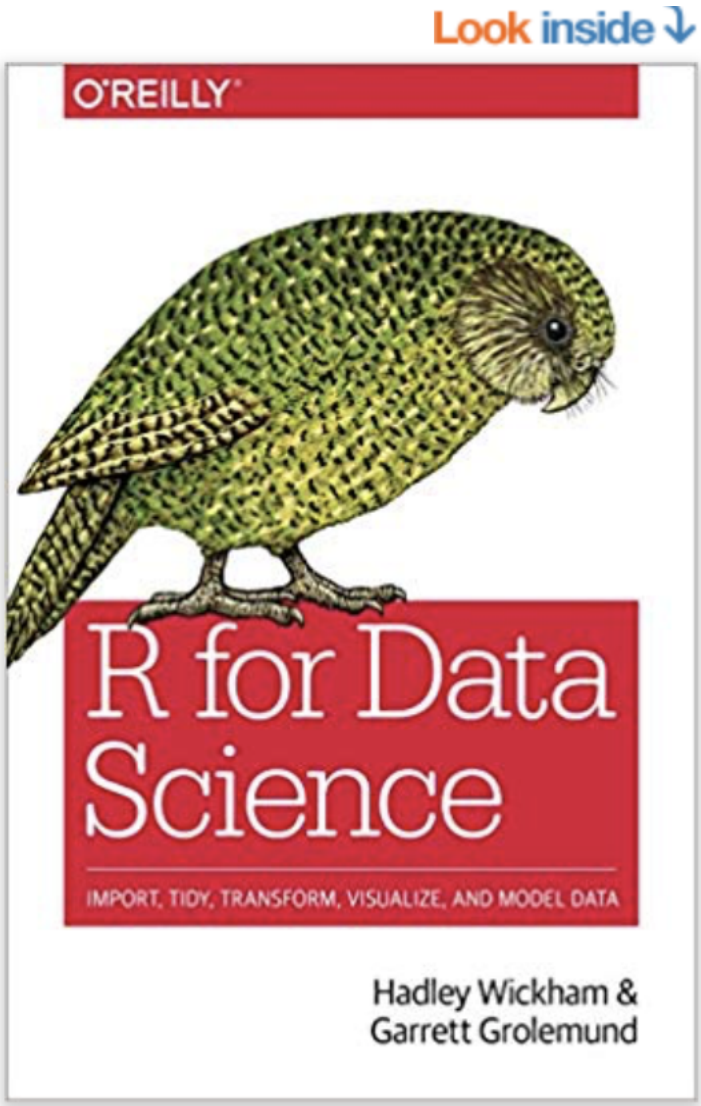
Tidyverse and dplyr

Laura Garcia Montoya

07/01/2019

Tidyverse

Tydiverse is a huge package in R developed by Garrett Grolemund and Hadley Wickham. They have a really great book to lear this package (you can find it online for free here: https://r4ds.had.co.nz). For this workshop I mostly rely on the material developed by them (chapter 5). In the ideal future, you will read the full book and be able to answer all practice questions. In the meantime, I present here the main takeaways from chapter 5.



For example, ggplot and dplyr are very useful packages that are part of the tidyverse package.

1. Prerequisites: Install and load relevant packages

```
install.packages("tidyverse") #Ignore if you already did this
library(tidyverse)
```

2. The datasets:

In order to see how dplyr works, we will use the same dataset that is used in the book. However, we will work in parallel withthe dataset we have been using through out the quarter. In real life your datasets will look more like the latter. You will have to be attentive to the fact that there will be two datasets uploaded in R.

2.1 Flights dataset

```
install.packages("nycflights13",repos = "http://cran.us.r-project.org")

library(nycflights13)
```

2.2 Our example dataset:

```
library(readxl)
setwd("~/Box Sync/PhD/TAship/FIFTH YEAR/TA - FALL/Database/")
data_example <- read_excel("Data_example_class.xls")
```

2.3 Basic information about datasets

Once, tydiverse is loaded, by typping the name of the dataset you will obtain a brief overwiew of the dataset: a tibble. It is a bit differente to what we are used to see in the past with command head(). "Tibbles are data frames, but slightly tweaked to work better in the tidyverse."

```
flights

data_example
```

3. dplyr

Six Basic functions:

''These six functions provide the verbs for a language of data manipulation.":

- select(): Pick variables by their names
- filter(): Pick observations by their values.
- arrange(): Reorder the rows
- mutate(): Create new variables with functions of existing variables
- summarise(): Collapse many values down to a single summary

- group_by(): changes the scope of each function from operating on the entire dataset to operating on it group-by-group.

We can combine these functions to get at more complex processes. The structure of the functions is similar:

input: original dataframe \ what to do: instructions \ output: a new data frame \

1. The first argument is a data frame.
2. The subsequent arguments describe what to do with the data frame, using the variable names (without quotes).
3. The result is a new data frame.

