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.

Answer:

```
library("nycflights13")
```

```
## Warning: package 'nycflights13' was built under R version 3.4.4
```

```
data(flights)
head(flights[["year"]])
```

```
## [1] 2013 2013 2013 2013 2013 2013
```

```
head(flights)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013     1     1      517            515         2      830
## 2  2013     1     1      533            529         4      850
## 3  2013     1     1      542            540         2      923
## 4  2013     1     1      544            545        -1     1004
## 5  2013     1     1      554            600        -6      812
## 6  2013     1     1      554            558        -4      740
## # ... with 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>
```

b.

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

Answer:

```r
class(flights)
```

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

```r
dim(flights)
```

```
## [1] 336776     19
```

```r
str(flights)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    336776 obs. of  19 variables:
##  $ year          : int  2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
##  $ month         : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ day           : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ dep_time      : int  517 533 542 544 554 554 555 557 557 558 ...
##  $ sched_dep_time: int  515 529 540 545 600 558 600 600 600 600 ...
##  $ dep_delay     : num  2 4 2 -1 -6 -4 -5 -3 -3 -2 ...
##  $ arr_time      : int  830 850 923 1004 812 740 913 709 838 753 ...
##  $ sched_arr_time: int  819 830 850 1022 837 728 854 723 846 745 ...
##  $ arr_delay     : num  11 20 33 -18 -25 12 19 -14 -8 8 ...
##  $ carrier       : chr  "UA" "UA" "AA" "B6" ...
##  $ flight        : int  1545 1714 1141 725 461 1696 507 5708 79 301 ...
##  $ tailnum       : chr  "N14228" "N24211" "N619AA" "N804JB" ...
##  $ origin        : chr  "EWR" "LGA" "JFK" "JFK" ...
##  $ dest          : chr  "IAH" "IAH" "MIA" "BQN" ...
##  $ air_time      : num  227 227 160 183 116 150 158 53 140 138 ...
##  $ distance      : num  1400 1416 1089 1576 762 ...
##  $ hour          : num  5 5 5 5 6 5 6 6 6 6 ...
##  $ minute        : num  15 29 40 45 0 58 0 0 0 0 ...
##  $ time_hour     : POSIXct, format: "2013-01-01 05:00:00" "2013-01-01 05:00:00" ...
```

```r
summary(flights)
```

```
##       year          month             day           dep_time
##  Min.   :2013   Min.   : 1.000   Min.   : 1.00   Min.   :   1
##  1st Qu.:2013   1st Qu.: 4.000   1st Qu.: 8.00   1st Qu.: 907
##  Median :2013   Median : 7.000   Median :16.00   Median :1401
##  Mean   :2013   Mean   : 6.549   Mean   :15.71   Mean   :1349
##  3rd Qu.:2013   3rd Qu.:10.000   3rd Qu.:23.00   3rd Qu.:1744
##  Max.   :2013   Max.   :12.000   Max.   :31.00   Max.   :2400
##                                                  NA's   :8255
##  sched_dep_time   dep_delay          arr_time    sched_arr_time
##  Min.   : 106   Min.   : -43.00   Min.   :   1   Min.   :   1
##  1st Qu.: 906   1st Qu.:  -5.00   1st Qu.:1104   1st Qu.:1124
##  Median :1359   Median :  -2.00   Median :1535   Median :1556
##  Mean   :1344   Mean   :  12.64   Mean   :1502   Mean   :1536
##  3rd Qu.:1729   3rd Qu.:  11.00   3rd Qu.:1940   3rd Qu.:1945
##  Max.   :2359   Max.   :1301.00   Max.   :2400   Max.   :2359
##                 NA's   :8255      NA's   :8713
##    arr_delay          carrier              flight       tailnum
##  Min.   : -86.000   Length:336776      Min.   :   1   Length:336776
##  1st Qu.: -17.000   Class :character   1st Qu.: 553   Class :character
##  Median :  -5.000   Mode  :character   Median :1496   Mode  :character
##  Mean   :   6.895                      Mean   :1972
##  3rd Qu.:  14.000                      3rd Qu.:3465
##  Max.   :1272.000                      Max.   :8500
##  NA's   :9430
##     origin              dest              air_time        distance   
##  Length:336776      Length:336776      Min.   : 20.0   Min.   :  17  
##  Class :character   Class :character   1st Qu.: 82.0   1st Qu.: 502  
##  Mode  :character   Mode  :character   Median :129.0   Median : 872  
##                                        Mean   :150.7   Mean   :1040  
##                                        3rd Qu.:192.0   3rd Qu.:1389  
##                                        Max.   :695.0   Max.   :4983  
##                                        NA's   :9430                  
##       hour           minute        time_hour                  
##  Min.   : 1.00   Min.   : 0.00   Min.   :2013-01-01 05:00:00  
##  1st Qu.: 9.00   1st Qu.: 8.00   1st Qu.:2013-04-04 13:00:00  
##  Median :13.00   Median :29.00   Median :2013-07-03 10:00:00  
##  Mean   :13.18   Mean   :26.23   Mean   :2013-07-03 05:22:54  
##  3rd Qu.:17.00   3rd Qu.:44.00   3rd Qu.:2013-10-01 07:00:00  
##  Max.   :23.00   Max.   :59.00   Max.   :2013-12-31 23:00:00  
##
```

