



## R-BASICS

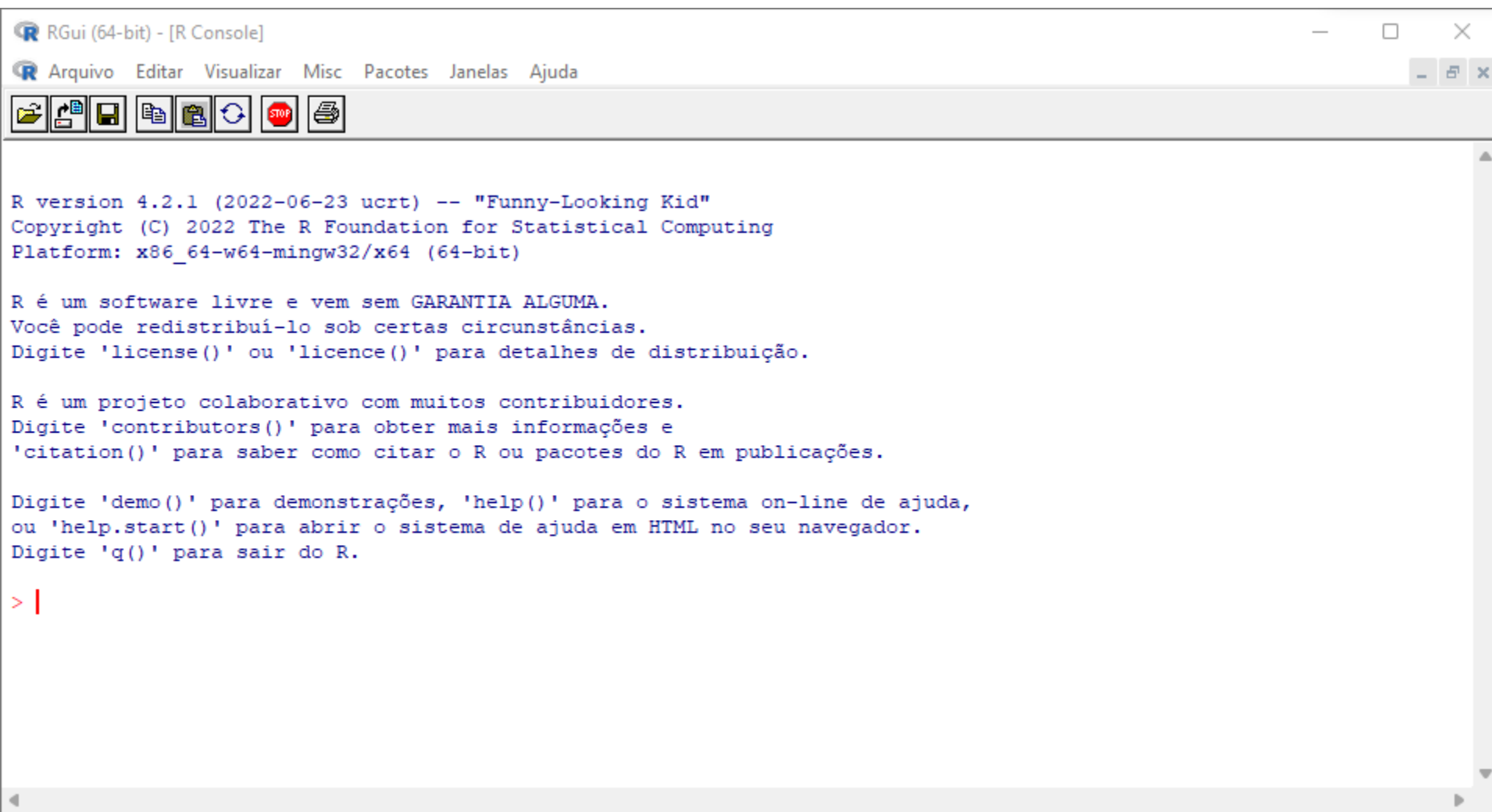
Eduardo Ogasawara  
[eduardo.ogasawara@cefet-rj.br](mailto:eduardo.ogasawara@cefet-rj.br)  
<https://eic.cefet-rj.br/~eogasawara>



## *Introduction to R*

- R is a programming language and free software environment for statistical computing
  - Supported by the R Foundation for Statistical Computing
- Created by Ross Ihaka and Robert Gentleman at Auckland University, New Zealand
- R was derived by S (Bell Laboratories - AT&T)
- R is a language broadly used by statisticians, data miners, and data scientists
- Current version 4.4

# R Console



```
RGui (64-bit) - [R Console]
Arquivo  Editar  Visualizar  Misc  Pacotes  Janelas  Ajuda

R version 4.2.1 (2022-06-23 ucrt) -- "Funny-Looking Kid"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)


R é um software livre e vem sem GARANTIA ALGUMA.
Você pode redistribuí-lo sob certas circunstâncias.
Digite 'license()' ou 'licence()' para detalhes de distribuição.

R é um projeto colaborativo com muitos contribuidores.
Digite 'contributors()' para obter mais informações e
'citation()' para saber como citar o R ou pacotes do R em publicações.

Digite 'demo()' para demonstrações, 'help()' para o sistema on-line de ajuda,
ou 'help.start()' para abrir o sistema de ajuda em HTML no seu navegador.
Digite 'q()' para sair do R.

> |
```

