Ajuntament de Barcelona

Data Massage - R

Introduction Data Massage with R

IT Academy

Barcelona Activa

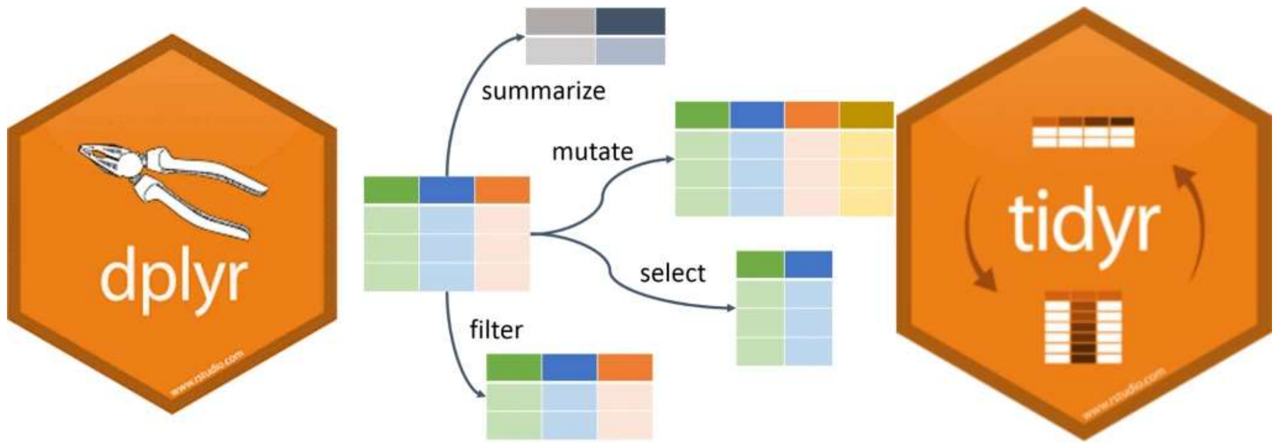
Unió Europea
Fons Europeu de Desenvolupament Regional

Index

Data massaging with R

Data Massaging is the process of extracting data to remove unneeded information or clean up a dataset to get into a usable format.

- To do it in R, we can use **dplyr** and **tidyr** packages

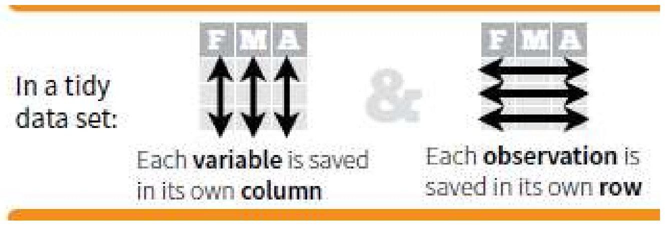


What is tidyr?

tidyr is a package that makes it easy to “tidy” your data.

Tidy data is data that’s easy to work with: it’s easy to munge (with **dplyr**), visualise (with **ggplot2** or ggvis) and model (with R’s hundreds of modelling packages). The two most important properties of tidy data are:

- Each variable forms a column
- Each observation forms a row

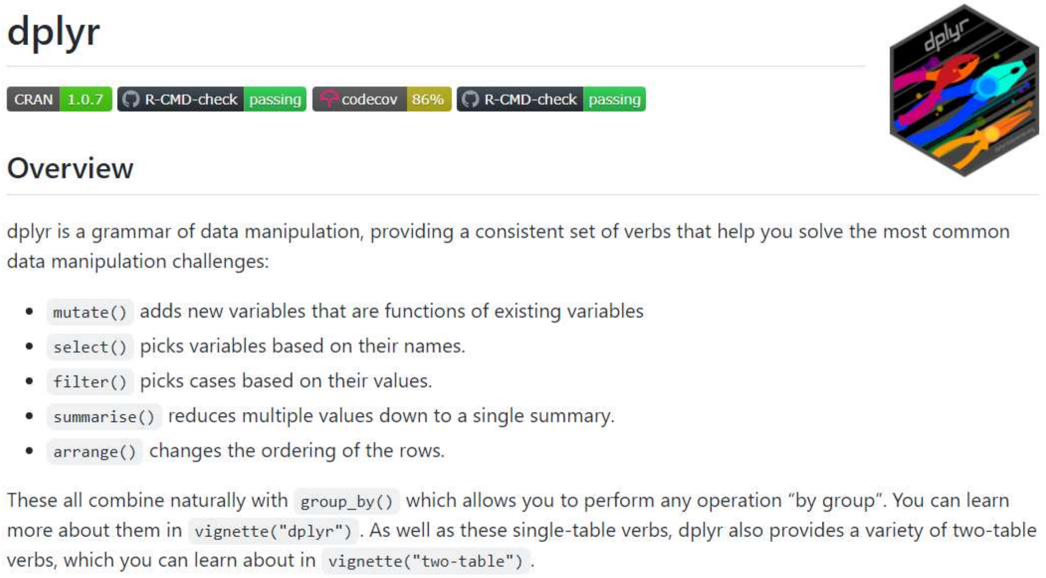

In a tidy data set:
F M A
Each variable is saved in its own column
&
F M A
Each observation is saved in its own row

