Text Mining and Sentiment Analysis

Prof. Annamaria Bianchi
A.Y. 2024/2025

Lecture 23

26 May 2025

UNIVERSITÀ DEGLI STUDI DI BERGAMO | Dipartimento di Scienze Economiche

Outline

Introduction to Web Scraping with R

Simple examples

Select the name of the main books (R4DS and TidyTextMining)

```
> html |>
+ html_elements(css = "div.mybook p i") |>
+ html_text2()
[1] "R4DS" "TidyTextMining"
```

In alternative it is also possible to specify the attribute in this way:

```
> html |>
+ html_element(css = "[class = 'mybook']") |>
+ html_elements(css = "p i") |>
+ html_text2()
[1] "R4DS" "TidyTextMining"
```

Simple examples

Select the list of the other books:

```
> html |>
+ html_elements(css = "li") |>
+ html_text2()
[1] "Book1" "Book2" "Book3"
```

Exercise. Make the same selection using XPath.

`html_table()` function

The function `html_table()` parses an html table into a data frame

```
html_table(x, header = NA, …)
```

`x` A document (from `read_html()`), node set (from `html_elements()`), node (from `html_element()`), or session (from `session()`).

`header` Use first row as header? If NA, will use first row if it consists of <th> tags. If TRUE, column names are left exactly as they are in the source document, which may require post-processing to generate a valid data frame.

When applied to a single element, `html_table()` returns a single tibble. When applied to multiple elements or a document, `html_table()` returns a list of tibbles.

Simple examples

Select the table:

```
> html |>
+ html_element(css = "table") |>
+ html_table()
# A tibble: 2 × 3
  Book   Author  Price
  <chr>  <chr>   <chr>
1 Title1 Author1 Price1
2 Title2 Author2 Price2
```

```
> html |>
+ html_element(xpath = "//table") |>
+ html_table()
```

Examples with real data

Two main types of webpages:

- **Static webpages** present the same content every time they are viewed.
- **Dynamic webpages** create content instantly in response to user input and present customized or updated information.

Let us start to consider static webpages, using the rvest functions considered so far.

Examples with real data - unibg

Let's scrape and save in a `.txt` file the history of our University.

First, it is necessary to define the url. Then, we read the url with the `read_html()` function

```
> url <- "https://en.unibg.it/about-us/university/history-and-identity"
> html <- read_html(url)
```

Then, we use the CSS selector gadget in order to find the right element. You can select the elements in different ways. The result is almost the same. Indeed, there are multiple ways for implementing the same task.

```
> text <- html |>
+ html_elements(css = "p , .aprichiudi-titolo, .page-header") |>
+ html_text2()
> text
> write(text, file = "text.txt")
```

Examples with real data – Wikipedia table

Let's scrape the table with the administrative division for the Lombardia region from Wikipedia:

```
> url = "https://en.wikipedia.org/wiki/Lombardy"
> html = read_html(url)
> # absolute path
> html |>
+ html_element(xpath =
"/html/body/div[2]/div/div[3]/main/div[3]/div[3]/div[1]/table[5]") |>
+ html_table(header = T)
```

```
# A tibble: 12 × 4
   `Province/Metropolitan city` Area                Population `Density (inh./km2)`
   <chr>                        <chr>               <chr>      <chr>
 1 Province of Bergamo          2,723 km2 (1,051 …  1,108,853  407.2
 2 Province of Brescia          4,784 km2 (1,847 …  1,265,077  264.4
 3 Province of Como             1,288 km2 (497 sq…  599,905    465.7
 4 Province of Cremona          1,772 km2 (684 sq…  361,610    204.4
 5 Province of Lecco            816 km2 (315 sq m…  340,251    416.9
 6 Province of Lodi             782 km2 (302 sq m…  229,576    293.5
 7 Province of Mantua           2,339 km2 (903 sq…  414,919    177.3
 8 Metropolitan City of Milan   1,575 km2 (608 sq…  3,259,835  2,029.7
 9 Province of Monza and Brianza 405 km2 (156 sq m… 864,557    2,134.7
10 Province of Pavia            2,965 km2 (1,145 …  548,722    185.1
11 Province of Sondrio          3,212 km2 (1,240 …  182,086    56.6
12 Province of Varese           1,211 km2 (468 sq…  890,234    735.1
```

Examples with real data – Wikipedia table

Try also with the relative path:

```
> # relative path
> html |>
+ html_element(xpath = "//table[5]") |>
+ html_table(header = T)
```

or with CSS selector…

Scraping multiple web pages

In this example, the objective is to identify and store relevant articles' information from the Journal of Statistical Software (Title, Abstract, Authors, Date and DOI).

https://www.jstatsoft.org/index

We go to Archives and we select volume 99. There are 15 articles, but for simplicity we only consider the first 5.

