

Iterating on data with `purrr` and `map`

Data Wrangling: Session 7

Kieran Healy

Statistical Horizons, December 2022

Load the packages, as always

```
library(here)      # manage file paths  
library(socviz)    # data and some useful functions  
library(tidyverse) # your friend and mine
```

Moar Data

More than one data file

Inside the data/ folder of the course packet is a folder named congress/

```
# A little trick from the fs package:  
fs::dir_tree(here("data", "congress"))  
  
## /Users/kjhealy/Documents/courses/data_wrangling/data/congress  
##   ├── 01_79_congress.csv  
##   ├── 02_80_congress.csv  
##   ├── 03_81_congress.csv  
##   ├── 04_82_congress.csv  
##   ├── 05_83_congress.csv  
##   ├── 06_84_congress.csv  
##   ├── 07_85_congress.csv  
##   ├── 08_86_congress.csv  
##   ├── 09_87_congress.csv  
##   ├── 10_88_congress.csv  
##   ├── 11_89_congress.csv  
##   ├── 12_90_congress.csv  
##   ├── 13_91_congress.csv  
##   ├── 14_92_congress.csv  
##   ├── 15_93_congress.csv  
##   ├── 16_94_congress.csv  
##   ├── 17_95_congress.csv  
##   ├── 18_96_congress.csv  
##   ├── 19_97_congress.csv  
##   ├── 20_98_congress.csv  
##   ├── 21_99_congress.csv  
##   ├── 22_100_congress.csv  
##   ├── 23_101_congress.csv  
##   ├── 24_102_congress.csv  
##   ├── 25_103_congress.csv  
##   ├── 26_104_congress.csv  
##   ├── 27_105_congress.csv  
##   ├── 28_106_congress.csv  
##   └── 29_107_congress.csv
```

More than one data file

Let's look at one.

```
read_csv(here("data", "congress", "17_95_congress.csv")) |>  
  janitor::clean_names() |>  
  head()  
  
## # A tibble: 6 × 25  
##   last     first    middle suffix nickn...¹ born   death sex   posit...² party state  
##   <chr>    <chr>    <chr>   <chr>  <chr>  <chr> <chr> <chr> <chr> <chr> <chr>  
## 1 Abdnor   James    <NA>    <NA>   <NA>   02/1... 11/0... M    U.S. R... Repu... SD  
## 2 Abourezk  James    George   <NA>    <NA>   02/2... <NA>   M    U.S. S... Demo... SD  
## 3 Adams    Brockman <NA>    <NA>   Brock   01/1... 09/1... M    U.S. R... Demo... WA  
## 4 Addabbo   Joseph   Patrick  <NA>    <NA>   03/1... 04/1... M    U.S. R... Demo... NY  
## 5 Aiken     George   David    <NA>    <NA>   08/2... 11/1... M    U.S. S... Repu... VT  
## 6 Akaka     Daniel   Kahiki... <NA>    <NA>   09/1... 04/0... M    U.S. R... Demo... HI  
## # ... with 14 more variables: district <chr>, start <chr>, end <chr>,  
## #   religion <chr>, race <chr>, educational_attainment <chr>, job_type1 <chr>,  
## #   job_type2 <chr>, job_type3 <chr>, job_type4 <chr>, job_type5 <lgl>,  
## #   mil1 <chr>, mil2 <chr>, mil3 <chr>, and abbreviated variable names  
## #   ¹nickname, ²position
```

We often find ourselves in this situation. We know each file has the same structure, and we would like to use them all at once.

Loops?

How to read them all in?

One traditional way, which we could do in R, is to write an explicit *loop* that iterated over a vector of filenames, read each file, and then joined the results together in a tall rectangle.

```
# Pseudocode

filenames <- c("01_79_congress.csv", "02_80_congress.csv", "03_81_congress.csv",
             "04_82_congress.csv" [etc etc])

collected_files <- NULL

for(i in 1:length(filenames)) {
  new_file <- read_file(filenames[i])
  collected_files <- append_to(collected_files, new_file)
}
```

Loops?

You may have noticed we have not written any loops, however.

While loops are still lurking there underneath the surface, what we will do instead is to take advantage of the combination of vectors and functions and *map* one to the other in order to generate results.

Speaking loosely, think of `map()` as a way of *iterating* without writing loops. You start with a vector of things. You feed it one thing at a time to some function. The function does whatever it does. You get back output that is the same length as your input, and of a specific type.

Mapping is just a kind of iteration

The `purrr` package provides a big family of mapping functions. One reason there are a lot of them is that `purrr`, like the rest of the tidyverse, is picky about data types.

Mapping is just a kind of iteration

The `purrr` package provides a big family of mapping functions. One reason there are a lot of them is that `purrr`, like the rest of the tidyverse, is picky about data types.

So in addition to the basic `map()`, which always returns a *list*, we also have `map_chr()`, `map_int()`, `map_dbl()`, `map_lgl()` and others. They always return the data type indicated by their suffix, or die trying.

Vectorized arithmetic again

The simplest cases are not that different from the vectorized arithmetic we're already familiar with.

```
a <- c(1:10)  
b <- 1  
  
# You know what R will do here  
a + b  
  
## [1] 2 3 4 5 6 7 8 9 10 11
```

Vectorized arithmetic again

The simplest cases are not that different from the vectorized arithmetic we're already familiar with.

```
a <- c(1:10)  
b <- 1  
  
# You know what R will do here  
a + b  
  
## [1] 2 3 4 5 6 7 8 9 10 11
```

R's vectorized rules add b to every element of a. In a sense, the **+** operation can be thought of as a function that takes each element of a and does something with it. In this case "add b".

Vectorized arithmetic again

We can make this explicit by writing a function:

```
add_b <- function(x) {  
  b <- 1  
  x + b # for any x  
}
```

Now:

```
add_b(x = a)  
## [1] 2 3 4 5 6 7 8 9 10 11
```

Vectorized arithmetic again

Again, R's vectorized approach means it automatically adds b to every element of the x we give it.

```
add_b(x = 10)
## [1] 11

add_b(x = c(1, 99, 1000))
## [1]    2   100 1001
```

Iterating in a pipeline

Some operations can't directly be vectorized in this way, which is why we need to manually iterate, or will want to write loops.

```
library(gapminder)
gapminder |>
  summarize(country_n = n_distinct(country),
            continent_n = n_distinct(continent),
            year_n = n_distinct(year),
            lifeExp_n = n_distinct(lifeExp),
            population_n = n_distinct(population))

## # A tibble: 1 × 5
##   country_n continent_n year_n lifeExp_n population_n
##       <int>        <int>    <int>      <int>        <int>
## 1        142           5     12       1626        4060
```

That's tedious to write! Computers are supposed to allow us to avoid that sort of thing.

Iterating in a pipeline

So how would we iterate this? What we want is to apply the `n_distinct()` function to each column of `gapminder`, but in a way that still allows us to use pipelines and so on.

```
library(gapminder)
gapminder |>
  summarize(n_distinct(country),
            n_distinct(continent),
            n_distinct(year),
            n_distinct(lifeExp),
            n_distinct(population))

## # A tibble: 1 × 5
##   `n_distinct(country)` `n_distinct(continent)` `n_distinct(yea...` `n_dis...` `n_dis...
##             <int>                  <int>                 <int>     <int>     <int>
## 1                142                      5                   12      1626      4060
## # ... with abbreviated variable names ¹`n_distinct(year)` ,
## #   ²`n_distinct(lifeExp)` , ³`n_distinct(population)`
```

Using `n_distinct()` in this context is an idea I got from Rebecca Barter's discussion of `purrr`.

Iterating in a pipeline

You'd use **across()**, like this:

```
gapminder |>  
  summarize(across(everything(), n_distinct))  
  
## # A tibble: 1 × 6  
##   country continent year lifeExp   pop gdpPercap  
##   <int>     <int> <int>   <int> <int>    <int>  
## 1     142       5    12    1626  1704      1704
```

Iterating in a pipeline

But you could also do this ...

```
map(gapminder, n_distinct)

## $country
## [1] 142
##
## $continent
## [1] 5
##
## $year
## [1] 12
##
## $lifeExp
## [1] 1626
##
## $pop
## [1] 1704
##
## $gdpPercap
## [1] 1704
```

Read it as "Feed each column of gapminder to the **n_distinct()** function.

(This is pretty much what **across()** is doing more nicely.)

Iterating in a pipeline

Or, in pipeline form:

```
gapminder |>  
  map(n_distinct)
```

```
## $country  
## [1] 142  
##  
## $continent  
## [1] 5  
##  
## $year  
## [1] 12  
##  
## $lifeExp  
## [1] 1626  
##  
## $pop  
## [1] 1704  
##  
## $gdpPercap  
## [1] 1704
```

You can see we are getting a *list* back.

Iterating in a pipeline

Or, in pipeline form:

```
result <- gapminder |>  
  map(n_distinct)  
  
class(result)  
## [1] "list"  
  
result$continent  
## [1] 5  
  
result[[2]]  
## [1] 5
```

Iterating in a pipeline

But we know `n_distinct()` should always return an integer. So we use `map_int()` instead of the generic `map()`.

```
gapminder |>  
  map_int(n_distinct)  
  
## # A tibble: 142 x 5  
##   country continent year lifeExp pop gdpPercap  
##   <dbl>     <dbl> <dbl>    <dbl> <dbl>      <dbl>  
## 1       1        5     12     1626    1704      1704
```

The thing about the `map()` family is that they can deal with all kinds of input types and output types.

