CeDoSIA SS2020 - Exercise Sheet 1: Introduction to R

Vangelis Theodorakis, Xueqi Cao, Daniela Andrade Salazar, Julien Gagneur

08 June, 2020

Contents

1 Vectors

First, create three named numeric vectors of size 10, 11 and 12 respectively in the following manner:

- One vector with the "colon" approach: *from:to*
- One vector with the `seq()` function: *seq(from, to)*
- And one vector with the `seq()` function and the `by` argument: *seq(from, to, by)*

For easier naming you can use the vector `letters` or `LETTERS` which contain the latin alphabet in small and capital, respectively. In order to select specific letters just use e.g. `letters[1:4]` to get the first four letters. Check their types. What is the outcome? Where do you think the difference comes from?

Then combine all three vectors in a list. Check the attributes of the vectors and the list. What is the difference and why?

Hint: If list elements have no names, we can access them with the double brackets and an index, e.g. `my_list[[1]]`

```
# Answer :

# A. Create vectors
vector.1 <- 1:10
names(vector.1) <- letters[vector.1]

vector.2 <- seq(1, 11)
names(vector.2) <- letters[vector.2]

vector.3 <- seq(1, 12, by = 1)
names(vector.3) <- letters[vector.3]

typeof(vector.1)
## [1] "integer"
typeof(vector.2)
## [1] "integer"
typeof(vector.3)
## [1] "double"

# B. Combine in a list
awesome.list <- list(vector.1, vector.2, vector.3)
attributes(vector.1)
## $names
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
attributes(vector.2)
## $names
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k"
attributes(vector.3)
## $names
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
attributes(awesome.list)
## NULL
attributes(awesome.list[[1]])
```

```
## $names
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"

## Why is the last vector of type double and not integer?
## By default seq returns integers from:to. But the `by`
## parameter returns always doubles

## myList got no names since we did not assign any compared to our vectors
```

2 Factors

```
f1 <- factor(letters)
levels(f1) <- rev(levels(f1))
f2 <- rev(factor(letters))
f3 <- factor(letters, levels = rev(letters))
```

The function `rev` reverses the order of an order-able object. What is the difference between f1, f2 and f3? Why?

```
# Answer :

f1 <- factor(letters)
levels(f1) <- rev(levels(f1))
# f1 goes from z - a, but the underlying encoding goes from z = 1 to a = 26
# We create the vector with the letters a to z and the mapped integer
# structure 1 to 26. THEN we reverse the levels = the mapping. As 1 becomes z
# and a becomes 26 the letters are mapped back to the unchanged integer
# structure and hence reversed.
f1
##  [1] z y x w v u t s r q p o n m l k j i h g f e d c b a
## Levels: z y x w v u t s r q p o n m l k j i h g f e d c b a

f2 <- rev(factor(letters))
# f2 goes from z - a, but the underlying encoding goes from a = 1 to z = 26
# We create the vector with the letters a to z and the mapped integer
# structure 1 to 26. Then we reverse the vector, i.e. the underlying integers,
# hence the vector gets reversed, but not the levels.
f2
##  [1] z y x w v u t s r q p o n m l k j i h g f e d c b a
## Levels: a b c d e f g h i j k l m n o p q r s t u v w x y z

f3 <- factor(letters, levels = rev(letters))
# f3 goes from a - z, but the underlying encoding goes from z = 1 to a = 26.
# We create the vector with the letters a to z BUT the mapped integer
# structure 26 to 1. Hence the levels but not the vector are reversed.
f3
##  [1] a b c d e f g h i j k l m n o p q r s t u v w x y z
## Levels: z y x w v u t s r q p o n m l k j i h g f e d c b a
```

```r
# Reversing f3 will give f1
rev(f3)
##  [1] z y x w v u t s r q p o n m l k j i h g f e d c b a
## Levels: z y x w v u t s r q p o n m l k j i h g f e d c b a
```

3 Computation on matrices

Create a 10 by 5 matrix which contains the the numbers from 1 to 50 column-wise. Name the rows as 'row_n' and columns as 'col_n'. Compute the mean and sum of each row and column. Add vector seq(60,100,10) as another row to the matrix.

