Estruturas de Dados em R

Luís Torgo
FEP, Universidade do Porto
ltorgo@liacc.up.pt

13 de Dezembro de 2006

O objectivo principal deste material é introduzir os alunos às principais estruturas de dados disponíveis na linguagem R.

1. Objectos do R

- O R é uma linguagem baseada em objectos.
  Tudo o que nós vamos usar no R está guardado na memória do computador sob a forma de um objecto.
- Todos os objectos em R têm um nome associado e podem armazenar diferentes tipos de coisas (números, texto, vectores, matrizes, expressões, funções, etc.).
- Para armazenar algo num objecto usamos o operador de **atribuição**,

```
> taxa.de.juro <- 4.5
```

- Para ver o conteúdo de um objecto (o que está guardado na memória sob um determinado nome) basta digitar o nome do objecto no *prompt* do R, carregando em seguida em Enter,

```
> taxa.de.juro

[1] 4.5
```

- A operação de atribuição é *destrutiva* no sentido que ao atribuir um novo valor a um objecto já existente, vamos perder o conteúdo que ele estava a armazenar anteriormente.
- Também podemos atribuir expressões numéricas a objectos. O resultado de tal operação é o de que o resultado do cálculo da expressão é colocado no objecto,

```
> z <- 5
> w <- z^2
> w

[1] 25


> i <- (z * 2 + 45)/2
> i

[1] 27.5
```

- Sempre que pretendemos atribuir o resultado de uma expressão a um objecto e em seguida ver o seu conteúdo (como nos exemplos acima), podemos em alternativa usar a seguinte sintaxe que nos poupa algum tempo,

```
> (w <- z^2)

[1] 25
```

- Não é necessário atribuir o resultado das expressões a objectos. Só faz sentido se pretendermos usar o resultado do cálculo mais tarde. Se pretendermos saber simplesmente o resultado do cálculo, podemos usar o R como uma espécie de calculadora,

```
> (34 + 90)/12.5

[1] 9.92
```

- Como a memória do computador é limitada podemos apagar os objectos sempre que não precisarmos mais deles.
- Podemos ver quais os objectos actualmente na memória do computador usando as funções `ls()` ou `objects()`.
- Se já não necessitamos de algum dos objectos podemos então apagá-lo com a função `rm()` como nos exemplos apresentados em seguida,

```
> ls()
[1] "i" "w" "y" "z"
> rm(y)
> rm(z,w)
> objects()
[1] "i"
```

- O nome dos objectos pode conter qualquer letra maiúscula ou minúscula, os dígitos 0 a 9 (excepto no início do nome), e também o ponto final ".".
- **Os nomes dos objectos em R são sensíveis às letras maiúsculas / minúsculas**.
  Cor e **cor** são dois objectos diferentes para o R.
- **Não pode usar espaços nos nomes dos objectos**

```
> taxa de câmbio <- 0.05
Error: syntax error
```

2. Vectores

- O objecto mais básico do R para guardar dados é o vector.
- Um vector é uma estrutura de dados que permite armazenar um conjunto de *valores do mesmo tipo* (por exemplo números) sob um mesmo nome.
- Os elementos podem depois ser acedidos individualmente usando um esquema de indexação.
- Estrutura de dados bastante útil para armazenar várias coisas relacionadas.
  Por exemplo guardar os lucros obtidos pela nossa empresa ao longo dos 12 meses do ano anterior.

- Todos os vectores em R têm um *modo* e um *tamanho*.
- O modo determina o tipo de valores guardado no vector.
  Podemos ter vectores com modo *character*, *logical*, *numeric* e *complex*.
  Ou seja, podemos ter vectores para armazenar os seguintes tipos de dados atómicos: conjuntos de caracteres, valores lógicos (**F** ou **T** ou **FALSE** ou **TRUE**), números inteiros ou reais, e números complexos.
- O tamanho de um vector é o número de elementos que ele contém, e pode ser obtido com a função `length()`.

