Ciência de Dados com R

Aulas 1 e 2 | Introdução ao R

Izabel Nolau
nolau@dme.ufrj.br

Ambientação ao R

Sobre o R

- O R é um software livre de linguagem interativa para computação estatística
- Possui uma vasta biblioteca de funções matemáticas, técnicas simples e sofisticadas para análise e visualização de dados
- Os usuários de R podem compartilhar seus códigos, contribuindo para o crescimento da comunidade!

Sobre o R

Vantagens:

- facilidade no manuseio e armazenamento de dados
- operadores vetoriais e matriciais
- ampla variedade de ferramentas para análise de dados
- linguagem de programação simples e eficiente
- integração com outros softwares
- facilidades gráficas
- calculadora

Desvantagens:

- pode ser lento (e.g., simulações intensivas)

Sobre o R

Vantagens:

- facilidade no manuseio e armazenamento de dados
- operadores vetoriais e matriciais
- ampla variedade de ferramentas para análise de dados
- linguagem de programação simples e eficiente
- integração com outros softwares
- facilidades gráficas
- calculadora

Desvantagens:

- pode ser lento (e.g., simulações intensivas)

RGui versus RStudio

- RGui é uma interface gráfica que vem com a instalação padrão do R para o usuário (Graphical User Interface - GUI)
- RStudio é um Ambiente de Desenvolvimento Integrado (IDE) com várias funcionalidades e gratuito
- Algumas vantagens do RStudio em relação ao RGui são:
  - Highlight do código
  - Autocomplete
  - Match automático de parenteses e chaves
  - Interface intuitiva para objetos, gráficos e script

RGui versus RStudio

Vamos abrir o RGui e o RStudio!

Ambiente

- Trabalha com linhas de comando:

você manda $\rightarrow$ ele executa!

- Os comandos podem ser digitados em duas janelas:

Console

(interface que interpreta o código da linguagem R)

ou *Script*

(arquivo usado para escrever, editar e salvar um código)

Operações Básicas

- `+` : soma
- `-` : subtração
- $*$: multiplicação
- `/` : divisão
- $\wedge$: potenciação
- `;` : separa diferentes comandos em uma mesma linha
- `#` : para inserir comentários

Funções e Pacotes

- Uma função é um **conjunto de instruções organizadas** para executar uma tarefa específica
- O R possui um grande número de funções embutidas e o usuário pode criar suas próprias funções
- Os pacotes são **coleções de funções** e bases de dados desenvolvidos pela comunidade
- Eles aumentam o poder do R melhorando as funcionalidades básicas do R ou adicionando novas

Funções do R Base

- `setwd(·)` : muda o diretório de trabalho (`"C:\\..."` ou `"C:/..."`)
- `getwd()` : mostra o diretório de trabalho
- `dir()` : lista todos os arquivos na pasta de trabalho atual
- `ls()` : lista o nome dos objetos criados na sessão atual
- `search()` : lista todos os pacotes carregados
- `library(·)` : ativa biblioteca/pacote instalado
- `rm(·)` : remove os objetos entre parênteses
- `rm(list=ls())` : remove todos os objetos, limpando a memória

Funções Matemáticas do R Base

- `sqrt(·)` : raiz quadrada
- `abs(·)` : valor absoluto
- `exp(·)` : exponencial
- `log(·, base = b)` : logaritmo em qualquer base `b`
- `factorial(n)` : $n!$

Objetos do R

Objetos

- O R é uma linguagem orientada a objetos, aos quais são atribuídos valores e expressões

- Atribuímos a um objeto de nome `x` o valor `10` fazendo

  `x <- 10` ou `x = 10`

- O nome do objeto precisa começar com uma letra, pode conter números, ponto, underline... Mas não pode conter símbolos como vírgula, ponto-e-vírgula ou espaço

- As informações ficam armazenadas na mémoria do computador, podendo ser acessadas, geradas, salvas, apagadas e manipuladas de diversas formas

Classes de Objetos

As classes mais simples de objetos são:

- `integer`: conjunto dos números inteiros, como 1 e 327
- `numeric`: conjunto dos números reais, como 0.32 e 4.5
- `logical`: TRUE e FALSE
- `character`: caracteres num geral, como “a”, “b”, “c”, “@”, “#”, “$”, “1”, “2”,...

R é uma linguagem sensível!