Generate another matrix with the same dimensions, containing random numbers between 1 and 100. Subtract this matrix from the first one.

Plot the covariance matrix of the columns of the resulting matrix with spearman correlation coefficients.

Hint: Check out functions paste0(), colMeans(), rowMeans(), colSums(), rowSums(), sample(), cor() and corrplot() (in package 'corrplot')

```r
# Answer :
m <- matrix(1:50, ncol = 5)
rownames(m) <- paste0("row_", 1:nrow(m))
colnames(m) <- paste0("col_", 1:ncol(m))
colMeans(m)
## col_1 col_2 col_3 col_4 col_5
##   5.5  15.5  25.5  35.5  45.5
rowMeans(m)
##  row_1  row_2  row_3  row_4  row_5  row_6  row_7  row_8  row_9 row_10
##     21     22     23     24     25     26     27     28     29     30
colSums(m)
## col_1 col_2 col_3 col_4 col_5
##    55   155   255   355   455
rowSums(m)
##  row_1  row_2  row_3  row_4  row_5  row_6  row_7  row_8  row_9 row_10
##    105    110    115    120    125    130    135    140    145    150

m <- rbind(m,seq(60,100,10))
n <- matrix(sample(1:100, length(m), replace=T), ncol = ncol(m))
m <- m-n
M <- cor(m, method = "spearman")
#corrplot::corrplot(M, method = "number")
```

4 Data frame manipulation

Create a 3 by 4 matrix that contains the numbers 1 to 12 and then convert it into a data frame. Assign zero to the elements at row 2 which are greater than 4. Set the rownames to "row1", "row2", "row3" and column names to "col1", "col2", "col3" and "col4". Assign 0 to all elements in columns "col3" and "col4". Add a new column named "Letters" with values c("A", "B", "C"). Inspect the structure of the data frame.

```
#Answer :
x <- matrix(1:12,3,4)
x <- as.data.frame(x)
x
##   V1 V2 V3 V4
## 1  1  4  7 10
## 2  2  5  8 11
## 3  3  6  9 12
x[2, x[2,]>4] <- 0
rownames(x) <- paste0("row",1:3)
colnames(x) <- paste0("col",1:4)
x[,paste0("col",3:4)] <- 0
x$Letters <- c("A","B","C")
```

5 Data frame operations

Compute the number of women who survived the Titanic. Start by loading the data into a data frame using the following command:

```
tab <- read.csv("../../extdata/titanic.csv")
## Error in file(file, "rt"): cannot open the connection
```

```
head(tab)
## Error in head(tab): object 'tab' not found
# number of females survived the Titanic
nrow(tab[tab$sex == "female" & tab$survived == 1,])
## Error in nrow(tab[tab$sex == "female" & tab$survived == 1, ]): object 'tab' not found

# total number of passangers
nrow(tab)
## Error in nrow(tab): object 'tab' not found
```

