Web Scraping with rvest

Clay Ford

Spring 2017

What is web scraping?

- Writing a program or script to automate the process of visiting multiple web pages and collecting data
- Also refers to the process of cleaning up and wrangling the data pulled from a web site so it's suitable for analysis
- This workshop uses R but can also be done with Python and other languages

Things to consider before web scraping

- If your data is on a few pages and will never change, a simple copy-and-paste might be faster
- Check if an API (Application Programming Interface) is available. An API allows you to retrieve data in a structured format. (Example: NY Times, Amazon, Twitter, census.gov)
- See if someone else has already written a script

HTML vs JavaScript

- This presentation shows how to scrape data from a page created with HTML/CSS
- Many sites now generate pages using JavaScript
- Sites generated with JavaScript are harder to scrape and require workarounds in R

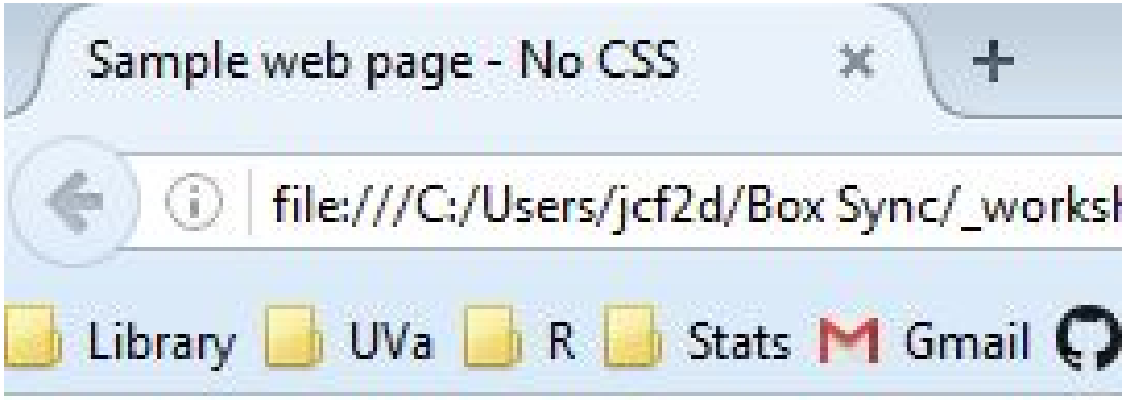
HTML and CSS

- With web scraping, we're downloading a raw HTML page, pulling out what we want, and wrangling into a structure we can analyze
- Many web sites use Cascading Style Sheets (CSS) to create the "look" of a web site
- It helps to know a little about HTML and CSS when web scraping
- To view source code of a web page, do something like right-click and select "View Page Source"

Basic HTML

```
1  <html>
2  <head>
3      <title>Sample web page - No CSS</title>
4  </head>
5  <body>
6  <p>Welcome to the <strong>No CSS</strong> web page.
7  <p>Talking about stuff is fun.
8  <p>I like you. Say more things about stuff.
9  </body>
10 </html>
```

Basic web page



Welcome to the **No CSS** web page.

Talking about stuff is fun.

I like you. Say more things about stuff.

