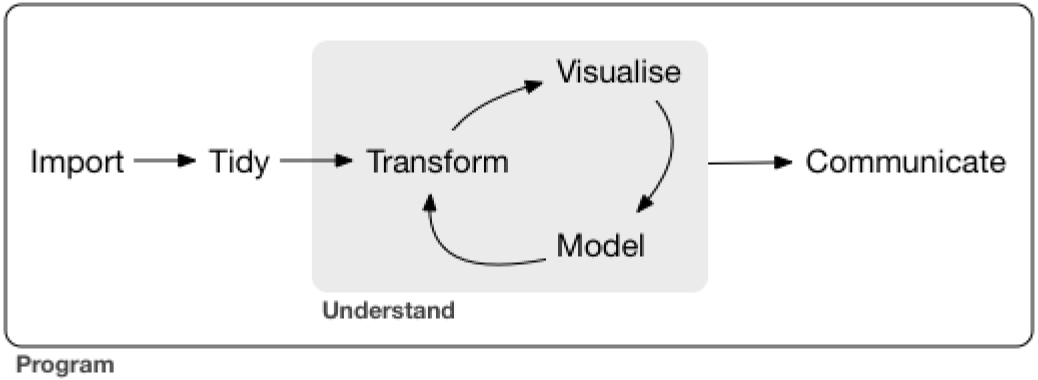
R -data grasping and wrangling

October 3, 2018

Descriptive statistics

Acquire a data culture

A typical Data Science project



Your opinion on...

Why and how to describe, visualise and summarise datasets ?

What is your strategy when you deal with a new dataset ?

How much time should you spend on these operations ?

What is (or should be) a dataset ?

At least 2 components :

What is (or should be) a dataset ?

At least 2 components :

- columns : variable, either discrete/continuous variable. They can be either predictors or outcomes

What is (or should be) a dataset ?

At least 2 components :

- columns : variable, either discrete/continuous variable. They can be either predictors or outcomes
- rows : observations, statistical units

What is (or should be) a dataset ?

At least 2 components :

- columns : variable, either discrete/continuous variable. They can be either predictors or outcomes
- rows : observations, statistical units

Definition of a statistical unit (INSEE's definition)

> "A statistical unit is a unit of observation or measurement for which data are collected or derived. The statistical unit is therefore the basic element for compiling and tabulating statistical data."

What is (or should be) a dataset ?

At least 2 components :

- columns : variable, either discrete/continuous variable. They can be either predictors or outcomes
- rows : observations, statistical units

Definition of a statistical unit (INSEE's definition)

> "A statistical unit is a unit of observation or measurement for which data are collected or derived. The statistical unit is therefore the basic element for compiling and tabulating statistical data."

A 3rd component might be :

- a reasoned approach when collecting the data (using experimental design, choosing exhaustiveness, real-time data...)

What is (or should be) a dataset ?

Let's imagine you'd like to test the efficiency of an e-mail marketing campaign. How could/should the analysed dataset look like ?

What is (or should be) a dataset ?

Let's imagine you'd like to test the efficiency of an e-mail marketing campaign. How could/should the analysed dataset look like ?

Homework : please read this blogpost

https://rtask.thinkr.fr/blog/the-ten-commandments-for-a-well-formatted-database/

Why describing datasets ?

Because of...

The Simpson's paradox

Let's dive into data and get our hands dirty !

These are the graduate school admission figures to university of California, Berkeley for the fall of 1973.

	Men		Women	
	Applicants	**Admitted**	**Applicants**	**Admitted**
Total	8442	**44%**	4321	35%

The Simpson's paradox

Let's dive into data and get our hands dirty !

These are the graduate school admission figures to university of California, Berkeley for the fall of 1973.

	Men		Women	
	Applicants	**Admitted**	**Applicants**	**Admitted**
Total	8442	**44%**	4321	35%

1. Is it a tidy dataset ? What is the statistical unit ?

The Simpson's paradox

Let's dive into data and get our hands dirty !

These are the graduate school admission figures to university of California, Berkeley for the fall of 1973.

	Men		Women	
	Applicants	Admitted	Applicants	Admitted
Total	8442	**44%**	4321	35%

1. Is it a tidy dataset ? What is the statistical unit ?

2. Any comments ?

The Simpson's paradox

Did you said gender bias ? What about this table :

The Simpson's paradox

Did you said gender bias ? What about this table :

Department	Men		Women	
	Applicants	Admitted	Applicants	Admitted
A	***825***	62%	108	**82%**
B	***560***	63%	25	**68%**
C	325	**37%**	***593***	34%
D	417	33%	375	**35%**
E	191	**28%**	***393***	24%
F	373	6%	341	**7%**

The Simpson's paradox

Did you said gender bias ? What about this table :

Department	Men		Women	
	Applicants	Admitted	Applicants	Admitted
A	***825***	62%	108	**82%**
B	***560***	63%	25	**68%**
C	325	**37%**	***593***	34%
D	417	33%	375	**35%**
E	191	**28%**	***393***	24%
F	373	6%	341	**7%**

1. What have we done here ?
2. Is it a tidy dataset ? What is the statistical unit ?
3. Any comments ?

The Simpson's paradox

In fact, when exposing new variables, we conclude to a counter-intuitive statement : **data show a bias in favor of women**

> Simpson's paradox, or the Yule–Simpson effect, is a phenomenon in probability and statistics, in which a trend appears in several different groups of data but disappears or reverses when these groups are combined. It is sometimes given the descriptive title reversal paradox or amalgamation paradox

Here is another example : https://dabblingwithdata.wordpress.com/2016/03/10/simpsons-paradox-and-the-importance-of-segmentation/

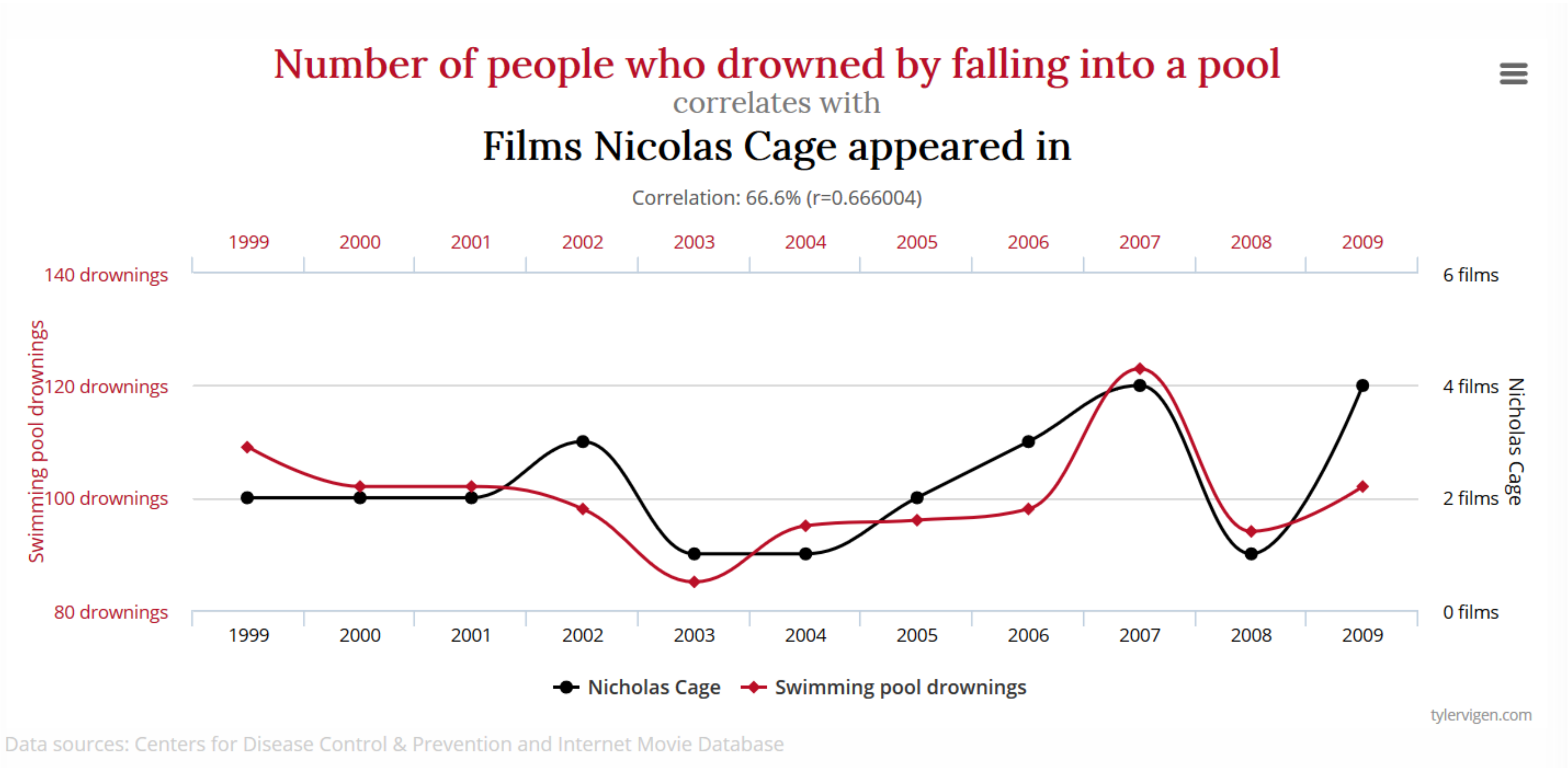
And a video : https://www.youtube.com/watch?v=ebEkn-BiW5k

Why describing datasets ?

Because of...

Correlation is not causality

Nicolas Cage is responsible for swimming pool drownings :



How to describe data in R ?

Inside your toolbox

Differentness

- range : `range()`
- variance : `var()`
- standard deviation : `sd()`

Sameness

- mean : `mean()`
- median : `median()`

Pay attention : all these functions eat vectors

How to describe data in R ?

Inside your toolbox :

```
library(tidyverse)
library(pander)
library(knitr)
library(skimr)
library(DT)
library(arsenal)
```

Try in a new Rmd file :

````
```{r results='asis'}
skim(iris) %>% pander()
```
````

How to describe data in R ?

Inside your toolbox :

```
library(tidyverse)
library(pander)
library(knitr)
library(skimr)
library(DT)
library(arsenal)
```

Try in a new Rmd file :

````
```{r results='asis'}
skim(iris) %>% pander()
```
````

````
```{r results='asis'}
skim(iris) %>% kable()
```
````

How to describe data in R ?

````
```{r}
skim_to_wide(iris) %>% datatable()
```
````

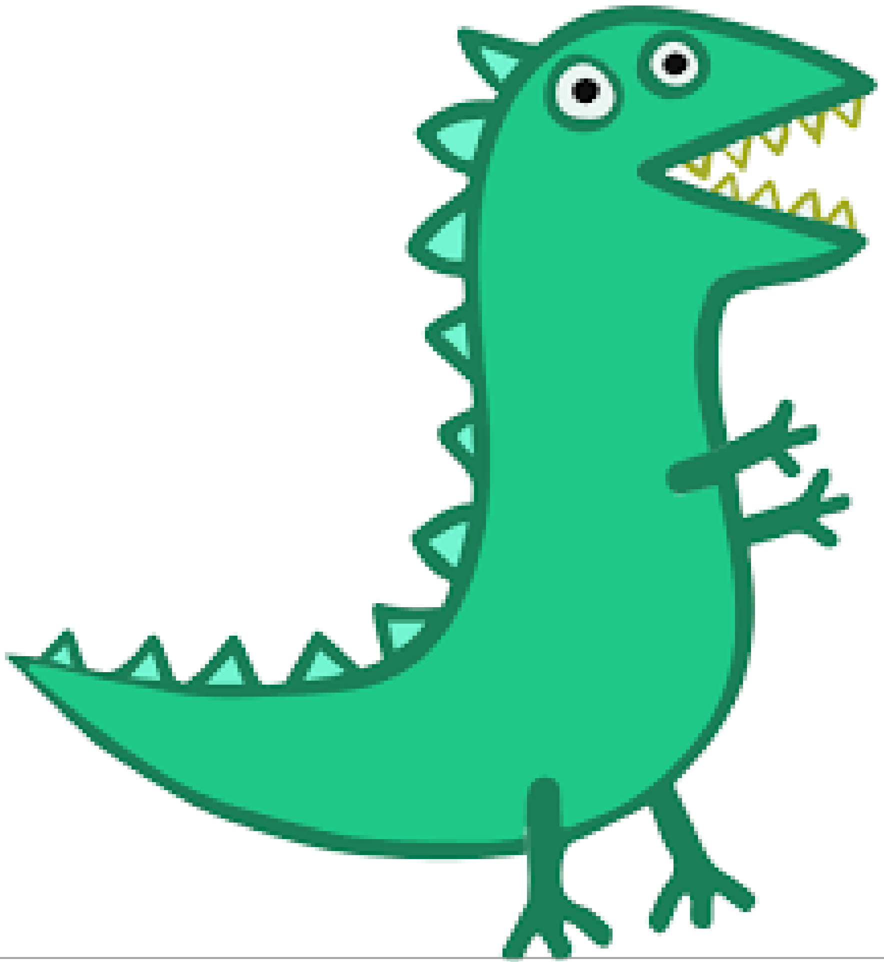
How to describe data in R ?

````
```{r}
skim_to_wide(iris) %>% datatable()
```
````

````
```{r results='asis'}
 cross_tab <- table(sample(c("nord","sud"),replace=TRUE,size=100),
 sample(c("yes","no"),replace=TRUE,size=100)
 )

 cross_tab %>%
  arsenal::freqlist() %>%
  summary()
```
````

Why visualising data ?

Because of dinosaurs :



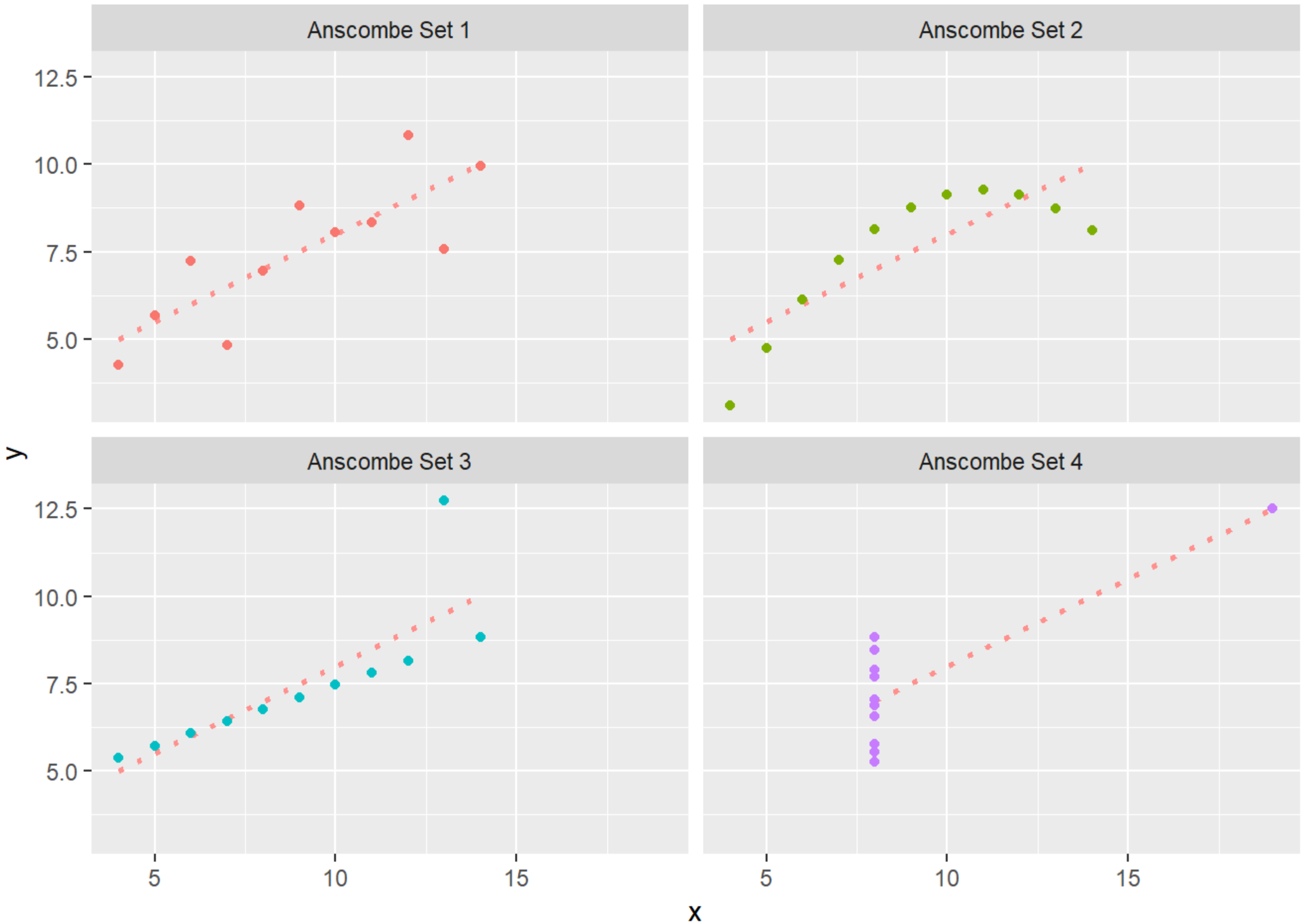
The Anscombe quartet

The Anscombe quartet are 4 x/y datasets. They were constructed in 1973 by the statistician Francis Anscombe to demonstrate both the importance of graphing data before analyzing it.

```
anscombe
```

```
#>    x1 x2 x3 x4    y1   y2    y3    y4
#> 1  10 10 10  8  8.04 9.14  7.46  6.58
#> 2   8  8  8  8  6.95 8.14  6.77  5.76
#> 3  13 13 13  8  7.58 8.74 12.74  7.71
#> 4   9  9  9  8  8.81 8.77  7.11  8.84
#> 5  11 11 11  8  8.33 9.26  7.81  8.47
#> 6  14 14 14  8  9.96 8.10  8.84  7.04
#> 7   6  6  6  8  7.24 6.13  6.08  5.25
#> 8   4  4  4 19  4.26 3.10  5.39 12.50
#> 9  12 12 12  8 10.84 9.13  8.15  5.56
#> 10  7  7  7  8  4.82 7.26  6.42  7.91
#> 11  5  5  5  8  5.68 4.74  5.73  6.89
```

The Anscombe quartet

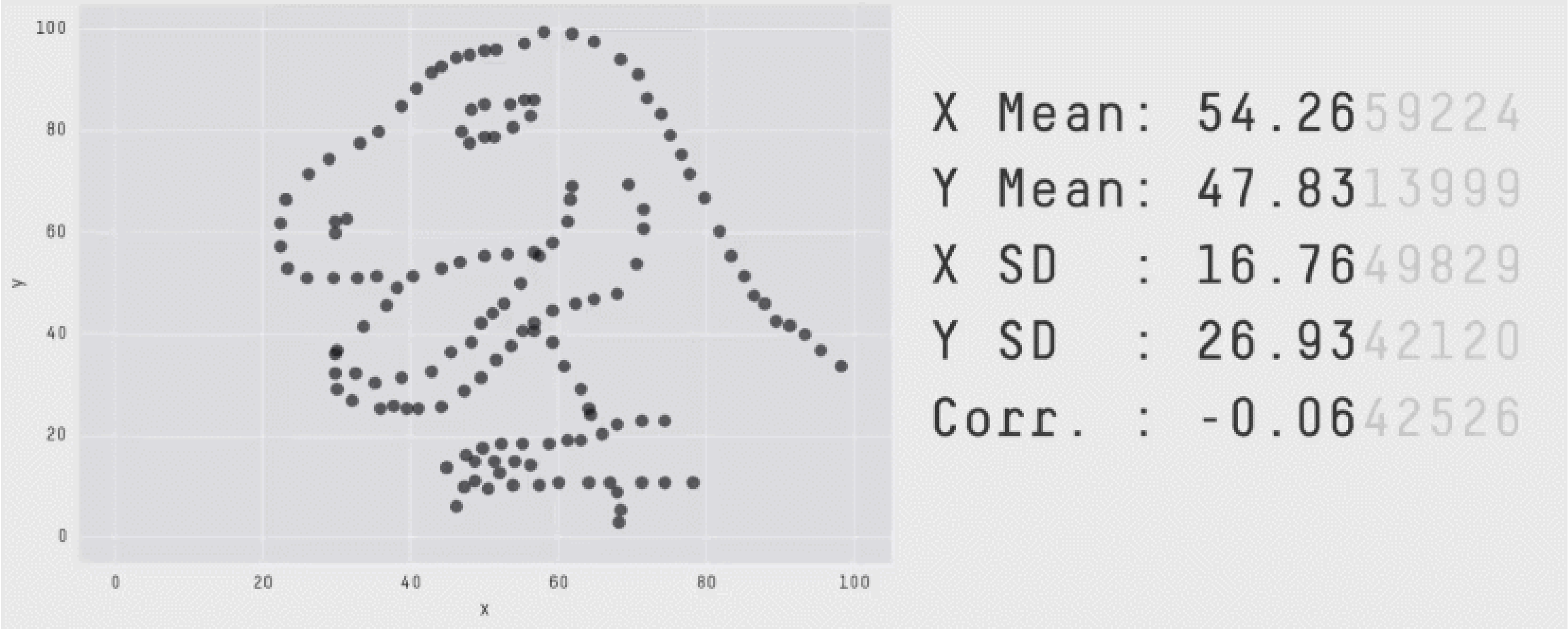


The Anscombe quartet

Set	mean_of_x	mean_of_y	variance_of_x	variance_of_y	correlation_x_y
1	9	7.500909	11	4.127269	0.8164205
2	9	7.500909	11	4.127629	0.8162365
3	9	7.500000	11	4.122620	0.8162867
4	9	7.500909	11	4.123249	0.8165214

Surprising, isn't it ?

