Lab Exercise - Web Scraping

Luis Correia - Student No. 1006508566

February 5th 2020

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

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.

```r
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 and (if you scroll down) a table of state prescription rates in 2008. Let's read in the html of this page.

```r
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 (23)}
##  [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\t<div class="d- ...
##  [6] <div class="container-fluid feature-area">\r\n\t\t<div class="container" ...
##  [7] <div class="container d-flex flex-wrap body-wrapper bg-white">\r\n\t\t<! ...
##  [8] <footer role="contentinfo" aria-label="Footer"><div class="container-flu ...
##  [9] <nav role="navigation" aria-label="Social Media" class="d-lg-none w-100  ...
## [10] <div id="metrics">\r\n    <!-- Google DAP inclusion -->\r\n    <script i ...
## [11] <script src="/TemplatePackage/contrib/libs/jquery/3.3.1/jquery.min.js">< ...
## [12] <script src="/TemplatePackage/contrib/libs/bootstrap/4.1.3/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=19-12-12T15:56: ...
## [15] <svg style="display:none" xmlns="http://www.w3.org/2000/svg" xmlns:xlink ...
## [16] <svg style="display:none" xmlns="http://www.w3.org/2000/svg" xmlns:xlink ...
## [17] <svg style="display:none" xmlns="http://www.w3.org/2000/svg" xmlns:xlink ...
## [18] <svg style="display:none" xmlns="http://www.w3.org/2000/svg" xmlns:xlink ...
## [19] <svg style="display:none" xmlns="http://www.w3.org/2000/svg" xmlns:xlink ...
## [20] <script>\r\n    \r\n    <U+FEFF>s.pageName=document.title; \r\ns.channel="Drug  ...
## ...
```

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">\t\t\t\t\t<div class="syndicat ...
```

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 ABBR</th>\n<th>2008 Prescribing Rate</th ...
##  [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:

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

table_text
```

```
##  [1] "State\nState ABBR\n2008 Prescribing Rate\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:

```r
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 ABBR 2008 Prescribing Rate
##  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:

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

```
  rename(prescribing_rate = x2008_prescribing_rate) %>%
  mutate(prescribing_rate = as.numeric(prescribing_rate))

d_prescriptions
```

```
## # A tibble: 51 x 3
##    state                state_abbr 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.

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`.

```
d_prescriptions <- d_prescriptions %>%
  mutate(year = 2008)
d_prescriptions
```

```
## # A tibble: 51 x 4
##    state                state_abbr 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 2009-2016 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.

Question 2

Make a vector of the urls for each year, storing them as strings. Here's some code to fill in the gaps (remember to remove eval = F):

```
# https://www.cdc.gov/drugoverdose/maps/rxcounty2009.html  - Base URL for 2009
years <- c(2009:2016)
base_url <- "https://www.cdc.gov/drugoverdose/maps/rxcounty"
year_urls <- paste0(base_url,years[1:8],".html")
year_urls
```

```
## [1] "https://www.cdc.gov/drugoverdose/maps/rxcounty2009.html"
## [2] "https://www.cdc.gov/drugoverdose/maps/rxcounty2010.html"
## [3] "https://www.cdc.gov/drugoverdose/maps/rxcounty2011.html"
## [4] "https://www.cdc.gov/drugoverdose/maps/rxcounty2012.html"
## [5] "https://www.cdc.gov/drugoverdose/maps/rxcounty2013.html"
## [6] "https://www.cdc.gov/drugoverdose/maps/rxcounty2014.html"
## [7] "https://www.cdc.gov/drugoverdose/maps/rxcounty2015.html"
## [8] "https://www.cdc.gov/drugoverdose/maps/rxcounty2016.html"
```

Question 3

By filling in the code below, extract the prescriptions data for the years 2008-2016, and store in the one tibble. Make sure you have a column for state, state abbreviation, prescription rate and year. (remember to remove eval = F)

Note: if you copy paste the code above and put it in the loop, you will get an error because the prescriptions data column name has the year in it. You can get around this however you want, but you can define column names based on a variable making use of `!!` e.g. `!!paste0("x",years[i],"_prescribing_rate")`

Another note: notice the last year is 2016, not 2017. If you look at the 2017 page, you'll notice the format of the column names has changed. So you would have to write some more code to deal with this special case. You don't have to do this for the lab, but if you want extra practice, maybe this would be a good exercise.

```
prescriptions_all_years <- c()

for(i in 1:length(years)){
  cdc <- read_html(year_urls[i])
  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("county", "state", "abbrev", "rate"), sep = "\n", extra = "drop")
  d_prescriptions <- rough_table %>%
    janitor::row_to_names(1) %>%
    janitor::clean_names() %>%
    rename(prescribing_rate = paste0("x",years[i],"_prescribing_rate")) %>%
```

```
    mutate(prescribing_rate = as.numeric(prescribing_rate), year = years[i])

  prescriptions_all_years <- bind_rows(prescriptions_all_years, d_prescriptions) # assuming your tibble
  Sys.sleep(1) # wait a sec until going again
}
```

Print the head and tail of your dataset (remember to remove eval = F)

```
head(prescriptions_all_years)
```

```
## # A tibble: 6 x 5
##   county             state fips_county_code prescribing_rate  year
##   <chr>              <chr> <chr>                       <dbl> <int>
## 1 Aleutians East, AK AK    02013                        NA    2009
## 2 Aleutians West, AK AK    02016                        NA    2009
## 3 Anchorage, AK      AK    02020                        74.6  2009
## 4 Bethel, AK         AK    02050                        NA    2009
## 5 Bristol Bay, AK    AK    02060                        NA    2009
## 6 Denali, AK         AK    02068                        NA    2009
```

```
tail(prescriptions_all_years)
```

```
## # A tibble: 6 x 5
##   county         state fips_county_code prescribing_rate  year
##   <chr>          <chr> <chr>                       <dbl> <int>
## 1 Sublette, WY   WY    56035                        58.6  2016
## 2 Sweetwater, WY WY    56037                        87.7  2016
## 3 Teton, WY      WY    56039                        62.7  2016
## 4 Uinta, WY      WY    56041                       105.   2016
## 5 Washakie, WY   WY    56043                        81.3  2016
## 6 Weston, WY     WY    56045                        46.7  2016
```

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 particularly if you have Catalina, you might run into issues.

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 'b826a9c83d4c4c0a956c153b1b52e939' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.034 seconds (Warm-up)
## Chain 1:                0.029 seconds (Sampling)
## Chain 1:                0.063 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'b826a9c83d4c4c0a956c153b1b52e939' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.034 seconds (Warm-up)
## Chain 2:                0.05 seconds (Sampling)
## Chain 2:                0.084 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'b826a9c83d4c4c0a956c153b1b52e939' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.057 seconds (Warm-up)
## Chain 3:                0.035 seconds (Sampling)
## Chain 3:                0.092 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'b826a9c83d4c4c0a956c153b1b52e939' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 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: 0.079 seconds (Warm-up)
## Chain 4:                0.041 seconds (Sampling)
## Chain 4:                0.12 seconds (Total)
## Chain 4:
```

```r
summary(mod)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: y ~ x
##    Data: d (Number of observations: 100)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     1.00      0.10     0.80     1.19 1.00     3802     2921
## x             2.07      0.10     1.88     2.28 1.00     3968     2780
```

```
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     1.01      0.07     0.89     1.17 1.00     4135     3156
##
## Samples were drawn 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).
```