Lab 3

Statistical Computing, 36-350

Friday September 18, 2015

Today's agenda: manipulating data frames; practicing iteration; practicing re-writing code; checking how reliable random methods are.

General instructions for labs. Upload an R Markdown file, named .Rmd", to Blackboard. You will give the commands to answer each question in its own code block, which will also produce plots that will be automatically embedded in the output file. Each answer must be supported by written statements as well as any code used. Include the name of your lab partner at the top of the file.

R Markdown setup. Open a new R Markdown file; set the output to HTML mode and click "Knit HTML". This should produce a web page with the knitting procedure executing your code blocks. You can edit this new file to produce your homework submission. Alternatively, you can start from the lab's R Markdown file posted on the course website, as a template.

Part I: Data frames

R includes a number of pre-specified data objects as part of its default installation. We will load and manipulate one of these, a data frame of 93 cars with model year 1993. Begin by ensuring that you can load this data with the commands

```
library(MASS)
data(Cars93)
```

Begin by examining the data frame with the command `View(Cars93)` to understand the underlying object. You must use functions and other commands to extract elements for this assignment; not a visual inspection using `View(Cars93)`.

1. What are the column names of the data frame `Cars93`? What are the number of rows in this data frame?
2. What is the mean price of a US-made car? A non-US-made car? (What are the units here?)
3. What is the biggest difference between highway MPG and city MPG? Which make of car achieves this difference?
4. How many different cars have no airbags? Of these, how many are US-made, and non-US-made?
5. Assuming that these cars are exactly as fuel efficient as this table indicates, find the cars that have the maximum, minimum and median distance travellable for highway driving. You will need at least two columns to work this out; why those two?

Part II: Reproducibility and functions

Some of the lectures have included examples of planning production for a factory that turns steel and labor into cars and trucks. Below is a piece of code that roughly optimizes a factory's output, given the available resources, using a `repeat` loop. It's embedded in a function to make it easier for you to run. We'll learn functions in detail a little later in the course.

```
factory.function = function (cars.output=1, trucks.output=1) {
  factory = matrix(c(40,1,60,3), nrow=2,
    dimnames=list(c("labor","steel"),c("cars","trucks")))
  available = c(1600,70); names(available) = rownames(factory)
  slack = c(8,1); names(slack) = rownames(factory)
  output = c(cars.output, trucks.output); names(output) = colnames(factory)

  passes = 0 # How many times have we been around the loop?
  repeat {
    passes = passes + 1 # Update our counter
    needed = factory %*% output # What do we need for that output level?
    # If we're not using too much, and are within the slack, we're done
    if (all(needed <= available) && all((available - needed) <= slack)) {
      break
    }
    # If we're using too much of everything, cut back by 10%
    if (all(needed > available)) {
      output = output * 0.9
      next
    }
    # If we're using too little of everything, increase by 10%
    if (all(needed < available)) {
      output = output * 1.1
      next
    }
    # If we're using too much of some resources but not others, randomly
    # tweak the plan by up to 10%
    # (Note that runif generates a random number, uniformly distributed;
    # NOT "run if"!)
    output = output * (1+runif(length(output),min=-0.1,max=0.1))
  }

  return(output)
}
```

5. Run the function above with the command `factory.function()` to obtain a default output value, starting from a very low initial planned output. What is the final output capacity obtained?

6. Repeat this five more times to obtain new output values. Do these answers differ from each other? If so why? If not, why not?

7. Right now, the number of `passes` is a value held within the function itself and not shared. Change the code so that the function returns a list that contains the final `output`, as well as the number `passes`.

8. Now, set the initial output levels to 10 cars and 19 trucks and run the code. What is the final output plan (`output`)? What is the final demand for resources (`needed`)? Note: for this last part, you will need to either calculate `needed` manually outside of the function, or have the function return `needed` in addition what it currently returns (`output` and `passes`).

9. Explicitly check that the plan from the last part is within budget and within the slack. Also, report how many iterations it took to converge (`passes`). How does this compare, roughly, to the number of iterations required with the default initial output settings (1 car and 1 truck)?

Part III (Bonus, optional)

10. Let's make the comparison between the number of iterations required in the last part a little more formal. Using a `for()` loop, run `factory.function()` 500 times with the default initial output settings (1 car and 1 truck), and record how many iterations it took each time, in a vector of length 500, called `factory.passes.cars1.trucks1`. Do the same but with a the initial output settings now being 10 cars and 19 trucks, saving the iterations required in a vector called `factory.passes.cars10.trucks19`.

(Note: in order to avoid running into a problem where `factory.function()` requires an enormous, possibly indefinite, number of iterations to finish, you may want to cap the maximum number of `passes` taken at some big number, like 100000. I.e., if it ever hits this big number, have the function just break out of the loop no matter what.)

11. Summarize the difference between the two vectors `factory.passes.cars1.trucks1` and `factory.passes.cars10.trucks19`, numerically. Say, what are their means, and standard deviations? Then, plot a histogram of each vector to graphically display their differences, using `hist()`. (Hint: to make two plots side by side, first call `par(mfrow=c(1,2))`, which sets the plotting window to have 1 row and 2 columns.)