Interacting with Web Data using R

Web APIs and Web Scraping (including RSelenium)

Spencer Lourens, Ph.D.

3/11/2019

Outline

- Why web data?
- Web APIs
  - Example, code run through
- Scraping static web content
  - css selectors, xpath, demos
  - rvest demo and example
- Scraping dynamic web content
  - Selenium web driver, and RSelenium (Selenium ported into R)
  - Example scraping dynamic content, iteration/scaling up

Why web data?

- In many applications, we may not have access to the data we need for an application of interest
  - Environmental concentration changes over time and space and their effect on health outcomes
  - Demographic data for adjustment and comparison across counties/states
  - Historical data regarding health behaviors and outcomes in a specific time/place
- In some settings, there are previous studies that have the data available and ready for download, but this isn't always the case
- We might see the data online in some spot, but perhaps need to gather many data from many slightly different locations

Why web "scraping"?

- Seems to require a LOT of copy and paste, labor (or time) intensive keystrokes/manipulations
- How long is this going to TAKE!?!?
- Can we somehow automate the process (i.e. automate the boring stuff), so that after we build our software, the data extraction/transformation process becomes "plug and play"?
- This is, in my opinion, the pinacle of web scraping, and a really really fun process to engage in
- This is also the topic of this webinar, extracting and transforming web data, either using an API provided by the data curators, or by scraping the data using "web scraping" techniques
  - **rvest** and **RSelenium** are used for this
- This webinar will be as interactive as possible - I'll do a lot of live demos, so cross your fingers!

Prelims

- The scripts used in this presentation and the presentation (and .Rmd which generated it) are available on github:
- https://github.com/slourens/ISDSWebinar

Web APIs

- There are thousands of web APIs available over the internet for accessing data for use in our own applications
- API - application programming interface - so basically an interace that allows YOU to program your own application utilizing external resources (data)
- The R programming language comes equipped with a package called **httr** (developed by Hadley Wickham) which allows executing cURL requests (mostly GET and POST requests, and in our case just GET requests) from within R
- Be careful, you don't NEED to recreate the wheel!
  - Someone else might have already developed an R API for the website you're interested in
  - Google search for "r API"

First Example - healthdata.gov API

- First, let's use the healthdata.gov API to extract data
- Navigate to: https://healthdata.gov/api
- We'll only use the data.json API exposed by healthdata.gov:
  - https://www.healthdata.gov/data.json
- healthdata.gov uses DKAN, whose documentation can be found here:
  - https://docs.getdkan.com/en/latest/apis/
- Many APIs require authentication, using access keys and secrets to prove you are who you say you are whenever you interact with the API - healthdata.gov has an open API, so you're not requireed to use access keys/secrets
- Reading through the documentation and experimenting with downloading / scripting API calls is usually necessary to understand how an API works - this takes time, but saves A LOT of time later
- There is a script available for us to go through that shows how to use this API: healthDataGOVAPI.R:

To the Script!

- To the Script! Demo the API calls, created functions

Scraping Static Content

- Let's do some examples - we'll use **rvest** (sounds like harvest when you pronounce it)
- Allows grabbing the HTML document from a web address and parsing it to extract data
- Works GREAT if the data is a part of the HTML document and not loaded through javascript, PHP, or another server-side request on page load
- We'll demonstrate with the https://airquality.weather.gov website
- Off to rvestExample.R!

XPath and CSS Selectors

- These two topics are both very fun and sometimes incredibly frustrating
- The document object model, i.e. DOM, is the structure of any web page on the internet - however, some websites use iframes, and other nested structures to embed content, so you have to approach each domain on a case by case basis
- What we do in this webinar will not automatically work on another website - BUT it will almost always work for other pages within the domain you've been learning on
- This means that other airquality.weather.gov URLs will be simple for us to scrape now, other calls to the healthdata.gov API should be simple for us to interact with (although the API access function is rudimentary and has not been thoroughly tested - it's just a starting point for further exploration, hopefully)
- At any rate, you can learn more at:
  - (CSS) - https://www.w3schools.com/cssref/css_selectors.asp
  - (XPath) - https://www.w3schools.com/xml/xpath_intro.asp

Scraping Dynamic Content

- In a lot of the examples I'm interested in (scraping job postings for data scientists, extracting financial data, etc.), the content is dynamically generated as a user interacts with a web page
- Consider the zillow website - you go to the main page, enter a city / state, or navigate directly to the city/state website, then interact with a UI to choose whether you want to rent, buy, and the characteristics / criteria you want your future abode to satisfy
- This leads to a bunch of dynamic changes to the content displayed on the website occurring, i.e. it leads to dynamic content generation
- In this case **rvest** is not an option, we need to use a more robust web scraping technology - one option is Selenium Web Driver, which has been ported into R through the RSelenium package, and is our last example we'll go through today!
- If you want to use RSelenium, I'd first go to the following page to get RSelenium installed: