

## Exercícios 2

### 1 Factores

Em diversos contextos, pode ser necessário lidar com um tipo de dados que não seja numérico, mas que o seu valor possa ser avaliado qualitativamente.

Por exemplo, num inquérito feito aos alunos, considere a questão: “O que acha dos docentes de CEGI?”, tendo as seguintes hipóteses de escolha “a) Terríveis”, “b) Meh.”, “c) über-cool!”. Este tipo de dados é então denominado categórico ou nominal. Neste caso em particular, têm ainda uma relação de ordem:

“Terríveis” < “Meh.” < “über-cool!”.

Em **R**, existe um tipo de dados indicado para estes casos, mais conveniente e vantajoso do que apenas ter um vector de caracteres: **factor**.

1. Fez-se um levantamento de algumas características fisionómicas dos alunos de uma turma. Crie um vector de factores, `cor.olhos`, a partir do seguinte vector:

```
c("azul", "castanho", "castanho", "castanho", "verde", "azul", "castanho")
```

**Solução:**

Para criar um vector de factores, a partir de um vector existente, usa-se a função `factor()`.

```
> cor.olhos = factor(c("azul", "castanho", "castanho", "castanho", "verde",  
+ "azul", "castanho"))  
> cor.olhos  
[1] azul      castanho castanho castanho verde     azul      castanho  
Levels: azul castanho verde
```

2. Verifique os níveis (categorias) criados.

**Solução:**

```
> levels(cor.olhos)  
[1] "azul"      "castanho" "verde"
```

3. Um outro conjunto de alunos foi registado no seguinte vector:

```
c("castanho", "castanho", "azul")
```

Defina os factores para este vector, mantendo os mesmos níveis.

**Solução:**

Neste caso, para forçar que os níveis iniciais sejam respeitados, é necessário especificá-los através do argumento `levels`.

```
> grupo2 = factor(c("castanho", "castanho", "azul"), levels=levels(cor.olhos))
> grupo2
[1] castanho castanho azul
Levels: azul castanho verde
```

4. Considerando ainda os mesmos níveis, transforme em factores os dados de um terceiro grupo de alunos:

```
c("azul", "castanho", "cinzento", "castanho")
```

O que aconteceu ao elemento na 3ª posição?

**Solução:**

```
> grupo3 = factor(c("azul", "castanho", "cinzento", "castanho"),
+ levels=levels(cor.olhos))
> grupo3
[1] azul      castanho <NA>      castanho
Levels: azul castanho verde
```

A categoria "cinzento" não existe nos níveis que estão a ser impostos, logo o elemento correspondente é convertido para NA no vector de factores.

5. No vector `gen` registou-se o género dos alunos:

```
gen = c("m", "f", "m", "m", "f", "f", "m")
```

Crie uma tabela de contingência<sup>1</sup>, mostrando a cor dos olhos por sexo.

**Solução:**

```
> (tabela = table(gen, cor.olhos))
cor.olhos
gen azul castanho verde
f      1          1      1
m      1          3      0
```

Os vectores passados para a função `table` não têm que ser obrigatoriamente constituídos por factores. Neste caso, a tabela de contingência está a ser criada através de um vector com caracteres e outro com factores.

6. Crie duas tabelas de frequências marginais, uma mostrando o número total de homens e mulheres, outra mostrando as contagens pela cor dos olhos.

**Solução:**

Este problema tem duas soluções semelhantes. Uma consiste em repetir a solução da questão anterior, passando como argumento apenas um dos vectores. A outra solução, aqui mostrada, parte da tabela resultante da solução anterior.

---

<sup>1</sup>[https://pt.wikipedia.org/wiki/Tabela\\_de\\_conting%C3%Aancia](https://pt.wikipedia.org/wiki/Tabela_de_conting%C3%Aancia)

```
# número total de homens e mulheres
> margin.table(tabela,1)
gen
f m
3 4
# contagens pela cor dos olhos
> margin.table(tabela,2)
cor.olhos
azul castanho verde
2      4      1
```

7. Selecciona os primeiros três elementos do vector `cor.olhos`. O que aconteceu aos níveis? Como manter apenas os níveis dos elementos seleccionados?

#### Solução:

```
> cor.olhos[1:3]
[1] azul      castanho castanho
Levels: azul castanho verde
```

Ao indexar um ou mais itens de um vector de factores, os níveis são mantidos. Para forçar que sejam considerados apenas os níveis correspondentes aos elementos seleccionados, é necessário usar o argumento `drop`.

```
> cor.olhos[1:3, drop=TRUE]
[1] azul      castanho castanho
Levels: azul castanho
```

8. Por fim, guardou-se a informação contendo a altura dos alunos no vector `alturas`. Esta informação é qualitativa, mas tem relação de ordem.

```
alturas = c("baixo", "baixo", "médio", "alto", "alto", "médio", "médio")
```

Transforme em factores, mantendo a relação de ordem. Compare a altura entre dois alunos.

