Exam 1 SCR (The midterm)

R-team

October 30, 2019

1. Last Year's Grades

Each worksheet in `filename.xlsx` is a data set representing the obtained grades of last year's students on the first exam (`Exam1`), the assignment (`Assignment`), and the second exam (`Exam2`). The data does not take into account the resit assignment, and the resit exam. For privacy reasons, we show fake names and fake ULCN numbers. Moreover, we added some noise to the data.

The fours columns of each data have the following names in the following order:`First Name` (column 1), `Last Name` (column 2), `ULCN` (column 3), and `Final Grade` (column 4).

1.1 Reading and Exploring the Data (20)

1.1a

To read all the data sets (work sheets of the excel file) in one-go, we use the function `readxl::read_excel()` and a `for` loop, and the function `assign()`. The data sets are stored eventually stored in the object variables `Exam1`, `Assignment`, and `Exam2`.

```
library(readxl) # line 1
sheets <- c("Exam1", "Assignment", "Exam2") # line 2
for (sheet in sheets) { # line 3
  file_name <- "0_data/190108_SCR_Grades.xlsx" # line 4
  assign(sheet, read_excel(file_name, sheet = sheet)[, 1:4]) # line 5
}; rm(sheet)
```

```
## New names:
## * `Overall Grade` -> `Overall Grade...5`
## * `Comments T2.2b` -> `Comments T2.2b...17`
## * `Comments T2.2b` -> `Comments T2.2b...19`
## * `Overall Grade` -> `Overall Grade...48`
```

Also run this code on your own console. Could you explain in at most two sentences per line what is happening at each line of code?

** Answer: **

line 1: load the library readxl, such that we can read in the excel file
line 2: create a character vector which has the sheet names of the workbook in the excel file..
line 3: start a for loop over the sheets in the work file, for each sheet in sheets something will be done.
line 4: store the addres to the `.xlsx` file in the object `file_name`
line 5: read the specific sheet of the excel file and assign it to an object which has the same name as the specific sheet.

1.1b The same Column names?

Verify with `R` code that the column names of the data sets `Exam1`, `Assignment`, and `Exam2` are the same. Your answer code should result in a logical vector of length 1 with the value `TRUE`.

Answer

```
E1A <- all(names(Exam1) == names(Assignment))
E1E2 <- all(names(Exam1) == names(Exam2))
E1A & E1E2
```

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

or

```
all.equal(colnames(Assignment),colnames(Exam1),colnames(Exam2)) #TRUE
```

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

1.1c How many students?

The number of observations (students) are 47 for the first exam, 41 for the assignment, and 40 for the second exam.

Use "ULCN" columns of the three data sets, `Exam1`, `Assignment`, and `Exam2`, and show with code

i. that there are 49 (unique) students in these data sets?

ii. that there are 37 students that took part in both exams AND the assignment.

Answer:

i.

```
all_ULCN <- unique(c(Exam1$ULCN, Assignment$ULCN, Exam2$ULCN))
length(all_ULCN)
```

```
## [1] 49
```

49 students were involved with the course.

ii.

```
E2A <- Exam2$ULCN %in% Assignment$ULCN
E2E1 <- Exam2$ULCN %in% Exam1$ULCN
sum(E2A & E2E1)
```

```
## [1] 37
```

There were 37 students that participated in the first take of the second exam and the first take of the assignment.

1.1d

Use a `for` loop to show the structure of each of the three data sets, make sure that the output for each data set is separated with a new line `"\n"`.

