Topic 10: Web Scraping

ISOM3390: Business Programming in R

HTML

> *"HTML is a markup language for describing web documents (web pages)."*
>
> *––– W3Schools*

HyperText Markup Language (HTML for short) documents are basically structured as follows:

```
<!DOCTYPE html>
<html>

<head><title>Sample HTML Page</title></head>

<body>
<h1>This is a heading.</h1>
<p>This is a typical paragraph.</p>
<p class = "notThisOne">
This is a paragraph of the "notThisOne" class.
</p>
<p id = "thisOne">
But I only want this <a href = "sample.html">paragraph</a>.
</p>
</body>

</html>
```

This is a heading.

This is a typical paragraph.

This is a paragraph of the "notThisOne" class.

But I only want this paragraph.

HTML Elements

HTML elements are written with a start tag, an end tag, and with the content in between: `<tagname>content</tagname>`.

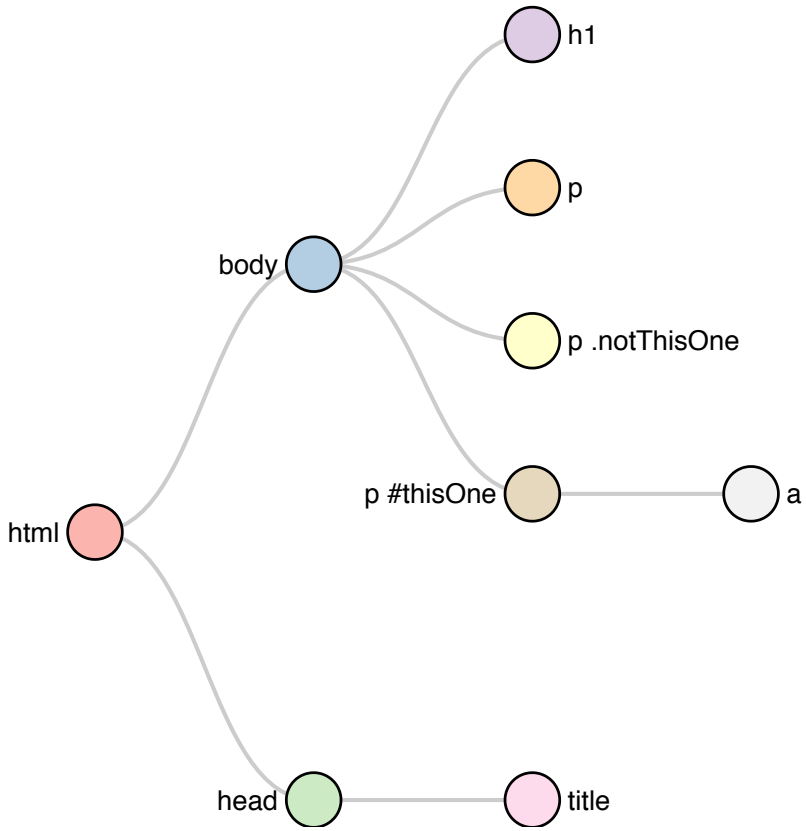
- `<h1>`, `<h2>`,…, `<h6>`: largest heading, second largest heading, etc.
- `<p>`: paragraph elements
- `<ul>` or `<ol>`: unordered or ordered bulleted list
- `<li>`: individual List item
- `<div>`: division or section
- `<table>`: table
- and many others …

The tags typically contain the textual content we wish to scrape.

```
<p> This paragraph represents a typical text paragraph in HTML form </p>
```

Tree Representation of HTML Nodes

These textual components are referred to as nodes of an HTML document.




```
<!DOCTYPE html>
<html>

<head><title>Sample HTML Page</title></head>

<body>
<h1>This is a heading.</h1><Br>
<p>This is a typical paragraph.</p>
<p class = "notThisOne">
This is a paragraph of the "notThisOne" class.
</p>
<p id = "thisOne">
But I only want this <a href = "sample.html">paragraph</a>.
</p>
</body>

</html>
```

Locating Desired Data

It is through these tags that we can locate desired textual information of HTML documents.

An HTML node can further have the `class` or `id` property.

```
<p>This is a typical paragraph.</p>
<p class = "notThisOne"> This is a paragraph of the "notThisOne" class.</p>
<p id = "thisOne"> But I only want this <a href = "sample.html">paragraph</a>.</p>
```

The difference between an `id` and a `class` is that an `id` is used to identify one element, whereas a `class` is used to identify more than one.

We can use the `class` or `id` property to differentiate the section we want from other sections.

For example, we want the paragraph with `id="thisOne"` and not `class="notThisOne"`, how do we get this content in R?

CSS

> *"CSS describes how HTML elements are to be displayed on screen, paper, or in other media."*
>
> *––– W3Schools*

Cascading Style Sheets (CSS for short) is a style sheet language for describing the presentation of a document written in a markup language.

```
h1 {
  color: royalblue;
  text-align: center; }

p {
  color: salmon;
  font-family: "Century Gothic", CenturyGothic, Geneva, AppleGothic, sans-serif; }

p.notThisOne {
  margin-bottom: 40px;
  font-family:"Times New Roman", Georgia, Serif; }

p#thisOne {
  color: #a1d99b;
  font-style: italic; }
```

This is a heading.

This is a typical paragraph.

This is a paragraph of the "notThisOne" class.

But I only want this paragraph.

Working Together

HTML dictates the content and structure of a webpage, while CSS modifies design and display of HTML elements.

example.css

```
h1 {
  color: royalblue;
  text-align: center; }

p {
  color: salmon;
  font-family: "Century Gothic", CenturyGothic, Geneva, AppleGothic, Serif; }

p.notThisOne {
  margin-bottom: 40px;
  font-family:"Times New Roman", Georgia, Serif; }

p#thisOne {
  color: #a1d99b;
  font-style: italic; }
```

sample.html

```
<!DOCTYPE html>
<html>

<head>
<link href="example.css" rel="stylesheet" type="text/css">
<title>Sample HTML Page</title>
</head>

<body>
<h1>This is a heading.</h1>
<p>This is a typical paragraph.</p>
<p class = "notThisOne">
This is a paragraph of the "notThisOne" class.</p>
<p id = "thisOne">
But I only want this <a href = "sample.html">paragraph</a>.
</p>
</body>

</html>
```

CSS Selectors

In CSS, selectors are patterns used to select the element(s) we want to style.