- Na maioria das situações vamos criar vectores com mais do que um elemento. Para isso precisamos de usar a função `c()` para indicar ao R os elementos que formam o vector separando-os por vírgulas,

```
> v <- c(4, 7, 23.5, 76.2, 80)
> v

[1]  4.0  7.0 23.5 76.2 80.0


> length(v)

[1] 5


> mode(v)

[1] "numeric"
```

- Todos os elementos de um vector têm que ser do mesmo tipo (*modo*).
- Caso tentemos criar um vector com elementos de tipo diferente o R vai forçá-los a ser do mesmo tipo, alterando-os,

```
> v <- c(4, 7, 23.5, 76.2, 80, "rrt")
> v

[1] "4"    "7"    "23.5" "76.2" "80"   "rrt"
```

- As *strings* em R são conjuntos de caracteres englobados por aspas ou plicas,

```
> w <- c("rrt", "ola", "isto e uma string")
> w

[1] "rrt"              "ola"              "isto e uma string"
```

- Todos os vectores podem ter um elemento especial que é o **NA** que representa um valor desconhecido. Por exemplo, se temos os lucros trimestrais de uma empresa guardados num vector, mas por alguma razão desconhecemos o seu valor no terceiro trimestre, poderíamos usar a seguinte instrução para criar esse vector,

```
> lucros <- c(234000, 245000, NA, 124500)
> lucros

[1] 234000 245000     NA 124500
```

- Os elementos de um vector podem ser acedidos através de um índice.
- Na sua forma mais simples este índice é um número indicando o elemento que pretendemos aceder. Esse número é colocado entre parêntesis rectos a seguir ao nome do vector,

```
> lucros[2]

[1] 245000
```

- Usando esta forma de aceder aos elementos individuais de um vector podemos alterar o conteúdo de um elemento particular de um vector,

```
> lucros[3] <- 45000
> lucros

[1] 234000 245000  45000 124500
```

- O tamanho de um vector já existente pode ser alterado atribuindo mais elementos a índices até agora inexistentes,

```
> w[5] <- "nova"
> w

[1] "rrt"                "ola"                "isto e uma string"
[4] NA                   "nova"
```

- Para diminuirmos o tamanho de um vector podemos usar a instrução de atribuição

```
> v <- c(45, 243, 78, 343, 445, 645, 2, 44, 56, 77)
> v

 [1]  45 243  78 343 445 645   2  44  56  77


> v <- c(v[5], v[7])
> v

[1] 445   2
```

- As posições de um vector podem ter nomes,

```
> v1 <- c(porto = 345, lisboa = 562, faro = 78)
> v1

 porto lisboa   faro
   345    562     78
```

- Também podemos dar os nomes à posteriori,

```
> v2 <- c(2.3, 4.5, 1.2)
> names(v2) <- c("p1", "p2", "p3")
> v2

 p1  p2  p3
2.3 4.5 1.2
```

2.1. Operações com Vectores

- Um dos aspectos mais poderosos da linguagem R é a possibilidade de "vectorizar" a maioria das suas funções.

- Isto quer dizer que a maioria das funções, ao serem aplicadas a um vector, produzem como resultado um vector de resultados.
  Vejamos um exemplo com a função `sqrt()` que serve para calcular raízes quadradas,

```
> v <- c(4, 7, 23.5, 76.2, 80)
> x <- sqrt(v)
> x

[1] 2.000000 2.645751 4.847680 8.729261 8.944272
```

- Esta característica do R pode ser usada para levar a cabo operações aritméticas envolvendo vectores,

```
> v1 <- c(4, 6, 87)
> v2 <- c(34, 32.4, 12)
> v1 + v2

[1] 38.0 38.4 99.0
```

- O que acontece se tentamos realizar operações envolvendo vectores de tamanho diferente? O R vai usar um regra de *reciclagem* dos valores do vector mais curto até este atingir o tamanho do maior,

```
> v1 <- c(4, 6, 8, 24)
> v2 <- c(10, 2)
> v1 + v2

[1] 14  8 18 26
```

