Data Acquisition with APIs in R

You can download this .qmd file from here. Just hit the Download Raw File button.

Credit to Brianna Heggeseth and Leslie Myint from Macalester College for a few of these descriptions and examples.

Getting data from websites

Option 1: APIs

When we interact with sites like The New York Times, Zillow, and Google, we are accessing their data via a graphical layout (e.g., images, colors, columns) that is easy for humans to read but hard for computers.

An **API** stands for **Application Programming Interface**, and this term describes a general class of tool that allows computers, rather than humans, to interact with an organization's data. How does this work?

- When we use web browsers to navigate the web, our browsers communicate with web servers using a technology called HTTP or Hypertext Transfer Protocol to get information that is formatted into the display of a web page.
- Programming languages such as R can also use HTTP to communicate with web servers. The easiest way to do this is via Web APIs, or Web Application Programming Interfaces, which focus on transmitting raw data, rather than images, colors, or other appearance-related information that humans interact with when viewing a web page.

A large variety of web APIs provide data accessible to programs written in R (and almost any other programming language!). Almost all reasonably large commercial websites offer APIs. Todd Motto has compiled an expansive list of Public Web APIs on GitHub, although it's about 3 years old now so it's not a perfect or complete list. Feel free to browse this list to see what data sources are available.

For our purposes of obtaining data, APIs exist where website developers make data nicely packaged for consumption. The language HTTP (hypertext transfer protocol) underlies APIs, and the R package `httr()` (and now the updated `httr2()`) was written to map closely to HTTP with R. Essentially you send a request to the website (server) where you want data from, and they send a response, which should contain the data (plus other stuff).

The case studies in this document provide a really quick introduction to data acquisition, just to get you started and show you what's possible. For more information, these links can be somewhat helpful:

- https://www.geeksforgeeks.org/functions-with-r-and-rvest/#
- https://nceas.github.io/oss-lessons/data-liberation/intro-webscraping.html

Wrapper packages

In R, it is easiest to use Web APIs through a **wrapper package**, an R package written specifically for a particular Web API.

- The R development community has already contributed wrapper packages for many large Web APIs (e.g. ZillowR, rtweet, genius, Rspotify, tidycensus, etc.)
- To find a wrapper package, search the web for "R package" and the name of the website. For example:
  - Searching for "R Reddit package" returns RedditExtractor
  - Searching for "R Weather.com package" returns weatherData
- rOpenSci also has a good collection of wrapper packages.

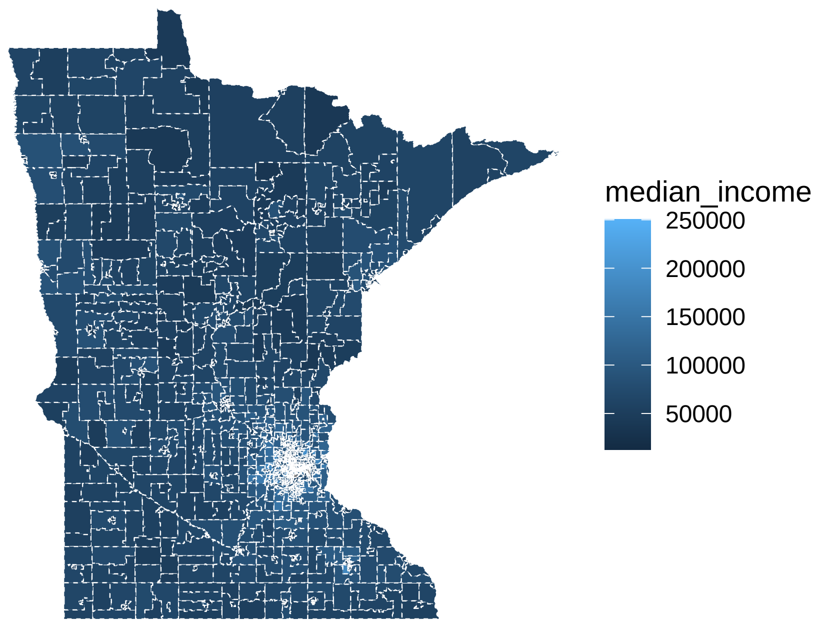
In particular, `tidycensus` is a wrapper package that makes it easy to obtain desired census information for mapping and modeling:

Obtaining raw data from the Census Bureau was that easy! Often we will have to obtain and use a secret API key to access the data, but that's not always necessary with `tidycensus`.

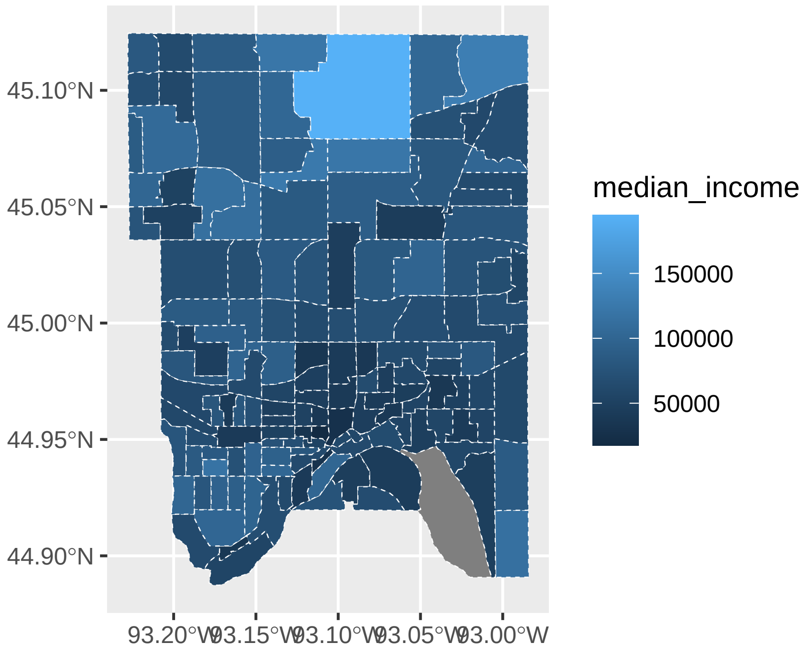
Now we can tidy that data and produce plots and analyses. Here's a decent place to get more information about the variable codes.

```
# Rename cryptic variables from the census form
sample_acs_data <- sample_acs_data |>
  rename(population = B01003_001E,
         population_moe = B01003_001M,
         median_income = B19013_001E,
         median_income_moe = B19013_001M)

# Plot with geom_sf since our data contains 1 row per census tract
#   with its geometry
ggplot(data = sample_acs_data) +
```

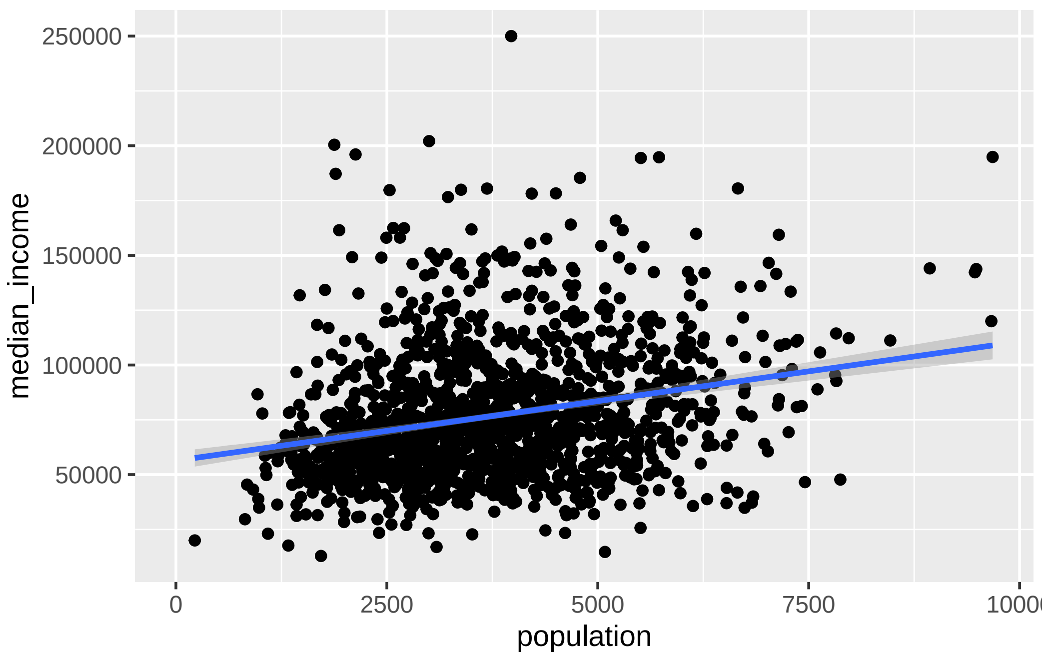
```
  geom_sf(aes(fill = median_income), colour = "white", linetype = 2) +
  theme_void()
```




```
# The whole state of MN is overwhelming, so focus on a single county
sample_acs_data |>
  filter(str_detect(NAME, "Ramsey")) |>
  ggplot() +
    geom_sf(aes(fill = median_income), colour = "white", linetype = 2)
```




```
# Look for relationships between variables with 1 row per tract
as_tibble(sample_acs_data) |>
  ggplot(aes(x = population, y = median_income)) +
    geom_point() +
    geom_smooth(method = "lm")
```



Extra resources:

- `tidycensus`: wrapper package that provides an interface to a few census datasets *with map geometry included!*
    - Full documentation is available at https://walker-data.com/tidycensus/
- `censusapi`: wrapper package that offers an interface to all census datasets
    - Full documentation is available at https://www.hrecht.com/censusapi/

`get_acs()` is one of the functions that is part of `tidycensus`. Let's explore what's going on behind the scenes with `get_acs()`…

Accessing web APIs directly

Getting a Census API key

Many APIs (and their wrapper packages) require users to obtain a **key** to use their services.

- This lets organizations keep track of what data is being used.
- It also **rate limits** their API and ensures programs don't make too many requests per day/minute/hour. Be aware that most APIs do have rate limits — especially for their free tiers.

Navigate to https://api.census.gov/data/key_signup.html to obtain a Census API key:

- Organization: St. Olaf College
- Email: Your St. Olaf email address

You will get the message:

> Your request for a new API key has been successfully submitted. Please check your email. In a few minutes you should receive a message with instructions on how to activate your new key.

Check your email. Copy and paste your key into a new text file:

- (In RStudio) File > New File > Text File (towards the bottom of the menu)
- Save as `census_api_key.txt` in the same folder as this `.qmd`.

You could then read in the key with code like this:

```
myapikey <- readLines("C:/Users/charl/Documents/SDS_264/census_api_key")
```

```
Warning in readLines("C:/Users/charl/Documents/SDS_264/census_api_key"):
incomplete final line found on
'C:/Users/charl/Documents/SDS_264/census_api_key'
```

Handling API keys

While this works, the problem is once we start backing up our files to GitHub, your API key will also appear on GitHub, and you want to keep your API key secret. Thus, we might use **environment variables** instead:

One way to store a secret across sessions is with environment variables. Environment variables, or envvars for short, are a cross platform way of passing information to processes. For passing envvars to R, you can list name-value pairs in a file called .Renviron in your home directory. The easiest way to edit it is to run:

```
file.edit("~/.Renviron")

Sys.setenv(PATH = "path", VAR1 = "value1", VAR2 = "value2")
```

The file looks something like

PATH = “path” VAR1 = “value1” VAR2 = “value2” And you can access the values in R using `Sys.getenv()`:

```r
Sys.getenv("VAR1")
#> [1] "value1"
```

Note that .Renviron is only processed on startup, so you'll need to restart R to see changes.

Another option is to use `Sys.setenv` and `Sys.getenv`:

```r
# I used the first line to store my CENSUS API key in .Renviron
#   after uncommenting - should only need to run one time
#Sys.setenv(CENSUS_API_KEY = "my personal key")
my_census_api_key <- Sys.getenv("CENSUS_API_KEY")
```

Navigating API documentation

Navigate to the Census API user guide and click on the "Example API Queries" tab.

Let's look at the Population Estimates Example and the American Community Survey (ACS) Example. These examples walk us through the steps to incrementally build up a URL to obtain desired data. This URL is known as a web API **request**.

https://api.census.gov/data/2019/acs/acs1?get=NAME,B02015_009E,B02015_009M&for=state:*

- `https://api.census.gov`: This is the **base URL**.
  - `http://`: The **scheme**, which tells your browser or program how to communicate with the web server. This will typically be either `http:` or `https:`.
  - `api.census.gov`: The **hostname**, which is a name that identifies the web server that will process the request.
- `data/2019/acs/acs1`: The **path**, which tells the web server how to get to the desired resource.
  - In the case of the Census API, this locates a desired dataset in a particular year.
  - Other APIs allow search functionality. (e.g., News organizations have article searches.) In these cases, the path locates the search function we would like to call.
- `?get=NAME,B02015_009E,B02015_009M&for=state:*`: The **query parameters**, which provide the parameters for the function you would like to call.
  - We can view this as a string of key-value pairs separated by `&`. That is, the general structure of this part is `key1=value1&key2=value2`.

key	value

key	value
get	NAME,B02015_009E,B02015_009M
for	state:*

Typically, each of these URL components will be specified in the API documentation. Sometimes, the scheme, hostname, and path (`https://api.census.gov/data/2019/acs/acs1`) will be referred to as the **endpoint** for the API call.

We will first use the `httr2` package to build up a full URL from its parts.

- `request()` creates an API request object using the **base URL**
- `req_url_path_append()` builds up the URL by adding path components separated by `/`
- `req_url_query()` adds the `?` separating the endpoint from the query and sets the key-value pairs in the query
  - The `.multi` argument controls how multiple values for a given key are combined.
  - The `I()` function around `"state:*"` inhibits parsing of special characters like `:` and `*`. (It's known as the "as-is" function.)
  - The backticks around `for` are needed because `for` is a reserved word in R (for for-loops). You'll need backticks whenever the key name has special characters (like spaces, dashes).
  - We can see from here that providing an API key is achieved with `key=YOUR_API_KEY`.

```
# Request total number of Hmong residents and margin of error by state
#   in 2019, as in the User Guide
CENSUS_API_KEY <- Sys.getenv("CENSUS_API_KEY")
req <- request("https://api.census.gov") |>
    req_url_path_append("data") |>
    req_url_path_append("2019") |>
    req_url_path_append("acs") |>
    req_url_path_append("acs1") |>
    req_url_query(get = c("NAME", "B02015_009E", "B02015_009M"), `for` = I("state:*"), key =
```

Why would we ever use these steps instead of just using the full URL as a string?

- To generalize this code with functions! (This is exactly what wrapper packages do.)
- To handle special characters

    - e.g., query parameters might have spaces, which need to be represented in a particular way in a URL (URLs can't contain spaces)

Once we've fully constructed our request, we can use `req_perform()` to send out the API request and get a **response**.

```r
resp <- req_perform(req)
resp
```

We see from `Content-Type` that the format of the response is something called JSON. We can navigate to the request URL to see the structure of this output.

- JSON (Javascript Object Notation) is a nested structure of key-value pairs.
- We can use `resp_body_json()` to parse the JSON into a nicer format.
    - Without `simplifyVector = TRUE`, the JSON is read in as a list.

```r
resp_json_list <- resp |> resp_body_json()
head(resp_json_list, 2)
```

```
[[1]]
[[1]][[1]]
[1] "NAME"

[[1]][[2]]
[1] "B02015_009E"

[[1]][[3]]
[1] "B02015_009M"

[[1]][[4]]
[1] "state"


[[2]]
[[2]][[1]]
[1] "Mississippi"

[[2]][[2]]
NULL

[[2]][[3]]
```

```
NULL

[[2]][[4]]
[1] "28"
```

```r
resp_json_df <- resp |> resp_body_json(simplifyVector = TRUE)
head(resp_json_df)
```

```
     [,1]          [,2]          [,3]          [,4]
[1,] "NAME"        "B02015_009E" "B02015_009M" "state"
[2,] "Mississippi" NA            NA            "28"
[3,] "Missouri"    "953"         "1141"        "29"
[4,] "Montana"     NA            NA            "30"
[5,] "Nebraska"    "412"         "477"         "31"
[6,] "Nevada"      "863"         "745"         "32"
```

```r
resp_json_df <- janitor::row_to_names(resp_json_df, 1)
head(resp_json_df)
```

```
     NAME            B02015_009E B02015_009M state
[1,] "Mississippi"   NA          NA          "28"
[2,] "Missouri"      "953"       "1141"      "29"
[3,] "Montana"       NA          NA          "30"
[4,] "Nebraska"      "412"       "477"       "31"
[5,] "Nevada"        "863"       "745"       "32"
[6,] "New Hampshire" NA          NA          "33"
```

All right, let's try this! First we'll grab total population and median household income for all census tracts in MN using 3 approaches

```r
# First using tidycenus
library(tidycensus)
sample_acs_data <- tidycensus::get_acs(
    year = 2021,
    state = "MN",
    geography = "tract",
    variables = c("B01003_001", "B19013_001"),
    output = "wide",
    geometry = TRUE,
    county = "Hennepin",   # specify county in call
    show_call = TRUE       # see resulting query
)
```

```r
# Next using httr2
req <- request("https://api.census.gov") |>
    req_url_path_append("data") |>
    req_url_path_append("2020") |>
    req_url_path_append("acs") |>
    req_url_path_append("acs5") |>
    req_url_query(get = c("NAME", "B01003_001E", "B19013_001E"), `for` = I("tract:*"), `in` =

resp <- req_perform(req)
resp
resp_json_df <- resp |> resp_body_json(simplifyVector = TRUE)
head(resp_json_df)
```

