Assignment 01

Statistical Computing and Empirical Methods

A word of advice

Think of the SCEM labs as going to the gym: if you pay for gym membership, but instead of working out you use a machine to lift the weights for you, you won't get the benefits.

ChatGPT, DeepSeek, Claude and other GenAI tools can provide answers to most of the questions below. Before you try that, please consider the following: answering the specific questions below is not the point of this assignment. Instead, the questions are designed to give you the chance to develop a better understanding of estimation concepts and a certain level of **statistical thinking**. These are essential skills for any data scientist, even if they end up using generative AI - to write an effective prompt and to catch the common (often subtle) errors that AI produces when trying to solve anything non-trivial.

A very important part of this learning involves not having the answers ready-made for you, but instead taking the time to actually search for the answer, trying something, getting it wrong, and trying again.

So, make the best use of this session. The assignments are not marked, so it is much better to try the yourself even if you get incorrect answers (you'll be able to correct yourself later when you receive feedback) than to submit a perfect, but GPT'd solution.

Before starting the assignment it is recommended that you first watch the video lectures corresponding to Week 1.

This assignment will be done in Rstudio. Please have it open before you start.

You don't need to submit this assignment.

1: Create a data frame

1. Within RStudio, find the Console window. In the Console, write a command to create a vector called `animals` containing the names of a few animals. What type of vector is this? When printed, your vector might look something like this:

```
## [1] "Snake" "Ostrich" "Cat" "Spider"
```

2. Create a vector of the same length called `num_legs` which gives the number of legs of the different animals. When printed, it may look something like this:

```
## [1] 0 2 4 8
```

3. Combine these two vectors in a data frame called `animals_df`, which has two columns - one with the names of animals, and another with their respective numbers of legs. The result should look something like this:

```
## animals num_legs
## 1 Snake 0
## 2 Ostrich 2
## 3 Cat 4
## 4 Spider 8
```

4. Check the Environment window and find out what objects (variables) are available. You should be able to find objects called `animals`, `num_legs`, and `animals_df`. Can you find the data type of these objects there?

The functions required by the tasks above can be found in the example below.

```
city_name <- c( "Bristol", "Manchester", "Birmingham", "London") # vector of city names
population <- c(0.5, 0.5, 1, 9) # vector of approximate populations (in millions of people)
cities_populations_df <- data.frame(city_name, population) # data frame
```

Note that R accepts either `<-` or = for variable attribution. Which one to use is a matter of stylistic preference.

2: 2 Check and delete objects

Apart from creating objects, you can also check and delete them. In this item, you will also learn how to find help documents of R functions.

1. After you finish creating the data frame required in **Item 1**, get a list of objects in the working environment.

- Function `ls()` returns a vector of character strings giving the names of the objects in the environment. Type `?ls` to check its usage.
- Do you find the objects `animals`, `num_legs`, and `animals_df` again?

2. Use function `rm()` (type `?rm` to see documentation and usage examples) to remove the object `num_legs` from the working environment. After that, check the list of objects in the current environment again, to verify that `num_legs` has been removed.

3. Remove all objects in the working environment. You can use a combination of `rm()` and `ls()` to do it (check the examples of use of function `rm()`).

3: Create a data frame in R Scripts

In **Item 1**, you created a data frame in the console in RStudio. Now you will create an R script and create a data frame within that script.

1. Remove all objects in the working environment (you should have already done it at the end of **Item 2**.

2. Go to `File -> New File -> R Script` (or press `ctrl + N`) to create your script. Save the file by clicking `File -> Save` (or press `ctrl + S`) and give the file a name of your choice, e.g., "myFirstRScript.R".
3. Now you can start writing code in the script. Create a data frame `animals_df`, using the same code you wrote in **Item 1**
4. You can run all the code within your script by clicking on the "Source" button at the top right of the script pane (alternatively, click `Code -> Source` or `ctrl + shift + S`).
5. Get a list of objects in the working environment. Check if it is the same list as what you got in **Item 2** Step 1?

4: Create a data frame in R Markdown

Now we'll create an R Markdown with HTML output as follows:

`File -> New File -> R Markdown`

This will open a create file window where you can choose:

- A title for your document;
- An author name (for example your name);
- An output format. Let's choose HTML.

Then click "OK" to create an R Markdown file. This will create a template project.

Explore the project and note its key features:

- There is a block of YAML header at the start of the document which gives key information, e.g., title and output format.
- You can create headings of sections with #, subsections with ## and so on.
- You can embed blocks of R code by using ```{r} before the R code, and ``` afterwards.
- By default, both the code and its output are displayed. If you only want to include the output but not the code itself include the option `echo = FALSE` in the code block prefix (i.e., ```{r, echo = FALSE}.
- If you don't want to include either the code or the output (e.g., to have something that executes silently) then include the option `include = FALSE`.
- You can include hyperlinks in the R Markdown document by writing `<url-name>` (outside of code blocks).

