Data import and export with R,
R
as a programming language

—

B83128 – Introduction to R scripting language (shortened version)

Lubomír Štěpánek[1, 2]

[1] Department of Biomedical Statistics
Institute of Biophysics and Informatics
First Faculty of Medicine
Charles University, Prague

[2] Department of Biomedical Informatics
Faculty of Biomedical Engineering
Czech Technical University in Prague

Content

1. Working directory
2. Data import and export
3. Conditionals
4. Loops
5. Warnings and errors
6. `*apply()` family
7. References

Working directory

- to get where my current working directory is

```
getwd()
```

- what my current working directory includes

```
dir()
```

- setting to a new working directory by a code

```
setwd("C:/.../my_working_directory")
```

- setting to a new working directory by a pop-up window

```
setwd(choose.dir())
```

Import and export of a plain text

- using functions `readLines()` and `writeLines()`
- we can load into R whatever with a plain text basis

```
    my_html <- readLines(
        con = paste(
            "https://ubi.lf1.cuni.cz/en",
            "introduction-to-r",
            sep = "/"
        ),
        encoding = "UTF-8"
    )
```

Import and export of a plain text

- saving and loading of a plain text format

```
writeLines(                 # I am saving to a file
    text = paste(
        "One R to rule them all",
        "one R to find them",
        "one R to bring them all",
        "and in the darkness bind them",
        sep = "\n"   # a new-line separator
    ),
    con = "my_text.txt"
)

my_loaded_text <- readLines(
    con = "my_text.txt",
    encoding = "UTF-8"
)                           # I am loading from a file
```

Import and export of a tabular format

- work-horse functions are `read.table()` and `write.table()`
- both the functions have many wrappers (`read.csv()` and `write.csv()`, `read.delim()` and `write.delim()`, respectively, etc.)

```
write.table(            # I am saving a data.frame
    x = mtcars,
    sep = ";",
    row.names = FALSE,
    file = "mtcars.csv" # or "mtcars.txt"
)

my_mtcars <- read.table(
    file = "mtcars.csv",
    sep = ";",
    header = TRUE
)                       # I am loading as a data.frame
```

Import and export of a tabular format

```
write.table(x = iris,
            file = "iris.txt")
```

- the function read.table() have some useful arguments

```
my_iris <- read.table(
    file = "iris.txt",
    sep = " ",
    header = TRUE,
    stringsAsFactors = FALSE,
    check.names = FALSE,
             # stops to check whether variables'
             # names are valid
    colClasses = "character"
             # coerces all variables to character
             # data type
)
```

Import and export of MS Excel® files (.xlsx)

- a package openxlsx could be used
  - it is necessary to install a tool Rtools following the links (with respect to a platform)

    https://cran.r-project.org/bin/windows/Rtools/

    https://cran.r-project.org/bin/macosx/tools/
- saving a data.frame to excel file (.xlsx) using the functions

```
    createWorkbook()
    addWorksheet(...)
    writeData(...)
    saveWorkbook(...)
```

- loading an excel file as a data.frame using

```
my_data <- read.xlsx(xlsxFile = "my_table.xlsx",
                     sheet = 1,   # or a sheet name
                     colNames = TRUE)
```

Export of a console output to a file

- using `sink()` – `sink()` commands or `capture.output()` command

```
(males <- rnorm(100, mean = 175, sd = 10))
(females <- rnorm(100, mean = 160, sd = 10))

t.test(males, females)

# console output to a file
capture.output(t.test(males, females),
               file = "t_test.txt")

# or similarly
sink("this_is_also_a_t_test.txt")
t.test(males, females)
sink()
```

Export a feasible R object to a TEX-ového code

- using an xtable table

```
library("xtable")

my_linear_model <- lm(mpg ~ hp + cyl,
                      mtcars)

xtable(my_linear_model)
xtable(my_linear_model, digits = 4)

# or a more complex TeX code
print(xtable(my_linear_model,
             digits = 4),
      floating = FALSE,
      tabular.environment = "tabular",
      hline.after = NULL,
      include.rownames = TRUE,
      include.colnames = TRUE)
```

Import of „exotic“ files to R session

- a package `foreign` is our friend
- many unusual formats are supported by the `foreign` package
    - Epi Info
    - Minitab
    - S
    - SAS, SPSS, STAT, Systat, Weka

```r
library("foreign")

# import of SPSS data
my_data <- read.spss(
    file = "a_file_from_SPSS.sav",
    to.data.frame = TRUE
)
```

A conditional *if*

- also called *if-statement*
- a decision construct based on truth-values of a predicate (logical condition), i. e. usually a relationship of a variable to some other variables or constants
- obecná syntaxe

```
      if(logical condition){
        a procedure when the logical condition is TRUE
      }else{
        a procedure when the logical condition is FALSE
      }
```

