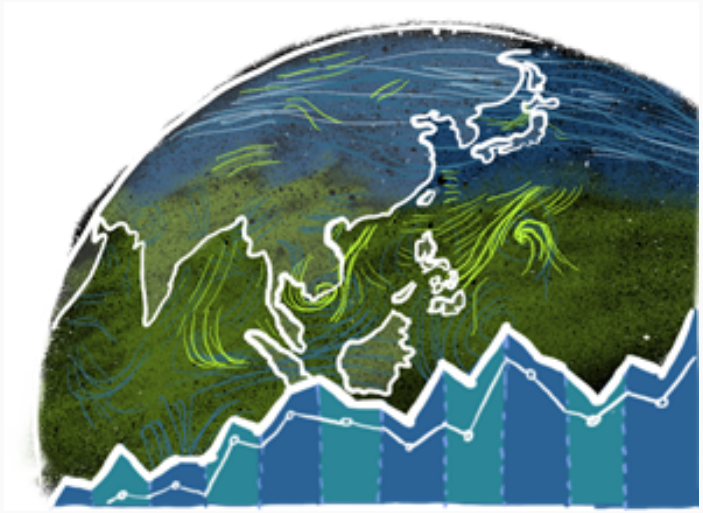


R session 01 - Introduction to R

Daniel Vaulot

2019-01-17

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CNRS • SORBONNE UNIVERSITÉ
Station Biologique de Roscoff

Outline

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

Introduction

- Who has used R before ?

Introduction

- Who has used R before ?
- What other programming language have you used before ?

Introduction

- Who has used R before ?
- What other programming language have you used before ?
- For those who are experts in R

Introduction

- Who has used R before ?
- What other programming language have you used before ?
- For those who are experts in R
  - please refrain to answer during this session...
  - help your neighbor...

Introduction

- Who has used R before ?
- What other programming language have you used before ?
- For those who are experts in R
  - please refrain to answer during this session...
  - help your neighbor...
- Two special slide formatting

> Your turn...

Introduction

- Who has used R before ?
- What other programming language have you used before ?
- For those who are experts in R
  - please refrain to answer during this session...
  - help your neighbor...
- Two special slide formatting

> Your turn...

> Warning

Computer languages

Mother Tongues

Tracing the roots of computer languages through the ages

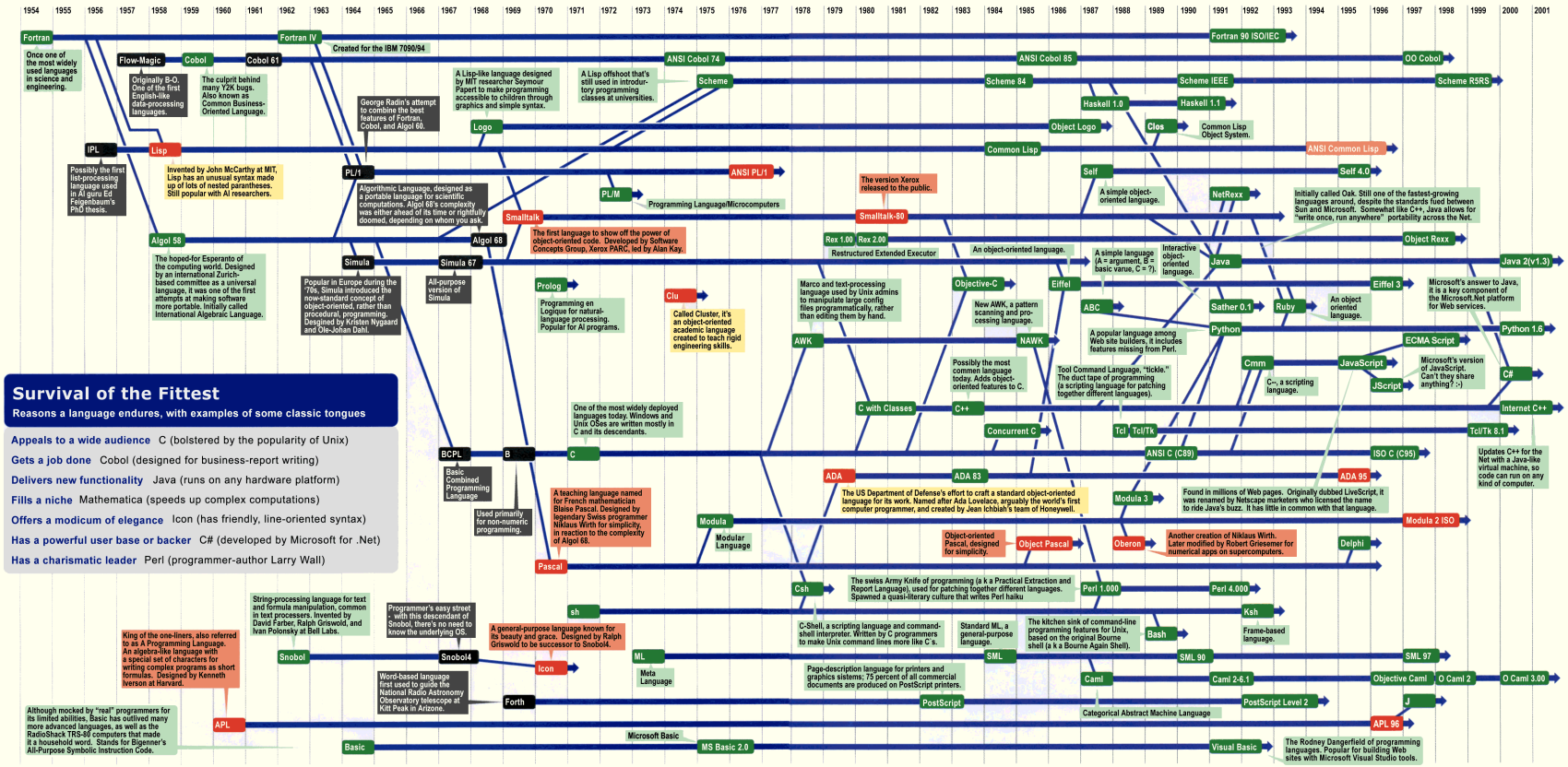
Just like half of the world's spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, Cobol, Java and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers-electronic lexicographers, if you will-aim to save, or at least document the lingo of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten lingua frangas. Among the most endangered are Ada, APL, B (the predecessor of C), Lsp, Oberon, Smalltalk, and Simula.

