ISOM 3390: Business Programming in R

Instructor: Bingjie Qian (PhD candidate)

Summer 2020, HKUST

Topic 10: Web Scraping

Nowadays, vast amount of information exists *online*, which is usually unstructured since it doesn't come in a neatly packaged spreadsheet. Fortunately, **HTML web pages** organize information in a certain way, so that we can scrape the information we want from web pages.

10.1 HTML Basics

HyperText Markup Language (HTML) is a markup language for describing web documents (web pages).

Open the HTML document "sample.html" (download from Canvas) in a *text editor*, it looks like this:

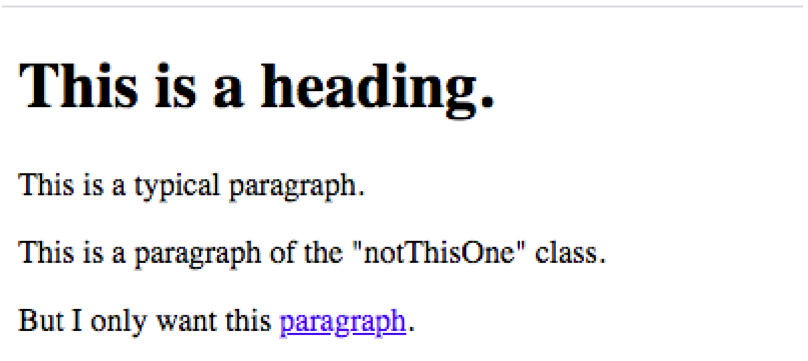
```
<!DOCTYPE html>
<html>

<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>
```

Open it with a *web browser*, it looks like this:



This is a heading.

This is a typical paragraph.

This is a paragraph of the "notThisOne" class.

But I only want this paragraph.

Fig. 1 Sample HTML Page

HTML Elements

An HTML document consists of HTML elements, which are written with the start tag, the end tag, and the content in between:

```
<tagname> content </tagname>
```

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

These elements typically contain the content we wish to scrape and may include attributes.

```
<tagname attribute1="value1" attribute2="value2"> content </tagname>
```

These HTML elements are the **nodes** of an HTML document.

