Intro to modeling in R - Session 1

Daniel Viana and Jessica Zamborain Mason

October, 2022

Table of Contents

Bootcamp overview

Welcome to the R bootcamp in Applied Marine Data Science. During this bootcamp we will introduce you to R and RStudio, show you how to input and wrangle data, and introduce you to the analysis and visualization of marine data.

This course will start from the beginning, asusming you have never used R before. If you have, that is great! but the first day may serve as a reminder of the basics.

Programming can sometimes be feel frustrating and tedious. We acknowledge immersing yourselves into computer programming can be a steep learning curve (we have been there…), but believe us when we say it is totally worth it! We are here to help you navigate this process so please always ask if you do not understand a step.

Let's get started!

Installing R and RStudio on your computer

If you haven't already, let's start by downloading R and RStudio on your computers.

Throughout this course we will use R through RStudio.

R is an open-source project of programming language, that is easily available for free download via the web. First, go to https://cran.r-project.org/ and click on the version of R that is suitable for your computer (e.g., Windows or Mac). If you pressed on the Mac link, press on the first link that appears after "Latest release:" (e.g., R-4.1.2.pkg). If you pressed on the Windows link, press the link names "base" just under "Subdirectories:". In the next

page, press on the link to download the latest version fo R (e.g., Download R 4.1.2 for Windows). R will start downloading on to your computer. Once it has finished downloading, open it and go through the installation process (.exe file in Windows). follow the default for your system type. For some, it will make you choose between 32 and 64-bit, which refers to the way a computer processor handles information.You can access your computer's system type by searching for "System information" in your windows search icon.

RStudio is an integrated development environment that uses R (some classify it as a more "user friendly" way of using R). It is also easily available for free download via the web. To download RStudio, go to https://www.rstudio.com/products/rstudio/download/. Click the download button under the RStudio Desktop Open Source License (Free). Then, under "All installers", choose the link that matches your computer (e.g., Windows or Mac). RStudio will download to your computer. Open it and go through the installation process (.exe file for Windows).

Congratulations! You now have R and RStudio installed on your computer. Now, we can start programming!

Starting RStudio and performing interactive calculations

Before opening RStudio, let's look briefly at R itself. Search for R within the applications of your computer (probably an R icon on your desktop). Click on it and R will open. When R is launched it opens the console window automatically. This has a few basic menus at the top, which you can check out later on your own. The console window is also where you enter commands (where the arrow is ">") for R to execute interactively, meaning that the command is executed and the result is displayed as soon as you hit the Enter key on your keyboard. For example, type:

```
2+2
```

```
## [1] 4
```

and press enter, R will return the value of 4. R "heard" you, understood what you wanted when you said "2+2", carried out the addition, and told you the answer.

Let's move on to RStudio. To do so, first we will close R. We do this by typing "q()" and pressing Enter.

When you quit R, it will ask you if you want to save the workspace (that is, all of the variables you have defined in this session); for now (and in general), say "no" in order to avoid clutter.

Similar to how we opened R, now we open RStudio (click the RStudio icon on your desktop or search for RStudio in your applications). On the left side, you may find something familiar. That is right, it is the console window, just like the one we saw in R.

However, in contrast to R, in RStudio you have other panels. On the bottom right, you will likely see some files. This is your working directory (where files related to the project are or will be stored). To check your working directory, go to the command window and type:

```
getwd()

## [1] "/Users/danielfviana/Rbootcamp_Madagascar/Intro sessions"
```

This tells you what RStudios's working directory is. Any outputs you generate should appear in this directory.

On the top right you have the environment. This is where you will see the data objects you create. Currently it is empty because we have not created any objects. Let's create one. In the console window, create a variable x that is equal to 2+2 and press Enter:

```
x=2+2
```

You will see that now your environment has one object called x that has a value of 4. R has stored this variable in your environment, and knows what x is (the result of 4). So if in your console window you type "x" and press Enter, R returns its value:

```
x

## [1] 4
```

Equivalently, you can use "<-" instead of "=" to assign variables, and you can also use print() to show what your variables are. For example, the two commands above can be replaced by:

```
x<-2+2
print(x)

## [1] 4
```

By default, a variable created this way is a vector (in this case of length 1), and it is numeric because we gave R a number. You can see what class of object you have by using the function class():

```
class(x)

## [1] "numeric"
```

This tells you x is a numeric variable. You could instead create a character string. For example:

```
x1="a"
class(x1)

## [1] "character"
```

Now we have a new object called "x1" that has the character "a" stored.

