1 - CRASH COURSE R

Noud van Giersbergen

University of Amsterdam

2021-03-29

General Information

- My name is Noud van Giersbergen. You can find me at home and I have an office REC E4.28. Email: N.P.A.vanGiersbergen@uva.nl.
- Format of this course: 12 lectures of 2h each (2 per week), plus computer labs
- Grading
  - The final grade is composed of:
  - final examination (60%); the grade for the final examination should be at least 5.0
  - average grade of the two computer assignments (each 20%)
- Attendance is required. Your attendance is registered automatically through Zoom!

- The grade for the computer assignments remains valid throughout the academic year in which the student attends the course; the assignments cannot be retaken.
- The final grade of the resit is composed of: resit (60%); average grade of the two computer assignments (40%).

Materials

- These lecture slides (available on Canvas)
- Videos (available on Canvas)
- books:
  - Davies (20016) The Book of R - A First Course in Programming and Statistics, ISBN: 9781593276515
  - Jones, Maillardet and Robinson (2014) Introduction to Scientific Programming and Simulation Using R (2nd ed), ISBN: 9781466569997
- Computer Labs
  - Mostly on Wednesday & Friday, this week Tuesday & Thursday

Motivation Programming and Numerical Analysis

- Algorithms are everywhere
  - Programming is fun!
- Many applications of optimization in economics
  - Minimize costs
  - Maximize revenues, etc.
  - Finding roots of $f'(x) = 0$
- Many applications of matrices in econometrics
  - Solving system of equations
  - OLS
- Application of integration
  - Determine probabilities $P(a \leq X \leq b)$
- Simulations
  - Statistical properties of statistical quantities (like estimators or test statistics)

