1. Introduction to R

Emmanuel Masavo DJEGOU

2025-04-06

Contents

1 Launching R

R can be run in two modes: **interactive** and **batch**. Here's a brief overview of both:

1.1 Interactive Mode

In interactive mode, you can run commands directly, and R will display results immediately in the console. This mode is useful for experimenting and quickly testing your ideas.

You can launch an R session in the following ways:

- On Linux or macOS, open a terminal and type `R`, then press Enter.
- On Windows, start R by double-clicking the R shortcut icon on your desktop or in the Start menu.

```
# Example of Interactive Mode: Creating a numeric vector and calculating its mean
y_data <- abs(rnorm(100))
mean(y_data)  # Output shown directly in the console
```

```
## [1] 0.8201506
```

The code generates 100 random values, takes their absolute values, and computes the mean. The `[1]` in the output shows the position of the first item in the line—useful for reading long outputs, where each line is numbered by its starting item.

R commands can be saved in a file, usually with a `.R` or `.r` extension. To run the code in a file like `Script.R`, use the command:

```
source("Script.R")
```

```
## [1] "The mean of y_data is: 0.806438082243052"
```

1.2 Batch Mode

You can automate R scripts by running them in **batch mode**, avoiding manual interaction. For example, save the following code in a file called `Graph-Making.R`:

- pdf("histogram.pdf") # Save the next plot to a PDF file named "histogram.pdf"
- hist(rnorm(100)) # Create a histogram of 100 random values from a standard normal distribution
- dev.off() # Finish and close the PDF file

Run the script from the command line with:

```
R CMD BATCH Graph-Making.R
```

2 Exploring Your First R Session

In this section, we will explore how to create and manipulate vectors, compute summary statistics, and visualize data using R.

```
# Creating a numeric vector
x <- c(1, 2, 4)

# Creating a new vector by repeating x and adding 8
q <- c(x, x, 8)

# Accessing the third element of x
print(x[3])
```

```
## [1] 4
```

```
# Subsetting: extracting the first two elements of x
subset_x <- x[1:2]
print(subset_x)
```

```
## [1] 1 2
```

```
# Calculating the mean of x
mean_x <- mean(x)
print(mean_x)
```

```
## [1] 2.333333
```

```
# Calculating the standard deviation of x
std_x <- sd(x)
print(std_x)
```

```
## [1] 1.527525
```

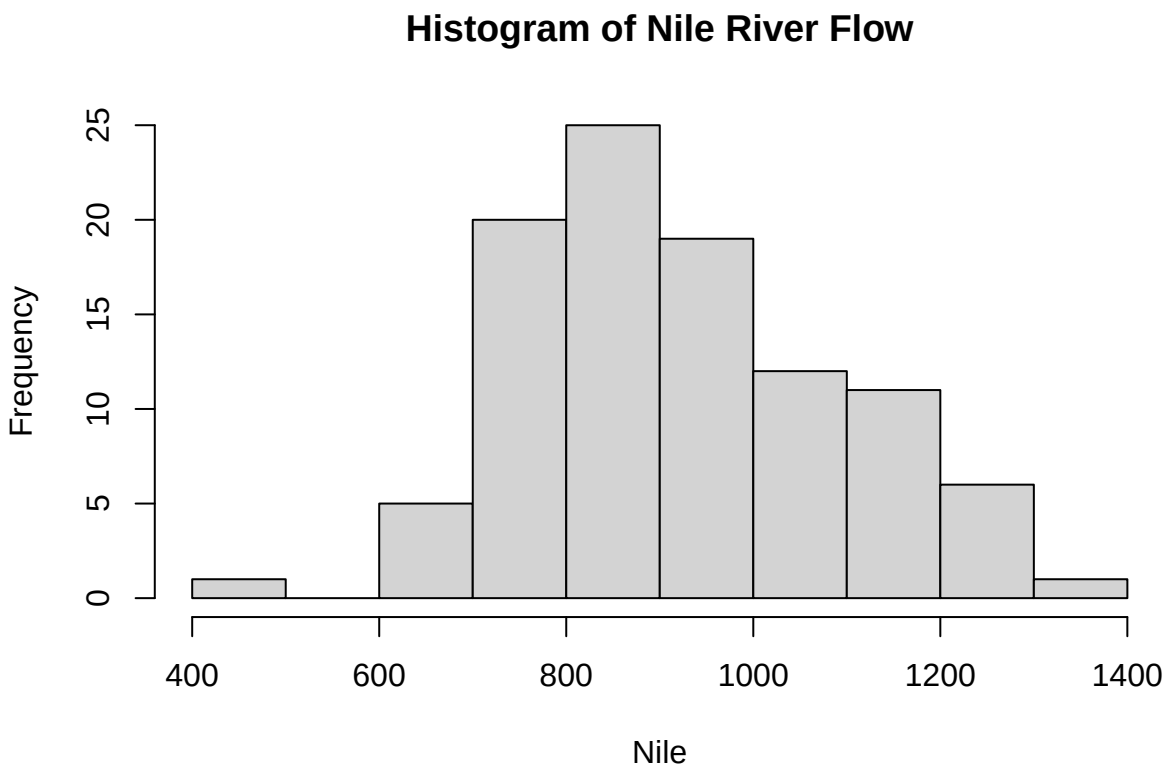
```
# Listing available datasets
data()

# Working with the "Nile" dataset
print(mean(Nile))      # Mean of the dataset
```

```
## [1] 919.35
```

