ID 543: Day 2

Introduction to R

Homework review

Today's goals

- Learn how to select variables with `select()`
- Learn how to subset data with `filter()`
- Learn how to use the pipe (`%>%` or `|>`) to chain functions together

Selecting the variables you want

- We've made tons of new variables!
- You don't want to keep them all!
- Luckily there's an easy way to select the variables you want: `select()`!

```
1 nlsy_subset <- select(nlsy, id, income, eyesight, sex, region)
2 nlsy_subset
```

```
# A tibble: 1,205 × 5
      id income eyesight   sex region
   <dbl>  <dbl>    <dbl> <dbl>  <dbl>
 1     3  22390        1     2      1
 2     6  35000        2     1      1
 3     8   7227        2     2      1
 4    16  48000        3     2      1
 5    18   4510        3     1      3
 6    20  50000        2     2      1
 7    27  20000        1     2      1
 8    49  23900        1     2      1
 9    57  23289        2     2      1
10    67  35000        1     1      1
```

`select()` syntax

- Like `mutate()`, the first argument is the dataset you want to select from
- Then you can just list the variables you want!
- Or you can list the variables you *don't* want, preceded by a negative sign (`-`)

```
1 select(nlsy_subset, -c(id, region))
```

```
# A tibble: 1,205 × 3
   income eyesight   sex
    <dbl>    <dbl> <dbl>
 1  22390        1     2
 2  35000        2     1
 3   7227        2     2
 4  48000        3     2
 5   4510        3     1
 6  50000        2     2
 7  20000        1     2
 8  23900        1     2
 9  23289        2     2
10  35000        1     1
```

`select()` syntax

There are also a lot of “helpers”!

- `var1:var10` (consecutively placed variables)
- `all_of()`/`any_of()`
- `starts_with()`
- `ends_with()`
- `contains()`
- `matches()` (like contains, but for regular expressions)
- `num_range()` (for patterns like `x01`, `x02`, ...)
- `everything()`
- `where(is.factor)` (or anything else)

Tip

Like the `fct_()` functions, you don’t need to memorize all these! Just know they exist and you can look them up when you need them.

`all_of()`

Notice that the variable names we used in `select()` weren't in quotation marks.

Let's say you have a vector of column names that you want. Then you can use `all_of()` to choose them.

```
cols_I_want <- c("age_bir", "nsibs", "region")
select(nlsy, all_of(cols_I_want))
```

```
# A tibble: 1,205 × 3
   age_bir nsibs region
     <dbl> <dbl>  <dbl>
 1      19     3      1
 2      30     1      1
 3      17     7      1
 4      31     3      1
 5      19     2      3
 6      30     2      1
 7      27     1      1
 8      24     6      1
 9      21     1      1
10      36     1      1
# ℹ 1,195 more rows
```

If you don't want an error if they don't exist, use `any_of()`.

`starts_with`, `ends_with`

You can also use `starts_with()` and `ends_with()` to select variables that start or end with a certain string.

```
1 new_subset <- select(nlsy, starts_with("sleep"), ends_with("d"))
```

What variables are in `new_subset`?

> **Note**
>
> Recall that `nlsy` has variables “id”, “glasses”, “eyesight”, “sleep_wkdy”, “sleep_wknd”, “nsibs”, “race_eth”, “sex”, “region”, “income”, “age_bir”, “eyesight_cat”, “glasses_cat”, “race_eth_cat”, “sex_cat”.

starts_with, ends_with

```
1 new_subset <- select(nlsy, starts_with("sleep"), ends_with("d"))
2 new_subset
```

```
# A tibble: 1,205 × 3
   sleep_wkdy sleep_wknd    id
        <dbl>      <dbl> <dbl>
 1          5          7     3
 2          6          7     6
 3          7          9     8
 4          6          7    16
 5         10         10    18
 6          7          8    20
 7          8          8    27
 8          8          8    49
 9          7          8    57
10          8          8    67
# ℹ 1,195 more rows
```

Note

Variables won't be repeated even if they meet multiple criteria!

