R COURSE

Introduction to R

Daniel Vaulot

2025-01-15

CNRS • SORBONNE UNIVERSITÉ
Station Biologique

UiO : University of Oslo

R sessions

- 01 - Introduction to R
- 02 - Data wrangling
- 03 - Data visualisation
- 04 - Markdown

R - Session 01

- What is R and why use R ?
- Resources
- Get started
- Fundamentals of R
- Data objects
- Vectors
- Operators
- Functions
- Packages

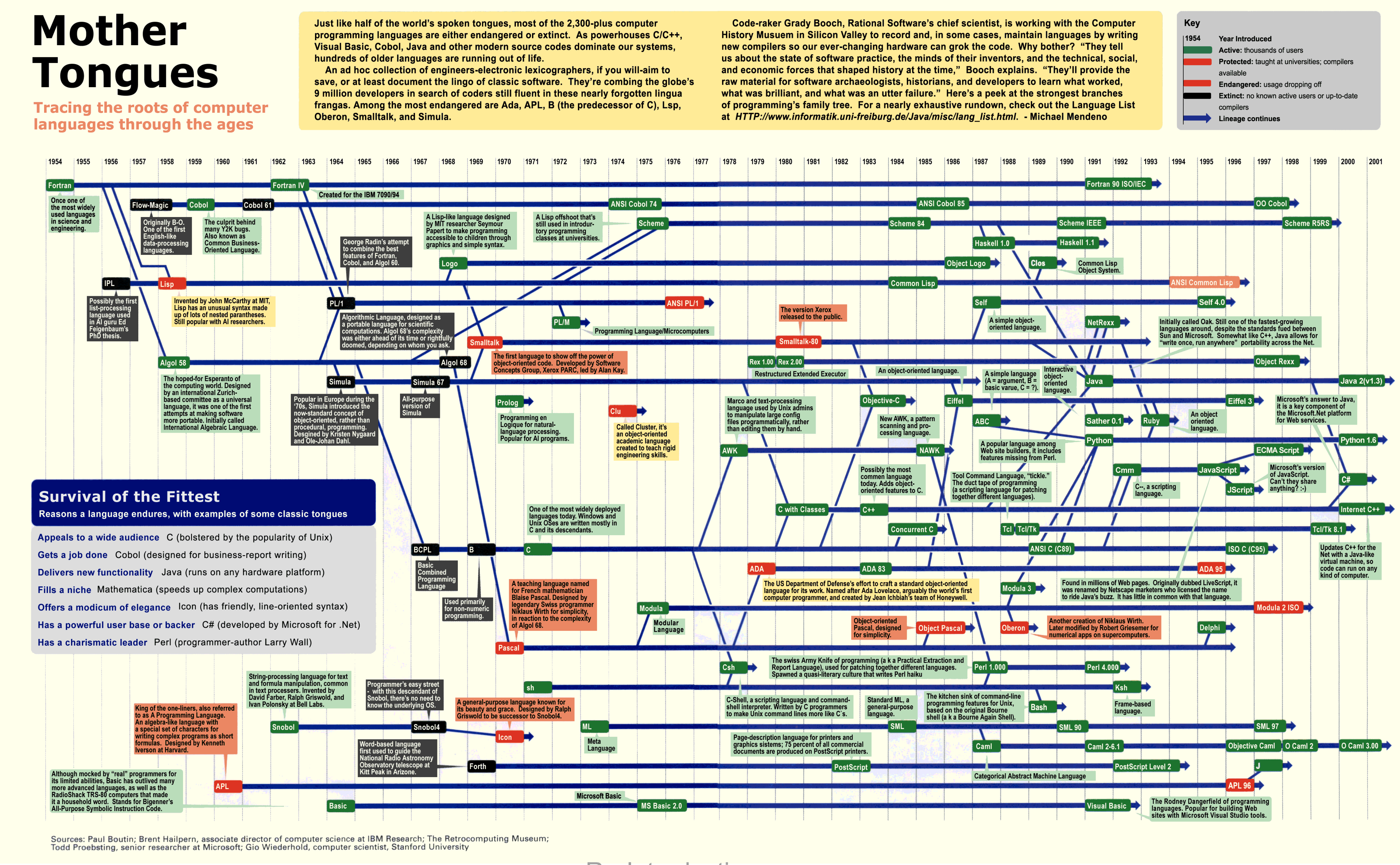
Introduction

- If you are an R guru:
  - Please refrain to answer during this session…
  - Help your neighbor

- Two special slide formatting
  - Your turn…
  - Warning

R

History of computer languages



History of R

- **Mid 1970s** - S Language for Statistical Computing conceived by John Chambers, Rick Becker, Trevor Hastie, Allan Wilks and others at Bell Labs
- **Early 1990's** - R was first implemented in the early 1990's by Robert Gentleman and Ross Ihaka, both faculty members at the University of Auckland.
- **1995** - Open Source Project
- **1997** - Managed by the R Core Group
- **2000** - First release of R
- **2011** - First release of R studio
- Historical notes - Paper from 1998

Why use R ?

- **Script vs. Menu driven software (e.g. Excel)**
  - Can be re-rerun with new data
  - Reproducible workflow
- **Open source**
  - Huge number of libraries
  - Tidy “universe” : tidyverse and ggplot2
    - Very easy to manipulate tables (select columns, create new variables)
    - High quality graphics
- **Work environment**
  - R studio
- **Document your data processing**
  - R markdown
  - Create HTML, pdf, presentations

What can you do with R ?

- **Science**
  - Statistics of course…
  - Data processing
  - Graphics
  - Time series analyses
  - Maps
  - Bioinformatics
- **But also**
  - Teach
  - Do a presentation
  - Write your CV
  - Build a web site
  - Write a book
  - Much more…