Get a vector of filenames

```
filenames <- dir(path = here("data", "congress"),
                  pattern = "*.csv",
                  full.names = TRUE)

filenames[1:15] # Just displaying the first 15, to save slide space

## [1] "/Users/kjhealy/Documents/courses/data_wrangling/data/congress/01_79_congress.csv"
## [2] "/Users/kjhealy/Documents/courses/data_wrangling/data/congress/02_80_congress.csv"
## [3] "/Users/kjhealy/Documents/courses/data_wrangling/data/congress/03_81_congress.csv"
## [4] "/Users/kjhealy/Documents/courses/data_wrangling/data/congress/04_82_congress.csv"
## [5] "/Users/kjhealy/Documents/courses/data_wrangling/data/congress/05_83_congress.csv"
## [6] "/Users/kjhealy/Documents/courses/data_wrangling/data/congress/06_84_congress.csv"
## [7] "/Users/kjhealy/Documents/courses/data_wrangling/data/congress/07_85_congress.csv"
## [8] "/Users/kjhealy/Documents/courses/data_wrangling/data/congress/08_86_congress.csv"
## [9] "/Users/kjhealy/Documents/courses/data_wrangling/data/congress/09_87_congress.csv"
## [10] "/Users/kjhealy/Documents/courses/data_wrangling/data/congress/10_88_congress.csv"
## [11] "/Users/kjhealy/Documents/courses/data_wrangling/data/congress/11_89_congress.csv"
## [12] "/Users/kjhealy/Documents/courses/data_wrangling/data/congress/12_90_congress.csv"
## [13] "/Users/kjhealy/Documents/courses/data_wrangling/data/congress/13_91_congress.csv"
## [14] "/Users/kjhealy/Documents/courses/data_wrangling/data/congress/14_92_congress.csv"
## [15] "/Users/kjhealy/Documents/courses/data_wrangling/data/congress/15_93_congress.csv"
```

And feed it to `read_csv()`

... using the variant of `map()` that returns data frames and tibbles.

```
df <- filenames |>
  map_dfr(read_csv, .id = "congress") |>
  janitor::clean_names()

df

## # A tibble: 20,580 × 26
##   congress last    first middle suffix nickname¹ born   death sex   posit...² party
##   <chr>     <chr>   <chr> <chr>  <chr>   <chr> <chr> <chr> <chr> <chr> <chr>
## 1 1        Abernet... Thom... Gerst... <NA>   <NA>   05/1... 01/2... M    U.S. R... Demo...
## 2 1        Adams     Sher... <NA>   <NA>   <NA>   01/0... 10/2... M    U.S. R... Repu...
## 3 1        Aiken     Geor... David   <NA>   <NA>   08/2... 11/1... M    U.S. S... Repu...
## 4 1        Allen     Asa    Leona... <NA>   <NA>   01/0... 01/0... M    U.S. R... Demo...
## 5 1        Allen     Leo    Elwood  <NA>   <NA>   10/0... 01/1... M    U.S. R... Repu...
## 6 1        Almond    J.    Linds... Jr.   <NA>   06/1... 04/1... M    U.S. R... Demo...
## 7 1        Andersen Herm... Carl   <NA>   <NA>   01/2... 07/2... M    U.S. R... Repu...
## 8 1        Anderson Clin... Presba <NA>   <NA>   10/2... 11/1... M    U.S. R... Demo...
## 9 1        Anderson John  Zuing... <NA>   <NA>   03/2... 02/0... M    U.S. R... Repu...
## 10 1       Andresen Augu... Herman <NA>   <NA>   10/1... 01/1... M    U.S. R... Repu...
## # ... with 20,570 more rows, 15 more variables: state <chr>, district <chr>,
## #   start <chr>, end <chr>, religion <chr>, race <chr>,
## #   educational_attainment <chr>, job_type1 <chr>, job_type2 <chr>,
## #   job_type3 <chr>, job_type4 <chr>, job_type5 <chr>, mil1 <chr>, mil2 <chr>,
## #   mil3 <chr>, and abbreviated variable names ¹nickname, ²position
```

Now witness the firepower of this fully armed and operational



method of type-safe functional iteration

read_csv() can do this directly now!

No map_df() required

```
tmp <- read_csv(filenames, id = "path",
                 name_repair = janitor::make_clean_names)

tmp |>
  mutate(congress = stringr::str_extract(path, "_\\d{2,3}_congress"),
         congress = stringr::str_extract(congress, "\\d{2,3}")) |>
  relocate(congress)

## # A tibble: 20,580 × 27
##   congress path      last    first   middle suffix nickname¹ born   death sex   posit...²
##   <chr>     <chr>    <chr>   <chr>   <chr>   <chr>   <chr>   <chr>   <chr>   <chr>
## 1 79       /Users/... Aber... Thom... Gerst... <NA>    <NA>    05/1... 01/2... M    U.S. R...
## 2 79       /Users/... Adams Sher... <NA>    <NA>    <NA>    01/0... 10/2... M    U.S. R...
## 3 79       /Users/... Aiken Geor... David   <NA>    <NA>    08/2... 11/1... M    U.S. S...
## 4 79       /Users/... Allen Asa   Leona... <NA>    <NA>    01/0... 01/0... M    U.S. R...
## 5 79       /Users/... Allen Leo   Elwood  <NA>    <NA>    10/0... 01/1... M    U.S. R...
## 6 79       /Users/... Almo... J.   Linds... Jr.   <NA>    <NA>    06/1... 04/1... M    U.S. R...
## 7 79       /Users/... Ande... Herm... Carl   <NA>    <NA>    01/2... 07/2... M    U.S. R...
## 8 79       /Users/... Ande... Clin... Presba <NA>    <NA>    10/2... 11/1... M    U.S. R...
## 9 79       /Users/... Ande... John   Zuing... <NA>    <NA>    03/2... 02/0... M    U.S. R...
## 10 79      /Users/... Andr... Augu... Herman <NA>    <NA>    10/1... 01/1... M    U.S. R...
## # ... with 20,570 more rows, 16 more variables: party <chr>, state <chr>,
## #   district <chr>, start <chr>, end <chr>, religion <chr>, race <chr>,
## #   educational_attainment <chr>, job_type1 <chr>, job_type2 <chr>,
## #   job_type3 <chr>, job_type4 <chr>, job_type5 <chr>, mil1 <chr>, mil2 <chr>,
## #   mil3 <chr>, and abbreviated variable names ¹nickname, ²position
```

Example: Iterating on the US Census

But mapped iteration is not just for local files!

```
## Register for a free Census API key
library(tidycensus)

out <- get_acs(geography = "county",
                variables = "B19013_001",
                state = "NY",
                county = "New York",
                survey = "acs1",
                year = 2005)

out

## # A tibble: 1 × 5
##   GEOID NAME           variable estimate    moe
##   <chr> <chr>          <chr>     <dbl> <dbl>
## 1 36061 New York County, New York B19013_001  55973  1462
```

Example: Iterating on the US Census

All counties in New York State for a specific year

```
out <- get_acs(geography = "county",
                variables = "B19013_001",
                state = "NY",
                survey = "acs1",
                year = 2005)

out

## # A tibble: 38 × 5
##   GEOID NAME           variable   estimate    moe
##   <chr> <chr>          <chr>      <dbl> <dbl>
## 1 36001 Albany County, New York B19013_001  50054  2030
## 2 36005 Bronx County, New York B19013_001  29228   853
## 3 36007 Broome County, New York B19013_001  36394  2340
## 4 36009 Cattaraugus County, New York B19013_001  37580  2282
## 5 36011 Cayuga County, New York  B19013_001  42057  2406
## 6 36013 Chautauqua County, New York B19013_001  35495  2077
## 7 36015 Chemung County, New York B19013_001  37418  3143
## 8 36019 Clinton County, New York B19013_001  44757  3500
## 9 36027 Dutchess County, New York B19013_001  61889  2431
## 10 36029 Erie County, New York   B19013_001  41967  1231
## # ... with 28 more rows
```

Example: Iterating on the US Census

What if we want the results for *every* available year?

First, a handy function: `set_names()`

```
x <- c(1:10)  
x  
## [1] 1 2 3 4 5 6 7 8 9 10  
  
x <- set_names(x, nm = letters[1:10])  
x  
## a b c d e f g h i j  
## 1 2 3 4 5 6 7 8 9 10
```

Example: Iterating on the US Census

By default, `set_names()` will label a vector with that vector's values:

```
c(1:10) |>  
  set_names()  
  
##  1  2  3  4  5  6  7  8  9 10  
## 1  2  3  4  5  6  7  8  9 10
```

Example: Iterating on the US Census

Meanwhile, `map_dfr()` has an `.id` argument that lets you add a row-identifier to whatever you are binding. Like this:

```
df <- 2005:2019 |>
  map_dfr(~ get_acs(geography = "county",
                     variables = "B19013_001",
                     state = "NY",
                     county = "New York",
                     survey = "acs1",
                     year = .x),
         .id = "id")

df

## # A tibble: 15 × 6
##   id    GEOID NAME          variable  estimate    moe
##   <chr> <chr> <chr>        <chr>      <dbl> <dbl>
## 1 1     36061 New York County, New York B19013_001  55973 1462
## 2 2     36061 New York County, New York B19013_001  60017 1603
## 3 3     36061 New York County, New York B19013_001  64217 2002
## 4 4     36061 New York County, New York B19013_001  69017 1943
## 5 5     36061 New York County, New York B19013_001  68706 1471
## 6 6     36061 New York County, New York B19013_001  63832 2125
## 7 7     36061 New York County, New York B19013_001  66299 1783
## 8 8     36061 New York County, New York B19013_001  67099 1640
## 9 9     36061 New York County, New York B19013_001  72190 2200
## 10 10   36061 New York County, New York B19013_001  76089 2016
## # ... with 5 more rows, and 1 more variable: state <chr>
```

Example: Iterating on the US Census

Our `id` column tracks the year. But we'd like it to *be* the year. So, we use `set_names()`, and also name it `year` when we create it:

```
df <- 2005:2019 |>
  set_names() |>
  map_dfr(~ get_acs(geography = "county",
                     variables = "B19013_001",
                     state = "NY",
                     county = "New York",
                     survey = "acs1",
                     year = .x),
         .id = "year") |>
  mutate(year = as.integer(year))
```

Example: Iterating on the US Census

df

```
## # A tibble: 15 × 6
##   year GEOID NAME      variable estimate    moe
##   <int> <chr> <chr>     <chr>      <dbl> <dbl>
## 1 2005 36061 New York County, New York B19013_001  55973 1462
## 2 2006 36061 New York County, New York B19013_001  60017 1603
## 3 2007 36061 New York County, New York B19013_001  64217 2002
## 4 2008 36061 New York County, New York B19013_001  69017 1943
## 5 2009 36061 New York County, New York B19013_001  68706 1471
## 6 2010 36061 New York County, New York B19013_001  63832 2125
## 7 2011 36061 New York County, New York B19013_001  66299 1783
## 8 2012 36061 New York County, New York B19013_001  67099 1640
## 9 2013 36061 New York County, New York B19013_001  72190 2200
## 10 2014 36061 New York County, New York B19013_001  76089 2016
## 11 2015 36061 New York County, New York B19013_001  75575 2566
## 12 2016 36061 New York County, New York B19013_001  77559 2469
## 13 2017 36061 New York County, New York B19013_001  85071 3520
## 14 2018 36061 New York County, New York B19013_001  85066 3480
## 15 2019 36061 New York County, New York B19013_001  93651 3322
```

