

University of BRISTOL

Getting started with R

A beginner's guide to R and R Studio

Henry W J Reeve

henry.reeve@bristol.ac.uk

Statistical Computing & Empirical Methods (EMATM0061) MSc in Data Science, Teaching block 1, 2021.

What will we cover today?

- We will introduce R and RStudio
- We will learn about the basic objects and operations
- We will learn how to write a simple function in R with control flow statements
- We will see how R facilitates a functional paradigm with call-by-value semantics
- We have a brief look at lazy evaluation
- We will give a few signposts for where to learn more.

Why R?

- R is a free, open source programming language designed for Statistical Computing.
- R provides a fantastic ecosystem for:
 - a) Graphics and data visualization
 - b) Efficient data wrangling
 - c) Statistical inference
 - d) Machine Learning
- Vast online community of contributors and enthusiasts!
- Remark: Python & Julia provide increasingly competitive alternatives.

Why RStudio?

- RStudio is a free, open source integrated development environment for R.
- RStudio provides:
 - a) A console with a command line interface
 - b) A source editor for writing, executing and debugging R scripts
 - c) Syntax highlighting, code completion, smart indentation
 - d) Reproducible analysis via knitr & R Markdown.
 - e) Convenient interface for version control via Git

Installing R and RStudio

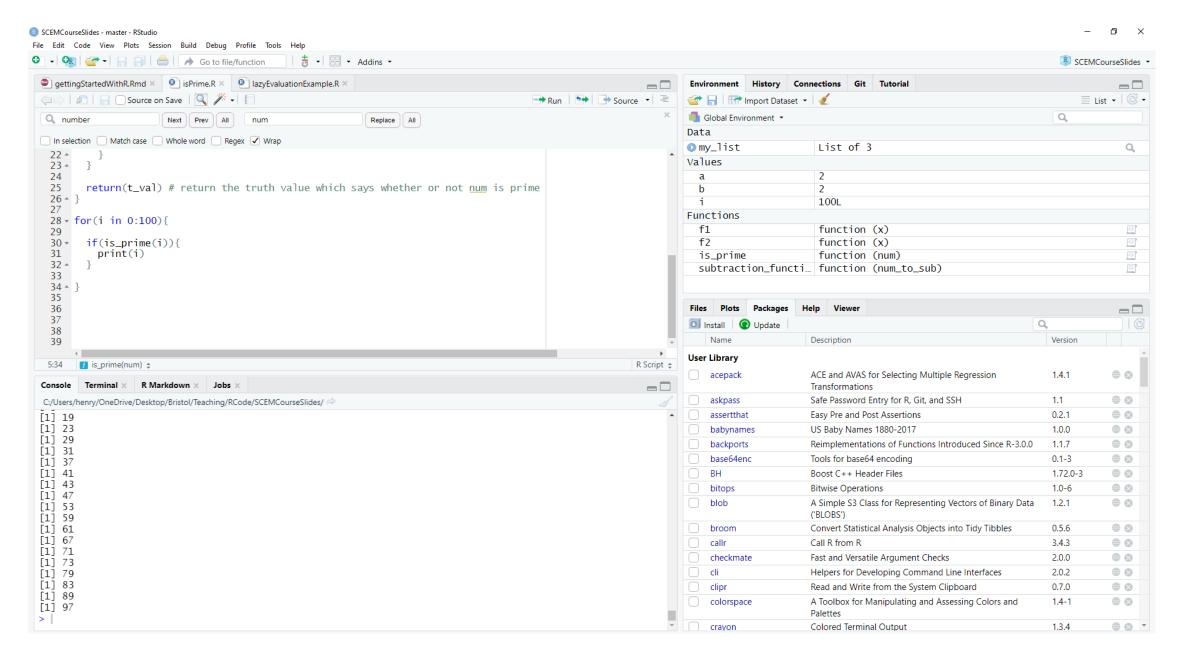
- You can install both R and RStudio in Windows, Linux or Mac OS X.
- First download and install R from the Comprehensive R Archive Network:

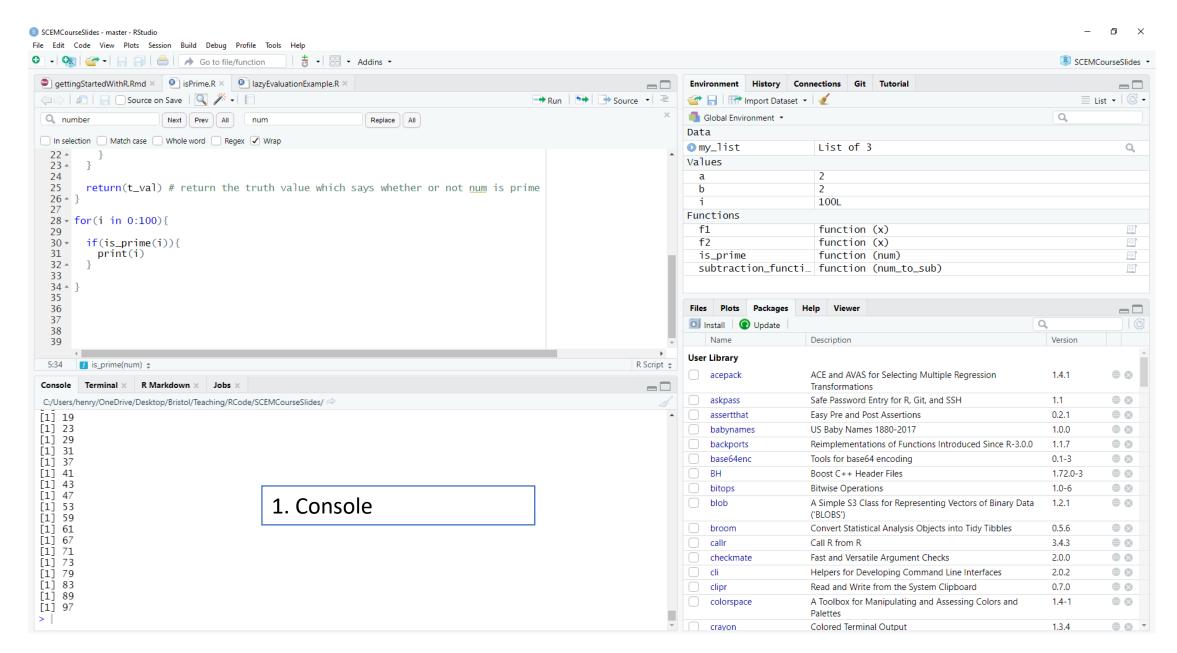
http://www.r-project.org

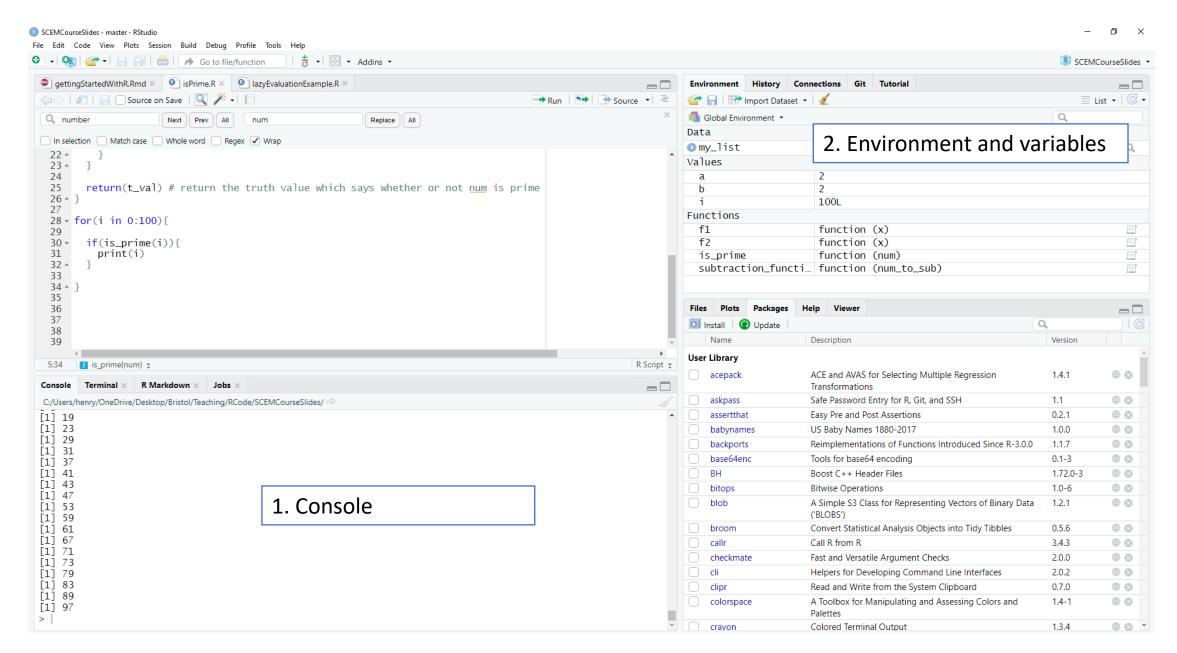
Then download and install Rstudio:

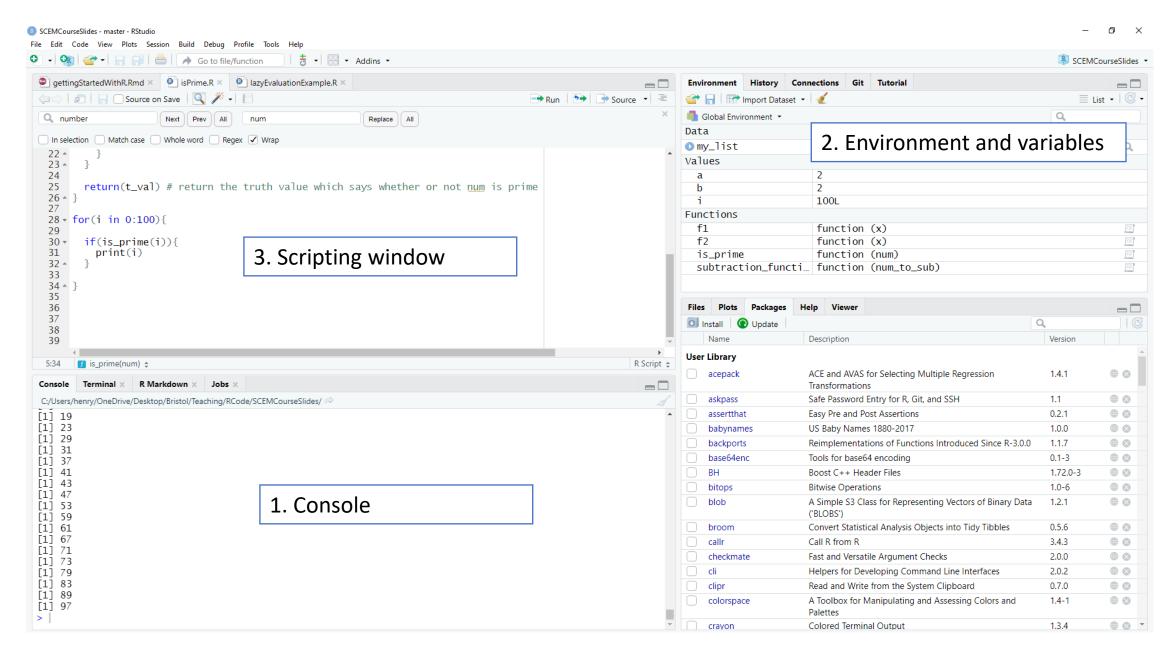
http://www.rstudio.com/download

I recommend that you pause the video here to install R & Rstudio.









Vectors

```
x \leftarrow c(3,7,4,2,1,2,-4,-5) # vector of numbers (use "<-" for assignment )
X
## [1] 3 7 4 2 1 2 -4 -5
y <- seq(5) # A vector of numbers generated as a sequence
У
## [1] 1 2 3 4 5
x[3] # You can access an element of a vector like this
## [1] 4
x[c(2,3)] # Or several elements like this
## [1] 7 4
x[1:4] # Or the first four elements like this
## [1] 3 7 4 2
```

Vectors

```
z <- c("Bristol", "Bath", "London") # You can have a vector of strings
Z
## [1] "Bristol" "Bath" "London"
w <- c(TRUE, FALSE, TRUE, FALSE) # Or a vector of Booleans
W
## [1] TRUE FALSE TRUE FALSE
a <- c(TRUE, 3, "Bristol") # You can't have a vector of mixed type!
а
## [1] "TRUE" "3" "Bristol"
mode(a) # You can test the type like this
## [1] "character"
```