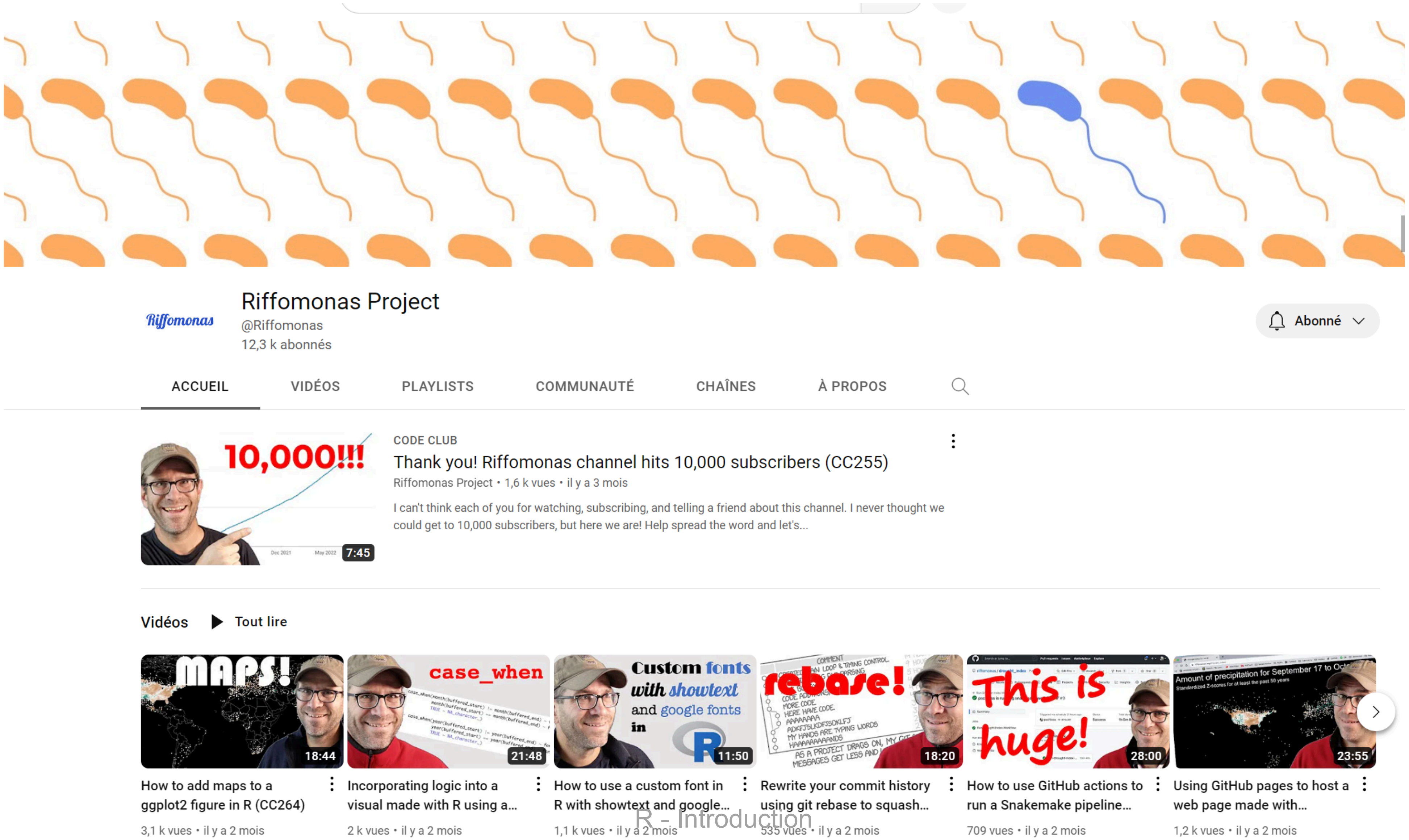
What can you do with R ?

Example of web page

:::