Code-raker Grady Booch, Rational Software's chief scientist, is working with the Computer History Musuem in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time," Booch explains. "They'll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure." Here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive rundown, check out the Language List at *HTTP://www.informatik.uni-freiburg.de/Java/misc/lang_list.html.* - Michael Mendeno

Key

- 1954 Year Introduced
- **Active:** thousands of users
- **Protected:** taught at universities; compilers available
- **Endangered:** usage dropping off
- **Extinct:** no known active users or up-to-date compilers
- **Lineage continues**



Survival of the Fittest

Reasons a language endures, with examples of some classic tongues

- **Appeals to a wide audience** C (bolstered by the popularity of Unix)
- **Gets a job done** Cobol (designed for business-report writing)
- **Delivers new functionality** Java (runs on any hardware platform)
- **Fills a niche** Mathematica (speeds up complex computations)
- **Offers a modicum of elegance** Icon (has friendly, line-oriented syntax)
- **Has a powerful user base or backer** C# (developed by Microsoft for .Net)
- **Has a charismatic leader** Perl (programmer-author Larry Wall)

Sources: Paul Boutin; Brent Hailpern, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gio Wiederhold, computer scientist, Stanford University

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

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

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

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

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
- **Share your data and workflow**
  - GitHub

Introduction

What can you do with R ?

What can you do with R ?

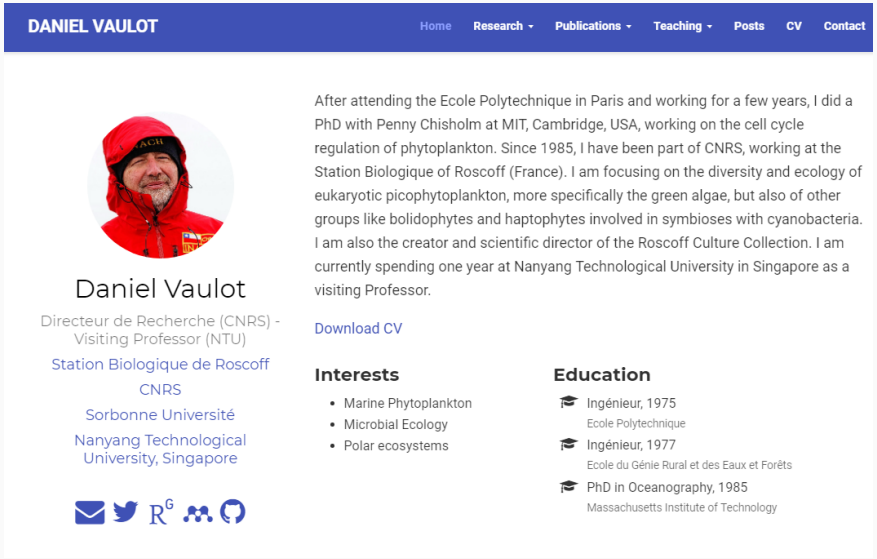
- **Science**
  - Statistics of course...
  - Data processing
  - Graphics
  - Time series analyses
  - Maps
  - Bioinformatics

What can you do with R ?

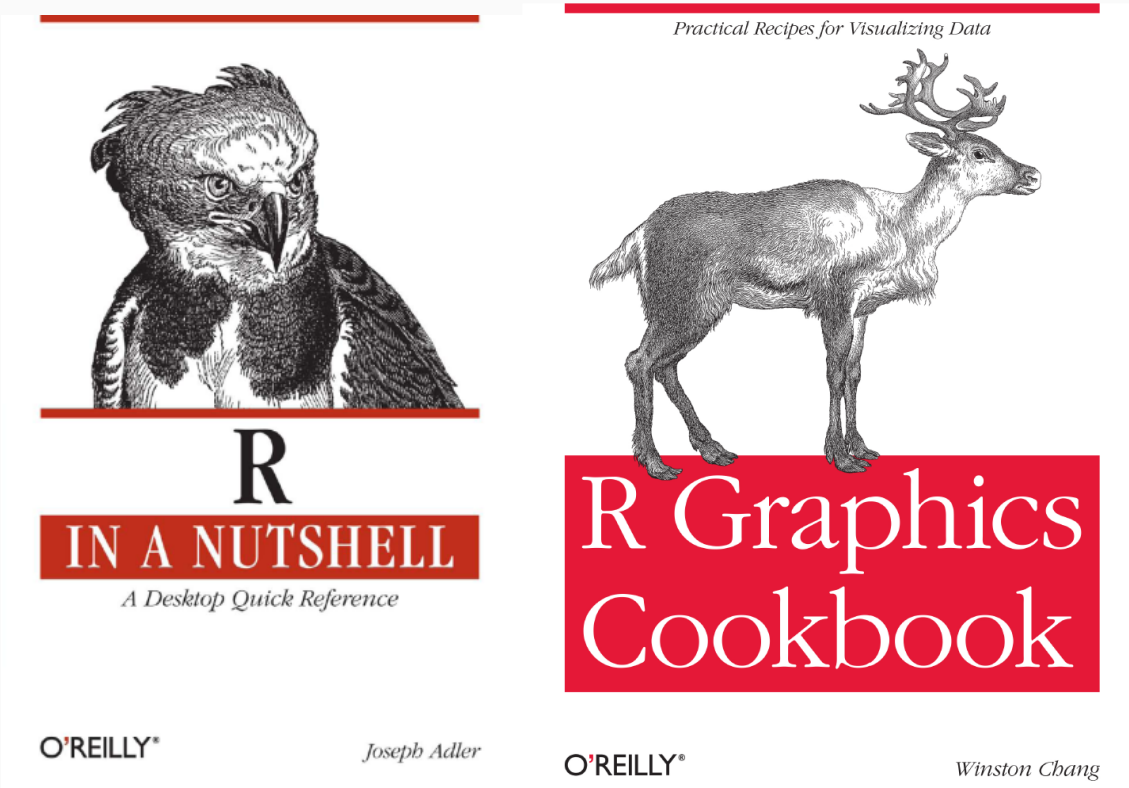
- **Science**
  - Statistics of course...
  - Data processing
  - Graphics
  - Time series analyses
  - Maps
  - Bioinformatics
- **But also**
  - Teach
  - Do a presentation
  - Write your CV
  - Build a web site
  - Write a book
  - Much more...

