Introduction to R

PROGRAMMING LANGUAGE FOR STATISTICAL COMPUTING AND GRAPHICS

VINCENT GARDEUX

BART DEPLANCKE

A bit of history

John M. Chambers

S programming language developed for data analysis

Ross Ihaka

R initial development as a mix between S and Scheme

Robert C. Gentleman

Bell Laboratories

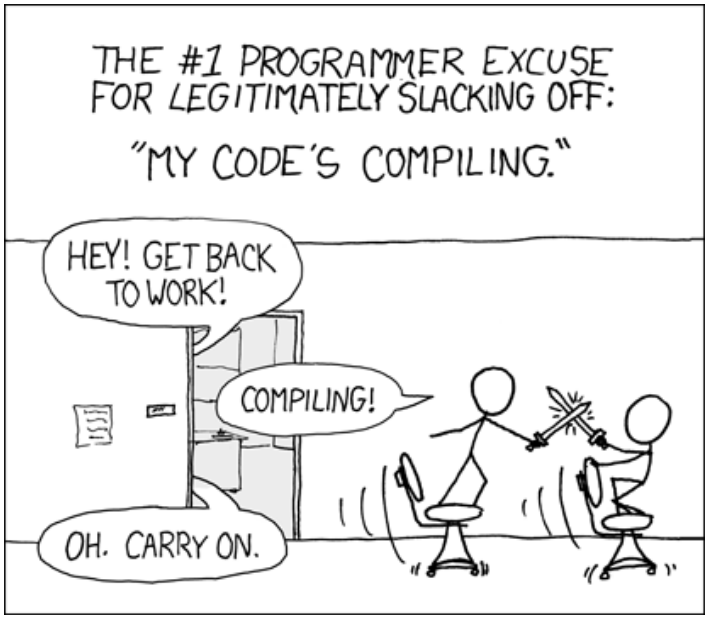
1976 — **S2** *"S-plus"* (1988) — **S3** (1992) — **S4** (1998) — **Sold to** ***Insightful corporation*** (2008)

R — **v1.0** (1995) — **v2.0** (2000) — **v3.0** (2004) — **R v. 3.5.1 (Feather Spray)** (2013) — July 2018

Why learning R?

- Contrary to SAS or SPSS, it is not only a statistical environment but a **complete language**
- It is not compiled such as C/C++ but **interpreted** (typically command-line interpreter)
- 14th language at TIOBE index

- Open source
- Advanced statistical language
- High quality graphical tools
- Large collection of packages



Where to find R resources?

R WEBSITES

- www.r-project.org 

Main project page & news

- cran.r-project.org The Comprehensive R Archive Network

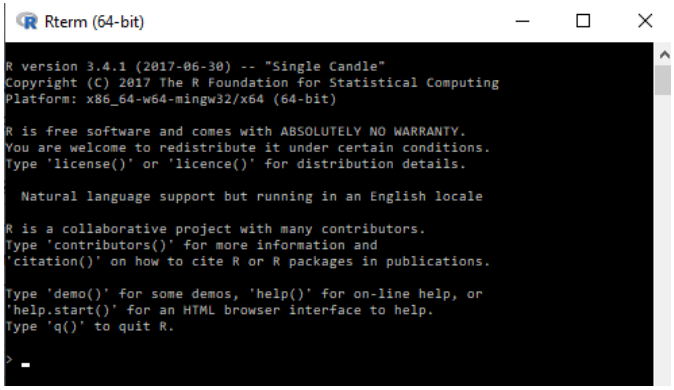
Download area for the R software itself, but also for extension packages

- *www.bioconductor.org* 

An extended series of packages, acting as an alternate of CRAN for bioinformatics tools (for analysis and comprehension of high-throughput genomic data)

How to set up your working environment?