But wait, what about dinosaurs ?



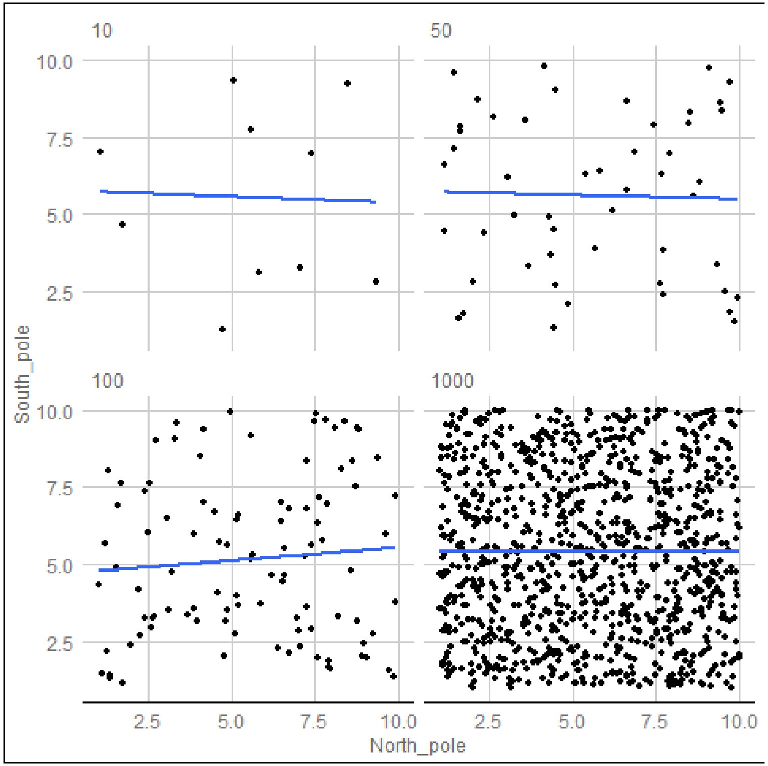
https://www.autodeskresearch.com/publications/samestats

> "...make both calculations and graphs. Both sorts of output should be studied; each will contribute to understanding" Anscombe, 1973

see dynamic version online

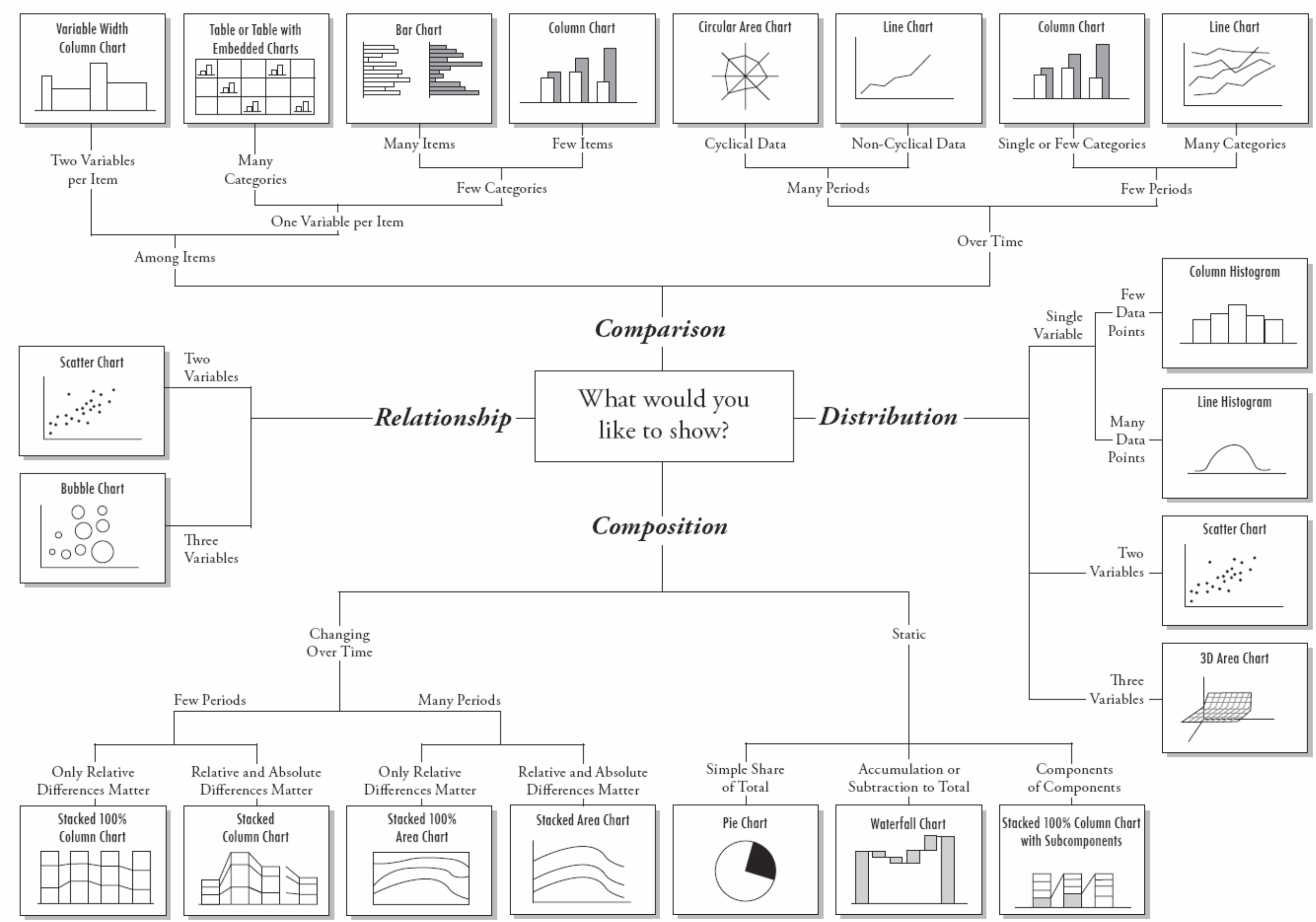
Other powers of datavisualising things

Correlation between random deviates from uniform distribution across four sample sizes. All values sampled from a uniform distribution. (source : https://crumplab.github.io/statistics/ - feel free to read the book)



see dynamic version online

What kind of plot ?



Another ressource : https://www.data-to-viz.com/

Do's and don't of data visualisation

Please do add…

- a title,
- labelled axes,
- a graduated scale that makes sense,
- a legend,
- the source of the data,
- and the units of measurement used

to your graphs.

Possibly, add

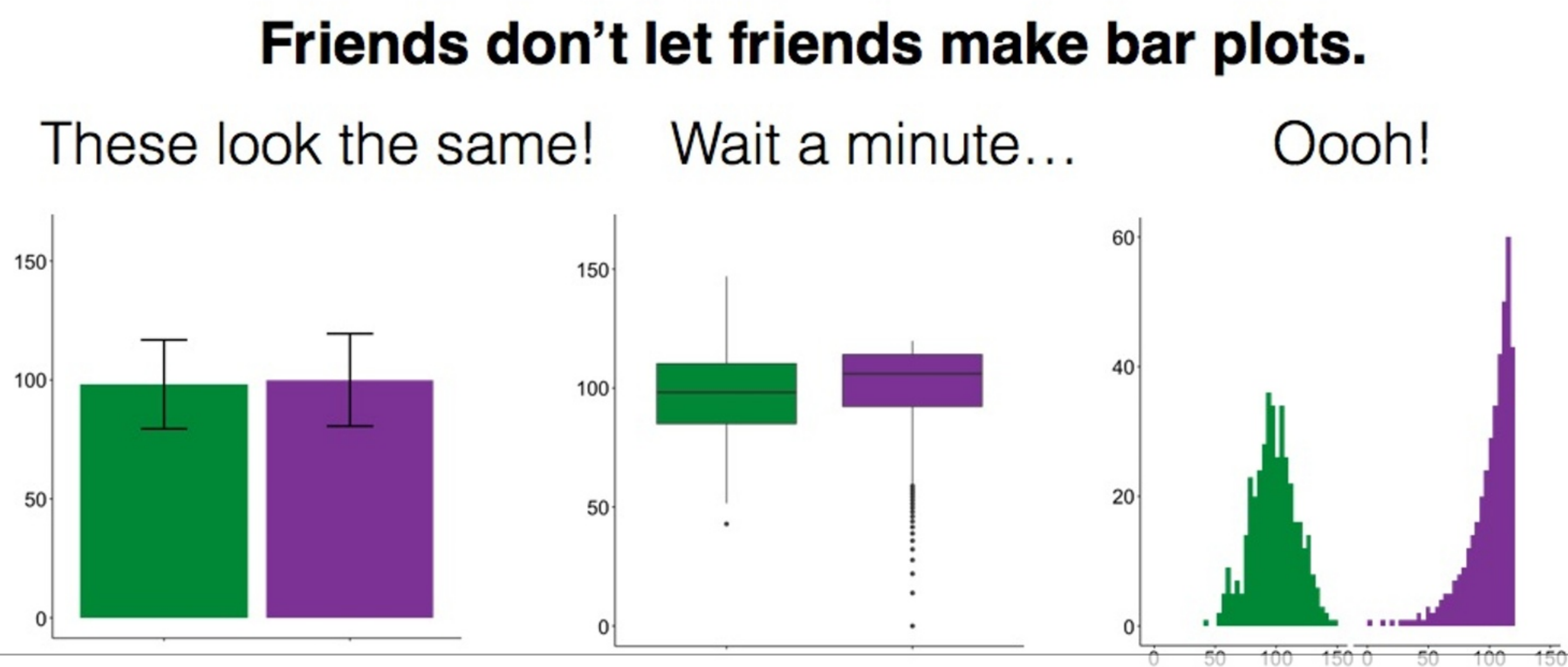
- a subtitle
- and annotations.

Be careful when choosing colors : https://hbr.org/2014/04/the-right-colors-make-data-easier-to-read

Do's and don't of data visualisation

Please, do not plot

- pie charts (also called the **bubble plague** - draw barplot instead)
- barbarplot (draw density or histogram instead)

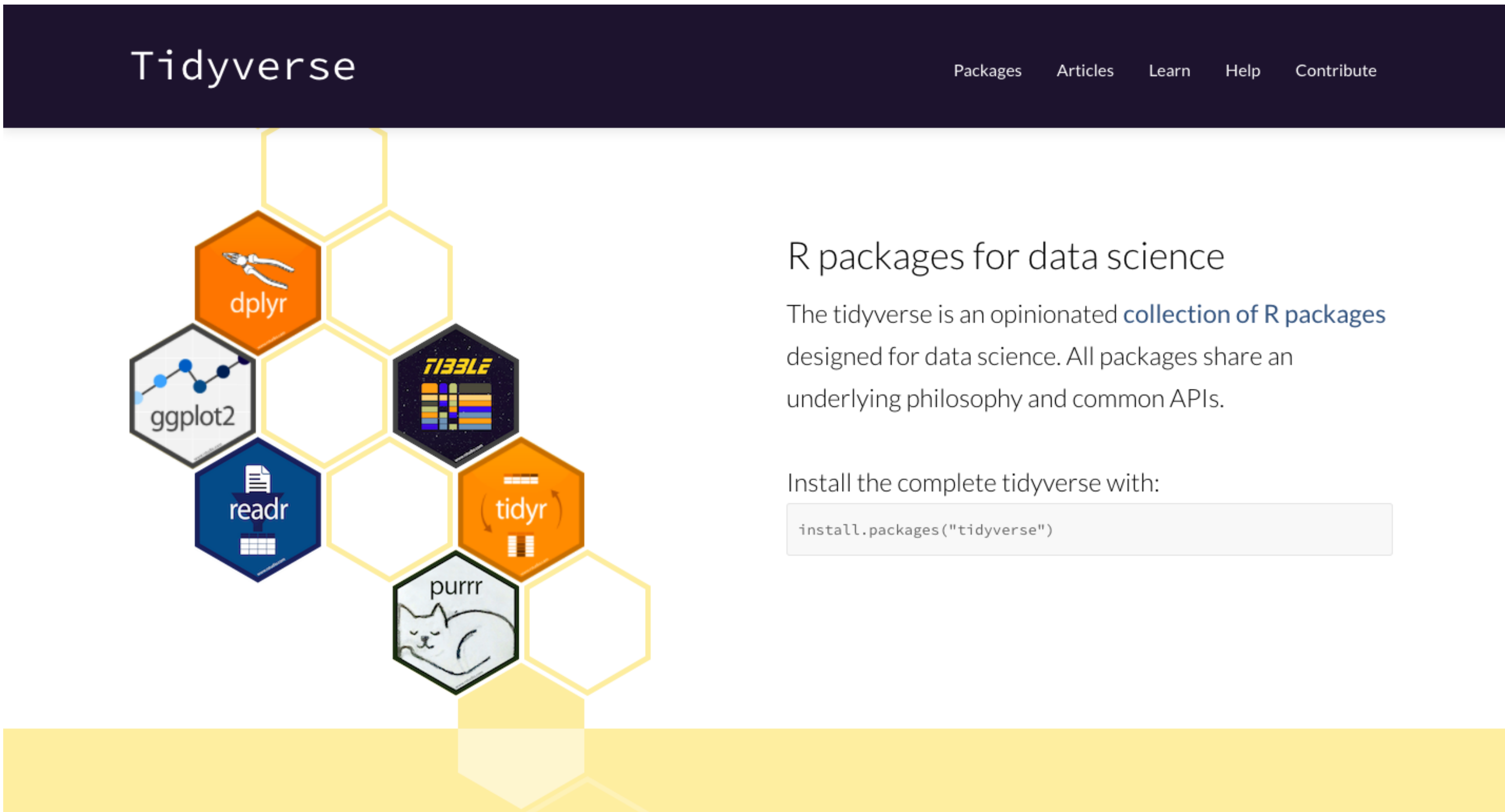


tidyverse & tidy data

Put your data in order

class: slide

What is the {tidyverse} ?



The {tidyverse} also contains ...

Import

- {readxl}
- {haven}
- {feather}
- {httr}
- {jsonlite}
- {rvest}
- {xml2}

Wrangle

- {stringr}
- {lubridate}
- {forcats}
- {hms}
- {blob}

Program

- {rlang}
- {magrittr}
- {glue}

Model

- {modelr}
- {broom}

```
install.packages("tidyverse")
```

The packages of {tidyverse}

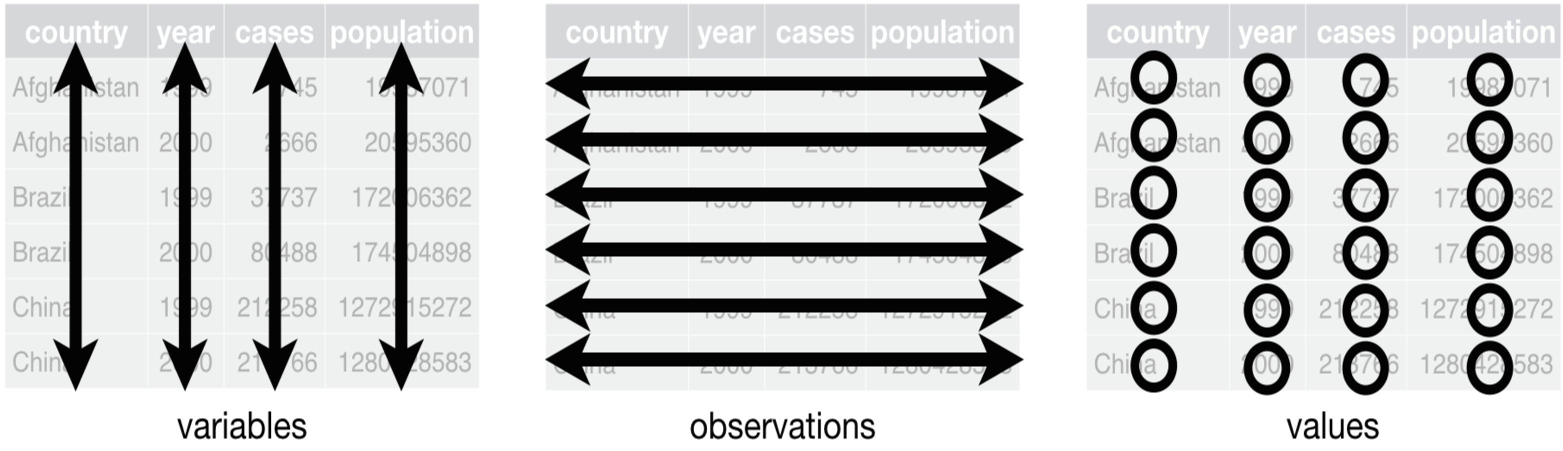
```
library(tidyverse)
```

```
#> -- Attaching packages --------------------------------------- tidyverse
1.2.1 --

#> v ggplot2 3.0.0     v purrr   0.2.5
#> v tibble  1.4.2     v dplyr   0.7.6
#> v tidyr   0.8.1     v stringr 1.3.1
#> v readr   1.1.1     v forcats 0.3.0

#> -- Conflicts ------------------------------------------
tidyverse_conflicts() --
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()    masks stats::lag()
```

{tidyverse}, Tidy == clean

> *All happy families are alike;*
> *each unhappy family is unhappy in its own way.*

Léon Tolstoï, _Anna Karénine_



And in data manipulation ?

