

# Curso de R básico

Sergio Pedro R Oliveira

2023-09-15

# SUMÁRIO

<b>1</b>	<b>Objetivo</b>	<b>1</b>
<b>2</b>	<b>Referência</b>	<b>2</b>
<b>3</b>	<b>Aula 01 - Introdução</b>	<b>3</b>
<b>4</b>	<b>Aula 02 - Instalação R e RStudio</b>	<b>4</b>
<b>5</b>	<b>Aula 03 - Conhecendo o R e o RStudio parte 1</b>	<b>5</b>
5.1	Configuração do <b>layout</b> do <b>RStudio</b>	5
5.2	Inserir <b>comentarios</b>	5
5.3	Compilar código	5
5.4	Limpar a tela do console	5
5.5	Atribuições	5
5.6	Tipagem de variáveis	5
<b>6</b>	<b>Aula 04 - Conhecendo R e o RStudio parte 2</b>	<b>6</b>
6.1	<b>Packages</b> - Bibliotecas/Pacotes	6
6.2	Pesquisa de função - função <b>Help</b>	7
6.3	<b>summary</b> - Resumo dos dados	7
6.4	Descobrir a classe de uma variável - <b>class</b>	7
6.5	Descobrir a estrutura de uma variável - <b>str</b>	8
<b>7</b>	<b>Aula 05 - Operadores</b>	<b>9</b>
7.1	Operadores básicos	9
7.2	Operadores lógicos	9
<b>8</b>	<b>Aula 06 á 09 - Tipo de dados</b>	<b>10</b>
8.1	Tipos de dados	10
8.1.1	Numerico - <b>numeric</b>	10
8.1.2	Caractere e String - <b>character</b>	10
8.1.3	Fatores - <b>factor</b>	11
8.1.4	Lógico	11
8.1.5	Data - date	12
8.2	Descobrir e converter tipos	13
8.2.1	as.tipo	13
8.2.2	is.tipo	13
<b>9</b>	<b>Aula 10 e 11 - Vetores e Listas</b>	<b>14</b>
9.1	Vetor	14
9.2	Lista	14
<b>10</b>	<b>Aula 12 e 13 - Matrizes e Data.frames</b>	<b>15</b>
10.1	Matrizes	15
10.2	Tabela de dados - Data.frames	15
10.3	Visualizar dados matriz e data.frame - <b>VIEW</b>	16
<b>11</b>	<b>Aula 14 e 15 - Filtros</b>	<b>17</b>
11.1	Vetor	17
11.2	Data.Frames	18
<b>12</b>	<b>Aula 16 - Condicionais IF, FOR e WHILE</b>	<b>19</b>
12.1	<b>IF</b>	19
12.2	Loop <b>FOR</b>	20

12.3 Loop <b>WHILE</b> . . . . .	20
<b>13 Aula 17 - Função e print</b>	<b>21</b>
13.1 Função - <b>function</b> . . . . .	21
13.2 Print . . . . .	21
<b>14 Andamento dos Estudos</b>	<b>22</b>
14.1 Assunto em andamento . . . . .	22

## LISTA DE FIGURAS

## LISTA DE TABELAS

1	Operadores Básicos . . . . .	9
2	Operadores Lógicos . . . . .	9

# 1 Objetivo

Estudo dirigido de básico de linguagem R.

## 2 Referência

Videoaulas de ‘curso R para iniciantes’.

### 3 Aula 01 - Introdução

- **R** é uma linguagem estatística e gráfica.
- É uma linguagem com um foco bem definido.
- Muito usada para **Big Data** e **Machine Learning**.
- Linguagem de código aberto.
- Muitos pacotes a disposição.
- Ambiente de desenvolvimento: **RStudio**.



## 4 Aula 02 - Instalação R e RStudio

- **R**  
<https://cran.r-project.org/>
- **RStudio**(IDE)  
<https://www.rstudio.com/>

## 5 Aula 03 - Conhecendo o R e o RStudio parte 1

### 5.1 Configuração do layout do RStudio

- Alterando o *layout* do **RStudio**:
  - opção **Tools**
  - **Global Options**
  - **Pane Layout**  
Reorganizar o *layout* das janelas do RStudio para melhor se adaptar ao usuário.

### 5.2 Inserir comentarios

- Inserir **comentarios**, utilizar ‘#’.

### 5.3 Compilar código

- Para executar um script:
  - Deixar o cursor sobre a linha que deve ser executada.
  - Selecionar o código que deve ser executado.  
Obs.: Atalho para executar ‘CTRL + Enter’.

### 5.4 Limpar a tela do console

- Comando que limpa a tela do console:
  - CTRL + L

### 5.5 Atribuições

(Tipagem automática/dinâmica)

- Atribuições de **variáveis** usar o sinal ‘->’ ou ‘<-’.
- Atribuição de **funções** utilizar o sinal ‘=’.

### 5.6 Tipagem de variáveis

- Tipagem das variáveis automática/dinâmica.
- Erro ao fazer operações com variáveis de tipos distintos.

## 6 Aula 04 - Conhecendo R e o RStudio parte 2

### 6.1 Packages - Bibliotecas/Pacotes

- São bibliotecas/pacotes com funções que podem ser baixados e instalados.
- Os pacotes/bibliotecas são instalados no sistema.
- As bibliotecas são baixadas com o comando no **R**:
  - Sintaxe:  
**install.packages** (“*nome\_do\_pacote*”)
- Para usar os pacotes é preciso declarar eles no script (chamar o pacote):
  - Sintaxe:  
**library**(“*nome\_do\_pacote*”)
- Baixado e declarado o pacote/biblioteca no script é possível agora usar as funções desse pacote.

## 6.2 Pesquisa de função - função `Help`

- `Help` '?', usado para pesquisar uma função na documentação do **R**.
- O resultado da pesquisa aparece no `Help`.
- Sintaxe:  
`?c`
- Caso o *Help* não ajude a descobrir a função, outro modo de pesquisar é recorrer ao site:  
<https://www.rdocumentation.org/>
  - Um site destinado a pesquisa e informações sobre a documentação do **R**.
  - Retorna informações sobre a função, estrutura, pacote a qual ela faz parte, etc.

## 6.3 `summary` - Resumo dos dados

- A função `summary()` retorna o resumo de variáveis.
- O retorno depende do argumento (se for um vetor, uma lista, um `data.frame`).
- O retorno para uma matriz ou `data.frame`, vai ser os métodos aplicados a cada campo/coluna.
- O retorno da função, no geral, retorna diversos métodos aplicados aos dados, tais como:
  - valor mínimo
  - 1º quantil
  - valor da mediana
  - valor da média
  - 3º quantil
  - valor máximo
- Sintaxe:  
`summary(nome_variavel)`

## 6.4 Descobrir a classe de uma variável - `class`

- A função `class()` retorna a que classe do objeto do argumento pertence.
- Basicamente diz se o objeto é numérico, string, vetor, lista, `data.frame`, matriz, ...
- Sintaxe:  
`class(argumento)`

## 6.5 Descobrir a estrutura de uma variavel - `str`

- A função `str()` retorna a estrutura do objeto do argumento.
- Retorna diversos dados, entre eles:
  - A classe do objeto.
  - Tamanho do objeto.
  - A lista, ou vetor, dos campos com o tipo e tamanho.
- Sintaxe:  
`str(argumento)`

## 7 Aula 05 - Operadores

### 7.1 Operadores básicos

Table 1: Operadores Básicos

Operador Básico	Símbolo
Soma	+
Subtração	-
Divisão	/
Multiplicação	*
Potencia	** ou ^
Raiz	sqrt(numero)
Exponencial	exp(numero)
Log na base n	log(numero,n)

### 7.2 Operadores lógicos

Table 2: Operadores Lógicos

Operador Lógico	Símbolo
Igual	==
Diferente	!=
Maior que	>
Maior ou igual	>=
Menor que	<
Menor ou igual	<=
E	&
OU	
Negação	!

- A resposta do sistema para uma operação lógica é *TRUE* (verdadeiro) ou *FALSE* (falso). Ou em termos numéricos 1 (verdadeiro), 0 (falso).
- No caso da negação (NOT), ele vem antes da operação como um todo.

– Exemplo:

!5 > 4

## 8 Aula 06 á 09 - Tipo de dados

### 8.1 Tipos de dados

#### 8.1.1 Numerico - numeric

- O **R** converte automaticamente variáveis numéricas para tipo `numeric`.
- São variáveis contando números tanto inteiros (*int*), quanto *float*.
- Variáveis do tipo `numeric` podem realizar operações matemáticas.
- É possível converter um número qualquer (*numeric*) para um número inteiro (*int*) utilizando a função uma função:

- `as.integer(variavel)`

- Observações:
  - A função `as.interger(variavel)` faz truncamento. CUIDADO!!!
  - Para fazer arredondamento, usar a função `round(variavel, qtd_decimal)`.

