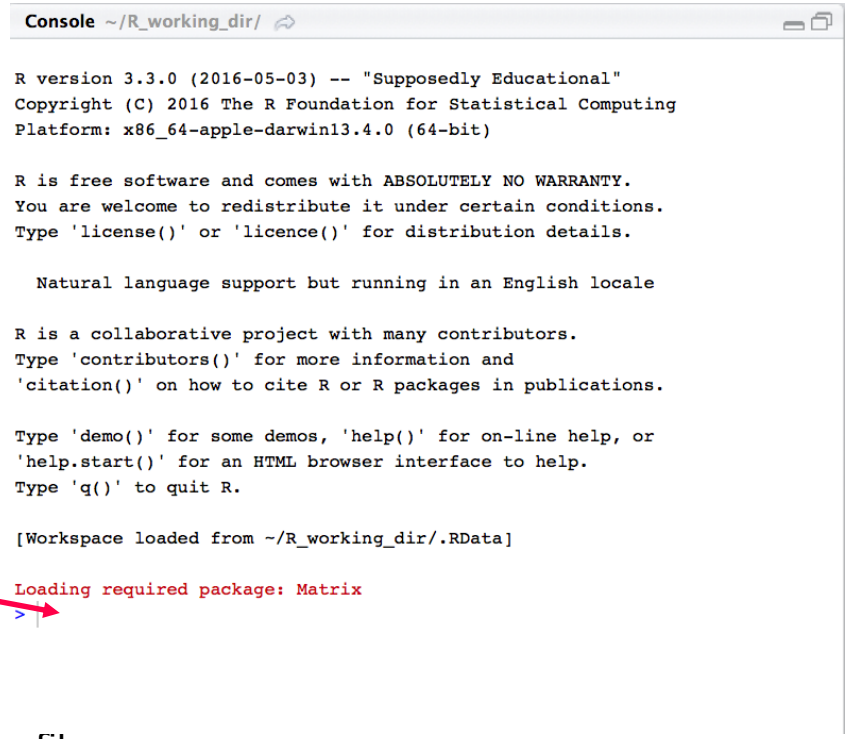
R: Basics

Andrea Passarella

(plus some additions by Salvatore Ruggieri)

Basic Concepts

- R is an interpreted scripting language
- Types of interactions
  - Console based
    - Input commands into the console
    - Examine results
  - Scripting
    - Sequence of statements in a text file
    - Use the "`source()`" command to process the file
    - Equivalent to provide the sequence of statements to the console
- How we will use it
  - Variables to store data
  - Functions (either existing in the packages or new ones written on purpose) to process data
  - (Limited) I/O with external files for
    - Input/output of data
    - scripting