Save your R Markdown file. Remove everything in your R Markdown except for the YAML section at the top with the title, author, date, and output, then do the following:

1. Insert a block of R code. In this block of code, create a data frame `animals_df`, using the same code you wrote in **Item 1**.
2. Insert another block of code to print your data frame `animals_df`.
3. You can then generate an HTML file with the "Knit" button just above your scripting window (or press `ctrl + shift + K`. Check what you have in the HTML file.

5: Matrix operations

Use the `seq()` function to generate a sequence of numbers starting at 12 and decreasing to 2 in steps of -2. Call this vector `x_vect`. You may want to run `?seq` or `help(seq)` to help you do this. The vector `x_vect` should look like this:

```
## [1] 12 10 8 6 4 2
```

Now convert the vector `x_vect` into a 2×3 matrix (i.e., with 2 rows and 3 columns) called `X`, using the `matrix()` function. The result should look like this:

```
##      [,1] [,2] [,3]
## [1,] 12   8      4
## [2,] 10   6      2
```

Next create a column-wise 2×2 matrix called `Y` consisting of a sequence of four numbers from 1 to 4. The matrix `Y` should look like this:

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

Create another 2×2 matrix called `Z` which looks as follows:

```
##   [,1] [,2]
## [1,] 4   8
## [2,] 6   10
```

Recall that for a 2×2 matrix

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

the transpose of this matrix (note the indices) is

$$A^T = \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix}$$

}

Use the function `t()` to compute Y^T and Z^T.

Matrix sums. Compute the matrix sums `Y + Z` and `Z + Y`. The result in both cases should be the same. We call such operations commutative. This means that reversing the order does not change the result.

Matrix multiplication. By default, R performs *element-wise* matrix multiplication, $A \odot B$. Try `Y * Z` and check the result.

To compute the proper matrix products `AB`, you need to use the matrix product operator, `%*%`. Try `Y %*% Z` and `Z %*% Y` and check the results. Are they the same? Is matrix multiplication commutative?

Recall that the matrix multiplication between two 2×2 matrices `A` and `B`, is defined as:

$$AB = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Now use R to compute the matrix product YX. What happens if you try to compute XY? Read (and try to understand) the error message you get.

Matrix inverse. Compute the matrix inverse Y^{-1} via the `solve()` function:

```
solve(Y)
```

You should get something like this

```
##  [,1] [,2]
## [1,] -2  1.5
## [2,] 1   -0.5
```

Next, check what you get from computing $Y^{-1}Y$ in R.

Now, compute $Y^{-1}X$. Your result should look like this:

```
##  [,1] [,2] [,3]
## [1,] -9 -7 -5
## [2,] 7 5 3
```

Can you do this (obtain the results of $Y^{-1}X$) without first computing Y^{-1}? Try running the help command on the `solve()` function by typing `?solve` into the R console to find out how to do this.

6: Writing a function within R

First, create an R script (as you did in **Item 3**), and save it. We'll use it to write an R function.

It is useful to know how a function is defined in R as well as basic control flow statements in R (e.g., if-else, for, while). Before you continue, you may want to first look ahead at the sample function called `is_prime`, provided at the end of this question, to get a sense of function definition and control flow statements.

1. Within your script, create a short function called `myFirstRFunc` which takes in a single numerical argument `n` and outputs the sum of all those numbers strictly below `n` which are divisible by 2 or 7. For example, if `n = 14` then it should be the sum of 2, 4, 6, 7, 8, 10, 12, so `myFirstRFunc(14)` should return 49. Make sure your comment your code in the function. You may want to include a check so that your function produces an error if it is given an argument which isn't a non-negative integer.

2. Run the script by clicking on the "Source". This will load your function into the R environment. Check what you get if you apply the function to 1000, i.e., `myFirstRFunc(1000)` (if you have been successful your function should output the answer 284787)

How to write a function in R? An example.

```
# This example function takes as input a positive integer and outputs Boolean
is_prime <- function(num){
  # Stop if the input is not a non-negative integer
  stopifnot(is.numeric(num), num %% 1 == 0, num >= 0)

  # Initialise truth value output with TRUE
  t_val <- TRUE

  if(num < 2){
    # Output FALSE if input is either 0 or 1
    t_val <- FALSE
  } else if (num > 2){
    # Check possible divisors i no greater than sqrt(num)
    for(i in 2:sqrt(num)){
      if(num %% i == 0){
        # if i divides num then num is not prime
        t_val <- FALSE
        break
      }
    }
  }

  # return the truth value which says whether or not num is prime
  return(t_val)
}
```

7: Futher R Markdown exercises

Next we will learn about how to include plots and mathematical formulae in R Markdown.

