

Introduction to Applied Data Science

Lecture 4: Getting Data, Web Scraping

Bas Machielsen
Utrecht University
2024-04-22

Web Scraping

Lecture 4: Web Scraping

- Overview of this class:
 - Lecture 1: Introduction to Data Science & R
 - Lecture 2: Introduction to Programming
 - Lecture 3: Getting Data, API & Databases
 - *This lecture:* Lecture 4: Getting Data, Web Scraping
 - Lecture 5: Transforming and Cleaning Data
 - Lecture 6: Spatial and Network Data
 - Lecture 7: Text Data & Text Mining
 - Lecture 8: Data Science Project

Introduction: What is Web Scraping?

- Web scraping is the process of extracting data from websites
- It involves **fetching HTML content** from web pages and then parsing and extracting the desired information
- R provides powerful libraries and tools for web scraping
- It allows us to **automate data collection** tasks, and extract structured data from unstructured sources
- The most important ingredient is the `rvest` package

```
library(pacman)  
p_load(rvest)
```

Step By Step

- Step by step, web scraping involves:
 - **Fetch HTML:** Download the HTML content of the web page.
 - **Parse HTML:** Parse the HTML content to extract the relevant elements.
 - **Extract Data:** Use **CSS selectors** to locate and extract the desired data.
 - **Process Data:** Process and clean the extracted data as needed.
 - **Analyze Data:** Analyze the extracted data using R's data manipulation and visualization tools. Strictly speaking, not part of the web scraping process *per se*.

HTML Code

- To do web scraping, you need to know some **basic HTML**.
- HTML stands for “Hyper Text Markup Language”.
- HTML page consists of **series of elements** which browsers use to interpret how to display the content.
- HTML tags are names of the elements surrounded by angle brackets like so: `<tagname>`
content goes here... `</tagname>`.
- Most HTML tags **come in pairs** and consist of opening and a closing tag, known as start tag and end tag, where the end tag name is preceded by the forward slash `/`.

HTML Structure

- Below is a visualization of a simple HTML page structure:

```
<html>
  <head>
    <title> Page title </title>
  </head>
  <body>
    <h1> Page title </h1>
    <p> This is a paragraph. </p>
    <p> This is another paragraph </p>
  </body>
</html>
```

- Pretty much all webpages look like this, but there are exceptions

Elements

- Elements are the **basic building blocks of HTML code**
- There are over 100 HTML elements:
- Every HTML page must be in an `<html>` element, and it must have two children: `<head>`, which contains document metadata like the page title, and `<body>`, which contains the content you see in the browser.
 - Block tags like `<h1>` (heading 1), `<p>` (paragraph), `<div>` (division), `` (ordered list) or `` (list) form the overall structure of the page.
 - Inline tags like `` (bold), `<i>` (italics), and `<a>` (links) formats text inside block tags.
- If you encounter a tag that you've never seen before, you can find out what it does with a little googling.

Example HTML Page

- Roughly, then, a **basic website** might look something like this:

```
<html>
  <head>
    Page Title
  </head>
  <body>
    <p>First Paragraph</p>
    <div>Second Paragraph</div>
    <ol>
      <li>Coffee</li>
      <li>Tea</li>
      <li>Milk</li>
    </ol>
  </body>
</html>
```

Attributes and Classes

- It is possible to write slightly more complicated HTML code
- Usually, people define HTML attributes inside HTML tags.
- These **attributes** provide additional information about HTML elements, such as hyperlinks for text, and width and height for images.
- Attributes are always defined in the start tag and come in `name="value"` pairs, like so:

```
<a href="https://www.example.com">This is a link</a>
```

 - Here `href` is the attribute (name) and its value is `"https://www.example.com"`
- HTML tags often also have **classes**
- We'll see examples of attributes and classes shortly
- The bottom line of web scraping is that we can **use these attributes and classes** later to find the piece of HTML code we need to **extract data**

