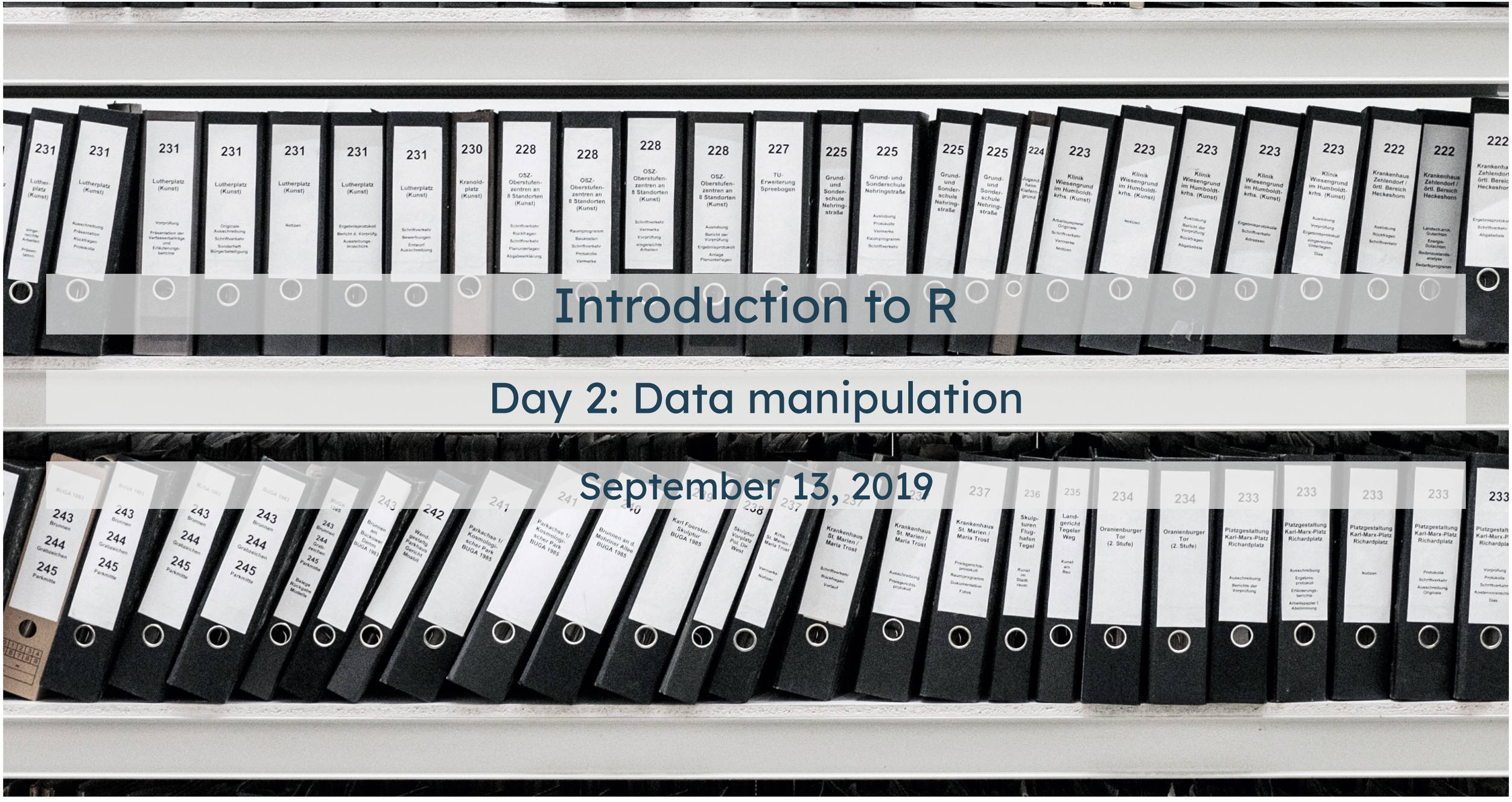


Introduction to R

Day 2: Data manipulation

September 13, 2019

Agenda

Day 1: Figures

Day 2: Selecting, filtering, and mutating

Day 3: Grouping and tables

Day 4: Functions

Day 5: Analyze your data

Agenda

Day 1: Figures

```
ggplot(data = {data}) +
      <geom>(aes(x = {xvar}, y = {yvar}, <characteristic> = {othvar}, ...),
             <characteristic> = "value", ...) +
      facet_<facettype>(vars({othvar})) +
      scale_<scalename>_<scaletype>(name = "name",
                                    <options> = c("options"),
                                    ...) +
      theme_<themename>()
```

