Statistical Computing with R

Lecture 2: R scripts and comments; matrices; data frames; functions (part 1); help pages

Mirko Signorelli
: mirkosignorelli.github.io
: statcompr [at] gmail.com

Mathematical Institute
Leiden University

Master in Statistics and Data Science (2023-2024)

Foreword

Foreword

Reminder:

extra class on Friday: 9.00-13.00 (Snellius 1.74)

Recap

Lecture 1:

- statistical computing: what is it?
- `R` and RStudio
- basic operations in `R`
- types of objects
- vectors

Today:

- `R` scripts and comments
- matrices (and linear algebra)
- data frames
- functions (part 1)
- consulting help pages

R scripts and comments

Matrices and linear algebra

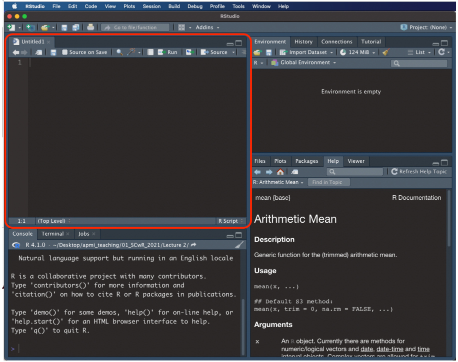
Data frames

Functions: part 1

Creating a script

Code typed in the console gets easily lost. How to “save” your R code?

- Simplest solution: R scripts = text files containing code that you would enter in the Console
- To create an R script: File > New File > R script



Codes and comments

R scripts can contain code and comments

- Code → instructions that are executed by R when you "run" them
- Comments → text / notes that make it easier to understand the code
- In each row, everything after the first hash (#) is a comment
- R identifies comments and does not execute them

```r
# below you find a mixture of code and comments
a = 1:20 # this is a numeric vector
b = letters[1:7] # this is a character
# Important: comments can span multiple lines
# as long as EVERY line starts with a hash (#).
# You can use Ctrl + Shift + C to add the # to multiple lines.
# You can also leave lines empty to separate chunks of code:

# Now, let's create a matrix:
M = matrix(1:4, 2, 2)
```

Writing clear comments: why is it important?

Comments can make code easier to read. They achieve two main goals:

1. help you understand your code when you come back to it
2. help others understand your code

$\Rightarrow$ try to write comments that also other people would understand!

Using comments to add structure to a script

In addition, comments can be used to “add structure” to a long script and make it easier to navigate it. Example:

```
######################
##### Exercise 1 #####
######################
### Step 1: draw random numbers from N(2, 1)
z = rnorm(100, mean = 2)
mean(z); var(z)
### Step 2: apply a linear transformation to z
x = 3*z - 4
### Step 3: check the mean and variance of x
mean(x) # E(X) = 3*E(Z) - 4 = 3*2 - 4 = 2
var(x)  # Var(X) = (3^2)*Var(Z) = 9

######################
##### Exercise 2 #####
######################
hist(cars$speed, 10)
#... and so on!
```

Saving a script

- R scripts can be saved using File > Save / Save as, or using the *floppy disk* icons
- Try to give your scripts a (short) name that will make sense when you (/ someone else) come back to it!
- Dos and don'ts:
  - script1.R, script2.R ✗
  - model fitting.R ✗
  - model_fitting.R ✓
  - 1_data_import.R, 2_data_cleaning.R, 3_model_estimation.R ✓

R scripts and comments

Matrices and linear algebra

Data frames

Functions: part 1

Matrices

- A matrix is a two-dimensional object whose elements (typically numbers!) are organized in rows and columns

$$A = \begin{bmatrix} 4 & 5 & 6 \\ 1 & 7 & 2 \end{bmatrix}$$

```
m1 = matrix(c(4:6, 1, 7, 2), nrow = 2, ncol = 3,
            byrow = T)
m1
```

```
##      [,1] [,2] [,3]
## [1,]    4    5    6
## [2,]    1    7    2
```

- Matrices are widely used in statistics and data science!

Selecting elements in a matrix

```r
m1
```

```
##      [,1] [,2] [,3]
## [1,]    4    5    6
## [2,]    1    7    2
```

```r
m1[2, 2]
```

```
## [1] 7
```

```r
m1[1, ]
```

```
## [1] 4 5 6
```

```r
m1[ , 2:3]
```

```
##      [,1] [,2]
## [1,]    5    6
## [2,]    7    2
```

Basic linear algebra

- In the next slides you will learn how to use `R` to compute some simple linear algebra operations (transposition, product, determinant, inversion)
  行列式
- If you don't know yet how such operations work: you will learn how to perform these computations manually during the *Linear Algebra* course

For now: don't worry if you don't fully understand how some of these computations are performed!

