Week 12: *Webscraping*

EMSE 4571 / 6571: Intro to Programming for Analytics

John Paul Helveston

April 16, 2026

Quiz 10

10:00

Write your name on the quiz!

Rules:

- Work alone; no outside help of any kind is allowed.
- No calculators, no notes, no books, no computers, no phones.

Week 12: *Webscraping*

1. Scraping static pages

2. Scraping multiple pages

BREAK

3. Using APIs

Some disclaimers (here for more details)

You're probably okay if the data is:

- Public
- Non-personal
- Factual

Otherwise, consult a lawyer and / or maybe don't scrape it.

Terms of service

Generally are not upheld, unless you **need an account to access the data**.

Copyright

Data is not copyright protected (in the US). But works are. Be careful.

Week 12: *Webscraping*

1. HTML basics

1. Scraping static pages

2. Scraping multiple pages

BREAK

3. Using APIs

Hyper**T**ext **M**arkup **L**anguage

```
<html>
<head>
  <title>Page title</title>
</head>
<body>
  <h1 id='first'>A heading</h1>
  <p>Some text &amp;amp; <b>some bold text.</b></p>
  <img src='myimg.png' width='100' height='100'>
</body>
```

HTML has a hierarchical structure formed by:

- Start and end **"tags"** (e.g. `<tag>` and `</tag>`)
- Optional attributes (e.g. `id='first'`)
- Contents (everything in between the start and end tag).

Common tags

- `<h1>` = Header level 1
- `<a>` = Url link
- `<b>` = **Bold** text
- `<i>` = *Italic* text
- `<p>` = Paragraph
- `<li>` = List item