What can you do with R ?

- **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...



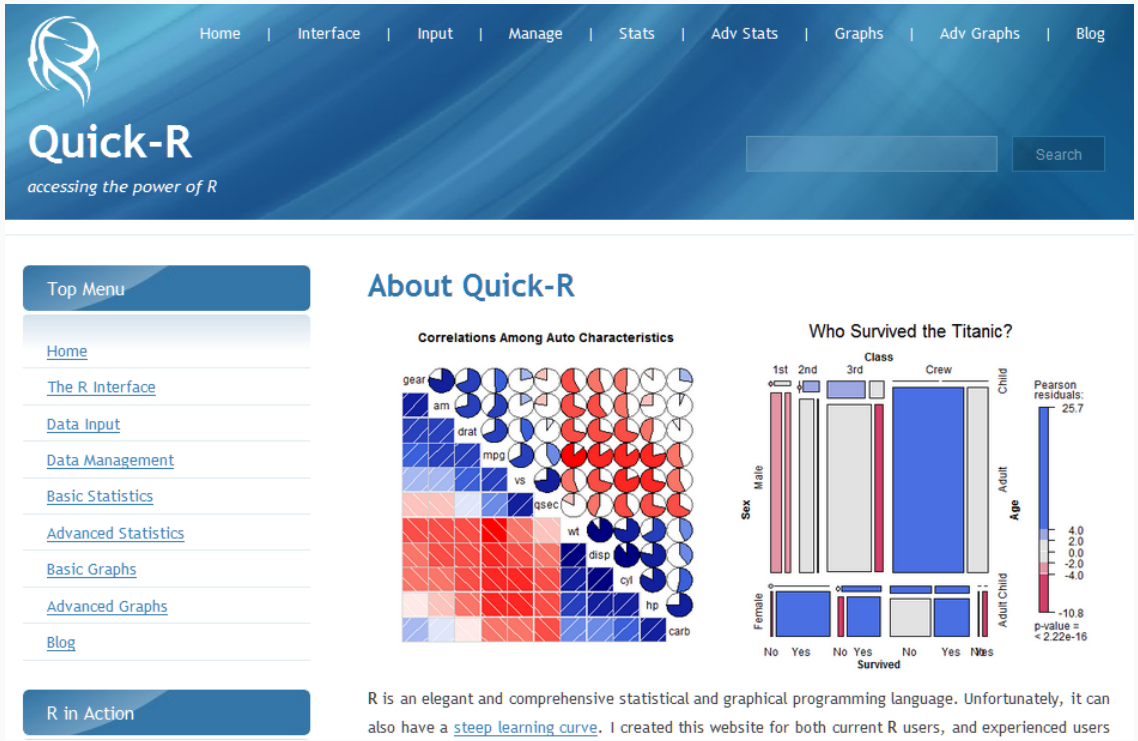
Books and Manuals



- R intro : Very good introduction to R, short and clear
- R in a nutshell : Many many receipes to solve all your questions
- R graphics cook book : very good for graphics

