BST02: Using R for Statistics in Medical Research

Part C: Functions and Programming

Nicole Erler   Eleni-Rosalina Andrinopoulou

Department of Biostatistics, Erasmus Medical Center

✉ n.erler@erasmusmc.nl   ✉ e.andrinopoulou@erasmusmc.nl

24 - 28 February 2020

Erasmus MC
University Medical Center Rotterdam

Recap Part B

Objects

- ► `vector`
- ► `matrix`
- ► `data.frame`
- ► `list`

Data Structures

- ► `numeric`
- ► `character`
- ► `integer`
- ► `logical`
- ► `factor`

Recap Part B

Objects

- ► `vector`
- ► `matrix`
- ► `data.frame`
- ► `list`

Operators

- ► +, -, *, /
- ► <-, =
- ► <, >, ==

Data Structures

- ► `numeric`
- ► `character`
- ► `integer`
- ► `logical`
- ► `factor`

Special Values

- ► `NA`
- ► `NaN`
- ► `Inf, -Inf`

Recap Part B

Objects
- ► `vector`
- ► `matrix`
- ► `data.frame`
- ► `list`

Operators
- ► +, -, *, /
- ► <-, =
- ► <, >, ==

Data Structures
- ► `numeric`
- ► `character`
- ► `integer`
- ► `logical`
- ► `factor`

Special Values
- ► `NA`
- ► `NaN`
- ► `Inf, -Inf`

Data Transformations
- ► rounding (`format()`)
- ► convert to factor (`factor()`)

Recap Part B

Objects
- ► `vector`
- ► `matrix`
- ► `data.frame`
- ► `list`

Operators
- ► +, -, *, /
- ► <-, =
- ► <, >, ==

Data Structures
- ► `numeric`
- ► `character`
- ► `integer`
- ► `logical`
- ► `factor`

Special Values
- ► `NA`
- ► `NaN`
- ► `Inf, -Inf`

Data Transformations
- ► rounding (`format()`)
- ► convert to factor (`factor()`)

Data Exploration
- ► `mean()`, `median()`, `sd()`, `IQR()`, …

Data Visualizations
- ► plotting packages
- ► plot types (`plot()`, `barplot()`, …)

Recap Part B

Objects
- ▶ `vector`
- ▶ `matrix`
- ▶ `data.frame`
- ▶ `list`

Operators
- ▶ `+, -, *, /`
- ▶ `<-, =`
- ▶ `<, >, ==`

Data Structures
- ▶ `numeric`
- ▶ `character`
- ▶ `integer`
- ▶ `logical`
- ▶ `factor`

Special Values
- ▶ `NA`
- ▶ `NaN`
- ▶ `Inf, -Inf`

Data Transformations
- ▶ rounding (`format()`)
- ▶ convert to factor (`factor()`)

Data Exploration
- ▶ `mean()`, `median()`, `sd()`, `IQR()`, …

Data Visualizations
- ▶ plotting packages
- ▶ plot types (`plot()`, `barplot()`, …)

Subsetting
- ▶ `[[...]], [...], $, …`

In this Section

- ▶ What are functions?
- ▶ Useful functions for data exploration
- ▶ Useful functions for data manipulations
- ▶ Writing functions
- ▶ Control-flow constructs
- ▶ The `apply` family
- ▶ Lots of practicing

Functions

Sometimes we want to perform the same action / manipulation on several objects.

- ► Option 1: copy & paste
    - ► a lot of work
    - ► susceptible to mistakes

Functions

Sometimes we want to perform the same action / manipulation on several objects.

- ► Option 1: copy & paste
  - ► a lot of work
  - ► susceptible to mistakes

- ► Option 2: **functions**

Functions

Sometimes we want to perform the same action / manipulation on several objects.

- ► Option 1: copy & paste
    - ► a lot of work
    - ► susceptible to mistakes

- ► Option 2: **functions**

What are functions?

- ► a group of (organized) R commands
- ► a (small) programm with flexible (= not pre-specified) input

Almost all commands in R are functions!

