# MY472 - Week 5: Using Data from the Internet

Pablo Barbera & Akitaka Matsuo

October 30, 2018

# Outline

**Last week:**

- Introductions to the web scraping

**Today: Advanced topics in web-scraping**

- More data types
    - XML and XPath selector
        - XML parsing example
    - JSON primer
- Browser-based scraping with Selenium
- Coding R with `dplyr`

# Other data formats

# Other data formats: XML

- XML = eXtensible Markup Language
- XML is used for distributing data over the Internet.
  - Examples:
    - RSS (web feeds): http://onlinelibrary.wiley.com/rss/journal/10.1111/(ISSN)1540-5907
    - SVG (graphic): https://upload.wikimedia.org/wikipedia/commons/b/be/BlankMap-LondonBoroughs.svg
    - epub (books)
    - Office documents (OpenOffice, MS)
- XML looks a lot like HTML, but more flexible (e.g. basically no preset definitions of tags).

# XML, Example 1 (no schema)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<notes>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
<note>
  <to>Jason</to>
  <from>Kelly</from>
  <heading>Offer</heading>
  <body>You won 10M. Contact us immediately.</body>
</note>
</notes>
```

- This file contains two notes, seems to have common structure for notes but you never know!

# 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>Don't forget me this weekend</body>
</note>
```

- This XML has a DTD (Document Type Definition)
- DTD is one of the XML schematic languages that are used as a validator of data input

# XML Example: Scraping newspaper websites

## RSS feeds

- **R**eally **S**imple **S**yndication, originally developed as a way to regularly check for new content on sites
- Includes list of entries (with some more information) and when they were updated
- Written in XML format (eXtensible Markup Language)
- Example: The Guardian RSS feed
  - We will scrape it in the lab

# Selecting XML/HTML nodes with XPath

- XPath: a syntax for defining parts of an XML document
  - Can be used to navigate through elements and attributes in an XML document.
  - Uses path expressions to navigate in XML document

**For web-scraping**

- Last week, we have seen CSS selector
  - Using selectorGadget and “Inspect”
- XPath can be used another type of selector
- Similar functionality
- Probably a bit more coding involved, but more powerful
- More useful for parsing xml than html
  - xml files are better formatted (while having less attributes)

# Xpath: Basic syntax

From W3schools

| Expression | Description |
|---|---|
| **nodename** | Selects all nodes with the name “nodename” |
| **/** | Selects from the root node |
| **//** | Selects nodes in the document from the current node that match the selection no matter where they are |
| **.** | Selects the current node |
| **..** | Selects the parent of the current node |
| **@** | Selects attributes |

# Xpath: Path examples

| Path Expression | Result |
| --- | --- |
| **bookstore** | Selects all nodes with the name “bookstore” |
| **/bookstore** | Selects the root element bookstoreNote: If the path starts with a slash ( / ) it always represents an absolute |
| **bookstore/book** | Selects all book elements that are children of bookstore |
| **//book** | Selects all book elements no matter where they are in the document |
| **bookstore//book** | Selects all book elements that are descendant of the bookstore element, no matter where they are under the bookstore element |
| **//@lang** | Selects all attributes that are named lang |

# Xpath: Predicates

| Path Expression | Result |
| --- | --- |
| **/bookstore/book[1]** | Selects the first book element that is the child of the bookstore element. |
| **/bookstore/book[last()]** | Selects the last book element that is the child of the bookstore element |
| **/bookstore/book[last()-1]** | Selects the last but one book element that is the child of the bookstore element |
| **/bookstore/book[position()<3]** | Selects the first two book elements that are children of the bookstore element |
| **//title[@lang]** | Selects all the title elements that have an attribute named lang |
| **//title[@lang='en']** | Selects all the title elements that have a “lang” attribute with a value of “en” |
| **/bookstore/book[price>35.00]** | Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00 |
| **/bookstore/book[price>35.00]/title** | Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00 |

# Xpath: Selecting unknown nodes with wildcards

| Wildcard | Description |
| --- | --- |
| * | Matches any element node |
| @* | Matches any attribute node |
| node() | Matches any node of any kind |

