Week 11: *Programming with Data*

🏛 EMSE 4571 / 6571: Intro to Programming for Analytics

👤 John Paul Helveston

📅 April 11, 2024

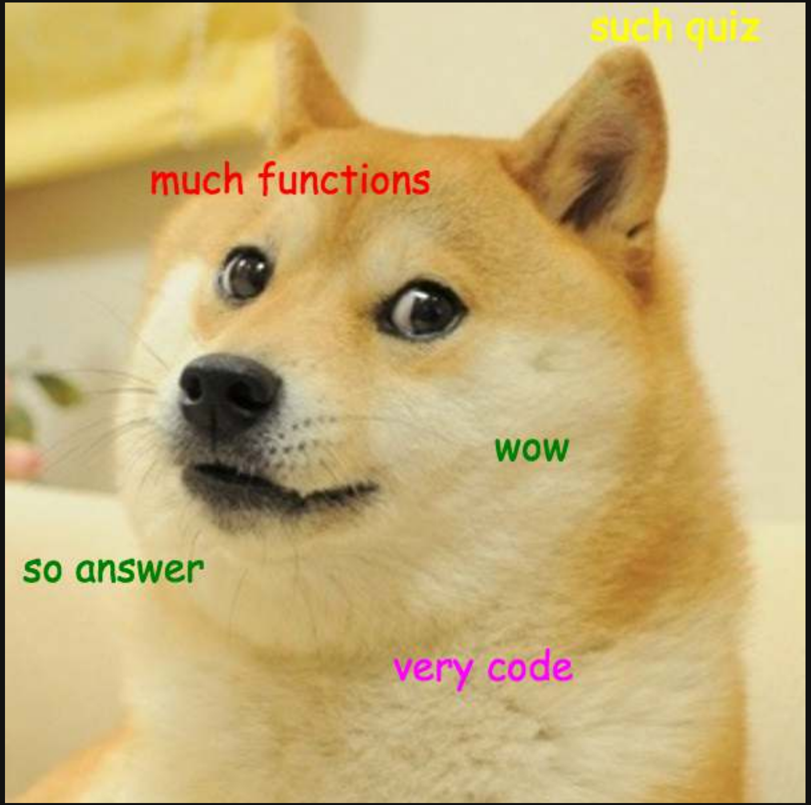
Quiz 6

10:00

Write your name on the quiz!

Rules:

- Work alone; no outside help of any kind is allowed.
- No calculators, no notes, no books, no computers, no phones.



Week 11: *Programming with Data*

1. Writing functions for data frames

2. Writing custom plot functions

BREAK

3. Iteration with purrr

Week 11: *Programming with Data*

1. Writing functions for data frames

2. Writing custom plot functions

BREAK

3. Iteration with purrr

Motivation

I want a summary of a variable in a data frame:

```
head(diamonds)
```

```
#> # A tibble: 6 × 10
#>   carat cut       color clarity depth table price     x     y     z
#>   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
#> 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
#> 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
#> 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
#> 4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
#> 5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
#> 6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

Motivation

I want a summary of a variable in a data frame:

```
length(diamonds$price)
```

```
#> [1] 53940
```

```
mean(diamonds$price)
```

```
#> [1] 3932.8
```

```
sd(diamonds$price)
```

```
#> [1] 3989.44
```

Motivation

I want a summary of a variable in a data frame:

```
diamonds %>%
  summarise(
    n = n(),
    mean = mean(price),
    sd = sd(price)
  )
```

```
#> # A tibble: 1 × 3
#>       n  mean    sd
#>   <int> <dbl> <dbl>
#> 1 53940 3933. 3989.
```

Motivation

I can get **grouped** summaries really easily now:

```
diamonds %>%
  group_by(cut) %>%
  summarise(
    n = n(),
    mean = mean(price),
    sd = sd(price)
  )
```

```
#> # A tibble: 5 × 4
#>   cut           n  mean    sd
#>   <ord>     <int> <dbl> <dbl>
#> 1 Fair       1610 4359. 3560.
#> 2 Good       4906 3929. 3682.
#> 3 Very Good 12082 3982. 3936.
#> 4 Premium   13791 4584. 4349.
#> 5 Ideal     21551 3458. 3808.
```

```
diamonds %>%
  group_by(color) %>%
  summarise(
    n = n(),
    mean = mean(price),
    sd = sd(price)
  )
```

```
#> # A tibble: 7 × 4
#>   color     n  mean    sd
#>   <ord> <int> <dbl> <dbl>
#> 1 D      6775 3170. 3357.
#> 2 E      9797 3077. 3344.
#> 3 F      9542 3725. 3785.
#> 4 G     11292 3999. 4051.
#> 5 H      8304 4487. 4216.
#> 6 I      5422 5092. 4722.
#> 7 J      2808 5324. 4438.
```

Convert this to a function

```r
diamonds %>%
  group_by(color) %>%
  summarise(
    n = n(),
    mean = mean(price),
    sd = sd(price)
  )
```

```
#> # A tibble: 7 × 4
#>   color     n  mean    sd
#>   <ord> <int> <dbl> <dbl>
#> 1 D      6775 3170. 3357.
#> 2 E      9797 3077. 3344.
#> 3 F      9542 3725. 3785.
#> 4 G     11292 3999. 4051.
#> 5 H      8304 4487. 4216.
#> 6 I      5422 5092. 4722.
#> 7 J      2808 5324. 4438.
```

```r
my_summary <- function(df, var) {
  df %>%
    group_by(var) %>%
    summarise(
      n = n(),
      mean = mean(price),
      sd = sd(price)
    )
}
```

...but this doesn't work

```r
my_summary <- function(df, var) {
  df %>%
    group_by(var) %>%
    summarise(
      n = n(),
      mean = mean(price),
      sd = sd(price)
    )
}

my_summary(diamonds, color)
```

```
#> Error in `group_by()`:
#> ! Must group by variables found in `.data`.
#> ✖ Column `var` is not found.
```

Solution: "embrace" your variables 🤗

```r
my_summary <- function(df, var) {
  df %>%
    group_by({{ var }}) %>%
    summarise(
      n = n(),
      mean = mean(price),
      sd = sd(price)
    )
}
```

```r
my_summary(diamonds, cut)
```

```
#> # A tibble: 5 × 4
#>   cut           n  mean    sd
#>   <ord>     <int> <dbl> <dbl>
#> 1 Fair       1610 4359. 3560.
#> 2 Good       4906 3929. 3682.
#> 3 Very Good 12082 3982. 3936.
#> 4 Premium   13791 4584. 4349.
#> 5 Ideal     21551 3458. 3808.
```

```r
my_summary(diamonds, color)
```

```
#> # A tibble: 7 × 4
#>   color     n  mean    sd
#>   <ord> <int> <dbl> <dbl>
#> 1 D      6775 3170. 3357.
#> 2 E      9797 3077. 3344.
#> 3 F      9542 3725. 3785.
#> 4 G     11292 3999. 4051.
```

Make it even more general!

```
my_summary <- function(df, group, var) {
  df %>%
    group_by({{ group }}) %>%
    summarise(
      n = n(),
      mean = mean({{ var }}),
      sd = sd({{ var }})
    )
}
```

```
my_summary(diamonds, group = cut, var = price)
```

```
#> # A tibble: 5 × 4
#>   cut           n  mean    sd
#>   <ord>     <int> <dbl> <dbl>
#> 1 Fair       1610 4359. 3560.
#> 2 Good       4906 3929. 3682.
#> 3 Very Good 12082 3982. 3936.
#> 4 Premium   13791 4584. 4349.
#> 5 Ideal     21551 3458. 3808.
```

Make it even more general!

```r
my_summary <- function(df, group, var) {
  df %>%
    group_by({{ group }}) %>%
    summarise(
      n = n(),
      mean = mean({{ var }}),
      sd = sd({{ var }})
    )
}
```

```r
my_summary(diamonds, group = color, var = carat)
```

```
#> # A tibble: 7 × 4
#>   color     n  mean    sd
#>   <ord> <int> <dbl> <dbl>
#> 1 D      6775 0.658 0.360
#> 2 E      9797 0.658 0.369
#> 3 F      9542 0.737 0.398
#> 4 G     11292 0.771 0.441
#> 5 H      8304 0.912 0.521
#> 6 I      5422 1.03  0.579
#> 7 J      2808 1.16  0.596
```

Use it on a different data frame!

