Bios 6301: Final

Alexander Thiemicke

2015-12-15

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.

```r
#1. Read and combine the projection data (five files) into one data set, adding a position column.
   wr <- read.csv('proj_wr15.csv')
   wr['position'] <- 'wr'
   te <- read.csv('proj_te15.csv')
   te['position'] <- 'te'
   rb <- read.csv('proj_rb15.csv')
   rb['position'] <- 'rb'
   qb <- read.csv('proj_qb15.csv')
   qb['position'] <- 'qb'
   k <- read.csv('proj_k15.csv')
   k['position'] <- 'k'
  # install.packages('plyr')
   library(plyr)
   projections <- rbind.fill(wr, te, rb, qb, k)
   x <- projections
   x[, 1:19][is.na(x[, 1:19])] <- 0

#2. The NFL season is 17 weeks long, and 10 weeks have been completed.  Each team plays 16 games and ha


#subset(projections, projections$Team['PIT'])
steelers <- projections[grep('PIT', projections$Team), ]
no <- projections[grep('NO', projections$Team), ]
cle <- projections[grep('CLE', projections$Team), ]
nyg <- projections[grep('NYG', projections$Team), ]
projnobye <- rbind(steelers, cle, nyg, no)
numpro <- sapply(projnobye, is.numeric)
projnobye[, numpro] <- projnobye[, numpro] * 10/16


projections <- projections[ ! grepl('PIT', projections$Team),]
projections <- projections[ ! grepl('CLE', projections$Team),]
projections <- projections[ ! grepl('NYG', projections$Team),]
projbye <- projections[ ! grepl('NO', projections$Team),]
numpro <- sapply(projbye, is.numeric)
projbye[, numpro] <- projbye[, numpro] * 9/16

projections <- rbind(projbye, projnobye)

#3. Sort and order the data by the `fpts` column descendingly.  Subset the data by keeping the top 20 k

#projsort <- projections[order(projections$fpts, decreasing = T)]
```

```r
projsort <- arrange(projections, desc(projections$fpts))

#projsort[duplicated(projsort$position)]
qb <- projsort[grep('qb', projsort$position), ]
wr <- projsort[grep('wr', projsort$position), ]
rb <- projsort[grep('rb', projsort$position), ]
te <- projsort[grep('te', projsort$position), ]
k <- projsort[grep('k', projsort$position), ]

qb <- qb[1:20,]
wr <- wr[1:60,]
rb <- rb[1:40,]
te <- te[1:20,]
k <- k[1:20,]
projections <- rbind(wr, te, rb, qb, k)
projections <- arrange(projections, desc(projections$fpts))

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

current <- read.csv('nfl_current15.csv')


#5. Merge the projected data with the observed data by the player's name. Keep all 160 rows from the pr

names(current)[names(current)=="Name"] <- "PlayerName"
total <- join(projections,current,by="PlayerName")
total[, 1:36][is.na(total[, 1:36])] <- 0
```

```
## Warning in `[<-.factor`(`*tmp*`, thisvar, value = 0): invalid factor level,
## NA generated
```

```
## Warning in `[<-.factor`(`*tmp*`, thisvar, value = 0): invalid factor level,
## NA generated
```

```r
total[20:21] <- list(NULL)

#6. Take the difference between the observed data and the projected data for each category. Split the d
#You will now have a list with five elements.  Each element will be a matrix or data.frame with 15 colu





data.frame(Name=c('field goals','field goals attempted','extra points','passing attempts','passing compl
              projected_col=c('fg','fga','xpt','pass_att','pass_cmp','pass_yds','pass_tds','pass_ints',
              observed_col=c("FGM","FGA","XPM","Att.pass","Cmp.pass","Yds.pass","TD.pass","Int.pass","
```

```
##                      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
```

```
field.goals <- total[, 'fg'] - total[, 'FGM']
field.goalsattempted <- total[, 'fga'] - total[, 'FGA']
extra.points <- total[,'xpt'] - total[, 'XPM']
passing.attempts <- total[, 'pass_att'] - total[, 'Att.pass']
passing.completions <- total[,'pass_cmp'] - total[, 'Cmp.pass']
passing.yards <- total[, 'pass_yds'] - total[, 'Yds.pass']
passing.touchdowns <- total[,'pass_tds'] - total[, 'TD.pass']
passing.interceptions <- total[, 'pass_ints'] - total[, 'Int.pass']
rushing.attempts <- total[,'rush_att'] - total[, 'Att.rush']
rushing.yards <- total[,'rush_yds'] - total[, 'Yds.rush']
rushing.touchdowns <- total[,'rush_tds'] - total[, 'TD.rush']
receiving.attempts <- total[,'rec_att'] - total[, 'Rec.catch']
receiving.yards <- total[,'rec_yds'] - total[, 'Yds.catch']
receiving.touchdowns <- total[,'rec_tds'] - total[, 'TD.catch']
fumbles <- total[,'fumbles'] - total[, 'Fmb']
total$fumbles <- fumbles
total$field.goals <- field.goals
total$field.goalsattempted <- field.goalsattempted
total$extra.points <- extra.points
total$passing.attempts <- passing.attempts
total$passing.completions <- passing.completions
total$passing.yards <- passing.yards
total$passing.touchdowns <- passing.touchdowns
total$passing.interceptions <- passing.interceptions
total$rushing.attempts <- rushing.attempts
total$rushing.yards <- rushing.yards
total$rushing.touchdowns <- rushing.touchdowns
total$receiving.attempts <- receiving.attempts
total$receiving.yards <- receiving.yards
total$receiving.touchdowns <- receiving.touchdowns
total[3:8] <- list(NULL)
total[6:28] <- list(NULL)
question2 <- total
total[1:2] <- list(NULL)
total[2] <- list(NULL)

qb <- total[grep('qb', total$position), ]
wr <- total[grep('wr', total$position), ]
rb <- total[grep('rb', total$position), ]
te <- total[grep('te', total$position), ]
k <- total[grep('k', total$position), ]
qb[2] <- list(NULL)
```

```
wr[2] <- list(NULL)
rb[2] <- list(NULL)
te[2] <- list(NULL)
k[2] <- list(NULL)



question3 <- list(qb, wr, rb, te, k)
```

