Intro to {purrr}

And a more explicit description of functional programming

Daniel Anderson

Week 3

Agenda

- Review Lab 1
- Thinking more about functional programming
    - A small example
- Introduce you to {purrr} and contrast it with the base functions we learned last week

Review Lab 1

Learning objectives

- Understand how `purrr::map` relates to `lapply` and `for` loops
- Understand the four 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 × 11
##    manufacturer model      displ  year   cyl trans      drv     cty   hw
##    <chr>        <chr>      <dbl> <int> <int> <chr>      <chr> <int> <int
##  1 honda        civic        1.6  1999     4 manual(m5) f        28    3
##  2 honda        civic        1.6  1999     4 manual(m5) f        25    3
##  3 honda        civic        1.8  2008     4 manual(m5) f        26    3
##  4 honda        civic        1.8  2008     4 auto(l5)   f        25    3
##  5 toyota       corolla      1.8  1999     4 manual(m5) f        26    3
##  6 toyota       corolla      1.8  2008     4 manual(m5) f        28    3
##  7 toyota       corolla      1.8  2008     4 auto(l4)   f        26    3
##  8 volkswagen   jetta        1.9  1999     4 manual(m5) f        33    4
##  9 volkswagen   new beetle   1.9  1999     4 manual(m5) f        35    4
## 10 volkswagen   new beetle   1.9  1999     4 auto(l4)   f        29    4
## # … with 1 more variable: class <chr>
```

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

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

- Produce a plot for each

```
plot_mfr <-
    ggplot(count_mfr, 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",
             x = "",
             y = "") +
        guides(fill = "none")
```

Assemble the plots

```
library(patchwork)
plot_car / (plot_mfr + 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: tibble [10 × 11] (S3: tbl_df/tbl/data.frame)
##   ..$ 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 1
##   ..$ cyl         : int [1:10] 4 4 4 4 4 4 4 4 4 4
##   ..$ trans       : chr [1:10] "manual(m5)" "manual(m5)" "manual(m5)" "a
##   ..$ 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" "s
##  $ 5: tibble [2 × 11] (S3: tbl_df/tbl/data.frame)
##   ..$ 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
##   ..$ cyl         : int [1:2] 5 5
```

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))
})
counts
```

```
## $`4`
## $`4`$mfr
## # A tibble: 3 × 2
##   manufacturer     n
##   <chr>        <int>
## 1 honda            4
## 2 toyota           3
## 3 volkswagen       3
##
## $`4`$car
## # A tibble: 4 × 2
```

Plots

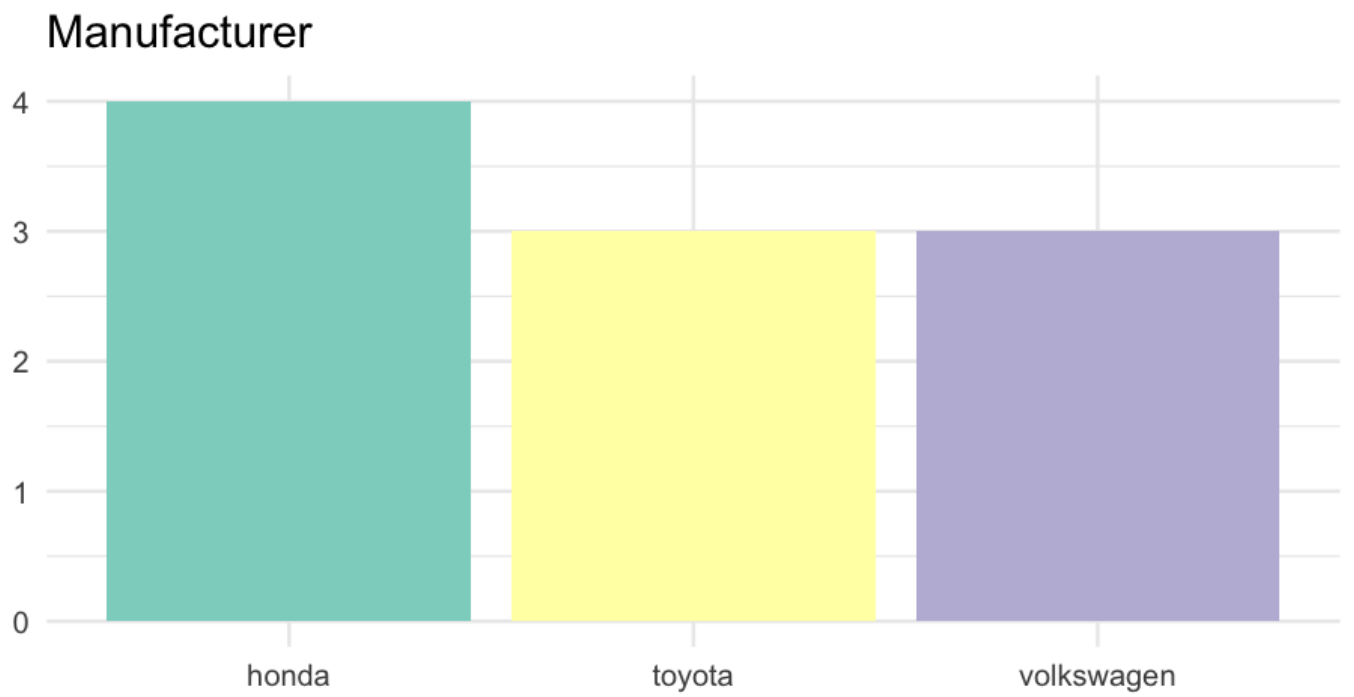
Let's write a couple functions

We'll mostly ignore how for now.

