R Scripting

Exercises for unit 2 - Data structures (II)

Marcus Wurzer

Please solve the following problems!

1. Try to answer quiz questions 4. and 5. at the beginning of the "Data structures" chapter of the coursebook: http://adv-r.had.co.nz/Data-structures.html

   See coursebook for answers!

2. Try to answer quiz questions 1. to 5. at the beginning of the "Subsetting" chapter of the coursebook: http://adv-r.had.co.nz/Subsetting.html

   See coursebook for answers!

3. Coursebook exercises:

   a. What are the six types of atomic vector? How does a list differ from an atomic vector?

      The six types are logical, integer, double, character, complex and raw. The elements of a list don't have to be of the same type.

   b. Test your knowledge of vector coercion rules by predicting the output of the following uses of `c()`:

```
c(1, FALSE)       # will be coerced to numeric   -> 1 0
c("a", 1)         # will be coerced to character -> "a" "1"
c(list(1), "a")   # will be coerced to a list with two elements of type double and character
c(TRUE, 1L)       # will be coerced to integer   -> 1 1
```

   c. Why is the default missing value, `NA`, a logical vector? What's special about logical vectors? (Hint: think about `c(FALSE, NA_character_)`.)

      It is a practical thought. When you combine `NA`s in `c()` with other atomic types they will be coerced like `TRUE` and `FALSE` to integer `(NA_integer_)`, double `(NA_real_)`, complex `(NA_complex_)` and character `(NA_character_)`. Recall that in R there is a hierarchy of recursion that goes logical -> integer -> double -> character. If `NA` were, for example, a character, including `NA` in a set of integers or logicals would result in them getting coerced to characters which would be undesirable. Making `NA` a logical means that involving an `NA` in a dataset (which happens often) will not result in coercion.

   d. What does `dim()` return when applied to a vector?

      `NULL`

   e. How would you describe the following three objects? What makes them different to `1:5`?

```
x1 <- array(1:5, c(1, 1, 5)) # 1 row, 1 column, 5 in third dimension
x2 <- array(1:5, c(1, 5, 1)) # 1 row, 5 columns, 1 in third dimension
x3 <- array(1:5, c(5, 1, 1)) # 5 rows, 1 column, 1 in third dimension
```

      They are of class array and so they have a `dim` attribute.

f. What does `as.matrix()` do when applied to a data frame with columns of different types?

   From `?as.matrix`: The method for data frames will return a character matrix if there is only atomic columns and any non-(numeric/logical/complex) column, applying as.vector to factors and format to other non-character columns. Otherwise the usual coercion hierarchy (logical < integer < double < complex) will be used, e.g., all-logical data frames will be coerced to a logical matrix, mixed logical-integer will give a integer matrix, etc.

g. Fix each of the following common data frame subsetting errors:

```
mtcars[mtcars$cyl = 4, ]        # = -> ==
mtcars[-1:4, ]                  # -1:4 -> -(1:4)
mtcars[mtcars$cyl <= 5]         # "," is missing
mtcars[mtcars$cyl == 4 | 6, ]  # 6 -> mtcars$cyl == 6
```

h. Why does `mtcars[1:20]` return an error? How does it differ from the similar `mtcars[1:20, ]`?

   In the first case `mtcars` is subsetted with a vector and the statement should return a data.frame of the first 20 columns in `mtcars`. Since `mtcars` only has 11 columns, the index is out of bounds, which explains the error. The biggest difference of `mtcars[1:20, ]` to the former case, is that now `mtcars` is subsetted with two vectors. In this case you will get returned the first 20 rows and all columns (like subsetting a matrix).

i. What does `df[is.na(df)] <- 0` do? How does it work?

   It replaces all `NA`s within df with the value `0`. `is.na(df)` returns a logical matrix which is used to subset `df`. Since you can combine subsetting and assignment, only the matched part of `df` (the `NA`s) is replaced with `0` entries.

j. How would you randomly permute the columns of a data frame? (This is an important technique in random forests.) Can you simultaneously permute the rows and columns in one step?

```
iris[sample(ncol(iris))] # permute rows
iris[sample(nrow(iris)), sample(ncol(iris)), drop = FALSE] # permute both at the same time
```

k. How could you put the columns in a data frame in alphabetical order?

```
iris[sort(names(iris))]
```

4. The following table is given:

	name	figure	height	weight	drinks	job
1	Susi	chubby	1.97	98	TRUE	student
2	Tim	chubby	1.67	89	n/a	student
3	Christine	beefy	1.90	71	TRUE	student
4	Mathias	skinny	1.81	86	TRUE	employed

(n/a = not available)

a. Generate the variables

   name as character string
   job as factor with the following categories: *student, employed, self-employed*
   figure as ordered factor (using function `ordered()`) with the following categories: *skinny, lean, slim, normal, chubby, beefy*
   height as numeric (continuous) variable
   weight as numeric (discrete) variable
   drinks as logical (binary) variable

   Pay attention to the correct specification of missing values!

