Intro to {purrr}

And a more explicit description of functional programming

Daniel Anderson
Week 3, Class 1

UNIVERSITY OF OREGON

Agenda

- Thinking more about functional programming
    - A small example
- Introduce you to {purrr} and contrast it with the base functions we learned last week

Learning objectives

- Understand how `purrr::map` relates to `lapply` and `for` loops
- Understand the six basic variants of `purrr::map`, when they should be used, and when they will fail
- Understand what functional programming is, and how `{purrr}` can help facilitate the process

Functional Programming

What is FP?

> decomposing a big problem into smaller pieces, then solving each piece with a function or combination of functions

- Adv-R

Example

Calculate top mpg manufactures

- First, we'll subset the data to 4 cylinders. This will be the dataset we'll solve the problem on:

```
library(tidyverse)
four_cyl <- filter(mpg, cyl == 4)
```

- Next, we'll filter for cases where the city miles per gallon is in the top 10% (i.e.,greater than or equal to the 90th percentile).

```
ninety <- four_cyl %>%
    filter(cty >= quantile(cty, probs = 0.9))
ninety
```

```
## # A tibble: 10 x 11
##    manufacturer model displ  year   cyl trans drv     cty   hwy fl    class
##    <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
##  1 honda        civic   1.6  1999     4 manu… f        28    33 r     subc…
##  2 honda        civic   1.6  1999     4 manu… f        25    32 r     subc…
##  3 honda        civic   1.8  2008     4 manu… f        26    34 r     subc…
##  4 honda        civic   1.8  2008     4 auto… f        25    36 r     subc…
##  5 toyota       coro…   1.8  1999     4 manu… f        26    35 r     comp…
##  6 toyota       coro…   1.8  2008     4 manu… f        28    37 r     comp…
##  7 toyota       coro…   1.8  2008     4 auto… f        26    35 r     comp…
##  8 volkswagen   jetta   1.9  1999     4 manu… f        33    44 d     comp…
##  9 volkswagen   new …   1.9  1999     4 manu… f        35    44 d     subc…
## 10 volkswagen   new …   1.9  1999     4 auto… f        29    41 d     subc…
```

- Now count the unique occurrences for each manufacturer, manufacturer/model, and class

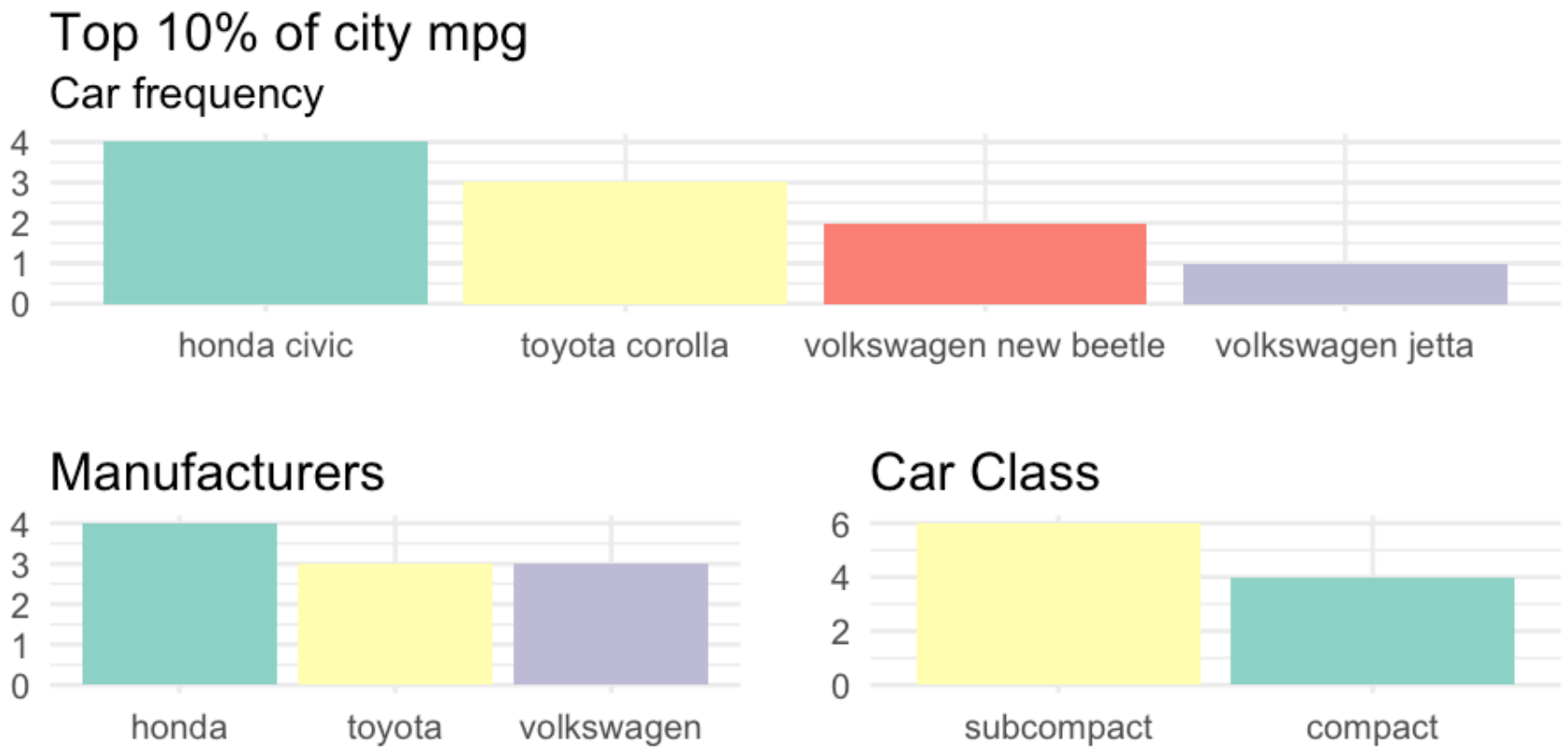
```
count_manufacturer <- count(ninety, manufacturer)
count_model <- count(ninety, manufacturer, model)
count_class <- count(ninety, class)
```

- Produce a plot for each

```
plot_manufacturer <-
    ggplot(count_manufacturer, aes(fct_reorder(manufacturer, -n), n)) +
        geom_col(aes(fill = manufacturer)) +
        scale_fill_brewer(palette = "Set3") +
        labs(title = "Manufacturers",
             x = "",
             y = "") +
        guides(fill = "none")

plot_car <- count_model %>%
    unite(car, manufacturer, model, sep = " ") %>%
    ggplot(aes(fct_reorder(car, -n), n)) +
        geom_col(aes(fill = car)) +
        scale_fill_brewer(palette = "Set3") +
        labs(title = "Top 10% of city mpg",
             subtitle = "Car frequency",
             x = "",
             y = "") +
        guides(fill = "none")

plot_class <-
    ggplot(count_class, aes(fct_reorder(class, -n), n)) +
        geom_col(aes(fill = class)) +
        scale_fill_brewer(palette = "Set3") +
        labs(title = "Car Class",
```

Assemble the plots

```
library(patchwork)
plot_car / (plot_manufacturer + plot_class)
```



Functional Programming Version

At least in spirit

Filter all

```
by_cyl <- split(mpg, mpg$cyl)
top_10 <- lapply(by_cyl, function(x) {
    filter(x, cty >= quantile(cty, probs = 0.9))
})
str(top_10)
```

```
## List of 4
##  $ 4:Classes 'tbl_df', 'tbl' and 'data.frame':    10 obs. of  11 variables:
##   ..$ manufacturer: chr [1:10] "honda" "honda" "honda" "honda" ...
##   ..$ model       : chr [1:10] "civic" "civic" "civic" "civic" ...
##   ..$ displ       : num [1:10] 1.6 1.6 1.8 1.8 1.8 1.8 1.8 1.9 1.9 1.9
##   ..$ year        : int [1:10] 1999 1999 2008 2008 1999 2008 2008 1999 1999 199
##   ..$ cyl         : int [1:10] 4 4 4 4 4 4 4 4 4 4
##   ..$ trans       : chr [1:10] "manual(m5)" "manual(m5)" "manual(m5)" "auto(l5)
##   ..$ drv         : chr [1:10] "f" "f" "f" "f" ...
##   ..$ cty         : int [1:10] 28 25 26 25 26 28 26 33 35 29
##   ..$ hwy         : int [1:10] 33 32 34 36 35 37 35 44 44 41
##   ..$ fl          : chr [1:10] "r" "r" "r" "r" ...
##   ..$ class       : chr [1:10] "subcompact" "subcompact" "subcompact" "subcompa
##  $ 5:Classes 'tbl_df', 'tbl' and 'data.frame':    2 obs. of  11 variables:
##   ..$ manufacturer: chr [1:2] "volkswagen" "volkswagen"
##   ..$ model       : chr [1:2] "jetta" "jetta"
##   ..$ displ       : num [1:2] 2.5 2.5
##   ..$ year        : int [1:2] 2008 2008
```