```r
library(palmerpenguins)

glimpse(penguins)
```

```
#> Rows: 344
#> Columns: 8
#> $ species           <fct> Adelie, Adelie,
#> $ island            <fct> Torgersen, Torg
#> $ bill_length_mm    <dbl> 39.1, 39.5, 40.
#> $ bill_depth_mm     <dbl> 18.7, 17.4, 18.
#> $ flipper_length_mm <int> 181, 186, 195,
#> $ body_mass_g       <int> 3750, 3800, 325
#> $ sex               <fct> male, female, f
#> $ year              <int> 2007, 2007, 200
```

```r
my_summary(penguins, sex, body_mass_g)
```

```
#> # A tibble: 3 × 4
#>   sex        n  mean    sd
#>   <fct>  <int> <dbl> <dbl>
#> 1 female   165 3862.  666.
#> 2 male     168 4546.  788.
#> 3 <NA>      11   NA    NA
```

```r
my_summary(penguins, species, bill_length_m
```

```
#> # A tibble: 3 × 4
#>   species        n  mean    sd
#>   <fct>     <int> <dbl> <dbl>
#> 1 Adelie      152  NA   NA
#> 2 Chinstrap    68  48.8  3.34
#> 3 Gentoo      124  NA   NA
```

Defining a filter condition

```r
filter_summary <- function(df, condition, var) {
  df %>%
    filter({{ condition }}) %>%
    summarise(
      n = n(),
      mean = mean({{ var }}, na.rm = TRUE),
      sd = sd({{ var }}, na.rm = TRUE)
    )
}
```

```r
filter_summary(penguins, species == 'Adelie', bill_length_mm)
```

```
#> # A tibble: 1 × 3
#>       n  mean    sd
#>   <int> <dbl> <dbl>
#> 1   152  38.8  2.66
```

Your turn - write the following functions

15:00

```
my_subset <- function(df, condition, cols)
```

Returns a subset of `df` by filtering the rows based on `condition` and only includes the select `cols`. Example:

```
nycflights13::flights %>%
  my_subset(month == 12, c("carrier", "flig
```

```
#> # A tibble: 5 × 2
#>   carrier flight
#>   <chr>    <int>
#> 1 B6         745
#> 2 B6         839
#> 3 US        1895
#> 4 UA        1487
#> 5 AA        2243
```

```
count_p <- function(df, group)
```

Returns a summary data frame of the count of rows in `df` by `group` as well as the percentage of those counts.

```
nycflights13::flights %>%
  count_p(carrier)
```

```
#> # A tibble: 6 × 3
#>   carrier     n      p
#>   <chr>   <int>  <dbl>
#> 1 UA      58665 0.174
#> 2 B6      54635 0.162
#> 3 EV      54173 0.161
#> 4 DL      48110 0.143
#> 5 AA      32729 0.0972
#> 6 MQ      26397 0.0784
```

Testing data frame functions

Function:

```r
my_summary <- function(df, group, var) {
  df %>%
    group_by({{ group }}) %>%
    summarise(
      n = n(),
      mean = mean({{ var }}),
      sd = sd({{ var }})
    )
}
```

Make two data frames and compare them

```r
test_my_summary <- function() {

  cat("Testing my_summary()...")

  df1 <- diamonds %>%
    my_summary(cut, price)

  df2 <- diamonds %>%
    group_by(cut) %>%
    summarise(
      n = n(),
      mean = mean(price),
      sd = sd(price)
    )

  stopifnot(identical(df1, df2))

  cat("passed!")
}

test_my_summary()
```

```
#> Testing my_summary()...passed!
```

Week 11: *Programming with Data*

1. Writing functions for data frames

2. Writing custom plot functions

BREAK

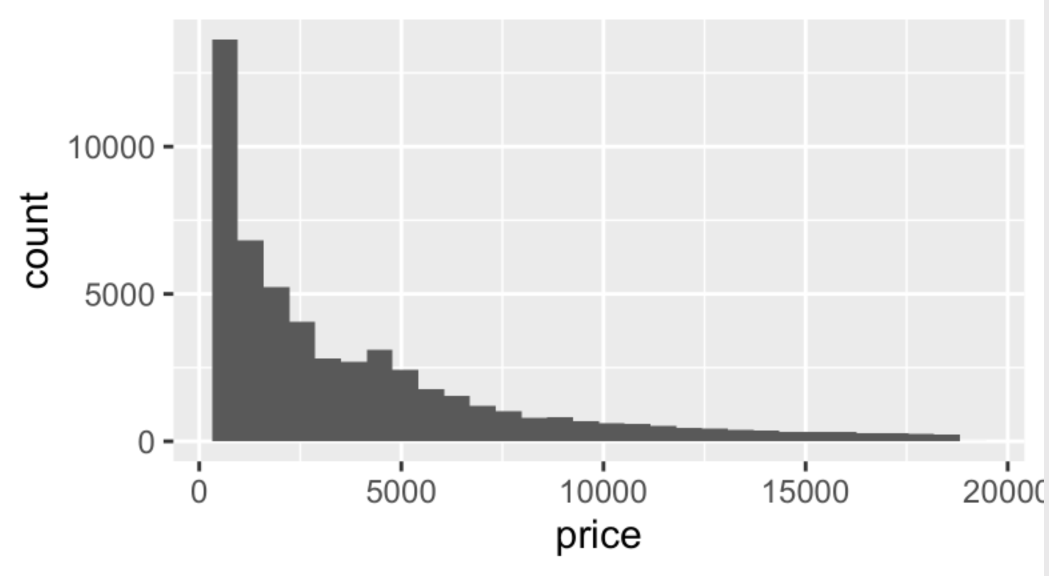
3. Iteration with purrr

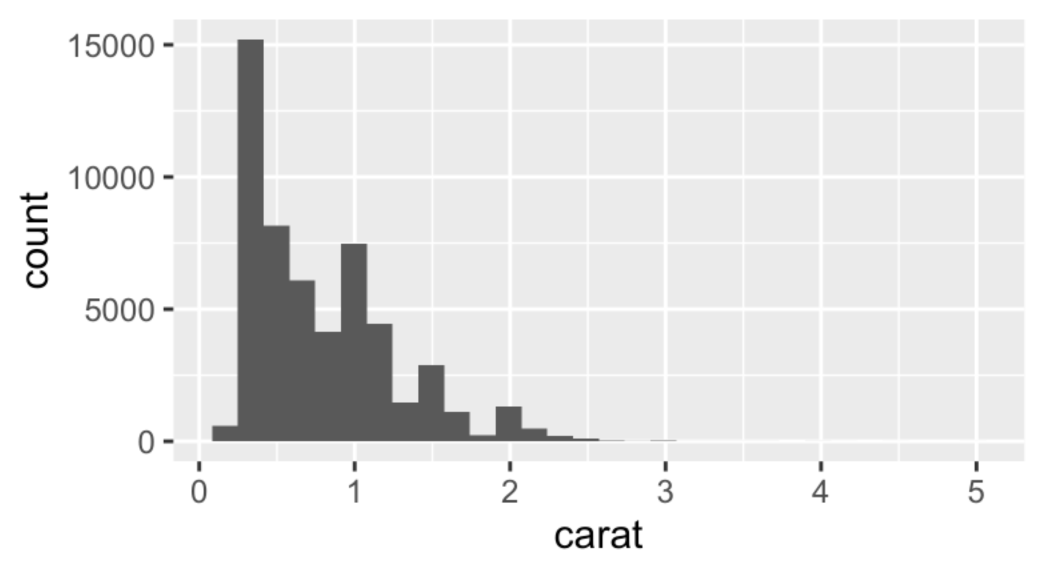
Motivation

I want to see a histogram of multiple variables