Yahoo channel - A must

- Riffomonas by Pat Schloss



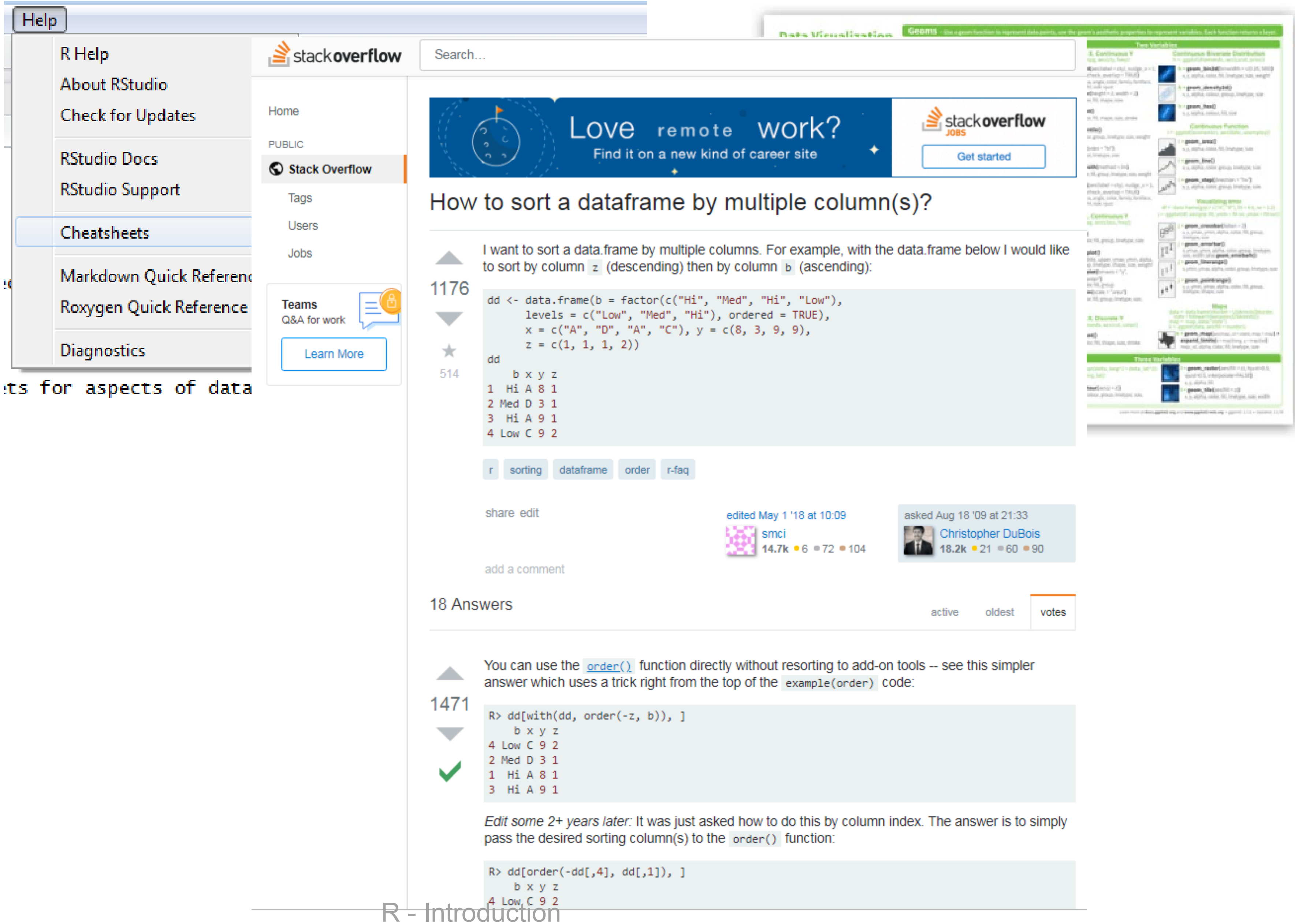
Help

Cheat sheets

- R basics
- ggplot2
- dplyr

Forum

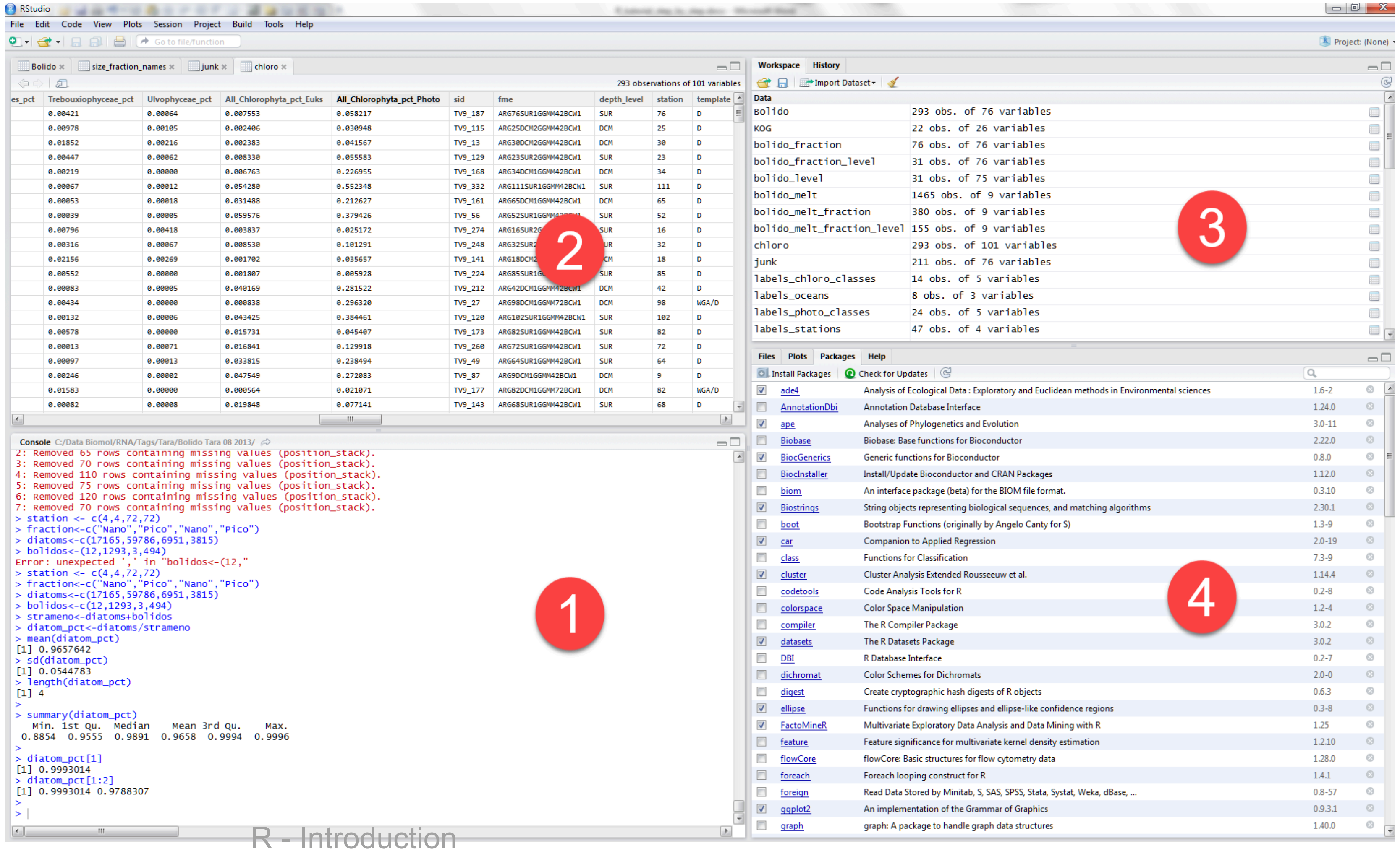
- Stack overflow
- Stat Blog
- R bloggers