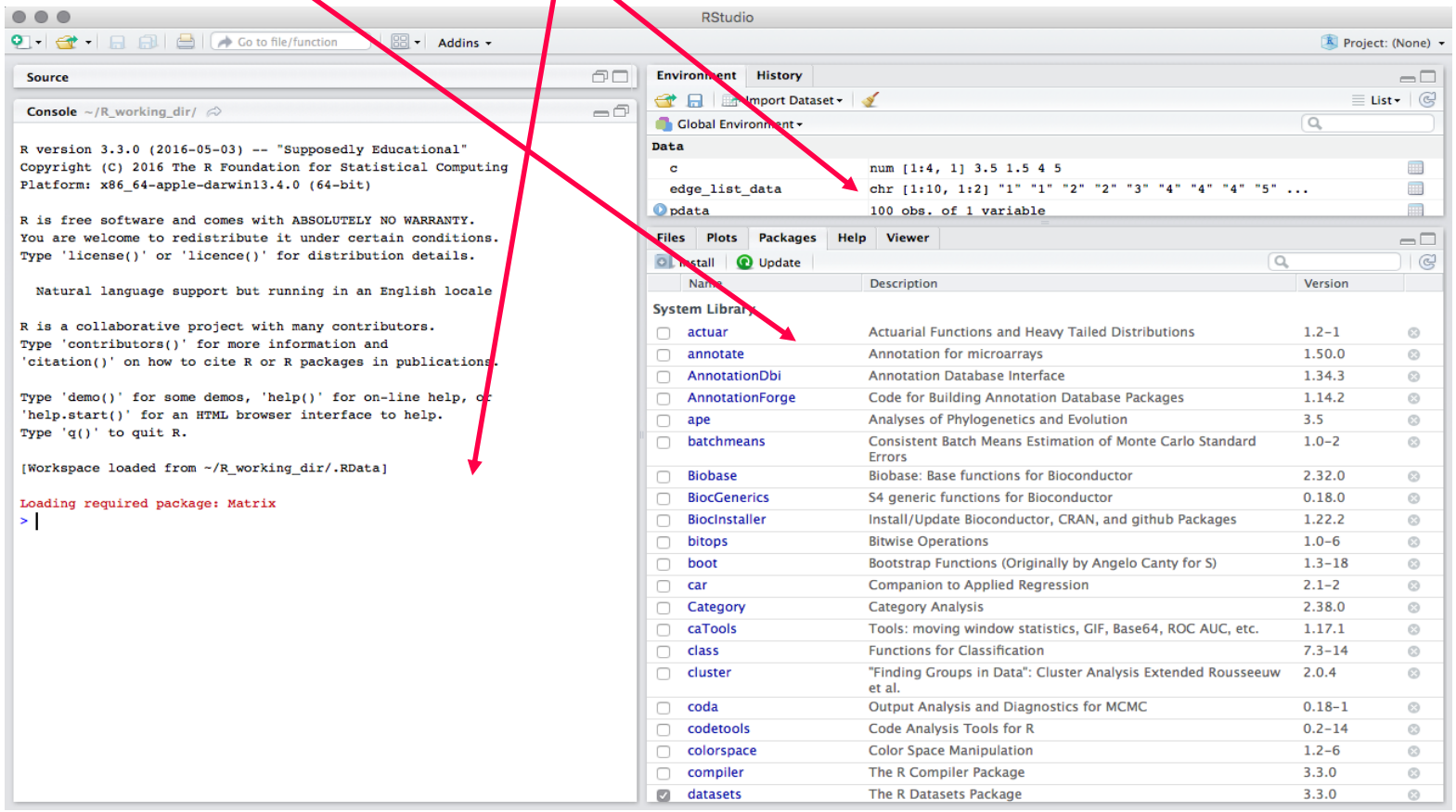
LAUNCH, HELP, SAVE, EXIT

- Launching the "R" application means running the interpreter shell

```
18:39 andrea R $ R

R version 3.3.0 (2016-05-03) -- "Supposedly Educational"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

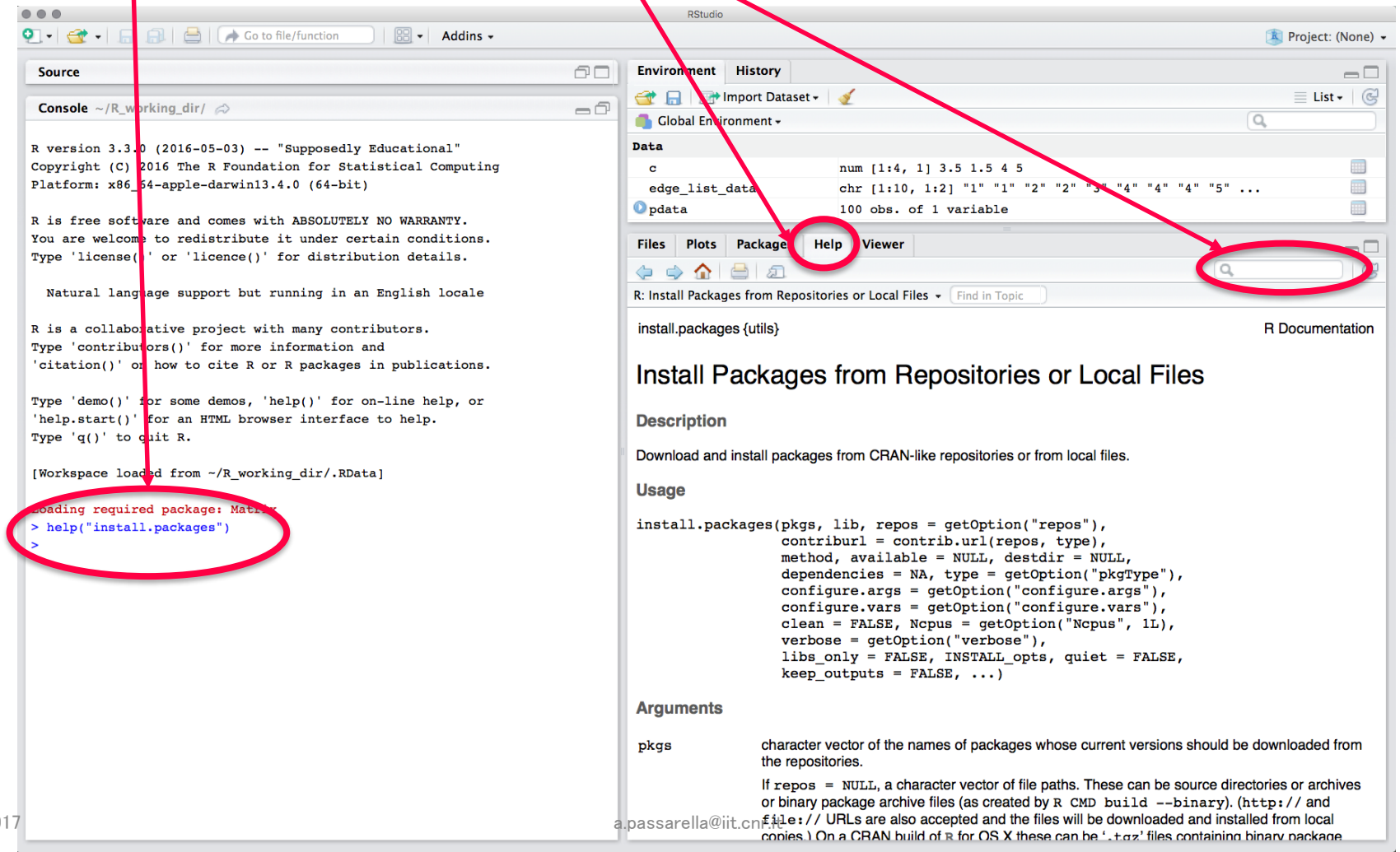
LAUNCH, HELP, SAVE, EXIT

- RStudio is a front-end to the language
  - Embeds the interpreter shell (Console)
  - Visualisation of available variables
  - Package installation
  - Help



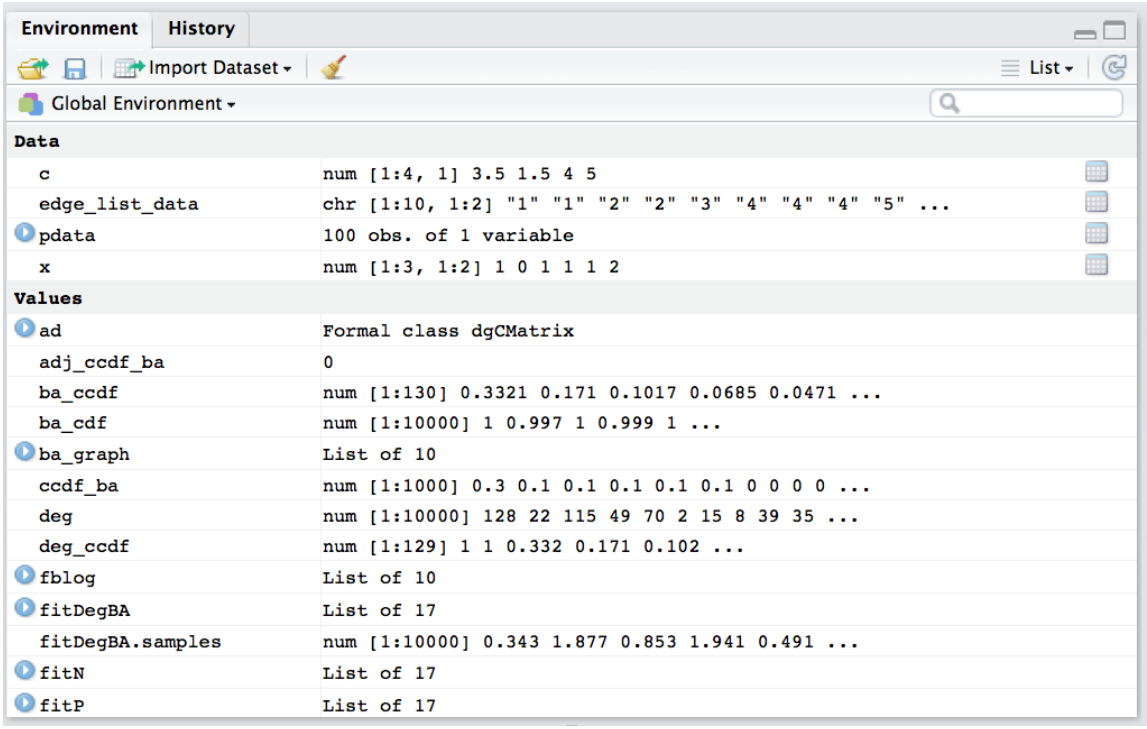
LAUNCH, HELP, SAVE, EXIT

- Help
  - Search with the user interface
  - `help()` function from the console



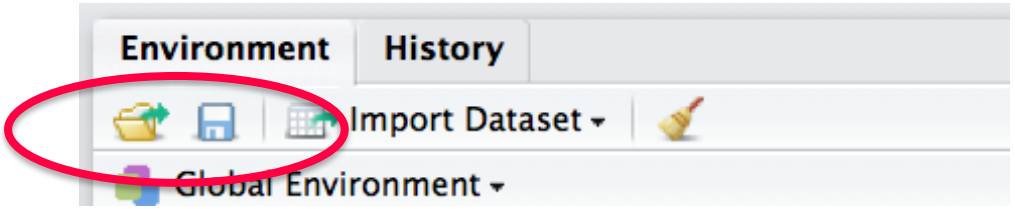
LAUNCH, HELP, SAVE, EXIT

- Workspace = set of data, function, … defined during a session
- The elements of the workspace are shown in the "Environment" pane
- or can be listed with `ls()` from the console



LAUNCH, HELP, SAVE, EXIT

- Workspaces can be saved and restored from previous sessions
  - Either through the UI in RStudio



  - or via `save.image()` and `load()` functions from the R console