More Complicated Example of an HTML

- What are the elements, attributes and classes in this HTML code?

```
<html>
  <head>
    <title>My First HTML Page</title>
  </head>
  <body>
    <h1>Hello, HTML!</h1>
    <p href="http://www.hello.com">This is a paragraph of text.</p>
    <div class="paragraph">This is also some enclosed text.</div>
    An Image</img>
  </body>
</html>
```

CSS Selectors

CSS Selectors

- The second **major ingredient** we need for web scraping is an understanding of CSS Selectors
- CSS is a language that describes how HTML elements should be displayed.
- One of the ways to **define useful shortcuts** for selecting HTML elements to style is by using CSS selectors.
- CSS selectors represent **patterns** for locating HTML elements.
- This is what we use to find particular attributes in a HTML page, and extract them or their text into R
- We will shortly see some examples of CSS selectors

CSS Selector Logic

- We have seen that CSS selectors are patterns used to select elements in an HTML document.
- They are commonly used in web scraping to specify which elements to extract.
 - **Simple selectors** include element selectors, class selectors, and ID selectors.
- **Element Selector**: Selects all elements of a specified type. Example: `p` selects all paragraph elements.
- **Class Selector**: Selects elements with a specific class attribute. Example: `.class` selects all elements with the class "class".
- **ID selector**: Selects an element with a specific attribute. For example: `#id` selects the element with the ID "id".

More CSS Selector Attributes

- A very important and often-used selector is the **attribute selector**
- The Attribute Selector (`[attribute=value]`) selects elements with a specific attribute-value pair.
 - Example: `[href="https://example.com"]` selects all elements in the page with the href attribute equal to `"https://example.com"`.
- Other Example: in the below code, the CSS Selector `[href="http://www.hello.com"]` would return the line: `<p href="http://www.hello.com">This is a paragraph of text.</p>`

```
<html>
  <head>
    <title>My First HTML Page</title>
  </head>
  <body>
    <h1>Hello, HTML!</h1>
    <p href="http://www.hello.com">This is a paragraph of text.</p>
    An Image</img>
  </body>
</html>
```

More CSS Selector Attributes

- There also exists a Partial Match Attribute Selector (`[attribute*="value"]`): this selects elements with an attribute value containing the specified substring.
 - Example: `[href*="example"]` selects all elements with the href attribute containing "example".
- You can also **select by text** in various ways:
 - Use the `:contains()` selector to select elements containing exact text. For example: `p:contains('Lorem ipsum')` selects all `<p>` elements containing the exact text "Lorem ipsum".
 - It is possible to do this case-insensitively: e.g. `p:contains('lorem ipsum' i)` selects all `<p>` elements containing the text "lorem ipsum" regardless of case.
 - You can use the `^` symbol to select elements where the text starts with a specific string. E.g. `a[href^="https://"]` selects all `<a>` elements whose href attribute starts with "https://".
 - You can use the `$` symbol to select elements where the text ends with a specific string. Example: `a[href$=".pdf"]` selects all `<a>` elements whose href attribute ends with ".pdf".

More Complicated CSS Selectors

- CSS selectors can also specify **relationships between elements**.
- Common relationships include parent-child (`>`) and descendant () selectors.
- A parent-child selector selects all `` elements that are direct children of a `` element. Example: `ul > li`
- A descendant selector selects all `` elements that are descendants of a `` element, regardless of their depth. Example: `ul li`
- An Adjacent Sibling Selector (`+`) selects the element that is immediately preceded by a specified sibling element. Example: `h2 + p` selects all `<p>` elements that are immediately preceded by an `<h2>` element.
- A General Sibling Selector (`~`) selects all sibling elements that follow a specified element. Example: `h2 ~ p` selects all `<p>` elements that follow an `<h2>` element at the same level in the tree, regardless of their position.

CSS Selectors Cheat sheet

- On a complicated web page, it is sometimes difficult to find the code for a good CSS selector
 - Potentially, you could use the automatized CSS selector in the Inspect window in your browser, which automatically generates CSS
 - Right Click the **Web Page > Click Inspect > Select the Element > Right-Click > Copy > CSS Selector**
 - This is often not robust, meaning that it does not find the element after the website owner makes some changes, but leaves the stuff you are looking for on it
- **Here** and **here** are CSS cheat sheets that help you find the right selector.
- But in general, the best way to get used to web scraping is by looking at real-life examples.

