STA 141A: Homework 1

Instructor: Akira Horiguchi

Student name: ABCDE FGHIJ; Student ID: 123456789

Oct 1, 2025 (Wednesday), 23:59 PDT

The assignment must be done in an R Markdown or Quarto document. The assignment must be submitted by the due date above by uploading:

1. a .pdf file in GRADESCOPE (if you can knit/compile your .rmd to a .html file only, please save the created .html file as a .pdf file (by opening the .html file -> print -> save to .pdf)).

Email submissions will not be accepted.

Each answer has to be based on `R` code that shows how the result was obtained. The code has to answer the question or solve the task. For example, if you are asked to find the largest entry of a vector, the code has to return the largest element of the vector. If the code just prints all values of the vector, and you determine the largest element by hand, this will not be accepted as an answer. No points will be given for answers that are not based on `R`. This homework already contains chunks for your solution (you can also create additional chunks for each solution if needed, but it must be clear to which tasks your chunks belong).

There are many possible ways to write `R` code that is needed to answer the questions or do the tasks, but for some of the questions or tasks you might have to use something that has not been discussed during the lectures or the discussion sessions. You will have to come up with a solution on your own. Try to understand what you need to do to complete the task or to answer the question, feel free to search the Internet for possible solutions, and discuss possible solutions with other students. It is perfectly fine to ask what kind of an approach or a function other students use. However, you are not allowed to share your code or your answers with other students. Everyone has to write the code, do the tasks and answer the questions on their own.

During the discussion sessions, you may be asked to present and share your solutions.

```
set.seed(2025*4)  # do not change this; this helps to reproduce the "random" results
```

1. Vectors and lists

Suppose that we have:

- four types of animals: `cat`, `dog`, `cow`, `squirrel`;
- four possible colors: `white`, `black`, `brown`, `red`;
- five possible attributes: `big`, `small`, `angry`, `cute`, `sad`.

a. Generate three random samples of size `50` from each of the three groups, so that you have a vector `Animal` containing 50 animals, a vector `Color` containing 50 colors and a vector `Attribute` containing 50 attributes.

```
# Your solution
```

b. By using the `sum()` function and logical operations, compute the number of animals that are cats or dogs.

```
# Your solution
```

c. Compute the relative frequency of cats, dogs, cows and squirrels in the sample.

```
# Your solution
```

d. Create a contingency table between `Animal` and `Attribute`. (No need to compute the marginal totals or the grand total.)

```
# Your solution
```

e. Put the three vectors together in a list of three elements called `mylist`, so that each vector is an element of the list. Use the command `length(mylist[1])` to print the length of the first vector. Is this code actually printing the length of the vector? Explain, and write the correct code to print the length of the first vector of the list.

```
# Your solution
```

2. Data frames

a. Create a data frame `df` with one numeric vector `index` and two character vectors `firstName` and `surname` with length 7, respectively. The vector `index` consists of randomly chosen values from 1 to 10 (the same values are NOT allowed to appear several times). The entries of `firstName` are randomly chosen from the names 'Ant', 'Bug', 'Cat' (the same values are allowed to appear several times), and the entries of `surname` are randomly chosen from the names 'Tolstoy' and 'Lee' (the same values are allowed to appear several times).

```
# Your solution
```

b. Select all elements of the data frame `df`, where the value of `index` is greater than 7.

```
# Your solution
```

c. Select all elements of the data frame `df`, where `firstName` is 'Ant'.

```
# Your solution
```

d. Select all elements of the data frame `df`, where both the value of `firstName` is 'Bug', and the `surname` is 'Lee'.

```
# Your solution
```

e. Use the function `cbind()` to add a new column to `df` called `middleName` randomly chosen from the names 'Davis' and `NA`. Store this new dataframe as `df2`.

```
# Your solution
```

f. Use the function `rbind()` to add a new row to `df2` with the values `index=11`, `firstName='Dog'`, `middleName='Leash'`, and `surname='Davis'`. Store this new dataframe as `df3`.

```
# Your solution
```

g. Return a character vector containing the names of the cars in the built-in dataset `mtcars` with horsepower (`hp`) equal to either 66, 110, 150, or 180. (Hint: use the `rownames()` function and the `%in%` operator.)

```
# Your solution
```