b. Generate a data frame `friends` from these variables that looks similar to the table given above and print it!

```
# a.
name <- c("Susi", "Tim", "Christine", "Mathias")
job <- factor(c(rep("student", 3), "employed"),
              levels = c("student", "employed", "self-employed"))
figure <- ordered(c("chubby", "chubby", "beefy", "skinny"),
                  levels = c("skinny", "lean", "slim", "normal", "chubby", "beefy"))
height <- c(1.97, 1.67, 1.90, 1.81)
weight <- c(98L, 89L, 71L, 86L)
drinks <- c(TRUE, NA, TRUE, TRUE)

# b.
friends <- data.frame(name, figure, height, weight, drinks, job)
friends
```

```
##        name figure height weight drinks      job
## 1      Susi chubby   1.97     98   TRUE  student
## 2       Tim chubby   1.67     89     NA  student
## 3 Christine  beefy   1.90     71   TRUE  student
## 4   Mathias skinny   1.81     86   TRUE employed
```

5. Dataset `chickwts` is included in R:

```
data(chickwts)
?chickwts
```

It is a data frame that contains two variables: The weight of chickens contingent upon various feed supplements (`weight` and `feed`).

a. Print the first 6 rows of the dataset.
b. Extract all factor levels of the variable `feed`.
c. Extract all rows of the data frame for chickens that are fed with **meatmeal** (Hint: `subset()`).
d. Extract the weight of the chickens that are fed with **casein** *or* **horsebean** (Hint: Operator for logical or: `|`).
e. Compute the mean chicken weight separately for each feed supplement (Hint: `aggregate()`).

```
# a.
head(chickwts)
```

```
##   weight      feed
## 1    179 horsebean
## 2    160 horsebean
## 3    136 horsebean
## 4    227 horsebean
## 5    217 horsebean
## 6    168 horsebean
```

```
# b.
levels(chickwts$feed)
```

```
## [1] "casein"    "horsebean" "linseed"   "meatmeal"  "soybean"   "sunflower"
```

```
# c.
subset(chickwts, feed == "meatmeal")
```

```
##    weight     feed
## 49    325 meatmeal
## 50    257 meatmeal
## 51    303 meatmeal
## 52    315 meatmeal
## 53    380 meatmeal
## 54    153 meatmeal
## 55    263 meatmeal
## 56    242 meatmeal
## 57    206 meatmeal
## 58    344 meatmeal
## 59    258 meatmeal
```

```r
# d.
subset(chickwts, feed == "casein" | feed == "horsebean", weight)
```

```
##    weight
## 1     179
## 2     160
## 3     136
## 4     227
## 5     217
## 6     168
## 7     108
## 8     124
## 9     143
## 10    140
## 60    368
## 61    390
## 62    379
## 63    260
## 64    404
## 65    318
## 66    352
## 67    359
## 68    216
## 69    222
## 70    283
## 71    332
```

```r
# or
subset(chickwts, feed %in% c("casein", "horsebean"), weight)
```

```
##    weight
## 1     179
## 2     160
## 3     136
## 4     227
## 5     217
## 6     168
## 7     108
## 8     124
## 9     143
## 10    140
## 60    368
## 61    390
## 62    379
## 63    260
## 64    404
## 65    318
## 66    352
## 67    359
## 68    216
## 69    222
## 70    283
## 71    332
```

```
# e.
aggregate(weight ~ feed, chickwts, mean)
```

```
##        feed   weight
## 1    casein 323.5833
## 2 horsebean 160.2000
## 3   linseed 218.7500
## 4  meatmeal 276.9091
## 5   soybean 246.4286
## 6 sunflower 328.9167
```

6. Use dataset `chickwts` again.
   a. Compute the mean weight of all chickens (Hint: function `mean()`).
   b. Use `weight` to generate a categorical variable `weight_cat` with the three categories `light`, `medium`, and `heavy`. All categories should approximately contain the same number of chickens (Hint: `cut()`).)
   c. Subsequently, combine the `light` and `medium` categories to a new category `standard`, and rename the `heavy` category to `premium`.
   d. How many "premium" chickens are there that have been fed with **meatmeal**? (Do not count manually, but use R functions!)

```
# a.
mean(chickwts$weight)
```

```
## [1] 261.3099
```

```
# b.
nrow(chickwts) # approx. 24 should be in each group
```

```
## [1] 71
```

```
idx <- round(nrow(chickwts) * c(1/3, 2/3)) # divide n by 3 to get
# the indices of the breaks
breaks <- sort(chickwts$weight)[idx] # sort the chickens by weight
# and get the breaks
weight_cat <- cut(chickwts$weight, c(-Inf, breaks, Inf),
                  labels = c("light", "medium", "heavy"))
table(weight_cat) # show frequencies of the categories
```

```
## weight_cat
##  light medium  heavy
##     24     23     24
```

```
# c.
levels(weight_cat)[1:2]
```

```
## [1] "light"  "medium"
```

```
levels(weight_cat)[1:2] <- "standard"
levels(weight_cat)[2] <- "premium"
```

```r
table(weight_cat)
```

```
## weight_cat
## standard  premium
##       47       24
```

```r
chickwts.prem <- chickwts[weight_cat == "premium", ]
summary(chickwts.prem$feed) # -> 4 chickens
```

```
##    casein horsebean   linseed  meatmeal   soybean sunflower
##         8         0         0         4         3         9
```

```r
# or
chickwts.prem <- subset(chickwts,
                        weight_cat == "premium" & feed == "meatmeal")
nrow(chickwts.prem)
```

