Problem Set 3 Solution

Andreas Bender, Philipp Kopper, Philip Studener

09 November 2023

Resources

Read chapter 3 and 5 in The Art of R Programming. You can leave out the Extended Examples once again and Section 5.3.

Keyboard Shortcuts

Try incorporating the following shortcuts into your workflow for this weeks exercise sheet:

(On Mac you can replace `CTRL` with `CMD` but don't have to)

Shortcut	What does it do?
`CTRL + Shift + N`	opens new R-Script file
`F1`	Go to function help page (cursor over function)
`F2`	Get function code in new tab
`CTRL + Shift + F`	Search for keyword in all files in your directory

Exercise 1

Consider the `bundesliga` vector from the first exercise sheet.

```r
bundesliga <- c(
    "FC Bayern"             = 55L,
    "BVB"                   = 51L,
    "RB Leipzig"            = 50L,
    "Borussia MGB"          = 49L,
    "Bayer 04"              = 47L,
    "FC Schalke 04"         = 37L,
    "VfL Wolfsburg"         = 36L,
    "SC Freiburg"           = 36L,
    "TSG Hoffenheim"        = 35L,
    "1. FC Köln"            = 32L,
    "Union Berlin"          = 30L,
    "Eintracht Frankfurt" = 28L,
    "Hertha Berlin"         = 28L,
    "FC Augsburg"           = 27L,
    "Mainz 05"              = 26L,
    "Fortuna Duesseldorf" = 22L,
    "Werder Bremen"         = 18L,
    "SC Paderborn"          = 16L)
```

a. Convert the `bundesliga` vector to a matrix with one column and 18 rows. Store this matrix in an object called `bundesliga_mat` (*Hint*: ARP, Section 3.1). The resulting matrix is given below:

```
bundesliga_mat <- matrix(bundesliga, ncol = 1)
```

```
bundesliga_mat
```

```
##       [,1]
##  [1,]   55
##  [2,]   51
##  [3,]   50
##  [4,]   49
##  [5,]   47
##  [6,]   37
##  [7,]   36
##  [8,]   36
##  [9,]   35
## [10,]   32
## [11,]   30
## [12,]   28
## [13,]   28
## [14,]   27
## [15,]   26
## [16,]   22
## [17,]   18
## [18,]   16
```

b. What is the data type of `bundesliga_mat`. What is its class?

```
typeof(bundesliga_mat)
```

```
## [1] "integer"
```

```
class(bundesliga_mat)
```

```
## [1] "matrix" "array"
```

c. Extract the points for the first 3 teams from this matrix. Then extract the last three teams. Store them in in the objects `first3` and `last3` respectively (*Hint: ARP, Section 3.2. Also consider PS01, E2 and think about the equivalent of `length` for matrices (e.g. ARP, Section 3.5)*). The resulting output is given below:

```
first3 <- bundesliga_mat[1:3, ]
last3 <- bundesliga_mat[(nrow(bundesliga_mat)-2):nrow(bundesliga_mat), ]
```

```
first3
```

```
## [1] 55 51 50
```

```
last3
```

```
## [1] 22 18 16
```

d. What is the data type of `first3` and `last3`. What is their class.

```
typeof(first3)
```

```
## [1] "integer"
```

```
typeof(last3)
```

```
## [1] "integer"
```

```
class(first3)
```

```
## [1] "integer"
```

```
class(last3)
```

```
## [1] "integer"
```

e. Repeat c., but make sure, that the class of `first3` and `last3` remains a matrix (*Hint*: See `?"["` and ARP, Section 3.6).

```
first3 <- bundesliga_mat[1:3, , drop = FALSE]
first3 <- head(bundesliga_mat, 3)
last3 <- bundesliga_mat[(nrow(bundesliga_mat)-2):nrow(bundesliga_mat), , drop = FALSE]
last3 <- tail(bundesliga_mat, 3)
class(first3)
```

```
## [1] "matrix" "array"
```

```
class(last3)
```

```
## [1] "matrix" "array"
```

f. Currently, `bundesliga_mat` only contains the information about the points, but not the team that collected them. Create a copy of `bundesliga_mat` called `bundesliga_mat2`. Store the team names as row names in `bundesliga_mat2` (*Hint*: ARP, Section 3.7). The resulting output is given below:

```
bundesliga_mat2 <- bundesliga_mat
rownames(bundesliga_mat2) <- names(bundesliga)
```

```
bundesliga_mat2
```

```
##                       [,1]
## FC Bayern               55
## BVB                     51
## RB Leipzig              50
## Borussia MGB            49
## Bayer 04                47
## FC Schalke 04           37
## VfL Wolfsburg           36
## SC Freiburg             36
## TSG Hoffenheim          35
## 1. FC Köln              32
## Union Berlin            30
## Eintracht Frankfurt     28
## Hertha Berlin           28
## FC Augsburg             27
## Mainz 05                26
## Fortuna Duesseldorf     22
## Werder Bremen           18
## SC Paderborn            16
```

g. Is data type, class or dimension (`?dim`) of `bundesliga_mat` different then the one of `bundesliga_mat2`?