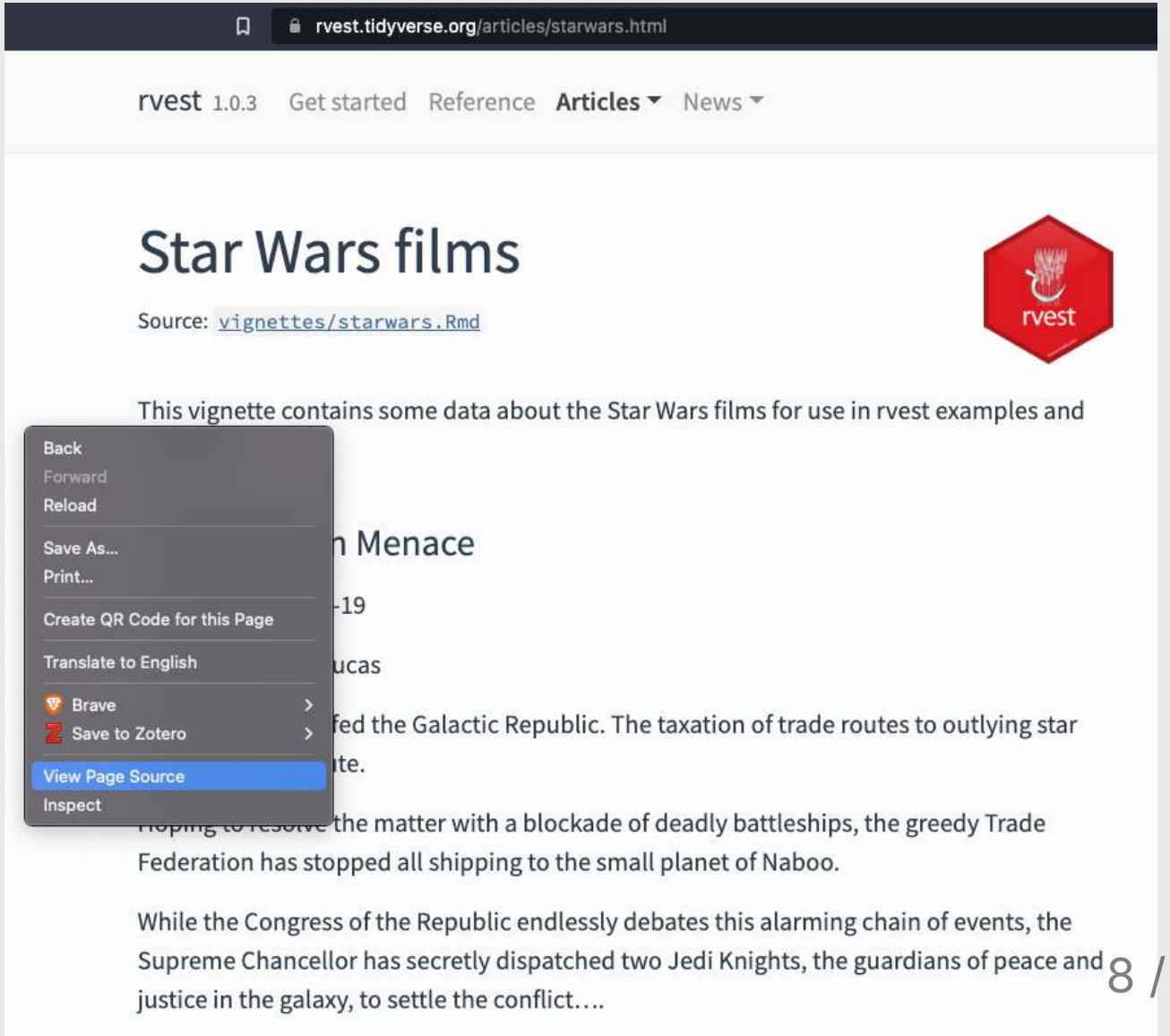
Attributes

- `id`: Element identifier, e.g. `<h1 id='first'>A heading</h1>`
- `class`: Styling class, e.g. `<h1 class='header'>A heading</h1>`

Quick example

- Go here
- Right-click, select "View Page Source"

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



Strategy: Use tags and classes to parse html

`source_code`

```
<html>
<head>
  <title>Page title</title>
</head>
<body>
  <h1 id='first'>A heading</h1>
  <p>Some text &amp;amp; <b>some bold text.</b><
  <img src='myimg.png' width='100' height='1
</body>
```

Use `{rvest}` package to parse html

```
library(rvest)

html <- read_html(source_code)

html %>%
  html_elements("h1")
```

```
#> {xml_nodeset (1)}
#> [1] <h1 id="first">A heading</h1>
```

Strategy: Use tags and classes to parse html

`source_code`

```
<html>
<head>
  <title>Page title</title>
</head>
<body>
  <h1 id='first'>A heading</h1>
  <p>Some text &amp;amp; <b>some bold text.</b><
  <img src='myimg.png' width='100' height='1
</body>
```

Use `{rvest}` package to parse html

```
library(rvest)

html <- read_html(source_code)

html %>%
  html_elements("p")
```

```
#> {xml_nodeset (1)}
#> [1] <p>Some text &amp;amp; <b>some bold text.
```

Dealing with multiple nodes (bullet list example)

`source_code`

```
<ul>
  <li><b>C-3PO</b> is a <i>droid</i> that we
  <li><b>R4-P17</b> is a <i>droid</i></li>
  <li><b>R2-D2</b> is a <i>droid</i> that we
  <li><b>Yoda</b> weighs <span class='weigh
</ul>
```

Rendered source code (in a browser)

- **C-3PO** is a *droid* that weighs 167 kg
- **R4-P17** is a *droid*
- **R2-D2** is a *droid* that weighs 96 kg
- **Yoda** weighs 66 kg

Dealing with multiple nodes (bullet list example)

`source_code`

```
<ul>
  <li><b>C-3PO</b> is a <i>droid</i> that we
  <li><b>R4-P17</b> is a <i>droid</i></li>
  <li><b>R2-D2</b> is a <i>droid</i> that we
  <li><b>Yoda</b> weighs <span class='weigh
</ul>
```

Use `{rvest}` package to parse html

```
library(rvest)

html <- read_html(source_code)

html %>%
  html_elements("li")
```

```
#> {xml_nodeset (4)}
#> [1] <li>\n<b>C-3PO</b> is a <i>droid</i>
#> [2] <li>\n<b>R4-P17</b> is a <i>droid</i>
#> [3] <li>\n<b>R2-D2</b> is a <i>droid</i>
#> [4] <li>\n<b>Yoda</b> weighs <span class=
```