If we inspect the first two articles, we can notice that there is a common pattern in the web address: `v<volume number>i<article issue number>`. We can exploit this pattern in order to iterate the scraping action over all (5) articles.
Moreover, we want to save all the web pages because they can be useful for future project. We will see an example without saving the web-pages and without a "for" cycle in the case study.
Here we use the SelectorGadget. Try with XPath.

Scraping multiple web pages

```
> url_vol <- "https://www.jstatsoft.org/article/view/v099i"
> issue <- str_c("0", seq(1, 5, 1))
> issue
[1] "01" "02" "03" "04" "05"

> url_list <- str_c(url_vol, issue)
> url_list
[1] "https://www.jstatsoft.org/article/view/v099i01"
[2] "https://www.jstatsoft.org/article/view/v099i02"
[3] "https://www.jstatsoft.org/article/view/v099i03"
[4] "https://www.jstatsoft.org/article/view/v099i04"
[5] "https://www.jstatsoft.org/article/view/v099i05"
> names <- str_c("vol99_issue", seq(1, 5, 1), ".html")
> names[1:5]
[1] "vol99_issue1.html" "vol99_issue2.html" "vol99_issue3.html" "vol99_issue4.html"
[5] "vol99_issue5.html"
> folder <- "jstat/"
> dir.create(folder)
```

Scraping multiple web pages

```
> for (i in 1:length(url_list)) {
+ if (!file.exists(str_c(folder, names[i]))) {
+ download.file(url_list[i], destfile = paste0(folder, + names[i]))
+ Sys.sleep(1)
+ }
+ }
> list.files(folder)
[1] "vol99_issue1.html" "vol99_issue2.html" "vol99_issue3.html" "vol99_issue4.html"
[5] "vol99_issue5.html"
```

Scraping multiple web pages

```
> list_files_path <- list.files(folder, full.names = TRUE)

> authors <- character()
> title <- character()
> date <- character()
> abstract <- character()
> doi <- character()
```

Scraping multiple web pages

```
> for (i in 1:length(list_files_path)) {
+ html <- read_html(list_files_path[i], encoding = "UTF8")
+ authors[i] <- html |> + html_element(css = ".authors_long strong") |>
+ html_text2()
+ title[i] <- html |>
+ html_element(css = ".page-header") |>
+ html_text2()
+ date[i] <- html |>
+ html_element(css = ".article-meta :nth-child(2) .col-sm-8") |>
+ html_text2()
+ abstract[i] <- html |>
+ html_element(css = ".article-abstract") |>
+ html_text2()
+ doi[i] <- html |>
+ html_element(css = ".row:nth-child(3) a") |>
+ html_text2()
+ }
> articles <- tibble(authors = authors, title = title, date = date,
+ abstract = abstract, doi = doi)
> View(articles)
```

Scraping dynamic web pages

In Dynamic web pages, the content which is displayed changes according to user's actions even if the source code remains the same. This is the case of X homepage which is updated automatically when you scroll down the page. Or, for example, when you select some options from a list and your results are filtered.

We cannot use the same techniques of static web pages because they focus on the scraping of the source code, which remains the same. It is the content which changes with our actions. Thus, scraping becomes more difficult or even impossible.

These pages are enriched with **AJAX** (Asynchronous JavaScript and XML) and Javascript tools which allow to interact with the page. From the perspective of the user experience this is an advantage, but it makes web scraping difficult. We do not study AJAX or JavaScript in details.

When you visit such web-pages, the "live" DOM tree changes according to your actions but it is not possible to access it directly using rvest. We need to use a technology that mimic an online session in order to render all the dynamic content and then scrape. This is **Selenium**.

Selenium is a Java-based software and can be used in R through the Rselenium package. In that way, we have a remote-control through R of Selenium which communicates with the web and can gather the live HTML DOM tree in any moment.

