# Lecture 1 - From Zero

Pietro Franceschi

FEM - UBC

11/02/2020

# Summary

- What is R
- What can I do with R
- Working with R in Rstudio
- R basic characteristics
- Visualizing data with ggplot

# What is R

*R is a **free software environment** for statistical computing and graphics, available at https://cran.r-project.org/*

- **open source** $\rightarrow$ huge community
- **statistical computing** $\rightarrow$ data mining/analysis
- **graphics** $\rightarrow$ data mining/visualization
- **programming** $\rightarrow$ language (command-line interpreter)

# Why R

## Pros

1. It's free!
2. Almost everything is ready
3. It easy to find "how to"s on the web

## Cons

1. Bugs errors and inconsistencies
2. It is not a true programming language
3. not always user friendly

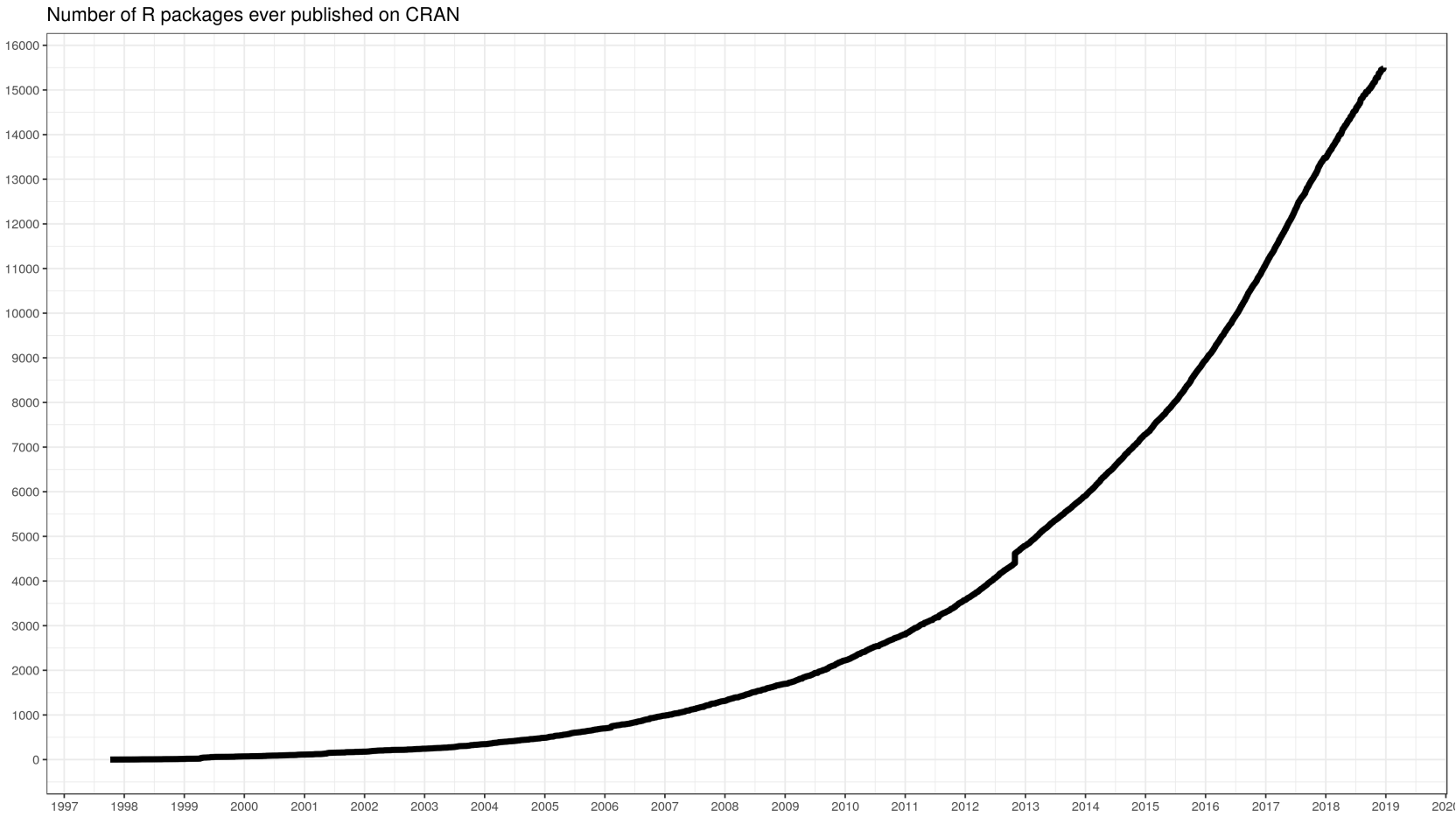
# R Packages

- New functionalities are added to R by using **packages**.
- A package is a set of **documented** functions to solve specific tasks

