Web Scraping

Stanley Sugiarto

Web Scraping

Web scraping involves extracting data from websites, typically utilizing the HTML and CSS elements present on web pages. It is a valuable technique for gathering various types of data, including both numeric and categorical data. To perform web scraping, the process typically involves the following steps:

1. Fetching: This involves retrieving the content of web pages, similar to how a browser fetches pages when a user accesses a website.
2. Extraction: Once the content is retrieved, we then extract the relevant data from it.
3. Utilization: The extracted data can then be used for various purposes, such as loading into a database, copying into a spreadsheet, or performing further analysis.

Web scraping typically involves interacting with HTML, CSS, and JavaScript code to extract desired information. HTML defines the structure of a web page, CSS determines its appearance, and JavaScript handles its behaviour. Automated web scraping processes can collect data more efficiently and accurately compared to manual methods, making it a popular technique for gathering data from the web. Examples of scraped data include names, phone numbers, business information, URLs, email addresses, and more.

Understanding HTML and CSS

HTML and CSS are the building blocks of web pages, so having an understanding of these languages is fundamental for successful web scraping.

1. Element Identification: HTML tags and CSS selectors pinpoint specific webpage elements for extraction.
2. Data Extraction: HTML structure guides data extraction from paragraphs, tables, and lists.
3. Attribute Understanding: Attributes like IDs and classes provide metadata for targeted data extraction.
4. Data Parsing: HTML and CSS knowledge aids in data cleaning and parsing for analysis.

Web Scraping in R

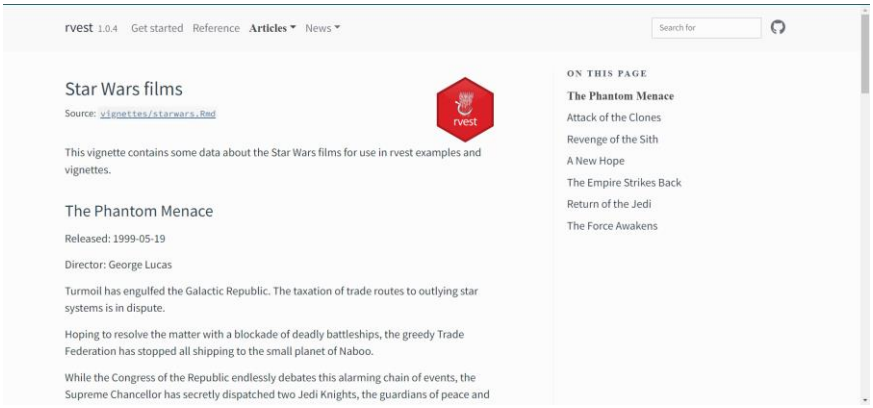
In this project, we will use R in RStudio to web scrape. There are several libraries in RStudio to do this, this includes rvest, RSelenium, and httr. With different kinds of usage and approaches. We will use rvest and tidyverse in this project, rvest will be used for fetching and handling data extraction from the internet while tidyverse is required to manipulate and wrangle the data (Wickham, H, 2023).

1. Install and load the required packages in RStudio,

```
install.packages('rvest')
install.packages('tidyverse')
install.packages('tidyr')


library(rvest)
library(tidyverse)
library(tidyr)

```

2. Choose a Webpage to Scrape, in this project we will try to scrape the rvest Starwars article. To start here are the link to these webpages;