All counts

```
counts <- lapply(top_10, function(x) {
    count_manufacturer <- count(x, manufacturer)
    count_class <- count(x, class)

    count_model <- count(x, manufacturer, model) %>%
        unite(car, manufacturer, model, sep = " ")


    return(list(mfr = count_manufacturer,
                car = count_model,
                class = count_class))
})
str(counts)
```

```
## List of 4
##  $ 4:List of 3
##   ..$ mfr  :Classes 'tbl_df', 'tbl' and 'data.frame':    3 obs. of  2 variables
##   .. ..$ manufacturer: chr [1:3] "honda" "toyota" "volkswagen"
##   .. ..$ n           : int [1:3] 4 3 3
##   ..$ car  :Classes 'tbl_df', 'tbl' and 'data.frame':    4 obs. of  2 variables
##   .. ..$ car: chr [1:4] "honda civic" "toyota corolla" "volkswagen jetta" "volk
##   .. ..$ n  : int [1:4] 4 3 1 2
##   ..$ class:Classes 'tbl_df', 'tbl' and 'data.frame':    2 obs. of  2 variables
##   .. ..$ class: chr [1:2] "compact" "subcompact"
```

Plots

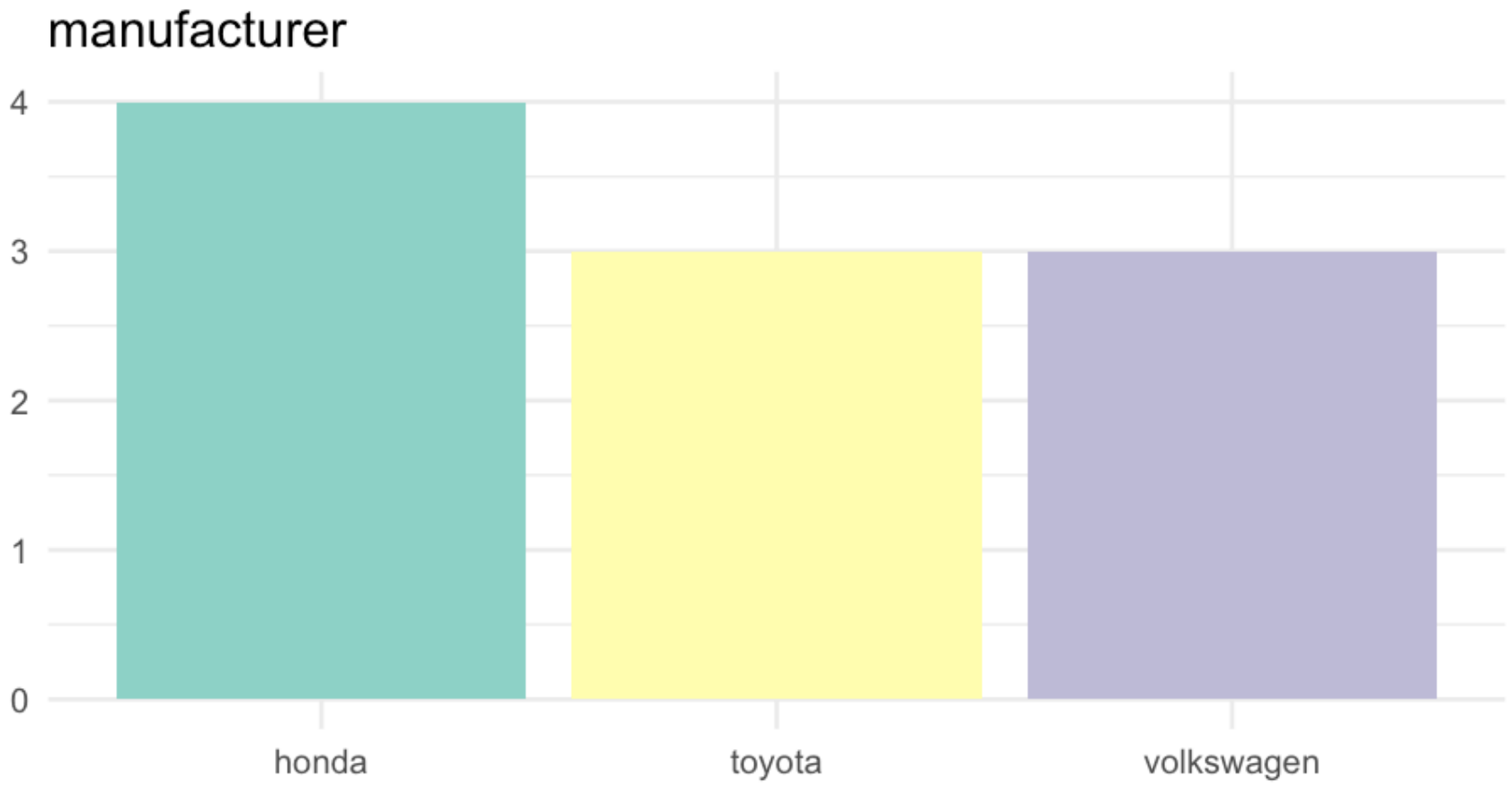
Let's write a couple functions

(I recognize we haven't talked about functions yet, but as a preview...)

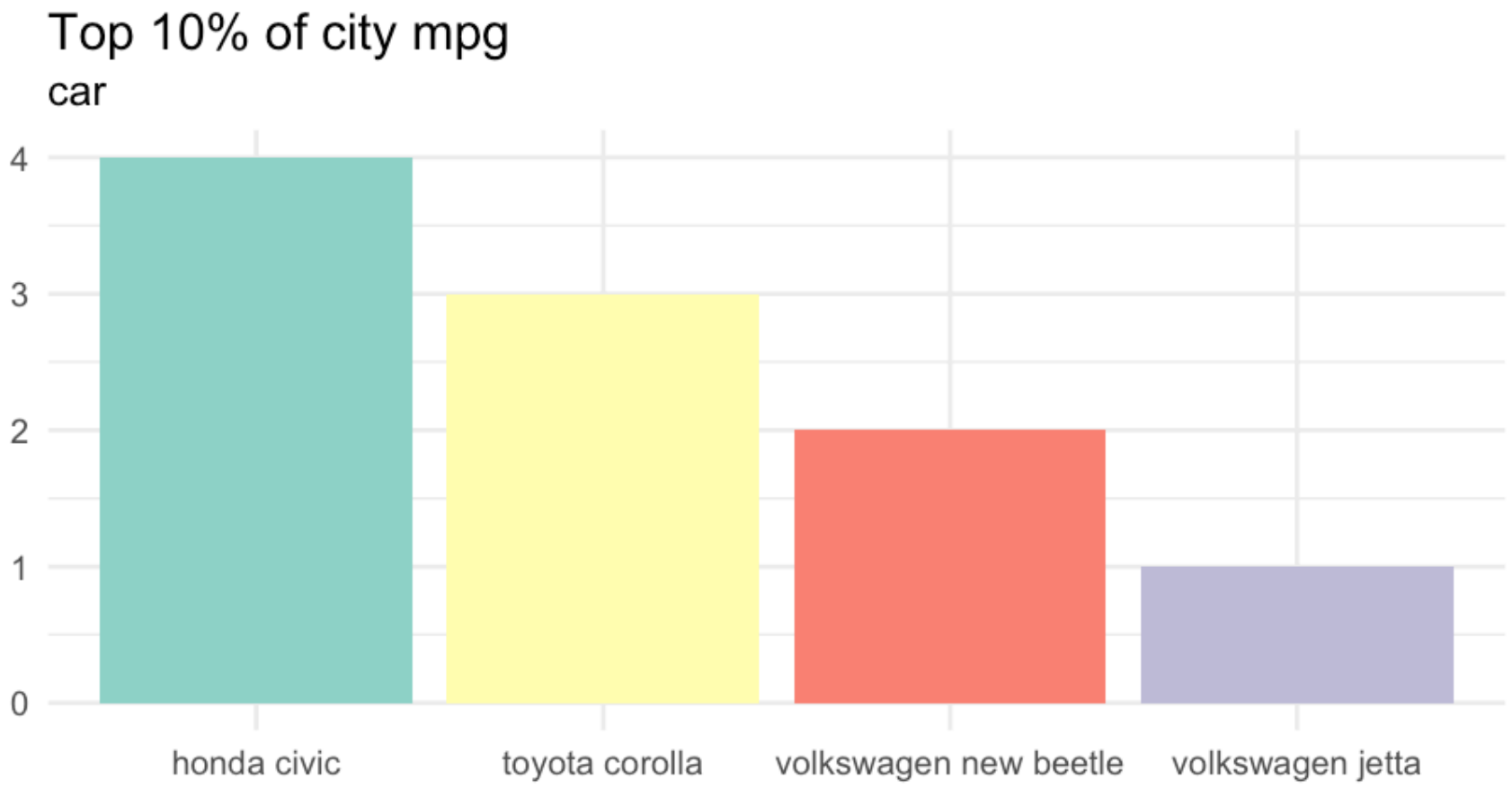
```
indiv_plot <- function(d, var) {
    p <- ggplot(d, aes_string(fct_reorder(d[[var]], -d[["n"]]),
                              d[["n"]])) +
        geom_col(aes_string(fill = var)) +
        scale_fill_brewer(palette = "Set3") +
        labs(title = var,
             x = "",
             y = "") +
        guides(fill = "none")

    if(var == "car") {
        p <- p + labs(title = "Top 10% of city mpg",
                      subtitle = var)
    }
    return(p)
}
```