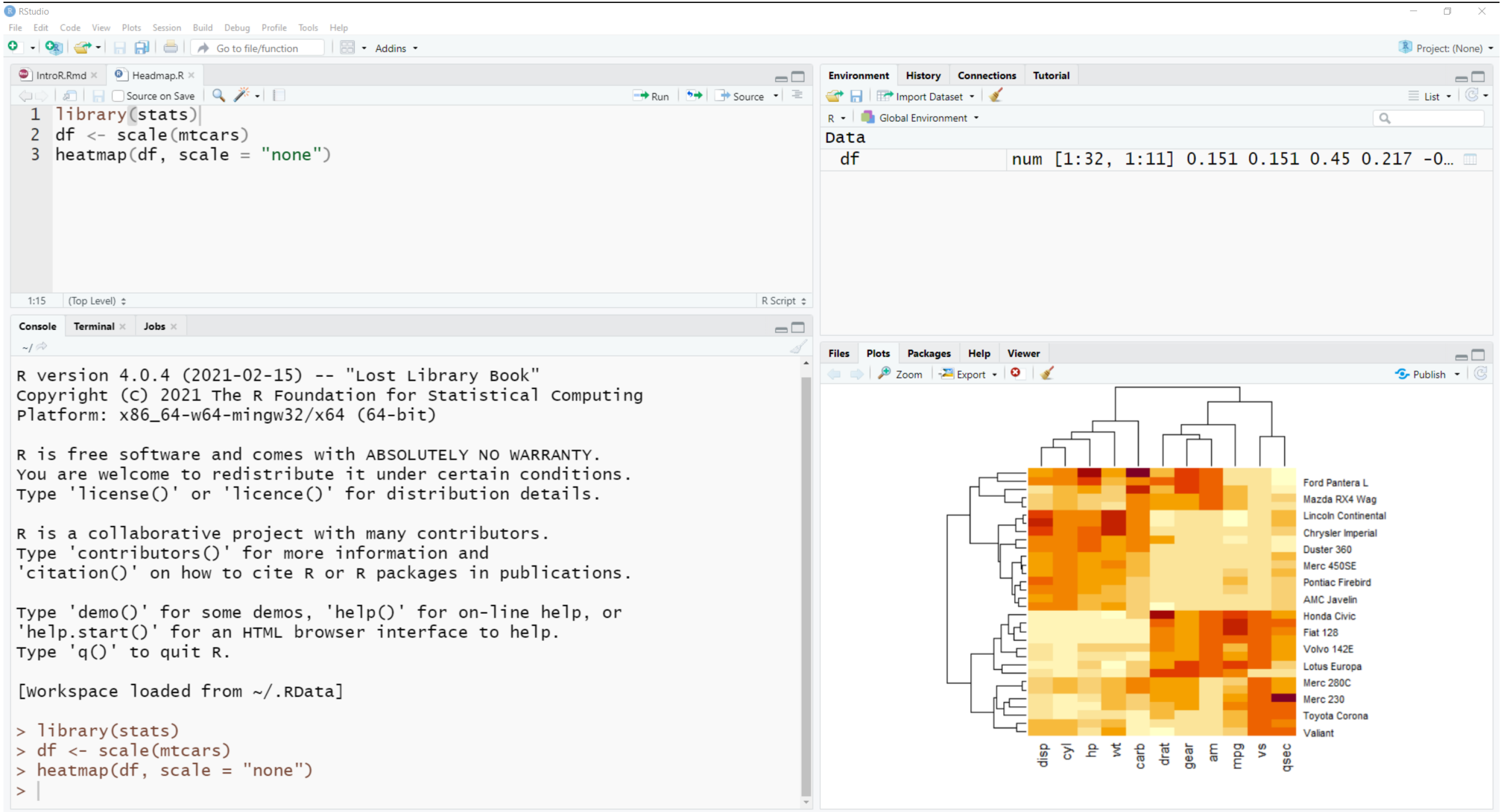
Software

R

- General R website: https://cran.r-project.org/
- Download for Windows (click on base and then "Download R 4.0.4 for Windows "): https://cran.r-project.org/bin/windows/
- Download for macOS: https://cran.r-project.org/bin/macosx/

RStudio

- You have to download the "Free RStudio Desktop" version from here (for Windows/macOS)
- https://rstudio.com/products/rstudio/download/#download



```r
library(stats)
df <- scale(mtcars)
heatmap(df, scale = "none")
```

```
Data
  df                num [1:32, 1:11] 0.151 0.151 0.45 0.217 -0...
```

```
R version 4.0.4 (2021-02-15) -- "Lost Library Book"
Copyright (C) 2021 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

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

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

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

[Workspace loaded from ~/.RData]

> library(stats)
> df <- scale(mtcars)
> heatmap(df, scale = "none")
> 
```

Introduction to R

Why R?

- R programming language is open source and cross-platform
- R is a very popular language in academia
- R has an extensive library of tools for data and database manipulation and wrangling
- R has many tools that can help in data visualization, analysis, and representation
- R is a language designed especially for statistical analysis and data reconfiguration; many new statistical method are first enabled through R libraries
- High-level language with a simple syntax, interactive. Hence ideal for rapid development
- Native R is usually slower than compiled languages like C++. But vector/matrix operations are pretty fast

R Basics

Variable

- A variable is a named memory location. It is assigned using `"="` (technically, `"="` binds the name on the LHS to the result of the expression on the RHS).

```r
a <- 2
a <- a+1
print(a)
```

```
[1] 3
```

- Variable names can be made up from letters, numbers, and the underscore. They may not start with a number. R is case-sensitive: `A` is not the same as `a`.

Built-in Types

Attributes and Methods

- Any R object has a *type*.
- One can use the `typeof` function to show the type of an object:

```r
typeof(a)
```

```
[1] "double"
```

R provides many functions to examine features of vectors and other objects, for example

- `class()` - what kind of object is it (high-level)?
- `typeof()` - what is the object’s data type (low-level)?
- `length()` - how long is it? What about two dimensional objects?

Basic Data Types and Data Structures

R has 5 basic data types:

- character: `"a"`, `"COVID"`
- numeric: `2`, `15.5`
- integer: `2L` (the `L` tells R to store this as an integer)
- logical: `TRUE`, `FALSE`
- complex: `1+4i` (complex numbers with real and imaginary parts)

R has many data structures. These include

- vector & matrix
- list
- data frame
- factors

Numeric Types

- Computers distinguish between integers and floating point numbers
- R integers cannot be arbitrary large
- R double floats are between ±2e308, but are stored with just 53 bits of precision
- Hence, not all real numbers can be represented, and floating point arithmetic is not exact

```r
.Machine$integer.max; as.integer(2147483648)
```

```
[1] 2147483647
```

```
Warning: NAs introduced by coercion to integer range
```

```
[1] NA
```

```r
1 - 1e-16 == 1
```

```
[1] FALSE
```

```r
1 - 1e-17 == 1
```

```
[1] TRUE
```