3. Matrices

a. Create two matrices `A`, `B`, with two rows and two columns each, where `A` contains the values 1 to 4, and `B` the values 5 to 8. Further, create a vector consisting of 6 elements which are randomly chosen from the values 1 to 100 (the same values are allowed to appear several times). Based on this vector, define a matrix `C` with 2 rows and 3 columns.

```
# Your solution
```

b. Return the matrices which are obtained by element-wise addition (`+`) and multiplication (`*`) of the matrices `A` and `B`.

```
# Your solution
```

c. Calculate the matrix products of `A` and `B`, and `B` and `C`.

```
# Your solution
```

d. Return the values in the 1st row of `A`, in the 2nd row of `B`, in the 1st column of `C`, and in the 1st row and 2nd column of `C`.

```
# Your solution
```

e. Use the `apply()` function to determine the row-wise sum of `C`.

```
# Your solution
```

4. Conditional and repetitive execution

a. Create a numeric vector `x` of 50 randomly generated integers between -3 and 3 (inclusive). Create a character vector `y` of length 50 whose values are `"negative"`, `"zero"`, and `"positive"` depending on the corresponding value of `x`. Specifically, if `x[i] < 0`, then `y[i]` should be `"negative"`; if `x[i] == 0`, then `y[i]` should be `"zero"`; and if `x[i] > 0`, then `y[i]` should be `"positive"`.

```
# Your solution
```

b. Use a `for` loop to calculate $\frac{1}{10}\sum_{i=3}^{12} 2^i$.

```
# Your solution
```

c. Use `for` loops to calculate $\frac{1}{1000}\sum_{i=2}^{9}\sum_{j=1}^{i} 4^{2i-3j}$.

```
# Your solution
```

d. Find the bug: The following `for` loop creates a vector that contains the sum of the first `n` numbers. In particular, if you set `n <- 10`, the `for` loop should return a vector of size `10` containing the values 1, (1+2), (1+2+3), ..., (1+2+3+4+5+6+7+8+9+10). Explain why this `for` loop does not create the desired vector, and write the correct code.

```
n <- 10
sums <- numeric(n)
for (i in 1:n) {
  sums <- sum(1:i)
}

# Your solution
```

e. Explain what the following code does:

```
n <- 10
x <- 1:(2*n)
while (x[1] < n) {
    x <- x[-1]
}
x
```

5. `lapply()` and `sapply()`

a. Redo questions 4b) and 4c), but instead of using `for` loops, use the `sapply()` function to perform the calculations.

```
# Your solution
```

b. Create a numeric vector `x` of 50 randomly generated integers between -3 and 3 (inclusive). Create a list `fun_list` of the following functions: `mean()`, `sd()`, `min()`, and `class()`. Use the `lapply()` function to apply the functions in `fun_list` to the vector `x` and return a list of the results.

```
# Your solution
```

c. Create a numeric vector `y` of 30 randomly generated numbers between 0 and 1 (hint: use `runif()`). Use the `sapply()` function to apply the functions in `fun_list` to the vector `y` and return a vector of the results. What is the difference between the first element of this vector output and the first element of the list output from part (b)?

```
# Your solution
```

d. Using the function `CreateDF()` defined below, use the `lapply()` function to create a data frame for each of the following values of beta: -5, -3, -1, 1, 3, 5. Store this list of data frames as `df_list`. Then use either the `Reduce()` function or the `purrr::reduce()` function to stack all the data frames in `df_list` into a single data frame called `final_df` that has two columns and `6*N` rows, where `N` is defined in the `CreateDF()` function (which is 10 in this case).

```
CreateDF <- function(beta) {
  N <- 10
  data.frame(beta = beta, gauss = rnorm(N, mean=beta))
}
# Your solution
```