ELEC IT-E1

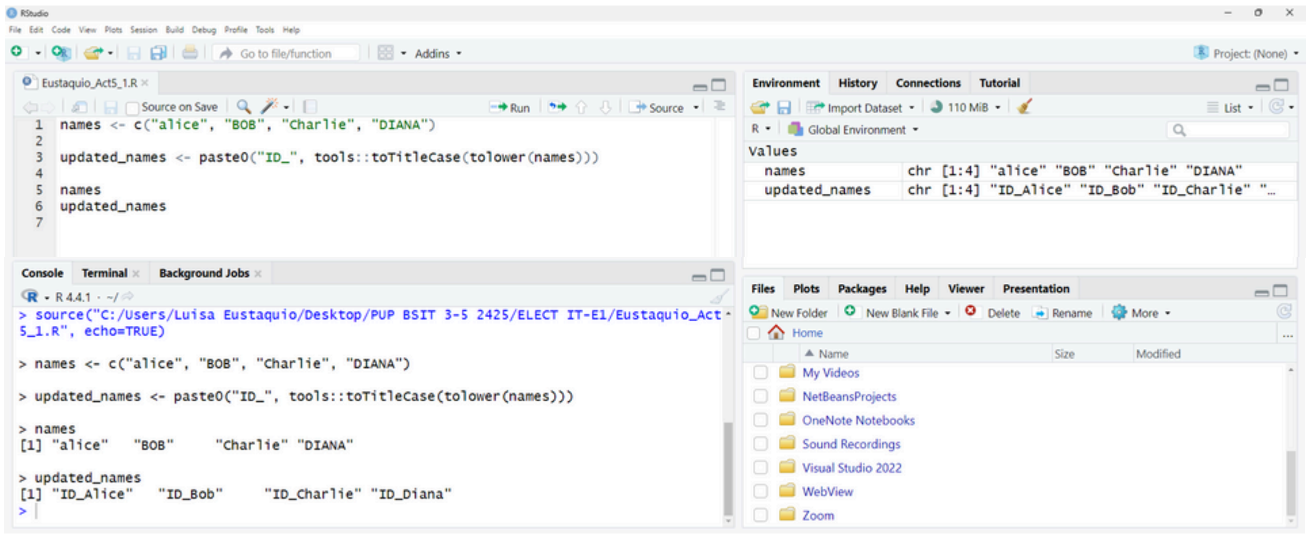
IT ELECTIVE 1

Name: Eustaquio, Maria Luisa O.	**Year & Section:** BSIT 3-5	**Date:** November 18, 2024

ACTIVITY #5 - USING BUILT-IN FUNCTIONS AND CONTROL STRUCTURES

Instructions: Explore R's built-in functions and control structures. Provide R code and a brief explanation for each task. Test your solutions in an R environment.

- Given a vector of names c("alice", "BOB", "Charlie", "DIANA"), write code to: Convert all names to title case (e.g., "Alice", "Bob"). Add a prefix "ID_" to each name, resulting in names like "ID_Alice".



