Introductory course on the R software

Benoit Liquet[1]

[1] University of Pau & Pays de L'Adour
Laboratory of Mathematics and Its Application
France

University Sebalas Maret
Department of Mathematics
Faculty of Mathematics and Natural Science
February 28, 2017

`http://benoit-liquet.github.io`

Goals of today lecture

Describing the instructions for

- elementary data manipulation ;
- extraction tool (direct and by logical mask) ;
- dealing with character strings.

Vector arithmetic

R can operate on vectors and matrices :

```
> x <- c(1,2,4,6,3)
> y <- c(4,7,8,1,1)
> x + y
[1]  5  9 12  7  4
```

Returns the vector of sums $(x_1 + y_1, \ldots, x_n + y_n)$.

This is one of the **main strengths** of R. It is called **vectorization**.

```
> M <- matrix(1:9,nrow=3)
> exp(M)
          [,1]      [,2]     [,3]
[1,]  2.718282  54.59815 1096.633
[2,]  7.389056 148.41316 2980.958
[3,] 20.085537 403.42879 8103.084
```

Vectorization is much quicker

```
> x <- rnorm(1000000)
> system.time(z<-0;for(i in 1:1000000) z <- z + x[i])
   user  system elapsed
  1.143   0.009   1.154
> z
[1] 367.689
> system.time(z <- sum(x))
   user  system elapsed
  0.002   0.000   0.003
> z
[1] 367.689
```

Recycling

```
# Vector of length 15:
> x <- c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)
# Vector of length 10:
> y <- c(1,2,3,4,5,6,7,8,9,10)
# Vector of length 15:
> x + y
 [1]  2  4  6  8 10 12 14 16 18 20 12 14 16 18 20

> matrix(1:4,ncol=3,nrow=3)
     [,1] [,2] [,3]
[1,]    1    4    3
[2,]    2    1    4
[3,]    3    2    1
```

Basic functions

- `length()` : returns the length of a vector.

```
> length(c(1,3,6,2,7,4,8,1,0))
[1] 9
```

- `sort()` : sorts the elements of a vector, in increasing or decreasing order.

```
> sort(c(1,3,6,2,7,4,8,1,0))
[1] 0 1 1 2 3 4 6 7 8
> sort(c(1,3,6,2,7,4,8,1,0),decreasing=TRUE)
[1] 8 7 6 4 3 2 1 1 0
```

- `rev()` : rearranges the elements of a vector in reverse order.

```
> rev(c(1,3,6,2,7,4,8,1,0))
[1] 0 1 8 4 7 2 6 3 1
```

Basic functions

• `order()`, `rank()` : the first function returns the vector of (increasing or decreasing) ranking indices of the elements. The second function returns the vector of ranks of the elements. In case of a tie, the ordering is always from left to right.

```
> vec <- c(1,3,6,2,7,4,8,1,0) ; names(vec) <- 1:9
> vec
1 2 3 4 5 6 7 8 9
1 3 6 2 7 4 8 1 0
> sort(vec)
9 1 8 4 2 6 3 5 7
0 1 1 2 3 4 6 7 8
> order(vec)
[1] 9 1 8 4 2 6 3 5 7
> rank(vec)
  1   2   3   4   5   6   7   8   9
2.5 5.0 7.0 4.0 8.0 6.0 9.0 2.5 1.0
```

Operations on matrices and data.frames

Perform the "**Do it yourself**" on page 89.

`http://biostatisticien.eu/springeR/Rbook-chap5.pdf`

Note : You may have to first install the executable file `http://biostatisticien.eu/springeR/Rtools29.exe` and then to install and load from within **R** the package `gdata`.

