Code Review

Jared Bennett

08 October, 2019

Homework 3 Notes and Code Review

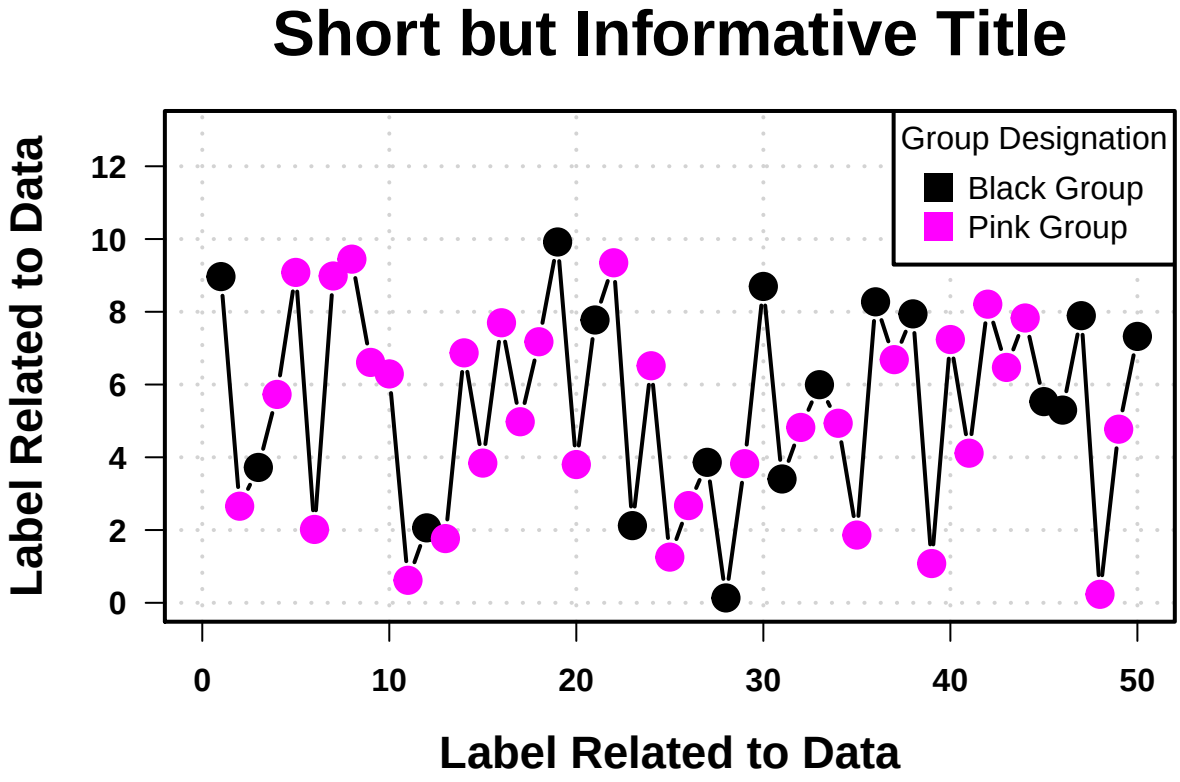
This section is mostly a focus on reviewing other students solutions, using PS3 as an example. We'll begin with things that I saw in homeworks, anything that didn't make sense to people, and then finish section with the code review.

