

# R for Data Science

Using the *tidyverse*

---

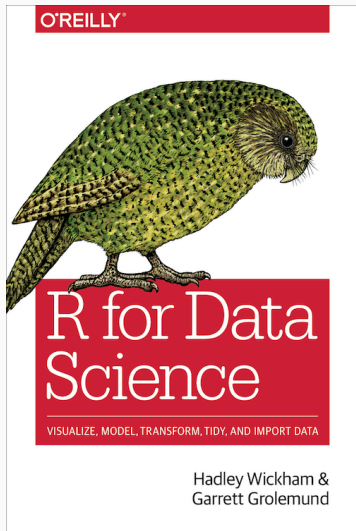
Matteo Sostero

May 18, 2018

Workshop material: [bit.ly/comos-r](https://bit.ly/comos-r)

Sant'Anna School of Advanced Studies, Pisa

# References

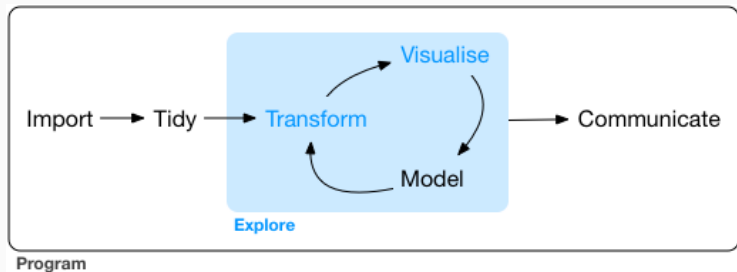


- Workshop material: [bit.ly/comos-r](https://bit.ly/comos-r)
- RStudio *Cheat sheets*  
[rstudio.com/resources/cheatsheets/](https://rstudio.com/resources/cheatsheets/)
- Book *R for Data Science* (Garrett Golemund, Hadley Wickham):
  - Online version:  
<http://r4ds.had.co.nz/>
  - Paperback: *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*, O'Reilly Media, 2017.

# Workshop outline

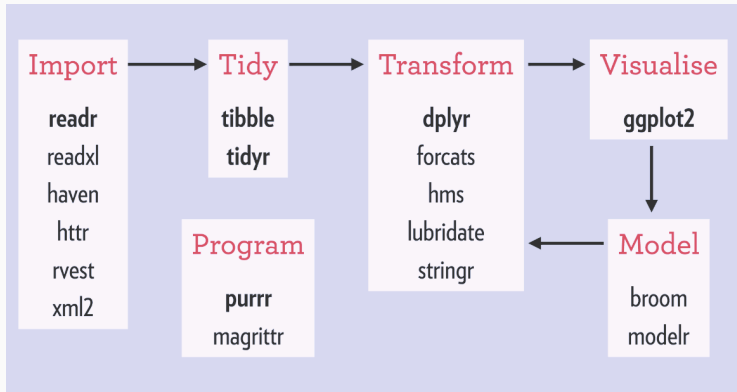
- Today:
  - Overview of Data Science workflow
  - *tidyverse* fundamentals
  - Data transformation with *dplyr*
  - Merging datasets with *join* operations
  - Data input strategies with *readr*
- Tomorrow:
  - String manipulation with *stringr*
  - Functional programming with *purrr*
  - Handling categorical 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)

# The Data Science Workflow



Credit *R for Data Science*

# The Tidyverse Ecosystem



credit [Joseph Rickert](#)

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 on JSS](#));
- **Code readability:** consistent, expressive 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”.

## 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.

RStudio shortcut: `ctrl` + `↑` + `M` or `⌘` + `↑` + `M` on macOS.

## Pseudocode: baking pipeline

<code>ingredients %&gt;%</code>	<code>{flour, water, eggs}</code>
<code>blend() %&gt;%</code>	<code>batter</code>
<code>cook() %&gt;%</code>	<code>whole cake</code>
<code>slice()</code>	<code>slice of cake</code>

The pipe allows function composition: `x %>% f() %>% g() ≡ g(f(x))`

If a function `h(x,y)` has more than one argument (or data is not the first argument), `.` is an explicit placeholder: `y %>% h(x, .) ≡ h(x, y)`.



# 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.

# Data structure

Vectors are the fundamental object class in R.

There are two types of vectors:

- **Atomic vectors**

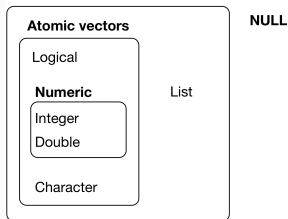
(homogeneous) of six types:

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.

- **Lists** (heterogeneous) which are sometimes called *recursive vectors* because lists can contain other lists.

## Vectors



# Functional programming with *purrr*

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

## (loose) mathematical definition:

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