CHAPTER 3: DATA MANIPULATION WITH DPLYR

https://jrnold.github.io/e4qf/data-transformation.html

CONFLICT WARNING:- When you load a new package such as tidyverse, you often get conflicts message that's printed. It tells you that dplyr overwrites some functions in base R. If you want to use the base version of these functions after loading dplyr, you'll need to use their full names: *stats::filter()* and *stats::lag()*.

In this chapter we are going to learn the five key dplyr functions that allow us to solve the vast majority of your data-manipulation challenges:

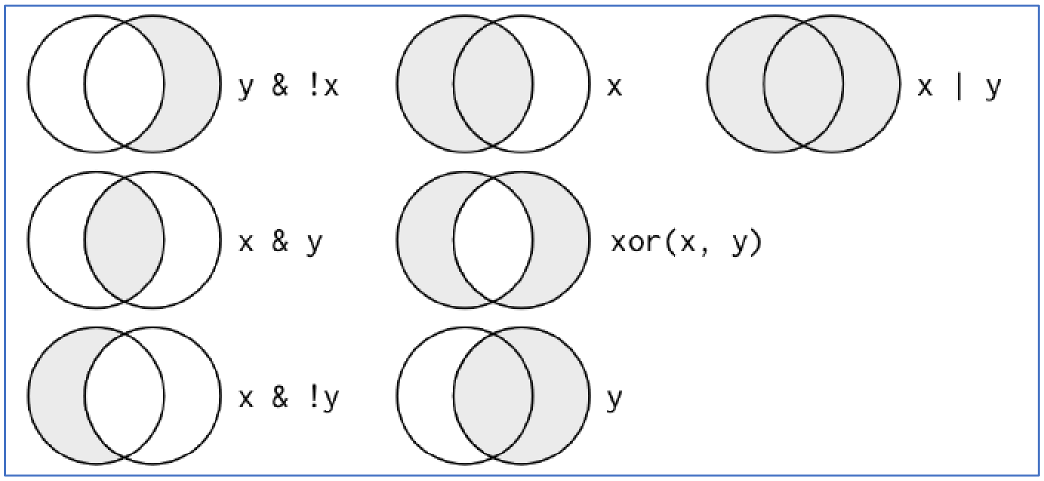
- Pick observations by their values (filter())
- Reorder the rows (arrange())
- Pick variables by their names (select())
- Create new variables with functions of existing variables (mutate())
- Collapse many values down to a single summary (summarize())

These can all be used in conjunction with group_by(), which changes the scope of each function from operating on the entire dataset to operating on it group-by-group. These six functions provide the verbs for a language of data manipulation.

All verbs work similarly:

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.

LOGICAL OPERATORS: Here "&" means "AND" and "|" means "OR"



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

The order of operations doesn't work like English. We can't write *filter(flights, month == 11 | 12)*, which you might literally translate into "finds all flights that departed in November or December." Instead it finds all months that equal 11 | 12, an expression that evaluates to TRUE. In a numeric context (like here), TRUE becomes one, so this finds all flights in January, not November or December.

A useful shorthand for this problem is x %in% y. This will select every row where x is one of the values in y. We could use it to rewrite the preceding code:

```
nov_dec <- filter(flights, month %in% c(11, 12))
```

Sometimes you can simplify complicated subsetting by remembering De Morgan's law: !(x & y) is the same as !x | !y, and !(x | y) is the same as !x & !y. For example, if you wanted to find flights that weren't delayed (on arrival or departure) by more than two hours, you could use either of the following two filters:

```
filter(flights, !(arr_delay > 120 | dep_delay > 120))
filter(flights, arr_delay <= 120, dep_delay <= 120)
```

Another important function is BETWEEN(). *between(x, left, right)* is equivalent to *x >= left & x <= right*

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:

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

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

```
arrange(flights, desc(arr_delay))
```

Missing values are always sorted at the end:

```
df <- tibble(x = c(5, 2, NA))
arrange(df, x)
arrange(df, desc(x))
```

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 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".
- *matches("(.)\\1")* selects variables that match a regular expression. This one matches any variables that contain repeated characters. You'll learn more about regular expressions in Chapter 11.
- *num_range("x", 1:3)* matches x1, x2, and x3.

We can use rename() to rename a column

```
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())
```

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

- *select(flights, dep_time, dep_delay, arr_time, arr_delay)*
- *select(flights, starts_with("dep_"), starts_with("arr_"))*
- *select(flights, matches("^(dep|arr)_(time|delay)$"))*

What happens if you include the name of a variable multiple times in a select() call? It ignores the duplicates, and that variable is only included once. No error, warning, or message is emitted.