Test it

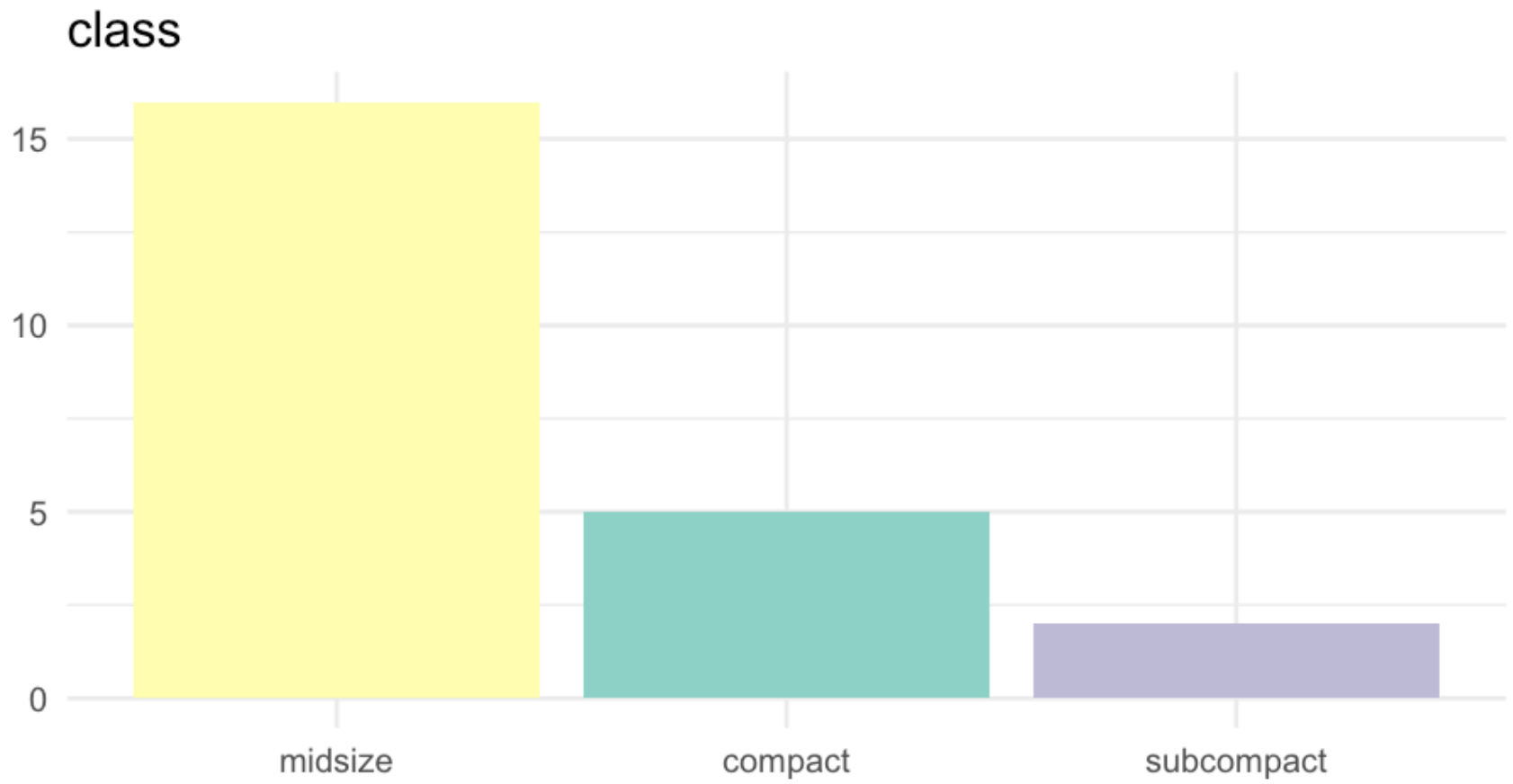
```
indiv_plot(counts[["4"]]$mfr, "manufacturer")
```



```
indiv_plot(counts[["4"]]$car, "car")
```



```
indiv_plot(counts[["6"]]$class, "class")
```

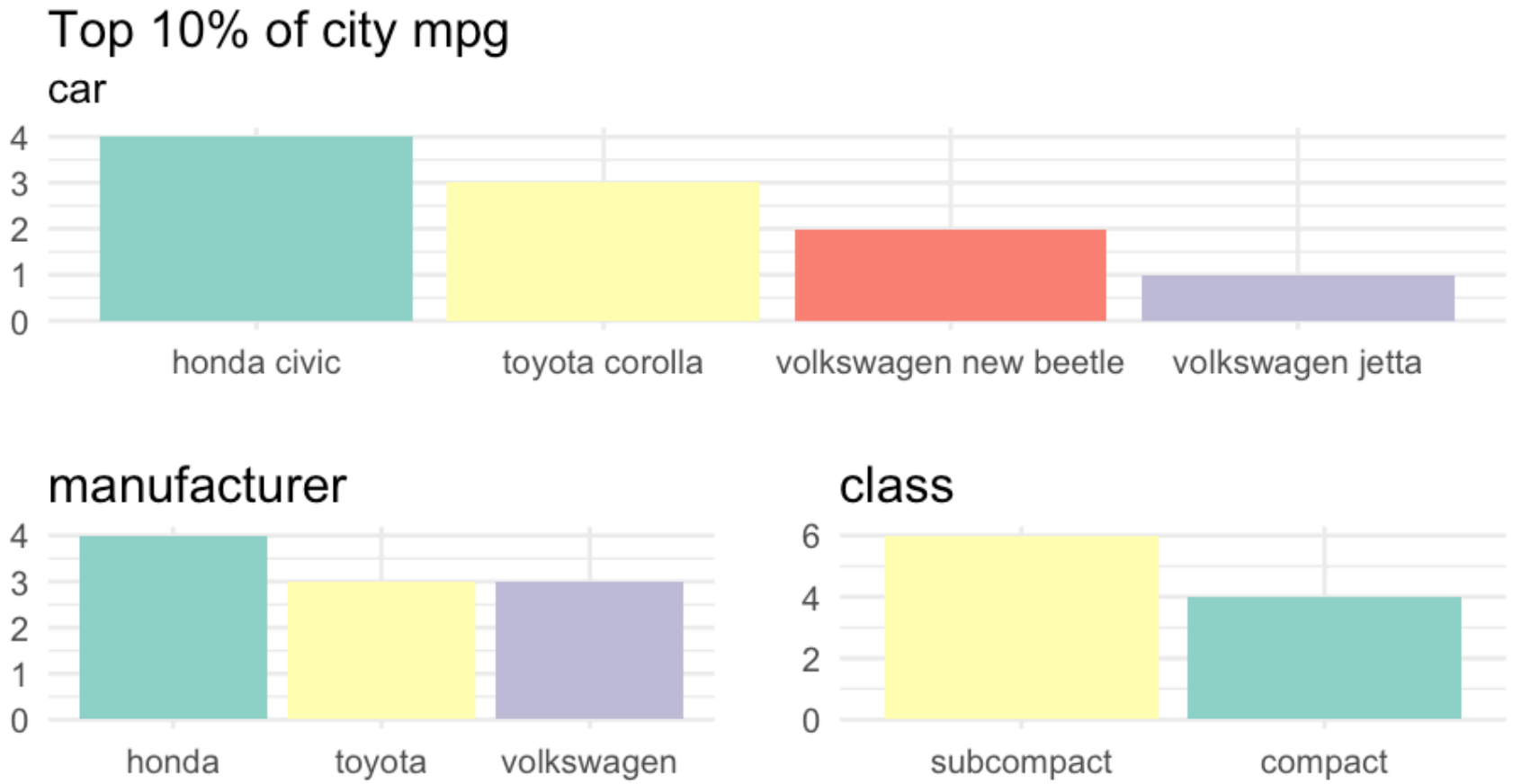


Compile plots function

```
full_plot <- function(l) {
    indiv_plot(l[["car"]], "car") / (
      indiv_plot(l[["mfr"]], "manufacturer") +
      indiv_plot(l[["class"]], "class")
    )
}
```

