R: Data Transformation 2

Sakol Suethanapornkul

11 November 2020

Quiz

For this quiz, we will use Pew Research Center's American Trend Panel Wave survey. The survey is downloadable from Pew's website. We will read in a SPSS file and remove labels from the file:

```
library(haven)

#read spss file with read_sav()

wtp59 <- read_sav("ATPW59.sav") %>%
  zap_label() %>%
  zap_labels()
```

Quiz

Just to make things easier, we will change column names to snake cases. We will make use of the package called `janitor`. Install the package and copy and paste the following command to R:

```
library(janitor)

wtp59 <- wtp59 %>%
  clean_names() #or janitor::clean_names()
```

Quiz

With the data frame in place, complete the following steps:

[1] Create a new data frame with just the questions from the W59 survey (which end with `_w59`);

[2] Drop questions about Youtube (which begin with `yt`) from the data frame;

[3] Move the last two columns on weight (`weight`) to third & fourth columns before `device_type_w59`;

[4] Drop two questions about a presidential candidate's quality (`demdealfght` and `repdealfght`); and

[5] Check how many columns are left. What is the first column? Last?

Recap

Let's remind ourselves again about vectors:

```
nums <- seq(from = 100, to = 200, by = 1.25)
length(nums)

is.numeric(nums)
is.vector(nums)

ord <- seq(from = 2, to = length(nums), by = 2)
nums[ord]

nums[nums >= mean(nums)]
nums >= mean(nums)

nums[[length(nums)]]
```

Recap

We focused on tibbles (aka data frames):

- How to create them;
- How to select columns from tibbles; and
- How to relocate columns

Recap

We focused on tibbles (aka data frames):

- How to create them;
- How to select columns from tibbles; and
- How to relocate columns

We focused on two functions: `select()` and `relocate()`:

```
library(tidyverse)
data(diamonds)

dia <- select(diamonds, starts_with("c") )

dia <- relocate(dia, price, .before = color)
```

Recap

EXERCISE: How do the two functions differ with respect to an end product?

```
select(diamonds, where(is.numeric) )

relocate(diamonds, where(is.numeric) )
```

Recap

ANSWER: `select()` subsets your data, narrowing them down to only those columns you specify, while `relocate()` returns a tibble with the same number of columns (and rows) as the original one.

Part I: `filter()`

Previously, we saw `select()` and `relocate()` which work with columns. In this part, we will look at functions that help us pick observations (i.e., *rows*) from tibbles.

Part I: `filter()`

`filter()` allows you subset and retain **rows** that meet your condition(s).

We are going to work with a new data set (`mpg`) and would like to select cars with 6 or more/less cylinders (in `cyl`):

```
data(mpg)
help(mpg)

filter(mpg, cyl > 6)

filter(mpg, cyl < 6)
```

Part I: `filter()`

`filter()` works with comparison operations we saw previously:

```
filter(mpg, manufacturer  == "nissan")
filter(mpg, manufacturer  %in% c("nissan", "subaru"))
filter(mpg, !manufacturer %in% c("nissan", "subaru"))
#notice !manufacturer

filter(mpg, hwy >= 22)

filter(mpg, between(hwy, 20, 30))
#this is equivalent to:
filter(mpg, hwy > 20 & hwy < 30)
```

Part I: `filter()`

And of course, you can combine more than one condition in `filter()`:

```
filter(mpg, year == 2008,  cyl == 6)    #same as
filter(mpg, year == 2008 & cyl == 6)

filter(mpg, year == 1999 | cyl != 6)
```

Part I: `filter()`

EXERCISE: Find cars that...

[1] are manufactured by Toyota or Honda;

[2] are made by Toyota or Honda and have 4 cylinders;

[3] are SUV and can get 15 miles or more per gallon in city;

[4] are four-wheel drive SUV; and

[5] run 17 miles an hour in city and 20 miles an hour on highway.

Part I: `filter()`

ANSWER:

```
filter(mpg,
       manufacturer %in% c("toyota", "honda"))

filter(mpg,
       manufacturer %in% c("toyota","honda") & cyl>4)

filter(mpg, class == "suv" & hwy > 20)

filter(mpg, class == "suv" & drv == "4")

filter(mpg, cty > 17 & hwy > 20)
```

NOTE: `drv` is a character vector.

Part I: `filter()`

One thing we haven't discussed more thoroughly is `NA`:

```
#NA is unknown
NA > 5
NA == NA

nums <- c(15, 20, 23, 24, NA, 26, NA, 31)

# NA is contagious
mean(nums)
sd(nums)
max(nums)
```

EXERCISE: How do we check if elements inside a vector are `NA`?

ANSWER: We use is.na().

```
nums <- c(15, 20, 23, 24, NA, 26, NA, 31)

is.na(nums)
```

In several functions, you can remove NA with na.rm = TRUE

```
mean(nums, na.rm = TRUE)
sd(nums, na.rm = TRUE)
```

Part I: `filter()`

`filter()` only includes rows that are `TRUE` of your condition(s). If you want rows with `NA`, you'll need to use `is.na()`:

```
#read in Pew's data, if you haven't done so

filter(wtp59, yearahed_w59 == 1)
filter(wtp59, yearahed_w59 == 1 | is.na(yearahed_w59))

# what's going on here?
filter(wtp59, yearahed_w59 == 1 & is.na(yearahed_w59))
```

Part II: arrange()

Just as relocate() sorts columns, arrange()does so with **rows**:

```
arrange(mpg, cty)
```

Use desc() inside arrange() to reorder in descending order:

```
arrange(mpg, desc(cty))
```

Part II: `arrange()`

You can reorder rows in more than one columns. `arrange()` reorders rows one column at a time, using the next column to break ties:

```
arrange(mpg, desc(cty), desc(hwy))
```