Resources

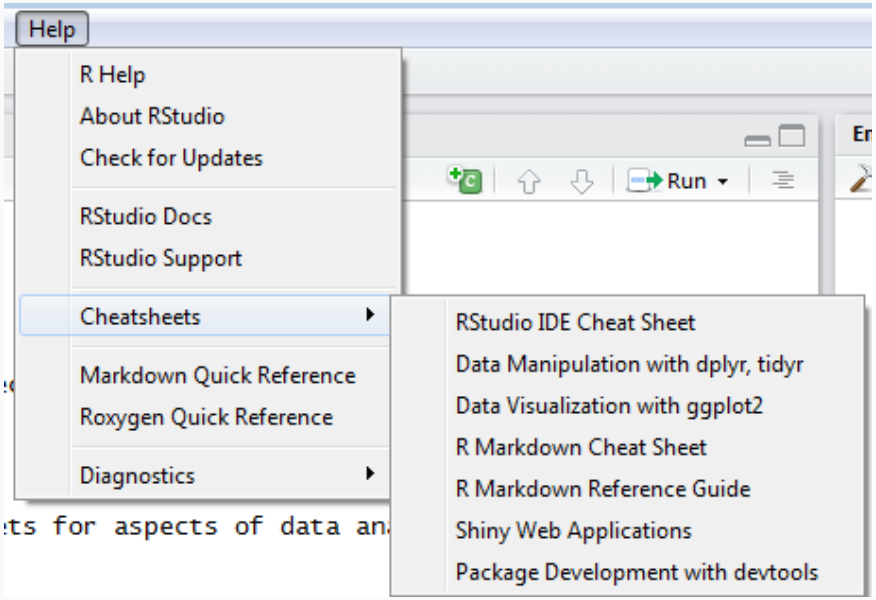
On line courses and web sites



- Coursera
- Pluralsight
- Quick-R, very simple

Resources

Cheat sheets



- R basics
- ggplot2
- dplyr

Resources

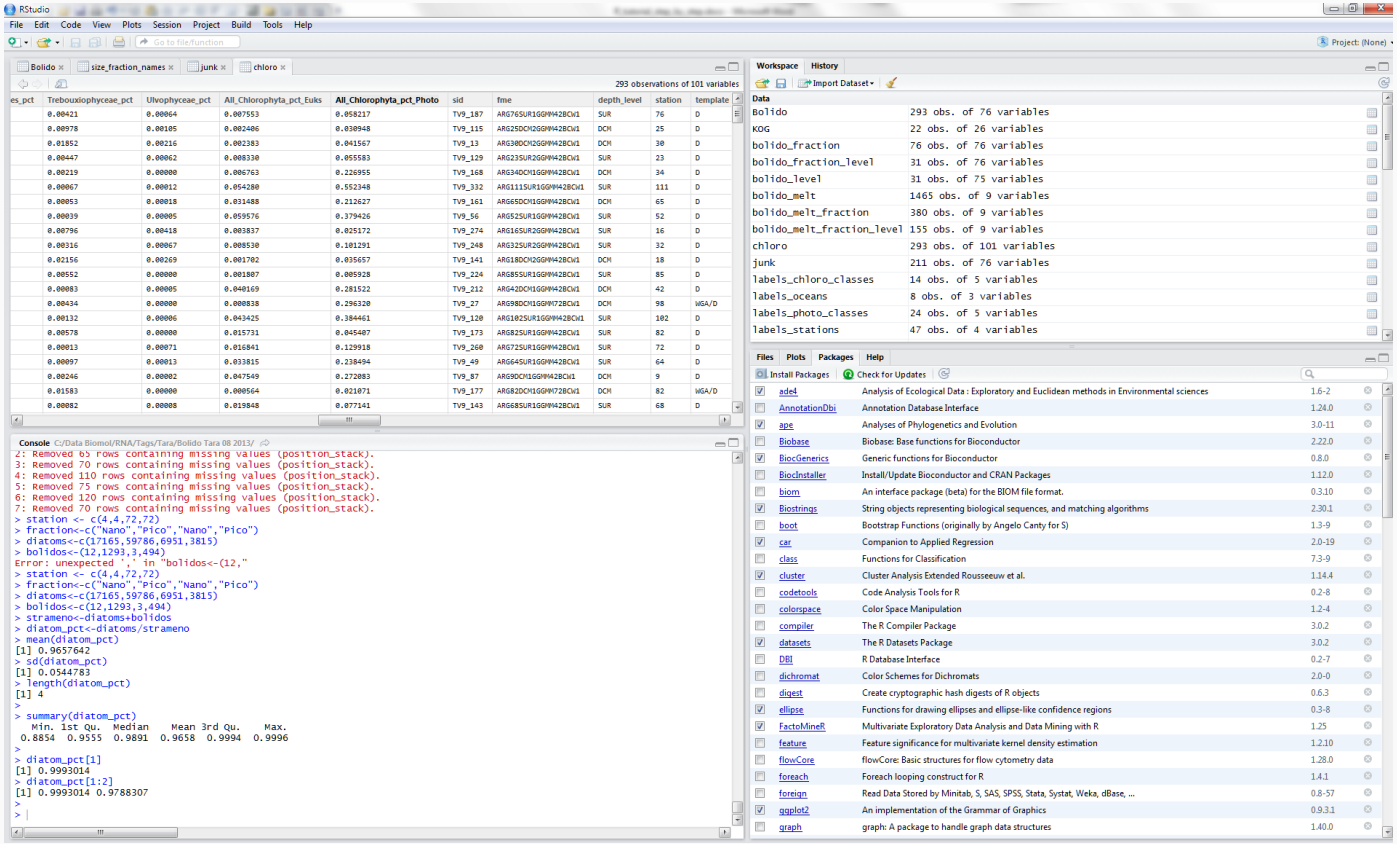
Forum

- https://stackoverflow.com/
- http://r-statistics.co/
- https://stats.stackexchange.com/
- https://www.r-bloggers.com/

