Week 7: XML, RSS, and Scraping Dynamic Websites

LSE MY472: Data for Data Scientists
https://lse-my472.github.io/

Autumn Term 2024

Ryan Hübert

Slides last updated: 30 September 2024

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** = e**X**tensible **M**arkup **L**anguage
- 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_whatis.asp

XML, Example 1

```
<?xml version="1.0" encoding="UTF-8"?>
<courses>
    <course>
        <title>Data for Data Scientists</title>
        <code>MY472</code>
        <year>2022</year>
        <term>Michaelmas</term>
        <description>A course about collecting, processing, and storing
    </course>
    <course>
        <title>Computer Programming</title>
        <code>MY470</code>
        <year>2022</year>
        <term>Michaelmas</term>
        <description>An introduction to programming.</description>
    </course>
</courses>
```

XML, Example 2 (with DTD)

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>The next assignment will be due on ... </body>
</note>
```

- This XML has a DTD (Document Type Definition)
- DTD is a schema language with relatively limited capabilities, XML Schema has more features
- Reference: <https://www.w3schools.com/xml/xml_dtd.asp, https://en.wikipedia.org/wiki/XML_schema>

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()`

Further XML examples

- Canadian members of parliament: https://www.ourcommons.ca/Members/en/search -> select "Export as XML"
- Scalable Vector Graphics - SVG (graphic): https://upload.wikimedia.org/wikipedia/commons/b/be/BlankMap-LondonBoroughs.svg
- epub (books)
- Office documents (OpenOffice, MS)
- RSS (web feeds -> next topic): http://onlinelibrary.wiley.com/rss/journal/10.1111/(ISSN)1540-5907

RSS

RSS

- **R**eally **S**imple **S**yndication
- 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

Types of XPath

- Absolute XPath: /html/body/div[2]/p[1]
- Relative XPath: //div[2]/p[1]

Our favourite website

```
<!DOCTYPE html>
<html>
    <head>

        <!-- CSS start -->
        <style>
        .text-about-web-scraping {
          color: orange;
        }
        .division-two h1 {
        color: green;
          }
        </style>
        <!-- CSS end -->

        <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 co
```

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

- `/`: Selects from the root node, e.g. /html/body/div[2]/p[1]
- `//`: Selects specific nodes from the document, e.g. //div[2]/p[1]
- `//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

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

Selector type	CSS selector	XPath
By tag	“h1”, “p”	“//h1”, “//p”
By class	“.division-two”	“//*[@class='division-two']”
By id	“#exemplary-id”	“//*[@id='exemplary-id']”
By tag with class or id	“div.division-two”	“//div[@class='division-two']”
Tag strucure (p as a child of div)	“div > p” or “div p”	“//div/p”
Tag strucure (p which is a second child of the div node with class name division-two)	“div.division-two > p:nth-of-type(2)”	“//div[@class='division-two']/p[2]”

Scraping with RSelenium

Why?

- Scenario 3
- Many websites cannot be scraped as easily as in scenarios 1 & 2 for various reasons
  - Forms
  - Authentication
  - Dynamic contents

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
- **RSelenium**: An R binding for Selenium
  - Launch a browser session and all communication will be routed through that browser session

Selenium drivers

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** and **Firefox** in headless mode)
   - ➜ Selenium in Python e.g. easily allows to run **Chrome** or **Firefox** in headless mode

Some key functions (1/2)

➜ **RSelenium package**

```
library("RSelenium")
```

➜ **Create browser instance with**

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

➜ **Navigate to url**

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

➜ **Find element**

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

Some key functions (2/2)

➜ **Click on 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"))
```

A Google search

Let us look at a simple example of `RSelenium` at work

```
library("RSelenium")

rD <- rsDriver(browser=c("firefox"), chromever = NULL)
driver <- rD$client

url <- "https://www.google.com/"
driver$navigate(url)

xpath_of_search_field <- "..."

search_box <- driver$findElement(using = "xpath",
                                 value = xpath_of_search_field)
search_box$sendKeysToElement(list("my472 lse"))

Sys.sleep(1)

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

Coding

Markdown files

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