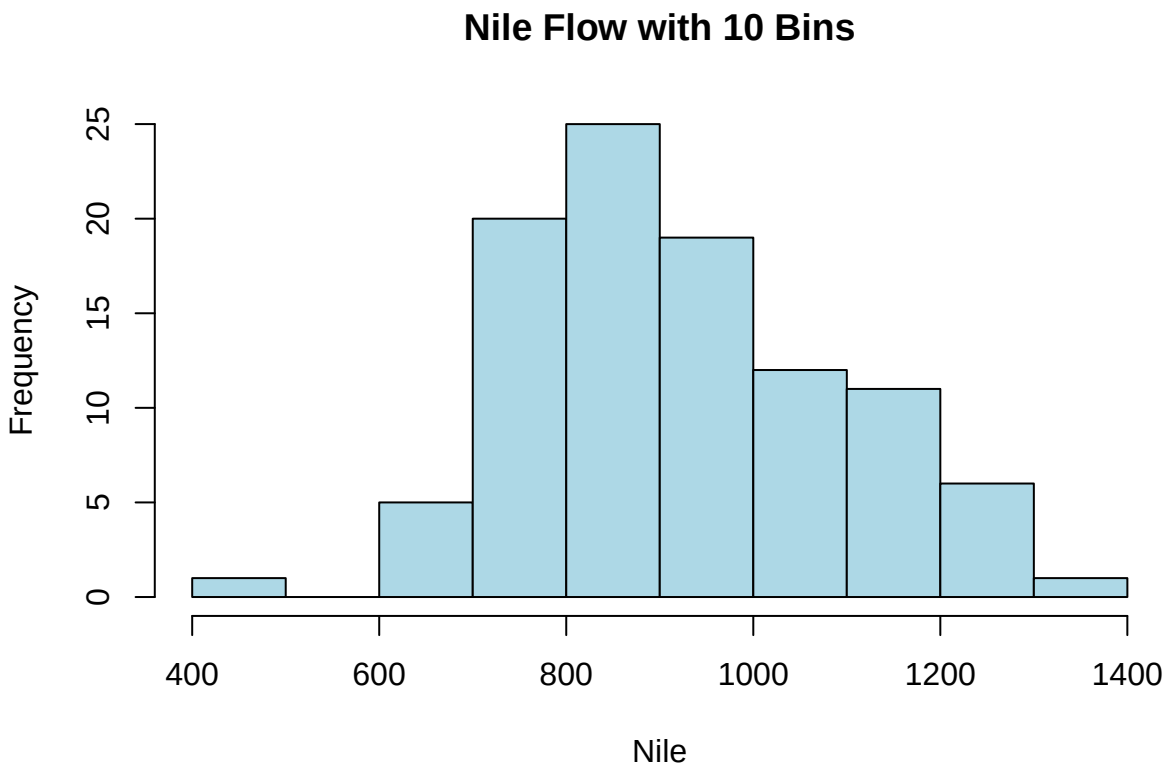
```
print(sd(Nile))        # Standard deviation
```

```
## [1] 169.2275
```

```r
# Plotting histogram
hist(Nile, main = "Histogram of Nile River Flow")
```




```r
hist(Nile, breaks = 10, col = "lightblue", main = "Nile Flow with 10 Bins")
```



3 Creating and Using Functions in R

```
# Function to count the number of odd numbers in a vector
oddcount <- function(x) {
     k <- 0
     for (n in x) {
          if (n %% 2 == 1) k <- k + 1
     }
     return(k)
}

# Testing the oddcount function
y <- c(1, 2, 3, 7, 9)
print(oddcount(y))
```

```
## [1] 4
```

```
# Modulo operator example
print(38 %% 7)
```

```
## [1] 3
```

```
# Global variable example
f <- function(x) return(z + y)
z <- 3
print(f(z))
```

```
## [1]  4  5  6 10 12
```

```
# Function with default arguments
g <- function(x, y = 2, z = TRUE) {
     print(paste("x:", x, "| y:", y, "| z:", z))
}

g(12, z = FALSE)
```

```
## [1] "x: 12 | y: 2 | z: FALSE"
```

```
# To quit R (uncomment if needed)
# q()
```