`starts_with`, `ends_with`

Use `&` in between multiple criteria if they have to meet *all* of them

```
1 select(nlsy, starts_with("sleep") & ends_with("d"))
```

```
# A tibble: 1,205 × 1
   sleep_wknd
        <dbl>
 1          7
 2          7
 3          9
 4          7
 5         10
 6          8
 7          8
 8          8
 9          8
10          8
# ℹ 1,195 more rows
```

where()

`where()` is a helper that lets you select variables based on their properties

```
1 select(nlsy, where(is.factor))
```

```
# A tibble: 1,205 × 4
   eyesight_cat glasses_cat            race_eth_cat            sex_cat
   <fct>        <fct>                  <fct>                   <fct>
 1 Excellent    Doesn't wear glasses   Non-Black, Non-Hispanic Female
 2 Very Good    Wears glasses/contacts Non-Black, Non-Hispanic Male
 3 Very Good    Doesn't wear glasses   Non-Black, Non-Hispanic Female
 4 Good         Wears glasses/contacts Non-Black, Non-Hispanic Female
 5 Good         Doesn't wear glasses   Non-Black, Non-Hispanic Male
 6 Very Good    Wears glasses/contacts Non-Black, Non-Hispanic Female
 7 Excellent    Doesn't wear glasses   Non-Black, Non-Hispanic Female
 8 Excellent    Wears glasses/contacts Non-Black, Non-Hispanic Female
 9 Very Good    Wears glasses/contacts Non-Black, Non-Hispanic Female
10 Excellent    Doesn't wear glasses   Non-Black, Non-Hispanic Male
# ℹ 1,195 more rows
```

Think back to how we named our factor variables. What's another way we could have selected them?

Rearranging variables

You can use these helpers to rearrange variables

```
1  select(nlsy, id, where(is.factor), where(is.numeric), everything())
```

```
# A tibble: 1,205 × 15
      id eyesight_cat glasses_cat          race_eth_cat sex_cat glasses eyesight
   <dbl> <fct>        <fct>                <fct>        <fct>     <dbl>    <dbl>
 1     3 Excellent    Doesn't wear glasses Non-Black, … Female        0        1
 2     6 Very Good    Wears glasses/conta… Non-Black, … Male          1        2
 3     8 Very Good    Doesn't wear glasses Non-Black, … Female        0        2
 4    16 Good         Wears glasses/conta… Non-Black, … Female        1        3
 5    18 Good         Doesn't wear glasses Non-Black, … Male          0        3
 6    20 Very Good    Wears glasses/conta… Non-Black, … Female        1        2
 7    27 Excellent    Doesn't wear glasses Non-Black, … Female        0        1
 8    49 Excellent    Wears glasses/conta… Non-Black, … Female        1        1
 9    57 Very Good    Wears glasses/conta… Non-Black, … Female        1        2
10    67 Excellent    Doesn't wear glasses Non-Black, … Male          0        1
# ℹ 1,195 more rows
# ℹ 8 more variables: sleep_wkdy <dbl>, sleep_wknd <dbl>, nsibs <dbl>,
#   race_eth <dbl>, sex <dbl>, region <dbl>, income <dbl>, age_bir <dbl>
```

Tip

If you end with `everything()`, it's basically saying "and everything else" in its original order.

Exercises

Subsetting data

We usually don't do an analysis in an *entire* dataset. We usually apply some eligibility criteria to find the people who we will analyze. One function we can use to do that in R is `filter()`.

```
1 wear_glasses <- filter(nlsy, glasses == 1)
2
3 nrow(wear_glasses)
```

```
[1] 624
```

```
1 summary(wear_glasses$glasses)
```

```
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
     1       1       1       1       1       1
```

`filter()` syntax

- Like the other functions, we give `filter()` the dataset first, then we give it a series of criteria that we want to subset our data on.
- As with `case_when()`, these criteria should be questions with `TRUE`/`FALSE` answers. We'll keep all those rows for which the answer is `TRUE`.
- If there are multiple criteria, we can connect them with `&` or just by separating with commas, and we'll get back only the rows that answer `TRUE` to all of them.

```
no_glasses <- filter(nlsy, glasses == 0)
nrow(no_glasses)
```

```
[1] 581
```

```
glasses_great_eyes <- filter(nlsy, glasses == 0, eyesight == 1)
nrow(glasses_great_eyes)
```

```
[1] 220
```

```
yesno_glasses <- filter(nlsy, glasses == 0, glasses == 1)
nrow(yesno_glasses)
```

```
[1] 0
```

