# Web scraping

Monica Alexander

February 5 2020

## Contents



# 1 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.

## 1.1 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)

# 2 Extracting data on opioid prescriptions from CDC

In Assignment 1 the `opioids` dataset contained data by state and year on the opioid prescription rate. I grabbed this data 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)
library(geofacet)
```

## 2.1 Getting the data for 2008

Have a look at the website at the url below. It shows a map and (if you scroll down) a table of state prescription rates in 2008. Let's read in the html of this page.

```
cdcpage <- "https://www.cdc.gov/drugoverdose/maps/rxstate2008.html"
cdc <- read_html(cdcpage)
cdc
```

```
## {html_document}
## <html lang="en-us" class="theme-purple">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
## [2] <body class="no-js">\r\n\t<div id="skipmenu">\r\n\t\t<a class="skippy sr- ...
```

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()
body_nodes
```

```
## {xml_nodeset (19)}
##  [1] <div id="skipmenu">\r\n\t\t<a class="skippy sr-only-focusable" href="#co ...
##  [2] <div class="container-fluid header-wrapper">\r\n\t\t\t\t<div class="cont ...
##  [3] <div class="container-fluid site-title">\r\n\t\t\t\t<div class="containe ...
##  [4] <nav role="navigation" aria-label="Mobile Nav" id="mobilenav" class="sti ...
##  [5] <div class="container breadcrumb-share">\r\n\t\t\t\t\t\t<div class="d- ...
##  [6] <div class="container-fluid feature-area">\r\n\t\t\t<div class="containe ...
##  [7] <div class="container d-flex flex-wrap body-wrapper bg-white">\r\n\t\t\t ...
##  [8] <footer class="" role="contentinfo" aria-label="Footer"><div class="cont ...
##  [9] <nav role="navigation" aria-label="Social Media" class="d-lg-none w-100  ...
## [10] <div id="metrics">\r\n\t<!-- Google DAP inclusion -->\r\n\t<script id="_ ...
## [11] <script src="/TemplatePackage/contrib/libs/jquery/latest/jquery.min.js"> ...
## [12] <script src="/TemplatePackage/contrib/libs/bootstrap/latest/js/bootstrap ...
## [13] <script src="/TemplatePackage/contrib/libs/cdc/ab/4.0.0/ab.js"></script>
## [14] <script src="/TemplatePackage/4.0/assets/js/app.min.js?v=21-02-03T18:34: ...
## [15] <svg viewbox="0 0 40 40" class="d-none"><radialgradient id="svg_ig_1" cx ...
## [16] <svg id="multicolor_icons" style="display:none" xmlns="http://www.w3.org ...
## [17] <script>\r\n    \r\n    <B>Error processing SSI file</B><BR>\r\n    \r\n ...
## [18] <script>\r\n\twindow.shortTitle = "U.S. State Opioid Dispensing Rates, 2 ...
## [19] <div class="modal fade" id="cdcExtLink" tabindex="-1" role="dialog" aria ...
```

We can keep going down to see the nodes within the nodes, just by piping again:

```
body_nodes[[7]] %>%
  html_children() %>%
  html_children() %>% `[[`(3) # pull the third element
```

```
## {html_node}
```

```
## <div class="row">
## [1] <div class="col content content-fullwidth">\n<div class="syndicate"><h1 i ...
```

## 2.2 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()

table_text
```

```
##  [1] "State\nState Abbreviation\nOpioid Dispensing Rate per 100\n"
##  [2] "Alabama\nAL\n126.1\n"
##  [3] "Alaska\nAK\n68.5\n"
##  [4] "Arizona\nAZ\n80.9\n"
##  [5] "Arkansas\nAR\n112.1\n"
##  [6] "California\nCA\n55.1\n"
##  [7] "Colorado\nCO\n67.7\n"
##  [8] "Connecticut\nCT\n68.7\n"
##  [9] "Delaware\nDE\n95.4\n"
## [10] "District of Columbia\nDC\n34.5\n"
```

