Web Scraping

Statistical Programming

Fall 2021

Dr. Colin Rundel


rvest
www.rstudio.com

Hypertext Markup Language

Most of the data on the web is still largely available as HTML - while it is structured (hierarchical) it often is not available in a form useful for analysis (flat / tidy).

```
<html>
  <head>
    <title>This is a title</title>
  </head>
  <body>
    <p align="center">Hello world!</p>
    <br/>
    <div class="name" id="first">John</div>
    <div class="name" id="last">Doe</div>
    <div class="contact">
      <div class="home">555-555-1234</div>
      <div class="home">555-555-2345</div>
      <div class="work">555-555-9999</div>
      <div class="fax">555-555-8888</div>
    </div>
  </body>
</html>
```

rvest

`rvest` is a package from the tidyverse that makes basic processing and manipulation of HTML data straight forward. It provides high level functions for interacting with html via the xml2 library.

Core functions:

- `read_html()` - read HTML data from a url or character string.
- `html_elements()` / `html_elements()` - select specified elements from the HTML document using CSS selectors.
- `html_table()` - parse an HTML table into a data frame.
- `html_text()` / `html_text2()` - extract tag's text content.
- `html_name` - extract tag's names.
- `html_attrs` - extract all of each tag's attributes.

html, rvest, & xml2

```
html =
'<html>
  <head>
    <title>This is a title</title>
  </head>
  <body>
    <p align="center">Hello world!</p>
    <br/>
    <div class="name" id="first">John</div>
    <div class="name" id="last">Doe</div>
    <div class="contact">
      <div class="home">555-555-1234</div>
      <div class="home">555-555-2345</div>
      <div class="work">555-555-9999</div>
      <div class="fax">555-555-8888</div>
    </div>
  </body>
</html>'


read_html(html)
```

```
## {html_document}
## <html>
```

css selectors

We will be using a tool called selector gadget to help up identify the html elements of interest - it does this by constructing a css selector which can be used to subset the html document.

Selector	Example	Description
element	`p`	Select all <p> elements
element element	`div p`	Select all <p> elements inside a element
element>element	`div > p`	Select all <p> elements with as a parent
.class	`.title`	Select all elements with class="title"
#id	`#name`	Select all elements with id="name"
[attribute]	`[class]`	Select all elements with a class attribute
[attribute=value]	`[class=title]`	Select all elements with class="title"

There are also a number of additional combinators and pseudo-classes that improve flexibility, see examples here

Selecting tags

```
read_html(html) %>% html_elements("p")
```

```
## {xml_nodeset (1)}
## [1] <p align="center">Hello world!</p>
```

```
read_html(html) %>% html_elements("p") %>% html_text()
```

```
## [1] "Hello world!"
```

```
read_html(html) %>% html_elements("p") %>% html_attrs()
```

```
## [[1]]
##    align
## "center"
```

```
read_html(html) %>% html_elements("p") %>% html_attr("align")
```

```
## [1] "center"
```

More selecting tags

```
read_html(html) %>% html_elements("div")
```

```
## {xml_nodeset (7)}
## [1] <div class="name" id="first">John</div>
## [2] <div class="name" id="last">Doe</div>
## [3] <div class="contact">\n      <div class="home">555-555-1234</div>\n       ...
## [4] <div class="home">555-555-1234</div>
## [5] <div class="home">555-555-2345</div>
## [6] <div class="work">555-555-9999</div>
## [7] <div class="fax">555-555-8888</div>
```

```
read_html(html) %>% html_elements("div") %>% html_text()
```

```
## [1] "John"
## [2] "Doe"
## [3] "\n      555-555-1234\n      555-555-2345\n      555-555-9999\n      555-555-8888\n    "
## [4] "555-555-1234"
## [5] "555-555-2345"
## [6] "555-555-9999"
## [7] "555-555-8888"
```

