

## How We Learned to Stop Worrying and Love Web Scraping

Nicholas J. DeVito, Georgia C. Richards, and Peter Inglesby

In research, time and resources are precious. Automating common tasks, such as data collection, can provide efficiency and repeatability, leading in turn to increased productivity and output. You will also end up with a sharable and reproducible method for data collection that can be verified, used, and expanded upon by others -- in other words a computationally reproducible data-collection workflow.

One of our recent projects analysing [coroners reports](#) to prevent future deaths required downloading over 3,000 PDFs to search for opioid-related deaths, a huge data collection task. In discussion with the larger team, we decided this was a good candidate for automation. With a few days of work, we were able to [write a computer program](#) that could quickly, efficiently, and reproducibly collect all the PDFs and create a spreadsheet that documented each case.

Such a tool is called a “web scraper” and our lab uses them regularly. We use them to collect information from [clinical trial registries](#), and also to enrich our [OpenPrescribing.net dataset](#) which tracks [primary care prescribing in England](#) -- tasks that would range from annoying to impossible without the help of some relatively simple code.

In the case of our coroner reports project, we could manually screen and save about 25 case reports every hour. Now, our program can save more than 1000 cases/hr while we work on other things, a 40-fold time savings. It also opens new opportunities for collaboration, because we can share the resulting database -- and we can keep that database up-to-date by rerunning our program as new PDFs are posted.

Here we offer some basics about web scraping and how you can start using it in your research projects.

### How does scraping work?

Web scrapers are computer programs that extract information from - that is, “scrape” - web sites. The structure and content of a webpage are encoded in Hypertext Markup Language (HTML), which you can see using your browser’s “view source” or “inspect element” functions. A scraper also understands HTML, and is able to parse and extract information from it. For example, you can program your scraper to extract specific fields of information in a website table to save in a spreadsheet file or download documents linked on the page.

A common scraping task would involve iterating over every possible URL from [www.example.com/data/1](#) to [www.example.com/data/100](#) (sometimes called “crawling”) and storing what you need from each page without the risk of human extraction errors. Once your program is written, you can recapture this data whenever you need, assuming the structure of the website stays mostly the same.

### How do I get started?

Some scraping tasks don't require programming. When you visit a web page in your browser, off-the-shelf browser extensions such as [webscraper.io](https://www.webscraper.io) let you click on the elements of the page that contain the data you're interested in. They can then automatically parse the relevant parts of the HTML and export the data as a spreadsheet. These are helpful but may also have certain functional limitations.

The alternative is to build your own scraper - a more difficult process, but one that offers greater control. We use [Python](#), but any modern programming language should work (for specific packages, [Requests](#) and [BeautifulSoup](#) work well together in Python; for R, try [rvest](#)). It's worth checking to see if anyone else has already written a scraper for your data source. If not, there's [no shortage](#) of [resources](#) and [free tutorials](#) to help you get started [no matter your preferred language](#).

As with most programming projects, there will be some trial-and-error, and different websites may use novel data structures or variations in how their HTML is implemented that will require tweaks to your approach. Yet this problem-solving aspect of development can be quite rewarding. As you get more comfortable with the process, overcoming these barriers will start to seem like second nature.

But be advised: depending on the number of pages, your internet connection, and the website's server, a scraping job could still take days. If you have access and know-how, running your code on a private server can help. On a personal computer, make sure to prevent your computer from sleeping and disrupting the internet connection. Also, think carefully about how your scraper can fail. Ideally, you should have a way to log failures so you know what worked, what didn't, and where to investigate further.

## Things to consider

*Can you get the data an easier way?*

Scraping all 300,000+ records off of ClinicalTrials.gov every day would be a massive job for our [FDAAA TrialsTracker](#) project. Luckily, ClinicalTrials.gov makes their full dataset [available for download](#); our software simply grabs that file once a day. We weren't so lucky with the data for our [EU TrialsTracker](#) and [scrape the EU registry monthly](#).

If there's no bulk download available, check to see if the website has an application programming interface (API). An API lets software interact with a website's data directly, rather than requesting the HTML. This can be much less burdensome than scraping individual web pages, though there may be a fee associated with API access (see, e.g., [Google's Map API](#)). In our work, the [PubMed API](#) is often useful. Alternatively, check if the website operators can provide the data to you directly.

*Can this website be scraped?*

Some websites don't make their data available directly in the HTML and may require some more advanced technique (resources like [StackOverflow](#) can help for specific questions). Other websites include protections like captchas and anti-denial-of-service ([DoS](#)) measures. A few

websites simply don't want to be scraped and are built to discourage it. It's also common to allow scraping but only if you follow certain rules, usually codified in something called a [robots.txt file](#).

#### *Are you being a courteous scraper?*

Every time your program requests data from a website, the underlying information needs to be "served" to you. While you can only move so quickly in a browser, a scraper could potentially send hundreds to thousands of requests per minute. Hammering a web server like that can slow, or entirely bring down, the website (essentially performing an unintentional DoS attack). This may get you temporarily, or even permanently, blocked from the website and you should take care to minimise the chances of harm. For instance, you can pause your program for a few seconds between each request. (Check the site's robots.txt file to see if it specifies a desired pause length.)

#### *Are the data restricted?*

Be sure to check for licensing or copyright restrictions on the extracted data. You might be able to use what you scrape, but it's worth checking that you can also legally share it. Ideally the website content license will be readily available. Whether or not you can share the data, you should share your code using services such as [GitHub](#) -- this is good open-science practice and ensures that others can discover, repeat, and build upon what you've done.

### **Conclusion**

We strongly believe more researchers should be developing code to help conduct their research and then [sharing it](#) with the community. If manual data collection has been an issue for your project, a web scraper may be the perfect solution and a great beginner coding project. Scrapers are complex enough to teach important lessons about software development, but common and well-documented enough so that beginners should feel confident experimenting.

Using web scrapers, we've saved hours of time, examined new and important research questions, and built reproducible methods that others can reuse, learn from, and expand upon. It can feel like a superpower when you run some relatively simple code on your computer and have it interact with the outside world. We hope this piece has given you the confidence to get started!

### **Bios**

[Nicholas J. DeVito](#) and [Georgia C. Richards](#) are doctoral candidates and researchers, and [Peter Inglesby](#) is a software engineer, at Ben Goldacre's [EBM DataLab](#) at the University of Oxford, UK.