For one of my projects I needed to download text from multiple websites. In this case, I used rvest and dplyr. Accessing the information you want can be relatively easy if the sources come from the same websites, but pretty tedious when the websites are heterogenous. The reason is how the content is kept in the HTML of the website (Disclaimer: I am not an expert *at all* on HTML or anything website related). Assume that you want to extract the title, author information, publish date, and of course the main article text. You can identify the location of that information via Cascading Style Sheets (CSS) or XML Path Language (XPath). As soon as you have the CSS or XPath locations, you can access it in R. The following text will walk you through an example and provide the relevant code.

**Where is the information I need?**

Assume you want to get the relevant information from an [article](https://www.theguardian.com/environment/2015/jan/08/mayors-failure-clean-up-londons-air-pollution-risks-childrens-health) from [The Guardian](https://www.theguardian.com/). Open this website in your browser. I recommend using Google Chrome because I will use a handy tool called [SelectorGadget](http://selectorgadget.com/" \t "_blank)which allows you to easily find the CSS or XPath information via point and click. You know exactly what you want on that website, i.e. title, author information, main text and publishing date. But how to get that into R? We’ll start by loading the HTML page into R using rvest.

library(rvest)

url <- "https://www.theguardian.com/environment/2015/jan/08/mayors-failure-clean-up-londons-air-pollution-risks-childrens-health"

# Read the HTML document using try to handle 404 errors

try(html\_document <- read\_html(url))

print(html\_document) # does not provide us the information we want. It just shows the HTML code.

## {xml\_document}

##

## [1] \n## [2]

Mayor's failure to clean up London's air pollution 'risks children's health'

 Select the SelectorGadget Addon in Google Chrome, then click on the headline. Make sure that only the content that you want to extract is highlighted in green or yellow, by clicking on the respective parts. At the end, you should see the HTML (CSS) code to access that information at the bottom. In our specific example, it should look like .content\_\_headline.

 Right click an empty part on the website and click **View page source**. Then search for headline text with Ctrl+F. This, most often, while provide you with multiple hits. Sometimes this is good if any of the other ways to access the information won’t work, since it provides you with alternatives. In our specific example, one outcome is the same as with the XPath method. The others, e.g. "headline":"Mayor's failure to clean up London's air pollution 'risks children's health'" are difficult to access, however.

**How can I retrieve the information I need?**

The following code shows you how to access the information found with the above means in R. First, with the XPath method, then with the CSS method. At the end, we will construct a data frame with that information.

library(dplyr)

# Specify the xpath content for the headline in title\_xpath

# Note that SelectorGadget provides you with: //\*[contains(concat( " ", @class, " " ), concat( " ", "content\_\_headline", " " ))], which is equivalent

title\_xpath <- "//h1[contains(@class, 'content\_\_headline')]"

title\_text <- html\_document %>%

html\_node(xpath = title\_xpath) # Only provides the node.

# In order to get the information we want, we need html\_text, which extracts attributes, text and tag name from html

title\_text <- title\_text %>%

html\_text(trim = T) # Stores title in title\_text

# Access author information (CSS)

author\_css <- ".tone-colour span" # Using SelectorGadget ('.byline span' does also work)

author\_text <- html\_document %>%

html\_node(css = author\_css) %>%

html\_text(trim = T) # Stores author in author\_text

# Access article text information (XPath)

body\_xpath <- "//div[contains(@class, 'content\_\_article-body')]//p" # '.js-article\_\_body > p' is also possible, but needs css option in html\_nodes

# The above location can be found when searching for the first two words of the article in the source code (or when inspecting the first to lines of the article).

# This provides you with the location information

body\_text <- html\_document %>%

html\_nodes(xpath = body\_xpath) %>%

html\_text(trim = T) %>%

paste0(collapse = "\n")

# Access publishing date information (XPath)

date\_xpath <- "//time" # '.content\_\_dateline-wpd--modified' does not work for some reason, although it is the output of SelectorGadget.