Let’s get started

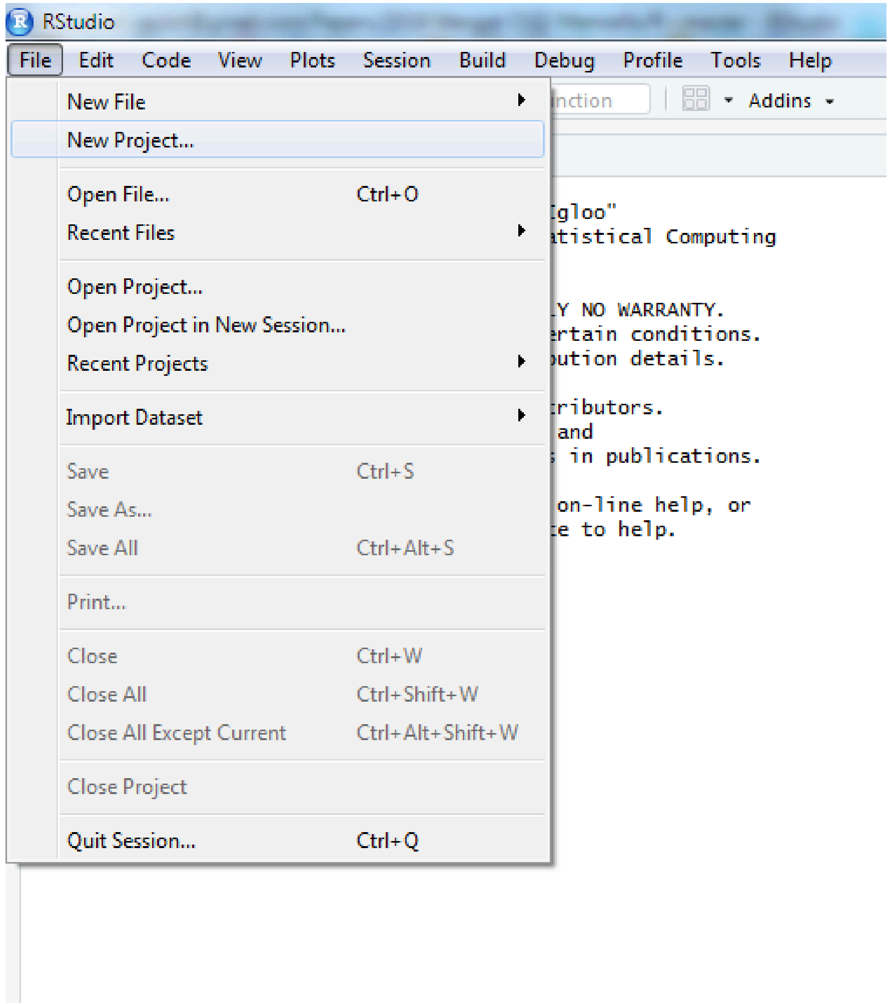
The R studio interface

- **Bottom left**
  - Console
- **Top left**
  - File editor for .R and .Rmd files
  - Data frame visualization
- **Top right**
  - Environment (i.e. R objects)
  - History
- **Bottom right**
  - Files
  - Plots
  - Packages
  - Help



Create a new project

- Open R studio
- Create new project for the course in a new directory
  - e.g. `Microbes course`



Your first script

Two ways to proceed

1. Type directly in command window

```
> print("Hello world")
```

```
[1] "Hello world"
```

2. Create a new script

> Type in script window
> * Select and execute (CTRL-R)
> * Source the script

R objects

Variables

Variables are abstracting your data.

Variables are **objects**

- Create a variable

```
> greeting = "Hello world"
> print(greeting)
```

```
[1] "Hello world"
```

- Update variable

```
> greeting = "Bonjour"
> print(greeting)
```

```
[1] "Bonjour"
```

Assignement

- Assignement done with **<-**

```
> x <- 1
> y <- 2
> x + y
```

```
[1] 3
```

```
> z <- x + y
> z
```

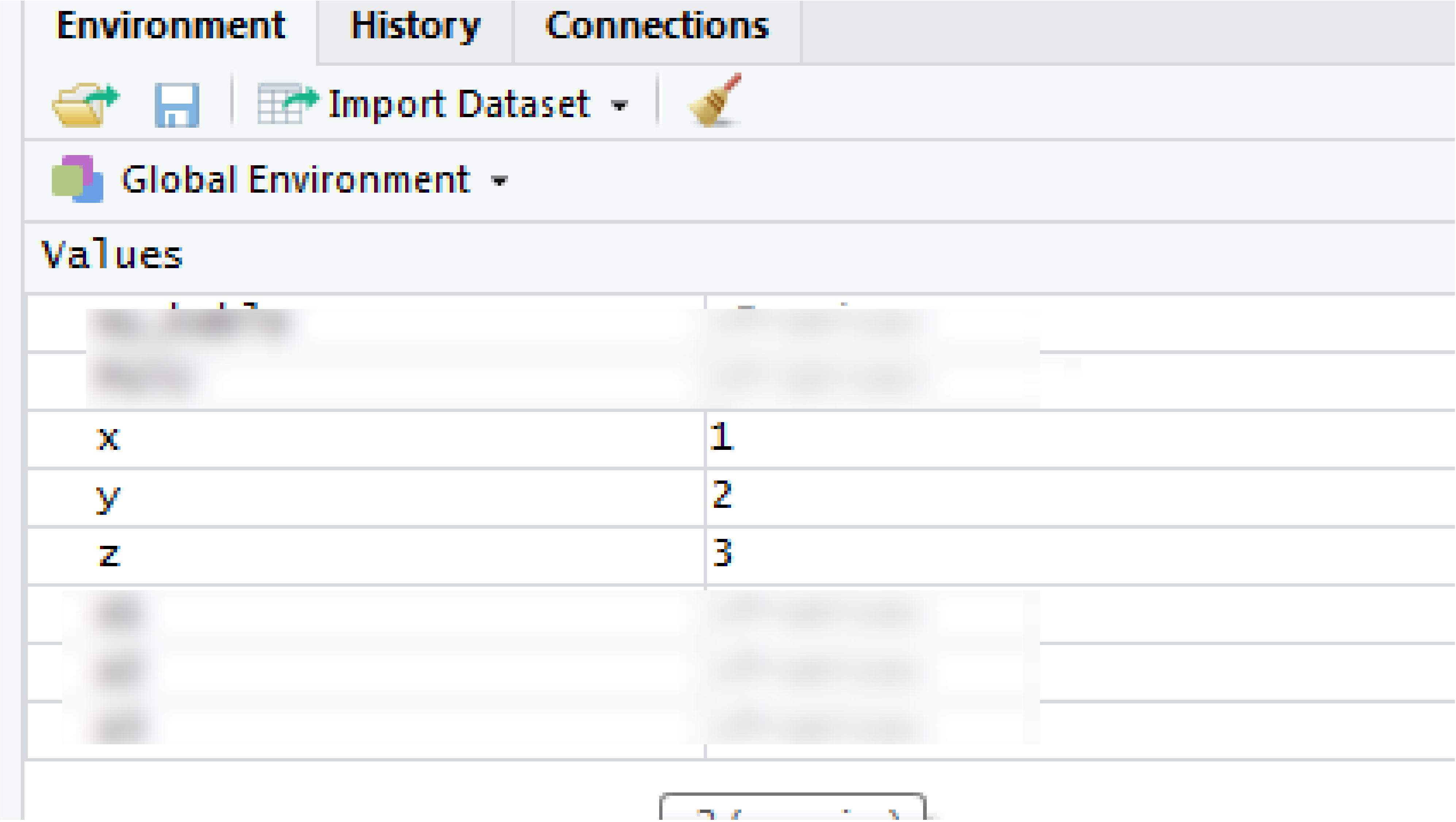
```
[1] 3
```

- **=** can be used instead of **<-** but refrain from it (not good style)

```
> z = x + y
```

Visualizing objects

You can view the values of the objects in R-studio environment window (top-right)



R is case sensitive

```
z
```

```
[1] 3
```

```
Z
```

```
Error in eval(expr, envir, enclos): object 'Z' not found
```

Rules for naming objects

- Use
  - letters
  - numbers
  - the dot
  - the underscore (not the minus sign !)
- Start always with a letter
  - `Myvariable`, `Myvariable1`, `Myvariable.1`,`Myvariable_01` are OK
  - `1Myvariable`, `My-variable`, `Myvariable@` are **not** OK

