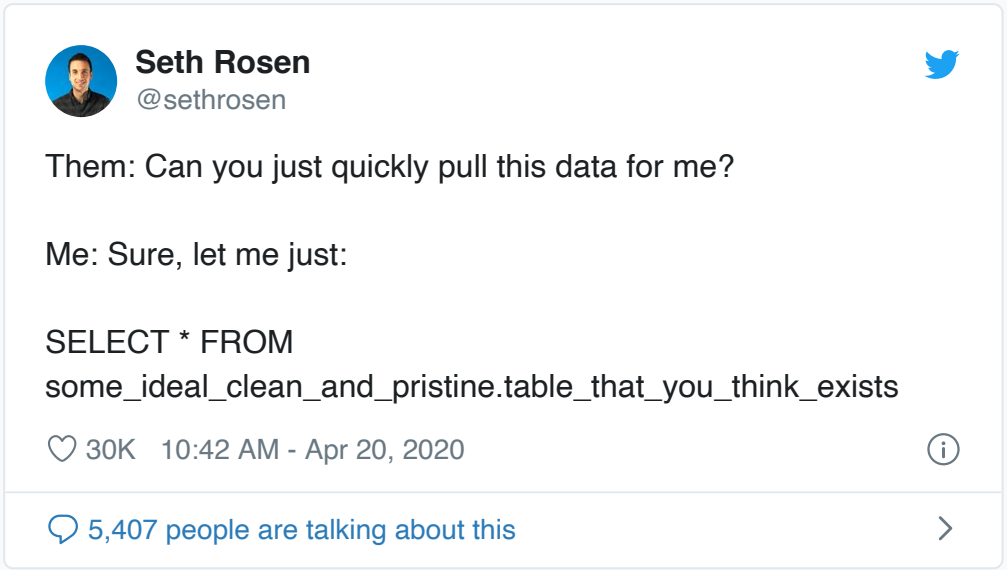
Data Wrangling in R

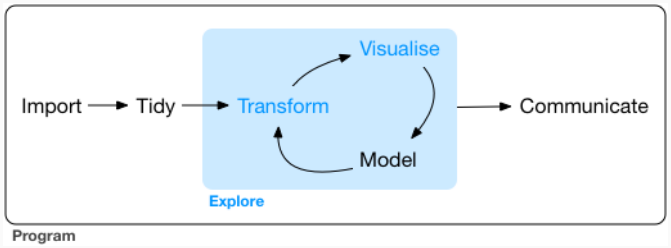
Connor Gilroy
2020-05-01

Data wrangling

What do we mean by *data wrangling*? (Or "data munging", or even "data janitoring"?)



Data wrangling is **everything you have to do to a data set to get it ready for analysis**



How to wrangle data in R

There's more than one way to do it! R has two main **dialects**:

#1. **base** R

```
head(data.frame(giving)[(giving$`2016_dollars`<0 & !is.na(giving$`2016_dollars`)),
                        c("transaction_amt","memo_text")], 100)
```

#2. the **tidyverse**

```
giving %>%
  filter(`2016_dollars` < 0, !is.na(`2016_dollars`)) %>%
  select(transaction_amt, memo_text) %>%
  head(100)
```

Many people mix the two. I'll teach a more pure tidyverse style.

Note: the tidyverse evolves over time. Use current packages that are well-established. Steer clear of retired packages (like **plyr**) or ones that are especially new (unless you need something cutting-edge)

Key concepts

tidyverse-style data transformation has 3 components

1. data (a data frame object)
2. verbs (functions)
3. the pipe (`%>%`)

```
# this
some_data %>%
  do_something()

# is the same as this
do_something(some_data)
```

Why bother with the pipe? You can combine steps into a *series* of transformations

Remember, if you want to keep something, you also have to give it a name:

```
some_new_data <-
  some_data %>%
  do_something() %>%
  do_something_else()
```

Simple **dplyr** verbs

`select()` - choose columns (variables) by name

```
gss %>% select(id, sex, race)

# remove columns with -name
gss %>% select(-year)

# there are select helpers too
gss %>% select(starts_with("vote"))
```

`filter()` - choose rows (observations) by some characteristics

```
gss %>% filter(age < 50)

# "a equals b" is `a == b`
gss %>% filter(race == "BLACK")

# `!` means "not"
gss %>% filter(!is.na(divorce))
```

`arrange()` - put data in order by one or more variables

```
gss %>% arrange(age, race)
```

Simple **dplyr** verbs

`select()` - choose columns (variables) by name

```
gss %>% select(id, sex, race)
```

```
## # A tibble: 4,761 x 3
##    id    sex    race
##    <fct> <fct>  <fct>
##  1 9     FEMALE BLACK
##  2 3001  FEMALE OTHER
##  3 6001  FEMALE BLACK
##  4 10    FEMALE OTHER
##  5 3002  FEMALE OTHER
##  6 6002  FEMALE WHITE
##  7 11    FEMALE BLACK
##  8 3003  FEMALE BLACK
##  9 6003  FEMALE BLACK
## 10 12    MALE   BLACK
## # … with 4,751 more rows
```