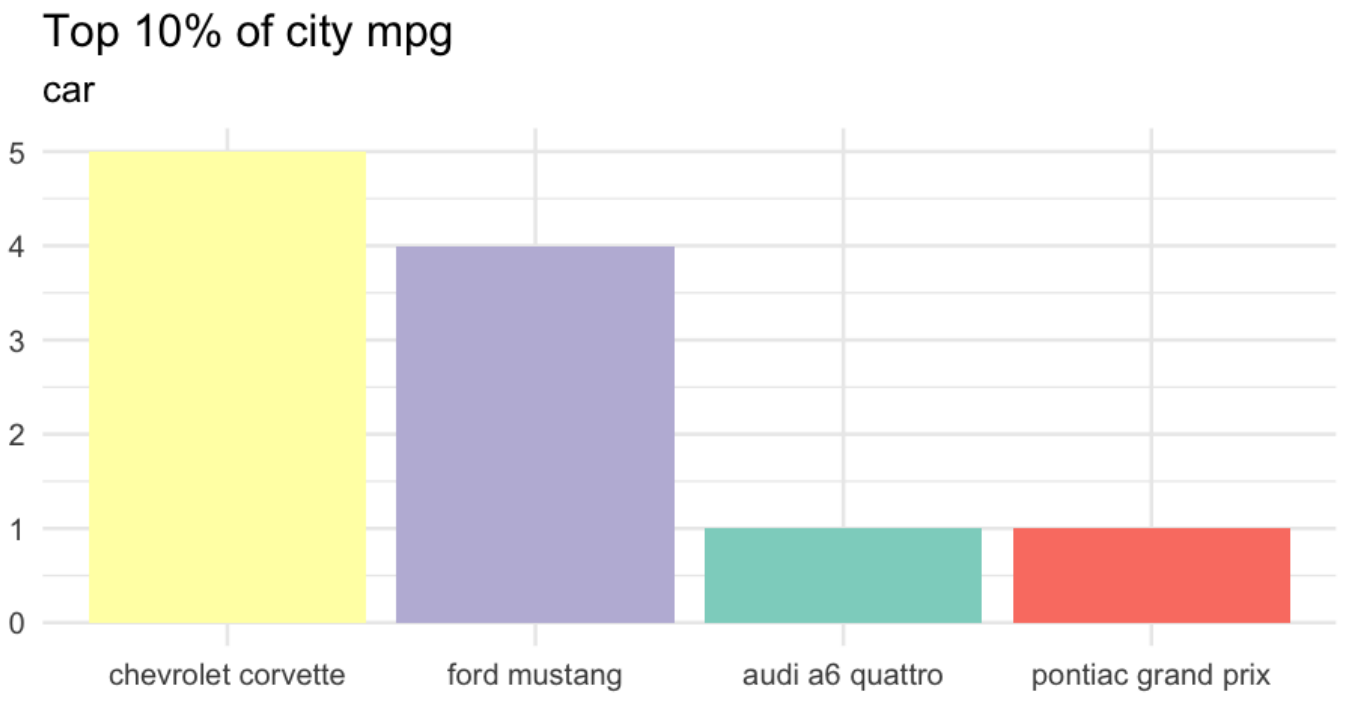
```
counts_plot <- function(counts_df) {
  var <- names(counts_df)[1]

  p <- ggplot(counts_df, aes(fct_reorder(!!sym(var), -n), n)) +
       geom_col(aes(fill = !!sym(var))) +
       scale_fill_brewer(palette = "Set3") +
       labs(title = stringr::str_to_title(var),
            x = "",
            y = "") +
       guides(fill = "none")

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

Test it

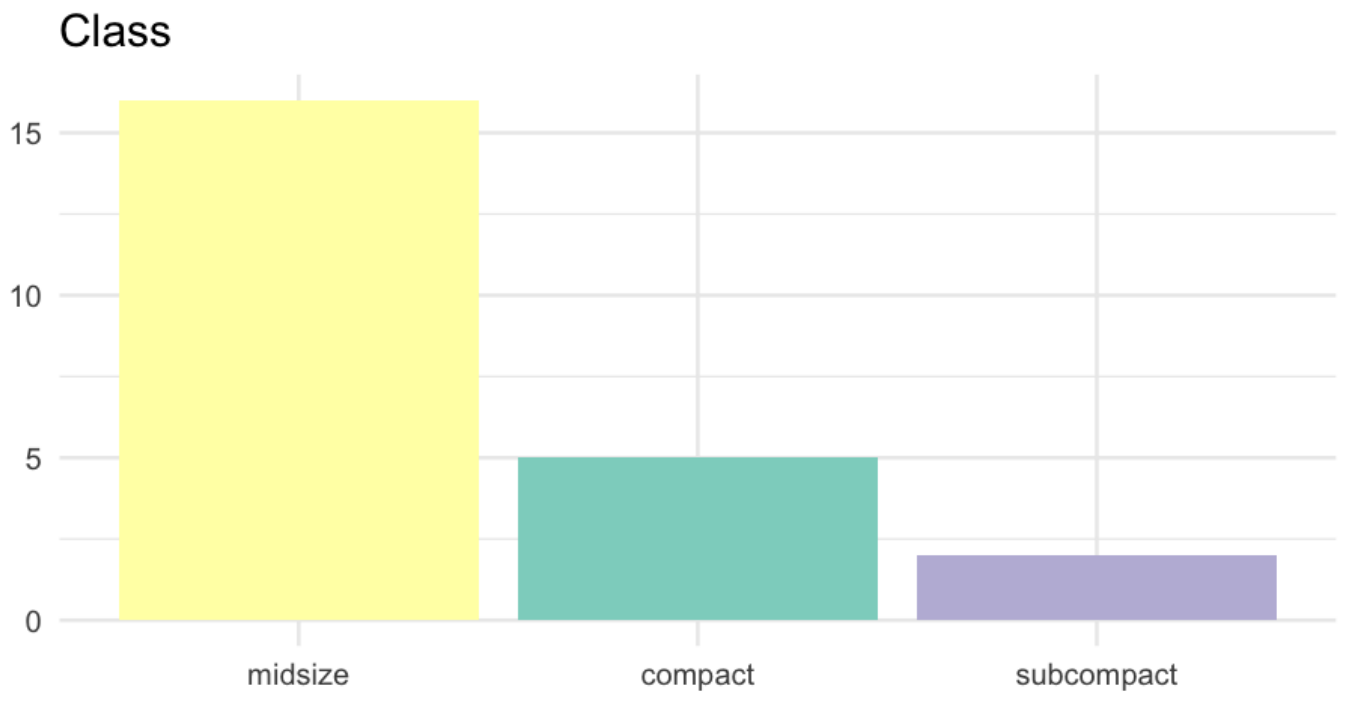
```
counts_plot(counts[["4"]]$mfr)
```



