Web Scraping with rvest

Shawn Santo

STT 301 Michigan State University

November 23, 2018

What is Web Scraping?

- Conversion of data in an unstructered format on the web to a structured format in R

- Most programming languages have web scraping capabilities

Ways to Scrape Data

- Copy-and-paste
- Text pattern matching
- Application Programming Interface (API)
- Document Object Model (DOM) parsing

Ways to Scrape Data into R

- rvest package
- xml2 package
- Rcrawler package
- Custom scrapers
  - twitteR package
  - Rlinkedin package
  - nflscrapR package

rvest package

- Written by Hadley Wickham
- Wrappers around the 'xml2' and 'httr' packages to make it easy to download, then manipulate, HTML and XML
  - `read_html`
  - `html_nodes`
  - `html_table`
  - `html_text`
- GitHub URL: `https://github.com/hadley/rvest`
- Vignette: `https://cran.r-project.org/web/packages/rvest/rvest.pdf`

Using rvest

- A starting guide for scraping with rvest

    1. Save the website's URL as a string
    2. Use `read_html` with the saved URL as an input
    3. Extract pieces out of the HTML document with `html_nodes` by specifying css selectors
    4. Use `html_text` or `html_table`

A Working Example

MI Municipality Population

We will scrape the 200 most populous municipalities in Michigan based on 2010 United States Census from `https://en.wikipedia.org/wiki/List_of_municipalities_in_Michigan_(by_population)` into R.

- Steps 1 - 2

```
mi.url <- "https://en.wikipedia.org/wiki/List_of_municipalities_in_Mic
mi.html <- read_html(mi.url)
```

- Steps 3 - 4

```
mi.table <- mi.html %>%
  html_nodes(css = "table") %>%
  html_table(fill = TRUE)
```

A Working Example

MI Municipality Population

There are a few tables pulled from the Wikipedia page. We will obtain the first table in the list and provide some variable names.

```r
mi.pop <- mi.table[[1]]

names(mi.pop) <- c("rank", "municipality",
                   "county", "population")
```

A Working Example

MI Municipality Population

Let's see the result.

```
head(mi.pop)

  rank     municipality                      county population
1    1          Detroit                       Wayne    713,777
2    2     Grand Rapids                        Kent    188,040
3    3           Warren                      Macomb    134,056
4    4 Sterling Heights                      Macomb    129,699
5    5          Lansing Clinton, Eaton & Ingham    114,297
6    6        Ann Arbor                   Washtenaw    113,934
```

We would want to clean this further and remove the commas from the `population` variable. This can be done with the `gsub` function.

A Closer Look at html_nodes

- `html_nodes(x, css, xpath)`
  - `x` - Either a document, a node set or a single node
  - `css`, `xpath` - Nodes to select
- What if I do not know what CSS is or what CSS selector to input?

What is CSS?

- CSS stands for Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen
- CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes
- CSS selectors are used to “find” (or select) HTML elements based on their element name, id, class, attribute, and more
- A good tutorial on CSS selectors is available at: `http://flukeout.github.io/`

CSS and SelectorGadget

- SelectorGadget is an open source tool that makes CSS selector generation and discovery on complicated sites easy
    - `http://selectorgadget.com/`
- The information you want does not have to be in a "nice" table
- SelectorGadget will predict the CSS selector you want/need

A Working Example

IMDb Top 100

We will scrape the top 100 movies of 2016 (sorted by populatiry) from `http://www.imdb.com/search/title?count=100&release_date=2016,2016&title_type=feature` into R using SelectorGadget.

- Access HTML

```r
imdb.url <- "http://www.imdb.com/search/title?count=100&release_date=2
imdb.html <- read_html(imdb.url)
```

- Scrape the rankings (use SelectorGadget to obtain the CSS selector)

```r
rankings <- imdb.html %>%
  html_nodes(css = ".text-primary") %>%
  html_text()

rankings[1:5]
```

```
[1] "1." "2." "3." "4." "5."
```

A Working Example

IMDb Top 100

- Scrape the titles (use SelectorGadget to obtain the CSS selector)

```
titles <- imdb.html %>%
  html_nodes(css = ".lister-item-header a") %>%
  html_text()

titles[1:5]

[1] "Fantastic Beasts and Where to Find Them"
[2] "Suicide Squad"
[3] "Split"
[4] "Sing"
[5] "Deadpool"
```

A Working Example
IMDb Top 100

- Scrape the ratings (use SelectorGadget to obtain the CSS selector)

```
ratings <- imdb.html %>%
  html_nodes(css = ".ratings-imdb-rating strong") %>%
  html_text()

ratings[1:5]
```

```
[1] "7.3" "6.1" "7.3" "7.1" "8.0"
```

A Working Example

IMDb Top 100

- Let's see the result.

```
imdb <- data.frame(rank = as.numeric(rankings),
                   title = titles,
                   rating = as.numeric(ratings))
head(imdb)

  rank                                 title rating
1    1 Fantastic Beasts and Where to Find Them    7.3
2    2                          Suicide Squad    6.1
3    3                                  Split    7.3
4    4                                   Sing    7.1
5    5                               Deadpool    8.0
6    6                  The Magnificent Seven    6.9
```

More on rvest

- Demos in rvest: `demo(package = "rvest")`
    - `demo("tripadvisor", package = "rvest")`
    - `demo("zillow", package = "rvest")`
- A detailed tutorial: `https://stat4701.github.io/edav/2015/04/02/rvest_tutorial/`
- RStudio blog: `https://blog.rstudio.com/2014/11/24/rvest-easy-web-scraping-with-r/`