Statistical Programming Languages 2022

Take-Home Exam

The take-home exam consists of the following five programming tasks. Solve all of the exercises using the file `SPL22_surname_name.R` as a template for your solutions. Rename the file accordingly and fill in your names and student ID (Matrikelnummer) at the designated place in the file. Append the number of the corresponding exercise so that you create a separate file for each exercise resulting in five files `SPL22_surname_name_1.R`, ..., `SPL22_surname_name_5.R`.

Your `R` code

- must solve the exercises without errors or warnings when executed from top to bottom; Exceptions are intended warnings or errors (e.g. in complex functions) and ill-set working directories;
- must provide self-contained solutions, i.e. the solutions to all exercises must be reproducible and obtained with `R` code and each file must be executable on its own (independent of other exercises);
- must only include one solution per task (otherwise only the first one will be considered);
- must be your own intellectual property; If parts of code were taken from external sources, they have to be cited properly using comments;
- must be comprehensible and efficient;
- must include reasonable comments, also to structure the code (subtasks) and to answer specific questions;
- must not use any additional packages[1], unless required/allowed explicitly in a specific exercise;
- must only include pure functions.

Note that

- errors or warnings can lead to 0 points for the respective task;
- using additional packages can lead to 0 points for the respective task;
- following this style-guide is part of the grading.

You have to submit your solutions of this take-home exam as **one zip file** until **24 June 2022, 11:59 p.m.** via Moodle.

Exercise	1	2	3	4	5	$\sum$
Points	3	22	20	26	40	111

Good Luck!

[1] You may use all packages loaded per default, namely, `stats`, `graphics`, `grDevices`, `utils`, `datasets`, `methods`, and `base`.

Exercise 1 *Session Info* [3 points]

Start a new `R` session and execute the following code:

```
> sink("my_session.txt")
> sessionInfo()
> sink()
```

Include the resulting file "my_session.txt" in your submission, i.e., in the zip file containing your solutions. Use comments to briefly describe in your own words, what kind of information the function `sessionInfo()` provides.

Exercise 2 *COVID-19* [22 points]

The file corona.csv contains information on the development of the COVID-19 pandemic in EU/EEA countries[2]. Amongst others, the following variables were observed in the data set:

Variable (original name)	New name	Description
`dateRep`	`date`	Observation date
`cases`		Number of newly infected people on this date
`deaths`		Number of people who died from COVID-19 on this date
`countriesAndTerritories`	`country`	Country/Territory of the observation
`popData2020`	`population`	Population of the Country/Territory (as of 2020)
`indicator14`		Number of `cases` per 100,000 residents over the last 14 days (including current date)

a) Set your working directory appropriately and read the data conveniently into a data frame `corona`. Delete all variables not contained in the table above and rename the remaining ones according to **New name**, if **New name** is not empty for this variable. Transform the variable `date` to class `POSIXct` appropriately. Are the classes of the other variables represented adequately? Briefly justify your answer. [4.5]

b) How many observations in the data set contain missing values? [2]

c) How many observations in the data set counted more than 300 `cases`? What's their share on the total number of observations? [1.5]

d) Use an appropriate test to decide whether the average value of `indicator14` differs in the countries Italy and Spain in April 2020 (significance level: 5%). Briefly justify your decision on whether the average value differs. [2.5]

e) Compute the total number of `deaths` per `country` in October 2021 and sort them in ascending order. [3]

f) In Germany, a 7 day indicator (`indicator7`) is used to evaluate the rate of new infections instead of `indicator14`. It is the number of `cases` per 100,000 residents over the last 7 days (including the current date) in one `country`, i.e.,

$$\texttt{indicator7}_i = \frac{\sum_{k=i-6}^{i} \texttt{cases}_k}{\texttt{population}_i} \cdot 100000$$

for the ith observation of the respective `country`. Create a new numeric variable `indicator7` containing this 7 day indicator and add it to the data frame `corona`. Adhere to the following:

- Make sure that the order of observations in the data frame `corona` stays unchanged.
- For the first 6 observed days of a `country`, the new variable `indicator7` has the value `NA`.
- Always use the observations of the last 7 days existent in the data frame for your computation. (Some `date`-`country`-combinations are non-existent in the data frame, e.g., 9th January 2022 for Denmark.) [6]

g) Add a new `factor` variable `alert_level` to the data frame `corona` which is

- "green" if `indicator7` is less than 35,

[2]Source: https://www.ecdc.europa.eu/en/publications-data/data-daily-new-cases-covid-19-eueea-country as of 12 April 2022.

- “yellow” if `indicator7` is at least 35 but less than 50,
- “red” if `indicator7` is at least 50 but less than 100,
- “darkred” if `indicator7` is at least 100.