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

Answer:

```
names(flights)
```

```
##  [1] "year"           "month"          "day"            "dep_time"      
##  [5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_time"
##  [9] "arr_delay"      "carrier"        "flight"         "tailnum"       
## [13] "origin"         "dest"           "air_time"       "distance"      
## [17] "hour"           "minute"         "time_hour"
```

```
sfooak_filter <- flights$dest%in%c("SFO", "OAK")
arrivaldelay_filter <- flights$arr_delay > 1
departure_filter <- (flights$dep_time > 0) & (flights$dep_time < 500)
twicemore_filter <- flights$arr_delay > (2 * flights$dep_delay)
```

d.

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

```
all_filter <- (sfooak_filter * arrivaldelay_filter * departure_filter * twicemore_filter)
all_filter_nas <- which(is.na(all_filter))
sum(all_filter[-all_filter_nas]) # nope!
```

Deal with NaN values

```
## [1] 0
```

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: $y_i = 2 + 0.1 * x_{i2} + 2 * x_{i3} + 1 * x_{i2} * x_{i3} + e_i$.

Answer:

```r
N <- 1e3
x1 <- c((N/2):1, 1:(N/2))
reduced_flights_carrier <- flights$carrier[indx.flights]
x2 <- reduced_flights_carrier %in% c("US", "UA", "AA")
x3 <- round(rnorm(N)^2)
e <- rnorm(N)
y <- 2 + 0.1*x2+2*x3+1*x2*x3

data.set <- cbind(data.set, x1, x2, x3, e, y)
```

b.

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

Answer:

```r
write.csv(data.set, "0_data/my_fake_data.csv")
```

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:

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

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

** Answer:**

```
#a:
any(x > y)
```

```
## [1] TRUE
```

```
#b:
any(x > 3 & y > 3)
```

```
## [1] TRUE
```

```
#c:
any(x > 3 | y > 3)
```

```
## [1] TRUE
```

```
#d:
all(x > 2 & y > 2)
```

```
## [1] FALSE
```

```
#e:
all(x > 2) || all(y > 2)
```

```
## [1] TRUE
```

```
#f:
any(x > 9 | y > 9)
```

```
## [1] TRUE
```

```
#g:
x[1] > 2 && y[1] > 2
```

```
## [1] TRUE
```

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.

Answer:

```
?paste
```

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"
```

Answer:

```
paste("A", 1:10, sep="_")
```

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

Answer:

```
paste(1:6, c("ODD", "EVEN"), sep=" = ")
```

```
## [1] "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).

```
blurpart <- function(img, rows, cols, q) {
  lrows <- length(rows)
  lcols <- length(cols)
  newimg <- img
  randomnoise <- matrix(nrow = lrows, ncol = lcols, runif(lrows * lcols))
```

```r
  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)`.

Answer:

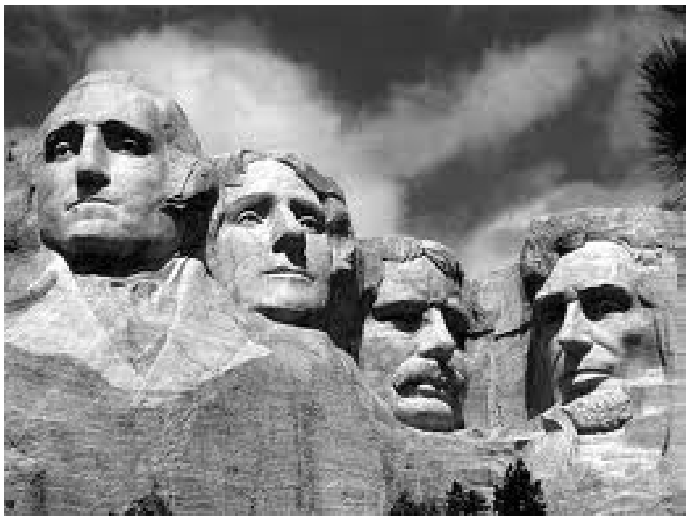
```r
if (!require(pixmap)){
install.packages("pixmap")
library(pixmap)
}
```