Tipos de Objetos

- Cada linguagem de programação possui seus próprios tipos de objetos para armazenar informações
- Esses objetos são então atribuídos à variáveis, para que sejam feitas operações
- Alguns dos objetos mais comumente utilizados são:
  - Vetores
  - Matrizes
  - DataFrames
  - Listas
  - ⋮

Vetores

- Os vetores são um dos tipos mais básicos de objetos na programação R
- Eles armazenam tipos de dados homogêneos como números inteiros e decimais, caracteres...
- Uma variável de elemento único também é um vetor

Conjuntos de elementos de uma mesma natureza!

Criando Vetores

Concatenação: `c(···)`

- `vetor.numerico = c(1, 4, -0.3, 9) # vetor numérico`
- `vetor.numerico[1] # primeiro elemento do vetor`
- `vetor.numerico[-1] # vetor sem o primeiro elemento`
- `vetor.numerico[1] = NA # vetor recebe um missing`
- `vetor.logico = c(TRUE, FALSE, TRUE) # vetor lógico`
- `vetor.caracter = c("oi", "olá", "tchau") # vetor de caracteres`

Quaisquer operações com vetores e matrizes é feita elemento-a-elemento

Criando Vetores

Sequência: `seq(from = ..., to = ..., by = ...)`

- `-10:10 # {-10, 9,…, 9, 10}`
- `seq(from = 1, to = 10, by = 2) # {1, 3, 5, 7, 9}`
- `seq(from = 5, to = 15, length = 3) # {5, 10, 15}`

Sequências permitem criar gráficos de funções conjuntamente com a função `plot(x)`

Criando Vetores

Repetição: `rep(···, times = ···, each = ···)`

- `rep(0, times = 4) # {0, 0, 0, 0}`
- `rep(c(1, 2, 3), times = 4) # {1, 2, 3,…, 1, 2, 3}`
- `rep(c(1, 3), times = c(8, 9)) # {1,…,1,3,…,3}`
- `rep(c(1, 4), each = 5) # {1,…,1,4,…,4}`
- `rep(1:4, each = 2, times = 3)`
  `# {1, 1, 2, 2, 3, 3, 4, 4,…, 1, 1, 2, 2, 3, 3, 4, 4}`
- `rep(c("a", "b"), times = 10) # {"a", "b",…,"a", "b"}`

Operações com Vetores

- `length(x)`: tamanho do vetor `x`
- `sort(x)`: ordena `x` em ordem crescente
- `rank(x)`: posições de cada elemento do vetor `x` ordenado
- `round(x, digits = 2)`: arredonda os valores do vetor `x`
- `min(x)`: elemento de valor mínimo de `x`
- `max(x)`: elemento máximo de `x`
- `which.min(x)` e `which.max(x)`: posição do mínimo e do máximo
- `sum(x)`: somatório dos elementos de `x`
- `prod(x)`: produtório dos elementos de `x`
- `unique(x)`: vetor com elementos não repetidos de `x`
- `x[x >= 10]`: elementos de `x` maiores ou iguais a `10`
- `which(x >= 10)`: posição dos elementos de `x` maiores ou iguais a `10`

Matrizes

- Elas também armazenam tipos de dados homogêneos como números inteiros e decimais, caracteres...
- Agora, os dados são armazenados em duas dimensões

Conjuntos de elementos de uma mesma natureza
organizados em linhas e colunas !

Criando Matrizes

Matriz: `matrix(···, nrow = ···, ncol = ···, byrow = ···)`

- `m = matrix(1:16, nrow = 4, ncol = 4) # matriz 4x4 ↓`
- `m = matrix(1:16, nrow = 4, ncol = 4,byrow = TRUE) # matriz →`
- `m[1, 4] # linha 1 e coluna 4`
- `m[1, ] # linha 1`
- `m[ , 1] # coluna 1`
- `m[-1, ] # todas as linhas exceto linha 1`
- `m[ , -1] # todas as colunas exceto coluna 1`
- `m[c(1, 3), ] # linhas 1 e 3`
- `m[ , c(1, 3)] # colunas 1 e 3`
- `m[1:2, 1:3] # linhas 1 a 2 e colunas de 1 a 3`

Criando Matrizes

Concatenação de matrizes: `cbind(···)` e `rbind(···)`

- `M1 = matrix(1, nrow = 2, ncol = 2)`

- ```
  M2 = cbind(M1, c(2, 2))
  # acrescenta uma coluna a matriz M1
  ```

- ```
  M3 = cbind(M1, M2)
  # concatena as matrizes M1 e M2 horizontalmente
  ```

- ```
  M4 = rbind(M1, c(2, 2))
  # acrescenta uma linha a matriz M1
  ```

- ```
  M5 = rbind(M1, M4)
  # concatena as matrizes M1 e M4 verticalmente
  ```