## Examples:

| Path Expression | Result |
| --- | --- |
| /bookstore/* | Selects all the child element nodes of the bookstore element |
| //* | Selects all elements in the document |
| //title[@*] | Selects all title elements which have at least one attribute of any kind |

# Comparison: XPath vs CSS selector

| Selector type | CSS selector | XPath |
|---|---|---|
| **By tag** | “h1”, “p” | “//h1”, “//p” |
| **By class** | “.display-name” | "//*[@class='display-name']" |
| **By id** | “#author-name” | "//*[@id='author-name']" |
| **By tag with class or id** | “h1#main-title” | “//h1[@id='main-title']”, |
| **Tag strucure (p as a child of div)** | “div > p” | “//div/p” |
| **Tag strucure (p which is a second child of main div)** | “div#main > p:nth-of-type(2)” | //div[@id=“main”/p[2] |

# Steps in XML (and html) parsing in R

1. Parse an XML file with `read_xml()` in `xml2` package
2. Select nodes with `xml_nodes()` in `rvest` package
3. Extract text using `xml_text()`

Let's see an example from the member list of Canadian parliament members (Link to XML)

# Other Data Formats: JSON

- JSON = **J**ava**S**cript **O**bject **N**otation
- Another format for data exchange in the net
- Lightweight, easy to read, less formatted
- Written with JavaScript object notation, but independent from any language
- Used in many APIs including (See here):
  - Twitter
  - Facebook
  - YouTube

# JSON, Primer

For example: https://petition.parliament.uk/petitions/200205.json

```
suppressMessages(library(jsonlite))
suppressMessages(library(tidyverse))
petition <- fromJSON("https://petition.parliament.uk/petitions/200205.json")
print(petition$data$attributes$signatures_by_constituency %>%
        head(2) %>% toJSON() %>% prettify())
```

```
## [
##     {
##         "name": "Edinburgh East",
##         "ons_code": "S14000022",
##         "mp": "Tommy Sheppard MP",
##         "signature_count": 154
##     },
##     {
##         "name": "Edinburgh North and Leith",
##         "ons_code": "S14000023",
##         "mp": "Deidre Brock MP",
##         "signature_count": 211
```

# Advanced scraping: Selenium

# Why?

- This is about scenario 3
- Many webpages cannot be scraped for many reasons:
    - Form
    - Authentication
    - Dynamic contents

Some form could be available for simple scraping, but often not.

# Selenium

- https://www.seleniumhq.org/
- 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

# Choice of Selenium Drivers

There are two general strategy to run scraper:

1. Headless browsers (without graphical interface)
   - **phantomJS**: headless browser (will not display website)
     - Capabilities: complete forms, write text, click on buttons or area of website etc navigate to new URL…
     - Since it's headless, you can set up the browser in the situation where you don't have visual device (i.e. Crawler on the cloud).
     - To check whether everything works, you need to take screenshot
   - Chrome can be run in headless mode
2. Normal browsers with selenium drivers
   - **Chrome**
   - **Firefox**

# Rselenium demo: Google search

Let's see a really simple example of `RSelenium`:

```
library(RSelenium)
driver<- rsDriver(browser=c("chrome"))
browser <- driver[["client"]]

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

search_field <- browser$findElement(using = "id", "lst-ib")
search_field$sendKeysToElement(list("london school of economics"))
Sys.sleep(5)
search_field$sendKeysToElement(list(key = "enter"))
```

# dplyr

From official page

`dplyr` is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:

- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names.
- `filter()` picks cases based on their values.
- `summarise()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.

# Why `dplyr`?

- Make your code readable. The syntax is designed for being able to read code from left to right (i.e. natural language)
- `%>%` chaining is particularly suitable for html/xml parsing:
  - `html_document %>% html_nodes() %>% html_table()`

# Lab 5

We will do:

- more xml parsing
- Web-scraping using RSelenium

Before the lab, please install Chrome/Firefox on your system.