Week 4: Web scraping

31/01/24

Introduction

Today we will be extracting some useful data from websites. There's a bunch of different ways to web-scrape, but we'll be exploring using the `rvest` package in R, that helps you to deal with parsing html.

Why is web scraping useful? If our research involves getting data from a website that isn't already in a easily downloadable form, it improves the reproducibility of our research. Once you get a scraper working, it's less prone to human error than copy-pasting, for example, and much easier for someone else to see what you did.

A note on responsibility

Seven principles for web-scraping responsibly:

1. Try to use an API.
2. Check robots.txt. (e.g. https://www.utoronto.ca/robots.txt)
3. Slow down (why not only visit the website once a minute if you can just run your data collection in the background while you're doing other things?).
4. Consider the timing (if it's a retailer then why not set your script to run overnight?).
5. Only scrape once (save the data as you go and monitor where you are up to).
6. Don't republish the data you scraped (cf datasets that create based off it).
7. Take ownership (add contact details to your scripts, don't hide behind VPNs, etc)

Extracting the Billboard top 100 songs list

We're going to scrape the Billboard top 100 songs list from september last year: https://www.billboard.com/charts/hot-100/2023-09-23/. While the data are nicely presented, there's no nice way of downloading the data for each year as a csv or similar form. So

let's use `rvest` to extract the data. We'll also load in `janitor` to clean up column names etc later on.

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.3     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.2     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

```
library(rvest)
```

```
Attaching package: 'rvest'

The following object is masked from 'package:readr':

    guess_encoding
```

```
library(janitor)
```

```
Attaching package: 'janitor'

The following objects are masked from 'package:stats':

    chisq.test, fisher.test
```

Firstly, read the html from the webpage:

```
link <- "https://www.billboard.com/charts/hot-100/2023-09-23/"
billboard_page <- read_html(link)
```

We want to extract information from the (interactive) table. To find where this was in the html, I made use of the 'Inspect' option when you right-click in Chrome. This was a relatively straightforward way of find the right class for each table item. Once I had that, I pulled all the html nodes of that class, converted to text, removed all "" (tabs), converted to a tibble and then separated the values. The final line of code gets rid of all empty columns.

```
# class of table objects
class <- "o-chart-results-list-row-container"

rough_table <- billboard_page |>
  html_nodes(xpath = paste0("//div[@class = '",class,"']"))|> # pull out table
  html_text() |> # convert to text
  str_remove_all("\t") |> # remove tabs
  as_tibble() |> # convert to tibble
  separate(value, into = paste0(1:1000), sep = "\n") |> # separate into columns
  select_if(function(x) !(all(is.na(x)) | all(x==""))) # remove empty columns
```

```
Warning: Expected 1000 pieces. Missing pieces filled with `NA` in 100 rows [1, 2, 3, 4,
5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...].
```

We're almost there, but some of the values are mis-aligned, because some songs have an additional label of 'new' or a 're-entry'. To tidy this up, I found it easiest to separate these songs out, clean up the table, and then join them back together.

```
# new songs tidying up
rough_new <- rough_table |>
  filter(`12`=="NEW")|>
  select_if(function(x) !(all(is.na(x)) | all(x==""))) |>
  rename(position = `5`,
         song_name = `43`,
         artist = `49`,
         last_week = `65`,
         peak_position = `72`,
         weeks_in_chart = `79`
         ) |>
  select(position, song_name, artist, last_week, peak_position, weeks_in_chart) |>
  mutate(label = "new")

# re-entry songs tidying up
rough_re <- rough_table |>
  filter(`12` == "RE-") |>
```

```
  rename(position = `5`,
         song_name = `45`,
         artist = `51`,
         last_week = `67`,
         peak_position = `74`,
         weeks_in_chart = `81`
         ) |>
  select(position, song_name, artist, last_week, peak_position, weeks_in_chart) |>
  mutate(label = "re-entry")

# everything else tidying up
rough_everything_else <- rough_table |>
  filter(`12` == "") |>
  mutate(`59` = ifelse(`59`=="", `61`, `59`),
         `66` = ifelse(`66`=="", `68`, `66`),
         `73` = ifelse(`73`=="", `75`, `73`))|>
  rename(position = `5`,
         song_name = `39`,
         artist = `45`,
         last_week = `59`,
         peak_position = `66`,
         weeks_in_chart = `73`
         ) |>
  select(position, song_name, artist, last_week, peak_position, weeks_in_chart) |>
  mutate(label = "NA")

# bind them all together and sort
clean_table <- bind_rows(rough_everything_else,
                         rough_new, rough_re) |>
  mutate(position = as.numeric(position)) |>
  arrange(position)

clean_table
```

```
# A tibble: 100 x 7
   position song_name        artist last_week peak_position weeks_in_chart label
      <dbl> <chr>            <chr>  <chr>     <chr>         <chr>          <chr>
 1        1 Vampire          Olivi~ 9         1             11             NA
 2        2 Paint The Town ~ Doja ~ 1         1             6              NA
 3        3 I Remember Ever~ Zach ~ 2         1             3              NA
 4        4 Fast Car         Luke ~ 3         2             25             NA
 5        5 Cruel Summer     Taylo~ 4         3             19             NA
```

```
 6        6 Last Night       Morga~ 5         1              33             NA
 7        7 Bad Idea Right?  Olivi~ 26        7              5              NA
 8        8 Snooze           SZA    7         7              40             NA
 9        9 Fukumean         Gunna  8         4              13             NA
10       10 Dance The Night  Dua L~ 6         6              16             NA
# i 90 more rows
```

