Web Scraping with R (1): Parsing HTML

Alex Sanchez and Francesc Carmona
Genetics Microbiology and Statistics Department
Universitat de Barcelona
October 2022

Introduction

- When scraping HTML we usuallyproceed in two steps:
    - First, inspect content on the Web and examine whether it is attractive for further analyses.
    - Second, import HTML files into R and extract information from them.
- Parsing HTML occurs at both steps
    - *by the browser* to display HTML content nicely, and also
    - *by parsers in R* to construct useful representations of HTML documents in our programming environment.

What is *parsing*

- Parsing involves *breaking down a text into its component parts of speech with an explanation of the form, function, and syntactic relationship of each part*. Wikipedia.

- Reading vs parsing, not just a semantic difference:

  - **reading** relies on functions that *do not care about the formal grammar that underlies HTML*, only recognizing the sequence of symbols included in the HTML file.

  - **parsing** employs programs that understand the special meaning of the mark-up structure reconstructing the HTML hierarchy within some R-specified structure.

Getting data (1): *Reading* an HTML file

- Read from the web using `readlines()` function:

```
##  [1] "<!DOCTYPE HTML PUBLIC \"-//IETF//DTD HTML//EN\">"
##  [2] "<html> <head>"
##  [3] "<title>Collected R wisdoms</title>"
##  [4] "</head>"
##  [5] ""
##  [6] "<body>"
##  [7] "<div id=\"R Inventor\" lang=\"english\" date=\"June/200
##  [8] "  <h1>Robert Gentleman</h1>"
##  [9] "  <p><i>'What we have is nice, but we need something ve
## [10] "  <p><b>Source: </b>Statistical Computing 2003, Reisens
```

readLines() is a *reading* function

- maps every line of the input file to a separate value in a character vector creating a flat representation of the document.
- it is *agnostic* about the different tag elements (name, attribute, values, etc.),
- it produces results that do not reflect the document's internal hierarchy *as implied by the nested tags* in any sensible way.

Getting data (2): *Parsing* an HTML file

- To achieve a useful representation of HTML files, we need to employ a program that:
  - understands the special meaning of the markup structures, and
  - reconstructs the implied hierarchy of an HTML file within some R-specific data structure.

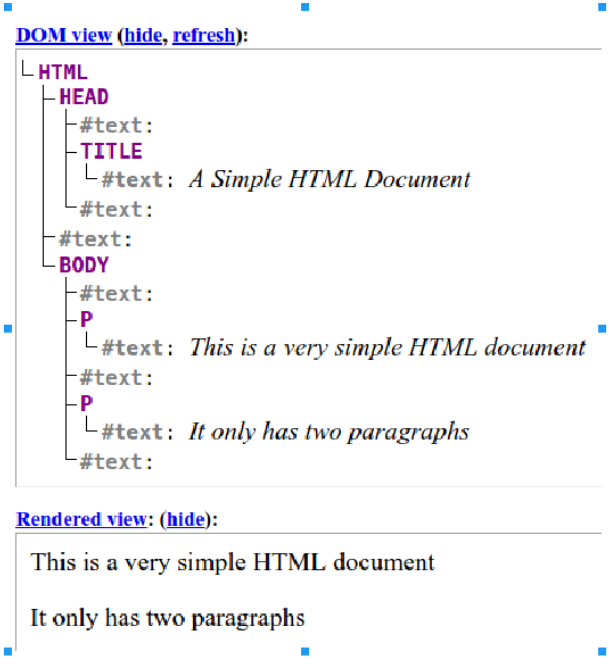
- This representation is also referred to as the *Document Object Model (DOM)*.

- A Document Object Model is a *queryable data object* that can be built from any HTML file and is useful for further processing of document parts.

A distraction: HTML tree structure

- A HTML document can be seen as a hierarchichal collection of tags which contain distinct elements.

```
<html>
<head>
<title>
A Simple HTML Document
</title>
</head>
<body>
<p>This is a very simple HTML
document</p>
<p>It only has two
paragraphs</p>
</body>
</html>
```



Hint: Paste the source code of the *fortunes.html* document in This viewer

DOM-style parsers

- Transformation from HTML code to the DOM is the task of a *DOM-style parser*.
- Parsers belong to a *general class of domain-specific programs that traverse over symbol sequences and reconstruct the semantic structure of the document within a data object of the programming environment*.
- Right now there are two mainstream packages that can be used for parsing HTML code
  - XML package by Duncan Temple and Debbie Nolan,
  - rvest package by Hadley Wickam,
  - and a few others that one can see at CRAN Task View: Web Technologies and Services.

Scrapping tools (I): The `XML` package

- The `XML` package provides an interface to `libxml2` a powerful parsing library written in C.
- The package is designed for two main purposes
  - parsing xml / html content
  - writing xml / html content (*we wonn't cover this*)

What can be achieved with `XML`?

- The `XML` package is useful at 4 major types of tasks:

  1. parsing xml / html content
  2. obtaining descriptive information about parsed contents
  3. navigating the tree structure (ie *accessing its components*)
  4. querying and extracting data from parsed contents

- The `XML` package can be used for both XML and HTML parsing.

Scraping tools (II): The `rvest` package

- `rvest` is an R package written by Hadley Wickam
- It facilitates the process of (i) *acquiring* data from web pages (not "from the web") and (ii) *parsing* the result into R.
- `rvest` is inspired to work with magrittr
- See more information on `rvest` at:
  - rvest package on Github
  - rvest documentation on DataCamp

Basic `rvest` capabilities

- Get the data: Create an html document from a url, a file on disk or a string containing html with `read_html()`.
- Extract elements using `html_element(s)()` and then
- Use `html_text2()` to extract the plain text contents of an HTML element
- Or use `html_attr(s)()` to retrieve the value of a single attribute
- Use `html_table` to read a table from within a page

More `rvest` capabilities

- Parse forms and set values with `html_form()`.
- Extract, modify and submit forms with `html_form()`, `set_values()` and `submit_form()`.
- Detect and repair encoding problems with `guess_encoding()` and `repair_encoding()`. Then pass the correct encoding into `html()` as an argument.

`rvest` Examples

- The easiest way to start with `rvest` is to try its own examples
  - Inspect the vignette "Web scraping 101" at the package website.
  - Go to the package github site and download the file `rvest.Rmd`.
  - Go through it step by step, either on the Rmarkdown or generating the associated R file with the `purl` command.