Another new function: `nrow()`

`nrow()` tells you how many rows are in a dataset

Of course, you can look in the environment pane, print a tibble, or use a function like `glimpse()`, but with `nrow()` you can get the number of rows in a dataset programatically

Tip

There's also `ncol()` for columns, or `dim()` for both!

Logicals in R

When we used `case_when()`, we got `TRUE`/`FALSE` answers when we asked whether a variable was `>` or `<` some number, for example.

When we want to know if something is

- equal: `==`
- not equal: `!=`
- greater than or equal to: `>=`
- less than or equal to: `<=`

We also can ask about multiple conditions with `&` (and) and `|` (or).

Behind the scenes of `filter()`

We are pulling out everyone for whom the statement evaluates to `TRUE`:

```
1 nlsy$glasses == 0
```

```
  [1]  TRUE FALSE  TRUE FALSE
TRUE FALSE  TRUE FALSE FALSE
TRUE  TRUE FALSE
 [13] FALSE  TRUE  TRUE  TRUE
TRUE FALSE FALSE FALSE  TRUE
FALSE  TRUE  TRUE
 [25]  TRUE  TRUE FALSE  TRUE
TRUE FALSE  TRUE FALSE FALSE
TRUE FALSE FALSE
 [37]  TRUE  TRUE  TRUE  TRUE
FALSE  TRUE FALSE  TRUE FALSE
TRUE  TRUE FALSE
 [49] FALSE  TRUE FALSE FALSE
TRUE  TRUE  TRUE FALSE  TRUE
FALSE  TRUE FALSE
 [61] FALSE  TRUE FALSE FALSE
FALSE  TRUE  TRUE FALSE FALSE
FALSE FALSE  TRUE
 [73] FALSE  TRUE FALSE  TRUE
TRUE FALSE  TRUE FALSE  TRUE
```

```
1 nlsy$eyesight == 1
```

```
  [1]  TRUE FALSE FALSE FALSE
FALSE FALSE  TRUE  TRUE FALSE
TRUE FALSE FALSE
 [13]  TRUE  TRUE  TRUE  TRUE
FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE
 [25]  TRUE  TRUE FALSE FALSE
TRUE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE
 [37] FALSE FALSE FALSE FALSE
FALSE  TRUE  TRUE  TRUE FALSE
TRUE  TRUE  TRUE
 [49] FALSE  TRUE FALSE  TRUE
TRUE FALSE FALSE  TRUE  TRUE
TRUE FALSE  TRUE
 [61]  TRUE FALSE FALSE FALSE
FALSE  TRUE  TRUE  TRUE  TRUE
FALSE  TRUE  TRUE
 [73]  TRUE FALSE FALSE  TRUE
TRUE FALSE FALSE FALSE FALSE
```

```
1 nlsy$glasses == 0
```

```
  [1]  TRUE FALSE FALSE FALSE
FALSE FALSE  TRUE FALSE FALSE
TRUE FALSE FALSE
 [13] FALSE  TRUE  TRUE  TRUE
FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE
 [25]  TRUE  TRUE FALSE FALSE
TRUE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE
 [37] FALSE FALSE FALSE FALSE
FALSE  TRUE FALSE  TRUE FALSE
TRUE  TRUE FALSE
 [49] FALSE  TRUE FALSE FALSE
TRUE FALSE FALSE FALSE  TRUE
FALSE FALSE FALSE
 [61] FALSE FALSE FALSE FALSE
FALSE  TRUE  TRUE FALSE FALSE
FALSE FALSE  TRUE
 [73] FALSE FALSE FALSE  TRUE
TRUE FALSE FALSE FALSE FALSE
```

