Lab 4

SHAOHAN CHANG

02-05-2023

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 data on opioid prescriptions from CDC

We're going to grab some data on opioid prescription rates from the CDC website. While the data are nicely presented and mapped, 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)
library(rvest)
library(janitor)
```

Getting the data for 2008

Have a look at the website at the url below. It shows a map of state prescription rates in 2008. Let's read in the html of this page.

```
cdcpage <- "https://www.cdc.gov/drugoverdose/rxrate-maps/state2008.html"
cdc <- read_html(cdcpage)
```

Note that it has two main parts, a head and body. For the majority of use cases, you will probably be interested in the body. You can select a node using `html_node()` and then see its child nodes using `html_children()`.

```
body_nodes <- cdc %>%
 html_node("body") %>%
 html_children()
```

Inspecting elements of a website

The above is still fairly impenetrable. But we can get hints from the website itself. Using Chrome (or Firefox) you can highlight a part of the website of interest (say, 'Alabama'), right click and choose 'Inspect'. That gives you info on the underlying html of the webpage on the right hand side. Alternatively, and probably easier to find what we want, right click on the webpage and choose View Page Source. This opens a new window with all the html. Do a search for the world 'Alabama'. Now we can see the code for the table. We can see that the data we want are all within `tr`. So let's extract those nodes:

```
cdc %>%
  html_nodes("tr")
```

```
## {xml_nodeset (52)}
##  [1] <tr>\n<th>State</th>\n<th>State Abbreviation</th>\n<th>Opioid Dispensing ...
##  [2] <tr>\n<td>Alabama</td>\n<td>AL</td>\n<td>126.1</td>\n</tr>\n
##  [3] <tr>\n<td>Alaska</td>\n<td>AK</td>\n<td>68.5</td>\n</tr>\n
##  [4] <tr>\n<td>Arizona</td>\n<td>AZ</td>\n<td>80.9</td>\n</tr>\n
##  [5] <tr>\n<td>Arkansas</td>\n<td>AR</td>\n<td>112.1</td>\n</tr>\n
##  [6] <tr>\n<td>California</td>\n<td>CA</td>\n<td>55.1</td>\n</tr>\n
##  [7] <tr>\n<td>Colorado</td>\n<td>CO</td>\n<td>67.7</td>\n</tr>\n
##  [8] <tr>\n<td>Connecticut</td>\n<td>CT</td>\n<td>68.7</td>\n</tr>\n
##  [9] <tr>\n<td>Delaware</td>\n<td>DE</td>\n<td>95.4</td>\n</tr>\n
## [10] <tr>\n<td>District of Columbia</td>\n<td>DC</td>\n<td>34.5</td>\n</tr>\n
## [11] <tr>\n<td>Florida</td>\n<td>FL</td>\n<td>84.3</td>\n</tr>\n
## [12] <tr>\n<td>Georgia</td>\n<td>GA</td>\n<td>86.3</td>\n</tr>\n
## [13] <tr>\n<td>Hawaii</td>\n<td>HI</td>\n<td>46.6</td>\n</tr>\n
## [14] <tr>\n<td>Idaho</td>\n<td>ID</td>\n<td>82.7</td>\n</tr>\n
## [15] <tr>\n<td>Illinois</td>\n<td>IL</td>\n<td>60.2</td>\n</tr>\n
## [16] <tr>\n<td>Indiana</td>\n<td>IN</td>\n<td>103.3</td>\n</tr>\n
## [17] <tr>\n<td>Iowa</td>\n<td>IA</td>\n<td>59.1</td>\n</tr>\n
## [18] <tr>\n<td>Kansas</td>\n<td>KS</td>\n<td>82.7</td>\n</tr>\n
## [19] <tr>\n<td>Kentucky</td>\n<td>KY</td>\n<td>136.6</td>\n</tr>\n
## [20] <tr>\n<td>Louisiana</td>\n<td>LA</td>\n<td>113.7</td>\n</tr>\n
## ...
```

Great, now we're getting somewhere. We only want the text, not the html rubbish, so let's extract that:

```
table_text <- cdc %>%
  html_nodes("tr") %>%
  html_text()
```

This is almost useful! Turning it into a tibble and using `separate` to get the variables into separate columns gets us almost there:

```
rough_table <- table_text %>%
  as_tibble() %>%
  separate(value, into = c("state", "abbrev", "rate"), sep = "\n", extra = "drop")
```

Now we can just divert to our standard tidyverse cleaning skills (`janitor` functions help here) to tidy it up:

```
d_prescriptions <- rough_table %>%
  janitor::row_to_names(1) %>%
  janitor::clean_names() %>%
```

```
  rename(prescribing_rate = opioid_dispensing_rate_per_100) %>%
  mutate(prescribing_rate = as.numeric(prescribing_rate))
```

Now we have clean data for 2008!

Take-aways

This example showed you how to extract a particular table from a particular website. The take-away is to inspect the page html, find where what you want is hiding, and then use the tools in `rvest` (`html_nodes()` and `html_text()` particularly useful) to extract it.

Question 1

Add a year column to `d_prescriptions`.

