Text as Data

Session 6: Data Collection – Scraping II und APIs

Mirko Wegemann

Universität Münster
Institut für Politikwissenschaft

03 June 2026

Logistics

- last week: first session on data acquisition (with *rvest*)
- today: further forms of data acquisition (scraping dynamic websites and APIs)
- first: short presentation by Jonah

Different Types of Web Scraping

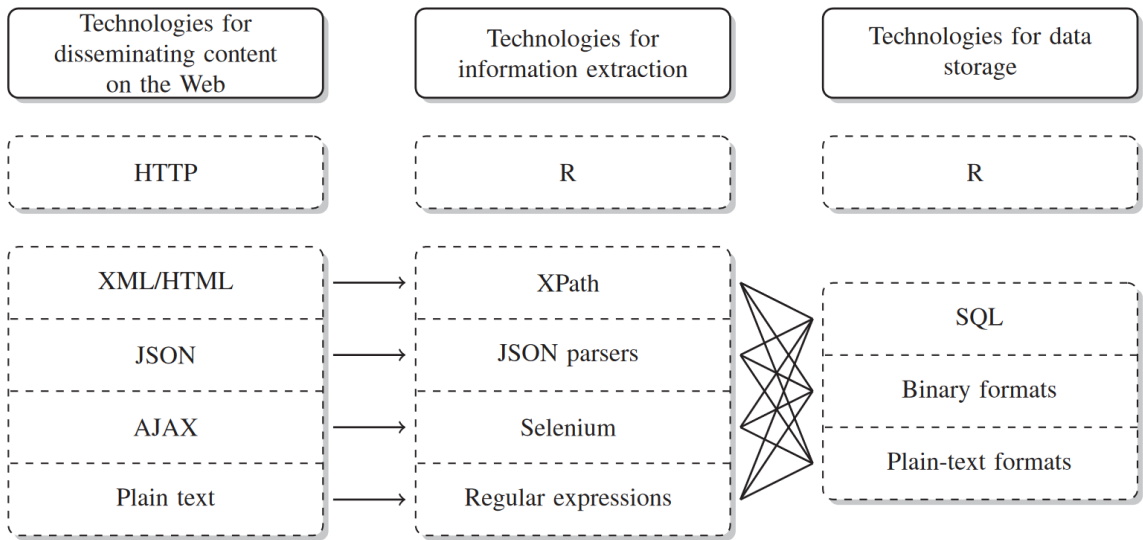
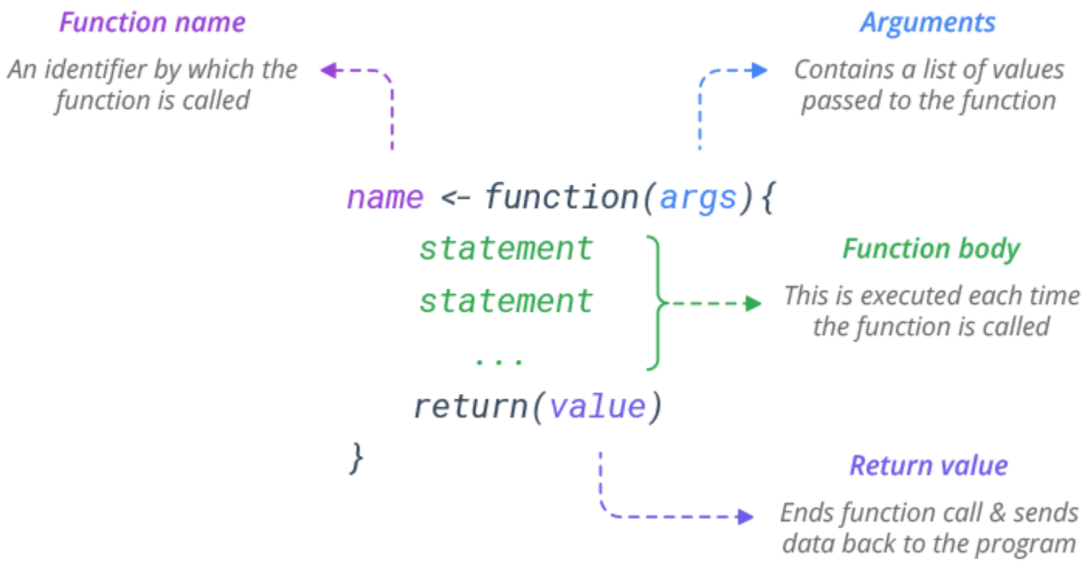


Figure 1.4 Technologies for disseminating, extracting, and storing web data

Munzert (2015, p. 10)