R can tell if you haven't finished a command and will give you a continuation prompt (+) to let you know you haven't finished. For example, type z=3+ and press Enter:

The prompt "+" appears, detailing that R is missing some information to finish assigning your variable. If you add a 5, for example:

```
z=3+5
```

Now R has understood that it has a variable z which has the value of 8. This usually happens if for example we miss a parenthesis.

Note that - variable names in R must begin with a letter, followed by alphanumeric characters. - R is case sensitive (x and X are not the same variable). - it helps to use descriptive variable names, such as CPUE for catch per unit effort instead of "x" or "y". - Calculations are done with variables as if they were numbers. R uses + for assition, - for substraction, * for multiplication, / for division, and ^ for exponentiation. For example:

```
y=4*10
w=y/x
```

R also has many built-in mathematical functions that operate on variables. For example log() calculates the natural logarithm, and sqrt() is used to calculate the square-root:

```
z1=sqrt(x)
z2=log(y)
```

For a complete list of functions in base R, when you have a chance (on your own time) type library(help = "base") in your console. This will pull up a documentation window and will show which functions are built within R.

You can get help on any R function by entering ?*functionname* or help(*functionname*). For example:

```
?sqrt
help(log)
```

You can also remove variables by using the function rm(). For example, if we wanted to delete the variable "x" from our environment, we would type:

```
rm(x)
```

You will see that "x" no longer appears in your environment, and if you type "x" and press Enter an error message will appear: "Error: object 'x' not found".

Feel free to play around a bit with all you have learnt so far when you have a chance.

Creating a project

To facilitate access to files we are creating during the bootcamp, we can create a project in Rstudio. When you create an R project, we can easily access all files anddata for each specific project without having to change the working directory.

To create an R project, click on “file”, then click on “New project…”, “New directory”, “New Project”. Next, you have to create a name for the project - for example “Rbootcamp”. Done! you have created a new project. With the new project, Rstudio created a folder in your computer (location will depend how this is setup in your Rstudio - you can also change this in “Preferences”). Next time youwant to work on this project, you can go to “Recent projects” or “Open Project” and selct the project you want to work on. This is where you will store all scripts and data we will use in the bootcamp.

Now lets create a folder in the project named “data”, where we can store all data files we will use. To do that, you can click on “New Folder” on the bottom right corner of Rstudio. Please copy and paste all data files to this folder (by coping/dragging files to thesource folder).

Creating a script

So far, we have done everything on the Console window. However, another way to give instructions to R is by creating scripts. A script contains stored instructions for R. You can tell R to read this file, and to carry out the instructions that it contains. It is also the best way to store everything you do in R so you can repeat it in the future. To create a script go to the top left corner of your screen, click on “File”, then click on “New File” and then click on “New Script”. You will see a script file opens up in the top left part of RStudio, pushing your Console window to the bottom left.

You can now type everything you have done so far in the Console window in the script file- that way you ensure you can reproduce everything in the future. In the script file type:

```
getwd()

## [1] "/Users/danielfviana/Rbootcamp_Madagascar/Intro sessions"

x=2+2
x

## [1] 4

x<-2+2
print(x)

## [1] 4

class(x)

## [1] "numeric"
```

```
x1="a"
class(x1)

## [1] "character"

z=3+5
y=4*10
w=y/x
z1=sqrt(x)
z2=log(y)
?sqrt
help(log)
rm(x)
```

You can select it all and click on the button that says “Run”. This will run all the selected sections and show the outputs in your Console window. Save your script to your working directory by clicking on the “Save” icon. Remember to assign a descriptive name to your Script such as “Rintro_file1”. Remember that these scripts need to be in the working directory, or R wont see them.

When writing in a computer language, it is often a good idea to write clarifying comments in plain language, to remind yourself why you did what you did, or to make a note for someone who is looking over your R code. To do this, though, you need to use a special character to say to R “Ignore this, this is information for someone else” You do this with the “#” character. Try it. Go to the command line in your Console, and type a comment without the #. For example, write “This is a comment” and press Enter.

R will give you an error. This is because R thought you were trying to give it instructions, but it didn’t understand what you wanted.

Now type “# This is a comment” and press Enter:

```
#This is a comment
```

R ignores you, and gives you a new prompt.

