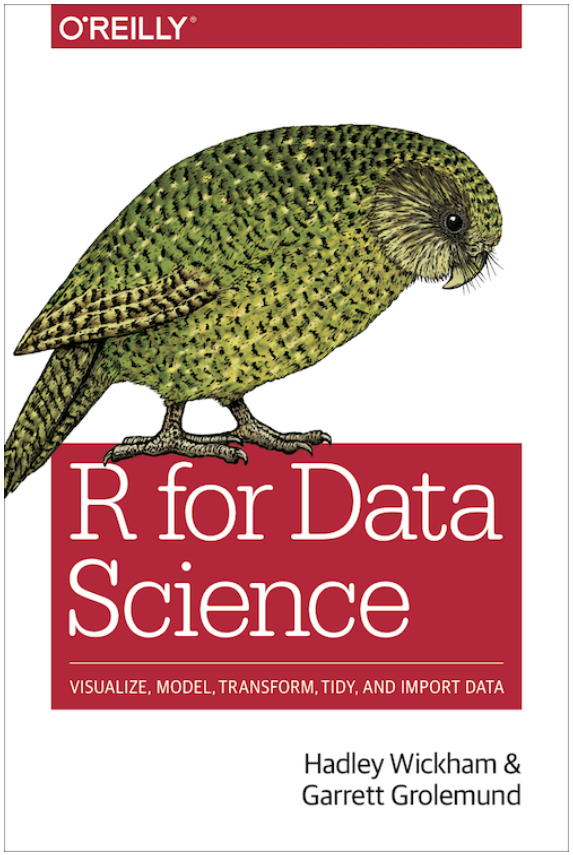
Socio-Informatics 348

Import
Web Scraping

Dr Lisa Martin

Department of Information Science
Stellenbosch University

Today's Reading



R for Data Science, Chapter 24

What is Web Scraping?

Web scraping is a technique for extracting data from web pages. While many sites provide APIs (often returning JSON), scraping works even when an API is not available.
In R, we typically use the `rvest` package. (Note: rvest is not in the core tidyverse, so you need to load it explicitly.)

```
library(tidyverse)
library(rvest)
```

Ethics & Legal Considerations

Key concerns:

- Terms of Service — some sites explicitly prohibit scraping
- Personally Identifiable Information — even if public, collecting names, emails, etc. has ethical risks
- Copyright — factual data is generally not copyrightable, but original content may be protected

Be respectful of server load (rate-limit your requests). The `polite` package can help with pausing between requests.

HTML Basics: Elements and Attributes

HTML (HyperText Markup Language) is a hierarchical markup language.

- Elements: e.g. `<p>`, `<h1>`, `<img>` etc.
- Attributes: named values in tags, e.g. `id='first'`, `class='title'`, `src='...'`
- Some attributes are especially helpful in scraping, e.g. `id`, `class`, `href`, `src`

Example:

```
<html>
<head>
  <title>Page title</title>
</head>
<body>
  <h1 id='first'>A heading</h1>
  <p>Some text &amp; <b>some bold text.</b></p>
  <img src='myimg.png' width='100' height='100'>
</body>
```

Extracting Data with rvest

1. **Get the HTML:**
   - Use `read_html(url)` to get an `xml_document`

```
html <- read_html("http://rvest.tidyverse.org/")
html
#> {html_document}
#> <html lang="en">
#> [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UT ...
#> [2] <body>\n    <a href="#container" class="visually-hidden-focusable">Ski ...
```

Extracting Data with rvest

2. Find Elements:

```
html <- minimal_html("
  <h1>This is a heading</h1>
  <p id='first'>This is a paragraph</p>
  <p class='important'>This is an important paragraph</p>
")
```

Extracting Data with rvest

2. **Find Elements:**
   - Use `html_elements()/html_element()` with CSS selectors to pick nodes
   - Use `html_text2()` to get text, or `html_attr()` to get an attribute
   - For HTML tables, use `html_table()`

```
html |> html_elements("p")
#> {xml_nodeset (2)}
#> [1] <p id="first">This is a paragraph</p>
#> [2] <p class="important">This is an important paragraph</p>
html |> html_elements(".important")
#> {xml_nodeset (1)}
#> [1] <p class="important">This is an important paragraph</p>
html |> html_elements("#first")
#> {xml_nodeset (1)}
#> [1] <p id="first">This is a paragraph</p>
```