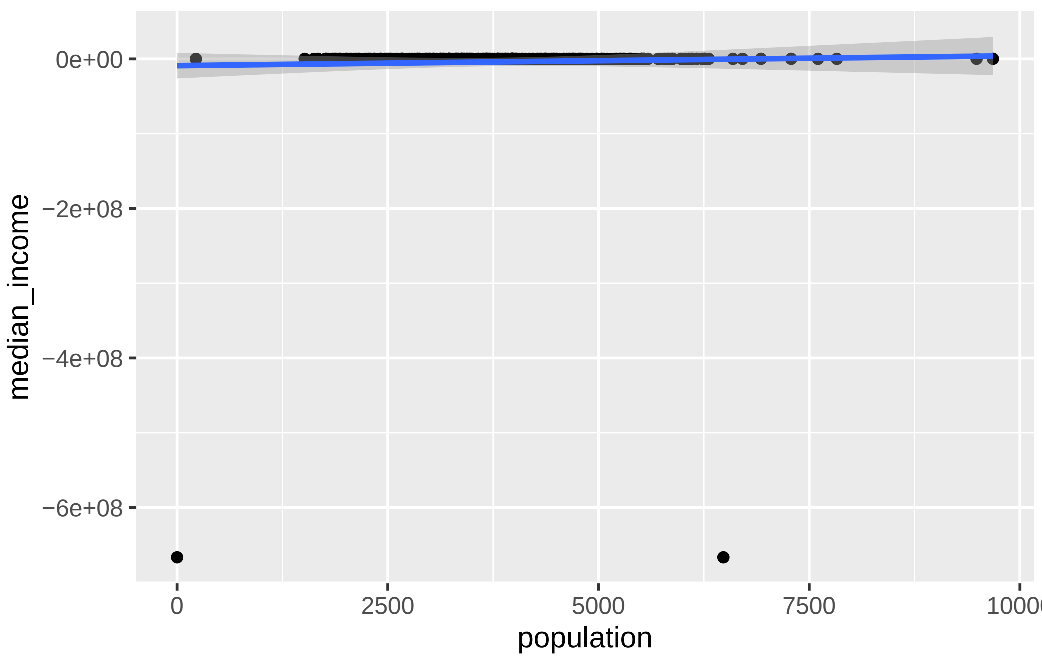
```
     [,1]                                               [,2]
[1,] "NAME"                                             "B01003_001E"
[2,] "Census Tract 1.01, Hennepin County, Minnesota" "3472"
[3,] "Census Tract 1.02, Hennepin County, Minnesota" "4992"
[4,] "Census Tract 3, Hennepin County, Minnesota"    "3404"
[5,] "Census Tract 6.01, Hennepin County, Minnesota" "4706"
[6,] "Census Tract 6.03, Hennepin County, Minnesota" "3301"
     [,3]          [,4]    [,5]     [,6]
[1,] "B19013_001E" "state" "county" "tract"
[2,] "70927"       "27"    "053"    "000101"
[3,] "46333"       "27"    "053"    "000102"
[4,] "82098"       "27"    "053"    "000300"
[5,] "71122"       "27"    "053"    "000601"
[6,] "96875"       "27"    "053"    "000603"
```

```r
resp_json_df <- janitor::row_to_names(resp_json_df, 1)
head(resp_json_df)
```

```
     NAME                                            B01003_001E B19013_001E
[1,] "Census Tract 1.01, Hennepin County, Minnesota" "3472"      "70927"
[2,] "Census Tract 1.02, Hennepin County, Minnesota" "4992"      "46333"
[3,] "Census Tract 3, Hennepin County, Minnesota"    "3404"      "82098"
[4,] "Census Tract 6.01, Hennepin County, Minnesota" "4706"      "71122"
[5,] "Census Tract 6.03, Hennepin County, Minnesota" "3301"      "96875"
[6,] "Census Tract 11, Hennepin County, Minnesota"   "2004"      "69509"
     state county tract
[1,] "27"  "053"  "000101"
[2,] "27"  "053"  "000102"
```

```
[3,] "27"  "053"  "000300"
[4,] "27"  "053"  "000601"
[5,] "27"  "053"  "000603"
[6,] "27"  "053"  "001100"
```

```
hennepin_httr2 <- as_tibble(resp_json_df) |>
  mutate(population = parse_number(B01003_001E),
         median_income = parse_number(B19013_001E)) |>
  select(-B01003_001E, -B19013_001E, -state, -county)

hennepin_httr2 |>
  ggplot(aes(x = population, y = median_income)) +
    geom_point() +
    geom_smooth(method = "lm")
```




```
summary(hennepin_httr2$population)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      0    2876    3714    3815    4651    9680
```

```r
summary(hennepin_httr2$median_income)
```

```
     Min.    1st Qu.     Median       Mean    3rd Qu.       Max. 
-666666666      61354      80966   -3966166     107232     250001 
```

```r
sort(hennepin_httr2$population)
```

```
  [1]    0  223 1514 1622 1672 1760 1766 1779 1798 1844 1848 1877 1897 1915 1926
 [16] 1935 1942 1973 2000 2004 2012 2013 2017 2038 2058 2061 2067 2092 2111 2123
 [31] 2130 2150 2163 2228 2235 2256 2272 2274 2280 2283 2295 2315 2339 2341 2357
 [46] 2399 2415 2416 2419 2460 2462 2476 2484 2499 2511 2511 2528 2532 2551 2570
 [61] 2594 2605 2625 2656 2658 2668 2670 2675 2681 2724 2738 2756 2763 2780 2796
 [76] 2808 2820 2822 2837 2848 2853 2865 2876 2878 2916 2935 2944 2950 2954 2969
 [91] 2971 2984 2994 3001 3036 3037 3038 3046 3047 3048 3075 3077 3119 3124 3127
[106] 3138 3150 3152 3162 3168 3193 3222 3224 3224 3225 3236 3251 3274 3298 3301
[121] 3305 3317 3317 3326 3331 3335 3341 3364 3372 3376 3379 3386 3404 3404 3418
[136] 3431 3439 3444 3454 3466 3472 3474 3486 3498 3512 3513 3557 3573 3574 3575
[151] 3585 3607 3628 3631 3634 3654 3656 3666 3671 3673 3676 3687 3703 3710 3714
[166] 3739 3750 3762 3764 3765 3799 3801 3805 3806 3808 3810 3811 3829 3832 3842
[181] 3853 3862 3877 3885 3890 3895 3896 3896 3903 3903 3913 3924 3930 3960 3967
[196] 3972 3974 3976 3978 3980 3989 3995 4008 4010 4013 4025 4036 4063 4086 4097
[211] 4098 4126 4132 4179 4200 4219 4228 4237 4273 4286 4295 4305 4319 4321 4326
[226] 4355 4359 4366 4371 4378 4385 4412 4441 4455 4460 4466 4472 4481 4503 4535
[241] 4584 4587 4591 4613 4622 4629 4651 4665 4671 4678 4693 4696 4706 4713 4718
[256] 4728 4747 4767 4769 4789 4789 4815 4855 4855 4874 4899 4919 4930 4972 4978
[271] 4983 4992 5030 5033 5041 5065 5085 5099 5107 5150 5195 5213 5244 5262 5267
[286] 5295 5305 5313 5364 5366 5385 5386 5415 5442 5459 5507 5510 5515 5541 5541
[301] 5587 5709 5725 5781 5821 5831 5872 5880 5980 6025 6069 6071 6102 6113 6166
[316] 6229 6249 6258 6265 6308 6482 6595 6709 6928 7286 7604 7828 9486 9680
```

```r
sort(hennepin_httr2$median_income)
```

```
  [1] -666666666 -666666666      14748      20000      22768      23256
  [7]      23391      25708      31513      31981      32321      32758
 [13]      34273      35368      35855      36700      37315      37346
 [19]      37413      38286      38554      39420      39605      39609
 [25]      39630      40400      40476      40603      40867      42426
 [31]      42550      42753      43036      43750      44867      45640
 [37]      46157      46333      46596      47139      47197      47688
 [43]      47857      48464      48690      48750      49028      49139
```

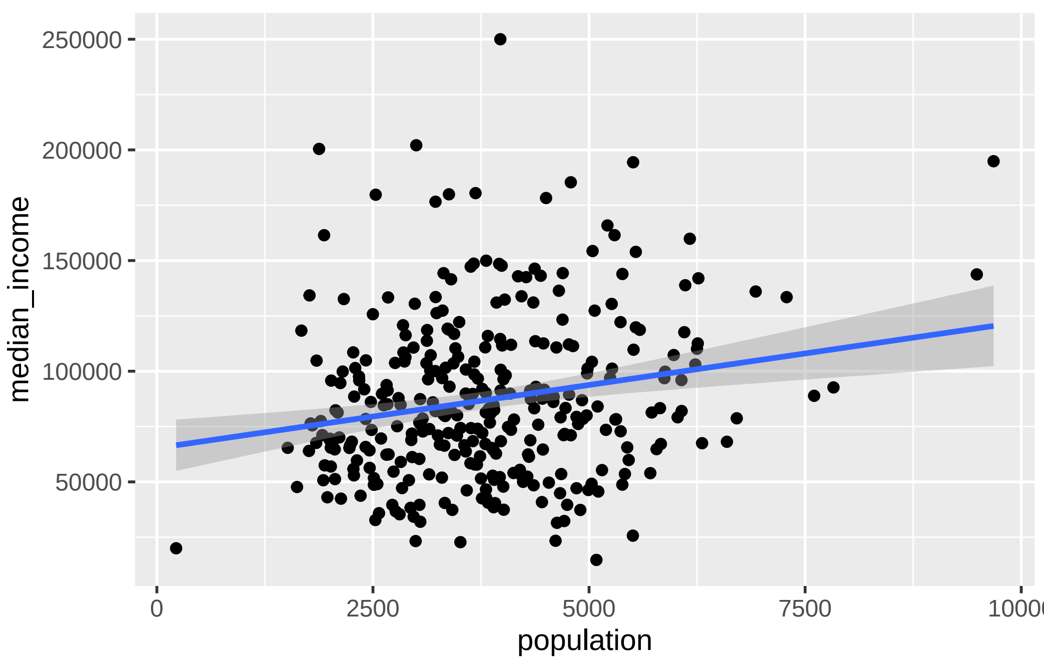
[49]	49659	50000	50741	50755	50935	51250
[55]	51513	51705	51923	52169	52304	52370
[61]	52781	52917	53393	53542	53564	53952
[67]	54026	54636	55321	55430	55833	56338
[73]	56955	57469	57802	57875	58426	59013
[79]	59704	59876	60375	61213	61354	61547
[85]	62188	62279	62404	62426	62770	63750
[91]	63990	64250	64333	64621	64676	64792
[97]	65323	65329	65395	65455	65590	65772
[103]	66364	66452	66549	66875	67102	67132
[109]	67473	67614	68114	68158	68369	68417
[115]	68434	68796	68913	68971	69509	69600
[121]	70089	70927	70970	71071	71122	71146
[127]	71250	71670	71818	72054	72102	72766
[133]	72853	73482	73514	73527	73897	73984
[139]	74286	74330	74817	75147	75556	75833
[145]	76111	76164	76417	76792	76839	77500
[151]	78137	78171	78333	78418	78509	78605
[157]	78728	79167	79191	79366	79750	80012
[163]	80080	80350	80966	81341	81341	81411
[169]	81977	82014	82098	82340	82527	83090
[175]	83250	83315	83380	84063	84569	84583
[181]	84792	85078	85221	85938	86106	86111
[187]	86904	87054	87390	87426	87599	87857
[193]	88431	88542	88895	89417	89740	89792
[199]	89891	89922	90167	91230	91250	91333
[205]	91637	91827	92019	92683	92941	93011
[211]	93750	94656	95750	95855	95980	96328
[217]	96378	96667	96856	96875	96983	97609
[223]	98137	98550	98986	99792	99853	100054
[229]	100329	100652	100761	101156	101194	101440
[235]	101578	103049	103531	103611	103750	104242
[241]	104306	104412	104795	104904	106310	106518
[247]	107232	107303	108476	108510	109722	110125
[253]	110339	110694	110729	110774	111364	111635
[259]	111950	112104	112557	112566	113563	113750
[265]	114550	115934	116281	116861	117631	118333
[271]	118594	118697	118828	119214	119821	120769
[277]	122180	122206	123312	125750	126250	127375
[283]	127396	130404	130486	131023	131042	132361
[289]	132604	133333	133472	133504	133859	134250
[295]	136012	136369	138848	141528	141984	142500
[301]	142889	143125	143744	143935	144282	144318

```
[307]     146328     147237     147672     148512     148611     149934
[313]     153917     154306     159857     161458     161471     165865
[319]     176580     178259     179743     179926     180463     185357
[325]     194417     194882     200438     202098     250001
```

```r
hennepin_httr2 <- hennepin_httr2 |>
  mutate(median_income = ifelse(median_income > 0, median_income, NA),
         population = ifelse(population > 0, population, NA))

hennepin_httr2 |>
  ggplot(aes(x = population, y = median_income)) +
    geom_point() +
    geom_smooth(method = "lm")
```

```
Warning: Removed 2 rows containing non-finite outside the scale range
(`stat_smooth()`).
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_point()`).
```



```r
# To make choropleth map by census tract, would need to download US Census
#   Bureau TIGER geometries using tigris package
```

```r
# Finally using httr
url <- str_c("https://api.census.gov/data/2020/acs/acs5?get=NAME,B01003_001E,B19013_001E&for=
acs5 <- GET(url)
details <- content(acs5, "parsed")
# details
details[[1]]  # variable names
```

```
[[1]]
[1] "NAME"

[[2]]
[1] "B01003_001E"

[[3]]
[1] "B19013_001E"

[[4]]
[1] "state"

[[5]]
[1] "county"

[[6]]
[1] "tract"
```

```r
details[[2]]  # list with information on 1st tract
```

```
[[1]]
[1] "Census Tract 1.01, Hennepin County, Minnesota"

[[2]]
[1] "3472"

[[3]]
[1] "70927"

[[4]]
[1] "27"
```

```
[[5]]
[1] "053"

[[6]]
[1] "000101"
```

```
name = character()
population = double()
median_income = double()
tract = character()

for(i in 2:330) {
  name[i-1] <- details[[i]][[1]][1]
  population[i-1] <- details[[i]][[2]][1]
  median_income[i-1] <- details[[i]][[3]][1]
  tract[i-1] <- details[[i]][[6]][1]
}
hennepin_httr <- tibble(
  name = name,
  population = parse_number(population),
  median_income = parse_number(median_income),
  tract = tract
)
```

On Your Own

1. Write a for loop to obtain the Hennepin County data from 2017-2021
2. Write a function to give choices about year, county, and variables

```
# function to allow user inputs

MN_tract_data <- function(year, county, variables) {
  tidycensus::get_acs(
    Sys.sleep(0.5),
    year = year,
    state = "MN",
    geography = "tract",
    variables = variables,
    output = "wide",
    geometry = TRUE,
```

```r
    county = county
  ) |>
    mutate(year = year)
}

# Should really build in checks so that county is in MN, year is in
#   proper range, and variables are part of ACS1 data set

my_data <- MN_tract_data(year = 2021,
              county = "Hennepin",
              variables = c("B01003_001", "B19013_001"))
```

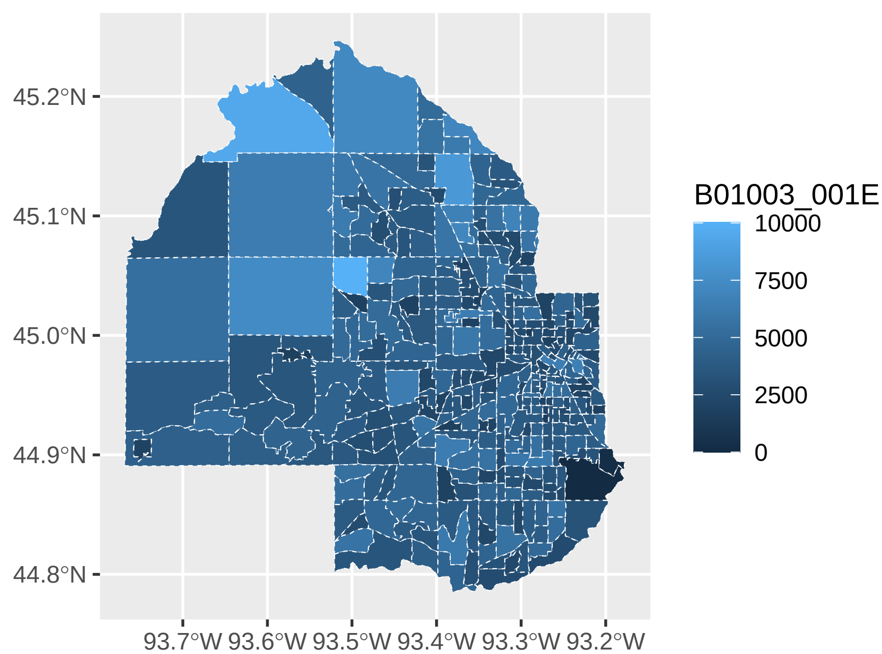
```
Getting data from the 2017-2021 5-year ACS

Downloading feature geometry from the Census website.  To cache shapefiles for use in future


  |
  |                                                                      |   0%
  |
  |=                                                                     |   2%
  |
  |=====                                                                 |   7%
  |
  |======                                                                |   9%
  |
  |==========                                                            |  14%
  |
  |===========                                                           |  16%
  |
  |============                                                          |  17%
  |
  |=============                                                         |  18%
  |
  |==============                                                        |  21%
  |
  |===============                                                       |  22%
  |
  |=================                                                     |  24%
  |
  |==================                                                    |  25%
```

```
|
|===================                         |  27%
|
|====================                        |  28%
|
|====================                        |  29%
|
|======================                      |  31%
|
|=======================                     |  32%
|
|=======================                     |  33%
|
|========================                    |  35%
|
|==========================                  |  36%
|
|===========================                 |  38%
|
|============================                |  40%
|
|=============================               |  41%
|
|=============================               |  42%
|
|===============================             |  44%
|
|================================            |  46%
|
|=================================           |  48%
|
|==================================          |  49%
|
|====================================        |  51%
|
|=====================================       |  52%
|
|======================================      |  55%
|
|=======================================     |  56%
|
|=========================================   |  58%
|
```

```
|=========================================                             |  59%
|
|==========================================                            |  60%
|
|===========================================                           |  62%
|
|============================================                          |  63%
|
|============================================                          |  64%
|
|==============================================                        |  65%
|
|===============================================                       |  67%
|
|===============================================                       |  68%
|
|================================================                      |  69%
|
|==================================================                    |  71%
|
|==================================================                    |  72%
|
|===================================================                   |  73%
|
|====================================================                  |  75%
|
|=====================================================                 |  75%
|
|=====================================================                 |  76%
|
|=======================================================               |  78%
|
|========================================================              |  80%
|
|=========================================================             |  82%
|
|==========================================================            |  83%
|
|===========================================================           |  84%
|
|============================================================          |  86%
|
|=============================================================         |  87%
```

```
  |
  |=============================================================        |  89%
  |
  |==============================================================       |  91%
  |
  |================================================================     |  93%
  |
  |=================================================================    |  95%
  |
  |==================================================================   |  96%
  |
  |===================================================================  |  97%
  |
  |==================================================================== |  99%
  |
  |=====================================================================| 100%
```

```r
ggplot(data = my_data) +
  geom_sf(aes(fill = B01003_001E), colour = "white", linetype = 2)
```




```r
my_data <- MN_tract_data(year = 2022,
              county = "Rice",
              variables = c("B01003_001", "B19013_001"))
```