Hint: Check out what happens in your console when you run get("Exam1")

Answer:

```
for(i in 1:length(sheets)){
  str(get(sheets[i]))
  cat("\n")
};rm(i)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    47 obs. of  4 variables:
##  $ First Name : chr  "Tommy" "Francina" "Jeni" "Darell" ...
##  $ Last Name  : chr  "Tejera" "Fegley" "Jameson" "Dipalma" ...
##  $ ULCN       : chr  "s1913973" "s1029448" "s2447584" "s2185592" ...
##  $ Final Grade: num  47 61 47 72 58 94 72 75 92 67 ...
##
## Classes 'tbl_df', 'tbl' and 'data.frame':    41 obs. of  4 variables:
##  $ First Name : chr  "Francina" "Jeni" "Darell" "Estell" ...
##  $ Last Name  : chr  "Fegley" "Jameson" "Dipalma" "Ertl" ...
##  $ ULCN       : chr  "s1029448" "s2447584" "s2185592" "s1927199" ...
##  $ Final Grade: num  69 91 54 80 53 51 50 74 41 71 ...
##
## Classes 'tbl_df', 'tbl' and 'data.frame':    40 obs. of  4 variables:
##  $ First Name : chr  "Francina" "Jeni" "Darell" "Estell" ...
##  $ Last Name  : chr  "Fegley" "Jameson" "Dipalma" "Ertl" ...
##  $ ULCN       : chr  "s1029448" "s2447584" "s2185592" "s1927199" ...
##  $ Final Grade: num  100 56 50 60 79 49 67 59 100 88 ...
```

1.1e Checking out Percentages

A student has passed the course for sure when 55 or more points were obtained out of each exam and assignment. Around 74.5% of the students have obtained 55 or higher on the first exam, and around 75.6% obtained 55 or higher on the Assignment, and around 87.5% obtained 55 or higher on the second exam.

Could you reproduce these percentages in R? (There is no need for any kind of loop here, you can just write three lines of code)

Answer:

```
100 * mean(Exam1$`Final Grade` >= 55)
```

```
## [1] 74.46809
```

```
100 * mean(Assignment$`Final Grade` >= 55)
```

```
## [1] 75.60976
```

```
100 * mean(Exam2$`Final Grade` >= 55)
```

```
## [1] 87.5
```

1.2 Merging data sets (10)

Without merging the data sets into one, it is less obvious to check how many students eventually scored 55 (or higher) on the exams and the assignment. Therefore, we want you to create one data set containing all the grades together. Since the `merge()` function is only made to merge two data frames, we need merge the three data frames into two steps.

1.2a

To ease the merging proces, first change the name of each `Final Grade` column of the data sets.

Do the following in R: set the `Final Grade` column of the `Exam1` data set to `"Exam1"`, and the `Final Grade` column of the `Assignment` data to `"Assignment"`, and the `Final Grade` column of the `Exam2` data to `"Exam2"`.

Answer:

```
names(Exam1)[4] <- "Exam1"
names(Assignment)[4] <- "Assignment"
names(Exam2)[4] <- "Exam2"
```

1.2b

Merge the Exam1 data and Assigment data with each other on the first name, last name, and ULCN. Ensure that all information of the merged data sets remains present in the new dat set.

Hint: if you are not familiar yet with the function `merge()`, take a look quickat its helpfile and its last examples.

Answer:

```
my_grades <- merge(
  x = Exam1,
  y = Assignment,
  all = TRUE
)
```

1.2c

Perform the merge again, but now merge your new data set with the remaining `Exam2` data set. Your final data set should be equal to the `the_grades` data set from the model answer variables. Check whether this is true by using the `all.equal()` function in R.

Answer:

```
my_grades <- merge(
  x = my_grades,
  y = Exam2,
  all = TRUE
)
all.equal(my_grades, the_grades)
```

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

1.3 Reproduce a Visualization (10)

If you did not succeed in 1.2, then use the data set `the_grades` from the model answer variables as your data set.

Can you reproduce Figure 1 which shows the visualization of the obtained grades by the students? The dashed red lines are plotted at a grade equal to 50 (indicating neccessary, but not sufficient requirements to pass the course).

