SCR week 2: homework exercises

The best way to master the `R` basics is to code yourself and `rep("practice", Inf)`. A good strategy to work on these exercises is to do them together with your colleagues, and discuss your (possibly) different strategies and solutions to the exercises. A few of the exercises are mandatory: they talk about some new things we've not covered in class, but which you need to know.

The other part is optional: these provide you with more material to practice your skills. These are of course highly recommended.

1. Packages, data, and more

a.

Install and load the package `nycflights13`. Take a look at the `nycflights13` package and `flights` object documentation using `help(package = 'nycflights13')` and take a look at the `flights` object, using `data(flights)`. Note: the `flights` data, is in a `tibble` format: a class similar to data.frame (something that differs is the way the data frame is `print`ed for example). It is enough to know you can interact with a `tibble` in mostly the same way as a data.frame. Try some things you know can do with data.frames on the `flights` object, to see this for yourself.

b.

Having read the documentation, explore the data (the `flights` object) by using functions like `class`, `dim`, `head`, `str` and `summary`.

c.

Let's do some basic filtering. Create logical vectors such that we can find all flights:

- to SFO or OAK
- delayed by more than an hour
- that departed between midnight (including midnight) and 5.00 am
- for which the arrival delay was more than twice the departure delay

d.

Combine the logical vectors to see if there are any flights for which *all* of the above are true. Are there any?

2. Creating variables and `seed`

First run the following code to create a data frame `data.set` with no variables:

```r
library("nycflights13")
data(flights)

set.seed(20161013)

N <- 1e3
indx.flights <- sample(1:nrow(flights), N)
data.set <- data.frame(row.names = 1:N)
```

Now, we would like to add the following variables.

- `x1`: numeric vector with components $N/2, N/2-1, N/2-2, \ldots, 2, 1, 1, 2 \ldots, N/2-2, N/2-1, N/2$.
- `x2`: a logical vector of lenght `N`. Extract the carrier information out of the `flights` data for the rownumbers that can be found in `indx.flights`. In the logical vector give each element that has 'US','UA' or 'AA' in the corresponding position in the carrier information vector a `TRUE`, all others `FALSE`. *HINT: using the binary operator* `%in%` *may be convenient for you*
- `x3`: a 'bad weather' indication variable. Draw N observations from a standard normal distribution, then square and round the results to the nearest integer (use the functions `rnorm` and `round`).
- `e`: a vector of N elements that come from a standard normal i.i.d.
- `y`: our own synthetic outcome variable representing arrival delay: $y_i = 2+0.1*x_{i2}+2*x_{i3}+1*x_{i2}*x_{i3}+e_i$.

b.

Write the data to a file called `my_fake_data.csv` in the `0_data` folder.

c.

Share you file with a fellow student and have them share their file with yours (e.g. send it via e-mail). Are they different? If so, how? Are they the same in surprising places? How about all the random numbers, are they different?

Answer: If you run the exact same code, given that we set a seed at the start of this exercise, you should get the exact same random numbers. However, if you've samped some random values in between final results, the first results might be the same, but the latter won't be.

Outro

We'll talk more about seeds (and `set.seed`) in a later lecture. For now it is enough to realise that setting a seed, means putting the random number generator of the computer into a particular state: you make sure two people get the same random numbers, by making sure they set the state of the generator in the same way, using `set.seed`.

3. Working with vectors

Given two vectors:

```r
x <- c(5, 2, 10, 4)
y <- c(3, 6, 3, 10)
```

&& and || only ever return a single (scalar, length-1 vector) TRUE or FALSE value, whereas | and & return a vector after doing element-by-element comparisons.

Using one (or some) of the operators `&`, `&&`, `|`, `||`, `>`, `any(), all()`, and write `R` code to test if:

a. elements in vector `x` are greater than elements in vector `y`;

b. elements in vectors `x` AND `y` are greater than 3;

c. elements in vectors `x` OR `y` are greater than 3;

d. all elements of vector `x` AND all elements of vector `y` are greater than 2;

e. all elements of vector `x` OR all elements of vector `y` are greater than 2;

f. there are any elements in vectors `x` or `y` that are greater than 9;

g. the first element of vector `x` AND the first element of vector `y` are greater than 2

4. `paste` and vector recycling

a.

Explore the helpfile and examples of the function `paste` if you are not comfortable with the paste function yet.

b.

Create the vector "varname" which has the following elements:

```
"A_1"  "A_2"  "A_3"  "A_4"  "A_5"  "A_6"  "A_7"  "A_8"  "A_9"  "A_10"
```

c.