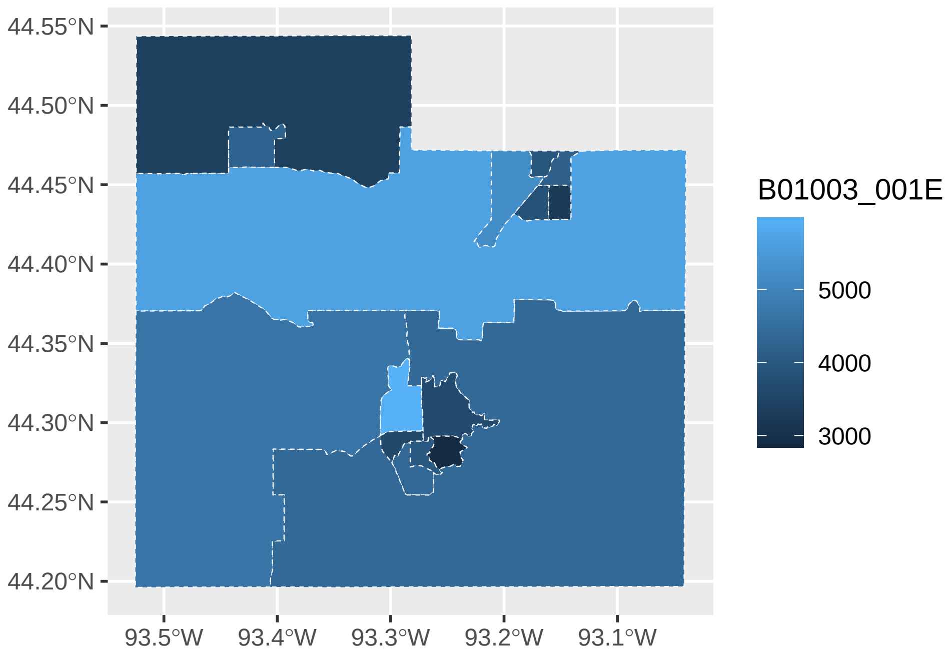
```
Getting data from the 2018-2022 5-year ACS
Downloading feature geometry from the Census website.  To cache shapefiles for use in future


  |
  |                                                                      |   0%
  |
  |=                                                                     |   2%
  |
  |==                                                                    |   3%
  |
  |====                                                                  |   5%
  |
  |=====                                                                 |   7%
  |
  |======                                                                |   9%
  |
  |=========                                                             |  13%
  |
  |==========                                                            |  14%
  |
  |===========                                                           |  16%
  |
  |============                                                          |  17%
  |
  |=============                                                         |  18%
  |
  |===============                                                       |  21%
  |
  |=================                                                     |  24%
  |
  |==================                                                    |  26%
  |
  |===================                                                   |  27%
  |
  |====================                                                  |  29%
  |
  |=====================                                                 |  30%
  |
  |======================                                                |  31%
  |
  |========================                                              |  34%
```

```
|
|=========================                                             |  35%
|
|==========================                                            |  37%
|
|==========================                                            |  38%
|
|===========================                                           |  39%
|
|============================                                          |  40%
|
|=============================                                         |  42%
|
|===============================                                       |  44%
|
|================================                                      |  46%
|
|=================================                                     |  47%
|
|==================================                                    |  48%
|
|===================================                                   |  50%
|
|====================================                                  |  51%
|
|====================================                                  |  52%
|
|=====================================                                 |  53%
|
|======================================                                |  55%
|
|=======================================                               |  56%
|
|========================================                              |  57%
|
|=========================================                             |  59%
|
|==========================================                            |  60%
|
|===========================================                           |  61%
|
|============================================                          |  63%
|
```

```
|=============================================                         |  64%
|
|=============================================                         |  65%
|
|==============================================                        |  66%
|
|===============================================                       |  68%
|
|================================================                      |  69%
|
|=================================================                     |  70%
|
|==================================================                    |  72%
|
|===================================================                   |  73%
|
|====================================================                  |  74%
|
|=====================================================                 |  76%
|
|======================================================                |  77%
|
|======================================================                |  78%
|
|=======================================================               |  79%
|
|=========================================================             |  81%
|
|=========================================================             |  82%
|
|==========================================================            |  83%
|
|===========================================================           |  85%
|
|============================================================          |  86%
|
|=============================================================         |  87%
|
|===============================================================       |  90%
|
|================================================================      |  92%
|
|==================================================================    |  95%
```

```
  |
  |===================================================================   |  96%
  |
  |====================================================================  |  98%
  |
  |===================================================================== |  99%
  |
  |======================================================================| 100%
```

```r
ggplot(data = my_data) +
  geom_sf(aes(fill = B01003_001E), colour = "white", linetype = 2)
```




```r
# Try other variables:
#  - B25077_001 is median home price
#  - B02001_002 is number of white residents
#  - etc.
# although the census codebook is admittedly quite daunting!
```

3. Use your function from (2) along with `map` and `list_rbind` to build a data set for Rice county for the years 2019-2021

```r
# To examine trends over time in Rice County
2019:2021 |>
  purrr::map(\(x)
    MN_tract_data(
      x,
      county = "Rice",
      variables = c("B01003_001", "B19013_001")
    )
  ) |>
  list_rbind()
```

```
Getting data from the 2015-2019 5-year ACS

Downloading feature geometry from the Census website.  To cache shapefiles for use in future


  |
  |                                                                      |   0%
  |
  |=                                                                     |   2%
  |
  |===                                                                   |   4%
  |
  |====                                                                  |   6%
  |
  |======                                                                |   8%
  |
  |===========                                                           |  16%
  |
  |============                                                          |  18%
  |
  |=============                                                         |  19%
  |
  |==============                                                        |  20%
  |
  |================                                                      |  22%
  |
  |================                                                      |  23%
  |
  |=================                                                     |  25%
  |
```

```
|====================                                                  |  29%
|
|=====================                                                 |  30%
|
|======================                                                |  32%
|
|=======================                                               |  33%
|
|========================                                              |  35%
|
|==========================                                            |  37%
|
|===========================                                           |  39%
|
|=============================                                         |  41%
|
|==============================                                        |  43%
|
|===============================                                       |  45%
|
|=================================                                     |  47%
|
|==================================                                    |  49%
|
|====================================                                  |  51%
|
|====================================                                  |  52%
|
|======================================                                |  54%
|
|========================================                              |  57%
|
|=========================================                             |  59%
|
|===========================================                           |  62%
|
|============================================                          |  63%
|
|==============================================                        |  66%
|
|================================================                      |  68%
|
|=================================================                     |  70%
```

```
  |
  |=================================================                     |  71%
  |
  |==================================================                    |  72%
  |
  |=====================================================                 |  75%
  |
  |=====================================================                 |  76%
  |
  |======================================================                |  78%
  |
  |========================================================              |  81%
  |
  |==========================================================            |  82%
  |
  |============================================================          |  86%
  |
  |=============================================================         |  88%
  |
  |===============================================================       |  90%
  |
  |=================================================================     |  92%
  |
  |==================================================================    |  94%
  |
  |==================================================================    |  95%
  |
  |====================================================================  |  97%
  |
  |===================================================================== |  98%
  |
  |======================================================================| 100%

Getting data from the 2016-2020 5-year ACS
Downloading feature geometry from the Census website.  To cache shapefiles for use in future


Getting data from the 2017-2021 5-year ACS


Downloading feature geometry from the Census website.  To cache shapefiles for use in future


         GEOID                                            NAME B01003_001E
```

1	27131070504	Census Tract 705.04, Rice County, Minnesota	3933
2	27131070400	Census Tract 704, Rice County, Minnesota	4511
3	27131070300	Census Tract 703, Rice County, Minnesota	4551
4	27131070503	Census Tract 705.03, Rice County, Minnesota	3348
5	27131070601	Census Tract 706.01, Rice County, Minnesota	3526
6	27131070800	Census Tract 708, Rice County, Minnesota	8101
7	27131070901	Census Tract 709.01, Rice County, Minnesota	5509
8	27131070700	Census Tract 707, Rice County, Minnesota	7165
9	27131070100	Census Tract 701, Rice County, Minnesota	7333
10	27131070602	Census Tract 706.02, Rice County, Minnesota	5211
11	27131070200	Census Tract 702, Rice County, Minnesota	5463
12	27131070902	Census Tract 709.02, Rice County, Minnesota	3160
13	27131070501	Census Tract 705.01, Rice County, Minnesota	4374
14	27131070501	Census Tract 705.01, Rice County, Minnesota	4272
15	27131070504	Census Tract 705.04, Rice County, Minnesota	3941
16	27131070801	Census Tract 708.01, Rice County, Minnesota	4456
17	27131070200	Census Tract 702, Rice County, Minnesota	5508
18	27131070701	Census Tract 707.01, Rice County, Minnesota	3057
19	27131070400	Census Tract 704, Rice County, Minnesota	4686
20	27131070300	Census Tract 703, Rice County, Minnesota	4737
21	27131070601	Census Tract 706.01, Rice County, Minnesota	3669
22	27131070102	Census Tract 701.02, Rice County, Minnesota	3786
23	27131070802	Census Tract 708.02, Rice County, Minnesota	3873
24	27131070702	Census Tract 707.02, Rice County, Minnesota	3872
25	27131070901	Census Tract 709.01, Rice County, Minnesota	5681
26	27131070503	Census Tract 705.03, Rice County, Minnesota	3185
27	27131070902	Census Tract 709.02, Rice County, Minnesota	2992
28	27131070101	Census Tract 701.01, Rice County, Minnesota	3428
29	27131070602	Census Tract 706.02, Rice County, Minnesota	5406
30	27131070902	Census Tract 709.02, Rice County, Minnesota	3212
31	27131070601	Census Tract 706.01, Rice County, Minnesota	3775
32	27131070503	Census Tract 705.03, Rice County, Minnesota	3035
33	27131070702	Census Tract 707.02, Rice County, Minnesota	3738
34	27131070901	Census Tract 709.01, Rice County, Minnesota	5858
35	27131070801	Census Tract 708.01, Rice County, Minnesota	4618
36	27131070501	Census Tract 705.01, Rice County, Minnesota	4242
37	27131070300	Census Tract 703, Rice County, Minnesota	4657
38	27131070200	Census Tract 702, Rice County, Minnesota	5419
39	27131070400	Census Tract 704, Rice County, Minnesota	4380
40	27131070701	Census Tract 707.01, Rice County, Minnesota	3028
41	27131070504	Census Tract 705.04, Rice County, Minnesota	3917
42	27131070101	Census Tract 701.01, Rice County, Minnesota	3417
43	27131070802	Census Tract 708.02, Rice County, Minnesota	3944

```
44 27131070102 Census Tract 701.02, Rice County, Minnesota        4201
45 27131070602 Census Tract 706.02, Rice County, Minnesota        5354
   B01003_001M B19013_001E B19013_001M                         geometry year
1          273       63989        9273 MULTIPOLYGON (((-93.19137 4... 2019
2          168       85952        2758 MULTIPOLYGON (((-93.40564 4... 2019
3          190       78343        4242 MULTIPOLYGON (((-93.52521 4... 2019
4          245       92321       14200 MULTIPOLYGON (((-93.16075 4... 2019
5          333       50368        9979 MULTIPOLYGON (((-93.17615 4... 2019
6          465       48403        7679 MULTIPOLYGON (((-93.29819 4... 2019
7          456       44417       10552 MULTIPOLYGON (((-93.30904 4... 2019
8          414       67868        9422 MULTIPOLYGON (((-93.27265 4... 2019
9          326       91667        8106 MULTIPOLYGON (((-93.52452 4... 2019
10         310       64479       12376 MULTIPOLYGON (((-93.22644 4... 2019
11         177      101359        4104 MULTIPOLYGON (((-93.5246 44... 2019
12         410       45230       12887 MULTIPOLYGON (((-93.30888 4... 2019
13         270       66188        9179 MULTIPOLYGON (((-93.16981 4... 2019
14         316       64792       13256 MULTIPOLYGON (((-93.16981 4... 2020
15         536       63500        7351 MULTIPOLYGON (((-93.1909 44... 2020
16         703       67625       23325 MULTIPOLYGON (((-93.29829 4... 2020
17         473      104011        5648 MULTIPOLYGON (((-93.5246 44... 2020
18         218       73750       13139 MULTIPOLYGON (((-93.26704 4... 2020
19         296       86094        3438 MULTIPOLYGON (((-93.40564 4... 2020
20         244       79068        4902 MULTIPOLYGON (((-93.52518 4... 2020
21         525       52936       10436 MULTIPOLYGON (((-93.17615 4... 2020
22         199       96023       13649 MULTIPOLYGON (((-93.44292 4... 2020
23         437       63924        8715 MULTIPOLYGON (((-93.28272 4... 2020
24         425       49811       16864 MULTIPOLYGON (((-93.27265 4... 2020
25         566       51595        9615 MULTIPOLYGON (((-93.30904 4... 2020
26         341      100516       11630 MULTIPOLYGON (((-93.16075 4... 2020
27         440       46750       15457 MULTIPOLYGON (((-93.30888 4... 2020
28         295      100563       15809 MULTIPOLYGON (((-93.52452 4... 2020
29         377       62078        5270 MULTIPOLYGON (((-93.22644 4... 2020
30         421       47059       15456 MULTIPOLYGON (((-93.30888 4... 2021
31         435       56319        4333 MULTIPOLYGON (((-93.17615 4... 2021
32         321      105952        8429 MULTIPOLYGON (((-93.16075 4... 2021
33         409       57126       13968 MULTIPOLYGON (((-93.27265 4... 2021
34         714       47344        9579 MULTIPOLYGON (((-93.30904 4... 2021
35         622       61193       23977 MULTIPOLYGON (((-93.29829 4... 2021
36         380       79063       15272 MULTIPOLYGON (((-93.16981 4... 2021
37         296       83911        7244 MULTIPOLYGON (((-93.52522 4... 2021
38         520      111711       10313 MULTIPOLYGON (((-93.5246 44... 2021
39         274       90179        4919 MULTIPOLYGON (((-93.40564 4... 2021
40         358       82500       20934 MULTIPOLYGON (((-93.26775 4... 2021
```

```
41         537      67219         9805 MULTIPOLYGON (((-93.1909 44... 2021
42         270     108490         1768 MULTIPOLYGON (((-93.52452 4... 2021
43         462      63679        12261 MULTIPOLYGON (((-93.28274 4... 2021
44         199      85789        20094 MULTIPOLYGON (((-93.44292 4... 2021
45         359      63835         4805 MULTIPOLYGON (((-93.22644 4... 2021
```

```
# Or a little more simply
2019:2021 |>
  purrr::map(MN_tract_data,
             county = "Rice",
             variables = c("B01003_001", "B19013_001")
            ) |>
  list_rbind()
```

```
Getting data from the 2015-2019 5-year ACS
Downloading feature geometry from the Census website.  To cache shapefiles for use in future

Getting data from the 2016-2020 5-year ACS

Downloading feature geometry from the Census website.  To cache shapefiles for use in future

Getting data from the 2017-2021 5-year ACS

Downloading feature geometry from the Census website.  To cache shapefiles for use in future

         GEOID                                        NAME B01003_001E
1  27131070504 Census Tract 705.04, Rice County, Minnesota        3933
2  27131070400    Census Tract 704, Rice County, Minnesota        4511
3  27131070300    Census Tract 703, Rice County, Minnesota        4551
4  27131070503 Census Tract 705.03, Rice County, Minnesota        3348
5  27131070601 Census Tract 706.01, Rice County, Minnesota        3526
6  27131070800    Census Tract 708, Rice County, Minnesota        8101
7  27131070901 Census Tract 709.01, Rice County, Minnesota        5509
8  27131070700    Census Tract 707, Rice County, Minnesota        7165
9  27131070100    Census Tract 701, Rice County, Minnesota        7333
10 27131070602 Census Tract 706.02, Rice County, Minnesota        5211
11 27131070200    Census Tract 702, Rice County, Minnesota        5463
12 27131070902 Census Tract 709.02, Rice County, Minnesota        3160
13 27131070501 Census Tract 705.01, Rice County, Minnesota        4374
14 27131070501 Census Tract 705.01, Rice County, Minnesota        4272
```

```
15 27131070504 Census Tract 705.04, Rice County, Minnesota       3941
16 27131070801 Census Tract 708.01, Rice County, Minnesota       4456
17 27131070200    Census Tract 702, Rice County, Minnesota       5508
18 27131070701 Census Tract 707.01, Rice County, Minnesota       3057
19 27131070400    Census Tract 704, Rice County, Minnesota       4686
20 27131070300    Census Tract 703, Rice County, Minnesota       4737
21 27131070601 Census Tract 706.01, Rice County, Minnesota       3669
22 27131070102 Census Tract 701.02, Rice County, Minnesota       3786
23 27131070802 Census Tract 708.02, Rice County, Minnesota       3873
24 27131070702 Census Tract 707.02, Rice County, Minnesota       3872
25 27131070901 Census Tract 709.01, Rice County, Minnesota       5681
26 27131070503 Census Tract 705.03, Rice County, Minnesota       3185
27 27131070902 Census Tract 709.02, Rice County, Minnesota       2992
28 27131070101 Census Tract 701.01, Rice County, Minnesota       3428
29 27131070602 Census Tract 706.02, Rice County, Minnesota       5406
30 27131070902 Census Tract 709.02, Rice County, Minnesota       3212
31 27131070601 Census Tract 706.01, Rice County, Minnesota       3775
32 27131070503 Census Tract 705.03, Rice County, Minnesota       3035
33 27131070702 Census Tract 707.02, Rice County, Minnesota       3738
34 27131070901 Census Tract 709.01, Rice County, Minnesota       5858
35 27131070801 Census Tract 708.01, Rice County, Minnesota       4618
36 27131070501 Census Tract 705.01, Rice County, Minnesota       4242
37 27131070300    Census Tract 703, Rice County, Minnesota       4657
38 27131070200    Census Tract 702, Rice County, Minnesota       5419
39 27131070400    Census Tract 704, Rice County, Minnesota       4380
40 27131070701 Census Tract 707.01, Rice County, Minnesota       3028
41 27131070504 Census Tract 705.04, Rice County, Minnesota       3917
42 27131070101 Census Tract 701.01, Rice County, Minnesota       3417
43 27131070802 Census Tract 708.02, Rice County, Minnesota       3944
44 27131070102 Census Tract 701.02, Rice County, Minnesota       4201
45 27131070602 Census Tract 706.02, Rice County, Minnesota       5354
   B01003_001M B19013_001E B19013_001M                       geometry year
1          273       63989        9273 MULTIPOLYGON (((-93.19137 4... 2019
2          168       85952        2758 MULTIPOLYGON (((-93.40564 4... 2019
3          190       78343        4242 MULTIPOLYGON (((-93.52521 4... 2019
4          245       92321       14200 MULTIPOLYGON (((-93.16075 4... 2019
5          333       50368        9979 MULTIPOLYGON (((-93.17615 4... 2019
6          465       48403        7679 MULTIPOLYGON (((-93.29819 4... 2019
7          456       44417       10552 MULTIPOLYGON (((-93.30904 4... 2019
8          414       67868        9422 MULTIPOLYGON (((-93.27265 4... 2019
9          326       91667        8106 MULTIPOLYGON (((-93.52452 4... 2019
10         310       64479       12376 MULTIPOLYGON (((-93.22644 4... 2019
11         177      101359        4104 MULTIPOLYGON (((-93.5246 44... 2019
```

```
12          410       45230        12887 MULTIPOLYGON (((-93.30888 4... 2019
13          270       66188         9179 MULTIPOLYGON (((-93.16981 4... 2019
14          316       64792        13256 MULTIPOLYGON (((-93.16981 4... 2020
15          536       63500         7351 MULTIPOLYGON (((-93.1909 44... 2020
16          703       67625        23325 MULTIPOLYGON (((-93.29829 4... 2020
17          473      104011         5648 MULTIPOLYGON (((-93.5246 44... 2020
18          218       73750        13139 MULTIPOLYGON (((-93.26704 4... 2020
19          296       86094         3438 MULTIPOLYGON (((-93.40564 4... 2020
20          244       79068         4902 MULTIPOLYGON (((-93.52518 4... 2020
21          525       52936        10436 MULTIPOLYGON (((-93.17615 4... 2020
22          199       96023        13649 MULTIPOLYGON (((-93.44292 4... 2020
23          437       63924         8715 MULTIPOLYGON (((-93.28272 4... 2020
24          425       49811        16864 MULTIPOLYGON (((-93.27265 4... 2020
25          566       51595         9615 MULTIPOLYGON (((-93.30904 4... 2020
26          341      100516        11630 MULTIPOLYGON (((-93.16075 4... 2020
27          440       46750        15457 MULTIPOLYGON (((-93.30888 4... 2020
28          295      100563        15809 MULTIPOLYGON (((-93.52452 4... 2020
29          377       62078         5270 MULTIPOLYGON (((-93.22644 4... 2020
30          421       47059        15456 MULTIPOLYGON (((-93.30888 4... 2021
31          435       56319         4333 MULTIPOLYGON (((-93.17615 4... 2021
32          321      105952         8429 MULTIPOLYGON (((-93.16075 4... 2021
33          409       57126        13968 MULTIPOLYGON (((-93.27265 4... 2021
34          714       47344         9579 MULTIPOLYGON (((-93.30904 4... 2021
35          622       61193        23977 MULTIPOLYGON (((-93.29829 4... 2021
36          380       79063        15272 MULTIPOLYGON (((-93.16981 4... 2021
37          296       83911         7244 MULTIPOLYGON (((-93.52522 4... 2021
38          520      111711        10313 MULTIPOLYGON (((-93.5246 44... 2021
39          274       90179         4919 MULTIPOLYGON (((-93.40564 4... 2021
40          358       82500        20934 MULTIPOLYGON (((-93.26775 4... 2021
41          537       67219         9805 MULTIPOLYGON (((-93.1909 44... 2021
42          270      108490         1768 MULTIPOLYGON (((-93.52452 4... 2021
43          462       63679        12261 MULTIPOLYGON (((-93.28274 4... 2021
44          199       85789        20094 MULTIPOLYGON (((-93.44292 4... 2021
45          359       63835         4805 MULTIPOLYGON (((-93.22644 4... 2021
```