R INTERFACE & IDE



- Default R interface

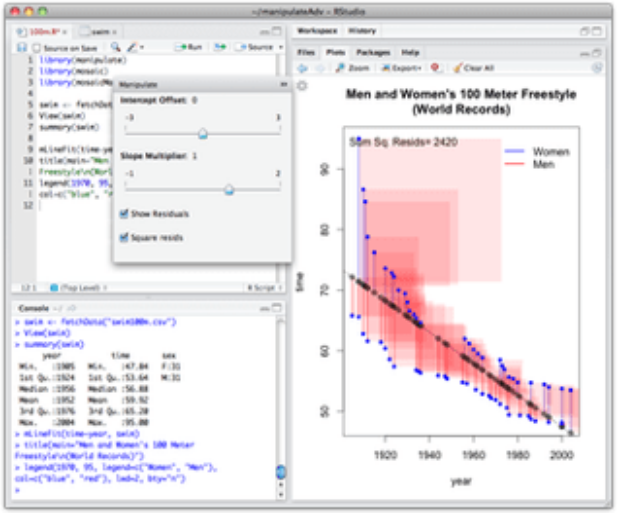
Directly available when installing R

- Text editor + Rscript

Use your favorite text editor (Notepad++, emacs, etc…) and run it with 'Rscript' tool (directly available when installing R)

- Advanced IDEs (**Rstudio**, …)

www.rstudio.org



Rstudio developed ***Shiny*** for web-based interactive visualization of data

Getting started

- Start by installing the last version of R for your machine (64 bits) from cran.r-project.org

- Install Rstudio Desktop (free version, 64 bits) from www.rstudio.com

- Run Rstudio and write directly in the terminal a simple command to check that both programs are correctly linked

```
> 2 + 2
[1] 4
```

- Check if you can install and load a package from CRAN (e.g. by installing the 'ggplot2' package)

```
> install.packages("ggplot2")
> require(ggplot2)
```

Basic R types and operations

1. Perform simple operations: `+ - / * ^ log() sqrt() exp()`

a. 2+2 **b.** 100*100 + 1/2 **c.** (100*100 + 1)/2 **d.** 2^10 **e.** log(2)

2. Vectors (ordered collection of data of same type): `c() x:y seq() rep()`

a. c(1,2,3,4,5) **b.** 1:100 **c.** c(1,1,2,2) * 3 **d.** 100:1 **e.** seq(1,100,5)

3. Operations/Functions on vectors: `sum() mean() median() sd() + - * / ^ log ...`

a. sum(1:100) **b.** mean(1:100000) **c.** length(seq(1,100,5))

4. Matrices & operations: `matrix() rowSums() colSums() + - * ^ log ...`

a. matrix(c(15,10,1,5), nrow=2)

b. rowSums(matrix(c(15,10,1,5), nrow=2))

c. dim(matrix(c(15,10,1,5), nrow=2))

5. Assignment / Variables: `<- =`

a.
```
tmpVal <- log(2)
exp(tmpVal)
## [1] 2
```

b.
```
median(x = 1:10)
x
## Error: object 'x' not found
```

c.
```
median(x <- 1:10)
x
## [1] 1 2 3 4 5 6 7 8 9 10
```

d.
```
a <- matrix(c(1,2,3,4), nrow=2)
a[,1]
a[2,2]
a[2,]
a[1,] + c(1,2)
```

Note: You can always ask some help for a specific function using the '?' symbol. e.g. ?median

Additional data structures: Lists

In addition to vectors and matrices, more complex data structures can be created in R

LISTS

- Collection of data of possibly different types

```
doe <- list(name="John", age=32, gender=factor("M", levels=c("M", "F")), married=F)
```

Character Numeric Factor with 2 levels Logic/Boolean

- Fields can be accessed using the '$' symbol

```
doe$name
## [1] "John"
```

Data types

As you may have noticed, variable types are automatically set when the variable is created. This may lead to issues when wrongly assigned.

