Fachhochschule Südwestfalen
University of Applied Sciences

Programmierung R - Exercise

Vectorization

May 10, SS 2022 || Hannah Behrens

Wir geben Impulse

Task 1

Your turn

1.1 Load the data set `Airquality3.csv` we have created at our second appointment from April 12, 2022 (Exercise *Data manipulation*).
1.2 Compute the mean of each numeric variable (use different `apply`- functions (`apply()`, `lapply()`, `sapply()` and `vapply()`)). How do the outputs differ?
1.3 Compute the median of `Temperature` dependent on the month.

Tasks 1.1 and 1.2 - answers

1.1

```r
# read csv-file named "Airquality3.csv" (exercise from April 12, 2022)
airqu3 <- read.table("Airquality3.csv", sep = ",")
```

1.2

```r
# the first six variables are numeric:
str(airqu3)
```

```
## 'data.frame':    112 obs. of  8 variables:
##  $ Ozone      : int  41 36 12 18 23 19 8 16 11 14 ...
##  $ Solar.R    : int  190 118 149 313 299 99 19 256 290 274 ...
##  $ Wind       : num  7.4 8 12.6 11.5 8.6 13.8 20.1 9.7 9.2 10.9 ...
##  $ Temperature: num  19.4 22.2 23.3 16.7 18.3 ...
##  $ Month      : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ Day        : int  1 2 3 4 7 8 9 12 13 14 ...
##  $ Month_name : chr  "May" "May" "May" "May" ...
##  $ Date       : chr  "1973-05-01" "1973-05-02" "1973-05-03" "1973-05-04" ...
```

Task 1.2 - apply() - answer

function apply()

```
# apply the mean function to the first six columns of airqu3
airqu3_apply <- apply(X = airqu3[,c(1:6)], MARGIN = 2, FUN = mean)
airqu3_apply
```

```
##       Ozone     Solar.R        Wind Temperature       Month         Day
##      41.902     185.830       9.929      25.392       7.205      15.857
```

```
typeof(airqu3_apply) # output: a vector
```

```
## [1] "double"
```

Task 1.2 - `lapply()` - answer

```r
airqu3_lapply <- lapply(X = airqu3[,c(1:6)], FUN = mean)
airqu3_lapply
```

```
## $Ozone
## [1] 41.9
##
## $Solar.R
## [1] 185.8
##
## $Wind
## [1] 9.929
##
## $Temperature
## [1] 25.39
##
## $Month
## [1] 7.205
##
## $Day
## [1] 15.86
```

function `lapply()`,
return value: a list

```r
typeof(airqu3_lapply) # output: a list
```

```
## [1] "list"
```

Task 1.2 - answer

```r
air_list <- list(Ozone = airqu3$Ozone, Solar.R = airqu3$Solar.R,
                 Wind = airqu3$Wind, Temperature = airqu3$Temperature,
                 Month = airqu3$Month, Day = airqu3$Day)
air_list
```

Saving `airqu3` as a list named `air_list`

```
## $Ozone
##   [1]  41  36  12  18  23  19   8  16  11  14  18  14  34   6  30  11   1  11
##  [19]   4  32  23  45 115  37  29  71  39  23  21  37  20  12  13 135  49  32
##  [37]  64  40  77  97  97  85  10  27   7  48  35  61  79  63  16  80 108  20
##  [55]  52  82  50  64  59  39   9  16 122  89 110  44  28  65  22  59  23  31
##  [73]  44  21   9  45 168  73  76 118  84  85  96  78  73  91  47  32  20  23
##  [91]  21  24  44  21  28   9  13  46  18  13  24  16  13  23  36   7  14  30
## [109]  14  18  20  20
##
## $Solar.R
##   [1] 190 118 149 313 299  99  19 256 290 274  65 334 307  78 322  44   8 320
##  [19]  25  92  13 252 223 279 127 291 323 148 191 284  37 120 137 269 248 236
##  [37] 175 314 276 267 272 175 264 175  48 260 274 285 187 220   7 294 223  81
##  [55]  82 213 275 253 254  83  24  77 255 229 207 192 273 157  71  51 115 244
##  [73] 190 259  36 212 238 215 203 225 237 188 167 197 183 189  95  92 252 220
##  [91] 230 259 236 259 238  24 112 237 224  27 238 201 238  14 139  49  20 193
## [109] 191 131 223 300
##
## $Wind
##   [1]  7.4  8.0 12.6 11.5  8.6 13.8 20.1  9.7  9.2 10.9 13.2 11.5 12.0 18.4 11.5
##  [16]  9.7  9.7 16.6  9.7 12.0 12.0 14.9  5.7  7.4  9.7 13.8 11.5  8.0 14.9 20.7
```