Test it

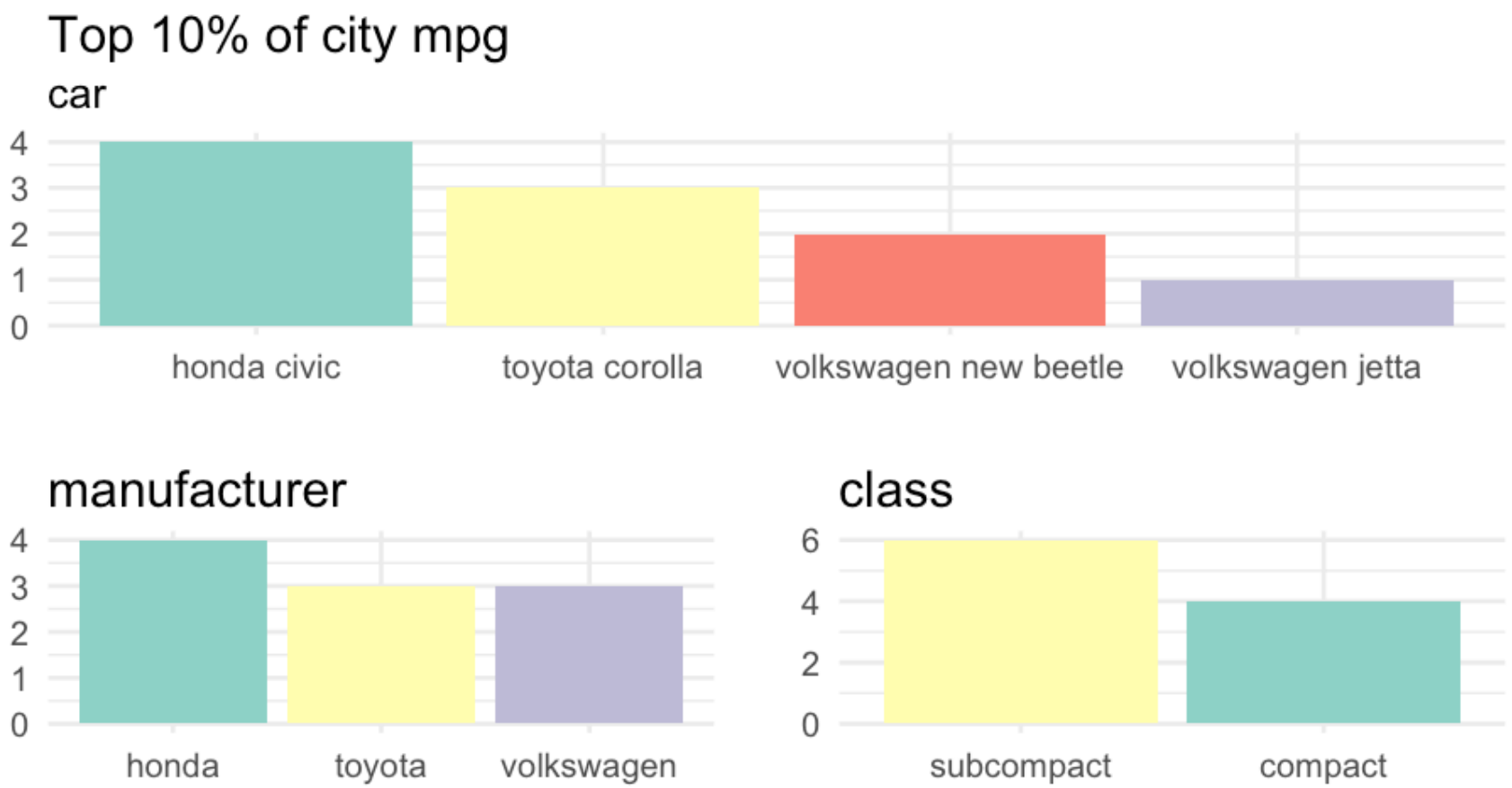
```
full_plot(counts[[1]])
```



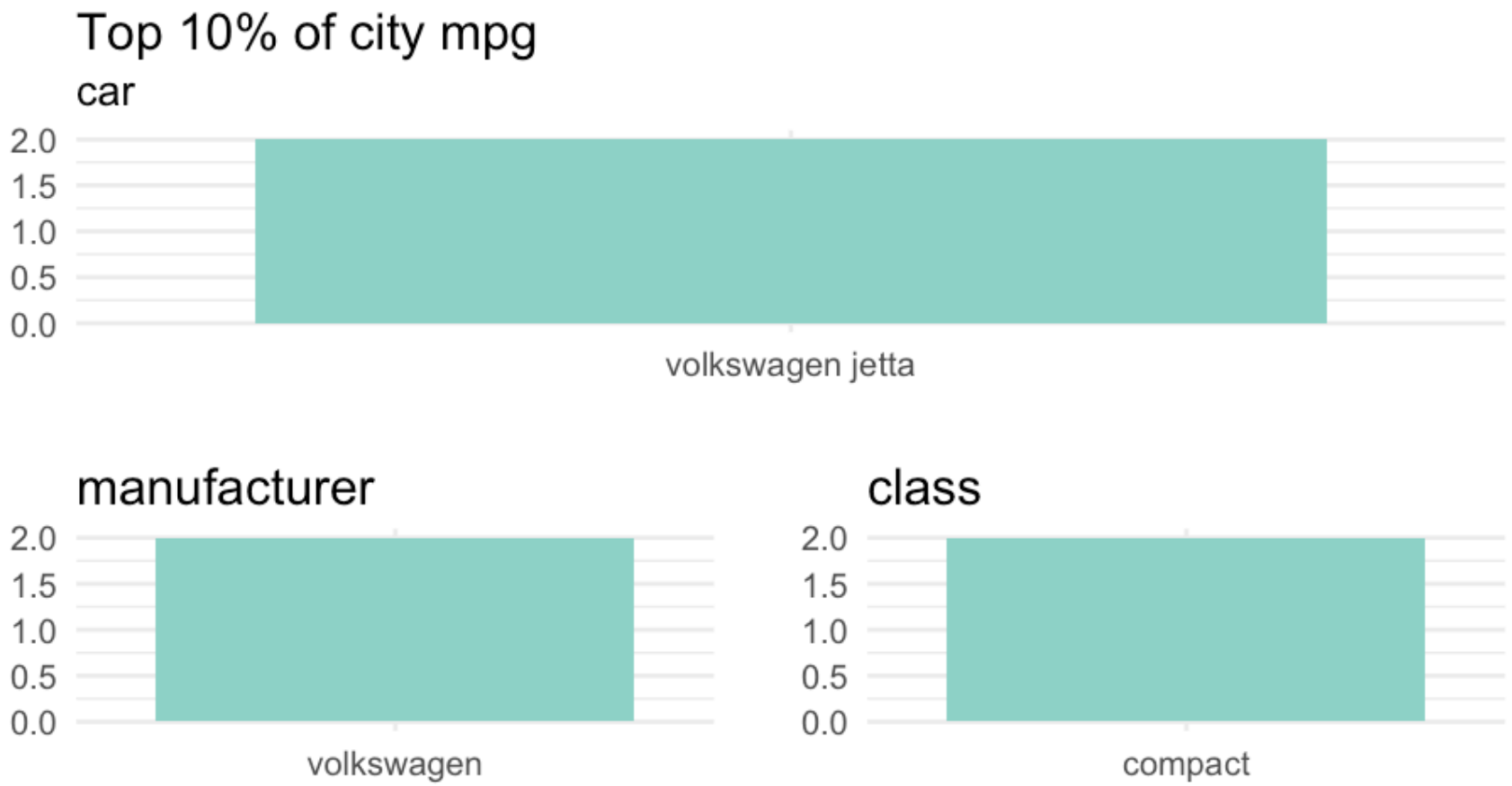
Finish up

```
plots <- lapply(counts, full_plot)
```

