# R for Absolute Beginners - Hands-on R Tutorial

**Presentation** · June 2018

DOI: 10.13140/RG.2.2.23615.36000



**2 authors**, including:

Isabel Duarte
University of Algarve
**60** PUBLICATIONS **863** CITATIONS

SEE PROFILE

# R for Absolute Beginners - Part 1/2

Syntax and Data Structures in R

*Authors: Isabel Duarte & Ramiro Magno | Collaborators: Bruno Louro & Rui Machado*

*4 June 2018*

## Contents



---

## Introduction

This mini hands-on tutorial serves as an introduction to R, covering the following topics:

- Online sources of information about R;
- Packages, Documentation and Help;
- Basics and syntax of R;
- Main R data structures: Vectors, Matrices, Data frames, Lists and Factors;
- Brief intro to R control-flow via Loops and Conditionals;
- Brief description of function declaration;
- Listing of some of the most commonly used built-in R functions.

This document will guide you through the initial steps toward using R. **RStudio** will be used as the development platform for this workshop since it integrates many functionalities that facilitate the learning process, and it is a free software, available for Linux, Mac and Windows. You can download it directly from: https://www.rstudio.com/products/rstudio/download/

This protocol is divided into **7 parts**, each one identified by a **Title**, **Maximum execution time** (in parenthesis), a brief **Task description** and the **R commands** to be executed. These will always be inside grey text boxes, with the font colored according to the R syntax highlighting.

Now, just *Keep Calm... and Good Work!*

## Online sources and other useful Bibliography

- **Links**
- R Project (The developers of R)
- Quick-R (Roadmap and R code to quickly use R)
- Cookbook for R (R code "recipes")
- Bioconductor workflows (R code for pipelines of genomic analyses)
- Advanced R (If you want to learn R from a programmers point of view)
- **Books**
- Introductory Statistics with R (*Springer*, Dalgaard, 2008)
- A first course in statistical programming with R (*CUP*, Braun and Murdoch, 2016)
- Computational Genome Analysis: An Introduction (*Springer*, Deonier, Tavaré and Waterman, 2005)
- R programming for Bioinformatics (*CRC Press*, Gentleman, 2008)
- R for Data Science: Import, Tidy, Transform, Visualize, and Model Data (*O'Reilly*, Wickham and Grolemund, 2017) (for advanced users)

## Basics

### General notes (about R and RStudio)

1. R is case sensitive - be aware of capital letters (**b** is different from **B**).
2. All R code lines starting with the # (hash) sign are interpreted as comments, and therefore not evaluated.

```
# This is a comment
# 3 + 4   # this code is not evaluated, so and it does not print any result
2 + 3     # the code before the hash sign is evaluated, so it prints the result (value 5)
```

```
[1] 5
```

3. Expressions in R are evaluated from the innermost parenthesis toward the outermost one (following proper mathematical rules).

```
# Example with parenthesis:
((2+2)/2)-2
```

```
[1] 0
```

```
# Without parenthesis:
2+2/2-2
```

```
[1] 1
```

4. Spaces matter in variable names — use a dot or underscore to create longer names to make the variables more descriptive, e.g. `my.variable_name`.

5. Spaces between variables and operators do not matter: `3+2` is the same as `3 + 2`, and `function (arg1 , arg2)` is the same as `function(arg1,arg2)`.

6. If you want to write 2 expressions/commands in the same line, you have to separate them by a `;` (semi-colon)

```
#Example:
3 + 2 ; 5 + 1
```

```
[1] 5
```

```
[1] 6
```

7. More recent versions of RStudio **auto-complete** your commands by showing you possible alternatives as soon as you type 3 consecutive characters, however, if you want to see the options for less than 3 chars, just press tab to display available options. **Tip: Use auto-complete as much as possible to avoid typing mistakes.**

8. There are 4 main vector **data types**: **Logical** (TRUE or FALSE); **Numeric** (eg. 1,2,3...); **Character** (eg. "u", "alg", "arve") and **Complex** (eg. 3+2i)

9. Vectors are ordered sets of elements. In R vectors are **1-based**, i.e. the first index position is number 1 (as opposed to other programming languages whose indexes start at zero).

10. R objects can be divided in two main groups: **Functions** and **Data-related objects**. Functions receive arguments inside circular brackets `( )` and objects receive arguments inside square brackets `[ ]`:

```
function (arguments)
data.object [arguments]
```

### Start/Quit RStudio

RStudio can be opened by double-clicking its icon.

The R environment is controlled by hidden files (files that start with a `.`) in the startup directory: `.RData`, `.Rhistory` and `.Rprofile` (optional).

- **.RData** is a file containing all the objects, data, and functions created during a work-session. This file can then be loaded for future work without requiring the re-computation of the analysis. (Note: it can potentially be a very large file);
- **.Rhistory** saves all commands that have been typed during the R session;
- **.Rprofile** useful for advanced users to customize RStudio behaviour.

It is always good practice to rename these files:

```
# DO NOT RUN
save.image (file="myProjectName.RData")
savehistory (file="myProjectName.Rhistory")
```

To quit R (close it), use the `q ()` function, and you will be asked if you want to save the workspace image (i.e. the `.RData` file):

```
q()
```

```
Save workspace image to ~/path/to/your/working/directory/.RData? [y/n/c]:
```

By typing `y` (yes), then the entire R workspace will be written to the `.RData` file (which can be very large). Often it is sufficient to just save an analysis script (i.e. a reproducible protocol) in an R source file. This way, one can quickly regenerate all data sets and objects for future analysis. The .RData file is particularly useful to save the results from analyses that require a long time to compute.

### Package repositories

In R, the fundamental unit of shareable code is the **package**. A package bundles together code, data, documentation, and tests, and is easy to share with others. These packages are stored online from which they can be easily retrieved and installed on your computer (R packages by Hadley Wickham). There are 2 main R repositories:

- The Comprehensive R Archive Network - CRAN (nearly 8500 packages)
- Bioconductor (>1560 packages in June 2018) (bioscience data analysis)

This huge variety of packages is one of the reasons why R is so successful: the chances are that someone has already solved a problem that you're working on, and you can benefit from their work by downloading their package for free.

In this course, we will not use any packages. However, if you continue to use R for bioinformatics analysis you will need to install Bioconductor. So for future reference, here is the code to install Bioconductor, and to set the repositories that you want to use when searching and installing packages:

```
##### THIS PROCESS MIGHT TAKE A VERY LONG TIME #####

# To Install Bioconductor, run the following code
   ## try http:// if https:// URLs are not supported
```

```r
source("https://bioconductor.org/biocLite.R")
biocLite()

# To set the other relevant repositories:
setRepositories()

# then follow the instructions and input the numbers corresponding to the requested repositories
 # (if you want to cover most packages, just use all listed repositories: 1 2 3 4 5 6 7 8 9)
```

#### Installing packages and Getting help

R has many built-in ways of providing help regarding its functions and packages:

```r
install.packages ("ggplot2")   # install the package called ggplot2
library ("ggplot2")    # load the library ggplot2
help (package=ggplot2) # help(package="package_name") to get help about a specific package
vignette ("ggplot2")   # show a pdf with the package manual (called R vignettes)

?qplot   # ?function to get quick info about the function of interest
```

#### Working environment

Your working environment is the place where the variables, functions, and data that you create are stored. More advanced users can create more than one environment.

```r
ls()    # list all objects in your environment
dir()   # list all files in your working directory
getwd() # find out the path to your working directory
setwd("/home/isabel") # example of setting a new working directory path
```

---

## Hands-on tutorial

## 1. Create an RStudio project (30 min)

To start we will open RStudio. This is an Integrated Development Environment - **IDE** - that includes syntax-highlighting **text editor** (1), an **R console** to execute code (2), as well as **workspace** and **history** management (3), and tools for **plotting** and **exporting** images, **browsing** the workspace, managing **packages** and **viewing** html/pdf files created within RStudio (4).

Projects are a great functionality, easing the transition between dataset analysis, and allowing a fast navigation to your analysis/working directory. To create a new project:

```
File > New Project... > New Directory > New Project
Directory name: r-absoluteBeginners
Create project as a subdirectory of: ~/
                          Browse... (directory/folder to save the workshop data)
Create Project
```

Projects should be personalized by clicking on the menu in the right upper corner. The general options - **R General** - are the most important to customize, since they allow the definition of the RStudio "behavior" when the project is opened. The following suggestions are particularly useful:

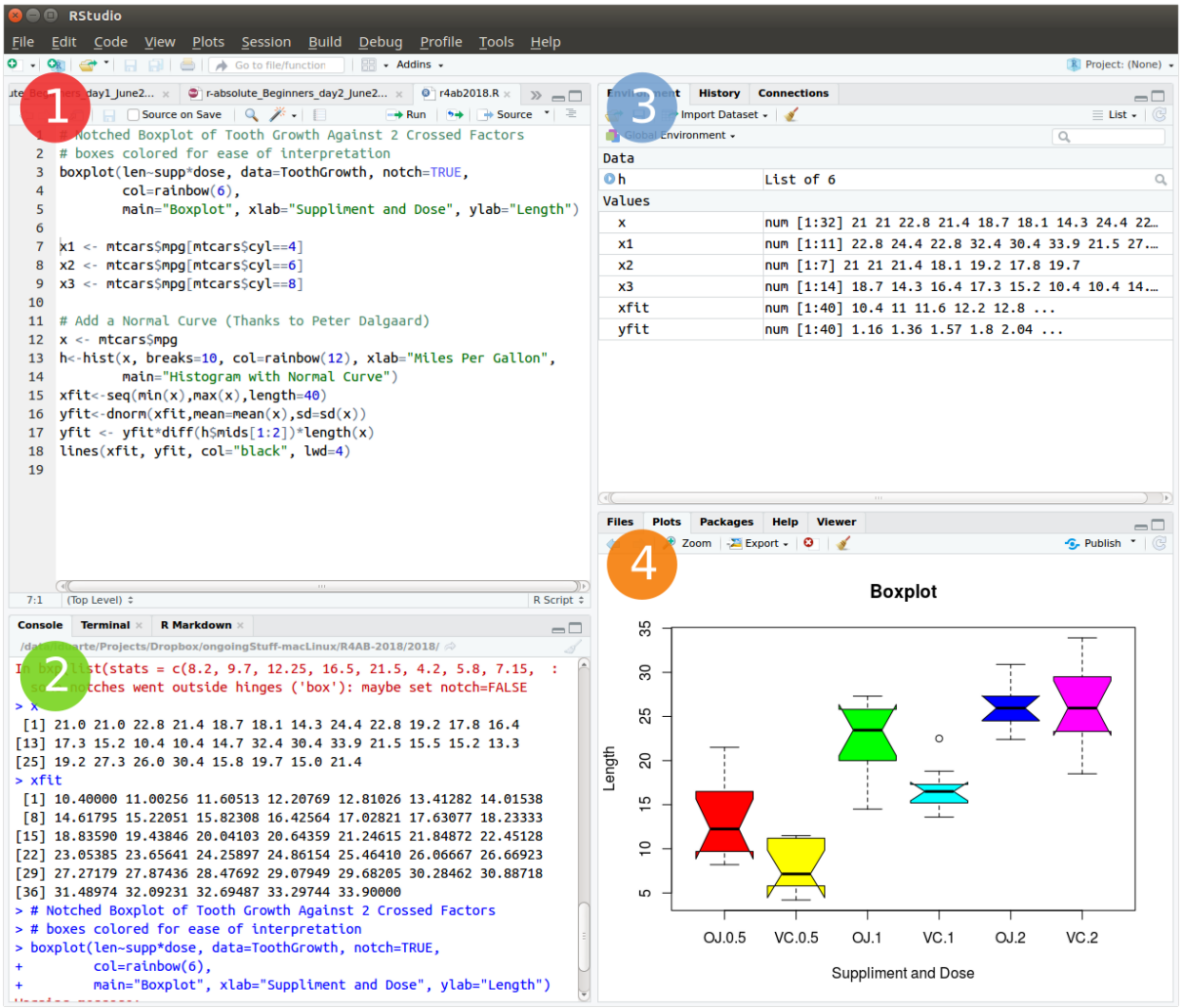



Figure 1: Figure 1: RStudio Graphical User Interface (GUI)