Or statements

To get the extreme values of eyesight (1 and 5), we would do something like:

```
1 extreme_eyes <- filter(nlsy, eyesight == 1 |
2                          eyesight == 5)
3 count(extreme_eyes, eyesight)
```

```
# A tibble: 2 × 2
  eyesight     n
     <dbl> <int>
1        1   474
2        5    19
```

We could of course do the same thing with a factor variable:

```
1 nlsy <- mutate(nlsy, region_cat = factor(region, labels = c("Northeast", "North Central", "Sout
2 some_regions <- filter(nlsy, region_cat == "Northeast" |
3                          region_cat == "South")
4 count(some_regions, region_cat)
```

```
# A tibble: 2 × 2
  region_cat     n
  <fct>      <int>
1 Northeast    206
2 South        411
```

Exercises

Multiple "or" possibilities

Often we have a number of options for one variable that would meet our eligibility criteria. R's special `%in%` function comes in handy here:

```
more_regions <- filter(nlsy, region_cat %in%
                         c("South", "West", "Northeast"))
```

If the variable's value is any one of those values, it will return `TRUE`.

> **Note**
>
> This is the same as saying `region_cat == "South" | region_cat == "West" | region_cat == "Northeast"`

More `%in%`

This is just a regular R function that works outside of the `filter()` function, of course!

```
1 7 %in% c(4, 6, 7, 10)
```

```
[1] TRUE
```

```
1 5 %in% c(4, 6, 7, 10)
```

```
[1] FALSE
```

This is just the same as writing:

```
1 7 == 4 | 7 == 6 | 7 == 7 | 7 == 10
```

```
[1] TRUE
```

Opposite of `%in%`

We can't say "not in" with the syntax `%!in%` or something like that. We have to put the `!` *before* the question to make it the opposite of what it otherwise would be

```
!7 %in% c(4, 6, 7, 10)
```

```
[1] FALSE
```

```
!5 %in% c(4, 6, 7, 10)
```

```
[1] TRUE
```

```
northcentralers <- filter(nlsy,
                          !region_cat %in%
                            c("South", "West", "Northeast"))
count(northcentralers, region_cat)
```

```
# A tibble: 1 × 2
  region_cat        n
  <fct>         <int>
1 North Central   333
```

Other questions

R offers a number of shortcuts to use when determining whether values meet certain criteria:

- `is.na()`: is it a missing value?
- `is.finite()` / `is.infinite()`: when you might have infinite values in your data
- `is.factor()`: asks whether some variable is a factor

You can find lots of these if you tab-complete `is.` or `is_` (the latter are tidyverse versions). Most you will never find a use for!

Putting it all together

We can of course `filter()` on multiple variables at once:

```
my_data <- filter(nlsy,
                  age_bir < 20,
                  sex != 1,
                  nsibs %in% c(1, 2, 3),
                  !is.na(sleep_wkdy))

summary(select(my_data, age_bir, sex, nsibs, sleep_wkdy))
```

```
   age_bir          sex          nsibs         sleep_wkdy
Min.   :14.00   Min.   :2   Min.   :1.000   Min.   : 2.000
1st Qu.:17.00   1st Qu.:2   1st Qu.:1.750   1st Qu.: 6.000
Median :18.00   Median :2   Median :2.000   Median : 7.000
Mean   :17.42   Mean   :2   Mean   :2.163   Mean   : 6.452
3rd Qu.:19.00   3rd Qu.:2   3rd Qu.:3.000   3rd Qu.: 7.000
Max.   :19.00   Max.   :2   Max.   :3.000   Max.   :10.000
```

Putting it all together

Spot the difference?

```
1 oth_dat <- filter(nlsy,
2                   (age_bir < 20) &
3                     (sex != 1 | nsibs %in% c(1, 2, 3)) &
4                     !is.na(sleep_wkdy))
5
6 summary(select(oth_dat, age_bir, sex, nsibs, sleep_wkdy))
```

```
   age_bir          sex            nsibs          sleep_wkdy
Min.   :13.00   Min.   :1.000   Min.   : 0.000   Min.   : 0.000
1st Qu.:16.00   1st Qu.:2.000   1st Qu.: 2.000   1st Qu.: 6.000
Median :17.00   Median :2.000   Median : 4.000   Median : 7.000
Mean   :17.24   Mean   :1.929   Mean   : 4.539   Mean   : 6.687
3rd Qu.:18.00   3rd Qu.:2.000   3rd Qu.: 6.000   3rd Qu.: 8.000
Max.   :19.00   Max.   :2.000   Max.   :16.000   Max.   :13.000
```