Remember vector recycling? Use the function `paste` and vector recycling for `c("ODD", "EVEN")` to create a vector that contains the following elements:

```
"1 = ODD" "2 = EVEN" "3 = ODD" "4 = EVEN" "5 = ODD" "6 = EVEN"
```

5. Enter the `matrix`

a.

Work through the extended example in 3.2.3 (from p. 63) about an image of Mount Rushmore. You can find the image file (`mtrush1.pgm`) in the `0_data` folder. You can find the function blurpart below (this is the corrected version of the downloaded code from the book).

```r
blurpart <- function(img, rows, cols, q) {
  lrows <- length(rows)
  lcols <- length(cols)
  newimg <- img
  randomnoise <- matrix(nrow = lrows, ncol = lcols, runif(lrows * lcols))
  newimg@grey[rows, cols] <- (1 - q) * img@grey[rows, cols] + q * randomnoise
  return(newimg)
}
```

Plot the variable (= objects) `mtrush1`, `mtrush2`, and `mtrush3` using e.g. `plot(mtrush1)`.

b.

Create object `mtrush4` using a different value of `q`. What happens if `q` is close to `0`? and if `q` is close to `1`?

c.

Now we want to keep President Roosevelt, but disguise the person on the left of the figure (Hint: the index of the rows equals 25:86; the index of the columns starts at 15).

Construct a function that works similarly to blurpart, but instead of replacing the pixels with random noise, changes the pixels so that the indicated pixels are really blurred.

Hint: in a blurred image, we lose all the 'sharp' edges, or that we lose contrast. Contrast is given by very light, or very dark pixels, a method of blurring might therefore be to 'pull' pixels to the average pixel intensity, and the more intense ones (towards light or dark) more heavily).

Construct the object (using the function blurpart) and plot it.

6. `set.seed` and coding mini simulation studies

This is an exercise about generating random normally distributed vectors and applying functions to rows and columns of a matrix.

Let matrix $\mathbf{X}$ be a matrix of 4 columns with normally distributed random variables with $\mu = 1$, $\sigma^2 = 3$, and number of N observations that is equal to 5000. Below, we show four ways to generate $\mathbf{X}$, the generated matrices are denoted with `X1`, `X2`, `X3` and `X4`, respectively.

```r
set.seed(123)
X1 <- cbind(
  rnorm(n = 5000, mean = 1, sd = sqrt(3)),
  rnorm(n = 5000, mean = 1, sd = sqrt(3)),
  rnorm(n = 5000, mean = 1, sd = sqrt(3)),
  rnorm(n = 5000, mean = 1, sd = sqrt(3))
)
```

```r
set.seed(123)
X2 <- matrix(
  rep(rnorm(n = 5000, mean = 1, sd = sqrt(3)), times = 4),
  nrow = 5000, ncol = 4
)
```

```r
set.seed(123)
nsamples <- 4
X3 <- matrix(
  rnorm(n = 5000 * nsamples, mean = 1, sd = sqrt(3)),
  nrow = 5000
)
```

```r
set.seed(123)
nsamples <- 4
X4 <- replicate(nsamples, {
  rnorm(n = 5000, mean = 1, sd = sqrt(3))
})
```

a.

Compare the column means and variances of the four matrices. Which two matrices have the same solution? Which matrix is not a matrix with four random variables? How come? Which coding example do you prefer and why?

b.

From matrix `X3`, create a new matrix `X3_new` that contains only the rows of `X3` for which at least 2 out of the 4 values are greater than 1. Give the dimensions of `X3_new`. Think back on logical indexing and filtering!

c.

Create a new matrix `X5` that is based on `X3`, where each column of `X3` is standardized (i.e. column mean equals 0 and standard deviation equals 1).

std <- apply(X3, 2, sd)

8. Matrices of `character` and `numeric` mode

Suppose we have the following matrix:

```
mat_values <- c(
  rep(2, 3),
  rep(3, 2),
  2, 3:1,
  rep(0, 3)
)
mat <- matrix(mat_values, nrow = 4
)
mat
```

```
##      [,1] [,2] [,3]
## [1,]    2    3    1
## [2,]    2    2    0
## [3,]    2    3    0
## [4,]    3    2    0
```

and the following character vector:

```
strg <- c("A" ,"C", "G", "T")
strg
```

```
## [1] "A" "C" "G" "T"
```

The values in matrix mat correspond to the characters of the vector 'strg' in the following way: the value 0 corresponds to A (Adenine), the value 1 corresponds to C (Cytosine), the value 2 corresponds to G (Guanine), and the value 3 corresponds to T (Thymine).

a.

Write a piece of `R` code that converts each value of `mat` into the corresponding character of 'strg'. Check the mode of the matrix.

b.