Like families, tidy datasets are all alike but every messy dataset is messy in its own way

- In an ordered dataset, individuals/observations are in rows, and variables in columns

cf The Ten Commandments for a well-formatted database

```
age <- c(25, 45, 31, 10,23,43,45,12)
sex <- c( "man", "man", "woman", "man","man", "man", "woman", "man")
tbl <- tibble("age" = age, "sex" = sex)
tbl
```

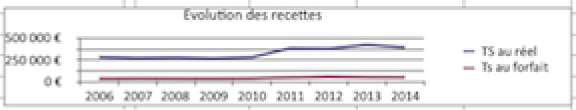
```
#> # A tibble: 8 x 2
#>     age sex
#>   <dbl> <chr>
#> 1    25 man
#> 2    45 man
#> 3    31 woman
#> 4    10 man
#> 5    23 man
#> 6    43 man
#> 7    45 woman
#> 8    12 man
```

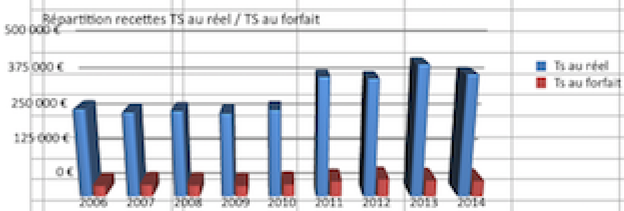
"non-tidy data"

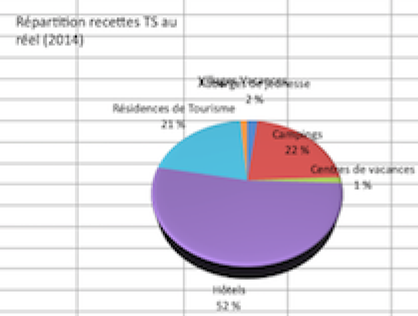
Feuil1 | Feuil2 | Feuil3

Annexe 1 - EVOLUTION DES RECETTE TAXES DE SEJOUR (données au 16/09/2015)

Les	2006	2007	2008	2009	2010	2011	2012	2013	2014
Auberges de jeunesse				6 447,90 €	7 177,50 €	9 002,40 €	6 610,80 €	7 609,20 €	8 198,00 €
Campings				75 080,71 €	80 234,50 €	97 733,10 €	78 362,26 €	87 744,60 €	88 135,96 €
Centres de vacances	34,20 €	87,90 €	6,90 €	41,70 €		4 769,00 €	4 923,50 €	5 123,40 €	5 237,90 €
Hôtels	159 656,90 €	150 164,77 €	152 246,70 €	147 121,10 €	152 191,10 €	212 908,61 €	225 131,23 €	224 025,69 €	208 148,30 €
Résidences de Tourisme	8 237,20 €	6 887,90 €	7 517,50 €	30 477,50 €	8 010,50 €	4 056,50 €	55 551,99 €	90 947,36 €	81 855,01 €
Villages Vacances	41 593,00 €	50 657,57 €	40 029,70 €	13 056,30 €	36 971,00 €	59 947,80 €	12 510,20 €	11 999,80 €	5 434,10 €
TOTAL TS AU REEL	286 375,40 €	275 854,52 €	280 483,93 €	272 225,21 €	284 584,60 €	388 417,41 €	**383 089,98 €**	**427 450,05 €**	**397 009,27 €**
TOTAL TS AU FORFAIT	35 386,40 €	38 228,59 €	36 562,68 €	35 547,93 €	39 898,91 €	50 122,36 €	**57 243,99 €**	**52 820,04 €**	**51 575,52 €**
TOTAL RECETTES	**321 761,80 €**	**314 083,11 €**	**317 046,61 €**	**307 773,14 €**	**324 483,51 €**	**438 539,77 €**	**440 333,97 €**	**480 270,09 €**	**448 584,79 €**







"non-tidy data"

A	B	C	D	E	F	G	H	I	J	K	L
Région	Consommation finale grande industrie (GWh)										
	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
Auvergne Rhône-Alpes	23 268	26 215	26 939	26 007	26 464	19 564	14 461	11 548	12 170	13 116	13 257
Bourgogne France-Comté	3 940	3 866	3 768	3 300	3 508	3 569	3 411	3 331	3 277	3 351	3 259
Bretagne	959	940	912	772	798	801	757	730	714	714	717
Centre-Val de Loire	1 485	1 486	1 398	1 394	1 384	1 447	1 453	1 402	1 386	1 341	1 261
Corse	-	-	-	-	-	-	-	-	-	-	-
Grand Est	12 224	12 052	11 611	9 050	9 800	9 834	9 057	8 643	8 580	8 323	8 278
Hauts de France	17 449	17 210	16 619	14 994	16 179	15 961	16 070	15 696	15 556	15 242	15 060
Ile-de-France	7 502	7 445	7 392	7 251	7 191	7 114	7 257	7 221	7 098	7 096	7 016
Normandie	6 355	6 376	6 293	6 062	5 944	5 832	5 545	5 596	5 560	5 632	5 441
Nouvelle Aquitaine	5 437	5 530	5 434	4 930	4 983	4 994	4 765	4 625	4 479	4 362	4 402
Occitanie	3 193	3 086	2 778	2 370	2 515	2 547	2 504	2 463	2 532	2 481	2 584
Pays de la Loire	2 226	2 213	2 210	2 036	2 079	2 103	2 118	2 086	2 156	2 185	2 180
Provence-Alpes-Côte d'Azur	8 996	8 974	8 988	8 225	8 409	8 613	8 062	8 151	8 214	8 051	8 392
France	93 034	95 393	94 342	86 391	89 254	82 381	75 460	71 492	71 722	71 894	71 846

A	M	N	O	P	Q	R	S
Région	Consommation finale PMI/PME (GWh)						
	2006	2007	2008	2009	2010	2011	2012
Auvergne Rhône-Alpes	22 637	22 698	22 942	22 246	23 220	22 781	23 062
Bourgogne France-Comté	7 858	7 887	7 739	7 483	7 762	7 645	7 721
Bretagne	8 343	8 564	8 816	8 694	9 039	8 904	9 039
Centre-Val de Loire	7 334	7 236	7 301	6 981	7 212	7 173	7 099
Corse	447	456	480	499	504	518	526
Grand Est	18 557	18 573	18 536	17 475	17 993	17 218	17 001
Hauts de France	16 594	16 474	16 687	15 739	16 306	16 166	16 265
Ile-de-France	29 622	29 806	30 685	31 191	32 063	30 507	31 191
Normandie	9 445	9 553	9 531	9 298	9 599	9 258	9 339
Nouvelle Aquitaine	13 077	13 227	13 644	13 127	13 631	13 395	13 851
Occitanie	10 823	10 949	11 225	10 719	11 052	11 517	11 767
Pays de la Loire	9 684	9 810	10 106	9 893	10 380	10 107	10 265
Provence-Alpes-Côte d'Azur	10 638	10 632	10 944	11 032	11 238	10 777	11 021
France	165 059	165 865	168 636	164 377	169 999	165 966	168 147

Auvergne Rhône-Alpes

Consommation finale grande industrie hors Eurodiff et CERN (GWh)

2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
12 629	12 465	12 321	10 234	11 035	11 336	10 930	10 706	11 070	11 455	11 723

Données 2016 | Données 2015 | données 2014

Notion of statistical unit

- **Population** = all the objects studied
- **Sample** = subpopulation of these same objects
- **Variable** = characteristic measured on each object
- **Statistical unit** = observations on objects for all variables

Datasets for statistical analysis reflect the notion of the statistical unit and must be constructed accordingly :

Statistical unit in line and variables in column

So,

- If the information is available somewhere, it should be included in the table.
- All data are grouped in a single table.
- The data is not broken down in multiple Excel tabs.
- Several tables must not be grouped together in the same workbook.

Tidying a messy dataset

What is a messy data set?

- The column headings are values/modalities.
- Several variables are stored in a single column.
- The variables are stored in rows AND columns.
- An observation is stored in several data tables.
- … (non-exhaustive list)…

Often, it's a *'clever'* combination of all.

{tidyverse} : About the "tibble" class

`dataframes` that are returned by one of the tidyverse import functions (`read_excel`, `read_csv`) are applied an additional class, they become `tbl`.

```
library(tibble)
iris_tbl <- as_tibble(iris)
class(iris)# [1] "data.frame"
```

```
#> [1] "data.frame"
```

```
class(iris_tbl)  #[1] "tbl_df" "tbl" "data.frame "
```

```
#> [1] "tbl_df"     "tbl"        "data.frame"
```

The display of these dataset is improved, only the first rows are displayed, an optimal number of columns is presented and the type of each variable is indicated.

About the "tibble" class

With `tbl`, when selecting a single column, the output keeps being a `tbl` object (no longer a vector like with r-base). No need to use `drop = FALSE`.

```
class(iris[,1]) # "numeric"
```

```
#> [1] "numeric"
```

```
class(iris[,1,drop = FALSE]) # "data.frame"
```

```
#> [1] "data.frame"
```

```
class(iris_tbl[,1])# "tbl_df" "data.frame"
```

```
#> [1] "tbl_df"     "tbl"        "data.frame"
```

However, not all R functions can support them yet, it is thus possible to delete the `tbl` class thanks to:

```
as.data.frame(iris_tbl)
```

About the "tibble" class

The `tribble` function allows to create tibbles in the other way:

```
df <- tribble(
  ~ x, ~ y, ~ z,
  1, 2, 3,
  4, 5, 6
)
df
```

```
#> # A tibble: 2 x 3
#>         x      y      z
#>     <dbl> <dbl> <dbl>
#> 1       1      2      3
#> 2       4      5      6
```

Import/Export data

Read and write data for analysis

Import data from flat files (1)

- Flat files are «text-oriented» databases (.txt and .csv extensions)
- They can be read in a simple text-editor (*e.g.* notepad)
- Each line in the file is a future row in the dataset
- They are structured with a field separator and a decimal separator
- In tidyverse, the {readr} package contains the set of necessary functions for this kind of import (and more)
- To import a file with this csv type, you need to use `read_delim` or, for convenience, `read_csv` or `read_csv2`
- The syntax looks like:

```
library(readr)
dataset <- read_delim(file = "filename.txt", delim = ";", col_names =
TRUE)
```

Import data from flat files (2)

- `read_csv` will use `delim=","`
- `read_csv2` will use `delim=";"`
- `read_tsv` will use `delim="\t"` (tabulation)

By default the function is intelligent and tries to guess the types of the columns (number, date, factor…). Parameter `col_type` allows to define them manually if necessary (see `vignette("column-types")`).

Exercise

Import `titanic_train.csv` from `/data` directory using `read_csv()` function, assign it to `titanic_train`

Show a summary of the dataset

Find the number of lines and columns

Find variables names

Import Excel .xls or .xlsx files (1)

- Package {readxl} :

```
library(readxl)
excel_sheets("Data_FE_CPU_messages.xlsx")
read_excel("Data_FE_CPU_messages.xlsx", sheet = 3)
read_excel("Data_FE_CPU_messages.xlsx", sheet = "FE1")
```

Import files from other statistical softwares

Packages {foreign} and {haven} should cover 99% of your needs

```
ls("package:haven", pattern = "^read")
```

```
"read_dta"    "read_por"    "read_sas"    "read_sav"    "read_spss"
"read_stata"
"read_xpt"
```

```
ls("package:foreign", pattern = "^read")
```

```
"read.arff"    "read.dbf"    "read.dta"    "read.epiinfo" "read.mtp"
"read.octave"  "read.S"      "read.spss"   "read.ssd"     "read.systat"
"read.xport"
```

Import data from SQL databases

Importing tables from a database requires 3 steps :

1. Establish a connexion with the database (handle)
2. List the tables in the database
3. Import the table you need (using SQL queries for example)

Several packages are available (`{RODBC}`, `{RMySQL}`,`{dbplyr}`,`{DBI}`, ⋯)

Export R objects (1)

`write_delim()` (and potentially variations like `write_csv()`) are used to export flat files

It's syntax is:

```
write_delim(
  objectname, #dataset to export
  path = "output.csv", #Destination
  delim = ";"
)
```

Export R objects (2)

R objects may be exported into R binaries files in order to load them quicker later on. They are stored in:

- `.RData` (serialized and deserialized using functions `save()` and `load()`)
- `.RDS` (serialized and deserialized using functions `write_rds()` et `read_rds()`)

The main differences are:

- `save()` saves a bunch of objects, or the entirety of objects in the environment with `save.image()`.
- /!\ *It's not possible to choose the name of the object using the `load()` function (can overwrite existing objects)* /!\
- `write_rds()` saves a unique object and `read_rds()` allows the user to choose the name of what is loaded.

Tip : Several objects to save ? Store them in a list and use `list2env()` to unlist them in the global environment `.GlobalEnv`

```
list2env(dframe, envir = .GlobalEnv)
```

Export R objects (3)

```
an_object  <- "character"
an_object
save(an_object, file =
"my_work.Rdata")
an_object <- 2
load("my_work.Rdata")
an_object
```

```
an_object  <- "character"
write_rds(an_object, path =
"my_work.RDS")

an_object <- 2
another_object <-
read_rds("my_work.RDS")
an_object
another_object
a_list <- list(my = 2, fair = 5,
lady = 6)
a_list
list2env(a_list, envir =
.GlobalEnv)
```

Export R objects (4)

The duet `dput()` and `dget()` writes and reads R instructions that builds R objects. It is the privileged way to submit small pieces of objects on Stackoverflow :

```
my_list <- list(my = 2, fair = 5, lady = 6)
dput(my_list, file = "myfile.dput")
dput(my_list)
my_other_list <- dget("myfile.dput")
dput(iris[,5])
```

Data Manipulation with {dplyr} in the tidyverse

First step to master the tidyverse

What is {dplyr} ?

- A package by Hadley Wickham
- An efficient «grammar» of data manipulation…
- …made lisible with the «pipe» (`%>%`)…
- …to produce pretty, fast and elegant code.

Loading the package

```r
library(tidyverse) # loads all the tidyverse
```

```
#> -- Attaching packages --------------------------------------- tidyverse
1.2.1 --

#> v ggplot2 3.0.0     v purrr   0.2.5
#> v tibble  1.4.2     v dplyr   0.7.6
#> v tidyr   0.8.1     v stringr 1.3.1
#> v readr   1.1.1     v forcats 0.3.0

#> -- Conflicts ------------------------------------------
tidyverse_conflicts() --
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()    masks stats::lag()
```

```r
library(dplyr) # loads only dplyr
```

The main verbs of the `{dplyr}` grammar

Verbs dedicated to observation manipulation (rows) :

- `arrange()`
- `filter()`

Verbs dedicated to variables manipulation (column) :

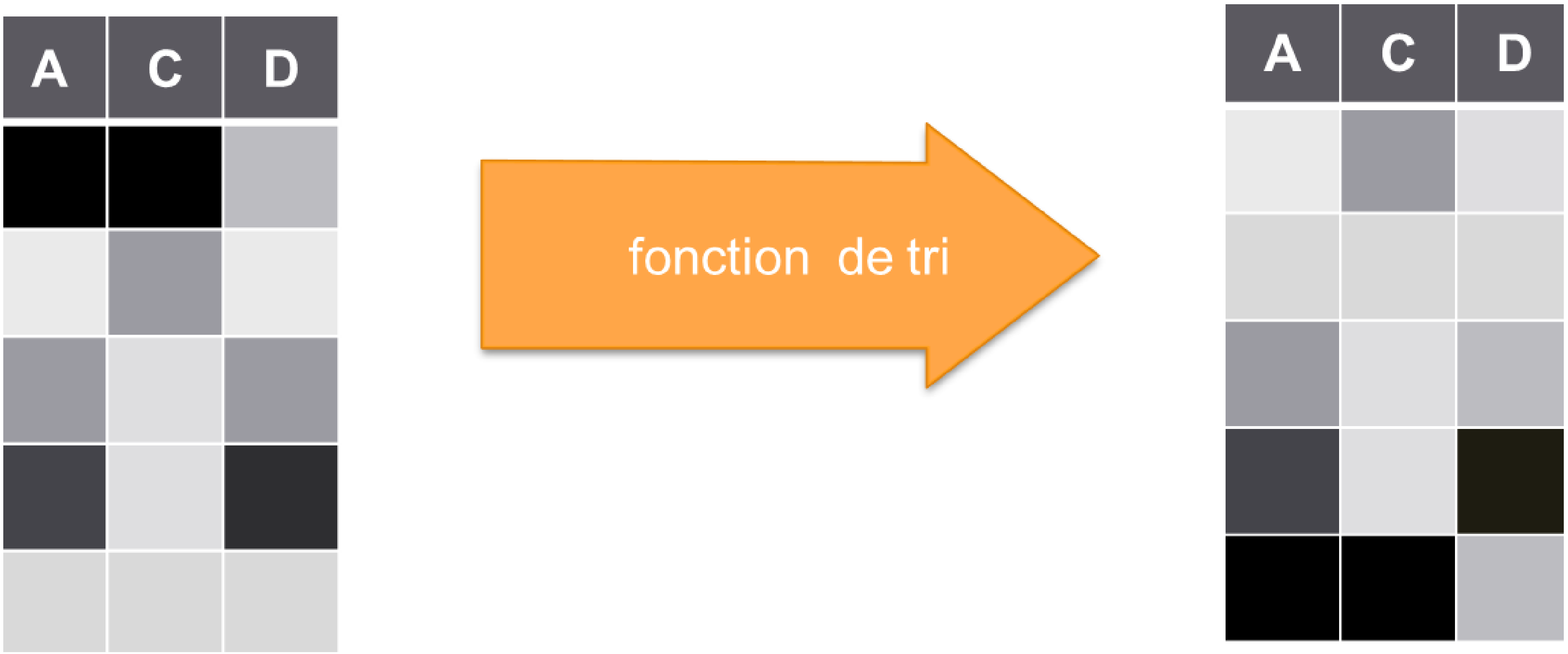
- `select()`
- `mutate()`

Verbs for combined operations on rows and columns :

- `summarise`
- `group_by`

> Each verb takes in first argument the dataset on which the action should be done

`arrange()` arranges the observations...



`arrange()` arranges the observations...

- in ascending order,
- or in descending order with `desc()`,
- combining several arranging criteria

```
arrange(the_data_frame, arranging_criteria_1, arranging_criteria_2)
```

Example :

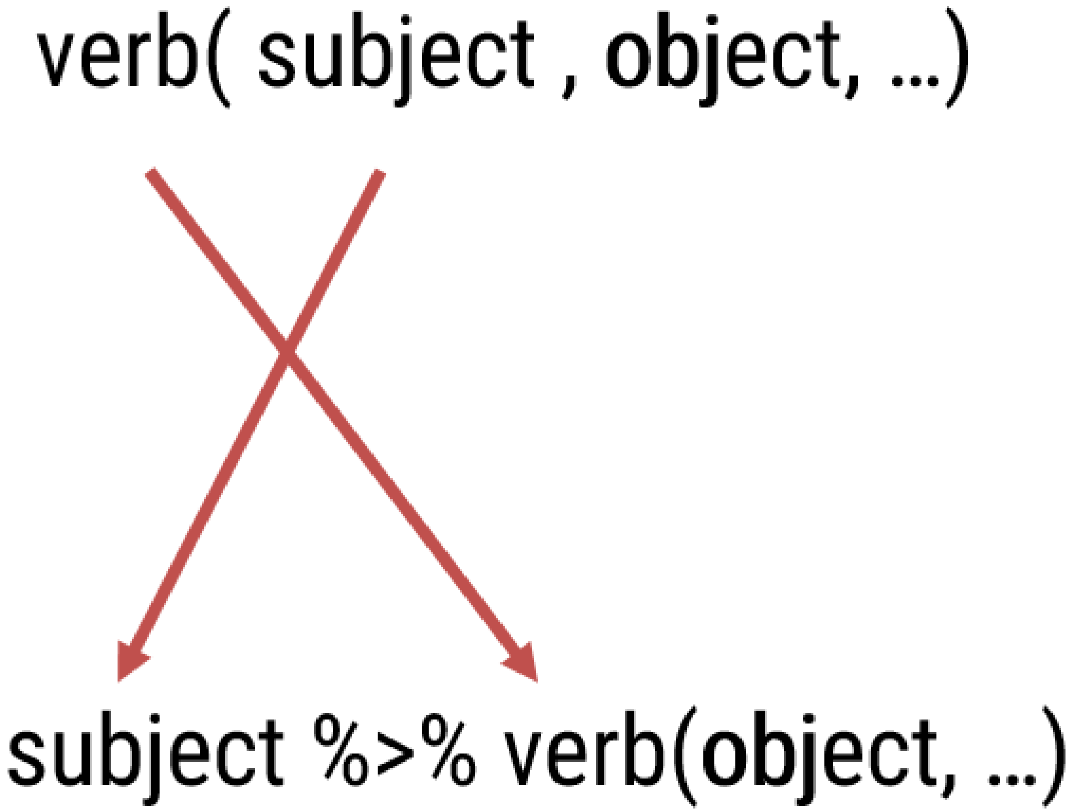
```
data(iris)
arrange(iris, Sepal.Length)
arrange(iris, desc(Sepal.Length), Petal.Length)
```

Write code like a narrative thread for you data : %>%

Let's digress for a moment

- keyboard shortcut : "Ctrl + Shift + M"

It's all about grammar...