Hint: Don't bother how things seem to scale out in your own produced plot

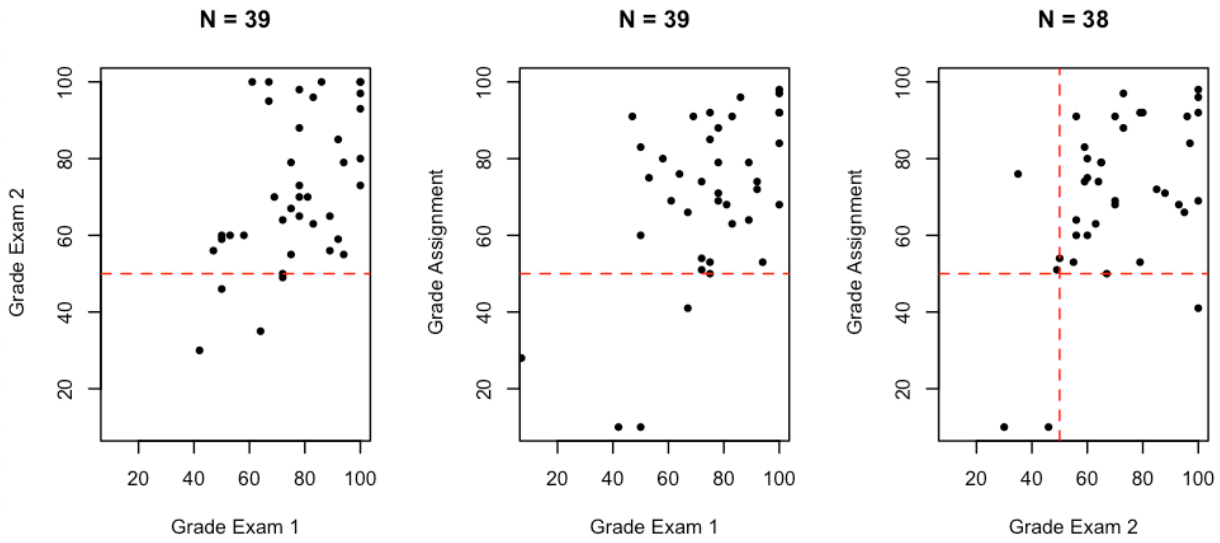


Figure 1: Scatterplots of the grades for Exam1, Exam2 and the Assignment. The red dashed line indicates the necessary (but not sufficient) minimum requirements to pass the course.