Arithmetic

- The basic arithmetic operations are `+`, `-`, `*`, `/`, and `^` for exponentiation:

```r
2*(3-1)^2
```

```
[1] 8
```

If any of the operands is a float, then R will convert the others to float, too:

```r
typeof(2L*3L)
```

```
[1] "integer"
```

```r
typeof(2L*3)
```

```
[1] "double"
```

Booleans

- A logical can take one of two values: `TRUE` or `FALSE`.
- They are returned by relational operators: `<`, `<=`, `>`, `>=`, `==` (equality), `!=` (inequality), and can be combined using the logical operators `&` (and), `|` (or), and `!` (not)

```
1<=2<4
```

```
Error: <text>:1:5: unexpected '<'
1: 1<=2<
        ^
```

```
1<=2 & 2<4
```

```
[1] TRUE
```

```
!(1<2)
```

```
[1] FALSE
```

Strings

- Strings hold text. They are constructed using either single or double quotes:

```
s1 <- "R"; s2 <- ' is easy'; line <- paste0(s1,s2)
line
```

```
[1] "R is easy"
```

```
nchar(line)
```

```
[1] 9
```

- Strings cannot be indexed: `line[3]` gives an error

```
substr(line,3,3)
```

```
[1] "i"
```

Vector

- Vectors are indexable collections of homogeneous things

```
x <- c(0, 1, 1, 2, 3, 5, 8, 13, 21)
```

- The function `length` returns the length of a vector

```
length(x)
```

```
[1] 9
```

- We can also pick out several elements ("*slicing*"). This works for all sequence types (lists, matrices)

```
x[2:4]
```

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

- Vectors of integers can be constructed using the `seq` function:

```r
seq(1,11,2)  # from, to [, by]
```

```
[1]  1  3  5  7  9 11
```

```r
1:10    # colon short-cut
```

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

```r
seq(1,3,l=5)   # from, to, length.out
```

```
[1] 1.0 1.5 2.0 2.5 3.0
```

```r
x1 <- seq(5); x2 <- seq(2); x1+x2  # recycle shorter vector!
```

```
Warning in x1 + x2: longer object length is not a multiple of shorter
object length
```

```
[1] 2 4 4 6 6
```

Lists

- Lists are the R objects which contain elements of different types like – numbers, strings, vectors and another list inside it

```
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2), list("green",12.3))
print(list_data)
```

```
[[1]]
[1] "Jan" "Feb" "Mar"

[[2]]
     [,1] [,2] [,3]
[1,]    3    5   -2
[2,]    9    1    8

[[3]]
[[3]][[1]]
[1] "green"

[[3]][[2]]
[1] 12.3
```

Naming List Elements

- The list elements can be given names and they can be accessed using these names

```
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2), list("green",12.3))
names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
list_data$A_Matrix
```

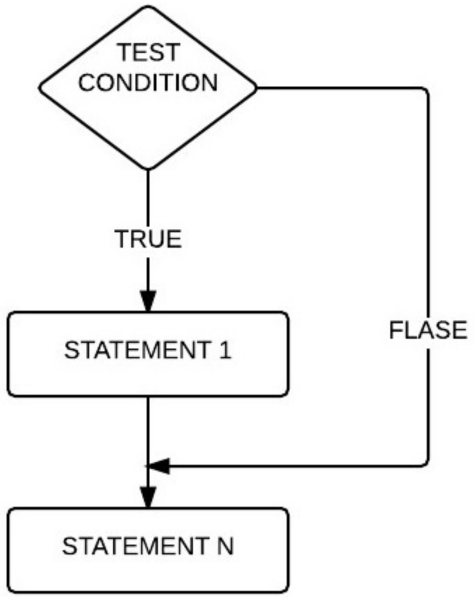
```
     [,1] [,2] [,3]
[1,]    3    5   -2
[2,]    9    1    8
```

```
list_data$'A Inner list'[[2]]
```

```
[1] 12.3
```