Functions in R



Tutorial on Functions
Graphics and Guide for functions

Functions for Automation

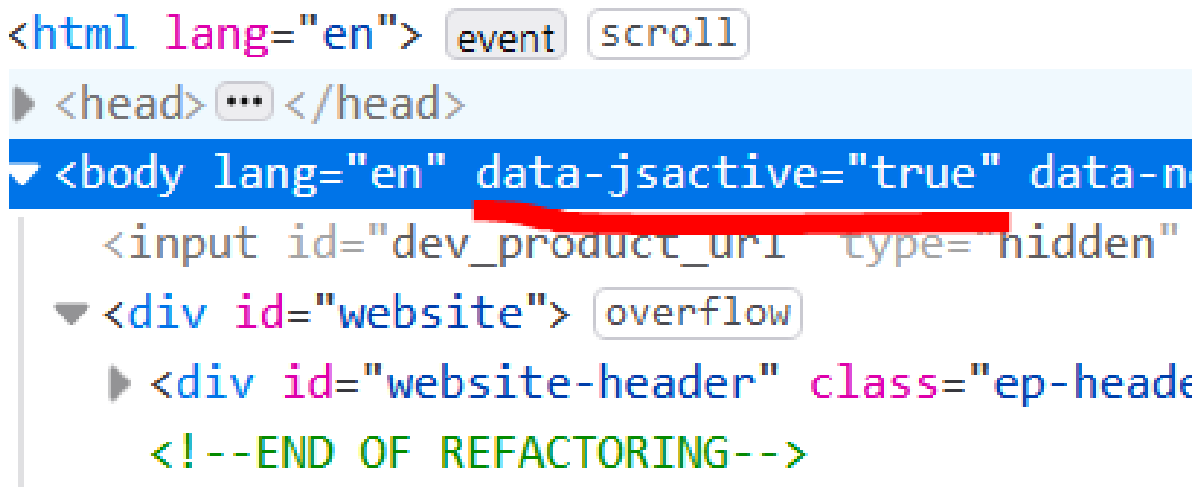
```
1  > h1_scrape <- function(url){
2    + html <- read_html(urls[[url]])
3    + links[url] <- html %>%
4    + html_node("h1") %>%
5    + html_text()
6    + }
7  >
8  > (links <- sapply(1:length(urls), h1_scrape))
9  [1] "Universitaet Muenster" "Hauptinhalt"
```

Dynamic Websites I

Sometimes websites only change when there's an interaction in a browser session (e.g., by clicking on certain objects). For these **websites**, `rvest` is not applicable.

Dynamic Websites II

Usually, such pages can be identified by their use of Javascript



Functionality of RSelenium

Solution?

Figure: RSelenium

RSelenium was originally developed to test websites; we use it to send commands to a virtual browser window (e.g., to click a button)

Before Using RSelenium

Whenever possible, the static approach is preferable to the dynamic one, as it is simpler and less error-prone:

- check other websites that might contain the same information
- search for other subdirectories that might store the same content in a basic HTML format
- check the Wayback Machine
- use a website's search function to list results (Example: press releases from the Die Linke parliamentary group)

RSelenium and wdman Pipeline

A typical workflow looks as follows:

- session configuration (for this, we need `wdman`)
- open browser
- call URL
- accept/reject cookies if necessary
- identify CSS selector for button and click (repeat if necessary)
- retrieve desired object (as before, we usually look for links to individual press releases)

Setup

- configure driver
- assign client to an easy-to-call object
- call website

```
1 > url <- "https://www.europarl.europa.eu/news/en"
2 > rd <- rsDriver(browser = "firefox", +
3          chromever = NULL, +
4          phantomver = NULL, +
5          port = sample(1:65535, 1), +
6          verbose = FALSE)
7 > browser <- rd[["client"]]
8 > browser$navigate(url)
```

And now in R!

Identifying and Clicking a Button

1. Find an element using its CSS selector
2. save it as an object
3. use the function clickElement()

```
1 > cookies <- browser$findElement(using = 'css
    selector', value='.epjs_agree:nth-child(1) span')
2 > cookies$clickElement()
```

How to Extract Text from a Dynamic Website

- identify element **e** with CSS selector (possible via SelectorGadget as before)
- elements usually contain many other pieces of information that are unnecessary for our purposes
- to extract only the plain text, we need to build a loop and extract only the text of the element (`getElementText()` works similarly to `html_text()` from `rvest`)

How to Extract Text from a Dynamic Website II

```
1  > paragraphs <- browser$findElements(using = 'css
       selector', value='p')
2  > text <- c()
3  > for(i in 1:length(paragraphs)){ +
4    text[i] <- paragraphs[[i]]$getElementText()[[1]] +
5    }
6  > text[1]
7  [1] "On Thursday afternoon, the Eurovision debate
       between the lead candidates for the presidency of
       the European Commission took place in the
       European Parliament."
```

How to Extract a Link from a Dynamic Website

- to extract only the link, we need the function `getElementAttribute()`, which is similar to `html_attr()` from `rvest`

```
> urls_euparl <- browser$findElements(using = 'css
    selector', value='.ep_title > a')
> urls_euparl2 <- c()
> for(i in 1:length(urls_euparl)){ +
  urls_euparl2[i] <-
      urls_euparl[[i]]$getElementAttribute('href')[[1]]
  }
```

Data Processing

Once we have downloaded the individual links to pages, we can often continue with `rvest` (which is easier to handle).