Install rstan and brms

We will be using the packages `rstan` and `brms` from next week. Please install these. Here's some instructions:

- https://github.com/paul-buerkner/brms
- https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started

In most cases it will be straightforward and may not need much more than `install.packages()`, but you might run into issues. Every Stan update seems to cause problems for different OS.

To make sure it works, run the following code:

```
library(brms)
```

```
Loading required package: Rcpp
```

```
Loading 'brms' package (version 2.18.0). Useful instructions
can be found by typing help('brms'). A more detailed introduction
to the package is available through vignette('brms_overview').
```

```
Attaching package: 'brms'
```

```
The following object is masked from 'package:stats':

    ar
```

```
x <- rnorm(100)
y <- 1 + 2*x + rnorm(100)
d <- tibble(x = x, y= y)

mod <- brm(y~x, data = d)
```

```
Compiling Stan program...

Start sampling


SAMPLING FOR MODEL '2b28b4a8f82b6598cd1c379da23e9b07' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 2.1e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.21 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:
Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0.017523 seconds (Warm-up)
Chain 1:                0.018954 seconds (Sampling)
Chain 1:                0.036477 seconds (Total)
Chain 1:

SAMPLING FOR MODEL '2b28b4a8f82b6598cd1c379da23e9b07' NOW (CHAIN 2).
Chain 2:
Chain 2: Gradient evaluation took 1e-05 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.1 seconds.
Chain 2: Adjust your expectations accordingly!
Chain 2:
Chain 2:
Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
```

```
Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 2:
Chain 2:  Elapsed Time: 0.017689 seconds (Warm-up)
Chain 2:                0.018405 seconds (Sampling)
Chain 2:                0.036094 seconds (Total)
Chain 2:

SAMPLING FOR MODEL '2b28b4a8f82b6598cd1c379da23e9b07' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 4e-06 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
Chain 3: Adjust your expectations accordingly!
Chain 3:
Chain 3:
Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 3:
Chain 3:  Elapsed Time: 0.018174 seconds (Warm-up)
Chain 3:                0.016471 seconds (Sampling)
Chain 3:                0.034645 seconds (Total)
Chain 3:

SAMPLING FOR MODEL '2b28b4a8f82b6598cd1c379da23e9b07' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 5e-06 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
Chain 4: Adjust your expectations accordingly!
```

```
Chain 4:
Chain 4:
Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 4:
Chain 4:  Elapsed Time: 0.018788 seconds (Warm-up)
Chain 4:                0.019158 seconds (Sampling)
Chain 4:                0.037946 seconds (Total)
Chain 4:
```

```
summary(mod)
```

```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: y ~ x
   Data: d (Number of observations: 100)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Population-Level Effects:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept     1.07      0.10     0.86     1.27 1.00     3628     3009
x             2.01      0.10     1.81     2.22 1.00     3240     2759

Family Specific Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma     1.02      0.07     0.89     1.18 1.00     3824     3416

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```