R basics and the tidyverse

Session 1 LOVE'R course

Pablo Raguet

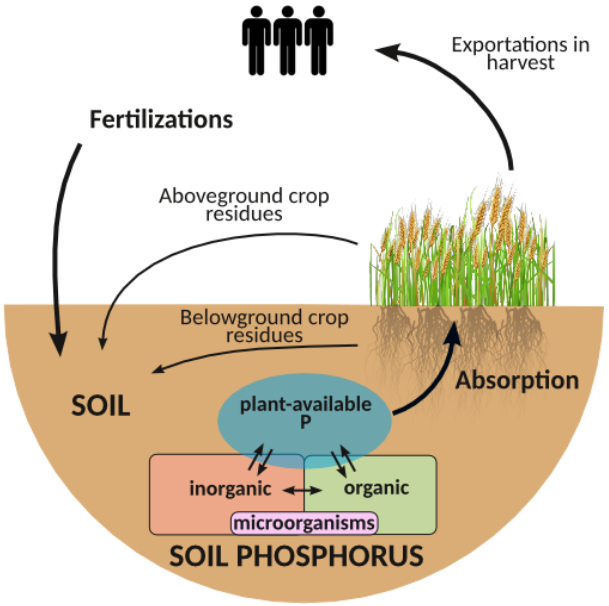
email: pablo.raguet@inrae.fr
github: Capra-Ibex/R-course-2022

original teacher: Tania L. Maxwell
website: tania-maxwell.github.io

07-11-2022

Who am I?

- Final year PhD student at INRAE, finishing a double degree program between the Université de Bordeaux and Université Laval in Québec
- Studying the dynamic and modelling of organic phosphorus in cropped soil under various field conditions.
- R enthusiast, coder



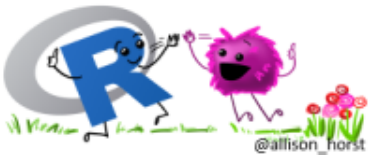
@allison_horst

R Course Objectives

- Understand the basis of the R language to be able to write your own scripts
- Optimize your dataset table (7/11)
- Choose a statistical model according to your scientific question (10/11 & 14/11)
- Create reproducible graphics (17/11)
- Know how to search for resources related to your question or code problem in R

How this course works

- This is an interactive course, which means you will be coding throughout the course
- When you see this symbol, you use code to answer a question that I've stated:



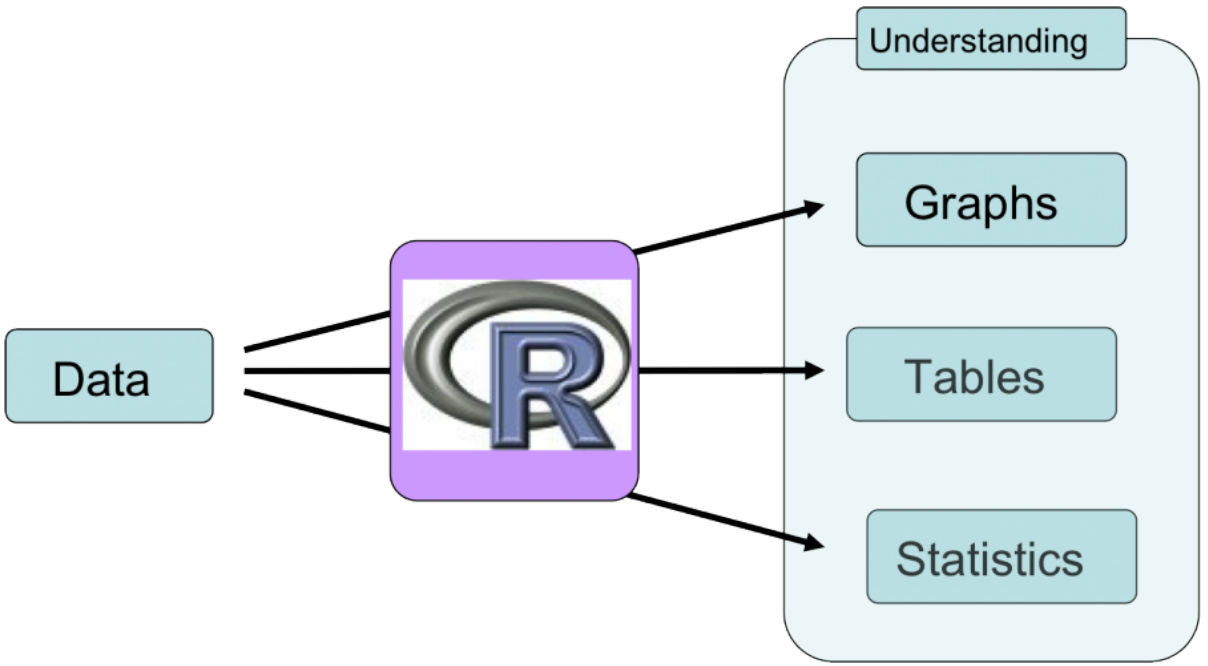
- Don't hesitate to ask questions
- There will be a final project to be done in groups of 2, with a presentation on November 24th (more details will be given at the end of this course).