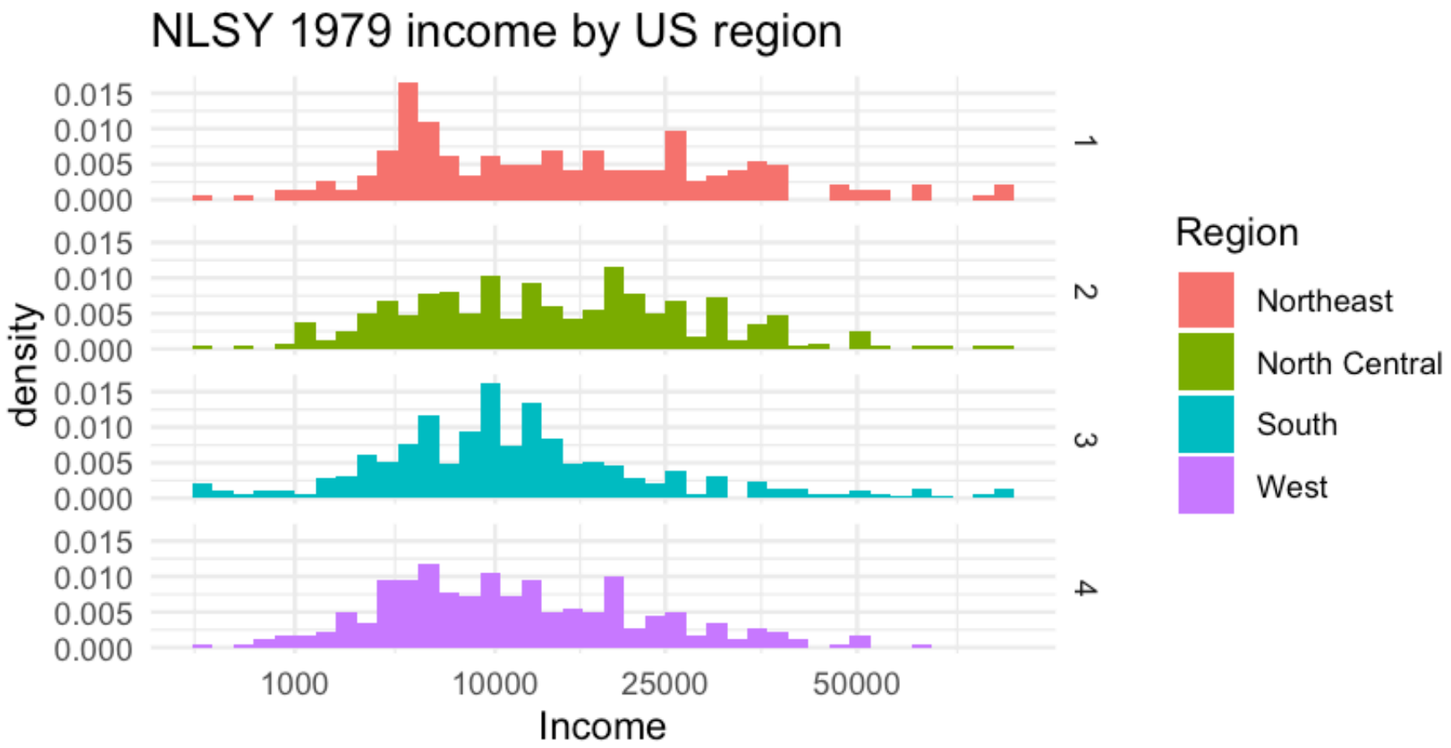
Agenda

Day 1: Figures

```
ggplot(data = nlsy) +
  geom_histogram(aes(x = income, y = ..density.., fill = factor(region)),
                 bins = 40) +
  scale_x_sqrt(breaks = c(1000, 10000, 25000, 50000)) +
  scale_fill_discrete(name = "Region",
                      labels = c("Northeast", "North Central", "South", "West")) +
  facet_grid(rows = vars(region)) +
  labs(x = "Income", title = "NLSY 1979 income by US region") +
  theme_minimal()
```

Agenda

Day 1: Figures



Agenda

Day 1: Figures

- We can build up figures using the `ggplot2` package by adding pieces using `+`
- These pieces follow a "grammar" which we use to map variables to the graph and specify colors, axes, etc.
- ggplot is not easy to master but with it you can do almost anything you want!
- There are lots of resources and examples online!

I don't think I've ever made a figure without Googling something! ("remove ggplot legend" is probably my most searched term ever)

I'm not alone!