Exercises

Putting it together

What does this sequence of code do?

```
nlsy2 <- mutate(nlsy,
                only = case_when(
                  nsibs == 0 ~ "yes",
                  .default = "no"))
nlsy3 <- select(nlsy2, id, contains("sleep"), only)
only_kids <- filter(nlsy3, only == "yes")
only_kids
```

```
# A tibble: 30 × 4
      id sleep_wkdy sleep_wknd only
   <dbl>      <dbl>      <dbl> <chr>
 1   458          7          8 yes
 2   653          6          7 yes
 3  1101          7          8 yes
 4  1166          5          6 yes
 5  2163          7          8 yes
 6  2442          7          9 yes
 7  2545          8          8 yes
 8  3036          5          8 yes
 9  3194          7          7 yes
```

Putting it together

What does this sequence of code do?

```
nlsy2 <- filter(nlsy, age_bir < 20)
nlsy3 <- select(nlsy2, id, contains("cat"), age_bir)
nlsy_final <- mutate(nlsy3,
                     age_bir_decade = case_when(
                        age_bir < 20 ~ "<20",
                        age_bir < 30 ~ "20-29",
                        age_bir < 40 ~ "30-39",
                        age_bir < 50 ~ "40-49",
                        age_bir >= 50 ~ "50+"
                        ),
                      age_bir_decade = fct_relevel(age_bir_decade,
                      "<20", "20-29", "30-39", "40-49", "50+")
                     )
```

Exercises

Sequence of functions

In any data management and/or analysis task, we perform a series of functions to the data until we get some object we want.

Sometimes this can be hard to read/keep track of.

The pipe

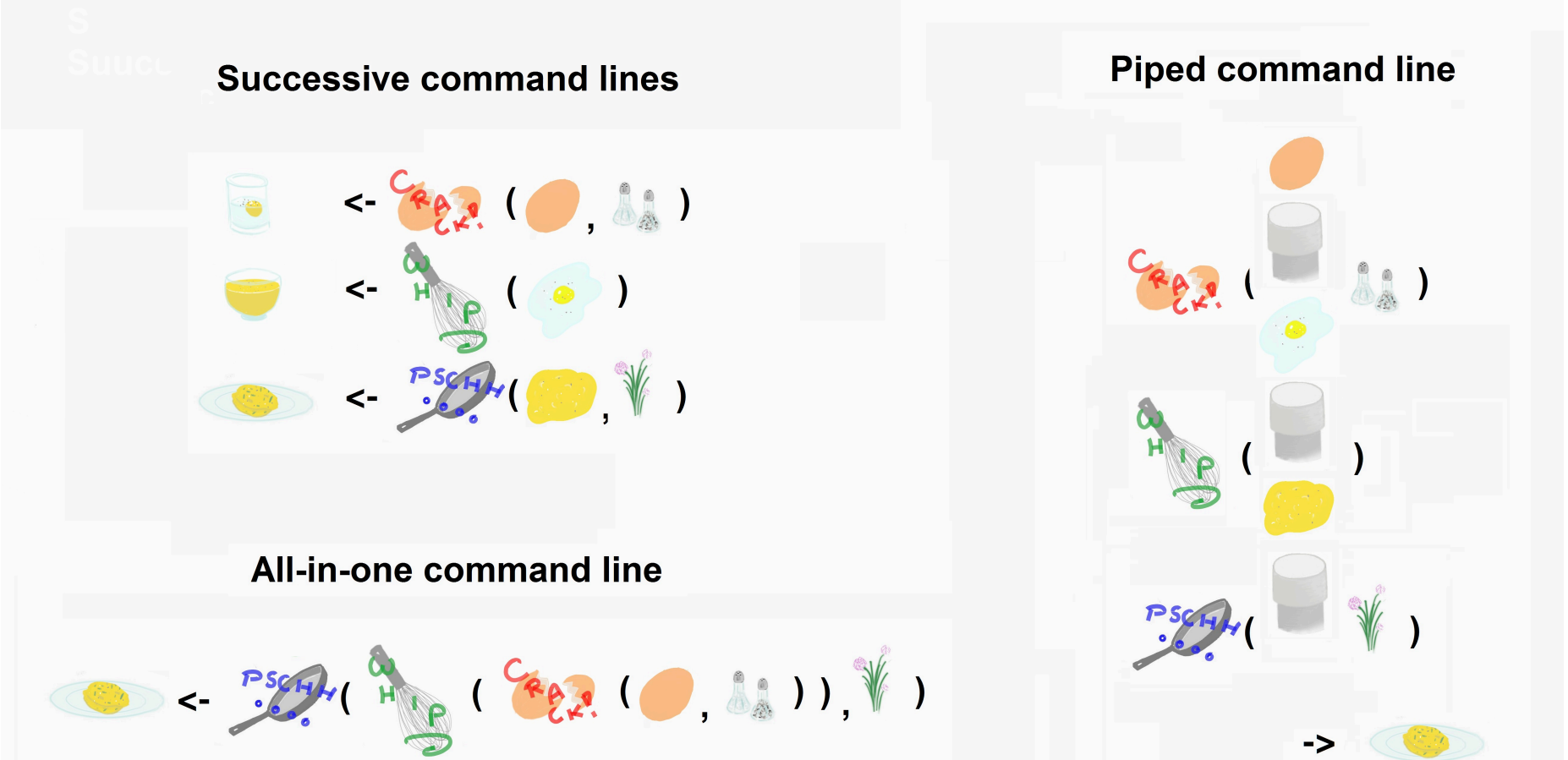
If you have experience with unix programming, you may be familiar with the version of the pipe there: `|`.