Extract the names with "b"

`source_code`

```
<ul>
  <li><b>C-3PO</b> is a <i>droid</i> that we
  <li><b>R4-P17</b> is a <i>droid</i></li>
  <li><b>R2-D2</b> is a <i>droid</i> that we
  <li><b>Yoda</b> weighs <span class='weigh
</ul>
```

Use `{rvest}` package to parse html

```
library(rvest)

html <- read_html(source_code)

html %>%
  html_elements("li") %>%
  html_element("b")
```

```
#> {xml_nodeset (4)}
#> [1] <b>C-3PO</b>
#> [2] <b>R4-P17</b>
#> [3] <b>R2-D2</b>
#> [4] <b>Yoda</b>
```

Extract the *text* with `html_text2()`

`source_code`

```
<ul>
  <li><b>C-3PO</b> is a <i>droid</i> that we
  <li><b>R4-P17</b> is a <i>droid</i></li>
  <li><b>R2-D2</b> is a <i>droid</i> that we
  <li><b>Yoda</b> weighs <span class='weigh
</ul>
```

Use `{rvest}` package to parse html

```
library(rvest)

html <- read_html(source_code)

html %>%
  html_elements("li") %>%
  html_element("b") %>%
  html_text2()
```

```
#> [1] "C-3PO"  "R4-P17" "R2-D2"  "Yoda"
```

Extract the weights using ".weight" class

`source_code`

```
<ul>
  <li><b>C-3PO</b> is a <i>droid</i> that we
  <li><b>R4-P17</b> is a <i>droid</i></li>
  <li><b>R2-D2</b> is a <i>droid</i> that we
  <li><b>Yoda</b> weighs <span class='weigh
</ul>
```

Use {rvest} package to parse html

```
library(rvest)

html <- read_html(source_code)

html %>%
  html_elements("li") %>%
  html_element(".weight") %>%
  html_text2()
```

```
#> [1] "167 kg" NA       "96 kg"  "66 kg"
```

Putting it together in a data frame

```
library(rvest)

items <- read_html(source_code) %>%
  html_elements("li")
```

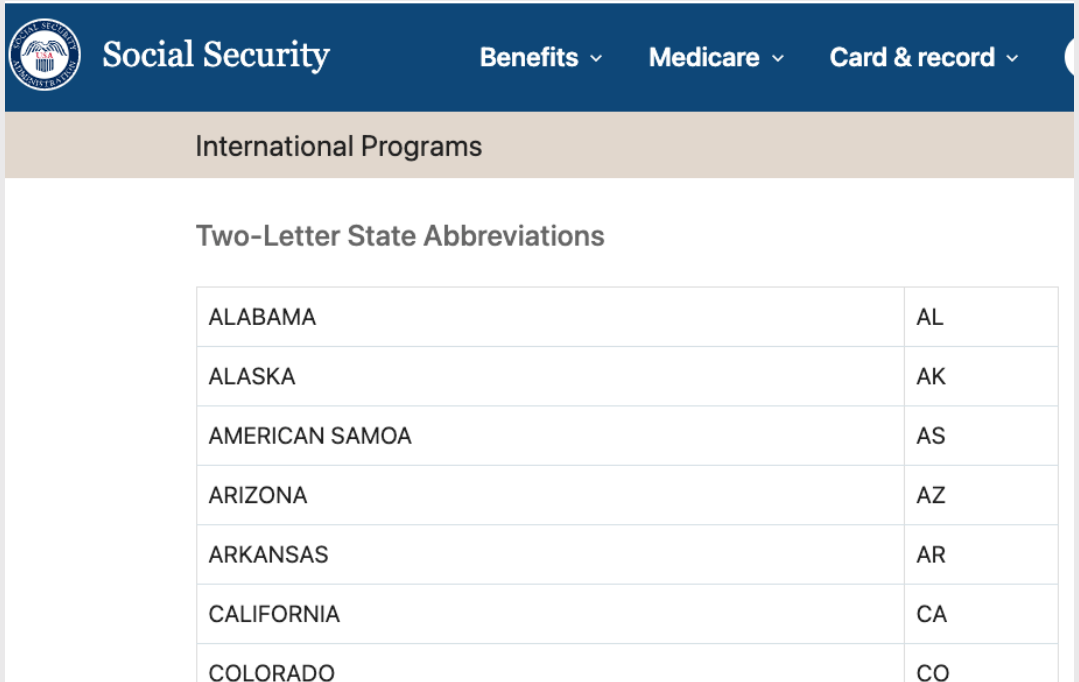
```
data <- tibble(
  name = items %>%
    html_element("b") %>%
    html_text2(),
  weight = items %>%
    html_element(".weight") %>%
    html_text2() %>%
    parse_number()
)

data
```

```
#> # A tibble: 4 × 2
#>   name   weight
#>   <chr>   <dbl>
#> 1 C-3PO     167
#> 2 R4-P17     NA
#> 3 R2-D2      96
#> 4 Yoda       66
```