Let's get started

Setup

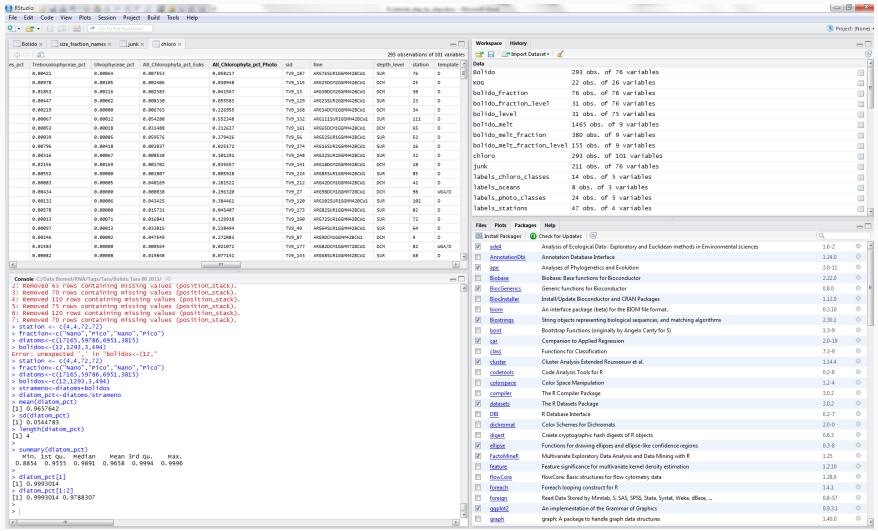
- Install R
- Install R studio



Let's get started

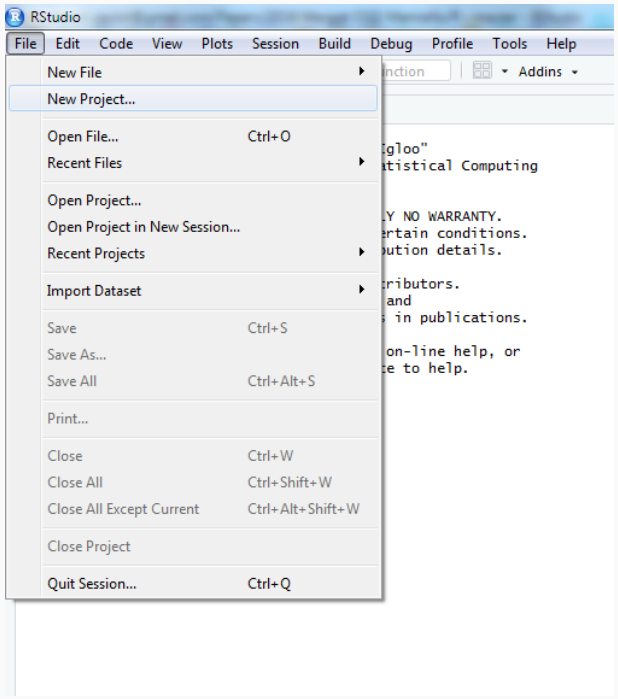
The R studio interface

- **Bottom left**
  - Console
- **Top left**
  - File editor
- **Top right**
  - Environment (i.e. R objects)
  - History
- **Bottom right**
  - Files
  - Plots
  - Packages
  - Help



Let's get started

Create a new project



- Open R studio
- Create new project for the course in a new directory
  - e.g. Data Science Class

Let's get started

Your first script

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

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

Two ways to proceed

1. Type directly in command window

Let's get started

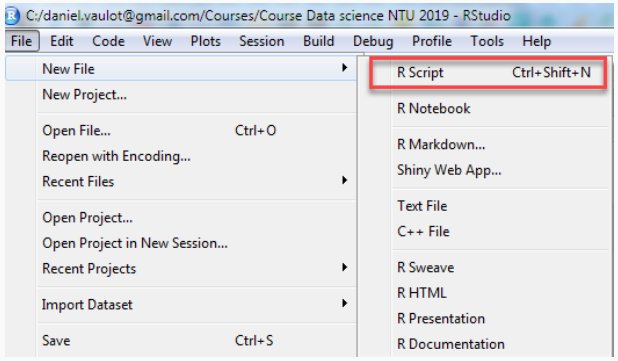
Your first script

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

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

Two ways to proceed

1. Type directly in command window
2. Create a new script



Type in script window, select and execute (CTRL-R)

Everything in R is an **object**

- Assignement done with **<-**

```
> x ← 1
> y ← 2
> x + y
```

```
[1] 3
```

Everything in R is an **object**

- Assignement done with **<-**

```
> x ← 1
> y ← 2
> x + y
```

```
[1] 3
```

```
> z ← x + y
> z
```

```
[1] 3
```

The R language

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

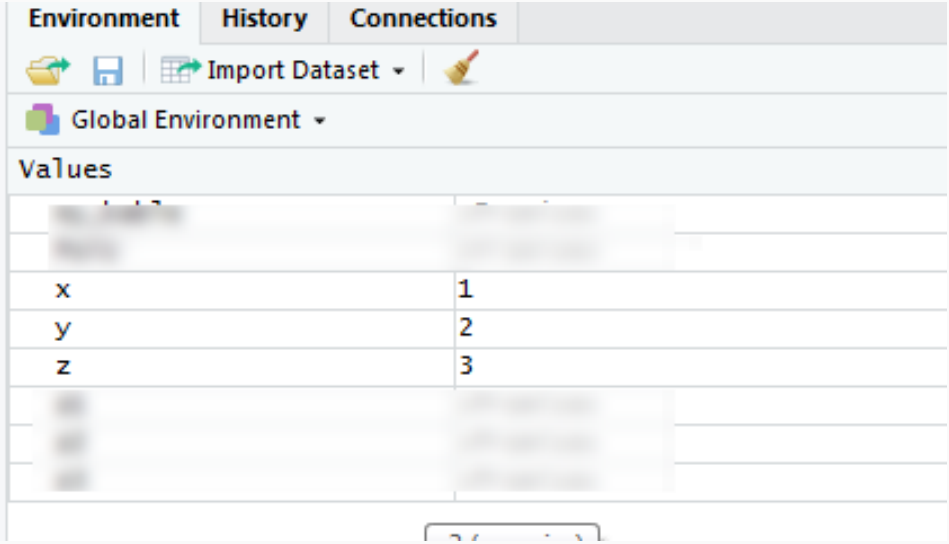
```
> z = x + y
```

The R language

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

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

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



R is **case sensitive**

```
> Z
```

R is **case sensitive**

```
> Z
```

```
> 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

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...

R objects

Data types

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

R objects

Data types

- **character**: "Daniel", "This is a course in R", 'Donald'
- **numeric**: 2, 15.5, 10e-3
- **integer**: 2L (the L tells R to store this as 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)

R objects

Data structures

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

Vectors

The basic R structure is a vector:

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

Vectors

The basic R structure is a vector:

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

A vector can with a single element only

$$[10]$$

Vectors

The basic R structure is a vector:

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

A vector can with a single element only

$$[10]$$

Assign a value to a vector

```
x ← 10
x
```

```
[1] 10
```

Assign several elements

```
x ← c(10, 20, 30)
x
```

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

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
```

Vectors

Assign characters

```
PoTU ← c("Donald", "Trump")
PoTU
```

```
[1] "Donald" "Trump"
```

Assign logical

```
flags ← c(TRUE, FALSE, TRUE)
flags
```

```
[1]  TRUE FALSE  TRUE
```

Access specific elements of a vector

First

```
x[1]
```

```
[1] 10
```

Access specific elements of a vector

First

```
x[1]
```

```
[1] 10
```

Range

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

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

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)
```

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
```

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** ?

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

Arithmetic Operators

We are performing vector operations !