Selector	Example	Explanation
element	`p`	Select all `<p>` elements
.class	`.notThisOne`	Select all elements with `class="notThisOne"`
#id	`#thisOne`	Select the element with `id="thisOne"`
[attribute]	`[id]`	Select all elements with an `id` attribute
element.class	`p.notThisOne`	select all `<p>` elements with `class="notThisOne"`
element#id	`p#thisOne`	Select all `<p>` elements with `id="thisOne"`

Combinator	Example	Explanation
`","`	`div, p`	Select all `<p>` elements as well as all `<div>` elements
`" "`	`div p`	Select all `<p>` elements inside `<div>` elements
`">"`	`div > p`	Select all `<p>` elements whose parent is a `<div>` element
`"+"`	`div + p`	Select the `<p>` element that immediately follows a `<div>` element
`"~"`	`div ~ p`	Select any `<p>` elements as long as they follow a `<div>` element

More use can be find here.

`rvest` for Web Scraping

`rvest` is a `tidyverse` package for Web scraping.

```
library(rvest)  # Load it explicitly; this also installs xml2, a package that rvest relies on

## Loading required package: xml2
```

It provides great functions (wrappers around the `xml2` and `httr` packages; both in the `tidyverse`) for parsing HTML documents and makes it easy to scrape data from HTML web pages.

The basic workflow is:

- Download the HTML and turn it into an XML file with `read_html()`;
- Extract specific nodes based on certain criteria with `html_nodes()`;
- Extract specific content from nodes with various functions, e.g., `html_text()` to get the text, `html_attr()` to get the attribute value.

Downloading the Document: Trump's Lies

JAN. 21 "I wasn't a fan of Iraq. I didn't want to go into Iraq." *(He was for an invasion before he was against it.)* **JAN. 21** "A reporter for Time magazine — and I have been on their cover 14 or 15 times. I think we have the all-time record in the history of Time magazine." *(Trump was on the cover 11 times and Nixon appeared 55 times.)* **JAN. 23** "Between 3 million and 5 million illegal votes caused me to lose the popular vote." *(There's no evidence of illegal voting.)* **JAN. 25** "Now, the audience was the biggest ever. But this crowd was massive. Look how far back it goes. This crowd was massive." *(Official aerial photos show Obama's 2009 inauguration was much more heavily attended.)* **JAN. 25** "Take a look at the Pew reports (which show voter fraud.)" *(The report never mentioned voter fraud.)* **JAN. 25** "You had millions of people that now aren't insured anymore." *(The real number is less than 1 million, according to the Urban Institute.)* **JAN. 25** "So, look, when President Obama was there two weeks ago making a speech,

Use `read_html()` to read an HTML document into R, returning an **XML** document.

```
(webpage <- read_html("https://www.nytimes.com/interactive/2017/06/23/opinion/trumps-lies.html"))
```

```
## {xml_document}
## <html lang="en" class="no-js page-interactive section-opinion page-theme-standard tone-opinion page-interactive-default limit-small layo
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset= ...
## [2] <body>\n<style>\n.lt-ie10 .messenger.suggestions {\n  display: block ...
```

Collecting All the Records

In the **HTML** code, every record is surrounded by the `<span>` tag of `class="short-desc"`:

```
<span class="short-desc">
      <strong> DATE </strong> LIE <span class="short-truth"><a href="URL"> EXPLANATION </a></span>
</span>
```

Use `html_nodes()` to identify all the `<span>` tags that belong to `class="short-desc"`.

```
results <- html_nodes(webpage, ".short-desc")
```

- The first argument is the HTML document or a node previously extracted from the document;
- The second argument is a CSS selector (or an **`XPath`** expression) a query language for selecting nodes from an XML document) to identify which nodes to select.

This returns all the XML nodes that contain the information that interests us.

```
## {xml_nodeset (180)}
##  [1] <span class="short-desc"><strong>Jan. 21 </strong>“I wasn't a fan o ...
##  [2] <span class="short-desc"><strong>Jan. 21 </strong>“A reporter for T ...
##  [3] <span class="short-desc"><strong>Jan. 23 </strong>“Between 3 millio ...
##  [4] <span class="short-desc"><strong>Jan. 25 </strong>“Now, the audienc ...
##  [5] <span class="short-desc"><strong>Jan. 25 </strong>“Take a look at t ...
##  [6] <span class="short-desc"><strong>Jan. 25 </strong>“You had millions ...
##  [7] <span class="short-desc"><strong>Jan. 25 </strong>“So, look, when P ...
##  [8] <span class="short-desc"><strong>Jan. 26 </strong>“We've taken in t ...
##  [9] <span class="short-desc"><strong>Jan. 26 </strong>“I cut off hundre ...
## [10] <span class="short-desc"><strong>Jan. 28 </strong>“The coverage abo ...
## [11] <span class="short-desc"><strong>Jan. 29 </strong>“The Cuban-Americ ...
## [12] <span class="short-desc"><strong>Jan. 30 </strong>“Only 109 people  ...
## [13] <span class="short-desc"><strong>Feb. 3 </strong>“Professional anar ...
## [14] <span class="short-desc"><strong>Feb. 4 </strong>“After being force ...
## [15] <span class="short-desc"><strong>Feb. 5 </strong>“We had 109 people ...
## [16] <span class="short-desc"><strong>Feb. 6 </strong>“I have already sa ...
## [17] <span class="short-desc"><strong>Feb. 6 </strong>“It's gotten to a  ...
## [18] <span class="short-desc"><strong>Feb. 6 </strong>“The failing @nyti ...
## [19] <span class="short-desc"><strong>Feb. 6 </strong>“And the previous  ...
## [20] <span class="short-desc"><strong>Feb. 7 </strong>“And yet the murde ...
## ...
```

Extracting Individual Details

The general structure of a single record is:

```
<strong> DATE </strong> LIE <span class="short-truth"><a href="URL"> EXPLANATION </a></span>
```

Let's extract each of the 4 parts of the 1st record.