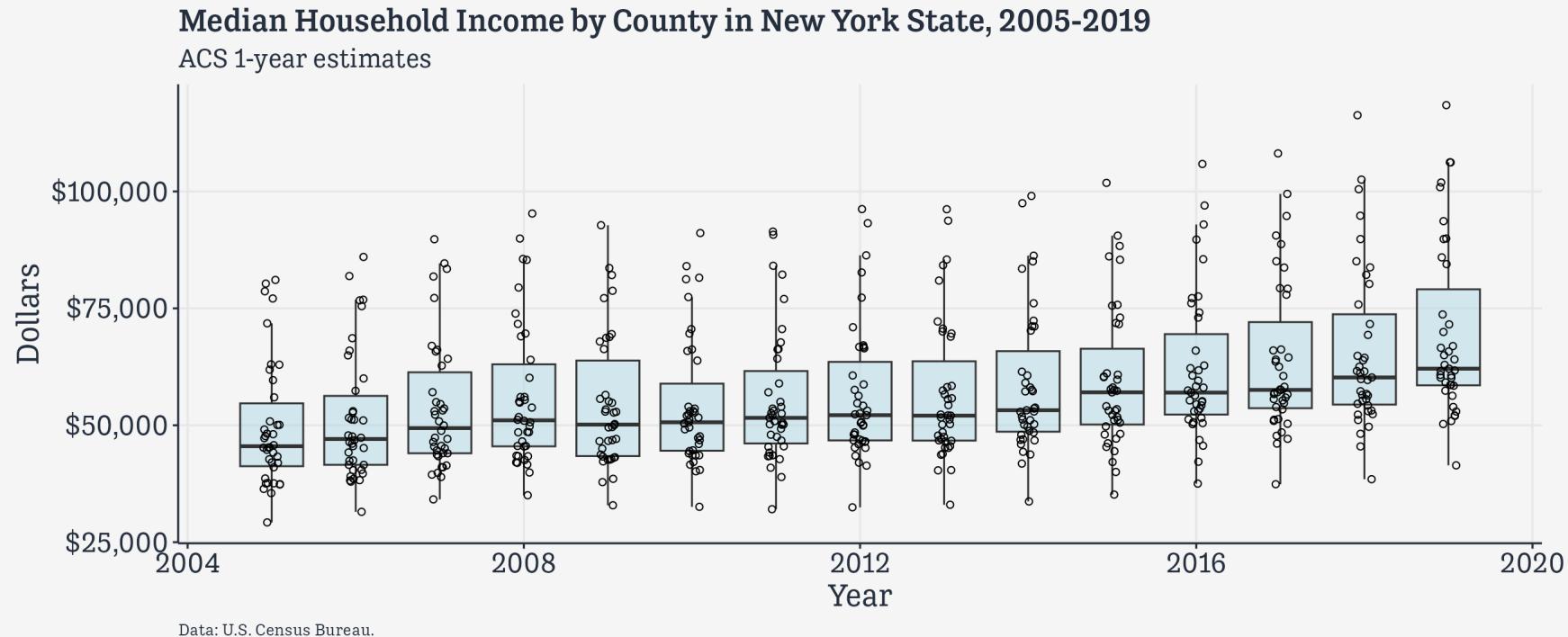
Now `year` is just the year. The `year` column will be created as a character vector, so we converted it back to an integer again at the end.

Example: Iterating on the US Census

```
p_out <- 2005:2019 |>
  set_names() |>
  map_dfr(~ get_acs(geography = "county", variables = "B19013_001",
                     state = "NY", survey = "acs1", year = .x), .id = "year") |>
  mutate(year = as.integer(year)) |>
  ggplot(mapping = aes(x = year, y = estimate, group = year)) +
  geom_boxplot(fill = "lightblue", alpha = 0.5, outlier.alpha = 0) +
  geom_jitter(position = position_jitter(width = 0.1), shape = 1) +
  scale_y_continuous(labels = scales::label_dollar()) + labs(x = "Year", y = "Dollars",
  title = "Median Household Income by County in New York State, 2005-2019",
  subtitle = "ACS 1-year estimates", caption = "Data: U.S. Census Bureau.")
```

Example: Iterating on the US Census

```
print(p_out)
```



Cleaning up congress

```
df <- filenames |>
  map_dfr(read_csv, .id = "congress") |>
  janitor::clean_names()

df |>
  select(born, death, start, end)

## # A tibble: 20,580 × 4
##   born      death     start     end
##   <chr>     <chr>     <chr>     <chr>
## 1 05/16/1903 01/23/1953 01/03/1945 01/03/1953
## 2 01/08/1899 10/27/1986 01/03/1945 01/03/1947
## 3 08/20/1892 11/19/1984 01/03/1945 01/03/1979
## 4 01/05/1891 01/05/1969 01/03/1945 01/03/1953
## 5 10/05/1898 01/19/1973 01/03/1945 01/02/1949
## 6 06/15/1898 04/14/1986 02/04/1946 04/17/1948
## 7 01/27/1897 07/26/1978 01/03/1945 01/03/1963
## 8 10/23/1895 11/11/1975 01/03/1941 06/30/1945
## 9 03/22/1904 02/09/1981 01/03/1945 01/03/1953
## 10 10/11/1890 01/14/1958 01/03/1945 01/14/1958
## # ... with 20,570 more rows
```

We'll use the **lubridate** package to sort these out.

Lubridate has a wide range of functions to handle dates, times, and durations.

Cleaning up congress

```
library(lubridate)

date_recodes <- c("born", "death", "start", "end")
df <- df |>
  mutate(across(any_of(date_recodes), mdy),
        congress = as.integer(congress) + 78)

df

## # A tibble: 20,580 × 26
##   congr...¹ last   first  middle suffix nickn...² born       death      sex posit...³
##   <dbl> <chr> <chr> <chr> <chr> <chr> <date>    <date>    <chr> <chr>
## 1     79 Aber... Thom... Gerst... <NA>   <NA> 1903-05-16 1953-01-23 M U.S. R...
## 2     79 Adams Sher... <NA>   <NA>   <NA> 1899-01-08 1986-10-27 M U.S. R...
## 3     79 Aiken Geor... David <NA>   <NA> 1892-08-20 1984-11-19 M U.S. S...
## 4     79 Allen Asa   Leona... <NA>   <NA> 1891-01-05 1969-01-05 M U.S. R...
## 5     79 Allen Leo   Elwood <NA>   <NA> 1898-10-05 1973-01-19 M U.S. R...
## 6     79 Almo... J. Linds... Jr. <NA>   <NA> 1898-06-15 1986-04-14 M U.S. R...
## 7     79 Ande... Herm... Carl <NA>   <NA> 1897-01-27 1978-07-26 M U.S. R...
## 8     79 Ande... Clin... Presba <NA>   <NA> 1895-10-23 1975-11-11 M U.S. R...
## 9     79 Ande... John Zuing... <NA>   <NA> 1904-03-22 1981-02-09 M U.S. R...
## 10    79 Andr... Augu... Herman <NA>   <NA> 1890-10-11 1958-01-14 M U.S. R...
## # ... with 20,570 more rows, 16 more variables: party <chr>, state <chr>,
## #   district <chr>, start <date>, end <date>, religion <chr>, race <chr>,
## #   educational_attainment <chr>, job_type1 <chr>, job_type2 <chr>,
## #   job_type3 <chr>, job_type4 <chr>, job_type5 <chr>, mil1 <chr>, mil2 <chr>,
## #   mil3 <chr>, and abbreviated variable names `¹congress`, `²nickname`, `³position`
```

Cleaning up congress

```
sessions <- tibble(congress = 79:116,
                    start_year = seq(1945, 2019, by = 2),
                    end_year = seq(1947, 2021, by = 2)) |>
  mutate(start_year = ymd(paste(start_year, "01", "03", sep = "-")),
        end_year = ymd(paste(end_year, "01", "03", sep = "-")))

sessions

## # A tibble: 38 × 3
##   congress start_year end_year
##       <int>     <date>    <date>
## 1       79 1945-01-03 1947-01-03
## 2       80 1947-01-03 1949-01-03
## 3       81 1949-01-03 1951-01-03
## 4       82 1951-01-03 1953-01-03
## 5       83 1953-01-03 1955-01-03
## 6       84 1955-01-03 1957-01-03
## 7       85 1957-01-03 1959-01-03
## 8       86 1959-01-03 1961-01-03
## 9       87 1961-01-03 1963-01-03
## 10      88 1963-01-03 1965-01-03
## # ... with 28 more rows
```

We're going to join these tables

The big table

```
df |>  
  select(congress, last, born)  
  
## # A tibble: 20,580 × 3  
##   congress last    born  
##   <dbl> <chr>   <date>  
## 1      79 Abernethy 1903-05-16  
## 2      79 Adams    1899-01-08  
## 3      79 Aiken    1892-08-20  
## 4      79 Allen    1891-01-05  
## 5      79 Allen    1898-10-05  
## 6      79 Almond   1898-06-15  
## 7      79 Andersen 1897-01-27  
## 8      79 Anderson 1895-10-23  
## 9      79 Anderson 1904-03-22  
## 10     79 Andresen 1890-10-11  
## # ... with 20,570 more rows
```

The smaller table

```
sessions  
## # A tibble: 38 × 3  
##   congress start_year end_year  
##   <int> <date>    <date>  
## 1      79 1945-01-03 1947-01-03  
## 2      80 1947-01-03 1949-01-03  
## 3      81 1949-01-03 1951-01-03  
## 4      82 1951-01-03 1953-01-03  
## 5      83 1953-01-03 1955-01-03  
## 6      84 1955-01-03 1957-01-03  
## 7      85 1957-01-03 1959-01-03  
## 8      86 1959-01-03 1961-01-03  
## 9      87 1961-01-03 1963-01-03  
## 10     88 1963-01-03 1965-01-03  
## # ... with 28 more rows
```