```
## Loading required package: pixmap
```

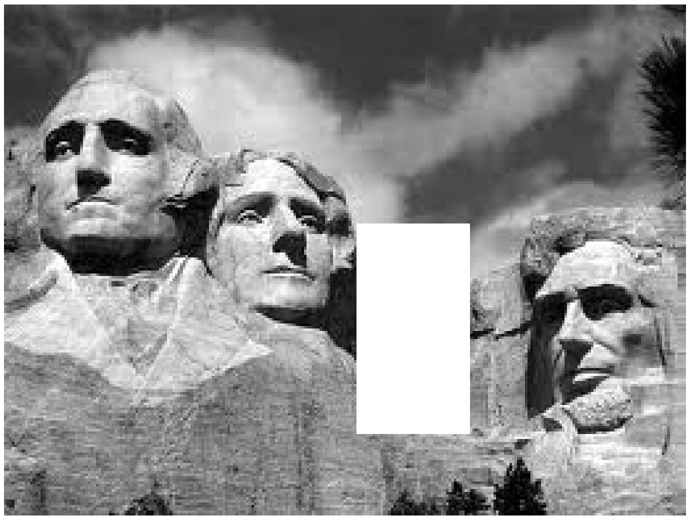
```r
mtrush1 <- read.pnm("0_data/mtrush1.pgm")
```

```
## Warning in rep(cellres, length = 2): 'x' is NULL so the result will be NULL
```

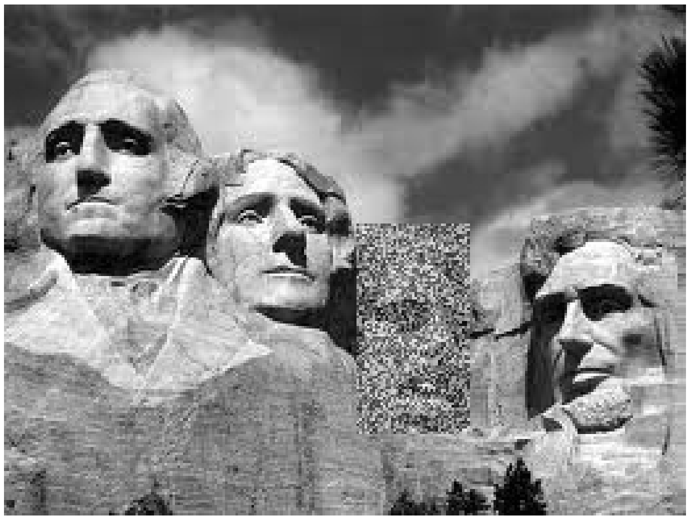
```r
plot(mtrush1)
```