Control Flow

- Control flow refers to the order in which commands are executed within a program
- Often we would like to alter the linear way in which commands are executed. Examples:
  1. *Conditional branch*: Code that is only evaluated if some condition is true
  2. *Loop*: Code that is evaluated more than once



Conditional Branch: The `if-else` statement

```r
x <- 3
#x <- as.integer(readline("Enter a number between 0 and 9: "))
if(x<0) {
  print("You have entered a negative number.")
} else if(x>9) {
  print("You have entered a number greater than 9.")
  } else {
    cat("Thank you. You entered",x)
  }
```

```
Thank you. You entered 3
```

- Notes:
  1. Code blocks are introduced by curly brackets and they are advised to be indented.
  2. The `if` block is executed if and only if the first condition is true
  3. The optional second `if` block is executed if and only if the first condition is false and the second one is true. There could be more than one.
  4. The optional `else` block is executed if and only if none of the others was.

While loops

- Similar to `if`, but jumps back to the `while` statement after the `while` block has finished

```
x <- 1 # set this to -1 to run
while( (x<0) | (x>9) ) {
  x <- as.integer(readline("Enter a number between 0 and 9: "))
  if(x<0) print("You have entered a negative number.")
  if(x>9) print("You have entered a number greater than 9.")
}
cat("Thank you. You entered",x)
```

```
Thank you. You entered 1
```

For Loops

- `For` loops are typically used to execute a block of code a pre-specified number of times; the colon operator (`:`) is often used in that case

```
squares = vector()
for(i in 1:10) {
    squares[i] <- i^2
}
print(squares)
```

```
 [1]   1   4   9  16  25  36  49  64  81 100
```

- Question: What does the following compute?

```
n <- 7
f <- 1
for(i in 1:n) f <- f*i
```

Functions

- User-defined functions are declared using the `function` keyword:

```r
mypower <- function(x, y){
  # Compute x^y
  return(x^y)
}
mypower(2, 3)   #positional arguments
```

```
[1] 8
```

```r
mypower(y=3, x=2)   #  named arguments
```

```
[1] 8
```

Several Outputs

```
plusminus <- function(a, b){
  return(list(a+b,a-b))    # return list
}
x <- plusminus(1, 2); cat(x[[1]],x[[2]])
```

```
3 -1
```

```
plusminus <- function(a, b){
  return(list(plus=a+b, minus=a-b))    # return named list
}
x <- plusminus(1, 2); cat(x$plus, x$minus)
```

```
3 -1
```

```
plusminus <- function(a, b){
  return(c(a+b,a-b))    # return vector
}
x <- plusminus(1, 2)
cat(x[1], x[2])
```

Keyword Arguments

- Functions can specify *default arguments*

```
mypower <- function(x, y=2){
  # Compute x^y
  return(x^y)
}
mypower(3)
```

```
[1] 9
```

```
mypower(3, 3)
```

```
[1] 27
```

Variable Scope

- Variables defined in functions are local (not visible in the calling scope)

```r
f <- function(){
  z <- 1
  cat("In f, z equals",z)
}
f()
z
```

```
Error in eval(expr, envir, enclos): object 'z' not found
```

```
In f, z equals 1
```

Calling Convention

- R uses a calling convention known as **Call by value**
  - Call by value method copies the value of an argument into the formal parameter of that function
  - Therefore, changes made to the parameter of the function do not affect the argument

```r
f <- function(y){
  y<-2;   cat("In f, y equals",y,"\n")
}
x <- 1
cat("Before f(x): x=",x,"\n")
f(x)
cat("After f(x): x=",x,"\n")
```

```
Before f(x): x= 1
In f, y equals 2
After f(x): x= 1
```

Anonymous Functions

- Anonymous functions are functions without a name (duh…) and whose function body is a single expression.
- They are often useful for functions that are needed only once (e.g., to return from a function, or to pass to a function

```r
sapply(c(1,2,4,8), function(x) { 3*x})
```

```
[1]  3  6 12 24
```

```r
integrate(function(x){x^2},0,2)
```

```
2.666667 with absolute error < 3e-14
```

Tomorrow PC Labs

- Attendance required
  - Have to submit answers of the exercises handed out at the start
- You are supposed to do the Exercises given in the book on your own
- If you haven't done so, please install R and RStudio
- Only join you own group (see Canvas for the groups)!