```
## [11] "Florida\nFL\n84.3\n"
## [12] "Georgia\nGA\n86.3\n"
## [13] "Hawaii\nHI\n46.6\n"
## [14] "Idaho\nID\n82.7\n"
## [15] "Illinois\nIL\n60.2\n"
## [16] "Indiana\nIN\n103.3\n"
## [17] "Iowa\nIA\n59.1\n"
## [18] "Kansas\nKS\n82.7\n"
## [19] "Kentucky\nKY\n136.6\n"
## [20] "Louisiana\nLA\n113.7\n"
## [21] "Maine\nME\n88.7\n"
## [22] "Maryland\nMD\n65.5\n"
## [23] "Massachusetts\nMA\n69.2\n"
## [24] "Michigan\nMI\n89.9\n"
## [25] "Minnesota\nMN\n56.5\n"
## [26] "Mississippi\nMS\n113.2\n"
## [27] "Missouri\nMO\n86.8\n"
## [28] "Montana\nMT\n85.3\n"
## [29] "Nebraska\nNE\n66.2\n"
## [30] "Nevada\nNV\n97.0\n"
## [31] "New Hampshire\nNH\n81.7\n"
## [32] "New Jersey\nNJ\n59.5\n"
## [33] "New Mexico\nNM\n71.4\n"
## [34] "New York\nNY\n48.4\n"
## [35] "North Carolina\nNC\n88.6\n"
## [36] "North Dakota\nND\n61.7\n"
## [37] "Ohio\nOH\n97.5\n"
## [38] "Oklahoma\nOK\n111.3\n"
## [39] "Oregon\nOR\n99.1\n"
## [40] "Pennsylvania\nPA\n76.5\n"
## [41] "Rhode Island\nRI\n82.9\n"
## [42] "South Carolina\nSC\n94.1\n"
## [43] "South Dakota\nSD\n52.1\n"
## [44] "Tennessee\nTN\n132.9\n"
## [45] "Texas\nTX\n71.3\n"
## [46] "Utah\nUT\n91.3\n"
## [47] "Vermont\nVT\n56.5\n"
## [48] "Virginia\nVA\n73.0\n"
## [49] "Washington\nWA\n86.6\n"
## [50] "West Virginia\nWV\n145.5\n"
## [51] "Wisconsin\nWI\n70.6\n"
## [52] "Wyoming\nWY\n81.0\n"
```

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")
rough_table
```

```
## # A tibble: 52 x 3
##    state               abbrev             rate
##    <chr>               <chr>              <chr>
##  1 State               State Abbreviation Opioid Dispensing Rate per 100
```

```
##  2 Alabama              AL                 126.1
##  3 Alaska               AK                 68.5
##  4 Arizona              AZ                 80.9
##  5 Arkansas             AR                 112.1
##  6 California           CA                 55.1
##  7 Colorado             CO                 67.7
##  8 Connecticut          CT                 68.7
##  9 Delaware             DE                 95.4
## 10 District of Columbia DC                 34.5
## # ... with 42 more rows
```

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))

d_prescriptions
```

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

Now we have clean data for 2008! Great success.

## 2.3 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.

## 2.4 Question 1

Add a year column to `d_prescriptions`.

**Answer**:

The code add the column is shown below:

```
d_prescriptions$"year" = 0
```

## 2.5 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 https://www.cdc.gov/drugoverdose/maps/rxrate-maps.html, on the right hand side there's hyperlinks to all the years under "U.S. State Prescribing 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.

## 2.6 Question 2

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

**Answer**:

The code add the URL is shown below:

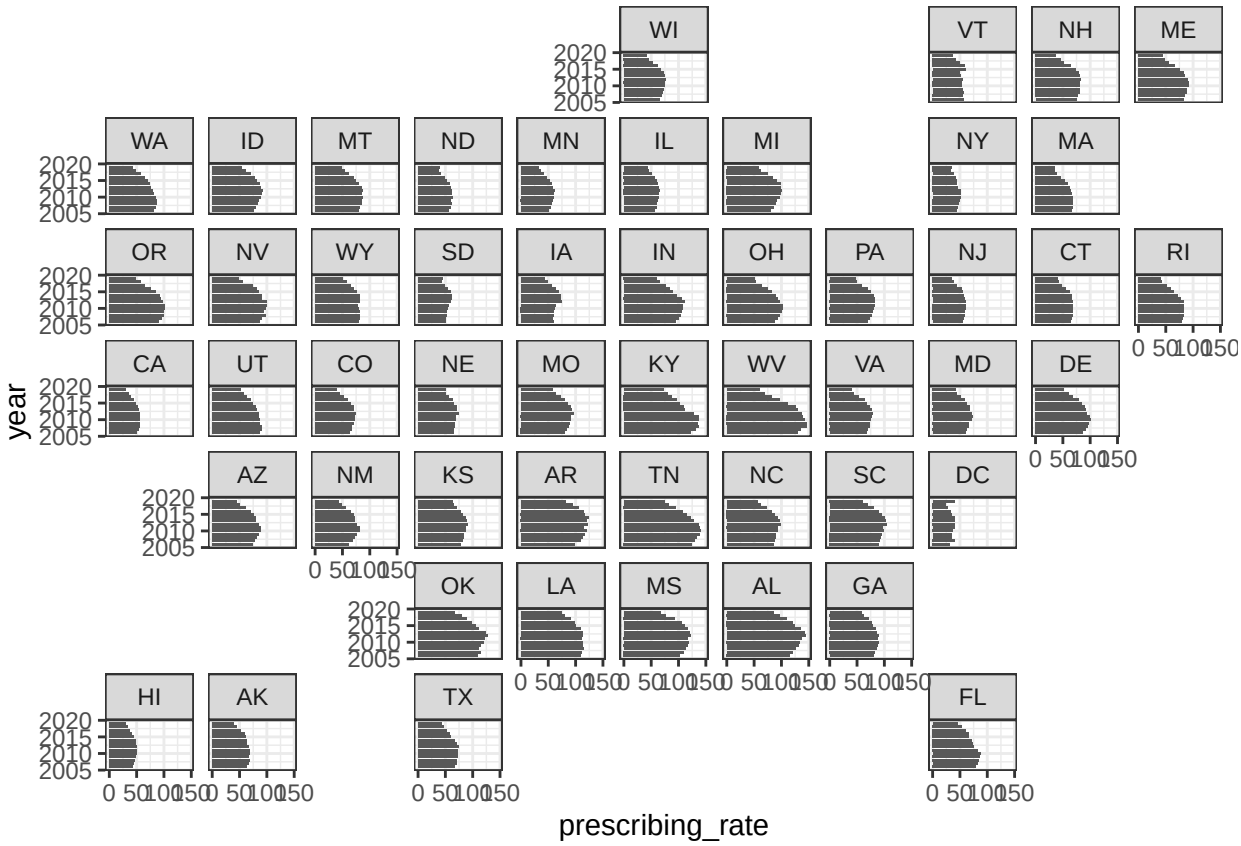
```
prefix = "https://www.cdc.gov/drugoverdose/maps/rxstate"
result = c()
for(year in c(2006:2019)){
  current = paste0(prefix, year, ".html")
  result = c(result, current)
}
```

## 2.7 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.

```
ggplot(all_prescriptions, aes(year, prescribing_rate)) +
  geom_col() +
  coord_flip() +
  facet_geo(~ state_abbreviation) +
  theme_bw()
```




# 3 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(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 '19a872829d9614a76e61d7b1c1300d34' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 2.3e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.23 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
```

```
## Error in sampler$call_sampler(args_list[[i]]) :
##   c++ exception (unknown reason)
## [1] "In addition: There were 13 warnings (use warnings() to see them)"
```

```
summary(mod)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: y ~ x
##    Data: d (Number of observations: 100)
##
## The model does not contain posterior samples.
```