```
counts_plot(counts[["8"]]$car)
```



```
counts_plot(counts[["6"]]$class)
```

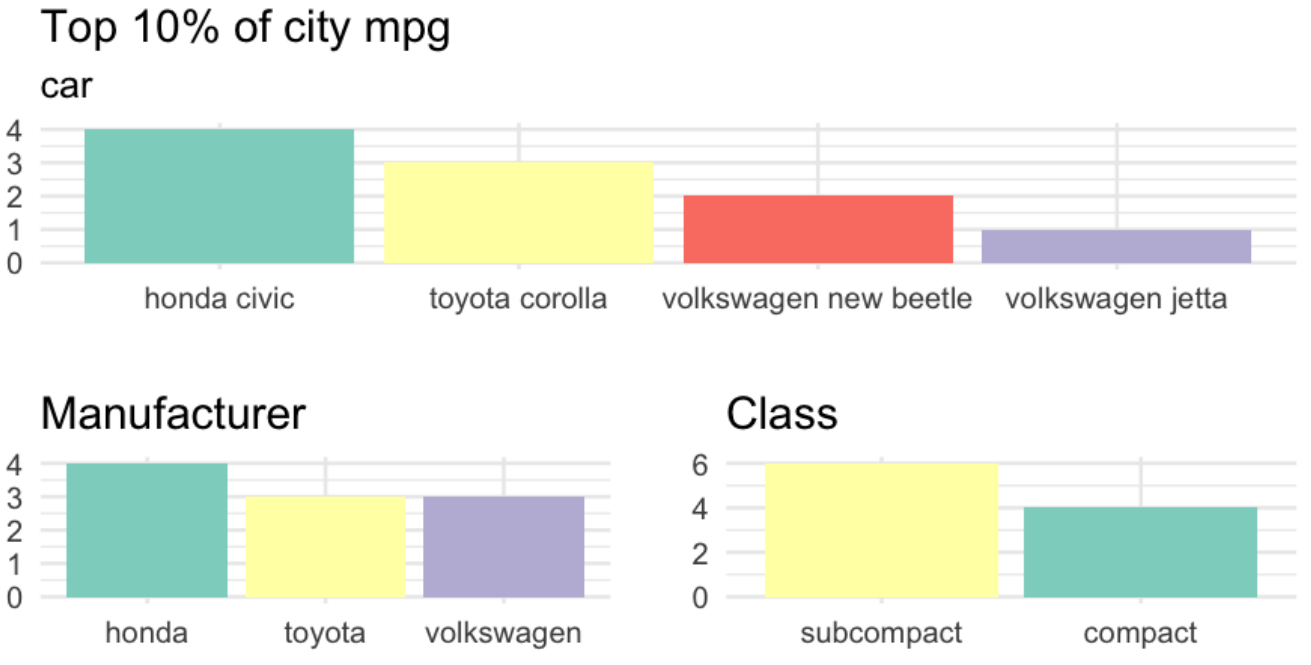


Compile plots function

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

Test it

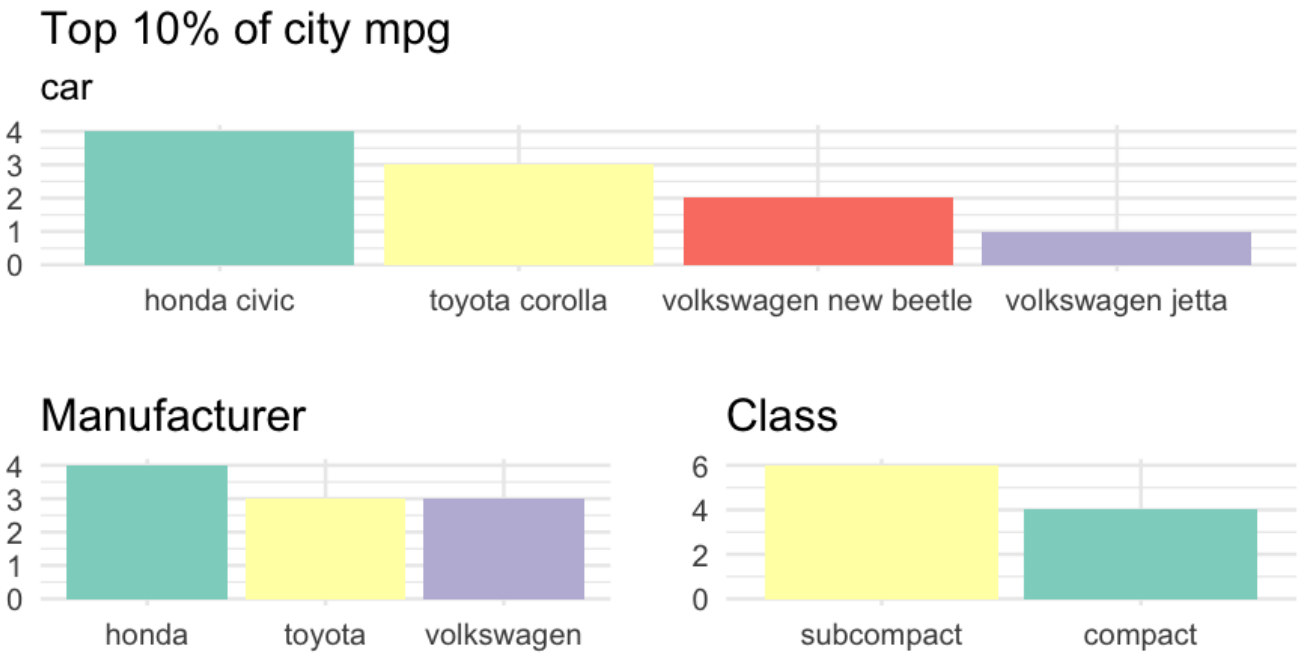