A conditional *if*

- for example

```
x = 1

if(x == 1){
  print("x is equal to 1")
}

# or
x = 2

if(x == 1){
  print(""x is equal to 1"")
}else{
  print(""x is not equal to 1"")
}
```

for() loop

- a flow-control construct for iteration of the same procedure many times
- it can be used when we know the number of the iterations of the procedure in advance
- a general syntax

```
      for(definition of an index space){

        a procedure for each atomic item
        of the index space

      }
```

for() loop

- for example

```
for(i in 1:5){

  print(i)

}

# or
for(my_letter in letters){

  print(
    paste(my_letter, "is fine", sep = " ")
  )

}
```

while() loop

- a flow-control construct for iteration of the same procedure many times
- it can be used when we don't know the number of the iterations of the procedure in advance
- a general syntax

```
index <- 1
while(logical condition){

  a procedure for each index,
  it can modify the logical condition
  and is run until the logical condition
  is TRUE

  index <- index + 1

}
```

while() loop

- for example

```
i <- 1
while(i <= 5){
  print(i)
  i <- i + 1
}

# or
my_letters <- letters
while(length(my_letters) > 0){

  print(
    paste(my_letters[1], "is fine", sep = " ")
  )
  my_letters <- my_letters[-1]

}
```

repeat-until loop

- a flow-control construct for iteration of the same procedure many times
- it can be used when we don’t know the number of the iterations of the procedure in advance
- very similar to the `while()` loop
- a general syntax

```
index <- 1
while(TRUE){
  if(stopping condition){break}

  a procedure for each index,
  it can generates the stopping condition

  index <- index + 1
}
```

repeat-until loop

- for example

```
i <- 1
while(TRUE){
  if(i == 6){break}
  print(i)
  i <- i + 1
}

# or
my_letters <- letters
while(TRUE){
  if(length(my_letters) == 0){break}
  print(
    paste(my_letters[1], "is fine", sep = " ")
  )
  my_letters <- my_letters[-1]
}
```

Warnings

- a warning is a text message returned by a function or procedure
- it is not a malignant error and does not stop the code execution

```r
log(-5)  # NaN; In log(-5) : NaNs produced
```

- we can define our own warnings

```r
getMyLog <- function(x){
  # it returns a natural logarithm of "x"
  if(x <= 0){
    cat(
      "x is non-positive, NaN will be output\n"
    )
  }
  return(suppressWarnings(log(x)))
}

getMyLog(-5)  # NaN;x is non-positive, NaN will be output
```

Errors

- an error is a text message returned by a function or procedure when an executing problem is occurring not allowing them to be executed henceforth

```r
  "1" + "1" # Error: non-numeric argument to binary
            # operator
```

- we can define our own errors

```r
sumUpTheSquares <- function(a, b){
  # it returns a sum of squares of "a" and "b"
  if(!is.numeric(a)){stop("a must be a number!")}
  if(!is.numeric(b)){stop("b must be a number!")}
  return(a ^ 2 + b ^ 2)
}

sumUpTheSquares(1, 2)   # 5
sumUpTheSquares(1, "2") # Error: b must be a number!
```

*apply() functions family

- a group of functions well optimized by early calling of C++ equivalents to R functions
- based on that, they are executed very quickly
- an `apply()` and `lapply()` function is most useful for us

```r
# returns means for each column of "mtcars" data.frame
x <- apply(mtcars, 2, mean)

# the same as above but not so elegant
x <- NULL
for(i in 1:dim(mtcars)[2]){
  x <- c(x, mean(mtcars[, i]))
}
names(x) <- colnames(mtcars)
```

A function apply()

- it returns a vector of a function FUN's results calculated for a matrix or a data frame X, particularly for its rows (MARGIN = 1), or columns (MARGIN = 2)
- a general syntax is apply(X, MARGIN, FUN, ...)

```
my_start <- Sys.time()
x <- apply(mtcars, 2, mean)
my_stop <- Sys.time(); my_stop - my_start # 0.019s

my_start <- Sys.time()
x <- NULL
for(i in 1:dim(mtcars)[2]){
  x <- c(x, mean(mtcars[, i]))
  names(x)[length(x)] <- colnames(mtcars)[i]
}
my_stop <- Sys.time(); my_stop - my_start # 0.039s
```

A function lapply()

- it returns a vector of a function FUN's results calculated for a vector or a list X
- a general syntax is lapply(X, FUN, ...)
- it could be used for a reformulation of a for() loop into a vectorized form
- it also helps with list addressing

```r
set.seed(1)
my_long_list <- lapply(
    sample(c(80:120), 100, TRUE),
        function(x) sample(
            c(50:150), x, replace = TRUE
        )   # a list of vectors of a random length
    )       # consisting of random numbers

lapply(my_long_list, "[[", 14)
    # the 14-th value of each list slot is returned
```

A replacement of for() loop by a lapply() function

- both calls are equivalent according to their outputs, but `lapply()` is significantly faster

```
# for loop
x <- NULL
for(i in 1:N){
  x <- c(x, FUN)
}

# lapply
x <- unlist(
  lapply(
    1:N,
    FUN
  )
)
```

A replacement of for() loop by a lapply() function

```r
# for loop
my_start <- Sys.time()

for_x <- NULL
for(i in 1:100000){for_x <- c(for_x, i ^ 5)}

my_stop <- Sys.time(); my_stop - my_start # 18.45s

# lapply
my_start <- Sys.time()

lapply_x <- unlist(lapply(
  1:100000, function(i) i ^ 5
))

my_stop <- Sys.time(); my_stop - my_start # 0.10s
```

References

Alain F. Zuur, Elena N. Ieno und Erik Meesters. *A Beginner's Guide to R*. Springer New York, 2009. DOI: 10.1007/978-0-387-93837-0. URL: https://doi.org/10.1007/978-0-387-93837-0.

Hadley Wickham. *Advanced R*. Boca Raton, FL: CRC Press, 2015. ISBN: 978-1466586963.

Thank you for your attention!

lubomir.stepanek@lf1.cuni.cz
lubomir.stepanek@fbmi.cvut.cz

▸ GitHub

https://github.com/LStepanek/B83128_Introduction_to_R_scripting_language