R basics

The multiple facets of R

R is a programming language, and RStudio is a graphical interface to help users with R

R is open source, which means there are plenty of great solutions to issues that can be found online



@CSBQ

Let's get started

Step 1: Create a new folder for the R course

Step 2: Open Rstudio, click on:

File → New File → R Script

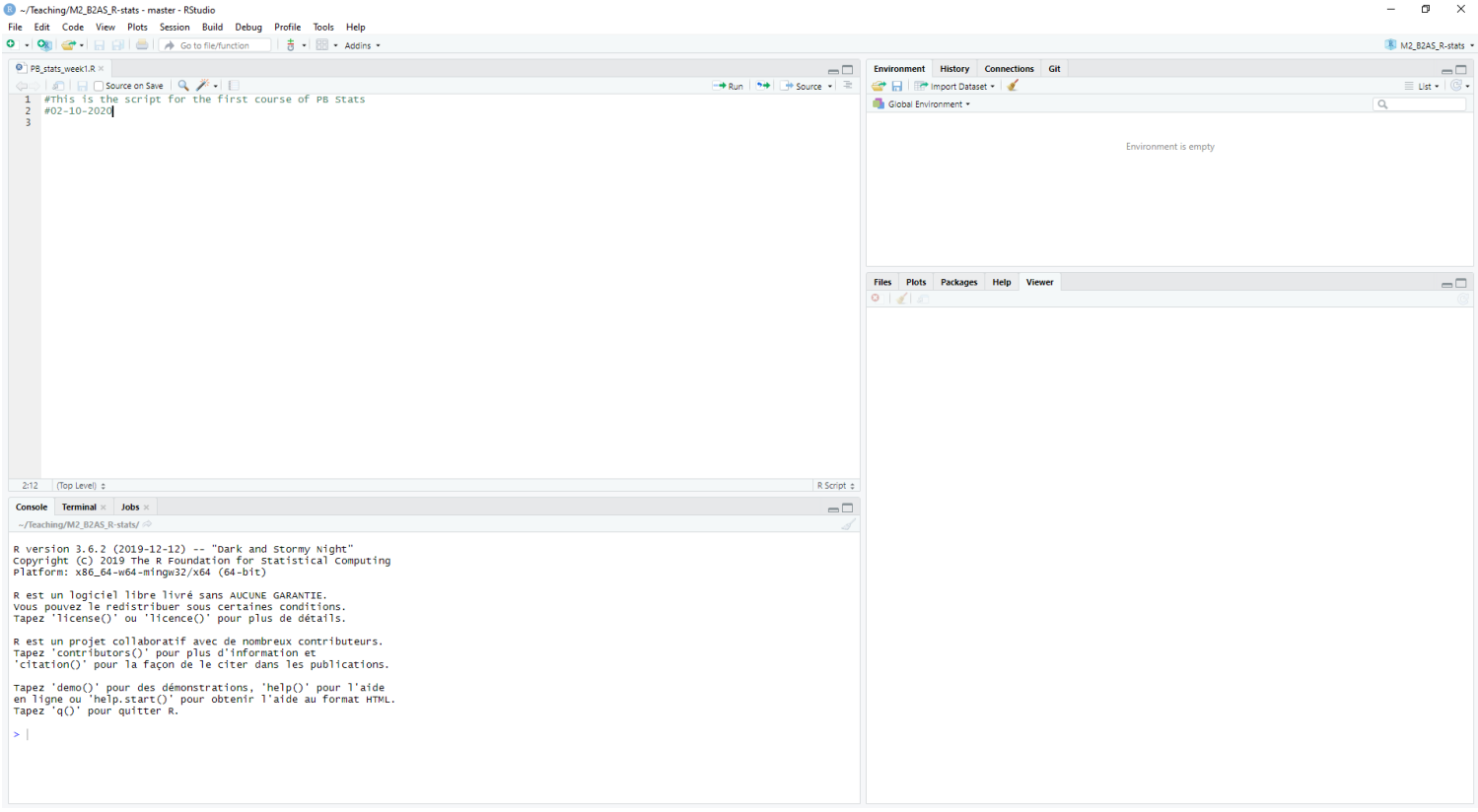
Step 3: Save the R Script with an easy-to-find name in the new-created folder