Tidy data graphic from R for Data Science

What is dplyr?

dplyr is a package which provides a set of tools for efficiently manipulating datasets in R

You can track developement progress at https://github.com/tidyverse/dplyr



dplyr

CRAN 1.0.7 | R-CMD-check passing | codecov 86% | R-CMD-check passing

Overview

dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:

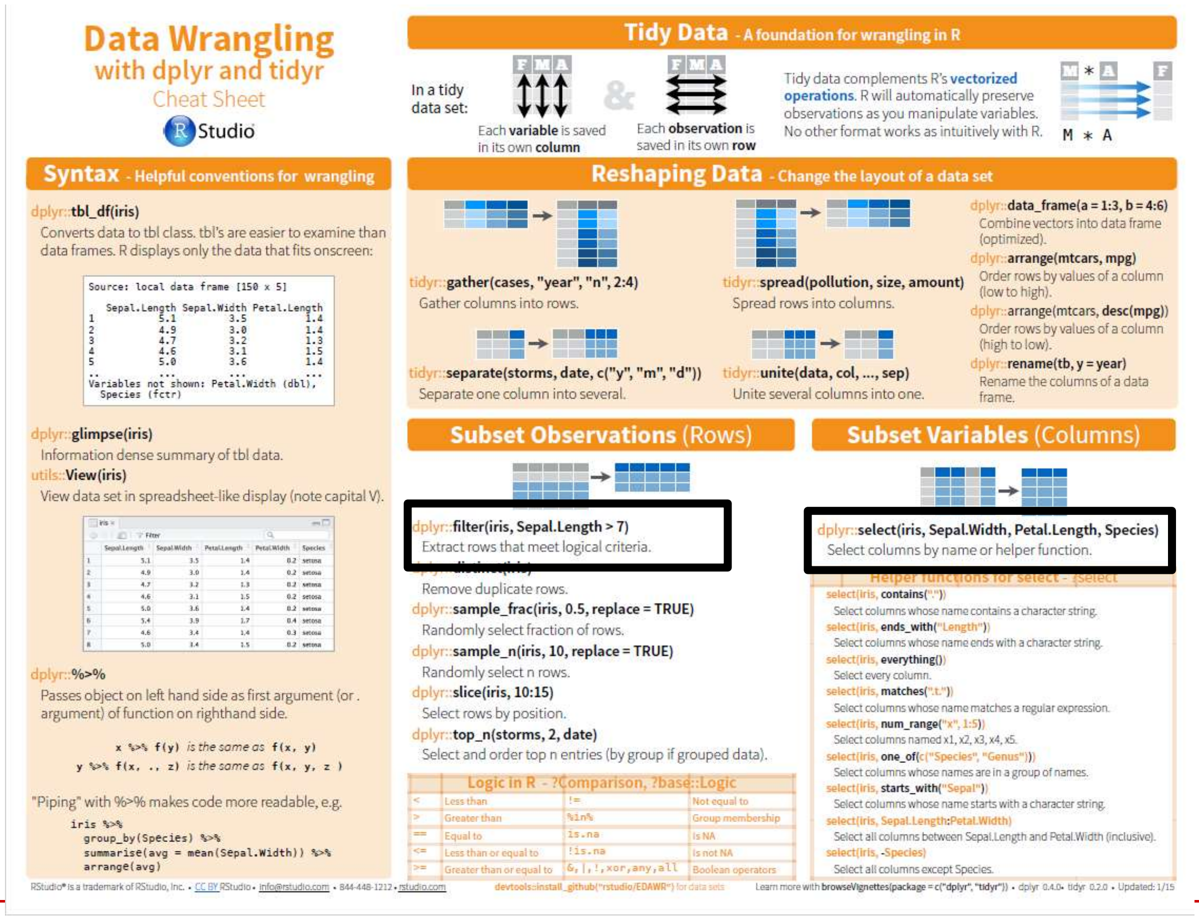
- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names.
- `filter()` picks cases based on their values.
- `summarise()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.

These all combine naturally with `group_by()` which allows you to perform any operation "by group". You can learn more about them in `vignette("dplyr")`. As well as these single-table verbs, dplyr also provides a variety of two-table verbs, which you can learn about in `vignette("two-table")`.