Merging columns of matrices or data.frames

The generic function is `cbind()`.

```
> cbind(1:4,5:8)
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```

Merging columns of matrices or data.frames

Try to merge these two tables using `cbind()` :

X1=

Id	GENDER	Weight
1	M	75
2	F	68
3	F	48
4	M	72

$\cup$

X2=

Id	GENDER	Height
1	M	182
2	F	165
3	F	160
4	M	178

What is the problem ?

Now, try using the `merge()` function.

Merging columns of matrices or data.frames

```
> X
  GENDER Height Weight Income
1      F    165     50     80
2      M    182     65     90
3      M    178     67     60
4      F    160     55     50
```

```
> Y
  GENDER Height Weight Salary
4      F    165     55     70
5      M    182     65     90
6      M    178     67     40
7      F    160     85     40
```

```
> merge(X,Y,by=c("GENDER","Weight"))
  GENDER Weight Height.x Income Height.y Salary
1      F     55      160     50      165     70
2      M     65      182     90      182     90
3      M     67      178     60      178     40
```

```
> merge(X,Y,by=c("GENDER","Weight"),all=TRUE)
  GENDER Weight Height.x Income Height.y Salary
1      F     50      165     80       NA     NA
2      F     55      160     50      165     70
3      F     85       NA     NA      160     40
4      M     65      182     90      182     90
5      M     67      178     60      178     40
```

Merging lines of matrices or data.frames

The generic function is `rbind()`.

```
> rbind(1:4,5:8)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
```

Merging lines of matrices or data.frames

```
> require(gtools)
> df1 <- data.frame(A=1:5, B=LETTERS[1:5])    # The square
                                              # brackets [] to
                                              # extract
> df2 <- data.frame(A=6:10, E=letters[1:5])   # elements will
                                              # be described
                                              # later.

> smartbind(df1, df2)
     A    B    E
1.1  1    A <NA>
1.2  2    B <NA>
1.3  3    C <NA>
1.4  4    D <NA>
1.5  5    E <NA>
2.1  6 <NA>    a
2.2  7 <NA>    b
2.3  8 <NA>    c
2.4  9 <NA>    d
2.5 10 <NA>    e
```

The function apply()

```
> ( X <- matrix(c(1:4, 1, 6:8), nr = 2) )
     [,1] [,2] [,3] [,4]
[1,]    1    3    1    7
[2,]    2    4    6    8
> apply(X, MARGIN=1, FUN=mean)
[1] 3 5
> apply(X, MARGIN=2, FUN=sum)
[1]  3  7  7 15
```

When the operation is summing or calculating the means of rows or columns, other possible functions are : `rowSums()`, `colSums()`, `rowMeans()`, `colMeans()`.

Perform the "**Do it yourself**" on page 93.

The function transform()

```
> ( X <- data.frame(Weight=c(80,75,60,52),Height=c(180,
+             170,165,150),Cholesterol=c(44,12,23,34),
+           Gender=c("Male","Male","Female","Female")) )
  Weight Height Cholesterol Gender
1     80    180          44   Male
2     75    170          12   Male
3     60    165          23 Female
4     52    150          34 Female
> ( X <- transform(X,Height=Height/100,
>            BMI=Weight/(Height/100)^2) )
  Weight Height Cholesterol Gender      BMI
1     80   1.80          44   Male 24.69136
2     75   1.70          12   Male 25.95156
3     60   1.65          23 Female 22.03857
4     52   1.50          34 Female 23.11111
```

Operations on lists : the function `lapply()`

```
> x <- list(a = 1:10, beta = exp(-3:3),
+           logic = c(TRUE,FALSE,FALSE,TRUE))
> lapply(x,mean) # Mean of each element of the list.
$a
[1] 5.5
$beta
[1] 4.535125
$logic
[1] 0.5
```

Logical and relational operations

Have a look at the Table 5.1 on page 98.

Operations on sets

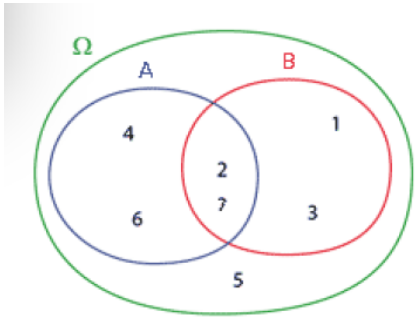

TABLE: Operations on sets.

Operation	R Instruction	Output
Membership : $a \in A$	`is.element(vec,A)`	`T F T`
Inclusion (subset) : $A \subset B$	`all(A %in% B)`	`F`
Superset : $A \supset B$	`all(B %in% A)`	`F`
Intersection : $A \cap B$	`intersect(A,B)`	`2 7`
Union : $A \cup B$	`union(A,B)`	`4 6 2 7 1 3`
Complement : $A \backslash B$	`setdiff(A,B)`	`4 6`
Symmetric difference : $(A \cup B) \backslash (A \cap B)$	`setdiff(union(A,B),intersect(A,B))`	`4 6 1 3`

Create `A`, `B` and `vec <- c(2,3,7)` and play with these functions !

Extracting from vectors