```
diamonds %>%
  ggplot() +
  geom_histogram((aes(x = price)))
```

```
diamonds %>%
  ggplot() +
  geom_histogram((aes(x = carat)))
```



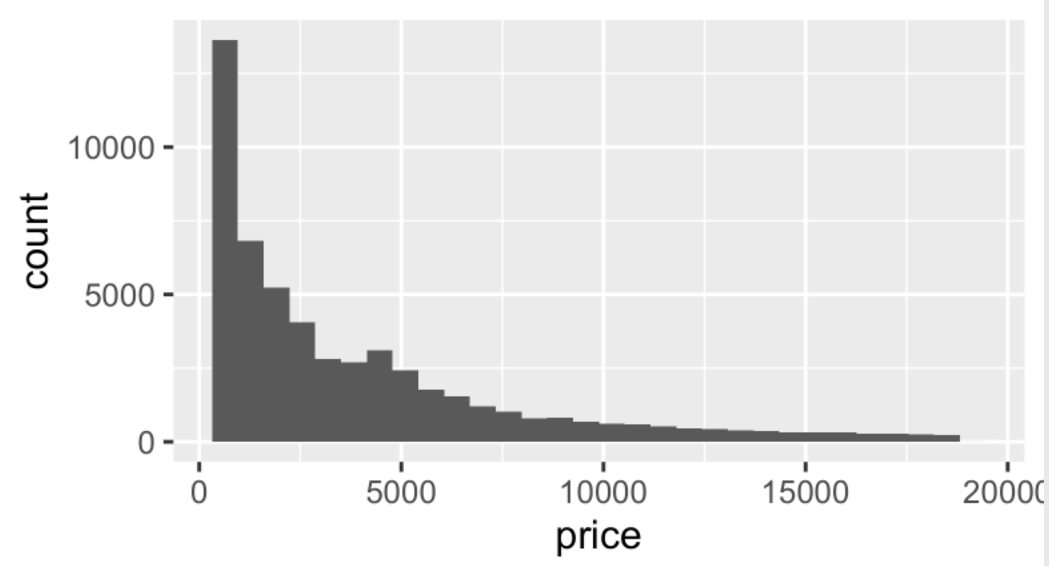


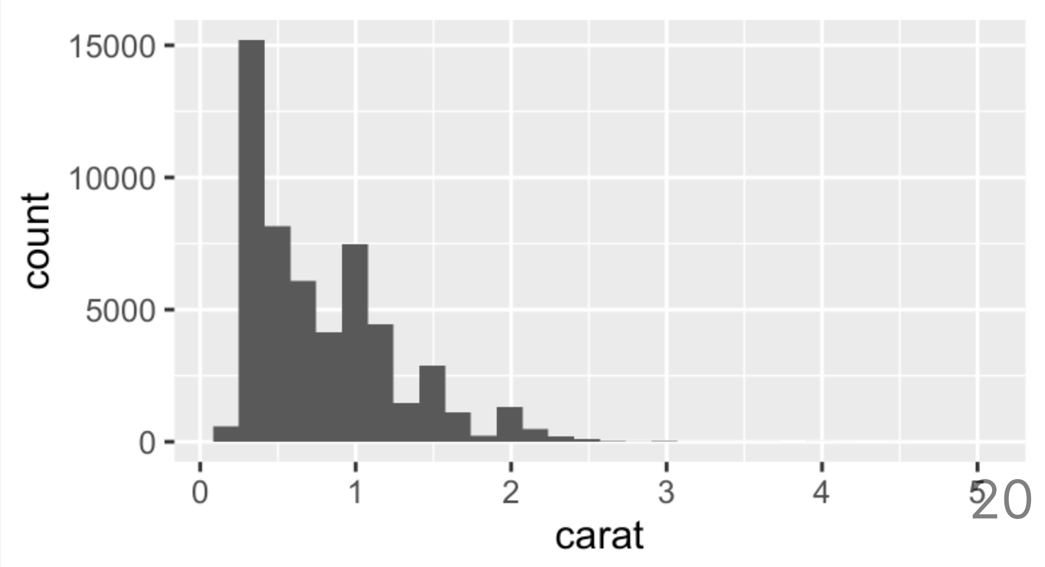
Convert this to a function

```
my_hist <- function(df, var) {
  df %>%
    ggplot() +
    geom_histogram((aes(x = {{ var }}))) # <<
}
```

```
my_hist(diamonds, price)
```

```
my_hist(diamonds, carat)
```



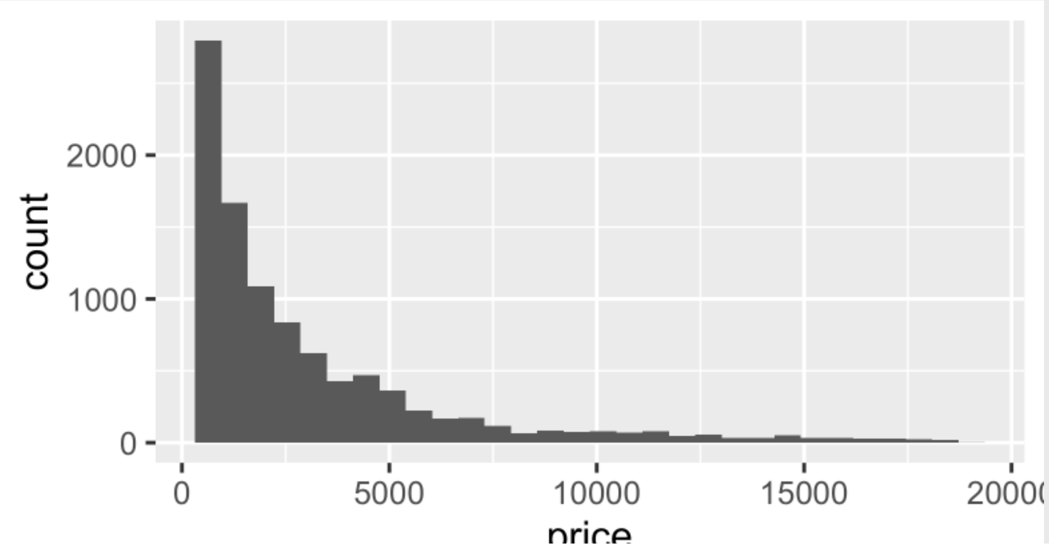


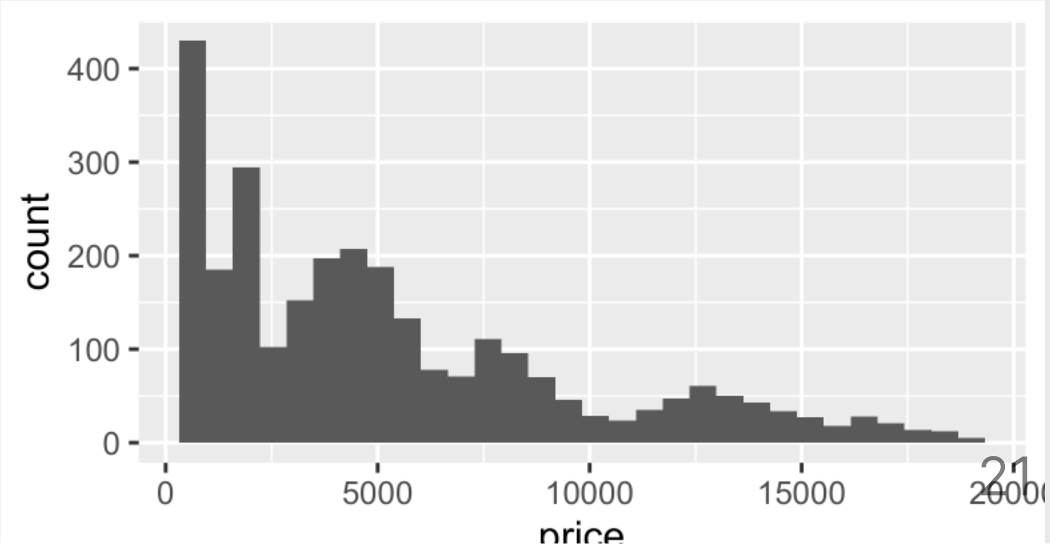
Combine with other functions

```
filtered_hist <- function(df, condition, var) {
  df %>%
    filter({{ condition }}) %>%
    ggplot() +
    geom_histogram((aes(x = {{ var }})))
}
```

```
filtered_hist(diamonds, color == "E", price
```

```
filtered_hist(diamonds, color == "J", price
```





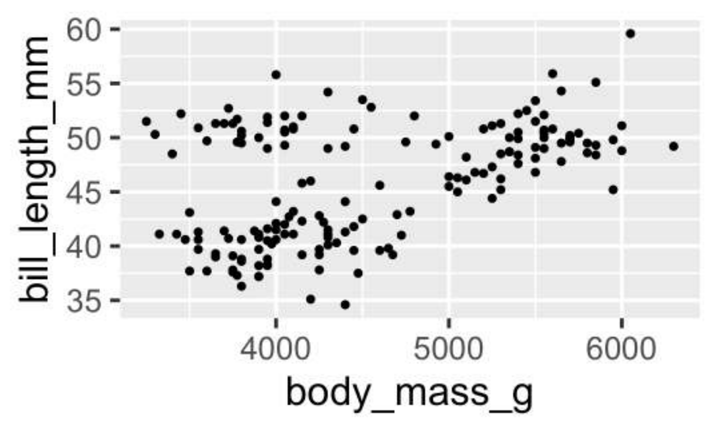
Your turn

10:00

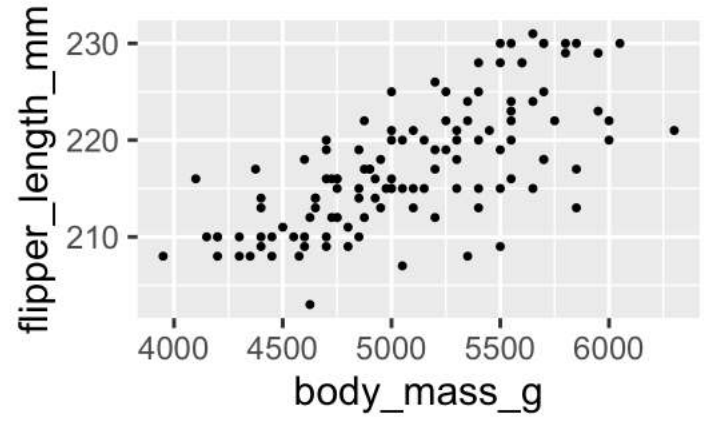
Write the function `filtered_scatter` which plots a scatterplot based on a condition, then use it for the two examples below.