We're going to **join** these tables

We will use **left_join()** which is what you want most of the time when you are looking to merge a smaller table with additional information into a larger main one.

```
df <- left_join(df, sessions) |>
  relocate(start_year:end_year, .after = congress)

## Joining, by = "congress"

df

## # A tibble: 20,580 × 28
##   congress start_year end_year   last    first middle suffix nickn...¹ born
##       <dbl>     <date>    <date>   <chr>   <chr>   <chr>   <chr>   <date>
## 1       79 1945-01-03 1947-01-03 Aberne... Thom... Gerst... <NA>   <NA>   1903-05-16
## 2       79 1945-01-03 1947-01-03 Adams   Sher... <NA>   <NA>   <NA>   1899-01-08
## 3       79 1945-01-03 1947-01-03 Aiken   Geor... David   <NA>   <NA>   1892-08-20
## 4       79 1945-01-03 1947-01-03 Allen   Asa    Leona... <NA>   <NA>   1891-01-05
## 5       79 1945-01-03 1947-01-03 Allen   Leo    Elwood  <NA>   <NA>   1898-10-05
## 6       79 1945-01-03 1947-01-03 Almond  J.    Linds... Jr.   <NA>   1898-06-15
## 7       79 1945-01-03 1947-01-03 Anders... Herm... Carl   <NA>   <NA>   1897-01-27
## 8       79 1945-01-03 1947-01-03 Anders... Clin... Presba <NA>   <NA>   1895-10-23
## 9       79 1945-01-03 1947-01-03 Anders... John   Zuing... <NA>   <NA>   1904-03-22
## 10      79 1945-01-03 1947-01-03 Andres... Augu... Herman <NA>   <NA>   1890-10-11
## ... with 20,570 more rows, 19 more variables: death <date>, sex <chr>,
##   position <chr>, party <chr>, state <chr>, district <chr>, start <date>,
##   end <date>, religion <chr>, race <chr>, educational_attainment <chr>,
##   job_type1 <chr>, job_type2 <chr>, job_type3 <chr>, job_type4 <chr>,
##   job_type5 <chr>, mil1 <chr>, mil2 <chr>, mil3 <chr>, and abbreviated
##   variable name `nickname`
```

Table joins

X	y
1	x1
2	x2
3	x3

1	y1
2	y2
4	y4

*Spiffy Join Animations courtesy [Garrick Aden-Buie](#)

Left join, `left_join()`

`left_join(x, y)`

1	x1	1	y1
2	x2	2	y2
3	x3	4	y4

All rows from x, and all columns from x and y. Rows in x with no match in y will have NA values in the new columns.

Left join (contd), `left_join()`

`left_join(x, y)`

1	x1	1	y1
2	x2	2	y2
3	x3	4	y4
		2	y5

If there are multiple matches between x and y, all combinations of the matches are returned.

Inner join, `inner_join()`

`inner_join(x, y)`

1	x1	1	y1
2	x2	2	y2
3	x3	4	y4

All rows from x where there are matching values in y, and all columns from x and y.

Full join, **full_join()**

`full_join(x, y)`

1	x1	1	y1
2	x2	2	y2
3	x3	4	y4

All rows and all columns from both x and y. Where there are not matching values, returns NA for the one missing.

Semi join, **semi_join()**

`semi_join(x, y)`

1	x1	1	y1
2	x2	2	y2
3	x3	4	y4

All rows from x where there are matching values in y, keeping just columns from x.

Anti join, **anti_join()**

anti_join(x, y)	
1	x1
2	x2
3	x3
1	y1
2	y2
4	y4

All rows from x where there are not matching values in y, keeping just columns from x.

Left join, `left_join()`

Most of the time you will be looking to make a `left_join()`

Missing Data

Never test for missingness with ==

The result of almost any operation involving a missing/unknown value will be missing/unknown.

```
df <- tribble(  
  ~subject, ~age,  
  "A", 20,  
  "B", 25,  
  "C", NA,  
  "D", 34  
)  
  
df  
  
## # A tibble: 4 × 2  
##   subject    age  
##   <chr>     <dbl>  
## 1 A          20  
## 2 B          25  
## 3 C          NA  
## 4 D          34
```

Never test for missingness with ==

The result of almost any operation involving a missing/unknown value will be missing/unknown.

```
# OK
df |>
  filter(age == 25)

## # A tibble: 1 × 2
##   subject    age
##   <chr>     <dbl>
## 1 B           25
```

Never test for missingness with ==

The result of almost any operation involving a missing/unknown value will be missing/unknown.

```
# OK
df |>
  filter(age == 25)

## # A tibble: 1 × 2
##   subject    age
##   <chr>     <dbl>
## 1 B          25

# Nope
df |>
  filter(age == NA)

## # A tibble: 0 × 2
## # ... with 2 variables: subject <chr>, age <dbl>

# E.g.
23 == NA

## [1] NA
```

Never test for missingness with ==

Always use **is.na()** instead

```
# Yes
df |>
  filter(is.na(age))

## # A tibble: 1 × 2
##   subject    age
##   <chr>     <dbl>
## 1 C          NA
```

A quick plug for **naniar** and **visdat**

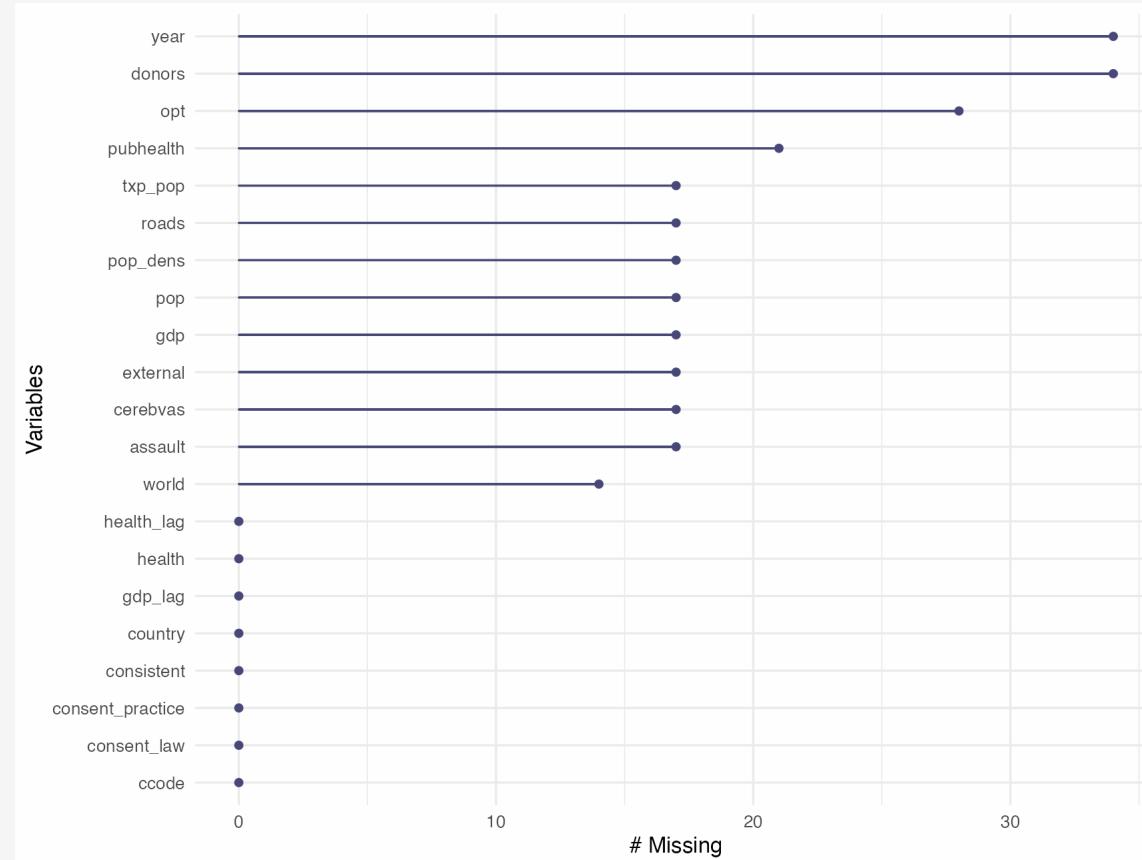
```
library(naniar)
library(visdat)

organdata

## # A tibble: 238 × 21
##   country    year    donors    pop  pop_d...¹    gdp  gdp_lag  health healt...²  pubhe...³
##   <chr>      <date>   <dbl>   <int>  <dbl>   <int>   <dbl>   <dbl>   <dbl>
## 1 Austral... NA        NA     17065  0.220  16774  16591  1300    1224    4.8
## 2 Austral... 1991-01-01 12.1    17284  0.223  17171  16774  1379    1300    5.4
## 3 Austral... 1992-01-01 12.4    17495  0.226  17914  17171  1455    1379    5.4
## 4 Austral... 1993-01-01 12.5    17667  0.228  18883  17914  1540    1455    5.4
## 5 Austral... 1994-01-01 10.2    17855  0.231  19849  18883  1626    1540    5.4
## 6 Austral... 1995-01-01 10.2    18072  0.233  21079  19849  1737    1626    5.5
## 7 Austral... 1996-01-01 10.6    18311  0.237  21923  21079  1846    1737    5.6
## 8 Austral... 1997-01-01 10.3    18518  0.239  22961  21923  1948    1846    5.7
## 9 Austral... 1998-01-01 10.5    18711  0.242  24148  22961  2077    1948    5.9
## 10 Austral... 1999-01-01 8.67   18926  0.244  25445  24148  2231    2077   6.1
## # ... with 228 more rows, 11 more variables: roads <dbl>, cerebvas <int>,
## # assault <int>, external <int>, txp_pop <dbl>, world <chr>, opt <chr>,
## # consent_law <chr>, consent_practice <chr>, consistent <chr>, ccode <chr>,
## # and abbreviated variable names `¹pop_dens`, `²health_lag`, `³pubhealth
```