Rules for naming objects

- Use consistent naming: five conventions
  - alllowercase: e.g. adjustcolor
  - period.separated: e.g. plot.new
  - **underscore_separated**: e.g. numeric_version
  - lowerCamelCase: e.g. addTaskCallback
  - UpperCamelCase: e.g. SignatureMethod
- Prefer third one, much more easy to read
  - Use **names** for objects : **last_name**
  - Use **verbs** for function : **build_name**
- Think about best order
  - e.g. prefer maybe **name_last** because then you can have name_first, name_full…
  - and you identify that all these objects are related to a name…

Data types

- **character**: “Daniel”, “This is a course in R”, ‘Joe Biden’
- **numeric**: 2, 15.5, 10e-3
- **integer**: 2L (the L tells R to store this is an integer)
- **date**: 2018-02-25
- **logical**: TRUE, FALSE
- **complex**: 1+4i (complex numbers with real and imaginary parts)

- **No data** “NA”
- **Not a number** “NaN” (e.g. division by zero)

Data structures

- **Vector**
- **List**
- **Matrix**
- **Data frames**
- **Function**

Vectors

Vectors

- The basic R structure is a vector (think as a column in Excel):

$$\begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix}$$

- A vector can contain only a single element

$$\begin{bmatrix} 10 \end{bmatrix}$$

- Assign a value to a vector

```r
x <- 10
x
```

```
[1] 10
```

Vectors

- Assign several elements

```
x <- c(10,20,30)
x
```

```
[1] 10 20 30
```

- Assign range

```
x <- 10:30
x
```

```
 [1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

- Assign characters

```
PoTU <- c("Jo", "Biden")
PoTU
```

```
[1] "Jo"     "Biden"
```

- Assign logical

```
flags <- c(TRUE, FALSE, TRUE)
flags
```

Access specific elements of a vector

- First

```
x[1]
```

```
[1] 10
```

- Range

```
x[1:5]
```

```
[1] 10 11 12 13 14
```

- Remove one element

```
x[-1]
```

```
 [1] 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

Determine object properties

Apply functions (we will come back to functions latter)

- **typeof()** - what is the object's data type (low-level)?
- **length()** - how long is it? What about two dimensional objects?

```
typeof(x)
length(x)
```

```
[1] "integer"
[1] 21
```

> What is the type and length of **PoTU** ?

Operators

Arithmetic Operators

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

We are performing vector operations !

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ .. \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 3 \\ .. \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 6 \\ .. \end{bmatrix}$$

Think about it as adding 2 columns in Excel.

Arithmetic Operators

- Vector one element

```
x <- 1
y <- 2
z <- x + y
z
```

```
[1] 3
```

- Vector several elements

```
# Two instructions on the same line
x <- 1:9;  y <- 1:9
z <- x + y
z
```

```
[1]  2  4  6  8 10 12 14 16 18
```

- Several instructions on same line separated by **;**
- The hastag **#** indicate a comment -> Use heavily to document your code
- However, it is even better to use R markdown (we will see it later)

> Use the other operators

Arithmetic Operators

- What happens when the vectors have different number of elements ?

```
x <- 1:9
y <- 1
z <- x + y
z
```

```
[1]  2  3  4  5  6  7  8  9 10
```

Equivalent to

```
y <- c(1,1,1,1,1,1,1,1,1)
```

The recycling rule…

Can we add logical ?

```r
x <- TRUE
y <- FALSE
z <- x + y
z
```

```
[1] 1
```

No error but…

The resulting variable is transformed to a **numeric**

> How you would show that ?

```r
typeof(x)
```

```
[1] "logical"
```

```r
typeof(z)
```

```
[1] "integer"
```

Logical Operators

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x | y	x OR y
x & y	x AND y
isTRUE(x)	test if X is TRUE

Logical Operators

```r
x <- TRUE
y <- FALSE
z1 <- x | y
z2 <- x == y
z1
```

```
[1] TRUE
```

```r
z2
```

```
[1] FALSE
```

Do not mix

- == which is logical operator
- = which is assignement

Can we add characters ?

```r
first <- "Jo"
last <- "Biden"
full <- first + last
```

```
Error in first + last: non-numeric argument to binary operator
```

Generates an error

> What can we do ?

Functions

Functions

Functions perform specific task on objects

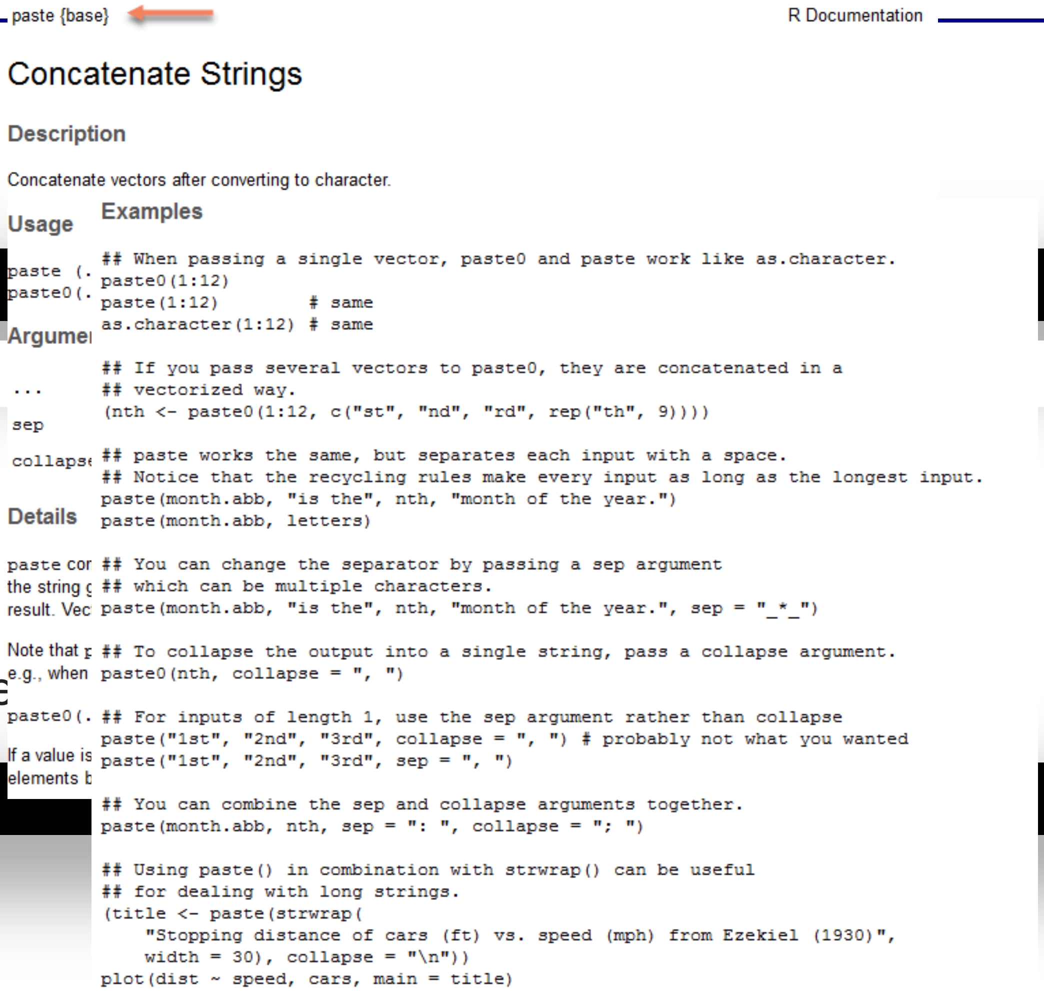
- e.g. to concatanate strings we use **paste()**

```
paste(first,last)
```

```
[1] "Jo Biden"
```