```r
mtrush2 <- mtrush1
rowindex <- 84:163
colindex <- 135:177
mtrush2@grey[rowindex, colindex] <- 1
plot(mtrush2)
```



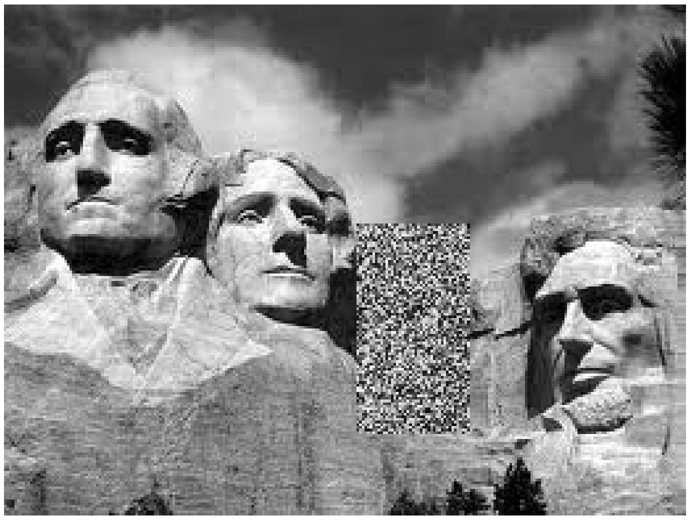
```r
mtrush3 <- blurpart(mtrush1, rowindex, colindex, 0.65)
plot(mtrush3)
```



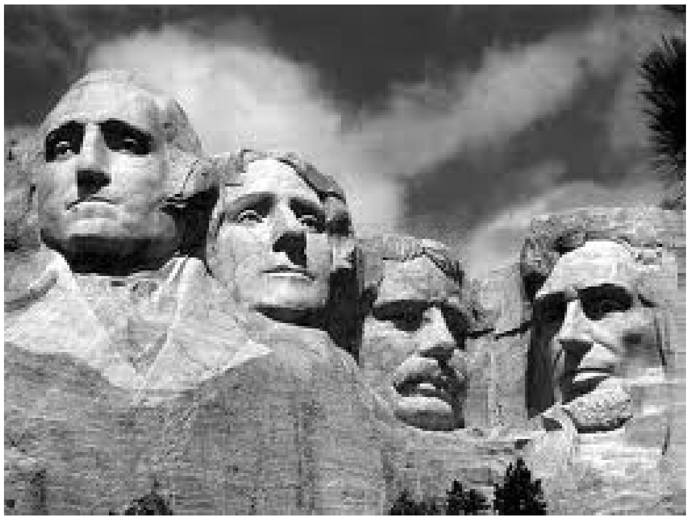
b.

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

```r
mtrush4a <- blurpart(mtrush1, rowindex, colindex, 0.90)
mtrush4b <- blurpart(mtrush1, rowindex, colindex, 0.10)
plot(mtrush4a) # high value of q is much random noise
```



```r
plot(mtrush4b) # low value of q is little random noise
```



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).

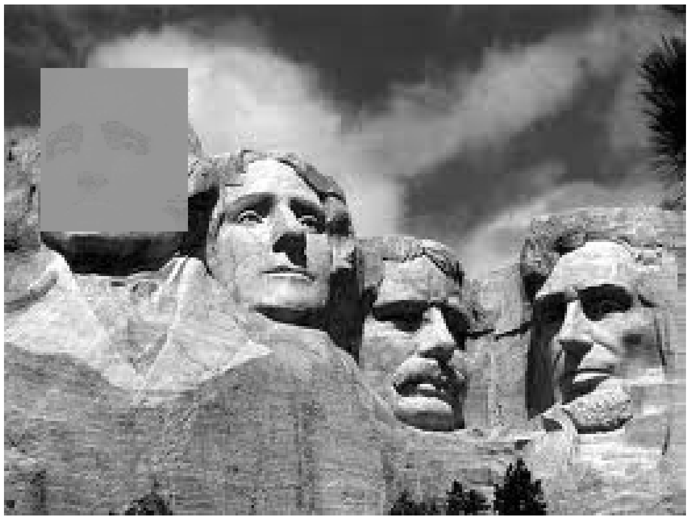
Construct a function that works similarly to blurpart, but instead of replacing the pixels with random noise, changes the pixels so that the indicated pixels are really blurred.

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.

Answer:

```r
rowindex <- 25:86
colindex <- 15:70

blurpart2 <- function(img, rows, cols, q) {
  lrows <- length(rows)
  lcols <- length(cols)
  newimg <- img
  # randomnoise <- matrix(nrow = lrows, ncol = lcols, runif(lrows * lcols))

  average.intensity <- mean(img@grey[rows, cols])
  difference <- img@grey[rows, cols] - average.intensity

  newimg@grey[rows, cols] <- average.intensity + sign(difference)*abs(difference)^(1/(1-q))

  return(newimg)
}

mtrush5 <- blurpart2(mtrush1, rowindex, colindex, 0.8)
plot(mtrush5)
```



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?

Answer:

```r
colMeans(X1)
```

```
## [1] 0.9990134 0.9927707 1.0141524 0.9543020
```

```r
colMeans(X2)
```

```
## [1] 0.9990134 0.9990134 0.9990134 0.9990134
```

```r
colMeans(X3)
```

```
## [1] 0.9990134 0.9927707 1.0141524 0.9543020
```

```r
colMeans(X4)
```

```
## [1] 0.9990134 0.9927707 1.0141524 0.9543020
```

```r
apply(X1, 2, var)
```

```
## [1] 2.967514 3.016716 3.005475 3.012159
```

```r
apply(X2, 2, var)
```

```
## [1] 2.967514 2.967514 2.967514 2.967514
```

```r
apply(X3, 2, var)
```

```
## [1] 2.967514 3.016716 3.005475 3.012159
```

```r
apply(X4, 2, var)
```

```
## [1] 2.967514 3.016716 3.005475 3.012159
```

`X2` consists of 4 times the same vector of N i.i.d sampled normal distribution values.

We, the instructors, would prefer readable code like:

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

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!

Answer:

```r
index <- rowSums(X3 > 1) >= 2
X3_new <- X3[index, ]
dim(X3_new)
```

```
## [1] 3375    4
```

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).

Answer:

```r
Standardize <- function(x){
  (x - mean(x)) / sd(x)
}
X5 <- apply(X3, 2, Standardize)
apply(X5, 2, mean)
```

```
## [1] -2.099183e-18 -3.464867e-17 -3.509051e-17 -3.636791e-17
```

```r
apply(X5, 2, var)
```

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

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

Suppose we have the following matrix:

```r
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.

Answer: First we try out a solution for the first column of mat:

```
x <- mat[, 1]
strg[x + 1]
```

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

Now we use this solution to apply it to all columns of mat:

```
matnew <- apply(mat, 2, function(x){
  strg[x + 1]
  })
matnew
```

```
##      [,1] [,2] [,3]
## [1,] "G"  "T"  "C"
## [2,] "G"  "G"  "A"
## [3,] "G"  "T"  "A"
## [4,] "T"  "G"  "A"
```

```
mode(matnew)
```

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

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"

Answer: First we try out a solution for the first row of mat:

```
x <- mat[1, ]
strg[x + 1]
```

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

```
paste(strg[x + 1], collapse = "")
```

```
## [1] "GTC"
```

Now we use this solution to apply it to all rows of mat:

```r
out <- apply(mat, 1, function(x){
  paste(strg[x + 1], collapse = "")
  })
out
```

```
## [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.

Answer:

```r
convert <- function(mat){
  strg <- c("A" ,"C", "G", "I")
  out <- apply(mat, 1, function(x){
    paste(strg[x + 1], collapse = "")
    })
  return(out)
  }
mat2 <- matrix(c(0:3, 3:0), nrow = 2, byrow = TRUE)
mat2
```

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

```r
convert(mat2)
```

```
## [1] "ACGI" "IGCA"
```

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`.

```r
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)
}
```

Answer:

```r
txt <- scan("0_data/testconcorda.txt", "")
txt
```

```
##  [1] "the"         "here"        "means"       "that"        "the"
##  [6] "first"       "item"        "in"          "this"        "line"
## [11] "of"          "output"      "is"          "item"        "in"
## [16] "this"        "case"        "our"         "output"      "consists"
## [21] "of"          "only"        "one"         "line"        "and"
## [26] "one"         "item"        "so"          "this"        "is"
## [31] "redundant"   "but"         "this"        "notation"    "helps"
## [36] "to"          "read"        "voluminous"  "output"      "that"
## [41] "consists"    "of"          "many"        "items"       "spread"
## [46] "over"        "many"        "lines"       "for"         "example"
## [51] "if"          "there"       "were"        "two"         "rows"
## [56] "of"          "output"      "with"        "six"         "items"
## [61] "per"         "row"         "the"         "second"      "row"
## [66] "would"       "be"          "labeled"
```

```r
wl <- list()
wl
```

```
## list()
```

```r
i <- 1
wrd <- txt[i]
wrd
```

```
## [1] "the"
```

```r
wl[[wrd]] <- c(wl[[wrd]], i)
wl[[wrd]]
```

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

```r
i <- 5
wrd <- txt[i]
wrd
```

```
## [1] "the"
```

```r
wl[[wrd]] <- c(wl[[wrd]], i)
wl[[wrd]]
```

```
## [1] 1 5
```

```r
wl <- findwords("0_data/testconcorda.txt")
wl[["that"]]
```

```
## [1]  4 40
```

This exercise may come back in a later week for you to re-program. The instructors prefer to use different code in R for `findwords()`, by using `scan()`, `table()`, `unique()`, `which()`, and `lapply()`.

10. pmax and play with logicals

We have the following data frame:

```r
set.seed(2009)
w <- runif(10)
x <- runif(10)
y <- runif(10)
```

```r
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:

```r
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`.

Answer:

Following the precedence in `R` of `&` (and) over `|` (or): there is overlap when `(a >= min(c , d)` and `a <= max(c, d))` or `(b >= min(c, d)` and `b <= max(c, d))` or `(c >= min(a, b)` and `c <= max(a, b))` or `(d >= min(a, b)` and `d <= max(a, b))`.