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

Arithmetic Operators

Vector one element

```
x ← 1
y ← 2
z ← x + y
z
```

```
[1] 3
```

Arithmetic Operators

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
```

Arithmetic Operators

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 separate by **;**
- The hastag **#** indicate a comment -> Use heavily to document your code

Arithmetic Operators

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 separate by **;**
- The hastag **#** indicate a comment -> Use heavily to document your code

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
```

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
```

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...

Operators

Can we add logical ?

```
x ← TRUE
y ← FALSE
z ← x + y
z
```

Operators

Can we add logical ?

```
x ← TRUE
y ← FALSE
z ← x + y
z
```

```
[1] 1
```

Can we add logical ?

It does not give an error but...

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

> How you would show that ?

Can we add logical ?

It does not give an error but...

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

> How you would show that ?

```
typeof(x)
```

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

```
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

Operators

Logical Operators

```
x ← TRUE
y ← FALSE
z1 ← x | y
z2 ← x == y
```

Logical Operators

```
x ← TRUE
y ← FALSE
z1 ← x | y
z2 ← x == y
```

```
[1] TRUE
```

```
[1] FALSE
```

Do not mix

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

Operators

Can we add characters ?

```
first ← "Donald"
last ← "Trump"
full ← first + last
```

Can we add characters ?

```
first ← "Donald"
last ← "Trump"
full ← first + last
```

Generates an error

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

Can we add characters ?

```
first ← "Donald"
last ← "Trump"
full ← first + last
```

Generates an error

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

> What can we do ?

Functions

Function perform specific task on objects

- e.g. to concatanate strings we use **paste0()**

Functions

Function perform specific task on objects

- e.g. to concatanate strings we use **paste0()**

```
paste0(first, last)
```

```
[1] "DonaldTrump"
```

Functions

Function perform specific task on objects

- e.g. to concatanate strings we use **paste0()**

```
paste0(first, last)
```

```
[1] "DonaldTrump"
```

- Functions take **arguments** and return an object called **result**
- To know the arguments use ?

```
 ? paste0()  # Do not forget the parenthesis
```

Functions

Function perform specific task on objects

- e.g. to concatanate strings we use **paste0()**

```
paste0(first, last)
```

```
[1] "DonaldTrump"
```

- Functions take **arguments** and return an object called **result**
- To know the arguments use ?

```
 ? paste0()  # Do not forget the parenthesis
```

> What happened ?

Functions

Function perform specific task on objects

- e.g. to concatanate strings we use **paste0()**

```
paste0(first, last)
```

```
[1] "DonaldTrump"
```

- Functions take **arguments** and return an object called **result**
- To know the arguments use ?

```
 ? paste0()  # Do not forget the parenthesis
```

> What happened ?

- Can go directly to Help panel and type function name

Help

paste {base} R Documentation

Concatenate Strings

Description

Concatenate vectors after converting to character.

Usage

```
paste (..., sep = " ", collapse = NULL)
paste0(..., collapse = NULL)
```

Arguments

`...`	one or more `R` objects, to be converted to character vectors.
`sep`	a character string to separate the terms. Not `NA_character_`.
`collapse`	an optional character string to separate the results. Not `NA_character_`.

Details

`paste` converts its arguments (*via* `as.character`) to character strings, and concatenates them (separating them by the string given by `sep`). If the arguments are vectors, they are concatenated term-by-term to give a character vector result. Vector arguments are recycled as needed, with zero-length arguments being recycled to `""`.

Note that `paste()` coerces `NA_character_`, the character missing value, to `"NA"` which may seem undesirable, e.g., when pasting two character vectors, or very desirable, e.g. in `paste("the value of p is ", p)`.

`paste0(..., collapse)` is equivalent to `paste(..., sep = "", collapse)`, slightly more efficiently.

If a value is specified for `collapse`, the values in the result are then concatenated into a single string, with the elements being separated by the value of `collapse`.

Help

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

We would like to write "Donald Trump" but we have :

```
paste0(first, last)
```

```
[1] "DonaldTrump"
```

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

Getting what you want

We would like to write "Donald Trump" but we have :

```
paste0(first, last)
```

```
[1] "DonaldTrump"
```

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

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

```
[1] "Donald Trump"
```

Functions

Write your own function

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

Write your own function

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

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

```
[1] 30
```

> If you write 3 times the same piece of code write a function...

End of lecture one

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

```
x ← 1:100
sum(x)
```

```
[1] 5050
```

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

```
x ← 1:100
sum(x)
```

```
[1] 5050
```

- Normal distribution

```
y ← rnorm(10, mean = 0, sd = 1)
y
```

```
 [1]  0.4885882 -0.6260146 -0.8855401 -1.2341267  0.3726551  0.8956950
 [7]  0.9124247  0.1755346  0.4628793 -1.5012981
```

Statistics

```
mean(y)
```

```
[1] 0.01007783
```

```
sd(y)
```

```
[1] 0.8875528
```

Statistics

```
mean(y)
```

```
[1] 0.01007783
```

```
sd(y)
```

```
[1] 0.8875528
```

Sample more points... 10,000 instead of 100

```
y ← rnorm(10000, mean = 0, sd = 1)
mean(y)
```

```
[1] 0.01251073
```