Good Practices

- Comment your code by using the # symbol
- Write in your script, and then click **run** to evaluate the code in the console

Tip: You can also press "Ctrl" and "Enter" to run your code

Explore R Studio



Tip: you can change the colors of RStudio in

Tools → Global Options → Appearance

Working directory

In RStudio

We use the `setwd()` function so that R knows where to look for the data table. We can use `getwd()` to see in which directory we currently are.

```
# getwd()
```

Tip: manually set your directory by clicking

Session → Set Working Directory → Choose Directory

Copy-paste the output from the R console into your script so that you only need to do it once.

```
# setwd("/directory/of/your/R-script/from/root")
```

Saved by the R project

Go up-left or up-right → Create new project → Choose Directory

Advantages:

- All your scripts, data, table and figure in one folder
- Relative path for working directory
- Code reproducibility
- Allow to use Git for code versioning (advanced use)

Give it a try:

```
getwd()
```

```
## [1] "C:/Users/praguet/Documents/THESE/Teachings/R-course-2022-prof"
```

Renv the package manager

Within a open R project:

Tools → Project Options → Environments → Tick *use renv*

Advantages:

- Local save of packages and their versions
- Limit code break due to package update

Tips:

Some renv functions (call: `renv::function()`):

Function	effects	Function	effects
activate()	activate renv	upgrade()	upgrade renv version
install()	install a package	update()	update packages
snapshot()	save packages versions	restore()	restore packages version

Structure your R project

Arrange your R scripts and folders:

- Name your scripts starting by a number (example: 01_name.R)

Tells you which script to run first !

- Examples of folders "Data", "Export", "Export/Figure"

Structure your code:

- Header
- Working directory check
- Loading packages (next slide)
- Data import (further slide)
- Divide your code in parts

Tips (dispensable):

- create a separate .R script for functions (`source("name_of_fun_file.R")`)
- Use the snippets for automatic code: copy/paste the Snippet.txt in

Tools → Edit Code Snippets

Loading the necessary package(s)

If you have not yet installed the metapackage `tidyverse`, you can do by clicking in the bottom right pannel:

Packages → Install, and typing the package name

You can also run it in your script without the `#`:

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

Or

```
# renv::install("tidyverse")
```

You need to tell R where to look in its library for the packages you will use in your script.

```
# library(tidyverse)
```

Tip: At the beginning of each script, only include the libraries which you will actually use.

R as a calculator

Type the following in your script and execute your code:

5*9-2+4

```
5*9-2+4
```

```
## [1] 47
```

What does the [1] mean ?

Hint: type `1:50` into your script

```
1:50
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```

The [1] is a way to **index** the output = to locate the place in the string of numbers

Objects in R

- You can save values as objects

```
var1 <- 5*9
var2 <- 2+3
```

- Objects are stored in your Global Environment "Values"

Create a new variable which is the multiplication of var1 and var2

```
var3 <- var1*var2
```

- What is the value of var3?

```
var3
```

```
## [1] 225
```

Arithmetic operators in R

Character		Character	
+	addition	/	division
-	substraction	%/%	integer division
^	exponent	%%	modulus
*	multiplication		

Examples of different operators:

```
10/3 ; 10%/%3 ; 10%%3
```

```
## [1] 3.333333
```

```
## [1] 3
```

```
## [1] 1
```

Tips for naming objects

- The first character should be a letter, i.e. var1 and not 1var
- The names should only include the following: letters, numbers, _ and .
- Avoid using accents in your code, as it will may have troubles encripting (which is why I recommend to code in English)
- Don't use special keywords as names of your object (i.e. break, function, if, etc.). They will highlight when you type it in R
- Choose names which add meaning to your code (because it is likely that in 3 months you will have to decipher your code)
- Use relatively short names (recommended)

Functions

A function is an argument with an input and an output

The arguments (what goes into the parentheses `()`) are objects (i.e. var1, var2, data frames, etc.)