Web Scraping in R

- All of this is fairly abstract, but it turns out this is **easy to implement** in R:
- To get started scraping, you'll need the URL of the page you want to scrape, which you can usually copy from your web browser:

```
library(rvest)
html ← read_html("http://www.uu.nl/")
html
```

```
## {html_document}
## <html lang="nl" dir="ltr" prefix="og: https://ogp.me/ns#">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">\n<li
## [2] <body class="full-width"> <noscript><iframe src="https://www.googletagmanager.co
```

Using CSS Selectors

- Let's use some of the selectors we've learned, for example the element selector for

```
<p>:
```

```
html ▷
```

```
html_elements('p') # Inside this function, you use the CSS Selector
```

```
## {xml_nodelist (6)}
```

```
## [1] <p> Onderzoeker Joeri Zwerts en collega's concluderen dit aan de hand van 1,3 mi
```

```
## [2] <p>Luister op 31 mei naar lezingen, muziek, film, comedy, poëzie en een lijsttre
```

```
## [3] <p>Een verdiepend artikel over de stijgende zeespiegel en hoe we het tij kunnen
```

```
## [4] <p>Niki Frantzeskaki wordt de nieuwe wetenschappelijk directeur van het strategi
```

```
## [5] <p>Hoogleraar Ingrid Robeyns maakt zich al jaren hard voor het limitarisme: <q>N
```

```
## [6] <p>Universiteit Utrecht<br>Heidelberglaan 8<br>3584 CS Utrecht<br>Nederland<br>T
```


Extracting Text And Attributes

- After having read in the HTML page and having found the correct elements to scrape, it is often a question of just extracting a particular text or a particular attribute:
- Text using the `html_text2()` function

```
html ▷  
  html_elements('.grid__item') ▷  
  html_text2() ▷  
  head(3)
```

```
## [1] ""
```

```
## [2] "FSC-certificering vergroot biodiversiteit tropische bossen\n\nOnderzoeker Joeri
```

```
## [3] "Veelgezocht\nUniversiteitsbibliotheek\nBotanische Tuinen\nVacatures\nPromoveren
```

Extracting Text And Attributes

- Attributes using the `html_attr()` function:

```
html ▷  
  html_elements('a.people-item__link') ▷  
  html_attr('href')
```

```
## [1] "/organisatie/festival-europa"  
## [2] "/organisatie/verdieping/boven-het-water-uitstijgen"  
## [3] "/nieuws/niki-frantzeskaki-benoemd-tot-nieuwe-wetenschappelijk-directeur-van-pat  
## [4] "/in-de-media/wat-is-het-limitarisme-hoogleraar-ingrid-robeysn-legt-het-uit"
```

Summarizing

- You can web scrape with only *three* functions:
 - `read_html()` for reading in the HTML code of a particular web page
 - `html_elements()` using CSS selectors to isolate the relevant pieces you want to scrape
 - `html_text2()` or `html_attr()` to extract the data you might be looking for

Tables

- If you're lucky, your data will be already stored in an HTML table, and it'll be a matter of just reading it from that table using the `read_table()` function:

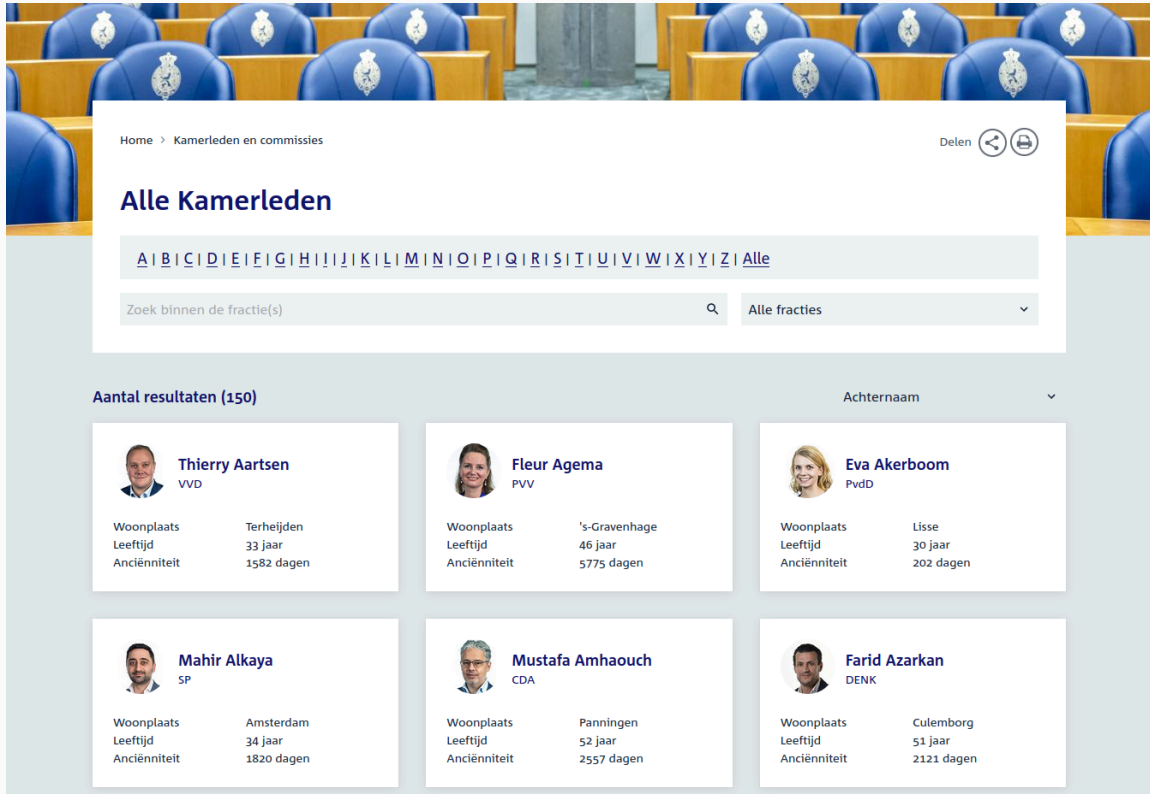
```
read_html('https://en.wikipedia.org/wiki/List_of_countries_by_GDP_(PPP)') ▷  
  html_table() ▷  
  pluck(2)
```

```
## # A tibble: 230 × 8  
##   `Country (or territory)` `UN region` `IMF[1][5]` `IMF[1][5]` `World Bank[6]` `Wor  
##   <chr>                   <chr>      <chr>      <chr>      <chr>          <chr>  
## 1 Country (or territory) UN region  Forecast   Year        Estimate      Year  
## 2 World                   —         185,677,122 2024       164,155,327 2022  
## 3 China                   Asia      35,291,015  [n 1]2024  30,327,320   [n 2  
## 4 United States           Americas  28,781,083  2024       25,462,700   2022  
## 5 India                   Asia      14,594,460  2024       11,874,583   2022  
## 6 Japan                   Asia      6,720,962   2024       5,702,287    2022  
## 7 Germany                 Europe    5,686,531   2024       5,309,606    2022  
## 8 Russia                  Europe    5,472,880   2024       5,326,855    2022  
## 9 Indonesia               Asia      4,720,542   2024       4,036,901    2022  
## 10 Brazil                 Americas  4,273,668   2024       3,837,261    2022  
## # i 220 more rows
```

Example: Scraping Lower House Members

Example: Lower House Data

- Have a look at this webpage ([Source](#))









Home > Kamerleden en commissies Delen

Alle Kamerleden

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | Alle

Zoek binnen de fractie(s) Alle fracties

Aantal resultaten (150) Achternaam

 Thierry Aartsen VVD	 Fleur Agema PVV	 Eva Akerboom PvdD
Woonplaats: Terheijden Leeftijd: 33 jaar Anciënniteit: 1582 dagen	Woonplaats: 's-Gravenhage Leeftijd: 46 jaar Anciënniteit: 5775 dagen	Woonplaats: Lisse Leeftijd: 30 jaar Anciënniteit: 202 dagen
 Mahir Alkaya SP	 Mustafa Amhaouch CDA	 Farid Azarkan DENK
Woonplaats: Amsterdam Leeftijd: 34 jaar Anciënniteit: 1820 dagen	Woonplaats: Panningen Leeftijd: 52 jaar Anciënniteit: 2557 dagen	Woonplaats: Culemborg Leeftijd: 51 jaar Anciënniteit: 2121 dagen