Write a piece of `R` code to perform the following operation: Concatenate (without spaces in between) A, C, G, T according to the values in each row of mat, in such a way that the output is the following:

[1] "GTC" "GGA" "GTA" "TGA"

c.

Write a function that concatenates A, C, G, I according to the values in the row of a general input matrix `mat` having elements in (0; 1; 2; 3). Check with an example if your function performs well.

9. The elements of a `list` object

Read example 4.2.4 (from p. 90) and try to understand the function `findwords` (which is given below). Perform the function step by step, using the text file `testconcorda.txt` (from the `0_data` folder). Finally, create an object `wl`, using the function `findwords` with `testconcorda.txt` as input. Inspect the class of `wl` and ask for the component `that`.

```
findwords <- function(tf) {
   # read in the words from the file, into a vector of mode character
   txt <- scan(tf, "")
   wl <- list()
   for (i in 1:length(txt)) {
      wrd <- txt[i]  # i-th word in input file
      wl[[wrd]] <- c(wl[[wrd]], i)
   }
   return(wl)
}
```

10. pmax and play with logicals

We have the following data frame:

```
set.seed(2009)
w <- runif(10)
x <- runif(10)
y <- runif(10)
z <- runif(10)
DF <- data.frame(a = w, b = x, c = y, d = z)
```

We define two intervals using the four columns of the data frame, namely we define the intervals $[\min(a, b), \max(a, b)]$, and $[\min(c, d), \max(c, d)]$. Add a new logical column in the data frame, which should be `TRUE` if the intervals overlap, and `FALSE` otherwise. The output should look like:

```
head(DF)
```

```
##             a          b         c          d overlap
## 1 0.197260832 0.03232136 0.5722249 0.05370368    TRUE
## 2 0.696829870 0.25971113 0.5922310 0.10296065    TRUE
## 3 0.607896252 0.57589595 0.8583711 0.90978420   FALSE
```

```
## 4 0.009547638 0.82870195 0.4836649 0.82281090    TRUE
## 5 0.429010613 0.67047141 0.4416763 0.74668683    TRUE
## 6 0.076557244 0.57599446 0.1430793 0.49561776    TRUE
```

Use logical operators and/or if you prefer shorter (and perhaps more readable) code, use the functions `pmin` and `pmax`.

11. Towards Programming: Importing Data from Excel using `readxl`

In the upper right pane of RStudio, there is a button called `Import Dataset`:

Figure 1: The option in RStudio you may want to explore.

Instead of you using code on the command line in your console (left-bottom panel), you can explore on how to import your data using this Graphical User Interface (GUI) in the RStudio editor.

a.

Use the above described GUI to produce the code with which you can import the sheet `NZA 2018` from the `Tarievenoverzicht.xlsx` file (see the 0_data folder). The `Tarievenoverzicht.xlsx` contains a pricelist of Health Insurances and the Nederlandse Zorg Autoriteit (NZA = Dutch Healthcare Agency) for specialized treatments in the area of Mental Health Care.

b.

If all went well with importing the `NZA 2018` data, you could see that you alos ran `library(readxl)`. An `R` package especially designed for importing data from Excel files.

In this package there is also a function that lists all sheets in the excel spreadsheet. One of the ways to find this function is to explore the helpfile of the package, `help(package = "readxl")`. Can you show with the correct `R` code the names of the sheets in `Tarievenoverzicht.xlsx`.

c.

Suppose we are only interested in importing the data of the `NZA` sheets. Use a combination of the functions `lapply()`, `grep`, and the answers of three previous sub-exercises, to import only the `NZA` sheets automatically with an `lapply` loop. Store the results in a variable `NZA_list`

d.

Repeat the previous exercise, but now make sure that the class of each data set in `NZA_list` is set equal to `data.frame` only, and let each data set consist of five variables:

```r
NZA_varnames <- c("code", "Productgroup", "tariff", "maxtariff")
```

You may use the `NZA_varnames`. Note that not all of the data sets have the fifth `maxtariff` variable. You'll need a strategy here to add or create this extra fifth variable yourself, within e.g. a customized function for `lapply()`.

It may be helpfull to first create your own custmized function, e.g. `ImportOneSheetNZA`, that performs the needed operations when you give it the address of `Tarievenoverzicht` and a sheet name as input parameters. To stay with the only already used functions in class and previous exercises, our customized function of th emodel answers consists of the functions `read_excel()`, `data.frame()`.

Hint: Instead of using `if`* statements or the *`ifelse()`* function, we just always created an extra column, but in the end only selected the columns that where consistent with* `NZA_varnames`.

Show that you managed, by neatly printing the new variable names of each data set.