Introduction to R and RStudio

Mohan Khanal

2025-07-29

Table of contents

---

1 Introduction to R and RStudio

This 2-hour online session introduces R and RStudio: how to install them, navigate the interface, and use packages and libraries. It includes demos and interactive practice.

1.1 Agenda

- **Part 1 (30 mins)** – Installation and Setup
- **Part 2 (20 mins)** – RStudio Interface Tour
- **Part 3 (20 mins)** – Basic R Commands
- **Part 4 (40 mins)** – R Packages and Libraries
- **Part 5 (10 mins)** – Practice and Q&A

1.2 Part 1: Installation and Setup (30 mins)

1.2.1 Installing R

- Go to https://cran.r-project.org
- Choose your OS: Windows, macOS, or Linux
- Download and install the latest version of R

1.2.2 Installing RStudio

- Go to https://posit.co/download/rstudio-desktop/
- Choose the **RStudio Desktop (Free)** version
- Install **RStudio after installing R**

> Tip: R is the programming language. RStudio is the environment that helps you work with R more easily.

1.3 Part 2: RStudio Interface Tour (20 mins)

When you open RStudio, you'll see four main panels:

1. **Source (Top-Left)** – Script Editor (`.R`, `.Rmd`)
2. **Console (Bottom-Left)** – Command Execution
3. **Environment/History (Top-Right)** – Lists variables and past commands
4. **Files/Plots/Packages/Help/Viewer (Bottom-Right)** – File manager, plot viewer, documentation, etc.

1.3.1 Live Demo

- Create an R script (`.R`)
- Type and run:

```
print("Hello, R world!")
```

```
[1] "Hello, R world!"
```

To output texts in R use single or double quotes

Similarly, to output numbers just type the numbers(without quotes)

```
1
```

```
[1] 1
```

```
2
```

```
[1] 2
```

```
3
```

```
[1] 3
```

To do simple calculations, add numbers together

```
1+2
```

```
[1] 3
```

1.4 Part 3: Basic R Concepts and Commands (20 mins)

1.4.1 Vector

sequence of data elements of the same basic type. Same thing as array in c or c++ or any other language. eg : 1, 3, 4, 5, ... 10.

- It is ordered set - will always have a beginning or end.

e.g. "Z" "f" "7" "yes" "A" "Ab"

- even if we try to put in a 7 as a number into a character vector R will automatically change it into a character.
- of course it has to have quotation marks.

1.4.2 Data type

- numeric: 7.2, 67, 711
- integer: 1L, 99L, 1000L (L declares this as an integer)
- complex: 7 + 12i, where "i" is the imaginary part
- character (a.k.a. string): "k", "R is exciting", "FALSE", "11.5"
- logical (a.k.a. boolean): TRUE or FALSE

```
# numeric
y <- 7.2
class(y)
```

```
[1] "numeric"
```

```
# integer
y <- 500L
class(y)
```

```
[1] "integer"
```

```
# complex
y <- 4i + 7
class(y)
```

```
[1] "complex"
```

```r
# character/string
y <- "Learning R is fun"
class(y)
```

```
[1] "character"
```

```r
# logical/boolean
y <- FALSE
class(y)
```

```
[1] "logical"
```

In R programming the numeric data type encompasses both integer and double (double-precision floating-point numbers).

1.4.3 Variables and Math

```r
x <- 10
y <- 5
x + y       # Addition
```

```
[1] 15
```

```r
x * y       # Multiplication
```

```
[1] 50
```

1.4.4 Using Functions

```r
mean(c(1, 2, 3, 4, 5))
```

```
[1] 3
```

```r
sum(1:10)
```

```
[1] 55
```

1.4.5 Creating Vector

- seq() - sequence e.g. seq(1,15) = 1: 15 = 1,2,,3,4,5,..., 15, seq(1,15,2) = 1,3,5,7,9,...15
- rep()- rep(3,50) = 3 50 times , we can replicate a character vector as well.
- rnorm() - used to create numeric vector containing random numbers drawn from a normal distribution - rnorm(10) - creates 10 random numbers from standard normal distribution.

1.4.6 Using Square[] bracket

```r
w <- c("a", "b", "c","d")
w
```

```
[1] "a" "b" "c" "d"
```

```r
w[1] #acess first - i.e 1
```

```
[1] "a"
```

```r
w[-3] # access except the third one
```

```
[1] "a" "b" "d"
```

1.4.7 Help System

```r
#?mean
#help("sum")
```

1.5 Part 4: Packages and Libraries (40 mins)

1.5.1 What are Packages?

Packages are collections of R functions, data, and compiled code in a well defined format. The directory where packages are stored is called the library.