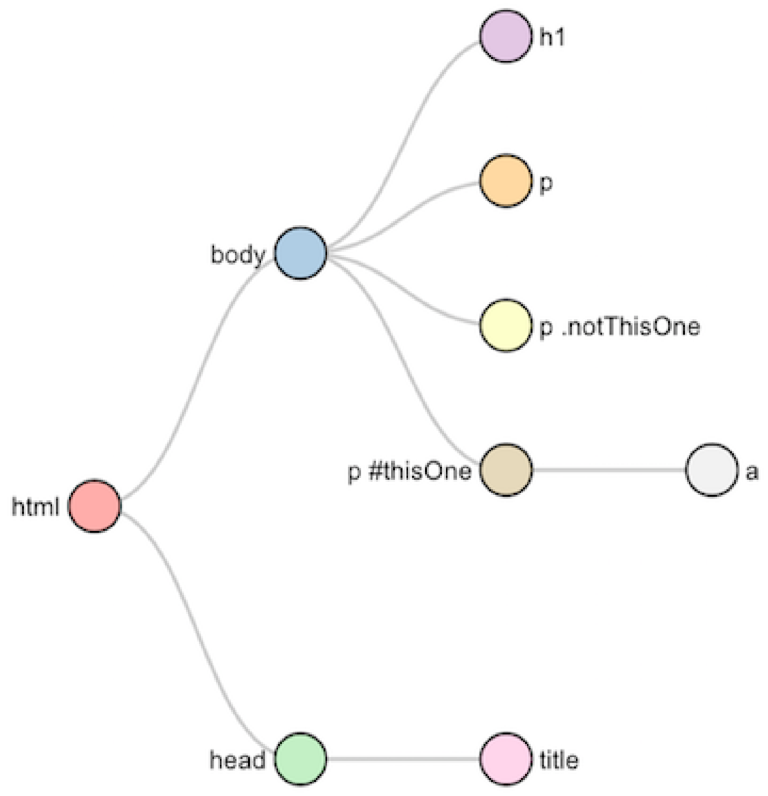


Fig. 2 Tree Representation of HTML Nodes

Locating Desired Information

It is through these tags that we can locate desired information of HTML documents. An HTML node can further have the `class` or `id` attribute.

```
<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` can be used to identify *more than one element*.

We can use the `class` or `id` attribute to differentiate the elements we want from other elements.

For example, we can select the paragraph with `id = "thisOne"` and not `class = "notThisOne"` .

10.2 CSS Basics

Cascading Style Sheets (CSS) is a style sheet language, and describes the presentation of a document written in a markup language.

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

Open the "sample.css" document in a *text editor*, it looks like this:

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

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

p.notThisOne {
  color: salmon; }

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

In the "sample.html" document, add `<link href="sample.css" rel="stylesheet" type="text/css">` into the head:

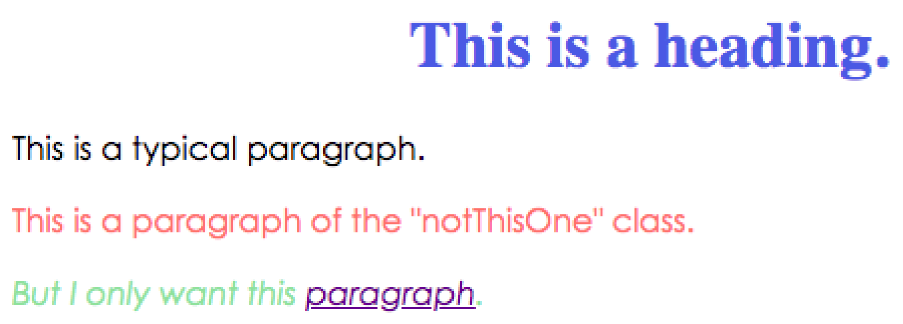
```
<!DOCTYPE html>
<html>

<head>
<link href="sample.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>
```

Then the HTML elements will be styled according to the CSS document "sample.css":


This is a heading.

This is a typical paragraph.

This is a paragraph of the "notThisOne" class.

But I only want this paragraph.

Fig. 3 HTML and CSS

CSS Selector

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 the `<p>` element with `id = "thisOne"`

Selectors can be combined.

Combinator	Example	Explanation
`","`	`div, p`	Select all `<div>` elements as well as all `<p>` elements

Combinator	Example	Explanation
`" "`	`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 found here: https://css-tricks.com/almanac/selectors/ (https://css-tricks.com/almanac/selectors/).

10.3 `rvest` for Web Scraping

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

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.0 ──
```

```
## ✓ ggplot2 3.3.2     ✓ purrr   0.3.4
## ✓ tibble  3.0.1     ✓ dplyr   1.0.0
## ✓ tidyr   1.1.0     ✓ stringr 1.4.0
## ✓ readr   1.3.1     ✓ forcats 0.5.0
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: xml2
```

```
##
## Attaching package: 'rvest'
```

```
## The following object is masked from 'package:purrr':
##
##     pluck
```

```
## The following object is masked from 'package:readr':
##
##     guess_encoding
```

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

The basic workflow is:

1. Download the HTML and turn it into an HTML document object with `read_html()`;

2. Extract specific nodes based on certain criteria with `html_nodes()`;
3. Extract specific content from nodes with various functions, e.g., `html_text()` to get the text, `html_attr()` to get the attribute values, `html_table()` to get the table.

Example 1: Trump's Lies

An article on New York Times: https://www.nytimes.com/interactive/2017/06/23/opinion/trumps-lies.html (https://www.nytimes.com/interactive/2017/06/23/opinion/trumps-lies.html).

Open the web page in Chrome, then you can right click and choose "Inspect" to view the html document. When you put your mouse on an html code element, the corresponding webpage component will be highlighted. This can help you quickly locate the html code you are interested in.

1. Download the HTML document

Use `read_html()` to read the HTML document into R:

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

```
## {html_document}
## <html lang="en" class="no-js page-interactive section-opinion page-theme-standard
tone-opinion page-interactive-default limit-small layout-xlarge app-interactive" item
id="https://www.nytimes.com/interactive/2017/06/23/opinion/trumps-lies.html" itemtype
="http://schema.org/NewsArticle" itemscope="" xmlns:og="http://opengraphprotocol.org/
schema/">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
## [2] <body>\n<style>\n.lt-ie10 .messenger.suggestions {\n  display: block !imp ...
```

2. Extract nodes from the HTML document

In the HTML document, every record of Trump's lie 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>
```

For example, the first record is:

```
<span class="short-desc">
<strong>Jan. 21 </strong>"I wasn't a fan of Iraq. I didn't want to go into Iraq." <sp
an class="short-truth"><a href="https://www.buzzfeed.com/andrewkaczynski/in-2002-dona
ld-trump-said-he-supported-invading-iraq-on-the" target="_blank">(He was for an invas
ion before he was against it.)</a></span>
</span>
```

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

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