To select the node for the `DATE`, use the `html_nodes()` function with the selector `"strong"`.

```
html_nodes(results[1], "strong")
```

```
## {xml_nodeset (1)}
## [1] <strong>Jan. 21 </strong>
```

Then use `html_text()` to extract only the text, with the `trim` argument active to trim leading and trailing spaces.

```
(date <- html_nodes(results[1], "strong") %>% html_text(trim = TRUE))
```

```
## [1] "Jan. 21"
```

Extracting the `LIE`

Use `xml_contents()` (from the `xml2` package) to extract the `LIE` (`xml2` is required by `rvest`, so it is not necessary to load it separately).

```
xml_contents(results[1])
```

```
## {xml_nodeset (3)}
## [1] <strong>Jan. 21 </strong>
## [2] “I wasn't a fan of Iraq. I didn't want to go into Iraq.”
## [3] <span class="short-truth"><a href="https://www.buzzfeed.com/andrewka ...
```

`xml_contents()` returns all the nodes that are part of `results[1]`.

We are interested in the `LIE`, which is the text of the second node:

```
xml_contents(results[1])[2] %>% html_text(trim = TRUE) %>% str_sub(2, -2)
```

```
## [1] "I wasn't a fan of Iraq. I didn't want to go into Iraq."
```

Extracting the `EXPLANATION` and the `URL`

For the `EXPLANATION`, select the text within the `<span>` tag that belongs to `class=".short-truth"`

```
(explanation <- results[1] %>% html_node(".short-truth") %>% html_text(trim = TRUE) %>%
    str_sub(2, -2))
```

```
## [1] "He was for an invasion before he was against it."
```

Note that the URL is an attribute within the `<a>` tag.

To get the `URL`, first use `html_nodes()` to select the `<a>` node, and then extract the value of the `href` attribute with the `html_attr()` function.

```
(url <- results[1] %>% html_node("a") %>% html_attr("href"))
```

```
## [1] "https://www.buzzfeed.com/andrewkaczynski/in-2002-donald-trump-said-he-supported-invading-iraq-on-the"
```

Building the Dataset

This process is extended to all the rest using a `for` loop or an `apply` function.

Eeach iteration creates a single data frame of 4 columns (for the `DATE`, the `LIE`, the `EXPLANATION`, and the `URL`) for each record:

```
records <- vector("list", length = length(results))

for (i in seq_along(results)) {
    date <- str_c(results[i] %>% html_nodes("strong") %>% html_text(trim = TRUE),
        ", 2017")
    lie <- str_sub(xml_contents(results[i])[2] %>% html_text(trim = TRUE), 2,
        -2)
    explanation <- str_sub(results[i] %>% html_nodes(".short-truth") %>% html_text(trim = TRUE),
        2, -2)
    url <- results[i] %>% html_nodes("a") %>% html_attr("href")
    records[[i]] <- data_frame(date = date, lie = lie, explanation = explanation,
        url = url)
}

df <- bind_rows(records)
```

Bind all data frames in the list together using the `bind_rows()` function from the `dplyr` package.

```
df
```

```
## # A tibble: 180 x 4
##    date    lie                 explanation              url
##    <chr>   <chr>               <chr>                    <chr>
##  1 Jan. 2… I wasn't a fan of … He was for an invasion … https://www.buzzf…
##  2 Jan. 2… A reporter for Tim… Trump was on the cover … http://nation.tim…
##  3 Jan. 2… Between 3 million … There's no evidence of … https://www.nytim…
##  4 Jan. 2… Now, the audience … Official aerial photos … https://www.nytim…
##  5 Jan. 2… Take a look at the… The report never mentio… https://www.nytim…
##  6 Jan. 2… You had millions o… The real number is less… https://www.nytim…
##  7 Jan. 2… So, look, when Pre… There were no gun homic… https://www.dnain…
##  8 Jan. 2… We've taken in ten… Vetting lasts up to two… https://www.nytim…
##  9 Jan. 2… I cut off hundreds… Most of the cuts were a… https://www.washi…
## 10 Jan. 2… The coverage about… It never apologized.     https://www.nytim…
## # ... with 170 more rows
```

Note that the column for the date is considered a character vector. It would be nice to have it as a **datetime** vector instead.

Lubridate

`Lubridate` is a `tidyverse` package that makes it easier to work with datetime data.

```
library(lubridate)  # Load it explicitly
```

`Lubridate` provides a collection of functions, , e.g., `ymd()`, `mdy()`, `dmy()`, etc., named with a sequence of initials of **m**onth, **d**ay, and **y**ear.

They match components of dates stored in character and numeric vectors, and transforms them to **Date** or **POSIXct** objects.

Here, we use `mdy()` to make the conversion:

```
df$date <- mdy(df$date)
glimpse(df)
```

```
## Observations: 180
## Variables: 4
## $ date        <date> 2017-01-21, 2017-01-21, 2017-01-23, 2017-01-25, 2...
## $ lie         <chr> "I wasn't a fan of Iraq. I didn't want to go into ...
## $ explanation <chr> "He was for an invasion before he was against it."...
## $ url         <chr> "https://www.buzzfeed.com/andrewkaczynski/in-2002-...
```

`html_table()` for Parsing HTML Tables

Some web pages display their data in an easy-to-read table.