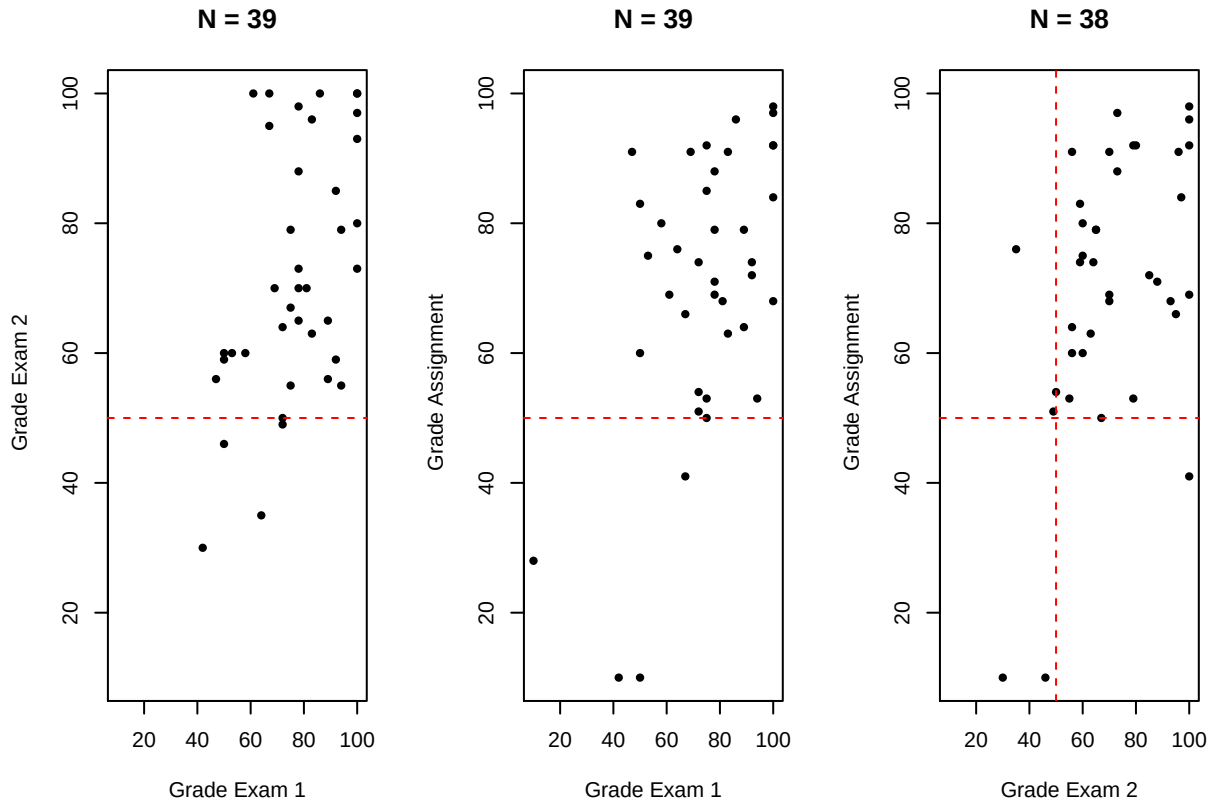
Answer:

```
# pdf("0_images/Grades.pdf", width = 480*2.5, height = 480)
par(mfrow = c(1,3))
plot(x = the_grades$Exam1, y = the_grades$Exam2,
     xlab = "Grade Exam 1", ylab = " Grade Exam 2",
     xlim = c(10, 100), ylim = c(10, 100),
     pch = 20, main = "N = 39"
)
abline(h = 50, col = "red", lty = 2)

plot(x = the_grades$Exam1, y = the_grades$Assignment,
     xlab = "Grade Exam 1", ylab = " Grade Assignment",
     xlim = c(10, 100), ylim = c(10, 100),
     pch = 20, main = "N = 39"
)
abline(h = 50, col = "red", lty = 2)

plot(x = the_grades$Exam2, y = the_grades$Assignment,
     xlab = "Grade Exam 2", ylab = " Grade Assignment",
     xlim = c(10, 100), ylim = c(10, 100),
```

```
     pch = 20, main = "N = 38"
)
abline(v = 50, col = "red", lty = 2)
abline(h = 50, col = "red", lty = 2)
```




```
#dev.off()
```

1.4 Pass / resit and the final course grade (20)

If you did not succeed in 1.2, then use the data set `the_grades` from the model answer variables as your data set.

1.4a

In the first question we've seen that there are 37 students who took part in both exams and the assignment. Could you verify this with code by using the newly merged data and by counting the number of observations that have an `NA` value for either one of the three grades?

Answer:

```
nrow(na.omit(the_grades))
```

```
## [1] 37
```

1.4b

To pass the course in one-take, a final SCR course grade of 55 (out of 100) is required. Let

E_1 := Grade Exam 1
E_2 := Grade Exam 2
$E = (E_2 + \max(E_1, E_2))/2$
A := Grade Assignment
Y := Course grade (without resits)

Then,

$$Y = \begin{cases} (2/3)E + (1/3)A, & \text{if } (E \geq 50) \cap (E_2 \geq 50) \cap (A \geq 50) \\ \min\left((2/3)E + (1/3)A,\ 50\right), & \text{otherwise.} \end{cases}$$

If either E_1, E_2, or A has a missing value (`NA`), then replace it with a 10 (out of 100).

Write a function which tells us whether the students have passed the course in one-take, or should (at least) do one resit. The function takes as input the `the_grades` data set, and gives as output a `list` that contains two entries: 1. a factor (pass/resit) of `nrow(the_grades)` elements, and 2. a numeric vector (final grade) of `nrow(the_grades)` elements. The labels (and levels) of the factor indicate whether a student has passed the course (`passed`) or needs to do at least one resit (`resit`).

Hint: don't bother about rounding the final course grade, also just implement the formula. If you don't understand why the grade is calculated this way, just ask it in the next course class.