Basics from tidyr and dplyr

Package	Function	Use
dplyr	select	select variables/columns
dplyr	filter	select observations/rows
dplyr	Mutate	transform or recode variables
dplyr	summarize / summarise	summarize data
dplyr	group_by	identify subgroups for further processing
tidyr	gather	convert wide format dataset to long format
tidyr	spread	convert long format dataset to wide format

You are going to use:



Data Wrangling
with dplyr and tidyr
Cheat Sheet
RStudio

Syntax - Helpful conventions for wrangling

dplyr::**tbl_df(iris)**
Converts data to tbl class. tbl's are easier to examine than data frames. R displays only the data that fits onscreen:

```
Source: local data frame [150 x 5]

  Sepal.Length Sepal.Width Petal.Length
1          5.1         3.5          1.4
2          4.9         3.0          1.4
3          4.7         3.2          1.3
4          4.6         3.1          1.5
5          5.0         3.6          1.4
..         ...         ...          ...
Variables not shown: Petal.Width (dbl),
  Species (fctr)
```

dplyr::**glimpse(iris)**
Information dense summary of tbl data.
utils::**View(iris)**
View data set in spreadsheet-like display (note capital V).

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa

dplyr::**%>%**
Passes object on left hand side as first argument (or . argument) of function on righthand side.

```
   x %>% f(y)   is the same as   f(x, y)
y %>% f(x, ., z)   is the same as   f(x, y, z )
```