```r
filtered_scatter <- function(df, condition, x, y)
```

```r
filtered_scatter(
  penguins, sex == "male",
  x = body_mass_g, y = bill_length_mm)
```

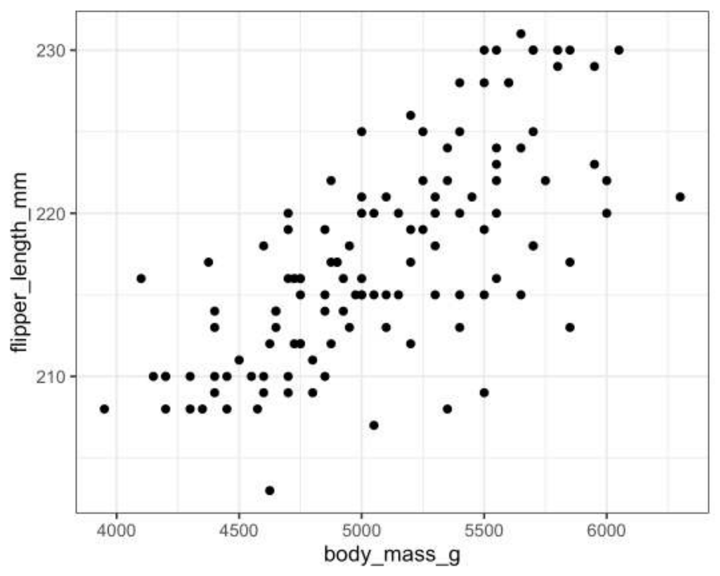


```r
filtered_scatter(
  penguins, species == "Gentoo",
  x = body_mass_g, y = flipper_length_mm)
```



You can add layers to your custom plot functions

```r
filtered_scatter(
  penguins, species == "Gentoo",
  x = body_mass_g, y = flipper_length_mm) +
  theme_bw()
```



Break

05:00

Week 11: *Programming with Data*

1. Writing functions for data frames

2. Writing custom plot functions

BREAK

3. Iteration with purrr

Much of this content is adapted from Shannon Pileggi's workshop at https://github.com/shannonpileggi/iterating-well-with-purrr

Iterating *without* purrr

Gapminder example

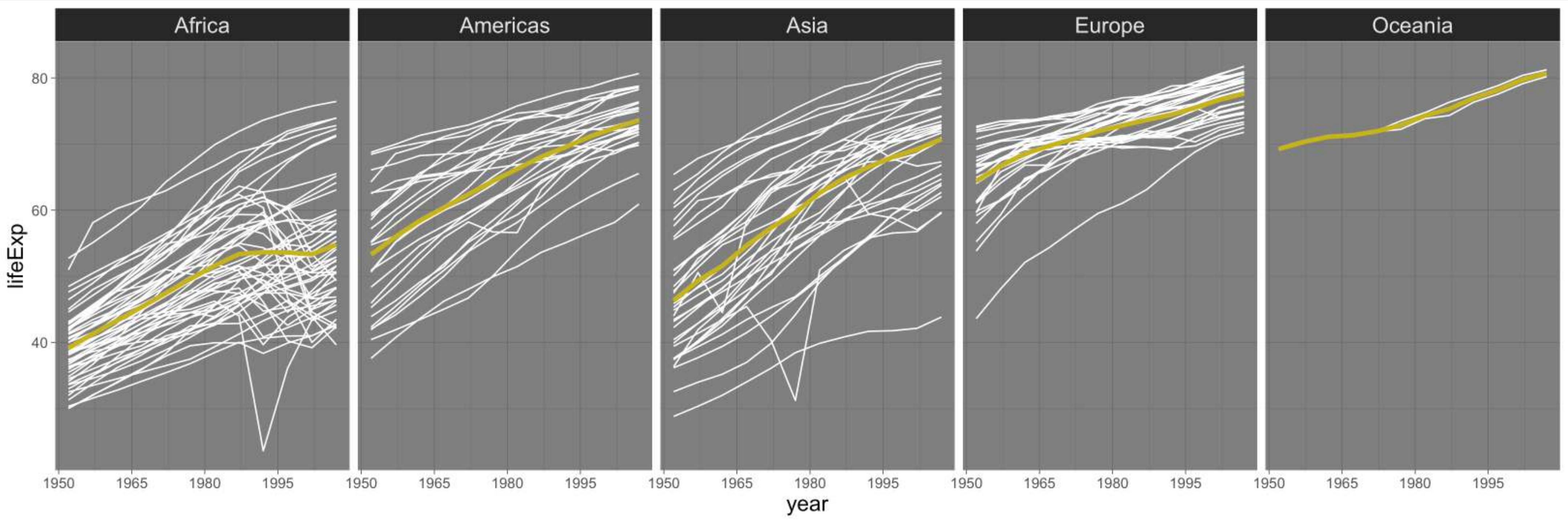
```r
library(gapminder)
library(tidyverse)

head(gapminder)
```

```
#> # A tibble: 6 × 6
#>   country     continent  year lifeExp      pop gdpPercap
#>   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
#> 1 Afghanistan Asia       1952    28.8  8425333      779.
#> 2 Afghanistan Asia       1957    30.3  9240934      821.
#> 3 Afghanistan Asia       1962    32.0 10267083      853.
#> 4 Afghanistan Asia       1967    34.0 11537966      836.
#> 5 Afghanistan Asia       1972    36.1 13079460      740.
#> 6 Afghanistan Asia       1977    38.4 14880372      786.
```

Hans Rosling discusses Gapminder data https://youtu.be/hVimVzgtD6w

Gapminder life expectancy



What am I doing here? Are there mistakes?

01:00

```
africa <- gapminder[gapminder$continent == "Africa", ]
africa_mm <- max(africa$lifeExp) - min(africa$lifeExp)

americas <- gapminder[gapminder$continent == "Americas", ]
americas_mm <- max(americas$lifeExp) - min(americas$lifeExp)

asia <- gapminder[gapminder$continent == "Asia", ]
asia_mm <- max(asia$lifeExp) - min(africa$lifeExp)

europe <- gapminder[gapminder$continent == "Europe", ]
europe_mm <- max(europe$lifeExp) - min(europe$lifeExp)

oceania <- gapminder[gapminder$continent == "Oceania", ]
oceania_mm <- max(europe$lifeExp) - min(oceania$lifeExp)

cbind(
  continent = c("Africa", "Asias", "Europe", "Oceania"),
  max_minus_min = c(africa_mm, americas_mm, asia_mm, europe_mm, oceania_mm)
)
```

1. What are the drawbacks of this code?

2. How would you do it instead?

```r
africa <- gapminder[gapminder$continent == "Africa", ]
africa_mm <- max(africa$lifeExp) - min(africa$lifeExp)

americas <- gapminder[gapminder$continent == "Americas", ]
americas_mm <- max(americas$lifeExp) - min(americas$lifeExp)

asia <- gapminder[gapminder$continent == "Asia", ]
asia_mm <- max(asia$lifeExp) - min(africa$lifeExp)

europe <- gapminder[gapminder$continent == "Europe", ]
europe_mm <- max(europe$lifeExp) - min(europe$lifeExp)

oceania <- gapminder[gapminder$continent == "Oceania", ]
oceania_mm <- max(europe$lifeExp) - min(oceania$lifeExp)

cbind(
  continent = c("Africa", "Asias", "Europe", "Oceania"),
  max_minus_min = c(africa_mm, americas_mm, asia_mm, europe_mm, oceania_mm)
)
```

An alternative solution

```
gapminder %>%
 group_by(continent) %>%
 summarize(max_minus_min = max(lifeExp) - min(lifeExp))
```

group_by approach

```
#> # A tibble: 5 × 2
#>   continent max_minus_min
#>   <fct>             <dbl>
#> 1 Africa             52.8
#> 2 Americas           43.1
#> 3 Asia               53.8
#> 4 Europe             38.2
#> 5 Oceania            12.1
```

previous approach

```
#>      continent max_minus_min
#> [1,] "Africa"  "52.843"
#> [2,] "Asias"   "43.074"
#> [3,] "Europe"  "59.004"
#> [4,] "Oceania" "38.172"
#> [5,] "Africa"  "12.637"
```

More iteration

```r
year <- 2017:2021
location <- c("Orlando", "San Diego", "Austin", "San Francisco", "remote")

conf <- rep("", length(year))
for (i in 1:length(conf)) {
 conf[i] <- paste0("The ", year[i], " RStudio Conference was in ", location[i], ".")
}
conf
```

```
#> [1] "The 2017 RStudio Conference was in Orlando."
#> [2] "The 2018 RStudio Conference was in San Diego."
#> [3] "The 2019 RStudio Conference was in Austin."
#> [4] "The 2020 RStudio Conference was in San Francisco."
#> [5] "The 2021 RStudio Conference was in remote."
```

Can you think of other ways to do this?

More iteration, cont.

