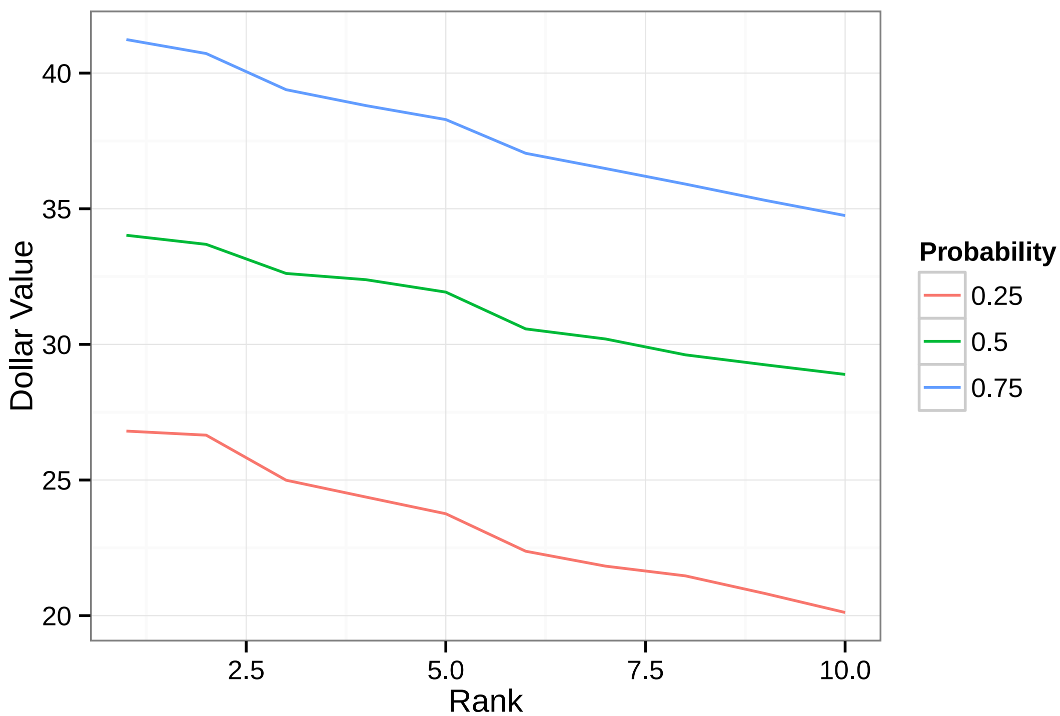

```
plot(ci, 'wr')
```




```
plot(ci, 'te')
```


Dollar Value
20
10
0
2.5
5.0
7.5
10.0
Rank
Probability
0.25
0.5
0.75

```
plot(ci, 'k')
```


Dollar Value
40
35
30
25
20
2.5
5.0
7.5
10.0
Rank
Probability
0.25
0.5
0.75