R TUTORIAL

涂玉臻 (Yu-Zhen Tu)

Research Assistant at Brain and Mind Laboratory, NTU

Community Mentor of "R programming" at Coursera

2016/10/12

OUTLINE

- Overview of R
- Input and evaluation
- Atomic classes of objects
- Data types
- Help documents
- Working directory
- Reading data
- Having a peek of data
- Extracting subsets
- Vectorized operations
- Control structures
- Writing functions
- References
- Panel discussion

OVERVIEW OF R

R as a language and a free software

2017/3/1

R PROGRAMMING LANGUAGE

- R is a dialect of S.
- S is a language developed by John Chambers at Bell Labs in 1976.
- Version 3 of S was rewritten in C language in 1988.
- R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand in 1991.
- The R Core Group was formed in 1997.
- R version 1.0.0 was released in 2000.
- R version 3.3.1 was released on June 21, 2016.

R IS A FREE SOFTWARE

- One is granted:
  1. The freedom to run the program for any purpose.
  2. The freedom to study how the program works.
  3. The freedom to share copies of the program.
  4. The freedom to improve or modify the program, and release the improvements to the public.
- "We call this free software because the user is free." (The Free Software Foundation, https://www.fsf.org)

INPUT AND EVALUATION

2017/3/1

INPUT

- Use "<-" as the assignment operator.
  - x <- 5
  - name <- "Yu-Zhen"
- "#" means that what comes afterwards in this line is a comment. Comments won't be executed.
  - name <- "Yu-Zhen" # keying my name in
  - # Nothing happens.
- Incomplete expressions

EVALUATION AND PRINTING

- x <- 5 # inputting the value of x
- x # auto-printing
- print(x) # explicit printing

ATOMIC CLASSES

Character, numeric, integer, complex, and logical

2017/3/1

FIVE BASIC CLASSES OF OBJECTS

- character: “Yu-Zhen”, “a”, “Hello world!”
- numeric: pi, 123, Inf
- integer: 10L, 123L
- complex: 7 + 3i
- logical: TRUE or FALSE
- Use the function class() to find out which class the object is.

MISSING VALUES

- NA (Not Available) and NaN (Not a Number)
- An NaN is an NA, but the converse is false.
- is.na() and is.nan() are used to test if objects are NA and NaN, respectively.

DATA TYPES

Vector, list, matrix, factor, and data frame

2017/3/1

VECTORS

- Use the function c() or vector() to create a vector.
- An integer sequence from m to n can be created by m:n.
- Vectors accept only the same class of elements.
- Coercion occurs when different classes are input.
  (logical < integer < numeric < complex < character)
- Explicit coercion can be performed with as.* functions.

LISTS

- Use list() to create a list.
- Lists are similar with vectors but accept different classes of elements.

MATRICES

- Matrices are vectors with two dimensions (row and column).
- Use matrix() to create a matrix.
- Matrices are constructed column-wise by default.
- Bind columns or rows by cbind() or rbind() respectively.

FACTORS

- factor() creates factors from vectors.
- The "level" argument can be specified to assign base level.
- Factors are very useful for analyzing categorical variables.
- Pay extra attention when coercing factors into numbers.

DATA FRAMES

- data.frame() can create one data frame.
- Data frames look like matrices but allow different classes of elements across columns.
- cbind() and rbind() are also effective for data frames.

HELP DOCUMENTS

HOW TO LOOK UP FUNCTIONS

- ? or help() gets the help document of the function.
  - Some arguments have default values.
  - Arguments can be passed by matching name or matching position.
- ?? or help.search() finds functions with the key word.
- Getting more help
  - Google
  - Statistics on telnet://ptt.cc
  - Stack Overflow http://stackoverflow.com/

WORKING DIRECTORY

RETRIEVE AND SET WORKING DIRECTORY

- Files will be saved to and retrieved from the working directory if no path is specified.
- getwd() retrieves current working directory path.
- setwd() sets another specific path as working directory.
- Let's create a folder and set it as the working directory for this tutorial! (Please avoid Chinese characters in your path.)

READING DATA

read.table()/write.table(), load()/save.image(), source()/dump()

READING TABLES

- read.table() or read.csv() inputs a table into R as a data frame.
- The important arguments include:
  - file: the file path or file.choose()
  - header: whether the first row should be regarded as names
  - sep: specified separator for the table (e.g. "," for .csv files)
- write.table() or write.csv() outputs data in a spreadsheet.

READING A SAVED WORKSPACE

- load() inputs previously saved workspace into R.
- Workspace contains objects we've assigned.
- save.image() saves the workspace as a .RData file.

READING WRITTEN CODES

- source() inputs a written .R file into R.
- dump() can be used to save .R files.
- We will resume the practice of using source() after we write our first function.

HAVING A PEEK OF DATA

SOME USEFUL FUNCTIONS

- summary() provides descriptive statistics. It also "produces result summaries of the results of various model fitting functions".
- head() and tail() "return the first or last parts of a vector, matrix, table, data frame or function".
- str() "compactly display the internal structure of an R object".

EXTRACTING SUBSETS

Prepare for lots of practice here!

SUBSETTING OPERATORS

- [] extracts elements by name, index(es), or logical values.
  - It can be used to select more than one element.
  - It usually returns an object as the same class as the original.
- [[]] extracts elements from lists and data frames.
  - It can extract only one element.
  - Returned objects are not necessarily the same class as the original.
- $ extracts elements by name.
- By default, subsetting a single row or a single column from a matrix will return a vector, unless the argument drop = FALSE.

VECTORIZED OPERATIONS

VECTORIZED OPERATIONS IN R

- x <- 1:3
- y <- 4:6
- x + y = ?
- a <- 1:4
- b <- 1:2
- a – b = ?
- m <- matrix(6, 2, 2)
- n <- matrix(1:4, 2, 2)
- m * n = ?
- m %*% n = ?

CONTROL STRUCTURES

Common ones: if, for, while
These three loops can be nested.

RELATIONAL AND LOGICAL OPERATORS

- == # equal to
- >= # greater than or equal to
- <= # less than or equal to
- > # greater than
- < # less than
- != # not equal to
- ! # not
- & # and
- | # or

IF

- if (<condition>) {
  # do something
  }
  else {
  # do something else
  }
- if (<condition 1>) {
  # do something
  }
  else if (<condition 2>) {
  # do something different
  }
  else {
  # do something else
  }

FOR

- A for loop takes an iteration variable, and assigns it successive values from a sequence or a vector.
-
```
x <- seq(0, 100, length.out = 5)
for (i in 1:5) {
        if (x[i] >= 60) {
                print("Pass")
        }
        else {
                print("Fail")
        }
}
```

WHILE

- while loops begin with testing a condition. If it is true, then execute the body. After the execution, the condition will be tested again.
- while loops can possibly cause infinite loops if not written properly.
- ```
  i <- 1
  while (i < 4) {
          i <- i + 1
          print(i)
  }
  ```
  ```

WRITING FUNCTIONS

Let's brainstorm and achieve everything we want!

2017/3/1

THE FIRST FUNCTION

- ```
  add <- function(x, y) {
          x + y
  }
  ```
- After inputting the .R file into R, add() is now able to add two numbers and print out the total.
- Try create a function, which can subset and print out elements greater than a specified number from a numeric vector.
- Let's brainstorm!
  ```

REFERENCES

REFERENCES

- R Programming on Coursera
  https://www.coursera.org/learn/r-programming
- The R Project
  https://www.r-project.org/
- The Free Software Foundation
  https://www.fsf.org/

PANEL DISCUSSION