Lists

```
first_list <- list(TRUE, 3, "Bristol") # lists can be of mixed type
first list
## [[1]]
## [1] TRUE
##
## [[2]]
## [1] 3
##
## [[3]]
## [1] "Bristol"
second_list <- list( t_value=TRUE, num_value=3, city = "Bristol") # lists members can be named like a dictionary
second_list$t_value
## [1] TRUE
second_list$num_value
## [1] 3
```

Matrices

```
M <- matrix(seq(10), 2, 5) # You can generate a 2 by 5 matrix
## [,1] [,2] [,3] [,4] [,5]
## [1,] 1 3 5 7 9
## [2,] 2 4 6 8 10
M[2,3] # The third element of the second row can be accessed directly
## [1] 6
M[,4] # Or we can inspect the entire four coloumn
## [1] 7 8
is.vector(M[2,]) # We can check that a selected row or coloumn is a vector
## [1] TRUE
```

Data frames

Data frames are powerful objects for representing and manipulating tabular data.

```
city_name <- c( "Bristol", "Manchester", "Birmingham", "London") # vector of city names
population <- c(0.5,0.5,1,9) # vector of populations

first_data_frame <-data.frame(city_name,population) # we can generate a data frame like this
first_data_frame
```

```
## city_name population
## 1 Bristol 0.5
## 2 Manchester 0.5
## 3 Birmingham 1.0
## 4 London 9.0
```

- Unlike matrices, columns are named & different columns may be of different type
- However, the cells within a column must be of the same type.

Arithmetic operations

```
(((4+2-1)*4)/2)^2 # Arithmetic operations - addition, subtraction, multiplication, division, exponentiation etc..
## [1] 100
a<-matrix(sample(1:10, 6, replace=T),2,3) # a random 2 by 3 matrix
b<-matrix(sample(1:10, 6, replace=T),2,3) # a second random 2 by 3 matrix
a*b # this performs element wise multiplication
    [,1] [,2] [,3]
## [1,] 15 49 15
## [2,] 6 20 9
a%*%t(b) # t(b) computes the transpose of b and %*% performs matrix multiplication
    [,1] [,2]
## [1,] 79 49
## [2,] 65 35
```

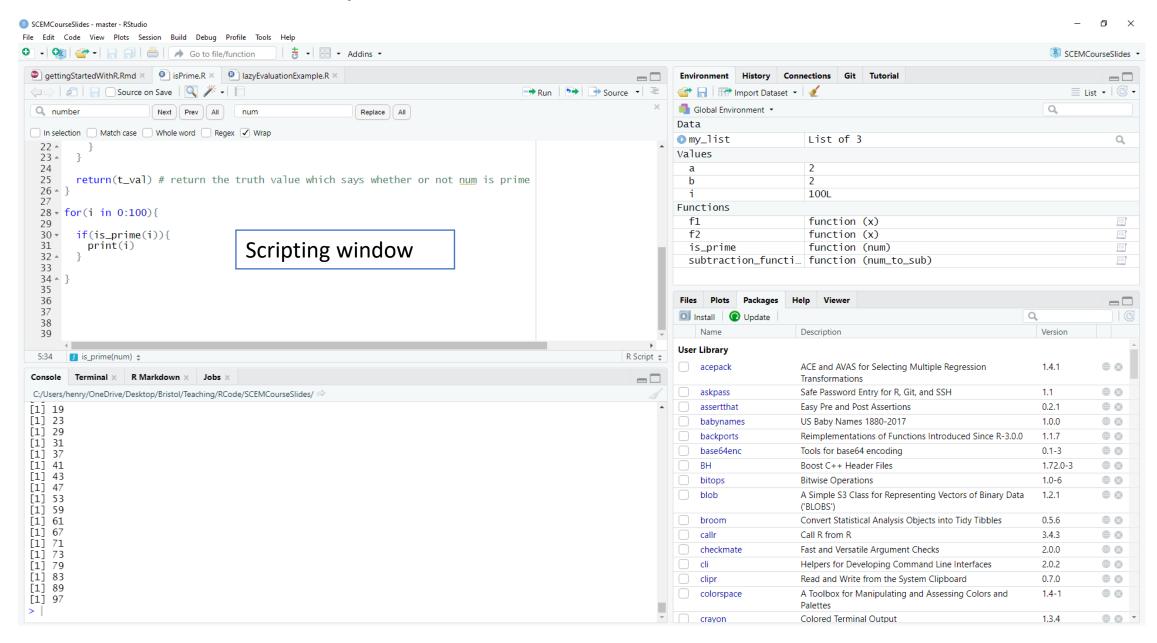
Boolean operations

```
a<-c(TRUE, TRUE, FALSE, FALSE) # a vector of Booleans
b<-c(TRUE, FALSE, TRUE, FALSE) # another vector of Booleans
!a # not a
## [1] FALSE FALSE TRUE TRUE
a&b # a and b
## [1] TRUE FALSE FALSE FALSE
alb # the inclusive or between a and b
## [1] TRUE TRUE TRUE FALSE
xor(a,b) # the exclusive or between a and b
## [1] FALSE TRUE TRUE FALSE
```