Webscraping Lower House Data

- After visiting the website and inspecting the HTML code (right click > inspect), we found that:
- Each Lower House member is located in a `<div>` of class `m-card__content`:
 - CSS selector: `div.m-card__content`
- Inside this `<div>`, the name is located in a `<h3>` and the political party is located in a ``
 - The latter `` has class `u-text-size--small`. The selector being `span.u-text-size--small`.
- The remaining data is stored inside a `<table>` object
- On the next slide, we will **extract** these data on the basis of these selectors and rearrange them, for each politician on the page

Scraping Code

- This is the code:

```
library(rvest, quietly = TRUE); library(tidyverse)

# Scrape the webpage
url ← "https://www.tweedekamer.nl/kamerleden_en_commissies/alle_kamerleden"
webpage ← read_html(url)
# Extract all the tables containing the data pieces
table ← html_elements(webpage, "div.m-card__content")
# Extract the data from each member separately
data ← lapply(table, function(x) {

  name_pol ← html_elements(x, 'h3') ▷ html_text() ▷ str_squish()
  party_pol ← html_elements(x, 'span.u-text-size--small') ▷ html_text()
  table ← html_elements(x, 'table') ▷ html_table() ▷ pluck(1)

  rearranged_table ← table ▷ pivot_wider(names_from=X1,
                                         values_from=X2)

  rearranged_table ▷
    mutate(name=name_pol, party=party_pol)

})

# Bind the data together
data ← bind_rows(data)
```

Inspect the Data

- This is what the dataset looks like:

```
data ▶ head(10)
```

```
## # A tibble: 10 × 5
##   Woonplaats      Leeftijd Anciënniteit name                party
##   <chr>          <chr>    <chr>          <chr>              <chr>
## 1 Drachten      61 jaar  139 dagen      Max Aardema         PVV
## 2 Terheijden    34 jaar  2049 dagen     Thierry Aartsen     VVD
## 3 Utrecht       41 jaar  139 dagen      Ismail el Abassi   DENK
## 4 's-Gravenhage 47 jaar  6242 dagen     Fleur Agema         PVV
## 5 Rotterdam     32 jaar  1119 dagen     Stephan van Baarle DENK
## 6 Eindhoven     38 jaar  139 dagen      Mpanzu Bamenga     D66
## 7 Amsterdam     41 jaar  2588 dagen     Thierry Baudet     FVD
## 8 Wassenaar     38 jaar  2364 dagen     Bente Becker       VVD
## 9 Groningen     41 jaar  2588 dagen     Sandra Beckerman   SP
## 10 Gouda        41 jaar  1119 dagen     Mirjam Bikker      ChristenUnie
```

Correct Code

- In comparison to the incorrect code offered to us by ChatGPT, we changed a couple of things:
- We changed the table object from:

```
table ← html_nodes(webpage, "table") %>% .[[1]]
```

to:

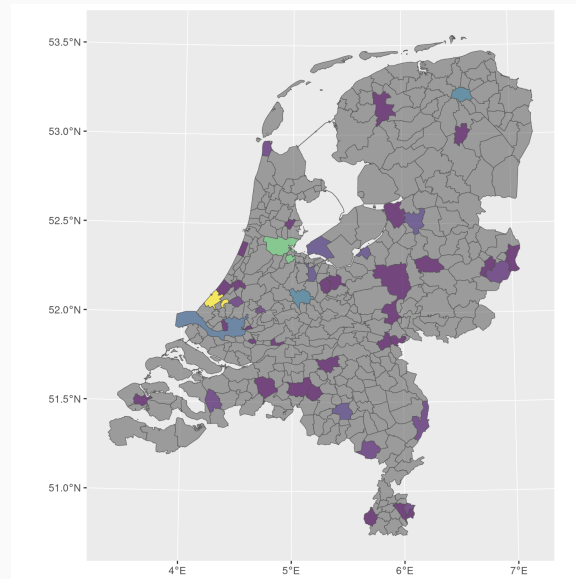
```
table ← html_elements(webpage, "div.m-card__content")
```