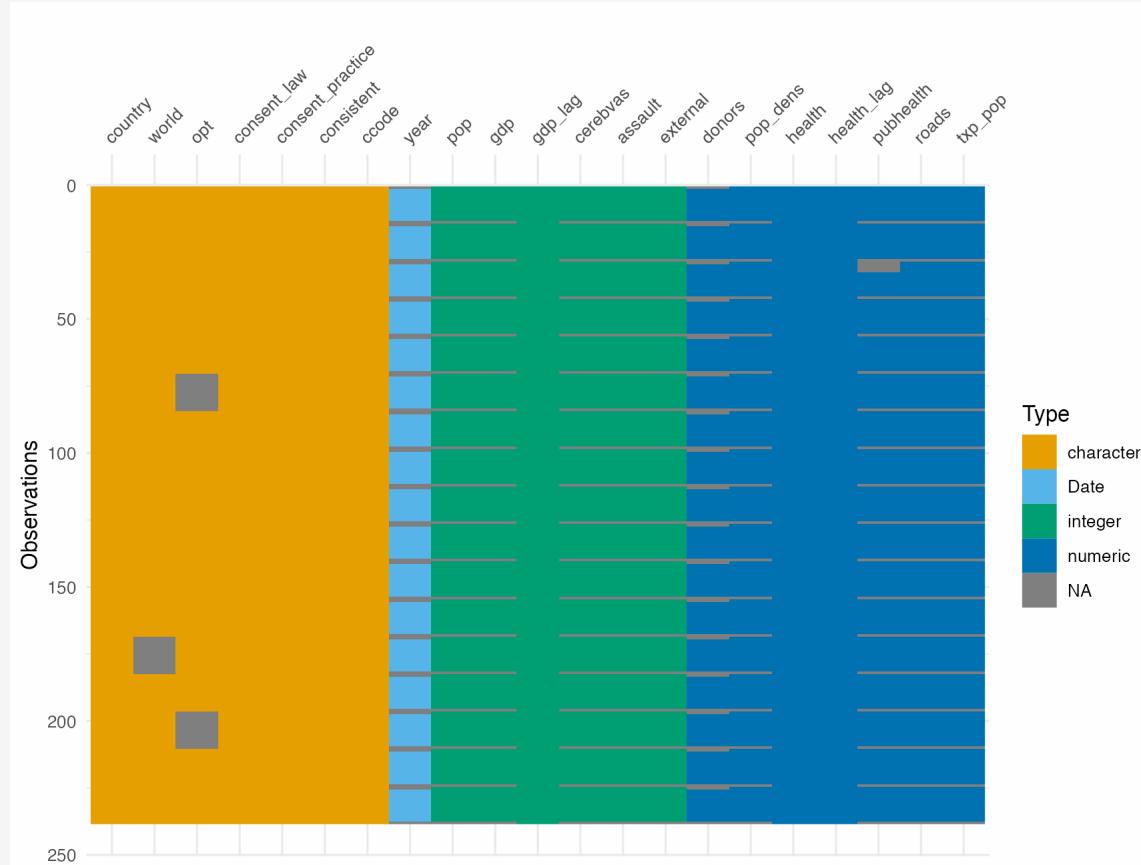
A quick plug for **naniar** and **visdat**

```
gg_miss_var(organdata)
```



A quick plug for **naniar** and **visdat**

```
vis_dat(organdata)
```



A quick plug for **naniar** and **visdat**

```
miss_var_summary(organdata)

## # A tibble: 21 × 3
##   variable n_miss pct_miss
##   <chr>     <int>    <dbl>
## 1 year       34     14.3
## 2 donors     34     14.3
## 3 opt        28     11.8
## 4 pubhealth   21     8.82
## 5 pop         17     7.14
## 6 pop_dens    17     7.14
## 7 gdp         17     7.14
## 8 roads        17     7.14
## 9 cerebvas    17     7.14
## 10 assault     17     7.14
## # ... with 11 more rows
```

A quick plug for **naniar** and **visdat**

```
miss_case_summary(organdata)

## # A tibble: 238 × 3
##   case n_miss pct_miss
##   <int>    <int>     <dbl>
## 1     84      12     57.1
## 2    182      12     57.1
## 3    210      12     57.1
## 4     14      11     52.4
## 5     28      11     52.4
## 6     42      11     52.4
## 7     56      11     52.4
## 8     70      11     52.4
## 9     98      11     52.4
## 10   112      11     52.4
## # ... with 228 more rows
```

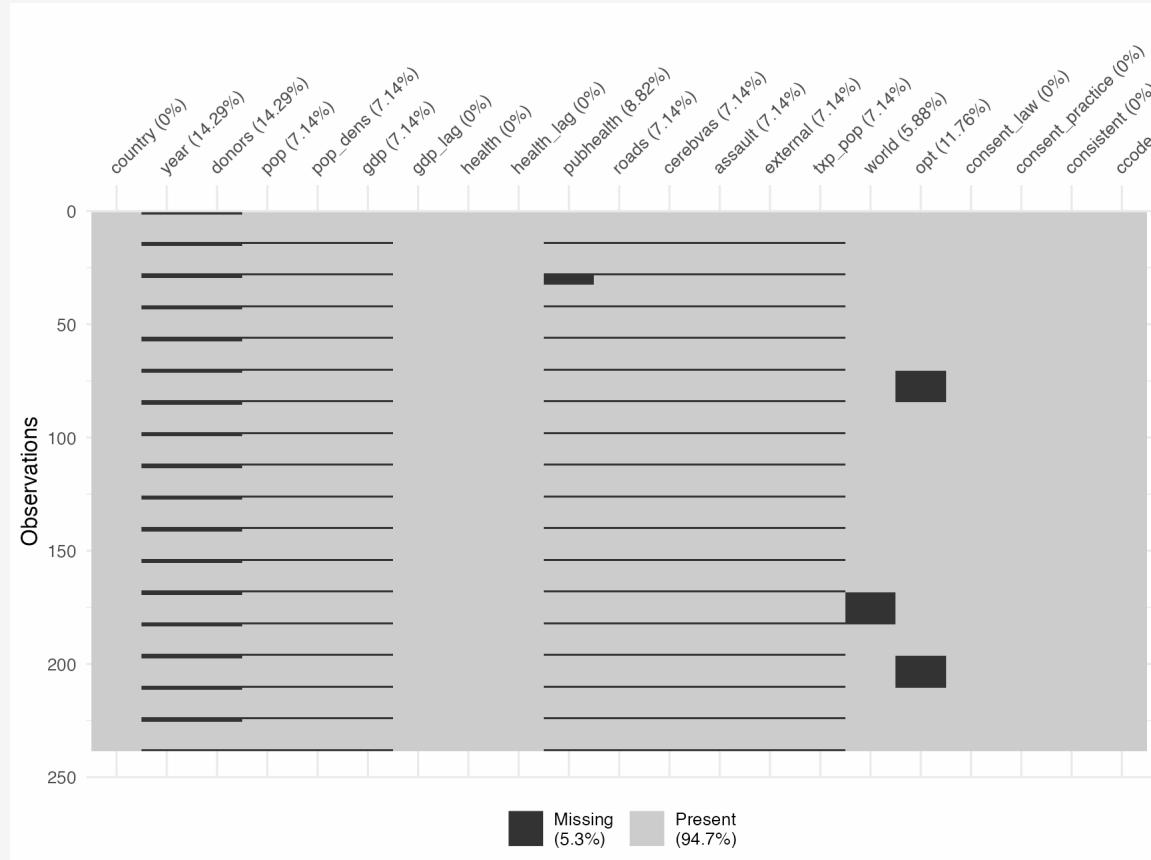
A quick plug for **naniar** and **visdat**

```
organdata |>
  select(consent_law, year, pubhealth, roads) |>
  group_by(consent_law) |>
  miss_var_summary()

## # A tibble: 6 × 4
## # Groups: consent_law [2]
##   consent_law variable n_miss pct_miss
##   <chr>       <chr>     <int>    <dbl>
## 1 Informed    year        16     14.3
## 2 Informed    pubhealth    8      7.14
## 3 Informed    roads        8      7.14
## 4 Presumed    year        18     14.3
## 5 Presumed    pubhealth   13     10.3
## 6 Presumed    roads        9      7.14
```

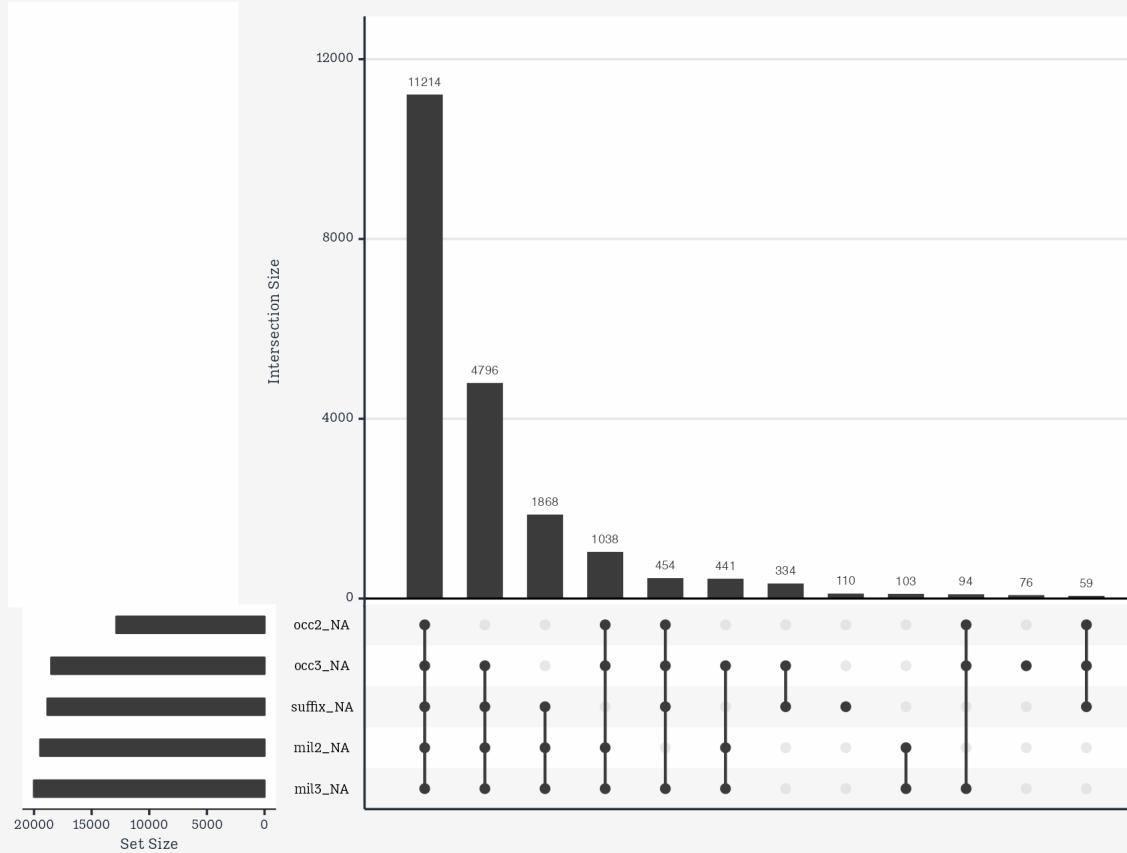
A quick plug for **naniar** and **visdat**

```
vis_miss(organdata)
```