```r
a.higher <- (DF[, 1] >= DF[, 3] | DF[, 1] >= DF[, 4])
a.lower <- (DF[, 1] <= DF[, 3] | DF[, 1] <= DF[, 4])
a.interv <- a.higher & a.lower

b.higher <- (DF[, 2] >= DF[, 3] | DF[, 2] >= DF[, 4])
b.lower <- (DF[, 2] <= DF[, 3] | DF[, 2] <= DF[, 4])
b.interv <- b.higher & b.lower

ab.interv <- b.interv | a.interv



c.higher <- (DF[, 3] >= DF[, 1] | DF[, 3] >= DF[, 2])
c.lower <- (DF[, 3] <= DF[, 1] | DF[, 3] <= DF[, 2])
c.interv <- c.higher & c.lower

d.higher <- (DF[, 4] >= DF[, 1] | DF[, 4] >= DF[, 2])
d.lower <- (DF[, 4] <= DF[, 1] | DF[, 4] <= DF[, 2])
d.interv <- d.higher & d.lower

cd.interv <- c.interv | d.interv

overlap <- ab.interv | cd.interv
```

```r
beyond.yz    <- pmax(w,x) <= pmin(y,z) | pmin(w,x) >= pmax(y,z)
beyond.wz  <- pmax(y,z) <= pmin(w,x) | pmin(y,z) >= pmax(w,x)
overlap <- !beyond.yz | !beyond.wz
```

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.

Answer:

```
library(readxl)
Tarievenoverzicht <- read_excel("0_data/Tarievenoverzicht.xlsx", sheet = "NZA 2018")
```

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`.

Answer:

```
excel_sheets("0_data/Tarievenoverzicht.xlsx")
```

```
##  [1] "DBC CODELIJST"    "Caresco 2018"     "DItzo 2018"
##  [4] "DItzo 2017"       "Salland-ENO 2018" "Salland-ENO 2017"
##  [7] "Friesland 2018"   "Friesland 2017"   "Z&Z 2018"
## [10] "Z&Z 2017"         "ZK 2018"          "ZK 2017"
## [13] "ZK 2016"          "ZK 2015"          "ZK 2014"
## [16] "VGZ 2018"         "VGZ 2017"         "VGZ 2016"
## [19] "VGZ 2015"         "VGZ 2014"         "Menzis 2018"
## [22] "Menzis 2017"      "Menzis 2016"      "Menzis 2015"
## [25] "CZ 2018"          "CZ 2017"          "CZ 2016"
## [28] "CZ 2015"          "CZ 2014"          "CZ totaal"
## [31] "Menzis totaal"    "NZA 2018"         "NZA 2017"
## [34] "NZA 2016"         "NZA 2015"         "NZA 2014"
```

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`

Answer:

```
path_to_file <- "0_data/Tarievenoverzicht.xlsx"
names_sheets <- excel_sheets(path_to_file)
NZA_sheets <- names_sheets[grep("NZA", names_sheets)]
NZA_list <- lapply(NZA_sheets, read_excel, path = path_to_file)
```

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:

```
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.

Answer:

```
path_to_file <- "0_data/Tarievenoverzicht.xlsx"
names_sheets <- excel_sheets(path_to_file)
NZA_sheets <- names_sheets[grep("NZA", names_sheets)]

sheet <- "NZA 2014"

ImportOneSheetNZA <- function(sheet, path = path_to_file) {
  dat_tmp <- read_excel(path = path, sheet = sheet)
  dat <- data.frame(
    dat_tmp,
    extra_var = NA
  )
  names(dat) <- NZA_varnames
  out <- dat[NZA_varnames]
  return(out)
}
NZA_list <- lapply(
  X = NZA_sheets,
  FUN = ImportOneSheetNZA, path = path_to_file
)
# showing the names of each data set in NZA_list:
do.call("rbind", lapply(NZA_list, names))
```

```
##      [,1]   [,2]           [,3]     [,4]
## [1,] "code" "Productgroup" "tariff" "maxtariff"
## [2,] "code" "Productgroup" "tariff" "maxtariff"
## [3,] "code" "Productgroup" "tariff" "maxtariff"
## [4,] "code" "Productgroup" "tariff" "maxtariff"
## [5,] "code" "Productgroup" "tariff" "maxtariff"
```