```
>
> load("~/R_working_dir/Untitled.RData")
> save.image("~/R_working_dir/Untitled.RData")
>
```

- Automatic actions (upon running/exiting from R/RStudio)
  - Load workspace from a file ".RData" in the working directory upon launch
  - Ask to save to ".RData" in the working directory upon exiting

Loading Datasets

- Function `data()` list the set of available dataset provided by the currently loaded packages
- `data(pippo)` loads data from pippo (the name of the dataset) in the current workspace
  - a variable (a dataframe, see later) called pippo is added to the workspace
- Depending on the dataset format, it might be needed to access the dataframe to "expand" it
  - E.g., `ls(pippo)`

VARIABLES

- Defined as they are needed
- Assignment operator, <−, or =
  - `a = 15` defines variable a, with value 15
    - From then on, `a` becomes available in the workspace
- Looking into variables
  - Type the name in the console

```
> a = 15
> a
[1] 15
>
```

- `summary(variable_name)` shows a summary, which depends on the type of the variable
  - e.g., if `p` is a set of values, `summary(p)` shows some reference percentiles of these values

```
> p
 [1] 0.4970180 0.2177386 0.1030616 0.4776593 0.7038415 0.9508472
 [7] 0.3151198 0.7208926 0.4440492 0.6947185
> summary(p)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1031  0.3474  0.4873  0.5125  0.7016  0.9508
>
```

VECTORS, ARRAYS

- Vectors are the most basic structure in R
  - a collection of values of the same type

Function `c()`, returns a collection of the arguments

```
> a = c(1,5,10)
> a
[1]  1  5 10
> b = c("mela", "pera", "albicocca", 5)
> b
[1] "mela"      "pera"      "albicocca" "5"
>
```

`a` is a vector of integers

`b` is a vector of character strings

- note the difference between
  - 5 in a
  - “5” in b

VECTORS, ARRAYS

- Arrays are vectors with given dimensions

```
> a = c(1,2,3,4,5,6,7,8,9,10)
> a
 [1]  1  2  3  4  5  6  7  8  9 10
> dim(a)
NULL
> dim(a)=10
> dim(a)
[1] 10
> a
 [1]  1  2  3  4  5  6  7  8  9 10
> dim(a)=c(2,5)
> a
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

Collection of values without any specific dimension attribute

Now gets a single dimension

2 dimensions, 2 rows, 5 columns

VECTORS, ARRAYS

- Arrays can be created more simply with `array()`

```
> a
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> b = array(seq(1:10), dim=c(2,5))
> b
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

`seq()` generates a sequence of values between the given extremes

`dim` parameter of the function to set the dimensions

- Matrices are arrays with 2 dimensions only
  - Note that arrays can have more than 2 dimensions

```
> c = matrix(seq(1:10),nrow = 2,ncol = 5)
> c
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

ACCESSING VECTOR/ARRAY ELEMENTS

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

- The `[]` operator
  - Start counting from 1, not from 0!

```
> c[1,3]
[1] 5
```
← Element with index (1,3)

```
> c[1,]
[1] 1 3 5 7 9
```
← All elements of the first row

```
> c[,2]
[1] 3 4
```
← All elements of the second column

NB: `c[,2]` is itself a vector, thus one can further index it

```
> c[,2][1]
[1] 3
```
← First element of c[,2] (equivalent to c[1,2])

ACCESSING VECTOR/ARRAY ELEMENTS

- Negative indices
  - `c[,-2]`: c with all columns but 2
  - In general, negative indices are excluded, e.g. `c[,c(-1;-3)]`

variable `c` combination function `c()`

```
> c[,-2]
     [,1] [,2] [,3] [,4]
[1,]    1    5    7    9
[2,]    2    6    8   10
> c[,c(-1,-3)]
     [,1] [,2] [,3]
[1,]    3    7    9
[2,]    4    8   10
>
```

- Range indices
  - `c[,2:4]`: all columns of matrix c between 2 and 4

```
> c[,2:4]
     [,1] [,2] [,3]
[1,]    3    5    7
[2,]    4    6    8
```

- Expressions as indices
  - `c[c>5]`: all values greater than 5
  - `c[c>5 & c<10]`: all values between 5 and 10
    - return value is a vector

```
> c[c>5]
[1]  6  7  8  9 10

> c[c>5 & c<10]
[1] 6 7 8 9
```