```r
year <- 2017:2021
location <- c("Orlando", "San Diego", "Austin", "San Francisco", "remote")
```

```r
paste0("The ", year, " RStudio Conference was in ", location, ".")
```

```
#> [1] "The 2017 RStudio Conference was in Orlando."
#> [2] "The 2018 RStudio Conference was in San Diego."
#> [3] "The 2019 RStudio Conference was in Austin."
#> [4] "The 2020 RStudio Conference was in San Francisco."
#> [5] "The 2021 RStudio Conference was in remote."
```

```r
glue::glue("The {year} RStudio Conference was in {location}.")
```

```
#> The 2017 RStudio Conference was in Orlando.
#> The 2018 RStudio Conference was in San Diego.
#> The 2019 RStudio Conference was in Austin.
#> The 2020 RStudio Conference was in San Francisco.
#> The 2021 RStudio Conference was in remote.
```

Introducing `purrr`

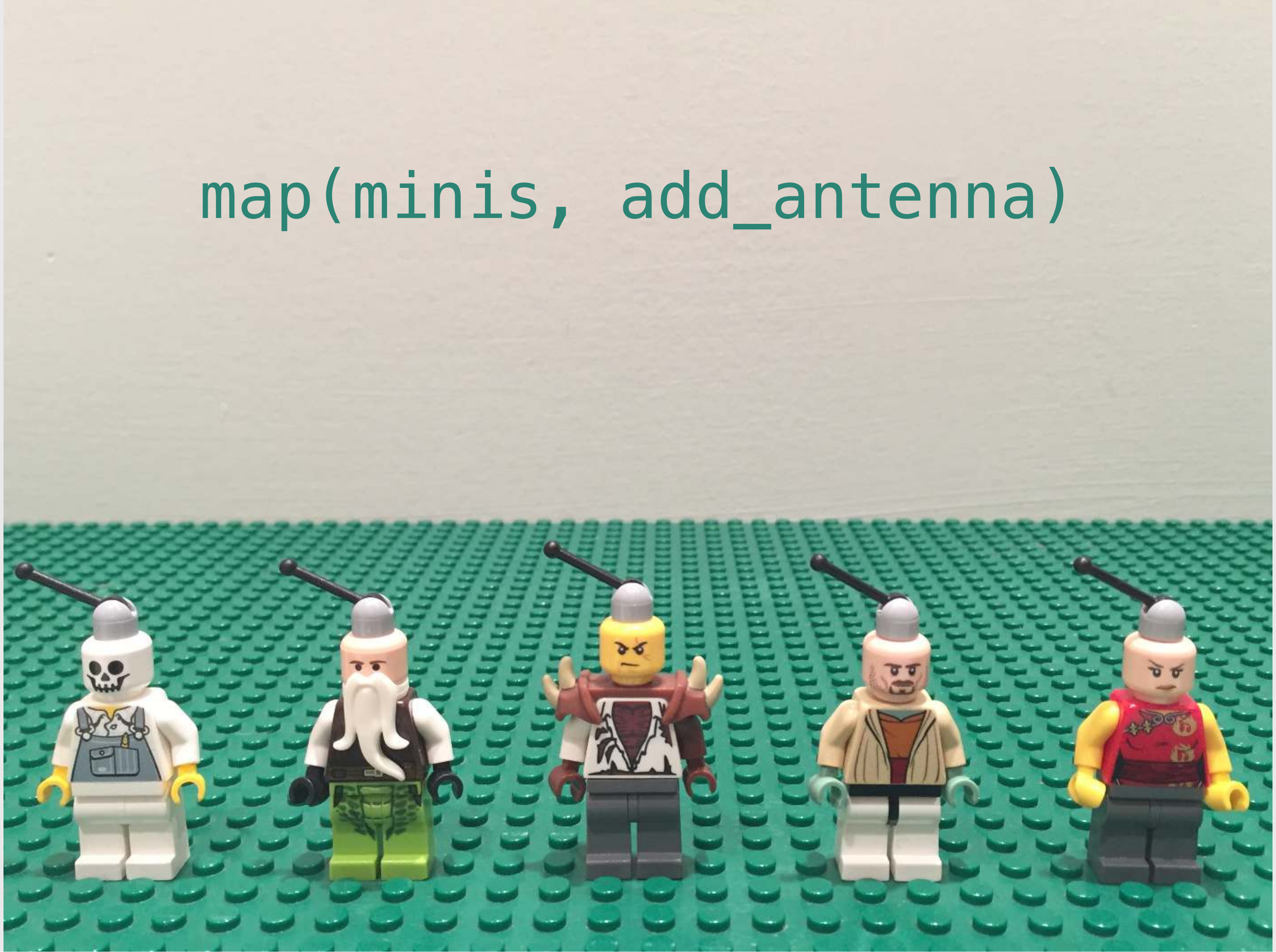
Loaded automatically with `library(tidyverse)`

```r
purrr::map(x, f, ...)
```

for every element of `x` do `f`

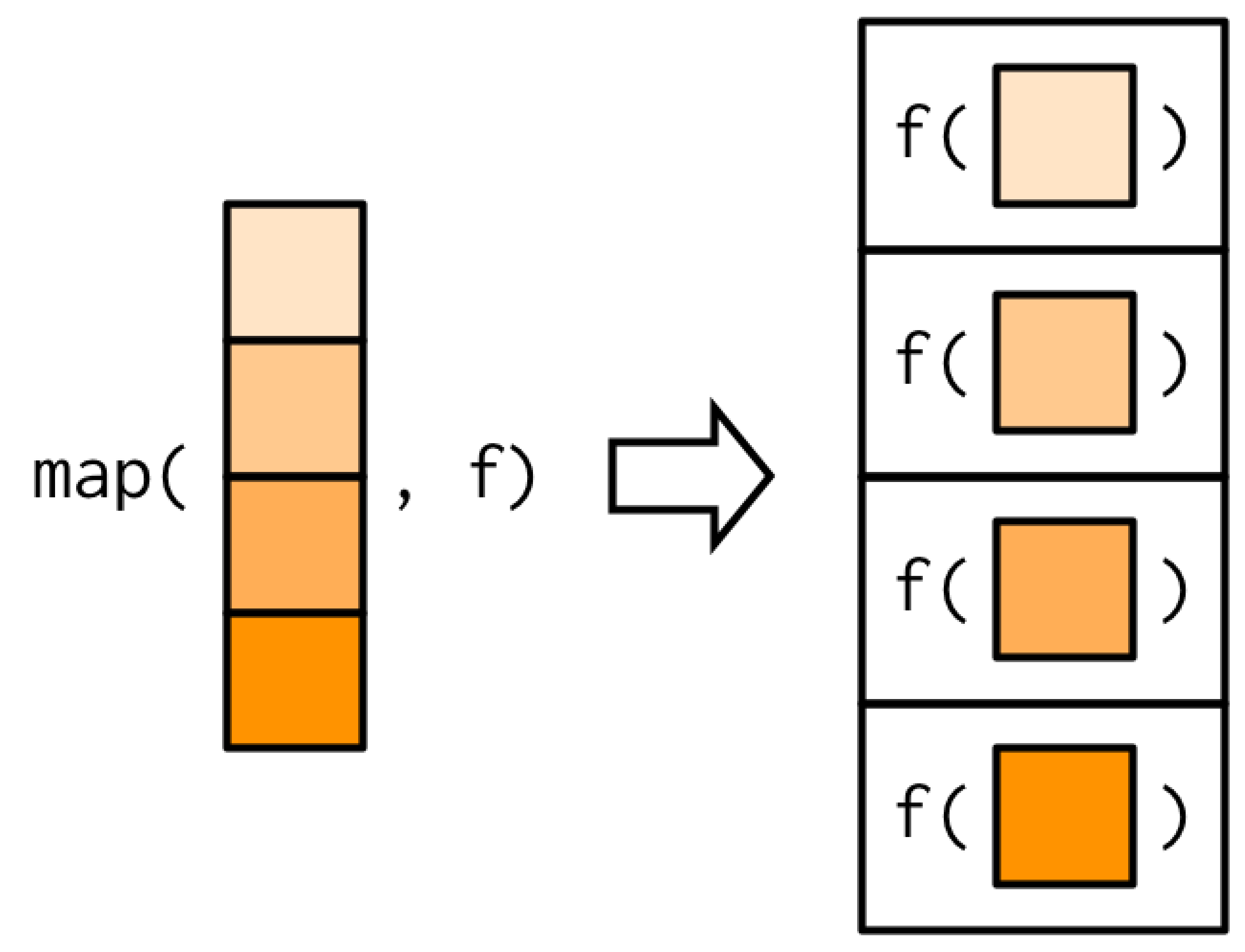
```
x = minis

f = add_antenna
```



```
map(minis, add_antenna)
```

for every element of x do f



map() returns a list

Vector example