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.

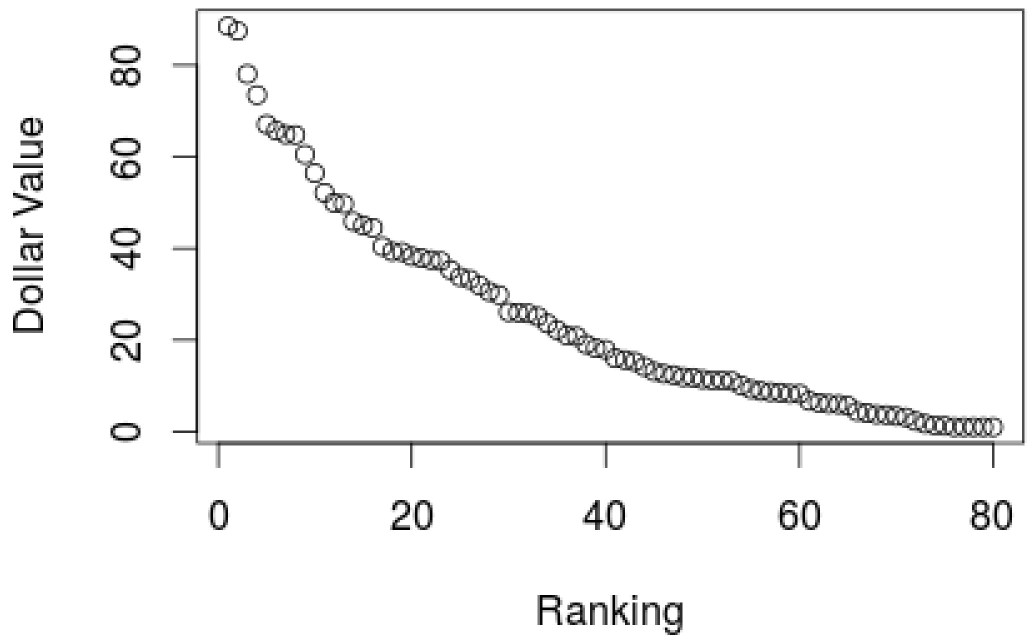
3. Create a function `buildValues` that takes 1 argument, a league object. It will modify the league object by calculating the dollar value of each player.

   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).