Thus,

```
data(iris)

arrange(iris, Sepal.Length)
arrange(iris, desc(Sepal.Length), Petal.Length)
```

is equivalent to :

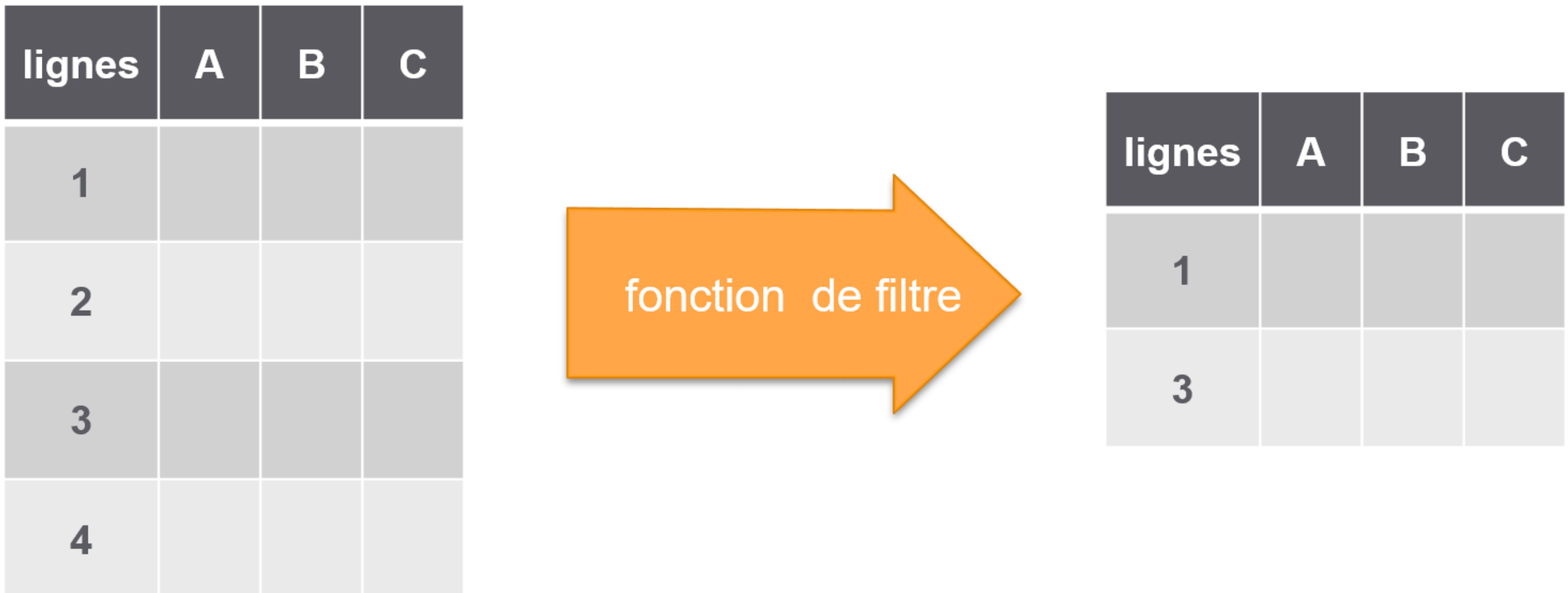
```
iris %>% arrange( Sepal.Length )
iris %>% arrange( desc(Sepal.Length), Petal.Length )
```

But also…

```
iris %>% head()
iris %>% names()
iris %>% dim()
iris %>% View()
iris %>% summary()
```

End of digression…

`filter()` reduces the height of the dataset



`filter()` reduces the height of the dataset

- filter operations are made with logical and boolean observations: `>`, `<`,`<=`, `>=`, `%in%`, `&`, `|` , `!`...

```
iris %>% filter( Species == "virginica" )
iris %>% filter( Species == "virginica", Petal.Width < 3 )
```

`|` means 'OR'

```
iris %>% filter( Species == "virginica" | Petal.Width > 1.3 )
```

```
#>    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
#> 1           7.0         3.2          4.7         1.4 versicolor
#> 2           6.4         3.2          4.5         1.5 versicolor
#> 3           6.9         3.1          4.9         1.5 versicolor
#> 4           6.5         2.8          4.6         1.5 versicolor
#> 5           6.3         3.3          4.7         1.6 versicolor
#> 6           5.2         2.7          3.9         1.4 versicolor
#> 7           5.9         3.0          4.2         1.5 versicolor
#> 8           6.1         2.9          4.7         1.4 versicolor
#> 9           6.7         3.1          4.4         1.4 versicolor
#> 10          5.6         3.0          4.5         1.5 versicolor
#> 11          6.2         2.2          4.5         1.5 versicolor
```

How to use `%in%` ?

```
iris %>% filter( Species %in% c("virginica","setosa") )
```

is equivalent to :

```
iris %>% filter( Species == "virginica" | Species == "setosa" )
```

Chain operations with `%>%`

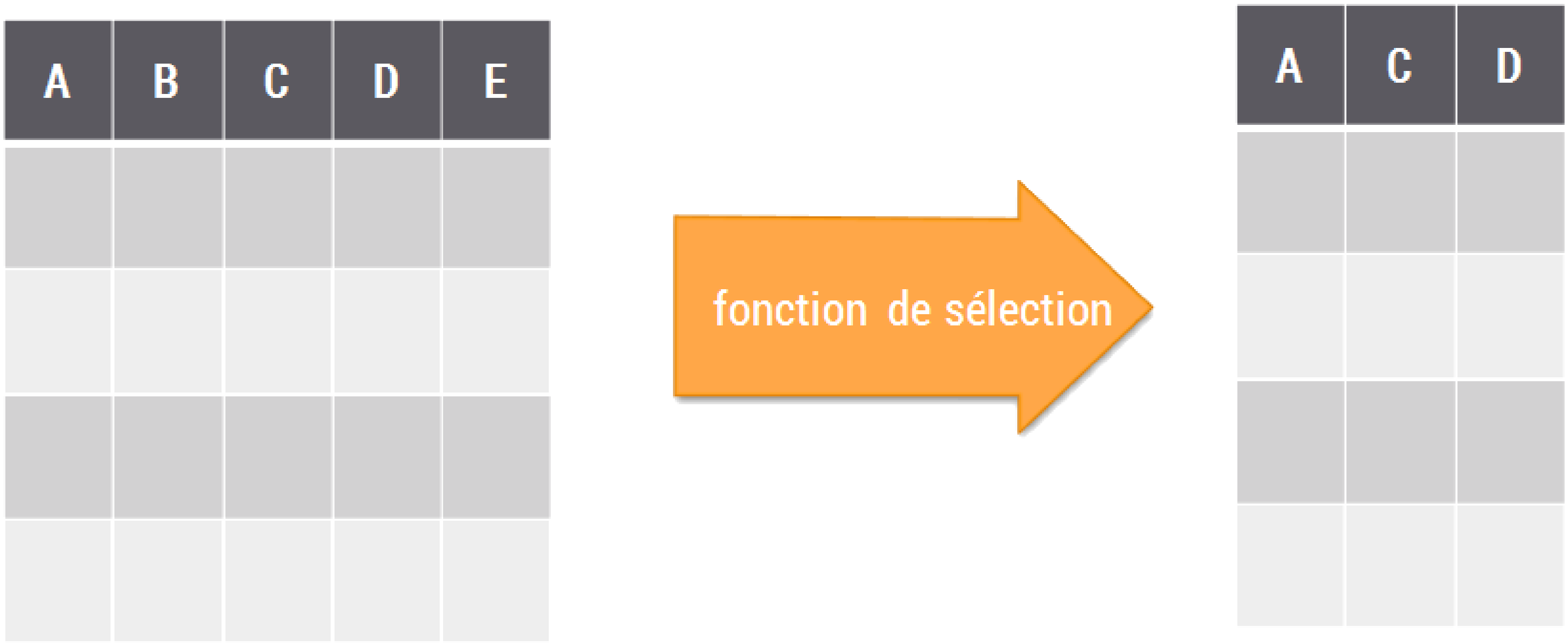
Second digression...

`%>%` avoids temporary objects, you can see it like a "then" in the narrative thread happening to your data

```
iris %>%
  arrange( Petal.Length ) %>%
  filter( Species == "setosa" ) %>%
  head()
```

```
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#> 1          4.6         3.6          1.0         0.2  setosa
#> 2          4.3         3.0          1.1         0.1  setosa
#> 3          5.8         4.0          1.2         0.2  setosa
#> 4          5.0         3.2          1.2         0.2  setosa
#> 5          4.7         3.2          1.3         0.2  setosa
#> 6          5.4         3.9          1.3         0.4  setosa
```

Reduce the dataset's width : `select()`



Reduce the dataset's width : `select()`

```
data(iris)
iris_tbl <- as.tbl( iris ) # transforms iris in a tibble

iris_tbl %>% select( Species, Sepal.Width, Petal.Width )
```

```
#> # A tibble: 150 x 3
#>    Species Sepal.Width Petal.Width
#>    <fct>         <dbl>       <dbl>
#>  1 setosa          3.5         0.2
#>  2 setosa          3           0.2
#>  3 setosa          3.2         0.2
#>  4 setosa          3.1         0.2
#>  5 setosa          3.6         0.2
#>  6 setosa          3.9         0.4
#>  7 setosa          3.4         0.3
#>  8 setosa          3.4         0.2
#>  9 setosa          2.9         0.2
#> 10 setosa          3.1         0.1
#> # ... with 140 more rows
```

"helpers" to select variables

- `-`
- `starts_with()`
- `ends_with()`
- `contains()`
- `matches()`
- `:`

Selection is simplified by "helpers" (1)

```
iris_tbl %>% select( -Petal.Length, -Petal.Width )
```

```
#> # A tibble: 150 x 3
#>    Sepal.Length Sepal.Width Species
#>           <dbl>       <dbl> <fct>
#>  1          5.1         3.5 setosa
#>  2          4.9         3   setosa
#>  3          4.7         3.2 setosa
#>  4          4.6         3.1 setosa
#>  5          5           3.6 setosa
#>  6          5.4         3.9 setosa
#>  7          4.6         3.4 setosa
#>  8          5           3.4 setosa
#>  9          4.4         2.9 setosa
#> 10          4.9         3.1 setosa
#> # ... with 140 more rows
```

Selection is simplified by "helpers" (2)

```
iris_tbl %>% select( starts_with("Petal") )
```

```
#> # A tibble: 150 x 2
#>    Petal.Length Petal.Width
#>           <dbl>       <dbl>
#>  1          1.4         0.2
#>  2          1.4         0.2
#>  3          1.3         0.2
#>  4          1.5         0.2
#>  5          1.4         0.2
#>  6          1.7         0.4
#>  7          1.4         0.3
#>  8          1.5         0.2
#>  9          1.4         0.2
#> 10          1.5         0.1
#> # ... with 140 more rows
```

Selection is simplified by "helpers" (3)

```
iris_tbl %>% select( -starts_with("Petal") )
```

```
#> # A tibble: 150 x 3
#>    Sepal.Length Sepal.Width Species
#>           <dbl>       <dbl> <fct>
#>  1          5.1         3.5 setosa
#>  2          4.9         3   setosa
#>  3          4.7         3.2 setosa
#>  4          4.6         3.1 setosa
#>  5          5           3.6 setosa
#>  6          5.4         3.9 setosa
#>  7          4.6         3.4 setosa
#>  8          5           3.4 setosa
#>  9          4.4         2.9 setosa
#> 10          4.9         3.1 setosa
#> # ... with 140 more rows
```

Selection is simplified by "helpers" (4)

```
iris_tbl %>% select( -ends_with("Width") )
```

```
#> # A tibble: 150 x 3
#>    Sepal.Length Petal.Length Species
#>           <dbl>        <dbl> <fct>  
#>  1          5.1          1.4 setosa 
#>  2          4.9          1.4 setosa 
#>  3          4.7          1.3 setosa 
#>  4          4.6          1.5 setosa 
#>  5          5            1.4 setosa 
#>  6          5.4          1.7 setosa 
#>  7          4.6          1.4 setosa 
#>  8          5            1.5 setosa 
#>  9          4.4          1.4 setosa 
#> 10          4.9          1.5 setosa 
#> # ... with 140 more rows
```

Selection is simplified by "helpers" (5)

```
iris_tbl %>% select( -contains("etal") )
```

```
#> # A tibble: 150 x 3
#>    Sepal.Length Sepal.Width Species
#>           <dbl>       <dbl> <fct>
#>  1          5.1         3.5 setosa
#>  2          4.9         3   setosa
#>  3          4.7         3.2 setosa
#>  4          4.6         3.1 setosa
#>  5          5           3.6 setosa
#>  6          5.4         3.9 setosa
#>  7          4.6         3.4 setosa
#>  8          5           3.4 setosa
#>  9          4.4         2.9 setosa
#> 10          4.9         3.1 setosa
#> # ... with 140 more rows
```

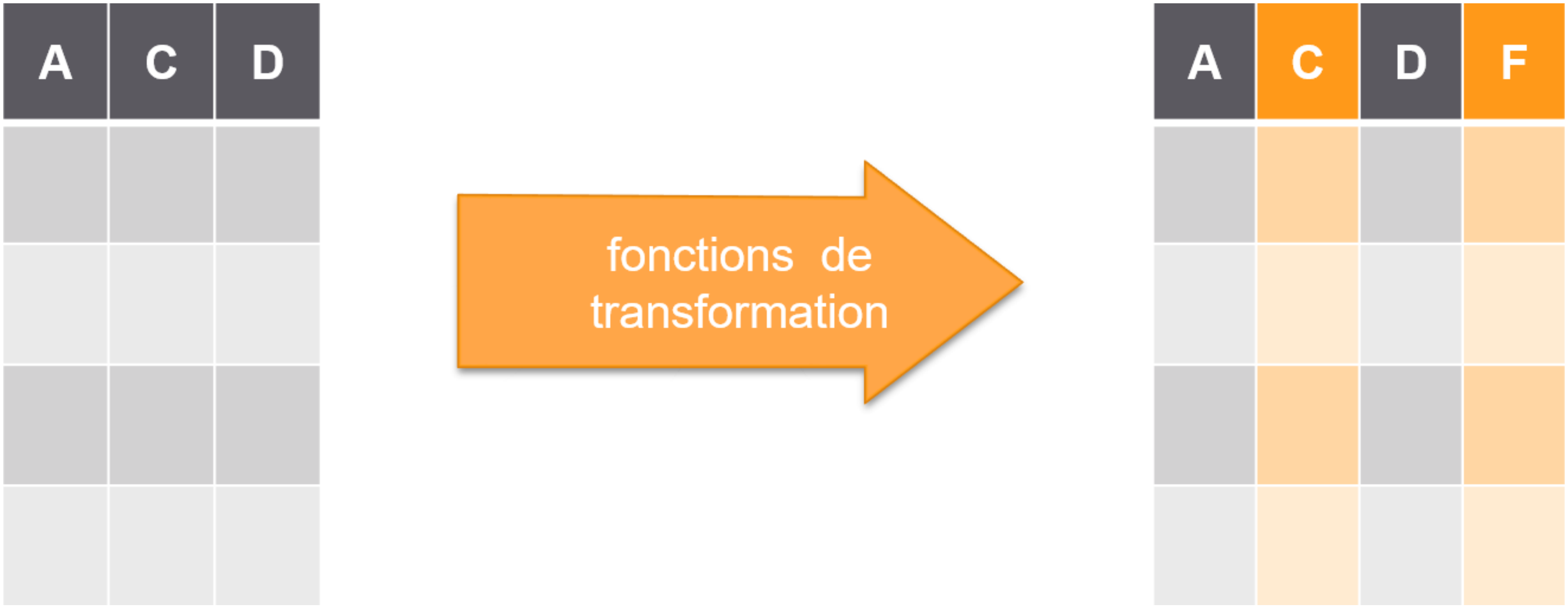
Selection is simplified by "helpers" (6)

```r
iris_tbl %>% select( Sepal.Width:Petal.Width )
```

```
#> # A tibble: 150 x 3
#>    Sepal.Width Petal.Length Petal.Width
#>          <dbl>        <dbl>       <dbl>
#>  1         3.5          1.4         0.2
#>  2         3            1.4         0.2
#>  3         3.2          1.3         0.2
#>  4         3.1          1.5         0.2
#>  5         3.6          1.4         0.2
#>  6         3.9          1.7         0.4
#>  7         3.4          1.4         0.3
#>  8         3.4          1.5         0.2
#>  9         2.9          1.4         0.2
#> 10         3.1          1.5         0.1
#> # ... with 140 more rows
```

`mutate()` to modify and create variables

`mutate()` add columns (and extend the dataset's width) or modifies existing columns



`mutate()` to modify and create variables

```
dataset %>% mutate( new_variable = dummy_operations(),
                    my_variable = other_operations(my_variable),
                    ...
                   )
```

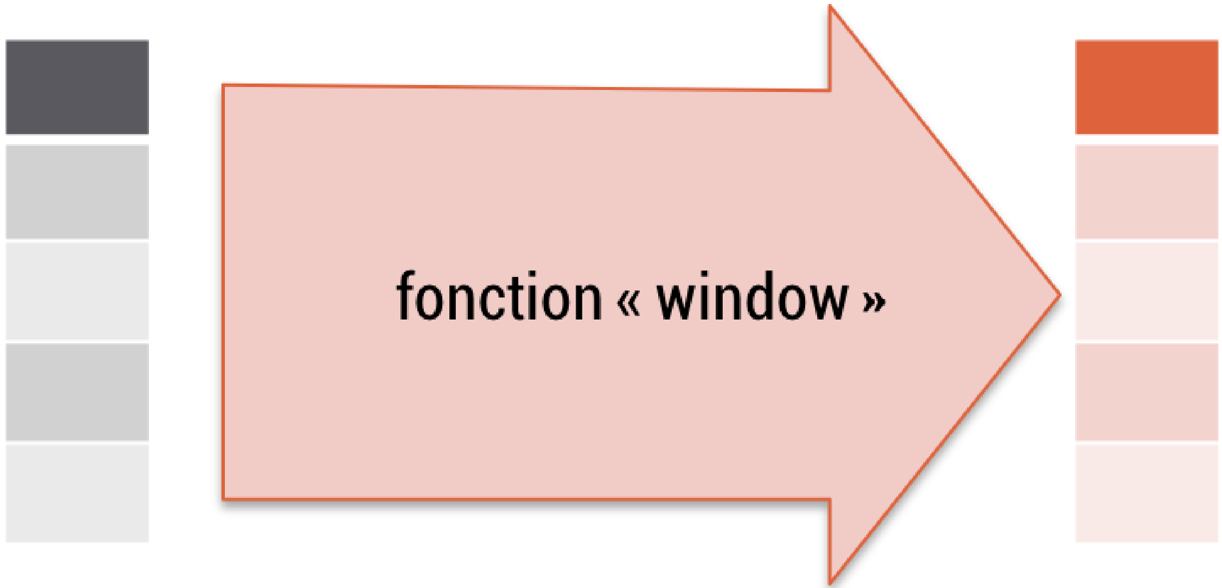
Example:

```
iris_tbl %>%
  mutate( Sepal.Length = Sepal.Length / 10,
          ratio_Sepal = Sepal.Length / Sepal.Width,
          ratio_Petal = Petal.Length / Petal.Width ) %>%
  head( 2 )
```

```
#> # A tibble: 2 x 7
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species ratio_Sepal
#>          <dbl>       <dbl>        <dbl>       <dbl> <fct>         <dbl>
#> 1         0.51         3.5          1.4         0.2 setosa        0.146
#> 2         0.49         3            1.4         0.2 setosa        0.163
#> # ... with 1 more variable: ratio_Petal <dbl>
```