#### 8.1.2 Caractere e String - character

- O **R** entende como variáveis do tipo *character* (caracter/string) todo dado que tiver entre aspas (“”).
- Mesmo números se tiverem entre aspas, serão lidos como *character*.
- Tipo *character* não faz operações matemáticas.
- Tipo *character* faz operações lógicas (igual á, diferente de, ...)
  - Espaços em branco fazem diferença para operações lógicas.

### 8.1.3 Fatores - factor

- Factor é o tipo categoria/enumerado.
- O tipo **factor** não se trata de numeros, mas sim de categorias classificar determinado registro.
- O tipo **factor** não se comporta como numeros, logo não é possível fazer operações matemáticas nele. Porém aceita operações lógicas.
- Para tipar uma variável (ou determinado vetor) como **factor** basta usar a função `as.factor(argumento)`.
- No caso da conversão de vetores para tipo factor dos elementos:
  - Elementos iguais serão considerados “níveis”.
  - A função `summary(vetor)`, numera o número de elementos no mesmo “nível”.

### 8.1.4 Lógico

- É possível salvar dentro de uma variável uma operação do tipo lógica.
- O que é salvo é o resultado da operação, os valores “*TRUE*” ou “*FALSE*”.
- O tipo lógico são palavras reservadas *TRUE* e *FALSE*. Se utiliza-las dentro de aspas elas se convertem em tipo character (“*TRUE*” e “*FALSE*”).
- Se converter um tipo lógico para numérico *TRUE* assume o valor 1 e *FALSE* o valor 0.
- Sintaxe (exemplo):  
`L <- variavel_1 < variavel_2`



### 8.1.5 Data - date

- O pacote `lubridate` permite o acesso ao tipo `date` (data) e funções para se trabalhar com esse tipo de variável.
- Normalmente passamos uma variável do tipo **character**, no formato de data (ex.: 2006-05-29), para o tipo **date**.
- Podemos passar do tipo **char** para o tipo **date** através das funções:
  - `date(variável_chr)`  
Converte variável do tipo `chr` em tipo `date`.  
Ex.: `data_date <- date(data_string)`
  - `as_date(variável_chr)`
  - `as.Date(variável_chr)`  
Função da base do **R** para converter variável do tipo `chr` em tipo `date`.
  - `dmy(variável_chr)`  
Converte variável `chr` no formato `date` (dia, mês e ano).  
Ex. formato: 21-10-2015.
  - `mdy(variável_chr)`  
Converte variável `chr` no formato `date` (mês, dia e ano).  
Ex. formato: 10-21-2015.
  - `myd(variável_chr)`  
Converte variável `chr` no formato `date` (mês, ano e dia).  
Ex. formato: 10-2015-21.
  - `ymd(variável_chr)`  
Converte variável `chr` no formato `date` (ano, mês e dia).  
Ex. formato: 2015-10-21.
  - `ydm(variável_chr)`  
Converte variável `chr` no formato `date` (ano, dia e mês).  
Ex. formato: 2015-21-10.
- Sintaxe:  
`data <- as.date(variável_char)`

## 8.2 Descobrir e converter tipos

### 8.2.1 `as.tipo`

- As funções começadas por “`as.`”, seguidas do “*tipo*” e a “*variavel*” como argumento, servem para converter variáveis para outros tipos.
- Sintaxe (exemplo):  
`as.factor(variavel)`
  - converte a variável de um tipo qualquer para uma variável do tipo factor.

### 8.2.2 `is.tipo`

- As funções começadas por “`is.`”, seguidas por “*tipo*” e a “*variavel*” como argumento, servem para descobrir/confirmar o tipo da variável.
- O retorno é “*TRUE*” ou “*FALSE*”.
- Sintaxe (exemplo):  
`is.factor(variavel)`
  - Retorna “*TRUE*” se verdadeiro, o tipo da variável for factor.
  - Ou retorna “*FALSE*” se falso, o tipo da variável não for factor.

## 9 Aula 10 e 11 - Vetores e Listas

### 9.1 Vetor

- Vetores são variáveis que aguardam diversos valores de mesmo tipo.
- No **R** o vetor é criado a partir da função `c()`.
- *Strings* e *caracteres* dentro do vetor devem ter seus valores entre aspas (“”).
- *length* indica o tamanho do vetor, o número de elementos dentro do vetor.
- Caso seja inserido um dado do tipo *character* para um vetor do tipo *numérico*, ele converte todo o vetor em tipo *character* (todos os elementos em *character*).
- Sintaxe (exemplo):  
`c(10,5,8,...)`
- Acessando um valor dentro de um vetor, uso a notação:  
`nome_do_vetor[posição]`