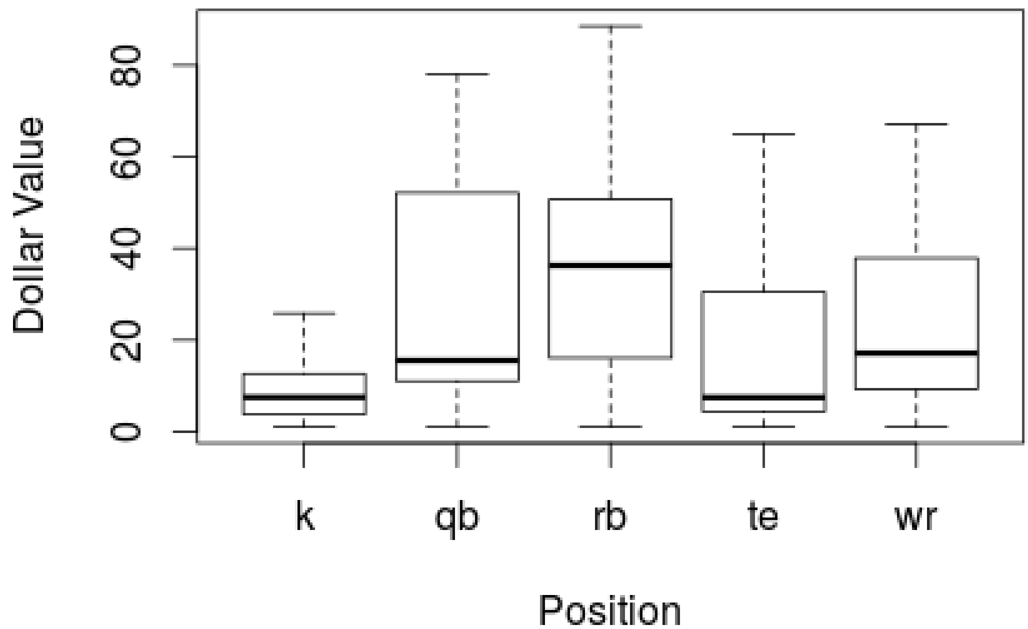
5. Create a `plot` method for the league class. Add minimal plotting decorations (such as axis labels).

   - Here's an example:



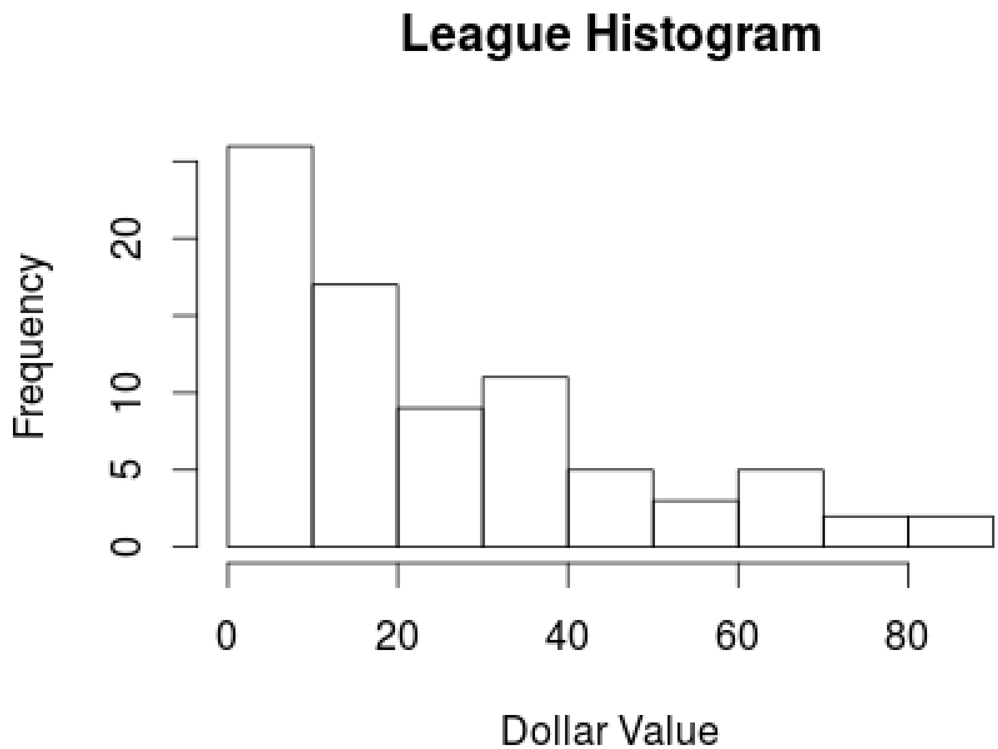
6. Create a `boxplot` method for the league class. Add minimal plotting decorations.