# In such a case just try to look for alternatives witht he other methods outlined above

library(lubridate) # to handle date information (important for later analysis including time)

date\_text <- html\_document %>%

html\_node(xpath = date\_xpath) %>%

html\_attr(name = "datetime") %>% # accesses the attribute information datetime in //time (different from html\_text above)

as.Date() %>%

parse\_date\_time(., "ymd", tz = "UTC")

# Store all information in a data frame called article

article <- data.frame(

url = url,

date = date\_text,

title = title\_text,

author = author\_text,

body = body\_text

)

print(as\_tibble(article))

## # A tibble: 1 x 5

## url

##

## 1 https://www.theguardian.com/environment/2015/jan/08/mayors-failure-clean-up

## # ... with 4 more variables: date , title , author ,

## # body

The next step would be to wrap this code in a function, in order to be able to run it for multiple The Guardian articles.

# Define the function

scrape\_guardian\_article <- function(url) {

try(html\_document <- read\_html(url))

title\_xpath <- "//h1[contains(@class, 'content\_\_headline')]"

title\_text <- html\_document %>%

html\_node(xpath = title\_xpath)

title\_text <- title\_text %>%

html\_text(trim = T)

author\_css <- ".tone-colour span"

author\_text <- html\_document %>%

html\_node(css = author\_css) %>%

html\_text(trim = T)

body\_xpath <- "//div[contains(@class, 'content\_\_article-body')]//p"

body\_text <- html\_document %>%

html\_nodes(xpath = body\_xpath) %>%

html\_text(trim = T) %>%

paste0(collapse = "\n")

date\_xpath <- "//time"

library(lubridate)

date\_text <- html\_document %>%

html\_node(xpath = date\_xpath) %>%

html\_attr(name = "datetime") %>%

as.Date() %>%

parse\_date\_time(., "ymd", tz = "UTC")

article <- data.frame(

url = url,

date = date\_text,

title = title\_text,

author = author\_text,

body = body\_text

)

return(article)

}

# Run the function for multiple links

articles <- data.frame()

links <- c("https://www.theguardian.com/environment/2015/jan/08/mayors-failure-clean-up-londons-air-pollution-risks-childrens-health", "https://www.theguardian.com/world/2016/dec/07/marshall-islands-natives-return-mass-exodus-climate-change", "https://www.theguardian.com/environment/2016/dec/14/queenslands-largest-solar-farm-plugs-into-the-grid-a-month-early")

for (i in 1:length(links)) { # Iterate over number of links

cat("Downloading", i, "of", length(links), "URL:", links[i], "\n")

article <- scrape\_guardian\_article(links[i]) # Use downloder function specified above for link[i]

articles <- rbind(articles, article) # Append new article to old

}

## Downloading 1 of 3 URL: https://www.theguardian.com/environment/2015/jan/08/mayors-failure-clean-up-londons-air-pollution-risks-childrens-health

## Downloading 2 of 3 URL: https://www.theguardian.com/world/2016/dec/07/marshall-islands-natives-return-mass-exodus-climate-change

## Downloading 3 of 3 URL: https://www.theguardian.com/environment/2016/dec/14/queenslands-largest-solar-farm-plugs-into-the-grid-a-month-early

print(as\_tibble(articles))

## # A tibble: 3 x 5

## url

##

## 1 https://www.theguardian.com/environment/2015/jan/08/mayors-failure-clean-up

## 2 https://www.theguardian.com/world/2016/dec/07/marshall-islands-natives-retu

## 3 https://www.theguardian.com/environment/2016/dec/14/queenslands-largest-sol

## # ... with 4 more variables: date , title , author ,

## # body

You can also modify the function to use it with lapply. To do that, use the following code modifications

# Change the return code in the functin defined above to:

articles <- rbind(articles, article)

# Run the function over vector of links

text\_df <- as.data.frame(lapply(links, scrape\_guardian\_article))

**How can I adapt this to other websites?**

Unfortunately, the above code won’t work for every website, probably not even for all The Guardian websites, because these websites are built differently. Main text will be stored just in p sometimes, whereas you will be more elaborated CSS path specifications on others. Depending on the number of different websites you want to scrape, it can be pretty tedious to write a function with the adequate CSS or XPath specifiers for everyone. However, as of now I do not know of a better way to do this (I would be grateful for tips and tricks, though). I tried using RSelenium, which sets up a server, navigates to the respective website and clicks or copies whatever you specify. However, in this case the algorithm cannot know perfectly what kind of information you want. Maybe there are machine learning methods that allow an algorithm to learn based on text on how to best identify the main text of a website, its title, etc. This sounds like a really interesting method. However, I am not yet aware of any such approaches.