R for life sciences. Chapter 7, dplyr and tidyr: tidyverse packages to manage data

2020-09-13

[**A new way of thinking R: The tidyverse**](#_83r95mbxzj5q) **2**

[Installation](#_isej2yhec45p) 3

[Cheatsheet](#_9llhzi24ok2g) 3

[The dplyr package](#_fjxvpnc4mnpl) 4

[Selecting columns: select()](#_t3639j6i75tk) 4

[filtering rows: filter()](#_mrutuynnbzy8) 5

[The magrittr pipe](#_xkh43ygon1kt) 6

[Add new variables: mutate()](#_7m71hqpjegat) 7

[Split-apply-combine data analysis: group\_by() and summarize() functions](#_q08swssnbec4) 7

[Order observations (rows): arrange()](#_6r58l0ebxnuw) 8

[The tidyr package](#_llbvwdw3xzlz) 9

[From long to wide format and viceversa: pivot\_longer() and pivot\_wider()](#_etrqav8eu6gd) 10

[Other tidyr functions](#_jqnhgprsc5wo) 12

[References](#_nrhs7wcw2ri) 12

[**Exercises**](#_8a14kjgss5p9) **12**

[**About this tutorial**](#_uime7c5gpt0a) **14**

# A new way of thinking R: The tidyverse

Bracket subsetting is handy, but it can be cumbersome and difficult to read, especially for complicated operations. The tidyverse package is an “umbrella-package” that installs several packages useful for data analysis which work together well such as tidyr, dplyr, ggplot2, tibble, etc. We have used already the ggplot2 package. Now we are going to learn about the two packages useful to work with data:

* **dplyr** is a package for making tabular data manipulation easier.
* **tidyr** enables you to swiftly convert between different data formats.

Commands from both packages work nicely with the magritte pipe (%>%) which is supposed to make code more readable.

## Installation

This packages are in cran, so it is easy to install them:

*#### Install packages --------------*  
install.packages(c("dplyr", "tidyr"), dependencies = TRUE)

Or you might want to install the whole tidyverse:

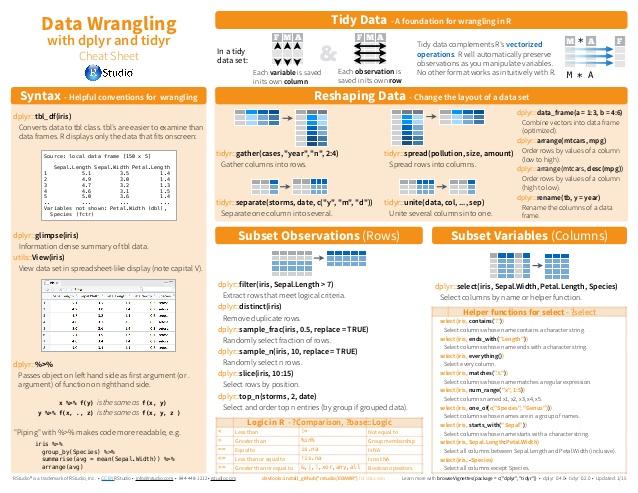
*#### Install packages --------------*  
install.packages(c("tidyverse"), dependencies = TRUE)

Remember that in your script the install.packages() commands should allways be commented and at the beginning of the script, and installed packages have to be loaded with library() before use.

*#### Packages ------------*  
*# install.packages(c("dplyr", "tidyr"), dependencies = TRUE)*  
  
library(dplyr)  
library(tidyr)

## Cheatsheet

There is a good cheatsheet here.

[](https://rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf)

## The *dplyr* package

The package **dplyr** provides easy tools for the most common data manipulation tasks. It can work with gigantic databases which enables to conduct queries directly, and pull back into R only what you need for analysis.

The package includes some commands named as verbs that coincide with the most common actions over data:

* select(): subset columns.
* filter(): subset rows on conditions.
* mutate(): create new columns by using information from other columns.
* group\_by() and summarize(): create summary statisitcs on grouped data.
* arrange(): order rows.

To work with dplyr we have to keep in mind that:

* The first argument is always a data frame.
* Rest of arguments indicate what do we want to do with this dataframe.
* Result will always have also data frame structure.

### Selecting columns: select()

This action consist in choosing a subset of variables (columns) from the data frame.