Task 1.2 - `lapply()` - answer

```r
airqu3_lapply2 <- lapply(X = air_list, FUN = mean)
airqu3_lapply2
```

```
## $Ozone
## [1] 41.9
##
## $Solar.R
## [1] 185.8
##
## $Wind
## [1] 9.929
##
## $Temperature
## [1] 25.39
##
## $Month
## [1] 7.205
##
## $Day
## [1] 15.86
```

applying `lapply()` to a list → a list is returned

```r
typeof(airqu3_lapply2) # output: a list
```

```
## [1] "list"
```

Task 1.2 - sapply() - answer

function sapply()

```r
airqu3_sapply <- sapply(X = airqu3[,c(1:6)], FUN = mean)
airqu3_sapply
```

```
##       Ozone     Solar.R        Wind Temperature       Month         Day
##      41.902     185.830       9.929      25.392       7.205      15.857
```

```r
typeof(airqu3_sapply) # output: a vector
```

```
## [1] "double"
```

Task 1.2 - sapply() - answer

applying sapply() to a list → a vector is returned when simplify = TRUE

```
airqu3_sapply2 <- sapply(air_list, FUN = mean)
airqu3_sapply2
```

```
##       Ozone     Solar.R        Wind Temperature       Month         Day
##      41.902     185.830       9.929      25.392       7.205      15.857
```

```
typeof(airqu3_sapply2) # output: a vector
```

```
## [1] "double"
```

What happens when simplify is set to FALSE?

Task 1.2 - sapply() - answer

```r
airqu3_sapply3 <- sapply(air_list, FUN = mean, simplify = FALSE)
airqu3_sapply3
```

Setting simplify = FALSE: a list is returned

```
## $Ozone
## [1] 41.9
##
## $Solar.R
## [1] 185.8
##
## $Wind
## [1] 9.929
##
## $Temperature
## [1] 25.39
##
## $Month
## [1] 7.205
##
## $Day
## [1] 15.86
```

```r
typeof(airqu3_sapply3) # output: a vector
```

```
## [1] "list"
```

Task 1.2 - `vapply()` - answer

function `vapply()`: specification of the return value (`FUN.VALUE`) is necessary

```r
airqu3_vapply <- vapply(X = airqu3[,c(1:6)], FUN = mean, FUN.VALUE = numeric(1))
airqu3_vapply
```

```
##       Ozone     Solar.R        Wind Temperature       Month         Day
##      41.902     185.830       9.929      25.392       7.205      15.857
```

```r
typeof(airqu3_vapply) # output: a vector
```

```
## [1] "double"
```

Task 1.3 - answer

1.3 Based on `Airquality3.csv`, compute the median of `Temperature` dependent on the month.

```
tapply(X = airqu3$Temperature, INDEX = airqu3$Month, FUN = median)
```

```
##     5     6     7     8     9
## 18.89 24.72 28.89 27.22 24.44
```

```
# or
tapply(X = airqu3$Temperature, INDEX = airqu3$Month_name, FUN = median)
```

```
##    August      July      June       May September
##     27.22     28.89     24.72     18.89     24.44
```

```
# --> is a big advantage compared to calculating the median of the temperature
# for each month piecewise like
median(airqu3$Temperature[which(airqu3$Month == 5)])
```

```
## [1] 18.89
```

```
# ... and so on
```

Task 2

Remember the data set `datasets::airquality` (R Core Team 2021a). It consists of the following variables

```
colnames(airquality)
```

```
## [1] "Ozone"   "Solar.R" "Wind"    "Temp"    "Month"   "Day"
```

which are numeric.

We are interested in all possible and different linear models when one of these six variables is considered as the dependent variable y. How can we achieve this?

How many combinations of different linear models exist for each y?

Task 2 - number of combinations - answer

For each y:

- N = 5 remaining variables as predictors to form linear models, e.g. y = `Ozone`:

```
lm(Ozone ~ 1 + Solar.R + Wind + Temp + Month + Day)
# but also
lm(Ozone ~ 1 + Solar.R + Wind + Temp + Month)
# as well as
lm(Ozone ~ 1 + Solar.R + Wind + Temp + Day)
# and so on...
```

- As we have seen, a linear model for `Ozone` can be created based on 5, 4, 3, 2 and 1 variable(s) as predictor(s).
- Each predictor appears only **once** in each linear model and the ordering **does not** play any role: So, the number of combinations when picking up 5 of 5, 4 of 5 predictors and so on is calculated based on the binomial coefficient $\binom{N}{n} = \frac{N!}{n! \cdot (N-n)!}$. (Fahrmeir et al. 2016, 187f.)