TYPES

- To get a variable type

```
class(doe$age)
## [1] "numeric"
```

- Or the types/components of a structure

```
str(doe)
## List of 4
##  $ name : chr "John"
##  $ age : num 32
##  $ gender : Factor w/ 2 levels "M","F": 1
##  $ married: logi FALSE
```

- To change a variable type (when possible)

```
doe$age = as.character(doe$age)
class(doe$age)
## [1] "character"
```

Additional data structures: data.frame

This is probably the most used data structure in bioinformatics. It is supposed to represent the typical data table that researchers come up with – like a spreadsheet

name	age	gender	married
John	32	M	F
Alison	42	F	T
Emma	22	F	F

It has rows and columns, data within each column has the same type.

```
my.data <- data.frame(name=c("John","Alison","Emma"),
              age=c(32, 42, 22),
              gender=factor(c("M","F","F"), levels=c("M", "F")),
              married=c(F,T,F))
```

Rows & columns are accessible like in a matrix (or by name in rownames/colnames)

```
my.data[1,]
my.data[,"gender"]
```

In addition, columns can be accessed/assigned using the '$' symbol

```
my.data$name
```

Accessing filesystem / Reading and Writing

FILESYSTEM

- getwd() and setwd() to change the working directory

```
getwd()
## [1] "C:/"
setwd("C:/Users/toto/Desktop")
```

- Listing files

```
dir()
## [1] "toto.txt" "tutu.tab"
dir(pattern=".txt")
## [1] "toto.txt"
```

- Writing a variable content in a text file

```
write.table(my.data, file="my.data.txt", sep="\t", quote=F, row.names=T)
```

- or in .RData binary format (not readable outside of R)

```
save(my.data, file="my.data.RData")
```

- Reading a text file, and put the content in a data.frame

```
my.data = read.table("my.data.txt", sep="\t", row.names=1, header=T)
```

- Load a .RData file

```
load("my.data.RData")
```

Note: There is a faster package available for reading (fread) and writing (fwrite) text files (specifically for large files):
> install.packages("data.table")

Conditional structures / subsetting

Conditional structures are similar to other programming languages.

Here we will use the *iris* dataset, directly available in R

```
iris
    Sepal.Length Sepal.Width  Petal.Length Petal.Width  Species
1   5.1          3.5          1.4          0.2          setosa
…
```

CONDITIONS

- if…else

```
if("setosa" %in% iris$Species){
    print(sum(iris$Species == "setosa"))
} else {
    print("Nope")
}
## 50
```

This is comparison
Do not use '=' for comparison

Questions V
1. What is this script doing?
2. How?
If needed, decompose it to understand it

- You can also use conditions to subset a data.frame

```
sub1 <- iris[iris$Species == "setosa", ]
sub2 <- iris[iris$Sepal.Width > 4, ]
```

- Works also with the *subset* function

```
sub1 <- subset(iris, Species == "setosa")
sub2 <- subset(iris, Sepal.Width > 4)
```

Loops

As any other programming language 'for' and 'while' loops are available.

FOR / WHILE

- Flexible way of writing code.

```
for(i in 1:20){
    print(i)
}
```

```
i = 1
while(i <= 20){
    print(i)
    i <- i + 1
}
```

- You can also iterate on a dataset rows/columns

```
cnt = 0
for(i in 1:nrow(iris)){
    if(iris[i, "Sepal.Length"] > 7){
        cnt <- cnt + 1
    }
}
cnt
## [1] 12
```

Functions

You can write your own functions in R, similarly to other programming languages. It is convenient for i) minimizing the code, ii) call the same function in other programs, iii) using the apply function