```
var4 <- sum(var1,var2,var3)
var4
```

```
## [1] 275
```

Functions have arguments that need a certain order (to be detailled later)

Some useful arithmetic function:

Function	*single results*	Function	*applied to each element of a vector*
sum()	addition	exp()	exponential
mean()	average	log()	logarythm
prod()	multiplication	sqrt()	square root
sd()	standard deviation	?	standard error

Advanced used of functions in R

Build your own function (1)

Create an object with the `function()` function. Example for coefficient of variation:

```
var5 <- c(2, 6, 1, 8)
```

```
Fun.cv <- function(x) {
  cv <- sd(x) / mean(x)
  return(cv)
}
Fun.cv(var5)
```

```
## [1] 0.7774207
```

Useful functions to build your own function:

Function	Effect	Function	Effect
print(x)	print the object `x`	stop(m)	stop the function and print the message `m`
return(x)	return the object `x` at the end of the function	warning(m)	print a warning with the message `m`

Build your own function (2)

The standard error function does not exist in R, how would you build that function ?

Use:

```
var5 <- c(2, 6, 1, 8) ; var5
```

```
## [1] 2 6 1 8
```

Tips:

$$se = \frac{\sigma}{\sqrt{n}}$$

With σ the vector standard deviation and `n` the `length()` of the vector.

Build your own function: solution(s)

- Simple solution

```
Fun.se.1 <- function(x){sd(x)/sqrt(length(x))}
```

```
Fun.se.1(var5)
```

```
## [1] 1.652019
```

- Complex solution

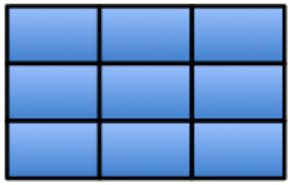
```
Fun.se.2 <- function(x){
  a <- (sd(x)/sqrt(length(x)))
  return(a)
}
```

```
Fun.se.2(var5)
```

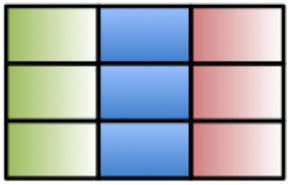
```
## [1] 1.652019
```

Data structures in R

Vector

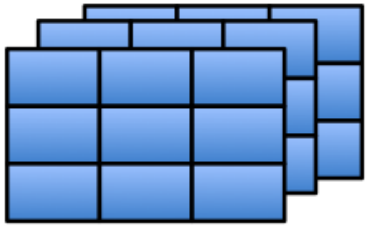


Matrix



Data frame

*Columns can be different modes



Array

List

- Vector
- Matrix
- Data frame
- Array

Source: R in Action, p.23

Let's create a data frame from scratch

Step 1: Create vectors

```
#Character vectors
plant_sp <- c("tomato", "tomato", "tomato", "tomato")

treatment <- c("stress", "stress", "control", "control")
```

```
#Numerical vector
diameter <- c(2.3, 3.4, 4.3, 4.8)
```

Step 2: Combine the vectors with the `data.frame()` function

```
plant_data <- data.frame(plant_sp, treatment, diameter) ; plant_data
```

```
##   plant_sp treatment diameter
## 1   tomato    stress      2.3
## 2   tomato    stress      3.4
## 3   tomato   control      4.3
## 4   tomato   control      4.8
```

Indexing

data_frame_name[`row`, `column`]

matrix[`row`, `column`]

vector[`element`]

list[[`element`]]

Example:

`plant_data[1, ]` → Extracts the first row

Note: empty index selects all the values

`plant_data[ ,3]` → Extracts the third column

Extract the second value from the third column

Solution

```
plant_data[2,3]
```

```
## [1] 3.4
```

Indexing

You can use this to easily remove columns or rows from your data frame.

Examples:

```
#remove column 1
plant_data_reduced1 <- plant_data[ , -1]
plant_data_reduced1
```

```
##   treatment diameter
## 1    stress      2.3
## 2    stress      3.4
## 3   control      4.3
## 4   control      4.8
```

```
#keep only row 3 and 4
plant_data_reduced2 <- plant_data[c(3,4), ]
plant_data_reduced2
```