Vectorized windows functions to be used with `mutate`

Vectorized functions take vectors as input and return vectors of the same length as output

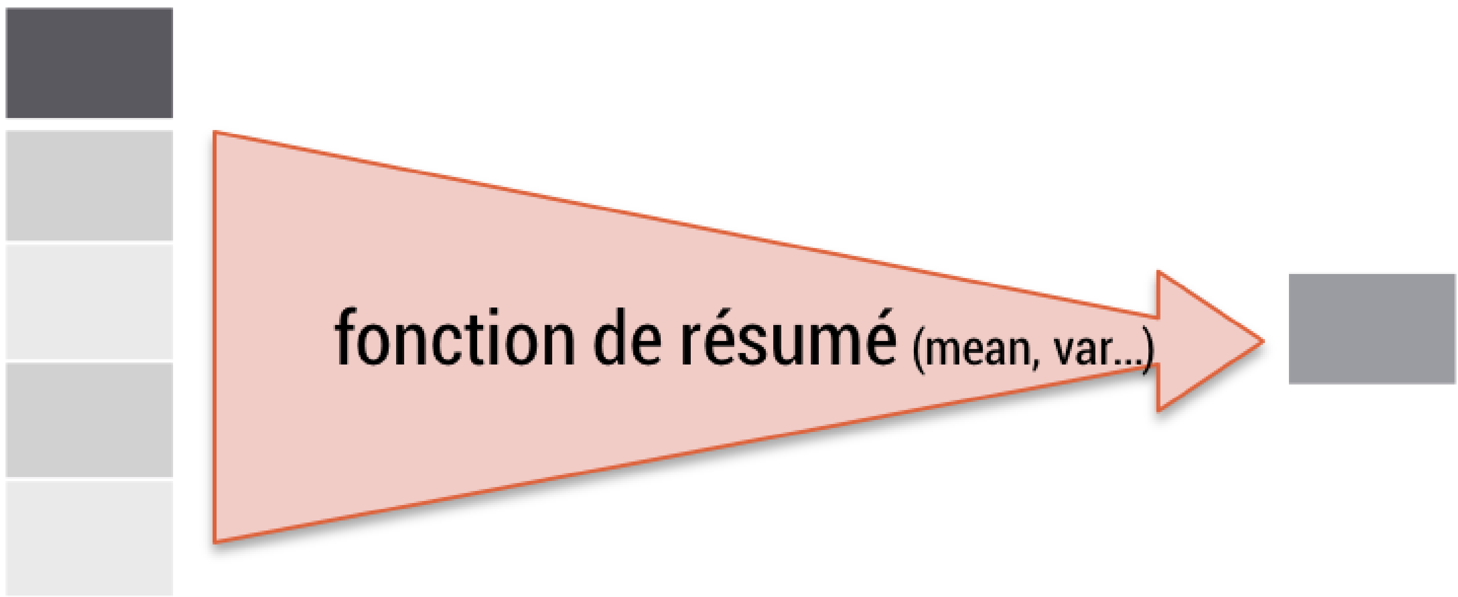


Some useful vectorized functions

- offsets: `lag()` (+1), `lead()` (-1)
- cumulative aggregates: `cumsum()`, `cumprod()` ...
- ranks: `dense_rank()`, `min_rank()` ...
- arithmetic operations and logical comparisons : +, −, *, and >, <,<=, >=
- misc: `case_when()`, `coalesce()`, `na_if()` ...

Summary functions: `summarise()`

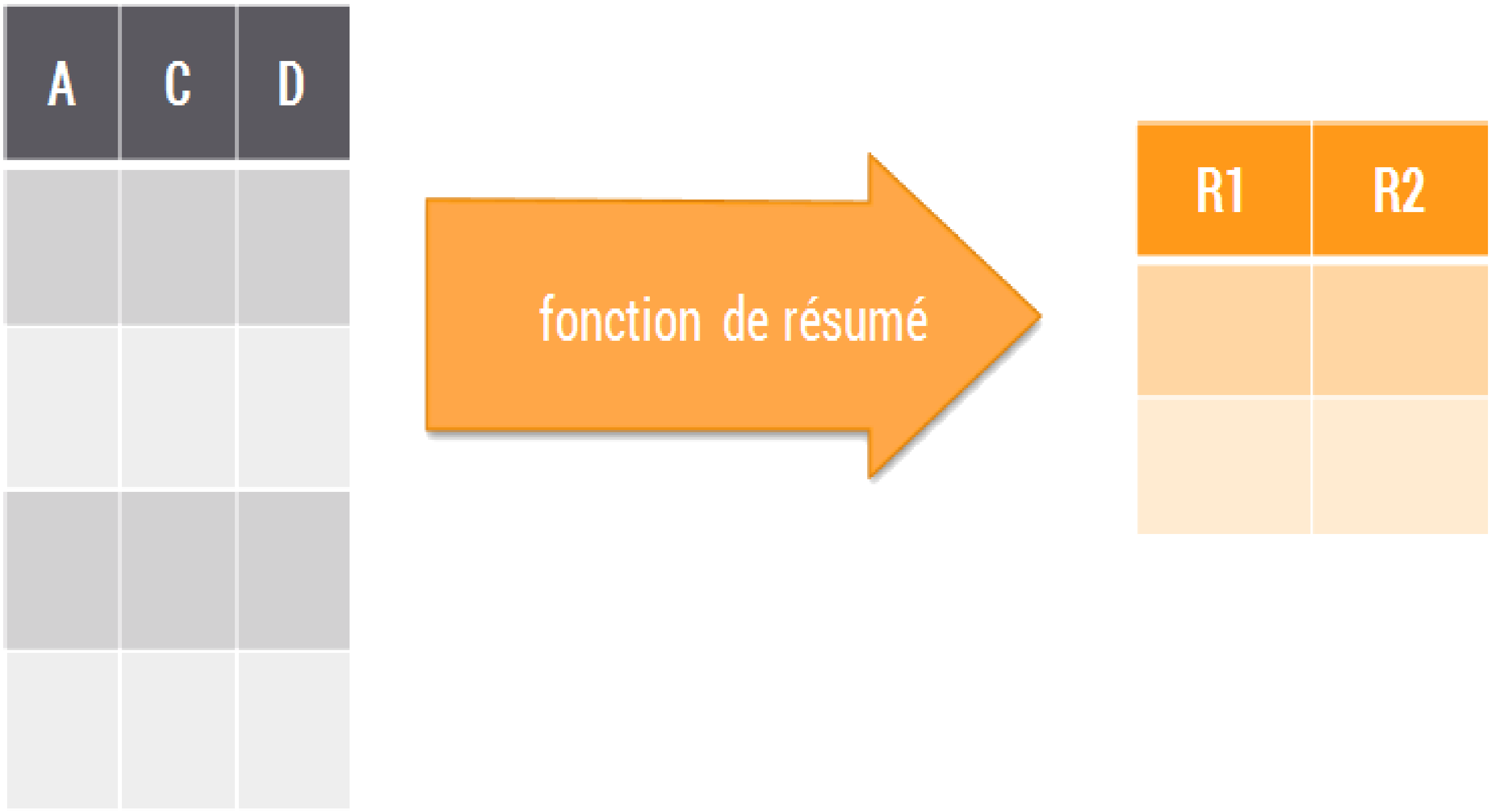
Summary functions take vectors as input but they return one (and only one) value



Some useful summary functions :

- Position parameters : `mean()`, `median()`
- Counts : `n()` without any parameter, only available inside `summarise`
- Spread parameters : `var()`, `sd()`, `IQR()`
- Ranks : `quantile()`, `min()`, `max()`

Summary functions: `summarise()`



Summary functions: `summarise()`

An example of `summarise()`

```
iris %>% summarise(
          average = mean( Sepal.Length ),
          variance = var( Sepal.Length ),
          count = n()
          )
```

```
#>    average  variance count
#> 1 5.843333 0.6856935   150
```

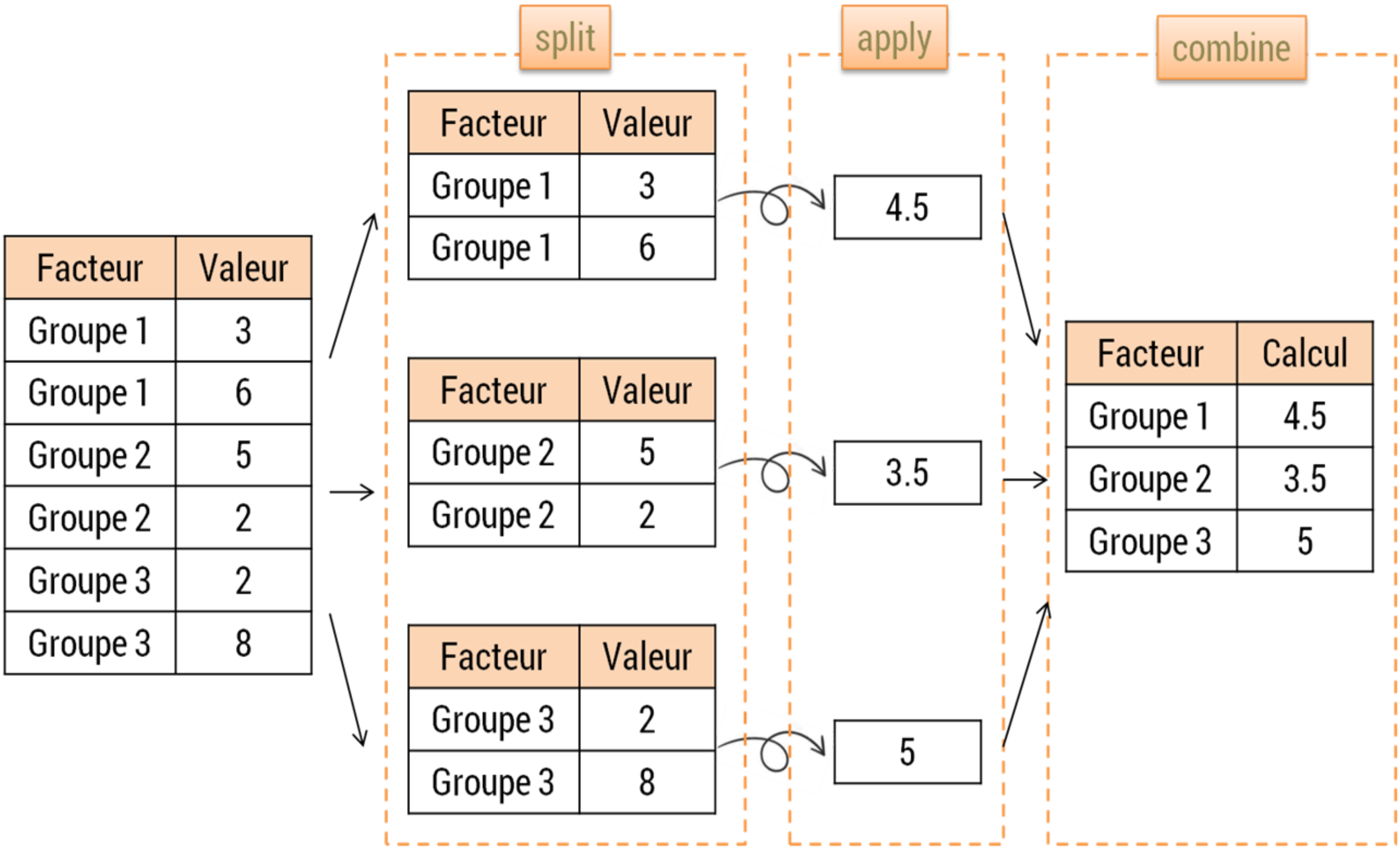
The adverb `group_by()`

> An adverbial phrase is a group of words operating adverbially, meaning that their syntactic function is to modify a verb, an adjective, or an adverb

Wikipedia

In other words, we're going to modify the conditions in which the verb operates.

The split - apply - combine strategy



The split - apply - combine strategy

The split-apply-combine, `{dplyr}` like...

... is a combination of `group_by()` + `summarise()`

```
iris %>%
  group_by( Species ) %>%
  summarise( average_Sepal.Length = mean( Sepal.Length ),
             variance_Sepal.Length = var( Sepal.Length ),
             count = n()
            )
```

```
#> # A tibble: 3 x 4
#>   Species    average_Sepal.Length variance_Sepal.Length count
#>   <fct>                     <dbl>                 <dbl> <int>
#> 1 setosa                     5.01                 0.124    50
#> 2 versicolor                 5.94                 0.266    50
#> 3 virginica                  6.59                 0.404    50
```

`group_by()` operates the «split» and `summarise()` operates the summarising to each subset

Example of data flow

```r
iris %>%
  filter( Species == "setosa" ) %>%
        select( -Species ) %>%
        mutate( surface_totale = Petal.Length * Petal.Width +
                 Sepal.Length * Sepal.Width,
                surface_totale = surface_totale * 100,
                surface_totale = round( surface_totale )
               ) %>%
  arrange( surface_totale )
```

```
#>     Sepal.Length Sepal.Width Petal.Length Petal.Width surface_totale
#> 1            4.5         2.3          1.3         0.3           1074
#> 2            4.3         3.0          1.1         0.1           1301
#> 3            4.4         2.9          1.4         0.2           1304
#> 4            4.4         3.0          1.3         0.2           1346
#> 5            4.4         3.2          1.3         0.2           1434
#> 6            4.8         3.0          1.4         0.1           1454
#> 7            4.6         3.1          1.5         0.2           1456
#> 8            4.8         3.0          1.4         0.3           1482
#> 9            4.9         3.0          1.4         0.2           1498
#> 10           4.6         3.2          1.4         0.2           1500
#> 11           4.8         3.1          1.6         0.2           1520
```

Some particular filters: `distinct()`

… removes duplicates from a dataset

```
iris_tbl %>% distinct()
```

```
#> # A tibble: 149 x 5
#>    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#>           <dbl>       <dbl>        <dbl>       <dbl> <fct>
#>  1          5.1         3.5          1.4         0.2 setosa
#>  2          4.9         3            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           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           3.4          1.5         0.2 setosa
#>  9          4.4         2.9          1.4         0.2 setosa
#> 10          4.9         3.1          1.5         0.1 setosa
#> # ... with 139 more rows
```

Some particular filters: `distinct()`

… removes duplicates from a dataset

```
iris_tbl %>% distinct( Species )
```

```
#> # A tibble: 3 x 1
#>   Species
#>   <fct>
#> 1 setosa
#> 2 versicolor
#> 3 virginica
```

Some particular filters: `sample_n()`

...randomly select a number of rows

```
iris_tbl %>% dim()
```

```
#> [1] 150   5
```

```
iris_tbl %>% sample_n( size = 60 )
```

```
#> # A tibble: 60 x 5
#>    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#>           <dbl>       <dbl>        <dbl>       <dbl> <fct>
#>  1          6.9         3.1          4.9         1.5 versicolor
#>  2          6.4         3.2          4.5         1.5 versicolor
#>  3          4.7         3.2          1.3         0.2 setosa
#>  4          5.6         3            4.5         1.5 versicolor
#>  5          6.4         2.9          4.3         1.3 versicolor
#>  6          4.4         2.9          1.4         0.2 setosa
#>  7          4.9         2.5          4.5         1.7 virginica
#>  8          5.1         3.7          1.5         0.4 setosa
#>  9          5.3         3.7          1.5         0.2 setosa
#> 10          5.5         2.3          4           1.3 versicolor
#> # ... with 50 more rows
```

... and `sample_frac()`

...randomly select a fraction of rows

```
iris_tbl %>% sample_frac( size = 0.7 ) # select 70% of rows
```

```
#> # A tibble: 105 x 5
#>    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#>           <dbl>       <dbl>        <dbl>       <dbl> <fct>
#>  1          5           2.3          3.3         1   versicolor
#>  2          4.8         3.4          1.9         0.2 setosa
#>  3          5.4         3.4          1.7         0.2 setosa
#>  4          4.8         3            1.4         0.1 setosa
#>  5          4.8         3.4          1.6         0.2 setosa
#>  6          6.3         3.3          6           2.5 virginica
#>  7          6.9         3.1          5.4         2.1 virginica
#>  8          5.7         2.5          5           2   virginica
#>  9          5.8         2.6          4           1.2 versicolor
#> 10          6.2         2.9          4.3         1.3 versicolor
#> # ... with 95 more rows
```

Some particular filters: `slice()`

…select rows by position

it is better to avoid working like this and give priority to the `filter` verb

```
iris %>% slice( 25:32 )
```

```
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#> 1          4.8         3.4          1.9         0.2  setosa
#> 2          5.0         3.0          1.6         0.2  setosa
#> 3          5.0         3.4          1.6         0.4  setosa
#> 4          5.2         3.5          1.5         0.2  setosa
#> 5          5.2         3.4          1.4         0.2  setosa
#> 6          4.7         3.2          1.6         0.2  setosa
#> 7          4.8         3.1          1.6         0.2  setosa
#> 8          5.4         3.4          1.5         0.4  setosa
```

Get the top : `top_n()`

…selects the n maximum observations for a variable. **(tie is kept)**

```
iris %>% top_n(2, Sepal.Length)
```

```
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
#> 1          7.7         3.8          6.7         2.2 virginica
#> 2          7.7         2.6          6.9         2.3 virginica
#> 3          7.7         2.8          6.7         2.0 virginica
#> 4          7.9         3.8          6.4         2.0 virginica
#> 5          7.7         3.0          6.1         2.3 virginica
```

Rename variables: `rename()`

```
data.frame %>% rename( new_name = old_name )
```

Example:

Rename variables: `rename()`

```
data.frame %>% rename( new_name = old_name )
```

Example:

```
iris_tbl %>% rename( petal_length = Petal.Length )
```

```
#> # A tibble: 150 x 5
#>    Sepal.Length Sepal.Width petal_length Petal.Width Species
#>           <dbl>       <dbl>        <dbl>       <dbl> <fct>
#>  1          5.1         3.5          1.4         0.2 setosa
#>  2          4.9         3            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           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           3.4          1.5         0.2 setosa
#>  9          4.4         2.9          1.4         0.2 setosa
#> 10          4.9         3.1          1.5         0.1 setosa
#> # ... with 140 more rows
```

Deeper in dplyr

mastering the tidyverse

Variants of `mutate()` like weapons of "mass construction"

When a transformation must be done on a batch of variables, some variants of `mutate()` can be useful :

- `mutate_all()`
- `mutate_at()`
- `mutate_if()`

mutate_if()

Use a condition to define which variables will be modified

```
iris_tbl %>% mutate_if( is.numeric, scale ) %>% head(3)
```

```
#> # A tibble: 3 x 5
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#>          <dbl>       <dbl>        <dbl>       <dbl> <fct>
#> 1       -0.898       1.02         -1.34       -1.31 setosa
#> 2       -1.14       -0.132        -1.34       -1.31 setosa
#> 3       -1.38        0.327        -1.39       -1.31 setosa
```

mutate_if()