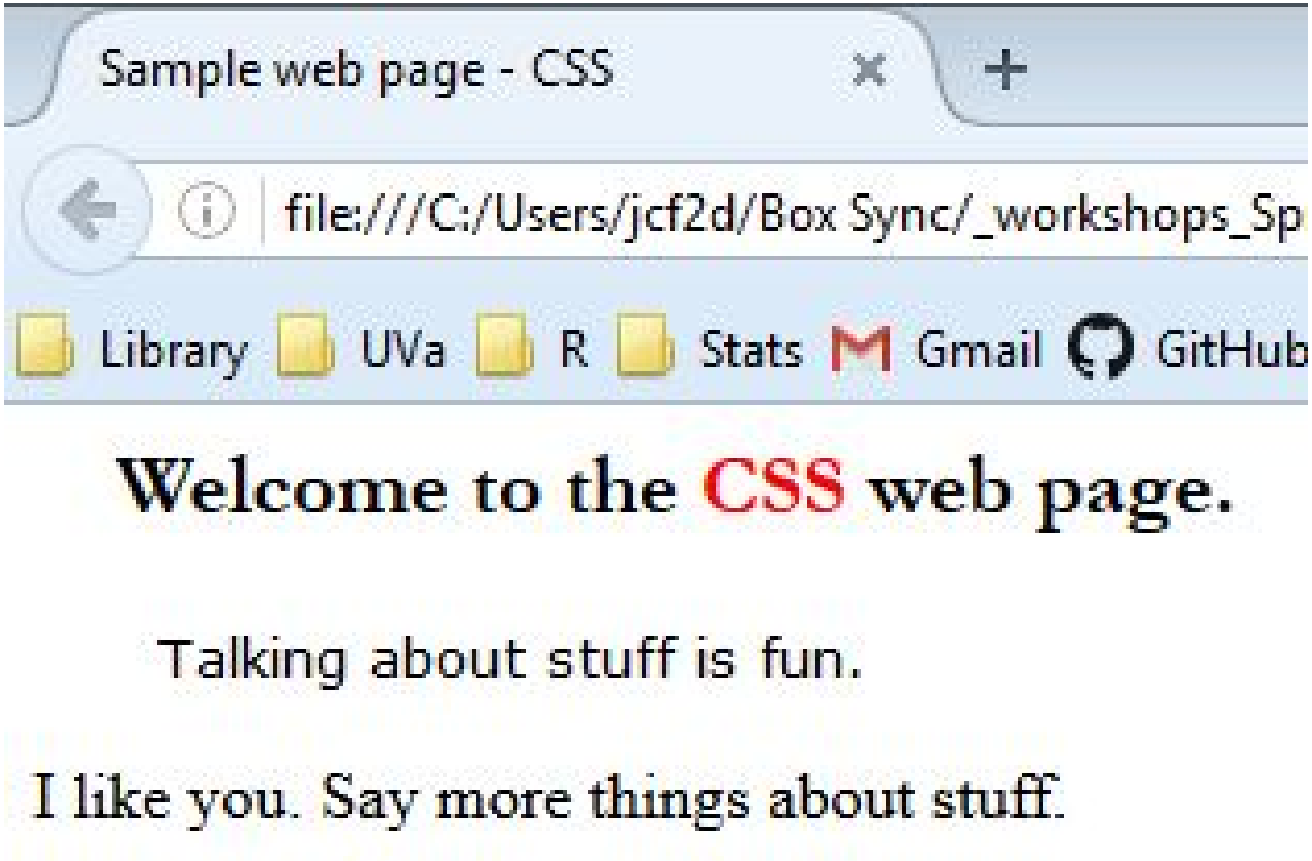
CSS example

```
1   /*a style sheet*/
2   /*id selector*/
3   #title {
4       font-family: Garamond, serif;
5       font-size: 1.3em;
6       font-weight: bold;
7       margin-left: 20px;
8   }
9   /*class selector*/
10  .p1 {
11      font-family: Verdana;
12      font-weight: normal;
13      font-size: 0.8em;
14      margin-left: 30px;
15  }
16  /*element selector*/
17  strong {
18      color: red;
19  }
```

Basic HTML with CSS

```
1  <html>
2  <head>
3      <link type="text/css" rel="stylesheet" href="stylesheet.css"/>
4      <title>Sample web page - CSS</title>
5  </head>
6  <body>
7  <p id="title">Welcome to the <strong>CSS</strong> web page.</p>
8  <p class="p1">Talking about stuff is fun.</p>
9  <p>I like you. Say more things about stuff.
10 </body>
11 </html>
```

Basic web page with CSS



The `rvest` package

- `rvest` is an R package that allows you to "scrape" content off a web page based on CSS selectors and HTML tags.
- `rvest` is a wrapper for `xml2` and `httr`, two other R packages.
- All three packages are in active development, which means new functions may be added, old functions deprecated/removed, and existing functions changed.
- Versions for this workshop
  - `rvest`: `0.3.2`
  - `xml2`: `1.1.1`
  - `httr`: `1.2.1`

CSS selectors

- CSS selectors are patterns used to select the element(s) we want to style
- In our CSS stylesheet above, we selected three elements:
  - `.p1` selected `class="p1"`
  - `#title` selected `id="title"`
  - `strong` selected `<strong>`
- CSS selectors can be more sophisticated. For example:
  - `h1 + p` selects all `<p>` elements placed immediately after `<h1>` elements
  - `a[href$=".pdf"]` selects all `<a>` elements whose link ends with `.pdf`
- **rvest uses CSS selectors to select text to scrape**

rvest Example

Extract the text with `id = "title"`

```
library(rvest)
page <- read_html("sample_CSS.html")
page %>% html_nodes(css = "#title") %>% html_text()
```

```
## [1] "Welcome to the CSS web page."
```

Note: `"sample_CSS.html"` is usually a web site URL.