```
## {xml_nodeset (180)}
##  [1] <span class="short-desc"><strong>Jan. 21 </strong>“I wasn't a fan of Ira ...
##  [2] <span class="short-desc"><strong>Jan. 21 </strong>“A reporter for Time m ...
##  [3] <span class="short-desc"><strong>Jan. 23 </strong>“Between 3 million and ...
##  [4] <span class="short-desc"><strong>Jan. 25 </strong>“Now, the audience was ...
##  [5] <span class="short-desc"><strong>Jan. 25 </strong>“Take a look at the Pe ...
##  [6] <span class="short-desc"><strong>Jan. 25 </strong>“You had millions of p ...
##  [7] <span class="short-desc"><strong>Jan. 25 </strong>“So, look, when Presid ...
##  [8] <span class="short-desc"><strong>Jan. 26 </strong>“We've taken in tens o ...
##  [9] <span class="short-desc"><strong>Jan. 26 </strong>“I cut off hundreds of ...
## [10] <span class="short-desc"><strong>Jan. 28 </strong>“The coverage about me ...
## [11] <span class="short-desc"><strong>Jan. 29 </strong>“The Cuban-Americans,  ...
## [12] <span class="short-desc"><strong>Jan. 30 </strong>“Only 109 people out o ...
## [13] <span class="short-desc"><strong>Feb. 3 </strong>“Professional anarchist ...
## [14] <span class="short-desc"><strong>Feb. 4 </strong>“After being forced to  ...
## [15] <span class="short-desc"><strong>Feb. 5 </strong>“We had 109 people out  ...
## [16] <span class="short-desc"><strong>Feb. 6 </strong>“I have already saved m ...
## [17] <span class="short-desc"><strong>Feb. 6 </strong>“It's gotten to a point ...
## [18] <span class="short-desc"><strong>Feb. 6 </strong>“The failing @nytimes w ...
## [19] <span class="short-desc"><strong>Feb. 6 </strong>“And the previous admin ...
## [20] <span class="short-desc"><strong>Feb. 7 </strong>“And yet the murder rat ...
## ...
```

This returns all the nodes that contain the information we are interested in (Trump's lies).

```
str(html_nodes)
```

```
## function (x, css, xpath)
```

Usage of `html_nodes()` :

- 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) to identify which nodes to select.

3. Extract content from the nodes

Let's first consider the first record:

```
results[1]
```

```
## {xml_nodeset (1)}
## [1] <span class="short-desc"><strong>Jan. 21 </strong>“I wasn't a fan of Iraq ...
```

Its structure is:

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

Now let's extract each of its 4 parts.

(1) `DATE`

To select the node for the `DATE` , use `html_nodes()` 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)
date
```

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

(2) `LIE`

Use `xml_contents()` to extract the `LIE` :

```
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/andrewkaczyns ...
```

`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:

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

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

(3) `EXPLANATION`

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)
explanation
```

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

(4) `URL`

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_nodes("a") %>% html_attr("href")
url
```

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

This process of extracting the `DATE`, `LIE`, `EXPLANATION` and `URL` for the first record is extended to all the rest records using a `for` loop or an `*apply` function.

Each 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), ", 20
17")
  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]] <- tibble(date = date, lie = lie, explanation = explanation, url = ur
l)
}

class(records)
```

```
## [1] "list"
```

```
length(records)
```

```
## [1] 180
```

```
records[[1]]
```

```
## # A tibble: 1 x 4
##   date     lie                    explanation          url
##   <chr>    <chr>                  <chr>                <chr>
## 1 Jan. 21… I wasn't a fan of I… He was for an invasi… https://www.buzzfeed.com/…
```

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

```
records <- bind_rows(records)
records
```

```
## # A tibble: 180 x 4
##    date    lie                   explanation                url
##    <chr>   <chr>                 <chr>                      <chr>
##  1 Jan. 2… I wasn't a fan of Ir… He was for an invasion be… https://www.buzzfee…
##  2 Jan. 2… A reporter for Time … Trump was on the cover 11… http://nation.time.…
##  3 Jan. 2… Between 3 million an… There's no evidence of il… https://www.nytimes…
##  4 Jan. 2… Now, the audience wa… Official aerial photos sh… https://www.nytimes…
##  5 Jan. 2… Take a look at the P… The report never mentione… https://www.nytimes…
##  6 Jan. 2… You had millions of … The real number is less t… https://www.nytimes…
##  7 Jan. 2… So, look, when Presi… There were no gun homicid… https://www.dnainfo…
##  8 Jan. 2… We've taken in tens … Vetting lasts up to two y… https://www.nytimes…
##  9 Jan. 2… I cut off hundreds o… Most of the cuts were alr… https://www.washing…
## 10 Jan. 2… The coverage about m… It never apologized.       https://www.nytimes…
## # … with 170 more rows
```

Example 2: Yahoo Finance

Yahoo Finance (https://finance.yahoo.com/quote/%5EGSPC/history?p=%5EGSPC (https://finance.yahoo.com/quote/%5EGSPC/history?p=%5EGSPC)) displays its data in an easy-to-read table.

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

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