- https://rvest.tidyverse.org/articles/starwars.html



3. HTTP GET request

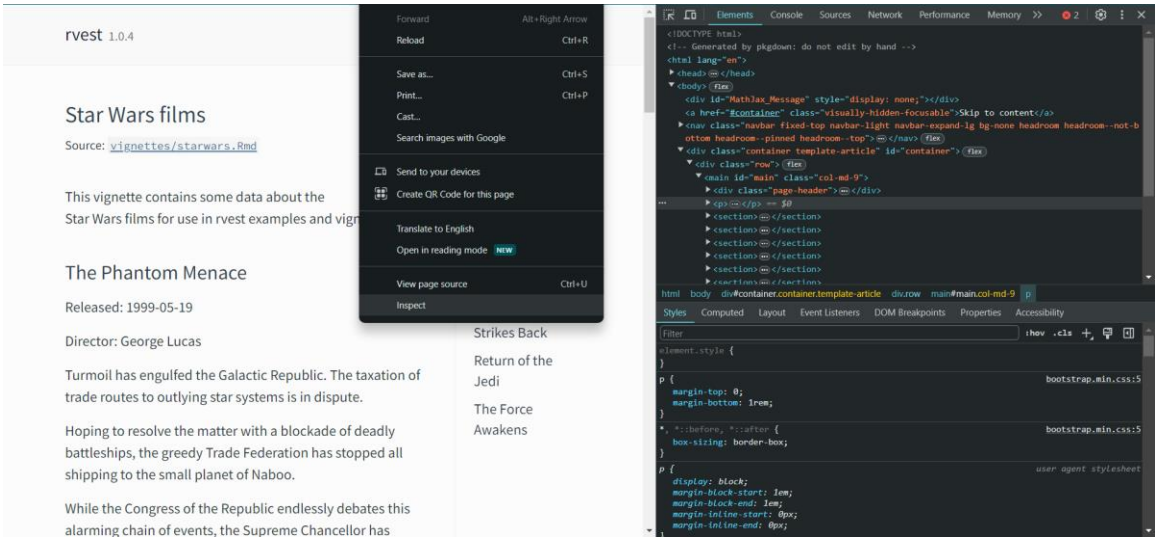
```
install.packages('rvest')
install.packages('tidyverse')
install.packages('tidyr')


library(rvest)
library(tidyverse)
library(tidyr)

URL <- "https://rvest.tidyverse.org/articles/starwars.html"
page <- read_html(URL)

print (page)
```

```
Console  Terminal  Background Jobs
R 4.3.2 · ~/
× dplyr::lag()           masks stats::lag()
i Use the conflicted package to force all conflicts to become errors
> library(tidyr)
> URL <- "https://rvest.tidyverse.org/articles/starwars.html"
> page <- read_html(URL)
> print (page)
{html_document}
<html lang="en">
[1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">\n<meta c ...
[2] <body>\n    <a href="#container" class="visually-hidden-focusable">Skip to content</ ...
>
```

This code uses the `read_html()` function to download and parse the HTML content of the webpage. By printing the content of `page`, it shows the parsed HTML code of the webpage. The *{html_document}* is the HTML code which is the underlying structure of this page.

4. Understanding CSS and XPath elements

Most Webpages are structured not only with HTML but with CSS and Javascript as well. Using CSS enables developer to create stunning and interactive webpages on top of the HTML underlying code. In the Star Wars film webpage, we can inspect the webpage by right-clicking and choose inspect.



Now we can inspect and understand the HTML code and CSS attribute in this page. We can see that ` <head> ….. </head> ` contains all the metadata of the page, while the `<body> …. </body>` contains all the information and data we require.

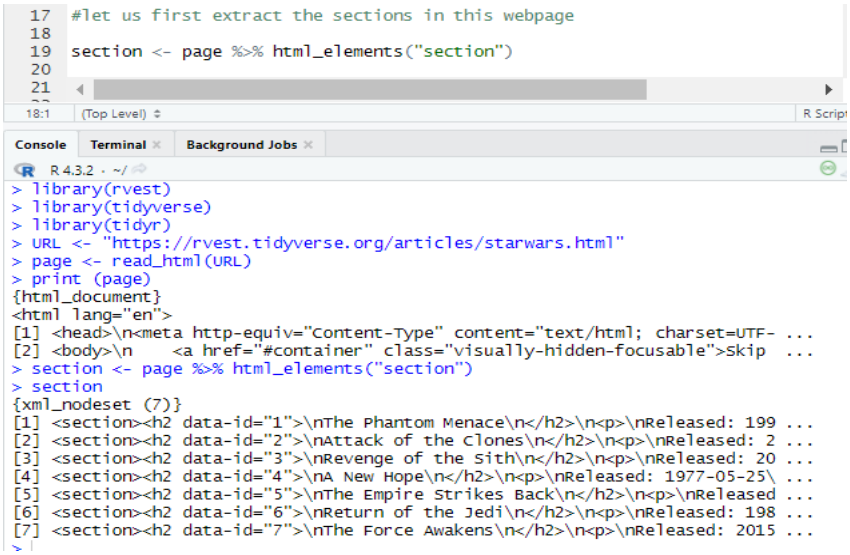
5. Parsing and finding attributes

   When we inspect the webpage, we can find that the body has several `<section>….</section>`. If we expand these sections, we can find that in contains

```
<section>
  <h2 data-id="1" id="the-phantom-menace"> The Phantom Menace </h2>
  <p> Released: 1999-05-19 </p>
  <p> == $0
    " Director: "
    <span class="director">George Lucas</span>
  </p>
  <div class="crawl">…</div>
</section>
```

Now we understand that the attribute `<h2> ….. </h2>` contains the movie title, `<p>….</p>` contains the release date, the other `<p>` paragraph contains the director, and `<div> … </div>` contains the intro. This structure is consistent for all similar content of Star Wars movie in this page. As visualized in the picture that ‘director’ is a class of “director”, while ‘intro’ is a class of “crawl”.

