

Week 7: XML, RSS, and Scraping Dynamic Websites

LSE MY472: Data for Data Scientists

<https://lse-my472.github.io/>

Autumn Term 2024

Ryan Hübert

Introduction

- Last week we discussed some examples of scraping tables or simple unstructured content
- To scrape some websites e.g. with forms or dynamic elements, we need more advanced tools
- This week we will discuss XML, RSS, and XPath, and use `RSelenium` for browser automation

Plan for today

- XML
- RSS
- XPath
- Scraping with (R)Selenium
- Coding

XML

XML

- XML = eXtensible Markup Language
- XML: Store and distribute data
- HTML: Display data
- XML looks a lot like HTML, but is more flexible
 - no predefined tags
 - author can invent tags to structure document

Reference and further information:

https://www.w3schools.com/xml/xml_what_is.asp

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<courses>
  <course>
    <title>Data for Data Scientists</title>
    <code>MY472</code>
    <year>2024</year>
    <term>Autumn</term>
    <description>A course about collecting, processing,
                  and storing data.</description>
  </course>
  <course>
    <title>Computer Programming</title>
    <code>MY470</code>
    <year>2024</year>
    <term>Autumn</term>
    <description>An introduction to programming.</description>
  </course>
</courses>
```

Steps in XML parsing in R

1. Parse an XML file with `read_xml()` in `xml2` package
2. Select elements with `html_elements()`
3. Extract text with `html_text()`

Some XML use cases

- Data storage, e.g. Canadian members of parliament:
<https://www.ourcommons.ca/Members/en/search> (select “Export as XML” at the bottom)
- Scalable Vector Graphics (SVG), e.g. London borough map
<https://upload.wikimedia.org/wikipedia/commons/b/be/BlankMap-LondonBoroughs.svg>
- ePub electronic books, e.g. Project Gutenberg books
- Office documents, e.g. OpenOffice, Microsoft Office
 - Notice difference between .doc and .docx
- RSS web feeds, e.g. [http://onlinelibrary.wiley.com/rss/journal/10.1111/\(ISSN\)1540-5907](http://onlinelibrary.wiley.com/rss/journal/10.1111/(ISSN)1540-5907)

RSS

RSS

- Really Simple Syndication
- Written in XML
- RSS feeds allow users to see new contents from a range of websites quickly and in one place
- RSS aggregators gather and sort RSS feeds
- RSS feed example: [The Guardian RSS feed](#) (more in the guided coding part)

Imaginary RSS feed

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">

<channel>

  <title>MY472 RSS Feed</title>
  <link>https://www.my472.blog/</link>
  <description>Blog about data</description>

  <item>
    <title>Article one</title>
    <link>https://www.my472.blog/article_1.html</link>
    <description>An introduction to data</description>
  </item>

  <item>
    <title>Article two</title>
    <link>https://www.my472.blog/article_2.html</link>
    <description>Some useful R functions</description>
  </item>

</channel>

</rss>
```

Based on: https://www.w3schools.com/xml/xml_rss.asp

XPath

Selecting XML/HTML nodes with XPath

- Last week we discussed CSS selectors to select elements, XPath offers another way
- Both XML and HTML document have a tree structure
- **XPath** (or XML Path Language) is a syntax for defining parts of the tree/document
- Can be used to navigate through elements and attributes
- For most things, you can use either CSS selectors or XPath, but XPath is (probably) more flexible/powerful

An example website

```
<!DOCTYPE html>
<html>
  <head>
    <title>A title</title>
  </head>
  <body>
    <div>
      <h1>Heading of the first division</h1>
      <p>A first paragraph.</p>
      <p>A second paragraph with some <b>formatted</b> text.</p>
      <p class="text-about-web-scraping">A third paragraph now
        containing some text about web scraping ...</p>
    </div>
    <div class="division-two">
      <h1>Heading of the second division</h1>
      <p>Another paragraph with some text.</p>
      <p class="text-about-web-scraping">A last paragraph
        discussing some web scraping ...</p>
    </div>
  </body>
</html>
```

In more detail: Some basic syntax (1/2)