    - 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`).

league <- function( stats, nTeams=10, cap, posReq, points) {
  x= stats

  attr(x, 'nTeams') <- nTeams
  attr(x, 'cap') <- cap
  attr(x, 'posReq') <- posReq
  attr(x, 'points') <- points
  x <- calcPoints(x)
  x <- buildValues(x)
  class(x) <- c('league', 'data.frame')
  return(x)
}

#2. Create a function `calcPoints` that takes 1 argument, a league object.  It will modify the league o



calcPoints <- function(league) {
  names(league) <- gsub('[.]', '', names(league))

  for(i in names(attr(league, 'points'))) {
    league[,sprintf("p_%s", i)] <- league[,i]*attr(league, 'points')[[i]]
  }
  league[,'points'] <- rowSums(league[,grep("^p_", names(league))])
  x2 <- league[order(league[,'points'], decreasing=TRUE),]
  for(i in names(attr(league, 'posReq'))) {
    ix <- which(x2[,'position'] == i)
    baseline <- (attr(league, 'posReq'))[[i]]*attr(league, 'nTeams')
    if(baseline == 0) {
      x2[ix, 'marg'] <- -1
    } else {
      x2[ix, 'marg'] <- x2[ix,'points'] - x2[ix[baseline],'points']
    }
  }
  x2
  #x3 <- x2[x2[,'marg'] >= 0,]
}



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


buildValues <- function(league) {

  valid <- league[,'marg'] >= 0
  league[valid,'value'] <- league[valid,'marg']*(attr (league, 'nTeams')* attr (league, 'cap')-sum(vali
  league[ !valid, 'value'] <- 0
  league <- league[order(league[,'value'], decreasing=TRUE),]
  rownames(league) <- NULL
  return(league)
```

```
}


#    As an example if a league has ten teams and requires one kicker, the tenth best kicker should be w

#4. Create a `print` method for the league class. It should print the players and dollar values (you ma
#1. Create a `plot` method for the league class. Add minimal plotting decorations (such as axis labels)

print.league <- function(league) {
  league <- league[league[,'marg'] >= 0,]
  print.data.frame(league[,c('PlayerName', 'value')])

}

plot.league <- function(league, ...) {
  valid <- league[,'marg'] >= 0
  plot(league[valid, 'value'], xlab = 'Ranking' , ylab = 'Dollar value' , title('plot'), ...)
}



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


boxplot.league  <- function(league) {
  league <- league[league[,'marg'] >= 0,]
  boxplot(league$value ~ league$position, xlab = 'Position', ylab = 'Dollar value')

}

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


hist.league <- function(league) {
  league <- league[league[,'marg'] >= 0,]
  hist(league$value, xlab = 'Dollar value', ylab = 'Frequency', main = 'League Histogram')
}
```

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.

```
#Using residuals from task 1, create a list of league simulations.  The simulations will be used to gen

#1. Create a function `addNoise` that takes 4 arguments: a league object, a list of residuals, number o

#The original league object contains a `stats` attribute. Each simulation will modify this by adding re

  #  As an example assume you want to simulate new projections for quarterbacks. The residuals for quar


addNoise <- function(league, residuals = question3, nsims, RNG=20151215) {
  set.seed(RNG)


  for (i in nsims) {
    rowsam <- sample(nrow(question3 [[1]]  ))[[i]]
    sims = c (attr (league, 'stats'), rowsam)
  }
  league <- sims
  league <- league[which(league<0)]=0

}


#2. Create a `quantile` method for the league class; it takes at least two arguments, a league object a


quantile.league <- function(league, probs = c(0.25, 0.5, 0.75), sims) {

  quantile(league$value)

}
```

```
#3. Create a function `conf.interval`; it takes at least two arguments, a league object and a probs vec

#    The new object will contain the output of `quantile`. However, results should be split by position

conf.interval <- function(league, probs, sims) {
  UseMethod(quantile)
  q <- quantile()
  return(league.conf.interval) <- q
}




#4. Create a `plot` method for the league.conf.interval class; it takes at least two arguments, a leagu

plot.league.conf.interval <- function(league.conf.interval, position){
  plot(league.conf.interval, xlab='Ranking', ylab = 'Dollar value')

}
```

```
* Here's an example:

    ![ciplot example](https://dl.dropboxusercontent.com/u/5044136/Bios301/ciplot_ex.png)
```

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')
```

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.