What just happened?

- loaded the `rvest` package (which also loads the `xml2` package)
- used the `read_html` function to read the HTML page (Note: `read_html` is an `xml2` function)
- used the `html_nodes` function to find those parts with `id = "title"`
- used the `html_text` function to extract just the text
- Notice the last three were "piped" together using the %>% operator. This takes the output of the previous function and inputs as the first argument to the next function.
- Use Ctrl+Shift+M (Win) or Cmd+Shift+M (Mac) to quickly insert %>%

Another `rvest` example

Extract the text with `class = "p1"`

```
page %>% html_nodes(css = ".p1") %>% html_text()
```

```
## [1] "Talking about stuff is fun."
```

Another `rvest` example

Extract the text tagged with `<strong>`

```
page %>% html_nodes(css = "strong") %>% html_text()
```

```
## [1] "CSS"
```

Another `rvest` example

Extract the text tagged with <p>

```
page %>% html_nodes(css = "p") %>% html_text()
```

```
## [1] "Welcome to the CSS web page."
## [2] "Talking about stuff is fun."
## [3] "I like you. Say more things about stuff.\r\n"
```

A common `rvest` process

1. Read in a web page with `read_html`
2. Pick out some portion of the page using `html_nodes` with a CSS selector
3. Pass to an extractor function such as `html_text` or `html_table`

`html_text` extracts text between HTML tags. `html_table` parses an HTML table into a data frame.

More CSS selector examples

- Select all HTML tables: `html_nodes(css = "table")`
- Select all `<strong>` elements inside of any `<p>`: `html_nodes("p strong")`
- Select all elements that have `<ul class="fancy">`: `html_nodes("ul.fancy")`
- Select all elements with `id="big"` that also have `class="wide"`: `html_nodes("#big.wide")`

To learn more, work through tutorial at
`http://flukeout.github.io/`;
See also this CSS selector reference:
`http://www.w3schools.com/cssref/css_selectors.asp`

How to figure out which CSS selectors to use

Three common approaches:

1. View page source (right click... View Page Source) and figure it out.
2. Use your browser's "Inspector" function. Chrome and Firefox have this.
3. Use SelectorGadget. SelectorGadget is a javascript bookmarklet that you add to your browser toolbar.

To install SelectorGadget

1. go to `https://cran.r-project.org/web/packages/rvest/vignettes/selectorgadget.html`
2. Drag the "SelectorGadget" link (under Installation) into your bookmark toolbar

That's it!

