Stat 133, Fall 2014
Homework 4: Web Caching PART 1: Lists and apply functions
Due Thursday Oct 2 at 11:55pm on bspace.

About the data

When you use a search engine to look for a Web page, the search engine looks through its cache.

The cache is created by regularly crawling the web and taking copies of every page that has changed since the last time it visited the site. It's time consuming and costly to crawl the web. In this assignment, you will analyze data collected to help answer the question: How often do you need to crawl the web in order to keep its stock fresh for the search engine?

The data collected are measurements on 500 web sites that were visited regularly over a period of time to see whether or not they had changed. We will study the properties of the changes. These data are in a list of 500 elements, one for each Web site. Each element is an integer vector that records whether the Web site had changed since the previous visit. For example,

```
> Cache500[[8]]
 [1]   3  23  30  32  39  42  45  62  78  82 147 151 155 175
```

indicates that the 8th element of `Cache500` first changed on the third visit; it changed a total of fourteen times, and the final change observed was at the 175th visit to the site.

What to turn in

I have provided a skeleton .R file for you to work from. Turn in just a completed version of this .R file with the name `[last name][first name]_HW4.R`, e.g. "`IbserHank_HW4.R`." Do not turn in any other files. Your answers to the ***boldface, italicized questions*** go at the top of the skeleton .R file in the space provided as *comments*. As you will check in problem 7, your .R file needs to run error-free from top to bottom and produce three .png files.

Assignment

Problem 0

Before starting the assignment, create a separate folder and save your .R file in there. Set this to be your working directory in R. In RStudio, you can do this by going to `Tools > Change Working Directory`. If you are more comfortable with specifying the filepath, you can run `setwd("filepath_of_my_folder")` in the console. Do not put this in your .R file.

We are doing this because this HW assignment will instruct you to generate plots straight to a file in your working directory, so this makes sure you don't lose track of where all of this output is going.

Problem 1

1a. Load the data into R from the site `http://www.stat.berkeley.edu/users/nolan/data/Cache500.rda`. These data are stored as an R object called `Cache500`. Read the data directly into R from the web; DO NOT download it first and then read in the local file.

1b. Check that `Cache500` is a list.

1c. Check that `Cache500` has 500 elements.

1d. Use one of the "apply" functions (`apply()`, `sapply()`, `tapply()`, etc.) to create a vector named `Cache500_class` which contains the class of each element in Cache500.

1e. Use `all()` and the vector `Cache500_class` from problem 1d to check that each element of `Cache500` is an integer vector.

Problem 2

Use one of the "apply" functions to complete each of these parts of Problem 2.

2a. Create a vector `numChanges` that gives the number of times each website in `Cache500` changed.

2b. Create a vector `firstChange` that gives the first time each website changed.

2c. Create a vector `lastChange` that gives the last time each website changed.

Problem 3

We want to see whether the Poisson model for random scatter describes the times at which a change was observed for our Web sites. If this is the case, the times should be uniformly distributed, like the x-coordinates of raindrops on a square of concrete. We will examine a subset of Web sites: those that have between 200 and 250 changes and where the last change was after the 700th visit. Use `which()` to create a vector named `whichOK` that gives the indices of the elements that meet these conditions.

Problem 4

According to the Poisson scatter model, the times of change should look roughly like uniform scatter on the interval from 0 to 720 (the interval of observation). Rather than make a histogram, it can be more informative to examine a quantile plot where we compare the quantiles of the distribution of observed changes to the quantiles of a uniform distribution. This problem will walk you through doing that.

4a. Pick one website in the `whichOK` set of sites to look at. Store its index in `Cache500` as `siteIndex`. For example, if `whichOK` included "23", that means the 23rd site in `Cache500` meets the criteria from Problem 3. If you were to focus on this site, you would set `siteIndex = 23`.

4b. Find the 0.01 quantiles of the website you chose and store them as a vector called `siteQuantiles`. The 0.01 quantiles mean a vector with the 1st percentile, 2nd percentile, ..., 99th percentile.

4c. Create the vector of 0.01 quantiles from the uniform distribution on the interval [0, 1]: 0.01, 0.02, ..., 0.99. Store these as `unifQuantiles`.