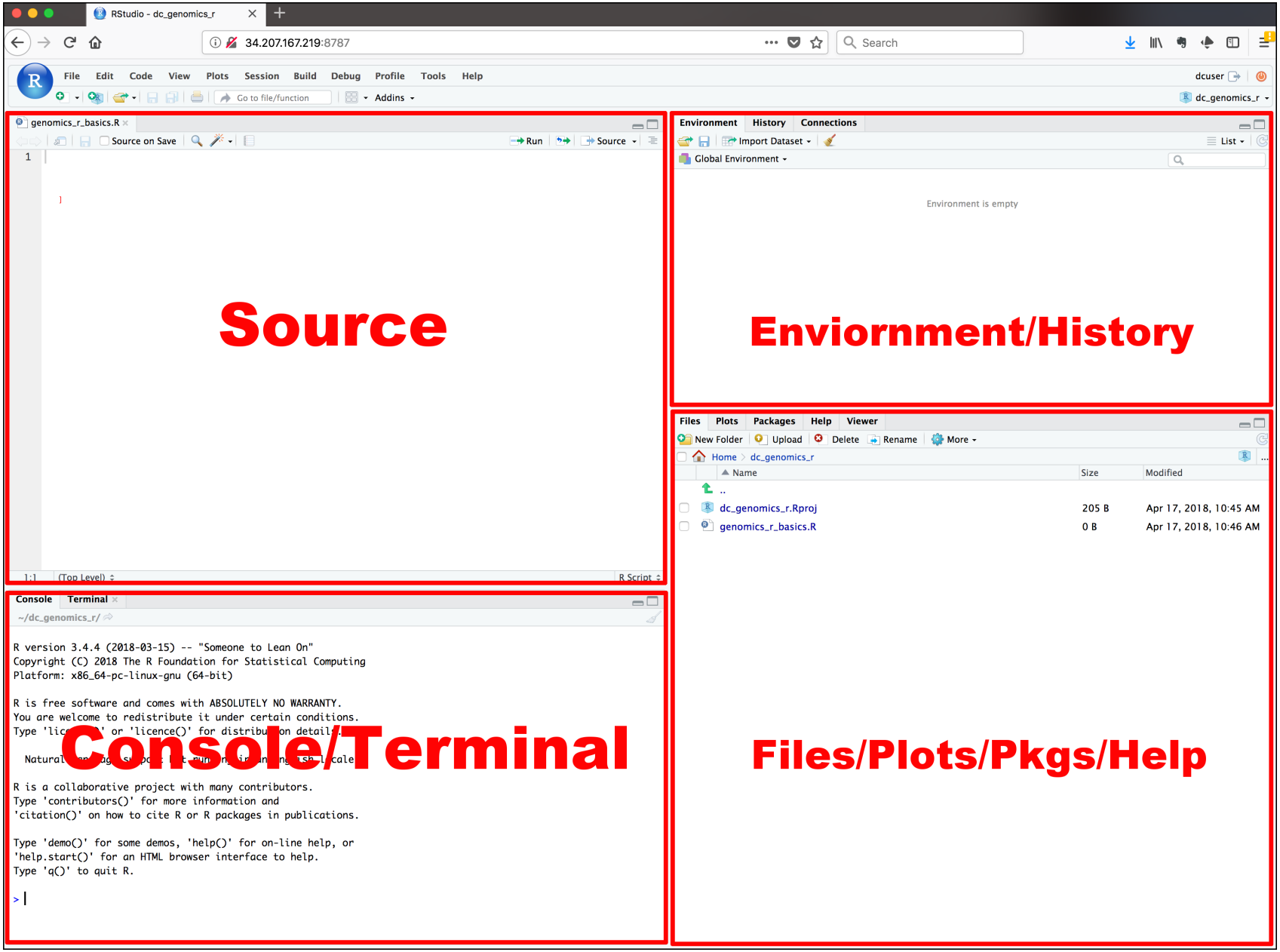
## Package sources

- CRAN - https://cran.r-project.org/
- Bioconductor - https://www.bioconductor.org/
- GitHub - https://github.com/

# Packages on CRAN



# RStudio


RStudio - dc_genomics_r
34.207.167.219:8787
Search
File Edit Code View Plots Session Build Debug Profile Tools Help
dcuser
Go to file/function
Addins
dc_genomics_r
genomics_r_basics.R
Source on Save
Run
Source
Source
Environment History Connections
Import Dataset
List
Global Environment
Environment is empty
Enviornment/History
Files Plots Packages Help Viewer
New Folder Upload Delete Rename More
Home dc_genomics_r
Name Size Modified
.. 
dc_genomics_r.Rproj 205 B Apr 17, 2018, 10:45 AM
genomics_r_basics.R 0 B Apr 17, 2018, 10:46 AM
Files/Plots/Pkgs/Help
Console Terminal
~/dc_genomics_r/
R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
Console/Terminal


LIVE
ON AIR

# How do we work

## Organize your work

- Keep track of what you do in the text editor
- Execute the commands in the console
- **Save** your code for next time ;-)

## Store your scripts

- **.R** files with this extension can be executed by R. The comments are marked by #
- **.Rmd** these files are typical of RStudio and consist of a mixture of text and code *chunks*


LIVE
ON AIR

# Key idea #1: Working directory

Is your reference directory

- reading files (data or code)
- writing files
- saving objects

```
## this function gets the working directory
getwd()
```

# Key idea #2: Workspace

Is the place (in memory) where R stores all the stuff you can use for the analysis.

- functions
- variables
- saving objects

R will "see" only things that are in the workspace!

# Populating the workspace

To put something in the workspace one uses either the "->" or "="

```
## this line of code creates a place called "a"
## in the workspace, filling it with the number 3

a <- 3
```

The content will show up in the "Environment" tab of Rstudio . . .

# Basic data Types

R can understand several basic **data types**

- Numeric: number with the comma
- Integer: number without comma
- (Complex number)
- Logical: TRUE/FALSE
- Character: a sequence of characters, everything between " "

# Assignment

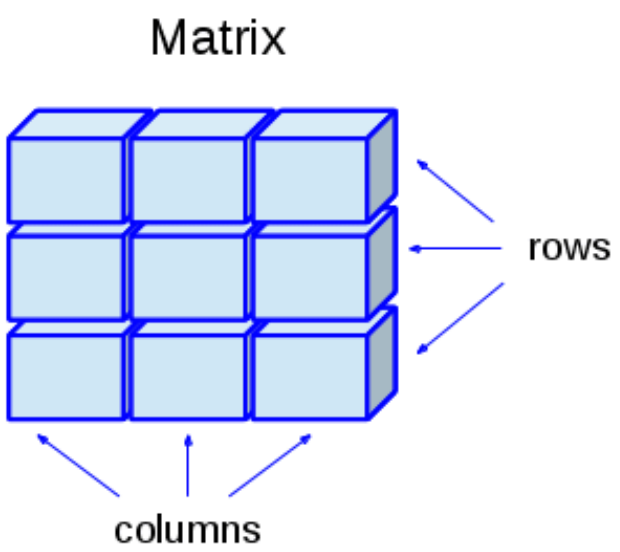
```
## create a set of basic objects
## operate on them with basic operations (+,-,*,/)
## to create character vectors use " ".

a <- 1
b <- 2

c <- a+b
```