Packages extend R's core functionality — like `ggplot2` for plotting, or `dplyr` for data manipulation.

1.5.2 Installing Packages

```
#install.packages("ggplot2")
#install.packages("dplyr")
#
```

1.5.3 Loading a Package

```
library(ggplot2)
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

using quotation mark is optional

1.5.4 Data Structure

** Vector: list of items

```r
# vector of fruits
fruits <- c("banana","apple", "orange")

#print fruits
fruits
```

```
[1] "banana" "apple"  "orange"
```

```r
#Vector length
length(fruits)
```

```
[1] 3
```

** Lists : collection of data which is ordered and changeable. To create a list, use the list() funciton:

```r
names <- list("apple","ball","cat")

#print the list
names
```

```
[[1]]
[1] "apple"

[[2]]
[1] "ball"

[[3]]
[1] "cat"
```

```r
# Append the list
append(names, "dog")
```

```
[[1]]
[1] "apple"

[[2]]
[1] "ball"
```

```
[[3]]
[1] "cat"

[[4]]
[1] "dog"
```

Notes : important codes
#dim() #class() #attribute()
#rm(list=ls()) #Ctrl+L = clear console #
Ctrl + + = zoom in - = Zoom out #view()
= in new tab ** Matrices : two dimensional
data set with columns and rows
::: {.cell}
“‘{.r .cell-code} # Create a matrix
firstmatrix <- matrix(c(1,2,3,4,5,6), nrow
= 3, ncol = 2)
Print the matrix firstmatrix “‘
::: {.cell-output .cell-output-stdout}
```
[,1] [,2] [1,]    1    4 [2,]    2
5 [3,]    3    6
```
::: :::
*****Note: c() function is used to
concatenate items together.
** DataFrames in R - It is used for storing
data tables. It can contain multiple data
types in multiple columns called fields. It
is a generalized form of a matrix. It is like
a table in excel sheets. It has column and
row names. The name of rows are unique
with no empty columns. The data stored
must be numeric, character or factor type.
DataFrames are heterogeneous. Example:
::: {.cell}
“‘{.r .cell-code} # creating company data
frame comp.data <- data.frame(#
company ids comp_id = c(1:3), # company
names comp_name = c(“XYZ”, “ABC”,
“KLM”), growth = c(16000, 14000, 12000),
company start dates comp_start_date
= as.Date(c(“02/05/25”, “04/04/25”,
“05/03/25”), format = “%m/%d/%y”))
print the data frame print(comp.data) “‘

::: {.cell-output .cell-output-stdout}
```
comp_id comp_name growth
comp_start_date 1       1       XYZ
16000      2025-02-05 2       2
ABC  14000      2025-04-04 3
3       KLM  12000      2025-05-03
```
::: :::

Exploring Data

::: {.cell}
```
{.r .cell-code} dat<-
datasets::iris head(dat)
```
::: {.cell-output .cell-output-stdout}
```
Sepal.Length Sepal.Width
Petal.Length Petal.Width Species 1
5.1          3.5           1.4
0.2  setosa 2           4.9
3.0           1.4          0.2
setosa 3           4.7         3.2
1.3          0.2  setosa 4
4.6          3.1           1.5
0.2  setosa 5           5.0
3.6           1.4          0.2
setosa 6           5.4         3.9
1.7          0.4  setosa
```
::: :::

::: {.cell}
```
{.r .cell-code} names(dat)
```
::: {.cell-output .cell-output-stdout}
```
[1] "Sepal.Length" "Sepal.Width"
"Petal.Length" "Petal.Width"
"Species"
```
:::

```
{.r .cell-code} #or colnames(dat)
```
::: {.cell-output .cell-output-stdout}
```
[1] "Sepal.Length" "Sepal.Width"
"Petal.Length" "Petal.Width"
"Species"
```
:::

```
{.r .cell-code} #counting column
ncol(dat)
```
::: {.cell-output .cell-output-stdout}
```
[1] 5
```

::: :::
Piping Concept : when we press % > % (without space) = %>% whose shortcut is ctrl+ shit +M (Windows) it help to run code of the right side to the left side file.
Calling an variable or column : We use $ like : df$var1