Text Mining and Sentiment Analysis

Prof. Annamaria Bianchi
A.Y. 2024/2025

Lecture 21

20 May 2025

Outline

rvest package

`rvest` helps scrape (or harvest) data from web pages. `rvest` is a member of the tidyverse but is not a core member, so you will need to load it explicitly

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

Parsing

- HTML documents use markup to store information and create the visual appearance of the webpage when opened in the browser.
- The first step in web scraping is to load and represent the contents of HTML files in an R session. To achieve a useful representation of HTML files, we need to employ a function that **understands** the special meaning of the markup structures and reconstructs the implied hierarchy of an HTML file within some R-specific data structure. This representation is also referred to as the **Document Object Model** (**DOM**). This transformation is called **parsing**.

Parsing

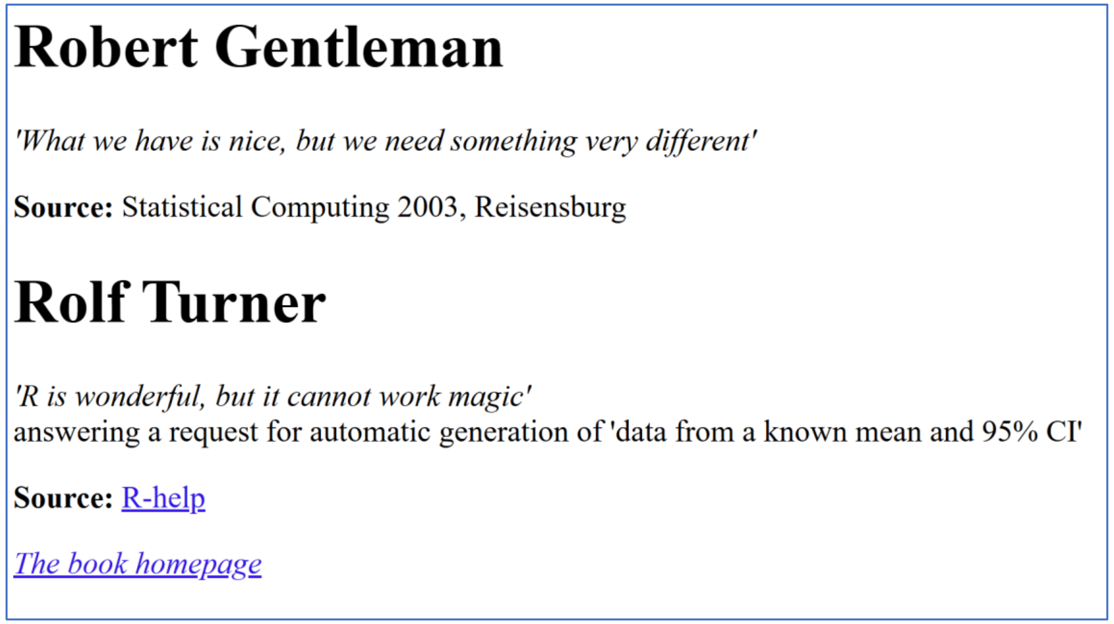
`read_html()` parses web documents and returns an xlm_document, which you'll then manipulate usign rvest functions. It works by performing a HTTP request then parsing the HTML received using the xml2 package. This is "static" scraping because it operates only on the raw HTML file.

`read_html(x,…)`

`x` Usually a string representing a URL or a path for an HTML file

Parsing

Let us consider the fortunes.html example from the Munzert et al. (2015) book:



Robert Gentleman

'What we have is nice, but we need something very different'

Source: Statistical Computing 2003, Reisensburg

Rolf Turner

'R is wonderful, but it cannot work magic'
answering a request for automatic generation of 'data from a known mean and 95% CI'

Source: R-help

The book homepage

```
> fortunes <- read_html("fortunes.html")
> class(fortunes)
[1] "xml_document" "xml_node"
> fortunes
{html_document}
<html>
[1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset ...
[2] <body>\n<div id="R Inventor" lang="english" date="June/2003">\n <h ...
[3] <div class="XTranslate"></div>
```

Xpath and CSS Selector

Two ways for extracting the relevant information from HTML files:

- XPath is a query language. It is based on the hierarchical structure in the DOM in order to specify the path.
- CSS Selectors identify elements on the basis of their style.

Assume we want to extract information from the <i> nodes (text written in *italics*) from the fortune.html file

`html_element()` function

`html_element()` and `html_elements()` find HTML element using CSS selectors or XPath expressions.

```
html_element(x, css, xpath)
```

```
html_elements(x, css, xpath)
```

`x`	Either a document, a node set or a single node.
`css, xpath`	Elements to select. Supply one of css or xpath depending on whether you want to use a CSS selector or XPath 1.0 expression.

The function `html_element()` extracts the first element with the specified characteristic and `html_elements()` extracts all the elements with that characteristic.