```
full_plot(counts[[1]])
```



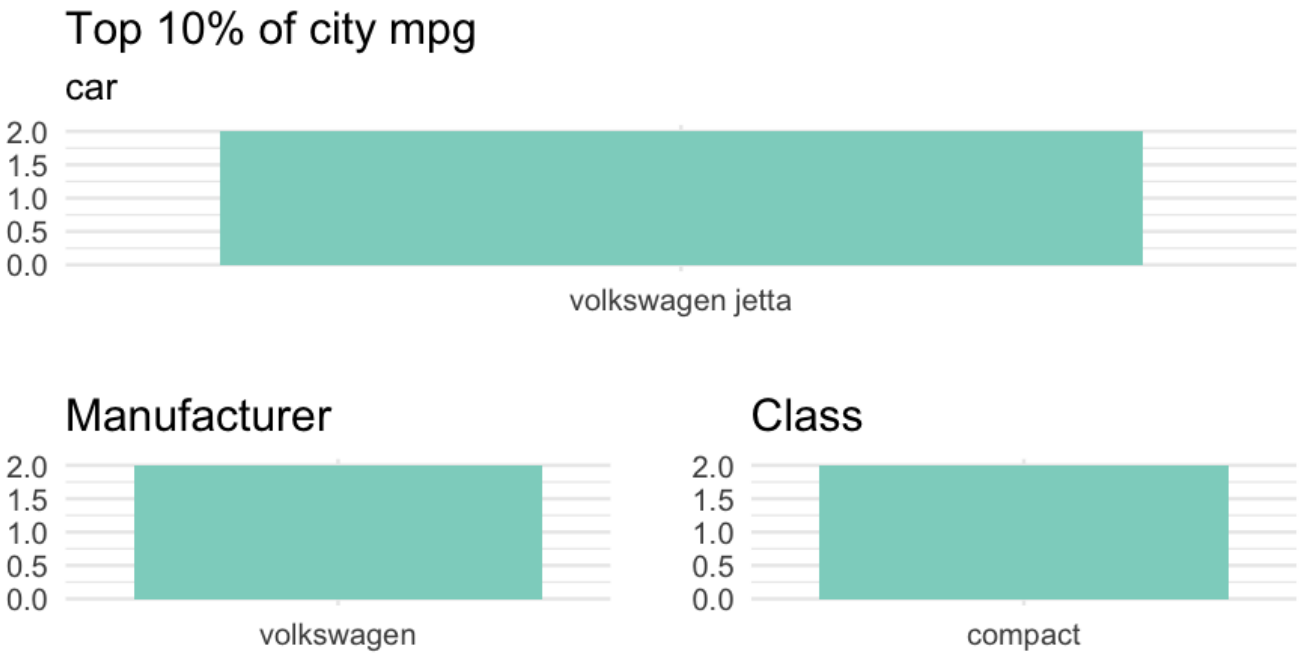
Finish up

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

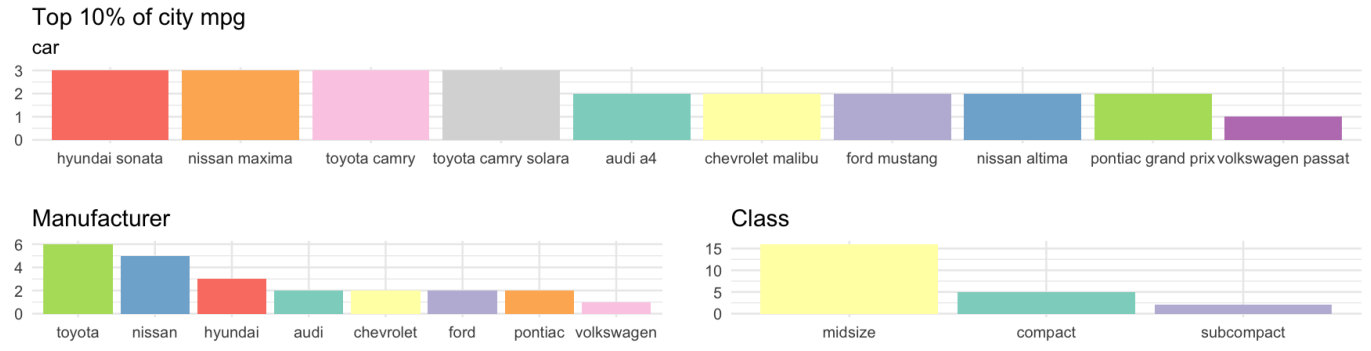
```
plots[[1]]
```


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