> I remembered how to remove legend titles in ggplot without looking it up AMA
>
> — Katharine Egan (@katharine_egan) November 15, 2018

> I have easily googled “remove legend ggplot” 500+ times. It’s my R kryptonite. I’m surprised google chrome doesn’t just open at that page or, at least, shout at me to remember this time. #rstats #ggplot2. Anyone else have a similar blind spot for a frequently used piece of code?
>
> — Ben L (@snoylnimajneb) January 4, 2019

> How many times do you need to google 'how to remove ggplot legend' before unlocking the achievement? #rstats #ggplot
>
> — Luke Browne (@lukembrowne) January 15, 2019

> I google how to remove the legend title from a ggplot every time. I once committed to copying it to a post-it note and sticking it to my monitor, which I did. Then I lost the post-it and have now returned to my previous behavior.
>
> — Thomas J. Leeper (@thosjleeper) July 7, 2019

Agenda

Day 1: Figures ✅

Day 2: Selecting, filtering, and mutating

Day 3: Grouping and tables

Day 4: Functions

Day 5: Analyze your data

Agenda

Day 1: Figures ✅

Day 2: Selecting, filtering, and mutating

a.k.a How to manipulate your data to look like you want it to look (without making mistakes!)

Example

```
... scale_fill_discrete(name = "Region",
                        labels = c("Northeast",
                                   "North Central",
                                   "South",
                                   "West"))
```

Many of you asked: but how do we know what order they're in?

```
table(nlsy$region)
```

```
##
##   1   2   3   4
## 206 333 411 255
```

Labeling "factor" variables

- R's version of categorical variables are called factors
- The function to make them is just `factor()`, as we saw in our figures

```
summary(nlsy$region)
summary(factor(nlsy$region))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   2.593   3.000   4.000
```

```
##   1   2   3   4
## 206 333 411 255
```

We want to make the store **`region`** as a factor permanently (and, later, give it better names...)

Creating a new variable

```
nlsy$region_factor <- factor(nlsy$region)
```

We can make a new variable out of anything, not just factors

```
nlsy$age_bir_cent <- nlsy$age_bir - mean(nlsy$age_bir)
nlsy$dataset <- "NLSY"
```

```
## # A tibble: 1,205 x 5
##    region region_factor age_bir age_bir_cent dataset
##     <dbl> <fct>            <dbl>        <dbl> <chr>
## 1       1 1                   19       -4.45  NLSY
## 2       1 1                   30        6.55  NLSY
## 3       1 1                   17       -6.45  NLSY
## 4       1 1                   31        7.55  NLSY
## 5       3 3                   19       -4.45  NLSY
## 6       1 1                   30        6.55  NLSY
## 7       1 1                   27        3.55  NLSY
## 8       1 1                   24        0.552 NLSY
## # … with 1,197 more rows
```

Very quickly your code can get overrun with dollar signs (and parentheses, and arrows)

```
baseline$momusbirth <- factor(ifelse(baseline$momusbirth == "NEVER KNEW MOTHER", NA, as.character(baseline$momusbirth)))
baseline$dadusbirth <- factor(ifelse(baseline$dadusbirth == "NEVER KNEW FATHER", NA, as.character(baseline$dadusbirth)))
baseline$dadusbirth <- factor(ifelse(baseline$dadusbirth == "OTHER COUNTRY", "IN OTHER COUNTRY", as.character(baseline$dadusbirth)))
baseline$southchild <- factor(baseline$southchild, levels = c("NO", "YES"))
baseline$rev_degree <- factor(ifelse(is.na(baseline$rev_degree), "None", as.character(baseline$rev_degree)))
baseline$hs_dip <- factor(ifelse(is.na(baseline$hs_dip), "None", as.character(baseline$hs_dip)))
baseline$hs_dip <- factor(ifelse(baseline$hs_dip %in% c("GED1HS2", "HS1GED2"), "BOTH", as.character(baseline$hs_dip)))

baseline$dadedu4 <- cut(baseline$dadedu,
                        breaks = c(0, 11.9, 12, 16, 100),
                        right = T, include.lowest = T,
                        labels = c("< 12 years", "12 years", "12-16 years", ">= 16 years"))

baseline$momedu4 <- cut(baseline$momedu,
                        breaks = c(0, 11.9, 12, 16, 100),
                        right = T, include.lowest = T,
                        labels = c("< 12 years", "12 years", "12-16 years", ">= 16 years"))

baseline$momedu4 <- factor(ifelse(is.na(baseline$momedu4), "Missing", as.character(baseline$momedu4)))
baseline$dadedu4 <- factor(ifelse(is.na(baseline$dadedu4), "Missing", as.character(baseline$dadedu4)))
baseline$childhealth <- factor(ifelse(is.na(baseline$childhealth), "Missing", as.character(baseline$childhealth)))
baseline$parentallove <- factor(ifelse(is.na(baseline$parentallove), "Missing", as.character(baseline$parentallove)))
baseline$urbanchild <- droplevels(factor(baseline$urbanchild, labels =
                                         c("Urban", "Rural", "Rural")))
baseline$physicalabuse2 <- factor(baseline$physicalabuse2)
baseline$alcoholic <- factor(baseline$alcoholic)
```