If you did not manage to solve exercise f) use the variable `indicator14` instead, but double the threshold values (e.g., use 70 as upper bound for “green”). [2.5]

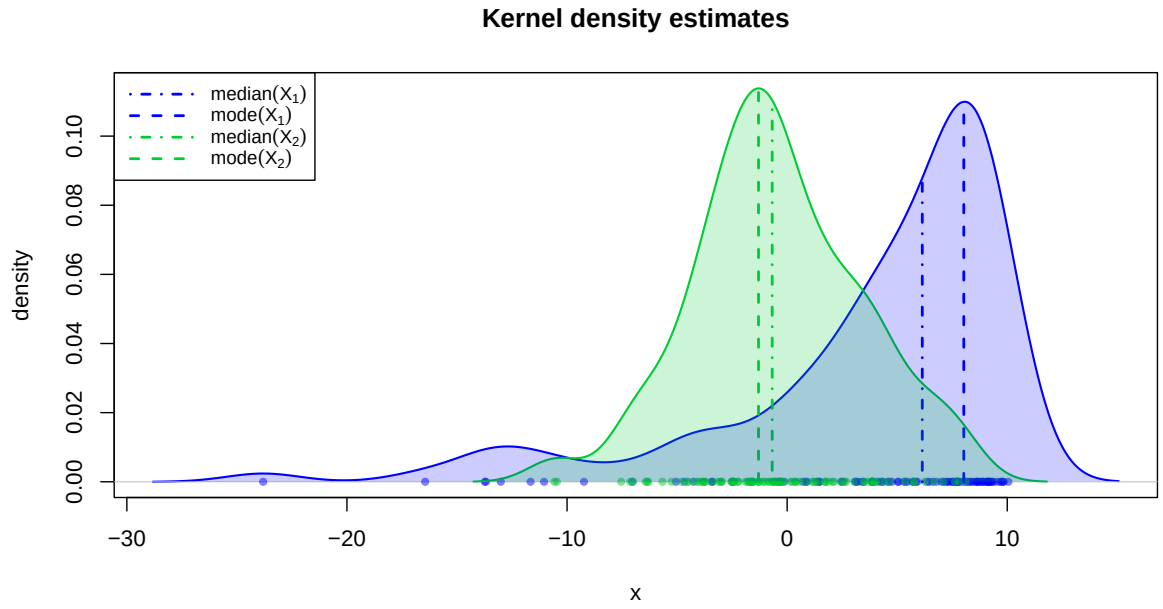
Exercise 3 *Plots* [20 points]

We consider two random variables $X_1 = 0.5U_1 - 3V + 10$ and $X_2 = 4U_2$ where U_1 and U_2 are standard normal random variables and V is exponentially distributed with rate $\lambda = 0.5$ (see `?rexp`).

a) Sample 100 independent observations of X_1 and X_2 each and compute kernel density estimates (`?density`) for both samples.

b) Compute the median and the mode for your samples of X_1 and X_2.
Hint: The mode is the value at which the density takes its maximum value

c) Reproduce the following plot. Note that the densities will look different, depending on the concrete sample from a). Make sure your code produces a meaningful plot even if you choose a new sample of X_1 and X_2.
Hint: You can use `polygon()` *to fill the area under the curve and* `rgb()` *to construct transparent colours.*



d) Produce a similar plot with the package `ggplot2`. Use your kernel density estimates from a) and not the function `geom_density`.

Exercise 4 *Expand polynomials* [26 points]

In this programming task you are asked to write functions that process a polynomial in latex code. Your functions should work for all inputs in the format as in the examples, not just for the examples. In particular, exponents may or may not be enclosed in curly brackets, and the input may contain blank spaces. You will be graded based on test cases that are similar but not identical to the examples. You can use the package `stringr` to solve this task.

a) Write a function `get_poly_structure` that converts a polynomial given as a character to a named vector. The entries of the vector shall correspond to the coefficients of the polynomial and be namend as `x^n`, where n is the corresponding exponent. Test your function on the following examples. Your output should be precisely the same as given below. Note that the last entry of the vector is not equal to zero.

```
> get_poly_structure("-3x^5 + 2x^2 - x^{10} - 1 + x")

 x^0  x^1  x^2  x^3  x^4  x^5  x^6  x^7  x^8  x^9 x^10
  -1    1    2    0    0   -3    0    0    0    0   -1

> get_poly_structure("4x^5-x^{ 3}+3x-10x^3+2x-4x^5")

x^0 x^1 x^2 x^3
  0   5   0 -11
```

