Filtering rows

The GRAPH Courses team

October 2022

This document serves as an accompaniment for a lesson found on https://thegraphcourses.org.

The GRAPH Courses is a project of the Global Research and Analyses for Public Health (GRAPH) Network, with the support of the World Health Organization (WHO) and other partners

Intro

Onward with the {dplyr} package, discovering the `filter` verb. Last time we saw how to `select` variables (columns) and today we will see how to keep or drop data entries, rows, using `filter`. Dropping abnormal data entries or keeping subsets of your data points is another essential aspect of data wrangling.

Let's go !

Learning objectives

1. You can use `dplyr::filter()` to keep or drop rows from a dataframe.
2. You can filter rows by specifying conditions on numbers or strings using relational operators like greater than (`>`), less than (`<`), equal to (`==`), and not equal to (`!=`).
3. You can filter rows by combining conditions using logical operators like the ampersand (`&`) and the vertical bar (`|`).
4. You can filter rows by negating conditions using the exclamation mark (`!`) logical operator.

5. You can filter rows with missing values using the `is.na()` function.

The Yaounde COVID-19 dataset

In this lesson, we will again use the data from the COVID-19 serological survey conducted in Yaounde, Cameroon.

```
yaounde <- read_csv(here::here('data/yaounde_data.csv'))
## a smaller subset of variables
yao <- yaounde %>%
  select(age, sex, weight_kg, highest_education, neighborhood,
         occupation, is_smoker, is_pregnant,
         igg_result, igm_result)
yao
```

Introducing `filter()`

We use `filter()` to keep rows that satisfy a set of conditions. Let's take a look at a simple example. If we want to keep just the male records, we run:

```
yao %>% filter(sex == "Male")
```

Note the use of the double equal sign == rather than the single equal sign =. The == sign tests for equality, as demonstrated below:

```
## create the object `sex_vector` with three elements
sex_vector <- c("Male", "Female", "Female")
## test which elements are equal to "Male"
sex_vector == "Male"
```

```
## [1]  TRUE FALSE FALSE
```

So the code `yao %>% filter(sex == "Male")` will keep all rows where the equality test `sex == "Male"` evaluates to `TRUE`.

It is often useful to chain `filter()` with `nrow()` to get the number of rows fulfilling a condition.

```
## how many respondents were male?
yao %>%
  filter(sex == "Male") %>%
  nrow()
```

```
## [1] 422
```

KEY POINT

The double equal sign, ==, tests for equality, while the single equals sign, =, is used for specifying values to arguments inside functions.

PRACTICE

(in RMD)

Filter the `yao` data frame to respondents who were pregnant during the survey. Store in `q1`.

How many respondents were female? (Use `filter()` and `nrow()`)

Relational operators

The == operator introduced above is an example of a "relational" operator, as it tests the relation between two values. Here is a list of some of these operators:

Operator	is TRUE if
A < B	A is **less than** B
A <= B	A is **less than or equal** to B
A > B	A is **greater than** B
A >= B	A is **greater than or equal to** B
A == B	A is **equal** to B
A != B	A is **not equal** to B
A %in% B	A **is an element of** B

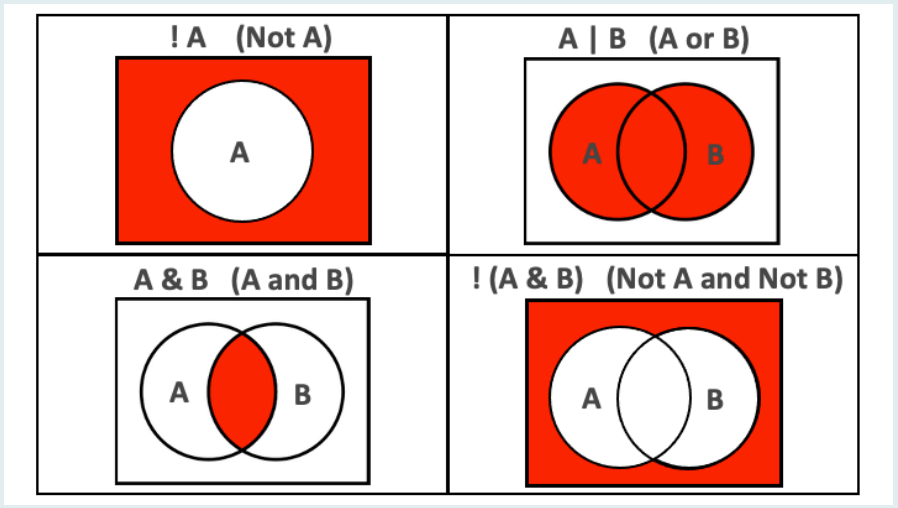

Fig: AND and OR operators visualized.

Let's see how to use these within `filter()`:

```
yao %>% filter(sex != "Male") ## keep rows where `sex` is not "Male"
```

```
yao %>% filter(age < 6) ## keep respondents under 6
```

```
yao %>% filter(age >= 70) ## keep respondents aged at least 70
```

```
## keep respondents whose highest education is "Primary" or "Secondary"
yao %>% filter(highest_education %in% c("Primary", "Secondary"))
```

PRACTICE (in RMD)

From `yao`, keep only respondents who were children (under 18).