Transposition

- Given a matrix A, its transposed matrix A^T can be computed using `t( )`

```
m1
```

```
##      [,1] [,2] [,3]
## [1,]    4    5    6
## [2,]    1    7    2
```

```
t(m1)
```

```
##      [,1] [,2]
## [1,]    4    1
## [2,]    5    7
## [3,]    6    2
```

Matrix product

- Matrix product of A and B: `A %*% B`

```
m2 = matrix(1:4, nrow = 2, ncol = 2, byrow = T)
m3 = matrix(1:6, nrow = 2, ncol = 3, byrow = T)
m2 %*% m3
```

```
##      [,1] [,2] [,3]
## [1,]    9   12   15
## [2,]   19   26   33
```

⚠ Matrix product possible only if A and B are conformable to matrix multiplication ⚠, i.e. if and only if

columns of left (1st) matrix (A) = # rows of right (2nd) matrix (B)

```
ncol(m2) == nrow(m3)
```

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

Determinant

- The determinant $|A|$ of a square matrix A can be computed using `det( )`

```
A = matrix(1:4, nrow = 2, ncol = 2)
A
```

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

```
det(A)
```

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

只适用于方阵

⚠ The determinant is defined only for square matrices! ⚠

Matrix inverse

- The inverse A^{-1} of a matrix A is a matrix such that $A^{-1}A = I$
- It can be computed using `solve( )`

```
A = matrix(1:4, nrow = 2, ncol = 2)
A.inv = solve(A)
A.inv
```

```
##      [,1] [,2]
## [1,]   -2  1.5
## [2,]    1 -0.5
```

```
A.inv %*% A
```

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

⚠ Inversion possible only for square matrices! ⚠

How to avoid conversion to vector (`drop = FALSE`)

- Problem: when you subset a matrix, ⚠ by default R will convert it to a vector if the resulting matrix has only 1 row / column: ⚠

```
m3
```

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

```
m3[ , 2]
```

```
## [1] 2 5
```

- Add `drop = FALSE` to prevent R from simplifying the subsetted matrix into a vector!

```
m3[ , 2, drop = F]
```

```
##      [,1]
## [1,]    2
## [2,]    5
```

Your turn

Exercises

1. Create the following matrix in R:

$$B = \begin{bmatrix} 3 & 2 & -5 \\ 4 & -2 & 0 \end{bmatrix}$$

2. Obtain $C = \begin{bmatrix} 3 & -5 \\ 4 & 0 \end{bmatrix}$ as a subset of B

3. Can you compute B^{-1}? Can't

Solutions

```r
# Ex 1
B = matrix(c(3, 2, -5, 4, -2, 0), nrow = 2,
           ncol = 3, byrow = T)
B
```

```
##      [,1] [,2] [,3]
## [1,]    3    2   -5
## [2,]    4   -2    0
```

```r
# Ex 2
C = B[ , c(1, 3)]
C
```

```
##      [,1] [,2]
## [1,]    3   -5
## [2,]    4    0
```

```r
# Ex 3
# B is not a square matrix, so we cannot compute
# its inverse. Try solve(B) and see what happens :)
```

R scripts and comments

Matrices and linear algebra

Data frames

Functions: part 1

Data frames

A data frame is a table containing data arranged as follows:

- each row is an observation
- each column is a variable

```
##        name age   country     job
## 1      Mark  25   Germany  waiter
## 2 Margaret  45 Australia    chef
## 3      Fang  32     China plumber
## 4     Pedro  19    Mexico student
```

Creating data frames

- Data frames can be created using the `data.frame( )` function:

```
df1 = data.frame(
  name=c("Mark", "Margaret", "Wang", "Pedro"),
  age=c(25, 45, 32, 19),
  country=c('Germany', 'Australia', 'China', ' Mexico'),
  job=c('waiter', 'chef', 'plumber', 'student'))
```

- You will usually need to work on (much) larger data frames. In those cases, it's better to store the data in an external file, and “import” the data in `R` from such file rather than using `data.frame( )`
- We will see how to do that in Lecture 3 ☺

Number of rows and of columns

- To get the size of the data frame you can use:

```
nrow(df1) # number of rows
```

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

```
ncol(df1) # number of columns
```

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

```
dim(df1)  # number of rows, and of columns
```

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

Number of rows and of columns (cont'd)

- `nrow( )`, `ncol( )` and `dim( )` work also with matrices:

```
A = matrix(1:6, nrow = 2, ncol = 3)
nrow(A) # number of rows
```