```r
addTen <- function(x) {
  return(x + 10)
}
```

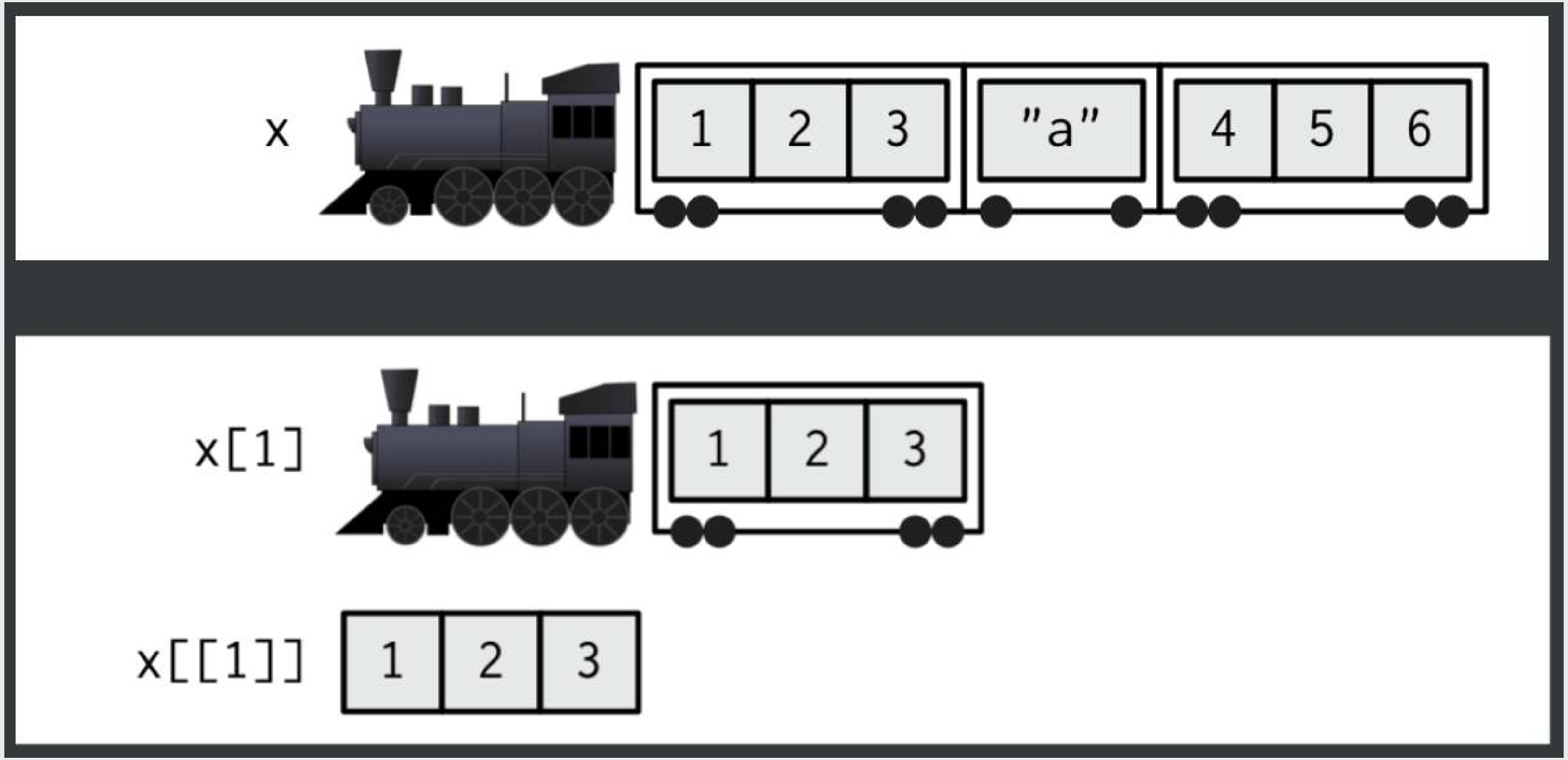
```r
numbers <- c(1, 7, 13)
map(numbers, addTen)
```

```
#> [[1]]
#> [1] 11
#> 
#> [[2]]
#> [1] 17
#> 
#> [[3]]
#> [1] 23
```

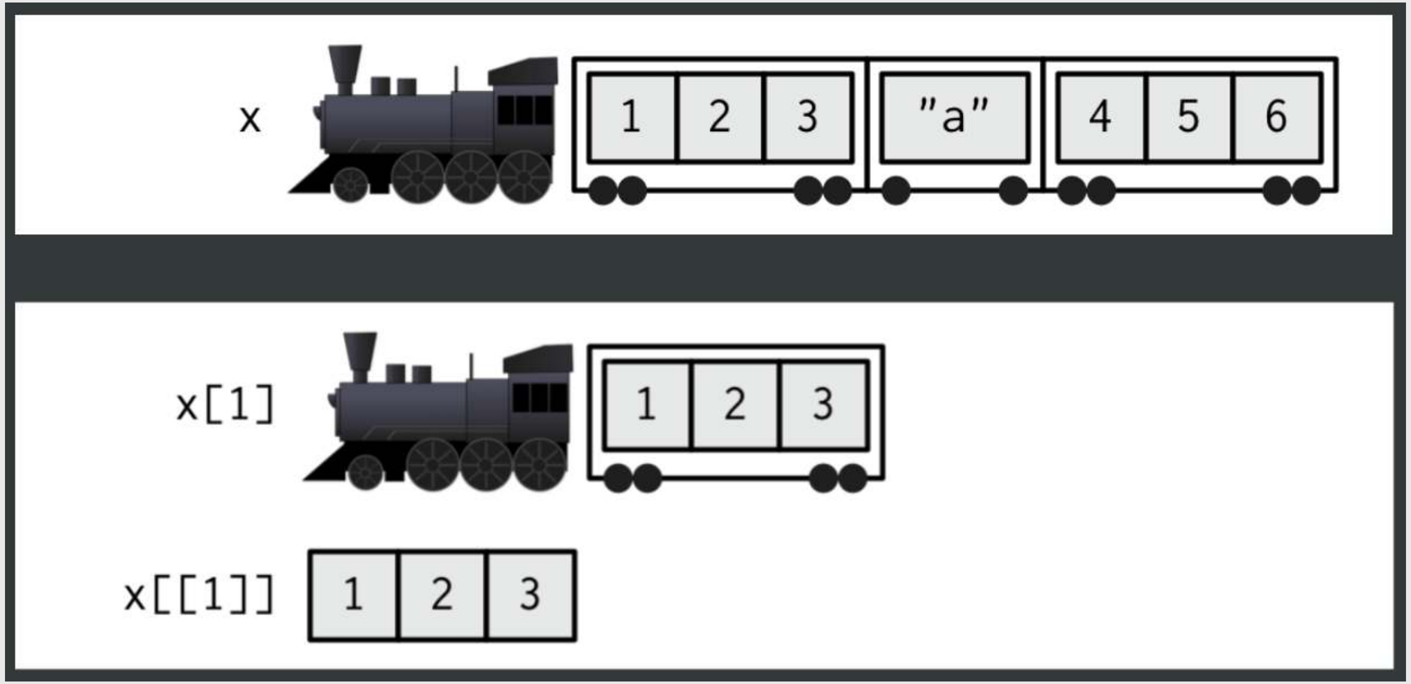
Working with lists feels like...

https://media.giphy.com/media/Bqn8Z7xdPCFy0/giphy.gif

Subsetting lists



source: https://shannonpileggi.github.io/iterating-well-with-purrr/#/subsetting-lists



```
x <- list(c(1, 2, 3), "a", c(4, 5, 6))
```

```
x[1]
```

```
#> [[1]]
#> [1] 1 2 3
```

```
x[[1]]
```

```
#> [1] 1 2 3
```

Example data: sw_people

```r
library(repurrrsive)
```

```r
sw_people
```

```
#> [[1]]
#> [[1]]$name
#> [1] "Luke Skywalker"
#> 
#> [[1]]$height
#> [1] "172"
#> 
#> [[1]]$mass
#> [1] "77"
#> 
#> [[1]]$hair_color
#> [1] "blond"
#> 
#> [[1]]$skin_color
#> [1] "fair"
#>
```

How many films was each Star Wars character in?