Simple **dplyr** verbs

`filter()` - choose rows (observations) by some characteristics

```
gss %>% filter(race == "BLACK")
```

```
## # A tibble: 665 x 15
##    firstid  wave year  id      age sex   race  divorce income income06 rincome vote00 vo
##    <fct>   <dbl> <fct> <fct> <dbl> <fct> <fct> <fct>   <fct>  <fct>    <fct>   <fct>  <f
##  1 9           1 2006  9        23 FEMA… BLACK <NA>    $2500… $40000 … $25000… INELI… VO
##  2 9           3 2010  6001     27 FEMA… BLACK NO      $2500… $60000 … $25000… IAP    VO
##  3 11          1 2006  11       81 FEMA… BLACK YES     <NA>   <NA>     <NA>    VOTED  VO
##  4 11          2 2008  3003     83 FEMA… BLACK YES     $2000… $20000 … <NA>    IAP    VO
##  5 11          3 2010  6003     85 FEMA… BLACK NO      <NA>   <NA>     <NA>    IAP    DI
##  6 12          1 2006  12       47 MALE  BLACK <NA>    <NA>   <NA>     <NA>    DID N… DI
##  7 12          2 2008  3004     49 MALE  BLACK <NA>    $1000… $12500 … <NA>    IAP    DI
##  8 13          1 2006  13       26 MALE  BLACK <NA>    $8000… $8 000 … <NA>    VOTED  VO
##  9 13          2 2008  3005     28 MALE  BLACK <NA>    $1000… $12500 … $10000… IAP    VO
## 10 13          3 2010  6005     30 MALE  BLACK <NA>    $2500… $30000 … $25000… IAP    VO
## # … with 655 more rows, and 2 more variables: vote08 <dbl>, earthsun <fct>
```

Simple **dplyr** verbs

`arrange()` - put data in order by one or more variables

```
gss %>% arrange(desc(age))
```

```
## # A tibble: 4,761 x 15
##    firstid  wave year  id      age sex   race  divorce income income06 rincome vote00 vo
##    <fct>   <dbl> <fct> <fct> <dbl> <fct> <fct> <fct>   <fct>  <fct>    <fct>   <fct>  <f
##  1 181         1 2006  181      89 MALE  WHITE NO      $2500… $40000 … <NA>    VOTED  VO
##  2 181         2 2008  3062     89 MALE  WHITE NO      $2500… $50000 … <NA>    IAP    VO
##  3 431         1 2006  431      89 FEMA… WHITE NO      $1500… $15000 … <NA>    VOTED  VO
##  4 431         2 2008  3137     89 FEMA… WHITE NO      $1500… $15000 … <NA>    IAP    VO
##  5 431         3 2010  6110     89 FEMA… WHITE NO      $1000… $12500 … <NA>    IAP    VO
##  6 771         1 2006  771      89 FEMA… OTHER NO      <NA>   <NA>     <NA>    DID N… DI
##  7 771         2 2008  3257     89 FEMA… WHITE NO      <NA>   <NA>     <NA>    IAP    DI
##  8 1101        2 2008  3356     89 FEMA… WHITE NO      $2500… $150000… <NA>    IAP    VO
##  9 1101        3 2010  6289     89 FEMA… WHITE NO      <NA>   <NA>     <NA>    IAP    VO
## 10 1292        3 2010  6342     89 MALE  WHITE NO      $2500… $50000 … <NA>    IAP    VO
## # … with 4,751 more rows, and 2 more variables: vote08 <dbl>, earthsun <fct>
```

Making summaries

`group_by()` creates groups from variables

`summarize()` aggregates a set of rows

Combine `group_by()` + `summarize()` to make group-wise summaries:

```
gss %>%
  group_by(sex, race) %>%
  summarize(age = mean(age))
```

```
## # A tibble: 6 x 3
## # Groups:   sex [2]
##   sex    race    age
##   <fct>  <fct> <dbl>
## 1 MALE   WHITE  50.0
## 2 MALE   BLACK  46.3
## 3 MALE   OTHER  38.9
## 4 FEMALE WHITE  50.6
## 5 FEMALE BLACK  46.6
## 6 FEMALE OTHER  42.4
```

Note: You can also use `group_by()` with `mutate()`. When might that be useful?

Making new variables

Create new variables with the verb `mutate()`:

```
new_data <-
  data %>%
  mutate(new_variable = some_function(old_variable))
```

The key is that the function needs to work on a *vector*. Why? Because columns of data frames are vectors, and mutate transforms a whole column.