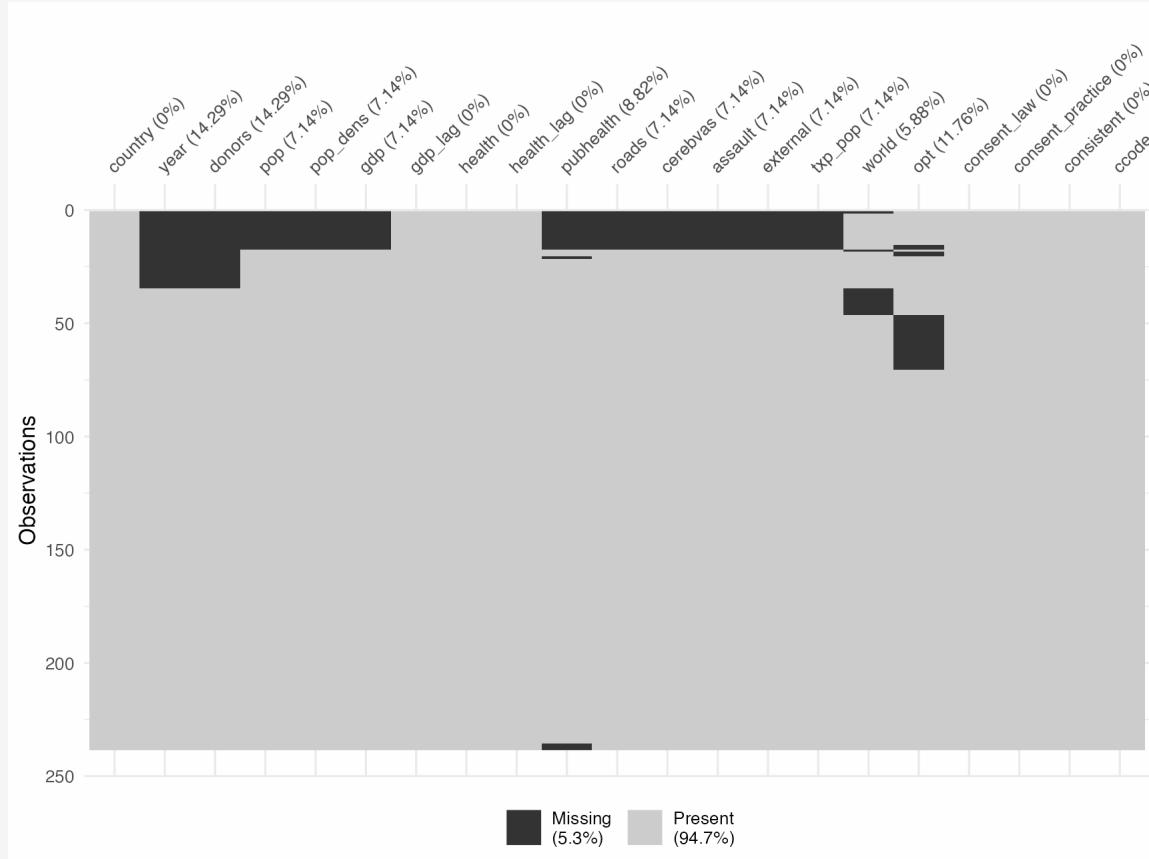
A quick plug for **naniar** and **visdat**

```
library(congress)
gg_miss_upset(congress)
```



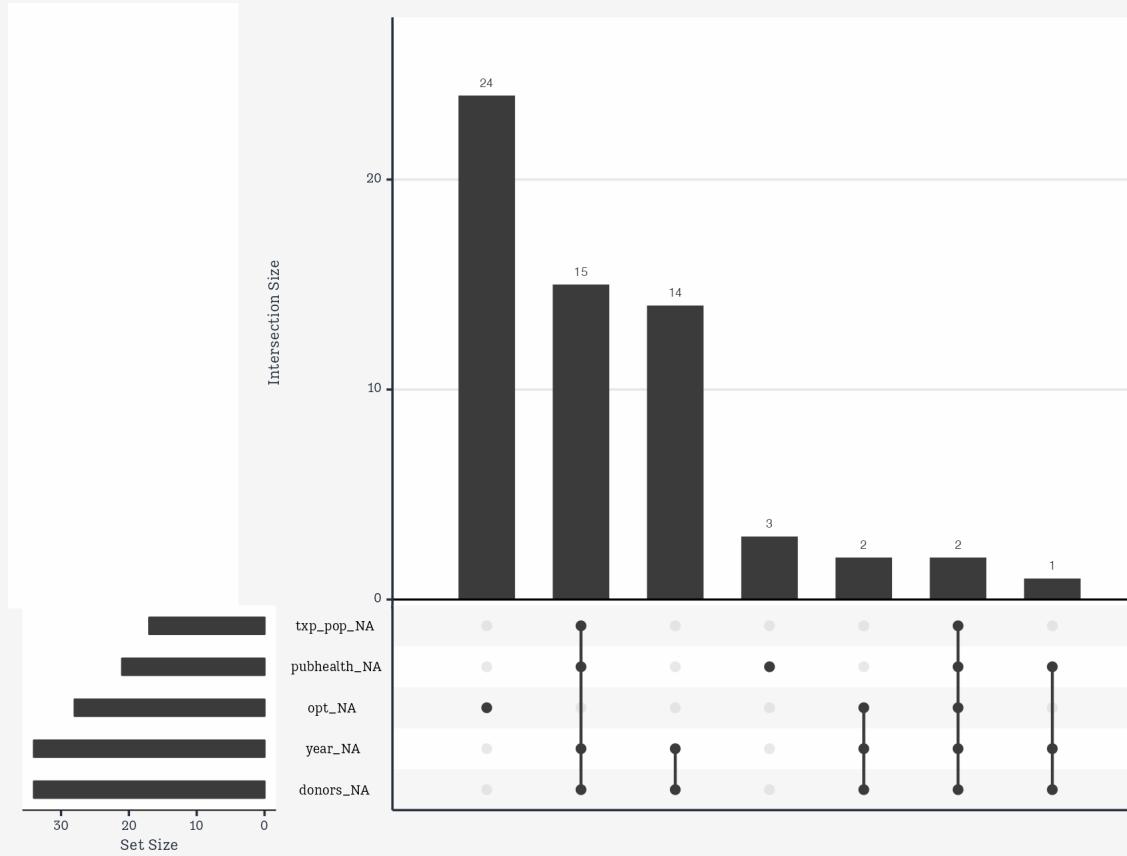
A quick plug for **naniar** and **visdat**

```
vis_miss(organdata, cluster = TRUE)
```



A quick plug for **naniar** and **visdat**

```
gg_miss_upset(organdata)
```

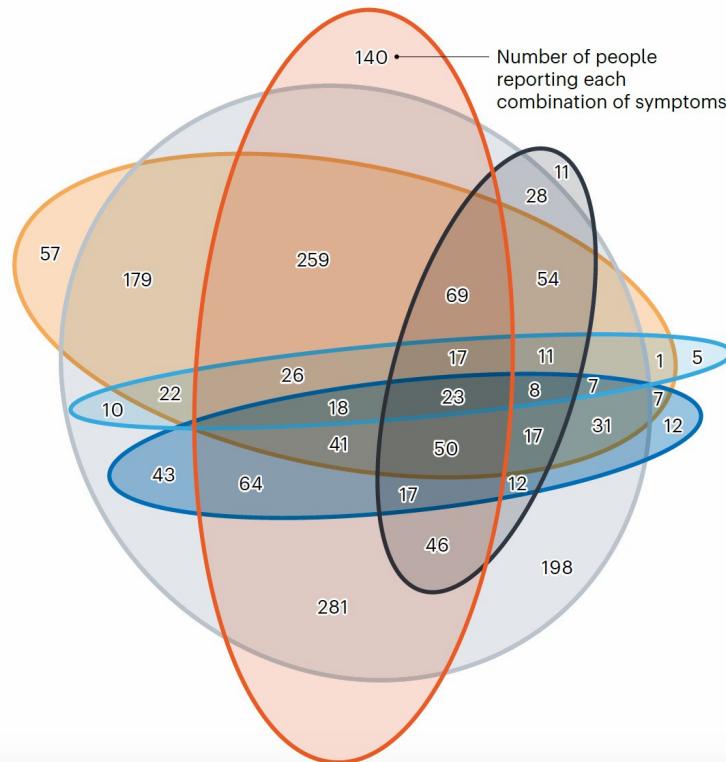


Upset plots and a bit of wrangling

TRACKING SYMPTOMS

On 7 April, around 60% of app users who tested positive for COVID-19 and reported symptoms had lost their sense of smell.