b) Write a function `expand_polynomial` that expands a polynomial given in factorized form with an arbitrary number of factors. Your function should except a character as input, with factors in round brackets and no multiplication symbol, and return the expanded polynomial in standard form (i.e simplified and with ordered exponents from highest two lowest). Your output should be precisely the same as given below.

```
> expand_polynomial("(x - 2)(x + 2)")

[1] "x^{2} - 4"

> expand_polynomial("(x + 1)^4")

[1] "x^{4} + 4x^{3} + 6x^{2} + 4x + 1"

> expand_polynomial("(2x + 1)(3 - x)(-2x - 5)")

[1] "4x^{3} - 31x - 15"

> expand_polynomial("(3x^4 + 5)(-2x + 1)^2")

[1] "12x^{6} - 12x^{5} + 3x^{4} + 20x^{2} - 20x + 5"
```

If the polynomial is already expanded, i.e. no "(" occurs, your function should return the polynomial in standard form.

```
> expand_polynomial("x^{ 3}+3x-10x^3+2x+4x^5")

[1] "4x^{5} - 9x^{3} + 5x"
```

Exercise 5 *Memory* [40 points]

The game *Memory* is a card game, which can be played by any number of persons (including 1). Pairs of equal cards are arranged randomly in a grid face down. In turn, each player chooses two cards and turns them face up, which is considered one move. If the cards are equal, then that player wins the pair and makes another move. If the cards are different, the cards are turned face down again and play passes to the next player. The game ends when the last pair has been picked up. The player with the most pairs wins. There may be a tie for first place.

Write a function `memory` that enables to play Memory via the `R` console for a specified grid dimension, possible cards and number of players. It should fulfill the following:

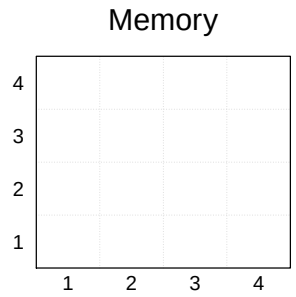
- The arguments of the function correspond to the number of rows (`n_row`; default: `4`), to the number of columns (`n_col`; default: `4`), to the possible card symbols (`pch`; default: `1:13`), to the possible card colors (`col`; default: `1:8`), and to the number of players (`n_player`; default: `2`). The basic structure is:

```
> memory <- function(n_row = 4, n_col = 4, pch = 1:13, col = 1:8,
+                    n_player = 2) {
+   # Your part
+ }
```

- The player who begins is selected randomly. When calling the function with appropriate parameters (see below), the following message is printed in the console (appropriately adapted, if another player starts) and an empty playing field of the specified dimension is plotted (in the Plots-tab of RStudio).

```
> memory()
Player 1 starts!
In each move you have to choose two cards.
```

  The empty playing field for 4 rows and 4 columns should look like this:



- Let $n := \texttt{n\_row} \cdot \texttt{n\_col}$. If n is odd, the function stops with the following error (hint: `?stop` or `?stopifnot`):

```
Error in memory(): n_row * n_col must be an even number.
```

  If n is even, each pair of equal cards corresponds to a unique combination of symbol (`pch`) and color (`col`), which is chosen randomly from all possible combinations of the symbols and colors passed to the function via the arguments `pch` and `col`. If the number of possible combinations (which is the product of the number of symbols and the number of colors) is smaller than $\frac{n}{2}$, the function stops with the following error (hint: `?stop` or `?stopifnot`):

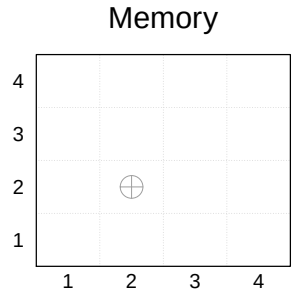
```
Error in memory(): Not enough different possible cards (combinations of pch and
col) were specified for the given size of the playing field.
```

Otherwise, the n cards are arranged randomly on the playing field by the function (invisible for players).

- The function requests a player to make their move, i.e., choose their two cards, via the console, where the player inserts the respective numbers of the row and the column (hint: `?scan`). The output in the console should be as follows (appropriately adapted):

```
Player 1, choose your first card (1: row, 2: column)!
1: 2
2: 2
```

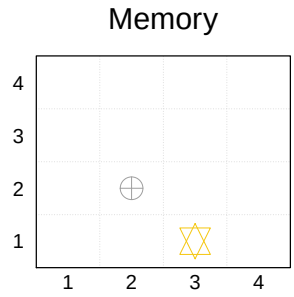
Immediately after the input, the chosen card is turned on the playing field, i.e., the respective symbol in the respective color appears at the selected position in the plot:



In the console, the player is asked to choose a second card:

```
Player 1, choose your second card (1: row, 2: column)!
1: 1
2: 3
```