One more example using an API key

Here's an example of getting data from a website that attempts to make imdb movie data available as an API.

Initial instructions:

- go to omdbapi.com under the API Key tab and request a free API key

- store your key as discussed earlier
- explore the examples at omdbapi.com

We will first obtain data about the movie Coco from 2017.

```
# I addeed: Sys.setenv(OMDB_KEY = "")
# I used the first line to store my OMDB API key in .Renviron
# Sys.setenv(OMDB_KEY = "paste my omdb key here")
myapikey <- Sys.getenv("OMDB_KEY")

# Find url exploring examples at omdbapi.com
url <- str_c("http://www.omdbapi.com/?t=Coco&y=2017&apikey=", myapikey)

coco <- GET(url)   # coco holds response from server
coco               # Status of 200 is good!
```

```
Response [http://www.omdbapi.com/?t=Coco&y=2017&apikey=d64645f3]
  Date: 2025-03-26 00:23
  Status: 200
  Content-Type: application/json; charset=utf-8
  Size: 1.04 kB
```

```
details <- content(coco, "parse")
details                          # get a list of 25 pieces of information
```

```
$Title
[1] "Coco"

$Year
[1] "2017"

$Rated
[1] "PG"

$Released
[1] "22 Nov 2017"

$Runtime
[1] "105 min"

$Genre
```

```
[1] "Animation, Adventure, Drama"

$Director
[1] "Lee Unkrich, Adrian Molina"

$Writer
[1] "Lee Unkrich, Jason Katz, Matthew Aldrich"

$Actors
[1] "Anthony Gonzalez, Gael García Bernal, Benjamin Bratt"

$Plot
[1] "Aspiring musician Miguel, confronted with his family's ancestral ban on music, enters t

$Language
[1] "English, Spanish"

$Country
[1] "United States, Mexico"

$Awards
[1] "Won 2 Oscars. 112 wins & 42 nominations total"

$Poster
[1] "https://m.media-amazon.com/images/M/MV5BMDIyM2E2NTAtMzlhNy00ZGUxLWI1NjgtZDY5MzhiMDc5NGU

$Ratings
$Ratings[[1]]
$Ratings[[1]]$Source
[1] "Internet Movie Database"

$Ratings[[1]]$Value
[1] "8.4/10"


$Ratings[[2]]
$Ratings[[2]]$Source
[1] "Rotten Tomatoes"

$Ratings[[2]]$Value
[1] "97%"
```

```
$Ratings[[3]]
$Ratings[[3]]$Source
[1] "Metacritic"

$Ratings[[3]]$Value
[1] "81/100"



$Metascore
[1] "81"

$imdbRating
[1] "8.4"

$imdbVotes
[1] "635,840"

$imdbID
[1] "tt2380307"

$Type
[1] "movie"

$DVD
[1] "N/A"

$BoxOffice
[1] "$210,460,015"

$Production
[1] "N/A"

$Website
[1] "N/A"

$Response
[1] "True"
```

```r
details$Year                    # how to access details
```

```
[1] "2017"
```

```
details[[2]]                          # since a list, another way to access
```

```
[1] "2017"
```

Now build a data set for a collection of movies

```
# Must figure out pattern in URL for obtaining different movies
#  - try searching for others
movies <- c("Coco", "Wonder+Woman", "Get+Out",
            "The+Greatest+Showman", "Thor:+Ragnarok")

# Set up empty tibble
omdb <- tibble(Title = character(), Rated = character(), Genre = character(),
       Actors = character(), Metascore = double(), imdbRating = double(),
       BoxOffice = double())

# Use for loop to run through API request process 5 times,
#   each time filling the next row in the tibble
#  - can do max of 1000 GETs per day
for(i in 1:5) {
  url <- str_c("http://www.omdbapi.com/?t=",movies[i],
               "&apikey=", myapikey)
  Sys.sleep(0.5)
  onemovie <- GET(url)
  details <- content(onemovie, "parse")
  omdb[i,1] <- details$Title
  omdb[i,2] <- details$Rated
  omdb[i,3] <- details$Genre
  omdb[i,4] <- details$Actors
  omdb[i,5] <- parse_number(details$Metascore)
  omdb[i,6] <- parse_number(details$imdbRating)
  omdb[i,7] <- parse_number(details$BoxOffice)   # no $ and ,'s
}

omdb
```

```
# A tibble: 5 x 7
  Title                Rated Genre          Actors Metascore imdbRating BoxOffice
  <chr>                <chr> <chr>          <chr>      <dbl>      <dbl>     <dbl>
1 Coco                 PG    Animation, A~ Antho~         81        8.4 210460015
2 Wonder Woman         PG-13 Action, Adve~ Gal G~         76        7.3 412845172
```

```
3 Get Out              R     Horror, Myst~ Danie~       85         7.8 176196665
4 The Greatest Showman PG    Biography, D~ Hugh ~       48         7.5 174340174
5 Thor: Ragnarok       PG-13 Action, Adve~ Chris~       74         7.9 315058289
```

```
#  could use stringr functions to further organize this data - separate
#    different genres, different actors, etc.
```

```
movies <- c("Up", "Cars", "Kung+Fu+Panda", "The+Emperor%27s+New+Groove", "Mulan")

omdb <- tibble(Title = character(), Released = character(), Runtime = character(), Plot = cha

for(i in 1:5) {
  url <- str_c("http://www.omdbapi.com/?t=",movies[i],
               "&apikey=", myapikey)
  Sys.sleep(0.5)
  onemovie <- GET(url)
  details <- content(onemovie, "parse")
  omdb[i,1] <- details$Title
  omdb[i,2] <- details$Released
  omdb[i,3] <- details$Runtime
  omdb[i,4] <- details$Plot
  omdb[i,5] <- parse_number(details$BoxOffice)
}
```

On Your Own (continued)

4. (Based on final project by Mary Wu and Jenna Graff, MSCS 264, Spring 2024). Start with a small data set on 56 national parks from kaggle, and supplement with columns for the park address (a single column including address, city, state, and zip code) and a list of available activities (a single character column with activities separated by commas) from the park websites themselves.

Preliminaries:

- Request API here
- Check out API guide

```
np_kaggle <- read_csv("Data/parks.csv")
```

You can download this .qmd file from here. Just hit the Download Raw File button.

Using rvest for web scraping

If you would like to assemble data from a website with no API, you can often acquire data using more brute force methods commonly called web scraping. Typically, this involves finding content inside HTML (Hypertext markup language) code used for creating webpages and web applications and the CSS (Cascading style sheets) language for customizing the appearance of webpages. We are used to reading data from .csv files.... but most websites have it stored in XML (like html, but for data). You can read more about it here if you're interested: https://www.w3schools.com/xml/default.asp

XML has a sort of tree or graph-like structure... so we can identify information by which `node` it belongs to (`html_nodes`) and then convert the content into something we can use in R (`html_text` or `html_table`).

Here's one quick example of web scraping. First check out the webpage https://www.cheese.com/by_type and then select Semi-Soft. We can drill into the html code for this webpage and find and store specific information (like cheese names)

```
session <- bow("https://www.cheese.com/by_type", force = TRUE)
result <- scrape(session, query=list(t="semi-soft", per_page=100)) |>
  html_node("#main-body") |>
  html_nodes("h3") |>
  html_text()
head(result)
```

```
[1] "American Cheese"      "Mozzarella"           "Taleggio"
[4] "Fontina Val d'Aosta" "Blue Cheese"          "Jarlsberg"
```

```
#> [1] "3-Cheese Italian Blend"  "Abbaye de Citeaux"
#> [3] "Abbaye du Mont des Cats" "Adelost"
#> [5] "ADL Brick Cheese"        "Ailsa Craig"
```

Four steps to scraping data with functions in the `rvest` library:

0. `robotstxt::paths_allowed()` Check if the website allows scraping, and then make sure we scrape "politely"
1. `read_html()`. Input the URL containing the data and turn the html code into an XML file (another markup format that's easier to work with).
2. `html_nodes()`. Extract specific nodes from the XML file by using the CSS path that leads to the content of interest. (use css="table" for tables.)
3. `html_text()`. Extract content of interest from nodes. Might also use `html_table()` etc.

Data scraping ethics

Before scraping, we should always check first whether the website allows scraping. We should also consider if there's any personal or confidential information, and we should be considerate to not overload the server we're scraping from.

Chapter 24 in R4DS provides a nice overview of some of the important issues to consider. A couple of highlights:

- be aware of terms of service, and, if available, the `robots.txt` file that some websites will publish to clarify what can and cannot be scraped and other constraints about scraping.
- use the `polite` package to scrape public, non-personal, and factual data in a respectful manner
- scrape with a good purpose and request only what you need; in particular, be extremely wary of personally identifiable information

See this article for more perspective on the ethics of data scraping.

When the data is already in table form:

In this example, we will scrape climate data from this website

The website already contains data in table form, so we use `html_nodes(. , css = "table")` and `html_table()`

```
# check that scraping is allowed (Step 0)
robotstxt::paths_allowed("https://www.usclimatedata.com/climate/minneapolis/minnesota/united-
```

```
 www.usclimatedata.com
```

```
[1] TRUE
```

```
# Step 1: read_html()
mpls <- read_html("https://www.usclimatedata.com/climate/minneapolis/minnesota/united-states/

# 2: html_nodes()
tables <- html_nodes(mpls, css = "table")
tables  # have to guesstimate which table contains climate info
```

```
{xml_nodeset (8)}
[1] <table id="monthly_table_one" class="table table-hover tablesaw tablesaw- ...
[2] <table id="monthly_table_two" class="table table-hover tablesaw tablesaw- ...
[3] <table class="table table-hover tablesaw tablesaw-mode-swipe mt-4 daily_t ...
[4] <table class="table table-hover tablesaw tablesaw-mode-swipe mt-4 history ...
[5] <table class="table table-striped table-hover tablesaw tablesaw-mode-swip ...
[6] <table class="table table-hover tablesaw geo_table">\n<thead><tr>\n<th> < ...
[7] <table class="table table-hover tablesaw datetime_table" data-tablesaw-hi ...
[8] <table class="table table-hover tablesaw monthly_summary_table" data-tabl ...
```

```r
# 3: html_table()
html_table(tables, header = TRUE, fill = TRUE)    # find the right table
```

```
[[1]]
# A tibble: 6 x 7
  ``                                       JanJa  FebFe  MarMa  AprAp  MayMa  JunJu
  <chr>                                    <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
1 Average high in ºF Av. high Hi            24     29     41     58     69     79
2 Average low in ºF Av. low Lo               8     13     24     37     49     59
3 Days with precipitation Days precip.~      8      7     11      9     11     13
4 Hours of sunshine Hours sun. Sun         140    166    200    231    272    302
5 Av. precipitation in inch Av. precip~      0.9    0.77   1.89   2.66   3.36   4.25
6 Av. snowfall in inch Snowfall Sn          12      8     10      3      0      0

[[2]]
# A tibble: 6 x 7
  ``                                        JulJu AugAu  SepSe  OctOc  NovNo  DecDe
  <chr>                                     <dbl> <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
1 Average high in ºF Av. high Hi            83     80     72     58     41     27
2 Average low in ºF Av. low Lo              64     62     52     40     26     12
3 Days with precipitation Days precip.~     10     10      9      8      8      8
4 Hours of sunshine Hours sun. Sun         343    296    237    193    115    112
5 Av. precipitation in inch Av. precip~      4.04   4.3    3.08   2.43   1.77   1.16
6 Av. snowfall in inch Snowfall Sn           0      0      0      1      9     12

[[3]]
# A tibble: 31 x 7
   Day    HighºF LowºF `Prec/moinch` `Prec/yrinch` `Snow/moinch` `Snow/yrinch`
   <chr>   <dbl> <dbl>         <dbl>         <dbl>         <dbl>         <dbl>
 1 1 Jan    23.8   8.3          0.04          0.04          0.39           1
 2 2 Jan    23.7   8.2          0.08          0.08          0.71           1.8
 3 3 Jan    23.6   8.1          0.12          0.12          1.1            2.8
```

```
 4 4 Jan    23.5   7.9           0.12           0.12           1.5            3.8
 5 5 Jan    23.5   7.8           0.16           0.16           1.81           4.6
 6 6 Jan    23.4   7.7           0.2            0.2            2.2            5.6
 7 7 Jan    23.4   7.6           0.24           0.24           2.6            6.6
 8 8 Jan    23.3   7.5           0.28           0.28           3.11           7.9
 9 9 Jan    23.3   7.4           0.28           0.28           3.5            8.9
10 10 Jan   23.3   7.3           0.31           0.31           3.9            9.9
# i 21 more rows

[[4]]
# A tibble: 26 x 6
   Day    HighºF LowºF Precip.inch Snowinch `Snow d.inch`
   <chr>   <dbl> <dbl> <chr>       <chr>            <dbl>
 1 01 Dec   32    19   0.07        1.61                 7
 2 02 Dec   27    12   0.00        0.00                 6
 3 03 Dec   37.9  19.9 0.00        0.00                 6
 4 04 Dec   39    24.1 0.00        0.00                 6
 5 05 Dec   37    21.9 0.00        0.00                 5
 6 06 Dec   32    17.1 0.00        0.00                 5
 7 07 Dec   42.1  21.9 0.00        0.00                 5
 8 08 Dec   41    30.9 0.00        0.00                 5
 9 09 Dec   34    -0.9 0.16        2.52                 5
10 10 Dec    8.1  -4   T           T                    7
# i 16 more rows

[[5]]
# A tibble: 9 x 4
  ``                                            `Dec 19`    ``    Normal
  <chr>                                         <chr>       <lgl> <chr>
1 "Average high temperature Av. high temp."     "29.9 ºF"   NA    "27 ºF"
2 "Average low temperature Av. low temp."       "14.6 ºF"   NA    "12 ºF"
3 "Total precipitation Total precip."           "0.39 inch" NA    "1.16 inch"
4 "Total snowfall Total snowfall"               "6.33 inch" NA    "12 inch"
5 ""                                            ""          NA    ""
6 "Highest max temperature Highest max temp."   "44.1 ºF"   NA    "-"
7 "Lowest max temperature Lowest max temp."     "8.1 ºF"    NA    "-"
8 "Highest min temperature Highest min temp."   "32.0 ºF"   NA    "-"
9 "Lowest min temperature Lowest min temp."     "-5.1 ºF"   NA    "-"

[[6]]
# A tibble: 10 x 3
   ``                   ``                 ``
   <chr>                <chr>              <lgl>
```

```
 1 Country              United States     NA
 2 State                Minnesota         NA
 3 County               Hennepin          NA
 4 City                 Minneapolis       NA
 5 Zip code             55401             NA
 6 Longitude            -93.27 dec. degr. NA
 7 Latitude             44.98 dec. degr.  NA
 8 Altitude - Elevation 840ft             NA
 9 ICAO                 -                 NA
10 IATA                 -                 NA

[[7]]
# A tibble: 6 x 3
  ``           ``               ``
  <chr>        <chr>            <lgl>
1 Local Time   07:23 PM         NA
2 Sunrise      07:05 AM         NA
3 Sunset       07:32 PM         NA
4 Day / Night  Day              NA
5 Timezone     Chicago -6:00    NA
6 Timezone DB  America/Chicago  NA

[[8]]
# A tibble: 6 x 2
  ``                           ``
  <chr>                        <chr>
1 Annual high temperature      55ºF
2 Annual low temperature       37ºF
3 Days per year with precip.   112 days
4 Annual hours of sunshine     2607 hours
5 Average annual precip.       30.61 inch
6 Av. annual snowfall          55 inch
```

```r
mpls_data1 <- html_table(tables, header = TRUE, fill = TRUE)[[1]]
mpls_data1
```

```
# A tibble: 6 x 7
  ``                                       JanJa FebFe MarMa AprAp MayMa JunJu
  <chr>                                    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Average high in ºF Av. high Hi             24    29    41    58    69    79
2 Average low in ºF Av. low Lo                8    13    24    37    49    59
3 Days with precipitation Days precip.~       8     7    11     9    11    13
```

```
4 Hours of sunshine Hours sun. Sun       140    166    200    231    272    302
5 Av. precipitation in inch Av. precip~    0.9    0.77   1.89   2.66   3.36   4.25
6 Av. snowfall in inch Snowfall Sn        12      8     10      3      0      0
```

```r
mpls_data2 <- html_table(tables, header = TRUE, fill = TRUE)[[2]]
mpls_data2
```

```
# A tibble: 6 x 7
  ``                                       JulJu AugAu  SepSe  OctOc  NovNo  DecDe
  <chr>                                    <dbl> <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
1 Average high in ºF Av. high Hi           83      80     72     58     41     27
2 Average low in ºF Av. low Lo             64      62     52     40     26     12
3 Days with precipitation Days precip.~    10      10      9      8      8      8
4 Hours of sunshine Hours sun. Sun        343     296    237    193    115    112
5 Av. precipitation in inch Av. precip~     4.04    4.3    3.08   2.43   1.77   1.16
6 Av. snowfall in inch Snowfall Sn          0       0      0      1      9     12
```