Logical Operators

- Standard set of operators of any programming language
  - ! Unary not
  - < Less than, binary
  - > Greater than, binary
  - == Equal to, binary
  - >= Greater than or equal to, binary
  - <= Less than or equal to, binary
  - & And, binary, vectorized
  - && And, binary, not vectorized
  - | Or, binary, vectorized
  - || Or, binary, not vectorized

Logical Operators: Vectorised vs Non-vectorised

- `c[c>5 & c<10]`: all values between 5 and 10
- Steps
  - `c>5`: a matrix of the same dimensions of c, with TRUE or FALSE values
  - `c<10`
  - `c>5 & c<10`: a matrix of the same dimensions of c, with the logical AND of the two expressions
  - `c[c>5 & c<10]`: select from c only the elements for which the indices are TRUE

```
> c[c>5 & c<10]
[1] 6 7 8 9
```

```
> c>5
      [,1]  [,2]  [,3] [,4] [,5]
[1,] FALSE FALSE FALSE TRUE TRUE
[2,] FALSE FALSE  TRUE TRUE TRUE
```

```
> c<10
     [,1] [,2] [,3] [,4]  [,5]
[1,] TRUE TRUE TRUE TRUE  TRUE
[2,] TRUE TRUE TRUE TRUE FALSE
```

```
> c>5 & c<10
      [,1]  [,2]  [,3] [,4]  [,5]
[1,] FALSE FALSE FALSE TRUE  TRUE
[2,] FALSE FALSE  TRUE TRUE FALSE
```

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

Logical Operators: Vectorised vs Non-vectorised

- `c[c>5 & c<10]`: all values between 5 and 10

```
> c[c>5 & c<10]
[1] 6 7 8 9
```

- Steps
  - `c>5`: a matrix of the same dimensions of c, with TRUE or FALSE values

```
> c>5
      [,1]  [,2]  [,3] [,4] [,5]
[1,] FALSE FALSE FALSE TRUE TRUE
[2,] FALSE FALSE  TRUE TRUE TRUE
```

  - `c<10`

```
> c<10
     [,1] [,2] [,3] [,4]  [,5]
[1,] TRUE TRUE TRUE TRUE  TRUE
[2,] TRUE TRUE TRUE TRUE FALSE
```

  - `c>5 & c<10`: a matrix of the same dimensions of c, with the logical AND of the two expressions
    - We need to do the logical AND on an element-by-element of the two matrices
    - This is obtained with the vectorised version of the operator, "&"

```
> c>5 & c<10
      [,1]  [,2]  [,3] [,4]  [,5]
[1,] FALSE FALSE FALSE TRUE  TRUE
[2,] FALSE FALSE  TRUE TRUE FALSE
```

  - `c>5 && c<10`: non-vectorised version
    - Applicable to single-element data
    - In case of vectors stops at the first element
    - Typically used for indices in control statements and loops

```
> c>5 && c<10
[1] FALSE
```

BUILDING MATRICES

- Sometimes useful to build matrices by stitching together existing arrays or matrices
  - `cbind()` joins together vectors/matrices by column
  - `rbind()` joins together vectors/matrices by row

```
> d = c(11,12)
> c = cbind(c,d,deparse.level = 0)
> c
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    5    7    9   11
[2,]    2    4    6    8   10   12
```

Do not assign names to columns

previous matrix `c`

vector `d`

LISTS, DATA FRAMES

- Lists are collections of arbitrary data types

```
> Lst <- list(name="Fred", wife="Mary",
+             no.children=3, child.ages=c(4,7,9))
> Lst
$name
[1] "Fred"

$wife
[1] "Mary"

$no.children
[1] 3

$child.ages
[1] 4 7 9
```

character string

integer

vector of 3 elements

```
> length(Lst$name)
[1] 1
> length(Lst$wife)
[1] 1
> length(Lst$no.children)
[1] 1
> length(Lst$child.ages)
[1] 3
```

Function `length()`

- size of the variable
- different from `dim()`

```
> d = c(11,12)
> length(d)
[1] 2
> dim(d)
NULL
```

LISTS, DATA FRAMES

- Data frames
  - lists whose components are all of the same length
  - If components are seen as columns of a matrix, all columns must have the same size
  - With respect to matrices, columns can be of different types

```
> name = c("Pietro", "Paolo", "Antonio")
> city = c("Pisa", "Pisa", "Ancona")
> age = c(25, 15, 34)
> df = data.frame(name,city,age)
> df
     name   city age
1  Pietro   Pisa  25
2   Paolo   Pisa  15
3 Antonio Ancona  34
```

Note the difference with the definition of `Lst`!

```
> df = data.frame(name="Fred", wife="Mary",
+              no.children=3, child.ages=c(4,7,9))
> df
  name wife no.children child.ages
1 Fred Mary           3          4
2 Fred Mary           3          7
3 Fred Mary           3          9
```

