Tidy R exercises

by courtesy of
"R for data science" - https://r4ds.had.co.nz/transform.html

October 29, 2019

1 Before you start

In this exercise you learn how to use extract and transform data using "tidyverse" functions. Tidyverse is not part of base-R, so you need to install it first by typing ‘install.packages("tidyverse")‘ into the console in Rstudio. After a sucessful install, type ‘library(tidyverse)‘ to load tidyverse into your session (Note that when using library(), the quotes around "tidyverse" are missing). If you want to use tidyverse functions in the future, you ALWAYS have to load it using library() at the start of your session. If you forgot to load it and try to use a tidyverse function, you will get an error message saying the function could not be found.

We will also use the "nycflights13" package. This data frame contains all 336,776 flights that departed from New York City in 2013. Install and load it the same way, using ‘install.packages()‘ and ‘library()‘.

You might notice that this data frame prints a little differently from other data frames you might have used in the past: it only shows the first few rows and all the columns that fit on one screen. (To see the whole dataset, you can run View(flights) which will open the dataset in the RStudio viewer). It prints differently because it’s a tibble. Tibbles are data frames, but slightly tweaked to work better in the tidyverse. For now, you don’t need to worry about the differences.

2 Extracting rows using filter()

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)
```

2.1 Comparisons

To use filtering effectively, you have to know how to select the observations that you want using the comparison operators. R provides the standard suite: >, >=, <, <=, != (not equal), and == (equal).

2.2 logical operators

Multiple arguments to filter() are combined with “and”: every expression must be true in order for a row to be included in the output. For other types of combinations, you’ll need to use Boolean operators yourself: & is “and”, | is “or”, and ! is “not”.

The following code finds all flights that departed in November or December:

```
filter(flights, month == 11 | month == 12)
```

2.3 Exercises - filter()

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)

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? (tip: find the documentation of between() by typing ?between)

3 arrange rows with 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, try:

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

Use desc() to re-order by a column in descending order:

```
arrange(flights, desc(dep_delay))
```

3.1 Exrecises - arrange()

(a) Sort flights to find the most delayed flights.

(b) Find the flights that left earliest.

(c) Sort flights to find the fastest flights.

(d) Which flights travelled the longest? Which travelled the shortest?

4 select columns with select()

It's not uncommon to get datasets with hundreds or even thousands of variables. In this case, the first challenge is often narrowing in on the variables you're actually interested in. select() allows you to rapidly zoom in on a useful subset using operations based on the names of the variables.

select() is not terribly useful with the flights data because we only have 19 variables, but you can still get the general idea:

```
# Select columns by name
select(flights, year, month, day)
# Select all columns between year and day (inclusive)
select(flights, year:day)
# Select all columns except those from year to day (inclusive)
select(flights, -(year:day))
```

There are a number of helper functions you can use within select():

- 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”.
- num_range("x", 1:3): matches x1, x2 and x3.

select() can be used to rename variables, but it’s rarely useful because it drops all of the variables not explicitly mentioned. Instead, use rename(), which is a variant of select() that keeps all the variables that aren’t explicitly mentioned:

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

Another option is to use select() in conjunction with the everything() helper. This is useful if you have a handful of variables you’d like to move to the start of the data frame.

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

4.1 Exercises - select()

(a) Brainstorm as many ways as possible to select dep_time, dep_delay, arr_time, and arr_delay from flights.

(b) What happens if you include the name of a variable multiple times in a select() call?

(c) 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")

5 Add new variables with mutate()

Besides selecting sets of existing columns, it’s often useful to add new columns that are functions of existing columns. That’s the job of mutate().

mutate() always adds new columns at the end of your dataset so we’ll start by creating a narrower dataset so we can see the new variables. Remember that when you’re in RStudio, the easiest way to see all the columns is View().

```
flights_sml <- select(flights,
  year:day,
  ends_with("delay"),
  distance,
  air_time
)
mutate(flights_sml,
  gain = dep_delay - arr_delay,
  speed = distance / air_time * 60
)
```

Note that you can refer to columns that you’ve just created:

```
mutate(flights_sml,
  gain = dep_delay - arr_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours
)
```

If you only want to keep the new variables, use transmute():

```
transmute(flights,
  gain = dep_delay - arr_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours
)
```

5.1 Exercises - mutate()

(a) Currently dep_time and sched_dep_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.

(b) Compare air_time with arr_time - dep_time. What do you expect to see? What do you see? What do you need to do to fix it?

(c) Compare dep_time, sched_dep_time, and dep_delay. How would you expect those three numbers to be related?

(d) What does 1:3 + 1:10 return? Why?