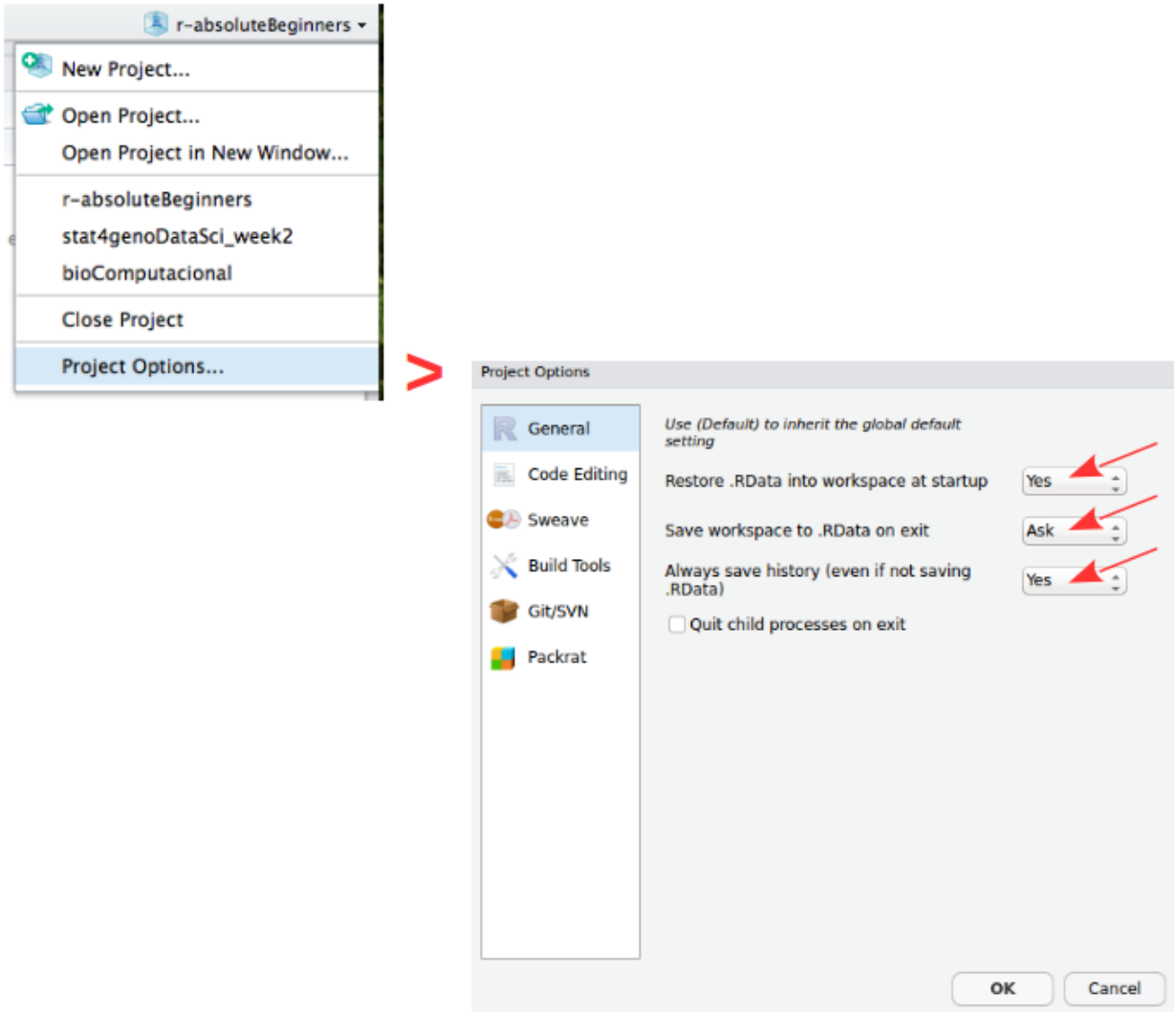



Figure 2: Figure 2: Customize Project

```
Restore .RData at startup - Yes (for analyses with +1GB of data, you should choose "No")
Save .RData on exit - Ask
Always save history - Yes
```

## 2. Operators (60 min)

**Important NOTE: Please create a new R Script file to save all the code you use for today's tutorial and save it in your current working directory. Name it: r4ab_day1.R**

### 2.1 Assignment operators

Values are assigned to named variables with an `<-` (arrow) or an `=` (equal) sign. In most cases they are interchangeable, however it is good practice to use the arrow since it is explicit about the direction of the assignment. If the **equal sign** is used, the assignment occurs from left to right.

```r
x <- 7      # assign the number 7 to a variable named x
x           # R will print the value associated with variable x

y <- 9      # assign the number 9 to the variable y

z = 3       # assign the value 3 to the variable z

42 -> lue   # assign the value 42 to the variable named lue

x ->  xx    # assign the value of x (which is the number 7) to the variable named xx
xx

my_variable = 5    # assign the number 5 to the variable named my_variable
```

### 2.2 Comparison operators

Allow the direct comparison between values:

| Symbol | Description |
|---|---|
| == | exactly the same (equal) |
| != | different (not equal) |
| < | smaller than |
| > | greater than |
| <= | smaller or equal |
| >= | greater or equal |

```r
1 == 1    # TRUE
1 != 1    # FALSE
x > 3     # TRUE (x is 7)
y <= 9    # TRUE (y is 9)
my_variable < z    # FALSE (z is 3 and my_variable is 5)
```

### 2.3 Logical operators

Compare logical (TRUE FALSE) values:

| Symbol | Description |
|---|---|
| & | AND |
| \| | OR |
| ! | NOT |

**QUESTION:** Are these TRUE, or FALSE?

```r
x < y & x > 10    # AND means that both expressions have to be true
x < y | x > 10    # OR means that only one expression must be true
!(x != y & my_variable <= y)  # yet another AND example
```

### 2.4 Arithmetic operators

R makes calculations using the following arithmetic operators:

| Symbol | Description |
|---|---|
| + | summation |
| - | subtraction |
| * | multiplication |
| / | division |
| ^ | powering |

```
3 / y   ## 0.3333333

x * 2   ## 14

3 - 4   ## -1

my_variable + 2   ## 7

2^z   ## 8
```

## 3. Data structures (120 min)

### 3.1 Vectors

The basic data structure in R is the **vector**, which requires all of its elements to be of the same type (e.g. all numeric; all character (text); all logical (TRUE FALSE)).

#### Creating vectors

| Function | Description |
|---|---|
| c | combine |
| : | integer sequence |
| seq | general sequence |
| rep | repetitive patterns |

```
x <- c (1,2,3,4,5,6)
x
```

```
[1] 1 2 3 4 5 6
```

```
class (x)   # this function outputs the class of the object
```

```
[1] "numeric"
```

```
y <- 10
class (y)
```

```
[1] "numeric"
```

```r
z <- "a string"
class (z)
```

```
[1] "character"
```

```r
# The results are shown in the comments next to each line

seq (1,6)   ## 1 2 3 4 5 6
seq (from=100, by=1, length=5)   ## 100 101 102 103 104

1:6    ## 1 2 3 4 5 6
10:1   ## 10  9  8  7  6  5  4  3  2  1

rep (1:2, 3)   ## 1 2 1 2 1 2
```

### Vectorized arithmetics

Most arithmetic operations in the R language are **vectorized**, i.e. the operation is applied **element-wise**. When one operand is shorter than the other, the shortest one is **recycled**, i.e. the values from the shorter vector are re-used in order to have the same length as the longer vector.
Please note that when one of the vectors is recycled, a **warning** is printed in the R Console. This warning is **not an error**, i.e. the operation has been completed despite the warning message.

```r
1:3 + 10:12
```

```
[1] 11 13 15
```

```r
# Notice the warning: this is recycling (the shorter vector "restarts" the "cycling")
1:5 + 10:12
```

```
Warning in 1:5 + 10:12: longer object length is not a multiple of shorter
object length
```

```
[1] 11 13 15 14 16
```

```r
x + y          # Remember that x = c(1 2 3 4 5 6) and y = 10
```

```
[1] 11 12 13 14 15 16
```

```r
c(70,80) + x
```

```
[1] 71 82 73 84 75 86
```

### Subsetting/Indexing vectors

Subsetting is one of the most powerfull features of R. It is the extraction of one or more elements, which are of interest, from vectors, allowing for example the filtering of data, the re-ordering of tables, removal of unwanted datapoints, etc. There are several ways of subsetting data.

**Note: Please remember that indices in R are 1-based (see introduction).**

```r
# Subsetting by indices
myVec <- 1:26 ; myVec
myVec [1]    # prints the first value of myVec
myVec [6:9]  # prints the 6th, 7th, 8th and 9th values of myVec

# LETTERS is a built-in vector with the 26 letters of the alphabet
myLOL <- LETTERS       # assign the 26 letters to the vector named myLOL
myLOL[c(3,3,13,1,18)]  # print the requested positions of vector myLOL
```

```
#Subsetting by same length logical vectors
myLogical <- myVec > 10 ; myLogical
# returns only the values in positions corresponding to TRUE in the logical vector
myVec [myLogical]
```

### Naming indexes of a vector

Referring to an index by name rather than by position can make code more readable and flexible. Use the function *names* to attribute names to each position of the vector.

```
joe <- c (24, 1.70)
names (joe)               ## NULL
names (joe) <- c ("age","height")
names (joe)               ## "age"    "height"
joe ["age"] == joe [1]    ## age   TRUE

names (myVec) <- LETTERS
myVec
# Subsetting by field names
myVec [c("A", "A", "B", "C", "E", "H", "M")] ## The Fibonacci Series :o)
```

### Excluding elements

Sometimes we want to retain most elements of a vector, except for a few unwanted positions. Instead of specifying all elements of interest, it is easier to specify the ones we want to remove. This is easily done using the minus sign.

```
alphabet <- LETTERS
alphabet   # print vector alphabet
```

```
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q"
[18] "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```

```
vowel.positions <- c(1,5,9,15,21)
alphabet[vowel.positions]    # print alphabet in vowel.positions
```

```
[1] "A" "E" "I" "O" "U"
```

```
consonants <- alphabet [-vowel.positions]  # exclude all vowels from the alphabet
consonants
```

```
 [1] "B" "C" "D" "F" "G" "H" "J" "K" "L" "M" "N" "P" "Q" "R" "S" "T" "V"
[18] "W" "X" "Y" "Z"
```

## 3.2 Matrices

Matrices are two dimensional vectors (tables), explicitly created with the `matrix` function. Just like one-dimensional vectors, they store **same-type elements**.

**IMPORTANT NOTE:** R uses a **column-major order** for the internal linear storage of array values, meaning that first all of column 1 is stored, then all of column 2, etc. This implies that, by default, when you create a matrix, R will populate the first column, then the second, then the third, and so on until all values given to the `matrix` function are used. This is the default behaviour of the matrix function, which can be changed via the `byrow` parameter (default value is set to FALSE).

```r
my.matrix <- matrix (1:12, nrow=3, byrow = FALSE)   # byrow = FALSE is the default (see ?matrix)
dim (my.matrix)   # check the dimension (size) of the matrix: number of rows and number of columns
my.matrix         # print the matrix

xx <- matrix (1:12, nrow=3, byrow = TRUE)
dim (xx)  # check if the dimensions of xx are the same as the dimensions of my.matrix
xx        # compare my.matrix with xx and make sure you understand what is hapenning
```

#### Subsetting/Indexing matrices

**Very Important Note:** The arguments inside the square brackets in matrices (and data.frames - see next section) are the `[row_number, column_number]`. If any of these is omitted, R assumes that all values are to be used.

```r
# Creating a matrix of characters
my.matrix <- matrix (LETTERS, nrow = 4, byrow = TRUE)
# Please notice the warning message (related to the "recycling" of the LETTERS)

my.matrix         # print the matrix
dim (my.matrix)   # check the dimensions of the matrix

# Subsetting by indices
my.matrix [,2]   # all rows, column 2 (returns a vector)
my.matrix [3,]   # row 3, all columns (returns a vector)
my.matrix [1:3,c(4,2)]   # rows 1, 2 and 3 from columns 4 and 2 (by this order) (returns a matrix)
```

### 3.3 Data frames

Data frames are the most flexible and commonly used R data structures, used to store datasets in spreadsheet-like tables.
In a `data.frame`, usually the observations are the rows and the variables are the columns. Unlike matrices, each column of a data frame can be a vector of different type (i.e. text, number, logicals, etc, can all be stored in the same data frame). Each column must to be of the same data type. Data frames are easily subset by index number or by column name.

```r
df <- data.frame (type=rep(c("case","control"),c(2,3)),time=rnorm(5))
# rnorm is a random number generator retrieved from a normal distribution

class (df)   ## "data.frame"
df
```

#### Subsetting/Indexing Data frames

**Remember:** The arguments inside the square brackets, just like in matrices, are the `[row_number, column_number]`. If any of these is omitted, R assumes that all values are to be used.

**NOTE:** R includes a package in its default base installation, named **"The R Datasets Package"**. This resource includes a diverse group of datasets, containing data from different fields: biology, physics, chemistry, economics, psychology, mathematics. These data are very useful to learn R. For more info about these datasets, run the following command: `library(help=datasets)`

```r
# Familiarize yourself with the iris dataset (built-in dataset with measurements of iris flowers)
iris
```

```r
# Subset by indices the iris dataset
iris [,3]   # all rows, column 3
iris [1,]   # row 1, all columns
iris [1:9, c(3,4,1,2)]   # rows 1 to 9 with columns 3, 4, 1 and 2 (in this order)

# Subset by column name (for data.frames)
iris$Species            #show only the species column
iris[,"Sepal.Length"]

# Select the time column from the df data frame created above
df$time      ## 0.5229577 0.7732990 2.1108504 0.4792064 1.3923535
```

### 3.4 Lists

Lists are very powerful data structures, consisting of ordered sets of elements, that can be arbitrary R objects (vectors, strings, functions, etc), and heterogeneous, i.e. each element of a different type.

```r
lst = list (a=1:3, b="hello", fn=sqrt)   # index 3 contains the function "square root"
lst
lst$fn(49)   # outputs the square root of 49
```

#### Subsetting/Indexing lists

```r
# Subsetting by indices
lst [1]     # returns a list with the data contained in position 1 (preserves the type of data as list)
class (lst[1])

lst [[1]]   # returns the data contained in position 1 (simplifies to inner data type)
class(lst[[1]])

# Subsetting by name
lst$b       # returns the data contained in position 1 (simplifies to inner data type)
class(lst$b)

# Compare the class of these alternative indexing by name
lst["a"]
lst[["a"]]
```

### 3.5 Data structure conversion

Data structures can be interconverted (coerced) from one type to another. Sometimes it is useful to convert between data structure types (particularly when using packages). R has several functions for such conversions:

```r
# To check the class of the object:
class(lst)

# To check the basic structure of an object:
str(lst)

# "Force" the object to be of a certain type:
 # (this is not valid code, just a syntax example)
as.matrix (myDataFrame)  # convert a data frame into a matrix
```

```r
as.numeric (myChar)      # convert text characters into numbers
as.data.frame (myMatrix) # convert a matrix into a data frame
as.character (myNumeric) # convert numbers into text chars
```

## 4. Loops and Conditionals in R (60 min)

### 4.1 `for()` and `while()` loops

R allows the implementation of **loops**, i.e. replicating instructions in an iterative way (also called cycles). The most common ones are `for()` loops and `while()` loops. The syntax for these loops is: `for (condition) { code-block }` and `while (condition) { code-block }`.