- É como se o vector `c(10,2)` fosse de facto `c(10,2,10,2)`!
- Se os tamanhos não são múltiplos um do outro, o R imprime um aviso no écran,

```
> v1 <- c(4, 6, 8, 24)
> v2 <- c(10, 2, 4)
> v1 + v2

[1] 14  8 12 34


Warning message:
longer object length
        is not a multiple of shorter object length in: v1 + v2
```

- Repare-se que um aviso não é um erro, o que quer dizer que a operação foi levada a cabo!
- Um número é em R um vector de tamanho 1!

```
> v1 <- c(4, 6, 8, 24)
> 2 * v1

[1]  8 12 16 48
```

2.2. Indexação de Vectores

- Vimos já como "extrair" um elemento de um vector indicando a sua posição entre parêntesis rectos.
- O R também nos permite usar vectores dentro desses parêntesis rectos. Estes vectores chamam-se vectores de índices.
- Existem vários tipos de vectores de índices.
- Os vectores de índices booleanos extraem de um vector os elementos correspondentes a posições verdadeiras.
  Vejamos um exemplo,

```
> x <- c(0, -3, 4, -1, 45, 90, -5)
> x
[1]  0 -3  4 -1 45 90 -5

> x > 0
[1] FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE

> x[x > 0]
[1]  4 45 90
```

- Tirando partido da gama de operadores lógicos disponíveis no R, podemos construir vectores de indexação lógicos mais complexos,

```
> x[x <= -2 | x > 5]
[1] -3 45 90 -5

> x[x > 40 & x < 100]
[1] 45 90
```

- O R também nos permite usar um vector de números inteiros,

```
> (v <- c("a", "b", "c", "d", "e", "f", "g", "h", "i", "j"))

 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"


> v[c(4, 6)]

[1] "d" "f"
```

- Podemos ainda usar um vector com números negativos, o que nos permite indicar quais os elementos a não obter como resultado da indexação,

```
> v[-1]

[1] "b" "c" "d" "e" "f" "g" "h" "i" "j"


> v[-c(4, 6)]

[1] "a" "b" "c" "e" "g" "h" "i" "j"
```

- Um vector também pode ser indexado por um vector de *strings* tirando partido do facto de o R permitir dar nomes aos elementos de um vector.
  Suponhamos que tínhamos um vector com as taxas de inflação de 5 países europeus,

```
> tx.infl <- c(Portugal = 2.5, França = 2, Espanha = 2.1, Itália = 2.3,
+     Alemanha = 1.9)
> tx.infl

Portugal   França  Espanha   Itália Alemanha
     2.5      2.0      2.1      2.3      1.9


> tx.infl["Portugal"]

Portugal
     2.5


> tx.infl[c("Espanha", "Alemanha")]

 Espanha Alemanha
     2.1      1.9
```

- Os índices podem ser vazios, o que tem o significado que todos os elementos são seleccionados. Por exemplo, se pretendemos preencher todas as posições de um vector com zeros podemos fazer **x[] <- 0**

2.3. Exemplo Ilustrativo do Uso de Vectores

Foi feito um estudo de mercado sobre o preço médio de um produto em diversas cidades do país, tendo esse estudo levado à obtenção dos seguintes valores:

Aveiro	Braga	Bragança	Porto	Coimbra	Covilhã	Leiria	Lisboa	Setúbal	Faro
10.3	10.6	11.5	12.3	9.9	9.3	11.4	10.9	12.1	9.1