Starting with R version 4.1, R has its own pipe: `|>`

The original pipe function in R `%>%` has been part of the tidyverse for a while and is originally from the `magrittr` package, named after René Magritte

We use the pipe to chain together steps

It's like a recipe for our dataset.[1]



1. Example from Lise Vaudor

Instead of successive command lines

```
nlsy2 <- mutate(nlsy,
                only = case_when(nsibs == 0 ~ "yes",
                                 .default = "no"))
nlsy3 <- select(nlsy2, id, contains("sleep"), only)
only_kids <- filter(nlsy3, only == "yes")
```

or all-in-one

```
only_kids <- filter(select(mutate(nlsy,
only = case_when(nsibs == 0 ~ "yes",
.default = "no")), id,
contains("sleep"), only),
only == "yes")
```

It's like reading a story (or nursery rhyme!)

```
foo_foo <- little_bunny()
bop_on(
  scoop_up(
    hop_through(foo_foo, forest),
    field_mouse),
  head)
```

VS

```
foo_foo |>
  hop_through(forest) |>
  scoop_up(field_mouse) |>
  bop_on(head)
```

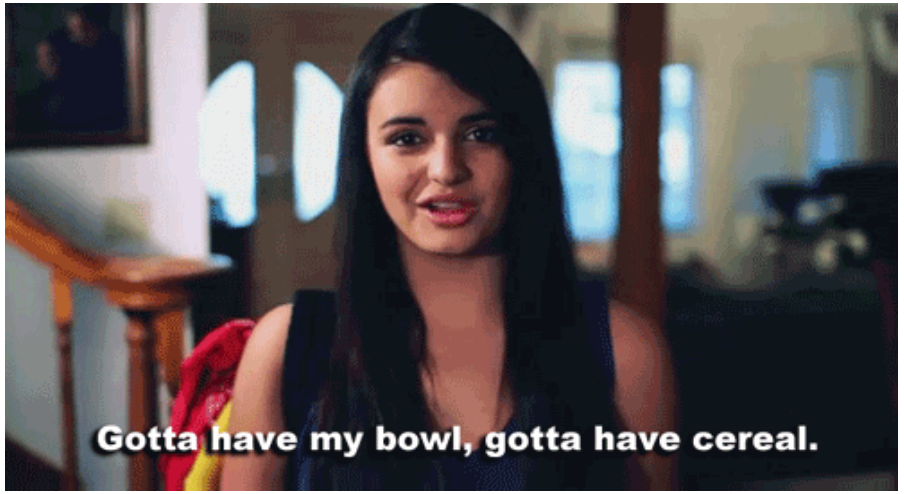
Example from Hadley Wickham

A natural order of operations

```
leave_house(
  get_dressed(
    get_out_of_bed(
      wake_up(me))))
```

```
me <- wake_up(me)
me <- get_out_of_bed(me)
me <- get_dressed(me)
me <- leave_house(me)
```

```
me  |>
  wake_up() |>
  get_out_of_bed() |>
  get_dressed() |>
  leave_house()
```



Example from Andrew Heiss

Using pipes with functions we already know

```r
nlsy2 <- mutate(nlsy, only = case_when(
  nsibs == 0 ~ "yes",
  .default = "no"))
nlsy3 <- select(nlsy2,
                id, contains("sleep"), only)
only_kids <- filter(nlsy3, only == "yes")
only_kids
```

```
# A tibble: 30 × 4
      id sleep_wkdy sleep_wknd only
   <dbl>      <dbl>      <dbl> <chr>
 1   458          7          8 yes
 2   653          6          7 yes
 3  1101          7          8 yes
 4  1166          5          6 yes
 5  2163          7          8 yes
 6  2442          7          9 yes
 7  2545          8          8 yes
 8  3036          5          8 yes
 9  3194          7          7 yes
10  3538          5          5 yes
# ℹ 20 more rows
```

```r
only_kids <- nlsy |>
  mutate(only = case_when(