Answer:

```
GiveCourseResults <- function(dat_grades) {

    A  <- dat_grades$Assignment
    A[is.na(A)] <- 10
    E1 <- dat_grades$Exam1
    E1[is.na(E1)] <- 10
    E2 <- dat_grades$Exam2
    E2[is.na(E2)] <- 10
    E <- (E2 + pmax(E1, E2)) / 2

    grade <- (2 / 3) * E + (1 / 3) * A
    pass <- A >= 50 & E2 >= 50 & E >= 50 & grade >= 55
    grade[!pass] <- pmin(50, grade[!pass])

    pass <- factor(pass + 1, labels = c("resit", "pass"))

    return(list(pass = pass, grade = grade))
}
```

1.4c

Create a table of the `factor` pass/resit to see how many people passed the course in one go, and also create a barplot of the `factor` with the title `"SCR Course in-one-go"`.

If you did not succeed in the previous task(s), then use the `pass` column `final_grades` that is in the model answer variables.

Answer:

```r
final_grades <- the_grades
final_grades$pass <- GiveCourseResults(the_grades)$pass
par(mfrow = c(1,1))
plot(final_grades$pass, main = "SCR Course in-one-go")
```

1.4d

Use the function `aggregate()` or `tapply()` to show the average grade of the students that passed the course, and of those that would have to do the resits.

Hint: if you did not succeed in the previous tasks, you may use the ***`CourseGrade`*** *column from the data set* ***`final_grades`***

Answer:

```r
final_grades$CourseGrade <- GiveCourseResults(the_grades)$grade
```

```r
tapply(final_grades$CourseGrade, final_grades$pass, mean, na.rm = TRUE)
```

```
##    resit     pass
## 33.14583 77.43434
```

2 Generate(d) Data

The data from the previous task consisted of generated names and ULCN numbers to protect privacy. In this second task we will ask you to generate UCLN numbers yourself, and to code your own function with which you can encrypt names.

2.1 About ULCN numbers (20)

Your ULCN number starts with an "s" and then a number of 7 digits. Let us describe the *population* of ULCN numbers from which we can generate a sample (like the ULCN numbers of the SCR course of last year). In this population 25% of the numbers are uniformly distributed over the interval `[1000001, 1699999]`, consisting of discrete values only. The ULCN numbers in this interval are also referred to as the *old* ULCN numbers. The remaining 75% of the numbers are uniformly distributed over the interval `[1700000, 2499999]`, also consisting of discrete values only. The numbers in this latter interval are referred to as the *new* ULCN numbers.

Knowing our "population" of ULCN numbers, we will ask you later to generate `49` unique ULCN numbers from this population. #### 2.1a

Create a vector object `ULCN_nrs` that consists of the numbers 1000001 to 2499999. Check whether the lenght of this vector is `1499999`. Then, use this `ULCN_nrs` vector to confirm with a logical expression that `699999` of the numbers are smaller than `1700000`, and `800000` are larger or equal to `1700000`.

Answer:

```
ULCN_nrs <- 1000001:2499999
ULCN_descr <- c(
  N = length(ULCN_nrs),
  N_old = sum(ULCN_nrs < 1700000),
  N_new = sum(ULCN_nrs >= 1700000)
)
ULCN_descr
```

```
##       N   N_old   N_new
## 1499999  699999  800000
```

2.1b

For the SCR-class we estimate that the ULCN numbers come from the same population as described above. Thus, around 25% of the students have an old ULCN number, and around 75% have a new ULCN number, the probability of drawing one specific old ULCN number is

```
prob_old <- 0.25 / 699999
```

and the probability of drawing specific new ULCN number is

```
prob_new <- 0.75 / 800000
```

Use the function `rep()` to create a vector object `probs_ULCN` that contains the probability of each ULCN number in `ULCN_nrs`. Also show that the sum of all probabilities in `probs_ULCN` is 1.