*### Select columns -----------------------*  
select(iris, Sepal.Length, Sepal.Width)

## Sepal.Length Sepal.Width  
## 1 5.1 3.5  
## 2 4.9 3.0  
...

It is possible to select a range of variables using :. For example to choose from Petal.Length to Sepal.Length.

*### Select columns by range*  
select(iris, Petal.Length:Sepal.Length)

## Petal.Length Sepal.Width Sepal.Length  
## 1 1.4 3.5 5.1  
## 2 1.4 3.0 4.9  
...

It is also possible to select all the variables except the ones with - before them.

*### Exclude variables from selection*  
select(iris, -Species, -Petal.Length)

## Sepal.Length Sepal.Width Petal.Width  
## 1 5.1 3.5 0.2  
## 2 4.9 3.0 0.2  
...

Other posibility is select variables with a pattern.

*### Use pattern to select*  
select(iris, contains('Petal'))

## Petal.Length Petal.Width  
## 1 1.4 0.2  
## 2 1.4 0.2  
...

Instead of contains(), you could use in a simmilar way: starts\_with(), ends\_with() or matches().

### filtering rows: filter()

To choose observations (rows) based on a specific criteria, use filter(). Three examples: the first command select all iris from setosa species, the second from setosa or virginica species and the third the iris from setosa with sepal length smaller than 5 mm.

*### Select rows ---------------------------*  
*### Select iris from setosa species*  
filter(iris, Species == 'setosa')  
  
*### Select iris from setosa or virginica species*  
filter(iris, Species == 'setosa' | Species == 'virginica')  
  
*### Select iris from setosa with sepal length smaller than 5 mm*   
filter(iris, Species == 'setosa', Sepal.Length < 5)

### The magrittr pipe

If you want to do several things to the same data frame, for example select and filter, there are three ways to do it: use nested functions, intermediate steps, or pipes.

You can nest functions (i.e. one function inside of another), like this:

*### Example of nesting functions --------------------*  
i <- select(filter(iris, Sepal.Length < 5),   
 Sepal.Width, Sepal.Length, Species)

This is handy, but can be difficult to read if too many functions are nested. Try don’t to use it.

With intermediate steps, you create a temporary data frame and use that as input to the next function, like this:

*### Example using temporary object i ------------------------*  
i <- filter(iris, Sepal.Length < 5)  
i <- select(i, Sepal.Width, Sepal.Length, Species)

This is readable, but can clutter up your workspace if you use different names for each step. If using the same name code is very simmilar to piped one.

The last option are pipes, which are a recent addition to R (they are not in base R and you need to install packages). Pipes let you take the output of one function and send it directly to the next, which is useful when you need to do many things to the same dataset. Pipes in R look like %>% and are made available via the magrittr package, installed automatically with dplyr. If you use RStudio, you can type the pipe with **Ctrl + Shift + M** if you have a PC or *Cmd + Shift + M* if you have a Mac.

*### Example using Magritte pipe ------------------------*  
i <- iris %>%   
 filter(Sepal.Length < 5) %>%   
 select(Sepal.Width, Sepal.Length, Species)

Some may find it helpful to read the pipe like the word “then”. In the above code, we use the pipe to send the iris dataset first through filter() to keep rows where sepal length is less than 5, then through select() to keep only the Sepal.Width, Sepal.Length and Species columns.

Since %>% takes the object on its left and passes it as the first argument to the function on its right, we don’t need to explicitly include the data frame as an argument to the filter() and select() functions any more.

You may also use non-dplyr functions with pipes.

*### Use non-dplyr functions*  
iris %>%   
 filter(Sepal.Length < 5) %>%   
 select(Sepal.Width, Sepal.Length, Species) %>%   
 head(4)

## Sepal.Width Sepal.Length Species  
## 1 3.0 4.9 setosa  
## 2 3.2 4.7 setosa  
## 3 3.1 4.6 setosa  
## 4 3.4 4.6 setosa

### Add new variables: mutate()

Frequently you’ll want to create new columns based on the values in existing columns, for example to do unit conversions, or to find the ratio of values in two columns. For this we’ll use mutate().

To create a new variable with the petals shape as the ratio between width and length, and select only the new variables and *Species*.

