# Lab 1- Introduction to RStudio, R operations and data exploration

## Learning Outcomes

At the end of the session, you will be able to:

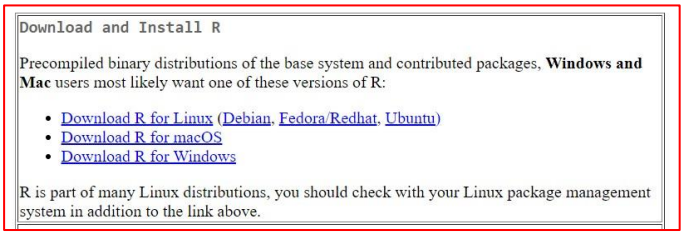
- Install, write, and run R program using RStudio.
- Explain the basic R syntax, variables, and mathematical operations.
- Explore and visualize a dataset using basic R code.

## Activity 1 – Introduction to R

1. *Using RStudio*

1.1. Step 1: Install R

- Download the R installer from https://cran.r-project.org/




Note: Example of installer for Windows setting: https://cran.r-project.org/bin/windows/base/

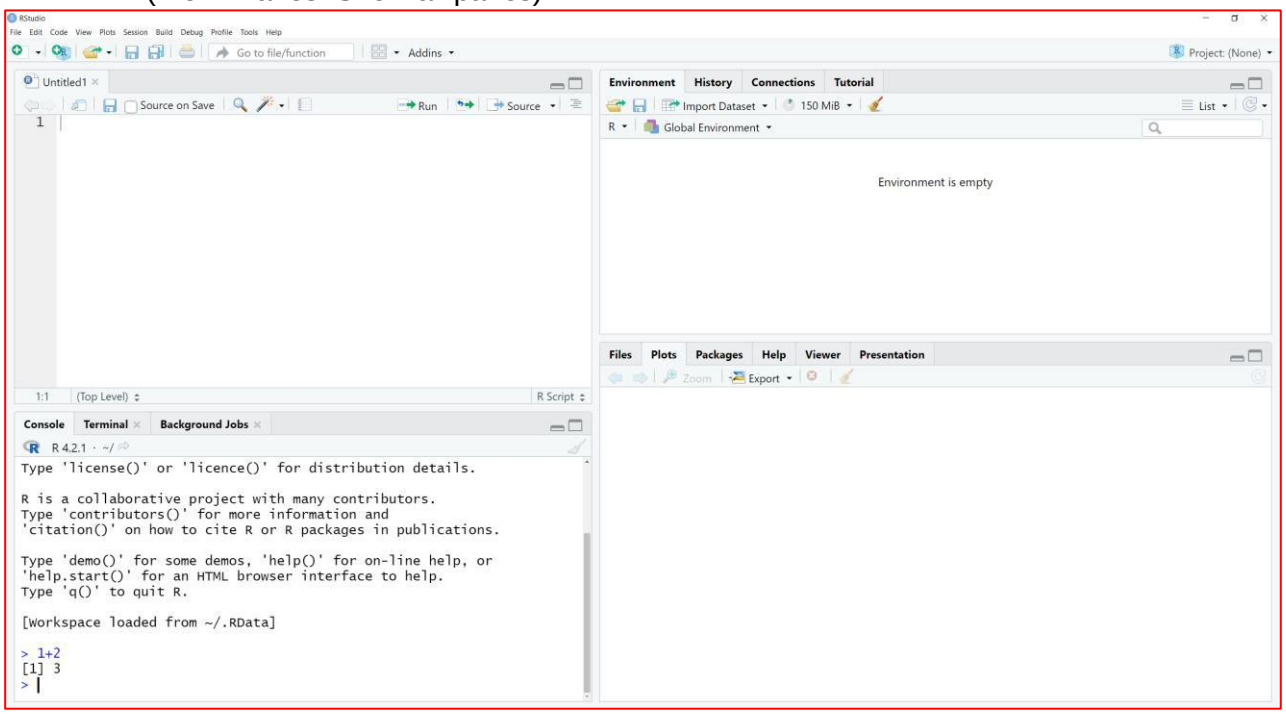
- Run the installer. Default settings are fine. (Make sure this step is done before proceeding to the next step)

1.2. Step 2: Install RStudio

- Download RStudio Desktop installer from https://www.rstudio.com/products/rstudio/download/
- Once the installation of R (step 1) has completed successfully (and not before), run the RStudio Desktop installer.

