# Data visualisation - part 4: data processing with dplyr

*Hannah Meyer*

*01/08/2020*

## Contents



## 1 Introduction

In the previous tutorial, you have learned how to visualise your data, from simple scatter plots with default settings to compound figures with elaborate color schemes and labels. For all of these, we used the data from Smith *et al* (2004) (made available at the following http://www.antigenic-cartography.org/)). I mentioned briefly in the previous exercises, that I had formated the data for us to work with. What I experssed here in a half-sentence, usually contains a lot of work, often more than the actual analysis: cleaning your data, bringing it into the right format, checking it for sanity. In the following sections, we will learn how to use 'dplyr' to reformat data into a 'tidy' format that we can use for visualisation and analysis.

For a more detailed description and additional examples, refer to chapter 5 in Hadley Wickham book 'R for Data Science' Wickham and Grolemund (2017). I would generally highly recommend this book, as it introduces concepts we have discussed in this course and beyond - its online version is available for free here!

## 2 dplyr: a grammar of data manipulation

The dplyr package is a core member of the tidyverse. dplyr's core functionality relies on six functions that let us solve the majority of data reformating. These functions, often described as the 'verbs for the language of data manipulation', are:

- `select()`: to select a subset of variables;
- `arrange()`: to change the order of observations;
- `mutate()`: to add new variables that are functions of existing variables;
- `filter()`: to subset observations based on their values;
- `summarise()`: to summarise observations to a single row;
- `group_by()`: to change the unit of analysis from the complete dataset to individual groups.

All verbs work in a similar fashion: * their first argument is a `tibble` or `data.frame`; * the following arguments describe the action to apply to that `tibble`/`data.frame` by specifying the variable names * the result is a new `tibble`/`data.frame`, dependent on the initial input

*Note*: If you want to save the results of a `dplyr` function, you have to use the assignment operator `<-`, as `dplyr` function never modify their input data.

## 2.1 Change the order of observations with `arrange()`

Using `arrange()`, we can change the order of rows. As input, `arrange` takes a `tibble` and a set of variable names - at least one is required. It will re-arrange the observations of the `tibble` based on variable selected for re-ordering. If you provided more than one variable to order the tibble by, each additional variable will be used to break ties in the values of preceding variables.

As a default, `arrange` orders the variables in ascending order. To re-order in descending order use `desc()`:

## 2.2 Select a subset of variables with `select()`

`select()` let's us pick a subset of variables. In the most simple case, we specify the `tibble` from which we want to `select()` variables, followed by the names of the variables that we want to pick:

To exclude a variable, use `select()` with the variable name preceded by a minus `-`.

In addition to that, there a number of functions that can be used in conjunction with `select()` that let us select multiple variables with common elements in their name. These include: * starts_with("year"): matches names that begin with "year". * ends_with("coordinate"): matches names that end with "coordinate". * contains("coord"): matches names that contain "coord". * num_range("X", 1:3): matches X1, X2 and X3.

## 2.3 Add new variables with `mutate()`

To add new variables to a `tibble`, we use `mutate`. Again, `mutate` first expects the name of the `tibble` to which we want to add a variable, followed by the name of the new variable, an equal sign `=` and the data to add. The data has to have the same number of observations as our input `tibble` and can be created by using a transformation of an existing variable:

*Note*: `mutate()` adds the column at the end of the `tibble`; if you want it at a different position use `select` to reorder the variables afterwards.

## 2.4 Select of subset of observations with `filter()`

Using `select` we could create a subset of the input data by selecting variables. `filter()` allows us to subset our input data by observations. As a first argument it takes the name of the `tibble`; this is followed by expressions that filter observations based on their value in the specified variables. Filtering uses the standard set of comparison operators available in R.