`rvest` has a handy tool that converts an HTML table to a data frame.

```
historical_prices <- read_html("https://finance.yahoo.com/quote/%5EGSPC/history?p=%5EGSPC") %>%
    html_nodes("table") %>% html_table(header = TRUE) %>% .[[1]] %>% as_tibble()
```

Date	Open	High	Low	Close*	Adj Close**	Volume
Nov 16, 2018	2,718.54	2,746.75	2,712.16	2,736.27	2,736.27	3,975,180,000
Nov 15, 2018	2,693.52	2,735.38	2,670.75	2,730.20	2,730.20	4,179,140,000
Nov 14, 2018	2,737.90	2,746.80	2,685.75	2,701.58	2,701.58	4,402,370,000
Nov 13, 2018	2,730.05	2,754.60	2,714.98	2,722.18	2,722.18	4,091,440,000
Nov 12, 2018	2,773.93	2,775.99	2,722.00	2,726.22	2,726.22	3,670,930,000
Nov 09, 2018	2,794.10	2,794.10	2,764.24	2,781.01	2,781.01	4,019,090,000
Nov 08, 2018	2,806.38	2,814.75	2,794.99	2,806.83	2,806.83	3,630,490,000
Nov 07, 2018	2,774.13	2,815.15	2,774.13	2,813.89	2,813.89	3,914,750,000

Showing 1 to 8 of 101 entries

Previous 1 2 3 4 5 … 13 Next

Exporting the Dataset

To export the dataset, we can use either the default `write.csv()` function, or the `write_csv()` function from the `readr` package.

```
str(write_csv)
```

```
## function (x, path, na = "NA", append = FALSE, col_names = !append)
```

```
write_csv(df, "trump_lies.csv")
```

Similarly, to retrieve the dataset, we can use either the default function `read.csv()` or the `read_csv()` function from the `readr` package.

```
str(read_csv)
```

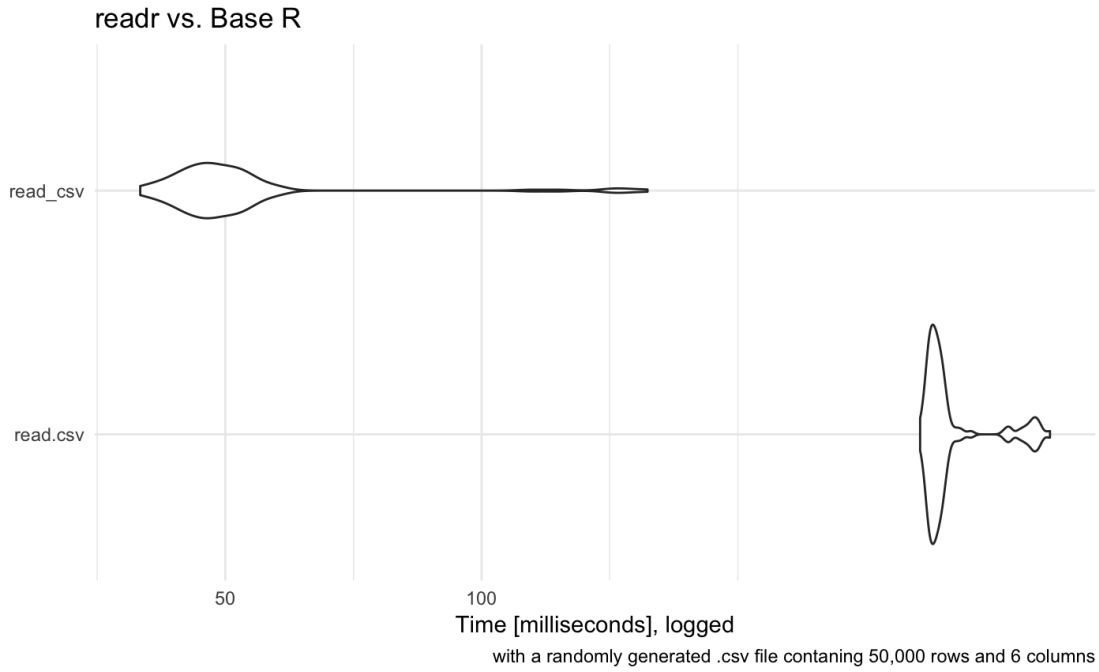
```
## function (file, col_names = TRUE, col_types = NULL, locale = default_locale(),
##     na = c("", "NA"), quoted_na = TRUE, quote = "\"", comment = "",
##     trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000, n_max),
##     progress = show_progress())
```

`readr` for Reading Rectangular Data

`readr` is a core package in the `tidyverse`.

It provides a fast and friendly way to read rectangular data (like csv, tsv, and fwf).

One of the main advantages of `readr` functions over base R functions is that they are typically much faster (up to 10x):



Managing Files

After we downloaded thousands of files, we want a good system to keep track of them.

Here are a couple of R functions that allow us to manage our files:

```
head(list.files(), 10)  #shows all of the files in the current directory
```

```
##  [1] "backup"
##  [2] "big-logo.png"
##  [3] "Business Programming in R - Google Search_files"
##  [4] "Business Programming in R - Google Search.htm"
##  [5] "css.png"
##  [6] "example.css"
##  [7] "hex-lubridate.png"
##  [8] "hex-readr.png"
##  [9] "hex-rvest.png"
## [10] "html.png"
```

```
head(list.files(recursive = TRUE), 10)  #recursive=TRUE shows subfiles
```

```
##  [1] "backup/HTMLDOMTree.png"
##  [2] "backup/Infographic-HTML-CSS.png"
##  [3] "big-logo.png"
##  [4] "Business Programming in R - Google Search_files/api.js"
##  [5] "Business Programming in R - Google Search_files/cb=gapi.loaded_0"
##  [6] "Business Programming in R - Google Search_files/cb=gapi(1).loaded_0"
##  [7] "Business Programming in R - Google Search_files/dn.js"
##  [8] "Business Programming in R - Google Search_files/dn(1).js"
##  [9] "Business Programming in R - Google Search_files/googlelogo_color_120x44dp.png"
## [10] "Business Programming in R - Google Search_files/images"
```

```
list.files(pattern = "\\.png$")  # adds criteria using regular expressions
```

```
## [1] "big-logo.png"       "css.png"            "hex-lubridate.png"
## [4] "hex-readr.png"      "hex-rvest.png"      "html.png"
## [7] "IMDb.png"           "SelectorGadget.png"
```

