The purpose of this section is to showcase the ability to scrape and process web data using `R`. We'll also talk a tiny bit about error handling along the way.

Web scraping

So far, we've used `R` to download PRISM weather data, a couple of polygon shapefiles, and data on population by ZIP code tabulation area. I do not really consider what we've done so far web scraping, as it is simply downloading data *that was intended to be downloaded as data.* I think web scraping becomes web scraping when the primary purpose of the "data" that is obtained by loading a URL, is to be viewed in a browser, or sampled for web pages, rather than downloaded and analyzed in statistical programs.

The first part of the section notes draw heavily from an old post[1] on a great blog by Pascal Mickelson and Scott Chamberlain, two biologists and experienced `R` users.

Note that the code below actually takes quite a while to run, since we need to download multiple large files from the server. If you are reading this in section while I'm demonstrating it, I do not recommend you run it! We don't want to crash their server.

XML

The simplest web information out there will find it's way to you in XML form. XML is a data format that's used a lot for storing data which is used on webpages, and many websites have public XML code. Our first example will look at downloading some of this and turning it into something useful.

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 `www.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(/install) the following libraries:

```
library(XML)
library(RCurl)
library(stringr)
library(ggplot2)

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 idea behind scraping data is to take web data like this and turn it into a dataset we can analyze. The function `getURL` from `RCurl` in the following code pulls this

[1] `https://github.com/sckott/sckott.github.com/blob/master/_drafts/2012-08-30-get-ecoevo-journal-titles.md`

response into `R` as a `character` string, and the outer functions `xmlParse` and `xmlToList` convert the output into a `list`. 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`.

```
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

  tryCatch(token <- set.res[["request"]][[".attrs"]][["resumptionToken"]], error = function(e){
    message("no more data")
    token <<- NULL
  })
}
```

This looks confusing. Don't panic, though. A `while` loop runs repeatedly as long as the condition in parentheses is satisfied. Here's what it does:

1. The `while` loop creates a url, `query` to pull from the URL above.
2. Gets the results, `xml.query`, and runs `xmlToList()` to turn those XML results into a list of journals, `set.res`.
3. Passes the names of the journals, `names`, to the list of names, `nameslist`. The list identifier for each set of names is `token`.
4. Tries to set the `token` variable equal to the `"resumptionToken"` field passed in the XML.
   (a) If successful, restarts the loop using the new `token`.
   (b) If unsuccessful, ends the loop.

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 <- unlist(nameslist)
length(allnames)
```

```
## [1] 38331
```

```
head(allnames)
```

```
##                           characters1                           characters2
##   "Journal of Neuroscience Research"               "Annals of Neurology"
##                           characters3                           characters4
##                     "Geoarchaeology"   "Applied Organometallic Chemistry"
##                           characters5                           characters6
## "Journal of Applied Polymer Science"    "Journal of Applied Econometrics"
```

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.

The ^ symbol says "the string must start here". The \\s means whitespace. The [] lets you specify a set of letters you are looking for, e.g., [Ee] means capital E OR lowercase e.

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

```
## [1] 623
```

I can't take credit for the following joke. But it's great nonetheless:

> *What in the hell? So many! I suppose that this is a good thing: at least one of the 623 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 623 patrons.* - Patrick Baylis

Here is a random sample of ten journals:

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

```
##  [1] "Journal of Asian Economics"
##  [2] "Supreme Court Economic Review"
##  [3] "Journal of Agricultural and Applied Economics"
##  [4] "Annals of Financial Economics"
##  [5] "Asian Journal of Research in Business Economics and Management"
##  [6] "Explorations in Economic History"
##  [7] "Handbook of Mathematical Economics"
##  [8] "Journal of Medical Economics"
##  [9] "Ecological Economics"
## [10] "Applied Economics Letters"
```

HTML

Now to HTML, which is what most webpages display their contents in. Google is definitely your friend for each situation where you're trying to work with actual webpages. We'll quickly go through

an example of downloading oil futures prices from a webpage.

```
url <- "http://www.cmegroup.com/trading/energy/crude-oil/brent-crude-oil.html"
tables <- tryCatch(readHTMLTable(url), error = function(e){
  message("Please check internet connection")
})
n.rows <- unlist(lapply(tables, function(t) dim(t)[1]))
oilPrices <- tables[[which.max(n.rows)]]
head(oilPrices)
```

```
##      Month  Options           Charts Last Change Prior Settle Open High
## 1 JUN 2015 JUN 2015 Show Price Chart    -      -        64.64    -    -
## 2 JUL 2015 JUL 2015 Show Price Chart    -      -        65.35    -    -
## 3 AUG 2015 AUG 2015 Show Price Chart    -      -        65.99    -    -
## 4 SEP 2015 SEP 2015 Show Price Chart    -      -        66.58    -    -
## 5 OCT 2015 OCT 2015 Show Price Chart    -      -        67.09    -    -
## 6 NOV 2015 NOV 2015 Show Price Chart    -      -        67.57    -    -
##   Low Volume Hi / Low Limit                 Updated
## 1   -      0       No Limit 01:06:49 CT 29 Apr 2015
## 2   -      0       No Limit 18:30:41 CT 28 Apr 2015
## 3   -      0       No Limit 01:06:49 CT 29 Apr 2015
## 4   -      0       No Limit 01:06:49 CT 29 Apr 2015
## 5   -      0       No Limit 18:30:41 CT 28 Apr 2015
## 6   -      0       No Limit 01:06:49 CT 29 Apr 2015
```

Easy! This time. The above code relies on the oil price table being the largest on the page. Sometimes the data you want isn't sitting on the page in a table at all! In these cases, you might have to use complicated regular expressions and string parsing methods. Yuck. Like I said, Google is your friend; failing that, ask on stackoverflow.

Selenium

Now the craziest tool of all, Selenium. Selenium is a browser automation tool, so it allows you to act (almost) exactly like a human would, clicking and typing in boxes. There are seldom cases where something like this is really necessary. The only time I have used it is to try and automate arbitrage trading on a prediction website. I succeeded in automating the trading, but found that the opportunities amounted to a few cents a day, which wouldn't have got me up to the $5 trading fee threshold.

On my machine, the `RSelenium` package automatically downloads the Selenium driver when you first use it. Hopefully this happens for you too. The example in the `collectAvailableCategories.R` file is something I cooked up for fun last Friday night instead of going out, because I'm trying to drink less. It looks on the USDA NASS online interface and determines what level of aggregation is available for each data type. Like I said, it's usually not necessary to use Selenium to achieve any particular goal, and this example is no exception - you can just download the full data set and go from there.