- Functions take **arguments** and return an object called **result**
- To know the arguments
    - Use “?”
    - Can also go directly to Help panel and type function name

```
? paste()   # Do not forget the parenthesis
```


paste {base}   R Documentation

Concatenate Strings

Description

Concatenate vectors after converting to character.

Usage

paste (.
paste0 (.

Argume

...

sep

collaps

Details

paste cor
the string
result. Vec

Note that
e.g., when

paste0 (.

If a value is
elements b

Examples

```
## When passing a single vector, paste0 and paste work like as.character.
paste0(1:12)
paste(1:12)        # same
as.character(1:12) # same

## If you pass several vectors to paste0, they are concatenated in a
## vectorized way.
(nth <- paste0(1:12, c("st", "nd", "rd", rep("th", 9))))

## paste works the same, but separates each input with a space.
## Notice that the recycling rules make every input as long as the longest input.
paste(month.abb, "is the", nth, "month of the year.")
paste(month.abb, letters)

## You can change the separator by passing a sep argument
## which can be multiple characters.
paste(month.abb, "is the", nth, "month of the year.", sep = "_*_")

## To collapse the output into a single string, pass a collapse argument.
paste0(nth, collapse = ", ")

## For inputs of length 1, use the sep argument rather than collapse
paste("1st", "2nd", "3rd", collapse = ", ") # probably not what you wanted
paste("1st", "2nd", "3rd", sep = ", ")

## You can combine the sep and collapse arguments together.
paste(month.abb, nth, sep = ": ", collapse = "; ")

## Using paste() in combination with strwrap() can be useful
## for dealing with long strings.
(title <- paste(strwrap(
    "Stopping distance of cars (ft) vs. speed (mph) from Ezekiel (1930)",
    width = 30), collapse = "\n"))
plot(dist ~ speed, cars, main = title)
```

Getting what you want

Let’s apply paste :

```r
paste(first,last)
```

```
[1] "Jo Biden"
```

- We would like to get “Jo_Biden”

> Can you read the help and suggest a change in the way we call the function ?

```r
paste(first,last, sep="_")
```

```
[1] "Jo_Biden"
```

Write your own function

> If you write 3 times the same piece of code, then write a function…

```
my_sum <- function(a, b) {
  c <- a + b
  return (c)
}
```

- **my_sum** : function name
- **a, b** : arguments
- instructions are enclosed by braces (**{}**)
- **return()** : the value(s) returned

- More compact way

```
my_sum <- function(a, b) {a + b}
```

Call your function

```r
my_sum(10, 20)
```

```
[1] 30
```

- better

```r
my_sum(a = 10, b = 20)
```

```
[1] 30
```

> Write a function to compute a product

Examples of functions

Most of the time you do not have to write functions because someone has already written one for what you want to do…

- Sum

```r
x <- 1:100
sum(x)
```

```
[1] 5050
```

- Sampling a normal distribution

```r
y <- rnorm(10, mean = 0, sd = 1)
y
```

```
 [1] -1.2082061 -0.1915338 -1.4461329 -1.5664386 -2.5408725 -1.4697788
 [7]  0.5294263  0.1391368  1.7930026 -0.8428267
```

Statistics

```r
mean(y)
sd(y)
```

```
[1] -0.6804224
[1] 1.258433
```

Sample more points… 10,000 instead of 100

```r
y <- rnorm(10000, mean = 0, sd = 1)
mean(y)
sd(y)
```