The following work flow returns a table of file types in a directory.

```
list.files(pattern="\\.") %>% # get files with "."
str_extract_all("([^.]+$)") %>% # cut the text after the "."
unlist() %>% table() %>% as.matrix() #unlist it, table it, matrix it
```

```
##      [,1]
## css     2
## csv     2
## htm     1
## html    4
## png     8
## Rmd     1
```

Working with File Directories

Use `paste0()` and `getwd()` to construct an absolute path. Consider:

```
write_lines(c("ISOM3390", "Business Programming in R"), paste0(getwd(), "/test/test.txt"))  # Save text
```

A folder called `test` must exist to make it work. In this case, we can wrapping it in `try()` so that the code won't break if this directory doesn't exist.

```
try(write_lines(c("ISOM3390", "Business Programming in R"), paste0(getwd(),
    "/test/test.txt")))
```

```
## Warning in open.connection(path, if (isTRUE(append)) "ab" else "wb"):
## cannot open file '.../Courses/ISOM3390/Lecture Notes/L9-Web Scraping
## /test/test.txt': No such file or directory
```

We can create a folder using `dir.create()`. Now `write_lines()` will work.

```
dir.create(paste0(getwd(), "/test"), recursive = TRUE)
write_lines(c("ISOM3390", "Business Programming in R"), paste0(getwd(), "/test/test.txt"))
```

The `test` folder can be deleted using `unlink()`:

```
unlink(paste0(getwd(), "/test"), recursive = TRUE)
```

Basic Encoding Issues

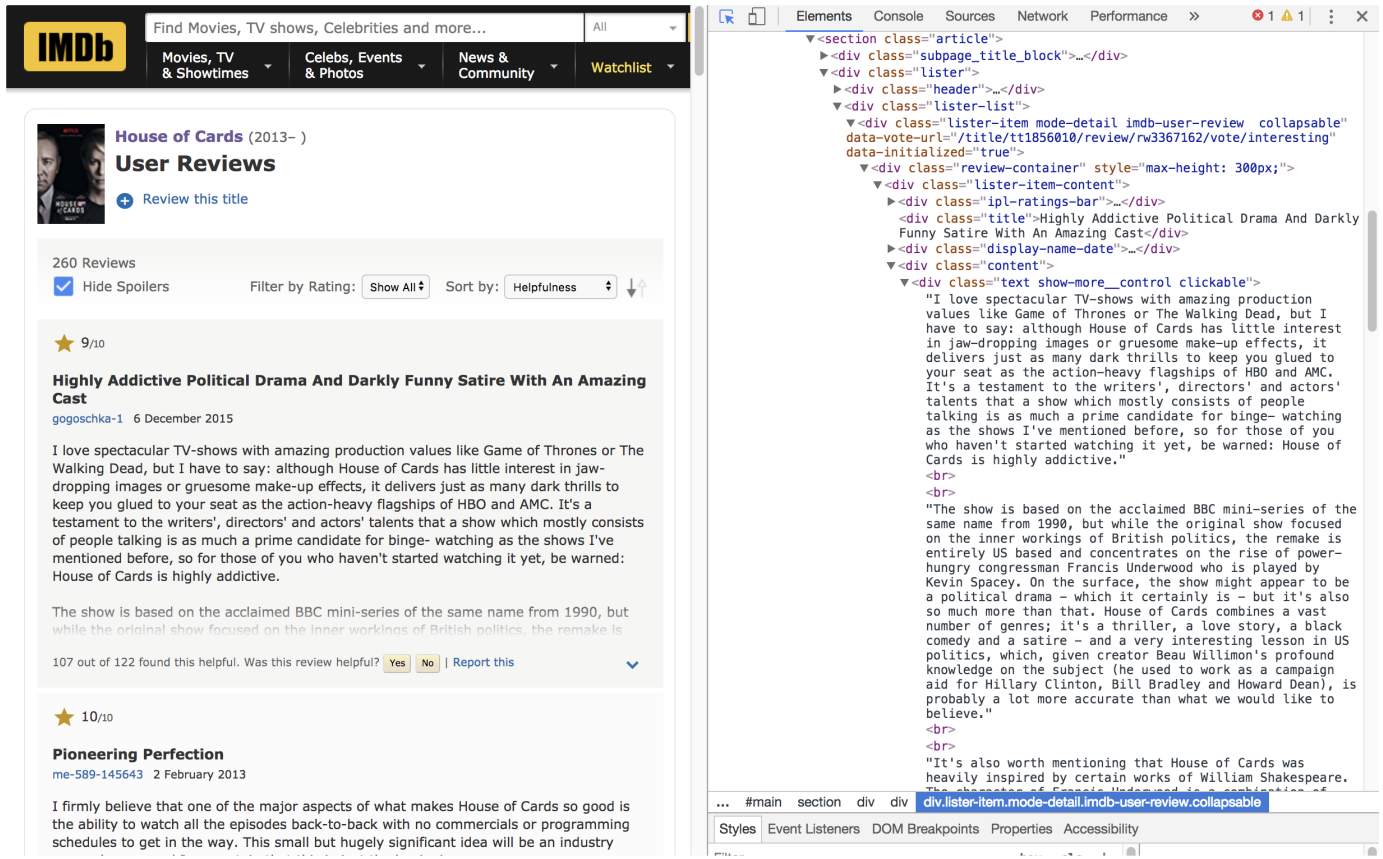
Working with non-Latin text (such as Chinese text, Japanese text, Arabic text, etc.) brings encoding problems.

