R: Data Transformation

Sakol Suethanapornkul

10 November 2020

Quiz

Complete the following steps:

[1] Create a vector of heights (in cm) of 15 people:

[2] Subset the first four elements from the vector:

[3] Subset the final element from the vector (**Hint**: use [[ ]]):

[4] Subset even-numbered elements from the vector:

[5] Check whether values are below an average height:

[6] Find mean and standard deviation of heights:

Answers

These are just some of the ways:

[1] `height <- c(150, 155, 160, ...)` [1]

[2] `height[1:4]`

[3] `height[[15]]` or
`height[[length(height)]]`

[4] `height[c(FALSE, TRUE)]` or
`height[seq(0, length(height), 2)]`

[5] `height < mean(height)`

[6] `mean(height)` and `sd(height)`

[1] A better answer is: height <- rnorm(n = 15, mean = 170, sd = 3)

Recap

So far, our focus has been on vectors:

- How to create them;
- How to subset them; and
- How to manipulate them.

Recap

So far, our focus has been on vectors:

- How to create them;
- How to subset them; and
- How to manipulate them.

If this isn't yet obvious, an operation on vectors is also a vector:

```
height <- rnorm(n = 10, mean = 170, sd = 3)
height <- round(height, digits = 1)

test   <- height[height > mean(height)]

is.vector(test)
```

Recap

We also discussed data frames (and tibbles).

Each data frame is a **named list of vectors of equal length**.

```
dat <- tibble(Class  = c("A", "A", "A",
                         "B", "B", "B"),
              Gender = c("M", "F", "M",
                         "F", "F", "M"),
              Test   = c(15, 20, 21,
                         16, 17, 18)
              )
```

EXERCISE: Check whether `Class` and `Gender` are vector.

Recap

ANSWER: We use `is.vector()` from base R to test if `x` is a vector

```
dat$Class
dat$Gender

is.vector(dat$Class)
```

NOTE: We can refer to each column of a data frame with a dollar sign ($).

Part I: Data frames

Once again, we create a data frame or tibble with `data.frame()` or `tibble()`:

```
class <- tibble(ID     = c(1, 2, 3, 4, 5),
                Gender = c("M", "F", "T", "T", "M"),
                Rating = c(4, 5, 3, 2, 4),
                Math   = c(25, 33, 29, 21, 31)
                )
```

Part I: Data frames

We can inspect data frames with some useful functions:

```
str(dat)
summary(dat)

head(dat, n = 3)
tail(dat, n = 3)
```

Part I: Data frames

But real data science questions require that you go beyond these simple functions! Let's load in some data set from `tidyverese`:

```
library(tidyverse)
data(diamonds)
```

EXERCISE: What can you learn about the diamonds data set from these two functions?

```
str(diamonds)
summary(diamonds)
```

NOTE: Read the package message. What warning message can you see?

Part I: Data frames

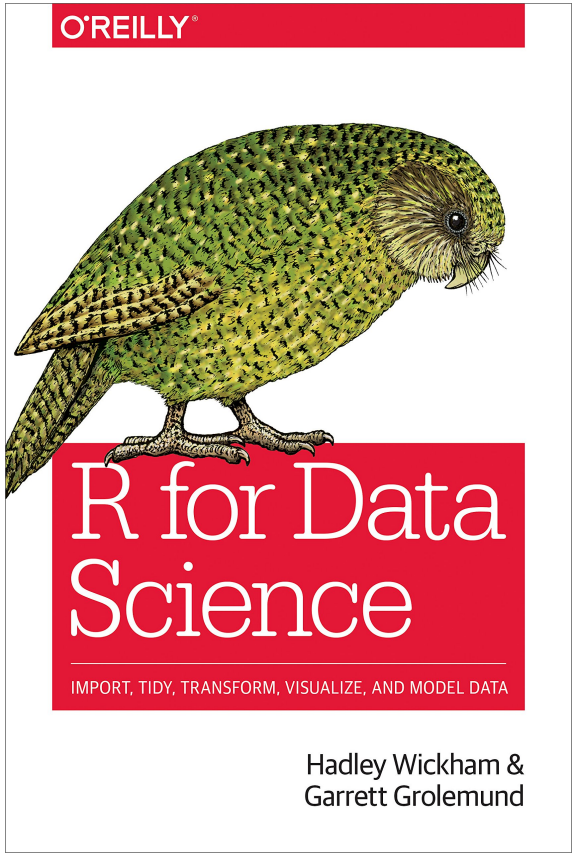
In several cases, you may want to:

- Subset a large data frame based on some information in columns x or y;
- Select only a few columns from the whole data frame;
- Calculate summary statistics by groups;
- And many more

And `str()` and `summary()` won't be that useful in this regard.

Part II: dplyr

This part corresponds to Chapter 3 in **R for Data Science**.



Part II: dplyr

Data transformation is an important first step toward a better understanding of your data!