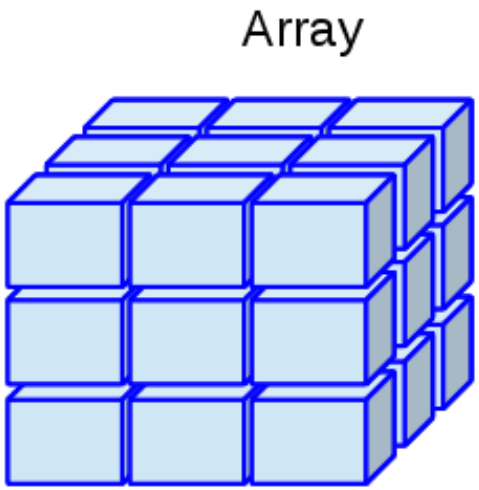
## Question time

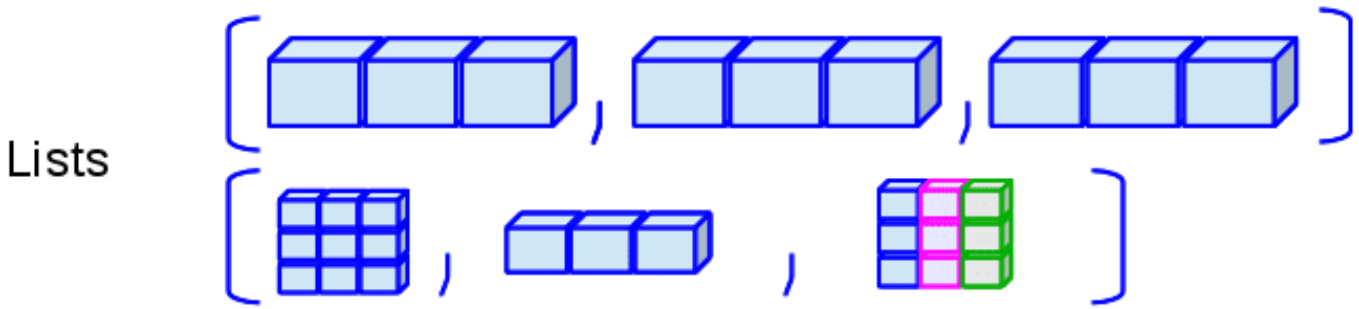
- can you combine object of different type?
- what does it happen if you sum two logical variables?
- is 1 different from "1"?

# Multidimensional Objects

Multidimensional objects are constructed by collecting together multiple basic data types



# Functions

Functions can be seen as "digestors", which handle some input producing output.

```
## a name followed by parenthesis is a function ...

pippo()
```

Inside the parenthesis you have **arguments** which determine the behavior of a function

```
## what does this do?
d <- sqrt(9)

## and this?
p <- seq(from = 1, to = 5, by = 2)
```

# Getting Help

The possible arguments of a function can be checked in the extensive and complete R help

```
## shows the help for the seq function
?seq
```

# Functions and multidimensional objects

**Multidimensional objects** are created by specific **functions**, with really evocative names ;-)

```
## create
myvector <- c(1,2,3,4)
mymatrix <- matrix(seq(1,9), ncol = 3)
mydataframe <- data.frame("col1" = 1:3,
                          "col2" = c("one","two","three"))
mylist <- list("hey" = seq(1:30), "today" = "monday")

## show
mymatrix
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
mydataframe
```

```
##   col1  col2
## 1    1   one
## 2    2   two
## 3    3 three
```

# Factors

Factors are character vectors with a limited number of values. For parsimony they are saved as numbers . . .

```
myfactor <- factor(rep(c("treated","ctrl"), each = 3))
myfactor
```

```
## [1] treated treated treated ctrl    ctrl    ctrl
## Levels: ctrl treated
```

***Important*** *if not specified the levels of the factor are ordered in alphabetical order. The importance of this aspect will be clear when we will start dealing with plots*

# Accessing Multidimensional Objects

- by **position**, giving the "coordinates"
- by **name** (more robust)

# Arrays, matrices and data frames

```
# like coordinates ...
mydataframe[1,]  ## first row
mydataframe[,1]  ## first column
mydataframe[,"col1"] ## the column  "col1"

# slicing
mydataframe[1:4,1]  ## the first four rows of the first column


# for data frames, the '$' symbol
# can be used to access the columns in a faster way
mydataframe$col1
```

# Lists

Lists are substantially unidimensional so we need only one index or one name

```
## here we need double brackets to get the element

mylist[[1]]  # the first element
mylist[1:3]  # from the first to the third element

mylist[["pippo"]] # the element named pippo

mylist$pippo  # also here the elemnt named pippo
```

# Assignment #2

```
# 1. create a dataframe with two columns,
# one with the short name of each month,
# the second with the number of days, the third with the (approx
# 2. extract the season and transform it into a factor
# 3. do the same with a list

data.frame()
factor()
list()
```