Most of the time, we just need to specify that the encoding is "UTF-8" when we load or save files, i.e. `readLines(link, encoding="UTF-8")`.

UTF stands for Unicode Transformation Format. The '8' means it uses 8-bit blocks to represent a character. UTF-8 encoding captures both plain English text and all other characters that are used (non-Latin letter, emojis, etc.)

We can check (or change) the encoding using the `Encoding()` function.

Example: IMDb Review Pages

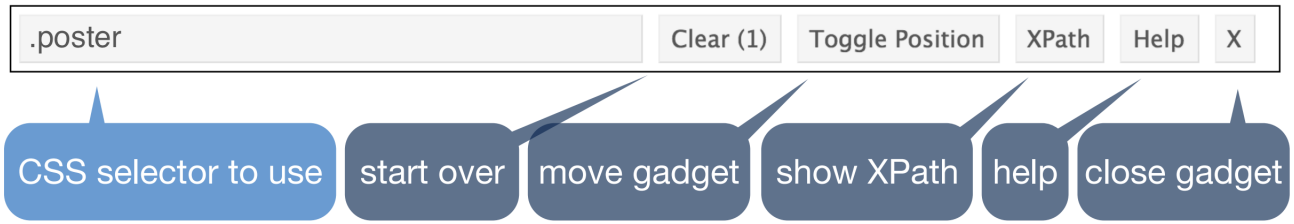


How can we make our HTML node selection process more focused?

Locating Specific Nodes

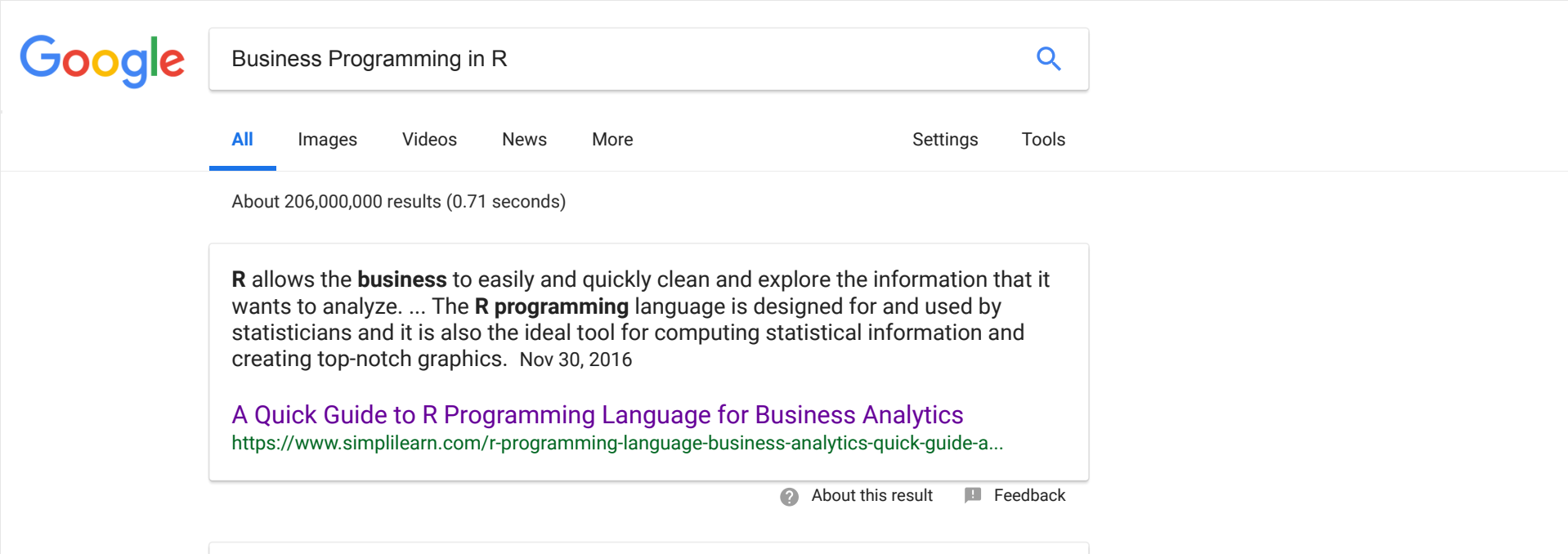
Two ways to find CSS selectors for an HTML element:

- Use browsers' developer tools (e.g., Chrome's **element selector**);
- Use a GUI tool called **SelectorGadget** that helps identify CSS selector combinations from a webpage. To install **SelectorGadget**:
  - Run `vignette("selectorgadget")`
  - Drag **SelectorGadget** link into the browser's bookmark bar

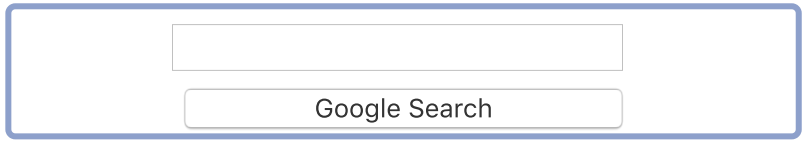


Dynamic Web Pages

We are increasingly encountering pages whose contents are dynamically generated within the user's Web browser; that is, the content is determined only when the page is rendered and is updated dynamically based on user interactions and inputs.



Working with HTML Forms



An HTML form allows a user to enter data that is sent to a server for processing.