```
##   plant_sp treatment diameter
## 3   tomato   control      4.3
## 4   tomato   control      4.8
```

Your turn

Create a new data frame with just rows 1 and 2, and columns 2 and 3 of the `plant_data` data frame

Solution (different ways to do this)

```
#option 1
reduced_option1<- plant_data[c(1,2), c(2,3)]
reduced_option1
```

```
##   treatment diameter
## 1    stress      2.3
## 2    stress      3.4
```

```
#option 2
reduced_option2<- plant_data[c(1,2), -1]
reduced_option2
```

```
##   treatment diameter
## 1    stress      2.3
## 2    stress      3.4
```

```
#etc...
```

Where to find help

R help and documentation:

- press `F1` with the cursor on the function
- execute `?function`

Online!

- Stackoverflow
- Cross-validated
- Google

"R how to ..." is my most frequent google search - and you almost always find an answer to your question.

Tidy data: using the `tidyverse` metapackage

How do we set up a data table?

Example: You are setting up an experiment with four plants of corn, and you are measuring the height (cm) at five different time points (T1, T2, T3, T4, T5) .

On a piece of paper, write down a draft of your data table

"Intuitive" data table

Plant	Height_T1_cm	Height_T2_cm	Height_T3_cm	Height_T4_cm	Height_T5_cm
CornA	10	12	15	25	30
CornB	9.5	11	...	...	...
CornC					
CornD					

More effective data table for R

Plant	Time	Height_cm
CornA	T1	10
CornA	T2	12
CornA	T3	15
CornA	T4	25
CornA	T5	30
CornB	T1	9.5
CornB	T2	11
...	...	...

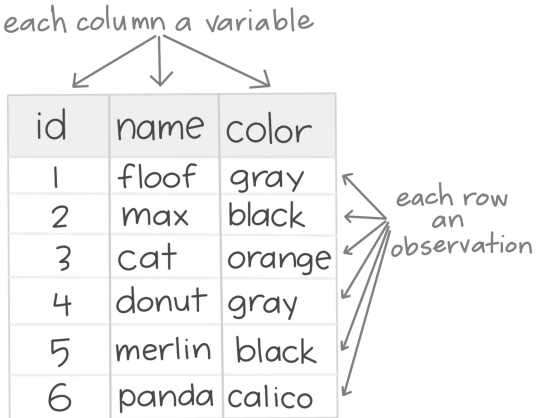
This is what we call 'tidy data' or 'long format'.

Introduction to 'tidy data'

> "TIDY DATA is a standard way of mapping the meaning of a dataset to its structure."
>
> –HADLEY WICKHAM

In tidy data:

- each variable forms a column
- each observation forms a row
- each cell is a single measurement



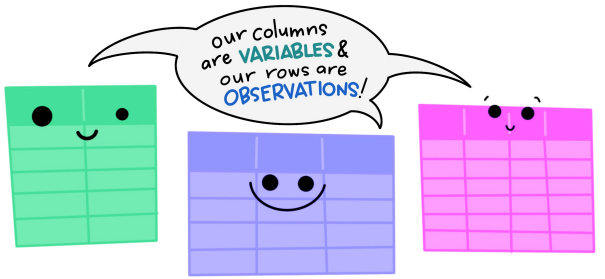
Wickham, H. (2014). Tidy Data. Journal of Statistical Software *59* (10). DOI: 10.18637/jss.v059.i10

@allison_horst

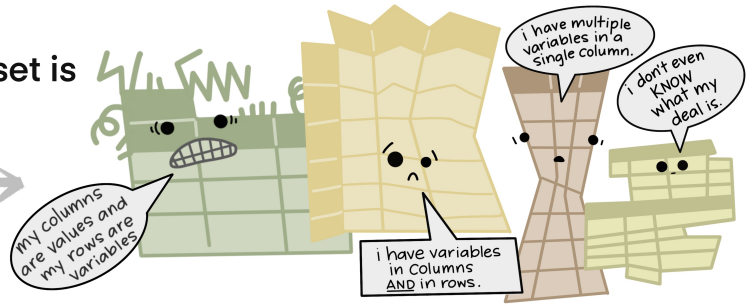
Why use 'tidy data'?

- Most efficient way for R to read your data
- Easier for automation and iteration
- Moving from Excel to R for all calculations : reproducible

The standard structure of tidy data means that "tidy datasets are all alike..."



"...but every messy dataset is messy in its own way."
–HADLEY WICKHAM



Importing a data file

If data is under Excel file format, save as a .csv or a .txt file.

We use the `read.table()` function. The "sep" (separator) will likely be different whether you are importing a .csv or a .txt file.

