StatR 502 Homework 7

Gregor Thomas

Due Thursday, Feb. 25, 2016, 6:30 pm

Submission guidelines: please submit a knitted PDF or Word document, and optionally your `.Rmd` file. As always, ask in the discussion forum if you're having trouble!

1. Nice plot

Create two nice, polished visualizations using the data for your final project. If you don't have your final project data yet, then you may use the `tips` data in the `reshape2` package. Polish your plots: pick out some nice colors, make sure labels are clear, and set it up so each plot tells a little story or highlights an interesting comparison.

2. Everyone's a critic

(a) Find a data graphic that you think is bad. Post it to the "Bad Visualizations" forum. On the scales of Question, Data, and Visualization, how do you rate it? Feel free to look on news sites, in journals, blogs, etc., just please don't take one from a site where it's already being criticized (e.g., junkcharts).

(b) (The hard part.) Find a data graphic that you think is *done well*. Post it to the "Good Visualizations" forum. On the scales of Question, Data, and Visualization, how do you rate it? What techniques are being used successfully? Are there any improvements you can suggest?

(c) Comment on at least one other post in each visualization forum. Do you agree with the rating of the person who posted it? Do you have other ideas for how it could be improved? *Of course, if you disagree, be tactful!*

3. A little more regex (this problem isn't as long as it looks, there's just some reading with it)

The vector `rstrings`, stored in `hw7data.RData` is a character vector of length 50. It's unfortunate that R calls "character" things that are called "string" in most every other programming language. For clarity, in the rest of this problem statement I will follow the non-R convention and refer to a single letter, number, space, or other symbol as a *character*, and to several characters together inside quotes as a *string*. This exercise is to practice using R's string-manipulation functions; you're welcome to use the base functions (which are all documented together in `?grep`), however I recommend using the `stringr` package for it's consistency of syntax. For lots and lots of details about R's regular expressions, see `?regexp`.

Regular expressions will be useful for this problem. A function will attempt to match a regular expression in a (vector of) string(s), like the `CTRL-F` "find" functionality. What is *done* with the match depends on the function; common tasks are to *detect* (return `TRUE` if there is a match), *extract* (pull out the match from the string) or *replace* (replace the match with something else). Basic matches are easy, for example if you want to look for an "a", the regular expression is simply `"a"`.

There are lots of special meta-characters, for example a period `.` is used to denote a single character. I keep a regular expressions cheat sheet taped up near my desk which reminds me of the meta-characters. (Note, if you want to use fancy, Perl-flavored regex with the `stringr` library, wrap your regex string in `perl()`. This is useful for complicated patterns where, for example, you want look-ahead or look-behind.)

Because a period is a meta-character, if you want to match a period with a regular expression *not in R* you must escape it with a backlash, `\.`. The backslash indicates that you don't want to use it's special meta-meaning. *In R*, `\` is also an escape character, which means for a regular expressions in R, you must escape every backslash. That is, to match a period using regular expressions in R, you would use `\\.`. Regular expressions often seem dense (at least to those of us who don't use them all the time), and doubling the backslashes makes things even worse. Somewhere on Stack Overflow I saw good advice for troubleshooting regular expressions in R: "just keep adding backslashes until it works."

If you have trouble with this, run the examples at the bottom of the `stringr` help pages. They usually do a pretty good job illustrating the functions.

A. Make a histogram of the *number characters* in the strings. (See `?str_length` or `?nchar`.) *Optional:* guess which distribution I used to simulate the length of these strings. Looking at the mean and the variance of the number of letters will provide a good hint.

B. Create a new variable, `rlets`, just has the letters (not the numbers) from `rstrings`. You could do this either by extracting the letters (see `?str_extact` and `?str_extract_all`) or by deleting the numbers (see `?str_replace` and `?str_replace_all`).

C. Make a barplot of the letters in present in all the strings (`?str_split`). (One histogram, not one per string. Put letter on the x-axis and counts on the y-axis.)

D. Which of the original strings start with a letter? (`?str_detect`)

E. Merge all the strings in `rlets` into one string separated by spaces. (See `?paste`, and make sure you understand the difference between the `sep` and `collapse` arguments.)

4. Versatility

Simulate a vector `cauch` of length 100 from the Cauchy distribution (use the default location of 0 and scale of 1). You will create a new vector `cauch.bins` that indicates whether each value of `cauch` is within 1 units of 0 with the label `"within 1"`, within 10 units of 0 with the label `"within 10"`, or farther away from 0 with the label `far`.

(a) Create `cauch.bins.ie` using `ifelse()`.

(b) Create `cauch.bins.sub` by initializing it to `NA`, then using `[` to fill in the categories one-at-a-time.

(c) Create `cauch.bins.cut` by using the `cut` function.

(d) Show that your results from a-c are all the same by using the `identical` function. (`identical` makes pairwise comparisons.)

Book problems

None! (Even though there is an Exercises section for Appendix B. What kind of book has exercises in the appendices?)