Lecture 22: Importing Data from the Web

36-350, Fall 2013

11 November 2013

Sometimes data is online in a nice, machine-readable format. Then it is very easy to import it into R. We have been doing this since the first assignment, without any fuss:

```
rain <- read.table("http://www.stats. uwo.ca/faculty/braun/data/rnf6080.dat")
```

All the commands like `read.table`, `read.csv`, etc., can take a URL just as easily as the name of a file on the local system.

Some webpages combine machine-readable formats with other parts which are intended to be read by human beings. (See, for instance, the example with the ANSS earthquake catalog in the handout on regular expressions.) Sometimes this can be dealt with simply, through arguments like `skip` to `read.table` and its kin. In slightly more complex cases, one needs to use some very basic text filtering, possibly involving regular expressions. (See, again, the example with the earthquake catalog.) Alternatively, especially if the file is not too large, save it, open it in a text editor, trim out everything which isn't formatted for the machine, and have R read the edited file.

Much of the time, however, the information one wants on the web is *not* in machine-readable format; rather it is intended for human consumption. For instance, in 2003 Valdis Krebs and collaborators published a famous study of networks of political books on Amazon, where two books are linked if they tend to be purchased by the same people[1]. Amazon displays this information for each book; to avoid needless controversy, I will show it for our textbook rather than a political tract (Figure 1).

This way of displaying the information is designed to be very appealing to human beings, but it's not convenient for the computer, especially if one wants to deal with dozens, or even thousands, of books. The craft of web scraping is about taking such *relatively* unstructured information, and turning it into rigidly formatted data.

[1] See Krebs's page at `http://www.orgnet.com/divided.html`, and references therein.

Figure 1: Part of Amazon's "also bought" information for Matloff's *The Art of R Programming*, as of November 2012.

```
<div class="shoveler" id="view_to_purchaseShvl">
    <div class="shoveler-heading">
        <h2>Customers Who Viewed This Item Also Bought</h2>
    </div>
<div class="shoveler-pagination" style="display:none">
<span> </span>
<span>
Page <span class="page-number"></span>  of  <span class="num-pages"></span>
<span class="start-over"> (<a href="#" onclick="return false;"
 class="start-over-link">Start over</a>) </span>
</span>
</div>
    <div class="shoveler-button-wrapper" id="view_to_purchaseButtonWrapper">
        <a class="back-button" href="#Back" style="display:none"
 onclick="return false;"><span class="swSprite
 s_shvlBack"><span>Back</span></span></a>
        <div class="shoveler-content">
            <ul tabindex="-1">
<li>
  <div class="new-faceout"  id="view_to_purchase_0596809158">
<a href="http://www.amazon.com/Cookbook-OReilly-Cookbooks-Paul-Teetor/dp/0596809158/ref=pd_vtp_b_1"
  class="sim-img-title" > <div class="product-image">
                      <img
 src="http://ecx.images-amazon.com/images/I/51ODN7aoMqL._SL500_PIsitb-sticker-arrow-big,TopRight,35,-73_OU01_SS100_.jpg"
 width="100" alt="" height="100" border="0" />
                    </div>
                    R Cookbook (O&#39;Reilly Cookbooks) </a>
    <div class="byline">
        <span class="carat">&#8250</span>  <a
 href="/Paul-Teetor/e/B004OOSH4E/ref=pd_vtp_b_bl_1">Paul Teetor</a>
    </div>
    <div class="rating-price"><span class="rating-stars"><span
     class="crAvgStars" style="white-space:no-wrap;"><span
     class="asinReviewsSummary" name="0596809158"><a
     href="http://www.amazon.com/Cookbook-OReilly-Cookbooks-Paul-Teetor/product-reviews/0596809158/ref=pd_vtp_b_cm_cr_acr_img_1">
     class="swSprite s_star_4_5 " title="4.4 out of 5 stars" ><span>4.4 out of
     5 stars</span></span></a> </span>(<a
     href="http://www.amazon.com/Cookbook-OReilly-Cookbooks-Paul-Teetor/product-reviews/0596809158/ref=pd_vtp_b_cm_cr_acr_txt_1">
    <div class="binding-platform">
        Paperback
    </div>
    <div class="pricetext">
        <span class="price">$32.68</span>
    </div>
  </div>
</li>
```

Figure 2: The beginning of the HTML source of the part of the Amazon page which displays the also-bought information. (A large number of blank lines have been deleted to save space, and I've added broken up some, but not all, of the very long lines.)

1 How to Scrape

Figure out what *exactly* you want to learn from the page This typically involved making an initially-vague idea about the desired information more precise.

To build a network of affiliated books, we want to know which books are likely to be bought by the same people. Amazon actually provides *four* kinds of information like this: "frequently bought together" items, based on what gets purchased together (i.e., included in the same order); "customers who viewed this item also bought" (shown above); "customers also bought items by" (which gives authors rather than specific books); and "what other items do customers buy after viewing this item". We will focus on the second one.

Understand how that information is organized on the webpage Specifically:

1. Find the information on *one* webpage.
2. What cues, as a human reader, did you use to find the desired information?
3. Where does that information appear in the *source* text of the webpage?

In the case of Matloff's book, scanning down the source text of the Amazon page finds a section which begins like Figure 2. This encodes a *lot* of information about each title, much of it related to catching the viewer's eye (and wallet) and of no essential concern to us, but now that we know about regular expressions, we can extract just what we need from it.