Now go ahead and comment your script accordingly. For example:

```
# Get working directory
getwd()

## [1] "/Users/danielfviana/Rbootcamp_Madagascar/Intro sessions"

# Create a variable x
x=2+2
# print x
x

## [1] 4

# Create a variable x
x<-2+2
```

```r
# print x
print(x)
```

```
## [1] 4
```

```r
# Output class of variable x (numeric)
class(x)
```

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

```r
# Create a variable x1
x1="a"
# Output class of variable x1 (character)
class(x1)
```

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

```r
# Create a variable z
z=3+5
# Create a variable y
y=4*10
# Create a variable w
w=y/x
# Create a variable z1 which is the square-root of x
z1=sqrt(x)
# Create a variable y which is the natural logarithm of y
z2=log(y)
# Obtain help about the  square-root function
?sqrt
help(log)
# Remove the variable x
rm(x)
```

Vectors

Vectors are 1 dimensional arrays of numbers.

Here are examples of ways in which vectors can be created

```r
#Create a vector  of a sequence from 1 to 5
x=1:5
x=c(1,2,3,4,5)
x=c(1:3, 4, 5)
x=seq(1, 5, by=1)
x
```

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

```r
#Create a vector of zeros with length 5
x=rep(0, 5)
x
```

```
## [1] 0 0 0 0 0
```

```
#Create an empty vector with length 5
x=vector(length=5)
x
```

```
## [1] FALSE FALSE FALSE FALSE FALSE
```

##Vector adressing

Often it is necessary to extract a specific entry or other part of a vector. This is done using brackets, for example:

```
#Create a vector  of a sequence from 1 to 5
x=1:5
#Extract first element of the vector
x[1]
```

```
## [1] 1
```

```
#Extract fifth element of the vector
x[5]
```

```
## [1] 5
```

```
#Extract elements 1 to 3 of the vector
x[1:3]
```

```
## [1] 1 2 3
```

```
#Extract elements 1, 3 and 5 of the vector
x[c(1,3,5)]
```

```
## [1] 1 3 5
```

```
#We can also set specific values within a vector. For example, we can set the
value for the third element of vector x as 0
x[3] = 0
x
```

```
## [1] 1 2 0 4 5
```

##Vector operations

We can perform operations with vectors such as divide, sum, multiply. When operating two vectors of the same size, it will apply the operation to elements in the same position.

Here are some examples

```
#Create a vector  of a sequence from 1 to 5
x=1:5
x
```

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

```r
#Create a vector  of a sequence from 6 to 10
y=6:10
y
```

```
## [1]  6  7  8  9 10
```

```r
#Multiply two vectors
z = x*y
z #This will do the following: 1*6, 2*7....
```

```
## [1]  6 14 24 36 50
```

```r
#Divide two vectors
z = x/y
z #This will do the following: 1/6, 2/7....
```

```
## [1] 0.1666667 0.2857143 0.3750000 0.4444444 0.5000000
```

```r
#Sum two vectors
z= x + y
z #This will do the following: 1+6, 2+7....
```

```
## [1]  7  9 11 13 15
```

```r
#When vectors have different sizes, r will repeat values of the smallest
vector and the resulting vector will have the same size as the largest
vector. For example:
x=1:5
w=6:12
k = x + w
```

```
## Warning in x + w: longer object length is not a multiple of shorter object
## length
```

```r
k
```

```
## [1]  7  9 11 13 15 12 14
```

```r
#Notice that it worked well for 1:5. However, for elements 6 and 7 it summed
with the first two elements of x (k[5:7] =  c(11+1, 12+2)). Even though r
gives you a warning, it does the calculations anyway.
```

#Matrices

Matrices are 2 dimensional arrays of numbers. All columns in a matrix must have the same mode (numeric, character, etc.) and the same length

Here is an example

```r
#Create a matrix with 2 rows and 3 columns
X=matrix(c(1,2,3,4,5,6),2,3)
#This takes the values 1 to 6 and reshapes them into a 2 by 3 matrix
X
```

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

```
#Note that values in the data vector are put into the matrix column-wise, by
default. You can change this by using the optional parameter byrow. For
example
X=matrix(c(1,2,3,4,5,6),2,3, byrow=TRUE)
X
```

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

```
#You can also create a matrix of zeros with 2 rows and 3 columns
X=matrix(0,nrow = 2, ncol= 3)
X
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    0    0    0
```