Prettier way to make new variables: `mutate()`

```
nlsy <- mutate(nlsy,
               region_factor = factor(region),
               age_bir_cent = age_bir - mean(age_bir),
               dataset = "NLSY"
               )
```

We can refer to variables within the same dataset without the $ notation

`mutate()` tips and tricks

You still need to store your dataset somewhere, so make sure to include the assignment arrow

- Good practice to make new copies with different names as you go along
- R is smart about data storage, so it won't actually copy all of your data (i.e., you won't run out of room with 50 copies of almost identical datasets)
- You can refer immediately to variables you just made:

```
nlsy_new <- mutate(nlsy,
                   age_bir_cent = age_bir - mean(age_bir),
                   age_bir_stand = age_bir_cent / sd(age_bir_cent)
                   )
```

My favorite R function: `case_when()`

I used to write endless strings of `ifelse()` statements

If A is TRUE, then B; if not, then if C is true, then D; if not, then if E is true, then F; if not,

Are you confused yet?

```
all_data$marstat <- factor(ifelse(all_data$marstat %in% c("Divorced", "DIVORCED"), "Divorced",
                   ifelse(all_data$marstat %in% c("Never Married", "NEVER MARRIED"), "Never Married",
                          ifelse(all_data$marstat %in% c("Separated", "SEPARATED"), "Separated",
                                 ifelse(all_data$marstat %in% c("Married", "MARRIED"), "Married",
                                        ifelse(all_data$marstat %in% c("Widowed", "WIDOWED"), "Widowed", NA))))))
```

case_when()

```
nlsy <- mutate(nlsy, slp_cat_wkdy = case_when(
                                                sleep_wkdy < 5 ~ "little",
                                                sleep_wkdy < 7 ~ "some",
                                                sleep_wkdy < 9 ~ "ideal",
                                                sleep_wkdy < 12 ~ "lots",
                                                TRUE ~ NA_character_
                                                )
               )

# note that table doesn't show NAs! can be dangerous!
table(nlsy$slp_cat_wkdy, nlsy$sleep_wkdy)
```

```
##
##             0   2   3   4   5   6   7   8   9  10  11  12  13
##   ideal     0   0   0   0   0   0 357 269   0   0   0   0   0
##   little    1   4  14  48   0   0   0   0   0   0   0   0   0
##   lots      0   0   0   0   0   0   0   0  32  14   1   0   0
##   some      0   0   0   0 136 326   0   0   0   0   0   0   0
```

`case_when()`

Syntax

- Ask a question (i.e., something that will give `TRUE` or `FALSE`) on the left-hand side of the `~`
- If `TRUE`, variable will take on value of whatever is on the right-hand side of the `~`
- Proceeds in order ... if TRUE, takes that value and stops
- If you want some default value, you can end with `TRUE ~ {something}`, which every observation will get if everything else is `FALSE`
- Must make everything the same type, including missing values (`NA_character_`, `NA_real_` generally)

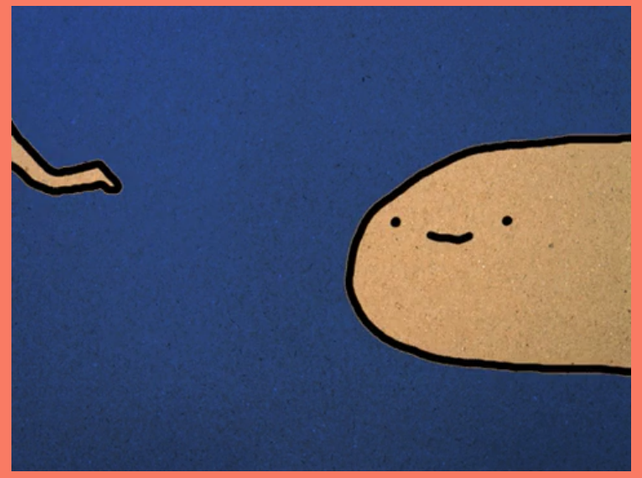
`case_when()`

Example:

```
nlsy <- mutate(nlsy, total_sleep = case_when(
                                              sleep_wknd > 8 & sleep_wkdy > 8 ~ 1,
                                              sleep_wknd + sleep_wkdy > 15 ~ 2,
                                              sleep_wknd - sleep_wkdy > 3 ~ 3,
                                              TRUE ~ NA_real_
                                              )
                )
```

- Which value would someone with `sleep_wknd = 8` and `sleep_wkdy = 4` go?
- What about someone with `sleep_wknd = 11` and `sleep_wkdy = 4`?
- What about someone with `sleep_wknd = 7` and `sleep_wkdy = 7`?