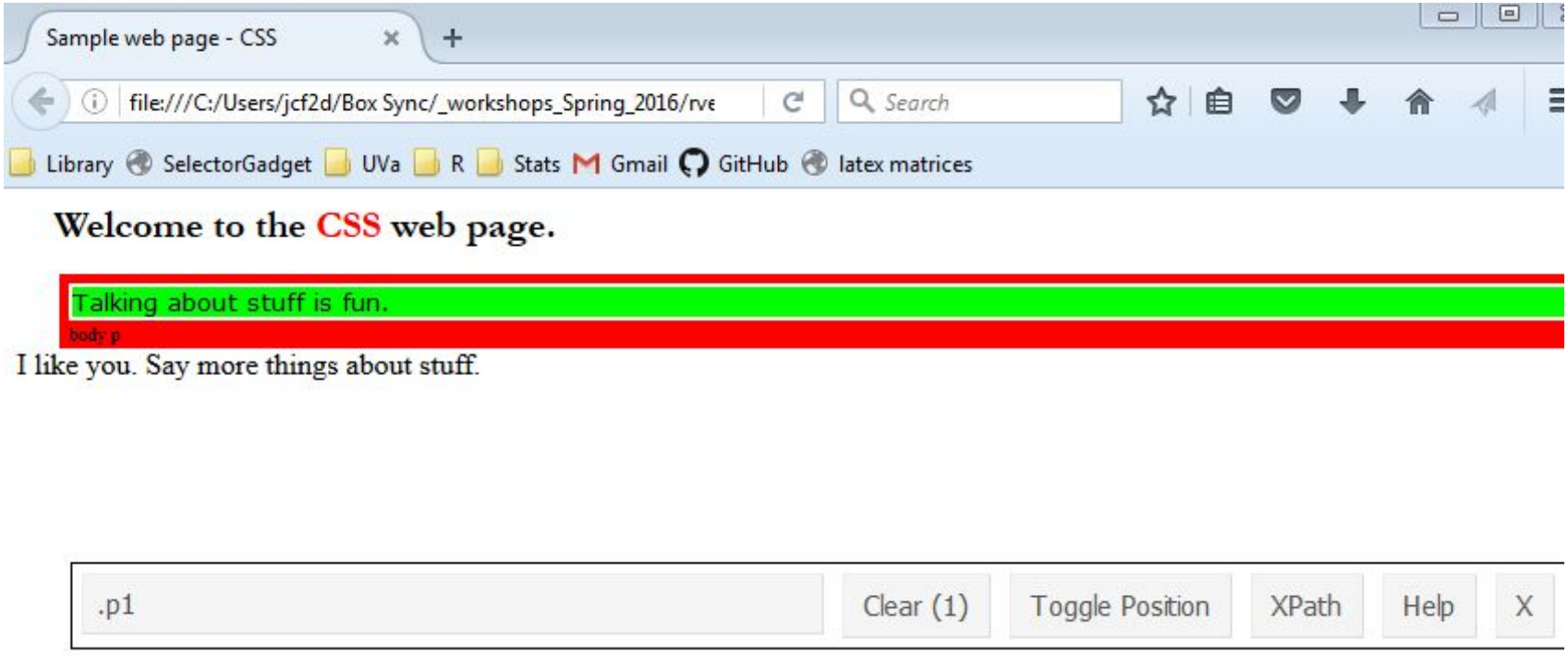
Basic way to use SelectorGadget

1. Click the SelectorGadget button in your bookmark bar to turn it on.
2. Hover over element you wish to select and click. It will turn green. SelectorGadget will reveal the CSS selector. All other matching items will be highlighted yellow.
3. If necessary, scroll around the document to find yellow elements that you don't want to match and click on them.

The `rvest` package has a vignette for using SelectorGadget with more details.