XPath

Main characteristics:

- **Absolute paths**: start with the root node and continue until the target one. Each note is separated by a slash. For example:

```
> html_elements(fortunes, xpath = "/html/body/div/p/i")
{xml_nodeset (2)}
[1] <i>'What we have is nice, but we need something very different'</i>
[2] <i>'R is wonderful, but it cannot work magic'</i>
```

- **Relative paths**: we use double slashes "//" to skip nodes. For example:

```
> html_elements(fortunes, xpath = "//p/i")
{xml_nodeset (2)}
[1] <i>'What we have is nice, but we need something very different'</i>
[2] <i>'R is wonderful, but it cannot work magic'</i>
```

You can choose absolute or relative paths according to your objective. Relative paths are faster to write but require more computation time if the file is big.

XPath

- **Wildcard operator**: it is the "*" whith matches any node.

```
> html_elements(fortunes, xpath = "/html/body/div/*/i")
{xml_nodeset (2)}
[1] <i>'What we have is nice, but we need something very different'</i>
[2] <i>'R is wonderful, but it cannot work magic'</i>
```

- **Predicates** and **node relations**: express conditions on the node. We do not study these cases.

CSS Selector

Definition: CSS means Cascading Style Sheets and it is used to describe the layout of an HTML document.

CSS is added to HTML in three ways:

- **Inline**: using the `style` attribute inside HTML element:

```
<h1 style="color:green;">Heading text is green</h1>
```

- **Internal**: using the `<style>` element in the `<head>` section:

```
<head>
  <title>Page title</title>
  <style>
.myDiv {
  border: 5px outset red;
  background-color: lightblue;
  text-align: center;
}
</style>
</head>
```

CSS Selector

- **External**: using a link to a style sheet in the `<head>` section

```
<head>
  <link rel="stylesheet" href="styles.css">
</head>
```

We do not focus on styles but we use the CSS selectors which define the elements to which CSS apply.

CSS includes a miniature language for selecting elements on a page called CSS selectors. CSS selectors define patterns for locating HTML elements.

CSS Selector

We will mainly focus on the following selectors:

- **Type selector**: selects all the elements with a given node name. We just indicate the name of element, for example `p` for paragraphs;
- **Class selector**: selects all the elements with a given class attribute; for example `.pkg` matches any element with this class;
- **ID selector**: selects all the elements with a given id; for example `#first` matches the element with that id.

A comprehensive list of CSS selectors, can be found here.

CSS selectors can be combined, so for example, `p.classname` will match all the paragraphs with the specified class.
An important CSS *combinator* is the space " ". It is called descendant combinator and it allows to select all the elements inside the previously specified element.

Another way to use CSS selectors is to use the CSS attribute value `[attribute_name="value"]`.

SelectorGadget

In order to find the CSS selector you need (or the XPath), you can simply go on the website

https://rvest.tidyverse.org/articles/selectorgadget.html

with Firefox, then click with the right mouse button on SelectorGadget and then “Bookmark this link” (or “Salva link con nome”).

When you are on the web page you want to scrape, you can click on Bookmark/SelectorGadget, then the gadget helps you finding the right selectors.

If you click on the element that you would like your selector to match, the element will turn green. Everything that is matched by the selectors becomes yellow and into brackets you can read the number of selected elements. To remove the element you can click again and it will turn red. There is also a button to clear the selection. If you hold shift it allows to select the elements inside of other selected ones.

`html_text()` and `html_text2()` function

There are two ways to retrieve text from a element: `html_text()` and `html_text2()`. `html_text2()` is usually what you want, but it is much slower than `html_text()` so for simple applications where performance is important you may want to use `html_text()` instead.