Accessing Elements of Lists and DataFrames

- `$` or `[[]]` operator
  - Selection of elements in a list or data frame
  - Either by position: `df[[1]]`
  - Or by name: `df[["name"]]`, `df$name`

```
> df[[1]]
[1] Pietro  Paolo   Antonio
Levels: Antonio Paolo Pietro
> df[["name"]]
[1] Pietro  Paolo   Antonio
Levels: Antonio Paolo Pietro
> df$name
[1] Pietro  Paolo   Antonio
Levels: Antonio Paolo Pietro
```

- Levels are the unique elements found, if defined

```
> df$city
[1] Pisa   Pisa   Ancona
Levels: Ancona Pisa
```

Adding removing elements from Lists/Data Frames

- Assigning `NULL` to an element drops that element

```
> df
     name   city age
1  Pietro   Pisa  25
2   Paolo   Pisa  15
3 Antonio Ancona  34
```

```
> df$age = NULL
> df
     name   city
1  Pietro   Pisa
2   Paolo   Pisa
3 Antonio Ancona
```

- Create a new element by just assigning values to the name of the new element

```
> df$age = NULL
> df
     name   city
1  Pietro   Pisa
2   Paolo   Pisa
3 Antonio Ancona
```

```
> age
[1] 25 15 34
> df$age = age
> df
     name   city age
1  Pietro   Pisa  25
2   Paolo   Pisa  15
3 Antonio Ancona  34
```

Modifying elements in a List/Data Frame

- `[[]]` or `$` operators return a vector
  - Whose elements can be managed with the normal index operators
    - E.g., `[]`

```
> df
     name   city age
1  Pietro   Pisa  25
2   Paolo   Pisa  15
3 Antonio Ancona  34
```

```
> df$age[1]
[1] 25
> df$age[1] = 10
> df
     name   city age
1  Pietro   Pisa  10
2   Paolo   Pisa  15
3 Antonio Ancona  34
```

Data Frames as Matrices

- Sometimes it is useful to access Data Frames as matrices

Names of the rows
- Access and modify via `rownames(df)`

```
> df
      name   city age
1   Pietro   Pisa  10
2    Paolo   Pisa  16
3 Antonio Ancona  35
```

Names of the columns
- Access and modify via `colnames(df)`

Matrix part of the data frame

Access and modify via the `[,]` operator

```
> df[,1]
[1] Pietro  Paolo   Antonio
Levels: Antonio Paolo Pietro
> df[2,]
   name city age
2 Paolo Pisa  16

> df[1,3] = 25
> df
      name   city age
1   Pietro   Pisa  25
2    Paolo   Pisa  16
3 Antonio Ancona  35
```

Select people whose age is greater than 16

```
> temp = df$age>16
> temp
[1]  TRUE FALSE  TRUE
> df[temp,]
      name   city age
1   Pietro   Pisa  25
3 Antonio Ancona  35
```

T/F index vectors can also be applied to columns!
- Select only those columns for which the condition is true

```
> df[,temp]
      name age
1   Pietro  25
2    Paolo  16
3 Antonio  35
```

ARITHMETIC OPERATIONS

- With arrays, element-by-element operation

```
> a = array(c(1,2,3))
> a
[1] 1 2 3
> b = array(c(8,9,10))
> b
[1]  8  9 10
```

```
> prod = a*b
> prod
[1]  8 18 30
```

- Same semantic with matrices

```
> c
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> d
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3   10    7    9
[2,]    2    4    6    8   10
```

```
> c*d
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    9   50   49   81
[2,]    4   16   36   64  100
```

- Use "%*%" for the standard matrix product form

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

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

```
> c %*% d
     [,1] [,2]
[1,]  140  220
[2,]  160  260
```

BASIC I/O

- Read values into a vector
  - `scan()` function

File "`sample.txt`"

```
FIRST LINE
1 10 20
30 40 50
```

```
> l = scan("sample.txt", skip=1)
Read 6 items
> l
[1]  1 10 20 30 40 50
```

Initial lines to skip

A path to the file to read
- if relative, the working directory is assumed
- Use `getwd()` for the name of the working directory
- Equivalent to `paste(getwd(),"/sample.txt",sep="")`

By default, elements are separated by white spaces or end-of-line
- can be modified through the `sep` argument

Basic I/O

- Read structured data into data frames
  - `read.table()` function

File "`sample.txt`"

```
first   second  third
1       10      20
30      40      50
```

```
> df_read = read.table("sample.txt",
+                      header = T)
> df_read
  first second third
1     1     10    20
2    30     40    50
```

Whether the first line should be used to get the column names

Writing Data Frames to Files

- `write.table()` function

Data frame to write

Where to write it

```
> write.table(df_read,"out_df.txt",
+             row.names = F,quote=F,sep='\t')
```