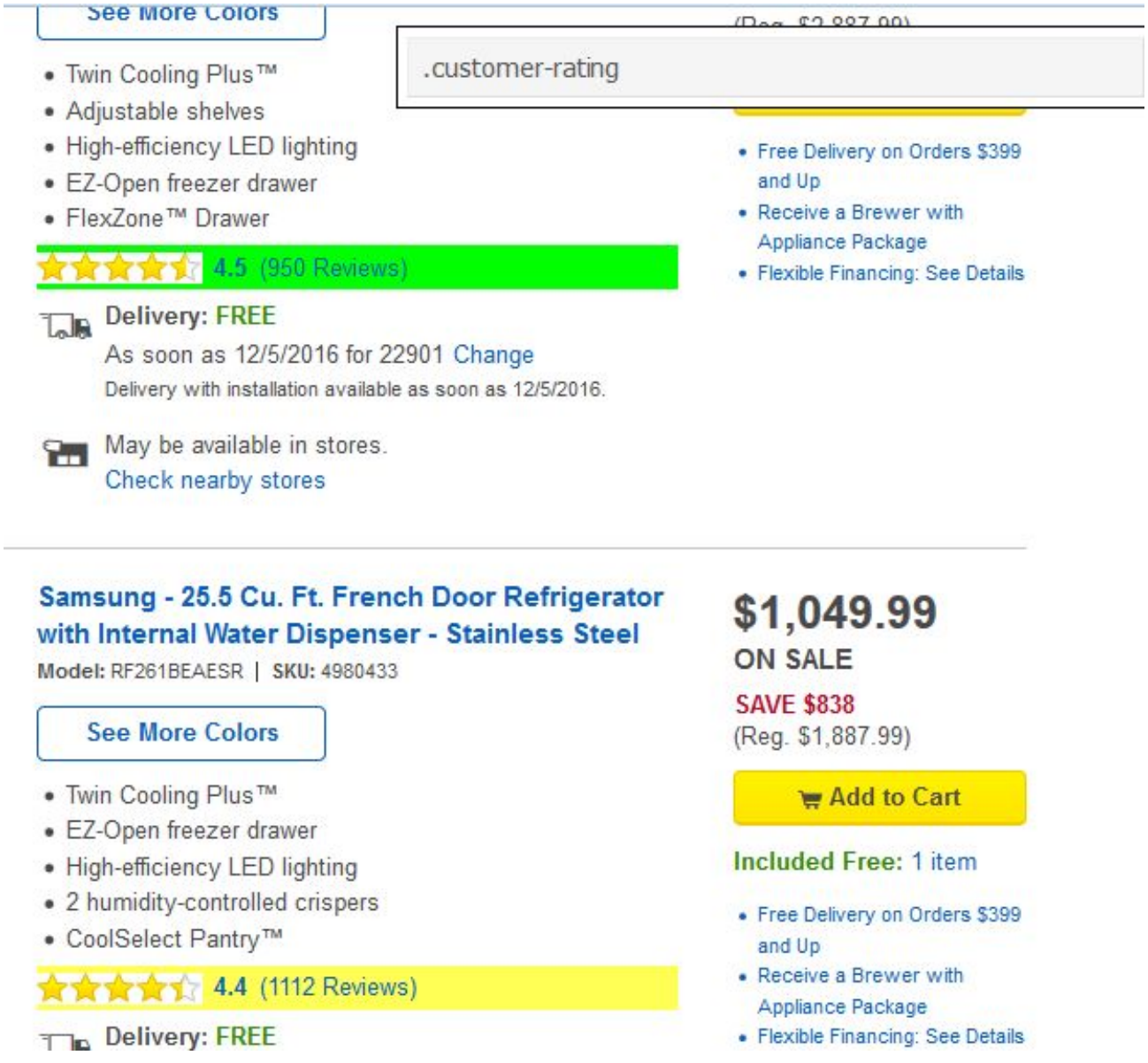
SelectorGadget example

Click the SelectorGadget button, click what you want to select, look at result in SelectorGadget bar (".p1")



SelectorGadget example

Determining selector for all reviews on BestBuy



Scraping tables

- HTML tables are usually in `<table>` tags
- `html_nodes("table")` will find all tables on a page
- If we want a specific table, say the 3rd one, we can use the `extract` function from the `magrittr` package to select it.
- Example: `html_nodes("table") %>% extract(3)`
- Once we have a table selected, we use `html_table` to convert the table to a data frame

Example of a table

Costliest earthquakes [edit]

This is a list or major earthquakes by the dollar value of property (public and private) losses directly attributable to the earthquake. Wherever possible, indirect and socioeconomic losses are excluded. Please note that damage estimates for particular earthquakes may vary through time as more data becomes available.

Rank	Property damages (inflation-adjusted) [*when?*][23]	Event	Location	Magnitude	Property damages (non-inflation-adjusted)
1	$312 billion	1995 Great Hanshin earthquake	Japan	6.9	$200 billion[24]
2	$249 billion	2011 Tōhoku earthquake and tsunami	Japan	9.1[4]	$235 billion[25][26]
3	$95 billion	2008 Sichuan earthquake	Sichuan, China	8.0	$86 billion[27]

Example of scraping a table

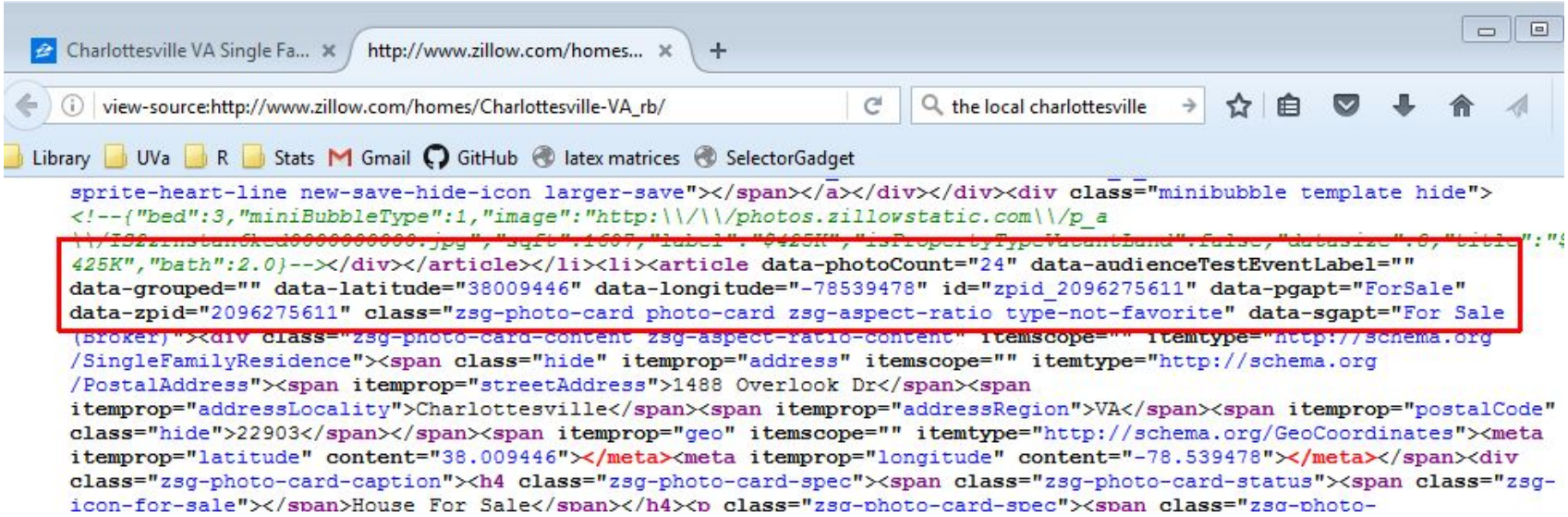
```
library(magrittr) # for extract function
URL <- "https://en.wikipedia.org/wiki/Lists_of_earthquakes"
page <- read_html(URL)
dat <- page %>% html_nodes("table") %>%
  extract(3) %>% html_table()
```