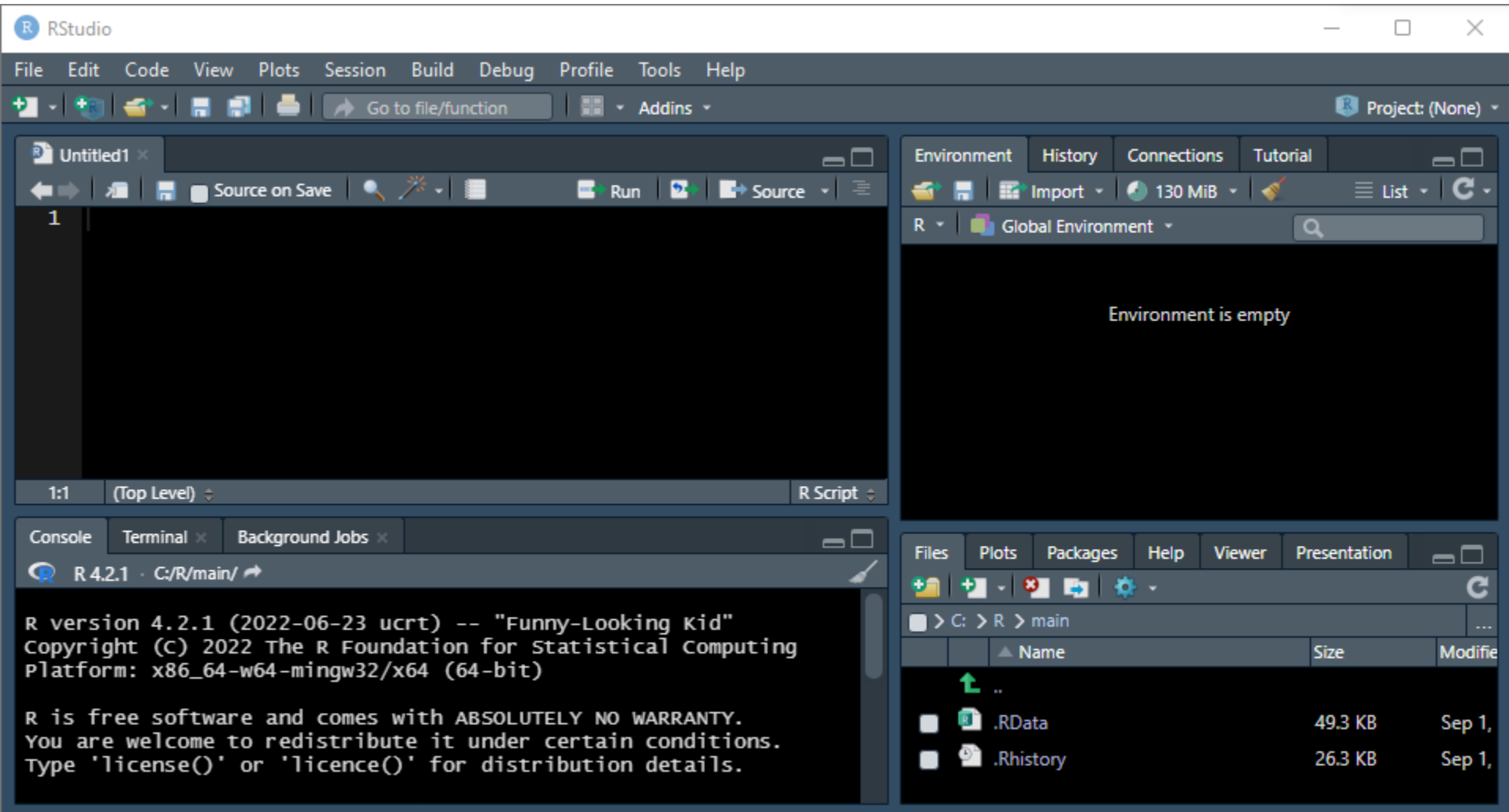
# ***R Console (Basic operations)***



The screenshot shows the RGui (64-bit) - [R Console] window. The title bar includes the R logo and the text "RGui (64-bit) - [R Console]". The menu bar contains "Arquivo", "Editar", "Visualizar", "Misc", "Pacotes", "Janelas", and "Ajuda". The toolbar has icons for file operations (open, save, print), editing (undo, redo), and a stop button. The console area displays the following R code and output:

```
> 1 + 2
[1] 3
> 1:10
[1] 1 2 3 4 5 6 7 8 9 10
> a <- 1
> a
[1] 1
> a + 2
[1] 3
> dado <- 1:6
> dado
[1] 1 2 3 4 5 6
> dado[1]
[1] 1
> dado[6]
[1] 6
> dado - 1
[1] 0 1 2 3 4 5
> |
```