Anosmia (loss of smell) Cough Fatigue
Diarrhoea Shortness of breath Fever



PHOTOGRAPH: PIER MARCO; DATA SOURCE: COVID SYMPTOM TRACKER TEAM

Upset plots and a bit of wrangling

```
symptoms <- c("Anosmia", "Cough", "Fatigue",
             "Diarrhea", "Breath", "Fever")
names(symptoms) <- symptoms
symptoms

##      Anosmia      Cough      Fatigue      Diarrhea      Breath      Fever
## "Anosmia"    "Cough"    "Fatigue"   "Diarrhea"    "Breath"    "Fever"
```

Upset plots and a bit of wrangling

```
# An Excel file!
dat <- readxl::read_xlsx(here("data", "symptoms.xlsx"))
dat |> print(n = nrow(dat))
```

```
## # A tibble: 32 × 2
##   combination           count
##   <chr>                <dbl>
## 1 Anosmia                  140
## 2 Cough                      57
## 3 Fatigue                   198
## 4 Diarrhea                   12
## 5 Breath                      5
## 6 Fever                      11
## 7 Cough&Fatigue              179
## 8 Fatigue&Fever               28
## 9 Breath&Fatigue              10
## 10 Diarrhea&Fatigue             43
## 11 Anosmia&Fatigue            281
## 12 Breath&Cough                 1
## 13 Anosmia&Diarrhea&Fatigue      64
## 14 Breath&Cough&Fatigue            22
## 15 Anosmia&Cough&Fatigue            259
## 16 Anosmia&Fever&Fatigue             46
## 17 Cough&Fever&Fatigue              54
## 18 Cough&Diarrhea                  7
## 19 Cough&Diarrhea&Fatigue             31
## 20 Anosmia&Breath&Cough&Fatigue       26
## 21 Anosmia&Cough&Fatigue&Fever          69
## 22 Anosmia&Breath&Cough&Diarrhea&Fatigue     18
## 23 Anosmia&Breath&Cough&Fatigue&Fever        17
```

Upset plots and a bit of wrangling

```
subsets <- dat |>
  pull(combination)

## Check if each subset mentions each symptom or not
symptom_mat <- map_dfc(subsets, str_detect, symptoms) |>
  data.frame() |>
  t() |> # transpose the result, this is a little gross, sorry
  as_tibble(.name_repair = "unique")

colnames(symptom_mat) <- symptoms
symptom_mat$count <- dat$count
```

Upset plots and a bit of wrangling

Now we have a table we can do something with.

```
symptom_mat |> print(n = nrow(symptom_mat))

## # A tibble: 32 × 7
##   Anosmia Cough Fatigue Diarrhea Breath Fever count
##   <lgl>    <lgl> <lgl>    <lgl>   <lgl> <dbl>
## 1 TRUE     FALSE  FALSE    FALSE   FALSE  140
## 2 FALSE    TRUE   FALSE    FALSE   FALSE  57
## 3 FALSE    FALSE  TRUE    FALSE   FALSE  198
## 4 FALSE    FALSE  FALSE   TRUE    FALSE  12
## 5 FALSE    FALSE  FALSE   FALSE  TRUE   5
## 6 FALSE    FALSE  FALSE   FALSE  FALSE  11
## 7 FALSE    TRUE   TRUE    FALSE  FALSE  179
## 8 FALSE    FALSE  TRUE    FALSE  FALSE  28
## 9 FALSE    FALSE  TRUE    FALSE  TRUE   10
## 10 FALSE   FALSE  TRUE    TRUE   FALSE  43
## 11 TRUE    FALSE  TRUE    FALSE  FALSE  281
## 12 FALSE   TRUE   FALSE   FALSE  TRUE   1
## 13 TRUE    FALSE  TRUE    TRUE   FALSE  64
## 14 FALSE   TRUE   TRUE    FALSE  TRUE   22
## 15 TRUE    TRUE   TRUE    FALSE  FALSE  259
## 16 TRUE    FALSE  TRUE    FALSE  FALSE  46
## 17 FALSE   TRUE   TRUE    FALSE  FALSE  54
## 18 FALSE   TRUE   FALSE   TRUE   FALSE  7
## 19 FALSE   TRUE   TRUE    TRUE   FALSE  31
## 20 TRUE    TRUE   TRUE    FALSE  TRUE   26
## 21 TRUE    TRUE   TRUE    FALSE  TRUE   69
## 22 TRUE    TRUE   TRUE    TRUE   FALSE  18
## 23 TRUE    TRUE   TRUE    FALSE  TRUE   17
```

Upset plots and a bit of wrangling

Uncounting tables

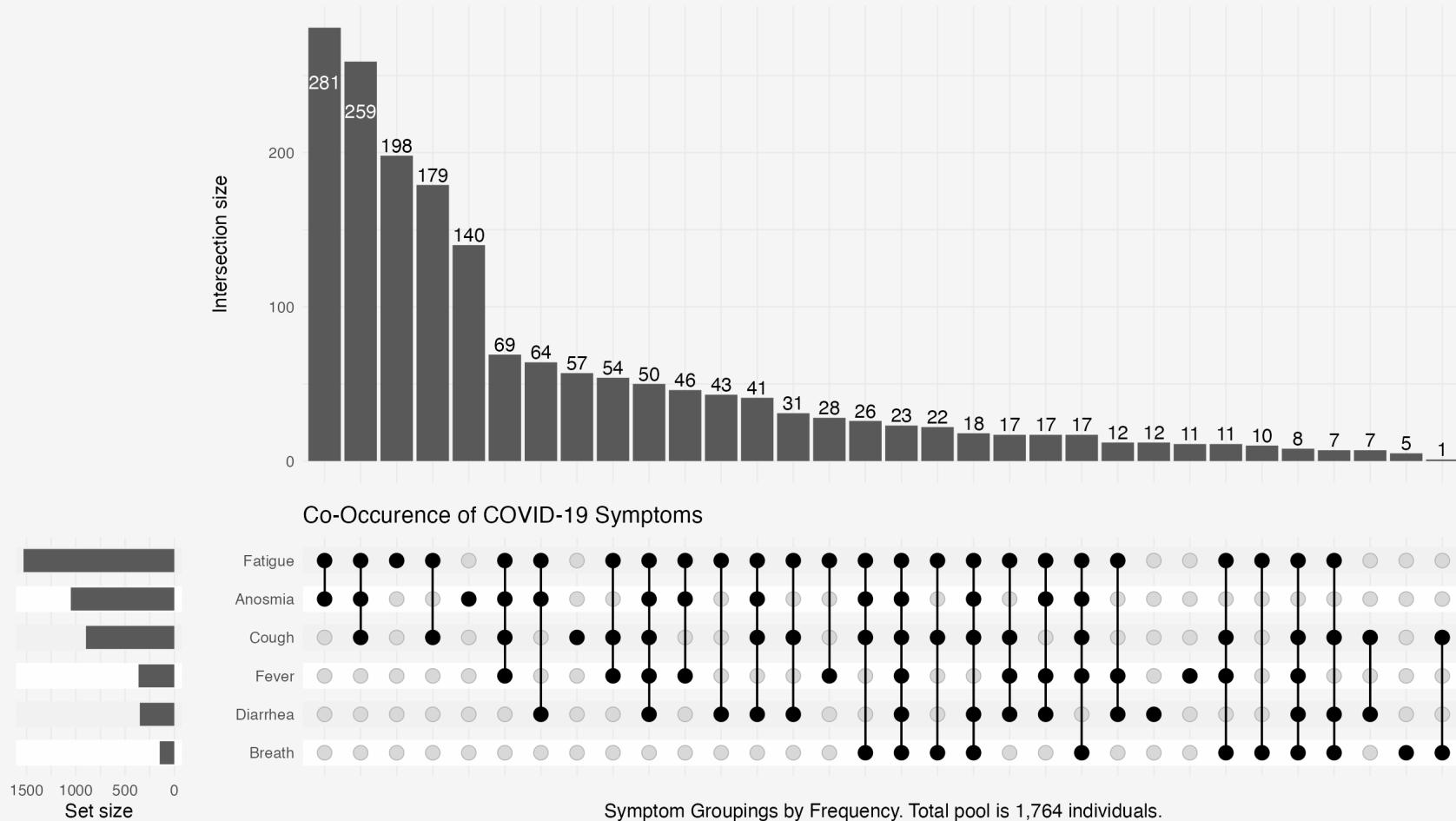
```
indvs <- symptom_mat |>  
  uncount(count)  
  
indvs  
  
## # A tibble: 1,764 × 6  
##   Anosmia Cough Fatigue Diarrhea Breath Fever  
##   <lgl>    <lgl> <lgl>    <lgl>    <lgl>  <lgl>  
## 1 TRUE     FALSE  FALSE    FALSE    FALSE  FALSE  
## 2 TRUE     FALSE  FALSE    FALSE    FALSE  FALSE  
## 3 TRUE     FALSE  FALSE    FALSE    FALSE  FALSE  
## 4 TRUE     FALSE  FALSE    FALSE    FALSE  FALSE  
## 5 TRUE     FALSE  FALSE    FALSE    FALSE  FALSE  
## 6 TRUE     FALSE  FALSE    FALSE    FALSE  FALSE  
## 7 TRUE     FALSE  FALSE    FALSE    FALSE  FALSE  
## 8 TRUE     FALSE  FALSE    FALSE    FALSE  FALSE  
## 9 TRUE     FALSE  FALSE    FALSE    FALSE  FALSE  
## 10 TRUE    FALSE  FALSE    FALSE    FALSE  FALSE  
## # ... with 1,754 more rows
```

Now we've reconstructed the individual-level observations.

Upset plots and a bit of wrangling

```
# devtools::install_github("krassowski/complex-upset")  
  
library(ComplexUpset)  
  
upset(data = indvs, intersect = symptoms,  
      name="Symptom Groupings by Frequency. Total pool is 1,764 individuals.",  
      min_size = 0,  
      width_ratio = 0.125) +  
  labs(title = "Co-Occurrence of COVID-19 Symptoms",  
       caption = "Data: covid.joinzoe.com/us | Graph: @kjhealy")
```

Upset plots and a bit of wrangling



Data: covid.joinzoe.com/us | Graph: @kjhealy

Models

This is not a **statistics** seminar!

I'll just give you an example of the sort of thing that many other modeling packages implement for all kinds of modeling techniques.

Again, the principle is tidy incorporation of models and their output.

Tidy regression output with **broom**

```
library(broom)
library(gapminder)

out <- lm(formula = lifeExp ~ gdpPercap + pop + continent,
          data = gapminder)
```

Tidy regression output with broom

We can't *do* anything with this, programatically.

```
summary(out)

##
## Call:
## lm(formula = lifeExp ~ gdpPercap + pop + continent, data = gapminder)
##
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -49.161  -4.486   0.297   5.110  25.175 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 4.781e+01 3.395e-01 140.819 < 2e-16 ***
## gdpPercap   4.495e-04 2.346e-05 19.158 < 2e-16 ***
## pop         6.570e-09 1.975e-09  3.326 0.000901 *** 
## continentAmericas 1.348e+01 6.000e-01 22.458 < 2e-16 ***
## continentAsia    8.193e+00 5.712e-01 14.342 < 2e-16 ***
## continentEurope   1.747e+01 6.246e-01 27.973 < 2e-16 ***
## continentOceania  1.808e+01 1.782e+00 10.146 < 2e-16 *** 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.365 on 1697 degrees of freedom
## Multiple R-squared:  0.5821,    Adjusted R-squared:  0.5806 
## F-statistic: 393.9 on 6 and 1697 DF,  p-value: < 2.2e-16
```

Tidy regression output with **broom**

```
library(broom)

tidy(out)

## # A tibble: 7 × 5
##   term      estimate std.error statistic p.value
##   <chr>     <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept) 4.78e+1  0.340     141.     0
## 2 gdpPercap    4.50e-4  0.0000235    19.2    3.24e- 74
## 3 pop         6.57e-9  0.00000000198    3.33   9.01e- 4
## 4 continentAmericas 1.35e+1  0.600     22.5    5.19e- 98
## 5 continentAsia  8.19e+0  0.571     14.3    4.06e- 44
## 6 continentEurope 1.75e+1  0.625     28.0    6.34e-142
## 7 continentOceania 1.81e+1  1.78      10.1   1.59e- 23
```

That's a *lot* nicer. Now it's just a tibble. We know those.