Forms are enclosed in the HTML `<form>` tag. This tag specifies where the data entered into the form should be submitted, and the method of submitting the data, `GET` or `POST`.

```
<form action="https://www.google.com/search" id="f" method="get">
<input id="q" name="q" type="text">
<input value="Google Search" aria-label="Google Search" name="btnG" type="submit">
</form>
```

Simulating User Interaction

Simulate a session in an html browser with `html_session()`:

```
session <- html_session("https://www.google.com/webhp?gl=us")
str(session[["response"]], max.level = 1)
```

```
## List of 10
##  $ url        : chr "https://www.google.com/webhp?gl=us"
##  $ status_code: int 200
##  $ headers    :List of 13
##   ..- attr(*, "class")= chr [1:2] "insensitive" "list"
##  $ all_headers:List of 1
##  $ cookies    :'data.frame': 2 obs. of  7 variables:
##  $ content    : raw [1:11539] 3c 21 64 6f ...
##  $ date       : POSIXct[1:1], format: "2018-11-18 17:55:01"
##  $ times      : Named num [1:6] 0 0.0091 0.0161 0.081 0.1408 ...
##   ..- attr(*, "names")= chr [1:6] "redirect" "namelookup" "connect" "pretransfer" ...
##  $ request    :List of 7
##   ..- attr(*, "class")= chr "request"
##  $ handle     :Class 'curl_handle' <externalptr>
##  - attr(*, "class")= chr "response"
```

Manipulating Forms

Extract the form that takes search queries with `html_form()`:

```
(search <- session %>% html_form() %>% .[[1]])
```

```
## <form> 'f' (GET /search)
##   <input hidden> 'ie': ISO-8859-1
##   <input hidden> 'hl': en
##   <input hidden> 'source': hp
##   <input hidden> 'biw':
##   <input hidden> 'bih':
##   <input text> 'q':
##   <input submit> 'btnG': Google Search
##   <input submit> 'btnI': I'm Feeling Lucky
##   <input hidden> 'gbv': 1
```

Modify the field(s) specified by the name-value pair(s) in `set_values()`:

```
(search <- search %>%
   set_values(q = "Business Programming in R"))
```

```
## <form> 'f' (GET /search)
##   <input hidden> 'ie': ISO-8859-1
##   <input hidden> 'hl': en
##   <input hidden> 'source': hp
##   <input hidden> 'biw':
##   <input hidden> 'bih':
##   <input text> 'q': Business Programming in R
##   <input submit> 'btnG': Google Search
##   <input submit> 'btnI': I'm Feeling Lucky
##   <input hidden> 'gbv': 1
```

Submit the form using `submit_form()`, and retrieve the parsed HTML response upon success:

```
search_result <- session %>% submit_form(search)
str(search_result[["response"]], max.level = 1)
```

```
## List of 10
##  $ url        : chr "https://www.google.com/search?ie=ISO-8859-1&hl=en&source=hp&q=Business%20Programming%20in%20R&btnG=Google%20Search&
##  $ status_code: int 200
##  $ headers    :List of 15
##   ..- attr(*, "class")= chr [1:2] "insensitive" "list"
##  $ all_headers:List of 1
##  $ cookies    :'data.frame': 4 obs. of  7 variables:
##  $ content    : raw [1:39614] 3c 21 64 6f ...
##  $ date       : POSIXct[1:1], format: "2018-11-18 17:55:01"
##  $ times      : Named num [1:6] 0 0.000042 0.000044 0.000256 0.077741 ...
##   ..- attr(*, "names")= chr [1:6] "redirect" "namelookup" "connect" "pretransfer" ...
##  $ request    :List of 7
##   ..- attr(*, "class")= chr "request"
##  $ handle     :Class 'curl_handle' <externalptr>
##  - attr(*, "class")= chr "response"
```

Parsing the Response

Use `content()` in the `httr` package to parse the response into an XML document, which can be processed with functions (e.g., `html_nodes()`) in `rvest`:

```
search_result[["response"]] %>% httr::content() %>% html_nodes(".g .r")
```

```
## {xml_nodeset (11)}
##  [1] <h3 class="r"><a href="/url?q=https://www.simplilearn.com/r-program ...
##  [2] <h3 class="r"><a href="/url?q=http://www.bm.ust.hk/isom/files/cours ...
##  [3] <h3 class="r"><a href="/url?q=http://www.bm.ust.hk/isom/files/cours ...
##  [4] <h3 class="r"><a href="/url?q=https://www.business-science.io/busin ...
##  [5] <h3 class="r"><a href="/url?q=https://www.computerworld.com/article ...
##  [6] <h3 class="r"><a href="/url?q=https://ust.space/review/ISOM3000A&am ...
##  [7] <h3 class="r"><a href="/url?q=https://www.simplilearn.com/r-program ...
##  [8] <h3 class="r"><a href="/url?q=https://www.coursera.org/learn/r-prog ...
##  [9] <h3 class="r"><a href="/url?q=https://www.edureka.co/blog/what-do-y ...
## [10] <h3 class="r"><a href="/url?q=https://www.r-project.org/about.html& ...
## [11] <h3 class="r"><a href="/url?q=https://dataseer.com/r-programming-fo ...
```

Navigating through a Website

Navigate around the search result with `jump_to()`, `follow_link()`, `back()`, `forward()`, etc.:

```
str(follow_link)

## function (x, i, css, xpath, ...)

search_result %>% follow_link("Next") %>% # find the first link containg this text
  .[["response"]] %>% httr::content() %>%
  html_nodes(".g .r")

## {xml_nodeset (10)}
##  [1] <h3 class="r"><a href="/url?q=https://www.r-project.org/about.html& ...
##  [2] <h3 class="r"><a href="/url?q=https://qz.com/1063071/the-great-r-ve ...
##  [3] <h3 class="r"><a href="/url?q=http://uc-r.github.io/introduction&am ...
##  [4] <h3 class="r"><a href="/url?q=https://dataseer.com/r-programming-fo ...
##  [5] <h3 class="r"><a href="/url?q=https://hackernoon.com/5-free-r-progr ...
##  [6] <h3 class="r"><a href="/url?q=https://www.datacamp.com/&amp;sa=U&am ...
##  [7] <h3 class="r"><a href="/url?q=https://searchbusinessanalytics.techt ...
##  [8] <h3 class="r"><a href="/url?q=https://www.youtube.com/watch%3Fv%3Dx ...
##  [9] <h3 class="r"><a href="/url?q=https://www.udemy.com/topic/r-program ...
## [10] <h3 class="r"><a href="/url?q=https://www.udemy.com/programming-r/& ...
```

For more details: `help(package = "rvest")`

Selenium: Web Browser Automator

Selenium is a **Web browser automation tool**.