Answer:

```r
prob_old <- 0.25 / ULCN_descr[["N_old"]]
prob_new <- 0.75 / ULCN_descr[["N_new"]]
probs_ULCN <- rep(
  c(prob_old, prob_new),
  c(ULCN_descr["N_old"], ULCN_descr["N_new"])
)
sum(probs_ULCN) # check whether probability sum up to 1.
```

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

2.1c

Use `ULCN_nrs` and `probs_ULCN` to generate 49 unique ULCN numbers accordingly. Set a seed `20191030` (representing the date of this exam).

Answer:

```r
set.seed(20191030)
my_ULCNs <- sample(ULCN_nrs, 49, prob = probs_ULCN)
```

2.1d

Change your generated ULCN numbers into a **sorted** character vector where each number also has the "s" up-front (see `my_ULCNs` from the model answer variables).

Answer:

```r
my_ULCNs <- sort(paste("s", my_ULCNs, sep = ""))
```

2.2 Decrypting Names (20)

It is common practice to make sure that privacy related data is encrypted. For encryption and decryption of the data a 'key' is used. In this task you will have to encrypt the data with such a key. Let the key be of class `data.frame`, of which `key20191030` here below is an example.

```r
key20191030 <- {
  set.seed(20191030)
  data.frame(
    decrypt = c(letters, LETTERS, 0:9, " ", NA, ".", "/", "-"),
    encrypt = sample(c(letters, LETTERS, 0:9, " ", NA, ".", "/", "-")),
    stringsAsFactors = FALSE
  )
}
```

You will have to use this particular `key20191030` to encrypt the names of the students we have used before.

2.2a Import the data in two columns

The names of the students can also be found in the `names.txt` file from the `0_data` folder. Import the content of this file into a `data.frame` of two columns (First name and Last name). Ensure that the columns of the `data.frame` are `character` vectors.

Answer:

```
student_names <- read.table("0_data/names.txt", stringsAsFactors = FALSE)
```

2.2b Using the key to encrypt the names

Write a function that encrypts any character vector (= first argument), based on a key `data.frame` (= second argument). As a default, set the second argument equal to the `key20191030` variable.

Show that your function can encrypt a character vector like `c("Tommy", "Francina")` correctly into `c("IkddN", "pKGunXuG")`.

OR when using an R version >= 3.6.1 (check yourself with `getRversion()`) the character vector should encrypt to `c("2755 ", "loVnQ/nV")` (for the explanation, see subtask **2.2c**)

Hint: first try to write the function just for one name only. When you've succeeded, then try to see on how you could apply this function in a loop on each element of the character vector (or list).

Answer:

```
Encrypt <- function(names_decr, key = key20191030) {

  # names_decr <- student_names$V1
  # key <- key20191030

  raw_names <- strsplit(names_decr, split = "")

  map_to_decr <- 1:nrow(key)
  names(map_to_decr) <- key$decrypt

  names_encr <- sapply(raw_names, function(decr_name) {
    # decr_name <- raw_names[[1]]
    name_encr <- key$encrypt[map_to_decr[decr_name]]
    name_encr <- paste(name_encr, collapse = "")
    return(name_encr)
  })

  return(names_encr)
}
```

```
getRversion()
names_encr <- Encrypt(c("Tommy", "Francina"))
names_encr
```

2.2c

Create a new `data.frame` with the encrypted names by running an implicit loop (`lapply()`) over the two columns of the original `data.frame` with the (decrypted) names. *Note that this needs a bit more coding then just a `for` loop.*

Your encrypted `data.frame` should contain the exact same names as `encr_students` from te model answer variables. If you did not succeed to load the orignal (decrypted names) into a data.frame, use the `student_names` data.frame from the model answer variables. Similarly if you did not manage to write your own encrypt function, then use the following function instead:

```
Surrogate <- function(names_decr, key) {
  load("0_data/model_answer_vars.RData")
  if(length(names_decr) != nrow(encr_students)) {
    stop("wrong input")
  }
  if(names_decr[1] == "Tommy") {
    out <- encr_students[, 1]
  } else if(names_decr[1] == "Tejera") {
    out <- encr_students[, 2]
  } else {
    stop("wrong input")
  }
  return(out)
}
```