*### Create new variables*  
iris %>%   
 mutate(Petal.Shape = Petal.Width / Petal.Length,  
 Sepal.Shape = Sepal.Width / Sepal.Length) %>%   
 select(Species, Petal.Shape, Sepal.Shape)

## Species Petal.Shape Sepal.Shape  
## 1 setosa 0.14285714 0.6862745  
## 2 setosa 0.14285714 0.6122449  
## 3 setosa 0.15384615 0.6808511  
...

### Split-apply-combine data analysis: group\_by() and summarize() functions

Many data analysis tasks can be approached using the split-apply-combine paradigm: split the data into groups, apply some analysis to each group, and then combine the results. dplyr makes this very easy through the use of the group\_by() and summarize() functions.

group\_by() collapses each group into a single-row summary of that group. It takes as arguments the column names that contain the categorical variables for which you want to calculate the summary statistics. So to compute the mean of Petal.Length by Species:

*### Use of group\_by and summarize ------------------*  
iris %>%   
 group\_by(Species) %>%   
 summarise(Mean.Petal.Length = mean(Petal.Length))

## `summarise()` ungrouping output (override with `.groups` argument)

## # A tibble: 3 x 2  
## Species Mean.Petal.Length  
## <fct> <dbl>  
## 1 setosa 1.46  
## 2 versicolor 4.26  
## 3 virginica 5.55

You can also group by multiple columns and create new variables.

*### grouping by multiple columns*  
iris %>%   
 mutate(Petal.Long = Petal.Length > 5) %>%   
 group\_by(Species, Petal.Long) %>%   
 summarise(Mean.Petal.Length = mean(Petal.Length),  
 n.Petals = length(Petal.Length),  
 sd.Petal.Length = sd(Petal.Length),  
 SE.Petal.Length = sd(Petal.Length) / sqrt(length(Petal.Length)))

## `summarise()` regrouping output by 'Species' (override with `.groups` argument)

## # A tibble: 5 x 6  
## # Groups: Species [3]  
## Species Petal.Long Mean.Petal.Length n.Petals sd.Petal.Length SE.Petal.Length  
## <fct> <lgl> <dbl> <int> <dbl> <dbl>  
## 1 setosa FALSE 1.46 50 0.174 0.0246  
## 2 versicolor FALSE 4.24 49 0.459 0.0655  
## 3 versicolor TRUE 5.1 1 NA NA   
## 4 virginica FALSE 4.87 9 0.158 0.0527  
## 5 virginica TRUE 5.70 41 0.489 0.0764

Check that there is an NA value in sd and SE variables, because the number of long petals in versicolor is only one, and therefore there is not standard deviation.

You might have noticed that the output of dplyr verbs is always a “tibble”. It is the tidyverse format for data.frame.

### Order observations (rows): arrange()

It is sometimes useful to arrange the rows of a data frame. For instance, to arrange the result of a query to inspect the values to put the longer petals first (the - inverts the order):

*### Order observations --------------*  
iris %>%   
 group\_by(Species) %>%   
 summarise(Mean.Petal.Length = mean(Petal.Length),  
 n.Petals = length(Petal.Length),  
 sd.Petal.Length = sd(Petal.Length),  
 SE.Petal.Length = sd(Petal.Length) / sqrt(length(Petal.Length))) %>%   
 arrange(-Mean.Petal.Length)

## `summarise()` ungrouping output (override with `.groups` argument)

## # A tibble: 3 x 5  
## Species Mean.Petal.Length n.Petals sd.Petal.Length SE.Petal.Length  
## <fct> <dbl> <int> <dbl> <dbl>  
## 1 virginica 5.55 50 0.552 0.0780  
## 2 versicolor 4.26 50 0.470 0.0665  
## 3 setosa 1.46 50 0.174 0.0246

## The *tidyr* package

The goal of tidyr is to help you create tidy data. Tidy data is data where:

1. Every column is variable.
2. Every row is an observation.
3. Every cell is a single value.

It seems easy. Now look at this data frame ¿Is it tidy?