Começemos por guardar esta informação num objecto do R que seja adequado,

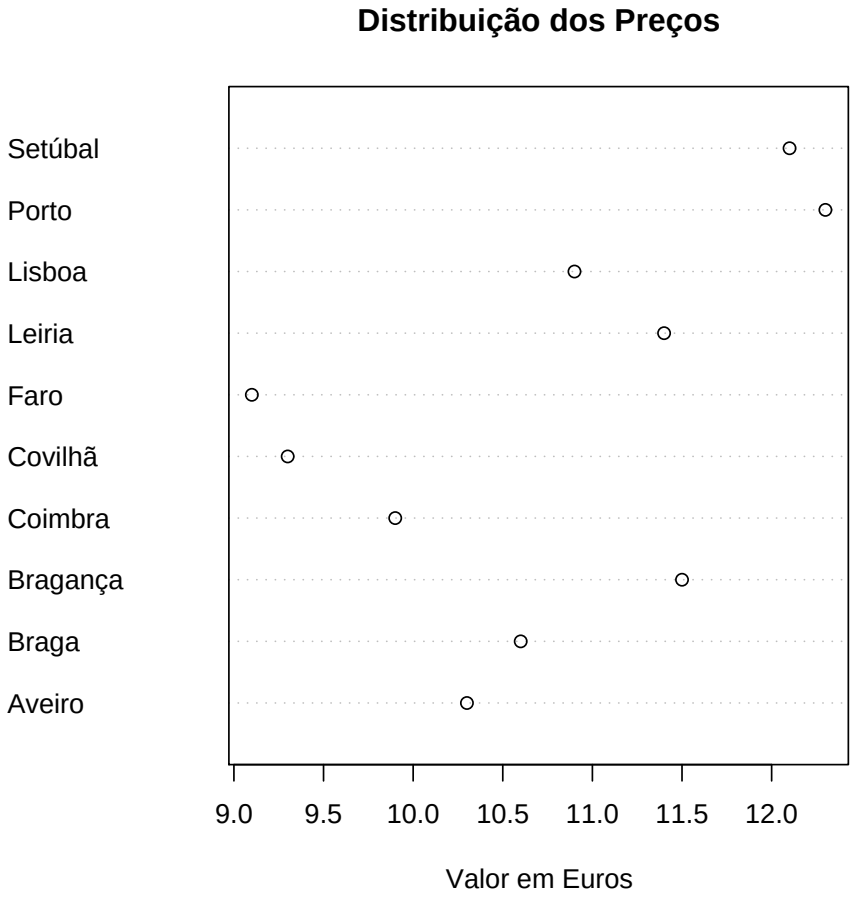
```
> (preços <- c(Aveiro = 10.3, Braga = 10.6, Bragança = 11.5,
+     Porto = 12.3, Coimbra = 9.9, Covilhã = 9.3, Leiria = 11.4,
+     Lisboa = 10.9, Setúbal = 12.1, Faro = 9.1))

  Aveiro    Braga Bragança    Porto  Coimbra  Covilhã   Leiria   Lisboa
    10.3    10.6     11.5     12.3      9.9      9.3     11.4     10.9
 Setúbal    Faro
    12.1     9.1
```

As estatísticas descritivas básicas destes dados,

```
> summary(preços)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   9.10   10.00   10.75   10.74   11.47   12.30
```

Uma visão gráfica global dos mesmos,

```
> dotchart(preços[sort(names(preços))], main = "Distribuição dos Preços",
+     xlab = "Valor em Euros")
```



As cidades com maior e menor preços,

```
> preços[c(which.min(preços), which.max(preços))]

 Faro Porto
  9.1  12.3
```

As cidades com preço acima da média,

```
> preços[preços > mean(preços)]

Bragança    Porto   Leiria   Lisboa  Setúbal
    11.5     12.3     11.4     10.9     12.1
```

As cidades com preço entre 10 e 11 euros,

```
> preços[preços > 10 & preços < 11]

Aveiro  Braga Lisboa
  10.3   10.6   10.9
```

As 50% de cidades que têm o preço mais "central",

```
> i <- summary(preços)[c(2, 5)]
> sort(preços[preços >= i[1] & preços <= i[2]])

Aveiro  Braga Lisboa Leiria
  10.3   10.6   10.9   11.4
```