Exercises 1



1. Using the NLSY data and `mutate()`, make a standardized (centered at the mean, and divided by the standard deviation) version of income.
2. Do the same thing, but using income on the log scale. Look at this variable using `summary()`. Can you figure out what happened? (Hint: look at log(income).)
3. Redo question 2, but if you are not able to calculate log(income) for an observation, replace it with a missing value (using `case_when()`). This time, when you standardize log(income), you'll have to use `na.rm = TRUE` to remove missing values both when you take the mean and the standard deviation.

OK, but what about those factors?!

Let's look at the variable we made describing someone's weekday sleeping habits:

```
 nlsy <- mutate(nlsy, slp_cat_wkdy = case_when(
                                                 sleep_wkdy < 5 ~ "little",
                                                 sleep_wkdy < 7 ~ "some",
                                                 sleep_wkdy < 9 ~ "ideal",
                                                 sleep_wkdy < 12 ~ "lots",
                                                 TRUE ~ NA_character_
                                                 )
                 )

 summary(nlsy$slp_cat_wkdy)
```

```
##    Length     Class      Mode
##      1205 character character
```

Character variables aren't very helpful in analysis

Like the {1, 2, 3, 4} region variable, we want to turn this variable into a categorical variable. This time it already comes with names!

```
# I'm just going to replace this variable, instead of making a new one,
# by giving it the same name a before
nlsy <- mutate(nlsy, slp_cat_wkdy = factor(slp_cat_wkdy))
summary(nlsy$slp_cat_wkdy)
```

```
##  ideal little   lots   some   NA's
##    626     67     47    462      3
```

Much better, but what's the deal with that order?

`forcats` package

- Tries to make working with factors safe and convenient
- Functions to make new levels, reorder levels, combine levels, etc.
- All the functions start with `fct_` so they're easy to find using tab-complete!
- Automatically loads with `library(tidyverse)`

Reorder factors

The `fct_relevel()` function allows us just to rewrite the names of the categories out in the order we want them (safely).

```
nlsy <- mutate(nlsy, slp_cat_wkdy_ord = fct_relevel(slp_cat_wkdy, "little",
                                                                  "some",
                                                                  "ideal",
                                                                  "lots"
                                                    )
               )
```

```
summary(nlsy$slp_cat_wkdy_ord)
```

```
## little   some  ideal   lots   NA's
##     67    462    626     47      3
```

```
levels(nlsy$slp_cat_wkdy_ord)
```

```
## [1] "little" "some"   "ideal"  "lots"
```

What if you misspell something?

```
 nlsy <- mutate(nlsy, slp_cat_wkdy_ord2 = fct_relevel(slp_cat_wkdy, "little",
                                                                    "same",
                                                                     "ideal",
                                                                     "lots"
                                                      )
                )
```

```
## Warning: Unknown levels in f: same
```

```
 summary(nlsy$slp_cat_wkdy_ord2)
```

```
## little  ideal   lots   some   NA's
##     67    626     47    462      3
```

```
 levels(nlsy$slp_cat_wkdy_ord2)
```

```
## [1] "little" "ideal"  "lots"   "some"
```

You get a warning, and levels you didn't mention are pushed to the end.

Other orders

While amount of sleep has an inherent ordering, region doesn't. Also, we still need to give the numbers names!

From the codebook, I know that:

```
nlsy <- mutate(nlsy, region_fact = factor(region),
                     region_fact = fct_recode(region_fact,
                                              "Northeast" = "1",
                                              "North Central" = "2",
                                              "South" = "3",
                                              "West" = "4"))
```

```
table(nlsy$region)
```

```
##
##   1   2   3   4
## 206 333 411 255
```

```
summary(nlsy$region_fact) # since table() doesn't show NAs
```

```
##     Northeast North Central         South          West
##           206           333           411           255
```

Other orders

So now I can reorder them as I wish -- how about from most people to least?

```
nlsy <- mutate(nlsy, region_fact = fct_infreq(region_fact))
summary(nlsy$region_fact)
```

```
##         South North Central          West     Northeast
##           411           333           255           206
```

Or the reverse of that?

```
nlsy <- mutate(nlsy, region_fact = fct_rev(region_fact))
summary(nlsy$region_fact)
```

```
##     Northeast          West North Central         South
##           206           255           333           411
```