# R Studio



# Run a block of code

The screenshot displays the RStudio interface with the following components:

- Source Editor:** Contains an R script named `01-primeiro-contato.R`. The code includes comments and arithmetic operations. Lines 1 through 12 are visible, with line 11 being the last line of code shown.
- Console:** Shows the output of the executed code. It displays the results of `1 + 2` (3), `1:10` (a vector of numbers 1 to 10), and `a <- 1` (1).
- Environment Pane:** Located on the right, it shows the current environment (Global Environment) and the values of variables. The variable `a` is listed with a value of 1.
- Files Pane:** Located at the bottom right, it shows the file structure of the project, including `..`, `.RData`, and `.Rhistory`.

```
1 #parte 1
2 1 + 2
3 1:10
4 a <- 1
5 a
6 a + 2
7 dado <- 1:6
8 dado
9 dado[1]
10 dado[6]
11 dado - 1
12
```

Console output:

```
> 1 + 2
[1] 3
> 1:10
[1] 1 2 3 4 5 6 7 8 9 10
> a <- 1
> a
[1] 1
>
```

Environment	History	Connections	Tutorial
R	Global Environment	123 MiB	
values			
a	1		

Files	Plots	Packages	Help	Viewer	Presentation
C: > R > main					
Name	Size	Modifie			
..					
.RData	49.3 KB	Sep 1,			
.Rhistory	26.3 KB	Sep 1,			

Execução de trechos marcados no código (botão run)

Veja a janela de variáveis do ambiente

# Run entire source file

The screenshot displays the RStudio environment with the following components:

- Source Editor:** Contains the script `01-primeiro-contato.R` with the following code:

```
1 #parte 1
2 1 + 2
3 1:10
4 a <- 1
5 a
6 a + 2
7 dado <- 1:6
8 dado
9 dado[1]
10 dado[6]
11 dado - 1
12
```
- Environment Pane:** Shows the Global Environment with the following values:

Variable	Value
a	1
dado	int [1:6] 1 2 3 4 5 6
- Console:** Shows the command `source("C:/Users/eduar/OneDrive/Aulas/R/01-primeiro-contato.R")` being executed.
- Files Pane:** Displays the file structure of the project, including `.RData` (49.3 KB) and `.Rhistory` (26.3 KB).

# ***R Packages***

---

- Packages are collections of functions made available as libraries
  - Published in the CRAN Repository
    - Quality control
    - good documentation
  - Uploaded from a GitHub repository
    - Versions under development
- Currently have more than 22000 packages
  - <https://cran.r-project.org/>
- R has a very active community
  - Several researchers, professors, programmers, and statisticians
- DAL Packages: 7 pacotes
  - <https://cran.r-project.org/web/packages/daltoolbox/index.html>
  - <https://cran.r-project.org/web/packages/harbinger/index.html>
  - <https://cran.r-project.org/web/packages/tspredict/index.html>



# ***R Package installation and loading***

---

## Package checking and instalation

```
if(!require(daltoolbox)) {  
  install.packages("daltoolbox")  
}
```



## Package loading

```
library(daltoolbox)
```



# Variable definitions

## Variable definition and assignment

```
weight <- 60  
height = 1.75  
subject <- "A"  
healthy <- TRUE
```



## Variable evaluation

```
weight
```



```
## [1] 60
```



## *Functions for type checking*

```
is.numeric(weight) # variable
```



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



```
is.double(weight)
```



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



```
is.integer(weight)
```



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



```
is.character(subject)
```



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



## *Functions for variable conversion*

---

```
weight <- as.integer(weight)  
is.integer(weight)
```



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



# Formulas and String formatting

Computing the body mass index (BMI) from the weight and height

```
#Body mass index (BMI)
bmi <- weight/height^2
bmi
```



```
## [1] 19.59184
```



Functions for string manipulation

```
message <- sprintf("%.1f", bmi)
print(message)
```



```
## [1] "19.6"
```



# Vectors

## definition

```
weight <- c(60, 72, 57, 90, 95, 72)
height <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
subject <- c("A", "B", "C", "D", "E", "F")
```



## evaluation

```
weight
```



```
## [1] 60 72 57 90 95 72
```



```
height
```



```
## [1] 1.75 1.80 1.65 1.90 1.74 1.91
```



```
subject
```



```
## [1] "A" "B" "C" "D" "E" "F"
```



# Vector indexes

vector of size 10

```
vec <- rep(0, 10)  
vec
```



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



vector length

```
length(weight)
```



```
## [1] 6
```



## Vector limits

from one to the length of the vector

```
weight[1]
```



```
## [1] 60
```



```
weight[length(weight)]
```



```
## [1] 72
```





## *Iterations: for loop*

from one to the length of weight

```
bmi <- 0
for (i in 1:length(weight)) {
  bmi[i] <- weight[i]/height[i]^2
}
```



evaluation of the bmi vector

```
bmi
```



```
## [1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630
```