For this part, we need `dplyr` which is part of the `tidyverse` package.

Part II: dplyr

`dplyr` consists of many "verbs". Each verb performs a specific function:

Verbs	Functions
`filter`	select rows by their values
`select`	select columns
`relocate`	changes column positions
`arrange`	sorts rows by their values
`mutate`	creates new columns
`summarize`	combines values to form a new one

There are many other helpers & functions. We will slowly go through them!

Part II: dplyr

These verbs are consistent; each function has the same "format":

Arguments	What is needed
1^{st}	Data frame
$2^{nd}+$	Columns to be selected (w/o quote)

The end result is always a **tibble**.

Part II: dplyr

We will begin slowly and go through each "verb" one by one. For each verb, we will also look at helper functions.

- For now, things may look silly, but bear with me;
- Later, we will string all these verbs together with %>% (pipe);
- And we will get a long line of commands to perform some cool stuff

Part II: `select()`

`select()` is useful when you want to select columns. There are four ways to use `select()`:

- pick columns by their *names*;
- pick columns by their *numbers*;
- pick columns by *functions*; and
- pick columns by *types*

Part II: `select()`

`select()` allows you to pick columns by their **names** (*while dropping others not chosen*):

```
diamonds

select(diamonds, x, y, z)

select(diamonds, c(x, y, z) )
```

If you want to create a new tibble from `select()`, you can use a name to refer to it:

```
diamonds_small <- select(diamonds, c(x, y, z) )
```

Part II: `select()`

`select()` works with a colon (:), and you can select between column a and column b:

```
select(diamonds, x, y, z)

select(diamonds, x:z)
```

EXERCISE: What do these codes do?

```
select(diamonds, -x)
select(diamonds, -c(x, z))

select(diamonds, !x)
select(diamonds, !c(x, z))
```

Part II: `select()`

`select()` also works with **column numbers**:

```
select(diamonds, 1, 3, 5)

select(diamonds, c(1, 3, 5) )

select(diamonds, 1:3)
```

This approach is generally not recommended.

EXERCISE: Why do you think `select()`ing by column numbers is not a good idea?

Part II: `select()`

`select()` also works with **functions** that select names:

Functions	What it does
`starts_with("car")`	select columns that starts with "car"
`ends_with("ty")`	select columns that end with "ty"
`matches("\\d.+")`	select columns with regular expressions
`contains("car")`	select columns with "car" in the name
`everything()`	select all columns

Part II: `select()`

EXERCISE: Write codes to perform the following actions.

You may combine helper functions with - and : to select columns:

[1] Select `carat`, `cut`, `color`, and `clarity` from `diamonds`;

[2] Select `carat`, and `cut`;

[3] Select `depth`, `table`, `price`, `x`, `y`, and `z`; and

[4] Select `table` and `price`

Part II: Pro Tip #1

Column names:

- can contain letters, numbers, _ (underscore), and . (period);
- must begin with letters;
- should be consistent (e.g., starts/ends with the same naming);
- do not contain spaces

Part II: Pro Tip #1

Suppose you collect the following information:

- *SES*, *GPA*, *test*, *family size*, and *gender* from students;
- *size*, *status*, *achievement*, *standing*, and *SES* from schools

EXERCISE: How would you name these variables in your data frame?

Part II: `select()`

And lastly, `select()` can be used to pick columns by **types**:

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

select(diamonds, where(is.factor) )

# What happens here?
select(diamonds, where(is.character) )
```

NOTE: Notice `where()` along with `is.*()` functions

Part II: `select()`

We can combine these four ways with Boolean operators (`&`, `|` or `!`) to select columns.

```
select(diamonds, starts_with("c") & where(is.factor)
       )

select(diamonds, starts_with("c") & !where(is.factor)
       )
```

Part II: relocate()

relocate() allows you to change the position of columns. The same four approaches to selecting columns apply:

```
relocate(diamonds, clarity)

relocate(diamonds, clarity, depth)

relocate(diamonds, 3:5)

relocate(diamonds, ends_with("e") )

relocate(diamonds, where(is.numeric) )

relocate(diamonds, starts_with("c") & where(is.factor)
         )
```

Part II: `relocate()`

You can tell `relocate()` where you want the columns to go with `.before =` and `.after =` :

```
relocate(diamonds, x, .before = cut)
relocate(diamonds, x, .after = cut)    #but

relocate(diamonds, x, .before = cut, .after = carat)
```

Part II: `select()` then `relocate()`

Let's string the two verbs together. At the end of this whole lesson, we will see a smarter way to do this with less code:

```
small_dia <- select(diamonds, starts_with("c") &
                      where(is.factor), x:z
                    )

small_dia <- relocate(small_dia, x:z, .before = color)
```

Part II: Homework

Let's read it real data from a survey. For now, just copy and paste the code below:

```
library(haven)

# read spss file (.sav) into R

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

Play around with this data set. Read an accompanying pdf.