```
library(dplyr)
d_prescriptions%>%
   mutate(year = 2008)
```

```
## # A tibble: 51 x 4
##    state                state_abbreviation prescribing_rate  year
##    <chr>                <chr>                         <dbl> <dbl>
##  1 Alabama              AL                            126.   2008
##  2 Alaska               AK                             68.5  2008
##  3 Arizona              AZ                             80.9  2008
##  4 Arkansas             AR                            112.   2008
##  5 California           CA                             55.1  2008
##  6 Colorado             CO                             67.7  2008
##  7 Connecticut          CT                             68.7  2008
##  8 Delaware             DE                             95.4  2008
##  9 District of Columbia DC                             34.5  2008
## 10 Florida              FL                             84.3  2008
## # ... with 41 more rows
```

Getting all the other years

Now I want you to get data for 2008-2019 and save it into one big tibble. If you go to cdc.gov/drugoverdose/rxrate-maps/index.html, on the right hand side there's hyperlinks to all the years under "U.S. State Opioid Dispensing Rate Maps".

Click on 2009. Look at the url. Confirm that it's exactly the same format as the url for 2008, except the year has changed. This is useful, because we can just loop through in an automated way, changing the year as we go.

Question 2

Make a vector of the urls for each year, storing them as strings.

```
urls <- NULL

a=2008:2019
b="https://www.cdc.gov/drugoverdose/rxrate-maps/state"
c=".html"

for(i in 1:12){
```

```
  urls[i]=paste(b,a[i],c, sep = "", collapse = "")
}
```

Question 3

Extract the prescriptions data for the years 2008-2019, and store in the one tibble. Make sure you have a column for state, state abbreviation, prescription rate and year. Note if you are looping over years/urls (which is probably the easiest thing to do), it's good practice to include a `Sys.sleep(1)` at the end of your loop, so R waits for a second before trying again.

Plot prescriptions by state over time.

```
library(tidyr)
library(dplyr)
tibble=list()

for (k in 1:12) {
  cdc <- read_html(urls[k])

  body_nodes <- cdc %>%
    html_node("body") %>%
    html_children()

  table_text <- cdc %>%
    html_nodes("tr") %>%
    html_text()

  rough_table <- table_text %>%
    as_tibble() %>%
    separate(value, into = c("state", "abbrev","rate"), sep = "\n", extra = "drop")

  d_prescriptions <- rough_table %>%
    janitor::row_to_names(1) %>%
    janitor::clean_names() %>%
    rename(prescribing_rate = opioid_dispensing_rate_per_100) %>%
    mutate(prescribing_rate =as.numeric(prescribing_rate))

  d_prescriptions=d_prescriptions%>%mutate(year=a[k])
  tibble[[k]]=d_prescriptions



  Sys.sleep(1)
}


data=tibble[[1]]
for (l in 2:12) {
  data=data%>%bind_rows(tibble[[l]])