PS3 Notes

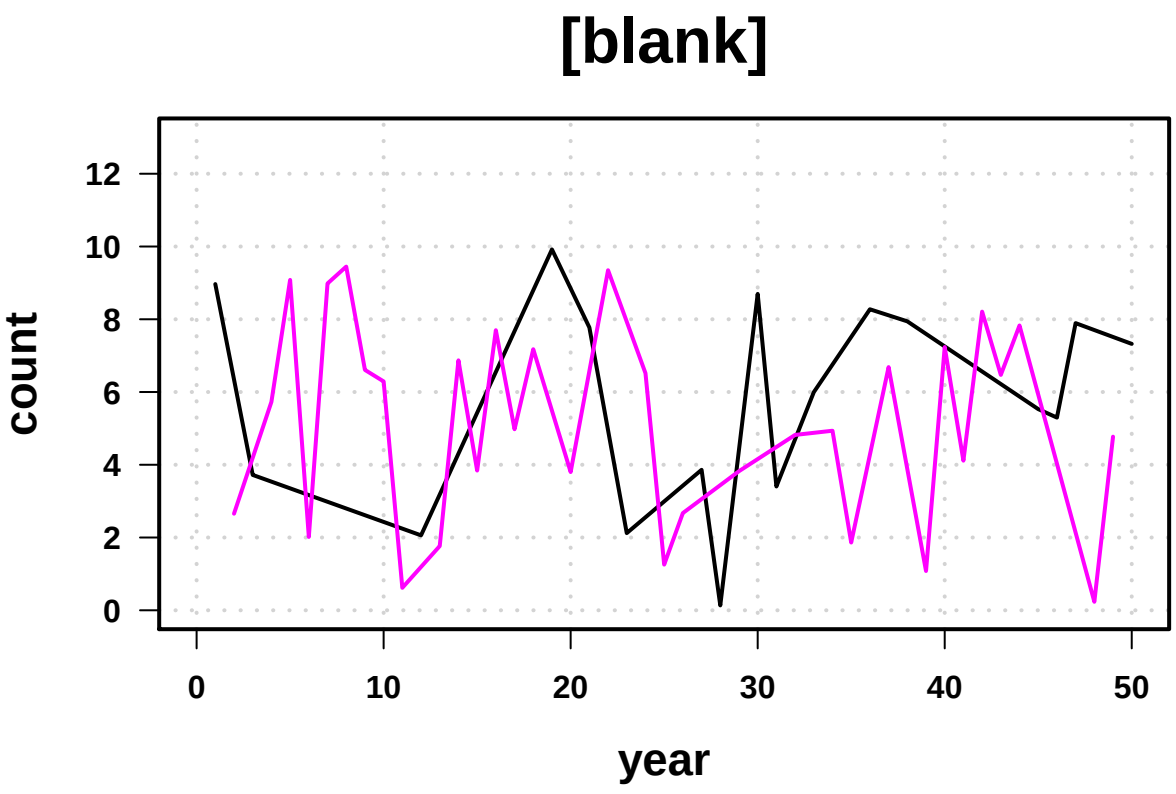
1. Problem Statements
   You **have to** answer the questions on the homework. Do whatever you'd like beyond that, be as creative as you wish, but **you must answer the questions that were asked**.
2. Include Code
   Most people include it as they go along, that's fine.
   If you find that messy, put an appendix at the end with all of the function definitions.
   This applies to code for plots as well.
3. Bonus/Extra Credit
   This is something additional, above and beyond the basic homework.
   Repeating the analysis already performed, but with different parameters, is not novel.
4. Plots and Comments
   Plots need to have a title, axis labels, and a key if there are several types of data.
   Code and comment lines should be ~80 characters in length.
   e.g.

```
####################
## Make Comment point
####################
# This is a bad comment
# Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labo

# This is a good comment
# Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
# incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis
# nostrud exercitation ullamco laboris nisi ut aliquip ...etc

####################
## Good plotting
####################
# setup things
n <- 50
x <- 1:n
y <- runif(n = n, min = 0, max = 10)
colChoice <- c("black","magenta")
cols <- sample(x = colChoice, size = n, replace = T, prob = c(1,2))
```

```r
# could use a key to set them
# colChoice <- c("black","magenta")
# key <- (y > 5)
# cols <- character(length = n)
# cols[key] <- colChoice[1]
# cols[!key] <- colChoice[2]

# parameters
par(las = 1, font.lab = 2, font.axis = 2, font.main = 2, cex.main = 2, cex.lab = 1.4, cex.axis = 1,
# par(las = 1, font.lab = 2, font.axis = 2, font.main = 2,
#     cex.main = 2, cex.lab = 1.4, cex.axis = 1, lwd = 2)

# plot
plot(x = x, y = y, type = "b",
     xlim = c(0,n), ylim = c(0,13),
     main = "Short but Informative Title",
     xlab = "Label Related to Data",
     ylab = "Label Related to Data",
     lwd = 2, pch = 19, cex = 1.75,
     panel.first = grid(),
     col = cols)
#box(lwd = 3)

# add legend
legend("topright", legend = c("Black Group","Pink Group"),
       title = "Group Designation", col = colChoice,
       pch = 15, pt.cex = 2, bg = "white")
```



```r
####################
## Bad plotting
####################
# same stuff, bad lines
black <- which(cols == "black")
pink <- which(cols == "magenta")

# settings
par(las = 1, font.lab = 2, font.axis = 2, font.main = 2,
    cex.main = 2, cex.lab = 1.4, cex.axis = 1, lwd = 2)

# plot
plot(x = x[black], y = y[black], type = "l",
     xlim = c(0,n), ylim = c(0,13),
     main = "[blank]",
     xlab = "year",
     ylab = "count",
     lwd = 2,
     panel.first = grid(),
     col = "black")

# plot second line
lines(x = x[pink], y = y[pink], type = "l", lwd = 2, col = "magenta")
```