### 9.2 Lista

- Listas são os objetos R que contêm elementos de diferentes tipos (diferente do vetor), como `_` números, strings, vetores e outra lista dentro dela.
- Uma lista também pode conter uma matriz ou uma função como seus elementos.
- *Strings* e *caracteres* dentro da lista devem ter seus valores entre aspas (“”).
- A lista é criada usando a função `list()`.
- Sintaxe (exemplo):  
`list(azul, 10, c(5,8,9), ...)`
- Acessando um valor dentro de uma lista.
  - Notação:  
`nome_da_lista [[posição]]`
  - Caso tenha uma lista ou vetor dentro de algum elemento:  
`nome_da_lista [[posição]][posição_dentro_da_lista_ou_vetor_do_elemento]`

## 10 Aula 12 e 13 - Matrizes e Data.frames

### 10.1 Matrizes

- Vetor com duas dimensões.
- A matriz aceita só um tipo de dado, assim como vetores.
- Caso entrar com dados de diversos *tipos*, ela se transforma numa matriz não numerica para comportar.
- Criar uma matriz:  

```
variavel <- matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
```

  - **data** = inserir dados, incluir listas ou vetores.  

```
data = c(vetor1,vetor2,vetor3)
```
  - **nrow** = numero de linhas.
  - **ncol** = numero de colunas.
  - **byrow** = lógica. Se FALSE (o padrão) a matriz é preenchida por colunas, caso contrário a matriz é preenchida por linhas.
  - **dimnames** = NULL ou uma lista de comprimento 2 fornecendo os nomes das linhas e colunas, respectivamente. Uma lista vazia é tratada como NULL e uma lista de comprimento um como nomes de linha.  

```
dimnames = list(c("lx","ly","lz"),c("c1","c2","c3"))
```
- Adicionando registros/linhas, usar a função `rbind()`.  

```
rbind(vetor1,vetor2,...)
```
- Adicionando campos/colunas, usar a função `cbind()`.  

```
cbind(vetor1,vetor2,...)
```
- Acessando dados dentro da matriz:
  - Acessando um elemento:  

```
variavel_da_matrix [linha, coluna]
```
  - Acessando uma linha:  

```
variavel_da_matrix [linha,]
```
  - Acessando uma coluna:  

```
variavel_da_matrix [, coluna]
```

### 10.2 Tabela de dados - Data.frames

- Na tabela de dados podemos ter dois ou mais *tipos* de dados misturados, sendo uma coluna/campo para cada tipo.
- Criando um data.frame a partir de vetores:  

```
variavel_df <- data.frame(nome_vetor1,nome_vetor2,...)
```

  - O nome das variaveis dos vetores serão o nome dos campos/colunas.

- Acessando dados dentro do `data.frame`:
  - Acessando um campo inteiro, pode usar o nome do campo:
    - \* retorna o vetor dos dados, com o *tipo* `data.frame`.  
`nome_do_df[numero_da_coluna]`
    - \* retorna o vetor com os dados, com o *tipo* original da coluna.  
`nome_do_df$nome_do_campo`
  - Acessando um elemento:  
`nome_do_df[linha, coluna]`
  - Acessando uma linha:  
`nome_do_df[linha,]`
- Excluindo uma coluna:  
`nome_do_df$nome_coluna <- NULL`
- Inserindo uma nova coluna:  
`nome_do_df$nome_nova_coluna <- valor`

### 10.3 Visualizar dados matriz e data.frame - VIEW

- Para apresentar os dados no console, basta apenas chamar o nome da variável com o dado armazenado.  
`nome_da_variavel`
- A função **VIEW** é usada para visualização de dados em formato planilha.  
`VIEW(variavel_matriz/data.frame)`

## 11 Aula 14 e 15 - Filtros

- Acessar, extrair e modificar dados em variáveis.

### 11.1 Vetor

- Acessar dados:
  - Acessar um elemento em determinada posição.  
`nome_vetor[posição_do_elemento]`
  - Acessar todos os elementos, menos determinada posição.  
`nome_vetor[ -posição_do_elemento_excluido]`
  - Dados entre posições (da *posição1* a *posição2*, *posição1* e *posição2* inclusas).  
`nome_vetor[posição1:posição2]`
  - Acessando a posição final do vetor com ajuda da função **length()** (devolve o tamanho do vetor, ou seja, a última posição).  
`nome_vetor[ posição1:length(nome_vetor)]`
  - Acessando posição determinada através de operação matemática.  
`nome_vetor[ length(nome_vetor)-2:length(nome_vetor)]`
  - Acessando dados através de condições.  
`nome_vetor[nome_vetor==condição]`  
`nome_vetor[nome_vetor!=condição]`  
`nome_vetor[nome_vetor>condição]`  
`nome_vetor[nome_vetor<=condição]`
- Observação:
  - A função **length()** devolve o tamanho do vetor, ou seja, a última posição.