Using the function `html_element() and html_elements()` we can extract the movie title, release date, director, and intro. Let us make a section object first which contains all the content we want.



```
17  #let us first extract the sections in this webpage
18
19  section <- page %>% html_elements("section")
20
21
```

```
Console  Terminal  Background Jobs
R 4.3.2 · ~/
> library(rvest)
> library(tidyverse)
> library(tidyr)
> URL <- "https://rvest.tidyverse.org/articles/starwars.html"
> page <- read_html(URL)
> print (page)
{html_document}
<html lang="en">
[1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF- ...
[2] <body>\n    <a href="#container" class="visually-hidden-focusable">Skip  ...
> section <- page %>% html_elements("section")
> section
{xml_nodeset (7)}
[1] <section><h2 data-id="1">\nThe Phantom Menace\n</h2>\n<p>\nReleased: 199 ...
[2] <section><h2 data-id="2">\nAttack of the Clones\n</h2>\n<p>\nReleased: 2 ...
[3] <section><h2 data-id="3">\nRevenge of the Sith\n</h2>\n<p>\nReleased: 20 ...
[4] <section><h2 data-id="4">\nA New Hope\n</h2>\n<p>\nReleased: 1977-05-25\ ...
[5] <section><h2 data-id="5">\nThe Empire Strikes Back\n</h2>\n<p>\nReleased ...
[6] <section><h2 data-id="6">\nReturn of the Jedi\n</h2>\n<p>\nReleased: 198 ...
[7] <section><h2 data-id="7">\nThe Force Awakens\n</h2>\n<p>\nReleased: 2015 ...
>
```

Since sections are denoted in html code we can use the function `html_text2()` to parse it into regular English (Wickham, H, 2023).

Now we can implement the pipe %>% operator to html_element(). To extract the titles of the movies we can use the selector “h2”. Since director and intro has a class then we use the dot (`.`) operator to denote the class, which would be “.director”. Therefore, our code should be like the following.

```
19  section <- page %>% html_elements("section")
20
21  Title <- section %>% html_element("h2") %>% html_text2()
22  Date <- section %>% html_element("p") %>% html_text2() %>%
23    str_remove("Released: ") %>% parse_date()
24
25  director <- section %>% html_element(".director") %>% html_text2()
26  intro <- section %>% html_element(".crawl") %>% html_text2()
```

6. Make a tibble to display the data

```
29 tibble(title = Title, release_date = Date, Director = director, Into = intro)
30
```

```
[2] <section><h2 data-id="2">\nAttack of the Clones\n</h2>\n<p>\nReleased: 2 ...
[3] <section><h2 data-id="3">\nRevenge of the Sith\n</h2>\n<p>\nReleased: 20 ...
[4] <section><h2 data-id="4">\nA New Hope\n</h2>\n<p>\nReleased: 1977-05-25\ ...
[5] <section><h2 data-id="5">\nThe Empire Strikes Back\n</h2>\n<p>\nReleased ...
[6] <section><h2 data-id="6">\nReturn of the Jedi\n</h2>\n<p>\nReleased: 198 ...
[7] <section><h2 data-id="7">\nThe Force Awakens\n</h2>\n<p>\nReleased: 2015 ...
> Title <- section %>% html_element("h2") %>% html_text2()
> Date <- section %>% html_element("p") %>% html_text2() %>% str_remove("Released: ") %>% parse_date()
> director <- section %>% html_element(".director") %>% html_text2()
> intro <- section %>% html_element(".crawl") %>% html_text2()
> tibble(title = Title, release_date = Date, Director = director, Into = intro)
# A tibble: 7 × 4
  title                 release_date Director         Into
  <chr>                 <date>       <chr>            <chr>
1 The Phantom Menace    1999-05-19   George Lucas     "Turmoil has engulfed the Galactic Republic. The ta…
2 Attack of the Clones  2002-05-16   George Lucas     "There is unrest in the Galactic Senate. Several th…
3 Revenge of the Sith   2005-05-19   George Lucas     "War! The Republic is crumbling under attacks by th…
4 A New Hope            1977-05-25   George Lucas     "It is a period of civil war. Rebel spaceships, str…
5 The Empire Strikes Back 1980-05-17 Irvin Kershner   "It is a dark time for the Rebellion. Although the …
6 Return of the Jedi    1983-05-25   Richard Marquand "Luke Skywalker has returned to his home planet of …
7 The Force Awakens     2015-12-11   J. J. Abrams     "Luke Skywalker has vanished. In his absence, the s…
>
```