With `%in%`, keep only respondents who live in the "Tsinga" or "Messa" neighborhoods.

Combining conditions with `&` and `|`

We can pass multiple conditions to a single `filter()` statement separated by commas:

```
## keep respondents who are pregnant and are ex-smokers
yao %>% filter(is_pregnant == "Yes", is_smoker == "Ex-smoker") ## only one row
```

When multiple conditions are separated by a comma, they are implicitly combined with an **and** (`&`).

It is best to replace the comma with `&` to make this more explicit.

```
## same result as before, but `&` is more explicit
yao %>% filter(is_pregnant == "Yes" & is_smoker == "Ex-smoker")
```

SIDE NOTE

Don't confuse:

- the "," in listing several conditions in filter `filter(A,B)` i.e. filter based on condition A and (`&`) condition B
- the "," in lists `c(A,B)` which is listing different components of the list (and has nothing to do with the `&` operator)

If we want to combine conditions with an **or**, we use the vertical bar symbol, `|`.

```
## respondents who are pregnant OR who are ex-smokers
yao %>% filter(is_pregnant == "Yes" | is_smoker == "Ex-smoker")
```

PRACTICE (in RMD)

Filter `yao` to only keep men who tested IgG positive.

Filter `yao` to children (under 18) and those whose highest education is primary school.

Negating conditions with `!`

To negate conditions, we wrap them in `!()`.

Below, we drop respondents who are children (less than 18 years) or who weigh less than 30kg:

```
## drop respondents < 18 years OR < 30 kg
yao %>% filter(!(age < 18 | weight_kg < 30))
```

The `!` operator is also used to negate `%in%` since R does not have an operator for **NOT in**.

```
## drop respondents whose highest education is NOT "Primary" or "Secondary"
yao %>% filter(!(highest_education %in% c("Primary", "Secondary")))
```

It is easier to read `filter()` statements as **keep** statements, to avoid confusion over whether we are filtering **in** or filtering **out**!

So the code below would read: "**keep** respondents who are under 18 or who weigh less than 30kg".

```
yao %>% filter(age < 18 | weight_kg < 30)
```

And when we wrap conditions in `!()`, we can then read `filter()` statements as **drop** statements.

So the code below would read: "**drop** respondents who are under 18 or who weigh less than 30kg".

```
yao %>% filter(!(age < 18 | weight_kg < 30))
```

Drop respondents who live in the Tsinga or Messa neighborhoods.

NA values

The relational operators introduced so far do not work with `NA`.

Let's make a data subset to illustrate this.

```
yao_mini <- yao %>%
  select(sex, is_pregnant) %>%
  slice(1,11,50,2) ## custom row order

yao_mini
```

In `yao_mini`, the last respondent has an `NA` for the `is_pregnant` column, because he is male.

Trying to select this row using `== NA` will not work.

```
yao_mini %>% filter(is_pregnant == NA) ## does not work
```

```
yao_mini %>% filter(is_pregnant == "NA") ## does not work
```

This is because `NA` is a non-existent value. So R cannot evaluate whether it is "equal to" or "not equal to" anything.

The special function `is.na()` is therefore necessary:

```
## keep rows where `is_pregnant` is NA
yao_mini %>% filter(is.na(is_pregnant))
```

This function can be negated with `!`:

```
## drop rows where `is_pregnant` is NA
yao_mini %>% filter(!is.na(is_pregnant))
```

SIDE NOTE

For tibbles, RStudio will highlight `NA` values bright red to distinguish them from other values:

```
# A tibble: 5 × 3
    age sex    is_pregnant
  <dbl> <chr>  <chr>
1    32 Male   NA
2    23 Female Yes
3    35 Male   NA
4    31 Female No
5    17 Female No response
```

A common error with `NA`

SIDE NOTE

`NA` values can be identified but any other encoding such as `"NA"` or `"NaN"`, which are encoded as strings, will be imperceptible to the functions (they are strings, like any others).

PRACTICE

(in RMD)

Keep all the responders who had missing records for the report of their smoking status

For some respondents the respiration rate, in breaths per minute, was recorded in the `respiration_frequency` column. From `yaounde`, drop those with a respiration frequency under 20.

Wrap Up !

Now you know the two essential verbs to `select()` columns and to `filter()` rows. This way you keep the variables you are interested in by selecting your columns and you keep the data entries you judge relevant by filtering your rows.

But what about modifying, transforming your data? We will learn about this in the next lesson. See you there!

Fig: Basic Data Wrangling: `select()` and `filter()`.

Contributors

The following team members contributed to this lesson:

LAURE VANCAUWENBERGHE

Data analyst, the GRAPH Network
A firm believer in science for good, striving to ally programming, health and education

ANDREE VALLE CAMPOS

R Developer and Instructor, the GRAPH Network

Motivated by reproducible science and education

KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about education

References

Some material in this lesson was adapted from the following sources:

- Horst, A. (2021). *Dplyr-learnr*. https://github.com/allisonhorst/dplyr-learnr (Original work published 2020)
- *Subset rows using column values—Filter*. (n.d.). Retrieved 12 January 2022, from https://dplyr.tidyverse.org/reference/filter.html

Artwork was adapted from:

- Horst, A. (2021). *R & stats illustrations by Allison Horst*. https://github.com/allisonhorst/stats-illustrations (Original work published 2018)