`html_table()` is awesome (if the site uses an HTML table)

Some pages have HTML tables in the source code, e.g.

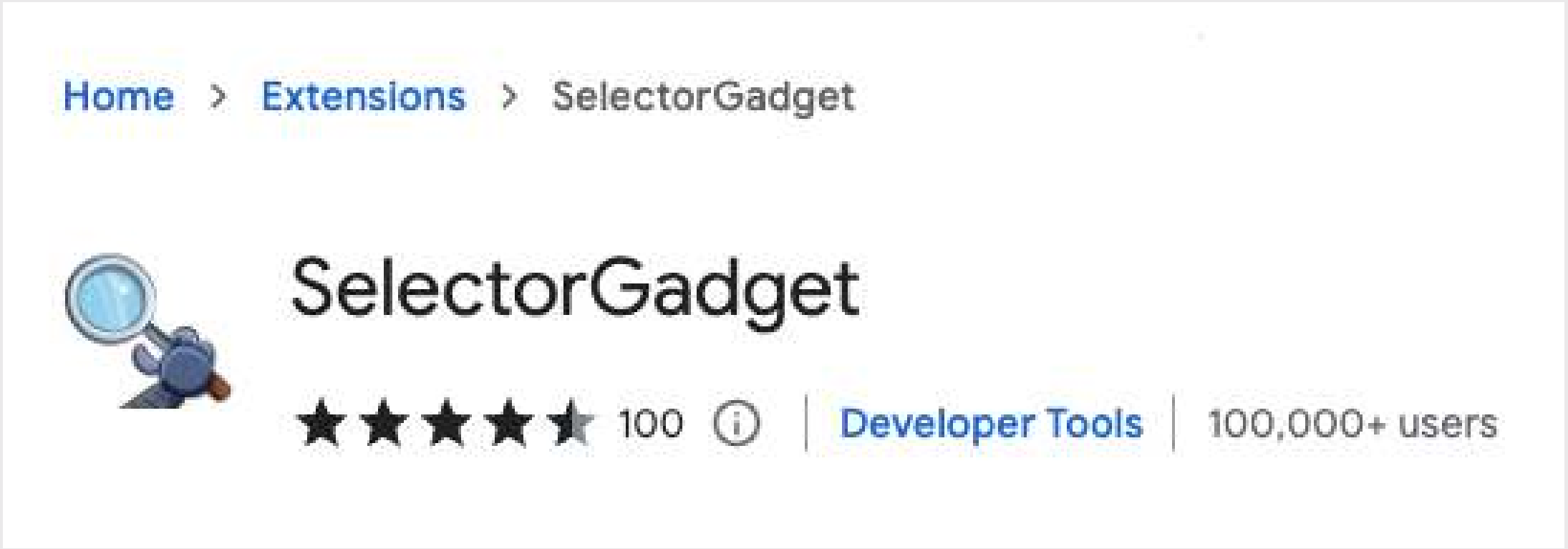
https://www.ssa.gov/international/coc-docs/states.html