1.3. Step 3: Check that R and RStudio are working

- Open RStudio. It should open a window that looks similar to image below (View>Panes>Show all panes).

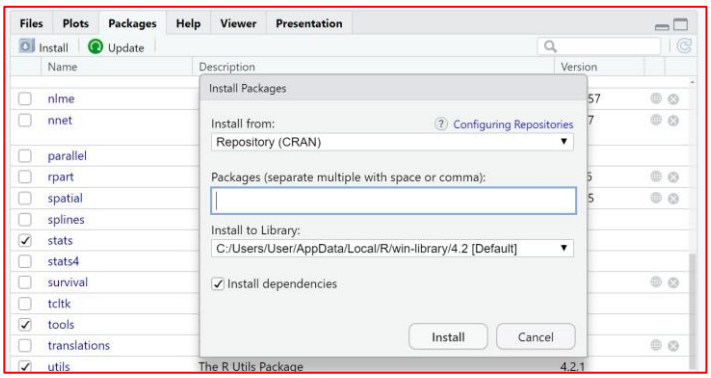



- Type '1+2' and hit enter. An output should appear. This means that R and RStudio are working.

- Differentiate the usage of R Script and R Console.

1.4. Step 4: Installation of R packages (Optional)

- Click on tab Packages>Install or Tools>Install, type in the package you want to install.




- Or type in Console *install.packages("<the package's name>")* tab.
- To use the packages type in Console `library`*("<the package's name>")*
- Find out what are the common packages in R for Data Science.

## 2. *Syntax, Variables and Operations*

2.1. Simple R Code

- Write and run the following in R Console:

```
2+3
print(“Hello World!”)
```

- Write and run the following code in R Script named test.R:

```
# My first program in R Programming
myString <- "Hello, World!"
print (myString)
```

2.2. R Variables

- Write and run the following in R Console. Make your conclusion about the code:

```
var.1 = 5
var_1 = 7
x = 1
print(ls())
print(ls(pattern=“var”))
```

2.3. Assignment Operations

| Operator | Description |
|---|---|
| <-, <<-, = | Leftwards assignment |
| ->, ->> | Rightwards assignment |

- Using the different leftwards/rightwards assignment operator in table above and run the following in R Console. Make your conclusion about the code:

| Variable | Value | R Syntax |
|---|---|---|
| Gender | Female | Gender<-“Female” |
| height | 152 | height<<-152 |

| | | |
|---|---|---|
| `Weight` | `81` | `Weight=81` |
| `f` | `3` | `3 -> f` |
| `x` | `23.5` | `23.5->> x` |
| `b` | `0 1 2 3 4 5` | `b <- seq(from = 0, to = 5)` |
| `c` | `0 2 4 6 8 10` | `c <- seq(from = 0, to = 10, by = 2)` |
| `v` | `2L` | `v = 2L` |
| `w` | `2+5i` | `w <<-2+5i` |
| `a` | `48 65 6c 6c 6f` | `a<-charToRaw("Hello")` |

- You can always check the class of an object by calling the function `class()`:

```
print(class(Gender))
print(class(height))
print(class(f))
print(class(x))
print(class(b))
print(class(v))
print(class(w))
```

2.4. Arithmetic Operations

| Operator | Description |
|---|---|
| + | Addition |
| – | Subtraction |
| * | Multiplication |
| ** or ^ | Exponentiation |
| / | Division |
| %% | Modulus (Remainder from division) |
| %/% | Integer Division |

