Workshop R

JEST

29/03/2018

Fatores

Os fatores são estruturas de dados que são utilizadas para categorizar os dados e armazená-los por níveis.

```
qualidade_comida <- c("low", "high", "medium", "high", "low", "medium", "high")
comida <- factor(qualidade_comida)
levels(comida);  #Os niveis são por outras palavras os nossos valores distintos.
```

```
## [1] "high"   "low"    "medium"
```

```
table(comida)  #A função table() mostra-nos os nossos valores distintos e o número de ocorrências de ca
```

```
## comida
##   high    low medium
##      3      2      2
```

```
comida <- factor(qualidade_comida, levels = c("low","medium", "high")) # podemos também definir nós qua
comida
```

```
## [1] low    high   medium high   low    medium high
## Levels: low medium high
```

```
levels(comida)
```

```
## [1] "low"    "medium" "high"
```

```
francesinhaEcompanhia <- comida[1]
atenas <- comida[length(qualidade_comida)] #a função length() recebe como entrada, por exemplo, um vect
#É a qualidade do restaurante Atenas é melhor do que a do Francesinhas&Companhia?
atenas > francesinhaEcompanhia
```

```
## Warning in Ops.factor(atenas, francesinhaEcompanhia): '>' not meaningful
## for factors
```

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

Para sabermos qual dos restaurantes tem melhor qualidade, temos de primeiro ordenar os *levels* (níveis). Para isso temos de passar "ordered = TRUE" na função *factor()*(Outra maneira seria utilizar a função ordered que faz o mesmo que a função *factor()* mas já ordena os niveis por *default*). Ao ativarmos a ordem dos níveis, estes serão colocados por uma ordem definida pelo utilizador.

```
comida <- factor(qualidade_comida, levels = c( "low","medium", "high"),ordered = TRUE)
comida
```

```
## [1] low    high   medium high   low    medium high
## Levels: low < medium < high
```

```
levels(comida)
```

```
## [1] "low"    "medium" "high"
```

```r
francesinhaEcompanhia <- comida[1]
atenas <- comida[length(qualidade_comida)]
atenas > francesinhaEcompanhia
```

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

Listas

Um objecto do tipo *list* em R é uma estrutura de dados que nos permite agrupar variaveis de diferentes tipos num mesmo objecto.

```r
lista <- list(c("Macario","Grifo","Poço"),TRUE,c(2,5,12.4,23),"Modulo de Introdução")  #Criação de uma
lista
```

```
## [[1]]
## [1] "Macario" "Grifo"   "Poço"
##
## [[2]]
## [1] TRUE
##
## [[3]]
## [1]  2.0  5.0 12.4 23.0
##
## [[4]]
## [1] "Modulo de Introdução"
```

```r
lista_NC <- list(nomes=c("Macario","Grifo","Poço"),trabalhador=TRUE,c(2,5,12.4,23),mensagem="Modulo de
lista_NC  #Lista com nomes nos componentes
```

```
## $nomes
## [1] "Macario" "Grifo"   "Poço"
##
## $trabalhador
## [1] TRUE
##
## [[3]]
## [1]  2.0  5.0 12.4 23.0
##
## $mensagem
## [1] "Modulo de Introdução"
```

Acesso a componentes em Listas (Indexação)

```r
lista_NC[2:3] # retorna os componentes desde o index 2 até ao index 3
```

```
## $trabalhador
## [1] TRUE
##
## [[2]]
## [1]  2.0  5.0 12.4 23.0
```

```r
lista_NC[[3]] # retorna todos os elementos ou componentes do componente na posição 3 da lista
```

```
## [1]  2.0  5.0 12.4 23.0
```

```
lista_NC$nomes  # retorna todos os elementos ou componentes do campo "nomes"
```

```
## [1] "Macario" "Grifo"   "Poço"
```

```
lista_NC$nomes[1]
```

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

Exercícios

1) Dedifir uma variável x como uma *list* com dois campos: nome="Grifo" e idade="21".

```
## $nome
## [1] "Grifo"
##
## $idade
## [1] 21
```

2) Ver o conteúdo de x, acedendo ao campo idade com "$" e "[[]]";

```
## [1] 21
```

```
## $idade
## [1] 21
```

Matrizes

O R permite definir matrizes de uma forma bastante similar ao conceito de vetor, mas agora com duas dimensões de dados. Uma matriz é uma coleção de elementos do mesmo tipo (numérico, sequencias de caracteres ou lógico), organizados por linhas e colunas. Para criar uma matriz usamos a função *matrix()* que toma como argumentos um vector, número de linhas e colunas.

```
#Primeiro utilize a documentação para ver como pode utilizar os argumentos de entrada na função matrix(
mat <- matrix(1:15,3,5)
mat
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    7   10   13
## [2,]    2    5    8   11   14
## [3,]    3    6    9   12   15
```