Functions

Some examples:

- ▶ mean()
- ▶ sum()
- ▶ plot()
- ▶ ...

```
class(mean)
## [1] "function"
class(sum)
## [1] "function"
class(plot)
## [1] "function"
```

Functions

Some examples:

- ▶ mean()
- ▶ sum()
- ▶ plot()
- ▶ ...

```
class(mean)
## [1] "function"
class(sum)
## [1] "function"
class(plot)
## [1] "function"
```

Even class() is a function:

```
class(class)
```

```
## [1] "function"
```

Useful Functions for Data Exploration

Link to Demo: Functions_DataExploration.R

Link to Practical:

Useful Functions for Data Exploration

Dimension
- ► `dim()`
- ► `nrow(), ncol()`
- ► `length()`

Data Structure
- ► `str()`
- ► `names(),`
- ► `head(), tail()`
- ► `is.data.frame(),`
  `is.list(),`
  `is.matrix()`
  `is.numeric(),`
  `is.ordered(),...`

Descriptives for Continuous Variables
- ► `summary()`
- ► `min(), max(),`
  `range()`
- ► `mean(), median(),`
  `quantile(), IQR()`
- ► `sd(), var()`
- ► `ave()`

Tables
- ► `table(),`
  `prop.table()`
- ► `addmargins(),`
  `ftable()`

for `matrix` & `data.frame`
- ► `summary()`
- ► `var(), cor(), cov2cor()`
- ► `colSums(), colMeans(),`
  `rowSums(), rowMeans()`

Duplicates & Comparison
- ► `duplicated()`
- ► `unique()`

Useful functions for Data Manipulation

Link to Demo: Functions_DataManipulation.R

Link to practical

Useful functions for Data Manipulation

Transformations
- ▶ `log()`, `log2()`, `log10()`
- ▶ `exp()`, `sqrt()`, `plogis()`

Splitting & Combining
- ▶ `split()`, `cut()`
- ▶ `cbind()`, `rbind()`
- ▶ `merge()`
- ▶ `subset()`
- ▶ `c()`
- ▶ `paste()`

repetition & sequence
- ▶ `rep()`, `seq()`
- ▶ `expand.grid()`

Transformation for objects
- ▶ `t()`
- ▶ `unlist()`, `unname()`
- ▶ `as.numeric()`, `as.matrix()`, `as.data.frame()`

Sorting
- ▶ `sort()`, `order()`, `rev()`, `rank()`

matrices
- ▶ `%*%`
- ▶ `diag()`, `det()`, `solve()`
- ▶ `upper.tri()`, `lower.tri()`

Writing Functions

To write your own function:

```r
myfun <- function(arguments) {
  syntax
}
```

For example:

```r
square <- function(x) {
  x^2
}
```

```r
square(3)
```

```
## [1] 9
```

Writing Functions

Functions do not always need an argument:

```r
random <- function() {
  rnorm(1)
}
```

```r
random()
## [1] 2.722814
random()
## [1] -0.6862098
random()
## [1] -0.5885114
```

Writing Functions

Functions can use multiple arguments:

```r
subtract <- function(x, y) {
  x - y
}
```

```r
subtract(x = 5.2, y = 3.3)
```

```
## [1] 1.9
```

Writing Functions

Multiple arguments are interpretet in the pre-defined order, unless they are named:

```
subtract(5.2, 1.2)
```

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

```
subtract(y = 5.2, x = 1.2)
```

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

Writing Functions

We can also define default values for arguments.

```r
multiply <- function(x, y = 2) {
  x * y
}
```

The default value is used when the user does not specify a value for that argument:

```r
multiply(x = 3, y = 3)
```

```
## [1] 9
```

```r
multiply(x = 3)
```

```
## [1] 6
```

Writing Functions

Link to Demo

Link to Practical

Control-flow constructs

- `if(cond) expr`
- `if(cond) cons.expr else (alt.expr)`
- `ifelse()`
- `for`
- `while`
- `repeat`
- `break`
- `next`

What is the apply Family

► Manipulate slices of data from matrices, arrays, lists and dataframes in a repetitive way avoiding explicit use of loop constructs
  ► An aggregating function, like for example the mean, or the sum
  ► Other transforming or subsetting functions
  ► Other vectorized functions, which return more complex structures like lists, vectors, matrices and arrays

What is the apply Family (cont'd)

apply(), lapply() , sapply(), tapply(), mapply()

But how and when should we use these?