7. Adding the gross revenue

Extracting from https://www.boxofficemojo.com/franchise/fr3125251845/ , we can use the similar syntax and implementing tidyverse to add the lifetime gross of Star Wars eps 1 to 7.

```
34 URL2 <- "https://www.boxofficemojo.com/franchise/fr3125251845/"
35 page <- read_html(URL2)
36
37 print(page)
38
39 summary_gross <- page %>% html_elements(".mojo-estimatable") %>% html_text2()
40 summary_title <- page %>% html_elements(".mojo-cell-wide") %>% html_text2() %>% parse_character()
41
42 summary_gross = summary_gross[-1]
43 summary_title = summary_title[-1]
44
45 tb2 = tibble(title = summary_title,
46        gross = parse_number(summary_gross, na = "-")
47   )
48 tb2 <- filter(tb2, gross >= 200000000) %>% arrange(title) %>% slice(-(1:2) , -7 , -11)
49
50 tb <- tb %>% mutate(gross = tb2$gross)
51 tb
52
```

```
+   )
> tb2 <- filter(tb2, gross >= 200000000) %>% arrange(title) %>% slice(-(1:2) , -7 , -11)
> tb <- tb %>% mutate(gross = tb2$gross)
> tb
# A tibble: 7 × 4
  title                 release_date Director            gross
  <chr>                 <date>       <chr>               <dbl>
1 The Phantom Menace    1999-05-19   George Lucas     431088295
2 Attack of the Clones  2002-05-16   George Lucas     302191252
3 Revenge of the Sith   2005-05-19   George Lucas     380270577
4 A New Hope            1977-05-25   George Lucas     307263857
5 The Empire Strikes Back 1980-05-17 Irvin Kershner   209398025
6 Return of the Jedi    1983-05-25   Richard Marquand 252583617
7 The Force Awakens     2015-12-11   J. J. Abrams     936662225
>
```

Selecting CSS attributes

The trickiest part in web scraping might be the part where we need to pick the selector for certain data or attributes. We need to practice in order to get ourselves familiar when it comes to selecting a selector.

In this case, the element has numerous attributes to make that specific part of the webpage. It is a trial-and-error process when picking the right selector. Which in this case the right element for the gross profit is `.mojo-estimatable` and `.mojo-cell-wide` for the title.

There are many difficulties when trying to scrape different kinds of web pages. Wikipedia has numerous to hundreds of links in its content which can be tricky for beginners. Some webpages like IMDB and newspaper webpages have different kinds of underlying structures. While understanding how to web scrape, there are some concepts we need to understand more to successfully web scrape.

Dynamic content, websites that heavily rely on JavaScript to load content dynamically can be challenging to scrape because the content may not be present in the initial HTML source code. Elements like buttons, forms, dropdown menus, and sliders that respond to user input in real time. Animated transitions, pop-ups, tooltips, and other visual effects enhance user interaction and experience. Content that updates automatically based on external factors such as stock prices, weather conditions, or social media feeds. Techniques like waiting for JavaScript to execute or using tools like Selenium may be necessary.

On the other hand, formatting and parsing HTML content is crucial to producing the right analysis. In our project example, the gross revenue is parsed as characters rather than numbers. Therefore, having a good understanding of data tidying is crucial.

Summary

When learning how to web scrape, I found that having many errors helped me to understand the concept more deeply. Having to encounter many obstacles is part of the learning process, especially with easy access to many resources in the internet. I think it is interesting to learn such a powerful skill to produce analysis and help people understand more. We can use this special skill to make predictions, visualizations, summaries, and conclusions without having to collect data. Mastering web scraping empowers you to leverage the wealth of information available on the web to generate valuable insights, drive informed decision-making, and contribute to advancements in various fields. Embracing the challenges and errors encountered along the learning journey ultimately leads to a deeper understanding and proficiency in this powerful skill.

Reference list

Wickham, H., Grolemund., G., & Çetinkaya-Rundel, M. (2023, July 18). *24 web scraping*. R for Data Science (2e) - 24 Web scraping. https://r4ds.hadley.nz/webscraping

Wickham, H. (n.d.). *Star wars films*. • rvest. https://rvest.tidyverse.org/articles/starwars.html

Franchise: Star wars. Box Office Mojo. (n.d.). https://www.boxofficemojo.com/franchise/fr3125251845/

Soetewey, A. (2023, January 16). *Web scraping in R*. Stats and R. https://statsandr.com/blog/web-scraping-in-r/#web-scraping-in-r