Introdução à Linguagem de Programação em R

Dia 2

Carina Silva e Ricardo Ribeiro
(*carina.silva@estesl.ipl.pt ricardo.ribeiro@estesl.ipl.pt*)
Escola Superior de Tecnologias da Saúde de Lisboa

Contents

Graphs

Basic Graphs

`lattice` Package

Contents

Graphs

Basic Graphs

`lattice` Package

Functions

Introduction

Iterations and Loops

Basic Graphs

One of the main reasons data analysts turn to R is for its strong graphic capabilities.

In R, graphs are typically created interactively.

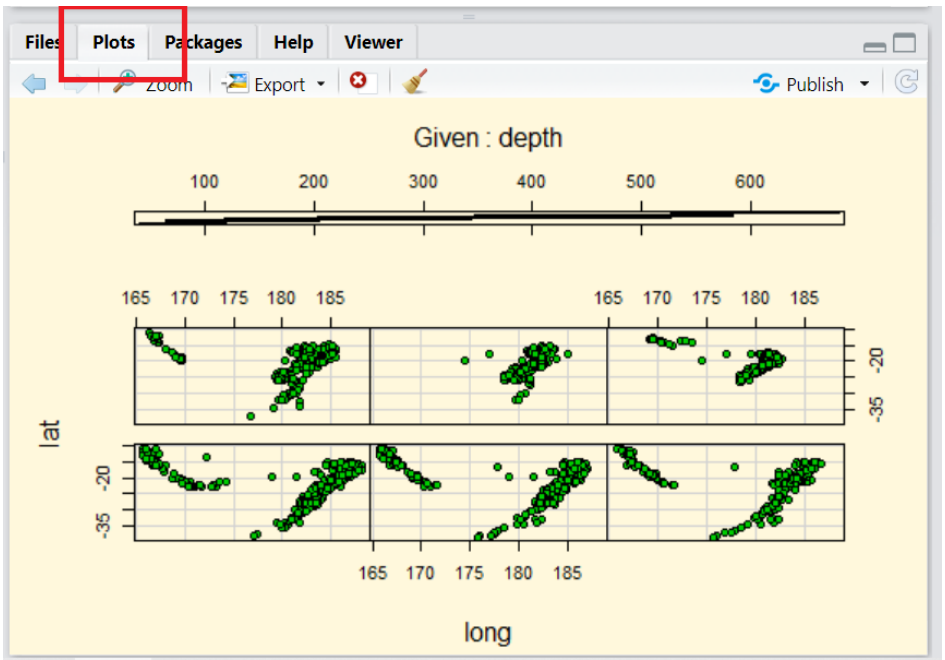
```
# Creating a Graph
>attach(mtcars)
>plot(wt, mpg)
>abline(lm(mpg~wt))
>title("Regression of MPG on Weight")
```

The `plot( )` function opens a graph window and plots weight vs. miles per gallon. The next line of code adds a regression line to this graph. The final line adds a title.

Basic Graphs

You can run the function `>demo(graphics)` to see several options.

When you produce a plot it will be seen in a window called *plots*:



Basic graphs

The function `plot()` is generic. The graph depends on the first argument of the function `plot()`.
Data base: CO2

- `data(CO2)`
- `plot(CO2$Type)`
- `plot(CO2$uptake)`
- `plot(CO2$Type,CO2$conc)`

Graph Functions

Graphics commands can be grouped into three categories:

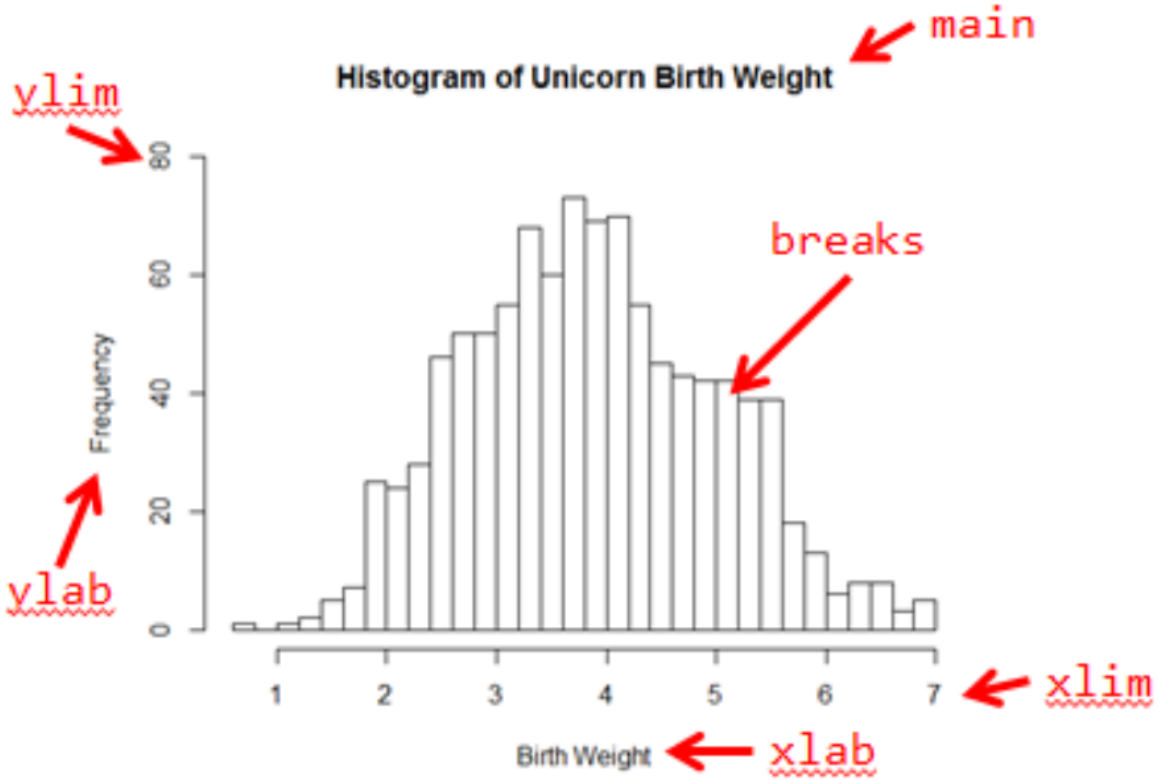
- High-level graphics functions: create new graphics in the graphic window,axes, labels, titles, etc.
- Low-level graphical functions: allow new information to be added to graphs already existing.
- Interactive graphic functions: allow adding or removing information in existing graphics.

Basic Graphs

The next arguments are very used is most graphical functions:

main=""	title
xlab="", ylab=""	variables labels
xlim=,ylim=	specifies the lower and upper limits of the axes
type=""	"l" lines
	"p" points connected by lines
	"o" lines over points
	"h" vertical lines
	"s" steps
col=	controls the color
lty=	controls the type of lines

Basic graphs



Legends

`legend()`

```
>plot(rnorm(10), type = "l")
>lines(rnorm(10), col = "red", lty = 2)
>title("Random numbers")
>legend("topright", c("1a série", "2a série"), lty =
+1:2,col = 1:2)
```

Several graphs on the same page

The argument `mfrow` is used to define line by line of the page where the graphs are putted. You could define column by column instead using `mfcol()` argument.

```
>par(mfrow=c(1,2))
>hist(CO2$uptake)
>pie(table(CO2$Plant))
```

Several graphs on the same page

You could split the window device by several sizes using the function `layout()`, where the principal argument is a matrix whose dimensions defines how the device will be divided.
`layout(matrix(1:4, 2, 2))`

You can use the function `layout.show()` to visualize the information.

```
layout.show(4)
```

Saving Graphs

You could save your graphs in several formats: .pdf, jpeg, ps:
Example:

```
>jpeg(file="plot.jpeg")
>plot(Plant)
>dev.off()
```

The image will be saved in your working directory.

Saving Graphs

To save a graphic:

- Click the Plots Tab window
- Click the Export button
- Choose your desired format
- Modify the export settings as you desire
- Click Save

Exercises

- Go to `https://github.com/CarinaSilva/Introducao-Linguagem-de-Programacao-em-R/tree/main/Dia_2`
- exercises_day2: Do exercise 1 to 8

lattice Package

Lattice is a powerful and elegant data visualization package for R programming, with an emphasis on multivariate data.

```
# Install
install.packages("lattice")
# Load
library("lattice")
```

lattice Package

The lattice package has a number of different functions to create different types of plot. For example, to create a scatterplot, use the `xyplot()` function. Notice that this is different from base graphics, where the `plot()` function creates a variety of different plot types.

lattice Package

The typical format is

`graph_type(formula, data=)`

formula: This is a formula typically of the form y $\sim$ x | z. It means to create a plot of y against x, conditional on z. In other words, create a plot for every unique value of z. Each of the variables in the formula has to be a column in the data frame that you specify in the data argument.
data: A data frame that contains all the columns that you specify in the formula argument.