Extracting Data with rvest

2. Find Elements:

`html_elements()` vs `html_element()`

```
html |> html_element("p")
#> {html_node}
#> <p id="first">
```

```
html |> html_elements("b")
#> {xml_nodeset (0)}
html |> html_element("b")
#> {xml_missing}
#> <NA>
```

Nesting Selections

StarWars Example:

```
html <- minimal_html("
  <ul>
    <li><b>C-3PO</b> is a <i>droid</i> that weighs <span class='weight'>167 kg</span></li>
    <li><b>R4-P17</b> is a <i>droid</i></li>
    <li><b>R2-D2</b> is a <i>droid</i> that weighs <span class='weight'>96 kg</span></li>
    <li><b>Yoda</b> weighs <span class='weight'>66 kg</span></li>
  </ul>
  ")
```

- Often you select a set of parent nodes (e.g. items), then within each you select sub-elements.
- Unordered list ($< ul >$) where each list item ($< li >$) contains some information about four characters from StarWars

Nesting Selections

StarWars Example:

```
characters <- html |> html_elements("li")
characters
#> {xml_nodeset (4)}
#> [1] <li>\n<b>C-3PO</b> is a <i>droid</i> that weighs <span class="weight"> ...
#> [2] <li>\n<b>R4-P17</b> is a <i>droid</i>\n</li>
#> [3] <li>\n<b>R2-D2</b> is a <i>droid</i> that weighs <span class="weight"> ...
#> [4] <li>\n<b>Yoda</b> weighs <span class="weight">66 kg</span>\n</li>
```

Nesting Selections

StarWars Example:

```
characters |> html_element("b")
#> {xml_nodeset (4)}
#> [1] <b>C-3PO</b>
#> [2] <b>R4-P17</b>
#> [3] <b>R2-D2</b>
#> [4] <b>Yoda</b>
```

Nesting Selections

StarWars Example:

```
characters |> html_element(".weight")
#> {xml_nodeset (4)}
#> [1] <span class="weight">167 kg</span>
#> [2] NA
#> [3] <span class="weight">96 kg</span>
#> [4] <span class="weight">66 kg</span>
```

```
characters |> html_elements(".weight")
#> {xml_nodeset (3)}
#> [1] <span class="weight">167 kg</span>
#> [2] <span class="weight">96 kg</span>
#> [3] <span class="weight">66 kg</span>
```

If a particular child is missing, `html_element()` returns `NA` rather than dropping the row.

Text and Attributes

- Use `html_text2()` to get the text content of a node

```
characters |>
  html_element("b") |>
  html_text2()
#> [1] "C-3PO"  "R4-P17" "R2-D2"  "Yoda"

characters |>
  html_element(".weight") |>
  html_text2()
#> [1] "167 kg" NA       "96 kg"  "66 kg"
```

Text and Attributes

- Use `html_attr()` to get an attribute value
- Note that these functions handle any escaped HTML entities (e.g. `&`) automatically

```
html <- minimal_html("
  <p><a href='https://en.wikipedia.org/wiki/Cat'>cats</a></p>
  <p><a href='https://en.wikipedia.org/wiki/Dog'>dogs</a></p>
")

html |>
  html_elements("p") |>
  html_element("a") |>
  html_attr("href")
#> [1] "https://en.wikipedia.org/wiki/Cat" "https://en.wikipedia.org/wiki/Dog"
```

Tables with html_table()

If the page already has a proper HTML table:

```
html <- minimal_html("
  <table class='mytable'>
    <tr><th>x</th>   <th>y</th></tr>
    <tr><td>1.5</td> <td>2.7</td></tr>
    <tr><td>4.9</td> <td>1.3</td></tr>
    <tr><td>7.2</td> <td>8.1</td></tr>
  </table>
  ")
```

```
html |>
  html_element(".mytable") |>
  html_table()
#> # A tibble: 3 × 2
#>       x     y
#>   <dbl> <dbl>
#> 1   1.5   2.7
#> 2   4.9   1.3
#> 3   7.2   8.1
```