Operações com Matrizes

- `dim(A)`: dimensão da matriz A
- `A%*%B`: multiplicação matricial
- `A*B`: multiplicação matricial elemento a elemento
- `2*A`: multiplicação de todos os elementos por 2
- `det(A)`: determinante de da matriz A
- `t(A)`: transposta da matriz A
- `rowSums(A)`: calcula a soma de cada linha da matriz A
- `colSums(A)`: calcula a soma de cada coluna da matriz A

DataFrames

- Armazenam dados tabulares bidimensionais
- Consistem em várias colunas, cada uma delas sendo um vetor
- As colunas podem ter diferentes classes de dados, diferentemente das matrizes

Similar às matrizes,
porém diferentes colunas podem
possuir elementos de naturezas diferentes!

Criando DataFrames

DataFrame: `data.frame(···)`:

```
dados = data.frame(c(27, 18, 23), c("Ana", "Eva", "Mel"))
# criando um DataFrame

names(dados) = c("Idade", "Nome")
# nomeando as colunas do DataFrame

dados = data.frame(Idade = c(27, 18, 23),
                   Nome = c("Ana", "Eva", "Mel"))
# criando um DataFrame de outra maneira

attach(dados)
# cria variáveis com os nomes das colunas do DataFrame
```

Listas

- Armazenam tipos de dados heterogêneos, incluvise os objetos apresentados anteriormente ou outras listas
- Esses os dados são armazenados em uma dimensão

Generalização de vetores,
representa uma coleção de objetos!

Criando Listas

Lista: `list(···)`

```
lista = list(c(1, 2, 3), matrix(1, 4, 3), "compras")
# lista com diferentes objetos

lista[[1]]
# acessa o primeiro objeto

lista[1]
# acessa o primeiro objeto, mas como uma lista

lista = list(a = c(1, 2, 3), b = matrix(1, 4, 3), c = "compras")
# lista com diferentes objetos nomeados

lista$a
# objeto chamado a
```

Estruturas de Controle

Estruturas de Controle

As estruturas de controle no R permitem controlar a execução do programa, dependendo das condições de execução. Algumas estruturas comuns são:

- `if, else if, else`: testa uma condição
- `for`: executa um loop um número fixo de vezes
- `while`: executa um loop enquanto uma condição for verdadeira
- `repeat`: executa um loop infinito
- `next`: pula uma iteração de um loop
- `break`: interrompe a execução de um loop
- `return`: sai de uma função

A maioria das estruturas de controle não é usada em sessões interativas, mas sim ao escrever funções.

Comandos `if`, `else if` e `else`

- Podemos comparar números através das expressões:

  `==` (igual) `!=` (diferente) `<` (menor) `>` (maior)
  `<=` (menor ou igual) `>=` (maior ou igual)

- Os operadores lógicos `|` (“ou”) e `&` (“e”) são úteis quando precisamos verificar mais de uma condição simultaneamente

- O comando `if` é utilizado para validar uma condição e então executar o código de acordo com o resultado:

```
if(condição verdadeira){
  comandos
}
```

Comandos `if`, `else if` e `else`

- Os comandos `else if` e `else` são utilizados juntamente com o comando `if`

```
if(condição verdadeira 1){
  comandos1
} else if (condição verdadeira 2){
  comandos2
} else{
  comandos3
}
```

Comando `for`

- O comando `for` cria um ciclo a partir de um contador

```
for(contador in sequência){
   comandos
}
```

Apply

- É uma função que retorna um vetor, matriz ou lista de valores obtidos aplicando uma função às margens de uma matriz ou um array

```
apply(X, MARGIN, FUN,...)
```

- `X`: uma matriz ou array
- `MARGIN`: um vetor dando os subscritos nos quais a função será aplicada: 1 indica linhas, 2 indica colunas, c (1, 2) indica ambos
- `FUN`: a função a ser aplicada

Criando Funções

A sintaxe básica de uma função do R é:

```
nome.da.funcao = function(argumentos){
  comandos
}
```

As diferentes partes de uma função são:

- Nome da Função: nome pelo qual a função é armazenada no R como um objeto
- Argumentos: valores recebidos pela função
- Corpo da Função: contém uma coleção de instruções que definem o que a função faz
- Valor de retorno: última expressão no corpo da função a ser avaliada e retornada

Criando Funções

- Para que a função retorne algum valor e não simplesmente execute uma ação, é necessário usar o comando `return()`, ou simplesmente colocar a variável que se quer retornar

Bons estudos! ☺