## 11.2 Data.Frames

- Acessar dados:
  - Acessando coluna/campo pelo numero da posição:  
`df[numero_da_coluna]`
  - Acessando registro pelo numero da linha:  
`df[numero_da_linha,]`
  - Acessando intervalo de colunas (posições inclusas):  
`df[numero_da_coluna_inicial:numero_da_coluna_final]`
  - Acessando intervalo de linhas (posições inclusas):  
`df[numero_da_linha_inicial:numero_da_linha_final,]`
  - Excluindo uma coluna do filtro:  
`df[-numero_da_coluna]`
  - Juntando linha e coluna para acessar dados:  
`df[numero_linha,numero_coluna]`
  - Acessando determinado registro de intervalo de colunas:  
`df[numero_linha,numero_da_coluna_inicial:numero_da_coluna_final]`
  - Acessando registro, excluindo determinada coluna:  
`df[numero_linha,-numero_coluna]`
  - Acessando intervalo de linhas e colunas:  
`df[numero_da_linha_inicial:numero_da_linha_final,numero_da_coluna_inicial:numero_da_coluna_final]`
  - Excluindo diversas colunas do filtro:  
`df[c(-numero_coluna1,-numero_coluna2,-numero_coluna3,...)]`
- Filtro por nome da coluna:
  - Acessando um elemento da coluna:  
`df$nome_coluna[numero_registro]`
  - Acessando um intervalo de registro:  
`df$nome_coluna[numero_registro_inicial:numero_registro_final]`
  - Acessar determinados registros usando operador lógico:  
`df$nome_coluna[valor==condição]`

## 12 Aula 16 - Condicionais IF, FOR e WHILE

### 12.1 IF

- **IF**

- “Se algo é verdade, faça isso...”
- Se uma *condição* for verdadeira algo deve ser executado.
- Sintaxe:

```
if (condição){  
  operação  
}
```

- **ELSE**

- “Se algo é verdade, faça isso..., **senão** faça aquilo outro”
- O **else** faz algo se a condição do **if** não for atendido.
- Sintaxe:

```
if (condição){  
  operação1  
} else {  
  operação2  
}
```

- **ELSE IF**

- “Se algo é verdadeiro faça isso ..., **caso** determinada *condição* faça aquilo outro, **senão** ...”
- Cria **casos** com *condições* a serem avaliadas, depois que anterior é analisada.
- Podem existir varios **else if**.
- Sintaxe:

```
if (condição1){  
  operação1  
} else if (condição2){  
  operação2  
} else {  
  operação3  
}
```



## 12.2 Loop FOR

- “Para cada posição do *vetor* faça isso...”
- **FOR** é usado para percorre um *vetor*.
- A variável *i* é incrementada a cada loop.
- Sintaxe:

```
for (i in vetor){  
  operação  
}
```

## 12.3 Loop WHILE

- “Enquanto a condição for verdadeira, execute...”
- Executa o loop enquanto uma determinada *condição* (operação lógica) for verdadeira.
- Sintaxe:

```
while(condição){  
  operação  
}
```

## 13 Aula 17 - Função e print

### 13.1 Função - function

- Criar funções tem o objeto de evitar ter que reescrever varias vezes as mesmas instruções, abreviar esse trabalho.
- Uma função é um conjunto de instruções organizadas em conjunto para executar uma tarefa específica.
- Podem ser passados diversos *parametros* para dentro da função, basta informar isso na criação da função. *Parametros* são valores ou variaveis que são colocados para dentro da função.
- As variaveis criadas e usadas dentro de **functions** (funções) são locais, não existem fora da função, deixam de existir depois que a função é executada.
- A instrução **return()** devolve o *argumento* para fora da função.
- Sintaxe:

```
nome_da_função <- function(parametro_1,...){  
  instruções  
  print(variavel)  
  return (variavel_argumento)  
}
```

### 13.2 Print

- A função **print(argumento)** imprime seu argumento na tela.
- É uma função útil para usar dentro de **functions** (funções), dado que as variaveis dentro de **function** são locais.  
Sintaxe:  
**print(variavel)**

## **14 Andamento dos Estudos**

### **14.1 Assunto em andamento**

Concluído.