```
map(sw_people, f = 🤷)
```

Workflow:

1. Do it for one element.

2. Find the general recipe.

3. Drop into `map()` to do for all.

1. Do it for one element

```
x <- sw_people[[1]]
x
```

```
#> $name
#> [1] "Luke Skywalker"
#>
#> $height
#> [1] "172"
#>
#> $mass
#> [1] "77"
#>
#> $hair_color
#> [1] "blond"
#>
#> $skin_color
#> [1] "fair"
#>
#> $eye_color
#> [1] "blue"
#>
```

View the variables we have to work with:

```
names(x)
```

```
#>  [1] "name"       "height"     "mass"
#>  [6] "eye_color"  "birth_year" "gender"
#> [11] "species"    "vehicles"   "starships
#> [16] "url"
```

Extract the films

```
x$films
```

```
#> [1] "http://swapi.co/api/films/6/" "http:
#> [3] "http://swapi.co/api/films/2/" "http:
#> [5] "http://swapi.co/api/films/7/"
```

1. Do it for one element

How many films was each Star Wars character in?

Character 1:

```
x <- sw_people[[1]]
length(x$films)
```

```
#> [1] 5
```

Character 2:

```
x <- sw_people[[2]]
length(x$films)
```

```
#> [1] 6
```

2. Find the general recipe

How many films was each Star Wars character in?

```
x <- sw_people[[1]]
length(x$films)
```

```
#> [1] 5
```

Recipe:

```
x <- sw_people[[index]]

length(x$films)
```

3. Drop into `map()` to do for all items in list.

Recipe:

```
x <- sw_people[[index]]

length(x$films)
```

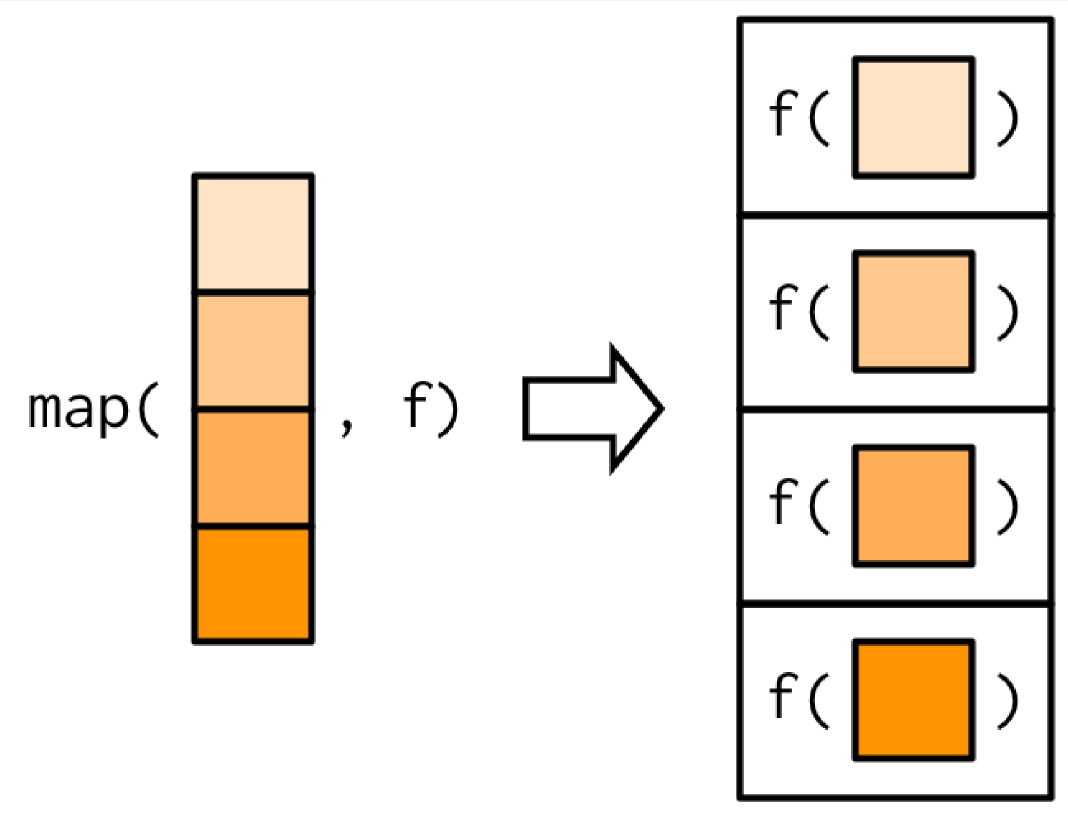
Do for all items in list:

```
get_film_length <- function(x) {
    return(length(x$films))
}

map(sw_people, get_film_length)
```

```
#> [[1]]
#> [1] 5
#>
#> [[2]]
#> [1] 6
#>
#> [[3]]
#> [1] 7
#>
#> [[4]]
```

for every element of x do f



```
get_film_length <- function(x) {
    return(length(x$films))
}

map(sw_people, get_film_length)
```

```
#> [[1]]
#> [1] 5
#>
#> [[2]]
#> [1] 6
#>
#> [[3]]
#> [1] 7
#>
#> [[4]]
#> [1] 4
#>
#> [[5]]
#> [1] 5
#>
```

Simplify it with "anonymous" functions

Version 1: **Custom function**

```
get_film_length <- function(x) {
    return(length(x$films))
}

map(sw_people, get_film_length)
```

```
#> [[1]]
#> [1] 5
#>
#> [[2]]
#> [1] 6
#>
#> [[3]]
#> [1] 7
#>
#> [[4]]
#> [1] 4
#>
```

Version 2: **Anonymous function**

```
map(sw_people, function(x) length(x$films))
```

```
#> [[1]]
#> [1] 5
#>
#> [[2]]
#> [1] 6
#>
#> [[3]]
#> [1] 7
#>
#> [[4]]
#> [1] 4
#>
#> [[5]]
#> [1] 5
#>
#> [[6]]
```

Anonymous functions

Three ways of specifying anonymous functions:

```r
map(sw_people, function(x) length( x$films)) # supported in base R
map(sw_people,        \(x) length( x$films)) # supported R > 4.1.0
map(sw_people,           ~ length(.x$films)) # supported in purrr
```

Quick practice

03:00

- How many `vehicles` does each Star Wars character have?

(use the `sw_people` list)

- How many `titles` does each character in Game of Thrones have?

(use the `got_chars` list)

Type specific map variants

```
map_int(sw_people, \(x) length(x$films))
```

```
#>  [1] 5 6 7 4 5 3 3 1 1 6 3 2 5 4 1 3 3 1 5 5 3 1 1 2 1 2 1 1 1 1 1 3 1 2 1 1 1 2
#> [39] 1 1 2 1 1 3 1 1 1 3 3 3 2 2 2 1 3 2 1 1 1 2 2 1 1 2 2 1 1 1 1 1 1 1 2 1 1 2
#> [77] 1 1 2 2 1 1 1 1 1 1 3
```

`map_lgl()`: Returns a *logical* vector

`map_int()`: Returns a *integer* vector

`map_dbl()`: Returns a *double* vector

`map_chr()`: Returns a *character* vector

Quick practice

03:00

Replace `map()` with type-specific `map()`.

```r
# What's each character's name?
map(got_chars, \(x) x$name)
map(sw_people, \(x) x$name)

# What color is each SW character's hair?
map(sw_people, \(x) x$hair_color)

# Is the GoT character alive?
map(got_chars, \(x) x$alive)

# Is the SW character female?
map(sw_people, \(x) x$gender == "female")

# How heavy is each SW character?
map(sw_people, \(x) x$mass)
```

How many films was each Star Wars character in?

```
map(sw_people, \(x) length(x$films))
```

```
#> [[1]]
#> [1] 5
#> 
#> [[2]]
#> [1] 6
#> 
#> [[3]]
#> [1] 7
#> 
#> [[4]]
#> [1] 4
#> 
#> [[5]]
#> [1] 5
#> 
#> [[6]]
#> [1] 3
#> 
#> [[7]]
```

Wait, which character?

Use a tibble to get the character name as well!

Returns a list of data frames:

```r
map(sw_people, \(x) tibble(
    name = x$name,
    n_vehicles = length(x$films)
  )
)
```

```
#> [[1]]
#> # A tibble: 1 × 2
#>   name           n_vehicles
#>   <chr>               <int>
#> 1 Luke Skywalker          5
#> 
#> [[2]]
#> # A tibble: 1 × 2
#>   name  n_vehicles
#>   <chr>      <int>
#> 1 C-3PO          6
#>
```

Use `map_df()` to merge the data frames