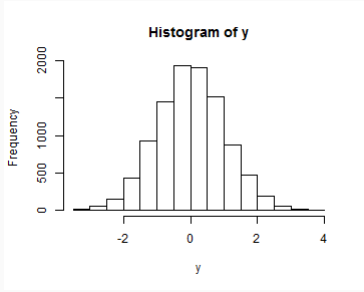
```
sd(y)
```

```
[1] 1.009886
```

Plot

Histogram

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

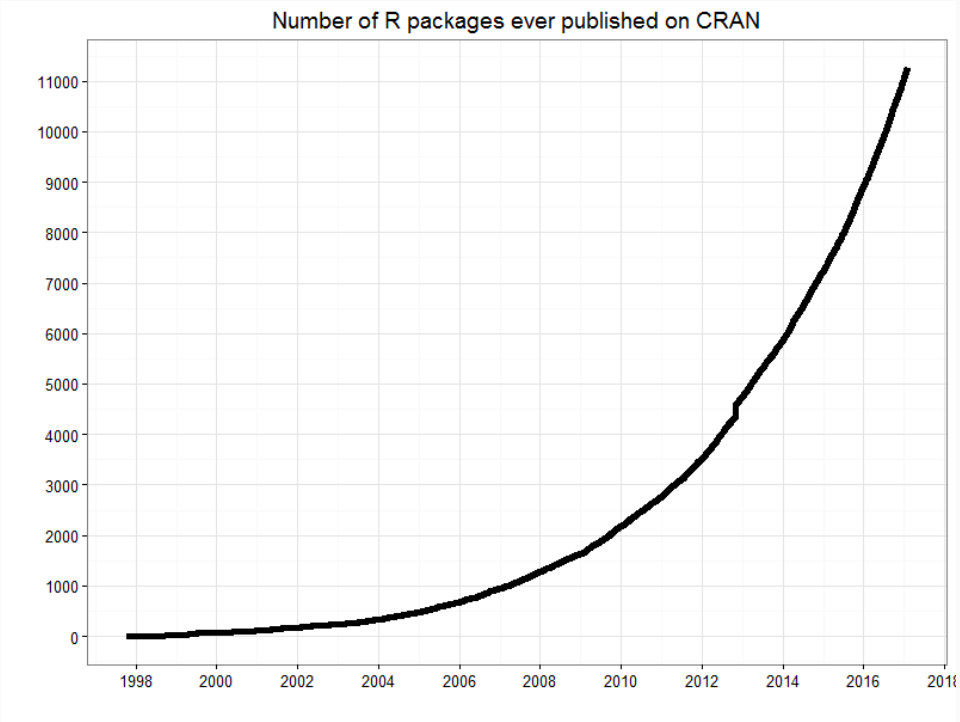


- What is this "library()"

Packages

Packages are set of functions that have a common goal

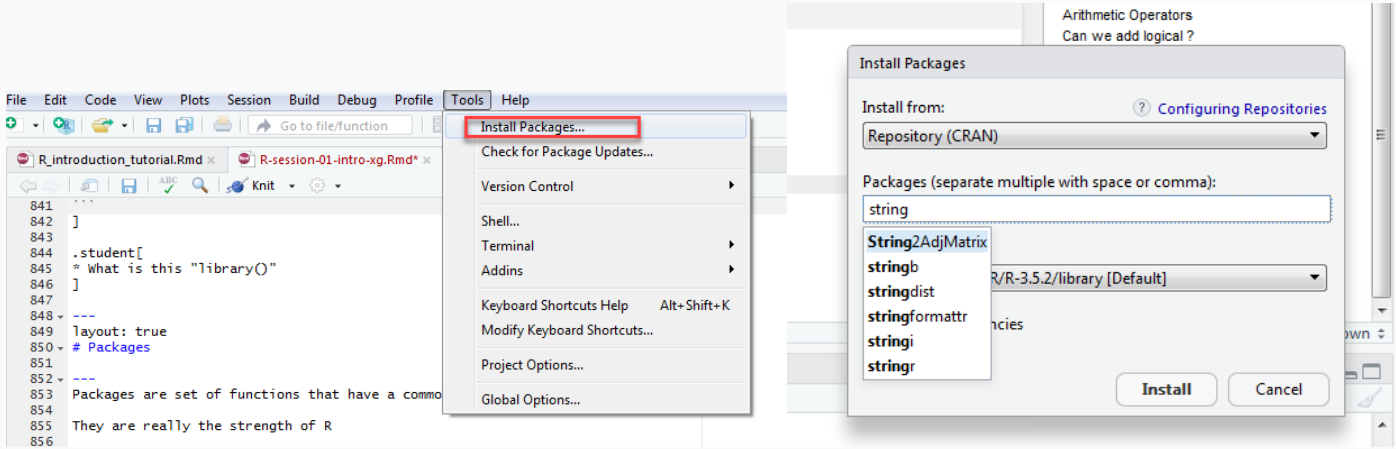
They are really the strength of R



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] "Donald Trump"
```

Using a package

To use functions from the package

- use the syntax `package::function`

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

```
[1] "Donald Trump"
```

- load the package with the library function

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

```
[1] "Donald Trump"
```

Using a package

To use functions from the package

- use the syntax `package::function`

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

```
[1] "Donald Trump"
```

- load the package with the library function

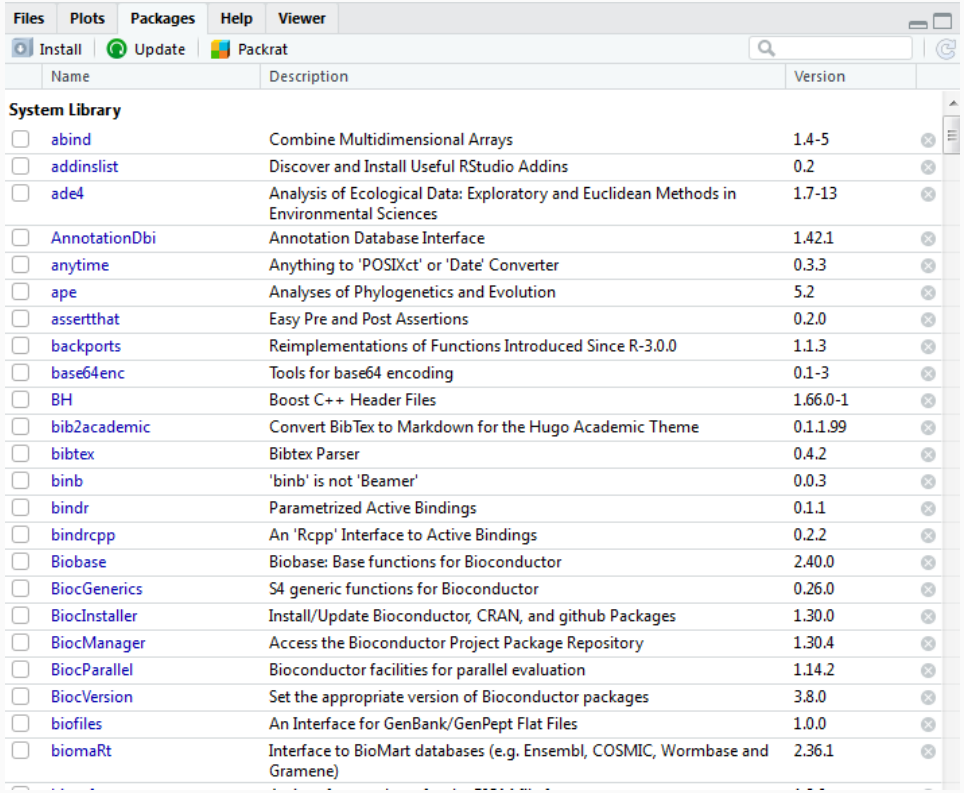
```
library(stringr)
str_c(first, last, sep = " ")
```