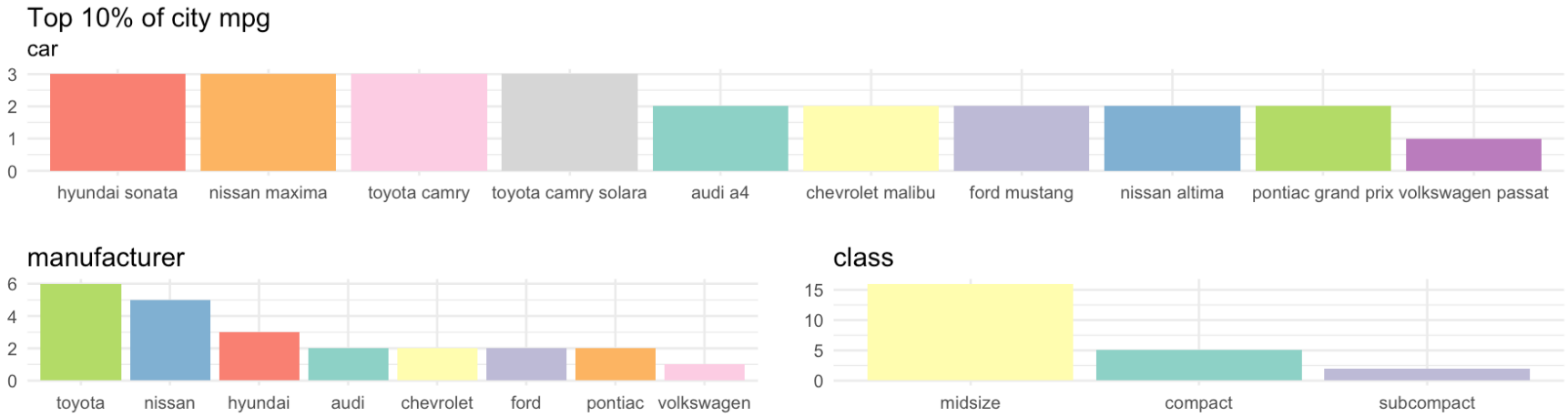
```
plots[[1]]
```


Top 10% of city mpg
car
4
3
2
1
0
honda civic
toyota corolla
volkswagen new beetle
volkswagen jetta
manufacturer
4
3
2
1
0
honda
toyota
volkswagen
class
6
4
2
0
subcompact
compact

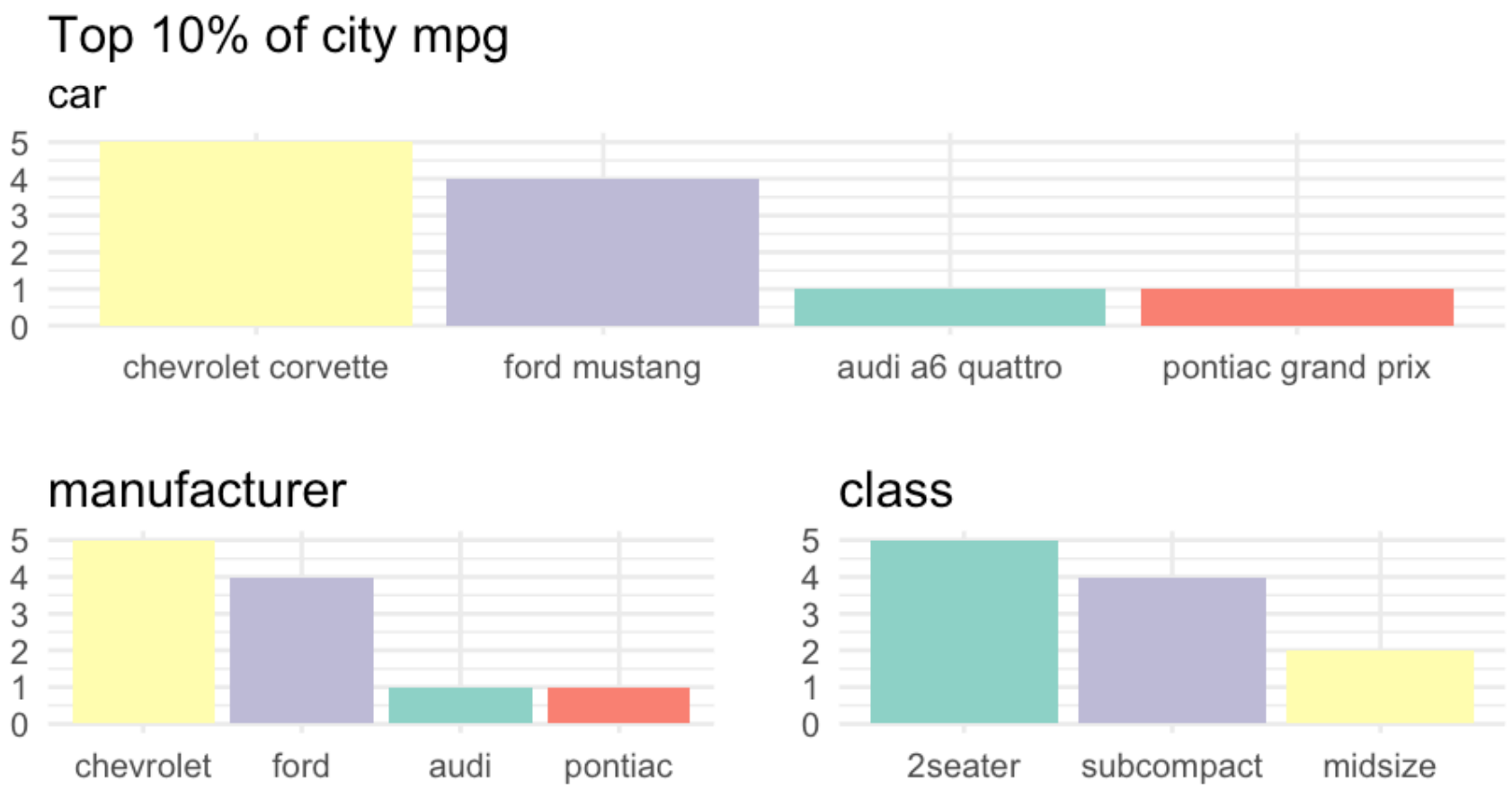
```
plots[[2]]
```


Top 10% of city mpg
car
2.0
1.5
1.0
0.5
0.0
volkswagen jetta
manufacturer
2.0
1.5
1.0
0.5
0.0
volkswagen
class
2.0
1.5
1.0
0.5
0.0
compact

```
plots[[3]]
```


Top 10% of city mpg
car
3
2
1
0
hyundai sonata
nissan maxima
toyota camry
toyota camry solara
audi a4
chevrolet malibu
ford mustang
nissan altima
pontiac grand prix
volkswagen passat
manufacturer
6
4
2
0
toyota
nissan
hyundai
audi
chevrolet
ford
pontiac
volkswagen
class
15
10
5
0
midsize
compact
subcompact

```
plots[[4]]
```


Top 10% of city mpg
car
5
4
3
2
1
0
chevrolet corvette
ford mustang
audi a6 quattro
pontiac grand prix
manufacturer
5
4
3
2
1
0
chevrolet
ford
audi
pontiac
class
5
4
3
2
1
0
2seater
subcompact
midsize

{purrr}

functionals

> a function that takes a function as input and returns a vector as output.