#### Solução:

```
> alturas = c("baixo", "baixo", "médio", "alto", "alto", "médio", "médio")
> alturas = factor(alturas, levels=c("baixo", "médio", "alto"), ordered=TRUE)
> alturas
[1] baixo baixo médio alto alto médio médio
Levels: baixo < médio < alto
> alturas[1] > alturas[2]
[1] FALSE
> alturas[1] > alturas[3]
[1] FALSE
> alturas[1] >= alturas[2]
[1] TRUE
> alturas[3] <= alturas[6]
[1] TRUE
```

Estas comparações não seriam possíveis caso não se tivesse usado o argumento `ordered`.

## 2 Matrizes

1. Uma matriz é uma extensão de um vector para duas dimensões. Na verdade, em **R**, um vector não tem dimensão. Considere o seguinte vector, que contém a pauta de uma disciplina:

```
notas = as.integer(rnorm(20, 12, 4))
```

Verifique as suas dimensões, usando a função `dim()`. Transforme-o numa matriz `pauta`, com 4 linhas e 5 colunas. Verifique agora as dimensões da matriz criada.

### Solução:

```
> dim(v)
NULL
> pauta = matrix(v, 4, 5)
> dim(pauta)
[1] 4 5
```

2. Junte nomes às linhas e às colunas da matriz `pauta`, em dois passos distintos.

### Solução:

```
> rownames(pauta) = paste(rep("Aluno", 4), 1:4)
> colnames(pauta) = letters[1:5]
> pauta
      a b c d e
Aluno 1  5 8 6 6 21
Aluno 2 13 15 12 19 11
Aluno 3 19 15 13 14 8
Aluno 4 11 11 4 8 13
```

3. Aceda aos elementos da matriz de diferentes formas:
  - duas posições
  - dois nomes
  - uma posição e um nome
  - uma linha inteira pela posição
  - uma coluna inteira pelo nome
  - primeiras 3 linhas, todas as colunas excepto a última

### Solução:

```

# 2ª linha, 3ª coluna
> pauta[2, 3]
[1] 12
# linha "Aluno 3", coluna "d"
> pauta["Aluno 3", "d"]
[1] 14
# 4ª linha, coluna "a"
> pauta[4, "a"]
[1] 11
# 3ª linha completa
> pauta[3, ]
a b c d e
19 15 13 14 8
# coluna "e" completa
> pauta[, "e"]
Aluno 1 Aluno 2 Aluno 3 Aluno 4
21      11      8      13
# primeiras 3 linhas, todas as colunas excepto a última (5ª)
> pauta[1:3, -5]
      a b c d
Aluno 1  5 8 6 6
Aluno 2 13 15 12 19
Aluno 3 19 15 13 14

```

4. Junte uma nova coluna à matriz. Verifique agora os nomes das colunas da matriz. Como adicionar um nome à coluna recentemente adicionada?

#### Solução:

```

> pauta = cbind(pauta, 10:13)
# a coluna é acrescentada mas fica sem nome atribuído
> pauta
      a b c d e
Aluno 1  5 8 6 6 21 10
Aluno 2 13 15 12 19 11 11
Aluno 3 19 15 13 14 8 12
Aluno 4 11 11 4 8 13 13
> colnames(pauta)
[1] "a" "b" "c" "d" "e" ""
> colnames(pauta)[6]
[1] ""
# aceder ao nome dessa coluna e modificá-lo
> colnames(pauta)[6] = "f"
> pauta
      a b c d e f
Aluno 1  5 8 6 6 21 10
Aluno 2 13 15 12 19 11 11
Aluno 3 19 15 13 14 8 12
Aluno 4 11 11 4 8 13 13

```

5. Junte uma nova linha à matriz, atribuindo-lhe logo um nome.

**Solução:**

```
> pauta = rbind(pauta, "Aluno 5"=sample(v,6))
> pauta
      a b c d e f
Aluno 1  5 8 6 6 21 10
Aluno 2 13 15 12 19 11 11
Aluno 3 19 15 13 14 8 12
Aluno 4 11 11 4 8 13 13
Aluno 5 6 19 5 21 8 8
```

Nota: a função `sample()` está a ser usada apenas para obter um conjunto de notas aleatoriamente, a partir do vector `v`.