Nesting tags

```
read_html(html) %>% html_elements("body div")
```

```
## {xml_nodeset (7)}
## [1] <div class="name" id="first">John</div>
## [2] <div class="name" id="last">Doe</div>
## [3] <div class="contact">\n      <div class="home">555-555-1234</div>\n       ...
## [4] <div class="home">555-555-1234</div>
## [5] <div class="home">555-555-2345</div>
## [6] <div class="work">555-555-9999</div>
## [7] <div class="fax">555-555-8888</div>
```

```
read_html(html) %>% html_elements("body>div")
```

```
## {xml_nodeset (3)}
## [1] <div class="name" id="first">John</div>
## [2] <div class="name" id="last">Doe</div>
## [3] <div class="contact">\n      <div class="home">555-555-1234</div>\n       ...
```

```
read_html(html) %>% html_elements("body div div")
```

```
## {xml_nodeset (4)}
## [1] <div class="home">555-555-1234</div>
## [2] <div class="home">555-555-2345</div>
## [3] <div class="work">555-555-9999</div>
```

CSS classes and ids

```
read_html(html) %>% html_elements(".name")
```

```
## {xml_nodeset (2)}
## [1] <div class="name" id="first">John</div>
## [2] <div class="name" id="last">Doe</div>
```

```
read_html(html) %>% html_elements("div.name")
```

```
## {xml_nodeset (2)}
## [1] <div class="name" id="first">John</div>
## [2] <div class="name" id="last">Doe</div>
```

```
read_html(html) %>% html_elements("#first")
```

```
## {xml_nodeset (1)}
## [1] <div class="name" id="first">John</div>
```

Mixing it up

```
read_html(html) %>% html_elements("[align]")
```

```
## {xml_nodeset (1)}
## [1] <p align="center">Hello world!</p>
```

```
read_html(html) %>% html_elements(".contact div")
```

```
## {xml_nodeset (4)}
## [1] <div class="home">555-555-1234</div>
## [2] <div class="home">555-555-2345</div>
## [3] <div class="work">555-555-9999</div>
## [4] <div class="fax">555-555-8888</div>
```

html_text() vs html_text2()

```
html = read_html(
  "<p>
    This is the first sentence in the paragraph.
    This is the second sentence that should be on the same line as the first sentence.<br>This third sent
  </p>"
)
```

```
html %>% html_text() %>% cat(sep="\n")
```

```
##
##     This is the first sentence in the paragraph.
##     This is the second sentence that should be on the same line as the first sentence.This third sentence sh
##
```

```
html %>% html_text2() %>% cat(sep="\n")
```

```
## This is the first sentence in the paragraph. This is the second sentence that should be on the same line as
## This third sentence should start on a new line.
```

html tables

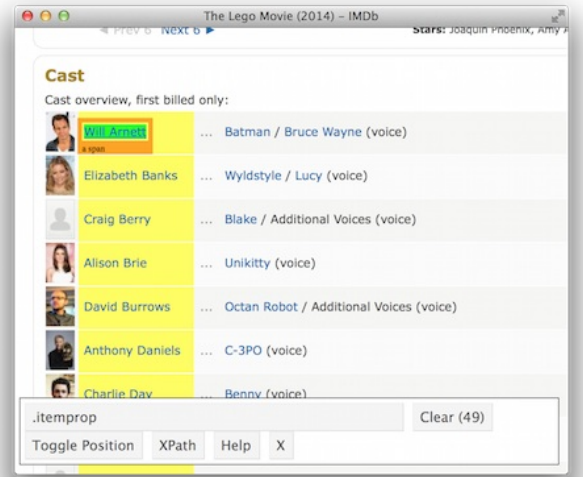
```
html_table =
'<html>
  <head>
    <title>This is a title</title>
  </head>
  <body>
    <table>
      <tr> <th>a</th> <th>b</th> <th>c</th> </tr>
      <tr> <td>1</td> <td>2</td> <td>3</td> </tr>
      <tr> <td>2</td> <td>3</td> <td>4</td> </tr>
      <tr> <td>3</td> <td>4</td> <td>5</td> </tr>
    </table>
  </body>
</html>'
```

```
read_html(html_table) %>% html_elements("table") %>% html_table()
```

```
## [[1]]
## # A tibble: 3 × 3
##       a     b     c
##   <int> <int> <int>
## 1     1     2     3
## 2     2     3     4
```

