Stat243: Problem Set 2, Due Friday Oct. 4

September 18, 2013

This covers material in Units 3 and 4 on R.
It's due at the start of class on 10/4.
Some guidelines on how to present your solutions:

1. Please use your knitr/Sweave/R Markdown solution from PS1, problem 5 as your template for how to format your solutions (only non-Statistics students are allowed to use R Markdown).
2. As with PS1, your solution should not just be code - you should have text describing how you approached the problem and what the various steps were.
3. In addition to the paper submission, which is the primary thing we will grade, please turn in your raw Latex/Markdown with embedded code electronically on bSpace (a file just containing the code with indication of what pieces of code are for what problems is fine too).
4. All your code should have comments indicating what each function or block of code does, and for any lines of code or code constructs that may be hard to understand, a comment indicating what that code does. You do not need to show exhaustive output but in general you should show short examples of what your code does to demonstrate its functionality. Depending on how long different pieces of your code are, you may want to have some of it be an appendix rather than part of the main solution.
5. Use functions as much as possible, in particular for any repeated tasks. We will grade in part based on the modularity of your code and your use of functions.

Please note my comments in the syllabus about when to ask for help and about working together.

Problems

1. Some questions about scoping.

   (a) Explain why the result of this is a vector of length 8.

```
a <- 8

genFun1 <- function() rnorm(a)

wrapFun1 <- function(data) {

    a <- length(data)
```

```
    return(genFun1())

}

wrapFun1(1:3)

## [1]  2.2155  0.2514  0.1526  0.4715  3.0137 -0.8437 -0.7464  0.1855
```

(b) Now explain why the result of this is a vector of length 3.

```
a <- 8

genFun2 <- function(x) rnorm(x)

wrapFun2 <- function(data) {

    a <- length(data)

    return(genFun2(x = a))

}

wrapFun2(1:3)

## [1] -0.8060 -0.1682  1.3328
```

(c) And why this is a vector of length 16.

```
a <- 8

genFun3 <- function(x = a * 2) rnorm(x)

wrapFun3 <- function(data) {

    a <- length(data)

    return(genFun3())

}

wrapFun3(1:3)

##  [1] -0.299274  1.703868 -0.276180  0.609983 -0.332622  0.799132 -1.2573
##  [8]  0.632928  0.008182  0.145242 -0.666172 -1.294892  0.312453 -0.8519
## [15] -0.682584  0.989831
```

2. Consider the result of running the following code in R.

```
sapply(0:3, function(x) {

ls(envir = sys.frame(x))

}

)
```

Interpret the output in light of our discussion of frames. Discuss what functions are being called, in what order, and what objects exist in the frame of each function. What are the numbers and names (the environment name in hexadecimal (base 16)) of each frame (note the names will change each time you run the code)? You may want to use *debug()* , *trace()* or *browser()* or to step through the code and see what is being executed when you call *sapply()*.

3. The file */scratch/users/paciorek/PUMS5_06.TXT.bz2* on the SCF filesystem is a zipped file containing household and person-specific records for California from the 2000 US Census, in particular a sample of 5% of the population. The meta data describing the file format is at http://www.census.gov/prod/cen2000/doc/pums.pdf - look for the data dictionary section to interpret the format of the rows in the data file. Your job is to take a random sample of n observations just from the household-level records (where the user can specify n), saving the result as an R data frame with the following columns extraced from the Census data file: BEDRMS, FINC, NPF, ROOMS, HHT, P18, P65. Format the fields of your data frame in a meaningful way - e.g., use factors for categorical variables as appropriate and for a variable such as FINC, you will probably want to have two columns, since FINC is a combination of a categorical and a numerc variable. Give your columns meaningful but concise names.
   Some rules for your solution:

   (a) You must not unzip the file when you read the data in. This is to mimic the situation where the file is too big when unzipped to fit on the hard drive. For purposes of writing your code, you can unzip the file to look at it, but note that "less" will probably show you the contents without unzipping it. If you would like to deal with the subsetting with UNIX command line tools, you can do this (and it may make your life easier, but you should figure out how to do this via piping without ever creating either the original or the subsetted file in unzipped form. bunzip2 and bzip2 can deal with compressed files in the bzip (.bz2) format.

   (b) You must only read the n observations into R. You are not allowed to read the entire file in, nor are you allowed to read it in in blocksthat contain observations other than those that you will retain.

   (c) Please make sure to write your code modularly. E.g., you might have an R function, say, 'sampleFile()' that does the sampling of rows from the file, given an input of the number of observations to sample and the name of the file. Then you could have a separate step of processing that processes a row to extract the elements of the row. Try to think of what the distinct tasks are and do each task as a separate function.

   (d) Make sure to set the random number seed using *set.seed()* so your sample is reproducible.

4. DRAMATIS PERSONAE: INSTRUCTOR, STUDENTS, GSI
   ACT 1. SCENE 1
   UC Berkeley
   INSTRUCTOR. [To STUDENTS] I set before thee the task of processing the plays of Shakespeare, to scrape as it were, in the manner of combing wool from a sheep. [To GSI] This should keep them busy.
   STUDENTS. [With gnashing of teeth] Nay, nay, thou shall not set us to this mighty task!
   ...
   ACT Post-Berkeley
   STUDENTS. [To INSTRUCTOR] We thank thee - you have set us upon a golden path, where beautiful data lie open to us in every nook of this World Wide Web.

   Translation:
   Your job is to extract data from the complete works of William Shakespeare, available as plain text at http://www.gutenberg.org/cache/epub/100/pg100.txt.
   [UNDER CONSTRUCTION]

   (a) Extract the plays (skip the information at the start of the file and the sonnets) into a character vector or a list, with one play per element.

   (b) Using your own functions and function from the apply() family, extract meta data about each play and the body of the play. The result should be in the form of a character vector with attributes or a list, with one play per element. By meta data I mean the year of the play, the title, the characters (i.e., the dramatis personae).

   (c) Extract the actual text spoken by the characters into R data structures. At the highest level you should have a list with each element being a play. Within that you should distinguish the acts and scenes, separate the names of the speakers from what they say, and distinguish the stage directions - the text that tells the actors what to do (e.g., "Enter Orlando and Adam", "Adam retires", "To Orlando"). Do not discard information such as the acts/scenes and stage directions but do strip it from the body of what is spoken.

   (d) Now use the constructed data object to calculate summary statistics about each play. These should include:

      i. The number of scenes and acts
      ii. The number of dramatis personae
      iii. The number of spoken chunks
      iv. The number of words spoken and average and standard deviation of number of words per chunk.
      v. The number of unique words, ignoring "stop" words, as specified at this URL [insert URL].
      vi. [More to come]

   (e) Plot some of your summary statistics as a function of time to see if their are trends in Shakespeare's plays over the course of his writing career.