COMMENT: Here the `encrypted_names.txt` was created based on a `key20191030` data.frame that was generated based on the default Random Number Generator (RNG) in R version 3.6.1. For versions older than 3.6.0 the encrypted names would have been different, unless the RNG was set as follows

```
RNGkind(sample.kind = "Round")
```

which is according to the default settings of R versions that are older than 3.6.0. To set the RNG back to its default settings

```
RNGkind(kind = "default", normal.kind = "default") # for R versions < 3.6.0
RNGkind(kind = "default", normal.kind = "default", sample.kind = "default") # for R >= 3.6.0
RNGkind()
```

```
## [1] "Mersenne-Twister" "Inversion"        "Rejection"
```

For more information, see https://bugs.r-project.org/bugzilla/show_bug.cgi?id=17494

Answer:

```
list_encr <- lapply(student_names, Encrypt, key = key20191030)
encr_students <- data.frame(do.call("cbind", list_encr))
if (FALSE) { write.table(
  encr_students,
  file = "0_data/encrypted_names.txt",
  col.names = FALSE, row.names = FALSE,
  quote = FALSE
)}
```

Or, when using the surrogate function:

```
list_encr <- lapply(student_names, Surrogate, key = key20191030)
encr_students <- do.call("cbind", list_encr)
```

2.2d

Write your encrypted names (or the `encr_students` object) into a text file and store them in the `0_data` folder. Your encrypted names should be the same as those in the file `encrypted_names.txt` from the `0_data` folder.

Answer:

```
write.table(
  encr_students,
  file = "0_data/encrypted_names.txt",
  col.names = FALSE, row.names = FALSE,
  quote = FALSE
)
```

3 Bonus: A Visual illusion (20)

Take a look at Figure 2, you probably see black dots appearing and dissappearing. This Figure is a typical visual illusion that you could create with `R` as well.

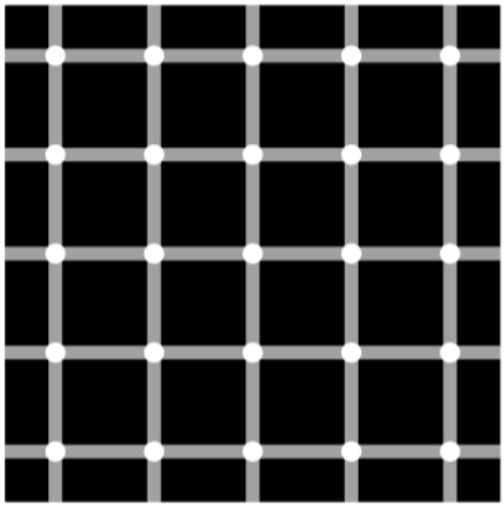


Figure 2: Visual illusion with Black Dots

Recreate the Figure 2 with your own code. For the plotting of the see-through white lines on a black background (making them to appear grey), use as color `"#FFFFFF90"`. Keep the aspect ratio of the vertical axis and that of the horizontal axis equal to 1 (`?plot`).

*Hint: full points can be obtained here if you do **NOT** use any implicit or explicit loops.*

Answer:

```
par(mfrow = c(1,1))
hsv_col <- "#FFFFFF90"
y <- rep(1:5, each = 5)
x <- rep(1:5, 5)
plot(x,y,
  xlim = c(0.5,5.5), ylim = c(0.5,5.5),
  bty = "n", type = "n", axes = FALSE,
  xlab = "", ylab = "",
  asp = 1
)
rect(xleft= 0.5, xright = 5.5, ybottom = 0.5, ytop = 5.5, col = "black")
segments(x0 = 1:5, y0 = 0.5, y1 = 5.5, lwd = 6, col = hsv_col)
segments(x0 = 0.5, x1 = 5.5, y0 = 1:5, lwd = 6, col = hsv_col)
points(x,y, pch = 16, col = "white", cex = 2)
```