

# WEBSCRAPING IN R AND PYTHON

Jed Rembold

Monday, November 18, 2024

# ANNOUNCEMENTS

- ⬡ Homework 10 due on Thursday night
  - ⬡ 1 problem, 1 extra credit
- ⬡ Projects!
  - ⬡ You have your project groups!
  - ⬡ I'll be leaving time in the latter half of every class this week for you to work with your group

# THE ART OF SCRAPING

# WEBSCRAPING

- ⬡ Webscraping is the act of extracting information from a webpage so that it can be collected or otherwise used elsewhere
- ⬡ It can take multiple forms, with varying degrees of complexity (most of which depends on the website)
  - ⬡ Extracting information from a table of data
  - ⬡ Extracting other information on a webpage that is not necessarily formatted
  - ⬡ Extracting information using a provided API endpoint
- ⬡ Our goal today is to touch on how you could do each of these in either Python or R
  - ⬡ In Python we'll be using the `requests`, `pandas`, `beautifulsoup` and `json` libraries
  - ⬡ In R we'll be using `httr`, `rvest` (which comes as part of tidyverse), and `jsonlite`





# STEP 1: GET THE HTML

- Regardless of what language you are using, the first step is to grab the necessary html
- This is exactly what your browser is doing when it accesses a webpage
- In Python, this is done using the `requests` library `get` function:

```
html = requests.get(url).text
```

- In R, this can be done using the `GET` function from `httr`:

```
html <- content(GET(url))
```

## STEP 2: UNDERSTANDING TAGS

- ⬡ HTML is comprised by information nested within what are called *tags*
  - ⬡ Tags can also be nested inside other tags
- ⬡ Extracting information from a webpage is frequently about knowing which tags have the information you want
- ⬡ Take advantage of the “Inspect” tool on most browsers, accessed by right clicking on a web page
  - ⬡ Will give you the option to explore both the html and mouse around the webpage and highlight the corresponding html tags
- ⬡ At the very least, you should normally try to identify the overall tag that surrounds your data of interest





# OPTION 1: DATA FROM TABLES

- One method in which data is frequently stored on a webpage is in tables
  - These are surrounded by the `< table >` tag
- So long as the table is fairly simple, both Python and R have very easy ways of grabbing the table information directly into a corresponding dataframe

---

Language	Example
----------	---------

Python	<code>df = pandas.read_html(html)</code>
--------	--

R	<code>df &lt;- html %&gt;% html_table</code>
---	--

---

- These will automatically correct for things like cells spanning multiple rows, which is very nice
- By default, both options technically return a list of dataframes for every table on the page

## OPTION 2: OTHER DATA ON A PAGE

- ⬡ Sometimes the data you want on from a page isn't clearly going to be the text in a table
  - ⬡ Maybe it is the url from a link, or an image, or any other text or number not in a table
- ⬡ In these cases you need to rely on the tag structure of the html document to select purely what you are interested
  - ⬡ You may also need to access the tag *attributes* to get information such as link or image urls
- ⬡ When selecting the tags you want, you can provide multiple separated by spaces to provide a hierarchy of what you are looking for
  - ⬡ Looking for `'tr td'` says you can all the `td` tags that are inside a `tr` tag
- ⬡ Gathering the data in this way may generate a list of content, but it won't generally create more complicated tables of information, so you would need to craft those yourself



## OPTION 2: SELECTING THE DESIRED TAGS

- ⬡ In Python, when parsing an html document by tag, it helps greatly to parse the raw html using the BeautifulSoup library
- ⬡ BeautifulSoup then gives you each methods to select only certain tags from the larger structure

```
soup = bs4.BeautifulSoup(html)
links = soup.select('td a')
```

- ⬡ In R, if you are using the `rvest` library, you just need to pass the html into the proper function: `html_elements`

```
links <- html %>% html_elements('td a')
```

## OPTION 2: INFORMATION FROM TAGS

- ⬡ Generally, you then want some information *from* those tags, either the enclosed text or some attribute
- ⬡ To get the text associated with a tag:

---

Language	Example
----------	---------

---

Python	<code>link_text = [tag.text for tag in links]</code>
--------	--

R	<code>link_text &lt;- only_links %&gt;% html_text</code>
---	--

---

- ⬡ To get an attribute value of a tag (content inside the `<` `>` parts of the tag)

---

Language	Example
----------	---------

---

Python	<code>link_url = [tag['href'] for tag in links]</code>
--------	--

R	<code>link_url &lt;- only_links %&gt;% html_attr('href')</code>
---	---

---





## OPTION 3: APIS

- ⬡ Sometimes the amount of information is just so large that it can't fit nicely on a webpage, or data providers don't want to make you "scrape" a webpage for the information
- ⬡ Instead, they might make available a public API where you can access the information
- ⬡ Most REST APIs look just like a web address, but if you navigate to that url, instead of getting HTML to render a webpage, you get the data directly, most often in a JSON format
  - ⬡ Some APIs also let you add extra information to the url to better specify *exactly* what information you want back
- ⬡ Some APIs will require you to register for a key, which is often free. This is to safeguard against people slamming their servers with billions of requests. **Be respectful in both your API and webscraping usage!**





## OPTION 3: ACCESSING API DATA

- ⬡ The information can be downloaded just like html information was downloaded
  - ⬡ `data = requests.get(url).text`
  - ⬡ `data <- read_html(url)`
- ⬡ Commonly, the output will be a string of JSON data, so it helps to use a JSON package to convert it to your language's natural data structures
  - ⬡ In Python

```
output = json.loads(data)
```

- ⬡ In R

```
output <- data %>% html_text %>% fromJSON
```

# PRACTICE TIME!

- ⬡ In your language of choice, see if you can:
  - ⬡ Extract information about our class schedule [here](#) into a dataframe that you could then export to CSV
  - ⬡ Suppose you only wanted to get the names of the tests from that table (the red options). Could you extract only those values?
  - ⬡ The API [here](#) will get you information on all the humans currently in space. Create a table of the names of the astronauts and what vessel they are currently on.

# PROJECTS!



# PROJECT TIME!

- ⬡ The remainder of class I have set aside for you to meet with your group and get going on your project
- ⬡ If you haven't already talked, your initial discussion probably needs to revolve around what data would you be interested in grabbing to analyze in some fashion.
- ⬡ Recall that you are shooting for bringing data from a *separate* source for each member in your group
  - ⬡ Different webpages that need to be scraped
  - ⬡ CSVs from different sources/datasets
  - ⬡ Different API endpoints