The second card is turned in the playing field, as well:



- If the chosen cards do not match (as in our example), the following message (appropriately adapted) appears in the console, passing the play to the next player:

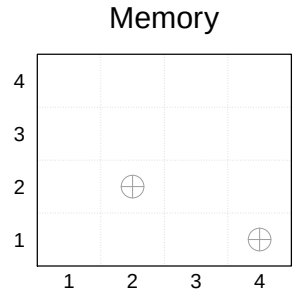
```
Wrong, Player 2 plays! Press [y], when you are ready to move on!
1:
```

After confirmation via the y-key (hint: `?scan`), the cards are turned face down again, i.e., the plot of the current playing field appears again. In our example, this corresponds to the empty playing field illustrated in the first plot. Via the console, the next player is asked to choose their cards:

```
Wrong, Player 2 plays! Press [y], when you are ready to move on!
1: y
Player 2, choose your first card (1: row, 2: column)!
```

```
1: 1
2: 4
Player 2, choose your second card (1: row, 2: column)!
1: 2
2: 2
```

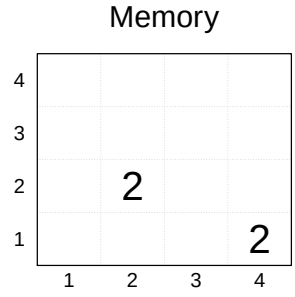
Again, the respective cards are turned immediately after the inputs via the console. In our example we get the following plot after turning both cards:



- If the chosen cards do match (as in our example) and are not the last pair of cards on the playing field (in this case, the game ends, see below), the following message (appropriately adapted) appears in the console:

```
Correct, Player 2 plays again! Current leaderboard:
       Player 1        Player 2
              0               1
Press [y], when you are ready to move on!
1:
```

The current leaderboard shows the number of pairs won per player (including all players, from 1 to `n_player`). After confirmation via the y-key, the matching cards are removed from the current playing field and assigned to the respective player, i.e., the respective positions in the plot are marked with the number of the player, resulting in the new current playing field:



In the console, the player is asked to choose the first card of their next move:

```
Correct, Player 2 plays again! Current leaderboard:
       Player 1        Player 2
              0               1
Press [y], when you are ready to move on!
1: y
Player 2, choose your first card (1: row, 2: column)!
1:
```

- The complete console output for our example up to this point looks like this:

```
> memory()
Player 1 starts!
In each move you have to choose two cards.
Player 1, choose your first card (1: row, 2: column)!
1: 2
2: 2
Player 1, choose your second card (1: row, 2: column)!
1: 1
2: 3
Wrong, Player 2 plays! Press [y], when you are ready to move on!
1: y
Player 2, choose your first card (1: row, 2: column)!
1: 1
2: 4
Player 2, choose your second card (1: row, 2: column)!
1: 2
2: 2
Correct, Player 2 plays again! Current leaderboard:
       Player 1        Player 2
              0               1
Press [y], when you are ready to move on!
1: y
Player 2, choose your first card (1: row, 2: column)!
1:
```

The game continues in the same manner.

- Every chosen card must be valid. In particular, the row and column numbers must be integers within the specified ranges and the position must contain a card that is part of the game (i.e., not yet won by a player). Otherwise, the function reacts with the following message and a new prompt until the chosen card is valid (recall that row 1, column 4 was already won by `Player 2` in our example):

```
Player 2, choose your first card (1: row, 2: column)!
1: 0
2: 1
Card not valid. Again:
1: 1
2: 4
Card not valid. Again:
1:
```

Similarly, the request "`Press [y], when you are ready to move on!`" (after both, non-matching and matching cards) reacts to any other input than the y-key by repeating the request:

```
Wrong, Player 2 plays! Press [y], when you are ready to move on!
1: 4
Press [y], when you are ready to move on!
1:
```

- The game ends, when the last two equal cards are chosen. They are removed from the current playing field and assigned to the respective player, i.e., the respective positions in the plot are marked with the number of the player, immediately after the second card was turned (without confirmation via the y-key). The function determines the winner(s) and terminates with one of the following messages (appropriately adapted) in the console, respectively, including the final leaderboard, which is the latest current leaderboard (see above):

```
Correct, Player 1 wins! Final leaderboard:
       Player 1        Player 2
              6               2
```

or

```
Correct, Player 1 and Player 2 are tied winners! Final leaderboard:
       Player 1        Player 2
              4               4
```

- Hints:
  - Outsource (self-contained) subproblems in separate functions, which are then called in the main function `memory`.
  - Test the function with different calls. In particular, make sure that it also works for non-default parameters.