Funcções *cbind()* e *rbind()*

A função *cbind()* permite-nos juntar vetores ou matrizes por colunas. O mesmo se repete para a função *rbind()* mas para linhas.

```
A <- matrix(1:4,2,2)
A
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
B <- matrix(4:8,2,2)
```

```
## Warning in matrix(4:8, 2, 2): data length [5] is not a sub-multiple or
## multiple of the number of rows [2]
```

```
B
```

```
##      [,1] [,2]
## [1,]    4    6
## [2,]    5    7
```

```
cbind(A,B)
```

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

```
rbind(A,B)
```

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

Indexação

```
procura_elementos <- matrix(1:50,byrow=TRUE,nrow=10)
procura_elementos
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]    1    2    3    4    5
##  [2,]    6    7    8    9   10
##  [3,]   11   12   13   14   15
##  [4,]   16   17   18   19   20
##  [5,]   21   22   23   24   25
##  [6,]   26   27   28   29   30
##  [7,]   31   32   33   34   35
##  [8,]   36   37   38   39   40
##  [9,]   41   42   43   44   45
## [10,]   46   47   48   49   50
```

```
procura_elementos[5,5]
```

```
## [1] 25
```

```
procura_elementos[1:3,4]
```

```
## [1]  4  9 14
```

```
procura_elementos[,3:ncol(procura_elementos)]
```

```
##       [,1] [,2] [,3]
##  [1,]    3    4    5
##  [2,]    8    9   10
##  [3,]   13   14   15
##  [4,]   18   19   20
##  [5,]   23   24   25
##  [6,]   28   29   30
##  [7,]   33   34   35
```

```
##  [8,]   38   39   40
##  [9,]   43   44   45
## [10,]   48   49   50
```

Funções com matrizes

```
dim(procura_elementos)   #retorna as dimensões da matriz
```

```
## [1] 10  5
```

```
mean(procura_elementos)  #retorna a media de todos os elementos da matriz
```

```
## [1] 25.5
```

```
sqrt(procura_elementos[1:5,1:5])  #faz a raiz quadrada sobre todos os elementos definidos no intervalo
```

```
##          [,1]     [,2]     [,3]     [,4]     [,5]
## [1,] 1.000000 1.414214 1.732051 2.000000 2.236068
## [2,] 2.449490 2.645751 2.828427 3.000000 3.162278
## [3,] 3.316625 3.464102 3.605551 3.741657 3.872983
## [4,] 4.000000 4.123106 4.242641 4.358899 4.472136
## [5,] 4.582576 4.690416 4.795832 4.898979 5.000000
```

```
apply(procura_elementos,1,sum) #ver documentação
```

```
##  [1]  15  40  65  90 115 140 165 190 215 240
```

```
mtx <- matrix(1:4,2,2)
t(mtx) #retorna a transposta da matriz
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

```
det(mtx) #retorna o determinante da matriz. Atenção: a matriz deve ser quadrada.
```

```
## [1] -2
```

Exemplo

```
precos <- c(1200,276,550,126.5,1050,241.5,60,13.8)
worten <- matrix(precos,byrow=TRUE,nrow=4)
colnames(worten) <- c("Produto","IVA")
rownames(worten) <- c("Televisão","Maquina de lavar","Computador","Aspirador")
worten
```

```
##                  Produto   IVA
## Televisão           1200 276.0
## Maquina de lavar     550 126.5
## Computador          1050 241.5
## Aspirador             60  13.8
```

```
Preco_com_IVA <- c(precos[1]+precos[2],precos[3]+precos[4],precos[5]+precos[6],precos[7]+precos[8])
worten <- cbind(worten,Preco_com_IVA)
worten
```

```
##                   Produto   IVA Preco_com_IVA
## Televisão            1200 276.0        1476.0
## Maquina de lavar      550 126.5         676.5
## Computador           1050 241.5        1291.5
## Aspirador              60  13.8          73.8
```

Multiplicação entre matrizes

```
A <- matrix(1:4,2,2)
B <- matrix(4:1,2,2)
C <- A%*%B
C
```

```
##      [,1] [,2]
## [1,]   13    5
## [2,]   20    8
```

Arrays

Um array é uma estrutura de dados que consegue armazenar dados em mais do que 2 dimensões. Por exemplo uma Matriz é por outras palavras um array bidimensonal, ou seja é um array com duas dimensões.

```
bidimensional <- array(1:12, c(4,3))  #Array bidimensional
bidimensional
```

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

```
tridimensinal <- array(1:12, c(4,3,2)) #Array Tridimensional
tridimensinal
```

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

Exercícios

2.1) Criar um vetor com valores de 1 a 6.

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

2.2) Traformar o vetor anterior numa matriz de dimensões 2x3.

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

2.3) Verificar dimensões da matriz

```
## [1] 2 3
```

2.4) Identificar o elemento que se encontra na linha 2, coluna 1.

```
## [1] 2
```

2.5) Calcular a soma dos elementos da linha 1

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

2.6) Criar um vector cujos elementos sejam as médias dos valores de cada coluna

```
## [1] 1.5 3.5 5.5
```

3.1) Criar um array tridimensinal com as dimensões 4, 3, 3 e os primeiros 36 numeros inteiros por ordem decrescente.

3.2) Calcular todos os elementos do array, cujo valor da segunda dimensão é 2.

```
## [1] 222
```