# *Iterations: while loop*

run while i is below or equal to the length of weight

```
bmi <- 0
i <- 1
while (i <= length(weight)) {
  bmi[i] <- weight[i]/height[i]^2
  i <- i + 1
}
```

```
bmi
```

```
## [1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630
```

## *Removing variables*

---

```
rm(bmi)  
exists("bmi")
```



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



## *Processing vectors at once*

---

```
bmi <- weight/height^2  
bmi
```



```
## [1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630
```



# Creating functions

```
name <- function(parameters) { body }
```

```
compute_bmi <- function(weight, height) {  
  bmi <- weight/height^2  
  return(bmi)  
}
```



calling it

```
bmi <- compute_bmi(60, 1.75)  
bmi
```



```
## [1] 19.59184
```



## *Function processing vectors*

---

```
bmi <- compute_bmi(weight, height)
bmi
```



```
## [1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630
```



## ***Example***

---

- Knowing that mean is computed as
  - $\mu(X) = \frac{\sum_{i=1}^n x_i}{n}$
- Create a function to compute the mean of a vector

## Example: function to compute the average

(iterating in all elements of the vector)

```
average <- function(vec) {  
  s <- 0  
  n <- length(vec)  
  for (x in vec) {  
    s <- s + x  
  }  
  return(s/n)  
}
```

invoking the function

```
avg_bmi <- average(bmi)  
avg_bmi
```

```
## [1] 23.13262
```



## *Example of a function to compute the average (improved version)*

manipulating vectors at once

```
average <- function(vec) {  
  s <- sum(vec)  
  n <- length(vec)  
  return(s/n)  
}
```



invoking the function

```
avg_bmi <- average(bmi)  
avg_bmi
```



```
## [1] 23.13262
```



## *Example: function using mean function*

Major statistical functions are available in R

```
average <- function(vec) {  
  return(mean(vec))  
}
```



invoking the function

```
avg_bmi <- average(bmi)  
avg_bmi
```



```
## [1] 23.13262
```



## *Not available (NA)*

Operations with NA lead to NA.

```
x <- c(10, NA, 13)
y <- average(x)
y
```

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

Addressing NA with na.rm=TRUE

```
average <- function(vec) {
  return(mean(vec, na.rm=TRUE))
}
```

```
x <- c(10, NA, 13)
y <- average(x)
y
```

```
## [1] 11.5
```

## Part II

# Plotting basic graphics

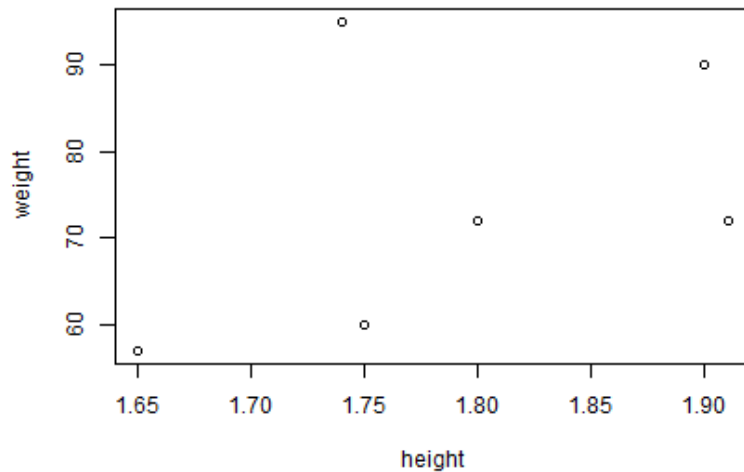
defining variables

```
weight <- c(60, 72, 57, 90, 95, 72)
height <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
bmi <- weight/height^2
```



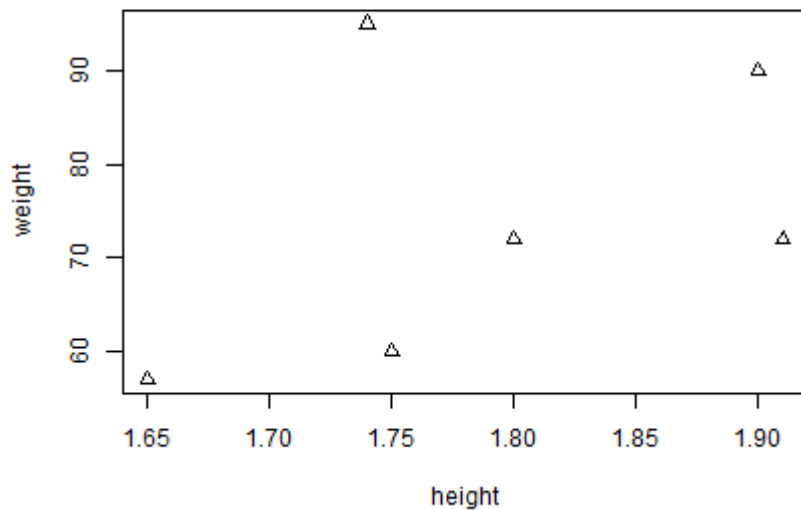
scatter plots

```
plot(height, weight)
```



# *Most functions contain many default parameters*

```
plot(height, weight, pch=2)
```



# All functions in R CRAN packages have help with examples

?base::plot



plot.default {graphics}

R Documentation

## The Default Scatterplot Function

### Description

Draw a scatter plot with decorations such as axes and titles in the active graphics window.