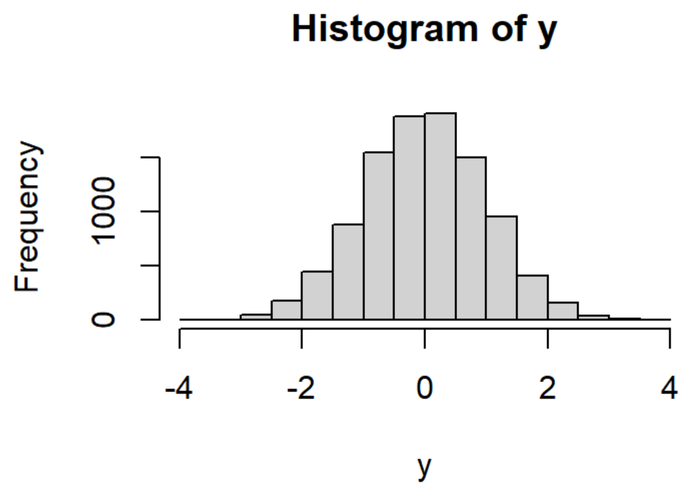
```
[1] -0.003926506
[1] 0.9976069
```

Plot

- Histogram

```
library(graphics)
hist(y)
```

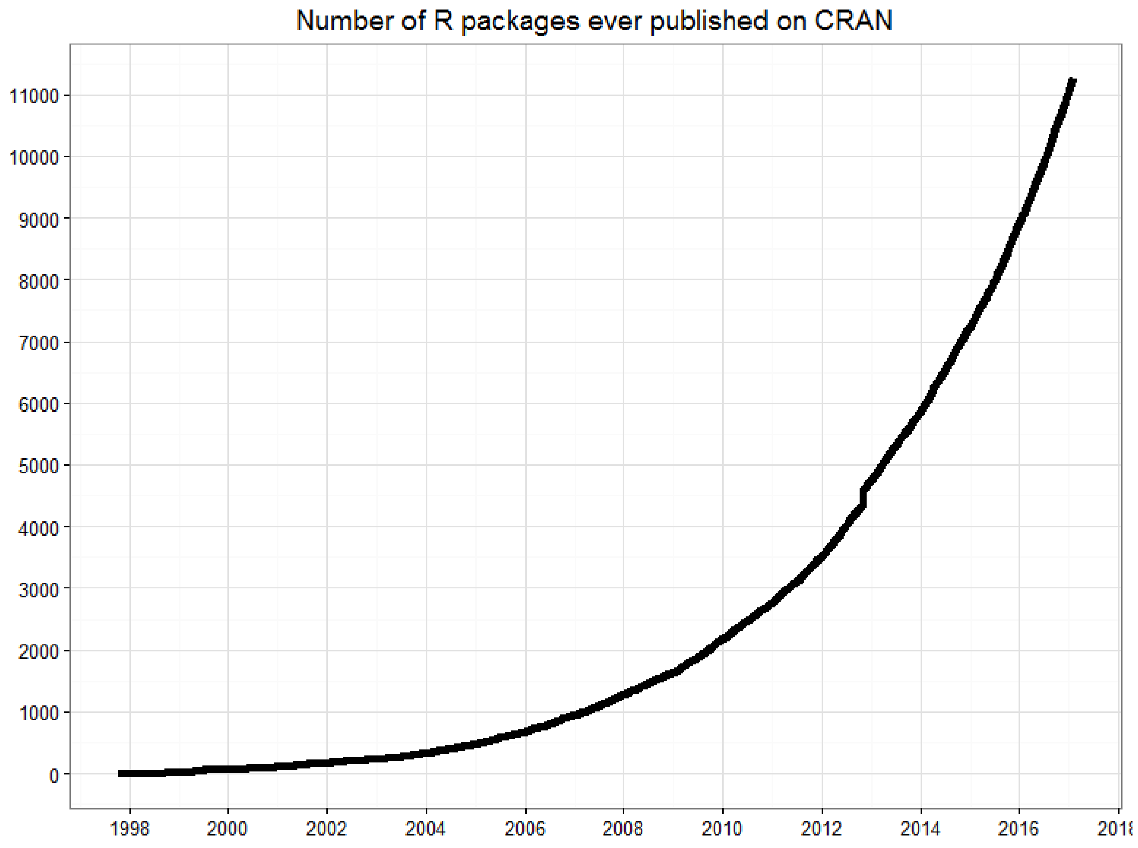
What is this “library()”



Packages

Packages

- Packages are set of functions that have a common goal.
- They are really the strength of R

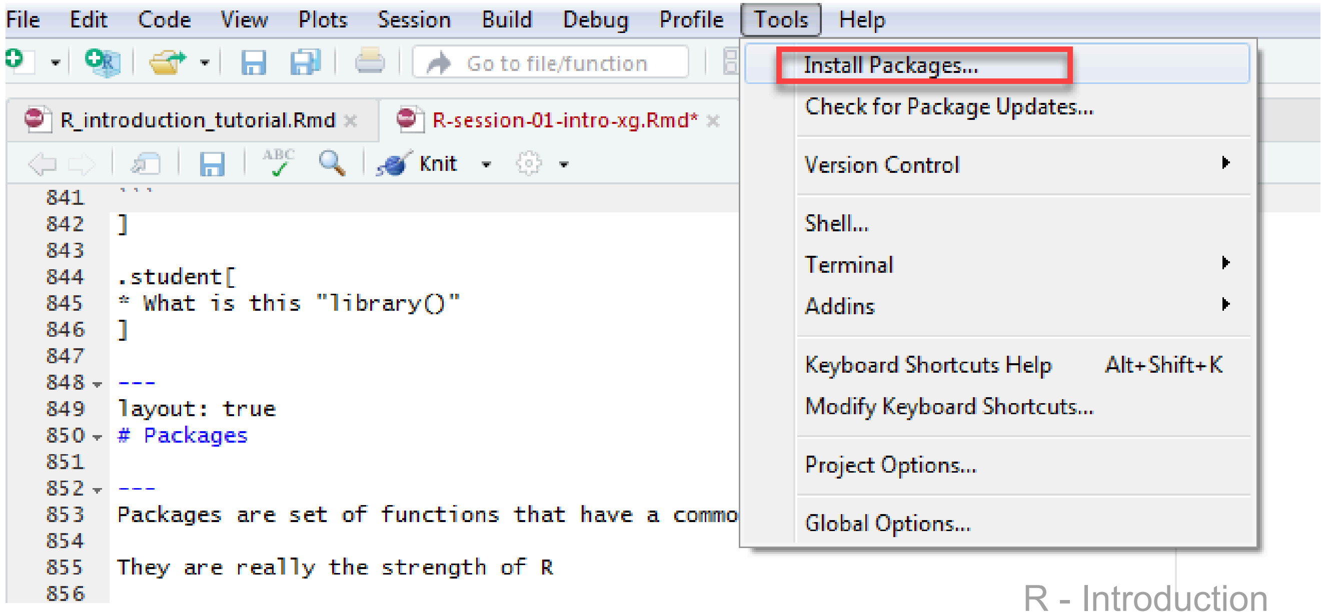


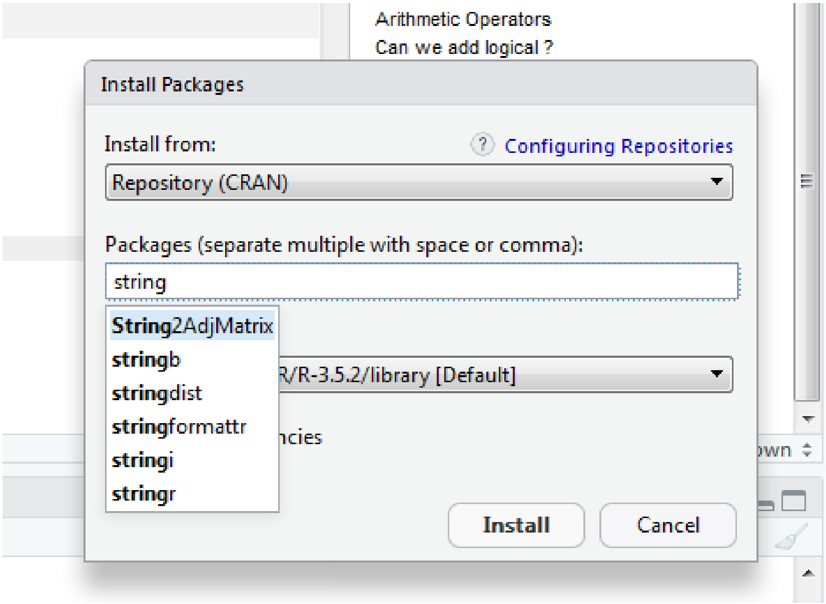
- And these are only the “official”” packages. You can find more on GitHub