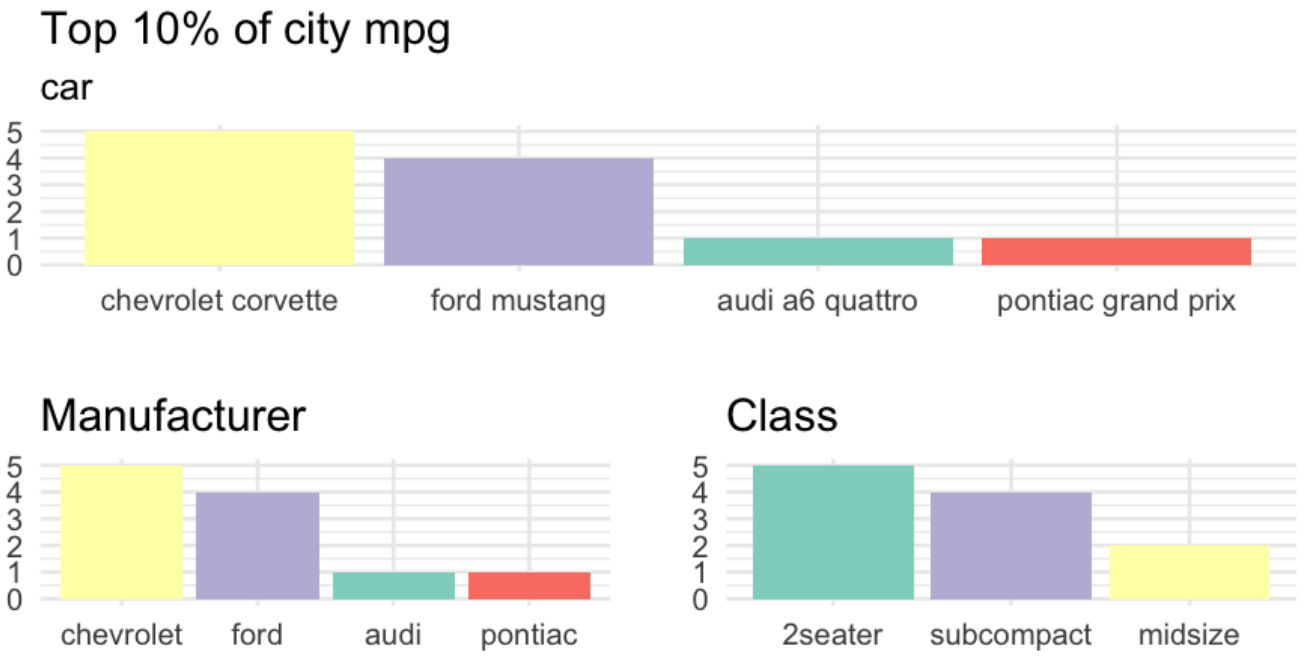
```
plots[[2]]
```


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.

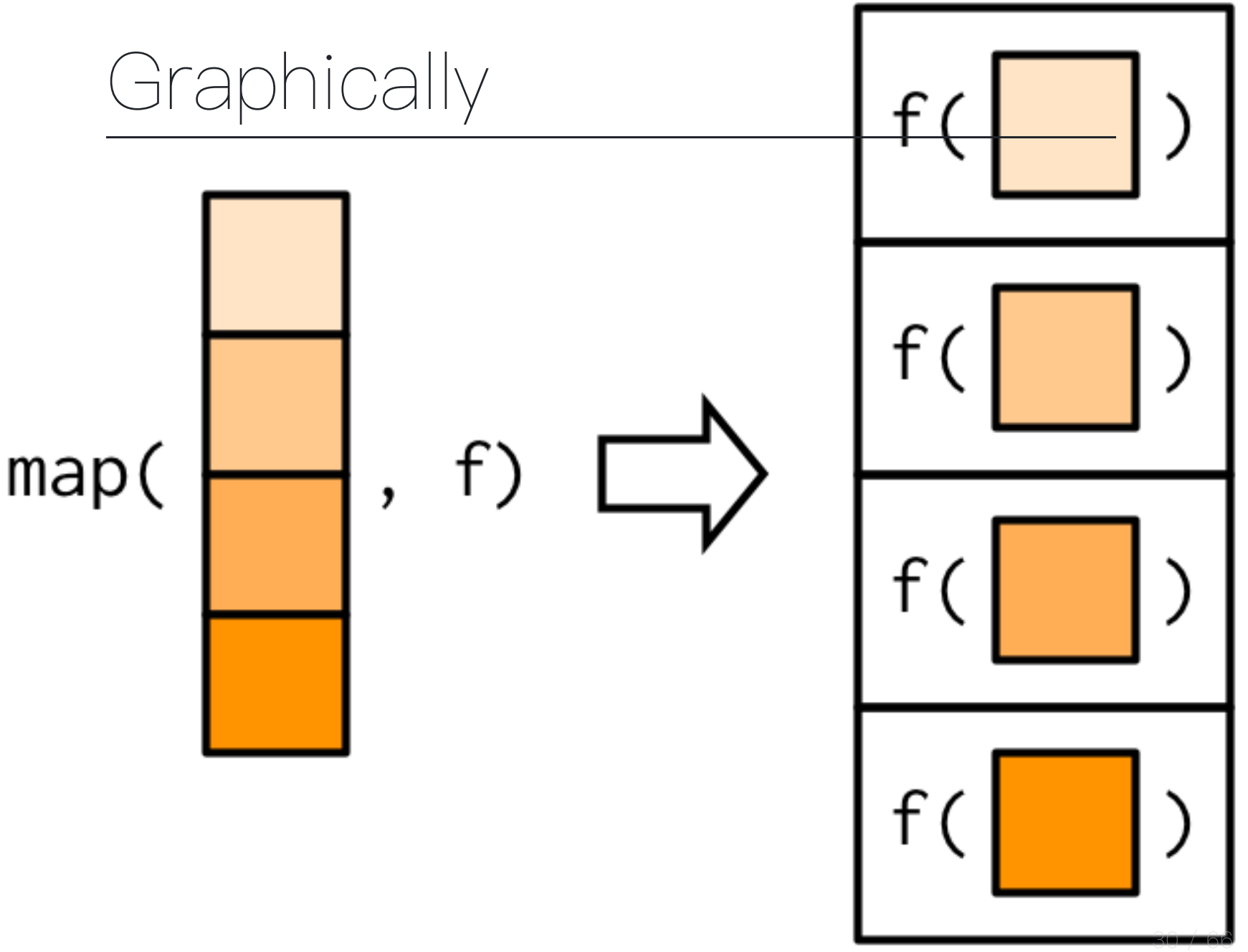
- What does this mean?

purrr::map

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