5. Object Initialization, Atomic Access, and Growth
   There are many ways to initialize objects in R, but there are more and less correct methods to do it.

```
####################
## Object Initialization
####################
n = 10000

# integers
x1 <- rep(as.integer(NA), n)
x2 <- rep(1L, n)
x3 <- c(1:n)
x3.5 <- 1:n
x4 <- vector(mode = "integer", length = n)
x5 <- integer(length = n)

# double
x6 <- double(length = n)
x6.5 <- numeric(length = n)

# logical
x7 <- logical(length = n)

# lists
#  There is not a good way to do this, sorry
x9 <- vector(mode = "list", length = n)

# benchmark for fun
microbenchmark::microbenchmark("repNA" = rep(as.integer(NA), n),
                               "rep" = rep(0L, n),
                               "repInt" = rep.int(x = 0L,times = n),
                               "int()" = integer(length = n),
                               "vec()" = vector(mode = "integer", length = n),
                               times = 1000)
```

```
## Unit: microseconds
##    expr   min      lq     mean  median      uq      max neval
##   repNA 9.395 18.1690 25.81256 18.6635 19.3440 3953.082  1000
##     rep 9.345 18.0300 18.61192 18.5190 19.2675   37.948  1000
##  repInt 6.812 15.4775 19.33494 15.9570 16.6495 3248.412  1000
##   int() 1.418 10.6965 11.29176 11.1845 11.9455   37.456  1000
##   vec() 1.553 10.7755 11.50994 11.3040 12.0075   31.225  1000
```

```
####################
## Object replacement/access
####################
# run this in TERMINAL!!!
x1 <- integer(length = n)
.Internal(inspect(x1))
x1 <- vapply(X = 1:n, FUN = as.integer, FUN.VALUE = integer(length = 1))
.Internal(inspect(x1)) # this will be different!!
x1[1:n] <- vapply(X = 1:n, FUN = as.integer, FUN.VALUE = integer(length = 1))
.Internal(inspect(x1)) # this will be the same
```

```r
####################
## Growing Objects
####################
# Initialize
myList <- list()
myVec <- numeric(length = 1)

# check length
cat("Length of list is:", length(myList),
    "\nLength of vector is:", length(myVec))
```

```
## Length of list is: 0
## Length of vector is: 1
```

```r
# loop
for(i in 1:n){
  myList[[i]] <- "HAHAHA, I'm growing!"
  myVec[i] <- as.integer(i)
  # effectively the same as append() or c(old, new)
}

# check length
cat("Length of list is:", length(myList),
    "\nLength of vector is:", length(myVec))
```

```
## Length of list is: 10000
## Length of vector is: 10000
```

6. Vectorizing
"Vectorizing" in R implies using functions that operate in parallel over certain objects. R is vectorized in many operations (addition, subtraction, multiplication, division). Many functions have built-in vectorization as well, make sure to check the function documentation.

```r
####################
## vectorized division
####################
x <- 1:100
y <- 100:1

mHold <- mapply(FUN = "/", x, y)
dHold <- x/y

all.equal(mHold, dHold)
```

```
## [1] TRUE
```

```r
identical(mHold, dHold)
```

```
## [1] TRUE
```

```r
####################
## vectorized multinomial
####################
myprobs <- matrix(data = c(0.1,0.1,0.8, 1/3,1/3,1/3), nrow = 2, byrow = T)

set.seed(0)
apHold <- t(apply(X = myprobs, MARGIN = 1, FUN = rmultinom, n = 1, size = 100))
```

```r
set.seed(0)
edHold <- extraDistr::rmnom(n = 2, size = 100, prob = myprobs)

all.equal(apHold, edHold)
```

```
## [1] TRUE
```

```r
identical(apHold, edHold) # why false?
```

```
## [1] FALSE
```

```r
# short benchmark
microbenchmark::microbenchmark("apply" = apply(X = myprobs, MARGIN = 1,
                                               FUN = rmultinom, n = 1, size = 100),
                               "vector" = extraDistr::rmnom(n = 2, size = 100,
                                                            prob = myprobs),
                               times = 100)
```

```
## Unit: microseconds
##    expr    min     lq     mean median      uq     max neval
##   apply 17.822 18.563 20.50875 19.195 19.7050 139.687   100
##  vector  6.388  6.763  7.83082  7.568  8.2885  28.932   100
```

PS3 Code Review

Today we will practice paired code review for PS3. In order for this to be of benefit, you will need to be as honest with your partner as possible. If something is actually hard to follow, tell them! If you thought that something they did was clever, also tell them!

Procedure

First, we will break into pairs of 2. This can be accomplished in one of two ways:

1. You pair up randomly with someone you don't know
2. You count off and I draw numbers for you
    - e.g., `matrix(data = sample(x = 1:n, size = n, replace = FALSE), ncol = 2)`

Second, you will each spend 15-20 minutes reading each other's code while your partner is available to answer questions you have about their work. I will float around and can make any clarifying points.

Finally, each of you will fill out this (also here https://tinyurl.com/statps3rev) google form summarizing the differences between yours and your partners work, noting anything that was particularly novel to you.

Some Guiding Questions

1. Is the code visually easy to break apart? Can you see the entire body of the functions without scrolling up/down left/right? The general rule-of-thumb is no more than 80 in either direction (80 character width, 80 lines. The first restriction should be practiced more religiously)
2. Are the data structures easy to parse? Do you understand what information is in which objects? Is the type of data appropriate?
3. Are the functions easy to parse? Is it clear what they are doing, what information is being passed to each function, and what the return value/objects are?
4. Look at the regex. Are there any edge cases that were missed? Was anything handled particularly well? Any novel expressions that you didn't think of?
5. Look at the plots. Are they well labeled? Is it easy to read information from them? Do they provide an assurance that the code is running correctly? Was there anything particularly neat about your partner's solution to the bonus?
6. EXTRA. Look at problem 2, compare/contrast your approaches to OOP. Explain the logic behind your design.