Task 2 - number of combinations - answer

From that it follows:

- When forming a model with 5 predictors, only 1 combination is possible: $\binom{5}{5} = \frac{5!}{5! \cdot (5-5)!} = 1$
- When forming a model with 4 predictors, 5 combinations are possible: $\binom{5}{4} = \frac{5!}{4! \cdot (5-4)!} = 5$

$\rightarrow$ So, for each *y*

```
combi_per_y <- choose(5,5) + choose(5,4) + choose(5,3) + choose(5,2) + choose(5,1)
combi_per_y
```

```
## [1] 31
```

different linear models can be created and examined.

Task 2 - implementation - step 1

Your turn

Based on the presented theory, the first step is to *combine the predictors* to create 31 linear models in the next step. These *combinations* should be saved in a list. How can we do this?

Task 2 - implementation - step 1 - answer

Combining the predictors to create 31 linear models

```r
## main program
data <- airquality
coln_names <- colnames(data) # get column names

y <- coln_names[1] # e.g. Ozone (column 1) as the dependent variable (y)
vars_grid <- coln_names[-1] # the remaining variables are predictors
variables_grid <- expand.grid(replicate(length(vars_grid), vars_grid,
                     simplify = FALSE)) # make a grid with all possible combinations
    # since each predictor can only be listed once in the linear model, remove replicates
variables_grid_unique <- apply(variables_grid, MARGIN = 1, FUN = unique)
# sort entries of the sublists since the order of the predictors does not play any role
variables_grid_sort <- lapply(variables_grid_unique, sort)
    # then remove replicates again
variables_grid_sort_unique <- unique(variables_grid_sort)
   # add y to the end of all sublists:
variables_grid_sort_unique_y <- lapply(variables_grid_sort_unique,
                                FUN = function(x){c(x, x[length(x)+1] <- y)})
air_grid_lm <- variables_grid_sort_unique_y
    # ... something is missing here
```

What to do next? Any ideas? Suggestions?

Task 2 - step 2

Your turn

Based on our list `air_grid_lm`, we have to create the formulas of each linear model with y (here: `Ozone`) as dependent variable. How can you implement it? Do it!

Task 2 - step 2 - implementation - answer

```r
# x: the first element(s) of each sublist (= predictors)
# y: the last element of each sublist
determiningLinMod <- function(x, y = x[length(x)], dataset = airquality){
  # predictors x1 ,..., x_{length(x)-1}
  predictors <- paste(x[-length(x)], collapse = " + ") # concatenate the predictors
  # by a + -sign
  make_formula <- paste(y, " ~ 1 + ", predictors)
  formula_lm <- as.formula(make_formula) # convert the string to a formula
  lin_model <- lm(formula_lm, data = dataset) # call lm()
  # return the different formulas as well as the model
  return(list(formula = formula_lm, model = lin_model))
}
```

```r
# apply the function determiningLinMod() to each combination listed in air_grid_lm:
lapply_lm_air <- lapply(X = air_grid_lm, FUN = determiningLinMod)
```

Task 2 - step 2 - implementation - answer

```r
## main program
data <- airquality
coln_names <- colnames(data) # get column names

y <- coln_names[1] # e.g. Ozone (column 1) as the dependent variable (y)
vars_grid <- coln_names[-1] # the remaining variables are predictors
variables_grid <- expand.grid(replicate(length(vars_grid), vars_grid,
                     simplify = FALSE)) # make a grid with all possible combinations
    # since each predictor can only be listed once in the linear model, remove replicates
variables_grid_unique <- apply(variables_grid, MARGIN = 1, FUN = unique)
# sort entries of the sublists since the order of the predictors does not play any role
variables_grid_sort <- lapply(variables_grid_unique, sort)
    # then remove replicates again
variables_grid_sort_unique <- unique(variables_grid_sort)
   # add y to the end of all sublists:
variables_grid_sort_unique_y <- lapply(variables_grid_sort_unique,
                                 FUN = function(x){c(x, x[length(x)+1] <- y)})
air_grid_lm <- variables_grid_sort_unique_y
# insert:
lapply_lm_air <- lapply(air_grid_lm, FUN = determiningLinMod)
```