##Matrix adressing

Often it is necessary to extract a specific entry or other part of a matrix. Like in vectors, this is done using brackets. However, this time we need to specify two elements, where the first element represents the rows and the second represents the columns, for example:

```
#Create a matrix with 2 rows and 3 columns
X=matrix(c(1,2,3,4,5,6),2,3)
X
```

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

```
#First column of matrix
X[,1]
```

```
## [1] 1 2
```

```
#First row of matrix
X[1,]
```

```
## [1] 1 3 5
```

```
#Second element of the first column
X[2,1]
```

```
## [1] 2
```

```
#Change value of the second element of the first column
X[2,1]=200
X
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]  200    4    6
```

```
#We can name the columns of the matrix (note that we need to use quotation
marks)
colnames(X) = c("a", "b", "c")
#Now we can also address the matrix by the name to extract the second element
of column "a"
X[2, "a"]
```

```
##   a
## 200
```

##cbind and rbind

Vectors of same size can be combined to form matrices. We can use the cbind function to combine by columns and the rbind function to combine by rows, for example:

```
#Create a vector of zeros and length 5
x=rep(0, 5)
#Create a vector of ones and length 5
y=rep(1, 5)
#Combine by rows
rbind(x,y)
```

```
##   [,1] [,2] [,3] [,4] [,5]
## x    0    0    0    0    0
## y    1    1    1    1    1
```

```
#Combine by columns
cbind(x,y)
```

```
##      x y
## [1,] 0 1
## [2,] 0 1
## [3,] 0 1
## [4,] 0 1
## [5,] 0 1
```

#Arrays

Arrays are similar to matrices but can have more than two dimensions. This can be useful when we want two matrices to be connected somehow.

```
#Create an array of zeros consisting of two matrices with 10 rows and 3
columns
x = array(0, dim=c(10, 3, 2))
```

```r
#we can name the columns (like in matrices)
colnames(x) = c("a", "b", "c")
x
```

```
## , , 1
##
##       a b c
##  [1,] 0 0 0
##  [2,] 0 0 0
##  [3,] 0 0 0
##  [4,] 0 0 0
##  [5,] 0 0 0
##  [6,] 0 0 0
##  [7,] 0 0 0
##  [8,] 0 0 0
##  [9,] 0 0 0
## [10,] 0 0 0
##
## , , 2
##
##       a b c
##  [1,] 0 0 0
##  [2,] 0 0 0
##  [3,] 0 0 0
##  [4,] 0 0 0
##  [5,] 0 0 0
##  [6,] 0 0 0
##  [7,] 0 0 0
##  [8,] 0 0 0
##  [9,] 0 0 0
## [10,] 0 0 0
```

```r
#Once we name the columns, we can address each column using the column name
#For example, we can set the first element of column "a" of the first matrix
to one
x[1,"a",1] = 1
#Now we can extract the first row of the first matrix
x[1,,1]
```

```
## a b c
## 1 0 0
```

#Data frames

A data frame is more general than a matrix, in that different columns can have different class (numeric, character, factor, etc.). One example where you would need a data frame is when you categorize your data somehow.

For example:

```r
#Create a data frame with one numeric and one factor columns
x = data.frame(1:3,c("a", "b", "c"))
#Name the data frame
names(x) = c("ID","categories")
x
```

```
##   ID categories
## 1  1          a
## 2  2          b
## 3  3          c
```

```r
#We can address data frames using a dollar sign ($)
x$ID
```

```
## [1] 1 2 3
```

#Lists

A list allows to create a variety of unrelated objects under one name. To address elements in the list we need to use either two brackets or address by the name of each element. For example:

```r
#Create a list with factors, a vector and a matrix
x <- list(name=c("Fred", "Joe"), vector=1:5, matrix=matrix(1,2,3))
x
```

```
## $name
## [1] "Fred" "Joe"
##
## $vector
## [1] 1 2 3 4 5
##
## $matrix
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    1    1    1
```

```r
#Extract first element
x[[1]]
```

```
## [1] "Fred" "Joe"
```

```r
#Extract element "name"
x["name"]
```

```
## $name
## [1] "Fred" "Joe"
```

```r
#We can also use a dollar sign ($) to address lists, in this case to extract
the element "name"
x$name
```

```
## [1] "Fred" "Joe"
```