}

data$state_abbreviation[460:615]=data$abbreviation[460:615]
```

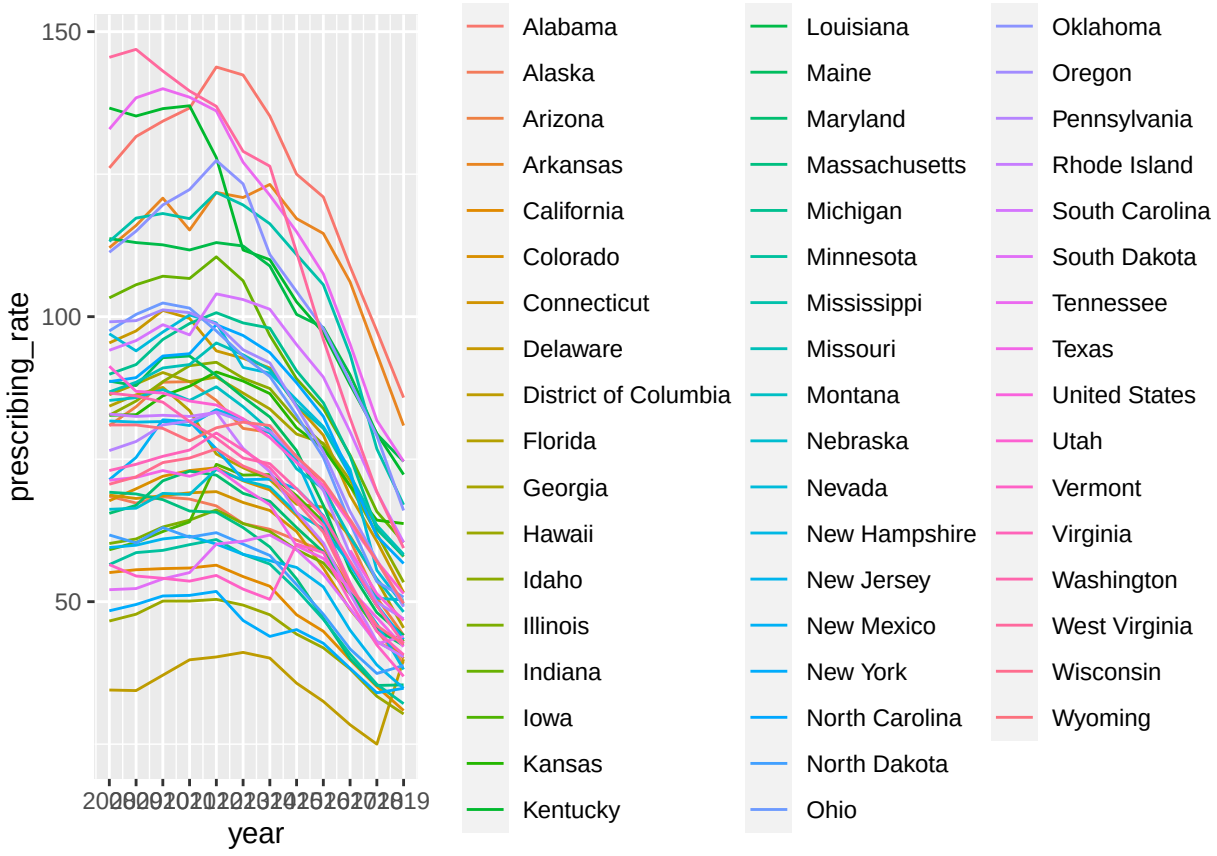
```r
data=data[,-5]



#plot
library(ggplot2)
unique(data$state)
```

```
##  [1] "Alabama"              "Alaska"               "Arizona"
##  [4] "Arkansas"             "California"           "Colorado"
##  [7] "Connecticut"          "Delaware"             "District of Columbia"
## [10] "Florida"              "Georgia"              "Hawaii"
## [13] "Idaho"                "Illinois"             "Indiana"
## [16] "Iowa"                 "Kansas"               "Kentucky"
## [19] "Louisiana"            "Maine"                "Maryland"
## [22] "Massachusetts"        "Michigan"             "Minnesota"
## [25] "Mississippi"          "Missouri"             "Montana"
## [28] "Nebraska"             "Nevada"               "New Hampshire"
## [31] "New Jersey"           "New Mexico"           "New York"
## [34] "North Carolina"       "North Dakota"         "Ohio"
## [37] "Oklahoma"             "Oregon"               "Pennsylvania"
## [40] "Rhode Island"         "South Carolina"       "South Dakota"
## [43] "Tennessee"            "Texas"                "Utah"
## [46] "Vermont"              "Virginia"             "Washington"
## [49] "West Virginia"        "Wisconsin"            "Wyoming"
## [52] "United States"
```

```r
data%>%group_by(state)%>%
  ggplot()+geom_line(aes(x=year,y=prescribing_rate,colour=state))+scale_x_continuous(breaks=seq(2008,20
```



Question 4: 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(devtools)
install_github("paul-buerkner/brms")
library(brms)

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

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

```
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 2.5e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.25 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: 0.012 seconds (Warm-up)
## Chain 1:                0.012 seconds (Sampling)
## Chain 1:                0.024 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 4e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.04 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: 0.012 seconds (Warm-up)
## Chain 2:                0.012 seconds (Sampling)
## Chain 2:                0.024 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 7e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.07 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: 0.012 seconds (Warm-up)
## Chain 3:                0.012 seconds (Sampling)
## Chain 3:                0.024 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 6e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.06 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: 0.012 seconds (Warm-up)
## Chain 4:                0.012 seconds (Sampling)
## Chain 4:                0.024 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     1.02      0.09     0.84     1.20 1.00     3742     2882
## x             2.02      0.08     1.86     2.17 1.00     3632     2652
```

```
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     0.89      0.06     0.78     1.03 1.00     3903     2936
##
## 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).
```