Insper EDUCAÇÃO EXECUTIVA ONLINE

Live Learning AO VIVO

Programação para Não Programadores

Aula 2

Prof. Magno Severino e Prof. Marina Muradian

15/04/2021

Objetivos de aprendizagem

- Conhecer as estruturas de dados básicas do `R`.
- Diferenciar tipos de estruturas de dados.
- Manipular estruturas de dados.

Revisão

Previamente, vimos alguns tipos de vetores para valores lógicos, caracteres e números.

Nessa seção, utilizaremos técnicas de manipulação de vetores e introduziremos o caso multidimensional para trabalhar com dataframes (o equivalente à planilhas).

Abaixo relembramos as operações que já foram feitas com vetores

```
10:5                    #sequência de números de 10 até 5
```

```
## [1] 10  9  8  7  6  5
```

```
c(1, 2:5, c(6, 7), 8)   #valores concatenados em um único vetor
```

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

Vetores

Existem funções para criar vetores de um determinado tipo e com tamanho específico. Todos os elementos deste vetor terá valor zero, `FALSE`, um caracter vazio, ou o equivalente à *nada/vazio* para aquele tipo. Veja abaixo duas maneiras de definir um vetor.

```
vector("numeric", 5) #cria um vetor numérico de 5 elementos
```

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

```r
numeric(5) #equivalente ao comando acima
```

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

```r
vector("logical", 5) #cria um vetor lógico de 5 elementos
```

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

```r
logical(5) #equivalente ao comando acima
```

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

```r
vector("character", 5) #cria um vetor de caracteres de 5 elementos
```

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

```r
character(5) #equivalente ao comando acima
```

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

Sequências

Podemos criar sequências mais gerais que aquelas criadas com o operador `:`. A função `seq` te permite criar sequências em diferentes maneiras Veja abaixo.

```r
seq(3, 12) #equivalente à 3:12
```

```
##  [1]  3  4  5  6  7  8  9 10 11 12
```

```r
seq(3, 12, 2) #o terceiro argumento indica a distância entre os elementos na lista.
```

```
## [1]  3  5  7  9 11
```

```r
seq(0.1, 0.01, -0.01)
```

```
##  [1] 0.10 0.09 0.08 0.07 0.06 0.05 0.04 0.03 0.02 0.01
```

Tamanhos

Todo vetor tem um tamanho, um número não negativo que representa a quantidade de elementos que o vetor contém. A função `length` retorna o tamanho de um dado vetor.

```r
length(1:5)
```

```
## [1] 5
```

```r
length(c(TRUE, FALSE, NA)) #observe que elementos NA também são contados no tamanho do vetor
```

```
## [1] 3
```

```r
frase <- c("Observe", "o", "resultado", "dos", "comandos", "abaixo")

length(frase)
```

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

```r
nchar(frase)
```

```
## [1] 7 1 9 3 8 6
```

Indexando vetores

A indexação é útil quando queremos acessar elementos específicos de um vetor. Considere o vetor

```r
x <- (1:5) ^ 2
```

Abaixo, três métodos de indexar os mesmos valores do vetor `x`.

```r
x[c(1, 3, 5)]

x[c(-2, -4)]

x[c(TRUE, FALSE, TRUE, FALSE, TRUE)]
```

Cuidado, acessar um elemento fora do tamanho do vetor não gera um erro no R, apenas `NA`.

```r
x[6]
```

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

A função which retorna a posição em que uma condição é verdadeira em um vetor

```r
x > 10
```

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

```r
which(x > 10)  #observe a diferenca no resultado deste comando, se comparado com o acima
```

```
## [1] 4 5
```

```r
which.max(x) #retrona a posição do maior elemento do vetor
```

```
## [1] 5
```

```r
which.min(x) #retrona a posição do menor elemento do vetor
```

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

Matrizes

Uma matriz é o equivalente à um vetor, porém em duas dimensões. Abaixo, um exemplo de definição de uma matriz com 4 linhas e 3 colunas (total de 12 elementos).

```r
?matrix
```

```r
uma_matriz <- matrix(
      1:12,
      nrow = 4, #ncol = 3 gera o mesmo resultado. Verifique!
)

class(uma_matriz)
```

```
## [1] "matrix"
```

```r
uma_matriz
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

Por padrão, ao criar uma matrix, o vetor passado como primeiro argumento preenche a matrix por colunas. Para preencher a matrix por linhas, basta especificar o argumento `byrow=TRUE`.

A função `dim` retorna um vetor de inteiros com as dimensões da variável.