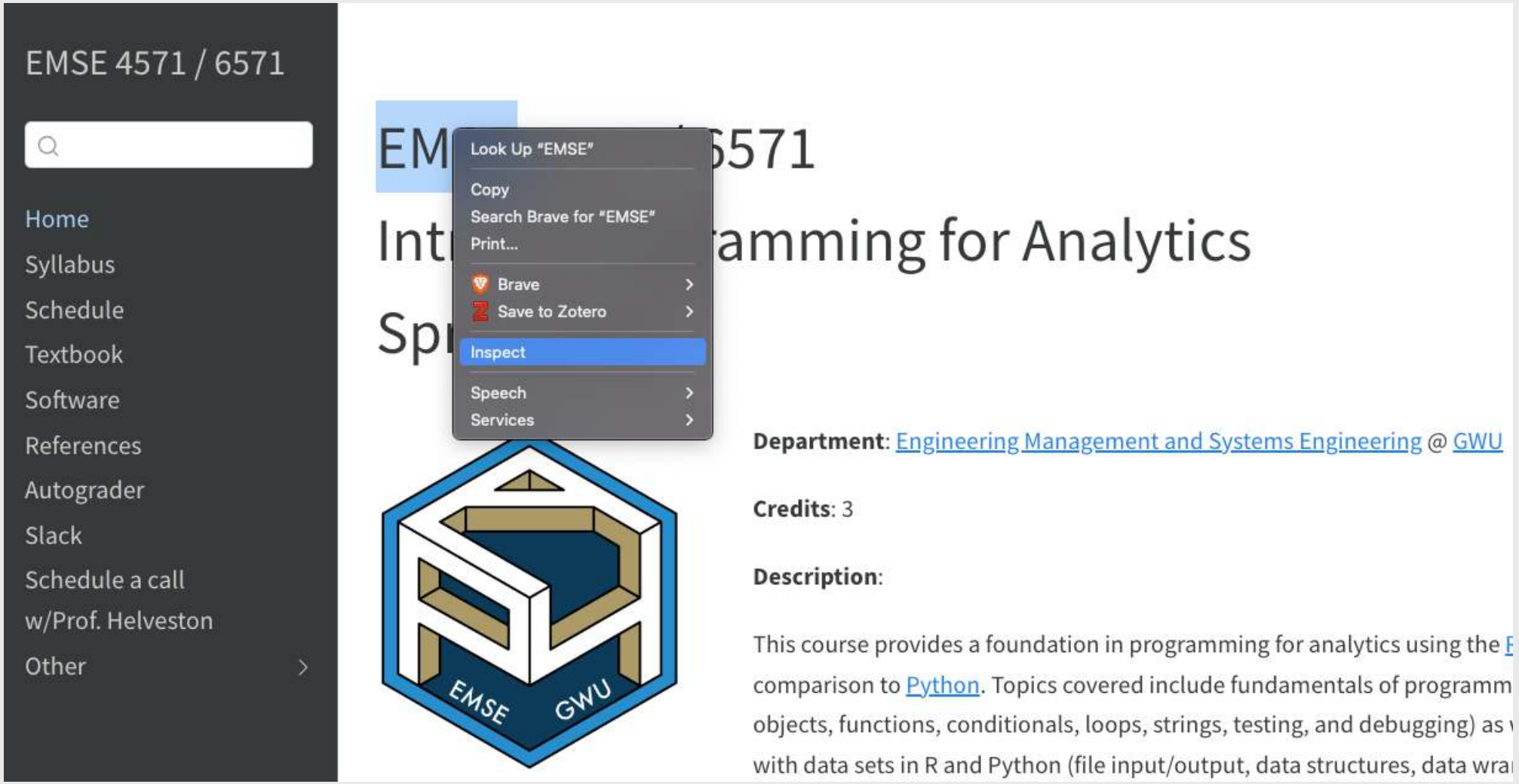

```
url <- "https://www.ssa.gov/international/c
df <- read_html(url) %>%
  html_table()

df
```

```
#> [[1]]
#> # A tibble: 56 × 2
#>    X1                   X2
#>    <chr>                <chr>
#>  1 ALABAMA              AL
#>  2 ALASKA               AK
#>  3 AMERICAN SAMOA       AS
#>  4 ARIZONA              AZ
#>  5 ARKANSAS             AR
#>  6 CALIFORNIA           CA
#>  7 COLORADO             CO
#>  8 CONNECTICUT          CT
#>  9 DELAWARE             DE
#> 10 DISTRICT OF COLUMBIA DC
#> # ℹ 46 more rows
```