Solution: No, data type, class and dimensions of the two matrices are the same.

h. Create another copy of `bundesliga_mat` called `bundesliga_mat3`. Add a second column to this matrix that contains the teams rank ("Tabellenplatz") names (*Hint*: ARP, Section 3.4.1). The resulting output is given below:

```
bundesliga_mat3 <- bundesliga_mat
bundesliga_mat3 <- cbind(bundesliga_mat3, 1:18)
```

```
bundesliga_mat3
```

```
##       [,1] [,2]
##  [1,]   55    1
##  [2,]   51    2
##  [3,]   50    3
##  [4,]   49    4
##  [5,]   47    5
##  [6,]   37    6
##  [7,]   36    7
##  [8,]   36    8
##  [9,]   35    9
## [10,]   32   10
## [11,]   30   11
## [12,]   28   12
## [13,]   28   13
## [14,]   27   14
## [15,]   26   15
## [16,]   22   16
## [17,]   18   17
## [18,]   16   18
```

i. What is the data type, class and dimension of `bundesliga_mat3`?

```
typeof(bundesliga_mat3)
```

```
## [1] "integer"
```

```
class(bundesliga_mat3)
```

```
## [1] "matrix" "array"
```

```
dim(bundesliga_mat3)
```

```
## [1] 18  2
```

j. Add a third column to `bundesliga_mat3`, that indicates in which German federal state the team is located. The `states` vector and the resulting output are given below:

```
c("BAY","NRW","SXN","NRW","NRW","NRW","NSX","BWB","BWB","NRW","BER","HES",
  "BER","BAY","RLP","NRW","BRE","NRW")
```

```
bundesliga_mat3 <- cbind(
    bundesliga_mat3,
    c("BAY","NRW","SXN","NRW","NRW","NRW","NSX","BWB","BWB","NRW","BER","HES",
      "BER","BAY","RLP","NRW","BRE","NRW"))
```

```
bundesliga_mat3
```

```
##       [,1] [,2] [,3]
##  [1,] "55" "1"  "BAY"
##  [2,] "51" "2"  "NRW"
##  [3,] "50" "3"  "SXN"
##  [4,] "49" "4"  "NRW"
##  [5,] "47" "5"  "NRW"
##  [6,] "37" "6"  "NRW"
##  [7,] "36" "7"  "NSX"
##  [8,] "36" "8"  "BWB"
##  [9,] "35" "9"  "BWB"
## [10,] "32" "10" "NRW"
## [11,] "30" "11" "BER"
## [12,] "28" "12" "HES"
## [13,] "28" "13" "BER"
## [14,] "27" "14" "BAY"
## [15,] "26" "15" "RLP"
## [16,] "22" "16" "NRW"
## [17,] "18" "17" "BRE"
## [18,] "16" "18" "NRW"
```

k. What is the data type, class and dimension of `bundesliga_mat3`? What changed compared to `bundesliga_mat` and `bundesliga_mat2` and why? Discuss why this is not a desirable behaviour.

```
typeof(bundesliga_mat3)
```

```
## [1] "character"
```

```
class(bundesliga_mat3)
```

```
## [1] "matrix" "array"
```

```
dim(bundesliga_mat3)
```

```
## [1] 18  3
```

Solution: Since matrices can only handle one data type, and a column of type character got added, the matrix gets converted to type chararcter. This is undesireable because it makes calculations on the numeric columns more difficult.

l. Currently, `bundesliga_mat3` doesn't have column names. Add column names `"points"`, `"rank"` and `"state"` to the matrix. The resulting output is given below:

```
colnames(bundesliga_mat3) <- c("points", "rank", "state")
```

```
bundesliga_mat3
```

```
##       points rank state
##  [1,] "55"   "1"  "BAY"
##  [2,] "51"   "2"  "NRW"
##  [3,] "50"   "3"  "SXN"
##  [4,] "49"   "4"  "NRW"
##  [5,] "47"   "5"  "NRW"
##  [6,] "37"   "6"  "NRW"
##  [7,] "36"   "7"  "NSX"
##  [8,] "36"   "8"  "BWB"
##  [9,] "35"   "9"  "BWB"
## [10,] "32"   "10" "NRW"
## [11,] "30"   "11" "BER"
## [12,] "28"   "12" "HES"
## [13,] "28"   "13" "BER"
## [14,] "27"   "14" "BAY"
## [15,] "26"   "15" "RLP"
## [16,] "22"   "16" "NRW"
## [17,] "18"   "17" "BRE"
## [18,] "16"   "18" "NRW"
```