"Piping" with %>% makes code more readable, e.g.

```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

Tidy Data - A foundation for wrangling in R

In a tidy data set: Each **variable** is saved in its own **column** & Each **observation** is saved in its own **row**

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.

M * A

Reshaping Data - Change the layout of a data set

tidyr::**gather(cases, "year", "n", 2:4)**
Gather columns into rows.

tidyr::**spread(pollution, size, amount)**
Spread rows into columns.

tidyr::**separate(storms, date, c("y", "m", "d"))**
Separate one column into several.

tidyr::**unite(data, col, ..., sep)**
Unite several columns into one.

dplyr::**data_frame(a = 1:3, b = 4:6)**
Combine vectors into data frame (optimized).
dplyr::**arrange(mtcars, mpg)**
Order rows by values of a column (low to high).
dplyr::**arrange(mtcars, desc(mpg))**
Order rows by values of a column (high to low).
dplyr::**rename(tb, y = year)**
Rename the columns of a data frame.

Subset Observations (Rows)

dplyr::**filter(iris, Sepal.Length > 7)**
Extract rows that meet logical criteria.
dplyr::**distinct(iris)**
Remove duplicate rows.
dplyr::**sample_frac(iris, 0.5, replace = TRUE)**
Randomly select fraction of rows.
dplyr::**sample_n(iris, 10, replace = TRUE)**
Randomly select n rows.
dplyr::**slice(iris, 10:15)**
Select rows by position.
dplyr::**top_n(storms, 2, date)**
Select and order top n entries (by group if grouped data).

Logic in R - ?Comparison, ?base::Logic			
<	Less than	!=	Not equal to
>	Greater than	%in%	Group membership
==	Equal to	is.na	Is NA
<=	Less than or equal to	!is.na	Is not NA
>=	Greater than or equal to	&,|,!,xor,any,all	Boolean operators

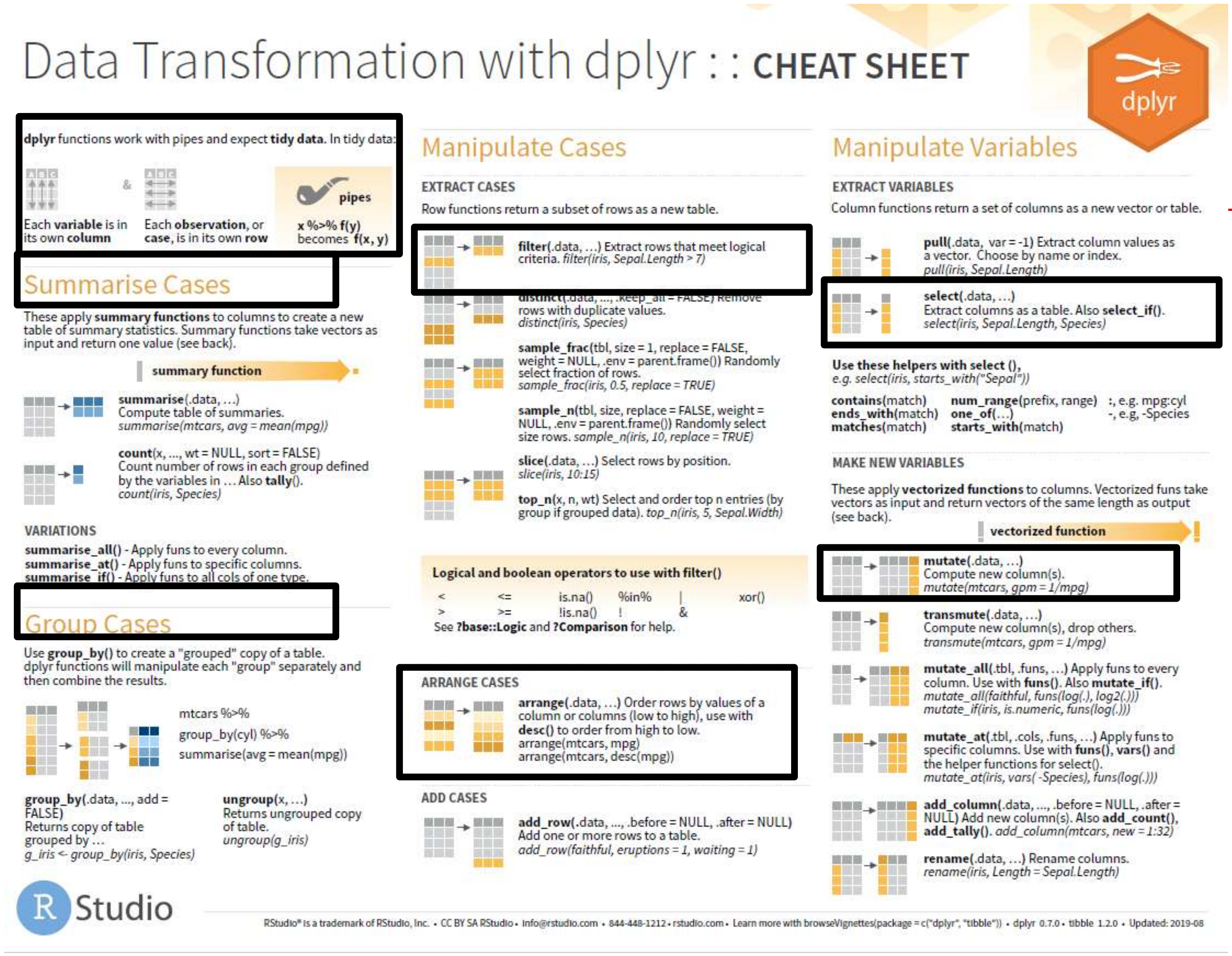
Subset Variables (Columns)

dplyr::**select(iris, Sepal.Width, Petal.Length, Species)**
Select columns by name or helper function.

Helper functions for select - ?select

select(iris, **contains(".")**)
Select columns whose name contains a character string.
select(iris, **ends_with("Length")**)
Select columns whose name ends with a character string.
select(iris, **everything()**)
Select every column.
select(iris, **matches(".t.")**)
Select columns whose name matches a regular expression.
select(iris, **num_range("x", 1:5)**)
Select columns named x1, x2, x3, x4, x5.
select(iris, **one_of(c("Species", "Genus"))**)
Select columns whose names are in a group of names.
select(iris, **starts_with("Sepal")**)
Select columns whose name starts with a character string.
select(iris, Sepal.Length**:**Petal.Width)
Select all columns between Sepal.Length and Petal.Width (inclusive).
select(iris, -Species)
Select all columns except Species.

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com   devtools::install_github("rstudio/EDAWR") for data sets   Learn more with browseVignettes(package = c("dplyr", "tidyr")) • dplyr 0.4.0 • tidyr 0.2.0 • Updated: 1/15



Data Transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

pipes

x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

```
mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))
```

group_by(.data, ..., add = FALSE)
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …)
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …) Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.
distinct(iris, Species)

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …) Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, ..., .before = NULL, .after = NULL)
Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1) Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)

select(.data, …)
Extract columns as a table. Also **select_if**().
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match)	**num_range**(prefix, range)	:, e.g. mpg:cyl
ends_with(match)	**one_of**(…)	-, e.g, -Species
matches(match)	**starts_with**(match)	

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, …)
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, …)
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …) Apply funs to every column. Use with **funs()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))

mutate_at(.tbl, .cols, .funs, …) Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))

add_column(.data, ..., .before = NULL, .after = NULL) Add new column(s). Also **add_count()**, **add_tally()**. *add_column(mtcars, new = 1:32)*

rename(.data, …) Rename columns.
rename(iris, Length = Sepal.Length)

RStudio

Barcelona Activa

Unió Europea
Fons Europeu
de Desenvolupament Regional

Barcelona
Activa

30 ANYS

barcelona.cat/barcelonactiva