For example, `+` works on vectors:

```
x <- c(1, 2, 3, 4)
y <- c(5, 6, 7, 8)
x + y
```

```
## [1]  6  8 10 12
```

Making new variables

Because `+` works on vectors, we can use `mutate()` to create a new column `z` by adding `x` and `y`:

```
d <- tibble(x, y)

d %>%
  mutate(z = x + y)
```

```
## # A tibble: 4 x 3
##       x     y     z
##   <dbl> <dbl> <dbl>
## 1     1     5     6
## 2     2     6     8
## 3     3     7    10
## 4     4     8    12
```

More on making new variables

Change the type of a variable

```
gss %>% mutate(age = as.numeric(age))
```

`as.numeric`, `as.character`, `as.factor` (and `as_factor`) ...

Test a TRUE/FALSE condition

```
gss %>% mutate(divorce_bin = ifelse(divorce == "YES", 1, 0))

gss %>% mutate(divorce_lgl = (divorce == "YES"),
               divorce_dbl = as.numeric(divorce_lgl))
```

Factors with **forcats**

Factors represent categorical data. forcats functions of the form `forcats::fct_*()` manipulate factors.

Text data with **stringr**

Characters (also called *strings*) represent text data. stringr functions of the form `stringr::str_*()` manipulate strings.

More on making new variables - factors

Factors with **forcats**

Factors represent categorical data. forcats functions of the form `forcats::fct_*()` manipulate factors. Useful functions include

- `fct_relevel`: change the order of the factor levels (categories). In an R model, the *first* level is the reference category
- `fct_drop`: get rid of levels that don't appear in the data
- `fct_recode`: manually change factor levels
- `fct_collapse`: manually combine factor levels
- `fct_reorder`: reorder levels based on a second variable (good for plotting)

```
gss_divorce <-
  gss %>%
  mutate(divorce = fct_drop(divorce),
         divorce = fct_relevel(divorce, "NO", "YES"))

levels(gss_divorce$divorce)
```

```
## [1] "NO"  "YES"
```

More on making new variables - strings

Text data with **stringr**

Characters (also called *strings*) represent text data. stringr functions of the form `stringr::str_*()` manipulate strings. Useful functions include

- `str_c`: combine strings
- `str_detect`: TRUE/FALSE if a pattern is in the string
- `str_extract`: pattern detection, but gives you the pattern itself
- `str_replace`, `str_replace_all`: replace a pattern

Note: To get the most out of working with text data, you'll need to learn a bit about ***regular expressions*** ("regex"), a general way for representing text patterns. For instance, with regex you can find:

- the beginning of a piece of text (`"^"`), or the end (`"$"`)
- any number (`"[0-9]"` or `"\\d"`)
- any number of numbers (`"[0-9]*"`, or `"[0-9]+"` for at least one)

Other good stuff

You might need to do these things, you might not. It's good to know they exist.

Reshape data with `tidyr::pivot_longer()` and `tidyr::pivot_wider()`

Sometimes it's better to have data in "long" form for visualization and "wide" form for modeling.

Join data sets with `dplyr::left_join()`

If you have more than one table, you can join them using identifying variables.

Apply functions with `purrr::map()`

If you want to use a normal, non-vector function inside mutate, the purrr package can "map" any function to a list or vector of values.

Note: `map()` returns a list, but often you want a vector---use `map_chr()`, `map_dbl()`, `map_lgl()`, ..., to get a vector column of the appropriate type.

An advanced example

If you want a challenge, try to understand how this code works. It fits a separate model to each wave of the gss panel data.

```
gss_nested <-
  gss %>%
  group_by(wave) %>%
  nest()

gss_nested %>%
  mutate(fit = map(data, ~lm(age ~ race + sex, data = .)))
```

```
## # A tibble: 3 x 3
## # Groups:   wave [3]
##    wave             data fit
##   <dbl> <list<df[,14]>> <list>
## 1     1    [1,992 × 14] <lm>
## 2     2    [1,514 × 14] <lm>
## 3     3    [1,255 × 14] <lm>
```

Rewriting code in a tidyverse style

```
test <-
  gss_panel10_long %>%
  dplyr::select("earthsun",
                names(gss_panel10_long)[grep("sci", names(gss_panel10_long))]) %>%
  filter(is.na(earthsun) == FALSE)

## Some of these we grabbed, dont't make sense, e.g., vissci
test <- test[, -which(colnames(test) == "uscitzn")]
test <- test[, -which(colnames(test) == "vissci")]

sum_sci <- apply(data.frame(test)[, -1], 1, sum, na.rm = TRUE)
sum(is.na(sum_sci))
```

```
test <-
  gss_panel10_long %>%
  dplyr::select(earthsun, contains("sci")) %>%
  filter(!is.na(earthsun)) %>%
  dplyr::select(-uscitzn, -vissci, -earthsun)

# TODO: find a tidier way to sum rows...
test %>%
  mutate(sum_sci = rowSums(., na.rm = TRUE)) %>%
  summarize(sum_sci = sum(is.na(sum_sci)))
```

Lab exercise

(in `data_wrangling.R`)