Installing a package

Download on your computer the package you need

- Install package **stringr** (to manipulate strings of characters)





Using a package

To use functions from the package

- use the syntax `package::function`

```
stringr::str_c(first,last, sep= " ")
```

```
[1] "Jo Biden"
```

OR

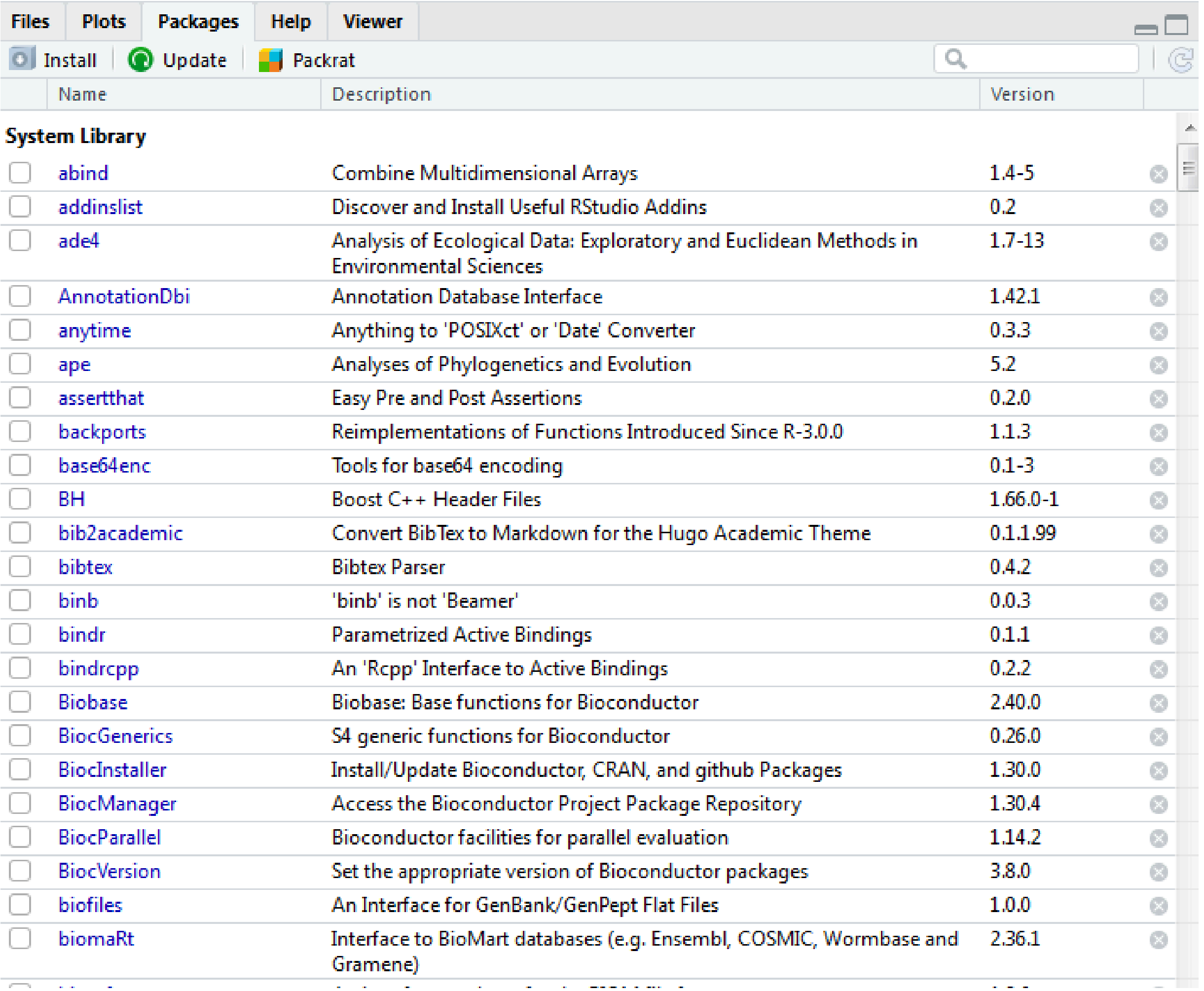
- load the package with the library function

```
library(stringr)
str_c(first,last, sep= " ")
```

```
[1] "Jo Biden"
```

> Sometimes functions from different libraries have similar names

List installed packages



Name	Description	Version
System Library		
abind	Combine Multidimensional Arrays	1.4-5
addinslist	Discover and Install Useful RStudio Addins	0.2
ade4	Analysis of Ecological Data: Exploratory and Euclidean Methods in Environmental Sciences	1.7-13
AnnotationDbi	Annotation Database Interface	1.42.1
anytime	Anything to 'POSIXct' or 'Date' Converter	0.3.3
ape	Analyses of Phylogenetics and Evolution	5.2
assertthat	Easy Pre and Post Assertions	0.2.0
backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.3
base64enc	Tools for base64 encoding	0.1-3
BH	Boost C++ Header Files	1.66.0-1
bib2academic	Convert BibTex to Markdown for the Hugo Academic Theme	0.1.1.99
bibtex	Bibtex Parser	0.4.2
binb	'binb' is not 'Beamer'	0.0.3
bindr	Parametrized Active Bindings	0.1.1
bindrcpp	An 'Rcpp' Interface to Active Bindings	0.2.2
Biobase	Biobase: Base functions for Bioconductor	2.40.0
BiocGenerics	S4 generic functions for Bioconductor	0.26.0
BiocInstaller	Install/Update Bioconductor, CRAN, and github Packages	1.30.0
BiocManager	Access the Bioconductor Project Package Repository	1.30.4
BiocParallel	Bioconductor facilities for parallel evaluation	1.14.2
BiocVersion	Set the appropriate version of Bioconductor packages	3.8.0
biofiles	An Interface for GenBank/GenPept Flat Files	1.0.0
biomaRt	Interface to BioMart databases (e.g. Ensembl, COSMIC, Wormbase and Gramene)	2.36.1

Recap

- R is case sensitive: Z != z
- Objects: data types vs data structures
- Vectors: think in vector operations
- Operators: arithmetic vs. logical
- Functions: try to practice

Next: 02 - Data wrangling

- Data frames
- Concept of tidy data
- Reading data
- Manipulating data
- Selecting columns
- Selecting rows