### Usage

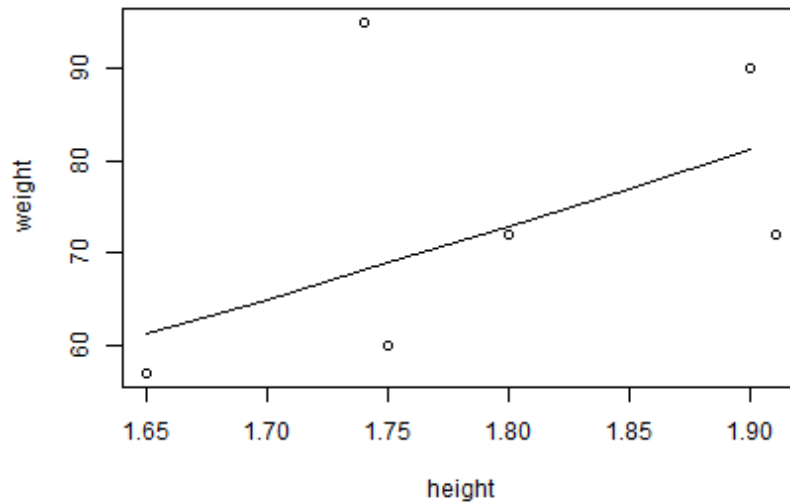
```
## Default S3 method:
plot(x, y = NULL, type = "p", xlim = NULL, ylim = NULL,
     log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
     ann = par("ann"), axes = TRUE, frame.plot = axes,
     panel.first = NULL, panel.last = NULL, asp = NA,
     xgap.axis = NA, ygap.axis = NA,
     ...)
```

### Arguments

- x, y** the **x** and **y** arguments provide the x and y coordinates for the plot. Any reasonable way of defining the coordinates is acceptable. See the function `xy.coords` for details. If supplied separately, they must be of the same length.
- type** 1-character string giving the type of plot desired. The following values are possible, for details, see

## *Canvas for plotting is still active until a new plot*

```
plot(height, weight)  
hh = c(1.65, 1.70, 1.75, 1.80, 1.85, 1.90)  
lines(hh, 22.5 * hh^2)
```





## ***Factors***

---

- Factors are variables in R that refer to categorical data
- Factors in R are stored as a vector of integer values with a corresponding set of character values to use when the factor is displayed
- Both numeric and character variables can be made into factors, but a factor's levels are always character values

# Factors

Factors are used to handle categorical data.

```
pain <- c(0,3,2,2,1)
fpain <- factor(pain,levels=0:3, ordered=TRUE)
fpain
```

```
## [1] 0 3 2 2 1
## Levels: 0 < 1 < 2 < 3
```

Levels provide correspondence between numerical values and categorical labels

```
levels(fpain) <- c("none","mild","medium","severe")
fpain
```

```
## [1] none   severe medium medium mild
## Levels: none < mild < medium < severe
```

## ***Example***

---

- Consider the height variable
  - Persons lower than 1.5 m are small
  - Persons greater than 1.9 m are tall
  - Persons in between are medium
  - Convert the height variable into a factor with small, medium, tall

# *Convert height to factor with levels: small, medium, high*

coding setting element by element

```
lev <- rep("", length(height))

for (i in 1:length(height)) {
  if (height[i] < 1.5)
    lev[i] <- "short"
  else if (height[i] < 1.9)
    lev[i] <- "medium"
  else
    lev[i] <- "tall"
}

lev <- as.factor(lev)
lev
```

```
## [1] medium medium medium tall   medium tall
## Levels: medium tall
```

## Using the function cut

```
lev <- cut(height, breaks=c(0, 1.5, 1.9, .Machine$double.xmax), ordered=TRUE)
lev
```



```
## [1] (1.5,1.9]      (1.5,1.9]      (1.5,1.9]      (1.5,1.9]      (1.5,1.9]      (1.9,1.8e+308]
## Levels: (0,1.5] < (1.5,1.9] < (1.9,1.8e+308]
```



```
levels(lev) <- c("short", "medium", "tall")
lev
```



```
## [1] medium medium medium medium medium tall
## Levels: short < medium < tall
```



# Matrix

Matrices can be filled from vectors or data frames.

```
x <- 1:9  
x
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

Converting a vector to matrix

```
dim(x) <- c(3,3)  
x
```

```
##      [,1] [,2] [,3]  
## [1,]    1    4    7  
## [2,]    2    5    8  
## [3,]    3    6    9
```

# Scalar multiplication

```
y <- 3*x  
y
```



```
##      [,1] [,2] [,3]  
## [1,]   3  12  21  
## [2,]   6  15  24  
## [3,]   9  18  27
```



# Matrix manipulation

Converting a vector to a matrix by row