```
> vec <- c(2,4,6,8,3)
> vec[2]
[1] 4
> "["(vec,2)          # Note: "[" is indeed a function.
[1] 4
> vec[-2]             # All elements except the second.
[1] 2 6 8 3
> vec[2:5]
[1] 4 6 8 3
> vec[-c(1,5)]
[1] 4 6 8
> vec[c(T,F,F,T,T)] # Extraction by logical mask.
[1] 2 8 3
> vec>4
[1] FALSE FALSE  TRUE  TRUE FALSE
> vec[vec>4]          # Extraction by logical mask.
[1] 6 8
```

Extracting from vectors

It is important to note here the syntactical simplicity of an instruction such as `x[y>0]`, which extracts from `x` all elements of index i such that $y_i > 0$.

```
> x <- 1:5
> y <- c(-1,2,-3,4,-2)
> x[y>0]
[1] 2 4
```

You need to learn to use as often as possible such constructions, which are called **logical masks**. There are two advantages : the code is easy to read, and execution is very fast.

Note : the functions `which()`, `which.min()` and `which.max()` are often very useful.

Replacement into vectors

```
> z
[1] 0 0 0 2 0
> z[c(1,5)] <- 1
> z
[1] 1 0 0 2 1
> z[which.max(z)] <- 0
> z
[1] 1 0 0 0 1
> z[z==0] <- 8  # The z_i such that
                # z_i is worth 0 are replaced with
                # 8.
> z
[1] 1 8 8 8 1
```

Insertion into vectors

```
> vecA <- c(1, 3, 6, 2, 7, 4, 8, 1, 0)
> vecA
[1] 1 3 6 2 7 4 8 1 0
> (vecB <- c(vecA, 4, 1))
 [1] 1 3 6 2 7 4 8 1 0 4 1
> (vecC <- c(vecA[1:4], 8, 5, vecA[5:9]))
 [1] 1 3 6 2 8 5 7 4 8 1 0

> a <- c()
> a <- c(a,2)
> a <- c(a,7)
> a
[1] 2 7
```

Perform the "**Do it yourself**" on page 102.

Extraction from matrices

Two methods are possible to extract elements from a matrix `X`. Each method has its own syntax.

1. *Extracting by indices :* `X[indr,indc]`, where `indr` is the vector of indices of rows and `indc` is the vector of indices of columns to extract. Omitting `indr` (respectively `indc`) means that all rows are selected (respectively all columns). Note that `indr` and `indc` can be preceded by a minus sign (`-`) to indicate elements not to extract.
2. *Extracting by logical mask :* `X[mask]`, where `mask` is a matrix of logical values `TRUE/FALSE` of the same size as `X`, indicating which elements to extract.

Extraction from matrices

```
> ( Mat <- matrix(1:12,nrow=4,ncol=3,byrow=TRUE) )
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
> Mat[2,3]          # Extracting the element at the
                    # intersection of row 2 and column 3.
[1] 6
> Mat[,1]           # All rows, and only column 1.
[1]  1  4  7 10
> Mat[c(1,4),]      # All columns, and rows 1 and 4.
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]   10   11   12
> Mat[3,-c(1,3)]    # Row 3 and column 2.
[1] 8
```

Extraction from matrices (logical mask)

```
> MatLogical <- matrix(c(TRUE,FALSE),nrow=4,ncol=3)
> MatLogical       # Is of the same size as Mat.
      [,1]  [,2]  [,3]
[1,]  TRUE  TRUE  TRUE
[2,] FALSE FALSE FALSE
[3,]  TRUE  TRUE  TRUE
[4,] FALSE FALSE FALSE
> Mat[MatLogical] # Make sure that you understand this
                  # instruction.
[1] 1 7 2 8 3 9
```

Extraction from matrices