Add and remove

Recall that we made it so that the sleep variable had missing values, perhaps because we thought they were outliers:

```
nlsy <- mutate(nlsy, slp_cat_wkdy_out =
                 fct_explicit_na(slp_cat_wkdy, na_level = "outlier"))
summary(nlsy$slp_cat_wkdy_out)
```

```
##   ideal  little    lots    some outlier
##     626      67      47     462       3
```

Or maybe we want to combine some levels that don't have a lot of observations in them:

```
nlsy <- mutate(nlsy, slp_cat_wkdy_comb = fct_collapse(slp_cat_wkdy,
                                                      "less" = c("little", "some"),
                                                      "more" = c("ideal", "lots")))
summary(nlsy$slp_cat_wkdy_comb)
```

```
## more less NA's
##  673  529    3
```

Add and remove

Or we can have R choose which ones to combine based on how few observations they have:

```
nlsy <- mutate(nlsy, slp_cat_wkdy_lump = fct_lump(slp_cat_wkdy, n = 2))
summary(nlsy$slp_cat_wkdy_lump)
```

```
## ideal  some Other  NA's
##   626   462   114     3
```

There are 25 **fct_** functions in the package. The sky's the limit when it comes to manipulating your categorical variables in R!

Exercises 2

1. Turn the eyesight variable into a factor variable. The numbers 1-5 correspond to excellent, very good, good, fair, and poor. Make sure that categories are in an appropriate order.
2. Use two different methods to combine the worst two categories of eyesight into one category.
3. Make a new categorical income variable with at least 3 levels (you can choose the cutoffs). Make a bar graph with this new variable where the bars are in the correct order from low to high and are colored increasingly dark shades of green. (Hint: http://colorbrewer2.org; `scale_color_brewer()`)

Selecting the variables you want

We've made approximately 1000 new variables!

You don't want to keep them all. You'll get confused, and when you go to summarize your data it will take pages.

Luckily there's an easy way to select the variables you want: `select()`!

```
nlsy_subs <- select(nlsy, id, income, eyesight, sex, region)
nlsy_subs
```

```
## # A tibble: 1,205 x 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
## # … with 1,199 more rows
```

`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 minus sign (`-`)
- There are also a lot of "helpers"!

```
select(nlsy_subs, -id, -region)
```

```
## # A tibble: 1,205 x 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
## # … with 1,196 more rows
```

one_of()

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

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

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

```
## # A tibble: 1,205 x 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
## # … with 1,196 more rows
```

Other select helpers

Do you have a lot of variables that are alike in some way? And you want to find all of them? Try:

- `starts_with()`
- `contains()`
- `ends_with()`

```
select(nlsy, starts_with("slp"))
```

```
## # A tibble: 1,205 x 6
##   slp_cat_wkdy slp_cat_wkdy_ord slp_cat_wkdy_or… slp_cat_wkdy_out slp_cat_wkdy_co…
##   <fct>        <fct>            <fct>             <fct>            <fct>
## 1 some         some             some              some             less
## 2 some         some             some              some             less
## 3 ideal        ideal            ideal             ideal            more
## 4 some         some             some              some             less
## 5 lots         lots             lots              lots             more
## 6 ideal        ideal            ideal             ideal            more
## 7 ideal        ideal            ideal             ideal            more
## # … with 1,198 more rows, and 1 more variable: slp_cat_wkdy_lump <fct>
```

Reordering variables

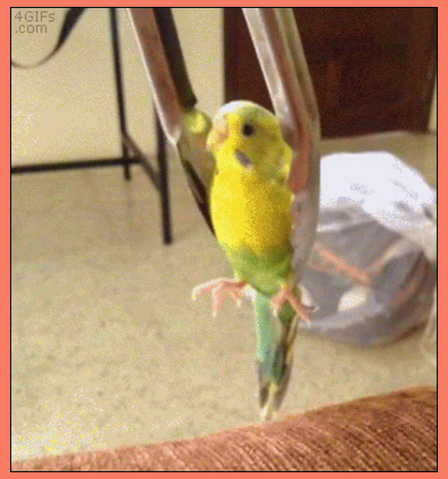
Sometimes you don't want to get rid of the other variables, you just want to move things around. Then use `everything()` as the last argument in `select()` to get all the rest.