lattice Package

graph_type is selected from the listed below:

graph_type	description	formula examples
barchart	bar chart	x~A or A~x
bwplot	boxplot	x~A or A~x
cloud	3D scatterplot	z~x*y|A
contourplot	3D contour plot	z~x*y
densityplot	kernal density plot	~x|A*B
dotplot	dotplot	~x|A
histogram	histogram	~x
levelplot	3D level plot	z~y*x
parallel	parallel coordinates plot	data frame
splom	scatterplot matrix	data frame
stripplot	strip plots	A~x or x~A
xyplot	scatterplot	y~x|A
wireframe	3D wireframe graph	z~y*x

lattice Package

◇∼x|A for each level of factor (A), it displays a numerical variable which is x;
◇ y∼x | A*B for every combination of factor A and B, there exists a relationship between the variables x and y.
◇ ∼x means display numeric variable x alone.

lattice Package

```
>my_data <- iris
>head(my_data)
# Default plot
>xyplot(Sepal.Length~ Petal.Length, data = my_data)
# Color by groups
>xyplot(Sepal.Length ~ Petal.Length, group = Species,
data = my_data, auto.key = TRUE)
```

lattice Package

```
# Show points ("p"), grids ("g") and smoothing line
# Change xlab and ylab
>xyplot(Sepal.Length ~ Petal.Length, data = my_data,
+ type = c("p", "g", "smooth"),
+ xlab = "Miles/(US) gallon", ylab = "Weight (1000
lbs)")
```

lattice Package

Multiple panels by groups: y ∼ x | group

```
>xyplot(Sepal.Length ~Petal.Length | Species,
+ group = Species, data = my_data,
+ type = c("p", "smooth"),
+ scales = "free")
```

lattice Package

```
>bwplot( Petal.Length ~ Species, data = my_data,
+ layout = c(4, 1), xlab = "Length")
 >bwplot( Petal.Length ~ Species, data = my_data,
+ layout = c(4, 1), xlab = "Length")
>bwplot( ~Petal.Length | Species, data = my_data,
+ layout = c(4, 1), xlab = "Length")
```

Exercises

- Go to `https://github.com/CarinaSilva/Introducao-Linguagem-de-Programacao-em-R/tree/main/Dia_2`
- exercises_day2: Do exercise 9 to 11.

ggplot2 Package

https://ggplot2.tidyverse.org/reference/

Functions

A function is a set of statements organized together to perform a specific task. R has a large number of in-built functions and the user can create their own functions.

An R function is created by using the keyword function. The basic syntax of an R function definition is as follows

```
function_name <- function(arg_1, arg_2, ...) {
   Function body
}
```

Functions

The different parts of a function are

- **Function Name** - This is the actual name of the function. It is stored in R environment as an object with this name.
- **Arguments** - An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional, that is, a function may contain no arguments. Also arguments can have default values.
- **Function Body** - The function body contains a collection of statements that defines what the function does.
- **Return Value** - The return value of a function is the last expression in the function body to be evaluated.

Functions

Example:

```
pow <- function(x, y) {
# function to print x raised to the power y
result <- x^y
print(paste(x,"raised to the power", y, "is", result))
}
```

Functions

How to call a function

```
>pow(8, 2)
[1] "8 raised to the power 2 is 64"
> pow(2, 8)
[1] "2 raised to the power 8 is 256"
```

Exercises

- Go to `https://github.com/CarinaSilva/Introducao-Linguagem-de-Programacao-em-R/tree/main/Dia_2`
- exercises_day2: Do exercise 12 to 14.

Iterations/Loops

Sometimes, iteration can not be avoided. The R commands for or while are useful in this situation. Here is an example of using for inside a function:

```
> Jsum <- function(x)          {
+   jsum <- 0
+   for(i in 1:length(x))  { jsum <- jsum + x[i] }
+   return(jsum)                  }
```

Note: R has its own function that performs this task, called sum. It will work MUCH faster than this one, especially on large vectors.

Iterations/Loops

When writing your own function, avoid iteration if you can; take advantage of R vectorized math and functions such as apply. It successively applies the function of your choice to each row (or column) of a matrix.

```
> x <-  matrix(1:12, 3, 4)
> apply(x, 2, mean)     # returns the mean of each column.
> apply(x, 1, mean)     # returns the mean of each row.
```