Now we wrap the 4 steps above into the `bow` and `scrape` functions from the `polite` package:

```r
session <- bow("https://www.usclimatedata.com/climate/minneapolis/minnesota/united-states/usm

result <- scrape(session) |>
  html_nodes(css = "table") |>
  html_table(header = TRUE, fill = TRUE)
mpls_data1 <- result[[1]]
mpls_data2 <- result[[2]]
```

Even after finding the correct tables, there may still be a lot of work to make it tidy!!!

[**Pause to Ponder:**] What is each line of code doing below?

```r
bind_cols(mpls_data1, mpls_data2) |>
  as_tibble() |>
  select(-`...8`) |>
  mutate(`...1` = str_extract(`...1`, "[^ ]+ [^ ]+ [^ ]+")) |>
  pivot_longer(cols = c(`JanJa`:`DecDe`),
               names_to = "month", values_to = "weather") |>
  pivot_wider(names_from = `...1`, values_from = weather) |>
  mutate(month = str_sub(month, 1, 3))  |>
  rename(avg_high = "Average high in",
         avg_low = "Average low in")
```

```
New names:
* `` -> `...1`
* `` -> `...8`
```

```
# A tibble: 12 x 7
   month avg_high avg_low `Days with precipitation` `Hours of sunshine`
   <chr>    <dbl>   <dbl>                     <dbl>               <dbl>
 1 Jan         24       8                         8                 140
 2 Feb         29      13                         7                 166
 3 Mar         41      24                        11                 200
 4 Apr         58      37                         9                 231
 5 May         69      49                        11                 272
 6 Jun         79      59                        13                 302
 7 Jul         83      64                        10                 343
 8 Aug         80      62                        10                 296
 9 Sep         72      52                         9                 237
10 Oct         58      40                         8                 193
11 Nov         41      26                         8                 115
12 Dec         27      12                         8                 112
# i 2 more variables: `Av. precipitation in` <dbl>, `Av. snowfall in` <dbl>
```

```
# Probably want to rename the rest of the variables too!
```

Leaflet mapping example with data in table form

Let's return to our example from `02_maps.qmd` where we recreated an interactive choropleth map of population densities by US state. Recall how that plot was very suspicious? The population density data that came with the state geometries from our source seemed incorrect.

Let's see if we can use our new web scraping skills to scrape the correct population density data and repeat that plot! Can we go out and find the real statewise population densities, create a tidy data frame, merge that with our state geometry shapefiles, and then regenerate our plot?

A quick wikipedia search yields this webpage with more reasonable population densities in a nice table format. Let's see if we can grab this data using our 4 steps to `rvest`ing data!

```
# check that scraping is allowed (Step 0)
robotstxt::paths_allowed("https://en.wikipedia.org/wiki/List_of_states_and_territories_of_the
```

```
 en.wikipedia.org
```

```
[1] TRUE
```

```
# Step 1: read_html()
pop_dens <- read_html("https://en.wikipedia.org/wiki/List_of_states_and_territories_of_the_Un

# 2: html_nodes()
tables <- html_nodes(pop_dens, css = "table")
tables  # have to guesstimate which table contains our desired info
```

```
{xml_nodeset (2)}
[1] <table class="wikitable sortable plainrowheaders sticky-header-multi stat ...
[2] <table class="nowraplinks hlist mw-collapsible mw-collapsed navbox-inner" ...
```

```
# 3: html_table()
html_table(tables, header = TRUE, fill = TRUE)    # find the right table
```

```
[[1]]
# A tibble: 61 x 6
   Location                Density Density Population `Land area` `Land area`
   <chr>                   <chr>   <chr>   <chr>      <chr>       <chr>
 1 Location                /mi2    /km2    Population mi2         km2
 2 District of Columbia    11,131  4,297   678,972    61          158
 3 New Jersey              1,263   488     9,290,841  7,354       19,047
 4 Rhode Island            1,060   409     1,095,962  1,034       2,678
 5 Puerto Rico             936     361     3,205,691  3,424       8,868
 6 Massachusetts           898     347     7,001,399  7,800       20,202
 7 Guam[4]                 824     319     172,952    210         543
 8 Connecticut             747     288     3,617,176  4,842       12,542
 9 U.S. Virgin Islands[4]  737     284     98,750     134         348
10 Maryland                637     246     6,180,253  9,707       25,142
# i 51 more rows

[[2]]
# A tibble: 11 x 2
   .mw-parser-output .navbar{display:inline;font-size:8~1 .mw-parser-output .n~2
   <chr>                                                  <chr>
 1 "List of states and territories of the United States"  "List of states and t~
 2 "Demographics"                                         "Population\nAfrican ~
 3 "Economy"                                              "Billionaires\nBudget~
 4 "Environment"                                          "Botanical gardens\nC~
 5 "Geography"                                            "Area\nBays\nBeaches\~
```

```
 6 "Government"                                            "Agriculture commissi~
 7 "Health"                                                "Changes in life expe~
 8 "History"                                               "Date of statehood\nN~
 9 "Law"                                                   "Abortion\nAge of con~
10 "Miscellaneous"                                         "Abbreviations\nAirpo~
11 "Category\n Commons\n Portals"                          "Category\n Commons\n~
# i abbreviated names:
#   1: `.mw-parser-output .navbar{display:inline;font-size:88%;font-weight:normal}.mw-parser-
#   2: `.mw-parser-output .navbar{display:inline;font-size:88%;font-weight:normal}.mw-parser-
```

```
density_table <- html_table(tables, header = TRUE, fill = TRUE)[[1]]
density_table
```

```
# A tibble: 61 x 6
   Location               Density Density Population `Land area` `Land area`
   <chr>                  <chr>   <chr>   <chr>      <chr>       <chr>
 1 Location               /mi2    /km2    Population mi2         km2
 2 District of Columbia   11,131  4,297   678,972    61          158
 3 New Jersey             1,263   488     9,290,841  7,354       19,047
 4 Rhode Island           1,060   409     1,095,962  1,034       2,678
 5 Puerto Rico            936     361     3,205,691  3,424       8,868
 6 Massachusetts          898     347     7,001,399  7,800       20,202
 7 Guam[4]                824     319     172,952    210         543
 8 Connecticut            747     288     3,617,176  4,842       12,542
 9 U.S. Virgin Islands[4] 737     284     98,750     134         348
10 Maryland               637     246     6,180,253  9,707       25,142
# i 51 more rows
```

```
# Perform Steps 0-3 using the polite package
session <- bow("https://en.wikipedia.org/wiki/List_of_states_and_territories_of_the_United_S

result <- scrape(session) |>
  html_nodes(css = "table") |>
  html_table(header = TRUE, fill = TRUE)
density_table <- result[[1]]
density_table
```

```
# A tibble: 61 x 6
   Location               Density Density Population `Land area` `Land area`
   <chr>                  <chr>   <chr>   <chr>      <chr>       <chr>
 1 Location               /mi2    /km2    Population mi2         km2
```

```
 2 District of Columbia   11,131  4,297   678,972    61          158
 3 New Jersey             1,263   488     9,290,841  7,354       19,047
 4 Rhode Island           1,060   409     1,095,962  1,034       2,678
 5 Puerto Rico            936     361     3,205,691  3,424       8,868
 6 Massachusetts          898     347     7,001,399  7,800       20,202
 7 Guam[4]                824     319     172,952    210         543
 8 Connecticut            747     288     3,617,176  4,842       12,542
 9 U.S. Virgin Islands[4] 737     284     98,750     134         348
10 Maryland               637     246     6,180,253  9,707       25,142
# i 51 more rows
```

Even after grabbing our table from wikipedia and setting it in a nice tibble format, there is still some cleaning to do before we can merge this with our state geometries:

```
density_data <- density_table |>
  select(1, 2, 4, 5) |>
  filter(!row_number() == 1) |>
  rename(Land_area = `Land area`) |>
  mutate(state_name = str_to_lower(as.character(Location)),
         Density = parse_number(Density),
         Population = parse_number(Population),
         Land_area = parse_number(Land_area)) |>
  select(-Location)
density_data
```

```
# A tibble: 60 x 4
   Density Population Land_area state_name
     <dbl>      <dbl>     <dbl> <chr>
 1   11131     678972        61 district of columbia
 2    1263    9290841      7354 new jersey
 3    1060    1095962      1034 rhode island
 4     936    3205691      3424 puerto rico
 5     898    7001399      7800 massachusetts
 6     824     172952       210 guam[4]
 7     747    3617176      4842 connecticut
 8     737      98750       134 u.s. virgin islands[4]
 9     637    6180253      9707 maryland
10     578      43915        76 american samoa[4]
# i 50 more rows
```

As before, we get core geometry data to draw US states and then we'll make sure we can merge our new density data into the core files.

```r
# Get info to draw US states for geom_polygon (connect the lat-long points)
states_polygon <- as_tibble(map_data("state")) |>
  select(region, group, order, lat, long)

# See what the state (region) levels look like in states_polygon
unique(states_polygon$region)
```

```
 [1] "alabama"              "arizona"              "arkansas"
 [4] "california"           "colorado"             "connecticut"
 [7] "delaware"             "district of columbia" "florida"
[10] "georgia"              "idaho"                "illinois"
[13] "indiana"              "iowa"                 "kansas"
[16] "kentucky"             "louisiana"            "maine"
[19] "maryland"             "massachusetts"        "michigan"
[22] "minnesota"            "mississippi"          "missouri"
[25] "montana"              "nebraska"             "nevada"
[28] "new hampshire"        "new jersey"           "new mexico"
[31] "new york"             "north carolina"       "north dakota"
[34] "ohio"                 "oklahoma"             "oregon"
[37] "pennsylvania"         "rhode island"         "south carolina"
[40] "south dakota"         "tennessee"            "texas"
[43] "utah"                 "vermont"              "virginia"
[46] "washington"           "west virginia"        "wisconsin"
[49] "wyoming"
```

```r
# Get info to draw US states for geom_sf and leaflet (simple features
#   object with multipolygon geometry column)
states_sf <- read_sf("https://rstudio.github.io/leaflet/json/us-states.geojson") |>
  select(name, geometry)

# See what the state (name) levels look like in states_sf
unique(states_sf$name)
```

```
 [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] "Puerto Rico"
```

```r
# See what the state (state_name) levels look like in density_data
unique(density_data$state_name)
```

```
 [1] "district of columbia"        "new jersey"
 [3] "rhode island"                "puerto rico"
 [5] "massachusetts"               "guam[4]"
 [7] "connecticut"                 "u.s. virgin islands[4]"
 [9] "maryland"                    "american samoa[4]"
[11] "delaware"                    "florida"
[13] "new york"                    "pennsylvania"
[15] "ohio"                        "northern mariana islands[4]"
[17] "california"                  "illinois"
[19] "hawaii"                      "north carolina"
[21] "virginia"                    "georgia"
[23] "indiana"                     "south carolina"
[25] "michigan"                    "tennessee"
[27] "new hampshire"               "washington"
[29] "texas"                       "kentucky"
[31] "wisconsin"                   "louisiana"
[33] "alabama"                     "missouri"
[35] "west virginia"               "minnesota"
[37] "vermont"                     "arizona"
[39] "mississippi"                 "oklahoma"
[41] "arkansas"                    "iowa"
[43] "colorado"                    "maine"
[45] "oregon"                      "utah"
[47] "kansas"                      "nevada"
[49] "nebraska"                    "idaho"
[51] "new mexico"                  "south dakota"
[53] "north dakota"                "montana"
[55] "wyoming"                     "alaska"
```

```
[57] "contiguous us"                 "50 states"
[59] "50 states and dc"              "united states"
```

```
# all lower case plus some extraneous rows
```

```
# Make sure all keys have the same format before joining: all lower case

states_sf <- states_sf |>
  mutate(name = str_to_lower(name))
```

```
# Now we can merge data sets together for the static and the interactive plots

# Merge with states_polygon (static)
density_polygon <- states_polygon |>
  left_join(density_data, by = c("region" = "state_name"))
density_polygon
```

```
# A tibble: 15,537 x 8
   region  group order   lat  long Density Population Land_area
   <chr>   <dbl> <int> <dbl> <dbl>   <dbl>      <dbl>     <dbl>
 1 alabama     1     1  30.4 -87.5     101    5108468     50645
 2 alabama     1     2  30.4 -87.5     101    5108468     50645
 3 alabama     1     3  30.4 -87.5     101    5108468     50645
 4 alabama     1     4  30.3 -87.5     101    5108468     50645
 5 alabama     1     5  30.3 -87.6     101    5108468     50645
 6 alabama     1     6  30.3 -87.6     101    5108468     50645
 7 alabama     1     7  30.3 -87.6     101    5108468     50645
 8 alabama     1     8  30.3 -87.6     101    5108468     50645
 9 alabama     1     9  30.3 -87.7     101    5108468     50645
10 alabama     1    10  30.3 -87.8     101    5108468     50645
# i 15,527 more rows
```

```
# Looks like merge worked for 48 contiguous states plus DC
density_polygon |>
  group_by(region) |>
  summarise(mean = mean(Density)) |>
  print(n = Inf)
```

```
# A tibble: 49 x 2
   region                  mean
   <chr>                  <dbl>
```

1	alabama	101
2	arizona	65
3	arkansas	59
4	california	250
5	colorado	57
6	connecticut	747
7	delaware	529
8	district of columbia	11131
9	florida	422
10	georgia	192
11	idaho	24
12	illinois	226
13	indiana	192
14	iowa	57
15	kansas	36
16	kentucky	115
17	louisiana	106
18	maine	45
19	maryland	637
20	massachusetts	898
21	michigan	178
22	minnesota	72
23	mississippi	63
24	missouri	90
25	montana	7.8
26	nebraska	26
27	nevada	29
28	new hampshire	157
29	new jersey	1263
30	new mexico	17
31	new york	415
32	north carolina	223
33	north dakota	11
34	ohio	288
35	oklahoma	59
36	oregon	44
37	pennsylvania	290
38	rhode island	1060
39	south carolina	179
40	south dakota	12
41	tennessee	173
42	texas	117
43	utah	42

```
44 vermont                70
45 virginia              221
46 washington            118
47 west virginia          74
48 wisconsin             109
49 wyoming                 6
```

```r
# Remove DC since such an outlier
density_polygon <- density_polygon |>
  filter(region != "district of columbia")


# Merge with states_sf (static or interactive)
density_sf <- states_sf |>
  left_join(density_data, by = c("name" = "state_name")) |>
  filter(!(name %in% c("alaska", "hawaii")))

# Looks like merge worked for 48 contiguous states plus DC and PR
class(density_sf)
```

```
[1] "sf"         "tbl_df"     "tbl"        "data.frame"
```

```r
print(density_sf, n = Inf)
```