6. Calcule a média dos valores em cada linha e em cada coluna.

**Solução:**

```
# por linha
> rowMeans(pauta)
      Aluno 1      Aluno 2      Aluno 3      Aluno 4      Aluno 5
9.333333 13.500000 13.500000 10.000000 11.166667
# por coluna
> colMeans(pauta)
      a      b      c      d      e      f
10.8 13.6 8.0 13.6 12.2 10.8
```

7. Experimente avaliar o resultado de algumas operações algébricas sobre a seguinte matriz:

```
ma = matrix(as.integer(runif(16, 1, 20)), 4, 4)
```

Exemplos:

- `ma + 1`
- `ma * 2`
- `ma * ma`
- `ma %*% ma`
- determinante, com a função `det()`
- transposta, com a função `t()`
- inverter, com a função `solve()`

**Solução:**

```
> ma = matrix(as.integer(runif(16, 1, 20)), 4, 4)
> ma
      [,1] [,2] [,3] [,4]
[1,]  11  13  14   1
```

```

[2,] 19 8 16 11
[3,] 11 5 18 18
[4,] 11 1 14 6
> ma + 1
      [,1] [,2] [,3] [,4]
[1,] 12 14 15 2
[2,] 20 9 17 12
[3,] 12 6 19 19
[4,] 12 2 15 7
> ma * 2
      [,1] [,2] [,3] [,4]
[1,] 22 26 28 2
[2,] 38 16 32 22
[3,] 22 10 36 36
[4,] 22 2 28 12
> ma %*% ma
      [,1] [,2] [,3] [,4]
[1,] 533 318 628 412
[2,] 658 402 836 461
[3,] 612 291 810 498
[4,] 360 227 506 310
> det(ma)
[1] 16476
> t(ma)
      [,1] [,2] [,3] [,4]
[1,] 11 19 11 11
[2,] 13 8 5 1
[3,] 14 16 18 14
[4,] 1 11 18 6
> solve(ma)
      [,1] [,2] [,3] [,4]
[1,] -0.04685603 0.12066035 -0.07137655 0.0007283321
[2,] 0.06724933 0.01335276 0.02731245 -0.1176256373
[3,] 0.04855547 -0.10949260 0.02603787 0.1145302258
[4,] -0.03860160 0.03204661 0.06554989 -0.0823015295

```

### 3 Arrays

Em **R**, as matrizes são uma extensão dos vectores para duas dimensões. Os arrays aumentam essa extensão para qualquer número de dimensões. Seguem as mesmas regras e possibilidades de indexação que os vectores e matrizes.

1. Crie um array com 50 elementos, distribuídos por 2 linhas, 3 colunas, e 5 camadas.

**Solução:**

```
array(1:50, dim=c(2, 3, 5))
```

2. Considere a matriz **pauta** da secção anterior. Junte várias cópias dessa matriz num array, como se, por exemplo, estivesse a agregar as pautas de diferentes anos, num só objecto.

### Solução:

```
# criar cópias da matriz pauta, com valores ligeiramente diferentes, só para ilustrar
pauta2014 = pauta - 1
pauta2015 = pauta + 1
pauta2016 = pauta * 0.8
pautas = array(c(pauta2014, pauta2015, pauta2016), dim=c(4, 5, 3))
```

3. Experimente aceder a elementos do array do mesmo modo que tentou para a matriz.

### Solução:

```
# notas de todos os alunos, na 2ª componente, em todos os anos
> pautas[, 2, ]
[,1] [,2] [,3]
[1,] 8 10 7.2
[2,] 14 16 12.0
[3,] 14 16 12.0
[4,] 8 10 7.2
# notas dos primeiros 3 alunos, em todas as componentes, ignorando o primeiro ano
> pautas[1:3, , -1]
, , 1

[,1] [,2] [,3] [,4] [,5]
[1,] 20 10 17 8 14
[2,] 14 16 12 7 14
[3,] 11 16 9 7 8

, , 2

[,1] [,2] [,3] [,4] [,5]
[1,] 15.2 7.2 12.8 5.6 10.4
[2,] 10.4 12.0 8.8 4.8 10.4
[3,] 8.0 12.0 6.4 4.8 5.6
```

## 4 Listas

As listas, ao contrário dos vectores, matrizes e arrays, permitem guardar elementos de diferentes tipos (e.g., "character", "numeric" e "logical") num único objecto. Assim, podem ser vistas como uma colecção ordenada de objectos.

Os objectos que constituem uma matriz são designados por componentes, são numerados, e podem ter um nome associado.

Considere, como exemplo, a lista **estudante**, que corresponde a uma ficha com os seguintes dados de um aluno:

**nro** 1999001

**nome** John Doe

**notas** 14, 3, 12, 10, 19



**inscrito** Sim

1. Crie a lista **estudante** com as informações do aluno.

**Solução:**

```
estudante = list(nro=1999001, nome="John Doe", notas=c(14, 3, 12, 10, 19),  
  inscrito=TRUE)
```

2. Visualize todos os componentes (ou atributos) da lista. Compare com o resultado da função **str()**, quando chamada com a mesma lista.