Part II: arrange()

Again, we can combine `filter()` and `arrange()` together:

```
mpg_small <- filter(mpg,
                    !manufacturer %in% c("toyota",
                                         "nissan")
                    )

mpg_small <- arrange(mpg_small, desc(year), desc(cty)
                     )
```

Part III: `mutate()`

So far, we have learned how to select columns and rows that meet our condition(s). For instance, we may create a new tibble by selecting columns and rows that are:

```
mpg_small <- select(mpg, starts_with("m"), drv:class)
mpg_small <- filter(mpg_small, class != "suv")
```

Part III: `mutate()`

But in a number of cases, we'd like to "do" something with our existing tibbles. This often involves creating new columns. We can easily do that with `mutate()`:

```
#create a column that repeats "EPA"
mutate(mpg_small, source = "EPA")

#or add this column back to the tibble
mpg_small <- mutate(mpg_small, source = "EPA")
```

EXERCISE: Why did we end up with `EPA` being repeated > 170 times?

Part III: `mutate()`

If you recall, arithmetic operation is vectorized in R:

```
(num1 <- 1:10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
num1 + c(2,4)
```

```
##  [1]  3  6  5  8  7 10  9 12 11 14
```

```
num1 + 15
```

```
##  [1] 16 17 18 19 20 21 22 23 24 25
```

Part III: `mutate()`

R expands a shorter vector to match the length of a longer one. This means that our `source = "EPA"` is expanded to match the length of `mpg_small`

Also, if you recall:

```
c(5, 6, 7, 8, 9) + c(10, 11, 12, 13, 14)
```

```
## [1] 15 17 19 21 23
```

Part III: `mutate()`

So, what do you think will happen here?

```
mutate(mpg_small, dif = hwy - cty)

#or add to an existing tibble
mpg_small <- mutate(mpg_small, dif = hwy - cty)
```

Part III: `mutate()`

You can reference this newly created variable right inside `mutate()`:

```
mutate(mpg_small, dif = hwy - cty,
                  dif_percent = dif / 100
       )
```

Part III: `mutate()`

EXERCISE: Create new variables that...

[1] subtract `hwy` from the maximum value of `hwy`;

[2] subtract `hwy` from the average of `hwy`

Then, spend some time thinking about each answer you've come up. How might your code work?

Part III: `mutate()`

ANSWER:

```
mutate(mpg, max_dif  = max(hwy) - hwy)

mutate(mpg, mean_dif = mean(hwy) - hwy)
```

If you know that, for example, `hwy` contains some `NA`, it is safe to:

```
mutate(mpg, mean_dif = max(hwy, na.rm = TRUE) - hwy)

mutate(mpg, mean_dif = mean(hwy, na.rm = TRUE) - hwy)
```

Part III: `mutate()`

How does this work? Let's return to vectors:

```
nums <- 11:20

max(nums) #max() gives the max value
```

```
## [1] 20
```

```
max(nums) - nums
```

```
##  [1] 9 8 7 6 5 4 3 2 1 0
```

Part III: `mutate()`

In a lot of cases, arithmetic operations can serve us well (with numeric vectors).

But in some cases, we may need to create a new column that is based on some condition of an existing column.

Part III: mutate()

That can be done with if_else():

```
gen    <- rep(c("M", "F"), times = 5)
scores <- 11:20

#dplyr::if_else(condition, true, false)

if_else(gen == "F", "female", "male")
if_else(scores > mean(scores), "above", "below")
```

NOTE: Use if_else() when there are just two options

Part III: `mutate()`

EXERCISE: Create new variables for `mpg_small` that...

[1] re-code `year` from 1999 to "90s" and from 2008 to "00s";

[2] classify `cty` into "above" and "below" based on `mean(cty)`; and

[3] change only "suv" in `class` into "big car"

Part III: mutate()

ANSWER:

```
mutate(mpg, decade = if_else(year == 1999,
                             "90s", "00s")
       )

mutate(mpg, cty_ave = if_else(cty > mean(cty),
                              "above", "below")
       )

mutate(mpg, class = if_else(class == "suv",
                            "big car", class)
       )
```

Part III: `mutate()`

In contrast to `if_else()`, `case_when()` can match several conditions:

```
nums <- 1:30

case_when(nums %% 10 == 0 ~ "ten",
          nums %% 5 == 0 ~ "five",
          nums %% 3 == 0 ~ "three",
          TRUE ~ as.character(nums)
          )
```

NOTE: `case_when()` moves from the specific to the general, so you need to put the most general on the last line.

Part III: `mutate()`

EXERCISE: What is going to happen here?

```
case_when(nums %% 10 == 0 ~ "ten",
          nums %% 5 == 0 ~ "five",
          nums %% 3 == 0 ~ "three"
          )
```

Part III: mutate()

You can apply case_when() and mutate() in some useful ways:

```
fruit <- c("banana", "apple", "mango")
food  <- c("hamburger", "pizza", "steak")

dish  <- c("hamburger", "banana", "steak", "rice",
           "dumpling", "apple", "sushi", "hamburger",
           "rice", "papaya", "orange")

case_when(dish %in% fruit ~ "fruits",
          dish %in% food ~ "foods",
          TRUE ~ "others")
```

Part III: `mutate()`

EXERCISE: Let's go back to the `mpg` data set. Some cars in the data frame are American brands while others are foreign brands.

[1] American brands: chevrolet, dodge, ford, jeep, lincoln;

[2] Japanese brands: honda, toyota, nissan;

[3] German brands: audi, volkswagen

There are other brands that are not in this list. **Create a new column for countries of these car brands**.

Q & A