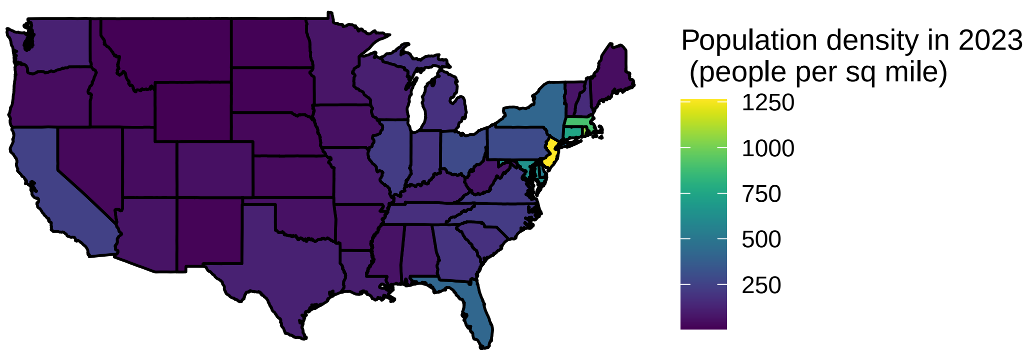
```
Simple feature collection with 50 features and 4 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -124.7066 ymin: 17.92956 xmax: -65.6268 ymax: 49.38362
Geodetic CRS:  WGS 84
# A tibble: 50 x 5
   name                                   geometry Density Population Land_area
 * <chr>                        <MULTIPOLYGON [°]>   <dbl>      <dbl>     <dbl>
 1 alabama              (((-87.3593 35.00118, -85.~   101      5108468     50645
 2 arizona              (((-109.0425 37.00026, -10~    65      7431344    113594
 3 arkansas             (((-94.47384 36.50186, -90~    59      3067732     52035
 4 california           (((-123.2333 42.00619, -12~   250     38965193    155779
 5 colorado             (((-107.9197 41.00391, -10~    57      5877610    103642
 6 connecticut          (((-73.05353 42.03905, -71~   747      3617176      4842
 7 delaware             (((-75.41409 39.80446, -75~   529      1031890      1949
 8 district of columbia (((-77.03526 38.99387, -76~ 11131       678972        61
 9 florida              (((-85.49714 30.99754, -85~   422     22610726     53625
```

10	georgia	(((-83.10919 35.00118, -83~	192	11029227	57513
11	idaho	(((-116.0475 49.00024, -11~	24	1964726	82643
12	illinois	(((-90.63998 42.51006, -88~	226	12549689	55519
13	indiana	(((-85.99006 41.75972, -84~	192	6862199	35826
14	iowa	(((-91.36842 43.50139, -91~	57	3207004	55857
15	kansas	(((-101.906 40.00163, -95.~	36	2940546	81759
16	kentucky	(((-83.90335 38.76931, -83~	115	4526154	39486
17	louisiana	(((-93.60849 33.01853, -91~	106	4573749	43204
18	maine	(((-70.70392 43.05776, -70~	45	1395722	30843
19	maryland	(((-75.99465 37.95325, -76~	637	6180253	9707
20	massachusetts	(((-70.91752 42.88797, -70~	898	7001399	7800
21	michigan	(((-83.45424 41.73234, -84~	178	10037261	56539
22	minnesota	(((-92.0147 46.7054, -92.0~	72	5737915	79627
23	mississippi	(((-88.47111 34.9957, -88.~	63	2939690	46923
24	missouri	(((-91.83396 40.60957, -91~	90	6196156	68742
25	montana	(((-104.0475 49.00024, -10~	7.8	1132812	145546
26	nebraska	(((-103.3246 43.00299, -10~	26	1978379	76824
27	nevada	(((-117.0279 42.00071, -11~	29	3194176	109781
28	new hampshire	(((-71.08183 45.3033, -71.~	157	1402054	8953
29	new jersey	(((-74.23655 41.14083, -73~	1263	9290841	7354
30	new mexico	(((-107.4213 37.00026, -10~	17	2114371	121298
31	new york	(((-73.34381 45.01303, -73~	415	19571216	47126
32	north carolina	(((-80.97866 36.56211, -80~	223	10835491	48618
33	north dakota	(((-97.22874 49.00024, -97~	11	783926	69001
34	ohio	(((-80.5186 41.9788, -80.5~	288	11785935	40861
35	oklahoma	(((-100.0877 37.00026, -94~	59	4053824	68595
36	oregon	(((-123.2113 46.17414, -12~	44	4233358	95988
37	pennsylvania	(((-79.76278 42.25265, -79~	290	12961683	44743
38	rhode island	(((-71.19684 41.67757, -71~	1060	1095962	1034
39	south carolina	(((-82.76414 35.0669, -82.~	179	5373555	30061
40	south dakota	(((-104.0475 45.94411, -96~	12	919318	75811
41	tennessee	(((-88.05487 36.49638, -88~	173	7126489	41235
42	texas	(((-101.8129 36.50186, -10~	117	30503301	261232
43	utah	(((-112.1644 41.99523, -11~	42	3417734	82170
44	vermont	(((-71.50355 45.01303, -71~	70	647464	9217
45	virginia	(((-75.39766 38.0135, -75.~	221	8715698	39490
46	washington	(((-117.0334 49.00024, -11~	118	7812880	66456
47	west virginia	(((-80.5186 40.63695, -80.~	74	1770071	24038
48	wisconsin	(((-90.41543 46.56848, -90~	109	5910955	54158
49	wyoming	(((-109.0808 45.00207, -10~	6	584057	97093
50	puerto rico	(((-66.44834 17.98433, -66~	936	3205691	3424

```
# Remove DC and PR
density_sf <- density_sf |>
  filter(name != "district of columbia" & name != "puerto rico")
```

Numeric variable (static plot):

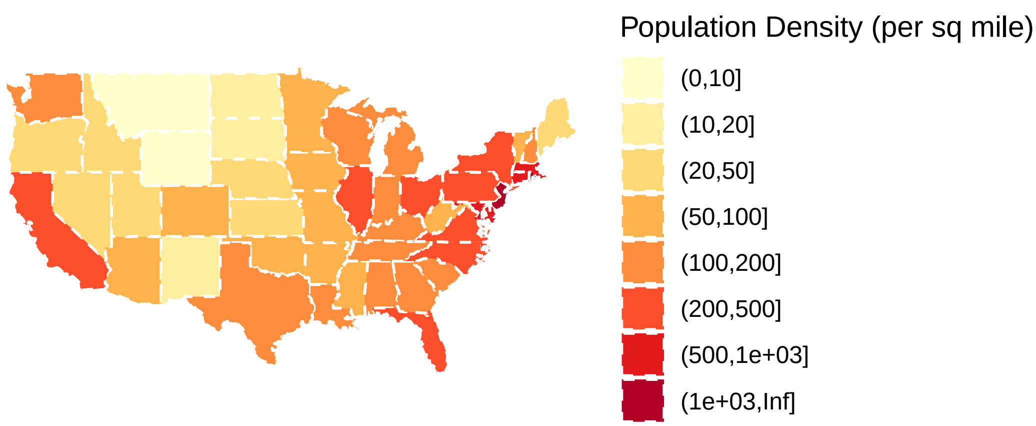
```
density_polygon |>
  ggplot(mapping = aes(x = long, y = lat, group = group)) +
    geom_polygon(aes(fill = Density), color = "black") +
    labs(fill = "Population density in 2023 \n (people per sq mile)") +
    coord_map() +
    theme_void() +
    scale_fill_viridis()
```



Remember that the original plot classified densities into our own pre-determined bins before plotting - this might look better!

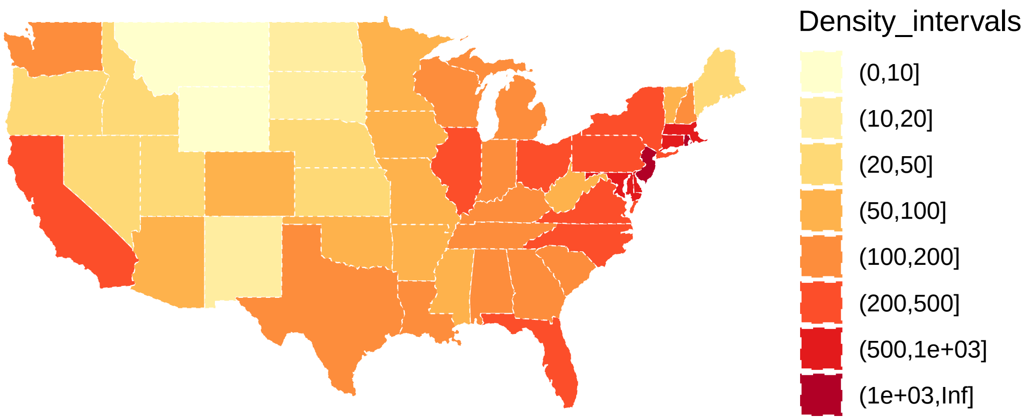
```
density_polygon <- density_polygon |>
  mutate(Density_intervals = cut(Density, n = 8,
          breaks = c(0, 10, 20, 50, 100, 200, 500, 1000, Inf)))

density_polygon |>
```

```r
  ggplot(mapping = aes(x = long, y = lat, group = group)) +
    geom_polygon(aes(fill = Density_intervals), color = "white",
                 linetype = 2) +
    labs(fill = "Population Density (per sq mile)") +
    coord_map() +
    theme_void() +
    scale_fill_brewer(palette = "YlOrRd")
```



We could even create a static plot using `geom_sf()` using `density_sf`:

```r
density_sf <- density_sf |>
  mutate(Density_intervals = cut(Density, n = 8,
          breaks = c(0, 10, 20, 50, 100, 200, 500, 1000, Inf)))

ggplot(data = density_sf) +
  geom_sf(aes(fill = Density_intervals), colour = "white", linetype = 2) +
  theme_void() +
  scale_fill_brewer(palette = "YlOrRd")
```

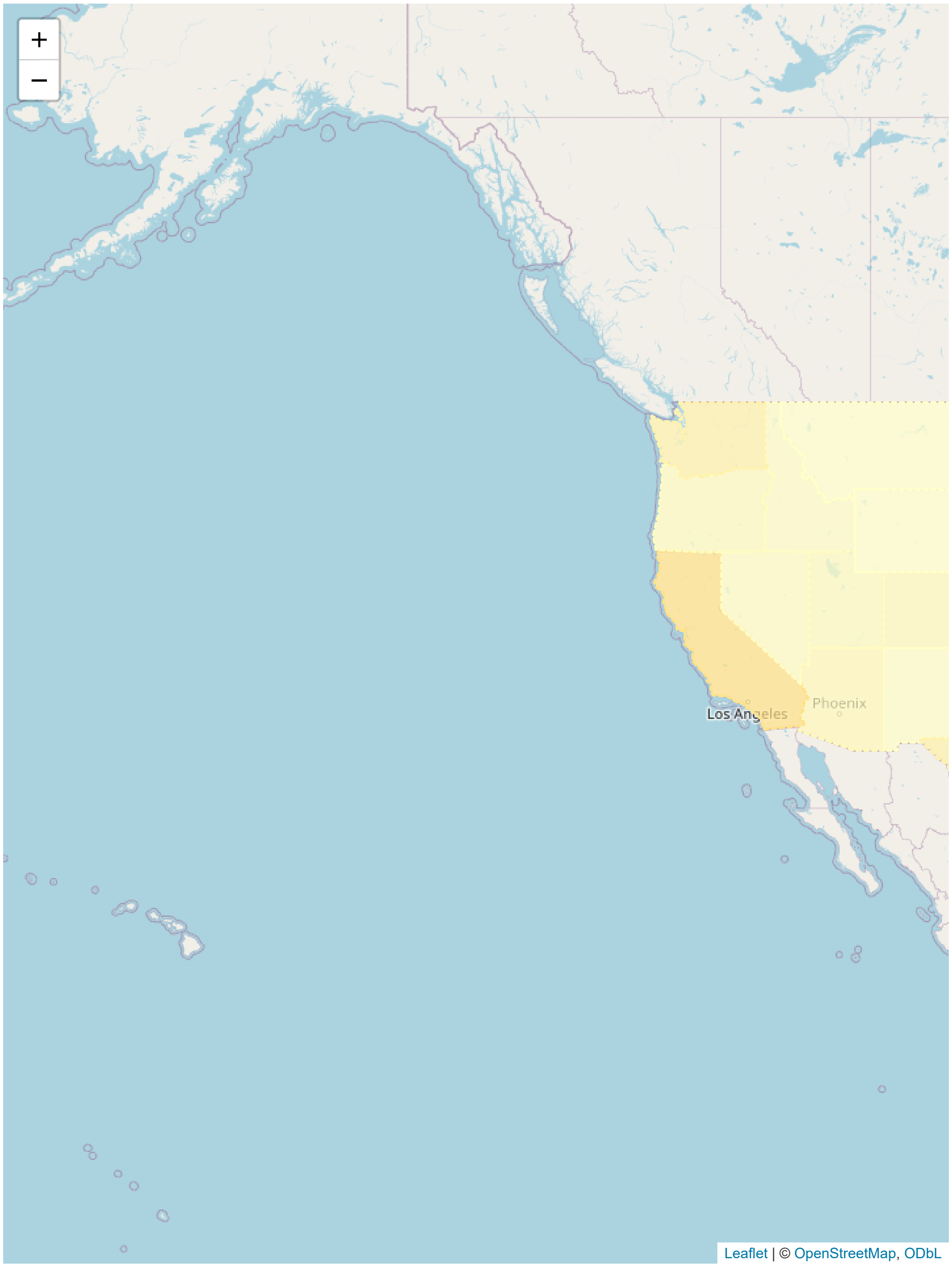


But… why not make an interactive plot instead?

```
density_sf <- density_sf |>
  mutate(labels = str_c(name, ": ", Density, " people per sq mile in 2023"))

labels <- lapply(density_sf$labels, HTML)
pal <- colorNumeric("YlOrRd", density_sf$Density)

leaflet(density_sf) |>
  setView(-96, 37.8, 4) |>
  addTiles() |>
  addPolygons(
    weight = 2,
    opacity = 1,
    color = ~ pal(density_sf$Density),
    dashArray = "3",
    fillOpacity = 0.7,
    highlightOptions = highlightOptions(
      weight = 5,
      color = "#666",
      dashArray = "",
      fillOpacity = 0.7,
      bringToFront = TRUE),
```

```
    label = labels,
    labelOptions = labelOptions(
      style = list("font-weight" = "normal", padding = "3px 8px"),
      textsize = "15px",
      direction = "auto"))
```

file:///C:\Users\charl\AppData\Local\Temp\RtmpWeplPp\file91bc6389c5\widget91bc5e2842aa.html s

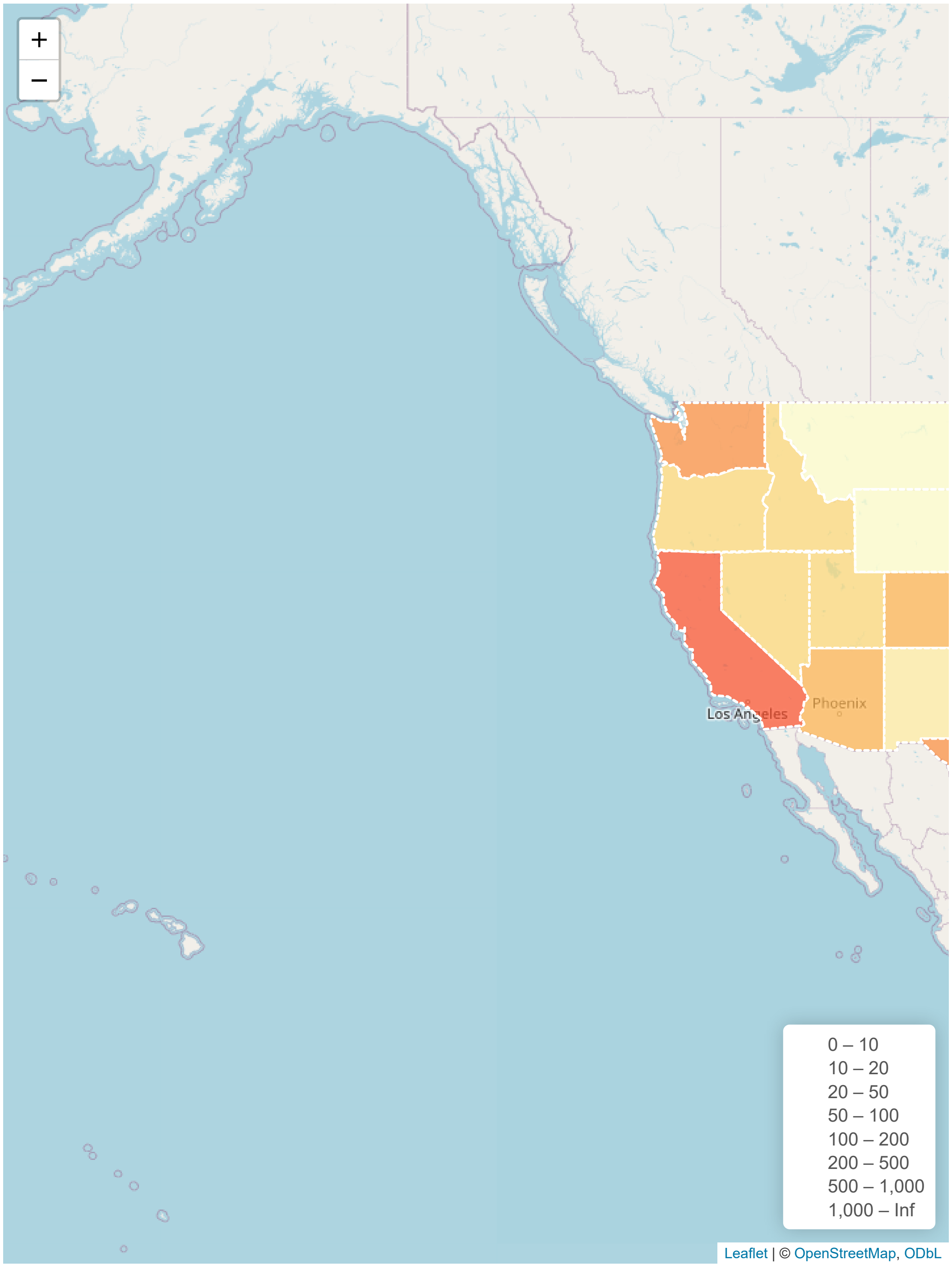

+
−
Los Angeles
Phoenix
Leaflet | © OpenStreetMap, ODbL

```
# should use addLegend() but not trivial without pre-set bins
```