**Solução:**

```
> estudante  
$nro  
[1] 1999001  
  
$nome  
[1] "John Doe"  
  
$notas  
[1] 14  3 12 10 19  
  
$inscrito  
[1] TRUE  
  
> str(estudante)  
List of 4  
 $ nro      : num 2e+06  
 $ nome     : chr "John Doe"  
 $ notas    : num [1:5] 14 3 12 10 19  
 $ inscrito: logi TRUE
```

A função **str()** (*structure*) apresenta o conteúdo de um objecto de uma forma mais compacta e fácil de ler.

3. Aceda ao primeiro componente da lista, usando a mesma sintaxe que nos vectores. Verifique o tipo de objecto desse resultado, usando a função **typeof()**.

**Solução:**

```
> estudante[1]  
$nro  
[1] 1999001  
  
> typeof(estudante[1])  
[1] "list"
```

```
> estudante[[1]]  
[1] 1999001  
> typeof(estudante[[1]])  
[1] "double"
```

Na primeira forma de indexação, o resultado é uma sublista contendo apenas a primeira componente. Na segunda forma, com duplos parêntesis rectos, a indexação resulta no acesso ao conteúdo da primeira componente.

4. Corrija o nome do primeiro componente para "número".

**Solução:**

```
names(estudante)[1] = "número"
```

5. Recebeu mais um dado do aluno: tem 33 anos de idade. Actualize a sua ficha com essa informação.

**Solução:**

```
estudante$idade = 33
```

6. Acrescente ainda a seguinte informação à ficha do aluno, em apenas um passo:

**sexo** masculino

**país** "Maria" e "Zé"

**Solução:**

```
estudante = c(estudante, list(sexo="m", país=c("Maria", "Zé")))
```

Este exemplo ilustra a utilidade da função `c()`, mostrando que também serve para juntar objectos de outros tipos para além de numéricos. Assim, neste caso, esta função está a ser usada para criar uma lista e não um vector.

7. O aluno recebeu a nota de mais um exame: 10 val. Adicione essa nota às restantes notas.

**Solução:**

```
estudante$notas = c(estudante$notas, 10)
```

8. Quantos componentes já tem a ficha do aluno?

**Solução:**

```
> length(estudante)
[1] 7
```

## 5 Data Frames

A estrutura de dados `data.frame` é das mais usadas em **R**, em boa parte, devido à sua versatilidade no tipo de objectos suportado e na indexação: é um tipo de dados derivado das listas, suportando elementos de diferentes tipos num só objecto; e adiciona as funcionalidades de indexação dos vectores.

É mais usado para guardar tabelas de dados, sendo parecido com uma matriz com nomes nas linhas e nas colunas, mas suportando elementos de diferentes tipos. Este tipo de tabela pode ser visto como uma tabela de uma base de dados, ou de uma folha de cálculo, em que cada linha corresponde a um registo diferente.

1. Transforme a matriz `pauta`, criada na secção 2, num `data.frame`, e guarde-o na variável `pautadf`.

**Solução:**

```
pautadf = data.frame(pauta)
```

2. Repita o exercício 2.3, agora aplicado à `pautadf`.

**Solução:**

A solução desta alínea é idêntica à do exercício 2.3.

3. Determine o número de alunos (linhas) e o número de componentes de avaliação (colunas). No exercício 2.1 referiu-se a função `dim()`. Procure por uma forma alternativa de encontrar o número de linhas e de colunas no ecrã de ajuda da função `dim()`.

**Solução:**

```
# número de linhas
> nrow(pautadf)
[1] 5
# número de colunas
> ncol(pautadf)
[1] 6
```

4. Os alunos foram aleatoriamente distribuídos em dois grupos, conforme mostra o bloco de código em baixo. Explique-o da melhor maneira que conseguir.

```
Grupos = sample(paste("Grupo",1:2), nrow(pautadf), replace=T)
pautadf = data.frame(pautadf, Grupos)
```

**Solução:**

A solução deste exercício fica para discussão entre os alunos.

5. Depois da execução das instruções dadas na alínea anterior, uma nova coluna surge no data frame `pautadf`. Verifique o tipo de objecto com que os elementos dessa coluna ficaram. O que aconteceu?

**Solução:**

```
> typeof(pautadf$Grupos)
[1] "integer"
> class(pautadf$Grupos)
[1] "factor"
```

A criação de um `data.frame` com uma coluna cujos elementos sejam do tipo `character`, converte essa coluna automaticamente para `factor`. Pode usar o argumento `stringsAsFactors`, da função `data.frame()`, para controlar este comportamento.

6. Utilize o editor interativo, disponibilizado pela função `edit()`, para alterar alguns valores (à sua escolha) do data frame.

**Solução:**

```
pautadf = edit(pautadf)
```

Com o editor interativo, é possível editar o conteúdo das células de uma tabela, como se fosse uma folha de cálculo.