- And we changed the function to pick up `Woonplaats`, `Ancienniteit` and `Leeftijd` more carefully: more about this soon
- In all cases, we made use of CSS selectors: we made use of `<h3>`, ``, and finally `<table>` tags!

Demonstration

- Let's first see if we can do something nice with this newly obtained dataset.

```
library(tmap, quietly = TRUE); library(sf, quietly = TRUE); data("NLD_muni")  
freq <- data > group_by(Woonplaats) > summarize(count = n())  
NLD_muni <- NLD_muni > left_join(freq, by = c("name" = "Woonplaats"))  
NLD_muni > ggplot() + geom_sf(alpha=0.7,aes(fill=count)) + scale_fill_continuous()
```



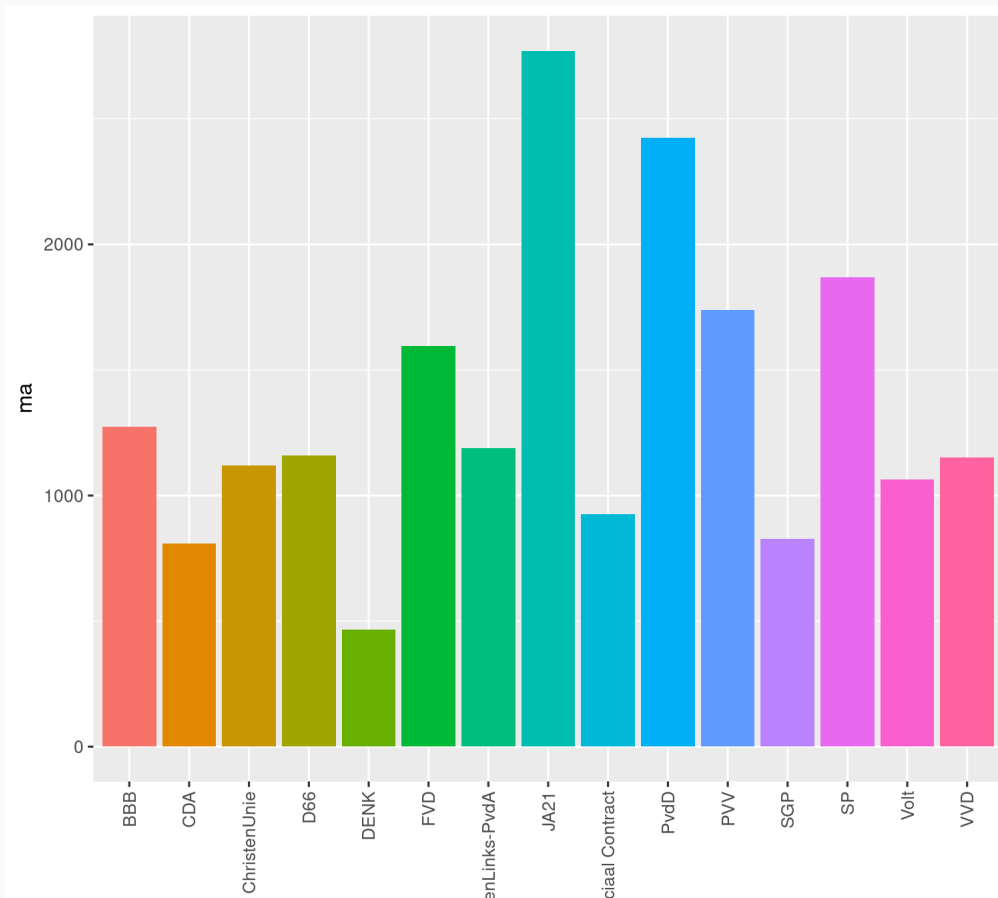
Demonstration

data ▷

```
group_by(party) ▷
```

```
summarize(ma = mean(parse_number(Anciënniteit))) ▷
```

```
ggplot(aes(x = party, y = ma)) + geom_col(aes(fill = party)) + theme(axis.text.x
```