```
## # A tibble: 101 x 7
##    Date         Open     High     Low      `Close*` `Adj Close**` Volume
##    <chr>        <chr>    <chr>    <chr>    <chr>    <chr>         <chr>
##  1 Jul 28, 2020 3,234.27 3,243.72 3,216.17 3,218.44 3,218.44      4,027,890,000
##  2 Jul 27, 2020 3,219.84 3,241.43 3,214.25 3,239.41 3,239.41      3,963,910,000
##  3 Jul 24, 2020 3,218.58 3,227.26 3,200.05 3,215.63 3,215.63      3,894,900,000
##  4 Jul 23, 2020 3,271.64 3,279.99 3,222.66 3,235.66 3,235.66      4,290,460,000
##  5 Jul 22, 2020 3,254.86 3,279.32 3,253.10 3,276.02 3,276.02      4,255,190,000
##  6 Jul 21, 2020 3,268.52 3,277.29 3,247.77 3,257.30 3,257.30      4,547,960,000
##  7 Jul 20, 2020 3,224.29 3,258.61 3,215.16 3,251.84 3,251.84      3,971,200,000
##  8 Jul 17, 2020 3,224.21 3,233.52 3,205.65 3,224.73 3,224.73      3,993,830,000
##  9 Jul 16, 2020 3,208.36 3,220.39 3,198.59 3,215.57 3,215.57      3,961,230,000
## 10 Jul 15, 2020 3,225.98 3,238.28 3,200.76 3,226.56 3,226.56      4,669,760,000
## # … with 91 more rows
```

10.4 Managing Files

Read and Write Files

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

```
str(write_csv)
```

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

```
write_csv(records, "trump_lies.csv")   # the csv file is saved in your working direct
ory
```

Similarly, to read the dataset, we can use either the function `read.csv()` from base R 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 = TRU
E,
##     skip = 0, n_max = Inf, guess_max = min(1000, n_max), progress = show_progress
(),
##     skip_empty_rows = TRUE)
```

```
records <- read_csv("trump_lies.csv")
```

```
## Parsed with column specification:
## cols(
##   date = col_character(),
##   lie = col_character(),
##   explanation = col_character(),
##   url = col_character()
## )
```

```
records
```

```
## # A tibble: 180 x 4
##    date    lie                        explanation                url
##    <chr>   <chr>                      <chr>                      <chr>
##  1 Jan. 2… I wasn't a fan of Ir… He was for an invasion be… https://www.buzzfee…
##  2 Jan. 2… A reporter for Time … Trump was on the cover 11… http://nation.time.…
##  3 Jan. 2… Between 3 million an… There's no evidence of il… https://www.nytimes…
##  4 Jan. 2… Now, the audience wa… Official aerial photos sh… https://www.nytimes…
##  5 Jan. 2… Take a look at the P… The report never mentione… https://www.nytimes…
##  6 Jan. 2… You had millions of … The real number is less t… https://www.nytimes…
##  7 Jan. 2… So, look, when Presi… There were no gun homicid… https://www.dnainfo…
##  8 Jan. 2… We've taken in tens … Vetting lasts up to two y… https://www.nytimes…
##  9 Jan. 2… I cut off hundreds o… Most of the cuts were alr… https://www.washing…
## 10 Jan. 2… The coverage about m… It never apologized.       https://www.nytimes…
## # … with 170 more rows
```

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

If we have a large number of files, we need a good system to keep track of them.

We can use `list.files()` to show all the files in the current directory.

```
list.files()
```

```
## [1] "10_Web Scraping.pdf"  "10_Web Scraping.Rmd"  "10_Web-Scraping.html"
## [4] "10_Web-Scraping.Rmd"  "images"               "sample.css"
## [7] "sample.html"          "trump_lies.csv"
```

Set `recursive = TRUE` to show subfiles:

```
list.files(recursive = TRUE)
```

```
##  [1] "10_Web Scraping.pdf"    "10_Web Scraping.Rmd"    "10_Web-Scraping.html"
##  [4] "10_Web-Scraping.Rmd"    "images/HTML_CSS.png"    "images/HTML_nodes.png"
##  [7] "images/HTML_sample.png" "sample.css"             "sample.html"
## [10] "trump_lies.csv"
```

Add criteria using regular expressions:

```
list.files(pattern = "\\.Rmd$")
```

```
## [1] "10_Web Scraping.Rmd" "10_Web-Scraping.Rmd"
```

Sometimes we want to call upon file directories to organize the scraped data; e.g., progrommatically create directories by date to store the data scraped on different days.

Use `str_c()` to construct an absolute path:

```
pathname <- str_c(getwd(), "/", Sys.Date())  # Sys.Date() returns the current date
```

Use `dir.create()` to create a new directory:

```
dir.create(pathname, recursive = TRUE)  # recursive = TRUE creates all ancestor direc
tories recursively
```

Save `records` in the new directory:

```
write_csv(records, str_c(pathname, "/trump_lies.csv"))
```

The directory can be deleted using `unlink()` :

```
unlink(pathname, recursive = TRUE)
```