*### Difference between long and wide format in tidy data -------------------*  
*### Number of insects in each trap each year. Wide format.*   
(d1 <- tibble(  
 trap = 1:10,  
 Year.2013 = c(12, 10, 5, 3, 15, 11, 12, 10, 7, 5),  
 Year.2014 = c(12, 3, 15, 1, 13, 4, 1, 16, 7, 13),  
 Year.2015 = c(12, 0, 5, 3, 15, 11, 12, 10, 7, 15),  
 Year.2016 = c(2, 10, 6, 3, 1, 11, 12, 10, 7, 10)))

## # A tibble: 10 x 5  
## trap Year.2013 Year.2014 Year.2015 Year.2016  
## <int> <dbl> <dbl> <dbl> <dbl>  
## 1 1 12 12 12 2  
## 2 2 10 3 0 10  
## 3 3 5 15 5 6  
## 4 4 3 1 3 3  
## 5 5 15 13 15 1  
## 6 6 11 4 11 11  
## 7 7 12 1 12 12  
## 8 8 10 16 10 10  
## 9 9 7 7 7 7  
## 10 10 5 13 15 10

Of course NOT. Year should be a variable with its own column. Also it should be numeric or Date type. Data should be given in this way:

*### Number of insects in each trap each year. Long format.*  
tibble(  
 trap = rep(1:10, 4),  
 year = c(rep(2013, 10), rep(2014, 10), rep(2015, 10), rep(2016, 10)),  
 numberInsects = c(12, 10, 5, 3, 15, 11, 12, 10, 7, 5,  
 12, 3, 15, 1, 13, 4, 1, 16, 7, 13,  
 12, 0, 5, 3, 15, 11, 12, 10, 7, 15,  
 2, 10, 6, 3, 1, 11, 12, 10, 7, 10))

## # A tibble: 40 x 3  
## trap year numberInsects  
## <int> <dbl> <dbl>  
## 1 1 2013 12  
## 2 2 2013 10  
## 3 3 2013 5  
## 4 4 2013 3  
## 5 5 2013 15  
## 6 6 2013 11  
## 7 7 2013 12  
## 8 8 2013 10  
## 9 9 2013 7  
## 10 10 2013 5  
## # … with 30 more rows

But if the data are already in one format you can convert to the other easily “pivotting”.

### From long to wide format and viceversa: pivot\_longer() and pivot\_wider()

From tidyr 1.0.0 “Pivotting” pivot\_longer() and pivot\_wider(), replace the older spread() and gather() functions. Which converts between long and wide forms.

From the previous example:

*### From wide to long format ------*  
d1.long <- d1 %>%   
 pivot\_longer(contains("Year"),   
 names\_to = "Year",   
 values\_to = "insects")  
  
*### Same cleaner ---------*  
d1.long <- d1 %>%   
 pivot\_longer(contains("Year"),   
 names\_to = "Year",   
 values\_to = "insects") %>%   
 separate(Year, into = c("na", "year"), convert = TRUE) %>%   
 select(-na)  
  
*### output*   
head(d1.long, 4)

## # A tibble: 4 x 3  
## trap year insects  
## <int> <int> <dbl>  
## 1 1 2013 12  
## 2 1 2014 12  
## 3 1 2015 12  
## 4 1 2016 2

*### Wide format again*   
d1.long %>%   
 pivot\_wider(names\_from = year, names\_prefix = "Year.",   
 values\_from = insects)

## # A tibble: 10 x 5  
## trap Year.2013 Year.2014 Year.2015 Year.2016  
## <int> <dbl> <dbl> <dbl> <dbl>  
## 1 1 12 12 12 2  
## 2 2 10 3 0 10  
## 3 3 5 15 5 6  
## 4 4 3 1 3 3  
## 5 5 15 13 15 1  
## 6 6 11 4 11 11  
## 7 7 12 1 12 12  
## 8 8 10 16 10 10  
## 9 9 7 7 7 7  
## 10 10 5 13 15 10

This would be useful for example to solve the exercise from the second tutorial:

*### Create the summary table ------*  
r2 <- iris %>%   
 pivot\_longer(Sepal.Length:Petal.Width,   
 names\_to = "NumVariable",   
 values\_to = "value") %>%   
 group\_by(Species, NumVariable) %>%   
 summarise(mean = mean(value),  
 st.error = sd(value) / sqrt(length(value)),  
 median = median(value),  
 maxim = max(value),  
 minim = min(value))

## `summarise()` regrouping output by 'Species' (override with `.groups` argument)

*### Print a nice table in rmarkdown ------*  
knitr::kable(r2, caption = "Summary table for numeric variables from iris data set.")