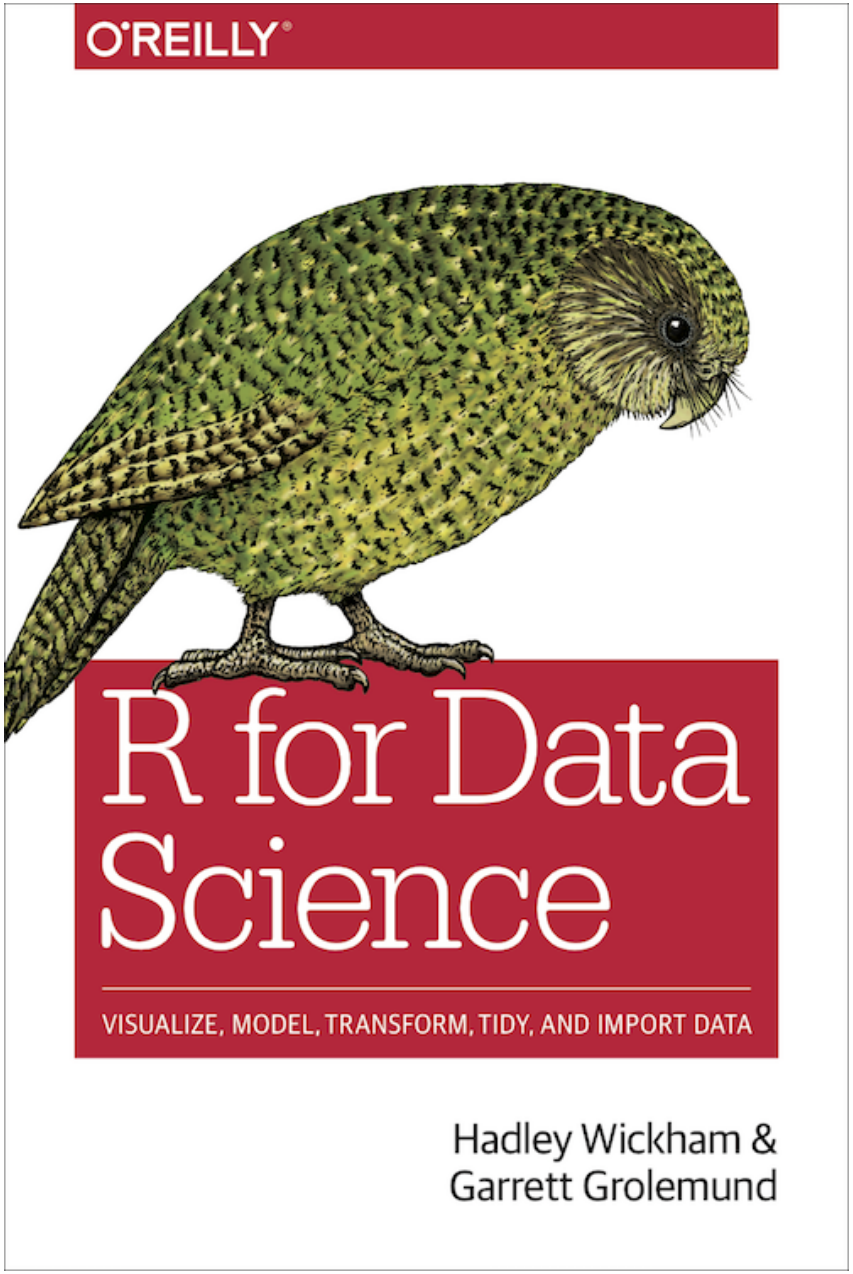
# Packages

The functionlities of R are expanded by "packages". Packages contain functions and, optionally data. To make a package available you have to

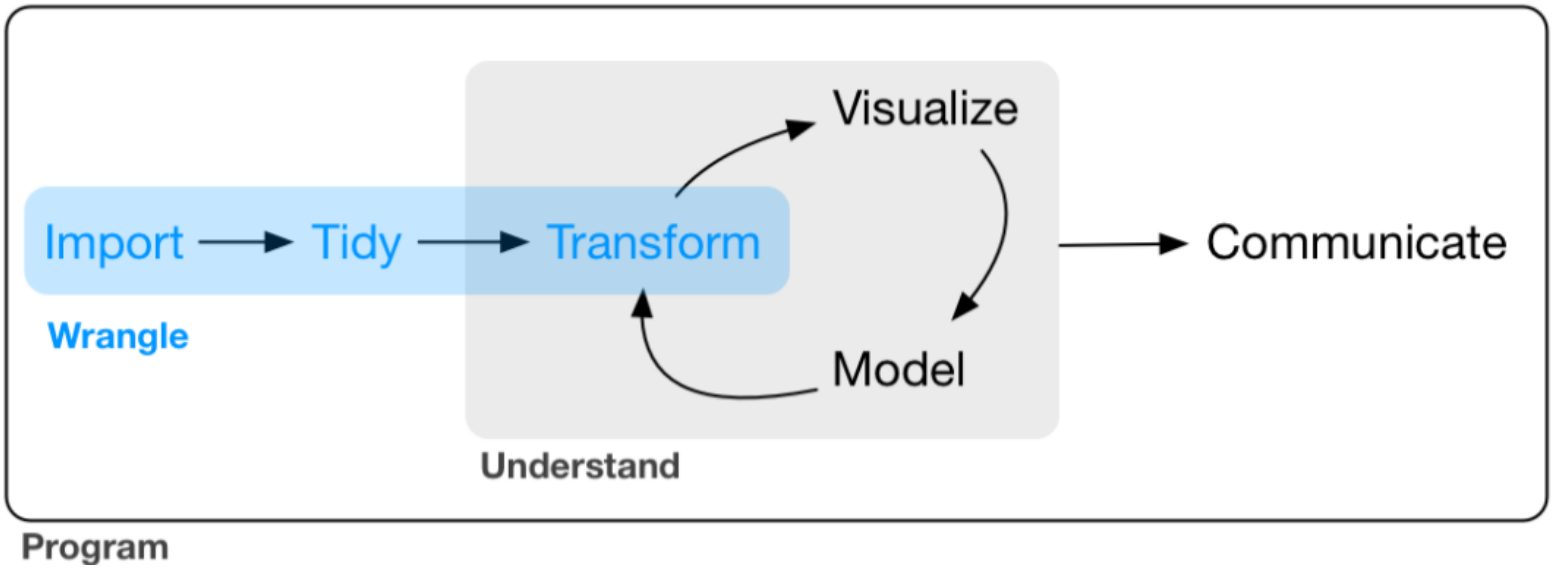
- download and compile it on your machine
- load it in the workspace

```
# install the package from CRAN on my machine
install.packages("tidyverse")

# makes the tools present in the package available on my workspace
library(tidyverse)

# load the mpg dataset in my workspace, otherwise I will not be able to use it!
data(mpg)
```

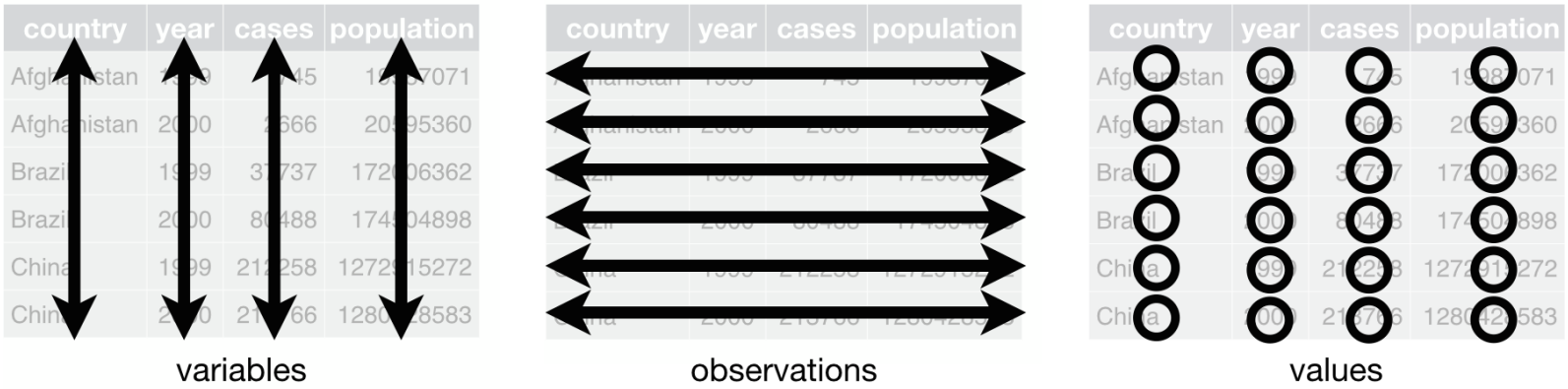
# My first Data visualization task



# Mind Map of data wrangling



# Tabular and tidy data




we can think of them as enormous and complicated excel tables

# mpg dataset

This dataset contains a subset of the fuel economy data that the EPA makes available on http://fueleconomy.gov. It contains only models which had a new release every year between 1999 and 2008 - this was used as a proxy for the popularity of the car.

```
## load the library
library(tidyverse)
## load the data
data(mpg)
## show their header, first 5 lines
head(mpg,5)
```