4 Key R Data Structures

```
# Vectors
# A vector in R is a collection of elements that must all have the same data type (mode).

# Scalar = one-element vector of mode numeric
x <- 8         # Scalar, a single numeric value
x              # Display the value of x
```

```
## [1] 8
```

```
# Character String = one-element vector of mode character
x <- c(5, 12, 13)  # Vector of numeric values
x                   # Display the vector
```

```
## [1]  5 12 13
```

```r
length(x)           # Returns the number of elements in the vector (3)
```

```
## [1] 3
```

```r
mode(x)             # Returns the mode of the vector, which is "numeric"
```

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

```r
y <- "cpt"          # Single character string, mode is character
length(y)           # Length of the string (1)
```

```
## [1] 1
```

```r
mode(y)             # Mode is "character"
```

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

```r
z <- c("cpt", "29 88")  # Character vector with two elements
length(z)               # Length of the vector (2)
```

```
## [1] 2
```

```r
mode(z)                 # Mode is "character"
```

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

```r
u <- paste("cpt","opt","f1")  # Concatenate strings into one character string
u                             # Display the concatenated string
```

```
## [1] "cpt opt f1"
```

```r
v <- strsplit(u, " ")  # Split the string into a list based on space
v                      # Display the result (list of substrings)
```

```
## [[1]]
## [1] "cpt" "opt" "f1"
```

```r
# Matrices = A two-dimensional array of numbers (rectangular array)
# Matrices are essentially vectors with two additional attributes: row and column numbers

m <- rbind(c(4, 5), c(0, 1))  # Create a matrix by binding rows
m                             # Display the matrix
```

```
##      [,1] [,2]
## [1,]    4    5
## [2,]    0    1
```

```r
n <- cbind(c(2, 8), c(3, 8))  # Create a matrix by binding columns
n                             # Display the matrix
```

```
##      [,1] [,2]
## [1,]    2    3
## [2,]    8    8
```

```r
# Matrix-multiplication operator
m %*% c(2, 3)  # Perform matrix multiplication with a vector (resulting in a matrix)
```

```
##      [,1]
## [1,]   23
## [2,]    3
```

```r
# Indexing matrices
m[1,1]  # Access element in the 1st row, 1st column of the matrix
```

```
## [1] 4
```

```r
m[2,1]  # Access element in the 2nd row, 1st column of the matrix
```

```
## [1] 0
```

```r
# Extracting Submatrices
m[1,]   # Extract the 1st row of the matrix
```

```
## [1] 4 5
```

```r
m[,2]   # Extract the 2nd column of the matrix
```

```
## [1] 5 1
```

