Data Science with R in the *tidyverse*

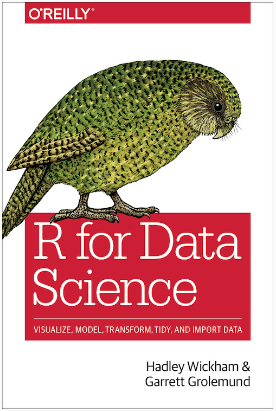
Workshop material: https://git.io/comos-r

Matteo Sostero

May 16-17-18, 2018

Sant'Anna School of Advanced Studies, Pisa

References



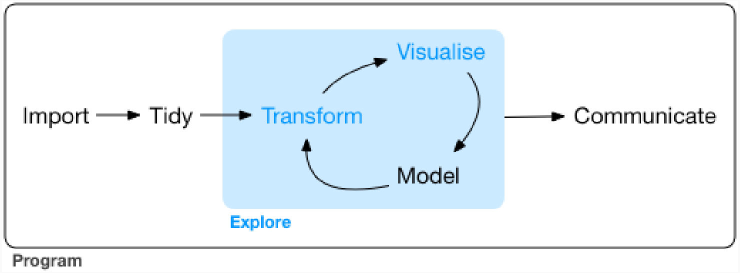
- Workshop material: https://git.io/comos-r
- RStudio *Cheat sheets:* https://rstudio.com/resources/cheatsheets/
- Book *R for Data Science* by Garrett Grolemund, Hadley Wickham:
    - Online version: http://r4ds.had.co.nz/
    - Paperback: *R for Data Science*, O'Reilly Media, 2017.

Workshop outline

- Today:
    - Overview of Data Science workflow
    - *tidyverse* fundamentals
    - Data transformation with *dplyr*
    - Merging datasets with *join* operations
    - Data input strategies with *readr*
- Tomorrow:
    - String manipulation with *stringr*
    - Functional programming with *purrr*
    - Handling categorical variables with *forcats*
    - Data tidying with *tidyr*
- Friday:
    - Data visualization with *ggplot2*
    - Model estimation and selection with *broom*
    - Web scraping with *rvest*
    - ? More dataviz? (maps)

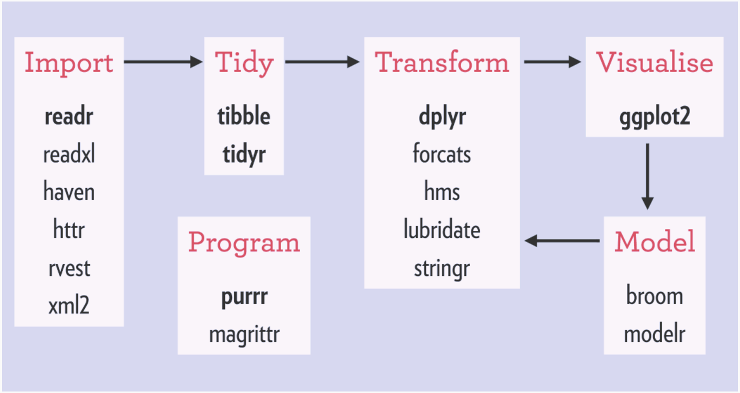
Data Science Workflow

The Data Science workflow



Credit: *R for Data Science*

The Tidyverse ecosystem



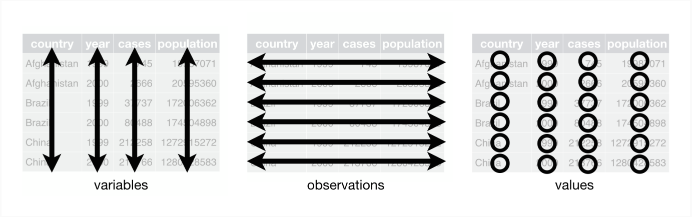
Credit: Joseph Rickert

Exploring the tidyverse

Tidyverse philosophy

The tidyverse is organised around a few principles:

- **Tidy data**: each variable is a column, each observation is a row, and each type of observational unit is a table (see article by Hadley);



Tidy data graphic from *R for Data Science*

- **Code readability**: consistent, expressive, verb-oriented syntax;
- **Functional programming**: functions applied to all elements of objects (no iterators); pass data along function pipeline %>%;
- **Compatibility**: classes and functions are mostly backward compatible with "Base R".

Tidyverse philosophical differences

Key differences with "Base R"

- `tibble` replaces `data.frame` class for rectangular datasets:
  - better preview printing;
  - "lazy and surly": less type coercion (strings not converted to factors); doesn't change variable names;
  - doesn't use `row.names`
- style: function names are in `snake_case`
  e.g.: `read_csv()` instead of `read.csv()`
- data is first argument of functions.

Building the pipeline

Tidyverse uses the *pipe* `%>%` to concatenate operations on data.

`%>%` builds chains of function by passing ("piping") the **output** of one function (ie, data) as **input** of the next function.

Pseudocode: pipeline for baking a cake

```
ingredients %>%             {flour, water, eggs}
  blend() %>%               dough
  cook()                    whole cake
```

The pipe `%>%` composes functions: `x %>% f(.) %>% g(.)` ≡ `g(f(x))`

The point `.` is an explicit placeholder for the argument being piped.
Useful if a function `h(z,x)` has more than one argument (or data is not the first argument): `x %>% h(z, .)` ≡ `h(z, x)`.
In most cases it can safely be omitted: `x %>% f() %>% g()`

RStudio shortcut for `%>%`: ctrl + ⇧ + M (or ⌘ + ⇧ + M on macOS).

Data manipulation with *dplyr*

In the tidyverse, *dplyr* provides a grammar of data manipulation, with 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 to perform any operation by groups of values.

R objects demystified

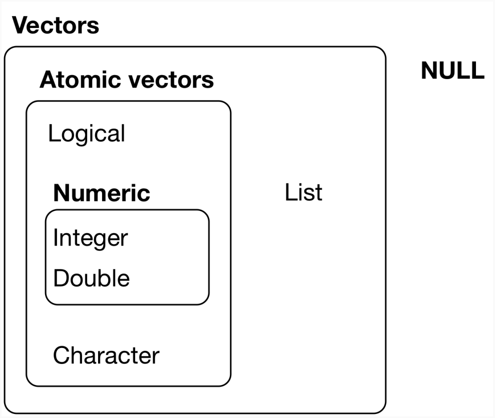
Data structure

Vectors are a fundamental object class in R and come in two types:

- **Atomic vector** (homogeneous) of type:
  1. *double*: most numbers
  2. *integer*: integer numbers
  3. *logical*: `TRUE, FALSE, NA`
  4. *character*: strings
  5. *complex*
  6. *raw*

  Integer and double vectors are also known as *numeric* vectors.
- **List** (heterogeneous) which are sometimes called *recursive vectors* because lists can contain other lists.

Data structure



Mapping functions to data

Functional programming with *purrr*

In *purrr*, the function `map(.x, .f)` maps (ie, applies) a **function** `.f` to every element of a **list** or **atomic vector** `.x`

(loose) mathematical definition:

- Given a set (in R, a `vector`) $X = \{x_1, x_2, \dots, x_n\}$
- And a function $f(x) \, : \, x \in X \mapsto f(x)$

$$\mathbf{Map}(X, f) \, : \, \left\{f(x_1), f(x_2), \dots, f(x_n)\right\}$$