CSS Selectors in the example

- If we focus once again on the [Dutch Lower House website](#), right-click with your mouse and then select `Inspect (Q)`
- A subscreen will pop-up with the html code underlying the website
- Your job is then to **find the elements** you want to scrape, and find the pattern
 - The first thing you realize is that all the information of one particular politician is located in a `<div>` of class `m-card__content`
 - Secondly, inside the `<div>`, there is a `<table>` containing three of the variables, which can be easily extracted by focusing on the `<table>` and then using the `html_table()` function in the `rvest` package
 - The other two attributes, the name and the party, are extracted using the `<h3>` and `` attributes in which they are located

Another Example: Wikipedia

Wikipedia Example

- Sometimes, we can also extract elements at once, without having to write a separate piece of code for each unit we're interested in
- The `rvest` package has the function `html_table`, allowing us to extract data from a table into a `data.frame` at once.
- To demonstrate, have a look at a Wikipedia page about football match outcomes:
[https://nl.wikipedia.org/wiki/Eredivisie_2023/24_\(mannenvoetbal\)](https://nl.wikipedia.org/wiki/Eredivisie_2023/24_(mannenvoetbal))
 - We will scrape the table that contains the scores (Dutch: *Stand*)

Wikipedia Example

- The CSS selector in this script says:
 - Look for the `<h3>` with the text 'Stand', then look for the next `<div>` afterward, and select the `<table>` inside that `<div>`.

```
url ← "https://nl.wikipedia.org/wiki/Eredivisie_2023/24_(mannenvoetbal)"
```

```
table ← url ▷
```

```
  read_html() ▷
```

```
  html_element(css = 'h3:contains("Stand") + div > table') ▷
```

```
  html_table()
```

- Note that we made use of the Adjacent Sibling Selector (+) and the Parent-Child selector (>)
 - Have a look at the HTML code of the page to see why: there are a lot of tables, and we only want this one!

Result

- The result you get is this:

```
table ▷ head(10)
```

```
## # A tibble: 10 × 12
##   Pos Pw Team Wed W G V Ptn DV DT `+/-` `Kwa
##   <int> <lgl> <chr> <int> <int> <int> <int> <chr> <int> <int> <chr> <chr>
## 1 1 1 NA PSV (Q) 30 26 3 1 81 95 17 +78 "Lea
## 2 2 2 NA Feyenoord (C, X) 30 22 6 2 72 77 23 +54 "Lea
## 3 3 3 NA FC Twente (Y) 30 18 6 6 60 56 30 +26 "Der
## 4 4 4 NA AZ (Z) 30 16 7 7 55 59 35 +24 "Tic
## 5 5 5 NA Ajax 30 13 9 8 48 63 56 +7 "Twe
## 6 6 6 NA N.E.C. 30 12 11 7 47 59 44 +15 "Pla
## 7 7 7 NA FC Utrecht 30 12 9 9 45 43 41 +2 "Pla
## 8 8 8 NA Go Ahead Eagles 30 11 9 10 42 44 39 +5 "Pla
## 9 9 9 NA Sparta Rotterdam 30 11 7 12 40 45 43 +2 "Pla
## 10 10 10 NA sc Heerenveen 30 10 6 14 36 50 56 -6 ""
```

Recapitulation

Recapitulation

- Today, we have seen various examples of **web scraping**, and how to get (usually free) data from the web
- We have also made an attempt to understand *how webpages are structured* in the form of HTML code
- We found a language that helps us *select* the relevant elements we are looking for on a HTML page and dissected its logic
- We discussed some functions helping us to **extract the relevant data** after applying the first two steps
- What have we not done?
 - Dynamic webpages / Selenium (more advanced)
 - Other sources of data, e.g. pdf (`tabulizer` package)