```r
# creating a for loop to calculate the first 12 values of the Fibonacci sequence
my.x <- c(1,1)
for (i in 1:10) {
  my.x <- c(my.x, my.x[i] + my.x[i+1])
  print(my.x)
}

# while loops will execute a block of commands until a condition is no longer satisfied
x <- 3 ; x
while (x < 9)
{
  cat("Number", x, "is smaller than 9.\n") # cat is a printing function (see ?cat)
   x <- x+1
}
```

### 4.2 Conditionals: `if()` statements

Conditionals allow running commands only when certain conditions are TRUE. The syntax is: `if (condition) { code-block }`.

```r
x <- -5 ; x
if (x >= 0) { print("Non-negative number") } else { print("Negative number") }
 # Note: The else clause is optional. If the command is run at the command-line,
  # and there is an else clause, then either all the expressions must be enclosed
  # in curly braces, or the else statement must be in line with the if clause.

# coupled with a for loop
x <- c(-5:5) ; x
for (i in 1:length(x)) {
  if (x[i] > 0) {
     print(x[i])
  }
  else {
   print ("negative number")
  }
}
```

### 4.3 Conditionals: `ifelse()` statements

The `ifelse` function combines element-wise operations (vectorized) and filtering with a condition that is evaluated. The major advantage of the `ifelse` over the standard if-then-else statement is that it is vectorized. The syntax is: `ifelse (condition-to-test, value-for-true, value-for-false)`.

```
# re-code gender 1 as F (female) and 2 as M (male)
gender <- c(1,1,1,2,2,1,2,1,2,1,1,1,2,2,2,2,2)
ifelse(gender == 1, "F", "M")
```

```
 [1] "F" "F" "F" "M" "M" "F" "M" "F" "M" "F" "F" "F" "M" "M" "M" "M" "M"
```

## 5. Functions (60 min)

R allows defining new functions using the `function` command. The syntax (in pseudo-code) is the following:

```
my.function.name <- function (argument1, argument2, ...) {
  expression1
  expression2
  ...
  return (value)
  }
```

Now, lets code our own function to calculate the average (or mean) of the values from a vector:

```
# Define the function
    # Please note that the function must be declared in the script before it can be used
my.average <- function (x) {
  average.result <- sum(x)/length(x)
  return (average.result)
}

# Create the data vector
my.data <- c(10,20,30)

# Run the function using the vector as argument
my.average(my.data)

# Compare with R built-in mean function
mean(my.data)
```

## 6. Loading data and Saving files (30 min)

Most R users need to load their own datasets, usually saved as table files (e.g. Excel, or .csv files), to be able to analyse and manipulate them. After the analysis, the results need to be exported/saved (eg. to view or use with other software).

```
# Inspect the esoph built-in dataset
esoph
dim(esoph)
colnames(esoph)

### Saving ###
# Save to a file named esophData.csv the esoph R dataset, separated by commas and
 # without quotes (the file will be saved in the current working directory)
```

```
write.table (esoph, file="esophData.csv", sep="," , quote=F)

# Save to a file named esophData.tab the esoph dataset, separated by tabs and without
 # quotes (the file will be saved in the current working directory)
write.table (esoph, file="esophData.tab", sep="\t" , quote=F)

### Loading ###
# Load a data file into R (the file should be in the working directory)
  # read a table with columns separated by tabs
my.data.tab <- read.table ("esophData.tab", sep="\t", header=TRUE)
 # read a table with columns separated by commas
my.data.csv <- read.csv ("esophData.csv", header=T)
```

*Note:* if you want to **load** or **save** the files in directories different from the working dir, just use (inside quotes) the full path as the first argument, instead of just the file name (e.g. "/home/Desktop/r_Workshop/esophData.csv").

## 7. Some great R functions to "play" with (60 min)

### 7.1 Using the `iris` buil-in dataset

```
# the unique function returns a vector with unique entries only (remove duplicated elements)
unique (iris$Sepal.Length)

# length returns the size of the vector (i.e. the number of elements)
length (unique (iris$Sepal.Length))

# table counts the occurrences of entries (tally)
table (iris$Species)

# aggregate computes statistics of data aggregates (groups)
aggregate (iris[,1:4], by=list (iris$Species), FUN=mean, na.rm=T)

# the %in% function returns the intersection between two vectors
month.name [month.name %in% c("CCMar","May", "Fish", "July", "September","Cool")]

# merge joins data frames based on a common column (that functions as a "key")
df1 <- data.frame(x=1:5, y=LETTERS[1:5]) ; df1
df2 <- data.frame(x=c("Eu","Tu","Ele"), y=1:6) ; df2
merge (df1, df2, by.x=1, by.y=2, all = TRUE)

# cbind and rbind (takes a sequence of vector, matrix or data-frame arguments
 # and combine them by columns or rows, respectively)
my.binding <- as.data.frame(cbind(1:7, LETTERS[1:7]))    # the '1' (shorter vector) is recycled
my.binding
my.binding <- cbind(my.binding, 8:14)[, c(1, 3, 2)] # insert a new column and re-order them
my.binding

my.binding2 <- rbind(seq(1,21,by=2), c(1:11))
my.binding2

# reverse the vector
rev (LETTERS)
```

```r
# sum and cumulative sum
sum (1:50); cumsum (1:50)
# product and cumulative product
prod (1:25); cumprod (1:25)

### Playing with some R built-in datasets (see library(help=datasets) )
iris   # familiarize yourself with the iris data

# mean, standard deviation, variance and median
mean (iris[,2]); sd (iris[,2]); var (iris[,2]); median (iris[,2])

# minimum, maximum, range and summary statistics
min (iris[,1]); max (iris[,1]); range (iris[,1]); summary (iris)

# exponential, logarithm
exp (iris[1,1:4]); log (iris[1,1:4])

# sine, cosine and tangent (radians, not degrees)
sin (iris[1,1:4]); cos (iris[1,1:4]); tan (iris[1,1:4])

# sort, order and rank the vector
sort (iris[1,1:4]); order (iris[1,1:4]); rank (iris[1,1:4])

# useful to be used with if conditionals
any (iris[1,1:4] > 2)   # ask R if there are any values higher that 2?
all (iris[1,1:4] > 2)   # ask R if all values are higher than 2

# select data
which (iris[1,1:4] > 2)
which.max (iris[1,1:4])
```

### 7.2 Using the esoph buil-in dataset

The `esoph` (Smoking, Alcohol and (O)esophageal Cancer data) built-in dataset presents 2 types of variables: *continuous numerical variables* (the number of cases and the number of controls), and *discrete categorical variables* (the age group, the tobacco smoking group and the alcohol drinking group). Sometimes it is hard to "categorize" continuous variables, i.e. to group them in specific intervals of interest, and name these groups (also called *levels*).

Accordingly, imagine that we are interested in classifying the number of cancer cases according to their occurrence: *frequent*, *intermediate* and *rare*. This type of variable recoding into *factors* is easily accomplished using the function `cut()`, which divides the range of x into intervals and codes the values in x according to which interval they fall.

```r
# subset non-contiguous data from the esoph dataset
esoph
summary(esoph)
# cancers in patients consuming more than 30 g/day of tobacco
subset(esoph$ncases, esoph$tobgp == "30+")
# total nr of cancers in patients older than 75
sum(subset(esoph$ncases, esoph$agegp == "75+"))

# factorize the nr of cases in 3 levels, equally spaced,
 # and add the new column named cat_ncases, to the dataset
```

```
esoph$cat_ncases <- cut (esoph$ncases,3,labels=c("rare","med","freq"))
summary(esoph)
```

END

# R for Absolute Beginners - Part 2/2

Summary Statistics and Graphics in R

*Authors: Isabel Duarte & Ramiro Magno | Collaborators: Bruno Louro & Rui Machado*

*5 June 2018*

## Contents



---

## Introduction

The goal of this tutorial is to get you acquainted with common first steps when dealing with a dataset in R, such as:

1. reading/loading data into R

2. inspecting R objects
3. cleaning and tidying-up the data
4. basic descriptive statistics.

To this end, you will be guided through various exercises. Each exercise's title indicates the time allocated to solve it. Since this time indication is an over-estimation of the time needed to properly answer each question, please take your time to think about it and discuss it with the instructors. The tutorial comprises 10 exercises, of which only the first nine are to be fully completed during the workshop. Feel free to proceed with Exercise 10 if you finish earlier; and if not, try to complete it at home. For now, keep calm and carry on... and remember, do not hesitate to ask any questions to the instructors!

**Important Note:** There are several alternative ways to accomplish these exercises in R; our suggestions are just one possibility, particularly targeted for beginners and using simple R functions, organized in small individual steps (without using any extra R packages). If you know another way that you find *more intuitive* (easier for you), please feel free to use it, just **make sure that you *truly* understand each command used**.

## Hands-on Exercises

This tutorial uses a dataset retrieved from the study *Reversal of ocean acidification enhances net coral reef calcification* by Albright et al., 2016.

### Exercise 0. Understand the context of your data (15 min)

*Please read the following introduction.*

A recent *in situ* experiment (Albright et al., 2016) has found that reducing the acidity of the seawater surrounding a natural coral reef in the southern Great Barrier Reef significantly increases calcification. In this experiment the acidity levels have been reverted to those characteristic of the pre-industrial era. By "turning back time", the authors demonstrate that, all else being equal, net coral-reef calcification would have been around 7% higher than current observations, suggesting that ocean acidification may already be diminishing coral-reef growth.

*One Tree Reef* encloses three lagoons, two of which are hydrologically distinct (i.e., separated by reef walls). At low tide, the water level drops below the outer reef crest, and the lagoons are effectively isolated from the ocean (Figure 2c). Since *First Lagoon* sits approximately 30 cm higher than *Third Lagoon*, gravity-driven, unidirectional flow results from *First Lagoon* over the reef flat separating the two lagoons, ending up in *Third Lagoon*. The study site is situated along a section of the reef wall separating *First* and *Third* Lagoons (Figure 2d).

### Exercise 1. Get the data (10 min)

Download the Supplementary Table 1 file containing the raw data for chemical and physical parameters measured (or calculated) for all days and station locations. Save it in a directory of your choice, and please keep the file's original name.

### Exercise 2. Format conversion (10 min)

Open the downloaded file (should be named `nature17155-s2.xlsx`) with a spreadsheet software program (e.g. Microsoft Excel or OpenOffice Calc) and export it as CSV (comma separated values). Save the exported

Figure 1: **Figure 1** | *One Tree Reef* in the southern Great Barrier Reef, Australia (Janice M. Lough, 2016).

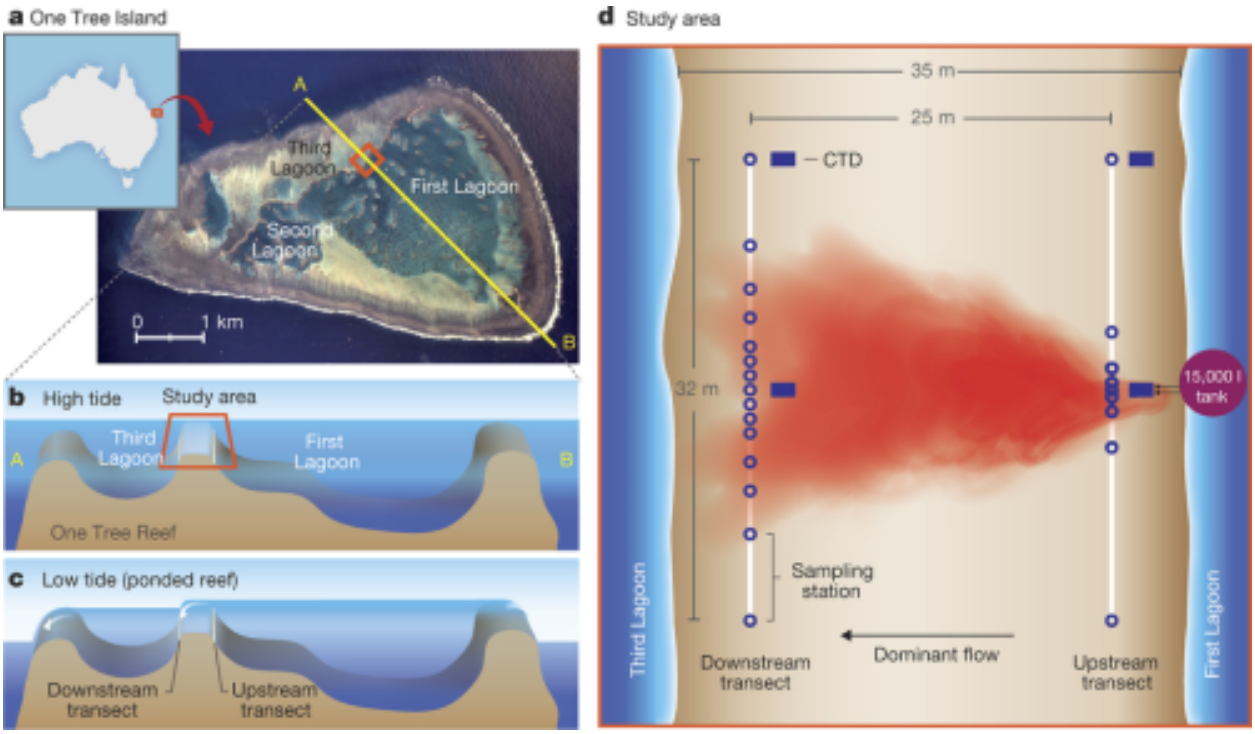



Figure 2: **Figure 2** | One Tree Reef in the southern Great Barrier Reef, Australia (Albright et al., 2016).

file in the same folder and name it `nature17155-s2.csv`.

### Exercise 3. Set working directory (15 min)

Open RStudio (if you have not done so already) and from the **R console** run a command (or a combination of commands) that confirms that the file `nature17155-s2.csv` is "visible" from R. If the working directory is not the one containing `nature17155-s2.csv`, then change to it accordingly.

**Hint: the functions `getwd`, `setwd`, `list.files`, `dir` and `file.exists` are your friends.**

### Exercise 4. Checking the data file structure (10 min)

The file `nature17155-s2.csv` has been saved as a CSV, which is a particular type of text file, where each value (i.e. column) is separated by a comma character (**,**). However, it is possible to have a few variations, the most relevant being: (i) the specific character used as value separator (e.g. comma, semi-colon (**;**), tab); (ii) whether values are quoted (**"**), (iii) whether the first line is a header (i.e. column names) or not. These details are decisive in order to correctly import the data into R.

So, in order to have a glance at how the data are formatted/organized in the dataset file `nature17155-s2.csv`, read (i.e. show) the first 3 lines in the R console:

```
readLines("nature17155-s2.csv", n = 3)
```

```
[1] "Station ID,Transect,Date,Type,T (C) in situ,Salinity,Alkalinity (umol/kg), Rhodamine (ppb),..."
[2] "D-16,Down,20140916,Control ,22.507,35.8542,2280.58,0.0507,8.1131,298.44,2273.95,..."
[3] "D-16,Down,20140917,Experiment ,22.995,35.8287,2253.42,0.1940,8.1206,298.4,2248.47,..."
```

### Exercise 5. Load data into R (20 min)

From the output of the last command one can see that the comma character is indeed separating the different columns. Notice however that data-fields with text containing commas are quoted, i.e. enclosed in quotation marks, so that those free-text-commas are not mistakingly used as column separators. Additionally, notice that the first line **is the header** of the dataset (column names) and **not an observation/data point**.

#### Exercise 5.1

Now, import the dataset using the function `read.table` with appropriate arguments, and save it in an R object named `reef`:

```
reef <- read.table("DATA/nature17155-s2.csv", header = TRUE, sep = ",", strip.white = TRUE)
# the strip.white argument removes trailing white spaces (spaces in the
# begining and end of each column)
```

#### Exercise 5.2

Next, let us inspect some of the imported data which has been saved in the variable `reef`.