m. To overcome the problem in k., convert `bundesliga_mat3` to a `data.frame` by applying the function `as.data.frame` and store the result in object `bundesliga_df`.

```r
bundesliga_df <- as.data.frame(bundesliga_mat3)
```

```r
bundesliga_df
```

```
##    points rank state
## 1      55    1   BAY
## 2      51    2   NRW
## 3      50    3   SXN
## 4      49    4   NRW
## 5      47    5   NRW
## 6      37    6   NRW
## 7      36    7   NSX
## 8      36    8   BWB
## 9      35    9   BWB
## 10     32   10   NRW
## 11     30   11   BER
## 12     28   12   HES
## 13     28   13   BER
## 14     27   14   BAY
## 15     26   15   RLP
## 16     22   16   NRW
## 17     18   17   BRE
## 18     16   18   NRW
```

n. What is the data type, class and dimension of `bundesliga_df`?

```r
typeof(bundesliga_df)
```

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

```r
class(bundesliga_df)
```

```
## [1] "data.frame"
```

```r
dim(bundesliga_df)
```

```
## [1] 18  3
```

o. Use the `apply` function in order to extract the class of each column in `bundesliga_df` (*HInt:* ARP, Section 3.3.1 and 5.2.4). The resulting output is given below:

```r
apply(bundesliga_df, 2, class)
```

```
##      points        rank       state 
## "character" "character" "character"
```

p. Internally, `data.frame` objects are stored as lists. Individual components of the list (columns in this case), can therefore be manipulated and overwritten the same way as list components (see ARP, Section 4.2.2). Use this knowledge to add two new columns to `bundesliga_df` that contain the points and rank as integer, rather then character. Call them `points_int` and `rank_int`. Delete the columns `points` and `rank`. The resulting output is shown below

```r
bundesliga_df$points_int <- bundesliga
bundesliga_df$rank_int   <- 1:18
bundesliga_df$points     <- bundesliga_df$rank <- NULL
```

```
bundesliga_df
```

```
##    state points_int rank_int
## 1    BAY         55        1
## 2    NRW         51        2
## 3    SXN         50        3
## 4    NRW         49        4
## 5    NRW         47        5
## 6    NRW         37        6
## 7    NSX         36        7
## 8    BWB         36        8
## 9    BWB         35        9
## 10   NRW         32       10
## 11   BER         30       11
## 12   HES         28       12
## 13   BER         28       13
## 14   BAY         27       14
## 15   RLP         26       15
## 16   NRW         22       16
## 17   BRE         18       17
## 18   NRW         16       18
```

```
str(bundesliga_df)
```

```
## 'data.frame':    18 obs. of  3 variables:
##  $ state     : chr  "BAY" "NRW" "SXN" "NRW" ...
##  $ points_int: int  55 51 50 49 47 37 36 36 35 32 ...
##  $ rank_int  : int  1 2 3 4 5 6 7 8 9 10 ...
```

q. Repeat o. to check the class of the columns of `bundesliga_df`. Discuss with us during the live-sessions why this contradicts the output of `str(bundesliga_df)`.

Exercise 2

Create the following data frame (*Hint*: ARP, Section 5.1). Call the data frame `df`.

```
name   <- c("Alex", "Lilly", "Mark", "Oliver", "Martha", "Lucas", "Caroline")
age    <- c(25, 31, 23, 52, 76, 49, 26)
height <- c(177, 163, 190, 179, 163, 183, 164)
weight <- c(57, 69, 83, 75, 70, 83, 53)
gender <- c("D", "F", "M", "M", "F", "M", "F")
df     <- data.frame(name, age, height, weight, gender)
```

```
df
```

```
##       name age height weight gender
## 1     Alex  25    177     57      D
## 2    Lilly  31    163     69      F
## 3     Mark  23    190     83      M
## 4   Oliver  52    179     75      M
## 5   Martha  76    163     70      F
## 6    Lucas  49    183     83      M
## 7 Caroline  26    164     53      F
```

```
str(df)
```

```
## 'data.frame':    7 obs. of  5 variables:
##  $ name  : chr  "Alex" "Lilly" "Mark" "Oliver" ...
##  $ age   : num  25 31 23 52 76 49 26
##  $ height: num  177 163 190 179 163 183 164
##  $ weight: num  57 69 83 75 70 83 53
##  $ gender: chr  "D" "F" "M" "M" ...
```