- Using the arithmetic operators and reuse the variable assignment and run the following in R Console. Make your conclusion about the code:

```
print(f+3)
print(height-x)
print(Weight*2)
print(b**2)
print(c^5)
m = height/100
print(Weight/(m**2))
BMI = Weight/(m**2)
print(b%%2)
print(c%/%2)
```

2.5. User Input

- Write and run the following in R Script:

```
name <- readline(prompt="Enter name: ")
age <- readline(prompt="Enter age: ")
# convert character into numeric
age <- as.numeric(age)
print(paste("Hi,", name, "this year you are", age, "years old."))
```

2.6. (i) Given two variables, num1=0.956786 and num2=7.8345901. Write a R code to display the num1 value in 2 decimal point number, and num2 value in 3 decimal point number (clue: use function round()).

(ii) Write R code to create a sequence of 20 numbers. The program will calculate and display the square of the number sequence

2.7. Extra

- Write and run the following in R Console. Make your conclusion about the code:

```
?paste
demo(graphics)
demo(image)
```

## Activity 2 – Basic data exploration

1. Go to Kaggle.com and download some historical data in CSV file. You may start with a sample data provided, seattle-weather.csv.

   Read the file into R as follows:

```
> data = read . csv ( " seattle-weather.csv" , header=TRUE)
> n = dim( data ) [ 1 ]
> n
```

   Observe the output. What is n value?
   Proceed with another line of command.

```
> data = data [n : 1 , ]
> data
```

   Observe the output and the changes. Is it a dataframe?

   A data frame is a rectangular collection of variables (in the columns) and observations (in the rows).

2. We will use matrices extensively in modeling, and here we examine the basic commands needed to create and manipulate matrices in R. We create a 4 x 3 matrix with random numbers as follows:

```
> x = matrix ( rnorm( 1 2 ) , 4 , 3 )
> x
[ , 1 ] [ , 2 ] [ , 3 ]
[ 1 , ] 0.0625034 0.9256896 2.3989183
[ 2 , ] 0.5371860 0.7497727 0.0857688
[ 3 , ] 1.0416409 1.6175885 3.3755593
[ 4 , ] 0.3244804 0.1228325 1.6494255
```

Transposing the matrix, notice that the dimensions are reversed:

```
> print ( t ( x ) , 3 )
```

Observe the output.
Produce a matrix based on the selected columns from seattle-weather.csv.

3. R has several systems for making graphs, but **ggplot2** is one of the most elegant and most versatile. **ggplot2** implements the grammar of graphics, a coherent system for describing and building graphs. With ggplot2, you can do more faster by learning one system and applying it in many places.

   Load the tidyverse by running this code:

   ```
   install.packages("tidyverse")
   library(tidyverse)
   ```

   You only need to install a package once, but you need to reload it every time you start a new session.

   Creating a ggplot. To plot weather, run this code to put *precipitation* on the x-axis and *wind* on the y-axis:

   ```
   ggplot(data) +
     geom_point(mapping = aes(x = precipitation, y = wind))
   ```

   Based on the plot, observe the relationship between precipitation and wind.

   **Exercise:**

   - Run ggplot(data = seattle-weather). What do you see?
   - How many rows are in seattle-weather? How many columns?
   - What does the *precipitation* variable describe?
   - Make a scatterplot of *temp_min* vs *wind*.
   - What happens if you make a scatterplot of *weather* vs *temp_max*? Why is the plot not useful?

You can convey information about your data by mapping the aesthetics in your plot to the variables in your dataset. For example, you can map the colors of your points to the weather variable to reveal the weather of each day.

```
ggplot(data) +
  geom_point(mapping = aes(x = precipitation, y = wind, color = weather))
```

**References:**

https://www.kaggle.com/datasets/ananthr1/weather-prediction
https://r4ds.had.co.nz/
https://rpubs.com/shailesh/mpg-exploration