Video 17: Webscraping with rvest

Stats 102A

Miles Chen, PhD

Package rvest

```
# install.packages("rvest")
library(rvest)
```

“harvesting” from the web, aka. web scraping

Documentation: https://rvest.tidyverse.org/

Recommended Reading:
https://rvest.tidyverse.org/articles/rvest.html

Use Selector Gadget with rvest:

https://rvest.tidyverse.org/articles/selectorgadget.html

rvest

rvest is great for fairly static webpages with well defined html tags.

rvest does not work as well for sites that are generated via javascript, e.g. sites with infinite scroll.

rvest does not work well for sites where you must be logged in to view content.

HTML and CSS

A very brief understanding of HTML can help you understand how rvest works.

Webpages are written in a markup language called HTML. Formatting is applied by using tags. To keep a page's format consistent, the designer will often use CSS classes and in a separate file specify how those classes should be displayed.

HTML and CSS

Here's a simplified version of what those tags may look like in the underlying HTML.

```
<ul class="flat-list buttons">
    <li class="first">
        <a href="#" class="comments">4612 comments</a>
    </li>
    <li class="share-button">
        <a href="#">share</a>
    </li>
    <li class="save-button">
        <a href="#">save</a>
    </li>
    <li class="hide-button">
        <a href="#">hide</a>
    </li>
    <li class="report-button">
        <a href="#">report</a>
    </li>
</ul>
```

Selector Gadget

The selector gadget works by looking at the CSS tags and selecting all tags on the webpage that has a specific tag.

I go to the site and click text that I want to capture. The selector gadget highlights everything it will capture in yellow. If there is stuff I do not want, I click that and selector gadget will adjust the tag to unselect.

So if I use the tag `.comments`, the selector gadget will identify all text that is surrounded by a tag with the class `.comments`. [Don't worry about the fact that there's a dot, the selector gadget will determine if there should or should not be a dot included in the tag to use.]

rvest

With the tag, we can go back to rvest and select the nodes.

```
old_reddit <- read_html("https://old.reddit.com/")
comment_counts <- old_reddit %>%
  html_nodes(".comments") %>%
  html_text()
comment_counts
```

```
 [1] "10233 comments" "5096 comments"  "4554 comments"  "560 comments"
 [5] "523 comments"   "3618 comments"  "1510 comments"  "2911 comments"
 [9] "775 comments"   "1523 comments"  "2720 comments"  "249 comments"
[13] "1511 comments"  "2707 comments"  "1138 comments"  "2535 comments"
[17] "2545 comments"  "481 comments"   "747 comments"   "2216 comments"
[21] "2207 comments"  "1667 comments"  "119 comments"   "1554 comments"
[25] "258 comments"
```

Using your scraped text

We can now extract these values with regular expressions (covered later).

```
1 library(stringr)
2 as.numeric(str_extract(comment_counts, "\\d+"))
```

```
 [1] 10233  5096  4554   560   523  3618  1510  2911   775  1523  2720   249
[13]  1511  2707  1138  2535  2545   481   747  2216  2207  1667   119  1554
[25]   258
```

Hard for rvest

Sites featuring infinite scroll don’t work well with rvest. For example, reddit.

Sites that require logging in to view content also do not work well. For example, Facebook, X/twitter.

Easier for rvest

rvest works best with fairly static, public pages.

The css selector

```
old_reddit <- read_html("https://old.reddit.com/")
old_reddit %>%
  html_nodes(".title.may-blank") %>%
  html_text()
```

```
 [1] "Who do you WISH would run for the President of America?"
 [2] "AITA for slapping a teenager?"
 [3] "Just a little Pissed!"
 [4] "Epstein Maxxing "
 [5] "This is a scorcher"
 [6] "Has anyone else completely lost faith in the American political system?"
 [7] "The last moments of a man who jumped into a tiger enclosure in 2014"
 [8] "What’s the fastest a movie has gone from “good” to “bad”?"
 [9] "The way this mows grass soothes my soul"
[10] "Sha Carri anchors USA s 4x100 WORLD TITLE"
[11] "JK Rowling says David Tennant is part of ‘gender Taliban’ after trans
rights support"
[12] "Body Reset Day"
[13] "Wonder What Happened to This Guy"
```

Other Webscraping Options

To work with javascript generated pages, you can use `RSelenium`, which is powerful but has a steeper learning curve.

Other language options:

- Python Beautiful Soup - very easy to learn
- Python Scrapy - more complex, can crawl across many pages
- Python Selenium - supports javascript generated pages

Automated browsing and scraping

rvest allows for automated sessions that can visit multiple pages and scrape the content from each page.

The general structure is simple:

1. Have rvest begin a browsing session.
2. Tell the session to visit a particular URL.
3. Tell rvest to scrape the content of the page and store it somewhere.
4. Tell the session to visit a new URL.
5. Repeat steps 3 and 4 until you have the content you desire.

The polite package

Many websites implement some sort of protection against DDOS attacks and bots that spam requests.

If you use rvest, you need to be careful not to generate too many requests. (A request is when your browser session requests some content from the server - usually a webpage.) If you generate too many requests, the server might rate-limit your IP address (temporary restriction on how many requests you are allowed to make) or possibly even ban/block you.

The package `polite` addresses some of these concerns by intentionally slowing down to avoid problems.

Starting a polite session

We begin by loading `library(polite)`.

We create a polite session with `session_name <- bow(url_goes_here)`

We can then scrape the HTML content using `scrape(session_name, content = "text/html; charset=UTF-8")`

If we want to visit a new page, we use `nod(session_name, path = new_url)`

The `polite` package will automatically make sure `nod()` does not request new urls too frequently or quickly.

rvest demo