```
x <- matrix(1:9,nrow=3,byrow=T)
x
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

transposing a matrix

```
x <- t(x)
x
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```



## ***Lists***

---

- Lists are the R objects which contain elements of different types, such as numbers, strings, vectors, matrix, data frame, and another list inside it
- A list can also contain a matrix or a function as its elements
- A list is created using the `list()` function
- List manipulation
  - Slicing a list `[]`
  - Accessing an element inside a list `[[ ]]`

# Creating a list

Lists are used to work with "objects"

```
a <- c(5260,5470,5640,6180,6390,6515,6805,7515,7515,8230,8770)
b <- c(3910,4220,3885,5160,5645,4680,5265,5975,6790,6900,7335)

mybag <- list(a, b, 0, "a")
mybag
```

```
## [[1]]
## [1] 5260 5470 5640 6180 6390 6515 6805 7515 7515 8230 8770
##
## [[2]]
## [1] 3910 4220 3885 5160 5645 4680 5265 5975 6790 6900 7335
##
## [[3]]
## [1] 0
##
## [[4]]
## [1] "a"
```

## *Adding elements into a list*

```
n <- length(mybag)
mybag[[n+1]] <- "b"
mybag
```



```
## [[1]]
## [1] 5260 5470 5640 6180 6390 6515 6805 7515 7515 8230 8770
##
## [[2]]
## [1] 3910 4220 3885 5160 5645 4680 5265 5975 6790 6900 7335
##
## [[3]]
## [1] 0
##
## [[4]]
## [1] "a"
##
## [[5]]
## [1] "b"
```



## List slicing

```
slice <- mybag[1]  
slice
```



```
## [[1]]  
## [1] 5260 5470 5640 6180 6390 6515 6805 7515 7515 8230 8770
```



```
is.list(slice)
```



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



## *Lists: Member reference*

Picking the first element

```
h <- mybag[[1]]  
h
```



```
## [1] 5260 5470 5640 6180 6390 6515 6805 7515 7515 8230 8770
```



An element can be evaluated. In this case, it is a vector.

```
is.vector(h)
```



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



```
is.list(h)
```



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



# Creating lists with attributes

They are properties on the list

```
mybag <- list(x=a, y=b, const=0, lit="a")  
mybag
```

```
## $x  
## [1] 5260 5470 5640 6180 6390 6515 6805 7515 7515 8230 8770  
##  
## $y  
## [1] 3910 4220 3885 5160 5645 4680 5265 5975 6790 6900 7335  
##  
## $const  
## [1] 0  
##  
## $lit  
## [1] "a"
```

# *Adding, accessing, and removing elements*

## Adding, accessing, and removing elements

```
mybag$c <- mybag$x - mybag$y  
mybag$const <- NULL  
mybag$lit <- NULL  
mybag
```



```
## $x  
## [1] 5260 5470 5640 6180 6390 6515 6805 7515 7515 8230 8770  
##  
## $y  
## [1] 3910 4220 3885 5160 5645 4680 5265 5975 6790 6900 7335  
##  
## $c  
## [1] 1350 1250 1755 1020 745 1835 1540 1540 725 1330 1435
```



## Part III



# Data frames

Data frames (tables) provide support for structured data.

```
a <- c(5260,5470,5640,6180,6390,6515,6805,7515,7515,8230,8770)
b <- c(3910,4220,3885,5160,5645,4680,5265,5975,6790,6900,7335)
```



```
data <- data.frame(A=a, B=b)
head(data)
```



```
##      A      B
## 1 5260 3910
## 2 5470 4220
## 3 5640 3885
## 4 6180 5160
## 5 6390 5645
## 6 6515 4680
```



## *Adding a column in a data frame*

```
data$c <- data$A + data$B  
head(data)
```

```
##      A      B      c  
## 1 5260 3910  9170  
## 2 5470 4220  9690  
## 3 5640 3885  9525  
## 4 6180 5160 11340  
## 5 6390 5645 12035  
## 6 6515 4680 11195
```

## *Removing a column of a data frame*

```
data$A <- NULL  
head(data)
```

```
##      B      c  
## 1 3910  9170  
## 2 4220  9690  
## 3 3885  9525  
## 4 5160 11340  
## 5 5645 12035  
## 6 4680 11195
```

# Reading a csv file

There are many functions for reading CSV, Excel, and RData formats.