```r
dim(uma_matriz)
```

```
## [1] 4 3
```

```r
nrow(uma_matriz) #retorna o número de linhas da matrix
```

```
## [1] 4
```

```r
ncol(uma_matriz) #retorna o número de colunas da matrix
```

```
## [1] 3
```

```r
length(uma_matriz) #retorna o número de elementos da matrix
```

```
## [1] 12
```

Indexação

A indexação de matrizes funciona de maneira similar à de vetores, com a diferença que agora precisam ser especificadas mais de uma dimensão.

```
uma_matriz[1, 1:3] #todos elementos da primeira linha
```

```
## [1] 1 5 9
```

```
uma_matriz[1, ] #todos elementos da primeira linha
```

```
## [1] 1 5 9
```

```
uma_matriz[, c(2, 3)] #todos elementos da segunda e terceira colunas
```

```
##      [,1] [,2]
## [1,]    5    9
## [2,]    6   10
## [3,]    7   11
## [4,]    8   12
```

A indexação pode ser feita também indicando os elementos que **não** devem ser selecionados. Basta usar o sinal negativo `-`.

```
uma_matriz[1, -2] #todos elementos da primeira linha, com excessão da segunda coluna
```

```
## [1] 1 9
```

```
uma_matriz[-1, ] #todas as linhas da matriz, com excessão da primeira
```

```
##      [,1] [,2] [,3]
## [1,]    2    6   10
## [2,]    3    7   11
## [3,]    4    8   12
```

Data frames

Vetores e matrizes contém elementos que são todos do mesmo tipo. Um dataframe permite armazenamento de diferentes tipos de dado em uma única variável. A grosso modo, dataframes são como uma tabela no Excel. Abaixo, um exemplo de criação de dataframe.

```
(um_data_frame <- data.frame(
      x = letters[1:5],       #coluna de caracteres
      y = rnorm(5),           #coluna de numeros
      z = runif(5) > 0.5      #coluna de logicos
))
```

```
##   x          y     z
## 1 a  1.7342511  TRUE
## 2 b -0.6651782 FALSE
## 3 c -1.5986199 FALSE
## 4 d -0.8859779  TRUE
## 5 e  0.4541187  TRUE
```

```r
class(um_data_frame)
```

```
## [1] "data.frame"
```

```r
rownames(um_data_frame) #nome das linhas do dataframe
```

```
## [1] "1" "2" "3" "4" "5"
```

```r
colnames(um_data_frame) #nome das colunas do dataframe
```

```
## [1] "x" "y" "z"
```

```r
length(um_data_frame) #retorna o numero de colunas do dataframe (diferente de matriz)
```

```
## [1] 3
```

```r
ncol(um_data_frame) #numero de linhas do dataframe
```

```
## [1] 3
```

```r
nrow(um_data_frame) #numero de colunas do dataframe
```

```
## [1] 5
```

```r
dim(um_data_frame) #dimensao do dataframe
```

```
## [1] 5 3
```

Indexação de dataframes

Os comandos abaixo selecionam o segundo e o terceiro elementos das duas primeiras colunas do dataframe criado anteriormente.