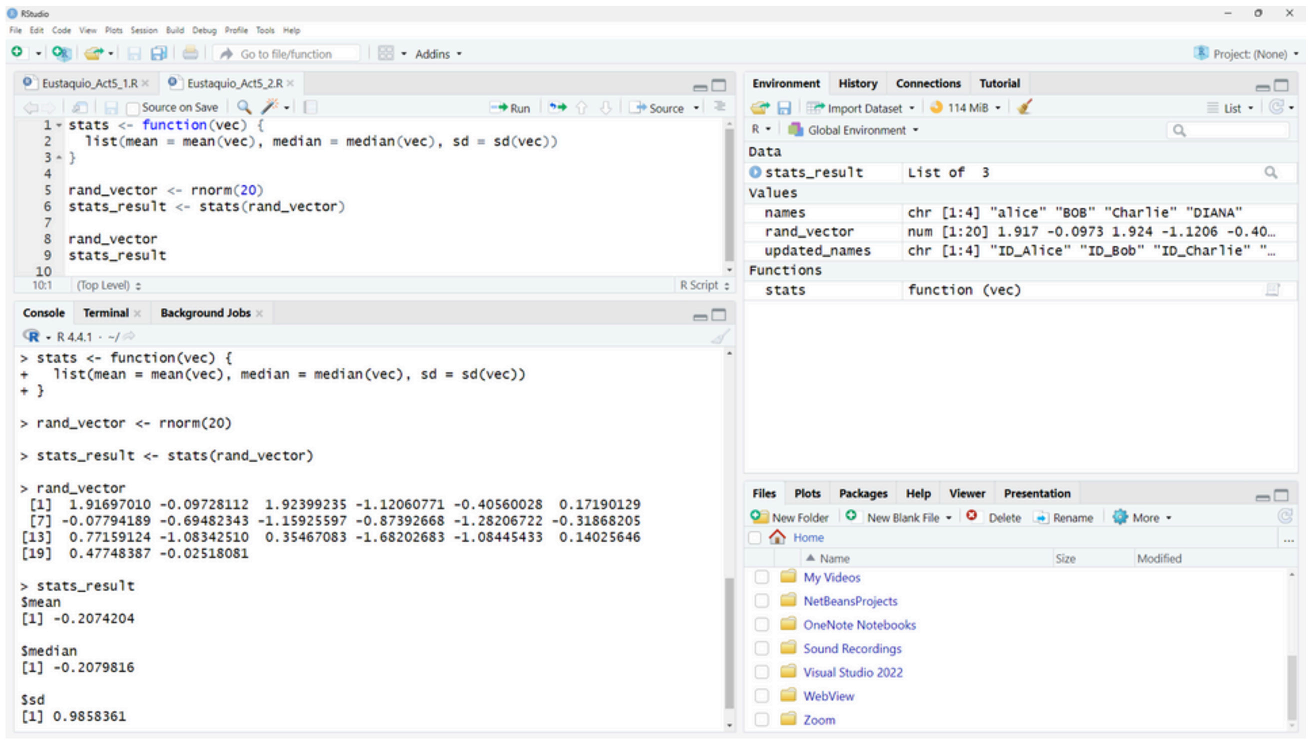
```
names <- c("alice", "BOB", "Charlie", "DIANA")

updated_names <- paste0("ID_", tools::toTitleCase(tolower(names)))

names
updated_names
```

```
> source("C:/Users/Luisa Eustaquio/Desktop/PUP BSIT 3-5 2425/ELECT IT-E1/Eustaquio_Act
5_1.R", echo=TRUE)

> names <- c("alice", "BOB", "Charlie", "DIANA")

> updated_names <- paste0("ID_", tools::toTitleCase(tolower(names)))

> names
[1] "alice"   "BOB"     "Charlie" "DIANA"

> updated_names
[1] "ID_Alice"   "ID_Bob"     "ID_Charlie" "ID_Diana"
> 
```

```
Values
  names          chr [1:4] "alice" "BOB" "Charlie" "DIANA"
  updated_names  chr [1:4] "ID_Alice" "ID_Bob" "ID_Charlie" "…
```

The code starts with a list of names in the **names vector**, some in uppercase and some in lowercase. It converts all the names to lowercase using **tolower()**, then changes them to title case (capitalizing only the first letter) using **tools::toTitleCase()**. Finally, it adds the prefix "ID_" to each name using **paste0()**. The original names vector stays the same, while the **updated_names vector** becomes c("ID_Alice", "ID_Bob", "ID_Charlie", "ID_Diana").

- Write an R function that takes a numeric vector as input and returns a list with the following: Mean, Median, Standard Deviation Test this function with a random vector of 20 values generated using rnorm()