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

```
dim(A)  # number of rows, and of columns
```

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

- For vectors, you can use `length( )`:

```
v = 7:20
length(v)
```

```
## [1] 14
```

Selecting rows and columns

- Subsetting with [and] similar to matrices:

```r
df1
```

```
##       name age   country     job
## 1     Mark  25   Germany  waiter
## 2 Margaret  45 Australia    chef
## 3     Wang  32     China plumber
## 4    Pedro  19    Mexico student
```

```r
df1[2, ]
```

```
##       name age   country  job
## 2 Margaret  45 Australia chef
```

```r
df1[3:4 , c(1, 3)]
```

```
##    name country
## 3  Wang   China
## 4 Pedro  Mexico
```

Selecting variables using their name

- Each column in a data frame has a name:

```
names(df1)
```

```
## [1] "name"     "age"      "country" "job"
```

- You can use `df.name$variable.name` to select the variable:

```
df1$country
```

```
## [1] "Germany"   "Australia" "China"      " Mexico"
```

- You can use variable names inside `[ , ]`:

```
df1[ , c('name', 'age')]
```

```
##        name age
## 1      Mark  25
## 2 Margaret  45
## 3      Wang  32
## 4     Pedro  19
```

Selecting a subset of rows

- Which observations meet a certain condition?

```
df1
```

```
##       name age   country     job
## 1     Mark  25   Germany  waiter
## 2 Margaret  45 Australia    chef
## 3     Wang  32     China plumber
## 4    Pedro  19    Mexico student
```

```
df1$age < 30
```

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

- To get the position of the `TRUE` elements, you can use `which( )`:

```
which(df1$age < 30)
```

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

Selecting a subset of rows (cont'd)

- You can use conditions yielding logical vectors to subset rows:

```
df1[df1$age < 30, ]
```

```
##    name age country     job
## 1  Mark  25 Germany  waiter
## 4 Pedro  19  Mexico student
```

- The same can be done with the function `subset( )`:

```
subset(df1, age < 30)
```

```
##    name age country     job
## 1  Mark  25 Germany  waiter
## 4 Pedro  19  Mexico student
```

More about subset( )

```
# how to subset both rows and columns:
subset(df1, age < 30, c('name', 'job'))
```

```
##    name     job
## 1  Mark  waiter
## 4 Pedro student
```

```
# how to subset only rows (= keep all columns):
subset(df1, age < 30)
```

```
##    name age country     job
## 1  Mark  25 Germany  waiter
## 4 Pedro  19  Mexico student
```

```
# how to subset only columns (= keep all rows):
subset(df1, T, c('name', 'job'))
```

```
##       name     job
## 1     Mark  waiter
## 2 Margaret    chef
## 3     Wang plumber
## 4    Pedro student
```

Viewing data frames

- You can click on a data frame in the Environment tab to view it as a table (or use `View(df_name)`)
- However, sometimes data frames can be too large to be easily viewed
- Workaround: you can use `head(data_frame, k)` to view their first *k* rows in the console:

```
head(Orange, 3)
```

```
##   Tree age circumference
## 1    1 118            30
## 2    1 484            58
## 3    1 664            87
```

Variable types

- Each variable (= column) in a data frame can be regarded as a vector
- ... and each vector (= variable) can be of a different type (we saw this in Lecture 1!)
- How to quickly view this? → use `str( )` or `is( )`:

```
str(df1)
```

```
## 'data.frame':    4 obs. of  4 variables:
##  $ name   : chr  "Mark" "Margaret" "Wang" "Pedro"
##  $ age    : num  25 45 32 19
##  $ country: chr  "Germany" "Australia" "China" " Mexico"
##  $ job    : chr  "waiter" "chef" "plumber" "student"
```

```
is(df1$age)
```

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

Your turn

Exercises

1. Create a data frame with info on the courses you are currently following: course name and number of ECs
2. Add to the data frame a variable with the teacher's name
3. Select the courses with $EC > 4$

Solutions

```r
# Ex 1
my.courses = data.frame(
  courses = c('Statistics and Probability',
              'Mathematics for Statisticians',
              'Statistical Computing with R'),
  EC = c(9, 3, 6))
# Ex 2
my.courses$teacher = c('Roula', 'Garnet', 'Mirko')
my.courses
```

```
##                         courses EC teacher
## 1    Statistics and Probability  9   Roula
## 2 Mathematics for Statisticians  3  Garnet
## 3  Statistical Computing with R  6   Mirko
```