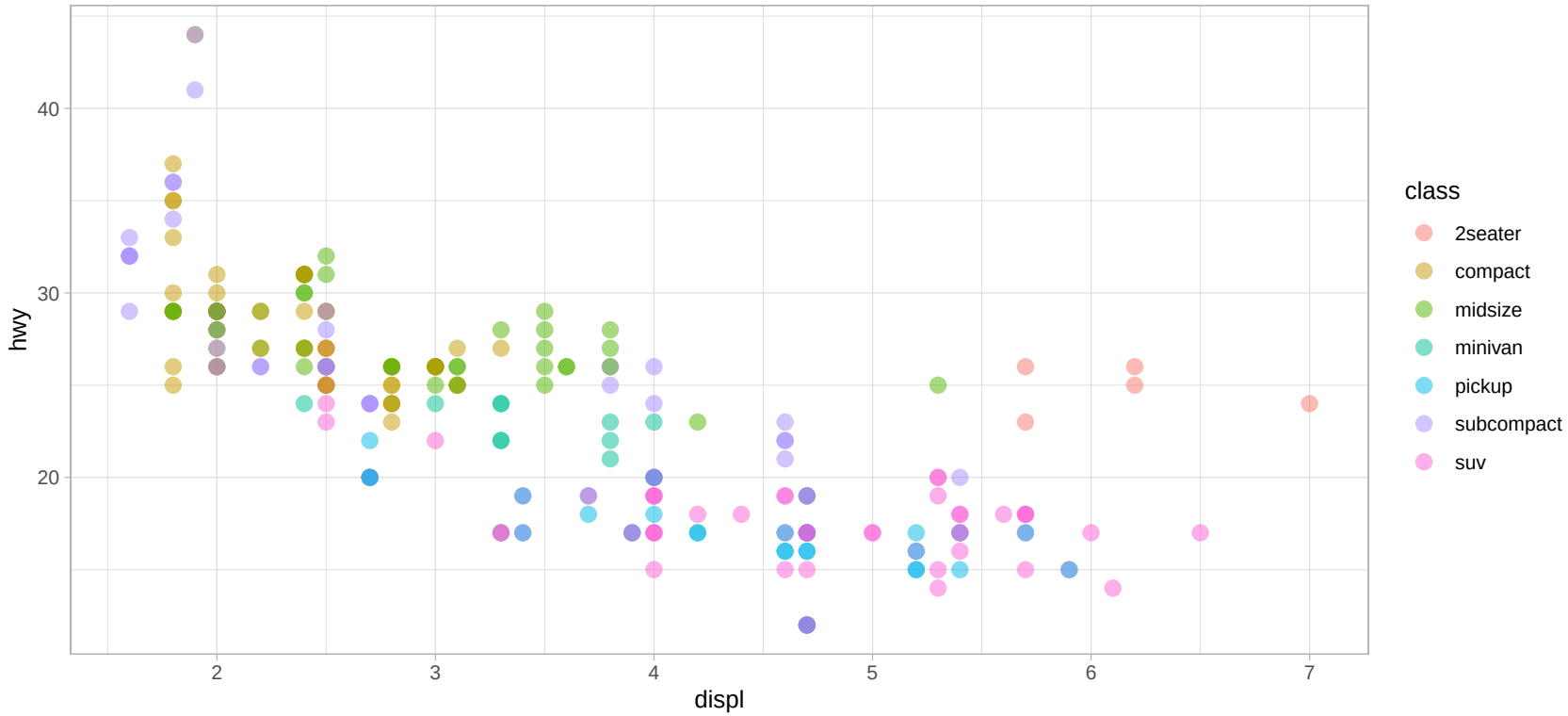
```
## # A tibble: 5 x 11
##   manufacturer model displ  year   cyl trans      drv     cty   hwy fl    class
##   <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
## 1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p     compa~
## 2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p     compa~
## 3 audi         a4      2    2008     4 manual(m6) f        20    31 p     compa~
## 4 audi         a4      2    2008     4 auto(av)   f        21    30 p     compa~
## 5 audi         a4      2.8  1999     6 auto(l5)   f        16    26 p     compa~
```

# Fast summary of my data

```
## this function gives a convenient summary of the data
summary(mpg)
```

```
##  manufacturer          model               displ           year
##  Length:234         Length:234         Min.   :1.600   Min.   :1999
##  Class :character   Class :character   1st Qu.:2.400   1st Qu.:1999
##  Mode  :character   Mode  :character   Median :3.300   Median :2004
##                                        Mean   :3.472   Mean   :2004
##                                        3rd Qu.:4.600   3rd Qu.:2008
##                                        Max.   :7.000   Max.   :2008
##       cyl           trans               drv                 cty
##  Min.   :4.000   Length:234         Length:234         Min.   : 9.00
##  1st Qu.:4.000   Class :character   Class :character   1st Qu.:14.00
##  Median :6.000   Mode  :character   Mode  :character   Median :17.00
##  Mean   :5.889                                         Mean   :16.86
##  3rd Qu.:8.000                                         3rd Qu.:19.00
##  Max.   :8.000                                         Max.   :35.00
##       hwy             fl                 class
##  Min.   :12.00   Length:234         Length:234
##  1st Qu.:18.00   Class :character   Class :character
##  Median :24.00   Mode  :character   Mode  :character
##  Mean   :23.44
##  3rd Qu.:27.00
##  Max.   :44.00
```

# A plot!


hwy
40
30
20
2
3
4
5
6
7
displ
class
2seater
compact
midsize
minivan
pickup
subcompact
suv