Here's an interactive plot with our own bins:

```
# Create our own category bins for population densities
#   and assign the yellow-orange-red color palette
bins <- c(0, 10, 20, 50, 100, 200, 500, 1000, Inf)
pal <- colorBin("YlOrRd", domain = density_sf$Density, bins = bins)

# Create labels that pop up when we hover over a state.  The labels must
#   be part of a list where each entry is tagged as HTML code.
density_sf <- density_sf |>
  mutate(labels = str_c(name, ": ", Density, " people / sq mile"))
labels <- lapply(density_sf$labels, HTML)

# If want more HTML formatting, use these lines instead of those above:
# states <- states |>
#   mutate(labels = glue("<strong>{name}</strong><br/>{density} people /
#   mi<sup>2</sup>"))
# labels <- lapply(states$labels, HTML)

leaflet(density_sf) |>
  setView(-96, 37.8, 4) |>
  addTiles() |>
  addPolygons(
    fillColor = ~pal(Density),
    weight = 2,
    opacity = 1,
    color = "white",
    dashArray = "3",
    fillOpacity = 0.7,
    highlightOptions = highlightOptions(
      weight = 5,
      color = "#666",
      dashArray = "",
      fillOpacity = 0.7,
      bringToFront = TRUE),
    label = labels,
    labelOptions = labelOptions(
      style = list("font-weight" = "normal", padding = "3px 8px"),
      textsize = "15px",
      direction = "auto")) |>
```

```
  addLegend(pal = pal, values = ~Density, opacity = 0.7, title = NULL,
    position = "bottomright")
```


+
−
Los Angeles
Phoenix
0 – 10
10 – 20
20 – 50
50 – 100
100 – 200
200 – 500
500 – 1,000
1,000 – Inf
Leaflet | © OpenStreetMap, ODbL

On Your Own

1. Use the `rvest` package and `html_table` to read in the table of data found at the link here and create a scatterplot of land area versus the 2022 estimated population. I give you some starter code below; fill in the "???" and be sure you can explain what EVERY line of code does and why it's necessary.

#| eval: FALSE

city_pop <- read_html("https://en.wikipedia.org/wiki/List_of_United_States_cities_by_population")

pop <- html_nodes(???, ???) html_table(pop, header = TRUE, fill = TRUE) # find right table pop2 <- html_table(pop, header = TRUE, fill = TRUE)[[???]] pop2

perform the steps above with the polite package

session <- bow("https://en.wikipedia.org/wiki/List_of_United_States_cities_by_population", force = TRUE)

result <- scrape(session) |> html_nodes(???) |> html_table(header = TRUE, fill = TRUE) pop2 <- result[[???]] pop2

pop3 <- as_tibble(pop2[,c(1:6,8)]) |> slice(???) |> rename(`State` = `ST`, `Estimate2023` = `2023estimate`, `Census` = `2020census`, `Area` = `2020 land area`, `Density` = `2020 density`) |> mutate(Estimate2023 = parse_number(Estimate2023), Census = parse_number(Census), Change = ??? # get rid of % but preserve +/-, Area = parse_number(Area), Density = parse_number(Density)) |> mutate(City = str_replace(City, "\[.*$“,” “)) pop3

pick out unusual points

outliers <- pop3 |> filter(Estimate2023 > ??? | Area > ???)

This will work if don't turn variables from chr to dbl, but in that

case notice how axes are just evenly spaced categorical variables

ggplot(pop3, aes(x = ???, y = ???)) + geom_point() + geom_smooth() + ggrepel::geom_label_repel(data = ???, aes(label = ???))

2. We would like to create a tibble with 4 years of data (2001-2004) from the Minnesota Wild hockey team. Specifically, we are interested in the "Scoring Regular Season" table from this webpage and the similar webpages from 2002, 2003, and 2004. Your final tibble should have 6 columns: player, year, age, pos (position), gp (games played), and pts (points).

You should (a) write a function called `hockey_stats` with inputs for team and year to scrape data from the "scoring Regular Season" table, and (b) use iteration techniques to scrape and combine 4 years worth of data. Here are some functions you might consider:

- `row_to_names(row_number = 1)` from the `janitor` package
- `clean_names()` also from the `janitor` package
- `bow()` and `scrape()` from the `polite` package
- `str_c()` from the `stringr` package (for creating urls with user inputs)
- `map2()` and `list_rbind()` for iterating and combining years

Try following these steps:

1) Be sure you can find and clean the correct table from the 2021 season.

```
# Step 0: Check that scraping is allowed
robotstxt::paths_allowed("https://www.hockey-reference.com/teams/MIN/2001.html")
```

```
 www.hockey-reference.com

[1] TRUE
```

```
# Step 1: read_html()
hockey_page <- read_html("https://www.hockey-reference.com/teams/MIN/2001.html")

# Step 2: html_nodes()
tables <- html_nodes(hockey_page, css = "table")
tables  # have to guesstimate which table contains our desired info
```

```
{xml_nodeset (6)}
[1] <table class="sortable stats_table" id="team_stats" data-cols-to-freeze=" ...
[2] <table class="sortable stats_table" id="team_stats_adv" data-cols-to-free ...
[3] <table class="sortable stats_table" id="roster" data-cols-to-freeze=",2"> ...
[4] <table class="stats_table sortable per_toggler soc" id="player_stats" dat ...
[5] <table class="stats_table sortable per_toggler soc" id="goalie_stats" dat ...
[6] <table class="stats_table sortable per_toggler soc" id="stats_misc_plus"  ...
```

```r
# Step 3: html_table()
html_table(tables, header = TRUE, fill = TRUE)    # find the right table
```

```
[[1]]
# A tibble: 2 x 29
  Team  AvAge    GP     W     L     T    OL   PTS `PTS%`    GF    GA   SRS   SOS
  <chr> <dbl> <int> <int> <int> <int> <int> <int>  <dbl> <int> <int> <dbl> <dbl>
1 Minn~  27.4    82    25    39    13     5    68  0.415   168   210 -0.42  0.09
2 Leag~  27.8    82    36    32    10     4    86  0.525   226   226 NA    NA
# i 16 more variables: `GF/G` <dbl>, `GA/G` <dbl>, PP <int>, PPO <int>,
#   `PP%` <dbl>, PPA <int>, PPOA <int>, `PK%` <dbl>, SH <int>, SHA <int>,
#   S <int>, `S%` <dbl>, SA <int>, `SV%` <dbl>, PDO <lgl>, SO <int>

[[2]]
# A tibble: 2 x 22
  Team  `S%`  `SV%` PDO   CF    CA    `CF%` xGF   xGA   aGF   aGA   axDiff SCF
  <chr> <lgl> <lgl> <lgl> <lgl> <lgl> <lgl> <lgl> <lgl> <lgl> <lgl> <lgl>  <lgl>
1 Minn~ NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA     NA
2 Leag~ NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA     NA
# i 9 more variables: SCA <lgl>, `SCF%` <lgl>, HDF <lgl>, HDA <lgl>,
#   `HDF%` <lgl>, HDGF <lgl>, `HDC%` <lgl>, HDGA <lgl>, `HDCO%` <lgl>

[[3]]
# A tibble: 38 x 11
   No.   Player    Birth Pos     Age Ht       Wt `S/C` Exp   `Birth Date` Summary
   <chr> <chr>     <chr> <chr> <int> <chr> <int> <chr> <chr> <chr>        <chr>
 1 40    Chris A~ ca CA D        25 6-0     205 L/-   R     June 26, 19~ 0 G, 0~
 2 45    Peter B~ cs CS RW       27 6-0     185 R/-   R     September 5~ 4 G, 2~
 3 3     Ladisla~ cs CS D        25 6-2     190 L/-   1     March 24, 1~ 2 G, 5~
 4 31    Zac Bie~ ca CA G        24 6-5     205 -/L   3     September 1~ 0-1-0,~
 5 36    Sylvain~ ca CA LW       26 6-2     215 L/-   3     May 21, 1974 3 G, 2~
 6 5     Brad Bo~ ca CA D        28 6-1     205 L/-   3     May 5, 1972  0 G, 1~
 7 32    Brian B~ us US LW       27 5-10    186 L/-   1     November 28~ 0 G, 0~
 8 15    J.J. Da~ ca CA D        35 5-10    192 L/-   15    October 12,~ 0 G, 0~
 9 34    Jim Dowd us US C        32 6-0     180 R/-   9     December 25~ 7 G, 2~
10 11    Pascal ~ ca CA LW       21 6-1     205 L/-   R     April 7, 19~ 1 G, 0~
# i 28 more rows

[[4]]
# A tibble: 40 x 22
   ``    ``    ``    ``    ``    Scoring Scoring Scoring ``    ``    Goals Goals
   <chr> <chr> <chr> <chr> <chr> <chr>   <chr>   <chr>   <chr> <chr> <chr> <chr>
```

```
 1 Rk    Play~ Age   Pos   GP    G       A       PTS     +/-   PIM   EVG   PPG
 2 1     Scot~ 31    RW    58    11      28      39      6     45    7     2
 3 2     Mari~ 18    LW    71    18      18      36      -6    32    12    6
 4 3     Ľubo~ 32    D     80    11      23      34      -8    52    7     4
 5 4     Wes ~ 30    C     82    18      12      30      -8    37    11    0
 6 5     Fili~ 24    D     75    9       21      30      -6    28    5     4
 7 6     Darb~ 28    LW    72    18      11      29      1     36    14    3
 8 7     Jim ~ 32    C     68    7       22      29      -6    80    7     0
 9 8     Antt~ 27    LW    82    12      16      28      -7    24    10    0
10 9     Stac~ 26    C     76    7       20      27      3     20    6     1
# i 30 more rows
# i 10 more variables: Goals <chr>, Goals <chr>, Assists <chr>, Assists <chr>,
#   Assists <chr>, Shots <chr>, Shots <chr>, `Ice Time` <chr>,
#   `Ice Time` <chr>, `` <chr>

[[5]]
# A tibble: 6 x 23
  ``    ``     ``    `Goalie Stats` `Goalie Stats` `Goalie Stats` `Goalie Stats`
  <chr> <chr>  <chr> <chr>          <chr>          <chr>          <chr>
1 "Rk"  Player "Age" GP             W              L              T/O
2 "1"   Jamie~ "29"  38             5              23             9
3 "2"   Manny~ "26"  42             19             17             4
4 "3"   Derek~ "21"  4              1              3              0
5 "4"   Zac B~ "24"  1              0              1              0
6 ""    Team ~ ""    82             25             44             13
# i 16 more variables: `Goalie Stats` <chr>, `Goalie Stats` <chr>,
#   `Goalie Stats` <chr>, `Goalie Stats` <chr>, `Goalie Stats` <chr>,
#   `Goalie Stats` <chr>, `Goalie Stats` <chr>, `Goalie Stats` <chr>,
#   `Goalie Stats` <chr>, `Goalie Stats` <chr>, `Goalie Stats` <chr>,
#   Scoring <chr>, Scoring <chr>, Scoring <chr>, `` <chr>, `` <chr>

[[6]]
# A tibble: 35 x 18
   ``    ``                 ``    ``    ``    Adjusted Adjusted Adjusted Adjusted
   <chr> <chr>              <chr> <chr> <chr> <chr>    <chr>    <chr>    <chr>
 1 Rk    Player             Age   Pos   GP    G        A        PTS      GC
 2 1     Scott Pellerin     31    RW    58    12       30       42       14.7
 3 2     Marián Gáborík     18    LW    71    20       19       39       16.1
 4 3     Ľubomír Sekeráš    32    D     80    12       25       37       13.4
 5 4     Wes Walz           30    C     82    20       13       33       14.5
 6 5     Filip Kuba         24    D     75    10       22       32       11.5
 7 6     Jim Dowd           32    C     68    8        23       31       10.6
 8 7     Darby Hendrickson  28    LW    72    20       12       32       14.2
```

```
 9 8     Antti Laaksonen   27    LW    82    13        17        30        11.7
10 9     Stacy Roest       26    C     76    8         21        29        10.1
# i 25 more rows
# i 9 more variables: `Plus/Minus` <chr>, `Plus/Minus` <chr>,
#   `Plus/Minus` <chr>, `Plus/Minus` <chr>, `Plus/Minus` <chr>,
#   `Point Shares` <chr>, `Point Shares` <chr>, `Point Shares` <chr>, `` <chr>
```

```
hockey_table <- html_table(tables, header = TRUE, fill = TRUE)[[1]]
hockey_table
```

```
# A tibble: 2 x 29
  Team  AvAge    GP     W     L     T    OL   PTS `PTS%`    GF    GA   SRS   SOS
  <chr> <dbl> <int> <int> <int> <int> <int> <int>  <dbl> <int> <int> <dbl> <dbl>
1 Minn~  27.4    82    25    39    13     5    68  0.415   168   210 -0.42  0.09
2 Leag~  27.8    82    36    32    10     4    86  0.525   226   226 NA    NA
# i 16 more variables: `GF/G` <dbl>, `GA/G` <dbl>, PP <int>, PPO <int>,
#   `PP%` <dbl>, PPA <int>, PPOA <int>, `PK%` <dbl>, SH <int>, SHA <int>,
#   S <int>, `S%` <dbl>, SA <int>, `SV%` <dbl>, PDO <lgl>, SO <int>
```

2) Organize your `rvest` code from (1) into functions from the `polite` package.

```
session <- bow("https://www.hockey-reference.com/teams/MIN/2001.html", force = TRUE)

result <- scrape(session) |>
  html_nodes(css = "table") |>
  html_table(header = TRUE, fill = TRUE)
```

```
No encoding supplied: defaulting to UTF-8.
```

```
hockey_table <- result[[1]]
hockey_table
```

```
# A tibble: 2 x 29
  Team  AvAge    GP     W     L     T    OL   PTS `PTS%`    GF    GA   SRS   SOS
  <chr> <dbl> <int> <int> <int> <int> <int> <int>  <dbl> <int> <int> <dbl> <dbl>
1 Minn~  27.4    82    25    39    13     5    68  0.415   168   210 -0.42  0.09
2 Leag~  27.8    82    36    32    10     4    86  0.525   226   226 NA    NA
# i 16 more variables: `GF/G` <dbl>, `GA/G` <dbl>, PP <int>, PPO <int>,
#   `PP%` <dbl>, PPA <int>, PPOA <int>, `PK%` <dbl>, SH <int>, SHA <int>,
#   S <int>, `S%` <dbl>, SA <int>, `SV%` <dbl>, PDO <lgl>, SO <int>
```

3) Place the code from (2) into a function where the user can input a team and year. You would then adjust the url accordingly and produce a clean table for the user.

```
hockey_stats <- function(team, year){
  base_front_url <- "https://www.hockey-reference.com/teams/"
  url <- str_c(base_front_url, team, "/", year, ".html")
  session <- bow(url, force = TRUE)

  result <- scrape(session) |>
    html_nodes(css = "table") |>
    html_table(header = TRUE, fill = TRUE)
  hockey_table <- result[[1]]
  hockey_table
}
```

```
hockey_stats("MIN", "2001")
```

```
No encoding supplied: defaulting to UTF-8.

# A tibble: 2 x 29
  Team  AvAge    GP     W     L     T    OL   PTS `PTS%`    GF    GA   SRS   SOS
  <chr> <dbl> <int> <int> <int> <int> <int> <int>  <dbl> <int> <int> <dbl> <dbl>
1 Minn~  27.4    82    25    39    13     5    68  0.415   168   210 -0.42  0.09
2 Leag~  27.8    82    36    32    10     4    86  0.525   226   226 NA    NA
# i 16 more variables: `GF/G` <dbl>, `GA/G` <dbl>, PP <int>, PPO <int>,
#   `PP%` <dbl>, PPA <int>, PPOA <int>, `PK%` <dbl>, SH <int>, SHA <int>,
#   S <int>, `S%` <dbl>, SA <int>, `SV%` <dbl>, PDO <lgl>, SO <int>
```

4) Use `map2` and `list_rbind` to build one data set containing Minnesota Wild data from 2001-2004.

```
specific_years <- c("2001","2002","2003","2004")
mn_hockey_data <- map2("MIN", specific_years, hockey_stats) |>
  list_rbind()
```

```
No encoding supplied: defaulting to UTF-8.
No encoding supplied: defaulting to UTF-8.
No encoding supplied: defaulting to UTF-8.
No encoding supplied: defaulting to UTF-8.
```

You can download this .qmd file from here. Just hit the Download Raw File button.

Credit to Brianna Heggeseth and Leslie Myint from Macalester College for a few of these descriptions and examples.

Using rvest for web scraping

Please see `08_table_scraping.qmd` for a preview of web scraping techniques when no API exists, along with ethical considerations when scraping data. In this file, we will turn to scenarios when the webpage contains data of interest, but it is not already in table form.