```
class(reef)
head(reef)  # inspect the first lines of reef
tail(reef)  # inspect the last lines of reef
```

**Exercise 6. Inspecting an R data frame (20 min)**

The imported data has been loaded as a data frame having several columns, such as `Station.ID`, `Transect`, `Date`, etc.. Notice that *special* characters like white spaces or parenthesis in the column names have been converted by R to dots (`.`).

Please note that in RStudio, data frames can be graphically inspected; by clicking on its name in the environment panel, a new tab opens in the text editor panel, showing its first 1000 lines; so try that too!

Examine other relevant information about the `reef` data frame. Note: This is a challenge to try to discover the functions that output these results.

```
[1] 526  16
```

```
 [1] "Station.ID"                    "Transect"
 [3] "Date"                          "Type"
 [5] "T..C..in.situ"                 "Salinity"
 [7] "Alkalinity..umol.kg."          "Rhodamine..ppb."
 [9] "Spec.pH..total."               "T..K..Spec.pH"
[11] "Alk..S.Normalized..umol.kg."   "Rhodamine..S.Normalized..ppb."
[13] "in.situ.pH..total."            "in.situ.pCO2..uatm."
[15] "in.situ.CT..umol.kg."          "in.situ.aragonite"
```

```
  [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"  "11"
 [12] "12"  "13"  "14"  "15"  "16"  "17"  "18"  "19"  "20"  "21"  "22"
 [23] "23"  "24"  "25"  "26"  "27"  "28"  "29"  "30"  "31"  "32"  "33"
 [34] "34"  "35"  "36"  "37"  "38"  "39"  "40"  "41"  "42"  "43"  "44"
 [45] "45"  "46"  "47"  "48"  "49"  "50"  "51"  "52"  "53"  "54"  "55"
 [56] "56"  "57"  "58"  "59"  "60"  "61"  "62"  "63"  "64"  "65"  "66"
 [67] "67"  "68"  "69"  "70"  "71"  "72"  "73"  "74"  "75"  "76"  "77"
 [78] "78"  "79"  "80"  "81"  "82"  "83"  "84"  "85"  "86"  "87"  "88"
 [89] "89"  "90"  "91"  "92"  "93"  "94"  "95"  "96"  "97"  "98"  "99"
[100] "100" "101" "102" "103" "104" "105" "106" "107" "108" "109" "110"
[111] "111" "112" "113" "114" "115" "116" "117" "118" "119" "120" "121"
[122] "122" "123" "124" "125" "126" "127" "128" "129" "130" "131" "132"
[133] "133" "134" "135" "136" "137" "138" "139" "140" "141" "142" "143"
[144] "144" "145" "146" "147" "148" "149" "150" "151" "152" "153" "154"
[155] "155" "156" "157" "158" "159" "160" "161" "162" "163" "164" "165"
[166] "166" "167" "168" "169" "170" "171" "172" "173" "174" "175" "176"
[177] "177" "178" "179" "180" "181" "182" "183" "184" "185" "186" "187"
[188] "188" "189" "190" "191" "192" "193" "194" "195" "196" "197" "198"
[199] "199" "200" "201" "202" "203" "204" "205" "206" "207" "208" "209"
[210] "210" "211" "212" "213" "214" "215" "216" "217" "218" "219" "220"
[221] "221" "222" "223" "224" "225" "226" "227" "228" "229" "230" "231"
[232] "232" "233" "234" "235" "236" "237" "238" "239" "240" "241" "242"
[243] "243" "244" "245" "246" "247" "248" "249" "250" "251" "252" "253"
[254] "254" "255" "256" "257" "258" "259" "260" "261" "262" "263" "264"
[265] "265" "266" "267" "268" "269" "270" "271" "272" "273" "274" "275"
[276] "276" "277" "278" "279" "280" "281" "282" "283" "284" "285" "286"
[287] "287" "288" "289" "290" "291" "292" "293" "294" "295" "296" "297"
[298] "298" "299" "300" "301" "302" "303" "304" "305" "306" "307" "308"
[309] "309" "310" "311" "312" "313" "314" "315" "316" "317" "318" "319"
[320] "320" "321" "322" "323" "324" "325" "326" "327" "328" "329" "330"
[331] "331" "332" "333" "334" "335" "336" "337" "338" "339" "340" "341"
[342] "342" "343" "344" "345" "346" "347" "348" "349" "350" "351" "352"
[353] "353" "354" "355" "356" "357" "358" "359" "360" "361" "362" "363"
```

```
[364] "364" "365" "366" "367" "368" "369" "370" "371" "372" "373" "374"
[375] "375" "376" "377" "378" "379" "380" "381" "382" "383" "384" "385"
[386] "386" "387" "388" "389" "390" "391" "392" "393" "394" "395" "396"
[397] "397" "398" "399" "400" "401" "402" "403" "404" "405" "406" "407"
[408] "408" "409" "410" "411" "412" "413" "414" "415" "416" "417" "418"
[419] "419" "420" "421" "422" "423" "424" "425" "426" "427" "428" "429"
[430] "430" "431" "432" "433" "434" "435" "436" "437" "438" "439" "440"
[441] "441" "442" "443" "444" "445" "446" "447" "448" "449" "450" "451"
[452] "452" "453" "454" "455" "456" "457" "458" "459" "460" "461" "462"
[463] "463" "464" "465" "466" "467" "468" "469" "470" "471" "472" "473"
[474] "474" "475" "476" "477" "478" "479" "480" "481" "482" "483" "484"
[485] "485" "486" "487" "488" "489" "490" "491" "492" "493" "494" "495"
[496] "496" "497" "498" "499" "500" "501" "502" "503" "504" "505" "506"
[507] "507" "508" "509" "510" "511" "512" "513" "514" "515" "516" "517"
[518] "518" "519" "520" "521" "522" "523" "524" "525" "526"

'data.frame':   526 obs. of  16 variables:
 $ Station.ID                   : Factor w/ 24 levels "D0","D1","D-1",..: 7 7 7 7 7 7 7 7 7 7 ...
 $ Transect                     : Factor w/ 2 levels "Down","Up": 1 1 1 1 1 1 1 1 1 1 ...
 $ Date                         : int  20140916 20140917 20140918 20140919 20140920 20140921 20140924 2
 $ Type                         : Factor w/ 2 levels "Control","Experiment": 1 2 2 2 1 2 2 2 2 2 ...
 $ T..C..in.situ                : num  22.5 23 23 24.5 24.6 ...
 $ Salinity                     : num  35.9 35.8 35.8 35.9 35.9 ...
 $ Alkalinity..umol.kg.         : num  2281 2253 2245 2189 2186 ...
 $ Rhodamine..ppb.              : num  0.0507 0.194 0.6468 0.3877 0.1912 ...
 $ Spec.pH..total.              : num  8.11 8.12 8.16 8.13 8.18 ...
 $ T..K..Spec.pH                : num  298 298 298 299 299 ...
 $ Alk..S.Normalized..umol.kg.  : num  2274 2248 2243 2182 2178 ...
 $ Rhodamine..S.Normalized..ppb.: num  0.0505 0.1935 0.6462 0.3864 0.1905 ...
 $ in.situ.pH..total.           : num  8.15 8.15 8.18 8.15 8.2 ...
 $ in.situ.pCO2..uatm.          : num  287 284 260 276 241 ...
 $ in.situ.CT..umol.kg.         : num  1932 1903 1879 1834 1801 ...
 $ in.situ.aragonite            : num  3.78 3.79 3.95 3.82 4.12 3.92 3.6 3.75 3.39 3.13 ...
```

**Exercise 7. Tidying-up the data (50 min)**

From the output of the previous commands it can be seen that there are 16 variables (columns). Each row refers to an observation. In this context, observations correspond to sampling stations where sets of measurements were taken in the reef-flat study area.

The first column of the `reef` data frame is the `Station.ID`, an ID reference that identifies each location. This ID is composed of two parts: (i) the first character, either U (referring to the upstream transect) or D (for downstream transect); (ii) the following chars indicate the position (in metres) of the sampling location relative to the tank. Since this information is pivotal for later analyses, it is useful to save the station positions in its own column, formatted as a numeric vector.

The following exercises (7.*) show one possible way of transforming the `Station.ID` column into a station position vector: (i) spliting the string in two parts (spliting the U or D from the numberic portion), (ii) extracting the position (second part) as a numeric vector and (iii) creating a new data frame (`reef2`) containing all original `reef` data plus the position as a new column. Try it for yourself, and make sure that you understand all the steps and code involved.

**Exercise 7.1**

Format the column names removing the extra dots between the text. To do this we will use the function `gsub` that finds patterns in text and replaces those patterns with other text (also called a string).

```
# Use gsub to substitute two consecutive dots with only one dot in the
# column names of reef Please run ?gsub to learn about regular expressions
# (regex) and how to use them.

colnames(reef) <- gsub("\\.\\.", "\\.", colnames(reef))
```

Extract the `Station.ID` column as a `character` vector.

```
station.id <- as.character(reef$Station.ID)
```

### Exercise 7.2

Use the `strsplit` function to split the ID string into its two relevant parts. The `split` argument indicates which characters are to be used to split the string. Please note that the characters used for the splitting are omitted (removed) leaving an empty string ("").

```
split.result.list <- strsplit(station.id, split = "[U,D]")
```

### Exercise 7.3

For each element in `station.id` we obtained two strings: the empty string `""` and a string with the position in meters. Since `strsplit` returns a list, we will *unlist* it to converts it to a character vector.

```
split.result.vector <- unlist(split.result.list)
```

### Exercise 7.4

Next we must remove the empty strings `""` in order to get the positions nicely arranged in a single vector.

```
station.position <- split.result.vector[split.result.vector != ""]
```

### Exercise 7.5

Since the positions are distances in meters, and we would like to use those values for future calculations, we must convert them from characters (text) to numeric.

```
station.position <- as.numeric(station.position)
```

### Exercise 7.6

Finally, to include the new vector in the reef data frame, we will create a new data frame (`reef2`) by combining columns (`cbind`) of the `reef` data frame with the newly created column (named `Station.Position`) as column number 2, followed by all other columns from the `reef` data frame (removing column 1 which is already present in position 1).

```
# create the new reef2 data frame by binding the first column of reef, with the
 # station.position vector and the rest of the reef data frame (without the 1st column)
reef2 <- cbind(reef[, 1], station.position, reef[,-1])

# check the column names attribute
names(reef2)
```

```
 [1] "reef[, 1]"                   "station.position"
 [3] "Transect"                    "Date"
 [5] "Type"                        "T.C.in.situ"
 [7] "Salinity"                    "Alkalinity.umol.kg."
 [9] "Rhodamine.ppb."              "Spec.pH.total."
[11] "T.K.Spec.pH"                 "Alk.S.Normalized.umol.kg."
[13] "Rhodamine.S.Normalized.ppb." "in.situ.pH.total."
[15] "in.situ.pCO2.uatm."          "in.situ.CT.umol.kg."
[17] "in.situ.aragonite"
```

```
# assign meaningfull column names to the first two columns
names(reef2)[c(1,2)] <- c("Station.ID","Station.Position")
# check the final column names
names (reef2)
```

```
 [1] "Station.ID"                  "Station.Position"
 [3] "Transect"                    "Date"
 [5] "Type"                        "T.C.in.situ"
 [7] "Salinity"                    "Alkalinity.umol.kg."
 [9] "Rhodamine.ppb."              "Spec.pH.total."
[11] "T.K.Spec.pH"                 "Alk.S.Normalized.umol.kg."
[13] "Rhodamine.S.Normalized.ppb." "in.situ.pH.total."
[15] "in.situ.pCO2.uatm."          "in.situ.CT.umol.kg."
[17] "in.situ.aragonite"
```

### Exercise 8. Exploring the data (1h30m)

#### Exercise 8.1

This reef experiment is a **case-control** study. How many observations (rows) are there for `Control` and `Experiment` days?

(*Hint*: `reef2$Type` contains dates which are Control or Experiment days; the `table` function discussed yesterday can be useful for counting).

(Answer: There are `166` and `360` observations for `Control` and `Experiment` days, respectively.)

#### Exercise 8.2

What is the time interval for this study?

(*Hint*: The column `Date` contains this information; `min`, `max` and/or `range` functions might help.)

(Answer: The study took place between 16/09/2014 and 10/10/2014.)

#### Exercise 8.3

This study comprises Control days (when no alkalinity is added to the solution pumped to the reef flat) and Experiment days (when 600 gram of NaOH is added). From the time interval of the study, how many days were "Control days", and how many days were "Experimental days"?

(*Hint*: `unique` and `length` functions will be useful. The **Date** and **Type** columns are pivotal).

(Answer: 7 were control days and 15 were experimental days.)

**Exercise 8.4**

Measurements were taken along two transects: upstream and downstream of the reef flat (Figure 3). Compare the spreading of the locations of the sampling stations up- and downstream of the reef flat.

(i) Are the two spreads alike?
(ii) Do you think there is an experimental design reason that explains this difference?
(iii) The standard deviation and the inter-quartile range seem to give contradictory results on which transect has its sampling stations more spread out. Which metric, in your opinion, best reflects your visual impression of spread?

