Curso de R básico

Sergio Pedro R Oliveira

2023-09-15

# 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 codigo 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 codigo

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

## 5.4 Limpar a tela do console

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

## 5.5 Atribuições

(Tipagem automatica/dinâmica)

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

## 5.6 Tipagem de variaveis

* Tipagem das variaveis automatica/dinâmica.
* Erro ao fazer operações com variaveis 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 possivel 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 variaveis.
* 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 metodos aplicados a cada campo/coluna.
* O retorno da função, no geral, retorna diversos metodos aplicados aos dados, tais como:  
  + valor mínimo
  + 1º quantil
  + valor da mediana
  + valor da media
  + 3º quantil
  + valor máximo
* Sintaxe:  
  summary(nome\_variavel)

## 6.4 Descobrir a classe de uma variavel - **class**

* A função class() retorna a que classe do objeto do argemunto pertence.
* Basicamente diz se o objeto é numerico, 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 vertor, dos campos com o tipo e tamanho.
* Sintaxe:  
  str(argumento)

# 7 Aula 05 - Operadores

## 7.1 Operadores básicos

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

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 logica é *TRUE* (verdadeiro) ou *FALSE* (falso). Ou em termos numericos 1 (verdadeiro), 0 (falso).
* No caso da negação (NOT), ele vem antes da operação como um todo.  
  + Exemplo:

# 8 Aula 06 á 09 - Tipo de dados

## 8.1 Tipos de dados

### 8.1.1 Numerico - **numeric**

* O **R** converte automaticamente variaveis númericas para tipo numeric.
* São variaveis contando números tanto inteiros (*int*), quanto *float*.
* Variaveis do tipo numeric podem realizar operações matemáticas.
* É possivel 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 truncamente. CUIDADO!!!
  + Para fazer arredondamento, usar a função round(variavel, qtd\_decimal).

### 8.1.2 Caractere e String - **character**

* O **R** entende como variaveis 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 possivel fazer operações matematicas nele. Porem 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 “niveis”.
  + A função summary(vetor), numera o numero de elementos no mesmo “nivel”.

### 8.1.4 Lógico

* É possivel salvar dentro de uma variavel 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 numerico *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 perimite 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 variaveis para outros tipos.
* Sintaxe (exemplo):  
  as.factor(variavel)  
  + converte a variavel de um tipo qualquer para uma variavel 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 variavel.
* O retorno é “*TRUE*” ou “*FALSE*”.
* Sintaxe (exemplo):  
  is.factor(variavel)  
  + Retorna “*TRUE*” se verdadeiro, o tipo da variavel for factor.
  + Ou retorna “*FALSE*” se falso, o tipo da variavel não for factor.

# 9 Aula 10 e 11 - Vetores e Listas

## 9.1 Vetor

* Vetores são variaves 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 numero de elementos dentro do vetor.
* Caso seja inserido um dado do tipo *character* para um vetor do tipo *numerico*, ele converte todo o vetor em tipo *character* (todos os elementos em *character*).
* Sintaxe (exemplo):  
  c(10,5,8,...)
* Acessando um valor dentro de uma vetor, uso a notaçao:  
  nome\_do\_vetor[posição]

## 9.2 Lista

* Listas sao os objetos R que contem elementos de diferentes tipos (diferente do vetor), como \_ numeros, strings, vetores e outra lista dentro dela.
* Uma lista tambem pode conter uma matriz ou uma funcao como seus elementos.
* *Strings* e *caracteres* dentro da lista devem ter seus valores entre aspas (““).
* A lista e criada usando a funçãoo list().
* Sintaxe (exemplo):  
  list(azul, 10, c(5,8,9), ...)
* Acessando um valor dentro de uma lista.  
  + Notaçao:  
    nome\_da\_lista [[posição]]
  + Caso tenha uma lista ou vetor dentro de algum elemento:  
    nome\_da\_lista [[posição]][posição\_dento\_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 veriaveis 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 penas chamar o nome da variavel 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 variaveis.

## 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 ultima posição).  
    nome\_vetor[ posição1:length(nome\_vetor)]
  + Acessando posição determinada atraves de operação matematica.  
    nome\_vetor[ length(nome\_vetor)-2:length(nome\_vetor)]
  + Acessando dados atraves 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\_lina,-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 deversas 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){  
  operção  
  }
* **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** derterminada *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 variavel *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* (operção lógica) for verdadeira.
* Sintaxe:

while(condição){  
operção  
}

# 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.