Recall the four steps to scraping data with functions in the `rvest` library:

0. `robotstxt::paths_allowed()` Check if the website allows scraping, and then make sure we scrape "politely"
1. `read_html()`. Input the URL containing the data and turn the html code into an XML file (another markup format that's easier to work with).
2. `html_nodes()`. Extract specific nodes from the XML file by using the CSS path that leads to the content of interest. (use css="table" for tables.)
3. `html_text()`. Extract content of interest from nodes. Might also use `html_table()` etc.

More scraping ethics

`robots.txt`

`robots.txt` is a file that some websites will publish to clarify what can and cannot be scraped and other constraints about scraping. When a website publishes this file, this we need to comply with the information in it for moral and legal reasons.

We will look through the information in this tutorial and apply this to the NIH robots.txt file.

From our investigation of the NIH `robots.txt`, we learn:

- `User-agent: *`: Anyone is allowed to scrape
- `Crawl-delay: 2`: Need to wait 2 seconds between each page scraped
- No `Visit-time` entry: no restrictions on time of day that scraping is allowed
- No `Request-rate` entry: no restrictions on simultaneous requests
- No mention of `?page=`, `news-events`, `news-releases`, or `https://science.education.nih.gov/` in the `Disallow` sections. (This is what we want to scrape today.)

robotstxt package

We can also use functions from the `robotstxt` package, which was built to download and parse robots.txt files (more info). Specifically, the `paths_allowed()` function can check if a bot has permission to access certain pages.

A timeout to preview some technical ideas

HTML structure

HTML (hypertext markup language) is the formatting language used to create webpages. We can see the core parts of HTML from the rvest vignette.

Finding CSS Selectors

In order to gather information from a webpage, we must learn the language used to identify patterns of specific information. For example, on the NIH News Releases page, we can see that the data is represented in a consistent pattern of image + title + abstract.

We will identify data in a web page using a pattern matching language called CSS Selectors that can refer to specific patterns in HTML, the language used to write web pages.

For example:

- Selecting by tag:
  - `"a"` selects all hyperlinks in a webpage ("a" represents "anchor" links in HTML)
  - `"p"` selects all paragraph elements
- Selecting by ID and class:
  - `".description"` selects all elements with `class` equal to "description"
    - The `.` at the beginning is what signifies `class` selection.
    - This is one of the most common CSS selectors for scraping because in HTML, the `class` attribute is extremely commonly used to format webpage elements. (Any number of HTML elements can have the same `class`, which is not true for the `id` attribute.)
  - `"#mainTitle"` selects the SINGLE element with **id** equal to "mainTitle"
    - The `#` at the beginning is what signifies `id` selection.

```
<p class="title">Title of resource 1</p>
<p class="description">Description of resource 1</p>

<p class="title">Title of resource 2</p>
<p class="description">Description of resource 2</p>
```

Warning: Websites change often! So if you are going to scrape a lot of data, it is probably worthwhile to save and date a copy of the website. Otherwise, you may return after some time and your scraping code will include all of the wrong CSS selectors.

SelectorGadget

Although you can learn how to use CSS Selectors by hand, we will use a shortcut by installing the Selector Gadget tool.

- There is a version available for Chrome–add it to Chrome via the Chome Web Store.
  - Make sure to pin the extension to the menu bar. (Click the 3 dots > Extensions > Manage extensions. Click the "Details" button under SelectorGadget and toggle the "Pin to toolbar" option.)
- There is also a version that can be saved as a bookmark in the browser–see here.

You might watch the Selector Gadget tutorial video.

Case Study: NIH News Releases

Our goal is to build a data frame with the article title, publication date, and abstract text for the 50 most recent NIH news releases.

Head over to the NIH News Releases page. Click the Selector Gadget extension icon or bookmark button. As you mouse over the webpage, different parts will be highlighted in orange. Click on the title (but not the live link portion!) of the first news release. You'll notice that the Selector Gadget information in the lower right describes what you clicked on. (If SelectorGadget ever highlights too much in green, you can click on portions that you do not want to turn them red.)

Scroll through the page to verify that only the information you intend (the description paragraph) is selected. The selector panel shows the CSS selector (`.teaser-title`) and the number of matches for that CSS selector (10). (You may have to be careful with your clicking–there are two overlapping boxes, and clicking on the link of the title can lead to the CSS selector of "a".)

[Pause to Ponder:] Repeat the process above to find the correct selectors for the following fields. Make sure that each matches 10 results:

- The publication date

  .date-display-single

- The article abstract paragraph (which will also include the publication date)

  .teaser-description

Retrieving Data Using `rvest` and CSS Selectors

Now that we have identified CSS selectors for the information we need, let's fetch the data using the `rvest` package similarly to our approach in `08_table_scraping.qmd`.

```
# check that scraping is allowed (Step 0)
robotstxt::paths_allowed("https://www.nih.gov/news-events/news-releases")
```

```
 www.nih.gov
```

```
[1] TRUE
```

```
# Step 1: Download the HTML and turn it into an XML file with read_html()
nih <- read_html("https://www.nih.gov/news-events/news-releases")
```

Finding the exact node (e.g. ".teaser-title") is the tricky part. Among all the html code used to produce a webpage, where do you go to grab the content of interest? This is where SelectorGadget comes to the rescue!

```
# Step 2: Extract specific nodes with html_nodes()
title_temp <- html_nodes(nih, ".teaser-title")
title_temp
```

```
{xml_nodeset (10)}
 [1] <h4 class="teaser-title"><a href="/news-events/news-releases/nih-researc ...
 [2] <h4 class="teaser-title"><a href="/news-events/news-releases/study-illum ...
 [3] <h4 class="teaser-title"><a href="/news-events/news-releases/nih-funded- ...
 [4] <h4 class="teaser-title"><a href="/news-events/news-releases/surgery-kid ...
 [5] <h4 class="teaser-title"><a href="/news-events/news-releases/nih-sponsor ...
 [6] <h4 class="teaser-title"><a href="/news-events/news-releases/topical-ste ...
 [7] <h4 class="teaser-title"><a href="/news-events/news-releases/tecovirimat ...
 [8] <h4 class="teaser-title"><a href="/news-events/news-releases/nih-central ...
 [9] <h4 class="teaser-title"><a href="/news-events/news-releases/nih-funded- ...
[10] <h4 class="teaser-title"><a href="/news-events/news-releases/longer-brea ...
```

```
# Step 3: Extract content from nodes with html_text(), html_name(),
#    html_attrs(), html_children(), html_table(), etc.
# Usually will still need to do some stringr adjustments
title_vec <- html_text(title_temp)
title_vec
```

```
 [1] "NIH researchers develop eye drops that slow vision loss in animals"
 [2] "Study illuminates the structural features of memory formation at cellular and subcellul
 [3] "NIH-funded study identifies potential new stroke treatment"
 [4] "Surgery in kids with mild sleep-disordered breathing tied to fewer doctor visits, meds"
 [5] "NIH-sponsored trial of Lassa vaccine opens"
 [6] "Topical steroid withdrawal diagnostic criteria defined by NIH researchers"
 [7] "Tecovirimat is safe but ineffective as treatment for clade II mpox"
 [8] "NIH centralizes peer review to improve efficiency and strengthen integrity "
 [9] "NIH-funded research team engineers new drug targeting pain sensation pathway"
[10] "Longer breastfeeding linked to blood-pressure lowering effects of certain infant gut ba
```

You can also write this altogether with a pipe:

```r
robotstxt::paths_allowed("https://www.nih.gov/news-events/news-releases")
```

```
 www.nih.gov

[1] TRUE
```

```r
read_html("https://www.nih.gov/news-events/news-releases") |>
  html_nodes(".teaser-title") |>
  html_text()
```

```
 [1] "NIH researchers develop eye drops that slow vision loss in animals"
 [2] "Study illuminates the structural features of memory formation at cellular and subcellul
 [3] "NIH-funded study identifies potential new stroke treatment"
 [4] "Surgery in kids with mild sleep-disordered breathing tied to fewer doctor visits, meds"
 [5] "NIH-sponsored trial of Lassa vaccine opens"
 [6] "Topical steroid withdrawal diagnostic criteria defined by NIH researchers"
 [7] "Tecovirimat is safe but ineffective as treatment for clade II mpox"
 [8] "NIH centralizes peer review to improve efficiency and strengthen integrity "
 [9] "NIH-funded research team engineers new drug targeting pain sensation pathway"
[10] "Longer breastfeeding linked to blood-pressure lowering effects of certain infant gut ba
```

And finally we wrap the 4 steps above into the `bow` and `scrape` functions from the `polite` package:

```r
session <- bow("https://www.nih.gov/news-events/news-releases", force = TRUE)

nih_title <- scrape(session) |>
  html_nodes(".teaser-title") |>
  html_text()
nih_title
```

```
 [1] "NIH researchers develop eye drops that slow vision loss in animals"
 [2] "Study illuminates the structural features of memory formation at cellular and subcellul
 [3] "NIH-funded study identifies potential new stroke treatment"
 [4] "Surgery in kids with mild sleep-disordered breathing tied to fewer doctor visits, meds'
 [5] "NIH-sponsored trial of Lassa vaccine opens"
 [6] "Topical steroid withdrawal diagnostic criteria defined by NIH researchers"
 [7] "Tecovirimat is safe but ineffective as treatment for clade II mpox"
 [8] "NIH centralizes peer review to improve efficiency and strengthen integrity "
 [9] "NIH-funded research team engineers new drug targeting pain sensation pathway"
[10] "Longer breastfeeding linked to blood-pressure lowering effects of certain infant gut ba
```

Putting multiple columns of data together.

Now repeat the process above to extract the publication date and the abstract.

```r
nih_pubdate <- scrape(session) |>
  html_nodes(".date-display-single") |>
  html_text()
nih_pubdate
```

```
 [1] "March 21, 2025" "March 20, 2025" "March 17, 2025" "March 17, 2025"
 [5] "March 17, 2025" "March 14, 2025" "March 12, 2025" "March 6, 2025"
 [9] "March 5, 2025"  "March 4, 2025"
```

```r
nih_description <- scrape(session) |>
  html_nodes(".teaser-description") |>
  html_text()
nih_description
```

```
 [1] "March 21, 2025 -     \n          Treatment shows potential to slow the progression of I
 [2] "March 20, 2025 -     \n          NIH-funded study uses cutting-edge imaging techniques
 [3] "March 17, 2025 -     \n          Preclinical study in rodents suggests that uric acid i
```

```
 [4] "March 17, 2025 -     \n          NIH-funded study supports use of adenotonsillectomy i
 [5] "March 17, 2025 -     \n          Lassa fever is a viral hemorrhagic disease that can b
 [6] "March 14, 2025 -     \n          Criteria may help guide treatment of dermatitis. "
 [7] "March 12, 2025 -     \n          NIH-sponsored trial data offer further evidence to he
 [8] "March 6, 2025 -     \n          The proposed approach is expected to save more than $6
 [9] "March 5, 2025 -     \n          Study of CB1 receptor has implications for chronic pai
[10] "March 4, 2025 -     \n          Nursing for at least six months may spur beneficial gu
```

Combine these extracted variables into a single tibble. Make sure the variables are formatted correctly - e.g. `pubdate` has `date` type, `description` does not contain the `pubdate`, etc.

```r
# use tibble() to put multiple columns together into a tibble
nih_top10 <- tibble(title = nih_title,
                    pubdate = nih_pubdate,
                    description = nih_description)
nih_top10
```

```
# A tibble: 10 x 3
   title                                                     pubdate description
   <chr>                                                     <chr>   <chr>
 1 "NIH researchers develop eye drops that slow vision loss~ March ~ "March 21,~
 2 "Study illuminates the structural features of memory for~ March ~ "March 20,~
 3 "NIH-funded study identifies potential new stroke treatm~ March ~ "March 17,~
 4 "Surgery in kids with mild sleep-disordered breathing ti~ March ~ "March 17,~
 5 "NIH-sponsored trial of Lassa vaccine opens"              March ~ "March 17,~
 6 "Topical steroid withdrawal diagnostic criteria defined ~ March ~ "March 14,~
 7 "Tecovirimat is safe but ineffective as treatment for cl~ March ~ "March 12,~
 8 "NIH centralizes peer review to improve efficiency and s~ March ~ "March 6, ~
 9 "NIH-funded research team engineers new drug targeting p~ March ~ "March 5, ~
10 "Longer breastfeeding linked to blood-pressure lowering ~ March ~ "March 4, ~
```

```r
# now clean the data
nih_top10 <- nih_top10 |>
  mutate(pubdate = mdy(pubdate),
         description = str_trim(str_replace(description, ".*\\n", "")))
nih_top10
```

```
# A tibble: 10 x 3
   title                                                  pubdate    description
   <chr>                                                  <date>     <chr>
 1 "NIH researchers develop eye drops that slow vision l~ 2025-03-21 Treatment ~
```

```
 2 "Study illuminates the structural features of memory ~ 2025-03-20 NIH-funded~
 3 "NIH-funded study identifies potential new stroke tre~ 2025-03-17 Preclinica~
 4 "Surgery in kids with mild sleep-disordered breathing~ 2025-03-17 NIH-funded~
 5 "NIH-sponsored trial of Lassa vaccine opens"           2025-03-17 Lassa feve~
 6 "Topical steroid withdrawal diagnostic criteria defin~ 2025-03-14 Criteria m~
 7 "Tecovirimat is safe but ineffective as treatment for~ 2025-03-12 NIH-sponso~
 8 "NIH centralizes peer review to improve efficiency an~ 2025-03-06 The propos~
 9 "NIH-funded research team engineers new drug targetin~ 2025-03-05 Study of C~
10 "Longer breastfeeding linked to blood-pressure loweri~ 2025-03-04 Nursing fo~
```

NOW - continue this process to build a tibble with the most recent 50 NIH news releases, which will require that you iterate over 5 webpages! You should write at least one function, and you will need iteration–use both a `for` loop and appropriate `map_()` functions from `purrr`. Some additional hints:

- Mouse over the page buttons at the very bottom of the news home page to see what the URLs look like.
- Include `Sys.sleep(2)` in your function to respect the `Crawl-delay: 2` in the NIH `robots.txt` file.
- Recall that `bind_rows()` from `dplyr` takes a list of data frames and stacks them on top of each other.

[**Pause to Ponder:**] Create a function to scrape a single NIH press release page by filling missing pieces labeled `???`:

```
# Helper function to reduce html_nodes() |> html_text() code duplication
get_text_from_page <- function(page, css_selector) {
  page |>
    html_nodes(css_selector) |>
    html_text()
}

# Main function to scrape and tidy desired attributes
scrape_page <- function(url) {
    Sys.sleep(2)
    page <- read_html(url)
    article_titles <- get_text_from_page(page, ".teaser-title")
    article_dates <- get_text_from_page(page, ".date-display-single")
    article_dates <- mdy(article_dates)
    article_description <- get_text_from_page(page, ".teaser-description")
    article_description <- str_trim(str_replace(article_description,
                                                ".*\\n",
                                                "")
```

```
                                    )

    tibble(
      title = article_titles,
      dates = article_dates,
      description = article_description
    )
}

scrape_page("https://www.nih.gov/news-events/news-releases")
```

```
# A tibble: 10 x 3
   title                                                  dates      description
   <chr>                                                  <date>     <chr>
 1 "NIH researchers develop eye drops that slow vision l~ 2025-03-21 Treatment ~
 2 "Study illuminates the structural features of memory ~ 2025-03-20 NIH-funded~
 3 "NIH-funded study identifies potential new stroke tre~ 2025-03-17 Preclinica~
 4 "Surgery in kids with mild sleep-disordered breathing~ 2025-03-17 NIH-funded~
 5 "NIH-sponsored trial of Lassa vaccine opens"           2025-03-17 Lassa feve~
 6 "Topical steroid withdrawal diagnostic criteria defin~ 2025-03-14 Criteria m~
 7 "Tecovirimat is safe but ineffective as treatment for~ 2025-03-12 NIH-sponso~
 8 "NIH centralizes peer review to improve efficiency an~ 2025-03-06 The propos~
 9 "NIH-funded research team engineers new drug targetin~ 2025-03-05 Study of C~
10 "Longer breastfeeding linked to blood-pressure loweri~ 2025-03-04 Nursing fo~
```

[**Pause to Ponder:**] Use a for loop over the first 5 pages:

```
pages <- vector("list", length = 6)
pos <- 0

for (i in 2025:2024) {
  for (j in 0:2) {
    pos <- pos + 1
    url <- str_c("https://www.nih.gov/news-events/news-releases?", i,
                 "&page=", j, "&1=")
    pages[[pos]] <- scrape_page(url)
  }
}

df_articles <- bind_rows(pages)
head(df_articles)
```

```
# A tibble: 6 x 3
  title                                                 dates      description
  <chr>                                                 <date>     <chr>
1 NIH researchers develop eye drops that slow vision los~ 2025-03-21 Treatment ~
2 Study illuminates the structural features of memory fo~ 2025-03-20 NIH-funded~
3 NIH-funded study identifies potential new stroke treat~ 2025-03-17 Preclinica~
4 Surgery in kids with mild sleep-disordered breathing t~ 2025-03-17 NIH-funded~
5 NIH-sponsored trial of Lassa vaccine opens             2025-03-17 Lassa feve~
6 Topical steroid withdrawal diagnostic criteria defined~ 2025-03-14 Criteria m~
```

[**Pause to Ponder:**] Use map functions in the purrr package:

```r
library(purrr)

base_url <- "https://www.nih.gov/news-events/news-releases?page="
urls_all_pages <- str_c(base_url, seq(0,5))

pages2 <- purrr::map(urls_all_pages, scrape_page)
df_articles2 <- bind_rows(pages2)
head(df_articles2)
```

```
# A tibble: 6 x 3
  title                                                 dates      description
  <chr>                                                 <date>     <chr>
1 NIH researchers develop eye drops that slow vision los~ 2025-03-21 Treatment ~
2 Study illuminates the structural features of memory fo~ 2025-03-20 NIH-funded~
3 NIH-funded study identifies potential new stroke treat~ 2025-03-17 Preclinica~
4 Surgery in kids with mild sleep-disordered breathing t~ 2025-03-17 NIH-funded~
5 NIH-sponsored trial of Lassa vaccine opens             2025-03-17 Lassa feve~
6 Topical steroid withdrawal diagnostic criteria defined~ 2025-03-14 Criteria m~
```