```
# Assessing the spread of stations' positions at the upstream transect by
# looking at the standard deviation, inter-quartile range and range

# Upstream transect
up.pos <- unique(reef2$Station.Position[reef2$Transect == "Up"])
# mean position of the upstream transect stations' positions
mean(up.pos)
```

```
[1] 0
```

```
# standard deviation of the upstream transect stations' positions
sd(up.pos)
```

```
[1] 8.284021
```

```
sqrt(var(up.pos))  # the standard deviation is the square root of the variance!
```

```
[1] 8.284021
```

```
# inter-quartile range of the upstream transect stations' positions
IQR(up.pos)
```

```
[1] 3
```

```
# range of the upstream transect stations' positions
range(up.pos)
```

```
[1] -16  16
```

```
# Downstream transect
dn.pos <- unique(reef2$Station.Position[reef2$Transect == "Down"])
# mean position of the upstream transect stations' positions
mean(dn.pos)
# standard deviation of the downstream transect stations' positions
sd(dn.pos)
# inter-quartile range of the downstream transect stations' positions
IQR(dn.pos)
# range of the downstream transect station positions
range(dn.pos)
```

**Answer:** The spread, as measured by the standard deviation $\sigma$, is surprisingly similar: upstream is $\sigma$~8.3 and downstream is $\sigma$~7.96. Accordingly, judging by the standard deviation alone, one might think that the sampling stations would be slightly more spread out along the upstream transect. However, judging from the picture, this contradicts our intuition, since the majority of upstream stations are very close to the centre. This could be explained by the fact that the standard deviation is known to be very sensitive to outliers; however, both transects have "outlier" stations at the edges: in positions -16 and 16 metres, as one can observe from the output of `range`. For this case, the inter-quartile range (IQR) proves to be a more robust

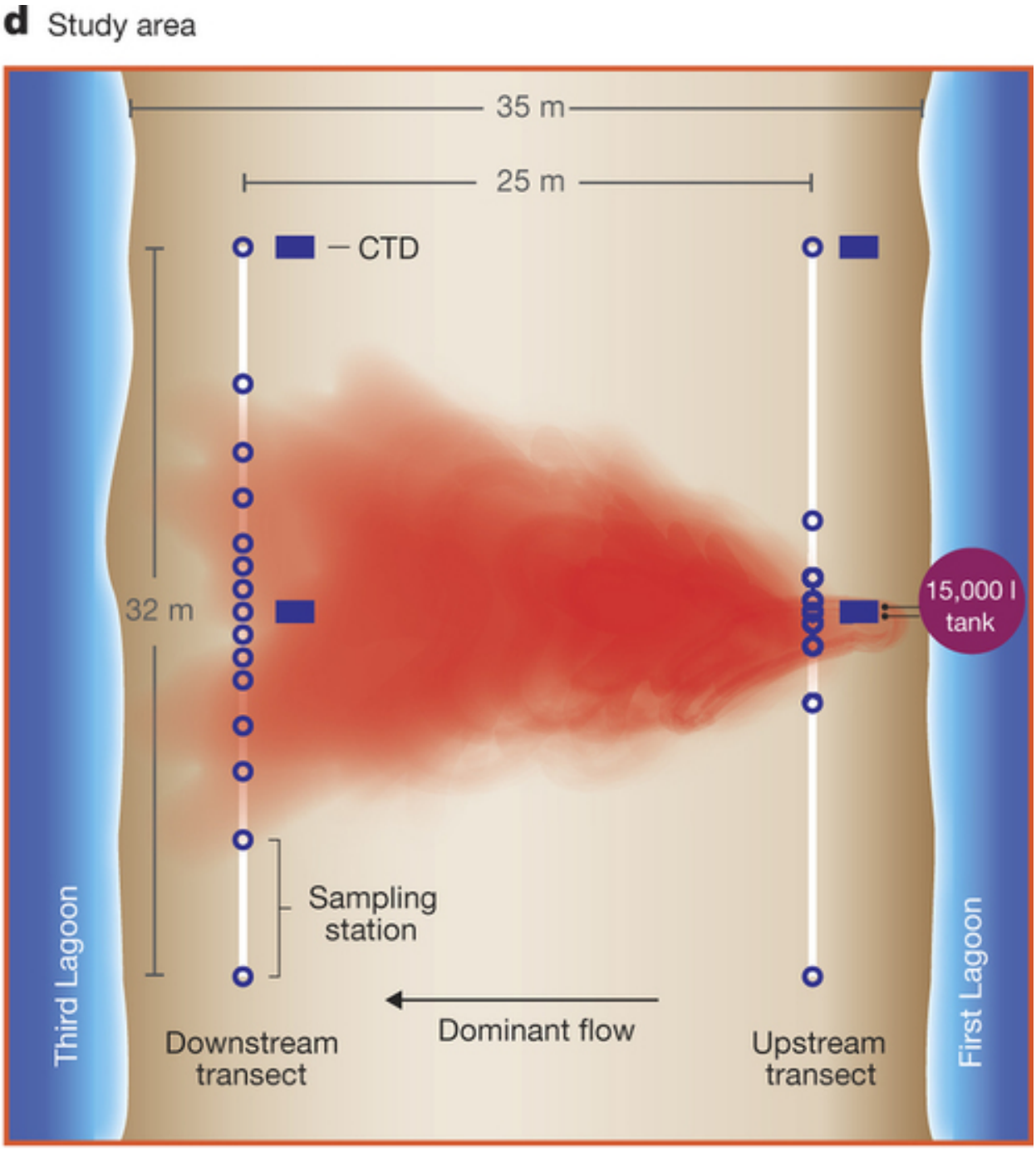



Figure 3: **Figure 3** | Sampling stations' locations (blue circles).

metric (IQR=3 upstream; IQR=8 downstream), working best at mathematically describing our intuition when observing Figure 3. This difference in spread between the two transects was probably a choice taken during the experimental design phase, reflecting the antecipated mixing and dilution of the solution as it flowed from upstream to downstream. Therefore it made sense to concentrate the sampling effort close to the source (upstream) and spread it out more at the downstream transect.

**Exercise 8.5**

`summary` is a very useful function that outputs the summary statistics for an R object. Try it on a subset of the `reef2` data frame: run `summary` for the variables: `Date`, `Type`, `Station.Position` and `Transect`. The output should look like this:

```
      Date                   Type     Station.Position    Transect
 Min.   :20140916   Control   :166   Min.   :-16.00000   Down:328
 1st Qu.:20140921   Experiment:360   1st Qu.: -3.75000   Up  :198
 Median :20140928                    Median :  0.00000
 Mean   :20140960                    Mean   : -0.01141
 3rd Qu.:20141005                    3rd Qu.:  3.00000
 Max.   :20141010                    Max.   : 16.00000
```

Appreciate how the output is differently presented for `Date` and `Station.Position` compared to `Type` and `Transect`. **Why is it differently presented?**

**Exercise 9. Exploring the data graphically (1h30m)**

By default, R base alone allows the plotting of several, highly customizable graphics. There are however many graphical packages developed by the community that greatly expand its plotting potential (e.g., ggplot2). Nevertheless, in this tutorial we will focus only on a few of the most common plots that can be generated with functions included in the base installation of R.

R graphics are created using a series of high- and low-level plotting commands. High-level commands create new plots via functions such as `plot`, `hist`, `boxplot`, or `curve`, whereas low-level functions add to an existing plot created with a high-level plotting function; examples are `points`, `lines`, `text`, `axis`, `arrows`, etc..

Graphical parameters are customizable via the function `par`, containing over 70 different customizable fields (for details, see `?par`). In this exercise we will look into a few of the common plotting functions: `plot`, `hist`, `boxplot` and `curve`; as well as several parameters that allow you to tweak the look 'n feel of your graphics.

**Exercise 9.0 Introduction**

Ocean acidification is the ongoing decrease in the pH of the Earth's oceans, caused by the uptake of carbon dioxide ($CO_2$) from the atmosphere. Seawater is slightly alkaline (pH $> 8$), and this acidification is a shift towards less alkaline conditions rather than acidic conditions (pH $< 7$). An estimated 30–40% of the carbon dioxide from human activity released into the atmosphere dissolves into oceans, rivers and lakes. To achieve chemical equilibrium, some of it reacts with the water to form carbonic acid. Some of these extra carbonic acid molecules react with a water molecule to give a bicarbonate ion and a hydronium ion, thus increasing ocean acidity (H+ ion concentration).

Aragonite is a carbonate mineral, one of the two common, naturally occurring, crystal forms of calcium carbonate, $CaCO_3$ (the other form being the mineral calcite). $CaCO_3$ saturation state $\Omega_{arag}$ was one of the chemical parameters measured at the sampling stations.

The saturation state of seawater with respect to aragonite can be defined as the product of the concentrations of dissolved calcium and carbonate ions in seawater divided by their product at equilibrium:

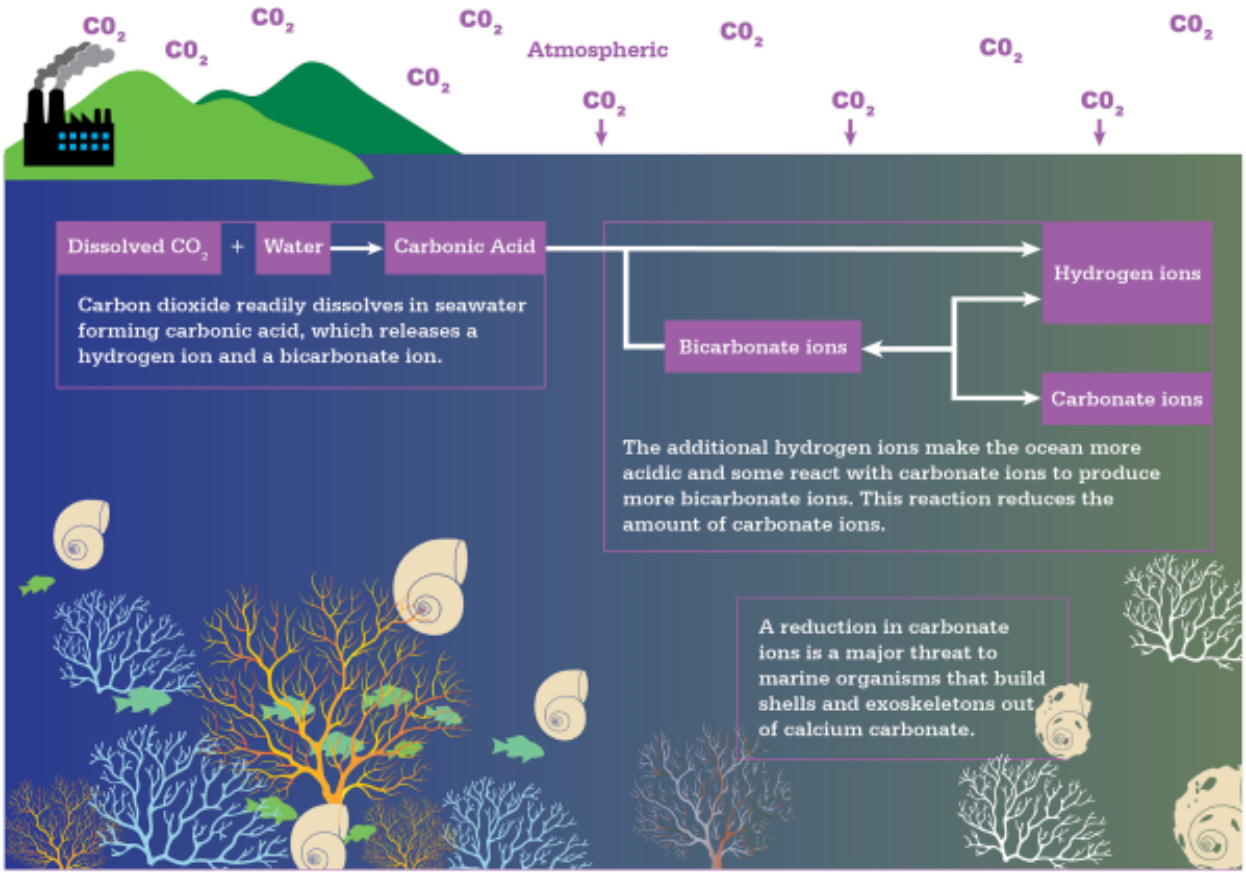


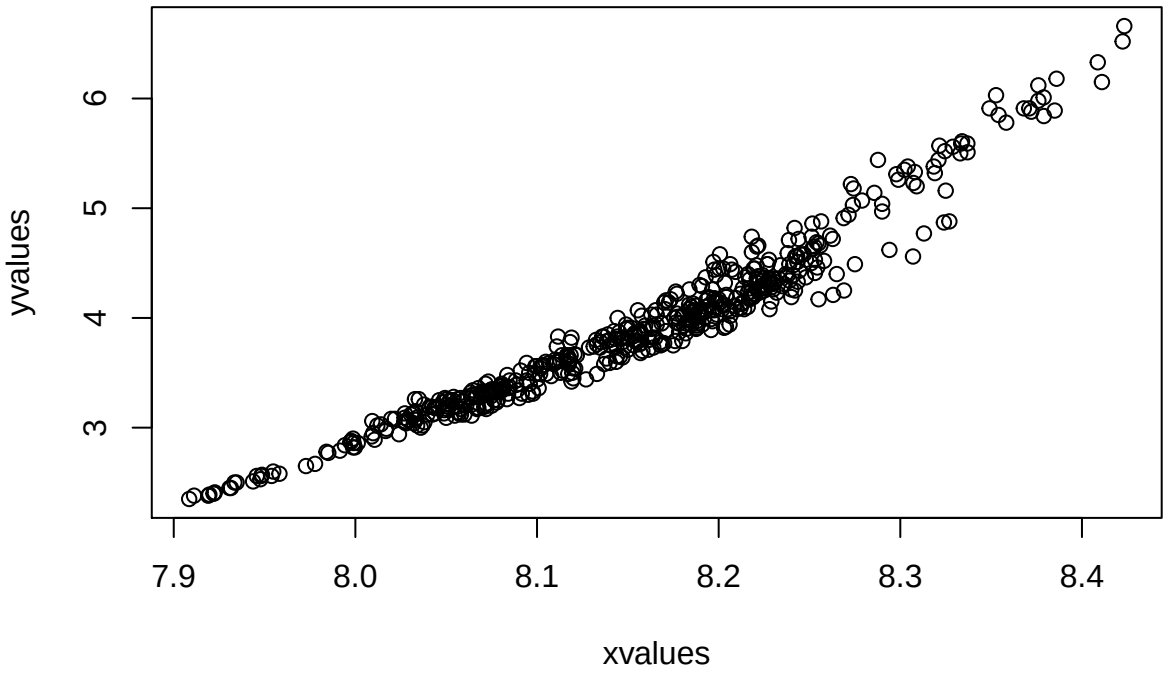

Figure 4: **Figure 4** | Ocean acidification and the resulting reduction in carbonate ions (climatecommission.angrygoats.net).

$$\Omega_{\text{arag}} = \frac{[\text{Ca}^{2+}][\text{CO}_3^{2-}]}{[\text{CaCO}_3]}$$

**Exercise 9.1 Scatterplots: Basic plotting with `plot`**

Lets see if the Albright et al. study recapitulates the reported effect of acidic conditions leading to lower levels of $CaCO_3$ (corresponding to a lower $\Omega_{\text{arag}}$). To this end we will plot the aragonite saturation state $\Omega_{\text{arag}}$ (`reef2$in.situ.aragonite`) vs pH (`reef2$in.situ.pH.total.`).

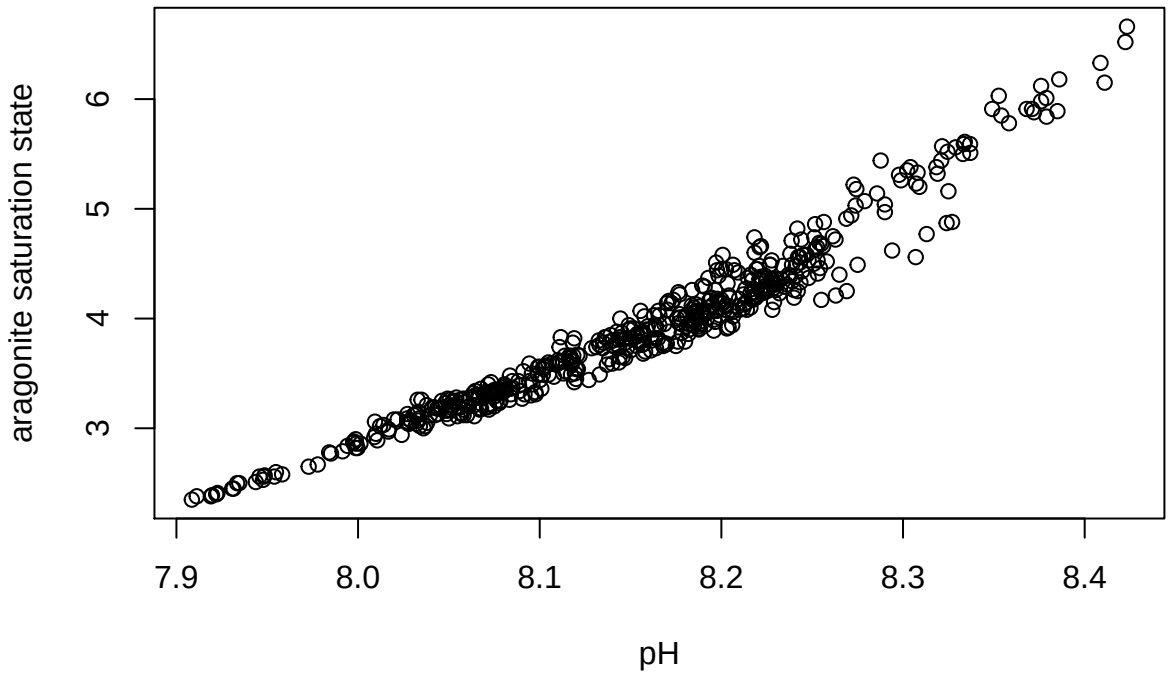
```
# xvalues: pH
xvalues <- reef2$in.situ.pH.total.
# yvalues: aragonite saturation state
yvalues <- reef2$in.situ.aragonite

