STATS 769
Web Scraping

Paul Murrell

The University of Auckland

August 22, 2019

Overview

- This section of the course (ONE lecture) explores how to access data from the web.

- One problem is that web resources present a variety of different interfaces (static web pages, web forms, and web APIs).
- Another problem is that web resources are delivered in a variety of different formats (flat text files, HTML, XML, and JSON).

Web versus Filesystem

For simple situations, R conveniently treats accessing a resource on the web like accessing a file on your filesystem.

- File-reading functions like `read.csv()` work seamlessly with locations on the file system and files on the web.

There are also functions specifically aimed at downloading a file from the web.

- `download.file()` can be used to download a file.
- `url()` can be used to open a "connection" to a web resource.

Web versus Filesystem

Simple web scraping is just as simple as reading files from the file system.

- Use text processing functions to generate URLs.
- Use `read.csv()` or `download.file()` to read the URLs.

Web versus Filesystem

Accessing a resource on the web can be more complicated:

- We may need to provide a username and password to access a resource.
- We may want to send more than just a URL to the web server (e.g., fill in a web form rather than just get a static URL).
- The resource may not be a simple flat text file (e.g., HTML, JSON, or XML).
- The web page might be "dynamic" (e.g., generated by client-side javascript)

We will look at some R packages that allow us to work with these more complex web scraping scenarios.

HTTP

- When we communicate with a web server, we send a **request** and receive a **response**.
- The request typically contains a URL for the resource we want, but it can also contain other information (e.g., username and password).
- The response consists of a header and a body.
- The body of a response could be the HTML for a web page, but it could be something else (e.g., a CSV file or even a binary object); the header contains information about the format of the body.
- The request can also have a body (e.g., to include information for an HTML form).

Authenticated HTTP Requests

The **httr** package allows us to request a URL and supply username and password.

- `response <- GET(url, authenticate(user, pwd))`

The package also has functions for unpacking the response.

- `content(response, as="parsed")`

Web Forms (GET)

A web page that contains a **form** requires us to send additional information as part of our request. A form is typically entered by hand, but it is sometimes possible to automate the process; there are two scenarios:

- Use `GET()` and include additional information in the URL.

  `GET(url)`
- There can be problems with providing information this way (e.g., escaping space characters).
  `URLencode()` can help.

Web Forms (POST)

A web page that contains a **form** requires us to send additional information as part of our request. A form is typically entered by hand, but it is sometimes possible to automate the process; there are two scenarios:

- Use `POST()` and include additional information in the body of the request.

  ```
  POST(url, body=list(...))
  ```

- We need to know the names of the inputs for the form.

Dynamic Web Pages

- If the content of a web page is generated dynamically (e.g., with javascript), automation can be harder (or impossible).
- One ray of hope is the **rdom** package (requires `phantomjs`), which allows us to generate the dynamic content **then** query the content.

Web APIs

- Use HTTP to access a set of (possibly dynamic) resources on a server.
- Not just requesting a static resource; asking the server to perform an action (and return a result).

- In terms of access, a Web API is just a predictable pattern of URLs (so we can just use text processing tools to generate requests).

Web APIs

In order to work with a Web API, we need to ...

- Read the API documentation (including Terms of Use!).
- Find the important methods and understand the inputs for the requests.
- Determine the format (e.g., JSON or XML) and structure of the results.
    - HTML
    - JSON
    - XML

The difference between a Web API and "web scraping" is that a Web API has a public description of a programmatic interface to a web server (rather than forcing us to guess or reverse-engineer an interface).

HTML

Assume you know what it looks like.
Problem is getting useful information out.

- The **rvest** package has a convenience function `html_table()`.
- The result is a **list** of data frames.
- Will not always work, in which case, treat HTML as XML and use **xml2**.

Addenda

There are R packages popping up all over the place that provide convenient interfaces to various data sources.

- The ROpenSci project provies a collection of packages: `https://ropensci.org/packages/`.

- Check the terms and conditions for sites.
- Some sites require registration and an access key.

Resources

- ROpenSci
  `https://ropensci.org/packages/`.
- Gaston Sanchez's "Getting Data from the Web with R"
  http://gastonsanchez.com/work/webdata/