The purpose of this section is to showcase the ability to scrape and process web data using R. The section notes draw heavily from a post on a great blog by Pascal Mickelson and Scott Chamberlain, two biologists and experienced R users.

Suppose we want to find the number of available economics journals. There are too many. Definitely. But suppose we want to find out just how many. To do this, we can visit crossref.org, which is a citation-linking network with a list of all journals and their Digital Object Identifiers (DOIs). We will query the list from within R and then parse the returned content to list journals with certain attributes. For this, we'll need to load the following libraries:

```
library(XML)
library(RCurl)
library(stringr)

options(show.error.messages = FALSE)
```

Note the useful option for code with loops, especially loops over remote queries, to globally suppress error messages. The next step is to repeatedly query crossref.org for journal titles. Try to copy and paste the base URL address (`baseurl`) into your browser: `http://oai.crossref.org/OAIHandler?verb=ListSets`. The result is a long XML form. The function `getURL` in the following code pulls this response into R as a string, and the outer functions `xmlParse` and `xmlToList` convert the output into an R data structure. There are too many entries to fit into a single query, so the `while` loop continues to query until there are no more results. The final results are stored in `nameslist`. We will review the code in section.

```
token <- "characters"
nameslist <- list()

while (is.character(token) == TRUE) {

  baseurl <- "http://oai.crossref.org/OAIHandler?verb=ListSets"

  if (token == "characters") {
    tok.follow <- NULL
  } else {
    tok.follow <- paste("&resumptionToken=", token, sep = "")
  }

  query <- paste(baseurl, tok.follow, sep = "")

  xml.query <- xmlParse(getURL(query))
  set.res <- xmlToList(xml.query)
  names <- as.character(sapply(set.res[["ListSets"]], function(x) x[["setName"]]))
  nameslist[[token]] <- names

  if (class(try(set.res[["request"]][[".attrs"]][["resumptionToken"]])) == "try-error") {
    stop("no more data")
  }
  else {
    token <- set.res[["request"]][[".attrs"]][["resumptionToken"]]
  }

}
```

How many journal titles are collected by this query? We first concatenate the results into a single list, and then find the total length:

```
allnames <- do.call(c, nameslist)
length(allnames)
```

```
: [1] 27289
```

Now, suppose that we are looking for just those journals with *economic* in the title. We rely on regular expressions, a common way to parse strings, from within `R`. The following code snippet detects strings with some variant of *economic*, both lower- and upper-case, and selects those elements from within the `allnames` list.

```
econtitles <- as.character(allnames[str_detect(allnames, "^[Ee]conomic|\\s[Ee]conomic")])
length(econtitles)
```

```
: [1] 461
```

What in the hell? So many! I suppose that this is a good thing: at least one of the 461 journals should accept my crappy papers. If I blindly throw a dart in a bar, it may not hit the dartboard, but it will almost certainly hit one of the 461 patrons. Here is a random sample of ten journals:

```
sample(econtitles, 10)
```

```
 [1] "Computer Science in Economics and Management"
 [2] "Advances in Theoretical Economics"
 [3] "Economic Systems"
 [4] "Journal of Agricultural Economics"
 [5] "Economic Quality Control"
 [6] "Contributions in Economic Analysis & Policy"
 [7] "Journal of Economic Interaction and Coordination"
 [8] "Asian Economic Journal"
 [9] "International Review of Economics"
[10] "African Journal of Economic Policy"
```

What are other things we can do with the data? Suppose we wanted to compare the relative frequencies of different subjects within journal titles. This offers a decent example for section, since we can refactor some of the code we already developed — a useful skill for writing clean code. We have already figured out how to count the number of journals for a particular regular expression. We can refactor the code into the following function, which accepts a regular expression and returns the length of the collection containint matching strings:

```
countJournals <- function(regex) {
  titles <- as.character(allnames[str_detect(allnames, regex)])
  return(length(titles))
}
```

Now the tedious process of converting a list of subjects into the appropriate regular expressions. If we have time, we'll write a function to do this conversion for us:

```
subj = c("economic", "business", "politic", "environment", "engineer", "history")
regx = c("^[Ee]conomic|\\s[Ee]conomic", "^[Bb]usiness|\\s[Bb]usiness",
  "^[Pp]olitic|\\s[Pp]olitic", "^[Ee]nvironment|\\s[Ee]nvironment",
  "^[Ee]ngineer|\\s[Ee]ngineer", "^[Hh]istory|\\s[Hh]istory")

subj.df <- data.frame(subject = subj, regex = regx)
```

Finally, we simply apply our refactored function to the regular expressions, and graph the result:

```
subj.df[["count"]] <- sapply(as.character(subj.df[["regex"]]), countJournals)
(g <- ggplot(data = subj.df, aes(x = subject, y = count)) + geom_bar())
```

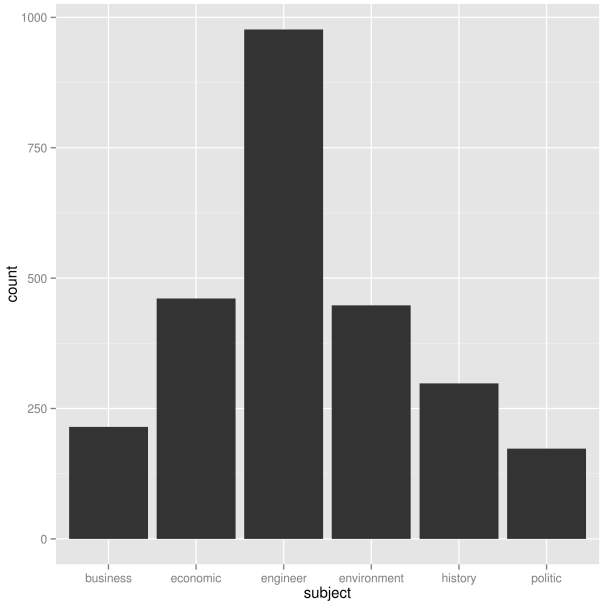

Figure 1: Journal count by subject search