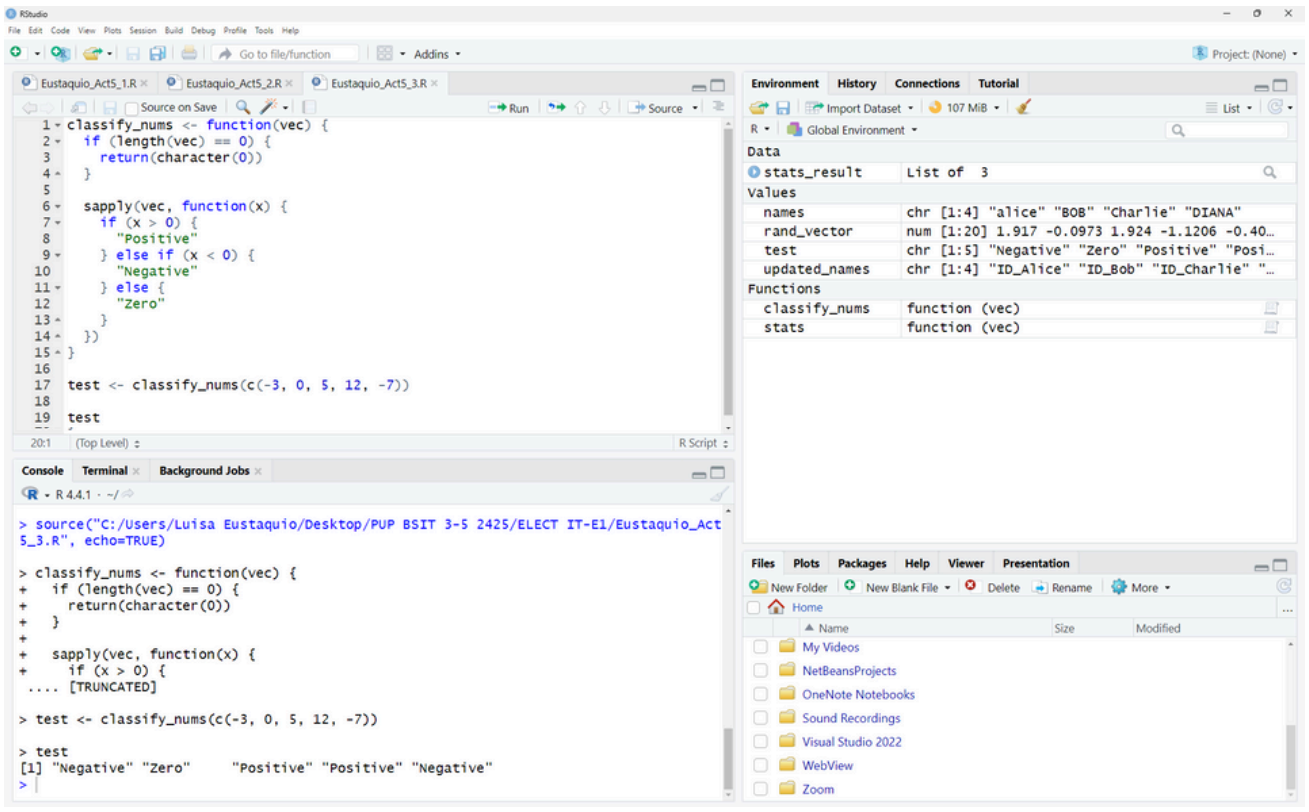
```
stats <- function(vec) {
  list(mean = mean(vec), median = median(vec), sd = sd(vec))
}

rand_vector <- rnorm(20)
stats_result <- stats(rand_vector)

rand_vector
stats_result
```

```
> stats <- function(vec) {
+   list(mean = mean(vec), median = median(vec), sd = sd(vec))
+ }

> rand_vector <- rnorm(20)

> stats_result <- stats(rand_vector)

> rand_vector
 [1]  1.91697010 -0.09728112  1.92399235 -1.12060771 -0.40560028  0.17190129
 [7] -0.07794189 -0.69482343 -1.15925597 -0.87392668 -1.28206722 -0.31868205
[13]  0.77159124 -1.08342510  0.35467083 -1.68202683 -1.08445433  0.14025646
[19]  0.47748387 -0.02518081

> stats_result
$mean
[1] -0.2074204

$median
[1] -0.2079816

$sd
[1] 0.9858361
```

```
Data
  stats_result   List of 3
Values
  names          chr [1:4] "alice" "BOB" "Charlie" "DIANA"
  rand_vector    num [1:20] 1.917 -0.0973 1.924 -1.1206 -0.40…
  updated_names  chr [1:4] "ID_Alice" "ID_Bob" "ID_Charlie" "…
Functions
  stats          function (vec)
```

The **stats function** takes a numeric vector as input and uses the **mean()**, **median()**, and **sd() functions** to calculate its respective mean, median, and standard deviation. The results are returned as a list using the **list() function**, which organizes these three values together. A random vector of 20 numbers is generated using **rnorm(20)** and passed to the stats function, which then outputs the calculations as a list, such as list(mean = ..., median = ..., sd = ...). The exact values will vary because the numbers are randomly generated.