```
data_co2 <- read.table("Data/Exemple/CO2_data_fromR.txt", header = TRUE, sep = "\t
```

- The dataset shows results of an experiment on the cold tolerance of grass.
- Grass samples from two regions (Quebec and Mississippi) were grown in either a chilled or nonchilled environment
- Their CO_2 uptake rate was tested at different CO_2 concentrations.

Exploring the data

Use the `str()` function to look at the structure of your data table.

What do you see?

```
str(data_co2)
```

```
## 'data.frame':    84 obs. of  5 variables:
##  $ Plant    : chr  "Qn1" "Qn1" "Qn1" "Qn1" ...
##  $ Type     : chr  "Quebec" "Quebec" "Quebec" "Quebec" ...
##  $ Treatment: chr  "nonchilled" "nonchilled" "nonchilled" "nonchilled" ...
##  $ conc     : int  95 175 250 350 500 675 1000 95 175 250 ...
##  $ uptake   : num  16 30.4 34.8 37.2 35.3 39.2 39.7 13.6 27.3 37.1 ...
```

There are four main data type:

- `int` = integer (whole numbers)
- `num` = numeric
- `chr` = character, alphabetical characters
- `Factor` = for both alphabetical and numerical characters, **can be placed in a certain order**

Selecting a column

The `$` symbol selects a column. But there are other ways to do this:

`data_co2$ID`,

`data_co2[ , "ID" ]`,

`data_co2[ , 1]` (because ID is the first column)

are **all the same**

Changing data types for a column

Let's say we want to make the ID number a **factor**, we can use the `as.factor()` function

- Note: this often happens when numbers are used to identify different levels in a column, but don't represent the number value

```
# testing the as.factor() function
# data_co2$ID <- as.factor(data_co2$ID)
```

Similarly, you can make a factor as a numeric value with:

`as.numeric()`, `as.character()`, `as.integer()`, etc.

More useful functions

- the `table()` and `levels()` are useful for numeric and factor columns, respectively

```
table(data_co2$conc)
```

```
##
##   95  175  250  350  500  675 1000
##   12   12   12   12   12   12   12
```

```
levels(data_co2$Treatment)
```

```
## NULL
```

- The `table(data_co2$conc)` gives us the number of observations for each value of `conc` = concentration.
- The `levels(data_co2$Treatment)` gives us all the levels in the `Treatment` column.

Data wrangling with `dplyr`

Let's say that you want to find the mean per group in your experiment.

How would you do this previously?

Because this data table is 'tidy', we can use `dplyr` for such data manipulations and calculations.

Important Concept : "piping" with %>%

```
New Data Table <- Data Table %>%
                  function1  %>%
                  function 2
                  etc...
```

This reads:

"Make a new data table, where you take my data table, apply function 1 to it, and then apply function 2 to the new subset made by function 1 "

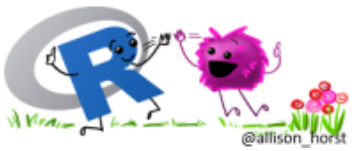
Example: means per group

```
data_co2_means<- data_co2 %>%
  group_by(Treatment) %>%
  summarise(mean = mean(uptake))

data_co2_means
```

```
## # A tibble: 2 × 2
##   Treatment   mean
##   <chr>      <dbl>
## 1 chilled     23.8
## 2 nonchilled  30.6
```

- `group_by()` is the first function
- `summarise()` is the second function, applied to each Treatment.
- `mean =` names the column in which the `mean(uptake)` is written



Make a new data table in which you calculate the mean of the CO2 uptake per plant Type (Quebec vs. Mississippi)

Solution

```
data_co2_means_type<- data_co2 %>%
  group_by(Type) %>%
  summarise(mean = mean(uptake))

data_co2_means_type
```

```
## # A tibble: 2 × 2
##   Type         mean
##   <chr>       <dbl>
## 1 Mississippi  20.9
## 2 Quebec       33.5
```

Challenge: means, se, n per group

Make a new data table in which you calculate the number of variables per observation (n), mean, and standard error of uptake per plant Type (Quebec vs. Mississippi)

Hint: use the `summarise_at()` and the `funs()` function:

summarise_at(c("uptake"), list(N = ~n(), Mean = ~mean(.), SE = ~(sd(.)/sqrt(n()))))