However, at the end of this section there is the following bit of HTML[2]:

```
<div id="view_to_purchaseSimsData" class="sims-data" style="display:none"
 data-baseAsin="1593273843" data-pageId="1593273843sr_1_1"
 data-popunder="true" data-reftag="pd_vtp_b"
 data-wdg="book_display_on_website"
 data-widgetName="view_to_purchase">0596809158,1119962846,1439810184,
 0387938362,0123814855,059680170X,1935182390,144931208X,0387790535,0387981403,
 0470510242,1439831769,111816430X,1446200469,159420411X,0470944889,1449311520,
 1441998896,0596802358,1441926127,1449319793,0321629302,0123748569,1590282418,
 0596153937,0538497815,1441977864,1599947250,0596529325,141297514X,0970601972,
 041587291X,1461406846,1584884509,0387747303,0387781889,1420070576,0615684378,
 1587788802,0387781706,1600490069,0387922970,0878933913,0470414359,193518220X,
 0615653634,1449303714,1441930086,0942154916,0521762936,052168689X,0123850819,
 0387848576,0387886974,1592576346,1617290181,0521518148,1449388345,1420068725,
 0470475358,0123814790,1449309038,0199230234,0321356683,1441915753,0470650931,
 1849513066,0321776402,0062731025,1441996494,0486653552,1400069289,0387759581,
 0970601980,0470141158,0262017180,0805816569,0387402721,0374275637,1449319262,
 0387759689,158488388X</div>
```

[2]It is actually all one line; I've added the linebreaks for printing.

This forbidding-looking block just gives the product ID numbers (ISBNs) of all the associated books. Realizing that that's what these are requires knowing that Amazon tracks books by ISBNs, and that these are ISBNs; but if we were uncertain, we might notice that the first numeral-string in this block, `0596809158`, is also the one which appears repeatedly in the display of the first associated book (Figure 2). As usual, though, the more one knows about the actual subject-matter, whether it be the book trade or bird tracking[3], the more sense one can make of the data.

At this stage, it may be worthwhile to see if one can figure out, from the source of the webpage, how to directly access the data files which are being processed to produce the humanly-readable page. (Often, but not always, this will be a database in the strict sense, as we'll come to in a few lectures.) Amazon, understandably enough, doesn't provide very ready access, or even cues to such access, but sites which are less worried about commercial rivals, or just less cautious, may provide more clues.

Build a function to automate your extraction of the information from one webpage Generally, this means using regular expressions based on what you've discovered about the organization of the page's source.

It helps to build up the regular expression gradually:

1. Use `paste` to combine sub-expressions;
2. Use parenthesized capture groups to extract particulars
3. Test your ability to extract *parts* of the desired information first, rather than everything at once (say, *one* related book)

You should use your humanly-extracted information from particular pages as test cases, and sore your results in a data frame.

Several of our design principles are coming together here. One is modular design: if there are multiple pieces of information we want from the page, figure out how to get them in stages, rather than having to come up with a single huge procedure all at once. Another is testing: because you have studied *this* web page already, you know what answer a correctly-working procedure should give when applied to the source, and can use that to guide your coding. Finally, storing in a data frame takes us all the way back to the idea that our data structures should keep related values together.

For the book-network example, the natural data frame would contain pairs of associated books, representing the information that "people who bought ISBN X also bought ISBN Y". (Note that this is not necessarily a symmetric relationship.)

[3] For a spectacular example of how failing to understand the measurement process can lead even excellent scientists astray, see Edwards *et al.* (2007).

Figure out all the relevant pages, and apply your function to them

1. Come up with a set of starting points. (E.g., books of known political valence.)
2. Follow, or generate, links to relevant pages. (One can build a URL for Amazon's page on a given book from its ISBN.) Determine whether or not to follow a link at all. (Amazon sells many other things besides books, and while there *might* be a political valence to, say, packs of incandescent light bulbs, it would also be reasonable to ignore such things.)
3. Determine whether those pages are of the same format, in which case they should be processed as before, or if they have a different format and need different handling. (All Amazon book pages are of basically the same format, but there are, for instance, review and discussion pages, which might be relevant for other projects.)
4. Determine when to stop. (A fixed number of pages, a fixed number of links, when memory is full, when a website is completely surveyed...)

Improve as you go Like everything else in programming, Web scraping is iterative. As you go along, you may discover cases you did not expect, where your initial extraction function does not work properly. But bugs become tests: record the problematic page, and write a test case for it, modifying your extraction function until it passes all your tests.

1.1 Data Storage and Reproducibility

Websites change, move, and disappear. If it's important to be able to reproduce your results on the same data later, it can be very worthwhile to store all the webpages you work with, and have your code work with the stored copies. This may however lead to legal issues regarding copyright, etc., which are outside my competence to advise you on.

2 Exercises

1. Write code which will extract the ISBNs of all the books associated with Matloff's book. *Hint:* try extracting the solid block of ISBNs first, and then get all of the individual ISBNs out of the block.
2. How can you generate an Amazon webpage for a book if you only know its ISBN and not its title?
3. How can you extract the title of a book from its Amazon webpage?
4. How can you tell, from an Amazon webpage, if the product being sold there is a book?

5. Put the pieces together: how would you build a network of political book affiliations?

References

Edwards, Andrew M., Richard A. Phillips, Nicholas W. Watkins, Mervyn P. Freeman, Eugene J. Murphy, Vsevolod Afanasyev, Sergey V. Buldyrev, M. G. E. da Luz, E. P. Raposo, H. Eugene Stanley and Gandhimohan M. Viswanathan (2007). "Revisiting Lévy flight search patterns of wandering albatrosses, bumblebees and deer." *Nature*, **449**: 1044–1048. URL `http://polymer.bu.edu/hes/articles/epwfmabdrsgv07.pdf`. doi:10.1038/nature06199.