- `/`: Selects from the root node, e.g. `/html/body/div[2]/p[1]`
 - This is how you construct an **absolute XPath**
- `//`: Selects specific nodes from the document, e.g. `//div[2]/p[1]`
 - This is how you construct a **relative XPath**
- `//div/*`: Selects all nodes which are immediate children of a div node
- `//div/p[last()]`: Selects the last paragraph nodes which are children of all div nodes
- `//h1[text()='Heading of the first division']`:
Selects the h1 tag with the text "Heading of the first division" in it

In more detail: Some basic syntax (2/2)

- `//div[@*]`: Selects all division nodes which have any attribute
- `//div[@class]`: Selects all division nodes which have a class attribute
- `//div[@class='division-two']`: Selects all division nodes which have a class attribute with name “division-two”
- `//*[[@class='division-two']`: Selects any node with a class attribute with name “division-two”
- etc.

Reference and full details:

https://www.w3schools.com/xml/xpath_syntax.asp

Comparison: XPath vs CSS selector

Type	CSS selector	XPath
By tag	h1	//h1
By class	.division-two	//*[@class='division-two']
By id	#exemplary-id	//*[@id='exemplary-id']
By tag with class	div.division-two	//div[@class='division-two']
Tag structure	div > p	//div/p
By child number	div > p:nth-of-type(3)	//div/p[3]

We can use either CSS selectors or XPath in rvest

```
read_html(url) %>%  
  html_elements(css = "div > p")
```

```
read_html(url) %>%  
  html_elements(xpath = "//div/p")
```

Comparison: XPath vs CSS selector

For example

```
suppressMessages(library(rvest))  
url <- "https://lse-my472.github.io/week05/data/css2.html"  
  
read_html(url) %>%  
  html_elements(css = "div > p:nth-of-type(3)") %>%  
  html_text()
```

```
## [1] "A third paragraph now containing some text about web scr"
```

```
read_html(url) %>%  
  html_elements(xpath = "//div/p[3]") %>%  
  html_text()
```

```
## [1] "A third paragraph now containing some text about web scr"
```

Scraping with R Selenium

Why?

- Recall 3 scenarios from the last lecture
 - Scenario 1: Data in table format
 - Scenario 2: Data in “unstructured” format
 - Scenario 3: “Hidden” behind web forms
- Many websites fall under scenario 3 because they have:
 - Forms
 - Authentication
 - Dynamic contents
- `RSelenium` is very useful for scenario 3

Selenium

- <https://www.selenium.dev/>
- A technology for browser automation
- General idea: **Browser control** to scrape dynamically rendered web pages
- Originally developed for web testing purposes
- R Selenium: An R binding for Selenium
 - Launch a browser session and all communication will be routed through that browser session

Selenium “WebDrivers”

1. Normal browsers

- Chrome
- Firefox
- etc.

2. Headless browser (will not display browser)

- Allows to set up the browser in a situation where you do not have a visual device (i.e. Crawler on the cloud) or do not need an open browser window
- Previously common headless browser: **phantomJS** (now e.g. just use Chrome or Firefox in headless mode)
- **Selenium in Python** easily allows to run Chrome or Firefox in headless mode

Set up and useful information

- To run RSelenium we are going to use Firefox because it tends to be less buggy than Chrome
- Selenium requires you to install the “Java Development Kit” from Oracle, which you can find here:
<https://www.oracle.com/java/technologies/downloads/>
- In this course, you will learn how to use selenium through R, but it's (probably) better to use python
 - Very similar process, just have to learn slightly different syntax
- In the coding session (next), I will show you how to get RSelenium up and running!

Some key functions (1/2)

Load the RSelenium library

```
library(RSelenium)
```

Create browser instance

```
rD <- rsDriver(browser=c("firefox"))  
driver <- rD$client
```

Navigate to a url

```
driver$navigate("https://www.lse.ac.uk/")
```

Find an element on a webpage

```
some_element <- driver$findElement(using = "xpath",  
                                     value = "...")
```


Some key functions (2/2)

Click on an element

```
some_element$clickElement()
```

Type text into box/element

```
search_box <- driver$findElement(using = "xpath",  
                                   value = "...")  
search_box$sendKeysToElement(list("some text"))
```

Press enter key

```
search_box$sendKeysToElement(list(key = "enter"))
```

Coding

Markdown files

- 01-newspaper-rss.Rmd
- 02-introduction-to-selenium.Rmd
- 03-selenium-lse.Rmd