```
## [[1]]
## [1] -0.443447
##
## [[2]]
## [1] -1.0505721 -0.3394564
##
## [[3]]
## [1] -0.4678440  0.1258665  1.0731389
```

Graphically



Comparison to `base::lapply`

lapply

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

```
## [[1]]
## [1] 0.5279669
##
## [[2]]
## [1] -0.924577 -1.766093
##
## [[3]]
## [1] -0.8753488  0.4032877 -0.2509419
```

map

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

```
## [[1]]
## [1] 0.8399047
##
## [[2]]
## [1] -1.411129  1.268880
##
## [[3]]
## [1] -1.5068100  1.0414819  0.2418696
```

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)))
```

{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
```

~ is used in place of `function(.x)`

More examples

Vary the n

```
map(1:3, ~rnorm(n = .x))
```

```
## [[1]]
## [1] 0.9697506
##
## [[2]]
## [1]  0.9260836 -0.8879916
##
## [[3]]
## [1] -0.9369674  1.0084796 -0.2488219
```

Vary the mean

```
map(1:3, ~rnorm(n = 1, mean =
```

```
## [[1]]
## [1] 0.9968307
##
## [[2]]
## [1] 4.104241
##
## [[3]]
## [1] 3.887219
```

vary the sd

```
map(1:3, ~rnorm(n = 1, sd = .x))
```

```
## [[1]]
## [1] -1.040311
##
## [[2]]
## [1] -2.498778
##
## [[3]]
## [1] 0.5607935
```

Extracting elements

Let's make a rather complicated list

```
l <- list(
  list(-1, x = 1, y = 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
```

Instead, you have to apply an anonymous 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_chr`
- `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
```

```
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"
```

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

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

```
## Error in `stop_bad_type()`:
## ! Result 3 must be a single string, not NULL of length 0
```

- Unless you set a default value

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

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

Quick note: 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
- Because (base) R likes to be inconsistent, here are the `NA` types

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

```
typeof(NA)
```

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

More quick examples

Please copy the code below so you have it locally.

```
df_list <- list(
  data.frame(var1 = 1:5),
  data.frame(var1 = 1:3),
  data.frame(var1 = 1)
)
```

Compute `mean(var1)`

For each data frame. You try first!

```
map(df_list, ~mean(.x$var1))
```

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

04:00

Return a vector

```
map_dbl(df_list, ~mean(.x$var1))
```

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

Try out coercion

```
map_chr(df_list, ~mean(.x$var1))
```

```
## [1] "3.000000" "2.000000" "1.000000"
```

```
map_lgl(df_list, ~mean(.x$var1))
```

```
## Error: Can't coerce element 1 from a double to a logical
```

```
# Manual override
map_lgl(df_list, ~as.logical(mean(.x$var1)))
```

```
## [1] TRUE TRUE TRUE
```

Some more examples

Please follow along

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

```
## # A tibble: 574 × 7
##    date         pce    pop psavert uempmed unemploy  year
##    <date>     <dbl>  <dbl>   <dbl>   <dbl>    <dbl> <dbl>
##  1 1967-07-01 506.7 198712    12.6     4.5     2944  1967
##  2 1967-08-01 509.8 198911    12.6     4.7     2945  1967
##  3 1967-09-01 515.6 199113    11.9     4.6     2958  1967
##  4 1967-10-01 512.2 199311    12.9     4.9     3143  1967
##  5 1967-11-01 517.4 199498    12.8     4.7     3066  1967
##  6 1967-12-01 525.1 199657    11.8     4.8     3018  1967
##  7 1968-01-01 530.9 199808    11.7     5.1     2878  1968
##  8 1968-02-01 533.6 199920    12.3     4.5     3001  1968
##  9 1968-03-01 544.3 200056    11.7     4.1     2877  1968
## 10 1968-04-01 544   200208    12.3     4.6     2709  1968
## # … with 564 more rows
```