Use with `funs()` to create new variables (and not to overwrite the existing one):

```
iris_tbl %>% mutate_if( is.numeric, funs( new = scale ) )%>% head(3)
```

```
#> # A tibble: 3 x 9
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#>          <dbl>       <dbl>        <dbl>       <dbl> <fct>
#> 1          5.1         3.5          1.4         0.2 setosa
#> 2          4.9         3            1.4         0.2 setosa
#> 3          4.7         3.2          1.3         0.2 setosa
#> # ... with 4 more variables: Sepal.Length_new <dbl>,
#> #   Sepal.Width_new <dbl>, Petal.Length_new <dbl>, Petal.Width_new <dbl>
```

Examples of `mutate_if()`

```
iris_tbl %>% mutate_if( is.numeric, round, digits = 0 ) %>% head()
```

```
#> # A tibble: 6 x 5
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#>          <dbl>       <dbl>        <dbl>       <dbl> <fct>
#> 1            5           4            1           0 setosa
#> 2            5           3            1           0 setosa
#> 3            5           3            1           0 setosa
#> 4            5           3            2           0 setosa
#> 5            5           4            1           0 setosa
#> 6            5           4            2           0 setosa
```

mutate_at()

Use variable names as quoted characters to define which variables will be modified

```
iris %>%
  mutate_at( c("Sepal.Length","Petal.Length"), as.factor ) %>%
  summary()
```

```
#>  Sepal.Length  Sepal.Width     Petal.Length  Petal.Width
#>  5       :10   Min.   :2.000   1.4     :13   Min.   :0.100
#>  5.1     : 9   1st Qu.:2.800   1.5     :13   1st Qu.:0.300
#>  6.3     : 9   Median :3.000   4.5     : 8   Median :1.300
#>  5.7     : 8   Mean   :3.057   5.1     : 8   Mean   :1.199
#>  6.7     : 8   3rd Qu.:3.300   1.3     : 7   3rd Qu.:1.800
#>  5.5     : 7   Max.   :4.400   1.6     : 7   Max.   :2.500
#>  (Other):99                    (Other):94
#>        Species
#>  setosa    :50
#>  versicolor:50
#>  virginica :50
#>
#>
#>
#>
```

mutate_at()

Use helpers of `select()` with `vars()`

```
iris %>%
  mutate_at( vars( starts_with( "Petal" ) ), as.factor ) %>%
  summary()
```

```
#>   Sepal.Length    Sepal.Width     Petal.Length  Petal.Width
#>  Min.   :4.300   Min.   :2.000   1.4    :13    0.2    :29
#>  1st Qu.:5.100   1st Qu.:2.800   1.5    :13    1.3    :13
#>  Median :5.800   Median :3.000   4.5    : 8    1.5    :12
#>  Mean   :5.843   Mean   :3.057   5.1    : 8    1.8    :12
#>  3rd Qu.:6.400   3rd Qu.:3.300   1.3    : 7    1.4    : 8
#>  Max.   :7.900   Max.   :4.400   1.6    : 7    2.3    : 8
#>                                  (Other):94    (Other):68
#>        Species
#>  setosa    :50
#>  versicolor:50
#>  virginica :50
#>
#>
#>
#>
```

About `case_when`

Rather than using `ifelse` embedded in a `mutate`, you can use `case_when`. The sequences are read in order. And are written as `condition ~ result`.

```
iris %>%
  mutate(
    new = case_when(
      Sepal.Length > 6 & Petal.Length > 3.1 ~ "long",
      Sepal.Width > 2.3 & Petal.Width > 1.8 ~ "wide" ,
      TRUE                                  ~ "usual"
    ),
    new = as.factor(new)
  ) %>%
  select(new) %>%
  summary()
```

```
#>      new
#>   long :61
#>   usual:84
#>   wide : 5
```

Variants of `summarise()` like weapons of mass "summarizing"

- `summarise_all()`
- `summarise_at()`
- `summarise_if()`

summarise_at & summarise_if

```
iris %>% summarise_at( vars( Sepal.Length:Petal.Width ), mean )
```

```
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width
#> 1     5.843333    3.057333        3.758    1.199333
```

```
iris %>% summarise_at( vars( starts_with( "Sepal" ) ), var )
```

```
#>   Sepal.Length Sepal.Width
#> 1    0.6856935   0.1899794
```

summarise_at & summarise_if

```
iris %>% summarise_at( vars( Sepal.Length:Petal.Width ), mean )
```

```
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width
#> 1     5.843333    3.057333        3.758    1.199333
```

```
iris %>% summarise_at( vars( starts_with( "Sepal" ) ), var )
```

```
#>   Sepal.Length Sepal.Width
#> 1    0.6856935   0.1899794
```

In order to apply several functions to the same `vars` :

```
iris %>% summarise_at( vars( starts_with( "Sepal" ) ), funs(var, mean) )
```

```
#>   Sepal.Length_var Sepal.Width_var Sepal.Length_mean Sepal.Width_mean
#> 1        0.6856935       0.1899794          5.843333         3.057333
```

summarise_at & summarise_if

```
iris %>% summarise_if( is.numeric, mean )
```

```
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width
#> 1     5.843333    3.057333        3.758    1.199333
```

```
iris %>% summarise_if( is.numeric, funs(mean,var,max) )
```

```
#>   Sepal.Length_mean Sepal.Width_mean Petal.Length_mean Petal.Width_mean
#> 1          5.843333         3.057333             3.758         1.199333
#>   Sepal.Length_var Sepal.Width_var Petal.Length_var Petal.Width_var
#> 1        0.6856935       0.1899794         3.116278       0.5810063
#>   Sepal.Length_max Sepal.Width_max Petal.Length_max Petal.Width_max
#> 1              7.9             4.4              6.9             2.5
```

```
iris %>% summarise_if( is.factor, nlevels )
```

```
#>   Species
#> 1       3
```

Example of data flow

```
iris %>%
    mutate( classe = cut( Sepal.Length, breaks =  3,
                          labels=c("small", "medium", "large") ) ) %>%
  group_by( classe ) %>%
  summarise_if( is.numeric, funs( min, max, mean ) )
```

```
#> # A tibble: 3 x 13
#>   classe Sepal.Length_min Sepal.Width_min Petal.Length_min Petal.Width_min
#>   <fct>             <dbl>           <dbl>            <dbl>           <dbl>
#> 1 small               4.3             2                1               0.1
#> 2 medium              5.6             2.2              1.2             0.2
#> 3 large               6.8             2.6              4.7             1.4
#> # ... with 8 more variables: Sepal.Length_max <dbl>,
#> #   Sepal.Width_max <dbl>, Petal.Length_max <dbl>, Petal.Width_max <dbl>,
#> #   Sepal.Length_mean <dbl>, Sepal.Width_mean <dbl>,
#> #   Petal.Length_mean <dbl>, Petal.Width_mean <dbl>
```

Mutating join with dplyr

join one table to columns from another

Common minimal example

```r
A <- data.frame(id = LETTERS[1:10], var1 = 1:10)
B <- data.frame(id = LETTERS[5:14], var2 = sample(1:10))
library(dplyr)
```

Let's illustrate our words with this minimal example :

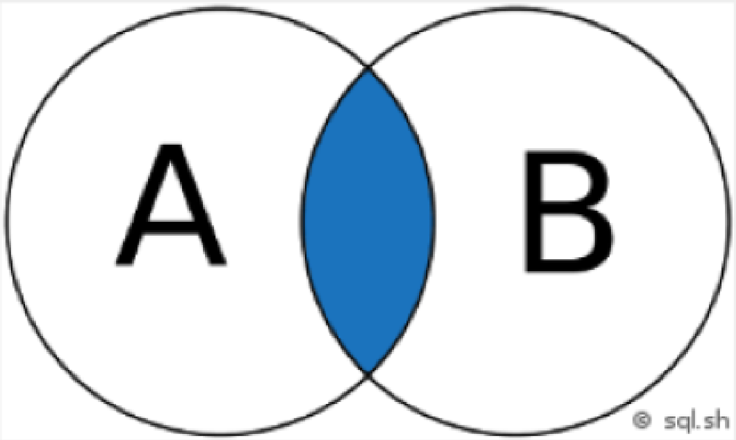
```r
A
```

```
#>    id var1
#> 1   A    1
#> 2   B    2
#> 3   C    3
#> 4   D    4
#> 5   E    5
#> 6   F    6
#> 7   G    7
#> 8   H    8
#> 9   I    9
#> 10  J   10
```

```r
B
```

```
#>    id var2
#> 1   E    9
#> 2   F    7
#> 3   G    4
#> 4   H   10
#> 5   I    2
#> 6   J    5
#> 7   K    6
#> 8   L    1
#> 9   M    8
#> 10  N    3
```

INNER JOIN – Retain only rows with matches

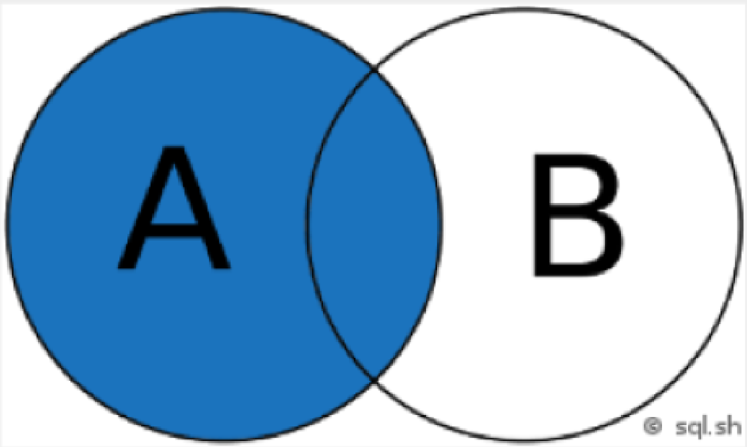


— Intersection de 2 ensembles

```
A %>% inner_join( B, by = "id" )
```

```
#>   id var1 var2
#> 1  E    5    9
#> 2  F    6    7
#> 3  G    7    4
#> 4  H    8   10
#> 5  I    9    2
#> 6  J   10    5
```

LEFT JOIN – matching values from B to A

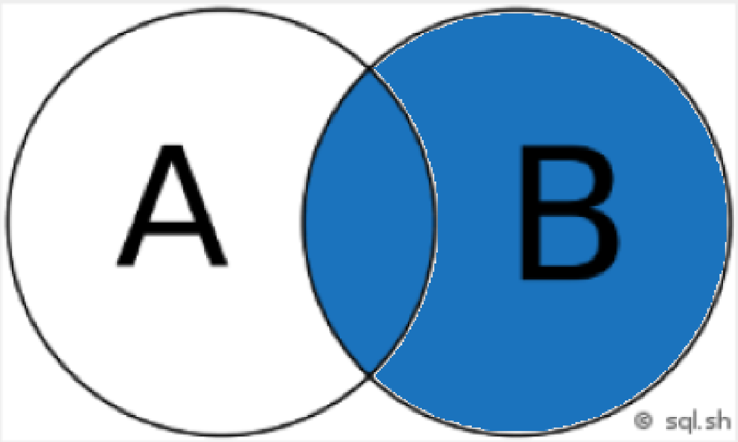


— Jointure gauche (LEFT JOINT)

```
A %>% left_join( B, by = "id" )
```

```
#>    id var1 var2
#> 1   A    1   NA
#> 2   B    2   NA
#> 3   C    3   NA
#> 4   D    4   NA
#> 5   E    5    9
#> 6   F    6    7
#> 7   G    7    4
#> 8   H    8   10
#> 9   I    9    2
#> 10  J   10    5
```

RIGHT JOIN – matching values from A to B

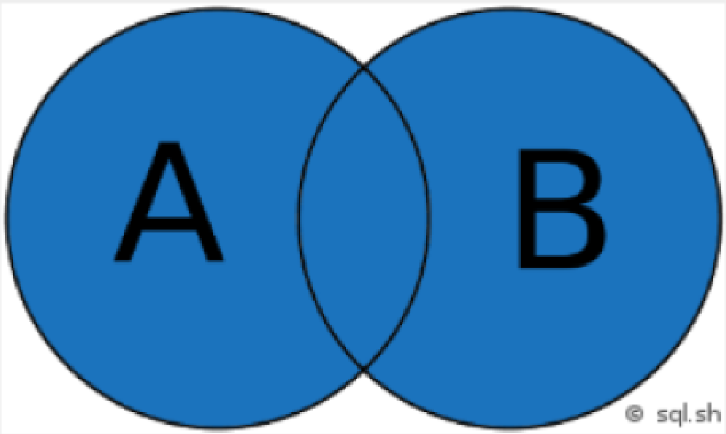


— Intersection de 2 ensembles

```
A %>% right_join( B, by = "id" )
```

```
#>    id var1 var2
#> 1   E    5    9
#> 2   F    6    7
#> 3   G    7    4
#> 4   H    8   10
#> 5   I    9    2
#> 6   J   10    5
#> 7   K   NA    6
#> 8   L   NA    1
#> 9   M   NA    8
#> 10  N   NA    3
```

FULL JOIN – retains all values and all rows

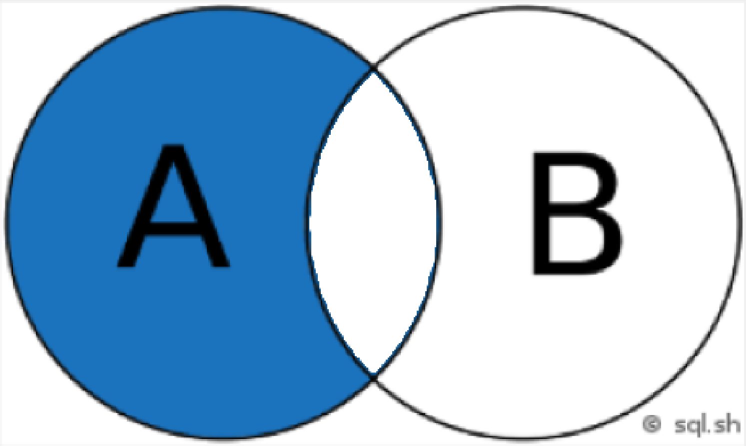


— Union de 2 ensembles

```
A %>% full_join( B, by = "id" )
```

```
#>    id var1 var2
#> 1   A    1   NA
#> 2   B    2   NA
#> 3   C    3   NA
#> 4   D    4   NA
#> 5   E    5    9
#> 6   F    6    7
#> 7   G    7    4
#> 8   H    8   10
#> 9   I    9    2
#> 10  J   10    5
#> 11  K   NA    6
#> 12  L   NA    1
#> 13  M   NA    8
#> 14  N   NA    3
```

ANTI JOIN – anti-matches values from B to A



— Jointure gauche (LEFT JOINT)

matches rows in A that are not in B

```
A %>% anti_join( B, by = "id" )
```

```
#>   id var1
#> 1  A    1
#> 2  B    2
#> 3  C    3
#> 4  D    4
```

ANTI JOIN example

`anti_join()` is very useful when separating a dataset in train and test datasets

```
train <- iris %>% sample_frac( 0.7 )
test <- iris %>% anti_join( train )
dim(train)
```

```
#> [1] 105   5
```

```
dim(test)
```

```
#> [1] 44  5
```

ANTI JOIN example

`anti_join()` is very useful when separating a dataset in train and test datasets

```
train <- iris %>% sample_frac( 0.7 )
test <- iris %>% anti_join( train )
dim(train)
```

```
#> [1] 105   5
```

```
dim(test)
```

```
#> [1] 44  5
```

It's even possible to respect a stratification in the sample:

```
train <- iris %>% group_by( Species ) %>% sample_frac( 0.7 )
test <- iris %>% anti_join( train )
dim(train)
```

```
#> [1] 105   5
```

```
dim(test)
```

Tidy your data

**

What are tidy data ?

- **Population** = wholeness of the objects we'd like to study
- **Sample** = subset of the population
- **Variable** = feature of an observation
- **Observation** = series of measures on a population object

In a tidy dataset,

Each variable is in its own column. Each observation is in its own row

- If some data are available somewhere, it should be in the dataset
- All the data on observations must be in a single dataset
- Data must not be spread in several spreadsheets or workbook

When should I tidy my dataset ?

- Headers are levels of a variable
- Several variables are stored in one column
- Variables are stored in rows AND columns
- A same observation is stored in several tables
- ⋯(this is not an exhaustive list)⋯

`fill()` completes unappropriate missing values

id	Année	Mois
1	2014	Janvier
2		Février
3		Mars
4	2015	Janvier
5		Février
6		Mars

id	Année	Mois
1	2014	Janvier
2	2014	Février
3	2014	Mars
4	2015	Janvier
5	2015	Février
6	2015	Mars

```r
library(tidyr)
dataset <- data.frame(id = 1:6,
annee = c("2014",NA,NA,"2015",NA,NA))
dataset %>% fill(annee)
```

`separate()` separates a column in multiple others

id	tension
1	12/8
2	12/7
3	14/4
4	18/10
5	13/8
6	12/8

id	PAS	PAD
1	12	8
2	12	7
3	14	4
4	18	10
5	13	8
6	12	8

```r
dataset <- data.frame(id = 1:6, tension =
c("12/8","12/7","14/4","18/10","13/8","12/8"))
dataset %>%
separate(tension,into = c("PAS","DAS"),sep = "/",remove = TRUE)
```

`unite()` gathers several columns into one

id	PAS	PAD
1	12	8
2	12	7
3	14	4
4	18	10
5	13	8
6	12	8

id	tension
1	12/8
2	12/7
3	14/4
4	18/10
5	13/8
6	12/8

```r
dataset <- data.frame(id = 1:6,
                      PAS = c("12", "12", "14", "18", "13","12"),
                      DAS = c("8", "7", "4", "10", "8", "8"))
dataset %>% unite("tension",PAS,DAS,sep = "/")
```

`extract()` extracts informations in a column

id	var
1	Q1_2014
2	Q2_2014
3	Q3_2014
4	Q1_2015
5	Q2_2015
6	Q3_2015

id	année
1	2014
2	2014
3	2014
4	2015
5	2015
6	2015

```
dataset <- data.frame(id = 1:6,
    var = c("Q1_2014","Q2_2014","Q3_2014","Q1_2015","Q2_2015","Q3_2015"))
dataset %>% extract(var,into = "année",regex = ".+_([[0-9]]+)")
dataset %>% extract(var,into = c("quarter","year"),regex = "(.+)_([[0-
9]]+)")
```

`complete()` expands missing combinations of variables

id	année	mois	temp
1	2012	Jan	12
2	2012	Mar	11
3	2013	Feb	15
4	2014	Jan	16
5	2014	Feb	18
6	2014	Mar	10

id	année	mois	temp
	2012	Feb	
1	2012	Jan	12
2	2012	Mar	11
3	2013	Feb	15
	2013	Jan	
	2013	Mar	
4	2014	Jan	16
5	2014	Feb	18
6	2014	Mar	10

```r
dataset <- data.frame(id = 1:6,
annee = c("2012","2012","2013","2014","2014","2014"),
mois = month.abb[c(1,3,2,1,2,3)],temp = c(12,11,15,16,18,10))
dataset %>% complete(annee, mois)
```

Reshape your data

from long to large, from large to long

Tidy your data from "long" to "large"

Ville	Mois	Température
Amsterdam	01_Janvier	2.9
Amsterdam	02_Février	2.5
Amsterdam	03_Mars	5.7
Lisbonne	01_Janvier	10.5
Lisbonne	02_Février	11.3
Lisbonne	03_Mars	12.8

Ville	01_Janvier	02_Février	03_Mars
Amsterdam	2.9	2.5	5.7
Lisbonne	10.5	11.3	12.8

Tidy your data from "long" to "large"

```
temp_europ <- data.frame(
Ville = c(rep("Amsterdam",3),rep("Lisbonne",3)),
Mois = c(rep(c("Janvier", "Février", "Mars"),2)),
Temperature = c(2.9,2.5,5.7, 10.5,11.3, 12.8))

library(tidyr)
temp_europ %>%
  spread( key = Mois, value = Temperature )
```

Tidy your data from "large" to "long"

id	sexe	contrôle	traitement
1	F	7.9	12.3
2	M	6.3	10.6
3	M	9.5	13.1
4	F	11.5	13.4

id	sexe	condition	glycémie
1	F	contrôle	7.9
2	M	contrôle	6.3
3	M	contrôle	9.5
4	F	contrôle	11.5
1	F	traitement	12.3
2	M	traitement	10.6
3	M	traitement	13.1
4	F	traitement	13.4

Tidy your data from "large" to "long"

```
experience <- data.frame(id = c(1,2,3,4),  sexe = c("F","M","M","F"),
                         control = c(7.9,6.3,9.5,11.5),
                         treatment = c(12.3,10.6,13.1,13.4))
library(tidyr)
experience %>%
  gather( key = condition, value = glycemia, control:treatment )
```

Summary

Data wrangling with {dplyr}

Dataset

`data.frame` or `tibble` :

- Statistical unit in row
- Variables in colums

col1	col2	col3
1	"a"	"01-01-2018"
2	"b"	"02-01-2018"
3	"c"	"03-01-2018"

=> Warning: the *dplyr* grammar does not apply to all the objects existing in R.

