# Scraping Dynamic Webpages

## Computational Text Analysis +

Theresa Gessler

University of Zurich | http://theresagessler.eu/ | @th_ges

2022-05-11

# Scrolling into infinity

- many webpages are *interactive*: they provide more content as we interact, e.g. scroll, click, ...
  - Facebook, some search engines (e.g. Bing image search)
- automatic reveal of more articles based on scrolling
  - similar with 'load more' buttons also in political sphere

→ Difficult to scrape as page content changes without the URL changing

# Selenium

- General idea: give R browser control to simulate your behavior
  - → scrape dynamically rendered web pages
- Originally developed for web testing purposes
  - automates browsing across platforms

## Procedure

- R launches a browser session
- interaction routed through that browser session
- you extract nodes or download the whole page

## Why?

- complete forms
- write text
- click on buttons or area of website
- scroll
- navigate to new URLs

# Selenium

- a whole set of projects
  - Selenium WebDriver: drive a browser
  - SeleniumGrid: running tests on multiple servers
  - Selenium IDE: Chrome and Firefox extension that allows recording and playing back tests
  - Selenium Remote Control: control web drivers on other computers

## Packages

- `RSelenium`: R bindings for Selenium web driver
- `wdman`: manages browser binary files needed for running selenium

# How Selenium works

- starting server and browser session
- navigating to page
- finding elements
- sending 'events' to elements
- getting the source code and extracting information

# How Selenium works

- **starting server and browser session**
- navigating to page
- finding elements
- sending 'events' to elements
- getting the source code and extracting information

# How Selenium works

## Starting server and browser session

- Selenium Server: standalone java program that allows to use a range of different browsers
- if you use `wdman`, R will download binaries for the browser you selected
- alternatively, you can download binary and reference it (*not recommended*)

```
library(RSelenium)
library(wdman)
```

- downloading browser binary with `wdman`
  - if you do not specify browser, it will download all of them
  - name the environment to which you assign the browser as you want - I chose `server`

```
# download just phantomjs
server <- phantomjs()
# download all browsers
server <- selenium()
```

# How Selenium works

## Starting server and browser session

- by assigning `remoteDriver()` to an object, you connect to browser session
  - standard: Firefox
  - name the object what you want but you will type this name a lot!
  - many people use `remDr`
  - you may need to change the port

```
# connect to browser
remDr <- remoteDriver(browserName = "phantomjs", port=4448L)
```

→ **Everything we did so far**:

```
library(RSelenium)
server <- phantomjs()
remDr <- remoteDriver(browserName = "phantomjs", port=4448L)
```

# How Selenium works

## Starting server and browser session

...so what is this *phantomjs* we are using?

- PhantomJS is a headless browser
  - browse webpages without an actual window: no graphical interface
- widely used with `RSelenium` due to its stability

## Downsides

- no longer under active development
- taking screenshots to check progress
- some features are blocked to phantomjs users - you may need to change your user agent
- and: far less fun when learning `RSelenium`

→ We will try using both a normal browser and PhantomJS today

# How Selenium works

- starting server and browser session
- **navigating to page**
- finding elements
- sending 'events' to elements
- getting the source code and extracting information

# How Selenium works

## Navigating to page

- simplest way to navigate is to use URL of page
- however, we can also refresh, navigate forward and backward

## Syntax

- commands are called *methods*
- attached to `remoteDriver` object with $ sign

```
remDr$Navigate("http://somewhere.com")

# other commands
remDr$goBack()
remDr$goForward()
remDr$refresh()
remDr$getCurrentUrl()
```

# How Selenium works

- starting server and browser session
- navigating to page
- **finding elements**
- sending 'events' to elements
- getting the source code and extracting information

# How Selenium works

## Finding elements

- if you want to use elements -e.g. write into a textbox - you need to find & assign them to an object
  - all commands to this element will be performed using *that environment*, not the `remoteDriver` environment
    - → so again, name it well!
  - `webElem` is a common name

```
webElem <- remDr$findElement(using = 'class name', "gbqfif")
webElem <- remDr$findElement("partial link text", "download R")
```

- objects can be found
  - by css selector, x-path, id or class
  - by name
  - by (partial) link text (anchor elements / links)

# How Selenium works

## Using ids as CSS selectors

- **ids** are unique to individual HTML tags, rather than shared (like a class)
- usually not interesting: typically, we are interested in multiple elements
- however, crucial for clicking buttons, finding text fields etc
  - → much more useful with `RSelenium` than for `rvest`-based scraping

## Usage

- in HTML code
  - `<h2 id="C4">Chapter 4</h2>`, also within page: `<a href="html_demo.html#C4">Jump to Chapter 4</a>`
- as CSS selector
  - `css="#id"`
- in `RSelenium`
  - use as CSS selector or directly
  - `remDr$findElement(using = 'id', "id")`

# How Selenium works

- starting server and browser session
- navigating to page
- finding elements
- **sending 'events' to elements**
- getting the source code and extracting information

# How Selenium works

## Sending 'events' to elements

- 'sending keys': typing
  - including sending 'enter' key: \uE007
  - see list of keys
- clicking on elements

```
# send text
webElem$sendKeysToElement(list("R Cran"))
# send text & enter
webElem$sendKeysToElement(list("R Cran", "enter"))
#click
webElem$clickElement()
```

# How Selenium works

## Sending 'events' to elements - application

- combinations of finding & sending keys can be used to scroll:
  - sending downward arrows
  - going to the end of HTML body

```
# scrolling a bit
webElem$sendKeysToElement(list(key = "down_arrow"))
# scrolling to end of page
webElem <- remDr$findElement("css", "body")
webElem$sendKeysToElement(list(key = "end"))
```

# How Selenium works

- starting server and browser session
- navigating to page
- finding elements
- sending 'events' to elements
- **getting the source code and extracting information**

# How Selenium works

## Source code and extracting information

- **screenshots**
  - central to working with headless browsers
- **getting source code**
  - preferable option: download HTML page source and save it for extraction
  - remedies the instability of `RSelenium`
- **directly extracting** elements from `RSelenium` session
  - `findElements()` / `findElement()` for selection of nodes
  - `getElementText()` for extracting text from individual nodes

```
# screen shots
remDr$screenshot(display = TRUE)
# getting source code
remDr$getPageSource()
# directly extracting elements
webElem <- remDr$findElements(using = "class", value="results")
values <- webElem[[1]]$getElementText()
```

So now, let us try step by step...