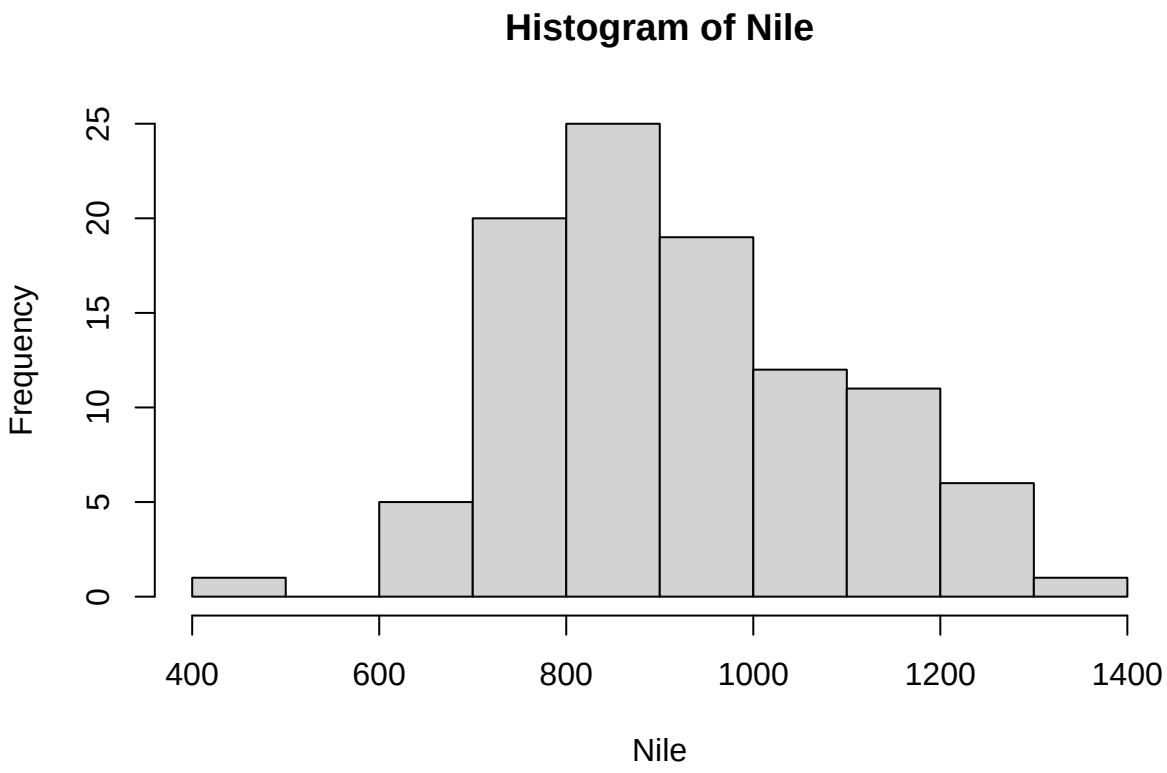
```r
# Lists = Containers that can hold multiple values of different types
x <- list(u=2, v="cpt")  # Create a list with numeric and character elements
x                        # Display the list
```

```
## $u
## [1] 2
##
## $v
## [1] "cpt"
```

```r
x$u                      # Access the value of element 'u' in the list
```

```
## [1] 2
```

```r
# Example of histogram
hist(Nile)               # Create a histogram of the Nile dataset
```



```r
hist_value <- hist(Nile) # Store the histogram object in hist_value
print(hist_value)        # Display the stored histogram object
```

```
## $breaks
##  [1]  400  500  600  700  800  900 1000 1100 1200 1300 1400
##
## $counts
##  [1]  1  0  5 20 25 19 12 11  6  1
##
## $density
##  [1] 0.0001 0.0000 0.0005 0.0020 0.0025 0.0019 0.0012 0.0011 0.0006 0.0001
##
## $mids
##  [1]  450  550  650  750  850  950 1050 1150 1250 1350
##
## $xname
## [1] "Nile"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

```r
# str() function displays the internal structure of any R object
str(hist_value)          # Display the structure of the histogram object
```

```
## List of 6
##  $ breaks  : int [1:11] 400 500 600 700 800 900 1000 1100 1200 1300 ...
##  $ counts  : int [1:10] 1 0 5 20 25 19 12 11 6 1
##  $ density : num [1:10] 0.0001 0 0.0005 0.002 0.0025 0.0019 0.0012 0.0011 0.0006 0.0001
##  $ mids    : num [1:10] 450 550 650 750 850 950 1050 1150 1250 1350
##  $ xname   : chr "Nile"
##  $ equidist: logi TRUE
##  - attr(*, "class")= chr "histogram"
```

```r
# Data Frame = A special type of list where each component is a column vector
# Create a data frame with columns "kids" and "ages"
df <- data.frame(list(kids = c("Easton", "Emma"), ages = c(25, 32)))
df                       # Display the data frame
```

```
##     kids ages
## 1 Easton   25
## 2   Emma   32
```

```r
df$kids                # Access the "kids" column
```

```
## [1] "Easton" "Emma"
```

```r
df$age          # Access the "ages" column (Note: Should be df$ages, not df$age)
```

```
## [1] 25 32
```

R is an object-oriented language, meaning it uses **objects** that belong to **classes**. In R, many objects use **S3 classes**, which are just regular lists with an extra label (the class name).

For example, the result of `hist()` is a list with components like `breaks` and `counts`, and its class is `"histogram"`.

```r
# Printing hist_value
print(hist_value)
```

```
## $breaks
##  [1]  400  500  600  700  800  900 1000 1100 1200 1300 1400
##
## $counts
##  [1]  1  0  5 20 25 19 12 11  6  1
##
## $density
##  [1] 0.0001 0.0000 0.0005 0.0020 0.0025 0.0019 0.0012 0.0011 0.0006 0.0001
##
## $mids
##  [1]  450  550  650  750  850  950 1050 1150 1250 1350
##
## $xname
## [1] "Nile"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

Classes are used because they work with **generic functions** like `summary()` and `plot()`. These functions behave differently depending on the object's class.

- `summary()` gives useful summaries tailored to each type of object (e.g., histogram, regression).
- `plot()` knows how to make the right kind of plot based on the object's class.

In short, **S3 classes and generic functions** help R handle different types of data in a consistent and flexible way.