Introduction to R

Week 3: Data manipulation

Louisa Smith

July 27 - July 31

Let's

wrangle

our data

Making variables in "Base R"

```
nlsy$region_factor <- factor(nlsy$region)
nlsy$income <- round(nlsy$income)
nlsy$age_bir_cent <- nlsy$age_bir - mean(nlsy$age_bir)
nlsy$index <- 1:nrow(nlsy)
nlsy$slp_wkdy_cat <- ifelse(nlsy$sleep_wkdy < 5, "little",
                            ifelse(nlsy$sleep_wkdy < 7, "some",
                                   ifelse(nlsy$sleep_wkdy < 9, "ideal",
                                          ifelse(nlsy$sleep_wkdy < 12, "lots"
                                          )
                                   )
                            )
```

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)
baseline$mentallyill <- factor(baseline$mentallyill)

full_data$momusbirth <- factor(ifelse(full_data$momusbirth == "NEVER KNEW MOTHER", NA, as.character(full_data$momusbirth)))
full_data$dadusbirth <- factor(ifelse(full_data$dadusbirth == "NEVER KNEW FATHER", NA, as.character(full_data$dadusbirth)))
```

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

```
library(tidyverse)
# mutate() is from dplyr
nlsy <- mutate(nlsy, # dataset
               region_factor = factor(region), # new variables
               income = round(income),
               age_bir_cent = age_bir - mean(age_bir),
               index = row_number()
               # could make as many as we want....
               )
```

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, ...

```
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))))))
```

> Are you confused yet?!

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_ # >= 12
                                              )
                )
# 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 `~`
  - `sleep_wkdy < 5`
- If `TRUE`, variable will take on value of whatever is on the right-hand side of the `~`
  - `~ "little"`
- 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`
  - `TRUE ~ NA_character_`
- 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`?

1

Your turn...

Exercises 3.1: Make some new variables!

What about 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

If the values are the desired labels, it's pretty straightforward: just use `factor()`

```
# 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
```

What if you misspell something?

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

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

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

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

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, the region variable is numeric, not a character!

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"))

summary(nlsy$region_fact)
```

```
##     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 levels

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
```

Remove levels

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
```

- Probably not a good idea for factors with in inherent order

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

2

Your turn...

Exercises 3.2: Make some new factors!

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 an exclamation point (`!`) or a minus sign (`-`)
- There are also a lot of "helpers"!

```
select(nlsy_subs, !c(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
## # … with 1,200 more rows
```

`all_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 `all_of()` to choose them.

```
cols_I_want <- c("age_bir", "nsibs", "region")
select(nlsy, all_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
## # … with 1,200 more rows
```

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

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()`
- `ends_with()`
- `contains()`
- `matches()` (like contains, but for regular expressions)
- `num_range()` (for patterns like `x01`, `x02`, ...)

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

```
## # A tibble: 1,205 x 7
##   slp_wkdy_cat slp_cat_wkdy slp_cat_wkdy_ord slp_cat_wkdy_or… slp_cat_wkdy_out
##   <chr>        <fct>        <fct>             <fct>             <fct>
## 1 some         some         some              some              some
## 2 some         some         some              some              some
## # … with 1,203 more rows, and 2 more variables: slp_cat_wkdy_comb <fct>,
## #   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 `relocate()`:

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

```
nlsy <- relocate(nlsy, id)
nlsy
```

```
## # A tibble: 1,205 x 26
##       id glasses eyesight sleep_wkdy sleep_wknd nsibs  samp race_eth   sex region
##    <dbl>   <dbl>    <dbl>      <dbl>      <dbl> <dbl> <dbl>    <dbl> <dbl>  <dbl>
## 1     3       0        1          5          7     3     5        3     2      1
## 2     6       1        2          6          7     1     1        3     1      1
## 3     8       0        2          7          9     7     6        3     2      1
## 4    16       1        3          6          7     3     5        3     2      1
## # … with 1,201 more rows, and 15 more variables: res_1980 <dbl>, res_2002 <dbl>,
## #   age_bir <dbl>, region_factor <fct>, age_bir_cent <dbl>, index <int>, slp_wkdy
## #   slp_cat_wkdy <fct>, total_sleep <dbl>, slp_cat_wkdy_ord <fct>, slp_cat_wkdy_c
## #   region_fact <fct>, slp_cat_wkdy_out <fct>, slp_cat_wkdy_comb <fct>,
```

Reordering variables

You can relocate multiple variables to the beginning, or specify where they should show up

Let's move `sex` and `region` to be after `id`:

```
relocate(nlsy, sex, region, .after = id)
```

```
## # A tibble: 1,205 x 26
##       id   sex region glasses eyesight sleep_wkdy sleep_wknd nsibs  samp race_eth
##    <dbl> <dbl>  <dbl>   <dbl>    <dbl>      <dbl>      <dbl> <dbl> <dbl>    <dbl>
## 1     3     2      1       0        1          5          7     3     5        3
## 2     6     1      1       1        2          6          7     1     1        3
## 3     8     2      1       0        2          7          9     7     6        3
## 4    16     2      1       1        3          6          7     3     5        3
## 5    18     1      3       0        3         10         10     2     1        3
## # … with 1,200 more rows, and 15 more variables: res_1980 <dbl>, res_2002 <dbl>,
## #   age_bir <dbl>, region_factor <fct>, age_bir_cent <dbl>, index <int>, slp_wkdy
## #   slp_cat_wkdy <fct>, total_sleep <dbl>, slp_cat_wkdy_ord <fct>, slp_cat_wkdy_c
## #   region_fact <fct>, slp_cat_wkdy_out <fct>, slp_cat_wkdy_comb <fct>,
## #   slp_cat_wkdy_lump <fct>
```

Select variables to do something to them

This can get a bit confusing (and "best practices" have recently changed), so we won't go into details, but you can put all these tools together:

```
nlsy <- mutate(rowwise(nlsy),
               sleep_avg = mean(c_across(starts_with("sleep"))))
select(nlsy, starts_with("sleep"))
```

```
## # A tibble: 1,205 x 3
##   sleep_wkdy sleep_wknd sleep_avg
##        <dbl>      <dbl>     <dbl>
## 1          5          7       6
## 2          6          7       6.5
## 3          7          9       8
## 4          6          7       6.5
## 5         10         10      10
## 6          7          8       7.5
## # … with 1,199 more rows
```

Select variables to do something to them

```
nlsy <- nlsy %>%
  mutate(across(starts_with("sleep"), ~ .x * 60, .names = "{col}_mins"))
select(nlsy, starts_with("sleep"))
```

```
## # A tibble: 1,205 x 6
##   sleep_wkdy sleep_wknd sleep_avg sleep_wkdy_mins sleep_wknd_mins sleep_avg_mins
##        <dbl>      <dbl>     <dbl>           <dbl>           <dbl>          <dbl>
## 1          5          7         6             300             420            360
## 2          6          7       6.5             360             420            390
## 3          7          9         8             420             540            480
## 4          6          7       6.5             360             420            390
## 5         10         10        10             600             600            600
## 6          7          8       7.5             420             480            450
## # … with 1,199 more rows
```

3

Your turn...

Exercises 3.3: Create some new datasets!

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 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.

```
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 is just a regular R function that 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%`

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
```

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
```

Resources

- Here’s an entire paper about working with factors in R.
- `forcats` cheat sheet
- Here's more info on the `select()` helpers
- Here are some useful functions for use with `mutate()`

4

Your turn...

Exercises 3.4: Create some new datasets!