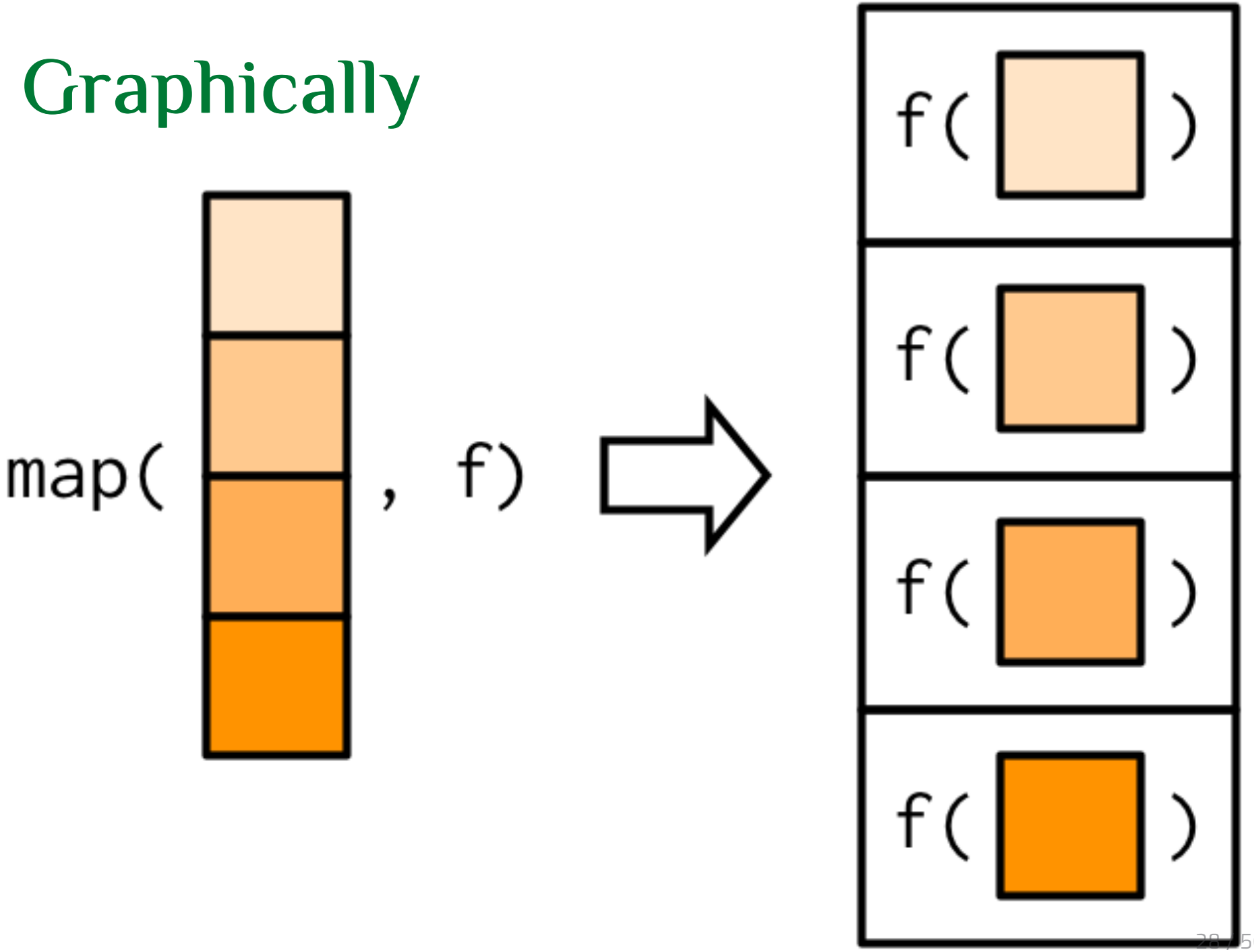
- What does this mean?

purrr::map

```
library(purrr) # loaded automatically by tidyverse
map(1:3, rnorm)
```

```
## [[1]]
## [1] 0.5941747
##
## [[2]]
## [1] 0.5532272 0.2522552
##
## [[3]]
## [1]  0.6873934 -0.1936925 -0.2495917
```

Graphically



Comparison to `base::lapply`

lapply

```
lapply(1:3, rnorm)
```

```
## [[1]]
## [1] -0.7838051
##
## [[2]]
## [1] -1.7543676  0.0689319
##
## [[3]]
## [1]  0.07790071 -0.65711040 -0.1227614
```

map

```
map(1:3, rnorm)
```

```
## [[1]]
## [1] -0.9771206
##
## [[2]]
## [1] -0.4845289  1.9154039
##
## [[3]]
## [1] -1.36694488  0.04390757 -0.0705060
```

side note: What exactly is going on here?

> The base equivalent to map() is lapply(). The only difference is that lapply() does not support the helpers that you'll learn about (next), so if you're only using map() from purrr, you can skip the additional dependency and use lapply() directly.

- Adv R

Equivalents

The following are equivalent

```
map(mtcars, function(x) length(unique(x)))

lapply(mtcars, function(x) length(unique(x)))
```

Equivalents

The following are equivalent

```
map(mtcars, function(x) length(unique(x)))

lapply(mtcars, function(x) length(unique(x)))
```

{purrr} also allows you to specify anonymous functions more succinctly using the formula interface

```
map(mtcars, ~length(unique(.x)))
```

```
## $mpg
## [1] 25
##
## $cyl
## [1] 3
##
## $disp
## [1] 27
##
## $hp
## [1] 22
```

Extracting elements

```
l <- list(
  list(-1, x = 1, y = c(2), z = "a"),
  list(-2, x = 4, y = c(5, 6), z = "b"),
  list(-3, x = 8, y = c(9, 10, 11))
)
```

Extracting elements

```
l <- list(
  list(-1, x = 1, y = c(2), z = "a"),
  list(-2, x = 4, y = c(5, 6), z = "b"),
  list(-3, x = 8, y = c(9, 10, 11))
)
```

Extract second element from each

```
map(l, 2)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 4
##
## [[3]]
## [1] 8
```

Doesn't work for `lapply`

```
lapply(l, 2)
```

```
## Error in match.fun(FUN): '2' is not a function, character or symbol
```

Doesn't work for `lapply`

```
lapply(l, 2)
```

```
## Error in match.fun(FUN): '2' is not a function, character or symbol
```

Instead, you have to apply an anoymous function

```
lapply(l, function(x) x[[2]])
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 4
##
## [[3]]
## [1] 8
```

Alternatively the following is also the same

```
lapply(l, `[[`, 2)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 4
##
## [[3]]
## [1] 8
```

Extract by name

```
map(l, "y")
```

```
## [[1]]
## [1] 2
##
## [[2]]
## [1] 5 6
##
## [[3]]
## [1]  9 10 11
```

Multiple arguments

```
map(l, list("y", 1))
```

```
## [[1]]
## [1] 2
##
## [[2]]
## [1] 5
##
## [[3]]
## [1] 9
```

{purrr} variants

Return a vector

- `map_dbl`
- `map_int`
- `map_char`
- `map_lgl`

```
str(l)
```

```
## List of 3
##  $ :List of 4
##   ..$  : num -1
##   ..$ x: num 1
##   ..$ y: num 2
##   ..$ z: chr "a"
##  $ :List of 4
##   ..$  : num -2
##   ..$ x: num 4
##   ..$ y: num [1:2] 5 6
##   ..$ z: chr "b"
##  $ :List of 3
##   ..$  : num -3
##   ..$ x: num 8
##   ..$ y: num [1:3] 9 10 11
```

```
str(l)
```

```
## List of 3
##  $ :List of 4
##   ..$  : num -1
##   ..$ x: num 1
##   ..$ y: num 2
##   ..$ z: chr "a"
##  $ :List of 4
##   ..$  : num -2
##   ..$ x: num 4
##   ..$ y: num [1:2] 5 6
##   ..$ z: chr "b"
##  $ :List of 3
##   ..$  : num -3
##   ..$ x: num 8
##   ..$ y: num [1:3] 9 10 11
```

```
map_dbl(l, "x")
```

```
## [1] 1 4 8
```

```
map_dbl(l, 1)
```

```
## [1] -1 -2 -3
```

Type match

- Coercion will occur if you request a different type

```
map_chr(l, "x")
```

```
## [1] "1.000000" "4.000000" "8.000000"
```

Type match

- Coercion will occur if you request a different type

```
map_chr(l, "x")
```

```
## [1] "1.000000" "4.000000" "8.000000"
```

- You'll get an error if element doesn't exist

```
map_chr(l, "z")
```

```
## Error: Result 3 must be a single string, not NULL of length 0
## Backtrace:
## [90m     [39m
## [90m  1. [39m├─rmarkdown::render(...)
## [90m  2. [39m| └─knitr::knit(...)
## [90m  3. [39m|   └─knitr:::process_file(text, output)
## [90m  4. [39m|     ├─base::withCallingHandlers(...)
## [90m  5. [39m|     ├─knitr:::process_group(group)
## [90m  6. [39m|     └─knitr:::process_group.block(group)
## [90m  7. [39m|       └─knitr:::call_block(x)
## [90m  8. [39m|         └─knitr:::block_exec(params)
```

- Unless you set a default value

```
map_chr(l, "z", .default = NA_character_)
```

```
## [1] "a" "b" NA
```

Quick note on missing values

- In the prior case, specifying `NA` would work, instead of `NA_character_`

Quick note on missing values

- In the prior case, specifying `NA` would work, instead of `NA_character_`
- Generally, I think it's better to specify the type.