```
> m <- matrix(c(1,2,3,1,2,3,2,1,3),3,3)
> m
     [,1] [,2] [,3]
[1,]    1    1    2
[2,]    2    2    1
[3,]    3    3    3
> which(m == 1)          # m is seen as the concatenation
                         # of its columns.
[1] 1 4 8
> which(m == 1,arr.ind=TRUE) #Outputs indices as couples
     row col
[1,]   1   1
[2,]   1   2
[3,]   2   3
```

Insertion into matrices

```
> m
     [,1] [,2] [,3]
[1,]    1    1    2
[2,]    2    2    1
[3,]    3    3    3
> m[m!=2] <- 0
> m
     [,1] [,2] [,3]
[1,]    0    0    2
[2,]    2    2    0
[3,]    0    0    0
> Mat <- Mat[-4,] ; Mat
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> m[Mat>7] <- Mat[Mat>7]
> m
     [,1] [,2] [,3]
[1,]    0    0    2
[2,]    2    2    0
[3,]    0    8    9
```

Perform the "**Do it yourself**" on page 105.

Extracting from Lists

```
> ( L <- list(12,c(34,67,8),Mat,1:15,list(10,11))  )
[[1]]
[1] 12
[[2]]
[1] 34 67  8
[[3]]
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[[4]]
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
[[5]]
[[5]][[1]]
[1] 10
[[5]][[2]]
[1] 11
> L[2]
[[1]]
[1] 34 67  8
> class(L[2])
[1] "list"
> L[c(3,4)]
[[1]]
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[[2]]
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

Extracting from Lists

```
> L[[2]]
[1] 34 67  8
> "[["(L,2)
[1] 34 67  8
> class(L[[2]])
[1] "numeric"
> L[[5]][[2]]
[1] 11
> L <- list(cars=c("FORD","PEUGEOT"),climate=
+                          c("Tropical","Temperate"))
> L[["cars"]]
[1] "FORD"    "PEUGEOT"
> L$cars
[1] "FORD"    "PEUGEOT"
> L$climate
[1] "Tropical"  "Temperate"
```

Inserting into Lists

```
> L$climate[2] <- "Continental"
> L
$cars
[1] "FORD"     "PEUGEOT"
$climate
[1] "Tropical"    "Continental"
```

Manipulating character strings

```
> ( string <- c("one","two","three") )
[1] "one"   "two"   "three"
> as.character(1:3)
[1] "1" "2" "3"
> string1 <- c("a","ab","B","bba","one","!@","brute")
> nchar(string1)   # Counts the number of symbols in each string.
[1] 1 2 1 3 3 2 5
> string1[nchar(string1)>2]
[1] "bba"   "one"   "brute"
> string2 <- c("e","D")
> paste(string1,string2) # Concatenates the strings.
[1] "a e"     "ab D"    "B e"     "bba D"   "one e"   "!@ D"
[7] "brute e"
> paste(string1,string2,sep="-")  # A separator can be included
                                  # between the strings.
[1] "a-e"     "ab-D"    "B-e"     "bba-D"   "one-e"   "!@-D"
[7] "brute-e"
> paste(string1,string2,collapse="",sep="")  # collapse is used to
                                             # concatenate the elmts
                                             # into a single string.

[1] "aeabDBebbaDonee!@Dbrutee"
```

Manipulating character strings

```
> substring("abcdef",first=1:3,last=2:4)
[1] "ab" "bc" "cd"
> strsplit(c("05 Jan","06 Feb"), split=" ")
[[1]]
[1] "05"  "Jan"
[[2]]
[1] "06"  "Feb"
> grep("i",c("Pierre","Benoit","Rems"))
[1] 1 2
> gsub("i","L",c("Pierre","Benoit","Rems"))
[1] "PLerre" "BenoLt" "Rems"
> sub("r","L",c("Pierre","Benoit","Rems"))
[1] "PieLre" "Benoit" "Rems"
```

Perform the "**Do it yourself**" on page 111.

Your turn to work !

Do the Exercises on pages 134-136.

Do the Worksheet (A, B, E, C) on pages 136-140.

`http://biostatisticien.eu/springeR/Rbook-chap5.pdf`