Tidy regression output with broom

```
out_conf <- tidy(out, conf.int = TRUE)
out_conf

## # A tibble: 7 × 7
##   term      estimate    std.error statistic   p.value conf.low conf.h...
##   <chr>     <dbl>       <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept) 4.78e+1  0.340        141.     0  4.71e+1  4.85e+1
## 2 gdpPerCap   4.50e-4  0.0000235     19.2  3.24e- 74  4.03e-4  4.96e-4
## 3 pop         6.57e-9  0.00000000198   3.33  9.01e- 4  2.70e-9  1.04e-8
## 4 continentAmericas 1.35e+1  0.600        22.5  5.19e- 98  1.23e+1  1.47e+1
## 5 continentAsia  8.19e+0  0.571        14.3  4.06e- 44  7.07e+0  9.31e+0
## 6 continentEurope 1.75e+1  0.625        28.0  6.34e-142 1.62e+1  1.87e+1
## 7 continentOceania 1.81e+1  1.78        10.1  1.59e- 23  1.46e+1  2.16e+1
## # ... with abbreviated variable name `conf.high`
```

Tidy regression output with broom

```
out_conf |>
  filter(term %nin% "(Intercept)") |>
  mutate(nicelabs = prefix_strip(term, "continent")) |>
  select(nicelabs, everything())

## # A tibble: 6 × 8
##   nicelabs term      estimate std.e...¹ stati...² p.value conf....³ conf....⁴
##   <chr>     <chr>      <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 gdpPercap gdpPercap  4.50e-4  2.35e-5   19.2   3.24e- 74  4.03e-4  4.96e-4
## 2 Pop       pop        6.57e-9  1.98e-9   3.33   9.01e- 4  2.70e-9  1.04e-8
## 3 Americas  continentAmericas 1.35e+1  6.00e-1   22.5   5.19e- 98  1.23e+1  1.47e+1
## 4 Asia      continentAsia   8.19e+0  5.71e-1   14.3   4.06e- 44  7.07e+0  9.31e+0
## 5 Europe    continentEurope  1.75e+1  6.25e-1   28.0   6.34e-142 1.62e+1  1.87e+1
## 6 Oceania   continentOceania 1.81e+1  1.78e+0   10.1   1.59e- 23  1.46e+1  2.16e+1
## # ... with abbreviated variable names ¹std.error, ²statistic, ³conf.low,
## #   ⁴conf.high
```

Grouped analysis and **list columns**

```
eu77 <- gapminder |> filter(continent == "Europe", year == 1977)
fit <- lm(lifeExp ~ log(gdpPercap), data = eu77)

summary(fit)

##
## Call:
## lm(formula = lifeExp ~ log(gdpPercap), data = eu77)
##
## Residuals:
##     Min      1Q      Median      3Q      Max
## -7.4956 -1.0306  0.0935  1.1755  3.7125
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 29.489     7.161   4.118 0.000306 ***
## log(gdpPercap) 4.488     0.756   5.936 2.17e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.114 on 28 degrees of freedom
## Multiple R-squared:  0.5572,    Adjusted R-squared:  0.5414
## F-statistic: 35.24 on 1 and 28 DF,  p-value: 2.173e-06
```

Grouped analysis and **list columns**

```
out_le <- gapminder |>
  group_by(continent, year) |>
  nest()

out_le

## # A tibble: 60 × 3
## # Groups:   continent, year [60]
##   continent   year data
##   <fct>     <int> <list>
## 1 Asia        1952 <tibble [33 × 4]>
## 2 Asia        1957 <tibble [33 × 4]>
## 3 Asia        1962 <tibble [33 × 4]>
## 4 Asia        1967 <tibble [33 × 4]>
## 5 Asia        1972 <tibble [33 × 4]>
## 6 Asia        1977 <tibble [33 × 4]>
## 7 Asia        1982 <tibble [33 × 4]>
## 8 Asia        1987 <tibble [33 × 4]>
## 9 Asia        1992 <tibble [33 × 4]>
## 10 Asia       1997 <tibble [33 × 4]>
## # ... with 50 more rows
```

Think of nesting as a kind of "super-grouping". Look in the object inspector.

Grouped analysis and **list columns**

It's still in there.

```
out_le |> filter(continent == "Europe" & year == 1977) |>  
unnest(cols = c(data))  
  
## # A tibble: 30 × 6  
## # Groups: continent, year [1]  
##   continent year country             lifeExp     pop gdpPercap  
##   <fct>     <int> <fct>           <dbl>     <int>     <dbl>  
## 1 Europe     1977 Albania          68.9    2509048    3533.  
## 2 Europe     1977 Austria          72.2    7568430    19749.  
## 3 Europe     1977 Belgium          72.8    9821800    19118.  
## 4 Europe     1977 Bosnia and Herzegovina 69.9    4086000    3528.  
## 5 Europe     1977 Bulgaria         70.8    8797022    7612.  
## 6 Europe     1977 Croatia          70.6    4318673    11305.  
## 7 Europe     1977 Czech Republic  70.7    10161915   14800.  
## 8 Europe     1977 Denmark          74.7    5088419    20423.  
## 9 Europe     1977 Finland          72.5    4738902    15605.  
## 10 Europe    1977 France           73.8    53165019   18293.  
## # ... with 20 more rows
```

Grouped analysis and **list columns**

Here we **map()** a custom function to every row in the data column.

```
fit_ols <- function(df) {  
  lm(lifeExp ~ log(gdpPerCap), data = df)  
}  
  
out_le <- gapminder |>  
  group_by(continent, year) |>  
  nest() |>  
  mutate(model = map(data, fit_ols))
```

Grouped analysis and **list** columns

out_le

```
## # A tibble: 60 × 4
## # Groups: continent, year [60]
##   continent year data          model
##   <fct>     <int> <list>        <list>
## 1 Asia       1952 <tibble [33 × 4]> <lm>
## 2 Asia       1957 <tibble [33 × 4]> <lm>
## 3 Asia       1962 <tibble [33 × 4]> <lm>
## 4 Asia       1967 <tibble [33 × 4]> <lm>
## 5 Asia       1972 <tibble [33 × 4]> <lm>
## 6 Asia       1977 <tibble [33 × 4]> <lm>
## 7 Asia       1982 <tibble [33 × 4]> <lm>
## 8 Asia       1987 <tibble [33 × 4]> <lm>
## 9 Asia       1992 <tibble [33 × 4]> <lm>
## 10 Asia      1997 <tibble [33 × 4]> <lm>
## # ... with 50 more rows
```

Grouped analysis and **list columns**

We can tidy the nested models, too.

```
fit_ols <- function(df) {  
  lm(lifeExp ~ log(gdpPerCap), data = df)  
}  
  
out_tidy <- gapminder |>  
  group_by(continent, year) |>  
  nest() |>  
  mutate(model = map(data, fit_ols),  
         tidied = map(model, tidy)) |>  
  unnest(cols = c(tidied)) |>  
  filter(term %nin% "(Intercept)" &  
        continent %nin% "Oceania")
```

Grouped analysis and **list columns**

out_tidy

```
## # A tibble: 48 × 9
## # Groups: continent, year [48]
##   continent year data           model term estim...¹ std.e..² stati...³ p.value
##   <fct>     <int> <list>          <lis> <chr>    <dbl>    <dbl>    <dbl>
## 1 Asia       1952 <tibble [33 × 4]> <lm>  log(...)  4.16    1.25    3.33  2.28e-3
## 2 Asia       1957 <tibble [33 × 4]> <lm>  log(...)  4.17    1.28    3.26  2.71e-3
## 3 Asia       1962 <tibble [33 × 4]> <lm>  log(...)  4.59    1.24    3.72  7.94e-4
## 4 Asia       1967 <tibble [33 × 4]> <lm>  log(...)  4.50    1.15    3.90  4.77e-4
## 5 Asia       1972 <tibble [33 × 4]> <lm>  log(...)  4.44    1.01    4.41  1.16e-4
## 6 Asia       1977 <tibble [33 × 4]> <lm>  log(...)  4.87    1.03    4.75  4.42e-5
## 7 Asia       1982 <tibble [33 × 4]> <lm>  log(...)  4.78    0.852   5.61  3.77e-6
## 8 Asia       1987 <tibble [33 × 4]> <lm>  log(...)  5.17    0.727   7.12  5.31e-8
## 9 Asia       1992 <tibble [33 × 4]> <lm>  log(...)  5.09    0.649   7.84  7.60e-9
## 10 Asia      1997 <tibble [33 × 4]> <lm>  log(...)  5.11    0.628   8.15  3.35e-9
## # ... with 38 more rows, and abbreviated variable names `¹estimate, `²std.error,
## #       `³statistic
```

Grouped analysis and **list columns**

```
out_tidy |>
  ungroup() |>
  sample_n(5)

## # A tibble: 5 × 9
##   continent year data           model  term estim...¹ std.e...² stati...³ p.value
##   <fct>     <int> <list>         <list> <chr>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 Asia       1977 <tibble [33 × 4]> <lm>    log(...) 4.87   1.03   4.75  4.42e-5
## 2 Asia       1997 <tibble [33 × 4]> <lm>    log(...) 5.11   0.628   8.15  3.35e-9
## 3 Americas   1997 <tibble [25 × 4]> <lm>    log(...) 5.39   0.859   6.28  2.08e-6
## 4 Asia       1962 <tibble [33 × 4]> <lm>    log(...) 4.59   1.24    3.72  7.94e-4
## 5 Asia       2002 <tibble [33 × 4]> <lm>    log(...) 5.44   0.696   7.82  8.05e-9
## # ... with abbreviated variable names `¹estimate, `²std.error, `³statistic
```