1. Create a new R Markdown file and save it using any name you prefer.

2. Within your R Markdown insert a section heading called “Wave plot”.
3. Insert a code block to do the following.

- Define a vector called x consisting of a sequence which starts at 0 and goes to 20 in increments of 0.01. You can do this using the `seq()` function.

- Next, create a vector called y, which is of the same length as x, such that the i-th entry of y is equal to the `sin` function of the i-th entry of x. This can be done via the R function `sin()`.

- Create a data frame called `sin_df` with two columns: x and y. You can inspect the first few rows of your data frame with the `head()` function, like this: `head(sin_df,3)`

```
##   x    y
## 1 0.00 0.000000000
## 2 0.01 0.009999833
## 3 0.02 0.019998667
```

If you have been successful you will observe a similar output after “Knitting” your R Markdown file.

- Write a statement to plot `sin_df`. The most basic way of doing this is using the `plot()` function. This aims to display a plot with x in the x-axis and y in the y-axis.

3. Insert the following mathematical formula into your Markdown file:

$$sin^2(x) + cos^2(x) = 1$$

R Markdown accepts syntax for mathematics. For a quick guide on how to include mathematical notation into your R Markdown, please check the Examples file provided in Blackboard. For a more complete guide, check https://bookdown.org/yihui/bookdown/markdown-extensions-by-bookdown.html#equations

8: Version control with RStudio and git (optional)

Next we will combine RStudio with git to create an R project with version control.

1. **Connect RStudio to git**. If you do not yet have a GitHub account sign up for one at https://github.com/. Next, connect your RStudio to git via R package *usethis*. Within RStudio perform the following steps within your R console:

```
install.packages("usethis") # Install the "usethis" R package
library(usethis) # Load the "usethis" R library
use_git_config(user.name = "Bob Smith", user.email = "bob@example.org") # Set
profile info
```

The first step installs the *usethis* R package. The second loads it into the R session. You may be prompted to also install *RTools* at this stage. If so, follow the links provided by RStudio. The third step sets your git profile information. The `user.name` can be the same as the username you used to set up your git profile, but it is not necessary. The `user.email` needs to be the same address as you used to set up your git profile.

2. **Create an R project with version control.** Create a project on GitHub. Go to https://github.com/ on your web browser and log into your git account. Create a new repository by clicking the green #New" button. Next:

– Give your repository a name (e.g., "firstRProject").

– Give your repository a description (e.g., "This is a repo for an R project").

– Check the box which says "Add a README file".

– Always be mindful of whether your repo is public or private. For now, set it as "public".

– Click the green "Create repository" button.

You have now created a git repository. Next click the green "Code" button on the repository page and copy the URL provided. This is the repository URL.

3. **Create a project in RStudio.** Go back to RStudio. Then go to:

```
File -> New Project... -> Version Control -> Git
```

– Under `Repository URL:` Paste the repository URL which you copied from your git repo. (if you have chosen your project to be private, you may be asked to input your account details. Follow the instructions to gain access).

– Under `Project directory name:` Choose a directory name.

– Under `Create project as a subdirectory of:` Choose where to store your project on your local machine.

– Check the box marked `Open in new session`.

– Click create project.

You have now created an R project with version control!

4. **Committing and pushing.** Within your new project click

```
File -> New File -> R Script
```

Type some R code in your new R script e.g.

```
a <- 1
```

Now save your new file by selecting `File -> Save` and give it a name. Then:

- Go to the git tab at the top right of your screen. Then click the "commit" button. On the left you will see a collection of check boxes. Here you can choose which files you want to add to your git repo. For now, just check all of the files.

- Write a commit message in the “Commit message” box (e.g., “This is my first commit”). It’s good practice to always include a descriptive commit message.

- Click the commit button. You should see a message which describes your changes. Commits act as a “snapshot” which records the current state of your repository.

You have now taken a local “snapshot” of your repository by performing a “commit”. Next you want to record this snapshot on the central repository. This is done by performing a “Push”. All you have to do is press the green “Push” button within the git panel in RStudio. You may be asked to enter your password or personal access tokens (you can create your token, if needed, at https://github.com/settings/tokens. You typically only have to do this once).

Now check that your push has been successful. Go back to your Github account on your browser, navigate to the *Repositories* section and click on the name of the repository you created in the previous stage of this assignment. You should see a record of your most recent commit.

Additionally, you may try to move your previous R script (from **Item 3**) into this project. You can then “commit” and “push” to upload your script to your git account.