# How we did it ..

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, col = class),
             size = 3,
             alpha = 0.5) +
  theme_light() +
  theme(aspect.ratio = 0.5)
```

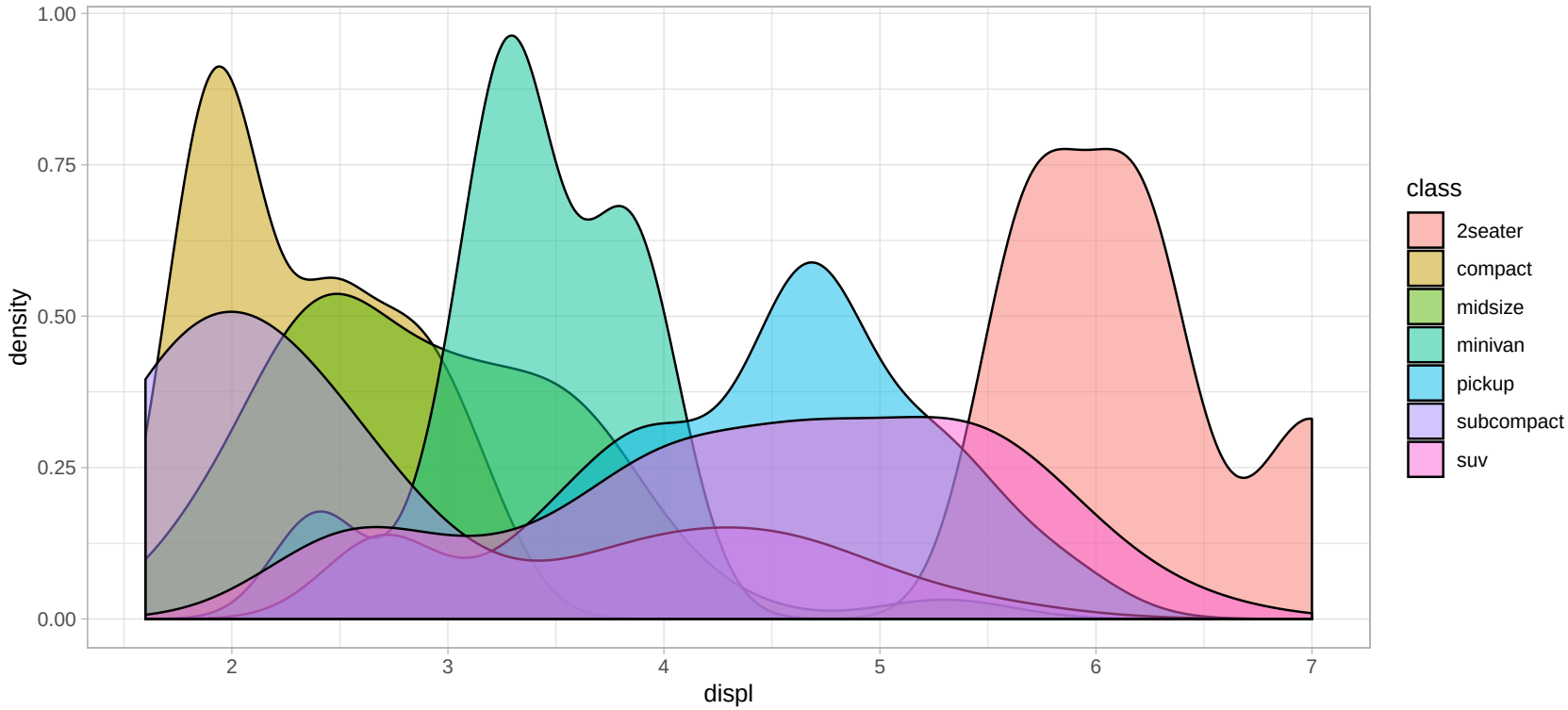
**Note** Here the functions are linked by +, this way of writing has been used to introduce a so called "grammar" of graphics

# Dissecting the command

```
## create an emty canvas ready for mpg
ggplot(data = mpg) +
  ## on the canvas plot points linking their eastetics to the column names
  geom_point(mapping = aes(x = displ, y = hwy, col = class),
             size = 3,
             alpha = 0.5) +
  ## add additional general aestetics
  theme_light() +  ## simple colors
  theme(aspect.ratio = 0.5) ## plot aspect ratio
```

- `ggplot()` - create the plot area
- geom_*something*() - add graphic elements to the plot
- `aes()` function to map graphical properties to columns in the data

# Now a density plot!



# On ggplotting

- manipulate "global" properties outside `aes()`
- link them with column inside `aes()`
- find the required and optional aesthetics in the help (e.g. `?geom_point`)

**Mind the apex!**

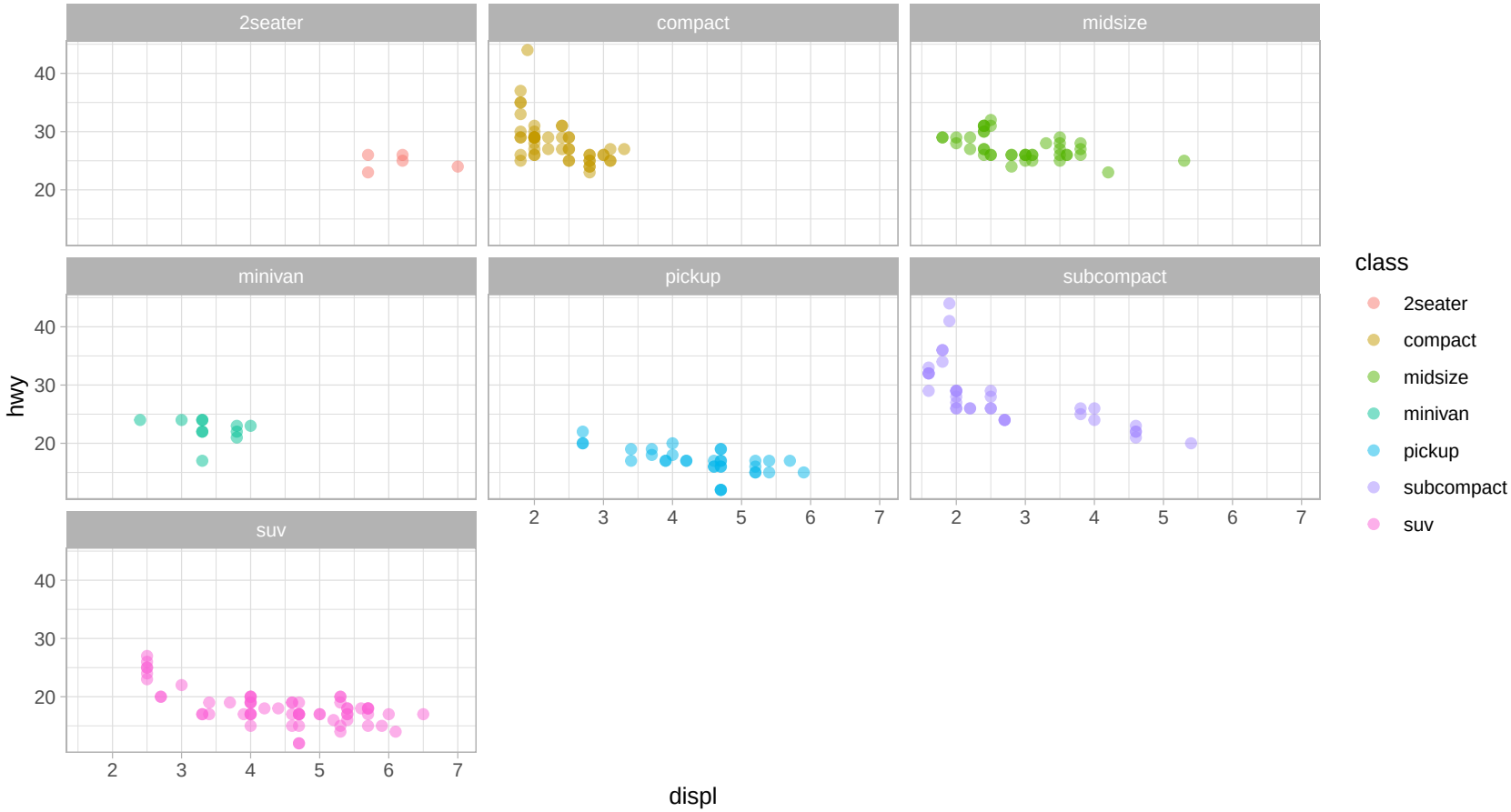
```
## when you specify the names of the columns
## pay attention to the apex!