Or maybe like this without using `extract`:

```
tables <- read_html(URL) %>% html_nodes("table")
dat <- tables[[3]] %>% html_table()
```

Extracting data from attributes

- Sometimes we want to scrape data stored in attributes.
- For example, on zillow.com, items like latitude, longitude and zillow id are stored as attribute values:



- We can use `html_attr` for this task.
- Use it like `html_text` (ie, after using `html_nodes`)

Scraping attribute values example

```
URL <- "http://www.zillow.com/homes/Charlottesville-VA_rb/"
page <- read_html(URL)
page %>% html_nodes("article") %>%
  html_attr("data-latitude")
```

```
##  [1] "38162223" "38060730" "38034126" "38090600" "380366
##  [7] "38049249" "38027108" "38029676" "38034483" "380337
## [13] "38032700" "38038456" "38072200" "38028690" "380722
## [19] "38063160" "38045340" "37998500" "38125687" "381592
## [25] "38019649" "38122987" "38030950"
```

Notice we need to convert to numeric and divide by 1,000,000 to get actual latitude values.

Web scraping multiple pages

- Often the data we want is located on multiple web pages.
- Example: a web site may only show 10 results at a time for a site search.
- Therefore we need to repeat our `rvest` code for each page.
- This requires some `R` programming to either loop or apply our `rvest` code for multiple pages.

A general strategy for scraping multiple pages

1. write R code to scrape data for one page
2. convert R code to a function
3. Get URLs for all pages you want to scrape
4. "lapply" function to a vector of URLs (or use a `for` loop to cycle through URLs)

We will demonstrate this approach in the R script.

Another approach is to use the `follow_link()` function to programmatically follow, say, a "Next" link to the next page.

Submitting forms

`rvest` also allows you to submit web page forms. For example, we might like to write an R script that allows us to submit the following form and scrape the results:

One process for submitting forms

1. Establish a session using `html_session`
2. Parse the form using `html_form`
3. Set the form values using `set_values`
4. Submit the form using `submit_form`

Example of submitting Virgo form and scraping book titles

```
uvaLibrary <- "http://search.lib.virginia.edu/catalog"
page <- html_session(uvaLibrary)
f0 <- html_form(page)[[1]]
f1 <- set_values(form = f0, q = "Asimov",
                 catalog_select = "catalog")
results <- page %>% submit_form(form = f1)
results %>% html_nodes("dd.titleField a") %>% html_text()
```

Notes: This scrapes just the book titles on page 1. `q` and `catalog_select` are form fields on Virgo.

web scraping heads-up

- Be paranoid. Assume something will go wrong. What works on one page may not work on another page.
- Be patient. Expect to use a lot of trial and error to figure out the best way to scrape a page.
- Continue to be patient. Expect your script to take time to run if you're scraping lots of pages.
- Be prepared for the web site to change without notice and render your script useless.
- Be prepared to spend time cleaning your data once the scraping is done.

Let's go to R!

Reference and further reading

- Tutorial for css selectors: `http://flukeout.github.io/`
- CSS Selector cheat sheet:
  `http://www.w3schools.com/cssref/css_selectors.asp`
- See the demos that come with the rvest package

Thanks for coming today!

- For help and advice with your data analysis: `statlab@virginia.edu`
- Sign up for more workshops or see past workshops: `http://data.library.virginia.edu/training/`
- Register for the Research Data Services newsletter to be notified of new workshops: `http://data.library.virginia.edu/newsletters/`