3. Factores

- Os factores proporcionam uma forma fácil e compacta de lidar com dados categóricos (ou nominais).
- Variáveis que podem tomar como valores possíveis um conjunto finito de etiquetas (por exemplo uma variável que armazene o estado civil de 30 indivíduos).
- Em R os factores têm associados a eles um conjunto de níveis, que são os valores possíveis que podem tomar.
- É importante usar factores para guardar variáveis discretas porque no R há várias funções (por exemplo funções ligadas à visualização de dados) que tiram partido disso.
- Para criar um factor fazemos,

```
> s <- factor(c("f", "m", "m", "m", "f", "m", "f", "m", "f", "f"))
> s

 [1] f m m m f m f m f f
Levels: f m


> mode(s)

[1] "numeric"
```

- Quando a nossa amostra não contêm todos os valores possíveis...

```
> outro.s <- factor(c("m", "m", "m", "m"), levels = c("f", "m"))
> outro.s

[1] m m m m
Levels: f m
```

- Por vezes temos a informação original codificada como números embora sejam variáveis discretas,

```
> estado.civil <- factor(c(1, 0, 1, 1, 1, 0, 0, 1, 1), labels = c("solteiro",
+     "casado"))
> estado.civil

[1] casado   solteiro casado   casado   casado   solteiro solteiro casado
[9] casado
Levels: solteiro casado
```

- Contando o número de ocorrências de cada valor...

```
> table(s)

s
f m
5 5


> table(outro.s)

outro.s
f m
0 4
```

- A função `table()` também pode ser usada para fazer tabulações cruzadas de dois factores, desde que estes tenham o mesmo tamanho.
  Imaginemos que temos um outro vector com a gama de idades dos indivíduos cujo sexo está armazenado em **s**. Podemos fazer uma tabulação cruzada da idade e do sexo dos 10 indivíduos, da seguinte forma,

```
> idade <- factor(c("adulto", "adulto", "jovem", "jovem", "adulto",
+     "adulto", "adulto", "jovem", "adulto", "jovem"))
> s

 [1] f m m m f m f m f f
Levels: f m


> idade

 [1] adulto adulto jovem  jovem  adulto adulto adulto jovem  adulto jovem
Levels: adulto jovem


> table(idade, s)

        s
idade    f m
  adulto 4 2
  jovem  1 3
```

- Também é possível obter facilmente frequências marginais e relativas
- Por exemplo, para saber o número total de adultos e jovens faríamos,

```
> tabela.cruzada <- table(idade, s)
> margin.table(tabela.cruzada, 1)

idade
adulto  jovem
     6      4
```

- Para obter a mesma informação relativamente ao sexo dos indivíduos, bastaria mudar o número 1, que indica que pretendemos os totais por linha (a primeira dimensão do objecto) da tabela cruzada, para um 2, para obtermos os totais por coluna da tabela cruzada,

```
> margin.table(tabela.cruzada, 2)

s
f m
5 5
```

- Relativamente a frequências relativas, podemos usar a função `prop.table`,

```
> prop.table(tabela.cruzada, 1)

        s
idade            f         m
  adulto 0.6666667 0.3333333
  jovem  0.2500000 0.7500000


> prop.table(tabela.cruzada, 2)

        s
idade      f   m
  adulto 0.8 0.4
  jovem  0.2 0.6


> prop.table(tabela.cruzada)

        s
idade      f   m
  adulto 0.4 0.2
  jovem  0.1 0.3
```

- Se preferirmos em percentagem...

```
> 100 * prop.table(tabela.cruzada)

        s
idade     f  m
  adulto 40 20
  jovem  10 30
```

4. Sequências

- O R tem vários mecanismos para gerar vários tipos de sequências, evitando a introdução de todos os elementos das mesmas.
- Por exemplo, imaginemos que pretendemos criar um vector com os número de 1 a 1000,

```
> x <- 1:1000
```

