Stat243: Problem Set 2, Due Monday Sep. 22

September 13, 2014

Comments:

- This covers Unit 3 and Unit 4 (not including text processing/regular expressions).
- It's due at the start of class on 9/22.
- Please note my comments in the syllabus about when to ask for help and about working together.

Formatting requirements

As discussed in the syllabus, please turn in (1) a copy on paper, as this makes it easier for us to handle AND (2) an electronic copy through Git following Jarrod's instructions.

1. Your task is to write a command-line tool that will subset and stratify from large zipped datafiles. The code should be entirely in R, but callable from the UNIX command line. Here are the specifications of what it should (and should not do):

   (a) The tool should read from a single zipped file without ever unzipping the entire file. It should write out to individual zipped files, one per stratum, without ever creating an unzipped file. Use the bzip2 format.

   (b) The tool should take an argument indicating which column of the input file is the stratifying variable.

   (c) The tool should take another argument indicating which column of the input file is the subsetting variable, and an additional argument indicating the (single) value of the variable for which the observations should be retained. Other observations whose value for that variable do not match should be discarded.

   (d) The tool should read from the input file in blocks, so that the entire input file is not read into R (and therefore into memory) at once.

   (e) For simplicity, don't write out any header rows to any of the output files (you may need to use *write.table()* instead of *write.csv()* to deal with this).

   To be concrete, you should use your tool to select all of the flight data for flights arriving at the San Francisco airport (SFO), stratified by year, for the airline flight data in the file http://www.stat.berkeley.edu/share/paciorek/AirlineData2006-2008.csv.bz2. This is a US government database of flight delay information and is a common testbed amongst statisticians for working with large datasets. I've subset the dataset to include only 2006-2008 to limit the size of the file you will need to download to your working machine. I.e., you should produce three files, *2006.csv.bz2*, *2007.csv.bz2*, and *2008.csv.bz2*. However, you may test out your code on the full dataset, http://www.stat.berkeley.edu/share/paciorek/AirlineDataAll.csv.bz2 if you're interested. That full file is 1.5 Gb zipped and has data for 1987-2008. More details on the dataset can be found at http://stat-computing.org/dataexpo/2009/, including metadata describing the fields.

Hints:

- This exercise is less complicated than it may sound, if you make good use of the R functionality covered in Units 3 and 4. The core code in my solution is about 15 lines of R code.
- The bz2 format can be handled by the bzip2 and bunzip2 tools.
- If you are reading in using *read.{csv,table}*, you will get an error if you try to continue reading after getting to the end of the file/connection. Here's a useful function to deal with that and allow your code to continue running despite the error (more details are in Section 5.5 of Unit 5):

```
dat <- try(read.csv(.......))

if (!class(dat) == "try-error") {

    # process 'dat'

} else {

    # you've read everything; do any post-processing you need to

}
```

- Also, you need to close any connections you are writing to before exiting.

2. This problem will have you thinking about scoping in R. Consider the following code:

```
myFuns <- vector("list", 3)

for (i in 1:length(myFuns)) {

    myFuns[[i]] <- function() {

        return(i)

    }

}

# First evaluation

for (j in 1:length(myFuns))
print(myFuns[[j]]())

## [1] 3
## [1] 3
## [1] 3
```

```
# Second evaluation

for (i in 1:length(myFuns))
print(myFuns[[i]]())

## [1] 1
## [1] 2
## [1] 3
```

(a) Explain the result of the first ('j') for loop. Why is the result 3 every time?

(b) Explain the result of the second ('i') for loop. In particular, where is the value of 'i' in the three 'myFuns' functions being found?

(c) Now consider the following code where the three functions are generated within another function. Why do both loops now give the same result and where is 'i' being found?

```
funGenerator <- function(len) {

    f <- vector("list", len)

    for (i in seq_len(len)) {

        f[[i]] <- function() {

            i

        }

    }

    return(f)

}



myFuns <- funGenerator(3)

# Third evaluation

for (j in 1:length(myFuns))
print(myFuns[[j]]())

## [1] 3
## [1] 3
## [1] 3
```

```
# Fourth evaluation

for (i in 1:length(myFuns))
print(myFuns[[i]]())

## [1] 3
## [1] 3
## [1] 3
```

3. 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, what the frame number is for each function, and what objects exist in the frame of each function.