4d. Make a scatterplot of the site quantiles vs. the uniform quantiles. Add a thick red line to the plot that goes through as many points as possible. You can just eyeball the line. (Hint: the intercept will probably be around 0 and the slope will probably be around the value of the last visit.) Use `png()` to plot to a file named `[last name][first name]_Prob4.png`, e.g. `IbserHank_Prob4.png`. A skeleton of the plotting code you will write is given in the provided .R file.

4e. ***Describe the shape of the plot. Is the scatterplot roughly a straight line?*** (Remember, answer these boldfaced, italicized questions in a separate .txt file)

Problem 5

5a. Use the `unlist()` function to create a vector called `lumpedChanges` with all of the time changes for the sites in `whichOK`.

5b. Store the 0.01 quantiles of `lumpedChanges` as a vector called `lumpedQuantiles`. Use the `unifQuantiles` vector from Problem 4 to create another quantile plot (just like part 4d). Just as in 4d, use `png()` to plot to a file named `[lastname][firstname]_Prob5.png`. You do not need to add a thick red line to this plot.

5c. ***How does this plot for multiple sites compare to the quantile plot for a single site you generated in problem 4: do the points follow a line better, worse, or no differently?***

Problem 6

Another way to check if our data is uniform is to compare our quantile plot to one where we know the data actually follows a uniform distribution. Also for comparison purposes, we'll look at a quantile plot where we know the data does NOT follow a uniform distribution. Continue working with the website that you chose in 4a.

6a. Use `numChanges` from Problem 3 to determine how many changes your website made. Generate that many values from a uniform distribution on the interval [0, 720], and store these as a vector called `simSiteChanges`. Since these represent simulated numbers of site changes, the resultant vector should only contain integers, i.e. no values like "23.481." Find the 0.01 quantiles as a vector called `simSiteQuantiles`.

6b. Likewise, generate the same number of values from a normal distribution with mean 0 and SD 1. These do not have to be integer-valued. Store these values as a vector called `normChanges`. Store the 0.01 quantiles as a vector called `normQuantiles`.

6c. Create plots as in 4d, but this time fit two plots in one image using the `mfrow` option in `par()`. On the left, make a quantile plot for the simulated changes using `simSiteQuantiles` vs. `unifQuantiles` from Problem 4. On the right, make a quantile plot for the random normal variables using `normQuantiles` and `unifQuantiles` from Problem 4. Use `png()` to plot to a file named `[lastname][firstname]_Prob6.png`. You do not need to add thick red lines to these plots.

6d. ***Compare these two plots to the lumped changes plot of problem 5. What can you conclude about the time of website changes from looking at these two simulated plots?***

Problem 7

Check that your code is reproducible by making sure it can be run from top to bottom. We'll also perform some tests to make sure that you have all of the variables we've asked for in this assignment, and they're of the right class and length. This last part is a spin-off of a programming concept called "unit testing," and is generally good coding practice. There are NO commands to turn in for this problem, but you will lose points if your code produces errors or does not pass the basic checks in the test file when the grader runs your code.

7a. Save your .R file and clear your workspace by running `rm(list=ls())` (do not include this in your .R file!). Delete any .pngs that were rendered to your working directory.

7b. Set your working directory to the folder where your .R file lives (and where you've been saving your .pngs). Run your .R file by typing `source("name_of_my_R_file.R")`. If this gives you errors, you'll need to go back and address them before you can proceed to the next part. Also check that your code produces three .png files with names like "`IbserHank_Prob#.png`" in your working directory.

7c. At this point, your .R file should run error-free from top to bottom, producing three .png files in the working directory along the way. Download `HW4_tests.R` and save it in the same directory. Clear your workspace by running `rm(list=ls())`. Source this file by running `source("HW4_tests.R")`. Run `testHW4("name_of_my_R_file.R")`.

The function `testHW4()` will source your .R file (running it from top to bottom) and check that each variable asked for in this homework exists and meets some basic requirements. You will see output for each problem, like this:

```
~~~~~~~~~TEST lumpedQuantiles~~~~~~~~~
PASS: lumpedQuantiles exists.
......PASS: correct length.
!.....FAIL: 1% percentile is too small.
```

Check that you have "PASS" for each test.