FUNCTIONS

- There is no disctinction in R between procedure and functions, the syntax is:

```
my.function.sub = function(x, y){
    return(x - y)
}
```

name

arguments/parameters

```
my.function.sub = function(x, y){
    s <- x - y
    s
}
```

if no *return*, it returns the last value

- You can then call your function with different parameters:

```
my.function.sub(2, -1)
## [1] 3

my.function.sub(2, -1, 9)
## Error
```

```
my.function.sub(2)
## [1] Error

my.function.sub(y=2, x=-1)
## -3
```

parameters can be ordered as needed, as soon as their name is used

Graphics / Plots

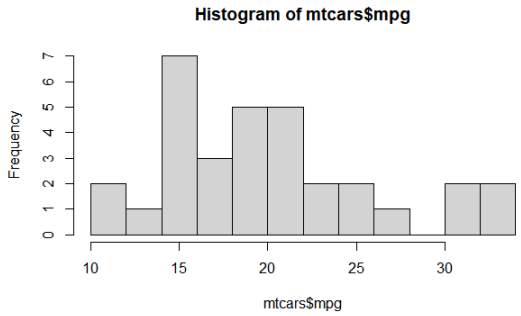
R provides inherently comprehensive graphics tools for visualizing and exploring scientific data (*graphics* base package)

Several powerful graphic packages extend these methods such as *ggplot2* but are slightly more complicated to use.

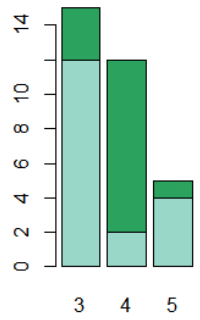
Here we will use the *mtcars* dataset, directly available in R

GRAPHICS

```
hist(mtcars$mpg, breaks=10, col="lightgrey")
```




```
barplot(table(mtcars$vs, mtcars$gear), col=c("#99d8c9", "#2ca25f"))
```

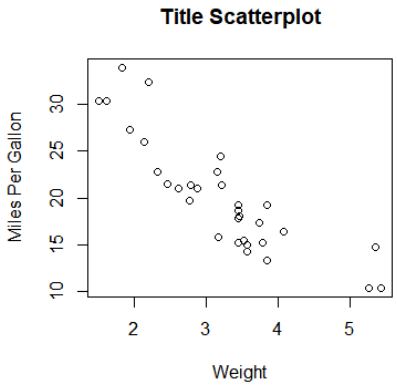


Note: Most of these plotting functions are explained in details here.

Graphics / Plots

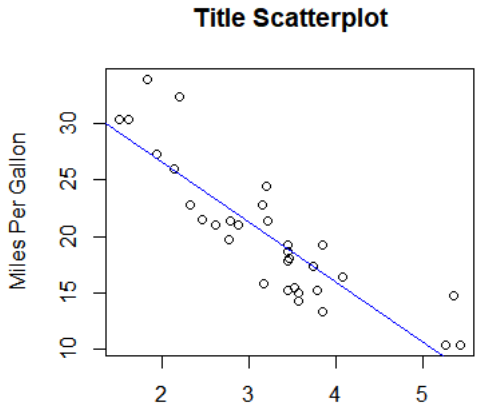
There are many options to add colors/text/labels/legends on the plots.

```
plot(mtcars$wt, mtcars$mpg, main="Title Scatterplot", xlab="Weight",
    ylab="Miles Per Gallon")
```



You can then perform a linear regression and add the regressed line to the existing plot.

```
regression.model = lm(mpg~wt, data=mtcars)
abline(regression.model, col="blue")
```



Note: You can save a plot directly to a .png or .pdf using the png() and pdf() functions

For additional R training

MOOCS

- ❖ **https://www.datacamp.com/courses/free-introduction-to-r**
- https://www.datacamp.com/courses/intermediate-r-practice
- … and others on datacamp.com (https://www.datacamp.com/tracks/r-programming)

- https://www.coursera.org/learn/r-programming/home/welcome

BOOKS / TUTOS

- https://cran.r-project.org/doc/contrib/Seefeld_StatsRBio.pdf (a bit old)
- ❖ **https://biostats.w.uib.no/tutorials/introduction-to-r/**
- https://www.r-project.org/doc/bib/R-books.html
- https://www.statmethods.net/r-tutorial/index.html

What a real programmer is:
https://www.youtube.com/watch?v=HluANRwPyNo

⇒ Check Google/Forums if you have any doubt or you are looking for a way to do something