6 Looping and writing your own functions

Write a function named "generateDataFrameSummary" which takes a data frame as input and outputs the medians of the rows and columns (NA values are discarded), and number of NA values in each row and column as a list.

```r
#Answer :

generateDataFrameSummary <- function(inputDF){

  return(list( rowMedians = apply(inputDF, 1, median, na.rm=T),
               colMedians = apply(inputDF, 2, median, na.rm=T),
               rowNANum   = apply(inputDF, 1, function(x){ sum(is.na(x)) }),
               colNANum   = apply(inputDF, 2, function(x){ sum(is.na(x)) })))

}

# test the function
df <- as.data.frame(matrix(rnorm(1000), nrow=50))
df[sample(c(T, F), nrow(df), replace=TRUE),
   sample(c(T, F), ncol(df), replace=TRUE)] <- NA
head(df)
##           V1          V2          V3         V4         V5        V6         V7
## 1 -0.1247252 -0.03369697 -1.63846635  1.1846881  0.4077047 -1.566437  0.2125627
## 2         NA          NA -0.23444217         NA         NA        NA -1.5233418
## 3         NA          NA  0.09825175         NA         NA        NA  0.2689837
## 4         NA          NA  0.79159661         NA         NA        NA -0.7469679
## 5 -0.3251337 -1.86513161 -0.82402457 -0.5296698 -1.8333851 -1.387791  0.3189131
## 6         NA          NA -0.34129167         NA         NA        NA -1.3462898
##            V8         V9         V10         V11        V12        V13
## 1  1.24074196 -0.4362921  0.45200986  0.40081830 -0.1770069 0.02173649
## 2  1.46491260         NA  1.01327830 -0.39540103 -0.7386880         NA
## 3 -0.05691632         NA -0.09312204 -0.79214624  0.1943725         NA
## 4  0.18395047         NA  0.91309656  0.30116232  1.5539355         NA
## 5 -0.41124726 -0.5265563  0.33640203 -0.77532851 -0.7823704 0.78016726
## 6  1.12799626         NA -0.06148710  0.06715794  0.6334482         NA
##          V14        V15         V16         V17         V18      V19        V20
## 1 -0.4892880  0.8505296  0.06721411  0.91042305 -0.54559218 1.120220  1.4093308
## 2 -0.3530381 -1.9535444  0.73151091 -0.08363822          NA       NA -1.7891092
## 3  0.9920929 -0.7307767  0.66080007 -0.93735494          NA       NA -0.9820880
## 4 -1.4683217 -1.4955113 -1.67329594  1.07616010          NA       NA  0.1954109
## 5  1.3293946  2.3617666 -0.29409099 -0.66651539 -0.03642664 0.617565 -0.8845018
## 6  0.7857377 -1.3475107  0.45137769  1.52312828          NA       NA  1.4605889
generateDataFrameSummary(df)
## $rowMedians
##  [1]  0.139888400 -0.353038054 -0.056916323  0.195410866 -0.468901757
##  [6]  0.451377692 -0.255290391  0.073613852 -0.059536615 -0.342390893
## [11] -0.232807456 -0.504808351  0.637012101 -0.481612844 -0.547877356
## [16] -0.165682845  0.222069366 -0.319885863  0.553734790  0.617228130
## [21] -0.852568114  0.215906972 -0.072562668 -0.128723754  0.057094350
## [26]  0.032161261 -0.163713527 -0.007021512  0.522737486 -0.289883752
## [31] -0.269763066  0.479099948  0.599860437 -0.093090754 -0.161380961
## [36] -0.003246077  0.343404237 -0.282538934  0.462060592  0.412874550
## [41] -0.033901956 -0.546574424 -0.262689241  0.213831060 -0.209326882
## [46] -0.214885722  0.083867533 -0.297863178  0.143093974 -0.163923893
##
## $colMedians
```

```
##          V1          V2          V3          V4          V5          V6
##  0.04109641  0.02235865 -0.22179269 -0.34394111 -0.78903514 -0.62003420
##          V7          V8          V9         V10         V11         V12
## -0.09087695 -0.02330022 -0.27192588 -0.00485598 -0.09377590  0.45040919
##         V13         V14         V15         V16         V17         V18
##  0.03812900 -0.15934218  0.08750901  0.01058144  0.06757430  0.34593335
##         V19         V20
##  0.11126602 -0.23716149
##
## $rowNANum
##  [1] 0 9 9 9 0 9 9 0 0 9 0 9 9 9 0 0 9 9 0 9 0 9 0 9 0 0 9 9 0 9 0 9 9 9 9 9 0 0
## [39] 0 9 9 9 0 9 9 0 9 0 0 9
##
## $colNANum
##  V1  V2  V3  V4  V5  V6  V7  V8  V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20
##  29  29   0  29  29  29   0   0  29   0   0   0  29   0   0   0   0  29  29   0
```