Quick note on missing values

- In the prior case, specifying `NA` would work, instead of `NA_character_`
- Generally, I think it's better to specify the type.
- **General programming rule:** The more strict the better (leads to fewer unexpected results)

Quick note on missing values

- In the prior case, specifying `NA` would work, instead of `NA_character_`
- Generally, I think it's better to specify the type.
- **General programming rule:** The more strict the better (leads to fewer unexpected results)
- Because (base) R likes to be inconsistent, here are they `NA` types

Type	NA value
character	NA_character_
integer	NA_integer_
double	NA_real_
logical	NA (see here)

Quick note on missing values

- In the prior case, specifying `NA` would work, instead of `NA_character_`
- Generally, I think it's better to specify the type.
- **General programming rule:** The more strict the better (leads to fewer unexpected results)
- Because (base) R likes to be inconsistent, here are they `NA` types

Type	NA value
character	NA_character_
integer	NA_integer_
double	NA_real_
logical	NA (see here)

```
typeof(NA)
```

```
## [1] "logical"
```

Some examples

Please follow along

```
econ <- economics %>%
    mutate(year = lubridate::year(date))
econ
```

```
## # A tibble: 574 x 7
##    date            pce    pop psavert  uempmed unemploy  year
##    <date>        <dbl>  <int>   <dbl>    <dbl>    <int> <dbl>
##  1 1967-07-01 507.4    198712    12.5 4.5           2944  1967
##  2 1967-08-01 510.5    198911    12.5 4.7           2945  1967
##  3 1967-09-01 516.3000 199113    11.7 4.600000      2958  1967
##  4 1967-10-01 512.9    199311    12.5 4.9           3143  1967
##  5 1967-11-01 518.1    199498    12.5 4.7           3066  1967
##  6 1967-12-01 525.800  199657    12.1 4.8           3018  1967
##  7 1968-01-01 531.5    199808    11.7 5.100000      2878  1968
##  8 1968-02-01 534.2    199920    12.2 4.5           3001  1968
##  9 1968-03-01 544.9    200056    11.6 4.100000      2877  1968
## 10 1968-04-01 544.6    200208    12.2 4.600000      2709  1968
## # … with 564 more rows
```

by_year

```
by_year <- split(econ, econ$year)
str(by_year)
```

```
## List of 49
##  $ 1967:Classes 'tbl_df', 'tbl' and 'data.frame':    6 obs. of  7 variables:
##   ..$ date    : Date[1:6], format: "1967-07-01" ...
##   ..$ pce     : num [1:6] 507 510 516 513 518 ...
##   ..$ pop     : int [1:6] 198712 198911 199113 199311 199498 199657
##   ..$ psavert : num [1:6] 12.5 12.5 11.7 12.5 12.5 12.1
##   ..$ uempmed : num [1:6] 4.5 4.7 4.6 4.9 4.7 4.8
##   ..$ unemploy: int [1:6] 2944 2945 2958 3143 3066 3018
##   ..$ year    : num [1:6] 1967 1967 1967 1967 1967 ...
##  $ 1968:Classes 'tbl_df', 'tbl' and 'data.frame':    12 obs. of  7 variables:
##   ..$ date    : Date[1:12], format: "1968-01-01" ...
##   ..$ pce     : num [1:12] 532 534 545 545 550 ...
##   ..$ pop     : int [1:12] 199808 199920 200056 200208 200361 200536 200706 200
##   ..$ psavert : num [1:12] 11.7 12.2 11.6 12.2 12 11.6 10.6 10.4 10.4 10.6 ...
##   ..$ uempmed : num [1:12] 5.1 4.5 4.1 4.6 4.4 4.4 4.5 4.2 4.6 4.8 ...
##   ..$ unemploy: int [1:12] 2878 3001 2877 2709 2740 2938 2883 2768 2686 2689 ..
##   ..$ year    : num [1:12] 1968 1968 1968 1968 1968 ...
##  $ 1969:Classes 'tbl_df', 'tbl' and 'data.frame':    12 obs. of  7 variables:
##   ..$ date    : Date[1:12], format: "1969-01-01" ...
##   ..$ pce     : num [1:12] 584 590 590 595 601 ...
##   ..$ pop     : int [1:12] 201760 201881 202023 202161 202331 202507 202677 202
```

Fit a simple model to each year

Notes:

- We'll discuss a more elegant way to do this later
- This is not (statistically) the best way to approach this problem
- It's a good illustration, and in my experience there are lots of times with this approach works well, even if this particular example is a bit artificial

What is the relation between personal consumption expenditures (`pce`) and the unemployment percentage?

What is the relation between personal consumption expenditures (`pce`) and the unemployment percentage?

Problem: We don't have the percentage. Let's compute!

You try first!

```
perc <- map(by_year, ~mutate(.x, percent = unemploy / pop))
str(perc)
```

```
## List of 49
##  $ 1967:Classes 'tbl_df', 'tbl' and 'data.frame':    6 obs. of  8 variables:
##   ..$ date    : Date[1:6], format: "1967-07-01" ...
##   ..$ pce     : num [1:6] 507 510 516 513 518 ...
##   ..$ pop     : int [1:6] 198712 198911 199113 199311 199498 199657
##   ..$ psavert : num [1:6] 12.5 12.5 11.7 12.5 12.5 12.1
##   ..$ uempmed : num [1:6] 4.5 4.7 4.6 4.9 4.7 4.8
##   ..$ unemploy: int [1:6] 2944 2945 2958 3143 3066 3018
##   ..$ year    : num [1:6] 1967 1967 1967 1967 1967 ...
##   ..$ percent : num [1:6] 0.0148 0.0148 0.0149 0.0158 0.0154 ...
##  $ 1968:Classes 'tbl_df', 'tbl' and 'data.frame':    12 obs. of  8 variables:
##   ..$ date    : Date[1:12], format: "1968-01-01" ...
##   ..$ pce     : num [1:12] 532 534 545 545 550 ...
##   ..$ pop     : int [1:12] 199808 199920 200056 200208 200361 200536 200706 200
##   ..$ psavert : num [1:12] 11.7 12.2 11.6 12.2 12 11.6 10.6 10.4 10.4 10.6 ...
##   ..$ uempmed : num [1:12] 5.1 4.5 4.1 4.6 4.4 4.4 4.5 4.2 4.6 4.8 ...
##   ..$ unemploy: int [1:12] 2878 3001 2877 2709 2740 2938 2883 2768 2686 2689 ..
##   ..$ year    : num [1:12] 1968 1968 1968 1968 1968 ...
##   ..$ percent : num [1:12] 0.0144 0.015 0.0144 0.0135 0.0137 ...
##  $ 1969:Classes 'tbl_df', 'tbl' and 'data.frame':    12 obs. of  8 variables:
##   ..$ date    : Date[1:12], format: "1969-01-01" ...
##   ..$ pce     : num [1:12] 584 590 590 595 601 ...
##   ..$ pop     : int [1:12] 201760 201881 202023 202161 202331 202507 202677 202
##   ..$ psavert : num [1:12] 10 9.4 9.9 9.5 10 10.9 11.7 11.5 11.5 11.3 ...
```

Fit the models

Fit a model of the form `lm(percent ~ pce)` to each year

You try first!

Fit the models

Fit a model of the form `lm(percent ~ pce)` to each year

You try first!

```
mods <- map(perc, ~lm(percent ~ pce, data = .x))
str(mods)
```

```
## List of 49
##  $ 1967:List of 12
##   ..$ coefficients : Named num [1:2] 8.39e-03 1.31e-05
##   .. ..- attr(*, "names")= chr [1:2] "(Intercept)" "pce"
##   ..$ residuals    : Named num [1:6] -0.000205 -0.000255 -0.000281 0.000677 0.0
##   .. ..- attr(*, "names")= chr [1:6] "1" "2" "3" "4" ...
##   ..$ effects      : Named num [1:6] -0.037041 0.00019 -0.000261 0.000737 0.000
##   .. ..- attr(*, "names")= chr [1:6] "(Intercept)" "pce" "" "" ...
##   ..$ rank         : int 2
##   ..$ fitted.values: Named num [1:6] 0.015 0.0151 0.0151 0.0151 0.0152 ...
##   .. ..- attr(*, "names")= chr [1:6] "1" "2" "3" "4" ...
##   ..$ assign       : int [1:2] 0 1
##   ..$ qr           :List of 5
##   .. ..$ qr   : num [1:6, 1:2] -2.449 0.408 0.408 0.408 0.408 ...
##   .. .. ..- attr(*, "dimnames")=List of 2
```

Extract coefficients

You try first

Hint: use `coef`. For example, see `coef(mods[[1]])`

Extract coefficients

You try first

Hint: use `coef`. For example, see `coef(mods[[1]])`

```
coefs <- map(mods, coef)
coefs[c(1:2, length(coefs))]
```

```
## $`1967`
##  (Intercept)          pce
## 8.388042e-03 1.307101e-05
##
## $`1968`
##   (Intercept)           pce
##  2.785458e-02 -2.495714e-05
##
## $`2015`
##   (Intercept)           pce
##  1.908717e-01 -1.350436e-05
```