Find elements with SelectorGadget



Find elements with "inspect"



Your turn

15:00

Scrape data on famous quotes from http://quotes.toscrape.com/

Your resulting data frame should have these fields:

- `quote`: The quote
- `author`: The author of the quote
- `about_url`: The url to the "about" page

```
#> Rows: 10
#> Columns: 3
#> $ quote     <chr> "“The world as we have created it is a process of our thinking. It ca
#> $ author    <chr> "Albert Einstein", "J.K. Rowling", "Albert Einstein", "Jane Austen",
#> $ about_url <chr> "http://quotes.toscrape.com/author/Albert-Einstein", "http://quotes.t
```

Week 12: *Webscraping*

1. HTML basics

1. Scraping static pages

2. Scraping multiple pages

BREAK

3. Using APIs

What if there is more than one page to scrape?

Iterate!

Iterative scraping!

1. Find the url pattern

2. Scrape one page

3. Iteratively scrape each page with `map_df()`

1. Find the url pattern

Example: http://quotes.toscrape.com/

url to page 2: http://quotes.toscrape.com/page/2

Pattern: `http://quotes.toscrape.com/page/` + `#`

I can *build* the url to any page with `paste()`:

```
root <- "http://quotes.toscrape.com/page/"
page <- 3
url <- paste(root, page, sep = "")
url
```

```
#> [1] "http://quotes.toscrape.com/page/3"
```

2. Scrape one page

Build the url to a single page:

```
root <- "http://quotes.toscrape.com/page/"
page <- 3
url <- paste(root, page, sep = "")
url
```

```
#> [1] "http://quotes.toscrape.com/page/3"
```

Scrape the data on that page:

```
quote_nodes <- read_html(url) %>%
    html_elements(".quote")
df <- tibble(
    quote = quote_nodes %>%
        html_element(".text") %>%
        html_text(),
    author = quote_nodes %>%
        html_element(".author") %>%
        html_text(),
    about_url = quote_nodes %>%
        html_element("a") %>%
        html_attr("href")
) %>%
    mutate(about_url = paste0(url, about_ur
```

3. Iteratively scrape each page with `map_df()`

Make a function to get data from a page:

```
get_page_data <- function(page) {
    root <- "http://quotes.toscrape.com/page/"
    url <- paste(root, page, sep = "")
    quote_nodes <- read_html(url) %>%
        html_elements(".quote")
    df <- tibble(
        quote = quote_nodes %>%
            html_element(".text") %>%
            html_text(),
        author = quote_nodes %>%
            html_element(".author") %>%
            html_text(),
        about_url = quote_nodes %>%
            html_element("a") %>%
            html_attr("href")
    ) %>%
        mutate(about_url = paste0(url, about_url))
    return(df)
}
```

Iterate with `map_df()`:

```
pages <- 1:10

df <- map_df(pages, \(x) get_page_data(x))
```

Your turn

15:00

Template code is provided to scrape data on F1 drivers for the 2025 season from https://www.formula1.com/en/results/2025/drivers

Your job is to extend it to scrape the data from seasons 2010 to 2025.