- Create a script that takes a numeric vector and classifies each number into: "Positive" if greater than 0 "Negative" if less than 0 "Zero" otherwise. Ensure the script handles edge cases (e.g., empty vector).



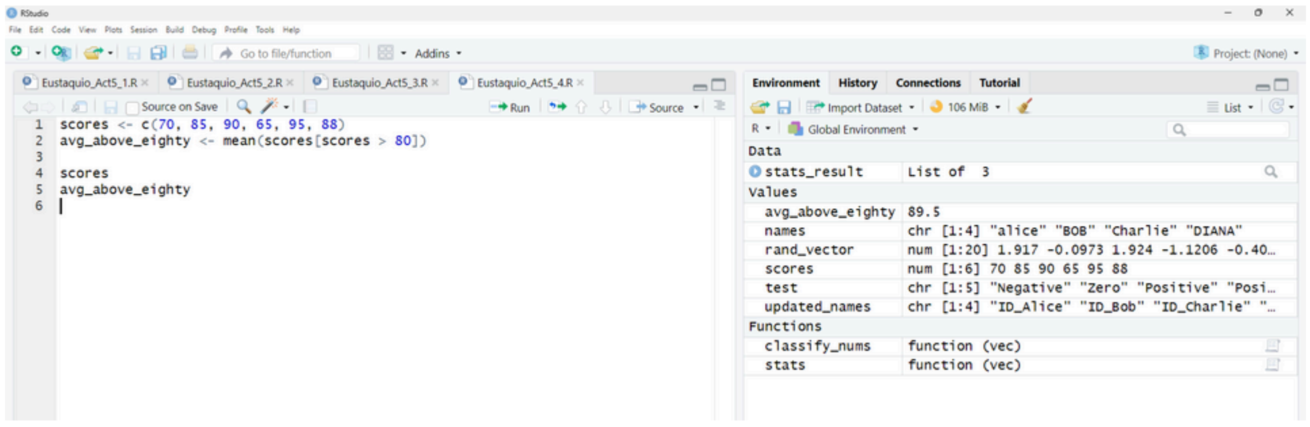
```r
classify_nums <- function(vec) {
  if (length(vec) == 0) {
    return(character(0))
  }

  sapply(vec, function(x) {
    if (x > 0) {
      "Positive"
    } else if (x < 0) {
      "Negative"
    } else {
      "Zero"
    }
  })
}

test <- classify_nums(c(-3, 0, 5, 12, -7))

test
```

```
> source("C:/Users/Luisa Eustaquio/Desktop/PUP BSIT 3-5 2425/ELECT IT-E1/Eustaquio_Act
5_3.R", echo=TRUE)

> classify_nums <- function(vec) {
+   if (length(vec) == 0) {
+     return(character(0))
+   }
+
+   sapply(vec, function(x) {
+     if (x > 0) {
 .... [TRUNCATED]

> test <- classify_nums(c(-3, 0, 5, 12, -7))

> test
[1] "Negative" "Zero"     "Positive" "Positive" "Negative"
>
```

```
Data
stats_result     List of  3
Values
  names          chr [1:4] "alice" "BOB" "Charlie" "DIANA"
  rand_vector    num [1:20] 1.917 -0.0973 1.924 -1.1206 -0.40…
  test           chr [1:5] "Negative" "Zero" "Positive" "Posi…
  updated_names  chr [1:4] "ID_Alice" "ID_Bob" "ID_Charlie" "…
Functions
  classify_nums  function (vec)
  stats          function (vec)
```