by_year

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

```
## List of 49
##  $ 1967: tibble [6 × 7] (S3: tbl_df/tbl/data.frame)
##   ..$ date    : Date[1:6], format: "1967-07-01" "1967-08-01" ...
##   ..$ pce     : num [1:6] 507 510 516 512 517 ...
##   ..$ pop     : num [1:6] 198712 198911 199113 199311 199498 ...
##   ..$ psavert : num [1:6] 12.6 12.6 11.9 12.9 12.8 11.8
##   ..$ uempmed : num [1:6] 4.5 4.7 4.6 4.9 4.7 4.8
##   ..$ unemploy: num [1:6] 2944 2945 2958 3143 3066 ...
##   ..$ year    : num [1:6] 1967 1967 1967 1967 1967 ...
##  $ 1968: tibble [12 × 7] (S3: tbl_df/tbl/data.frame)
##   ..$ date    : Date[1:12], format: "1968-01-01" "1968-02-01" ...
##   ..$ pce     : num [1:12] 531 534 544 544 550 ...
##   ..$ pop     : num [1:12] 2e+05 2e+05 2e+05 2e+05 2e+05 ...
##   ..$ psavert : num [1:12] 11.7 12.3 11.7 12.3 12 11.7 10.7 10.5 10.6 10
##   ..$ 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: num [1:12] 2878 3001 2877 2709 2740 ...
##   ..$ year    : num [1:12] 1968 1968 1968 1968 1968 ...
##  $ 1969: tibble [12 × 7] (S3: tbl_df/tbl/data.frame)
##   ..$ date    : Date[1:12], format: "1969-01-01" "1969-02-01" ...
##   ..$ pce     : num [1:12] 584 589 589 594 600 ...
##   ..$ pop     : num [1:12] 201760 201881 202023 202161 202331 ...
##   ..$ psavert : num [1:12] 10.3 9.7 10.2 9.7 10.1 11.1 11.8 11.5 11.6 11
```

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 where 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 over time?

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

You try first!

04:00

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

```
## List of 49
##  $ 1967: tibble [6 × 8] (S3: tbl_df/tbl/data.frame)
##   ..$ date    : Date[1:6], format: "1967-07-01" "1967-08-01" ...
##   ..$ pce     : num [1:6] 507 510 516 512 517 ...
##   ..$ pop     : num [1:6] 198712 198911 199113 199311 199498 ...
##   ..$ psavert : num [1:6] 12.6 12.6 11.9 12.9 12.8 11.8
##   ..$ uempmed : num [1:6] 4.5 4.7 4.6 4.9 4.7 4.8
##   ..$ unemploy: num [1:6] 2944 2945 2958 3143 3066 ...
##   ..$ year    : num [1:6] 1967 1967 1967 1967 1967 ...
##   ..$ percent : num [1:6] 0.0148 0.0148 0.0149 0.0158 0.0154 ...
##  $ 1968: tibble [12 × 8] (S3: tbl_df/tbl/data.frame)
##   ..$ date    : Date[1:12], format: "1968-01-01" "1968-02-01" ...
##   ..$ pce     : num [1:12] 531 534 544 544 550 ...
##   ..$ pop     : num [1:12] 2e+05 2e+05 2e+05 2e+05 2e+05 ...
##   ..$ psavert : num [1:12] 11.7 12.3 11.7 12.3 12 11.7 10.7 10.5 10.6 10
##   ..$ 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: num [1:12] 2878 3001 2877 2709 2740 ...
##   ..$ year    : num [1:12] 1968 1968 1968 1968 1968 ...
##   ..$ percent : num [1:12] 0.0144 0.015 0.0144 0.0135 0.0137 ...
##  $ 1969: tibble [12 × 8] (S3: tbl_df/tbl/data.frame)
##   ..$ date    : Date[1:12], format: "1969-01-01" "1969-02-01" ...
##   ..$ pce     : num [1:12] 584 589 589 594 600 ...
##   ..$ pop     : num [1:12] 201760 201881 202023 202161 202331 ...
##   ..$ psavert : num [1:12] 10.3 9.7 10.2 9.7 10.1 11.1 11.8 11.5 11.6 11
##   ..$ uempmed : num [1:12] 4.4 4.9 4 4 4.2 4.4 4.4 4.4 4.7 4.5 ...
##   ..$ unemploy: num [1:12] 2718 2692 2712 2758 2713 ...
##   ..$ year    : num [1:12] 1969 1969 1969 1969 1969 ...
```

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] 8.40e-03 1.31e-05
##   .. ..- attr(*, "names")= chr [1:2] "(Intercept)" "pce"
##   ..$ residuals    : Named num [1:6] -0.000205 -0.000255 -0.000281 0.000
##   .. ..- attr(*, "names")= chr [1:6] "1" "2" "3" "4" ...
##   ..$ effects      : Named num [1:6] -0.037041 0.00019 -0.000261 0.00073
##   .. ..- 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
##   .. .. .. ..$ : chr [1:6] "1" "2" "3" "4" ...
```

03:00

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
## 8.397192e-03 1.307101e-05
##
## $`1968`
##    (Intercept)           pce
##   2.784626e-02 -2.496983e-05
##
## $`2015`
##    (Intercept)           pce
##   1.176573e-01 -7.482080e-06
```