Solutions (cont'd)

```r
# Ex 3
# option 1:
my.courses[my.courses$EC > 4, ]
```

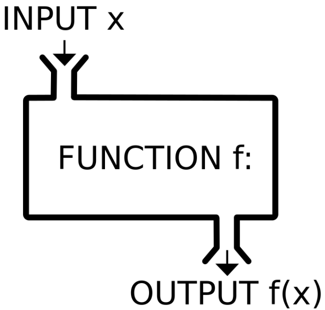
```
##                          courses EC teacher
## 1    Statistics and Probability  9   Roula
## 3 Statistical Computing with R  6   Mirko
```

```r
# option 2:
subset(my.courses, EC > 4)
```

```
##                          courses EC teacher
## 1    Statistics and Probability  9   Roula
## 3 Statistical Computing with R  6   Mirko
```

Functions

- Computations in `R` are mostly performed through functions
- A function f is a list of instructions that given one or more inputs x produces one or more outputs $y = f(x)$



Mathematical functions

- Functions in `R` work in a way similar to mathematical functions:
  1. $y = f(x) = x^2 + 3x - 2$
  2. $z = g(x) = \frac{x-2}{x+1}$

```
y = function(x) x^2 + 3*x - 2
z = function(x) (x-2)/(x+1)
```

- If, for example, the input is $x = 2$:
  1. $y = f(2) = 2^2 + 3 * 2 - 2 = 8$
  2. $z = g(2) = \frac{2-2}{2+1} = 0$

```
y(2); z(2)
```

```
## [1] 8
## [1] 0
```

Built in functions

For didactic purposes, we can distinguish 3 types of functions:

1. "built-in functions", available as soon as you install `R` ("base `R`")
2. functions from `R` packages that are not included in base R
3. user-defined functions

Today we begin with (1). We will cover (2) and (3) in the next lecture ☺

Examples of built-in functions

- We have already used several built-in functions. For example:

```
c(5:9, -7, 334)
seq(5, 20, by = 2)
matrix(1:100, 10, 10)
t(...)
solve(...)
data.frame(...)
nrow(...)
subset(...)
```

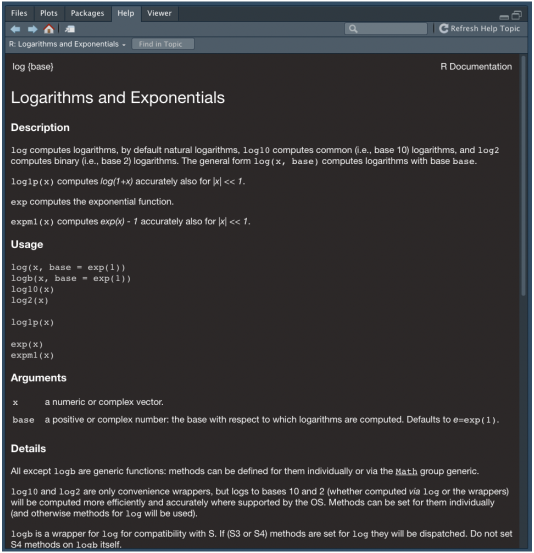
Help pages

- Built-in functions and functions from `R` packages always have a (structured!) help page
- To access the help page, use: `?function-name`

```
?dim
?mean
?log
```

- The help page will open in the Help tab (or in a browser)

Help pages (cont'd)



log {base} R Documentation

Logarithms and Exponentials

Description

`log` computes logarithms, by default natural logarithms, `log10` computes common (i.e., base 10) logarithms, and `log2` computes binary (i.e., base 2) logarithms. The general form `log(x, base)` computes logarithms with base `base`.

`log1p(x)` computes *log(1+x)* accurately also for |x| << 1.

`exp` computes the exponential function.

`expm1(x)` computes *exp(x) - 1* accurately also for |x| << 1.

Usage

```
log(x, base = exp(1))
logb(x, base = exp(1))
log10(x)
log2(x)

log1p(x)

exp(x)
expm1(x)
```

Arguments

`x`	a numeric or complex vector.
`base`	a positive or complex number: the base with respect to which logarithms are computed. Defaults to *e*=`exp(1)`.

Details

All except `logb` are generic functions: methods can be defined for them individually or via the `Math` group generic.

`log10` and `log2` are only convenience wrappers, but logs to bases 10 and 2 (whether computed *via* `log` or the wrappers) will be computed more efficiently and accurately where supported by the OS. Methods can be set for them individually (and otherwise methods for `log` will be used).

`logb` is a wrapper for `log` for compatibility with S. If (S3 or S4) methods are set for `log` they will be dispatched. Do not set S4 methods on `logb` itself.