For now :

- Data manipulation : all *tidyverse* functions
- Statistical analyses : base and *tidyverse* functions
- Handling of models results : *often* base functions (see {broom})

Dataset

`data.frame` or `tibble`:

col1	col2	col3
1	"a"	"01-01-2018"
2	"b"	"02-01-2018"
3	"c"	"03-01-2018"

- `x %>% filter(conditions)` filters the lines for which the conditions are `TRUE`
- `x %>% select(col1)` selects the column named "col1".
- `x %>% mutate(new = col1 * 2)` add a new column named "new".
- `x %>% group_by(col2) %>% summarise(n = n())` applies a function (*number of lines here*) for each group identified in column "col2".

Graphics with {ggplot2}

The ultimate graphic design tool

What is {ggplot2} ?

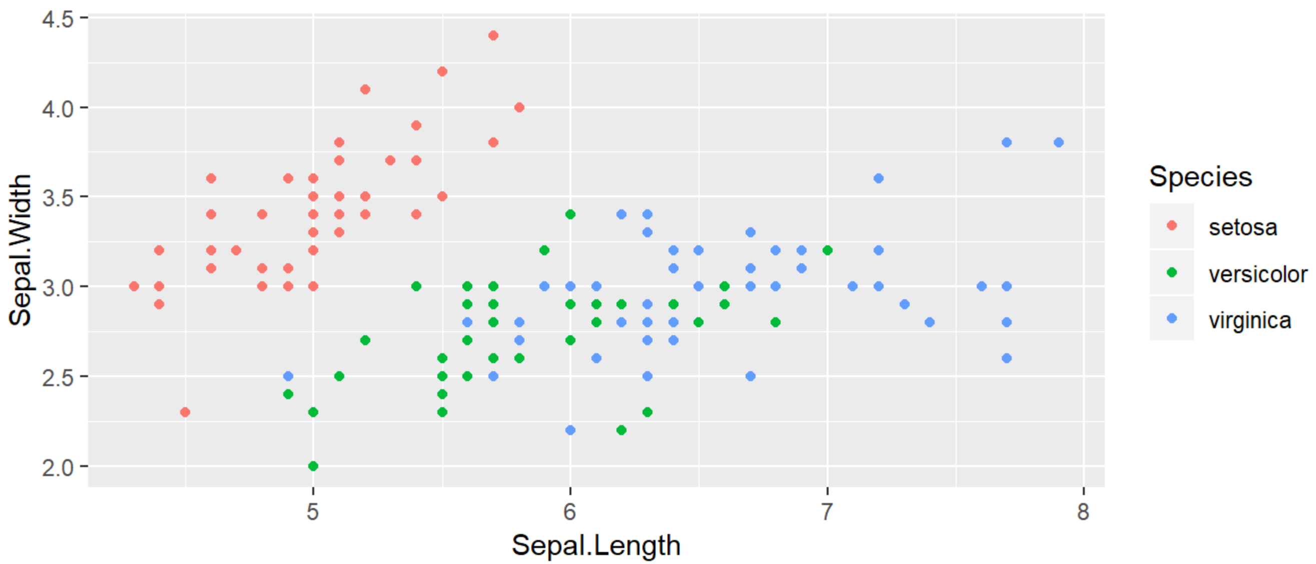
{ggplot2} is one of the most popular graphics package because:

- Code is compact.
- Rendering is pretty.
- Usage is intuitive.

How does it looks like ?

Example:

```
library(ggplot2)
ggplot( data = iris ) +
  aes( x = Sepal.Length, y = Sepal.Width, color = Species ) +
  geom_point()
```



How to build a ggplot ?

In {ggplot2}, graphics are layers, superimposed using `+` :

- data
- aesthetics
- geometric layer: geom_XXX (ex : `geom_point()` ; `geom_line()` ...)
- statistics layer : stat_XXX (ex : `stat_smooth()` ...)
- scale layer : scale_XXX (ex : `scale_x_log10()` ...)
- facetting layer : facet_XXX (ex : `facet_grid()` ; `facet_wrap()` ...)
- coordinate layer : coord_XXX (ex : `coord_flip()` ...)
- theme layer : theme_ (ex : `theme_mininal()` ...)

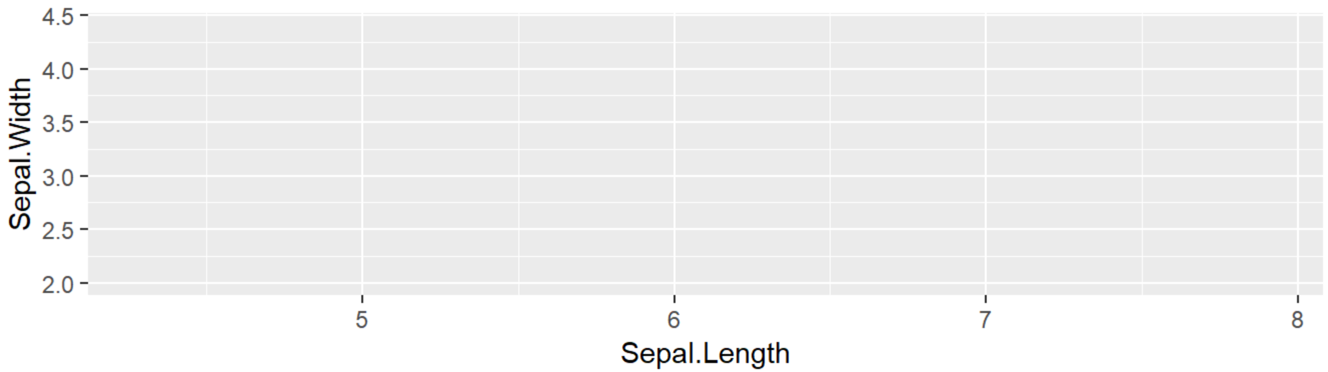
The ggplot2 paradigm :

The `ggplot()` function

`ggplot` function may include 2 parameters:

- `data` : the dataset containing data to plot (a dataframe)
- aesthetic mapping (describing how variables in the `data` are mapped to visual properties, and must be includes in `aes()`).

```
p <- ggplot(data = iris) +
  aes(x = Sepal.Length, y = Sepal.Width, color = Species)
  p
```



`p` doesn't draw data: layers must be added !

geom_*()

`geom` are layers that materialize the plot. Some common geoms:

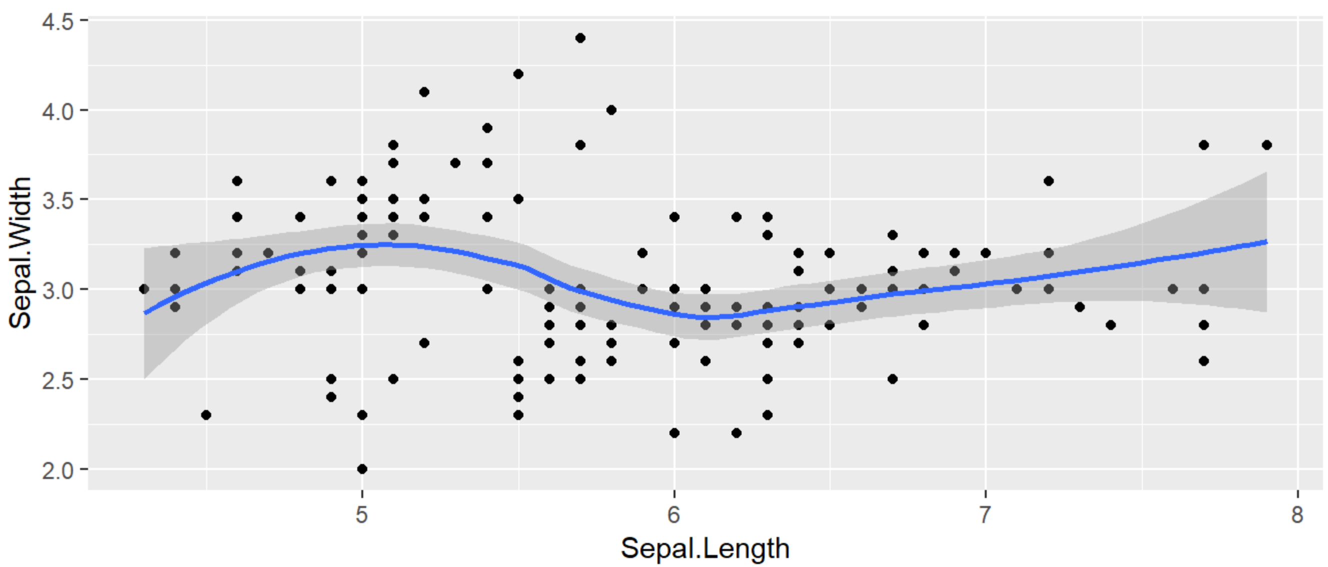
- "histogram" : histogram (default if 1 dimension)
- "point" : scatterplot (default if 2 dimensions)
- "smooth" : adds a smoothed density estimates
- "boxplot"
- "freqpoly" : frequency polygon
- "density"
- "bar" : barcharts, counts frenquencies of x
- "col" : barcharts, frenquencies are provided by the user on y axis
- "text" : adds some text

Link to available geoms

geom()

Feel free to combine multiple `geom_*`:

```
library(ggplot2)
ggplot(data = iris) +
  aes(x = Sepal.Length, y = Sepal.Width) +
  geom_point() +
  geom_smooth()
```

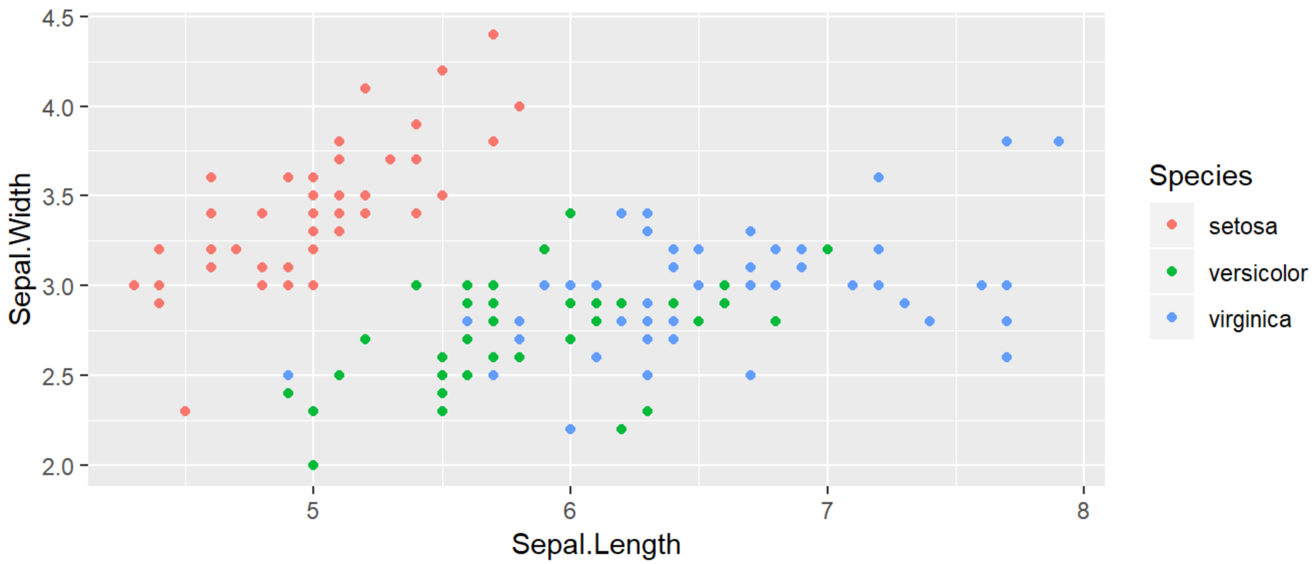


Taking into account categories...

...with colors:

- `color` argument in `aes()`

```
ggplot(data = iris) +
  aes(x = Sepal.Length, y = Sepal.Width, color = Species) +
  geom_point()
```

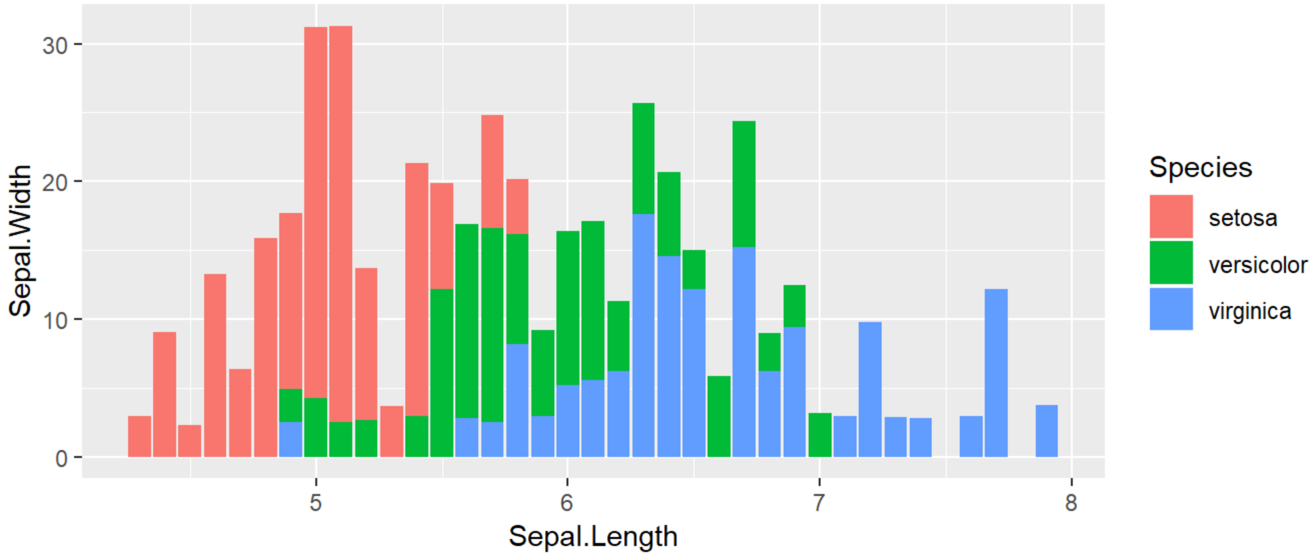


Taking into account categories...

...with colors:

- `fill` argument in `aes()`

```
ggplot(data = iris,
       aes(x = Sepal.Length, y = Sepal.Width, fill = Species)) +
  geom_col()
```

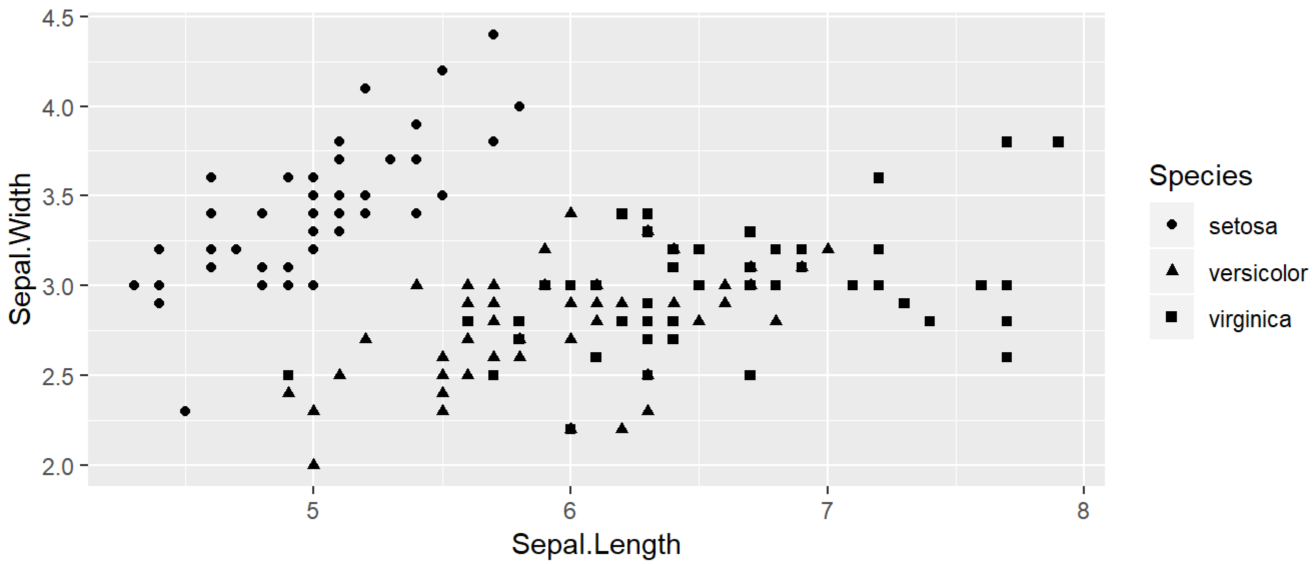


Taking into account categories...

...with shapes:

- `shape` argument in `aes()`

```
ggplot(data = iris) +
  aes(x = Sepal.Length, y = Sepal.Width, shape = Species) +
  geom_point()
```

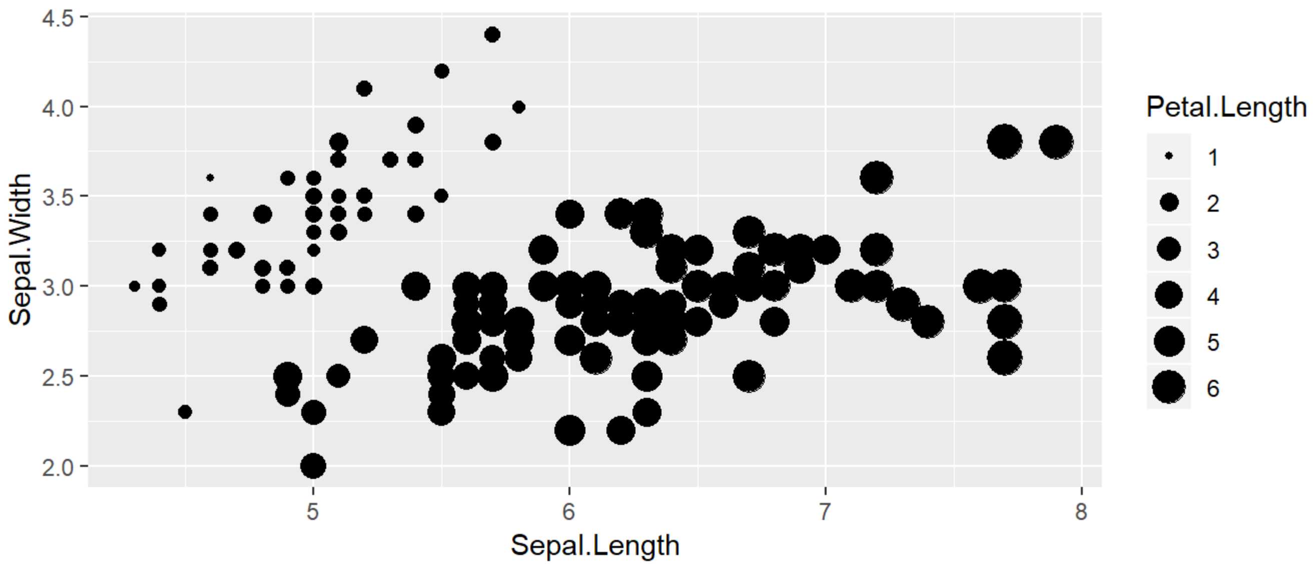


Taking into account categories...