```
wine = read.table(  
  "http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data",  
  header = TRUE, sep = ",")  
colnames(wine) <- c('Type', 'Alcohol', 'Malic', 'Ash',  
  'Alcalinity', 'Magnesium', 'Phenols',  
  'Flavanoids', 'Nonflavanoids',  
  'Proanthocyanins', 'Color', 'Hue',  
  'Dilution', 'Proline')  
  
head(wine)
```

##	Type	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthocyanins	Color	H
## 1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.
## 2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.
## 3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.
## 4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.
## 5	1	14.20	1.76	2.45	15.2	112	3.27	3.39	0.34	1.97	6.75	1.
## 6	1	14.39	1.87	2.45	14.6	96	2.50	2.52	0.30	1.98	5.25	1.

# ***Saving and loading Rdata files***

saving a data frame

```
save(wine, file="wine.RData")
```



removing a data frame from memory

```
rm(wine)
```



loading it

```
load("wine.RData")  
head(wine, 3)
```



```
##   Type Alcohol Malic  Ash Alcalinity Magnesium Phenols Flavanoids Nonflavanoids Proanthocyanins Color H  
## 1    1   13.20  1.78 2.14      11.2      100    2.65      2.76          0.26          1.28 4.38 1.  
## 2    1   13.16  2.36 2.67      18.6      101    2.80      3.24          0.30          2.81 5.68 1.  
## 3    1   14.37  1.95 2.50      16.8      113    3.85      3.49          0.24          2.18 7.80 0.  
##   Dilution Proline  
## 1     3.40    1050  
## 2     3.17    1185  
## 3     3.45    1480
```



## *Exporting data.frame into csv file*

---

```
write.table(wine, file="wine.csv", row.names=FALSE, quote = FALSE, sep = ",")
```



## Filtering vectors

```
a <- c(5260,5470,5640,6180,6390,6515,6805,7515,7515,8230,8770)
b <- c(3910,4220,3885,5160,5645,4680,5265,5975,6790,6900,7335)
data <- data.frame(a = a, b = b)
head(data[data$a > 7000,])
```

```
##      a    b
## 8  7515 5975
## 9  7515 6790
## 10 8230 6900
## 11 8770 7335
```

## Filtering data frames

```
a <- c(5260,5470,5640,6180,6390,6515,6805,7515,7515,8230,8770)
b <- c(3910,4220,3885,5160,5645,4680,5265,5975,6790,6900,7335)
data <- data.frame(a = a, b = b)
head(data[data$a > 7000,])
```

```
##      a    b
## 8  7515 5975
## 9  7515 6790
## 10 8230 6900
## 11 8770 7335
```



## *filtering data frames (rows and columns)*

---

```
head(data[data$a > 7000,c(1)])
```



```
•  
## [1] 7515 7515 8230 8770
```



# ***Benchmarking performance of matrix and data frames***

---

```
rheight <- rnorm(100000, 1.8, sd=0.2)  
rweight <- rnorm(100000, 72, sd=15)
```



## *Computing a entire column at once in a data frame*

```
start_time <- Sys.time()

hw <- data.frame(height=rheight, weight=rweight)
hw$bmi <- hw$weight/hw$height^2

end_time <- Sys.time()
end_time - start_time
```



```
## Time difference of 0.002799988 secs
```



```
object.size(hw)
```



```
## 2400984 bytes
```



# Computing a column in a data frame cell by cell

processing cell by cell

```
start_time <- Sys.time()
hw <- data.frame(height=rheight, weight=rweight)
for (i in 1:nrow(hw)) {
  hw$bmi[i] <- hw$weight[i]/hw$height[i]^2
}
end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 12.53602 secs
```

# Converting a data frame to a matrix, processing it, and going back

```
start_time <- Sys.time()
hw <- data.frame(height=rheight, weight=rweight)

hw <- as.matrix(hw)

hw <- cbind(hw, 0)
for (i in 1:nrow(hw)) {
  hw[i,3] <- hw[i,2]/hw[i,1]^2
}

hw <- as.data.frame(hw)

end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 0.272609 secs
```

## Part IV

# Pipelines

The operator `|>` creates a pipeline.

The first parameter of the next invoked function receives the data from the pipeline.

Library *dplyr* contains a set of functions that support relational algebra operations.

```
flight_data <- read.table(text = "Year Quarter Flights Delays
2016 1 11 6
2016 2 12 5
2016 3 13 3
2016 4 12 5
2017 1 10 4
2017 2 9 3
2017 3 11 4
2017 4 25 15
2018 1 14 3
2018 2 12 5
2018 3 13 3
2018 4 15 4",
header = TRUE, sep = "")
```



## *Displaying the data frame*

```
head(flight_data)
```



```
##   Year Quarter Flights Delays
## 1 2016      1      11      6
## 2 2016      2      12      5
## 3 2016      3      13      3
## 4 2016      4      12      5
## 5 2017      1      10      4
## 6 2017      2       9      3
```





# Basic Query

```
library(dplyr)
```



```
result <- flight_data |>  
  filter(Delays > 5) |>  
  select(Year, Quarter, Flights)  
head(result)
```



```
##   Year Quarter Flights  
## 1 2016         1      11  
## 2 2017         4      25
```



## Aggregated query

```
result <- flight_data |>  
  group_by(Year) |>  
  summarize(mean = mean(Flights), sd = sd(Flights))  
head(result)
```

```
## # A tibble: 3 × 3  
##   Year mean  sd  
##   <int> <dbl> <dbl>  
## 1  2016   12  0.816  
## 2  2017  13.8  7.54  
## 3  2018  13.5  1.29
```

# Tables join

## Store table