You also need to **install Java** https://www.java.com/en/download/

Scraping dynamic web pages – German parliament

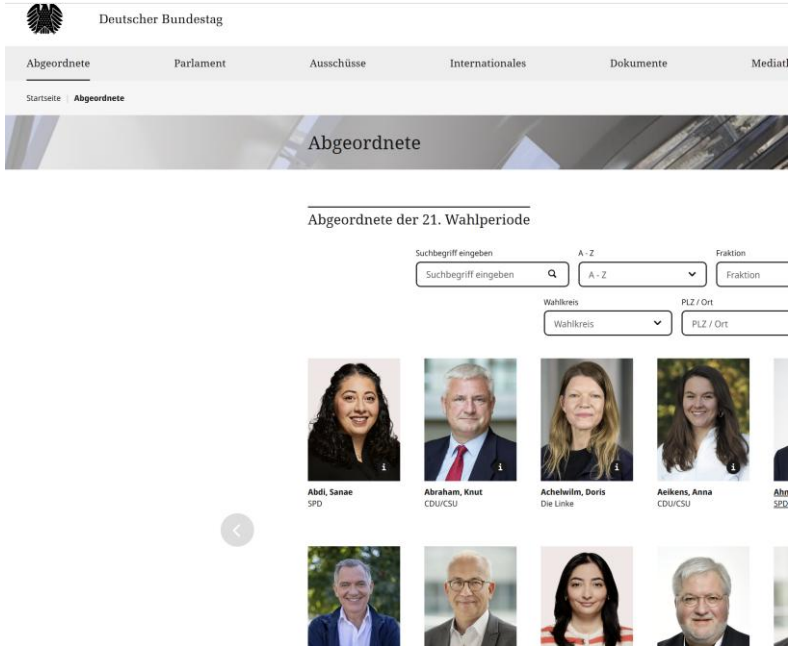
The objective is to scrape all the names of elected politicians. In order to get the list, we need to click on the "list" button (upper right in the page). This is a Dynamic Web Page and to do that, we use RSelenium.

```
> library(RSelenium)
> library(netstat)

> rD <- rsDriver(browser = "firefox",
+ verbose = FALSE,
+ port = free_port(),
+ chromever = NULL)

> remDr <- rD[["client"]]

> url <- "https://www.bundestag.de/abgeordnete/"
> remDr$navigate(url)
```



Scraping dynamic web pages – German parliament

We need to identify and click the list button through R.

```
> # identify and click the list button
> button <- remDr$findElement(using = "css", value = ".icon-list-bullet")
> button$clickElement()

> # save the live DOM tree
> output <- remDr$getPageSource(header = TRUE)
> write(output[[1]], file = "parliament.html")

> # close the connection
> remDr$close()
> rD$server$stop()
[1] TRUE
```

Scraping dynamic web pages – German parliament

Then, we can find and visualize the results

```
> parliament |>
+ html_elements(".bt-list-holder > li:nth-child(1) > a:nth-child(1) > div:nth-
child(1) > div:nth-child(1) > h3:nth-child(1)") |>
+ html_text2()
[1] "\r Abdi, Sanae\r"

> parliament |>
+ html_elements("ul.bt-list-holder li h3") |>
+ html_text2()
[1] "\r Abdi, Sanae\r"
[2] "\r Abraham, Knut\r"
[3] "\r Achelwilm, Doris\r«
….
```

Of course, you should clean and re-arrange the text.

Scraping dynamic web pages – Pew Research Statistics

The objective is to scrape the table which show what percentage of the Total U.S. adult population (and by party) gets news about the coronavirus outbreak on social media.

https://www.pewresearch.org/pathways-2020/covidthreat_a/political_party/democrat_lean_dem/

This can be done in different ways. Here we search for the survey question “Getting COVID-19 news on social media” in the “1. Search for a survey question” query field.

Scraping dynamic web pages – Pew Research Statistics

```
> # set up connection and start browser to navigate the page
> rD <- rsDriver(browser = "firefox",
+ verbose = FALSE,
+ port = free_port(),
+ chromever = NULL)
> remDr <- rD[["client"]]

> url <- "https://www.pewresearch.org/pathways-
2020/covidthreat_a/political_party/democrat_lean_dem/"
> remDr$navigate(url)
```

Scraping dynamic web pages – Pew Research Statistics

```
> # identify and clear search field
> field <- remDr$findElement(using = "css", ".prompt")
> field$clearElement()
> search = "Getting COVID-19 news on social media"
> field$sendKeysToElement(list(search))
> # click on the new tab in order to search
> click <- remDr$findElement(using = "css", ".result")
> click$clickElement()

> # identify the list of political party
> css <- "div.flex-direction-column:nth-child(1) > div:nth-child(2) > div:nth-
child(1) > i:nth-child(2)"
> list <- remDr$findElement(using = "css", value = css)
> list$clickElement()

> # and select u.s. adults
> css <- "div.visible:nth-child(3) > div:nth-child(1) > a:nth-child(1)"
> elem <- remDr$findElement(using = "css", value = css)
> elem$clickElement()
```

Scraping dynamic web pages – Pew Research Statistics

```
> # select the table output
> css <- "a.active:nth-child(2)"
> elem <- remDr$findElement(using = "css", value = "table")
> elem$clickElement()

> # save the output
> output <- remDr$getPageSource(header = TRUE)
> write(output[[1]], file = "covid_news.html")
> # close the connection
> remDr$close()
> rD$server$stop()
[1] TRUE
```

Scraping dynamic web pages – second example

Then, we can parse and clean the data:

```
> covid_news <- read_html("covid_news.html", encoding = "utf-8")
> covid_news |>
+ html_elements("table") |>
+ html_table()
```

```
[[1]]
# A tibble: 1 × 6
  ``                Often Sometimes `Hardly ever` Never Refused
  <chr>             <chr> <chr>     <chr>         <chr> <chr>
1 Total U.S. adults 19%   27%       19%           33%   1%
```