    nsibs == 0 ~ "yes",
    TRUE ~ "no")) |>
  select(id, contains("sleep"), only) |>
  filter(only == "yes")
only_kids
```

```
# A tibble: 30 × 4
      id sleep_wkdy sleep_wknd only
   <dbl>      <dbl>      <dbl> <chr>
 1   458          7          8 yes
 2   653          6          7 yes
 3  1101          7          8 yes
 4  1166          5          6 yes
 5  2163          7          8 yes
 6  2442          7          9 yes
 7  2545          8          8 yes
 8  3036          5          8 yes
 9  3194          7          7 yes
10  3538          5          5 yes
# ℹ 20 more rows
```

Pipes fill first argument of the next function with what is being piped in

```
1 help(mutate)
2 help(select)
3 help(filter)
```

Usage

`mutate(.data, ...)`

`select(.data, ...)`

`filter(.data, ...)`

Pipes fill first argument of the next function with what is being piped in

```r
nlsy2 <- mutate(nlsy,
          only = case_when(
           nsibs == 0 ~ "yes",
           .default = "no"))
```

```r
only_kids <- nlsy  |>
  mutate(only = case_when(
    nsibs == 0 ~ "yes",
    .default = "no"))
```

Pipes fill first argument of the next function with what is being piped in

```r
nlsy2 <- mutate(nlsy,
          only = case_when(
           nsibs == 0 ~ "yes",
           .default = "no"))
nlsy3 <- select(nlsy2,
        id, contains("sleep"),
                  only)
only_kids <- filter(nlsy3,
          only == "yes")
```

```r
only_kids <- nlsy  |>
  mutate(only = case_when(
     nsibs == 0 ~ "yes",
     .default = "no")) |>
  select(id,
         contains("sleep"),
         only)  |>
  filter(only == "yes")
```

Exercises

Today's summary

- `select()` lets you choose the variables you want
- `filter()` lets you choose the rows you want
- `mutate()` lets you create new variables
- the pipe (`|>` or `%>%`) lets you chain these functions together

Additional functions

- `starts_with()`, `ends_with()`, `contains()`, `matches()`, `num_range()`, `all_of()`, `any_of()`, `everything()`, `where()`: helpers for `select()`
- `is.na()`, `is.finite()`, `is.infinite()`, `is.factor()`: helpers for `filter()`
- `%in%`: for multiple `==` or `!=` possibilities