```
data_co2_means_se <- data_co2 %>%
  group_by(Type) %>%
  summarise_at(c("uptake"),
               list(N = ~n(),
                    Mean = ~mean(.),
                    SE = ~(sd(.)/sqrt(n()))))
data_co2_means_se
```

```
## # A tibble: 2 × 4
##   Type            N  Mean    SE
##   <chr>       <int> <dbl> <dbl>
## 1 Mississippi    42  20.9  1.21
## 2 Quebec         42  33.5  1.49
```

Terminology

`dplyr::` means we are using a function from the dplyr package without loading all the package.

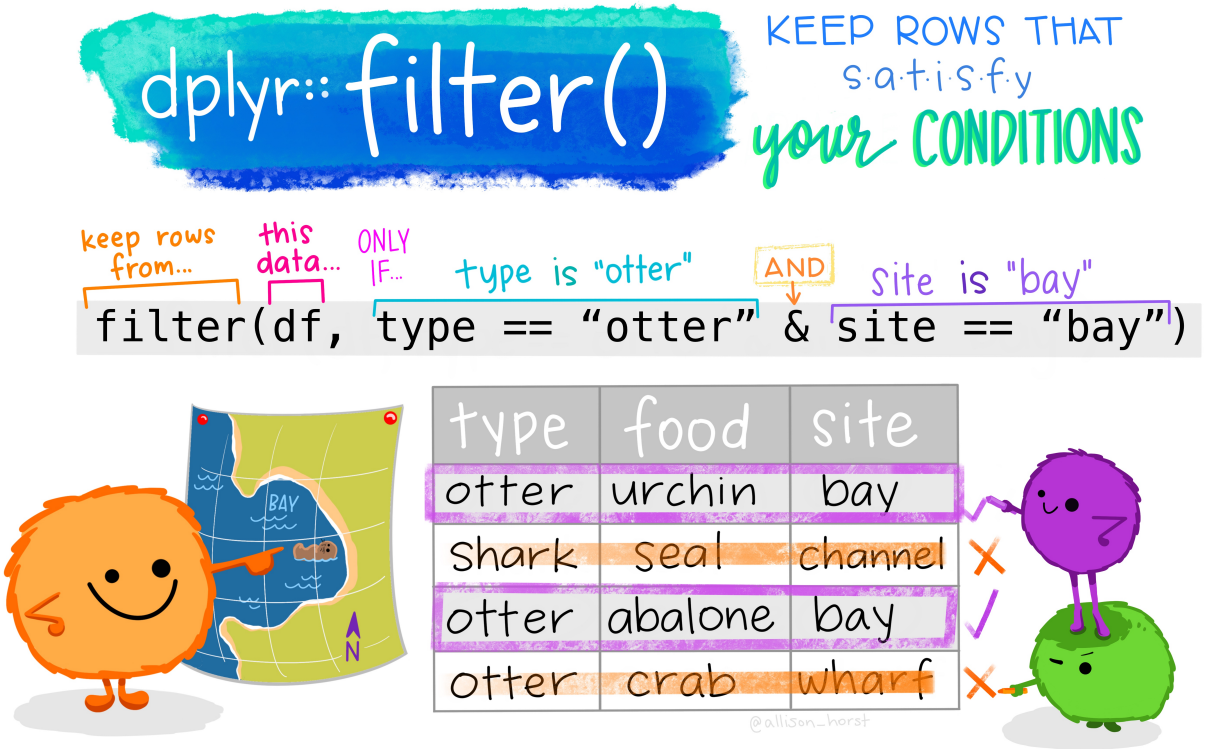
x %>% f(y)is the same as f(x, y),

```
 data_co2 %>%
  group_by(Type)
```

```
is equal to
```

```
dplyr::group_by(data_co2, Type)
```

Filtering a data table



@allison_horst

Your turn

Using the previous tutorial, filter the data table for the nonchilled treatment in Quebec?

Hint:

```
new_df <- dplyr:: filter(df, ...)  is the same as
new_df<- df %>%
  filter(...)
```

Solution

```
quebec_nonchilled<- data_co2 %>%
  filter(Type == "Quebec" & Treatment == "nonchilled")
head(quebec_nonchilled )
```

```
##   Plant   Type  Treatment conc uptake
## 1   Qn1 Quebec nonchilled   95   16.0
## 2   Qn1 Quebec nonchilled  175   30.4
## 3   Qn1 Quebec nonchilled  250   34.8
## 4   Qn1 Quebec nonchilled  350   37.2
## 5   Qn1 Quebec nonchilled  500   35.3
## 6   Qn1 Quebec nonchilled  675   39.2
```

And what if we want to directly find the n, mean, and standard error?