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

7. A digitized black and white image can be represented by a binary matrix, with "0" symbolizing color value "white" and "1" color value "black".
   a. Create an 800x600 matrix and randomly fill it with black and white values.
   b. Compute the grey value (= mean color value) of the whole matrix.
   c. Compute the grey value for a certain image section, namely the lower right quadrant of the picture (Hint: Use horizontal and vertical splitting to get the four equally sized image sections).
   d. Invert the picture to get a negative (0 becomes 1 and vice versa). Compute the grey value again.

```r
# a.
X <- matrix(sample(0:1, 800 * 600, replace = TRUE), nrow = 800)
dim(X)
```

```
## [1] 800 600
```

```r
table(X)
```

```
## X
##      0      1
## 239273 240727
```

```r
# b.
mean(X)
```

```
## [1] 0.5015146
```

```r
# c.
row.idx <- (nrow(X) / 2 + 1):nrow(X)
col.idx <- (ncol(X) / 2 + 1):ncol(X)
X_lr <- X[row.idx, col.idx]
mean(X_lr)
```

```
## [1] 0.5017833
```

```r
# d.
X_i <- -1 * X + 1
mean(X_i)
```

```
## [1] 0.4984854
```

8. Use the following R-code to generate a matrix of temperature measurements (in °C) for the 31 daily average high temperatures of July in two cities:

```r
set.seed(1)
temps <- matrix(c(round(rnorm(31, 26, 5), 1), round(rnorm(31, 22, 4), 1)),
                nrow = 2, byrow = TRUE)
rownames(temps) <- c("City A", "City B")
colnames(temps) <- paste("July", 1:31)
```

Unfortunately, many of the entries have been lost when reading the data…:

```r
temps[sample(1:length(temps), 20)] <- NA
```

a. Compute mean, minimum, and maximum for all entries and separately for both cities.
b. Compute the maximum temperature for the second half of the month (starting with the 16^th of July)
c. Save the observational units in a list having two components (one for each city). The list should only contain the actual measurements and reference dates (i.e., without missings).

```r
# a.
mean(temps, na.rm = TRUE)
```

```
## [1] 24.5881
```

```r
min(temps, na.rm = TRUE)
```

```
## [1] 14.9
```

```r
max(temps, na.rm = TRUE)
```

```
## [1] 34
```

```r
mean(temps[1, ], na.rm = TRUE)
```

```
## [1] 26.26667
```

```r
mean(temps[2, ], na.rm = TRUE)
```

```
## [1] 22.35
```

```r
min(temps[1, ], na.rm = TRUE)
```

```
## [1] 14.9
```

```r
min(temps[2, ], na.rm = TRUE)
```

```
## [1] 16.5
```

```r
max(temps[1, ], na.rm = TRUE)
```

```
## [1] 34
```

```r
max(temps[2, ], na.rm = TRUE)
```

```
## [1] 29.9
```

```r
# b.
max(temps[, 16:ncol(temps)], na.rm = TRUE)
```

```
## [1] 32.8
```

```r
# c.
temps.list <- list("City A" = na.omit(temps[1, ]),
                   "City B" = na.omit(temps[2, ]))
temps.list
```

```
## $`City A`
##  July 2  July 3  July 4  July 5  July 6  July 7  July 9 July 10 July 11 July 12
##    26.9    21.8    34.0    27.6    21.9    28.4    28.9    24.5    33.6    27.9
## July 13 July 14 July 16 July 18 July 19 July 21 July 23 July 24 July 25 July 27
##    22.9    14.9    25.8    30.7    30.1    30.6    26.4    16.1    29.1    25.2
## July 28 July 29 July 30 July 31
##    18.6    23.6    28.1    32.8
## attr(,"na.action")
##  July 1  July 8 July 15 July 17 July 20 July 22 July 26
##       1       8      15      17      20      22      26
## attr(,"class")
## [1] "omit"
##
## $`City B`
##  July 1  July 2  July 3  July 4  July 7  July 8  July 9 July 10 July 15 July 16
##    21.6    23.6    21.8    16.5    21.8    26.4    25.1    21.3    19.2    23.5
## July 18 July 19 July 22 July 23 July 25 July 27 July 28 July 29
##    21.6    25.5    23.4    17.5    29.9    17.8    24.3    21.5
## attr(,"na.action")
##  July 5  July 6 July 11 July 12 July 13 July 14 July 17 July 20 July 21 July 24
##       5       6      11      12      13      14      17      20      21      24
## July 26 July 30 July 31
##      26      30      31
## attr(,"class")
## [1] "omit"
```