Let's move `id` to be the first column:

```
select(nlsy, id, everything())
```

```
## # A tibble: 1,205 x 25
##       id glasses eyesight sleep_wkdy sleep_wknd nsibs  samp race_eth   sex region income
##    <dbl>   <dbl>    <dbl>      <dbl>      <dbl> <dbl> <dbl>    <dbl> <dbl>  <dbl>  <dbl>
## 1     3       0        1          5          7     3     5        3     2      1  22390
## 2     6       1        2          6          7     1     1        3     1      1  35000
## 3     8       0        2          7          9     7     6        3     2      1   7227
## 4    16       1        3          6          7     3     5        3     2      1  48000
## 5    18       0        3         10         10     2     1        3     1      3   4510
## 6    20       1        2          7          8     2     5        3     2      1  50000
## # … with 1,199 more rows, and 14 more variables: res_1980 <dbl>, res_2002 <dbl>,
## #   age_bir <dbl>, region_factor <fct>, age_bir_cent <dbl>, dataset <chr>,
## #   slp_cat_wkdy <fct>, total_sleep <dbl>, slp_cat_wkdy_ord <fct>, slp_cat_wkdy_ord2 <fct>,
## #   region_fact <fct>, slp_cat_wkdy_out <fct>, slp_cat_wkdy_comb <fct>,
## #   slp_cat_wkdy_lump <fct>
```

Exercises 3



1. Create mean-centered versions of "age_bir", "nsibs", "income", and the two sleep variables. Use the same ending (e.g., "_cent") for all of them. Then make a new dataset of just the centered variables using `select()` and a helper.
2. You may have added a lot of variables to the original dataset by now. Create a dataset called `nlsy_orig` that contains only the variables we started off with, using the vector of names we originally used to name the columns and the `one_of()` helper.
3. Look at `help(select)`. You'll notice that `rename()` is a related function. Looking at the examples to help, rename "age_bir" to "age_1st_birth" without making a new column.

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()`.

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

 nrow(wear_glasses)
```

```
## [1] 624
```

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

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

`filter()` syntax

- Like the others, 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.

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

nrow(yesno_glasses)
```

```
## [1] 0
```

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

nrow(glasses_great_eyes)
```

```
## [1] 254
```

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).

Or statements

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

```
extreme_eyes <- filter(nlsy, eyesight == 1 | eyesight == 5)
table(extreme_eyes$eyesight)
```

```
##
##   1   5
## 474  19
```

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

```
some_regions <- filter(nlsy, region_fact == "Northeast" | region_fact == "South")
table(some_regions$region_fact)
```

```
##
##     Northeast          West North Central         South
##           206             0             0           411
```

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_fact %in% c("South", "West", "Northeast"))
table(more_regions$region_fact)
```

```
##
##     Northeast          West North Central         South
##           206           255             0           411
```

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

More `%in%`

This function works outside of the `filter()` function, of course!

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

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

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

```
## [1] FALSE
```

Opposite of `%in%`

This is annoying. We can't say "not in" with the syntax `%!in%` or something like that. We have to put the `!` before the question to basically 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_fact %in% c("South", "West", "Northeast"))
 table(northcentralers$region_fact)
```

```
##
##     Northeast          West North Central         South
##             0             0           333             0
```

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

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

summary(select(my_data, age_bir_cent, sex, nsibs, slp_cat_wkdy))
```

```
##   age_bir_cent          sex        nsibs       slp_cat_wkdy
##  Min.   :-9.4481   Min.   :2   Min.   :1.000   ideal :109
##  1st Qu.:-5.4481   1st Qu.:2   1st Qu.:2.000   little: 14
##  Median :-4.4481   Median :2   Median :2.000   lots  :  6
##  Mean   :-3.8249   Mean   :2   Mean   :2.174   some  : 78
##  3rd Qu.:-1.4481   3rd Qu.:2   3rd Qu.:3.000
##  Max.   : 0.5519   Max.   :2   Max.   :3.000
```

Putting it all together

```
oth_dat <- filter(nlsy,
                  (age_bir_cent < 1) &
                  (sex != 1 | nsibs %in% c(1, 2, 3)) &
                  !is.na(slp_cat_wkdy))

summary(select(oth_dat, age_bir_cent, sex, nsibs, slp_cat_wkdy))
```

```
##   age_bir_cent           sex            nsibs        slp_cat_wkdy
##  Min.   :-10.4481   Min.   :1.000   Min.   : 0.000   ideal :306
##  1st Qu.: -6.4481   1st Qu.:2.000   1st Qu.: 2.000   little: 40
##  Median : -3.4481   Median :2.000   Median : 3.000   lots  : 26
##  Mean   : -3.8518   Mean   :1.817   Mean   : 3.982   some  :230
##  3rd Qu.: -1.4481   3rd Qu.:2.000   3rd Qu.: 5.000
##  Max.   :  0.5519   Max.   :2.000   Max.   :16.000
```

Exercises 4

1. Create a dataset with all the observations that get over 7 hours of sleep on both weekends and weekdays *or* who have an income greater than/equal to 20,000 and less than/equal to 50,000.
2. Create a dataset that consists *only* of the missing values in `slp_cat_wkdy`. Check how many rows it has (there should be 3!).
3. Look up the `between()` function in help. Figure out how to use this to answer question 1, when choosing people whose income is between 20,000 and 50,000. Check to make sure you get the same number of rows.