SelectorGadget

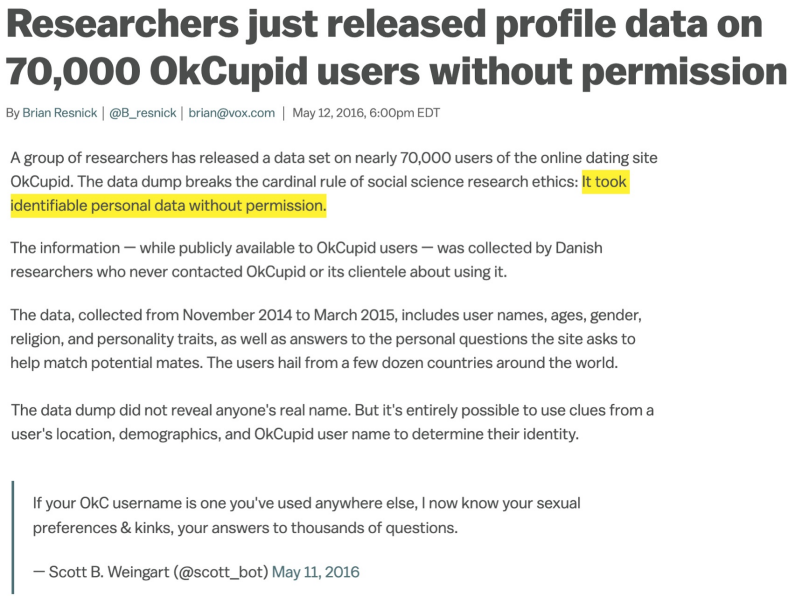
This is a javascript based tool that helps you interactively build an appropriate CSS selector for the content you are interested in.



selectorgadget.com

Web scraping considerations

"Can you?" vs "Should you?"



Researchers just released profile data on 70,000 OkCupid users without permission

By Brian Resnick | @B_resnick | brian@vox.com | May 12, 2016, 6:00pm EDT

A group of researchers has released a data set on nearly 70,000 users of the online dating site OkCupid. The data dump breaks the cardinal rule of social science research ethics: It took identifiable personal data without permission.

The information — while publicly available to OkCupid users — was collected by Danish researchers who never contacted OkCupid or its clientele about using it.

The data, collected from November 2014 to March 2015, includes user names, ages, gender, religion, and personality traits, as well as answers to the personal questions the site asks to help match potential mates. The users hail from a few dozen countries around the world.

The data dump did not reveal anyone's real name. But it's entirely possible to use clues from a user's location, demographics, and OkCupid user name to determine their identity.

> If your OkC username is one you've used anywhere else, I now know your sexual preferences & kinks, your answers to thousands of questions.
>
> — Scott B. Weingart (@scott_bot) May 11, 2016

Source: Brian Resnick, Researchers just released profile data on 70,000 OkCupid users without permission, Vox.

"Can you?" vs "Should you?"

Scraping permission & `robots.txt`

There is a standard for communicating to users if it is acceptable to automatically scrape a website via the [robots exclusion standard] or `robots.txt`.

You can find examples at all of your favorite websites: google, facebook, etc.

These files are meant to be machine readable, but the `polite` package can handle this for us.

```
polite::bow("http://google.com")
```

```
## <polite session> http://google.com
##     User-agent: polite R package - https://github.com/dmi3kno/polite
##     robots.txt: 281 rules are defined for 4 bots
##    Crawl delay: 5 sec
##   The path is scrapable for this user-agent
```

```
polite::bow("http://facebook.com")
```

```
## <polite session> http://facebook.com
##     User-agent: polite R package - https://github.com/dmi3kno/polite
##     robots.txt: 479 rules are defined for 20 bots
##    Crawl delay: 5 sec
##   The path is not scrapable for this user-agent
```

Example - Rotten Tomatoes

For the movies listed in **Popular Steaming Movies** list on `rottentomatoes.com` create a data frame with the Movies' titles, their tomatometer score, and whether the movie is fresh or rotten, and the movie's url.

Exercise

Using the url for each movie, now go out and grab the number of reviews, the runtime, and number of user ratings.

If you finish that you can then try to scrape the mpaa rating and the audience score,.