plot(xvalues, yvalues)
```




From the generated plot it is clear that the two variables are indeed correlated. The higher the pH, the higher the $\Omega_{\text{arag}}$. Notice how the axes' labels were automatically set based on the name of the variables passed as arguments to the `plot` function.

To change the axes' labels, you may specify them explicitly by setting `plot`'s arguments: `xlab` and `ylab`.

Generate the following plot:




**Exercise 9.1.1 More parameters for `plot`**

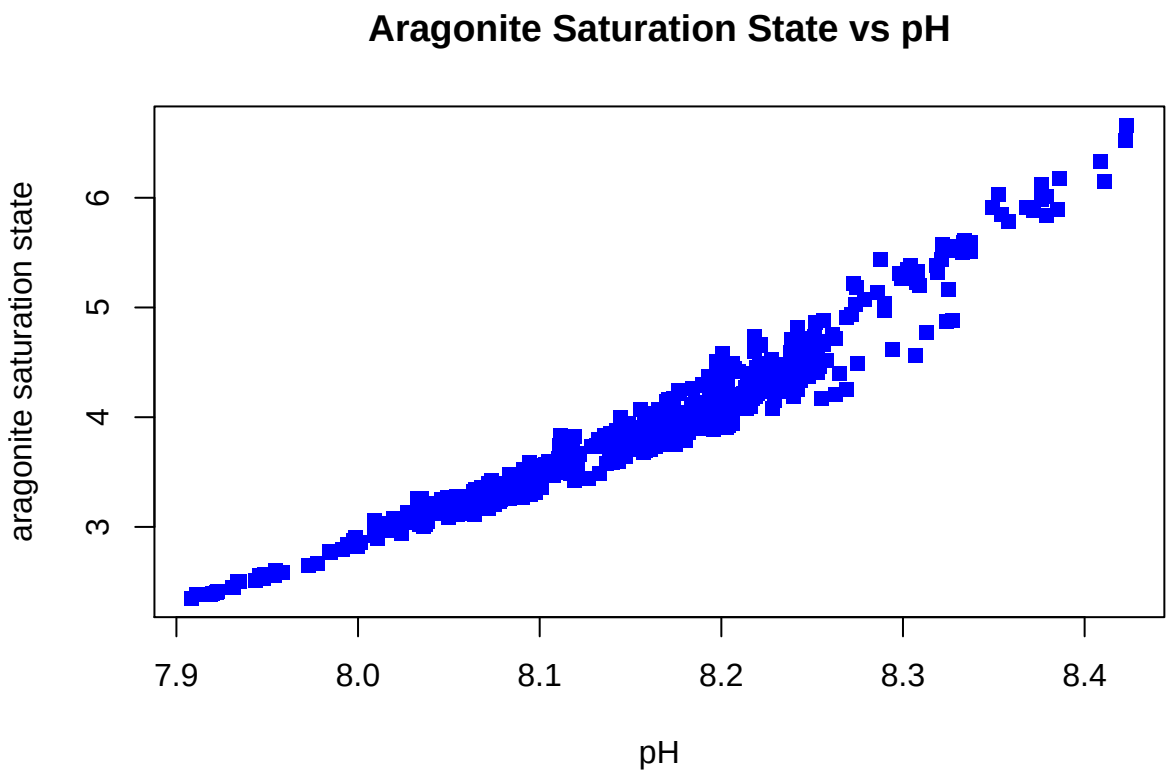
There are many parameters that allow you to customise your plot (see `?par`). Here are some of the most commonly used:

| Argument | Description |
|---|---|
| `main` | an overall title for the plot |
| `type` | what type of plot should be drawn: `"p"` points, `"l"` lines, `"n"` no plotting (see `?plot`) |
| `sub` | a sub title for the plot |
| `xlab` | a title for the x axis |
| `ylab` | a title for the y axis |
| `asp` | the y/x aspect ratio |
| `cex` | plotting text and symbols magnification factor relative to the default |

| Argument | Description |
|---|---|
| `cex.axis` | magnification to be used for axis annotation relative to the current setting of cex |
| `axes` | whether to draw axes (`TRUE`) or not (`FALSE`) |
| `xlim` | x axis range (should be a vector of two numbers: xmin and xmax, respectively) |
| `ylim` | y axis range (should be a vector of two numbers: ymin and ymax, respectively) |
| `pch` | either an integer or a single character to be used as the defaults symbol in plotting points (see `?points`) |
| `col` | set plotting color of each point (see named colors with `colors()`) |

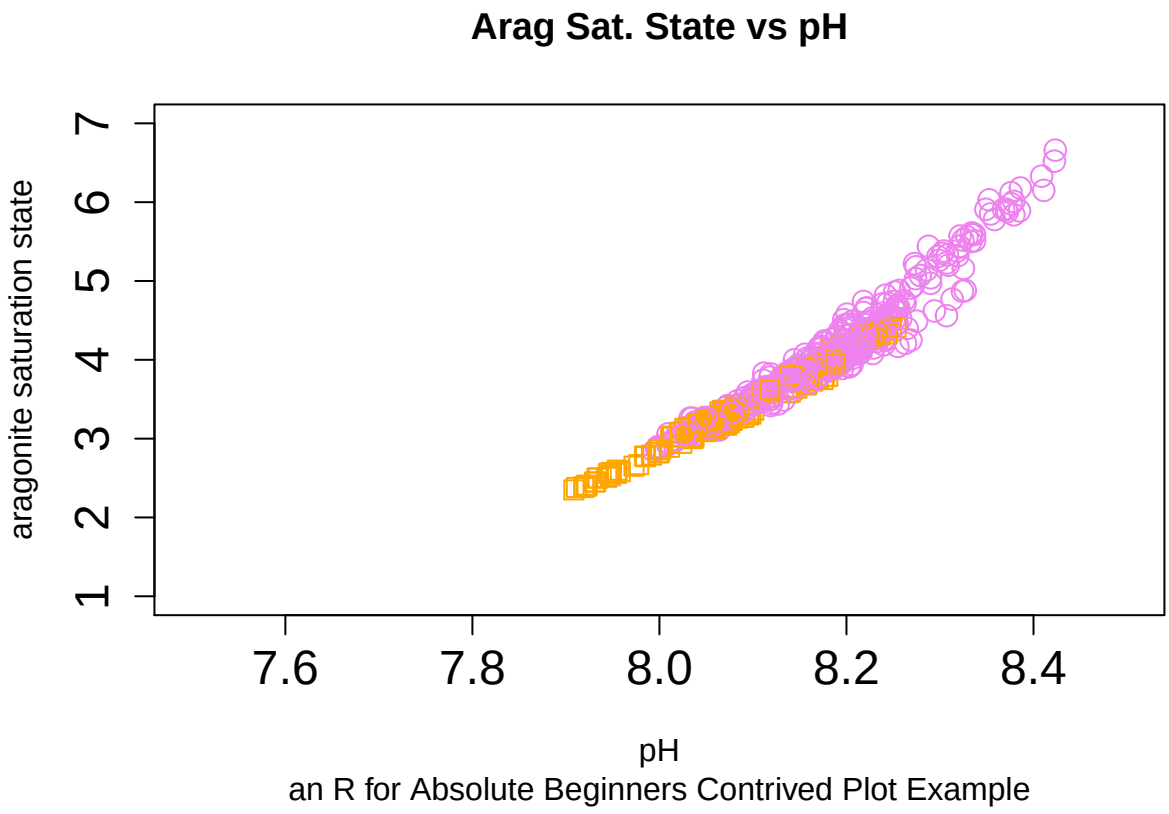
*Note:* The *demo("graphics")* command shows examples of available plots in R, together with the R code that can be used to generate it. The *colors ()* command shows the names of the available colors.

Try them out! Start by adding a main title, changing the type of points and their color.




Here is a contrived example using many parameters at once.

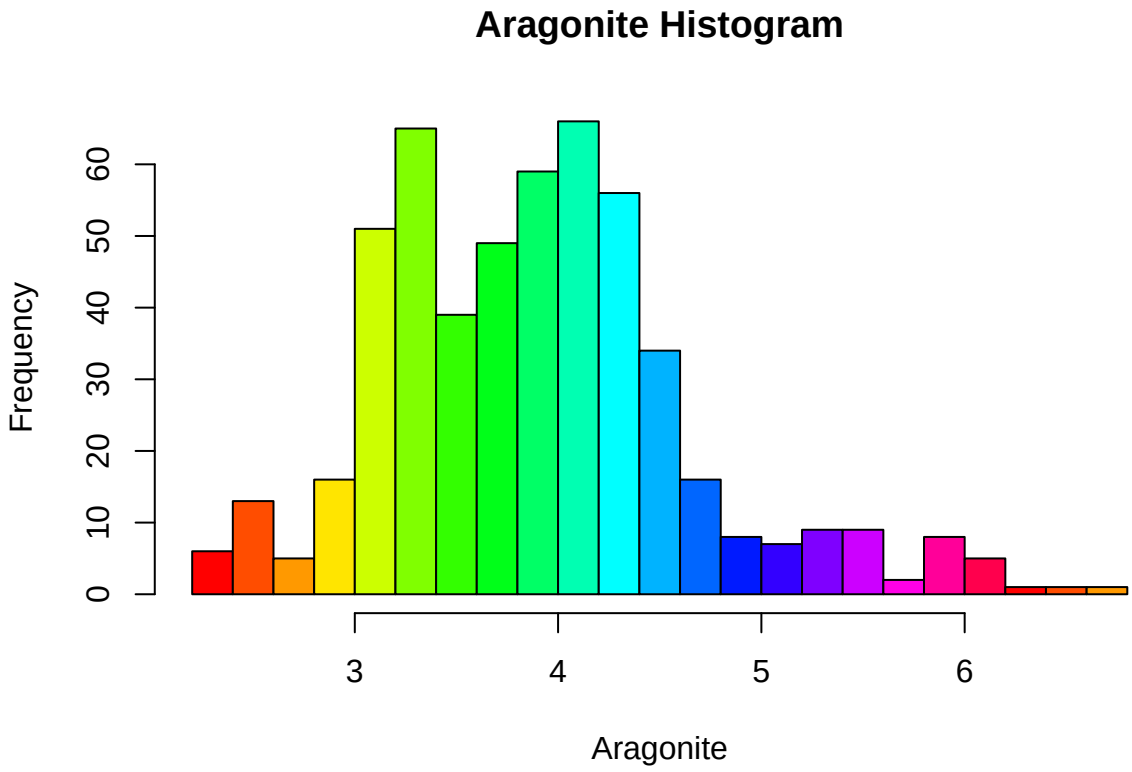
```
# define logical vector based on experiment type: TRUE ("Control"), FALSE ("Experiment")
type.logical <- reef2$Type == "Control"
# check if "violet" is a named color: "violet" %in% colors()
# define colors according to experiment type (either "Control" or "Experiment")
plot.colours <- ifelse(type.logical, "orange", "violet")
# define type of symbol for plotting points (see more options with ?points)
plot.points <- ifelse(type.logical, 22, 1)