Summary table for numeric variables from iris data set.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Species | NumVariable | mean | st.error | median | maxim | minim |
| setosa | Petal.Length | 1.462 | 0.0245598 | 1.50 | 1.9 | 1.0 |
| setosa | Petal.Width | 0.246 | 0.0149038 | 0.20 | 0.6 | 0.1 |
| setosa | Sepal.Length | 5.006 | 0.0498496 | 5.00 | 5.8 | 4.3 |
| setosa | Sepal.Width | 3.428 | 0.0536078 | 3.40 | 4.4 | 2.3 |
| versicolor | Petal.Length | 4.260 | 0.0664554 | 4.35 | 5.1 | 3.0 |
| versicolor | Petal.Width | 1.326 | 0.0279665 | 1.30 | 1.8 | 1.0 |
| versicolor | Sepal.Length | 5.936 | 0.0729976 | 5.90 | 7.0 | 4.9 |
| versicolor | Sepal.Width | 2.770 | 0.0443778 | 2.80 | 3.4 | 2.0 |
| virginica | Petal.Length | 5.552 | 0.0780497 | 5.55 | 6.9 | 4.5 |
| virginica | Petal.Width | 2.026 | 0.0388414 | 2.00 | 2.5 | 1.4 |
| virginica | Sepal.Length | 6.588 | 0.0899270 | 6.50 | 7.9 | 4.9 |
| virginica | Sepal.Width | 2.974 | 0.0456079 | 3.00 | 3.8 | 2.2 |

See vignette("pivot") for more details.

### Other tidyr functions

You won’t need them by now, but is good that you know they exist, so you know where to look when needed.

* “Rectangling”, which turns deeply nested lists (as from JSON) into tidy tibbles. See unnest\_longer(), unnest\_wider(), hoist(), and vignette("rectangle") for more details.
* Nesting converts grouped data to a form where each group becomes a single row containing a nested data frame, and unnesting does the opposite. See nest(), unnest(), and vignette("nest") for more details.
* Splitting and combining character columns. Use separate() and extract() to pull a single character column into multiple columns; use unite() to combine multiple columns into a single character column.
* Make implicit missing values explicit with complete(); make explicit missing values implicit with drop\_na(); replace missing values with next/previous value with fill(), or a known value with replace\_na().

## References

* <https://dplyr.tidyverse.org/>
* <https://tidyr.tidyverse.org/>
* <https://datacarpentry.org/R-ecology-lesson/03-dplyr.html>

# Exercises

1. In a new Rmarkdown document, using pipes, dplyr and ggplot2, do a barplot for the mean of each of the numeric variables in “iris”. Add error bars to the barplot with the standard error.
2. Do anova test and posthoc and represent in each figure if diferences between species are significant or not in each case.
3. Is a barplot the best way to represent these data? Which would be the best one? Do it.

# About this tutorial

Cite as: Alfonso Garmendia (2020) R for life sciences. Chapter 7, dplyr and tidyr: tidyverse packages to manage data. <http://personales.upv.es/algarsal/R-tutorials/07_Tutorial-7_R-dplyr-tidyr.html>.

Available also in other formats (pdf, docx, …): <https://drive.google.com/drive/folders/19w914WCg8BVTVBE_zpgShmg2vpjguV1e?usp=sharing>.

Other simmilar tutorials: <https://garmendia.blogs.upv.es/r-lecture-notes/>

Originals are in bitbucket repository: <https://bitbucket.org/alfonsogar/tea_daa_tutorials>.

Document written in [Rmarkdown](http://rmarkdown.rstudio.com/), using [Rstudio](https://www.rstudio.com/).

* System: R version 4.0.2 (2020-06-22) x86\_64-pc-linux-gnu (64-bit) Ubuntu 18.04.5 LTS.
* Base packages: stats 4.0.2, graphics 4.0.2, grDevices 4.0.2, utils 4.0.2, datasets 4.0.2, methods 4.0.2, base 4.0.2.
* Other loaded packages: tidyr 1.1.0, dplyr 1.0.0, ggplot2 3.3.2, knitr 1.29, rmarkdown 2.3, googledrive 1.0.1.



This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](http://creativecommons.org/licenses/by-nc-sa/4.0/).