Challenge

Deal with the disaster that are the residence categories across NLSY years!

Sometimes when a study is conducted across many years, the questions and/or possible answers change slightly. This is **really annoying**. R to the rescue!

```
##                          res_1980                                  res_2002
##  OWN DWELLING UNIT          :3198   OWN DWELLING UNIT                   :7057
##  DORM, FRATERNITY, SORORITY: 493   RESPONDENT IN PARENT HOUSEHOLD      : 382
##  ABOARD SHIP, BARRACKS      : 432   JAIL                                : 110
##  BACHELOR, OFFICER QUARTERS: 113   OTHER TEMPORARY INDIVIDUAL QUARTERS:  91
##  ON-BASE MIL FAM HOUSING    :  70   OTHER INDIVIDUAL QUARTERS           :  57
##  (Other)                    : 194   (Other)                             :  27
##  NA's                       :8186   NA's                                :4962
```

Challenge

```
 levels(nlsy_full$res_1980)
```

```
##  [1] "ABOARD SHIP, BARRACKS"      "BACHELOR, OFFICER QUARTERS" "DORM, FRATERNITY, SORORITY"
##  [4] "HOSPITAL"                   "JAIL"                       "OTHER TEMPORARY QUARTERS"
##  [7] "OWN DWELLING UNIT"          "ON-BASE MIL FAM HOUSING"    "OFF-BASE MIL FAM HOUSING"
## [10] "ORPHANAGE"                  "RELIGIOUS INSTITUTION"      "OTHER INDIVIDUAL QUARTERS"
## [13] "PARENTAL"                   "HHI CONDUCTED WITH PARENT"  "R IN PARENTAL HOUSEHOLD"
```

```
 levels(nlsy_full$res_2002)
```

```
##  [1] "OPEN BAY OR TROOP BARRACKS, ABOARD SHIP"
##  [2] "BACHELOR ENLISTED OR OFFICER QUARTERS"
##  [3] "DORMITORY, FRATERNITY OR SORORITY"
##  [4] "HOSPITAL"
##  [5] "JAIL"
##  [6] "OTHER TEMPORARY INDIVIDUAL QUARTERS"
##  [7] "OWN DWELLING UNIT"
##  [8] "ON-BASE MILITARY FAMILY HOUSING"
##  [9] "OFF-BASE MILITARY FAMILY HOUSING"
## [10] "CONVENT, MONASTERY, OTHER RELIGIOUS INSTITUTE"
## [11] "OTHER INDIVIDUAL QUARTERS"
## [12] "RESPONDENT IN PARENT HOUSEHOLD"
```

res_1980

value	label
1	ABOARD SHIP, BARRACKS
2	BACHELOR, OFFICER QUARTERS
3	DORM, FRATERNITY, SORORITY
4	HOSPITAL
5	JAIL
6	OTHER TEMPORARY QUARTERS
11	OWN DWELLING UNIT
12	ON-BASE MIL FAM HOUSING
13	OFF-BASE MIL FAM HOUSING
14	ORPHANAGE
15	RELIGIOUS INSTITUTION
16	OTHER INDIVIDUAL QUARTERS
17	PARENTAL
18	HHI CONDUCTED WITH PARENT
19	R IN PARENTAL HOUSEHOLD

res_2002

value	label
1	OPEN BAY OR TROOP BARRACKS, ABOARD SHIP
2	BACHELOR ENLISTED OR OFFICER QUARTERS
3	DORMITORY, FRATERNITY OR SORORITY
5	JAIL
4	HOSPITAL
6	OTHER TEMPORARY INDIVIDUAL QUARTERS (SPECIFY)
11	OWN DWELLING UNIT
12	ON-BASE MILITARY FAMILY HOUSING
13	OFF-BASE MILITARY FAMILY HOUSING
15	CONVENT, MONASTERY, OTHER RELIGIOUS INSTITUTE
16	OTHER INDIVIDUAL QUARTERS (SPECIFY)

Challenge

We'll eventually want to be able to work with these two factor variables together, so we want them to have the same levels.

Your job is to do your best to make each of them into a variable you would like to work with if you were analyzing this data. This may involve combining categories, changing names, etc.

Then make a dataset with only your two better versions of these variables. Only include observations that have a non-missing observation in both years.