### 2.4.1 Comparisons

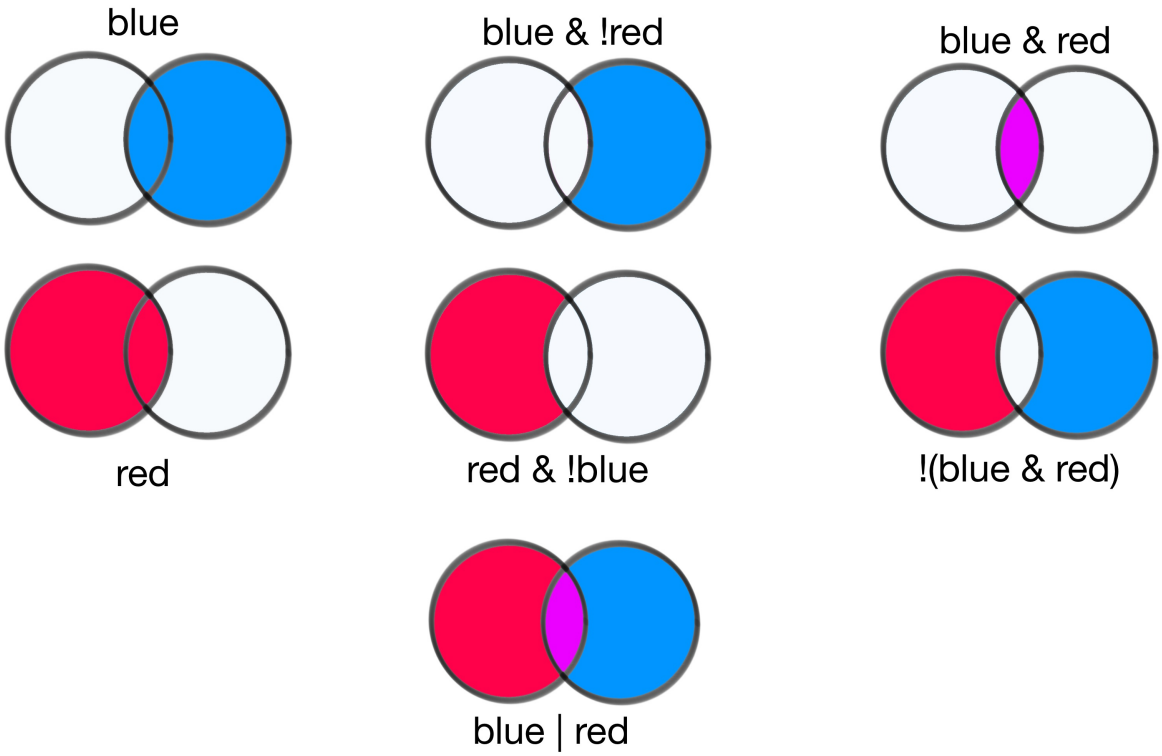
Comparison operators in R: * `==`: equal to * `!=`: not equal to * `>`: greater than * `>=`: greater or equal than * `<`: less than * `<=`: less or equal than

*Note*: When testing for equality, make sure to use `==` and not a simple `=`!

For instance, the following code let's us select all observations of viruses in circulation after 1988:

In addition to these comparison operators, we can also use boolean operators tp filter the input data. The simplest boolean operator is intrinsic to the `filter` function: multiple arguments to `filter()` are combined with and, i.e. every expression has to be true for an observation to be kept in the output. For instance:

Any more complicated combinations, like in circulation before 1987 but not BILTHOVEN, have to be constructed using the logical operators and `&`, or `|` and not `!`. The graphic below shows a complete overview of selection logical subsets of observations for two variables "red" and "blue":




### 2.5 Group analyses and summarise observations with `group_by` and `summarise()`

`group_by` and `summarise()` often go hand in hand: using `group_by`, we can first group observation based on values in the specified variable and then apply a summary statistic on this group. This might sound a bit complicated, so let's have a look at an example:

## 3 Creating a workflow with pipes

```
circulation_summary <- coord %>%
    group_by(cluster) %>%
    summarise(start=min(year),
              end=max(year),
              location=n_distinct(location)) %>%
    mutate(start=as.numeric(start)) %>%
    mutate(end=as.numeric(end)) %>%
    arrange(start) %>%
    mutate(cluster=factor(cluster)) %>%
    mutate(cluster=fct_inorder(cluster))

p <- ggplot(circulation_summary)
p + geom_segment(aes(x=start, xend=end, y=cluster, yend=cluster)) +
    xlab("Year") +
```

```
    ylab("Antigenic cluster") +
    theme_bw()
```

Wickham, Hadley, and Garrett Grolemund. 2017. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. 1st ed. O'Reilly Media, Inc.