Task 2 - step 2 - implementation - results

```r
length(lapply_lm_air) # 31 combinations expected (see some slides before)
```

```
## [1] 31
```

```r
lapply_lm_air
```

```
## [[1]]
## [[1]]$formula
## Ozone ~ 1 + Solar.R
## <environment: 0x00000000255a5cd0>
##
## [[1]]$model
##
## Call:
## lm(formula = formula_lm, data = data)
##
## Coefficients:
## (Intercept)      Solar.R
##      18.599        0.127
##
##
##
## [[2]]
## [[2]]$formula
## Ozone ~ 1 + Solar.R + Wind
```

Task 2 - step 3 - parallelization

Very nice! We get all possible linear models for the variable `Ozone` of the `airquality` data set (R Core Team 2021a).

But now, we want to go a step further by parallelizing the determination of the 31 linear models for `Ozone`.

How can we achieve this?

Task 2 - step 3 - parallelization

The function `parallel::mclapply()` (R Core Team 2021b) is available which works as follows (The example is taken from Peng (2020)):

```r
 r <- mclapply(1:10, function(i) {
         Sys.sleep(10)  # Do nothing for 10 seconds
}, mc.cores = 1)
r
```

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

Task 2 - step 3 - parallelization

```r
parallelized_lm_air <- parallel::mclapply(X = air_grid_lm,
                                          FUN = determiningLinMod, mc.cores = 1)
parallelized_lm_air
```

Instead of using `lapply()` `mclapply` (R Core Team 2021b) is used.
Parallelization based on `mclapply` is not possible on Windows.

```
## [[1]]
## [[1]]$formula
## Ozone ~ 1 + Solar.R
## <environment: 0x00000000279626e0>
##
## [[1]]$model
##
## Call:
## lm(formula = formula_lm, data = data)
##
## Coefficients:
## (Intercept)      Solar.R
##      18.599        0.127
##
##
##
## [[2]]
## [[2]]$formula
## Ozone ~ 1 + Solar.R + Wind
## <environment: 0x00000002799f010>
##
## [[2]]$model
```

Task 2 - step 3 - parallelization

```
length(parallelized_lm_air)
```

```
## [1] 31
```

The expectations have been fulfilled!

Task 3

The following list is given:

```r
L <- list(money = c(250, 124, 360, 720, 340, 340),
          hours = c(19, 12, 30, 48, 26, 25),
          idx = c(1:6),
          name = c("Paul", "Emma", "Mia", "John", "Kim", "Maxi"))
```

Your turn

Make use of the `*apply()`-functions and

a) compute the total sum of `money`, `hours` and `idx` and return a list.

b) compute the mean of `money` and `hours` and return a vector.

c) multiply each element of `money` and `hours` by 1.5 and return a matrix.

Tasks 3a) and 3b) - answers

```r
# 3.a)
lapply(X = L[1:3], FUN = sum)
```

```
## $money
## [1] 2134
##
## $hours
## [1] 160
##
## $idx
## [1] 21
```

```r
# 3.b)
sapply(X = L[1:2], FUN = mean)
```

```
##  money  hours
## 355.67  26.67
```

```r
# vapply(L[1:2], mean, FUN.VALUE=numeric(1)) # alternative
# --> the value specified by FUN.VALUE (here: numeric(1)) and the actual value returned
# (here: the mean of money and hours) have to match
```

Task 3.c) - answer

```r
sapply(X = L[1:2], FUN = function(x){x * 1.5})
```

```
##      money hours
## [1,]   375  28.5
## [2,]   186  18.0
## [3,]   540  45.0
## [4,]  1080  72.0
## [5,]   510  39.0
## [6,]   510  37.5
```

For home

Document the code (main program (slide 23) and function `determiningLinMod()`) as you have learned in the last lecture and exercise about *Software Development*.

You can also think of wrapping the code from slide 23 in a function.

References

Buchwitz, B. 2021. *Computational Statistics*. https://bchwtz.github.io/bchwtz-cswr/.

Fahrmeir, L., C. Heumann, R. Künstler, I. Pigeot, and G. Tutz. 2016. *Statistik. Der Weg Zur Datenanalyse*. 8th ed. Springer-Lehrbuch. Berlin: Springer. https://doi.org/10.1007/978-3-662-50372-0.

Peng, R. D. 2020. *R Programming for Data Science. 22.3 the Parallel Package*. Vienna, Austria: R Foundation for Statistical Computing. https://bookdown.org/rdpeng/rprogdatascience/parallel-computation.html#the-parallel-package.

R Core Team. 2021a. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

———. 2021b. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.