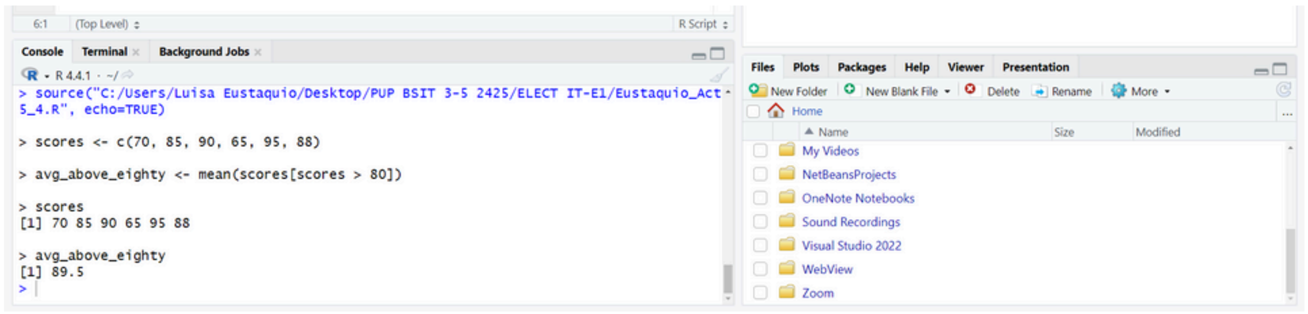
The **function classify_nums** takes a numeric vector and classifies each number as "Positive," "Negative," or "Zero." It first checks if the vector is empty using **length(vec) == 0**. If the vector is empty, it returns an empty character vector. Otherwise, it uses **sapply()** to iterate over each element of the vector and applies a **conditional check**. Numbers greater than 0 are labeled "Positive," less than 0 are labeled "Negative," and equal to 0 are labeled "Zero." For the input c(−3, 0, 5, 12, −7), the function returns a **character vector**: c("Negative", "Zero", "Positive", "Positive", "Negative").

- Given a vector of scores c(70, 85, 90, 65, 95, 88), write code to calculate the average score only for scores above 80.



```r
scores <- c(70, 85, 90, 65, 95, 88)
avg_above_eighty <- mean(scores[scores > 80])

scores
avg_above_eighty
```

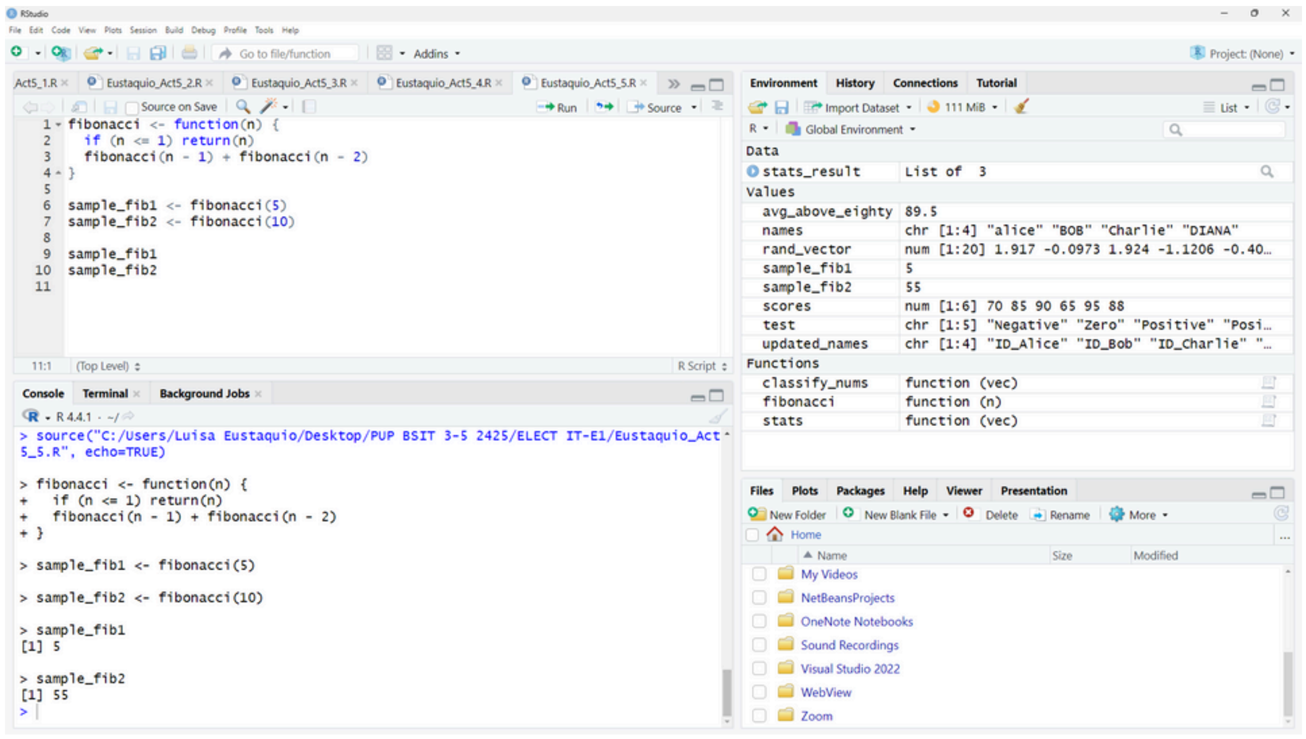
```
Data
stats_result      List of  3
Values
  avg_above_eighty 89.5
  names            chr [1:4] "alice" "BOB" "Charlie" "DIANA"
  rand_vector      num [1:20] 1.917 -0.0973 1.924 -1.1206 -0.40…
  scores           num [1:6] 70 85 90 65 95 88
  test             chr [1:5] "Negative" "Zero" "Positive" "Posi…
  updated_names    chr [1:4] "ID_Alice" "ID_Bob" "ID_Charlie" "…
Functions
  classify_nums    function (vec)
  stats            function (vec)
```



```
> source("C:/Users/Luisa Eustaquio/Desktop/PUP BSIT 3-5 2425/ELECT IT-E1/Eustaquio_Act
5_4.R", echo=TRUE)

> scores <- c(70, 85, 90, 65, 95, 88)

> avg_above_eighty <- mean(scores[scores > 80])

> scores
[1] 70 85 90 65 95 88

> avg_above_eighty
[1] 89.5
>
```