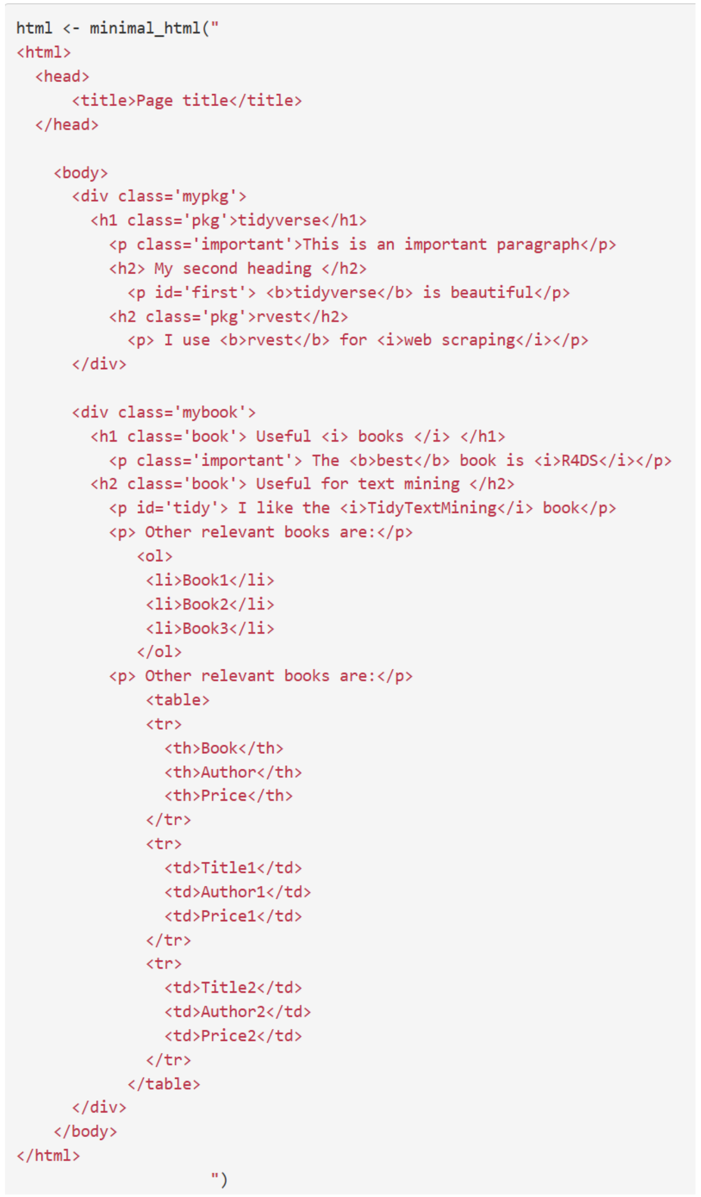
```
html_text(x,…)

html_text2(x,…)
```

`x` A document, node, or node set.

A character vector the same length as x

Simple examples

I used the function `minimal_html` in order to create an HTML document. In the e-learning you will also find the corresponding html file that you can open and inspect (minimal_html_example).



```
html <- minimal_html("
<html>
  <head>
      <title>Page title</title>
  </head>

    <body>
      <div class='mypkg'>
        <h1 class='pkg'>tidyverse</h1>
          <p class='important'>This is an important paragraph</p>
          <h2> My second heading </h2>
            <p id='first'> <b>tidyverse</b> is beautiful</p>
          <h2 class='pkg'>rvest</h2>
            <p> I use <b>rvest</b> for <i>web scraping</i></p>
      </div>

      <div class='mybook'>
        <h1 class='book'> Useful <i> books </i> </h1>
          <p class='important'> The <b>best</b> book is <i>R4DS</i></p>
        <h2 class='book'> Useful for text mining </h2>
          <p id='tidy'> I like the <i>TidyTextMining</i> book</p>
          <p> Other relevant books are:</p>
             <ol>
              <li>Book1</li>
              <li>Book2</li>
              <li>Book3</li>
             </ol>
          <p> Other relevant books are:</p>
              <table>
              <tr>
                <th>Book</th>
                <th>Author</th>
                <th>Price</th>
              </tr>
              <tr>
                <td>Title1</td>
                <td>Author1</td>
                <td>Price1</td>
              </tr>
              <tr>
                <td>Title2</td>
                <td>Author2</td>
                <td>Price2</td>
              </tr>
            </table>
      </div>
    </body>
</html>
                     ")
```

>

Simple examples

```
> html <- read_html("minimal_html_example.html")
> html
{html_document} <html>
[1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset ...
[2] <body>\n<div class="mypkg">\n<h1 class="pkg">tidyverse</h1>\n<p cla ...
```

Extract and display the *head* and the *body*:

```
> html |>
+ html_element(css = "head") |>
+ html_text()
[1] " Page title "
```

```
> html |>
+ html_element(css = "body") |>
+ html_text2()
```

Simple examples

Select all the paragraphs:

```
> html |>
+ html_elements(css = "p") |>
+ html_text2()
[1] "This is an important paragraph" "tidyverse is beautiful"
[3] "I use rvest for web scraping" "The best book is R4DS"
[5] "I like the TidyTextMining book" "Other relevant books are:"
[7] "Other relevant books are:"

> html |>
+ html_elements(xpath = "//p") |>
+ html_text2()
```

Simple examples

Select all the paragraphs from the `div` with `class="mypkg"`;

```
> html |>
+ html_element(css = "div.mypkg") |>
+ html_elements(css = "p") |>
+ html_text2()
[1] "This is an important paragraph" "tidyverse is beautiful"
[3] "I use rvest for web scraping"
```

In alternative you can use the descendant combinators:

```
> html |>
+ html_elements(css = "div.mypkg p") |>
+ html_text2()
[1] "This is an important paragraph" "tidyverse is beautiful"
[3] "I use rvest for web scraping"
```

>

Exercise

Select the first `h2` from the `div` with `class=“mypkg”.`

Exercise for you

With reference to the `html` file, write proper code to do the following:

1. Select all the headers h1;
2. Select the important paragraph from the div with class="mybook";
3. Select the paragraph with id="tidy".

Make all the required selections using both CSS Selector and XPath.