Exercise 4: Scraping

Philipp Kling

18.06.2020

Contents

```r
library(knitr)

## Global options
options(max.print="75")
opts_chunk$set(echo=FALSE,
                 cache=FALSE,
               prompt=FALSE,
               tidy=TRUE,
               comment=NA,
               message=FALSE,
               warning=FALSE)
opts_knit$set(width=75)
rm(list = ls())
# load("/home/philipp/Documents/data/vignettes/data/vignettes_RDATA.Rda")
```

Preparation

Installation of required packages

```r
# toInstall <- c('httr','jsonlite', 'yaml','rtweet','xml2','rvest')
# 'devtools','xml2','rvest' install.packages(pkgs = toInstall)
library(httr)
library(jsonlite)
library(RSelenium)
library(rvest)
library(xml2)
library(yaml)
library(rtweet)
```

Useful ressources

- Selector Gadget: https://selectorgadget.com/
- R Packages: https://ropensci.org/packages/
- Maybe: Google Chrome Developer Selector (Inspect, Copy Xpath)
- *xml2*-package to retrieve data from XML-files
- *rvest*-package to retrieve data from parsed HTML-files
- *RSelenium*-package to interact with dynamic websites.

Scraping The Guardian

Goal: retrieving every article headline from The Guardian about the UK (UK Section of the website)

Procedure

- Install Selector Gadget
- Visit the url and inspect it: Multiple pages with headlines and links to articles; no pageination end
- Download website and parse it to an XML file
- Selector Gadget and select one headline
- Copy XPath and use *rvest* to extract nodes
- Inspect result and use *rvest* and *xml2* to extract headlines and links to articles.
- Iterate through pages and repeat for every page
- Catch error if end is reached (page not existent)

Inspection

```r
url <- "https://www.theguardian.com/uk-news/all"
# browseURL(url)
```

Download static website (HTML) and parse it

```r
url_parsed <- xml2::read_html(url)
```

Copy XPath from SelectorGadget and use *rvest* to extract nodes

```r
nodes <- rvest::html_nodes(url_parsed, xpath = "//*[contains(concat( \" \", @class, \" \" ), concat( \"
```

Extract Headlines and Links

- *rvest::html_name* gets the categories of the nodes (similar to *xml2::xml_name*)
- *rvest::html_text* extracts the text included in the nodes (similar to *xml2::xml_text*)
- *rvest::html_attr* extracts the name of the attributes (similar to *xml2::xml_attr*)

Headlines

```r
# Headlines
headline_nodes <- nodes[rvest::html_name(nodes) == "span"]
headlines <- rvest::html_text(headline_nodes)
```

Links

```r
# Links
link_nodes <- nodes[rvest::html_name(nodes) == "a"]  # Keep only those entries who are links
links <- xml2::xml_attr(link_nodes, "href")
```

Paste them together

```r
page_df <- cbind.data.frame(headline = headlines, link = links)
```

Define a function ‘get__article__urls__from__page’ that does all the steps above

```r
get_article_urls_from_page <- function(page_nbr) {
    url <- paste("https://www.theguardian.com/uk-news?page=", page_nbr, sep = "")

    # Save the file for reproducibility
    download.file(url, destfile = paste("../data/ex4/scraping-guardian-uknews-",
        Sys.Date(), "-page-", page_nbr, ".html", sep = ""))

    # Download and parse
    url_parsed <- xml2::read_html(url)

    # Extract nodes
    nodes <- rvest::html_nodes(url_parsed, xpath = "//*[contains(concat( \" \", @class, \" \" ), concat

    # Narrow down headlines
    headline_nodes <- nodes[rvest::html_name(nodes) == "span"]
    headlines <- rvest::html_text(headline_nodes)

    # Narrow down links
    link_nodes <- nodes[rvest::html_name(nodes) == "a"]  # Keep only those entries who are links
    links <- xml2::xml_attr(link_nodes, "href")

    # Paste them together
    page_df <- cbind.data.frame(headline = headlines, link = links)

    # Return the result
    return(page_df)
}
```

Outline: Iterate through pages

- Note: this seems possible to a maximum of page 1900 which is about October, 11, 2018 (Date: 2020-04-22). Apparently there are 506,781 articles, which should amount to 25339 pages (we can, thus, get about 7% of the articles via this method).
- Note: page = 1 represents the "all"-page previously scraped.

```
page_df <- data.frame()
for (i in 1:3) {
    to_add <- get_article_urls_from_page(i)
    # Add them to the existing dataframe
    page_df <- rbind(page_df, to_add)
}
save(page_df, file = paste("../data/ex4/theguardian-uknews-p1-p3-", Sys.Date(),
    ".Rda", sep = ""))
```

Outline: Scrape until a specific date is reached.