02:00

Extract slopes

AKA – the coefficient that is not the intercept

You try first

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

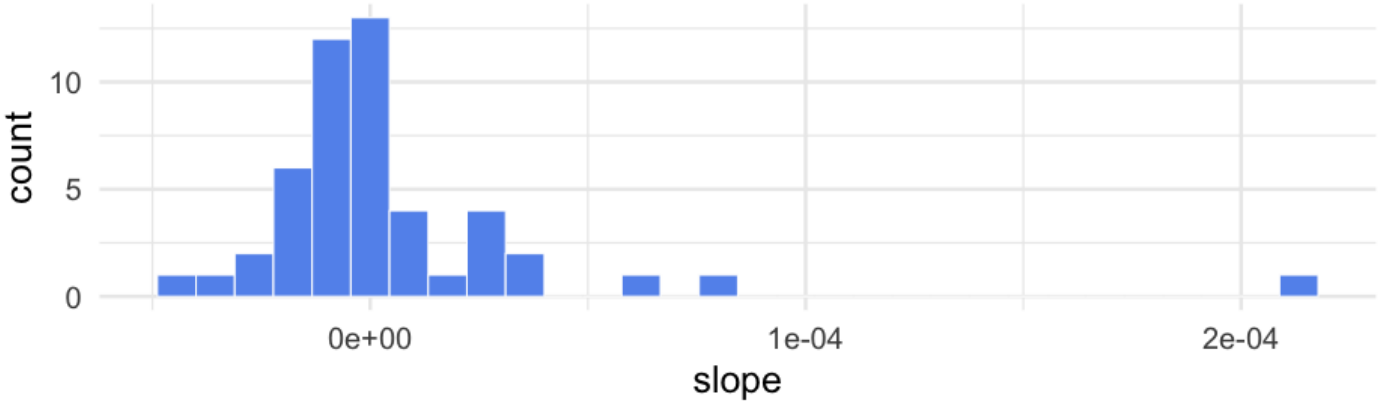
```
##          1967          1968          1969          1970          1971
##  1.307101e-05 -2.496983e-05  3.201794e-05  2.136662e-04  1.177357e-05 -2
##          1973          1974          1975          1976          1977
## -6.047148e-06  8.050869e-05 -1.643992e-06  9.413733e-06 -3.366894e-05 -1
##          1979          1980          1981          1982          1983
##  5.011539e-06  2.350410e-05  3.150845e-05  6.516215e-05 -4.402208e-05 -1
##          1985          1986          1987          1988          1989
## -6.926535e-06 -6.782516e-06 -1.980421e-05 -7.174276e-06  4.436762e-06  2
##          1991          1992          1993          1994          1995
##  1.842612e-05  6.091330e-07 -1.371548e-05 -2.102375e-05  7.718004e-07 -5
##          1997          1998          1999          2000          2001
## -9.936855e-06 -2.283149e-06 -2.893646e-06 -1.601844e-06  3.001365e-05  2
##          2003          2004          2005          2006          2007
## -1.306818e-06 -3.640025e-06 -3.679151e-06 -3.368536e-06  4.187576e-06 -1
##          2009          2010          2011          2012          2013
##  2.230078e-05 -5.844266e-06 -5.647401e-06 -1.070319e-05 -1.536441e-05 -8
```

02:00

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 × 8
##   date         pce    pop psavert uempmed unemploy  year    percent
##   <date>     <dbl>  <dbl>   <dbl>   <dbl>    <dbl> <dbl>      <dbl>
## 1 1967-07-01 506.7 198712    12.6     4.5     2944  1967 0.01481541
## 2 1967-08-01 509.8 198911    12.6     4.7     2945  1967 0.01480562
## 3 1967-09-01 515.6 199113    11.9     4.6     2958  1967 0.01485589
## 4 1967-10-01 512.2 199311    12.9     4.9     3143  1967 0.01576933
## 5 1967-11-01 517.4 199498    12.8     4.7     3066  1967 0.01536858
## 6 1967-12-01 525.1 199657    11.8     4.8     3018  1967 0.01511592
##
## $`1968`
## # A tibble: 12 × 8
##    date         pce    pop psavert uempmed unemploy  year    percent
##    <date>     <dbl>  <dbl>   <dbl>   <dbl>    <dbl> <dbl>      <dbl>
##  1 1968-01-01 530.9 199808    11.7     5.1     2878  1968 0.01440383
##  2 1968-02-01 533.6 199920    12.3     4.5     3001  1968 0.01501100
##  3 1968-03-01 544.3 200056    11.7     4.1     2877  1968 0.01438097
##  4 1968-04-01 544   200208    12.3     4.6     2709  1968 0.01353093
```

```
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
##   8.397e-03    1.307e-05
##
##
## $`1968`
##
## Call:
## lm(formula = percent ~ pce, data = .x)
##
## Coefficients:
## (Intercept)          pce
##   2.785e-02   -2.497e-05
##
##
## $`1969`
##
## Call:
## lm(formula = percent ~ pce, data = .x)
##
```

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

```
## $`1967`
##  (Intercept)          pce
## 8.397192e-03 1.307101e-05
##
## $`1968`
##   (Intercept)           pce
##  2.784626e-02 -2.496983e-05
##
## $`1969`
##   (Intercept)           pce
## -5.362991e-03  3.201794e-05
##
## $`1970`
##   (Intercept)           pce
## -0.1180577869  0.0002136662
##
## $`1971`
##  (Intercept)          pce
## 1.594909e-02 1.177357e-05
##
## $`1972`
##   (Intercept)           pce
##  0.0440400392 -0.0000270806
##
```

```
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 -2.496983e-05  3.201794e-05  2.136662e-04  1.177357e-05 -2
##          1973          1974          1975          1976          1977
## -6.047148e-06  8.050869e-05 -1.643992e-06  9.413733e-06 -3.366894e-05 -1
##          1979          1980          1981          1982          1983
##  5.011539e-06  2.350410e-05  3.150845e-05  6.516215e-05 -4.402208e-05 -1
##          1985          1986          1987          1988          1989
## -6.926535e-06 -6.782516e-06 -1.980421e-05 -7.174276e-06  4.436762e-06  2
##          1991          1992          1993          1994          1995
##  1.842612e-05  6.091330e-07 -1.371548e-05 -2.102375e-05  7.718004e-07 -5
##          1997          1998          1999          2000          2001
## -9.936855e-06 -2.283149e-06 -2.893646e-06 -1.601844e-06  3.001365e-05  2
##          2003          2004          2005          2006          2007
## -1.306818e-06 -3.640025e-06 -3.679151e-06 -3.368536e-06  4.187576e-06 -1
##          2009          2010          2011          2012          2013
##  2.230078e-05 -5.844266e-06 -5.647401e-06 -1.070319e-05 -1.536441e-05 -8
##          2015
## -7.482080e-06
```

More Practice

If we have time before the lab

Practice

- Compute the standard deviation of every mtcars column.
- 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

Lab 2

Midterm quiz next week