The code starts with a vector scores containing a list of numeric values. It uses a **condition scores > 80** to filter only the scores greater than 80, resulting in a **subset of the vector: c(85, 90, 95, 88)**. The **mean()** function then calculates the average of this subset. The original scores vector remains unchanged, and the calculated avg_above_eighty is 89.5, representing the average of the values above 80.

- Write a recursive R function to compute the nth Fibonacci number. Demonstrate the function with inputs n = 5 and n = 10.



```
fibonacci <- function(n) {
  if (n <= 1) return(n)
  fibonacci(n - 1) + fibonacci(n - 2)
}

sample_fib1 <- fibonacci(5)
sample_fib2 <- fibonacci(10)

sample_fib1
sample_fib2
```

```
> source("C:/Users/Luisa Eustaquio/Desktop/PUP BSIT 3-5 2425/ELECT IT-E1/Eustaquio_Act
5_5.R", echo=TRUE)

> fibonacci <- function(n) {
+   if (n <= 1) return(n)
+   fibonacci(n - 1) + fibonacci(n - 2)
+ }

> sample_fib1 <- fibonacci(5)

> sample_fib2 <- fibonacci(10)

> sample_fib1
[1] 5

> sample_fib2
[1] 55
>
```

```
Data
stats_result      List of  3
Values
  avg_above_eighty  89.5
  names             chr [1:4] "alice" "BOB" "Charlie" "DIANA"
  rand_vector       num [1:20] 1.917 -0.0973 1.924 -1.1206 -0.40…
  sample_fib1       5
  sample_fib2       55
  scores            num [1:6] 70 85 90 65 95 88
  test              chr [1:5] "Negative" "Zero" "Positive" "Posi…
  updated_names     chr [1:4] "ID_Alice" "ID_Bob" "ID_Charlie" "…
Functions
  classify_nums     function (vec)
  fibonacci         function (n)
  stats             function (vec)
```

The **fibonacci function** calculates the nth Fibonacci number using **recursion**. If the input n is less than or equal to 1, it directly returns n. Otherwise, it calculates the Fibonacci number by summing the results of the function called with **n - 1** and **n - 2**. For fibonacci(5), the result is 5 (the fifth Fibonacci number in the sequence 0, 1, 1, 2, 3, 5). For fibonacci(10), the result is 55 (the tenth Fibonacci number in the sequence).