It seems unpractical to scrape pages based on their number. Instead it would be more helpful if we could stop at a specific date. To do so we:

- define a variable which enables and breaks the scraping: 'scrape'
- use the regular expression developed in exercise 2 to extract the dates from the URLs.
- set the 'scrape'-variable to FALSE if the median of the retrieved dates is smaller than the specified stop date.

```
get_uknews <- function(stop_date = Sys.Date() - 7) {
    scrape <- TRUE
    i <- 1
    page_df <- data.frame()  # Empty dataframe that we append to.


    while (scrape) {
        # Scrape the content of the current page
        to_add <- get_article_urls_from_page(i)

        # Create date variable
        to_add$date <- lubridate::as_date(stringr::str_extract(string = to_add$link,
            pattern = "[:digit:]{4}\\/[:lower:]{3}\\/[:digit:]{2}"))

        # Add them to the existing dataframe
        page_df <- rbind(page_df, to_add)

        # Increment the iterator for the pages
        i <- i + 1

        # Finally: Check if the recently added data was predominantly from before
        # the desired date. If so: set scrape to be FALSE and end the while loop.
        if (median(to_add$date) < stop_date) {
            scrape <- FALSE
        }
    }
    # Do a final cleanup: remove every entry that is from before the desired
    # date.
    page_df <- page_df[page_df$date >= stop_date, ]
    return(page_df)
}
stop <- Sys.Date() - 2
res_page_df <- get_uknews(stop_date = stop)
table(res_page_df$date)
save(page_df, file = paste("../data/ex4/theguardian-uknews-", stop, "--", Sys.Date(),
    ".Rda", sep = ""))
```

Scrape dynamic pages

When accessing websites, we often need to interact with websites first to be able to retrieve the desired information. Examples are:

- Websites that require a log in to access the desired information.
- Content that loads only until we scroll down far enough.
- Systematic searches that require interactions with search parameters

Example here: use Selenium, to specify a search on IMDB that looks for movies containing the word "action".

Preparation

In order to scrape content from dynamic websites, we require Selenium WebDriver. Getting the Selenium WebDriver to work is not trivial. Below we provide some tutorials that worked for us. Some basic intuition:

- **Selenium WebDriver** provides a framework to open a virtual browser and interact with it in an automated way. This is needed if we first need to interact with a website before being able to access the desired information (e.g. dynamic websites).
- **RSelenium** is then the interface for Selenium to work in R.
- **Docker** is a program that provides a virtual environment that runs an application This makes it handy when sharing applications between different computers and operating systems. Docker is not necessary, but very helpful and we advice to use it.

Optionally, we might require viewer programs to make the Selenium WebDriver visible to us.

Windows

- Install Docker: https://www.docker.com/products/docker-desktop
- Install RSelenium: https://github.com/ropensci/RSelenium
- Start a docker container:
  - Start docker
  - Open terminal/ switch to the Terminal tab in RStudio
  - Type: docker run -d -p 4445:4444 selenium/standalone-chrome
- Load Rselenium and get a remote driver:

```
library(RSelenium)
remDr <- RSelenium::remoteDriver(remoteServerAddr = "localhost", port = 4445L,
    browserName = "chrome")
remDr$open()
```

Mac

- I am unable to test whether the following tutorial makes sense, but you might have a look here: https://www.raynergobran.com/2017/01/rselenium-mac-update/

Linux

- Installation Rselenium-Docker worked with this Guide on Ubuntu (2020-04-16):
  - https://rpubs.com/johndharrison/RSelenium-Docker
- Installation of Docker (https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-18-
  - sudo apt update
  - sudo apt install apt-transport-https ca-certificates curl software-properties-common
  - curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
  - sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
  - sudo apt update
  - apt-cache policy docker-ce
  - sudo apt install docker-ce
  - Test it with: sudo docker run hello-world
- Install a VNC Viewer
  - sudo apt-get update
  - sudo apt-get install vinagre
- Pull an image with docker:
  - sudo docker pull selenium/standalone-firefox-debug
- Start container
  - sudo docker run -d -p 4445:4444 -p 5901:5900 selenium/standalone-firefox-debug
- Test it:

```r
remDr <- RSelenium::remoteDriver(port = 4445L)
remDr$open(silent = TRUE)
remDr$navigate("http://www.google.com/ncr")
remDr$getTitle()
# close connection
remDr$closeServer()
```