How To Use apply() in R

► Operates on Matrices and Data Frames

```
mat <- matrix(1:6, 3, 3)
mat

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

```
apply(mat, 2, sum)

[1]  6 15  6
```

```
mat <- matrix(1:6, 3, 3)
mat

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

```
apply(mat, 1, sum)

[1]  6  9 12
```

How To Use apply() in R (cont'd)

▶ Operates on Matrices and Data Frames

```
mat <- matrix(1:6, 3, 3)
mat

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

```
apply(mat, 2, mean)
```

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

```
mat <- matrix(1:6, 3, 3)
mat

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

```
apply(mat, 1, mean)
```

```
[1] 2 3 4
```

How To Use apply() in R (cont'd)

► You can also apply your functions

```
mat <- matrix(1:6, 3, 3)
mat

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

apply(mat, 2, function(x)
         sum(x)/(length(x)-1))

[1] 3.0 7.5 3.0
```

```
mat <- matrix(1:6, 3, 3)
mat

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

apply(mat, 1, function(x)
         sum(x)/(length(x)-1))

[1] 3.0 4.5 6.0
```

How To Use lapply() in R

- ▶ Apply a given function to every element of a list and obtain a list as result
- ▶ The difference with apply():
    - ▶ It can be used for other objects like data frames, lists or vectors
    - ▶ The output returned is a list

How To Use lapply() in R (cont'd)

```r
myList <- list(x <- c(1:6),
               y = c("m", "f"),
               z = c(30, 4, 23))
myList
```

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

$y
[1] "m" "f"

$z
[1] 30  4 23
```

```r
myList <- list(x <- c(1:6),
               y = c("m", "f"),
               z = c(30, 4, 23))
lapply(myList, length)
```

```
[[1]]
[1] 6

$y
[1] 2

$z
[1] 3
```

How To Use lapply() in R (cont'd)

```r
myList <- list(x <- c(1:6),
               y = c("m", "f"),
               z = c(30, 4, 23))
myList
```

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

$y
[1] "m" "f"

$z
[1] 30  4 23
```

```r
myList <- list(x <- c(1:6),
               y = c("m", "f"),
               z = c(30, 4, 23))
lapply(myList, median)
```

```
[[1]]
[1] 3.5

$y
[1] NA

$z
[1] 23
```

How To Use sapply() in R

▶ sapply() is similar to lapply(), but it tries to simplify the output

```
myList <- list(x <- c(1:6),
               y = c("m", "f"),
               z = c(30, 4, 23))
myList
```

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

$y
[1] "m" "f"

$z
[1] 30  4 23
```

```
myList <- list(x <- c(1:6),
               y = c("m", "f"),
               z = c(30, 4, 23))
sapply(myList, length)
```

```
  y z
6 2 3
```

```
sapply(myList, median)
```

```
        y    z
 3.5   NA 23.0
```

How To Use tapply() in R

▶ Apply a function to subsets of a vector and the subsets are defined by some other vector, usually a factor

```
tapply(pbc$bili, pbc$sex, mean)
```

```
       m        f
2.865909 3.262567
```

```
tapply(pbc$age, pbc$sex, median)
```

```
       m        f
54.00137 50.19302
```

How To Use tapply() in R (cont'd)

▶ You can also apply your functions

```
tapply(pbc$bili, pbc$sex, function(x) sum(x)/(length(x)-1))
```

```
       m        f
2.932558 3.271314
```

How To Use mapply() in R

- ▶ Multivariate apply
- ▶ Its purpose is to be able to vectorize arguments to a function that is not usually accepting vectors as arguments
- ▶ mapply() applies a function to multiple list or multiple vector arguments

```
mapply(length, pbc)
```

id	time	status	trt	age	sex	ascites	hepato
418	418	418	418	418	418	418	418
spiders	edema	bili	chol	albumin	copper	alk.phos	ast
418	418	418	418	418	418	418	418
trig	platelet	protime	stage				
418	418	418	418				

How To Use mapply() in R (cont'd)

```r
myList <- list(x <- c(1:6),
               y = c("m", "f"),
               z = c(30, 4, 23))
mapply(length, myList, SIMPLIFY = FALSE)
```

```
[[1]]
[1] 6

$y
[1] 2

$z
[1] 3
```

Useful Summary: Apply Family

Vectors
- ► `tapply()`
- ► `sapply()`
- ► `lapply()`
- ► `mapply()`

Matrices
- ► `apply()`
- ► `tapply()`
- ► `lapply()`
- ► `sapply()`
- ► `mapply()`

Data frames
- ► `apply()`
- ► `tapply()`
- ► `lapply()`
- ► `sapply()`
- ► `mapply()`

Lists
- ► `lapply()`
- ► `sapply()`
- ► `mapply()`