```
quebec_nonchilled_mean<- data_co2 %>%
  filter(Type == "Quebec" & Treatment == "nonchilled") %>%
  summarise_at(c("uptake"),
               list(N = ~n(),
                    Mean = ~mean(.),
                    SE = ~(sd(.)/sqrt(n()))))
quebec_nonchilled_mean
```

```
##    N     Mean     SE
## 1 21 35.33333 2.0941
```

Logics and relations in R

Can be used within the functions (i.e. when using `filter()`)

Character	Logics	Character	Relations
==	Less than	<	Less than
!=	Not equal to	>	Greater than
&	and	>=	Greater than or equal to
|	or	<=	Less than or equal to

Some other logics:

- `is.na()`: is NA (and other is.something() operators)
- `%in%`: group membership
- `!`: negation (e.g. `!si.na()`: is not NA)

Useful function:

- `which()`
- `grep()`, `grepl()` and other regexpr functions

More examples

Pick one or both questions, and use the appropriate function to answer them:

1. What is the mean `uptake` for each treatment type, for concentration values greater than or equal to 350?

2. Add a new column to the data frame, which multiplies the `conc` by the `uptake`
   Hint: use the `mutate()` function

Google "Rstudio dplyr cheatsheet" to get the following pdf, which may help
https://rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

Answers

```
# question 1: mean uptake for each treatment type,
# for concentration values greater than or equal to 350?
treat_above350 <- data_co2 %>%
  filter(conc >= 350) %>%
  group_by(Treatment) %>%
  summarise(mean = mean(uptake))
treat_above350
```

```
## # A tibble: 2 × 2
##   Treatment   mean
##   <chr>      <dbl>
## 1 chilled     27.6
## 2 nonchilled  35.9
```

```
# question 2: add a new column to the data frame
# which multiplies the `conc` by the `uptake`
data_co2 <- data_co2 %>%
  mutate(mult = conc*uptake)
head(data_co2, 3)
```

```
##   Plant   Type  Treatment conc uptake mult
## 1   Qn1 Quebec nonchilled   95   16.0 1520
## 2   Qn1 Quebec nonchilled  175   30.4 5320
## 3   Qn1 Quebec nonchilled  250   34.8 8700
```

So how does R compare to Excel?

@dinosaur

Manipulating data in R is reproducible and traceable, which means it is less prone to user errors.

Other useful information

- R for data science: https://r4ds.had.co.nz/ (the other of this book made the tidyverse!)
- Making nice-looking tables from your data: https://rfortherestofus.com/2019/11/how-to-make-beautiful-tables-in-r/
- R references: https://cran.r-project.org/doc/contrib/Baggott-refcard-v2.pdf
- Solutions to common tasks and problems in analyzing data in R: http://www.cookbook-r.com/
- Other useful websites for using dplyr: https://rpubs.com/bradleyboehmke/data_wrangling https://seananderson.ca/2014/09/13/dplyr-intro/ https://www.tidyverse.org/

Group project

- In pairs (groups of 2), select 2 papers which use linear regressions, ANOVAs and/or mixed models in their statistical analyses (we will go through these in class)
- Send the papers to me by Thursday, Nov. 10th (pablo.raguet@inrae.fr), I will choose 1 of those papers for you to prepare a critique
- On Monday, Nov. 24th, give a 12min presentation and answer 5-10min of questions (P. Raguet)

What to present

- Summarize the research aims and objectives of the paper: are they clear? Does the introduction give a good overview of the subject?
- Critique the methods and statistical analyses: are they sufficiently detailed? Are they appropriate for the research question?
- Explain the nature of the data (fixed vs. random factors, replications, etc.)
- Are the statistical tests appropriate? Why or why not?
- Are the results valid?
- How could you improve the paper?

Next session: choosing your statistical test