# draw contrived plot example
plot(xvalues, yvalues, xlab = "pH", ylab = "aragonite saturation state",
     main = "Arag Sat. State vs pH", sub = "an R for Absolute Beginners Contrived Plot Example",
     xlim = c(7.5, 8.5), ylim = c(1, 7), cex = 1.5, cex.axis = 1.5, pch = plot.points,
     col = plot.colours)
```



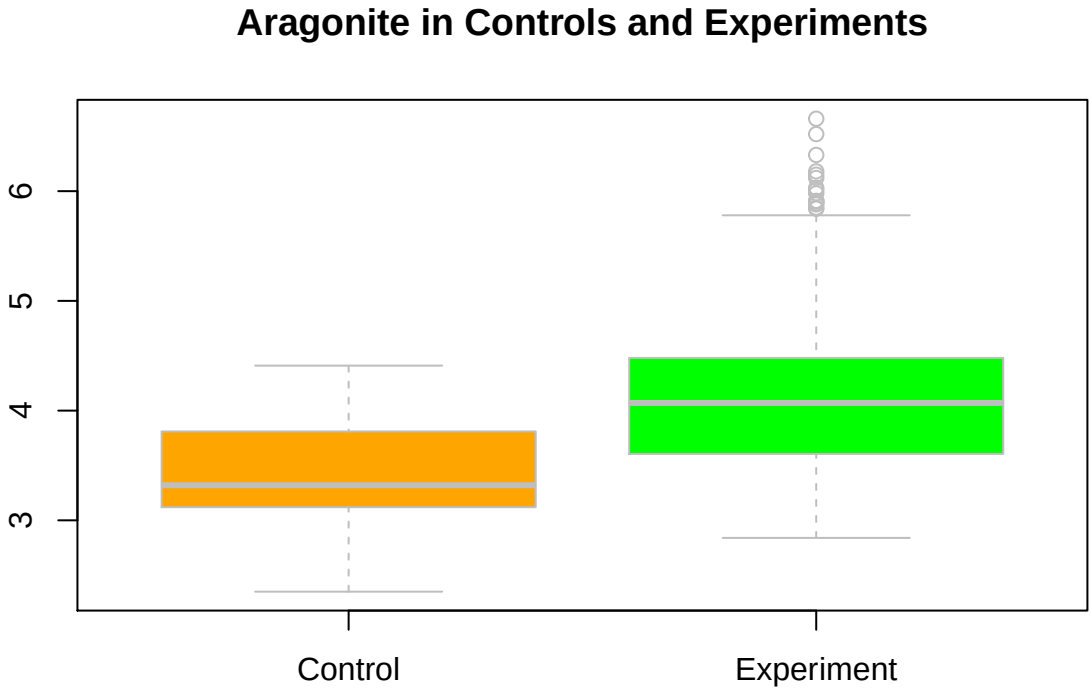

### 9.2 Histograms and Boxplots

The function *hist ()* shows the frequency (number of occurrences) of each observation; and the function *boxplot ()* shows the distribution of the occurrences in each category (agegp, alcgp and tobg).

```
# basic histogram, with labels, title and bars each with a different color
# (rainbow function), using ~ 20 breaks
hist(reef2$in.situ.aragonite, breaks = 20, xlab = "Aragonite", main = "Aragonite Histogram",
    col = rainbow(20))
```




```
# basic boxplot of the cases per age group
boxplot(reef2$in.situ.aragonite ~ reef2$Type, main = "Aragonite in Controls and Experiments",
    border = "gray", lwd = 1, col = c("orange", "green"))
```

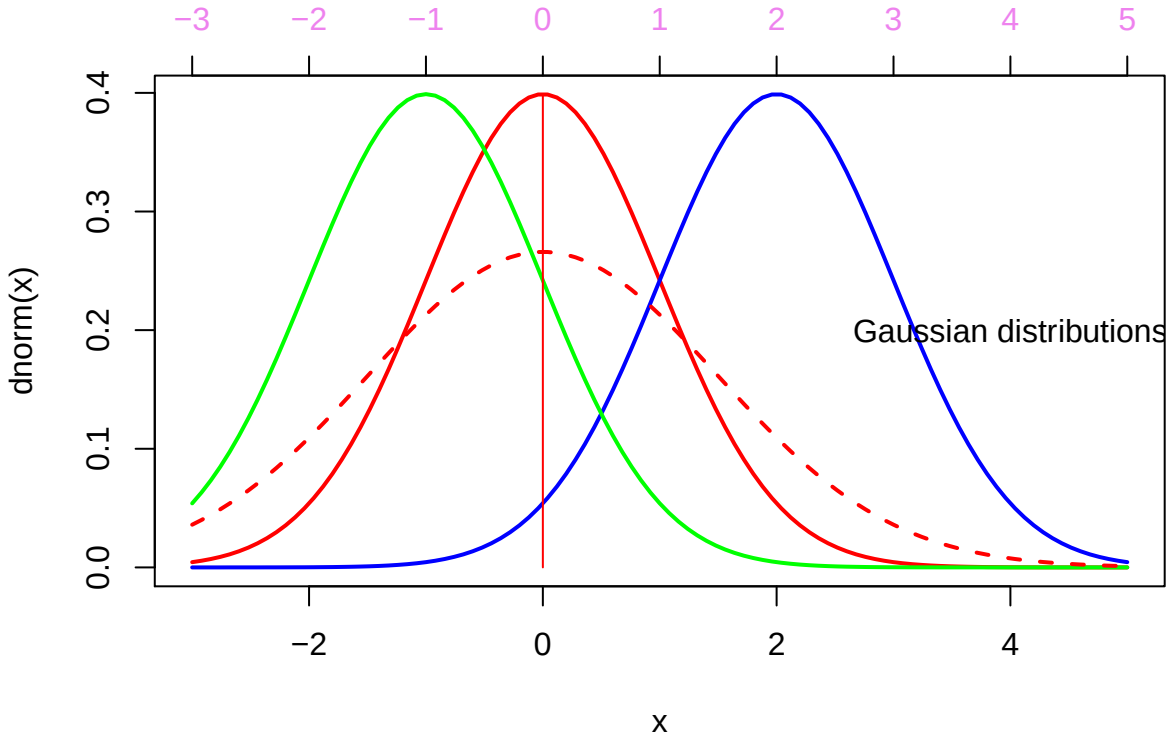



### 9.3 Curves

These are continuous plots (usually of known statistical distributions, like the Gaussian (dnorm), gamma,

beta, etc). Here we will see how to add lines and text to the plot (in specific locations/coordinates), as well as an extra axis on top with a different color.

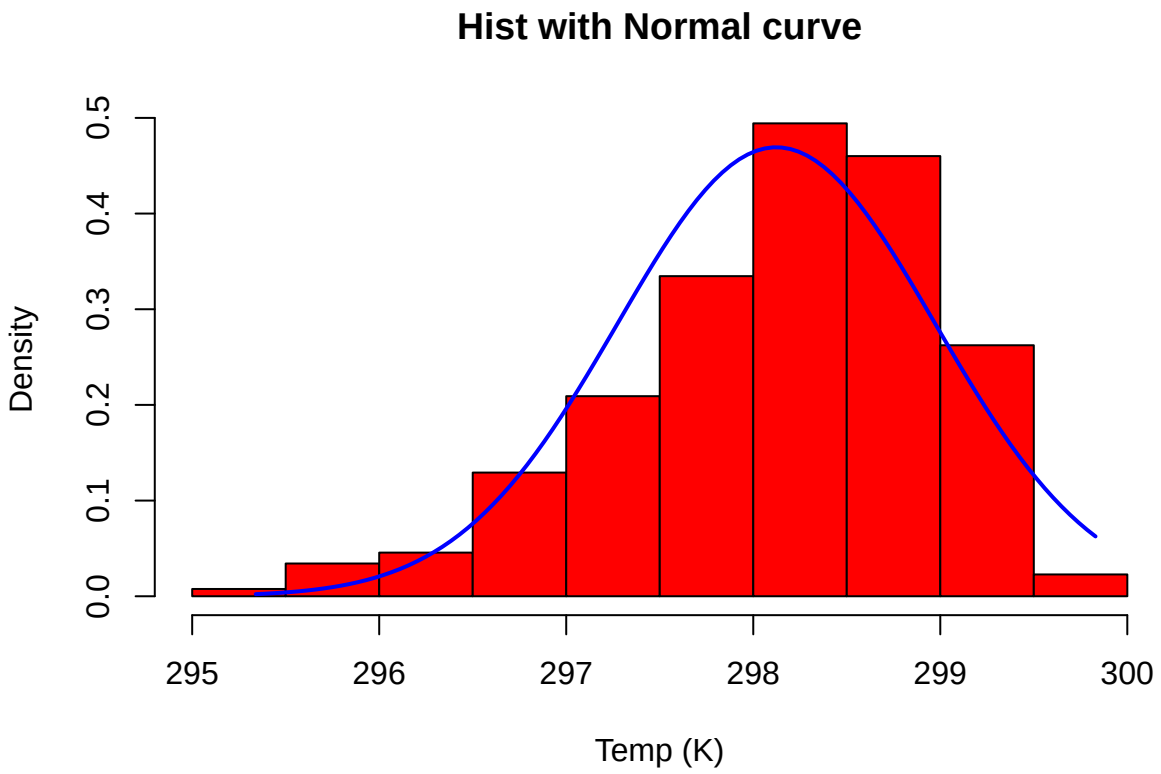
```
# multiple normal distribution curves, different mean and sd, and plot them
# in the same plot (add = TRUE)
curve(dnorm, from = -3, to = 5, lwd = 2, col = "red")
curve(dnorm(x, mean = 2), lwd = 2, col = "blue", add = TRUE)
curve(dnorm(x, mean = -1), lwd = 2, col = "green", add = TRUE)
curve(dnorm(x, mean = 0, sd = 1.5), lwd = 2, lty = 2, col = "red", add = TRUE)
# add a vertical line at the mean of the standard 'red' distribution
lines(c(0, 0), c(0, dnorm(0)), lty = 1, col = "red")
# add free text to the plot, in coordinates x=4, y=0.2
text(4, 0.2, "Gaussian distributions")
# add extra axis, on top (side 3), from -3 to 5, with tick-marks from -3 to
# 5, and colored violet
axis(3, -3:5, seq(-3, 5), col.axis = "violet")
```




### 9.4 Multiple Graphs

Different types of graphs can be combined in the same plotting area. Start by trying to plot an histogram of the temperature (in Kelvin) together with the gaussian distribution that best fits it (with appropriate mean and standard deviation).