 "ciao" # - string
 'ciao' # - string
 `ciao` # - the "name" ogf the column
```


LIVE
ON AIR

# Assignment #3

1. Play around with mpg changing the type of plots
2. Associate aesthetics to categorical or continuous properties
3. Make a boxplot of "displ" as a function of the class of the vehicle
4. Just play and ask!

# Splitting the plot in subplots



# Here the trick

```
## group specific subplots
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, col = class),
             alpha = 0.5, size = 2) +
  facet_wrap(~class) +  ## !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  theme_light() +
  theme(aspect.ratio = 0.5)
```

> *facet_wrap() and facet_grid() can be used to split the content of a plot according to one or more categorical variables*

Faceting can be also used with a clever trick to display multiple variables in the same multiplot . . .

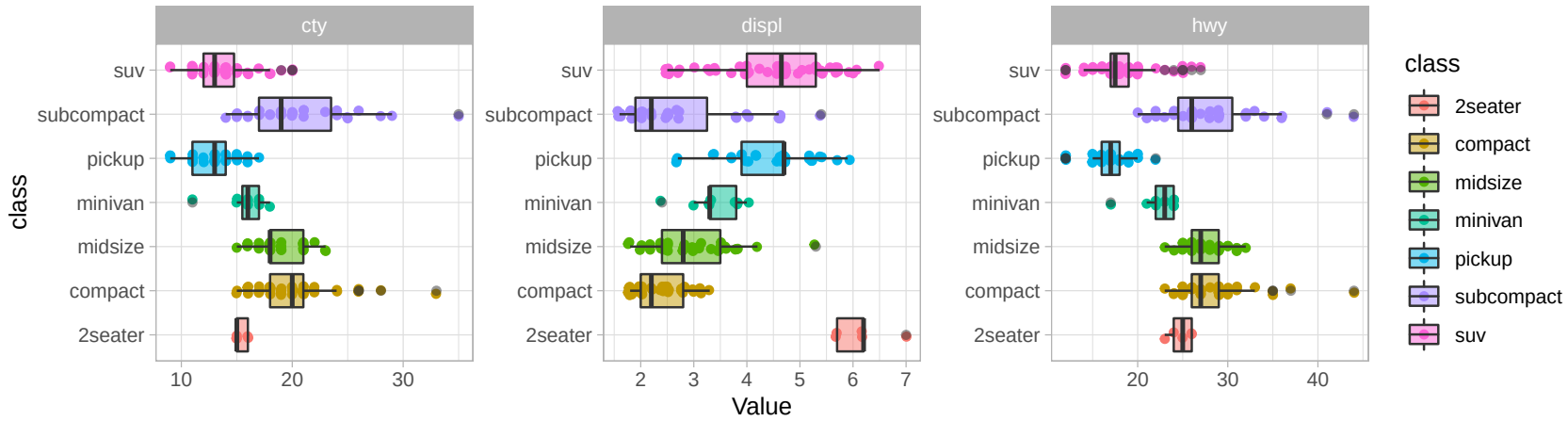
# Long and Wide data.frames

The same *tidy* data.frame can be organized in a **wide** and a **long** format. Topically we prefer to work with wide data, but long formats can be extremely handy ...

| country | year | cases |
|---|---|---|
| Afghanistan | 1999 | 745 |
| Afghanistan | 2000 | 2666 |
| Brazil | 1999 | 37737 |
| Brazil | 2000 | 80488 |
| China | 1999 | 212258 |
| China | 2000 | 213766 |

| country | 1999 | 2000 |
|---|---|---|
| Afghanistan | 745 | 2666 |
| Brazil | 37737 | 80488 |
| China | 212258 | 213766 |

table4


cty
displ
hwy
class
suv
subcompact
pickup
minivan
midsize
compact
2seater
10
20
30
2
3
4
5
6
7
Value
20
30
40
class
2seater
compact
midsize
minivan
pickup
subcompact
suv


LIVE
ON AIR

# Edgar Anderson's Iris Data

This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris.

The species are Iris **setosa, versicolor, and virginica**.