Further Possibilities with RSelenium

In principle, we can mimic any browser function with RSelenium. A frequently helpful function is, for example, filling out forms on pages (such as a search field).

```
1 > browser$navigate(url)
2 > search <- browser$findElement(using = 'css
    selector', value="#search-field")
3 > search$clearElement
4 >
    search$sendKeysToElement(list("economy",key="enter"))
```

What are APIs?

Application **P**rogramming **I**nterfaces are interfaces provided by website operators that allow us to download and further process data.

- mostly aimed at web developers who can, for example, develop apps that access data from a website
- accordingly, it is possible that data interesting for researchers has not always been made accessible (e.g., we cannot download full texts via the NYT API)

Access to APIs

Access to APIs can usually be obtained through registration.

- some registrations are free and simple (e.g., Manifesto Project or NYT)
- others are heavily restricted and only conditionally free (e.g., X API)

Using the API

After registration, we usually receive an access key through which we can access the API.

- Manual access via JSON functions (`jsonlite`), in combination with API documentation (e.g., NYT Article Search API)
- Many APIs have so-called wrappers; implementations that greatly simplify their use in `R`

API: Setting the API Key

Use the `usethis` package to store your API key in the project environment using the `edit_r_environ()` function. This serves as security against misuse, so that your key is not openly visible in the code.

```
> if (!require('usethis')) install.packages("usethis")
> edit_r_environ()
```

For the NYT API, you would write NYTIMES_API_KEY="YOUR_KEY" into the environment (replace "YOUR_KEY" with your API key). Now you must save the environment and restart RStudio.

API: Defining and Sending a Query

```
1 > if (!require("jsonlite"))
    install.packages("jsonlite") # manual handling of
    APIs
2 > url_query <-
    paste0("https://api.nytimes.com/svc/search/v2/
    articlesearch.json?q=trump&api-key=",
    Sys.getenv("NYTIMES_API_KEY"))
3 > query <- fromJSON(url_query)
```

API: Converting Data into a Dataset

We receive the data in a list, which in turn consists of several lists (including meta-information about the API request). We are usually interested in the data, which in this case we can convert into a `data.frame` via the nested structure.

```
> df <- data.frame(query$$response$$docs)
```

API: Automation

Even with APIs, we often have to build loops to obtain more than a certain number of data points.

```
1  > df <- c()
2  > for(i in 0:5){
3    url_query <-
         paste0("https://api.nytimes.com/svc/search/v2/
         articlesearch.json?q=trump&api-key=",
         Sys.getenv("NYTIMES_API_KEY"), "&page=", i)
         query
4    df_temp <- data.frame(date =
         query$$response$$docs$$pub_date, url =
         query$$response$$docs$$web_url, summary =
         query$$response$$docs$$abstract, lead_paragraph
         = query$$response$$docs$$lead_paragraph)
5    df <- rbind(df, df_temp)
6    Sys.sleep(12)
7  }
```

The Manifesto Project

Figure: The Manifesto Project

The Manifesto Project is an international research project that has collected and coded election programs from more than 1,000 parties in 68 countries.

ManifestoR

The Manifesto Project offers an API through which we can access the project's corpus (with more than 1.9 million "quasi-sentences").

- Access either again via `jsonlite`
- ...or `manifestoR`, an `R` wrapper

Here we only discuss the implementation using `manifestoR`.

And now in R!

Setting the API Key

As before, we must first store and set the API key in our environment variable.

```
1 > if (!require("manifestoR"))
      install.packages("manifestoR")
2 > mp_setapikey(key=Sys.getenv("MP_API_KEY"))
```

Data Access

The data of the Manifesto corpus can be retrieved quite easily using the `mp_corpus` function. We must always **specify** which text segments we need (either via *countryname* or *party*).

```
1 > german_mp <- mp_corpus_df(countryname=="Germany")
```

Outlook

- we have learned different ways to collect web-data
- next week, we will prepare data for text analysis
- Literature:
  1. Denny, M., & Spirling, A. (2017, September). Text Preprocessing for Unsupervised Learning: Why It Matters, When It Misleads, and What to Do about It. https://doi.org/10.2139/ssrn.2849145
  2. Hvitfeldt, E., & Silge, J. (2022). *Supervised Machine Learning for Text Analysis in R*. CRC Press – Chapters 2-4

Literature I

Denny, M., & Spirling, A. (2017, September). Text Preprocessing for Unsupervised Learning: Why It Matters, When It Misleads, and What to Do about It. https://doi.org/10.2139/ssrn.2849145

Hvitfeldt, E., & Silge, J. (2022). *Supervised Machine Learning for Text Analysis in R*. CRC Press.

Munzert, S. (2015). *Automated Data Collection with R: A Practical Guide to Web Scraping and Text Mining* (1st ed.). Wiley.