```
# plot the histogram
hist(reef$T.K.Spec.pH, col = "red", xlab = "Temp (K)", freq = F, main = "Hist with Normal curve")
# calculate the x and y values
temp.x <- seq(min(reef$T.K.Spec.pH), max(reef2$T.K.Spec.pH), length = 100)
temp.y <- dnorm(temp.x, mean = mean(reef2$T.K.Spec.pH), sd = sd(reef2$T.K.Spec.pH))
# plot the normal curve using the function lines
lines(temp.x, temp.y, col = "blue", lwd = 2)
```



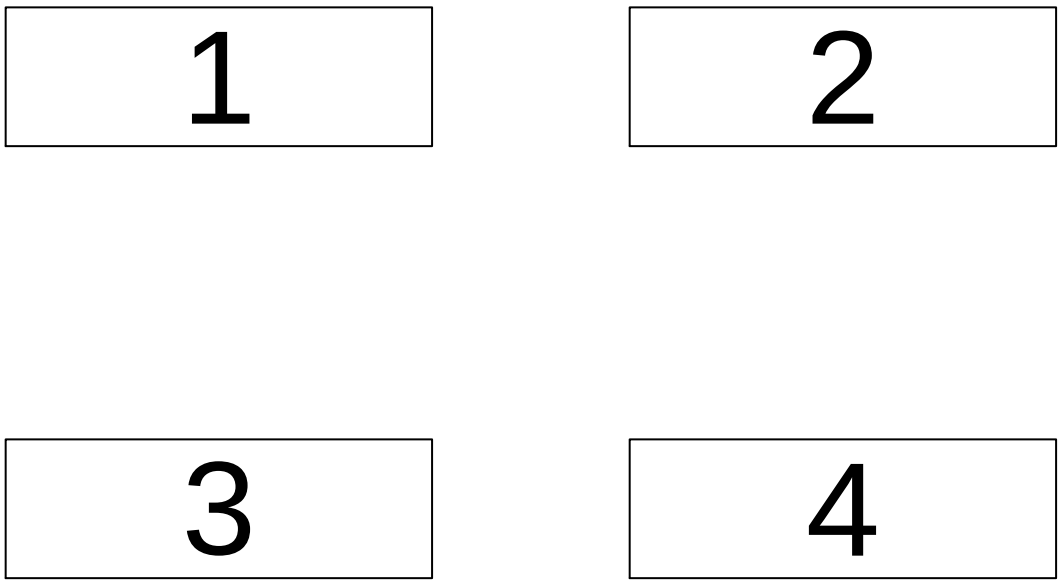

**Exercise 9.4.1 Arranging several plots in one page**

To create a page with several plots located in side-by-side panels, we can use the function `par` with one of the following parameters: `par(mfrow=c(r,c))` or `par(mfcol=c(r,c))`. `mfrow` adds images per line, from left to right, and `mfcol` adds per column, from top to bottom.

Make sure you understand how `mfrow` parameter is specifying the order by which the plots fill up the layout.

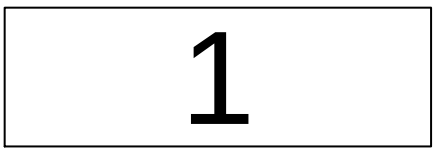
```
# make a 2 by 2 array of plot panels
# fill up row by row
par(mfrow = c(2,2))
# create a new plot, type="n" means plot none
# first plot
#plot(c(0.5),type="n", axes = FALSE, ann=FALSE)
plot.new()
text(0.5, 0.5, "1", cex = 5)
box()
# second plot
plot.new()
#plot(c(0.5),type="n", axes = FALSE, ann=FALSE)
text(0.5, 0.5, "2", cex = 5)
box()
# third plot
plot.new()
#plot(c(0.5),type="n", axes = FALSE, ann=FALSE)
text(0.5, 0.5, "3", cex = 5)
box()
# fourth plot
plot.new()
#plot(c(0.5),type="n", axes = FALSE, ann=FALSE)
```

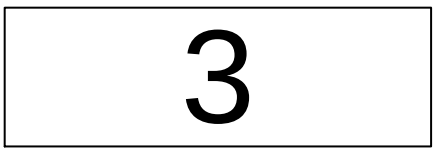
```
text(0.5, 0.5, "4", cex = 5)
box()
```

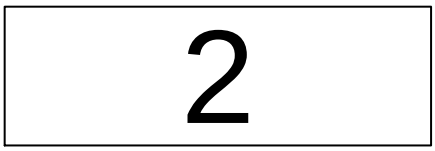



Here is the same example but with `mfcol`.

```
# make a 2 by 2 array of plot panels
# fill up column by column
par(mfcol = c(2,2))
# create a new plot, type="n" means plot none
# first plot
plot(c(1),type="n", axes = FALSE, ann=FALSE)
text(1, 1, "1", cex = 5)
box()
# second plot
plot(c(1),type="n", axes = FALSE, ann=FALSE)
text(1, 1, "2", cex = 5)
box()
# third plot
plot(c(1),type="n", axes = FALSE, ann=FALSE)
text(1, 1, "3", cex = 5)
box()
# fourth plot
plot(c(1),type="n", axes = FALSE, ann=FALSE)
text(1, 1, "4", cex = 5)
box()
```



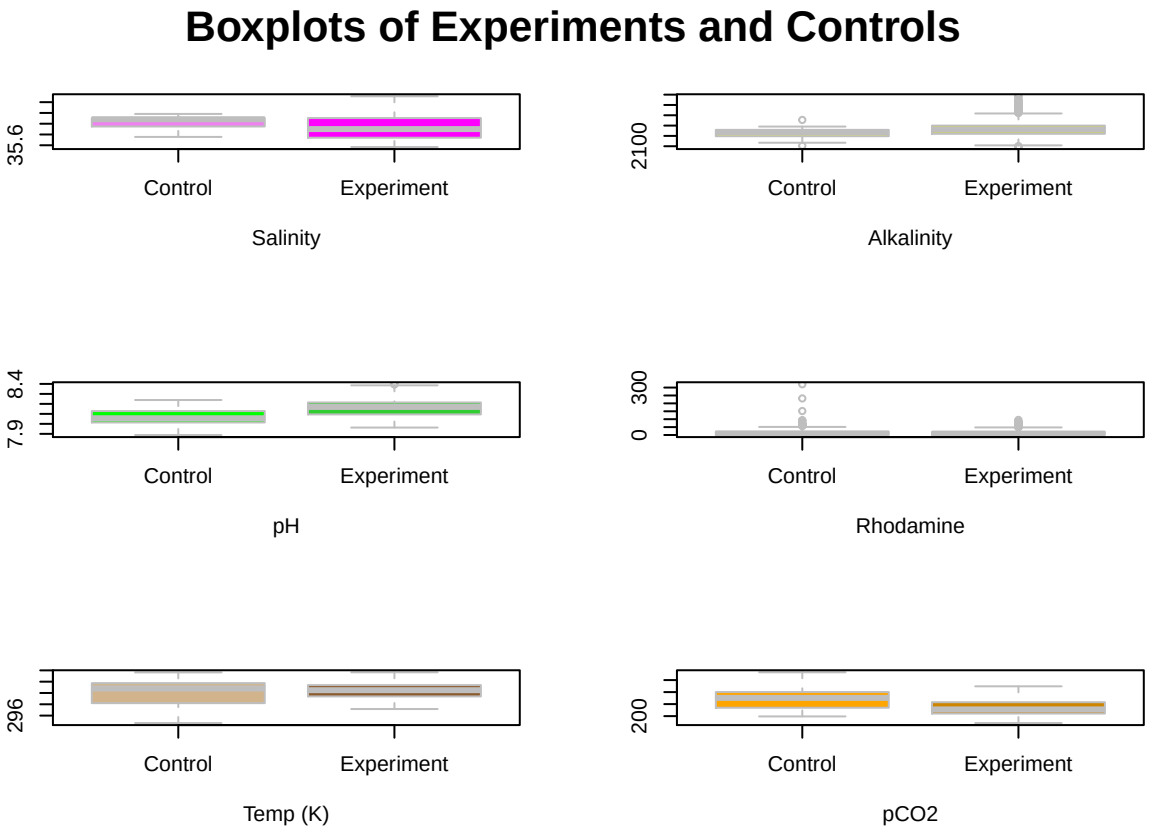


Once terminated the panel plots, we must revert the graphical parameters to its default values, so that we can go back to plotting one chart per page.

```
# reset the graphical display parameters to 1 row and 1 column
par(mfrow = c(1, 1))
```

Now lets try to plot the real data from our tutorial dataset.

```
# set the graphical display parameters to 3 rows and 2 columns
par(mfrow = c(3, 2))  # mfrow adds plots per row, from left to right
# draw boxplots for experiments and controls, per each group
boxplot(reef2$Salinity ~ reef2$Type, xlab = "Salinity", border = "gray", lwd = 1,
    col = c("violet", "magenta"))
boxplot(reef2$Alkalinity.umol.kg. ~ reef2$Type, xlab = "Alkalinity", border = "gray",
    lwd = 1, col = c("yellow", "yellow2"))
boxplot(reef2$Spec.pH.total. ~ reef2$Type, border = "gray", xlab = "pH", lwd = 1,
    col = c("green", "limegreen"))
boxplot(reef2$Rhodamine.ppb. ~ reef2$Type, border = "gray", xlab = "Rhodamine",
    lwd = 1, col = c("blue", "lightskyblue"))
boxplot(reef2$T.K.Spec.pH ~ reef2$Type, border = "gray", xlab = "Temp (K)",
    lwd = 1, col = c("tan", "tan4"))
boxplot(reef2$in.situ.pCO2.uatm. ~ reef2$Type, border = "gray", xlab = "pCO2",
    lwd = 1, col = c("orange", "orange3"))
# add a title outside of the plotting area
title("Boxplots of Experiments and Controls", outer = TRUE, line = -2, cex.main = 2)
```




**Exercise 9.5 Export and save plots**

RStudio allows the visualization of the plots before exporting/saving them to an image file (i.e. a bitmap such as a `jpeg` or `png` which becomes pixelated when zoomed in), or as `pdf` or `svg` (vectorial formats that can be zoomed and stretched to infinity, without loosing image quality). However, pdf files don't always export correctly, so they require some "trial and error" until one can get the "perfect" image.

To save plots you can quickly go to the *Plots* tab in the Workspace window and click on *Export*. However, this approach is not easily reproducible and can be time-consuming if the plots need to be generated many times. Alternatively, in order to save the plots programmatically, you can just wrap the plotting functions between two commands: `pdf` (or other depending on file format desired) and `dev.off`.

```
# pdf(...) or jpeg(...), png(...), svg(...), etc..

# Start graphics device
pdf("aragonite_ctrl_exp.pdf", width=7, height=5)
# draw boxplots for experiments and controls, per each group
boxplot (reef2$in.situ.aragonite ~ reef2$Type, main="Aragonite in Controls and Experiments",
         border="gray", lwd=1, col=c("orange","blue"))
dev.off () # close graphics device (stop writing to file)
```

Save one of the previous plots as pdf and open it with your pdf viewer (e.g. Acrobat Reader).

**Exercise 10. Extra study**

**Exercise 10.1 Write a function of your own**

To assess the fraction of alkalinity taken up by the reef, a passive tracer, i.e. the non-reactive dye Rhodamine WT, was mixed with ambient sea water in the tank. Rhodamine WT concentration was then measured fluorometrically. Given that this measurement is temperature dependent, it needs to be corrected. The following formula provides this correction:

$$F_r = F_s e^{k(T_s - T_r)}$$

where $F_r$ and $F_s$ are the fluorescences at the reference and sample temperatures, $T_r$ and $T_s$ (in Kelvin), and $k = 0.026$ per Kelvin (2.6% correction per Kelvin).

**Challenge:** Write a function named `f.r` that returns the $F_r$ value given as input $F_s$, $T_r$ and $T_s$. Also, include an argument in the function that allows changing the accepted temperature units from Kelvin (default) to Celsius.

Use your function `f.r` to calculate the temperature-corrected Rhodamine concentrations. *Hint*: $F_s$ is `reef2$Rhodamine.ppb.`, $T_r$ and $T_s$ are `T.C.in.situ` and `T.K.Spec.pH`, respectively. Plot the calculated temperature-corrected Rhodamine concentrations versus `reef2$Rhodamine.S.Normalized.ppb.`.

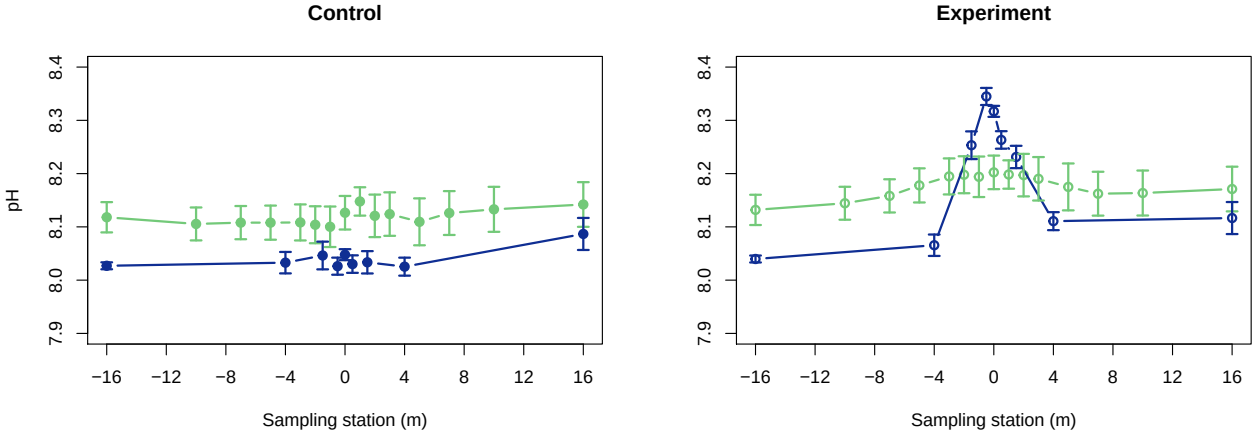
### Exercise 10.2 Scatterplots with Error Bars

R base does not provide plots with error bars. However, it is easy (however laborious) to take advantage of the `arrows` function to mimic the same effect.

```
arrows(x, avg - sdev, x, avg + sdev, length = 0.05, angle = 90, code = 3)
```

In the code above `x` is a vector of x-positions, and `avg-sdev` and `avg+sdev` are vectors of the lower and upper y-positions of the error bars.

Using many of the elements you have seen today try to generate this set of two plots. See how these plots compare with those of *Figure 2c-d* from Albright et al., 2016.




END