```
stores <- data.frame(  
  city = c("Rio de Janeiro", "Sao Paulo", "Paris", "New York", "Tokyo"),  
  value = c(10, 12, 20, 25, 18))  
head(stores)
```

```
##           city value  
## 1 Rio de Janeiro   10  
## 2      Sao Paulo   12  
## 3         Paris   20  
## 4      New York   25  
## 5         Tokyo   18
```

## Division table

```
divisions <- data.frame(  
  city = c("Rio de Janeiro", "Sao Paulo", "Paris", "New York", "Tokyo"),  
  country = c("Brazil", "Brazil", "France", "US", "Japan"))  
head(divisions)
```

```
##           city country  
## 1 Rio de Janeiro  Brazil  
## 2      Sao Paulo  Brazil  
## 3         Paris  France  
## 4      New York     US  
## 5         Tokyo   Japan
```

# Merge function

The function *merge* can be used to join data frames. It can be used to produce inner, left, right, and outer joins.

```
stdiv <- merge(stores, divisions, by.x="city", by.y="city")  
head(stdiv)
```

```
##           city value country  
## 1      New York    25      US  
## 2         Paris    20  France  
## 3 Rio de Janeiro    10  Brazil  
## 4      Sao Paulo    12  Brazil  
## 5         Tokyo    18   Japan
```

## Aggregating merged data frame

```
result <- stdiv |> group_by(country) |>  
  summarize(count = n(), amount = sum(value))  
head(result)
```



```
## # A tibble: 4 × 3  
##   country count amount  
##   <chr>   <int> <dbl>  
## 1 Brazil     2     22  
## 2 France     1     20  
## 3 Japan      1     18  
## 4 US         1     25
```



# Part V

# Statistical analysis

There are many statistical tests in R. One of the most used is the t-test. It checks if the mean of observations is not different from a theoretical value.

```
weight <- c(60, 72, 57, 90, 95, 72)
height <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
bmi <- weight/height^2
```



```
t.test(bmi, mu=22.5)
```



```
##
##      One Sample t-test
##
## data:  bmi
## t = 0.34488, df = 5, p-value = 0.7442
## alternative hypothesis: true mean is not equal to 22.5
## 95 percent confidence interval:
##  18.41734 27.84791
## sample estimates:
## mean of x
##  23.13262
```



# Python + R integration

Python code at retic.py

```
import pyreadr
import pandas

def add(x, y):
    return x + y

def read_rdata_mem(data):
    x = data["x"]
    print(x)
    y = data["y"]
    data["z"] = x + y
    return(data)
```



Library reticulate enables seamless integration with Python

```
library(reticulate)
source_python('https://raw.githubusercontent.com/eogasawara/analise-dados/refs/heads/main/python/retic.py')
x <- add(5, 10)
x
```



```
## [1] 15
```





# Python + R dataset integration

```
data <- data.frame(x = c(1:5), y=c(11:15))  
dfm <- read_rdata_mem(data)
```



```
## 0    1  
## 1    2  
## 2    3  
## 3    4  
## 4    5  
## Name: x, dtype: int32
```



```
head(dfm)
```

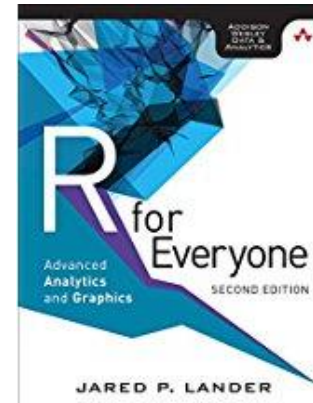
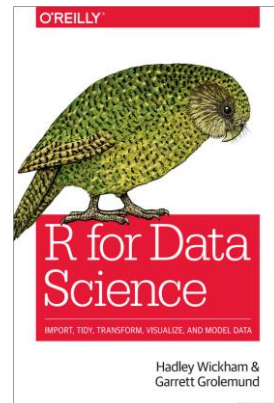
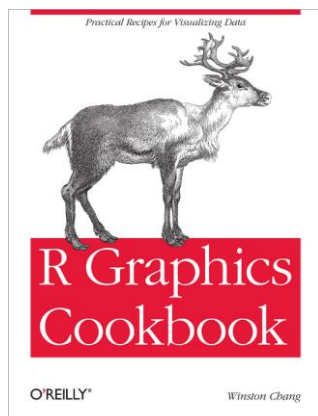


```
##   x  y  z  
## 1 1 11 12  
## 2 2 12 14  
## 3 3 13 16  
## 4 4 14 18  
## 5 5 15 20
```



# Main References

Código: <https://github.com/eogasawara/analise-dados/blob/main/examples/2-R-Basics.md>



Further reading: Hands-On Programming with R, Chapters 1-12

Slides e vídeos em: <https://eic.cefet-rj.br/~eogasawara/analise-de-dados>



- <https://rstudio-education.github.io/hopr/index.html>
- <https://r-graphics.org>