Web Scraping Workshop

Jihed Ncib

3 October 2023

Introduction

Web scraping is an important tool for data collection. In this workshop, I will give a general introduction on how to scrape and crawl websites and how to use loops. However, I will first go over some important aspects of HTML and the structure of web pages as well as some important (and maybe familiar for some of you) functions that will be required for this tutorial. Let's first load our packages

```
library(tidyverse)
library(rvest)
```

Important Functions

data.frame() creates data frames, tightly coupled collections of variables which share many of the properties of matrices and of lists, used as the fundamental data structure by most of R's modeling software.
rbind() takes a sequence of vector, matrix or data-frame arguments and combine by columns or rows, respectively.
paste() / paste0() concatenate vectors after converting to character.
str_sub() takes a portion of a string and str_remove() removes a portion of a string.

```
str_sub("Jihed", 1, 2)
```

```
## [1] "Ji"
```

```
str_remove("Jihed", "J")
```

```
## [1] "ihed"
```

```
paste("Jihed", "Ncib")
```

```
## [1] "Jihed Ncib"
```

```
paste("Jihed", "Ncib", sep = "_")
```

```
## [1] "Jihed_Ncib"
```

```r
paste0("Jihed", "Ncib")
```

```
## [1] "JihedNcib"
```

```r
month = "Nov"
year = "2021"
paste0(month, "-", year)
```

```
## [1] "Nov-2021"
```

FOR Loop

A for-loop is a control flow statement for specifying iteration, which allows code to be executed repeatedly.

```r
print("Monday")
```

```
## [1] "Monday"
```

```r
print("Tuesday")
```

```
## [1] "Tuesday"
```

```r
print("Wednesday")
```

```
## [1] "Wednesday"
```

```r
print("Thursday")
```

```
## [1] "Thursday"
```

```r
print("Friday")
```

```
## [1] "Friday"
```

```r
print("Saturday")
```

```
## [1] "Saturday"
```

```r
print("Sunday")
```

```
## [1] "Sunday"
```

```r
days = c("Monday", "Tuesday", "Wedndesday", "Thursday", "Friday", "Saturday", "Sunday")
for (i in 1:7) {
  print(days[i])
}
```

```
## [1] "Monday"
## [1] "Tuesday"
## [1] "Wedndesday"
## [1] "Thursday"
## [1] "Friday"
## [1] "Saturday"
## [1] "Sunday"
```

HTML: The front-end syntax

Most, if not all, websites use some form of HTML. It is the default syntax language to design webpages along with CSS to edit the layout and Javascript to make dynamic pages.
Webpages are based on HTML elements. These are nodes written using a tag in the HTML document. "html, head, title, body, h2, p" are all elements because they are represented by tags.
Tags (or elements) are used to select which part of the webpage to scrape.
Examples: view-source:https://jihedncib.net/
To start scraping, you first need to store the HTML code of the webpage in a variable. We do that by specifying our target url using read_html()

```
read_html("https://ccss.ku.edu.tr/people/")
```

```
## {html_document}
## <html lang="en-US" xmlns:fb="https://www.facebook.com/2008/fbml" xmlns:addthis="https://www.addthis.
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
## [2] <body class="page-template-default page page-id-451 masthead-fixed full-w ...
```

Then we look for the HTML tag or element that we want to select. We use html_nodes() to select the part of the webpage that we want to scrape. Let's try h4 to scrape names and titles of faculty members at CCSS.

```
read_html("https://ccss.ku.edu.tr/people/") %>% html_nodes("h4")
```

```
## {xml_nodeset (7)}
## [1] <h4>Assoc. Prof. Erdem Yörük</h4>
## [2] <h4>Asst. Prof. Merih Angın</h4>
## [3] <h4>Assoc. Prof. Ergin Bulut</h4>
## [4] <h4>Assoc. Prof. Gizem Ergin</h4>
## [5] <h4>Asst. Prof. Güneş Ertan</h4>
## [6] <h4>Assoc. Prof. Mustafa Erdem Kabadayı</h4>
## [7] <h4>Sinemis Temel (PhD Candidate)</h4>
```

Different types of elements can be scraped off a webpage. They can be text elements, tables, links, etc. To scrape text, the function html_text() is used.

```
read_html("https://ccss.ku.edu.tr/people/") %>% html_nodes("h4") %>% html_text()
```

```
## [1] "Assoc. Prof. Erdem Yörük"            "Asst. Prof. Merih Angın"
## [3] "Assoc. Prof. Ergin Bulut"            "Assoc. Prof. Gizem Ergin"
## [5] "Asst. Prof. Güneş Ertan"             "Assoc. Prof. Mustafa Erdem Kabadayı"
## [7] "Sinemis Temel (PhD Candidate)"
```