```r
um_data_frame[2:3, -3]
```

```
##   x          y
## 2 b -0.6651782
## 3 c -1.5986199
```

```r
class(um_data_frame[2:3, -3]) #observe que o resultado é um dataframe
```

```
## [1] "data.frame"
```

```r
um_data_frame[c(FALSE, TRUE, TRUE, FALSE, FALSE), c("x", "y")]
```

```
##   x          y
## 2 b -0.6651782
## 3 c -1.5986199
```

```r
um_data_frame[2:3, 1]
```

```
## [1] b c
## Levels: a b c d e
```

```r
class(um_data_frame[2:3, 1]) #como selecionamos um vetor, a classe é a mesma do elemento selecionado
```

```
## [1] "factor"
```

É possivel selecionar uma coluna pelo seu nome, e também um subconjunto de seus elementos.

```r
um_data_frame$x #seleciona a coluna x
```

```
## [1] a b c d e
## Levels: a b c d e
```

```r
um_data_frame$x[2:3] #seleciona os elementos 2 e 3 da coluna x
```

```
## [1] b c
## Levels: a b c d e
```

Manipulação de dataframes

Considere o novo dataframe abaixo.

```r
novo_data_frame <- data.frame( #mesmas colunas que o dataframe anterior, ordem diferente
  z = rlnorm(5), #números distribuídos seguindo a distribuição lognormal
  y = sample(5), #número 1 a 5 distribuidos em uma ordem aleatória
  x = letters[3:7])

rbind(um_data_frame, novo_data_frame) #qual a dimensão deste elemento?
```

```
##    x          y         z
## 1  a  1.7342511 1.0000000
## 2  b -0.6651782 0.0000000
## 3  c -1.5986199 0.0000000
## 4  d -0.8859779 1.0000000
## 5  e  0.4541187 1.0000000
```

```
## 6  c  3.0000000 0.7289481
## 7  d  1.0000000 0.7797106
## 8  e  4.0000000 0.3657587
## 9  f  5.0000000 1.2087237
## 10 g  2.0000000 1.7674061
```

```
cbind(um_data_frame, novo_data_frame) #qual a dimensão deste elemento?
```

```
##   x          y     z         z y x
## 1 a  1.7342511  TRUE 0.7289481 3 c
## 2 b -0.6651782 FALSE 0.7797106 1 d
## 3 c -1.5986199 FALSE 0.3657587 4 e
## 4 d -0.8859779  TRUE 1.2087237 5 f
## 5 e  0.4541187  TRUE 1.7674061 2 g
```

Para mesclar dois dataframes, devemos considerar uma coluna que contenha algum identificador único para cada elemento.

```
?merge
```

```
merge(um_data_frame, novo_data_frame, by = "x")
```

```
##   x        y.x   z.x       z.y y.y
## 1 c -1.5986199 FALSE 0.7289481   3
## 2 d -0.8859779  TRUE 0.7797106   1
## 3 e  0.4541187  TRUE 0.3657587   4
```

```
merge(um_data_frame, novo_data_frame, by = "x", all = TRUE) #o que o argumento all=TRUE faz?
```

```
##   x        y.x   z.x       z.y y.y
## 1 a  1.7342511  TRUE        NA  NA
## 2 b -0.6651782 FALSE        NA  NA
## 3 c -1.5986199 FALSE 0.7289481   3
## 4 d -0.8859779  TRUE 0.7797106   1
## 5 e  0.4541187  TRUE 0.3657587   4
## 6 f         NA    NA 1.2087237   5
## 7 g         NA    NA 1.7674061   2
```

Exercício

a) Instale a biblioteca `nycflights13` utilizando o comando abaixo.

```
install.packages("nycflights13")
```

b) Carregue no seu ambiente a biblioteca instalada.

```
library(nycflights13)
```

c) Utilizando os comandos abaixo, verifique o conteúdo dos dataframes `flights` e `airports`.

```
?flights
?airports
```

d) Filtre os voos que aconteceram em 25/01/2013 e armazene-os na variável `natal`.

e) Quantos voos partiram de Nova Iorque em 25/12/2013?

f) Obtenha um sumário da coluna `dep_delay`. Há dados faltantes? Se sim, remova-os.

g) Obtenha o nome do aeroporto de destino do voo com maior atraso de partida em 25/12/2013. *Dica: mescle os dados de `flights` com `airports`.*