It opens a browser of our choice and "drives" it to perform tasks as a human being would, such as:

- Clicking buttons
- Entering information in forms
- Searching for specific information on the web pages

Because the Selenium server is a standalone JAVA program, we need to download and install the **Java SE Development Kit** to run it.

Download the latest Selenium standalone server binary manually. Look for `selenium-server-standalone-x.xx.x.jar`.

Starting a Selenium Server

- Open the OS console (e.g., **Command Prompt** in Windows or **Terminal** on a MacOS computer) and navigate to where the binary is stored and run: `java -jar selenium-server-standalone-x.xx.x.jar`.
- For some browsers, we also need a driver program (e.g., chromedriver) that acts as a bridge between the browser and the Selenium server.
- If the browser to use requires a driver program, execute the above command with the appropriate option to locate the required driver program, e.g., `-Dwebdriver.chrome.driver=[relative path to chromedriver]`.
- By default, the Selenium Server listens for connections on port `4444`. We can change it to `4445` using the `-port` option.

Run: `java -Dwebdriver.chrome.driver=chromedriver -jar selenium-server-standalone-x.xx.x.jar -port 4445`

Working with Selenium in R

The **`RSelenium`** package allows us to connect to the Selenium Server and program its behaviors from within R.

```
library(RSelenium)
```

To connect to the running server, use the `remoteDriver()` function to instantiate a new `remoteDriver` object with appropriate options:

```
remDr <- remoteDriver(remoteServerAddr = "localhost", port = 4445L, browserName = "chrome")  # 'firefox', 'internet explorer', 'iphone', et
str(remDr, max.level = 1)
```

```
## Reference class 'remoteDriver' [package "RSelenium"] with 17 fields
##  and 74 methods, of which 60 are  possibly relevant:
##    acceptAlert, addCookie, buttondown, buttonup, checkError, checkStatus, click, close, closeall, closeServer,
##    closeWindow, deleteAllCookies, deleteCookieNamed, dismissAlert, doubleclick, errorDetails, executeAsyncScript,
##    executeScript, findElement, findElements, getActiveElement, getAlertText, getAllCookies, getCurrentUrl,
##    getCurrentWindowHandle, getLogTypes, getPageSource, getSession, getSessions, getStatus, getTitle, getWindowHandles,
##    getWindowPosition, getWindowSize, goBack, goForward, initialize, initialize#errorHandler, log, maxWindowSize,
##    mouseMoveToLocation, navigate, obscureUrlPassword, open, phantomExecute, queryRD, quit, refresh, screenshot,
##    sendKeysToActiveElement, sendKeysToAlert, setAsyncScriptTimeout, setImplicitWaitTimeout, setTimeout,
##    setWindowPosition, setWindowSize, show#envRefClass, showErrorClass, switchToFrame, switchToWindow
```

Opening the Browser

Use `remDr`'s `open()` method to send a request to the Selenium server to start the browser:

```
remDr$open(silent = TRUE)
```

We can query the status of the remote server using the `getStatus()` method:

```
remDr$getStatus() %>% str()
```

```
## List of 5
##  $ ready  : logi TRUE
##  $ message: chr "Server is running"
##  $ build  :List of 3
##   ..$ revision: chr "e82be7d358"
##   ..$ time    : chr "2018-11-14T08:25:53"
##   ..$ version : chr "3.141.59"
##  $ os     :List of 3
##   ..$ arch   : chr "x86_64"
##   ..$ name   : chr "Mac OS X"
##   ..$ version: chr "10.13.2"
##  $ java   :List of 1
##   ..$ version: chr "1.8.0_152"
```

Navigating through Webpages

Navigate to a url using `navigate()`:

```
remDr$navigate("http://www.imdb.com/title/tt1856010/reviews")
```

We navigate to a second page:

```
remDr$navigate("https://www.imdb.com/title/tt0944947/reviews")
remDr$getCurrentUrl()
```

```
## [[1]]
## [1] "https://www.imdb.com/title/tt0944947/reviews"
```

We can go back and forth using the methods `goBack()` and `goForward()`.

We can use the `refresh()` method to refresh the current page:

```
remDr$refresh()
```

Locating HTML Elements

A number of methods can be used for searching. We can search by `id`, `name`, or `class`:

```
loadmore <- remDr$findElement(using = "id", value = "load-more-trigger")  # using = 'name' or 'class'
```

Or using css selector or XPath:

```
loadmore <- remDr$findElement(using = "css", ".ipl-load-more__button")  # using = 'xpath' when using XPath
```

The method returns a `webElement` object:

```
class(loadmore)
```

```
## [1] "webElement"
## attr(,"package")
## [1] "RSelenium"
```

Sending Events to Elements

Mimic clicking the link to load new items (iteratively until all reviews are loaded):

```
loadmore$clickElement()
```

Supported events include:

- Sending text
- Sending key Presses
- Sending mouse events

More examples can be found here

Returning the Page Source

Use the `getPageSource()` to get the source of the last loaded page.

```
page_source <- remDr$getPageSource() %>% .[[1]]
```

Use `read_html()` to return an XML document as before:

```
page_source %>% read_html()
```

```
## {xml_document}
## <html xmlns="http://www.w3.org/1999/xhtml" xmlns:og="http://ogp.me/ns#" xmlns:fb="http://www.facebook.com/2008/fbml" class=" scriptsOn">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">\n<script async="" src="https://images ...
## [2] <body id="styleguide-v2" class="fixed">\n\n        \n\n<script>\n    if (typeof uet == 'function') {\n      uet(" ...
```

Close the conenction after use:

```
remDr$close()
```