Your final dataset should look like this:

```
#> # A tibble: 6 × 7
#>    year position driver          nationality team            points abb
#>   <dbl>    <int> <chr>           <chr>       <chr>            <int> <chr>
#> 1  2025        1 Lando Norris    GBR         McLaren            423 NOR
#> 2  2025        2 Max Verstappen  NED         Red Bull Racing    421 VER
#> 3  2025        3 Oscar Piastri   AUS         McLaren            410 PIA
#> 4  2025        4 George Russell  GBR         Mercedes           319 RUS
#> 5  2025        5 Charles Leclerc MON         Ferrari            242 LEC
#> 6  2025        6 Lewis Hamilton  GBR         Ferrari            156 HAM
```

Intermission

05:00

Week 12: *Webscraping*

1. HTML basics

1. Scraping static pages

2. Scraping multiple pages

BREAK

3. Using APIs

Hopefully you won't need to scrape

Before you start scraping, ask...

1. Is there a formatted dataset I can download?
(e.g. see this page)

2. Is there an API I can use?

Application Programming Interface (API)

A set of defined rules that enable different applications to communicate (and pass data) with each other

Basically, APIs make it easier to get data from the web

APIs use the `url` to "ask" a website for data

Example: Stock market prices from https://www.alphavantage.co/

API Request:
https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={symbol}&apikey={api_key}&datatype=csv

- `function`: The time series of your choice
- `symbol`: Stock price symbol (e.g. `NFLX` = Netflix)
- `apikey`: Your API key (have to register to get one)
- `datatype`: `csv` or `json`

Set up your `.env` file

1. Make a `.env` file:

```
# Create an empty .env file
file.create(".env")
```

2. Open the file to edit:

```
# Create an empty .env file
file.edit(".env")
```

Store your API key

3. Register for a key here: https://www.alphavantage.co/support/#api-key

4. Store your key in the file:

```
ALPHAVANTAGE_API_KEY=ZF33JCWPWWQDX4LW
```

Get your API key

5. Load your `.env` variables:

```
dotenv::load_dot_env()
```

6. Retrieve your key:

```
api_key <- Sys.getenv("ALPHAVANTAGE_API_KEY")
```

Using your key to get data

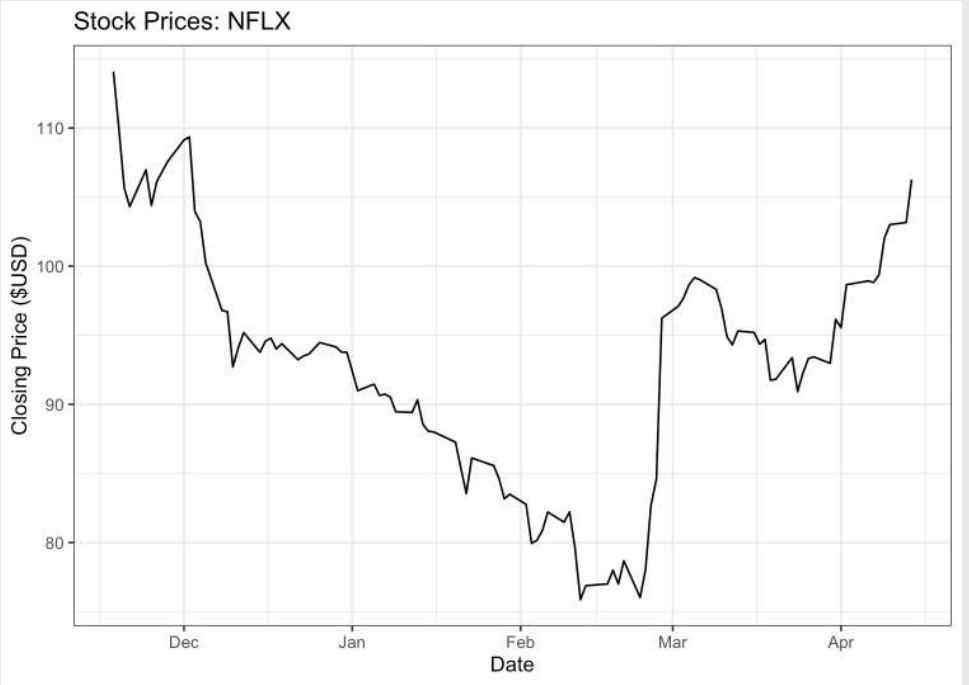
```
api_key <- Sys.getenv("ALPHAVANTAGE_API_KEY")
symbol <- "NFLX" # Netflix

# Build the url data request

url <- paste0(
  "https://www.alphavantage.co/query",
  "?function=TIME_SERIES_DAILY",
  "&symbol=", symbol,
  "&apikey=", api_key,
  "&datatype=csv"
)

# Read in the data

df <- readr::read_csv(url)
```

```
glimpse(df)
```

```
#> Rows: 100
#> Columns: 6
#> $ timestamp <date> 2026-04-14, 2026-
#> $ open      <dbl> 103.120, 103.032,
#> $ high      <dbl> 106.5700, 103.6650
#> $ low       <dbl> 103.0400, 102.0600
#> $ close     <dbl> 106.28, 103.16, 10
#> $ volume    <dbl> 40534865, 26042804
```