Whether to put row names (usually numbers) in front of rows

Whether to put quotes around character strings

Use tab as separator

File "`out_df.txt`"

first	second	third
1	10	20
30	40	50

WRITING VECTORS, LISTS, OR MATRICES

- `write()` function

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

Object to write

Where to write it

```
> write(c, "out_matrix.txt",
+       sep='\t', ncolumns = dim(c)[2])
```

Use tab as separator

Number of columns in the output file

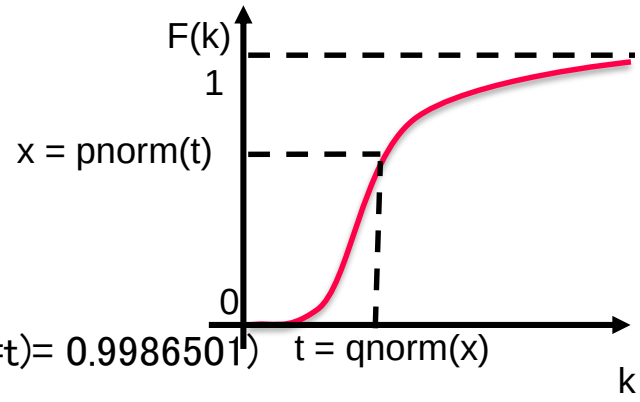
- Here equal to the number of columns of the matrix
  - Same with function `ncol(c)`

File "`out_matrix.txt`"

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

PROBABILITY DISTRIBUTIONS

- R includes a family of functions to manage the most popular distributions
- Given a specific distribution (e.g., normal, named "`norm`" in R)
  - rnorm(100, mean=0, std=1)
    - Generates 100 samples from a normal distribution with mean 0 and standard deviation 1
  - dnorm(3, mean=0, std=1)
    - Density function computed at 3 (f(3))
  - pnorm(3, mean=0, std=1)
    - Distribution function computed at 3 (F(3) = P(X<=3) = 0.9986501)
  - qnorm(0.9986501, mean=0, std=1)
    - Percentile corresponding to 0.9986501 (t s.t. P(X<=t)= 0.9986501)



- Given a set of values in a vector x
  - `mean(x)` gives the average
  - `sd(x)` gives the standard deviation
  - Summary(x) gives a summary of the main percentiles of the distribution

PROBABILITY DISTRIBUTIONS

- Parameters to the p,q,r,d functions depend on the particular distribution

Distribution	R name	additional arguments
beta	`beta`	`shape1, shape2, ncp`
binomial	`binom`	`size, prob`
Cauchy	`cauchy`	`location, scale`
chi-squared	`chisq`	`df, ncp`
exponential	`exp`	`rate`
F	`f`	`df1, df2, ncp`
gamma	`gamma`	`shape, scale`
geometric	`geom`	`prob`
hypergeometric	`hyper`	`m, n, k`
log-normal	`lnorm`	`meanlog, sdlog`
logistic	`logis`	`location, scale`
negative binomial	`nbinom`	`size, prob`
normal	`norm`	`mean, sd`
Poisson	`pois`	`lambda`
signed rank	`signrank`	`n`
Student's t	`t`	`df, ncp`
uniform	`unif`	`min, max`
Weibull	`weibull`	`shape, scale`
Wilcoxon	`wilcox`	`m, n`

CONDITIONAL STATEMENT

- General form
  - If (statement1)
    statement2
    else
    statement3

- Example
  - if (x > 0) {
    count = count+1
    x = x+1
    print(x)
    } else {
    count = count-1
    x = x-1
    print(x)
    }

```
> x = 5
> count = 1
> if (x > 0) {
+     count = count + 1
+     x = x + 1
+     print(x)
+ } else {
+     count = count - 1
+     x = x-1
+     print(x)
+ }
[1] 6
```

Loop Statement

- While loop
  - `while (expression) statement`

```
> x = 5
> count = 2
> while (count > 0) {
+     x = x-1
+     count = count-1
+ }
> print(x)
[1] 3
```

- For loop
  - for (name in statement1) statement2

NB: `statement1` is typically a set of values

```
> df$age
[1] 25 15 34
> for (i in 1:length(df$age)) {
+     df$age[i] = df$age[i] + 1
+ }
> df$age
[1] 26 16 35
```

FUNCTIONS

- General form
  - `name <- function(arg_1, arg_2, ...)`
    `expression`
- Return the max of two arguments

```
> max = function(x,y) {
+     if(x>y)
+         ret = x
+     else
+         ret = y
+     ret
+ }
> max(3,5)
[1] 5
```