Now take a break!



Your first R script



Functions

```
is prime <- function(num) {</pre>
# Function which takes as input a positive integer and outputs Boolean - TRUE if and only if the input is prime.
  stopifnot(is.numeric(num), num%%1==0, num>=0) # Stop if the input is not a positive integer
  t val <- TRUE # Initialise truth value output with TRUE
 if(num<2){
    t val<-FALSE # Output FALSE if input is either 0 or 1
    }else if(num>2) {
      for (i in 2:sqrt(num)) { # Check possible divisors i no greater than sqrt(num)
        if(num%%i==0){
          t val<-FALSE
                  # if i divides num then num is not prime
  return(t val) # return the truth value which says whether or not num is prime
is prime (39) #Now we can use our function to check if 39 is prime.
```

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

Call by value semantics

• In R arguments in functions are passed with call-by-value semantics.

```
a <- seq (5,2) # Create a vector
demo func 1 <- function(x) {
 x[2] \leftarrow -10 # Set the second value of the input to -10
 print(x)
demo func 1(a) # Apply demo func 1 to a
## [1] 5 -10 3 2
a # Note that the value of a is unchanged.
## [1] 5 4 3 2
```

This facilitates a functional programming style with limited side effects.

Lazy evaluation

Lazy evaluation enables efficiency but has some surprising consequences.

```
subtraction_function <- function(num_to_sub){
    output_function <- function(x) {
        return (x-num_to_sub)
    } # a function with input x and output x minus num_to_sub
    return(output_function) #output this function
}
a<-1 # initialise a
f1 <- subtraction_function(a) # construct a function which subtracts a
print(f1(2)) # evaluate function at 2</pre>
## [1] 1
```

```
a<-2 # modify a
print(f1(2)) # doesn't change the function</pre>
```

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

Lazy evaluation

```
subtraction_function <- function(num_to_sub) {
  output_function <- function(x) {
    return (x-num_to_sub)
} # a function with input x and output x minus num_to_sub
  return(output_function) #output this function
}
a<-1 # initialise a
f1 <- subtraction_function(a) # construct a function which subtracts a
print(f1(2)) # evaluate function at 2</pre>
```

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

```
a<-2 # modify a
print(f1(2)) # doesn't change the function
```

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

```
b<-1 # now initialise a new variable b
f2 <- subtraction_function(b) # construct a function which outputs b
b<-2 # change the value of b
print(f2(2)) # evaluating the function reveals that the second choice of b was used.</pre>
```

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

How can we learn more?

• Almost every R function has an associated help function which can be accessed via

```
> ?name_of_function
> help(name_of_function)
```

A fantastic resource to learn more about R is the Swirl package:

```
> install.packages("swirl")
> library(swirl)
```

Another great resource is StackOverflow for R:

https://stackoverflow.com/questions/tagged/r

What have we covered?

- We installed R and RStudio and started our first session.
- We learned about the basic objects vectors, lists, matrices & data frames.
- We considered basic operations arithmetic and Boolean
- We learnt how to write a simple function with control flow statements
- We saw that R facilitates a functional paradigm with call-by-value semantics
- We briefly looked at lazy evaluation and its surprising consequences
- We discussed a few resources for learning more the help function, Swirl, stack etc.

That's plenty for today! – now onto the exercises.



University of BRISTOL

Thanks for listening,

Henry W J Reeve

henry.reeve@bristol.ac.uk

Include EMATM0061 in the subject of your email.

Statistical Computing & Empirical Methods