...with size:

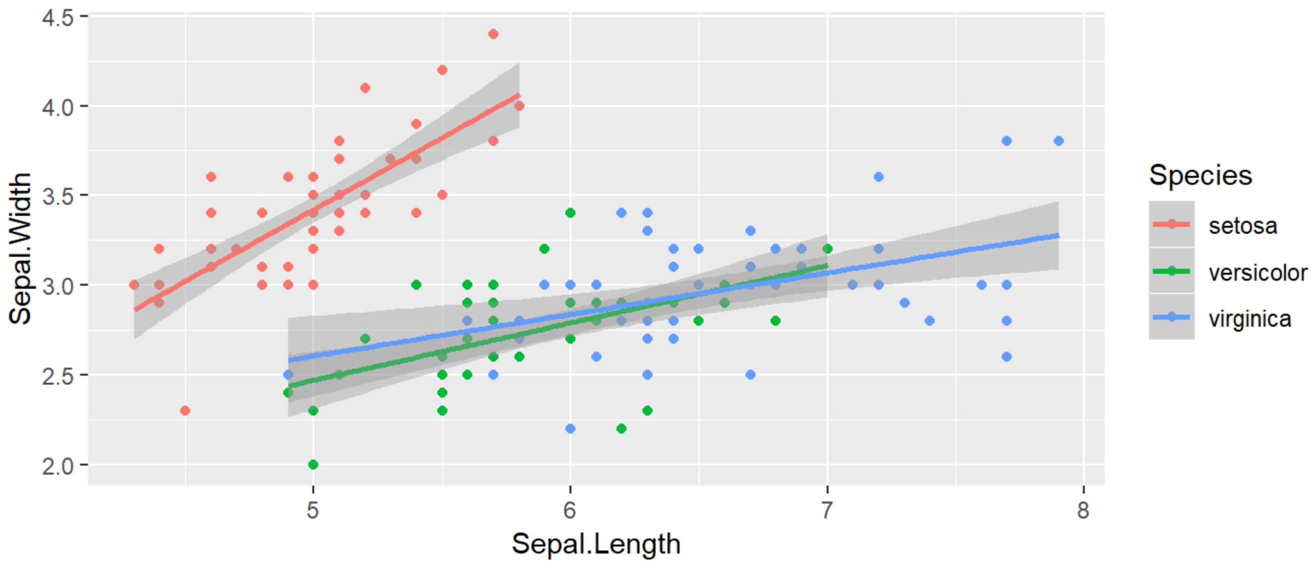
- `size` argument in `aes()`

```
ggplot(data = iris) +
  aes(x = Sepal.Length, y = Sepal.Width, size = Petal.Length) +
  geom_point()
```



Statistical smoothing

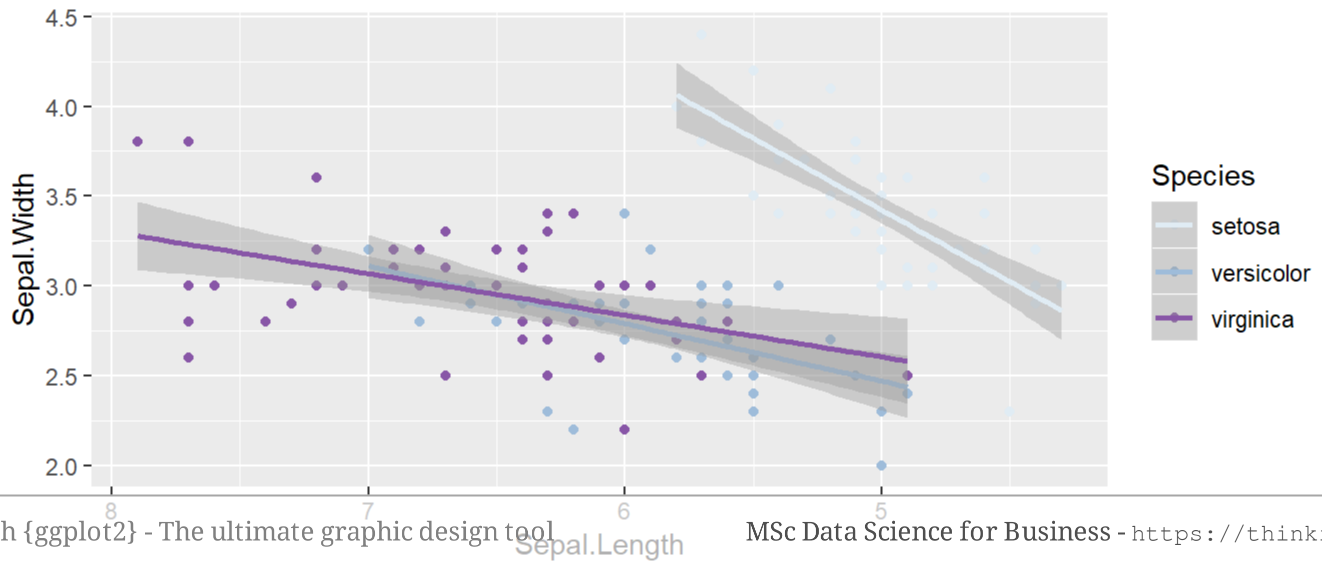
```
ggplot(data = iris) +
  aes(x = Sepal.Length, y = Sepal.Width, color = Species) +
  geom_point() +
  geom_smooth(method = "lm")
```



Scale layer

This layer operates transformations on the scales of the graphics: `scale_x_*`, `scale_y_*`, `scale_fill_*`, `scale_color_*`...

```r
ggplot(data = iris) +
  aes(x = Sepal.Length, y = Sepal.Width, color = Species) +
  geom_point() +
  geom_smooth(method = "lm")  +
  scale_colour_brewer(palette = 3) +
  scale_x_reverse()
```



Facetting

This layer "slices" the plot into several plots. Two functions will do the job:

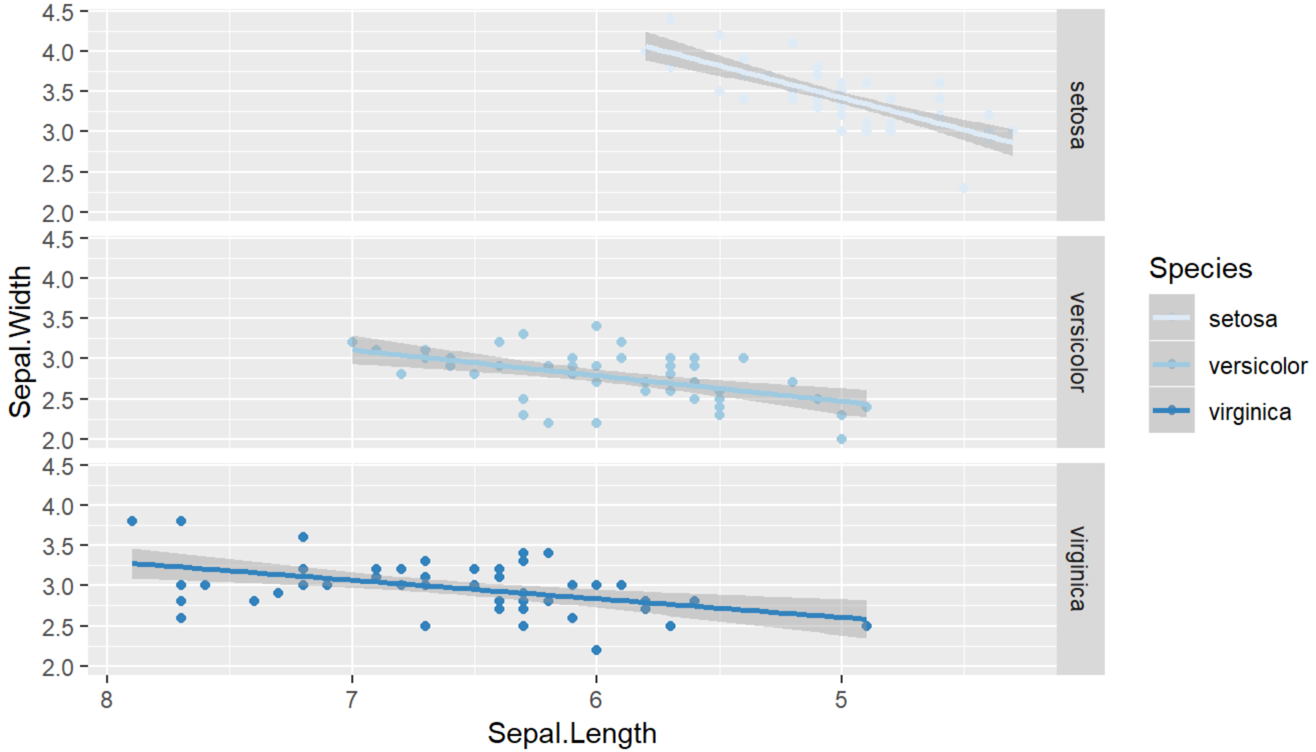
- `facet_grid` : forms a matrix of panels defined by row and column
- `facet_wrap` : wraps a 1d sequence of panels into 2d

```
ggplot(data = iris) +
  aes(x = Sepal.Length, y = Sepal.Width, color = Species) +
  geom_point() +
  geom_smooth(method = "lm")  +
  scale_colour_brewer(palette = 4) +
  scale_x_reverse() +
  facet_grid(Species ~ .)
```

Facetting

This layer "slices" the plot into several plots. 2 functions will do the job:

- `facet_grid` : forms a matrix of panels defined by row and column
- `facet_wrap` : wraps a 1d sequence of panels into 2d



Coordinates

This layer manages the coordinates of the plot

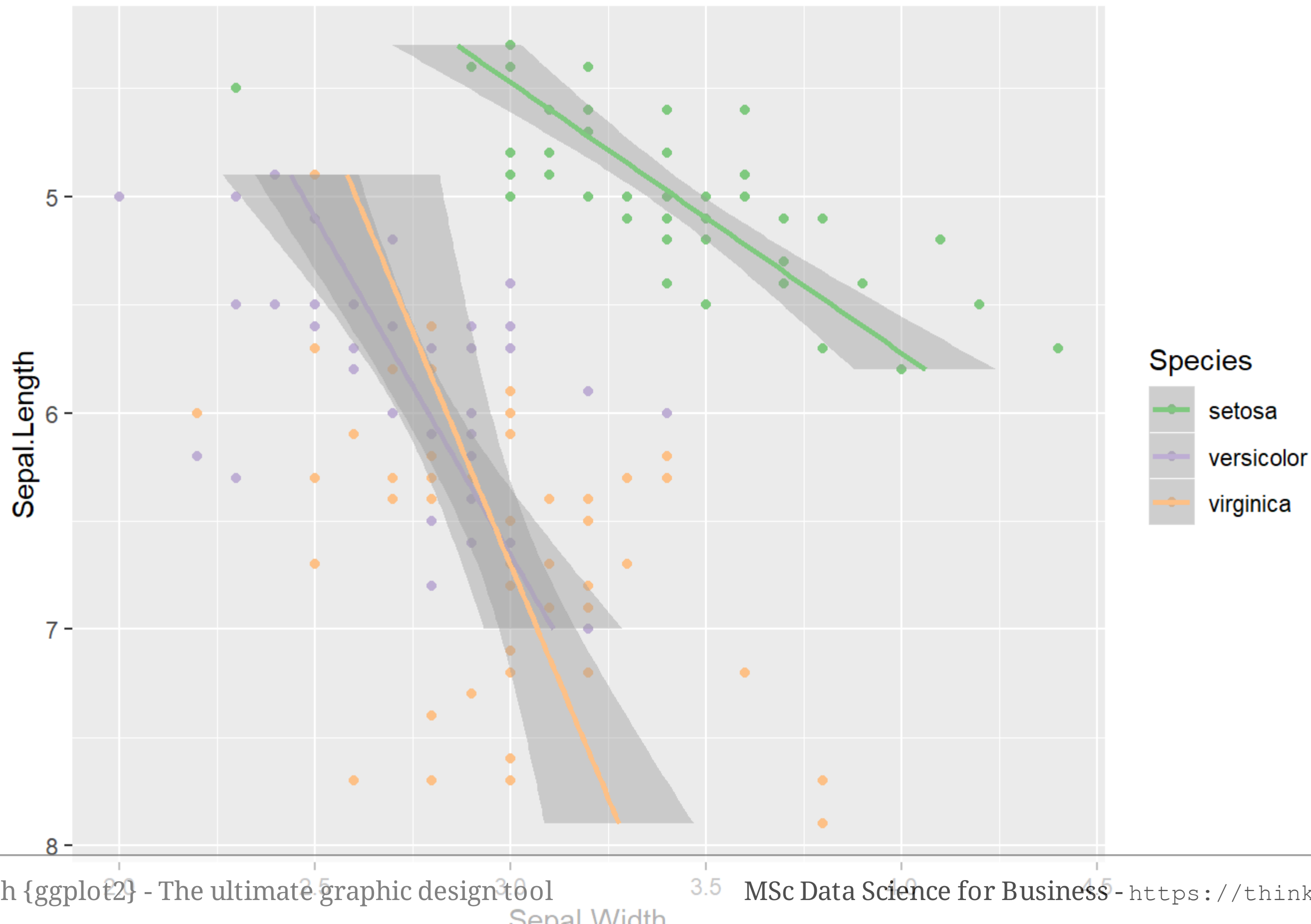
- `coord_flip` : switches x and y.

```
ggplot(data = iris) +
  aes(x = Sepal.Length, y = Sepal.Width, color = Species) +
  geom_point() +
  geom_smooth(method = "lm")  +
  scale_colour_brewer() +
  scale_x_reverse() +
  facet_grid(Species ~ .) +
  coord_flip()
```

Coordinates

This layer manages the coordinates of the plot.

- `coord_flip` : switches x and y.



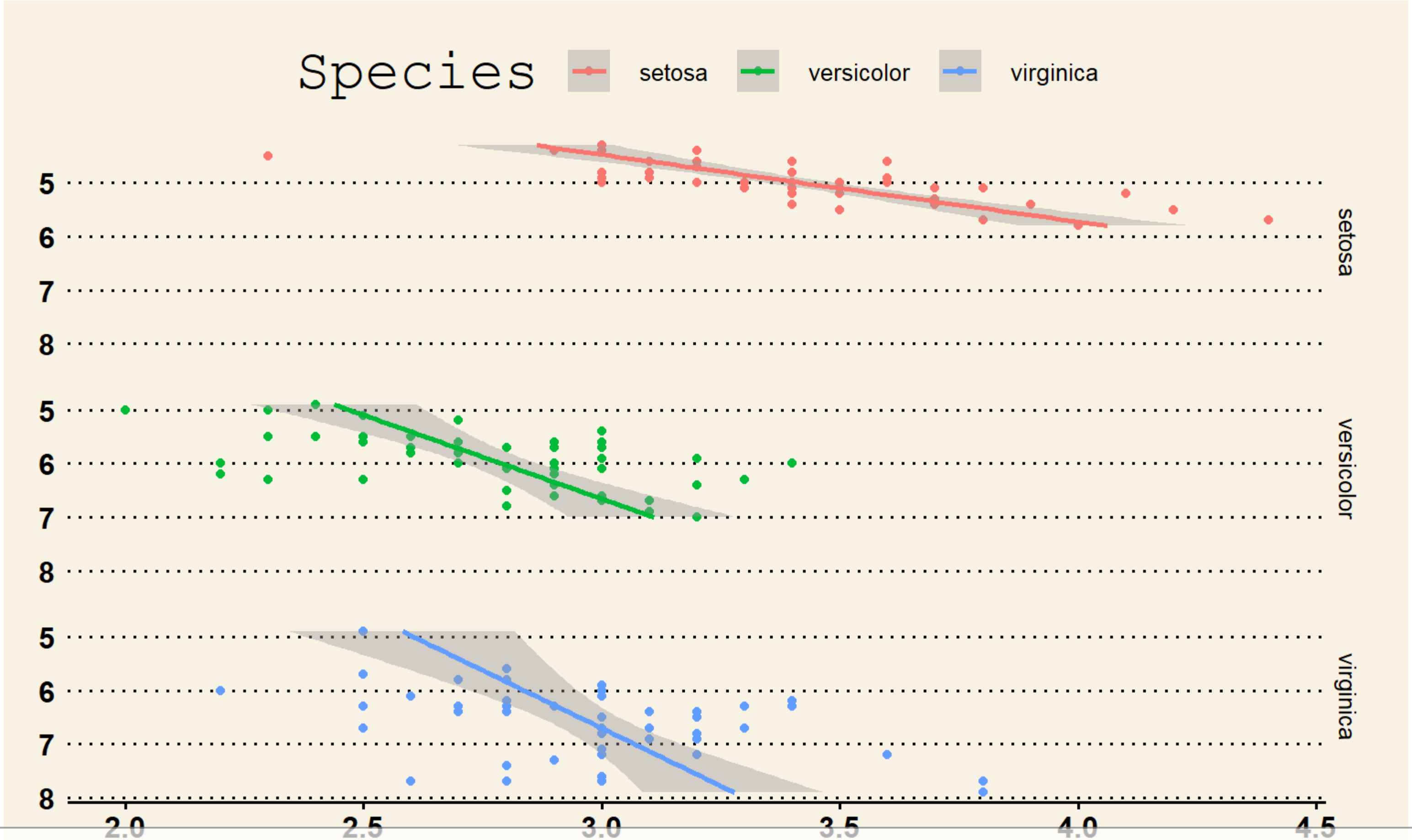
Theme

Manages the cosmetic aspect of the `ggplot`. You can design yours or use `{ggtheme}` one's:

```
library(ggthemes)
ggplot(data = iris) +
  aes(x = Sepal.Length, y = Sepal.Width, color = Species) +
  geom_point() +
  geom_smooth(method = "lm")  +
  scale_x_reverse() +
  facet_grid(Species ~ .) +
  coord_flip() +
  theme_wsj()
```

Theme

Themes manage the cosmetic aspect of the `ggplot`. You can design yours or use one of `{ggtheme}`'s:



Save graphs from {ggplot2}

```
ggsave( filename = "mongraph.png", dpi = 96 ) # dpi is the resolution. Default to 300.
```

...and find some help :

- http://docs.ggplot2.org/current/index.html
- http://docs.ggplot2.org/current/index.html
- https://stackoverflow.com/questions/tagged/ggplot2

Going further

Numerous extensions to ggplot exists :

- {ggiraph}, {gganimate} : dynamic graph
- {ggTimeSeries} : to plot time series
- http://www.ggplot2-exts.org/

Diane Beldame

diane@thinkr.fr

06 23 83 10 61