- Return the max and whether it was first or second argument

```
> max = function(x,y) {
+     if(x>y) {
+         m = x
+         pos = "first"
+     } else {
+         m = y
+         pos = "second"
+     }
+     ret = c(pos,m)
+     ret
+ }
```

```
> max(3,5)
[1] "second" "5"
> max(5,3)
[1] "first" "5"
```

Default and Named Arguments

- Functions may be defined with default arguments

```
> max = function(x,y=0) {
+     if(x>y) {
+         m = x
+         pos = "first"
+     } else {
+         m = y
+         pos = "second"
+     }
+     ret = c(pos,m)
+     ret
+ }
```

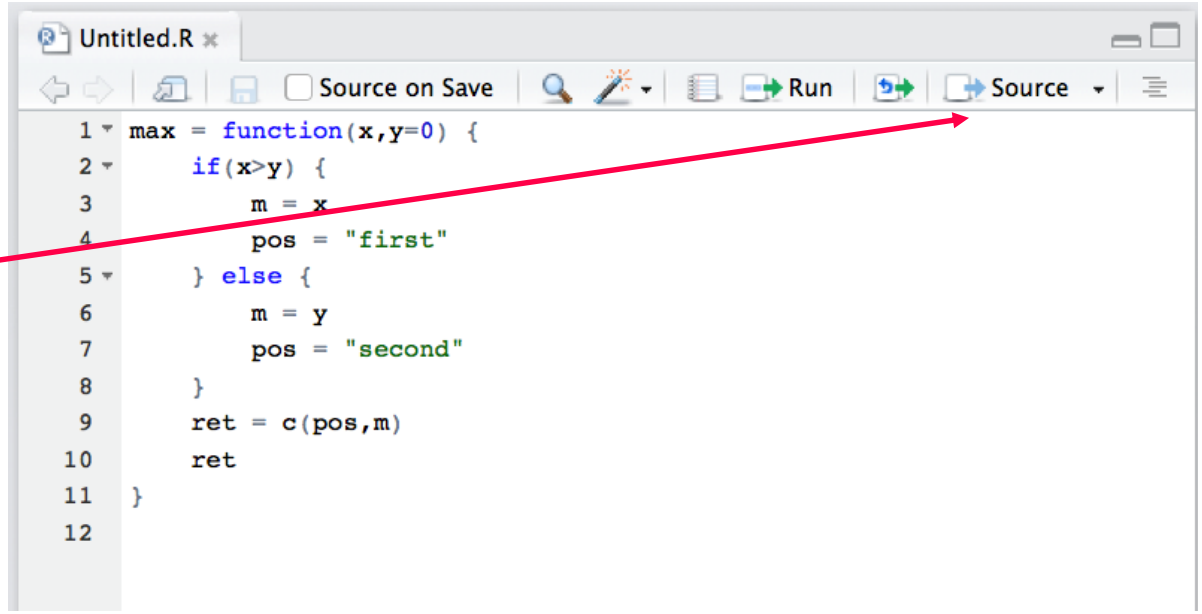
```
> max(5)
[1] "first" "5"
> max(-10)
[1] "second" "0"
> max(-10,-20)
[1] "first" "-10"
```

- Parameters can also be given by name (instead of by position)

```
> max(-10,-20)
[1] "first" "-10"
> max(y=-10,x=-20)
[1] "second" "-10"
```

SCRIPTING

- For non-toys use, most likely you want to
  - Write a script with a set of R statements
  - Execute the script and get the results
- Writing a script
  - Write the script as a text file in any text editor
    - NOT using Word, using a real file text editor
  - Use the file editor integrated in RStudio
- Execute the script
  - Using the `source()` function
  - Loading the script file into the editor and "sourcing" from there



EXERCISE

1. Install (if needed) the MASS package and load it
2. Load the "Animals" data set
3. Calculate the ratio between animals' brain size and their body size, adding the result as a new column called "`proportions`" to the Animals data frame
4. Calculate average and standard deviation of the "`proportions`"
5. Remove the column "`proportions`" from the data frame
6. Select animals with body size > 100
7. Get a list of animals' names with body size > 100 and brain size > 100

EXERCISE

8. Find the average body and brain size for the first 10 animals in the dataset

9. Write a function that returns a list of two elements containing the mean value and the standard deviation of a vector of elements
   - Apply this to the body and brain sizes of Animals

10. Create a vector called `body_norm` with 100 samples from a Normal random variable with average and standard deviation equal to those of body sizes in the Animals dataset
    - print the summary of the generated dataset
    - compare the summary with another dataset of 100 samples with same average and sd = 1

11. Save the Animals data frame to a file named "`animals_a.txt`" with row and column names

12. Create a copy of the file named "`animals_b.txt`", then
    - modify some data in it
    - Read the file into a new data frame, Animals_b
    - Write a function that returns the rows that differ between Animals and Animals_b

13. Save the workspace to a file, clean the workspace, restore the workspace from the file