- Note: conversion guesses variable types; you can turn that off (`convert = FALSE`) and clean manually

Finding the Right CSS Selectors

Picking a good selector is often the hardest part. Useful tools:

- **SelectorGadget** — a bookmarklet that helps you click on positives/negatives to generate selectors
- Browser DevTools (Right-click > Inspect / "Copy as selector")

Case Study: IMDB Top Films

- Tabular and with embedded attributes

Case Study: IMDB Top Films

```r
url <- "https://web.archive.org/web/20220201012049/https://www.imdb.com/chart/top/"
html <- read_html(url)

table <- html |>
  html_element("table") |>
  html_table()
table
#> # A tibble: 250 × 5
#>   ``    `Rank & Title`                        `IMDb Rating` `Your Rating`   ``
#>   <lgl> <chr>                                        <dbl> <chr>           <lgl>
#> 1 NA    "1.\n      The Shawshank Redempt…           9.2 "12345678910\n… NA
#> 2 NA    "2.\n      The Godfather\n      …           9.1 "12345678910\n… NA
#> 3 NA    "3.\n      The Godfather: Part I…           9   "12345678910\n… NA
#> 4 NA    "4.\n      The Dark Knight\n    …           9   "12345678910\n… NA
#> 5 NA    "5.\n      12 Angry Men\n       …           8.9 "12345678910\n… NA
#> 6 NA    "6.\n      Schindler's List\n   …           8.9 "12345678910\n… NA
#> # ℹ 244 more rows
```

Case Study: IMDB Top Films

```r
ratings <- table |>
  select(
    rank_title_year = `Rank & Title`,
    rating = `IMDb Rating`
  ) |>
  mutate(
    rank_title_year = str_replace_all(rank_title_year, "\n +", " ")
  ) |>
  separate_wider_regex(
    rank_title_year,
    patterns = c(
      rank = "\\d+", "\\. ",
      title = ".+", " +\\(",
      year = "\\d+", "\\)"
    )
  )
```

Case Study: IMDB Top Films

```
ratings
#> # A tibble: 250 × 4
#>   rank  title                     year  rating
#>   <chr> <chr>                     <chr>  <dbl>
#> 1 1     The Shawshank Redemption 1994     9.2
#> 2 2     The Godfather            1972     9.1
#> 3 3     The Godfather: Part II   1974     9
#> 4 4     The Dark Knight          2008     9
#> 5 5     12 Angry Men             1957     8.9
#> 6 6     Schindler's List         1993     8.9
#> # ℹ 244 more rows
```

Case Study: IMDB Top Films

Note: sometimes additional information is hidden in attributes not visible in the table itself.

```
html |>
  html_elements("td strong") |>
  head() |>
  html_attr("title")
#> [1] "9.2 based on 2,536,415 user ratings"
#> [2] "9.1 based on 1,745,675 user ratings"
#> [3] "9.0 based on 1,211,032 user ratings"
#> [4] "9.0 based on 2,486,931 user ratings"
#> [5] "8.9 based on 749,563 user ratings"
#> [6] "8.9 based on 1,295,705 user ratings"
```

Case Study: IMDB Top Films

```
ratings |>
  mutate(
    rating_n = html |> html_elements("td strong") |> html_attr("title")
  ) |>
  separate_wider_regex(
    rating_n,
    patterns = c(
      "[0-9.]+ based on ",
      number = "[0-9,]+",
      " user ratings"
    )
  ) |>
  mutate(
    number = parse_number(number)
  )
```

Note the "magic" happening in the first `mutate()` call.

Case Study: IMDB Top Films

```
#> # A tibble: 250 × 5
#>   rank  title                    year  rating  number
#>   <chr> <chr>                    <chr>  <dbl>   <dbl>
#> 1 1     The Shawshank Redemption 1994     9.2 2536415
#> 2 2     The Godfather            1972     9.1 1745675
#> 3 3     The Godfather: Part II   1974     9   1211032
#> 4 4     The Dark Knight          2008     9   2486931
#> 5 5     12 Angry Men             1957     8.9  749563
#> 6 6     Schindler's List         1993     8.9 1295705
#> # ℹ 244 more rows
```