- View this website in Vinagre:
  - Start Vinagre in ubuntu programs
  - Select Protocoll = VNC; Host = 127.0.0.1:5901 Options: View Only
  - The password is 'secret'

Load required packages

```r
library(RSelenium)
library(xml2)
library(rvest)
```

Scrolling the New York Times

Navigate to the New York Times homepage.

```r
remDr <- RSelenium::remoteDriver(port = 4445L)
remDr$open(silent = TRUE)
```

```r
remDr$navigate("https://www.nytimes.com/")
output <- remDr$getPageSource(header = TRUE)  # Download the page at this time.
write(output[[1]], file = "../data/ex4/nytimes1.html")
```

Use the CSS File and its contents to scroll to the end of the current section. Do this several times

```r
webElem <- remDr$findElement("css", "body")
for (i in 1:10) {
    webElem$sendKeysToElement(list(key = "end"))
    Sys.sleep(4)
}
output <- remDr$getPageSource(header = TRUE)  # Download the page at this later time
write(output[[1]], file = "../data/ex4/nytimes2.html")

# close connection
remDr$closeServer()
```

Compare both downloads

- Both have content at the top of the website, but only the second one contains information about the bottom of the website.

```r
c_bs <- xml2::read_html("../data/ex4/nytimes1.html", encoding = "utf8")
c_as <- xml2::read_html("../data/ex4/nytimes2.html", encoding = "utf8")

# Language element: 'ENGLISH' (or anything on top of the page)
english_bs <- rvest::html_nodes(c_bs, xpath = "//*[@id=\"app\"]/div[2]/div/header/section[1]/div[3]/ul/
rvest::html_text(english_bs)
```

```
[1] "English"
```

```r
english_as <- rvest::html_nodes(c_as, xpath = "//*[@id=\"app\"]/div[2]/div/header/section[1]/div[3]/ul/
rvest::html_text(english_as)
```

```
[1] "English"
```

```r
# Get something in the bottom of the page (e.g. 'Today's Opinion' link)
opinion_bs <- rvest::html_nodes(c_bs, xpath = "//*[@id=\"site-index\"]/div/div[2]/div/section[2]/ul/li[
rvest::html_text(opinion_bs)
```

```
character(0)
```

```r
opinion_as <- rvest::html_nodes(c_as, xpath = "//*[@id=\"site-index\"]/div/div[2]/div/section[2]/ul/li[
rvest::html_text(opinion_as)
```

```
[1] "Today's Opinion"
```

Scraping IMDB

Navigate to the search section of the IMDB site.

```r
remDr <- RSelenium::remoteDriver(port = 4445L)
remDr$open(silent = TRUE)
remDr$navigate("https://www.imdb.com/search/")
```

Find the field where we can input the search. Put “action” into the field.

```
xpath <- "//*[(@id = \"query\")]"
searchElem <- remDr$findElement(using = "xpath", value = xpath)
writeFrom <- searchElem$sendKeysToElement(list("action"))  # enter action into the field
```

Extract the search button and then ‘click’ it.

```
xpath <- "//*[contains(concat( \" \", @class, \" \" ), concat( \" \", \"btn-raised--primary\", \" \" ))
searchElem <- remDr$findElement(using = "xpath", value = xpath)
resultsPage <- searchElem$clickElement()
```

Download the results of the search.

```
output <- remDr$getPageSource(header = TRUE)
write(output[[1]], file = "../data/ex4/imdb-datafilms.html")

# close connection
remDr$closeServer()
```

Import the results into R and show the first couple of hits.

```
# parse index table
content <- xml2::read_html("../data/ex4/imdb-datafilms.html", encoding = "utf8")
items <- rvest::html_nodes(content, xpath = ".mode-detail")
items <- rvest::html_nodes(content, xpath = "//*[contains(concat( \" \", @class, \" \" ), concat( \" \"
# children <- rvest::html_children(items)
text <- rvest::html_text(items)
```