a. The variable `gender` should be of class factor. Change the variable accordingly if necessary and change the levels to `female`, `male` and `diverse` instead of `F`, `M` and `D`.

```
df$gender <- factor(df$gender, levels = c("F", "M", "D"), labels = c("female", "male", "diverse"))
df$gender
```

```
## [1] diverse female  male    male    female  male    female 
## Levels: female male diverse
```

```
df[["gender"]]
```

```
## [1] diverse female  male    male    female  male    female 
## Levels: female male diverse
```

```
df[, "gender"]
```

```
## [1] diverse female  male    male    female  male    female 
## Levels: female male diverse
```

b. Create a data frame that contains the column `working`, that indicates whether the person has a job (`"Yes"`) or not (`"No"`). Call the data frame `df2`. The expected output is given below:

```
df2 <- data.frame(working = c("Yes","No","No","Yes","Yes","No","Yes"))
```

```
df2
```

```
##   working
## 1     Yes
## 2      No
## 3      No
## 4     Yes
## 5     Yes
## 6      No
## 7     Yes
```

c. Combine the two data frames `df` and `df2` columnwise. Store the result in object `df3`. The resulting output is given below:

```
df3 <- cbind(df, df2)
df3
```

```
##       name age height weight  gender working
## 1     Alex  25    177     57 diverse     Yes
## 2    Lilly  31    163     69  female      No
## 3     Mark  23    190     83    male      No
## 4   Oliver  52    179     75    male     Yes
## 5   Martha  76    163     70  female     Yes
## 6    Lucas  49    183     83    male      No
## 7 Caroline  26    164     53  female     Yes
```

d. Transform the `working` column to type `logical`, that contains `TRUE` if the person is working (`Yes`), and `FALSE` if the person is not working (`No`)).

```
df3$working <- df3$working == "Yes"
```

e. How many rows and columns does `df3` have?

```
dim(df3)
```

```
## [1] 7 6
```

```
# Oder:
nrow(df3)
```

```
## [1] 7
```

```
ncol(df3)
```

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

f. What data type and class is each column? *Hint: Use* `sapply`*, ARP, Section 4.4.1 (remember, columns of data frames are equivalent to components of lists)*

```
str(df3)
```

```
## 'data.frame':    7 obs. of  6 variables:
##  $ name   : chr  "Alex" "Lilly" "Mark" "Oliver" ...
##  $ age    : num  25 31 23 52 76 49 26
##  $ height : num  177 163 190 179 163 183 164
##  $ weight : num  57 69 83 75 70 83 53
##  $ gender : Factor w/ 3 levels "female","male",..: 3 1 2 2 1 2 1
##  $ working: logi  TRUE FALSE FALSE TRUE TRUE FALSE ...
```

```
sapply(df3, typeof)
```

```
##        name         age      height      weight      gender     working
## "character"    "double"    "double"    "double"   "integer"   "logical"
```

```
sapply(df3, class)
```

```
##        name         age      height      weight      gender     working
## "character"   "numeric"   "numeric"   "numeric"    "factor"   "logical"
```

g. Create a new column `bmi` for which you calculate the BMI for each person (see https://de.wikipedia.org/wiki/Body-Mass-Index).

```
df3$bmi <- df3$weight / (df3$height / 100)^2
```

h. Create a subset of `df3` that only contains entries of males. Call it `males_df` (*Hint*: ARP, Section 5.2.1).

```
males_df <- subset(df3, gender == "male")
# oder
males_df <- df3[df3$gender == "male", ]
```

i. Print out all the rows in `males_df` which have a `bmi` over 23.

```
males_df[males_df$bmi > 23, ]
```

```
##     name age height weight gender working      bmi
## 4 Oliver  52    179     75   male    TRUE 23.40751
## 6  Lucas  49    183     83   male   FALSE 24.78426
```

```
# Oder
subset(males_df, bmi > 23)
```

```
##     name age height weight gender working      bmi
## 4 Oliver  52    179     75   male    TRUE 23.40751
## 6  Lucas  49    183     83   male   FALSE 24.78426
```

j. Print out a subset of `males_df` that only contains the columns `name` and `bmi`.

```
males_df[, c("name", "bmi")]
```

```
##     name      bmi
## 3   Mark 22.99169
## 4 Oliver 23.40751
## 6  Lucas 24.78426
```

Session info

```
sessionInfo()
```

```
## R version 4.3.0 (2023-04-21)
## Platform: x86_64-apple-darwin20 (64-bit)
## Running under: macOS Big Sur 11.7.10
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/libRlapack.dylib;  LAPACK
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: Europe/Berlin
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] compiler_4.3.0  fastmap_1.1.1   cli_3.6.1       tools_4.3.0
##  [5] htmltools_0.5.5 rstudioapi_0.14 yaml_2.3.7      rmarkdown_2.21 
##  [9] knitr_1.43      xfun_0.39       digest_0.6.31   rlang_1.1.1    
## [13] evaluate_0.21
```