Stringr functions can be useful here to clean the output data: Let's try removing the titles from the names that we just scraped.

```
read_html("https://ccss.ku.edu.tr/people/") %>% html_nodes("h4") %>% html_text() %>% str_remove_all(., "
```

```
## [1] "Erdem Yörük"                    "Merih Angın"
## [3] "Ergin Bulut"                    "Gizem Ergin"
## [5] "Güneş Ertan"                    "Mustafa Erdem Kabadayı"
## [7] "Sinemis Temel (PhD Candidate)"
```

What if we would like to scrape the links? They can be very useful, especially if you're scraping multiple pages.
The "a" tag and the "href" attribute are used to insert hyperlinks in HTML.

```
read_html("https://ccss.ku.edu.tr/people/") %>% html_nodes("a") %>% html_attr("href")
```

Notice how this retrieves ALL the links on the webpage. What if I need only certain ones? In this case, we're interested in faculty members' university web pages.
HTML IDs and classes are used to identify different sections and elements on a webpage. IDs should be preceded by # and classes by .
Let's use selector gadget to only target the desired content.

```
read_html("https://ccss.ku.edu.tr/people/") %>% html_nodes(".award-content a") %>% html_attr("href")
```

With the data that we currently have, we create a dataframe containing two columns: names and links to websites:

```
name = read_html("https://ccss.ku.edu.tr/people/") %>% html_nodes("h4") %>% html_text()
website = read_html("https://ccss.ku.edu.tr/people/") %>% html_nodes(".award-content a") %>% html_attr("
ccss_faculty = as.data.frame(name, website)
ccss_faculty
```

Selector Gadget

It is very useful to have a broad undertsanding of how HTML tags and elements work. But there's a tool that we can use to select different elements of a webpage without having to go through all the code: https://chrome.google.com/webstore/detail/selectorgadget/mhjhnkcfbdhnjickkkdbjoemdmbfginb
Let's see how this works.
Example 2: News headlines from Hurriyet https://www.hurriyet.com.tr/dunya/

```
news_titles = read_html("https://www.hurriyet.com.tr/dunya/") %>% html_nodes("#content h2") %>% html_tex
news_titles[1:10]
```

```
##  [1] "Azerbaycan duyurdu:  İşte Karabağ'daki Ermeni nüfusun yeniden entegrasyon planının detayları"
##  [2] "Kaşıntı için gittiği hastanede kötü haberi öğrendi! İşte gizli hastalığı ve ele veren belirtisi
##  [3] "İran onayladı: 2 bin mahkuma af"
##  [4] "İspanyol yazar herkesi şaşırttı, hayattayken toprağa verildi! 'Üzerime toprak attıklarında bir
```

```
##  [5] "114 kişinin öldüğü düğünün yeni görüntüleri yayınlandı"
##  [6] "Trump bugün hakim karşısına çıkacak"
##  [7] "Ermeni model Armine Harutyunyan terör örgütüne katıldı... Azerbaycan'ı tehdit etti!"
##  [8] "Son dakika: Nobel Tıp Ödülü'nü kazanan isimler belli oldu"
##  [9] "Meksika'da facia... Kilisenin çatısı çöktü: 9 ölü, 50 yaralı"
## [10] "AB bakanları Kiev'de toplanıyor"
```

Scraping multiple pages

Notice how the previous website we scraped contains multiple pages with news articles. It would be a lot of work to copy paste the URL for each page and run the scraping code on at a time.
'for' loops come into play when we need to scrape multiple pages in one call.
Our base URL in this case will be "https://www.hurriyet.com.tr/dunya/?p=". THe 'for' loop will then add the page number at the end using the seq() function.
In this example, we'll also see how to scrape multiple fields and combine them in one dataframe.
Let's create an empty dataframe to store our data in first.

```
columns = c("headline","description")
hurriyet_news = data.frame(matrix(nrow = 0, ncol = length(columns)))
colnames(hurriyet_news) = columns
```

```
for (tr_news in seq(from = 1, to = 3, by = 1)) {
  link = paste0("https://www.hurriyet.com.tr/dunya/?p=",tr_news)
  headline = link %>% read_html() %>% html_nodes("#content h2") %>% html_text()
  description = link %>% read_html() %>% html_nodes("#content p") %>% html_text()
  temp = data.frame(headline, description)
  hurriyet_news = rbind(hurriyet_news, temp)
  rm(temp)
}
```