```
## you get it with
data(iris)
## visualize data
head(iris,5)
```

```
##   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
```

# Assignment #4

1. Look for correlation between Sepal.Length and Sepal.Width (Petal.Length and Petal.Width) for the three iris varieties
2. Make a faceted boxplot showing the four iris properties as a function of the species

# Read and Write data

## Reading

- in RStudio directly import data tables with the *Import Dataset* command present in the `Environment` tab
- from the command line several `read` function are available
- to avoid strange behavior keep your files as simple (tidy!) as possible ... no colors, no merged cells, ...

## Writing

- the object in the environment can be saved in compressed `.RData` format
- tables can be written with `write.csv`, `write.table` (or `write_csv` and `write_table` from the `readr` package)
- the function from `readr` are more efficient and handle better the row names

# Example

```
## save obj1 and obj2 in the mydata.RData file ...
save(obj1,obj2,file = "mydata.RData")
```

```
library(readr)
## here the data are in a csv called "wines.csv"
wines <- read_csv("data/wines.csv")

head(wines,5)
```

```
## # A tibble: 5 x 14
##   alcohol malic_acid   ash ash_alkalinity magnesium tot._phenols flavonoids
##     <dbl>      <dbl> <dbl>          <dbl>     <dbl>        <dbl>      <dbl>
## 1    13.2       1.78  2.14           11.2       100         2.65       2.76
## 2    13.2       2.36  2.67           18.6       101         2.8        3.24
## 3    14.4       1.95  2.5            16.8       113         3.85       3.49
## 4    13.2       2.59  2.87           21         118         2.8        2.69
## 5    14.2       1.76  2.45           15.2       112         3.27       3.39
## # ... with 7 more variables: non.flav._phenols <dbl>, proanth <dbl>,
## #   col._int. <dbl>, col._hue <dbl>, OD_ratio <dbl>, proline <dbl>, class <chr>
```

List of Chemical properties of a group of three types of wines (Barolo, Barbera and Grignolino)

# Assignment #5

1. Get the wines data from GitHub (wines.csv)
2. Import the data into R
3. make a text summary of your data (`summary()`)
4. Plot the relation between proanthocyanidins and total phenols for the three types of wines.
5. Can you do a boxplot of the different properties of the wines (also here remember wide and narrow data.frames)?

# Data Carpentry

*With the term "data carpentry" we identify all the set of operations/manipulations we currently do during the process of data exploration*

## Typical Operations

- Select some columns (variables)
- Select some rows
- Transform some of the columns (e.g. sum them . . . )
- Calculate some statistics on a group of samples

# The old way . . .

n the "standard" data analysis workflow, when several steps of transformation are needed the output of each transformation are saved and become the input of the subsequent step. This is time and memory inefficient . . .

```
## suppose you want to make a sequence from 1 to the square root of 10
a <- 10
b <- sqrt(a)
c <- seq(1,b,1)

## we create intermediate ancillary objects
```

# Piping

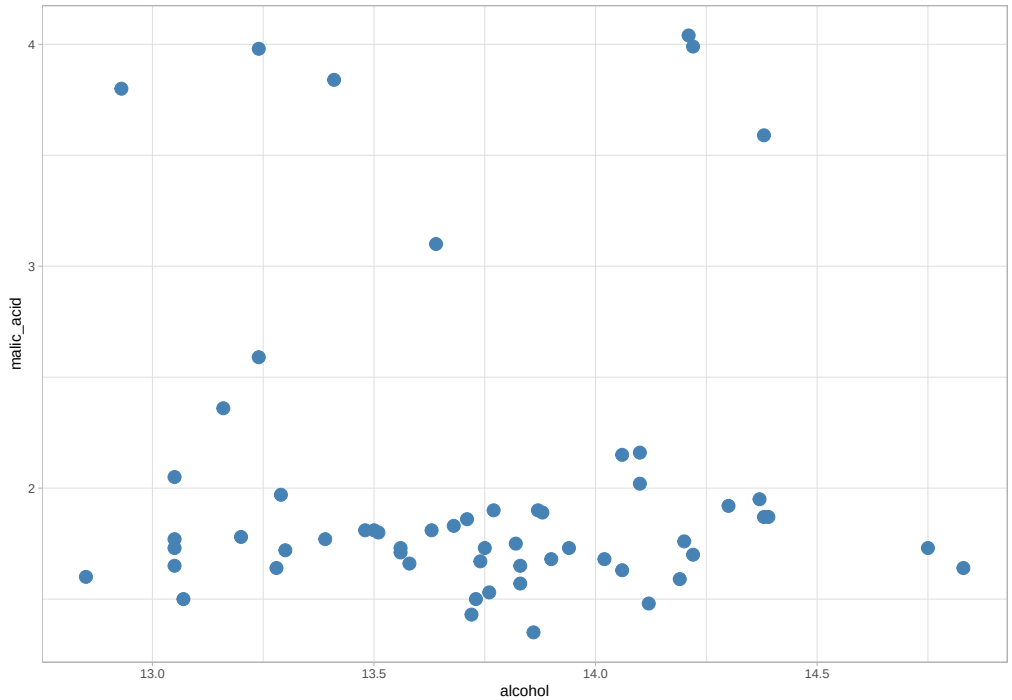
Better would be to "pipe" the output of a function to the input of another function . . . no intermediate saving, but also a code easier to read . . .

```
## pipe
%>%
```

This is the "pipe" operator which is added to R when you use tidyverse (actually magritteR . . . )

# Plumber at work . . .

```
wines %>%
  filter(class == "Barolo") %>%        ## get only the Barolos
  ggplot() +                           ## the data comes from the pipe!
  geom_point(mapping = aes(x = alcohol, y = malic_acid),
             col = "steelblue", size = 4) +
  theme_light()
```



# Pipes

- Very compact and clear writing
- I'm not creating permanent intermediate objects
- It follows my "psychological" logic ... I'm not changing the data, but only digesting them ...

# First carpenter tools

- select() Used to include, exclude columns. to exclude just put a (-) before the name.
- filter() Used to focus only a subset of the rows depending on a criterion
- mutate() Used to modify the content of a column or combine columns together


LIVE
ON AIR

# Assignment #6

1. Load the iris data
2. Calculate the ratios sepal.width/sepal.length and petal.width/petal.length (with **mutate()**!)
3. Plot the two ratios for all the iris variety
4. Save the modified table as .csv
5. Open it with Excel and see if you managed to save the new columns

**Important** Everything should be done with pipes! So only one command!