```
[1] "Donald Trump"
```

> Sometimes functions from different libraries have similar names

Packages

List installed packages



Other objects

- List
- Matrix
- Factors
- **Data frames**

Data frames

What is it ?

- Table mixing different types of columns (an Excel table...)
- However within a column all values are similar, e.g. numeric, logical, character

Data frames

What is it ?

- Table mixing different types of columns (an Excel table...)
- However within a column all values are similar, e.g. numeric, logical, character

```
df ← data.frame(label = letters[1:6], id = 1:6, value = rnorm(6, mean = 0,
    sd = 1), flag = c(TRUE, FALSE), stringsAsFactors = FALSE)
df
```

```
  label id      value  flag
1     a  1  0.8002749  TRUE
2     b  2 -0.1723698 FALSE
3     c  3  1.0188527  TRUE
4     d  4 -1.4748408 FALSE
5     e  5  0.5381787  TRUE
6     f  6  0.8350807 FALSE
```

- We will not get into the factor at this time, why we set `stringsAsFactors = FALSE`
- Notice the recycling rule ?

Data frames

Useful functions

```
dim(df)  # returns the dimensions of data frame
```

```
[1] 6 4
```

```
nrow(df)  # number of rows
```

```
[1] 6
```

```
ncol(df)  # number of columns
```

```
[1] 4
```

Data frames

Useful functions

```
str(df)  # structure of data frame - name, type and preview of data in each column
```

```
'data.frame':    6 obs. of  4 variables:
 $ label: chr  "a" "b" "c" "d" ...
 $ id   : int  1 2 3 4 5 6
 $ value: num  0.8 -0.172 1.019 -1.475 0.538 ...
 $ flag : logi  TRUE FALSE TRUE FALSE TRUE FALSE
```

```
colnames(df)  # columns names
```

```
[1] "label" "id"    "value" "flag"
```

Access specific column

- Use `$` notation

```
df$value
```

```
[1]  0.8002749 -0.1723698  1.0188527 -1.4748408  0.5381787  0.8350807
```

Access specific column

- Use `$` notation

```
df$value
```

```
[1]  0.8002749 -0.1723698  1.0188527 -1.4748408  0.5381787  0.8350807
```

- Use the `df[i,j]` notation

```
df[, 3]
```

```
[1]  0.8002749 -0.1723698  1.0188527 -1.4748408  0.5381787  0.8350807
```

```
df[, "value"]
```

```
[1]  0.8002749 -0.1723698  1.0188527 -1.4748408  0.5381787  0.8350807
```

Data frames

Access specific value

- Use `$` notation

```
df$label[5]
```

```
[1] "e"
```

```
df$label[1:5]
```

```
[1] "a" "b" "c" "d" "e"
```

Access specific value

- Use `$` notation

```
df$label[5]
```

```
[1] "e"
```

```
df$label[1:5]
```

```
[1] "a" "b" "c" "d" "e"
```

- Use the `df[i,j]` notation

```
df[5, 1]
```

```
[1] "e"
```

```
df[1:5, "value"]
```

```
[1]  0.8002749 -0.1723698  1.0188527 -1.4748408  0.5381787
```

Filter the data

```
df[df$id ⩽ 3, ]
```

```
  label id      value  flag
1     a  1  0.8002749  TRUE
2     b  2 -0.1723698 FALSE
3     c  3  1.0188527  TRUE
```

Select lines for which the label is c

Filter the data

```
df[df$id ⩽ 3, ]
```

```
  label id      value  flag
1     a  1  0.8002749  TRUE
2     b  2 -0.1723698 FALSE
3     c  3  1.0188527  TRUE
```

Select lines for which the label is c

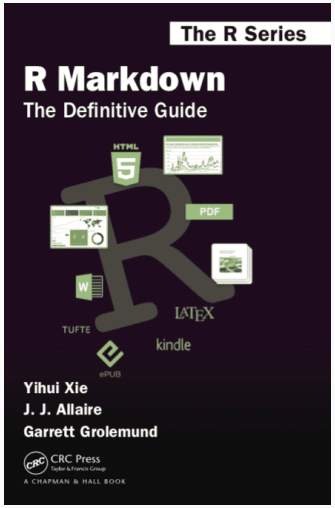
```
df[df$label == "c", ]
```

```
  label id    value flag
3     c  3 1.018853 TRUE
```

Next time: Markdown

What you will learn :

- Mix text, R code and R output in a single document
- Produce documents as HTML, pdf or even Word from the same template



- Please install the following packages and their dependencies
  - rmarkdown (will install also knitr)
  - tinytex (Latex)
- Read the installation instruction : https://bookdown.org/yihui/rmarkdown/installation.html
- Have a look at the book https://bookdown.org/yihui/rmarkdown