Scraping a table

html_table() is used to retrieve complete tables.
This Wikipedia page contains tables detailing (current and historic) Turkish political parties.

```
turkish_parties = read_html("https://en.wikipedia.org/wiki/List_of_political_parties_in_Turkey") %>% htr
```

Notice how it scrapes all the tables that are on the web page.
To select your desired table(s), you simply specify its index number.

```
turkish_parties = read_html("https://en.wikipedia.org/wiki/List_of_political_parties_in_Turkey") %>% htr
turkish_parties = turkish_parties[[5]]
turkish_parties
```

Scraping data that is not on the specified webpage

Sometimes, you need to scrape data on a parent page and a child page simultaneously. This is when html_attr() comes into play. Example 5: https://www.oireachtas.ie/en/debates/questions/
For this workshop, I will filter the results to only see questions asked 4 July 2023:
https://www.oireachtas.ie/en/debates/questions/?page=1&datePeriod=dates&questionType=all&toDate=04%2F07%2F2023&fromDate=04%2F07%2F2023&departmentToggle=member&resultsPerPage=50&viewBy=question
(Always examine the URLs when scraping multiple pages!)
We first need to collect the 'secondary' urls for each parliamentary questions.
As done previously, we create an empty dataframe to store our data in.

```
columns = c("quest_url")
parliamentary_questions = data.frame(matrix(nrow = 0, ncol = length(columns)))
colnames(parliamentary_questions) = columns
```

We have 15 pages that contain parliamentary questions, so we use a 'for' loop (after examining the URL pattern for the pages!). For the sake of time, we will only do the first 3 pages today.

```
for (parl_quest in seq(from = 1, to = 3, by = 1)) {
  link = paste0("https://www.oireachtas.ie/en/debates/questions/?page=",parl_quest,"&datePeriod=dates&qu
  quest_url = link %>% read_html() %>% html_nodes(".u-btn-secondary") %>% html_attr("href")
  temp = data.frame(quest_url)
  parliamentary_questions = rbind(parliamentary_questions, temp)
  rm(temp)
}
```

Notice how the URLs don't contain the main domain name of the website? Let's try again.

```
columns = c("quest_url")
parliamentary_questions = data.frame(matrix(nrow = 0, ncol = length(columns)))
colnames(parliamentary_questions) = columns

for (parl_quest in seq(from = 1, to = 3, by = 1)) {
  link = paste0("https://www.oireachtas.ie/en/debates/questions/?page=",parl_quest,"&datePeriod=dates&qu
  quest_url = link %>% read_html() %>% html_nodes(".u-btn-secondary") %>% html_attr("href") %>% paste0(
  temp = data.frame(quest_url)
  parliamentary_questions = rbind(parliamentary_questions, temp)
  rm(temp)
}
```

We now have the secondary links, let's retrieve our data, also using a loop to make things easier. This time we'll add sys.sleep in order to not overwhelm the website. This basically adds a time interval (in seconds) between calls.

```
columns = c("mp_name", "quest_title")
questions_content = data.frame(matrix(nrow = 0, ncol = length(columns)))
colnames(questions_content) = columns

for (quest in seq(from = 1, to = length(parliamentary_questions$quest_url), by = 1)) {
  mp_name = parliamentary_questions$quest_url[quest] %>% read_html() %>% html_nodes(".from") %>% html_t
```

```
  quest_title = parliamentary_questions$quest_url[quest] %>% read_html() %>% html_nodes(".c-hero__title
  temp = data.frame(mp_name, quest_title)
  questions_content = rbind(questions_content, temp)
  rm(temp)
}
```

Let's do something with these data: Create a plot showing the top 10 MPs who posted parliamentary questions in this period:

```
frequency_mps = questions_content %>% count(mp_name) %>% arrange(desc(n))
frequency_mps = frequency_mps[1:10,]

ggplot(frequency_mps, aes(x = fct_reorder(mp_name, n), y = n, colour = n)) +
  geom_point(aes(size = n)) +
  coord_flip() +
  scale_color_gradient(low = "green1", high = "green4") +
  scale_y_continuous(limits = c(10,20), breaks = seq(10,20,2)) +
  labs(x = NULL, y = "Number of questions posted on 4 July 2023") +
  theme_minimal() +
  theme(legend.position = "none")
```