Web Scraping, part 1: files and APIs

Web scraping is extracting information from webpages, usually (but not always) as tables of data that you can save to csv files, json/xml files or databases.

Design it first, then scrape it

When you start on any piece of code, try asking yourself some design questions first; definitely do this if you're thinking about something as potentially complex as web scraping code.   So you've seen a dataset hiding in a website - it might be a table of data that you need, or lists of data, or data spread across multiple pages on the site. Here are some questions for you:

1. Do you need to write scraper code at all?

- Is the dataset very small - if you're talking about 10 values that don't get updated, writing a scraper will take longer than just typing all the numbers into a spreadsheet.
- Has the site owner made this data available via an API (search for this on the site) or downloadable datafiles, either on this site or connected sites? APIs are much much faster (and less messy) to use than writing scraper code.
- Did someone already scrape this dataset and put it online somewhere else? Check sites like scraperwiki.com and datahub.io.

2. What do you need to scrape?

- What format is the data currently in? Is it on html webpages, or in documents attached to the site? Is it in excel or pdf files?
- Does the data on this site get updated? e.g. do you need to scrape this data regularly, or just once?

3. What's your maintenance plan?

- Where are you planning to put the data when you've scraped it? How are you going to make it available to other people (this being the good and polite thing to do when you've scraped out some data)? How will you tell people that the data is one-off or regularly updated, and who's responsible for doing that?
- Who's going to maintain your scraper code? Website owners change their sites all the time, often breaking scraper code - who's going to be around in a year, two years etc to fix this?

Reading files

Okay. Questions over. Let’s get down to business, working from easy to less-easy, starting with "you got to the website, there's a file waiting for you to download and you only need to do this once".

You've got the data, and it’s a CSV file

Lucky you: pretty much any visualization package and language out there can read CSV files. You’ll still have to check (e.g. look for things like messed-up text, be suspicious if all the biggest files the same size, etc) and clean the data (e.g. check that your date columns all contain formatted dates, you have the right number of codes for gender - and no, its not always two - etc) but as far as scraping goes, you’re done here.

You’ve got the data, and it’s a file with loads of brackets in it

Also, the file extension (the part after the last “.” in a filename) is probably “json". This is a json file - not all data packages will read in this format, so you might have to convert it to CSV (and it might not quite fit the rows-by-columns format so you’ll have to do some work there too), but again, no scraping needed.

You’ve got the data, and it’s a file with loads of s in it

Either you’ve got html files (look for obvious things like HTMl tabs: , ,

,

, etc and text outside the opening and closing brackets) or you’ve got an xml file. Another big hint is if the file extension is “.xml”. Like json, xml is read in by many but not all data visualization packages, and might need converting to csv files; a few quirks make this a little harder than converting json, but there’s a lot of help out there on this online.

You’ve got the data and it’s a PDF file

Ah, the joys of scraping PDF files. PDF files are difficult because even though they *look* like text files on your screen, they’re not nearly as tidy as that behind the scenes. You need a PDF converter: these take PDF files and (usually) convert them into machine-readable formats (CSV, Excel etc). This means that the 800-page PDF of data tables someone sent you isn't necessarily the end of your plan to use that data.

First, check that your PDF can be scraped. Open it, and try to select some of the text in it (as though you were about to cut and paste). If you can select the text, that's a good sign - you can probably scrape this pdf file. Try one of these:

- If you've got a small, one-off PDF table, either type it in yourself or use Acrobat's "save as tables" function
- If you've got just one large PDF to scrape, try a tool like pdftables or cometdocs
- If you want to use open source code - and especially if you want to contribute to an open-source project for scraping PDF data, use Tabula.

If you can't select the text, that's bad: the PDF has probably been created as an image of the text - your best hope at this point is using OCR software to try to read it in.

Using APIs

An Application Programming Interface (API) is a piece of software that allows websites and programs to communicate with each other.

So how do you check if a site or group has an API? Usually a Google search for their name plus "API" will do, but you might also want to try http://api.theirsitename.com/ and http://theirsitename.com/api (APIs are often in these places on a website). If you still can't find an API, try contacting the group and asking them if they have one.

Using APIs without coding

APIs are often used to output datasets requested using a RESTful interface, where the dataset request is contained in the address (URL) used to ask for it. For example, http://api.worldbank.org/countries/all/indicators/SP.RUR.TOTL.ZS?date=2000:2015 is a RESTful call to the World Bank API that gives you the rural population (as a percentage) of all countries for the years 2000 to 2015 (try it!). If you've entered in the RESTful URL and you've got a page with all the data that you need, you don't have to code: just save the page to a file and use that. Note that you could also get the information on rural populations from http://data.worldbank.org/indicator/SP.RUR.TOTL.ZS but you won't be able to use that directly in your program (although most good data pages like this will also have a "download" button someone too).

APIs with code

Using code to access an API means you can read data straight into a program and either process it there or combine and use it with other datasets (as a "mashup"). I'll use Python here, but more other modern languages (PHP, Ruby etc) have functions that do the same things. So, in Python, we're going to use the requests library (here's why).

```
import requests  worldbank_url = "
http://api.worldbank.org/countries/all/indicators/SP.RUR.TOTL.ZS?date=200
0:2015"  r = requests.get(github_url)
```

Erm. That was it. The variable r contains several things: information about whether your request to the API was successful or not, error codes if it isn't, the data you requested if it is. You can quickly check for success by seeing if r.ok is equal to True; if it is, your data is in r.text; if it isn't, then look at r.status_code and r.text, take a big deep breath and start working out why (you'll probably see 200, 400 and 500 status codes: here's a list to get you started).

Many APIs will offer a choice of formats for their output data. The World Bank API outputs its data in XML format unless you ask for either json (add "&format=json" to the end of worldbank_url) or CSV (add "&format=csv"), and it's always worth checking for these if you don't want to handle a site's default format.

Sometimes you'll need to give a website a password before you can get data from its API. Here, you add a second parameter to requests.get, "authenticating" that it's you using the API:

```
import requests  r = requests.get(
'https://api.github.com/user', auth=('yourgithubname',
 'yourgithubpassword'))  dataset = r.text
```

That's enough to get you started. The second half of this post is on scraping websites when you don't have an API. In the meantime, please use the above to get yourself some more data. Places you could start at include HDX and datahub.io (these are both CKAN APIs).