Help pages (cont'd)

A help page typically comprises at least the following fields:

- **Description**: it provides a short description of what the function does
- **Usage**: it shows the arguments of the function and the default values of an argument
- **Arguments**: an explanation of what each argument is / should be
- **Details**: more detailed information about the function
- **Examples**: code that you can run to get an idea of how to use the function

Functions for numeric inputs

Function	Description
`abs(x)`	Absolute value
`log(x, base = k)`	Logarithm with base k
`exp(x)`	Exponential: e^x
`sin(x)`	Sine
`cos(x)`	Cosine
`tan(x)`	Tangent
`round(x, digits = k)`	Rounds to the k-th digit
`ceiling(x)`	Rounds to the next integer
`floor(x)`	Rounds to the previous integer

```r
floor(4.72)
```

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

```r
round(log(5:10, base = 2), 3)
```

```
## [1] 2.322 2.585 2.807 3.000 3.170 3.322
```

Functions to manipulate character inputs

Function	Description
`paste(..., sep = '')`	Concatenates strings
`toupper(x)`	Uppercase
`tolower(x)`	Lowercase
`substr(x, start, stop)`	Extracts substrings

```r
substr('Good morning', 3, 7)
```

```
## [1] "od mo"
```

```r
paste('hello', c('Maria', 'Josh'), sep = ' ')
```

```
## [1] "hello Maria" "hello Josh"
```

Your turn

Exercises

1. Let $v = (2\ 5\ 8)$. Compute $log_5(v)$ and round the results to the third digit
2. Compute $log_2(\sqrt{e^x + 5})$ for $x = 1, 2, 3$.
3. Let `v = c('random', 'this is', 'SENTENCE', 'just a potato')`. Use the character functions in the previous slide to generate the following string: 'this is just a random sentence'

Solutions

```r
# Ex. 1
round(log(c(2, 5, 8), base = 5), digits = 3)
```

```
## [1] 0.431 1.000 1.292
```

```r
# Ex. 2
x = 1:3
log(sqrt(exp(x) + 5), base = 2)
```

```
## [1] 1.474140 1.815497 2.324392
```

```r
# Ex. 3
v = c('random', 'this is', 'SENTENCE', 'just a potato')
paste(v[2], substr(v[4], 1, 6), v[1], tolower(v[3]))
```

```
## [1] "this is just a random sentence"
```

Functions to compute descriptive statistics

Function	Description
`table(x, y, useNA = 'ifany')`	Frequency tables
`sum(x, na.rm = F)`	Sum all elements in x
`mean(x, na.rm = F)`	Arithmetic mean
`median(x, na.rm = F)`	Median
`var(x, na.rm = F)`	Variance
`sd(x, na.rm = F)`	Standard deviation
`min(x, na.rm = F)`	Minimum
`max(x, na.rm = F)`	Maximum
`range(x, na.rm = F)`	Range

Most functions have the argument `na.rm`:

- `na.rm = F` (⚠ default!): if `NA`s are present in `x`, the output is `NA`
- `na.rm = T`: if present, `NA`s are ignored and the computation is performed excluding them

Descriptive statistics functions (cont'd)

```r
marks = c(7.5, 9, NA, 8, 6.5, 8, 7.5)
table(marks)
```

```
## marks
## 6.5 7.5   8   9
##   1   2   2   1
```

```r
mean(marks) # NB: default na.rm = FALSE!!!
```

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

```r
mean(marks, na.rm = T) # set na.rm = TRUE to skip NAs :)
```

```
## [1] 7.75
```

```r
sd(marks, na.rm = T)
```

```
## [1] 0.8215838
```

```r
range(marks, na.rm = T)
```

```
## [1] 6.5 9.0
```

Your turn

Exercises

The `iris` data frame, pre-loaded in `R`, contains measurements of 150 iris plants published by R. Fisher in 1936. Type `?iris` in the console for more details about this data frame.

1. How many variables does the data frame contain? What are their names?
2. Compute the frequency distribution of plants by species
3. Compute the mean, median and range of petal length

Solutions

```
# Ex. 1
ncol(iris); names(iris)
```

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

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length"
## [4] "Petal.Width"  "Species"
```

```
# Ex. 2
table(iris$Species)
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

```
# to view also the NA values, specify: useNA = 'ifany'
table(iris$Species, useNA = 'ifany') # no NAs here :)
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

Solutions (cont'd)

```r
# Ex. 3
mean(iris$Petal.Length)
```

```
## [1] 3.758
```

```r
median(iris$Petal.Length)
```

```
## [1] 4.35
```

```r
range(iris$Petal.Length)
```

```
## [1] 1.0 6.9
```