Extract slopes

AKA - the coefficient that is not the intercept

You try first

Extract slopes

AKA - the coefficient that is not the intercept

You try first

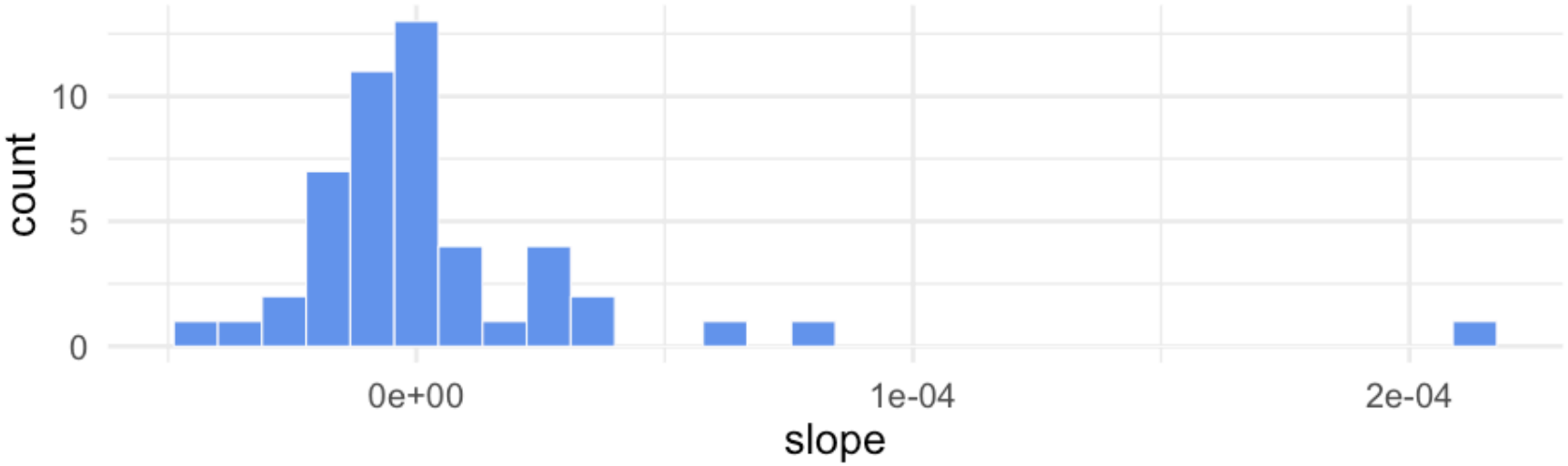
```
slopes <- map_dbl(coefs, 2)
slopes
```

```
##          1967          1968          1969          1970          1971
##  1.307101e-05 -2.495714e-05  3.188337e-05  2.130891e-04  1.176607e-05
##          1972          1973          1974          1975          1976
## -2.694906e-05 -5.988616e-06  8.064082e-05 -1.529659e-06  9.390419e-06
##          1977          1978          1979          1980          1981
## -3.343402e-05 -1.149169e-05  4.999290e-06  2.361504e-05  3.211201e-05
##          1982          1983          1984          1985          1986
##  6.278562e-05 -4.447567e-05 -1.688549e-05 -6.769360e-06 -6.697402e-06
##          1987          1988          1989          1990          1991
## -1.959304e-05 -7.205655e-06  4.437364e-06  2.609293e-05  1.842054e-05
##          1992          1993          1994          1995          1996
##  6.051336e-07 -1.369550e-05 -2.092534e-05  7.668013e-07 -5.571115e-06
##          1997          1998          1999          2000          2001
## -9.922250e-06 -2.262969e-06 -2.872153e-06 -1.595609e-06  2.935469e-05
##          2002          2003          2004          2005          2006
```

Plot

- I trust you can do this

```
relation <- tibble(year = names(slopes),
                   slope = slopes)

ggplot(relation, aes(slope)) +
    geom_histogram(fill = "cornflowerblue",
                   color = "white")
```



Piping

We could also have done the previous in a pipeline.

```
by_year %>%
    map(~mutate(.x, percent = unemploy / pop))
```

```
## $`1967`
## # A tibble: 6 x 8
##   date            pce    pop psavert  uempmed unemploy  year    percent
##   <date>        <dbl>  <int>   <dbl>    <dbl>    <int> <dbl>      <dbl>
## 1 1967-07-01 507.4    198712    12.5 4.5          2944  1967 0.01481541
## 2 1967-08-01 510.5    198911    12.5 4.7          2945  1967 0.01480562
## 3 1967-09-01 516.3000 199113    11.7 4.600000     2958  1967 0.01485589
## 4 1967-10-01 512.9    199311    12.5 4.9          3143  1967 0.01576933
## 5 1967-11-01 518.1    199498    12.5 4.7          3066  1967 0.01536858
## 6 1967-12-01 525.800  199657    12.1 4.8          3018  1967 0.01511592
##
## $`1968`
## # A tibble: 12 x 8
##    date            pce    pop psavert  uempmed unemploy  year    percent
##    <date>        <dbl>  <int>   <dbl>    <dbl>    <int> <dbl>      <dbl>
##  1 1968-01-01 531.5   199808    11.7 5.100000     2878  1968 0.01440383
##  2 1968-02-01 534.2   199920    12.2 4.5          3001  1968 0.01501100
##  3 1968-03-01 544.9   200056    11.6 4.100000     2877  1968 0.01438097
```

```
by_year %>%
    map(~mutate(.x, percent = unemploy / pop)) %>%
    map(~lm(percent ~ pce, data = .x))
```

```
## $`1967`
##
## Call:
## lm(formula = percent ~ pce, data = .x)
##
## Coefficients:
## (Intercept)          pce
##   8.388e-03    1.307e-05
##
##
## $`1968`
##
## Call:
## lm(formula = percent ~ pce, data = .x)
##
## Coefficients:
## (Intercept)          pce
##   2.785e-02   -2.496e-05
##
##
## $`1969`
##
## Call:
```

```
by_year %>%
    map(~mutate(.x, percent = unemploy / pop)) %>%
    map(~lm(percent ~ pce, data = .x)) %>%
    map(coef)
```

```
## $`1967`
##  (Intercept)          pce
## 8.388042e-03 1.307101e-05
##
## $`1968`
##   (Intercept)           pce
##  2.785458e-02 -2.495714e-05
##
## $`1969`
##   (Intercept)           pce
## -5.308326e-03  3.188337e-05
##
## $`1970`
##   (Intercept)           pce
## -0.1178905392  0.0002130891
##
## $`1971`
##  (Intercept)          pce
## 1.594169e-02 1.176607e-05
##
## $`1972`
##   (Intercept)           pce
```

```
slopes <- by_year %>%
    map(~mutate(.x, percent = unemploy / pop)) %>%
    map(~lm(percent ~ pce, data = .x)) %>%
    map(coef) %>%
    map_dbl(2)
slopes
```

```
##          1967          1968          1969          1970          1971
##  1.307101e-05 -2.495714e-05  3.188337e-05  2.130891e-04  1.176607e-05
##          1972          1973          1974          1975          1976
## -2.694906e-05 -5.988616e-06  8.064082e-05 -1.529659e-06  9.390419e-06
##          1977          1978          1979          1980          1981
## -3.343402e-05 -1.149169e-05  4.999290e-06  2.361504e-05  3.211201e-05
##          1982          1983          1984          1985          1986
##  6.278562e-05 -4.447567e-05 -1.688549e-05 -6.769360e-06 -6.697402e-06
##          1987          1988          1989          1990          1991
## -1.959304e-05 -7.205655e-06  4.437364e-06  2.609293e-05  1.842054e-05
##          1992          1993          1994          1995          1996
##  6.051336e-07 -1.369550e-05 -2.092534e-05  7.668013e-07 -5.571115e-06
##          1997          1998          1999          2000          2001
## -9.922250e-06 -2.262969e-06 -2.872153e-06 -1.595609e-06  2.935469e-05
##          2002          2003          2004          2005          2006
##  2.025961e-06 -1.264229e-06 -3.763984e-06 -3.742069e-06 -3.346146e-06
##          2007          2008          2009          2010          2011
##  4.200462e-06 -1.503429e-05  2.331515e-05 -5.152339e-06 -5.596069e-06
##          2012          2013          2014          2015
## -8.750080e-06 -1.380735e-05 -1.056487e-05 -1.350436e-05
```

Practice (if any time remains)

- Compute the standard deviation of every mtcars column.
- Use the following list of formulas to fit multiple models to mtcars

```
formulas <- list(mpg ~ disp,
                 mpg ~ I(1 / disp),
                 mpg ~ disp + wt,
                 mpg ~ I(1 / disp) + wt)
```

- Copy and run the following code to obtain 50 bootstrap samples

```
bootstrap <- function(df) {
  df[sample(nrow(df), replace = TRUE), , drop = FALSE]
}

samples <- map(1:50, ~bootstrap(mtcars))
```

- Fit the following model to each bootstrap sample: `mpg ~ disp`
- Extract R^2 and plot the distribution