```r
map_df(sw_people, \(x) tibble(
    name = x$name,
    n_vehicles = length(x$films)
  )
)
```

```
#> # A tibble: 87 × 2
#>    name               n_vehicles
#>    <chr>                   <int>
#>  1 Luke Skywalker              5
#>  2 C-3PO                       6
#>  3 R2-D2                       7
#>  4 Darth Vader                 4
#>  5 Leia Organa                 5
#>  6 Owen Lars                   3
#>  7 Beru Whitesun lars          3
#>  8 R5-D4                       1
#>  9 Biggs Darklighter           1
```

Your turn

15:00

Try to answer these questions:

1. Which SW film has the most characters? (use `sw_films`)
2. Which SW species has the highest average lifespan? (use `sw_species`)
3. Which GoT character(s) have been played by multiple actors? (use `got_chars`)

Sometimes you really need do something on each row

Use a `for` loop to iterate across each row in a data frame:

```r
for (i in 1:nrow(df)) {
  row <- df[i,]
  # Do stuff with row
}
```

Example: tagging a new daily covid case record

```r
covid_dc <- read_csv(here::here('data', 'us_covid.csv')) %>%
  filter(state == 'District of Columbia') %>%
  select(-state)

head(covid_dc)
```

```
#> # A tibble: 6 × 6
#>   date         day cases_daily deaths_daily cases_total deaths_total
#>   <date>     <dbl>       <dbl>        <dbl>       <dbl>        <dbl>
#> 1 2020-01-23     1           0            0           0            0
#> 2 2020-01-24     2           0            0           0            0
#> 3 2020-01-25     3           0            0           0            0
#> 4 2020-01-26     4           0            0           0            0
#> 5 2020-01-27     5           0            0           0            0
#> 6 2020-01-28     6           0            0           0            0
```

Example: tagging a new daily covid case record

Initialize new column

```r
covid_dc$new_record <- FALSE

glimpse(covid_dc)
```

```
#> Rows: 403
#> Columns: 7
#> $ date         <date> 2020-01-23, 2020-01
#> $ day          <dbl> 1, 2, 3, 4, 5, 6, 7,
#> $ cases_daily  <dbl> 0, 0, 0, 0, 0, 0, 0,
#> $ deaths_daily <dbl> 0, 0, 0, 0, 0, 0, 0,
#> $ cases_total  <dbl> 0, 0, 0, 0, 0, 0, 0,
#> $ deaths_total <dbl> 0, 0, 0, 0, 0, 0, 0,
#> $ new_record   <lgl> FALSE, FALSE, FALSE,
```

Now loop through each row and check if a new record is met

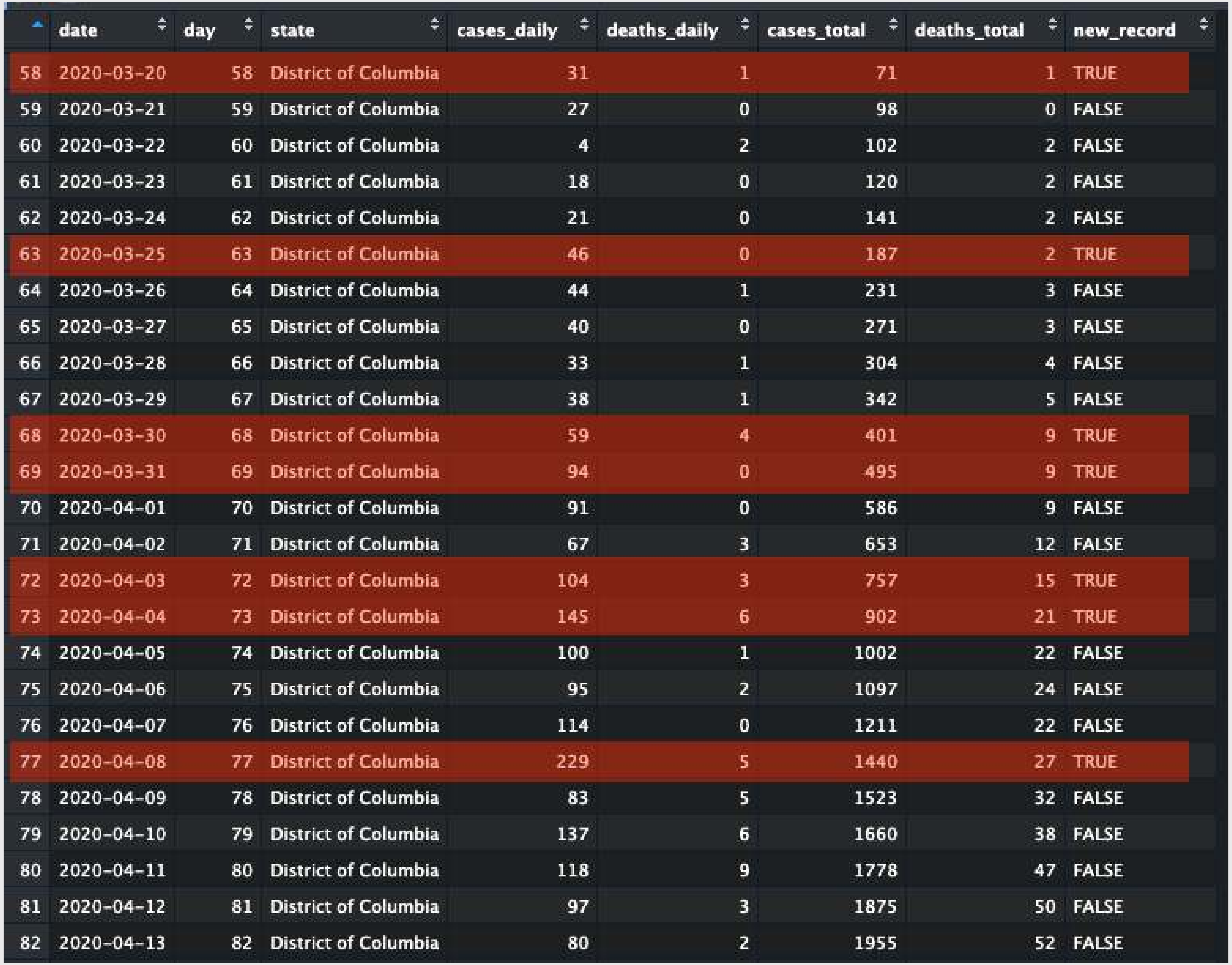
```r
record <- 0
for (i in 1:nrow(covid_dc)) {

  # Get the nubmer of cases on row i
  num_cases <- covid_dc[i,]$cases_daily

  # Check if new record is met
  if (num_cases > record) {

    # Update new record in covid_dc
    covid_dc[i, ]$new_record <- TRUE

    # Update new record
    record <- num_cases
  }
}
```



	date	day	state	cases_daily	deaths_daily	cases_total	deaths_total	new_record
58	2020-03-20	58	District of Columbia	31	1	71	1	TRUE
59	2020-03-21	59	District of Columbia	27	0	98	0	FALSE
60	2020-03-22	60	District of Columbia	4	2	102	2	FALSE
61	2020-03-23	61	District of Columbia	18	0	120	2	FALSE
62	2020-03-24	62	District of Columbia	21	0	141	2	FALSE
63	2020-03-25	63	District of Columbia	46	0	187	2	TRUE
64	2020-03-26	64	District of Columbia	44	1	231	3	FALSE
65	2020-03-27	65	District of Columbia	40	0	271	3	FALSE
66	2020-03-28	66	District of Columbia	33	1	304	4	FALSE
67	2020-03-29	67	District of Columbia	38	1	342	5	FALSE
68	2020-03-30	68	District of Columbia	59	4	401	9	TRUE
69	2020-03-31	69	District of Columbia	94	0	495	9	TRUE
70	2020-04-01	70	District of Columbia	91	0	586	9	FALSE
71	2020-04-02	71	District of Columbia	67	3	653	12	FALSE
72	2020-04-03	72	District of Columbia	104	3	757	15	TRUE
73	2020-04-04	73	District of Columbia	145	6	902	21	TRUE
74	2020-04-05	74	District of Columbia	100	1	1002	22	FALSE
75	2020-04-06	75	District of Columbia	95	2	1097	24	FALSE
76	2020-04-07	76	District of Columbia	114	0	1211	22	FALSE
77	2020-04-08	77	District of Columbia	229	5	1440	27	TRUE
78	2020-04-09	78	District of Columbia	83	5	1523	32	FALSE
79	2020-04-10	79	District of Columbia	137	6	1660	38	FALSE
80	2020-04-11	80	District of Columbia	118	9	1778	47	FALSE
81	2020-04-12	81	District of Columbia	97	3	1875	50	FALSE
82	2020-04-13	82	District of Columbia	80	2	1955	52	FALSE

Preview HW 11