Using your key to get data

```
df %>%
    ggplot() +
    geom_line(
      aes(
        x = timestamp,
        y = close
      )
    ) +
    theme_bw() +
    labs(
        x = "Date",
        y = "Closing Price ($USD)",
        title = paste0("Stock Prices: ", symbol)
    )
```



Want something else?

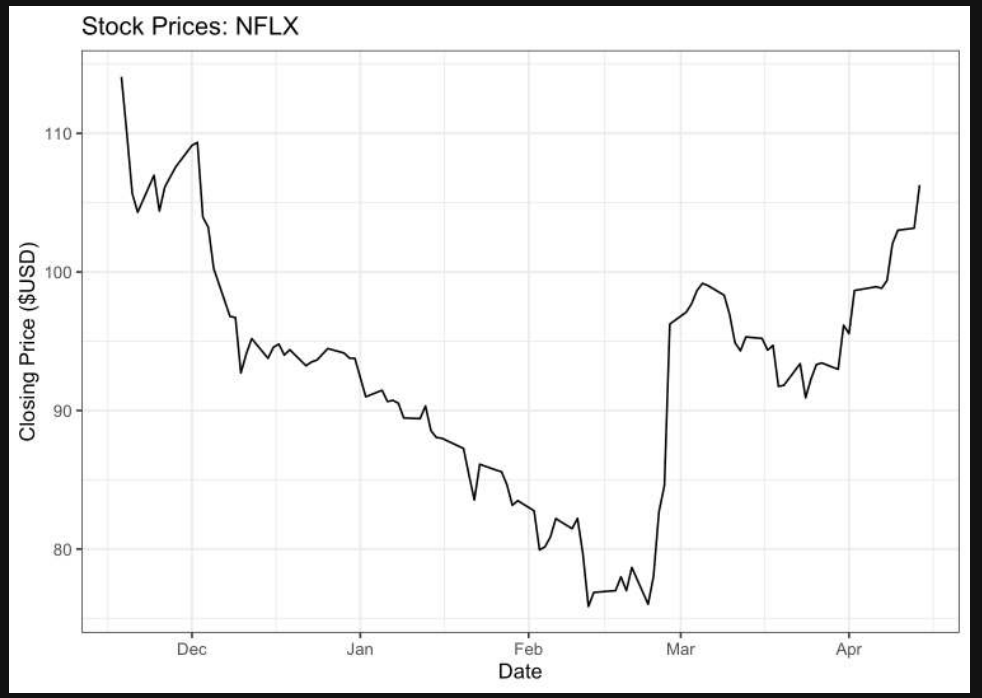
Read the docs!

https://www.alphavantage.co/documentation/

Your turn

20:00

1. Make your .env file:
   `file.create(".env")`
2. Edit your .env file:
   `file.edit(".env")`
3. Register for a key:
   https://www.alphavantage.co/support/#api-key
4. Store your key, e.g.
   `ALPHAVANTAGE_API_KEY=ZF33JCWPWWQDX4LW`
5. Load your .env file: `dotenv::load_dot_env()`
6. Load your API key:
   ```
   api_key <-
   Sys.getenv("ALPHAVANTAGE_API_KEY")
   ```
7. Build the url to request historical stock prices for a stock of your choice
8. Read in the data, then make this a stock plot with ggplot



HW12