3.1 Filter

"Allows you to subset observations based on their values. The first argument is the name of the data frame. The second and subsequent arguments are the expressions that filter the data frame. For example, we can select all flights on January 1st with:"\

```
filter(flights, month == 1, day==1)
```

Or all the observations of the country Afghanistan

```
filter(data_example, CountryCode == "AFG")
```

Note: {dplyr} executes the filtering operation and returns a new data frame. dplyr functions never modify their inputs, so if you want to save the result, you'll need to use the assignment operator $< -$:

```
jan1 <- filter(flights, month == 1, day == 1)
```

If you want R to print the object, you can wrap the whole thing in parenthesis. Let's try it with my birthday!

```
(feb22 <- filter(flights, month == 2, day == 22))
```

You can also combine logical operators with these functions,

```
summer_months<-filter(flights, month=7 | month==8 | month==9)
```

3.1.1 Practice Exercises: Filter

(From the book)

1. Find all flights that:
   (a) Had an arrival delay of two or more hours
   (b) Flew to Houston (IAH or HOU)
   (c) Were operated by United, American, or Delta
   (d) Departed in summer (July, August, and September)
   (e) Arrived more than two hours late, but didn't leave late
   (f) Were delayed by at least an hour, but made up over 30 minutes in flight
   (g) Departed between midnight and 6am (inclusive)
2. Another useful dplyr filtering helper is between(). What does it do? Can you use it to simplify the code needed to answer the previous challenges?

3. How many flights have a missing departure time? What other variables are missing? What might these rows represent?

(From World Bank dataset)

1. Find all countries that:
   (a) in 2001, had more than 70
   (b) In any year between 2005 and 2008, had less than 1000 Electric power consumption (kWh per capita) (variable name - aEGUSEELECKHPC)
   (c) In any year between 2005 and 2008, had more than 10000 Electric power consumption (kWh per capita) (variable name - aEGUSEELECKHPC)

3.2 Arrange

"arrange() works similarly to filter() except that instead of selecting rows, it changes their order. It takes a data frame and a set of column names (or more complicated expressions) to order by. If you provide more than one column name, each additional column will be used to break ties in the values of preceding columns:"

```
arrange(flights, year, month, day)

arrange(data_example, year)
```

What if you wanted descending order according to year as opposed to ascending?

```
arrange(flights, desc(year), month, day)
```

Note: Missing values will always be sorted at the end.

3.2.1 Practice Exercises: Arrange

(From the book)

1. How could you use arrange() to sort all missing values to the start? (Hint: use is.na()).
2. Sort flights to find the most delayed flights. Find the flights that left earliest.
3. Sort flights to find the fastest flights.
4. Which flights travelled the longest? Which travelled the shortest?

3.3 Select

"select() allows you to rapidly zoom in on a useful subset using operations based on the names of the variables."

```
select(flights, year, month, day)
```

Alternatively, you can specify a range of variables.

```
select(flights, year:day)

select(data_example, CountryName:aENATMCO2EKDGD )
```

Or, you can provide a list of variables you do not want to select.

```
select(flights, -(year:day))
```

3.3.2 Helper functions

In addition, there are a number of helper functions you can use within select():

For example: starts_with("abc"): matches names that begin with "abc".

ends_with("xyz"): matches names that end with "xyz".

contains("ijk"): matches names that contain "ijk".

matches("(.)\1"): selects variables that match a regular expression. (more details in book)

3.3.2 Rename

A variant of the function select() is the function renamed(). It can be use to rename variables.Like this:

```
rename(flights, tail_num = tailnum)
```

By using select() in conjunction with everyting(), allows you to reorder your variables:

```
rename(flights, time_hour, air_time, everything())
```

3.3.3 Practice Exercises

(from the book)

1. Brainstorm as many ways as possible to select dep_time, dep_delay, arr_time, and arr_delay from flights.
2. What happens if you include the name of a variable multiple times in a select() call?
3. What does the one_of() function do? Why might it be helpful in conjunction with this vector?

   vars< − c("year", "month", "day", "dep_delay", "arr_delay")
4. Does the result of running the following code surprise you? How do the select helpers deal with case by default? How can you change that default?

   "'r select(flights, contains("TIME")) "'

3.4 Mutate: Adding new variables

mutate() allows you to create new variables as function of existing one. It will add new columns to the end of dataset.

```
flights<-mutate(flights, gain=dep_delay - arr_delay, speed=distance/air_time*60)

View(flights)
```

3.4.1 Practice Exercise

(From the book)

1. Currently departure time and scheduled departure time are convenient to look at, but hard to compute with because they're not really continuous numbers. Convert them to a more convenient representation of number of minutes since midnight.

2. Compare air time with arrival time - departure time. What do you expect to see? What do you see? What do you need to do to fix it?
3. Compare departure time, scheduled departured time, and departure delay. How would you expect those three numbers to be related?

3.5 Summarise

This function collapses data to a single row.

```
summarise(flights, delay = mean(dep_delay, na.rm = TRUE))
```

summarise() is more useful if we combine it with group_by(). "This changes the unit of analysis from the complete dataset to individual groups. Then, when you use the dplyr verbs on a grouped data frame they'll be automatically applied "by group". For example, if we applied exactly the same code to a data frame grouped by date, we get the average delay per date:"

```
by_day <- group_by(flights, year, month, day)
summarise(by_day, delay = mean(dep_delay, na.rm = TRUE))
by_day
```