- Devemos ter algum cuidado com a precedência do operador ":" em relação aos operadores aritméticos. O exemplo seguinte ilustra os "perigos" que corremos,

```
> 10:15 - 1

[1]  9 10 11 12 13 14


> 10:(15 - 1)

[1] 10 11 12 13 14
```

- O operador ":" também pode ser usado para gerar sequências descendentes,

```
> 5:0

[1] 5 4 3 2 1 0
```

- Para gerar sequências com números reais podemos usar a função `seq()`,

```
> seq(-4, 1, 0.5)

 [1] -4.0 -3.5 -3.0 -2.5 -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0
```

- Esta função tem muitas outras possibilidades para explicitar a sequência que pretendemos. Seguem-se alguns exemplos (poderá também explorar as potencialidades de ajuda do R, digitando o comando `? seq` que mostra o *help* relativamente à função seq),

```
> seq(from = 1, to = 5, length = 4)

[1] 1.000000 2.333333 3.666667 5.000000


> seq(from = 1, to = 5, length = 2)

[1] 1 5


> seq(length = 10, from = -2, by = 0.2)

 [1] -2.0 -1.8 -1.6 -1.4 -1.2 -1.0 -0.8 -0.6 -0.4 -0.2
```

- Uma outra função bastante útil para gerar sequências é a função `rep()`,

```
> rep(5, 10)

 [1] 5 5 5 5 5 5 5 5 5 5


> rep("sim", 3)

[1] "sim" "sim" "sim"


> rep(1:3, 2)

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

- A função `gl()` pode ser usada para gerar sequências envolvendo factores. A sintaxe desta função é `gl(k,n)`, em que `k` é o número de níveis do factor e `n` o número de repetições de cada nível. Vejamos dois exemplos,

```
> gl(3, 5)

 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels: 1 2 3


> gl(2, 5, labels = c("nao", "sim"))

 [1] nao nao nao nao nao sim sim sim sim sim
Levels: nao sim
```

- O R tem uma série de funções para gerar sequências aleatórias de acordo com uma série de funções de distribuição de probabilidade.
- Essas funções têm a forma genérica `rfunc(n, par1, par2, ...)`, em que `n` é o número de dados a gerar, e `par1, par2, ...` são valores de alguns parâmetros que a função específica a ser usada possa precisar.
- Por exemplo, se pretendemos 10 números gerados aleatoriamente de acordo com uma distribuição normal de média 0 e desvio padrão unitário, podemos fazer,

```
> rnorm(10)

 [1]  1.08464333 -2.40738914  0.29537221 -0.09906322  0.15261368  1.29400349
 [7] -0.22959089  0.76073892  1.64477460 -0.62999262
```

- Se preferirmos 10 números provenientes de uma distribuição normal com média 10 e desvio padrão 3, faríamos

```
> rnorm(10, mean = 10, sd = 3)

 [1]  9.710987  4.511571  8.479486 11.754482  4.292229 12.100119  8.109798
 [8]  9.728588  9.006856  9.093789
```

- De igual modo para obter 5 números obtidos de forma aleatória de uma distribuição t de Student com 10 graus de liberdade, fazemos

```
> rt(5, df = 10)

[1] -0.65607431 -2.92087322  0.13349023  1.73650499 -0.03083171
```

- O R tem muitas mais funções para outras distribuições de probabilidade, bem como funções semelhantes para obter a densidade de probabilidade, as densidades acumuladas e os quartis das distribuições.

5. Matrizes

- Por vezes os nossos dados ficam melhor representados em estruturas com mais do que uma dimensão.
- As matrizes arranjam a informação em duas dimensões.
- Em R as matrizes não são mais do que vectores com uma propriedade especial que é a dimensão.
- Vejamos um exemplo: suponhamos que temos doze números correspondentes às vendas trimestrais durante o último ano, em três lojas.

```
> vendas <- c(45, 23, 66, 77, 33, 44, 56, 12, 78, 23, 78, 90)
> vendas
```

```
 [1] 45 23 66 77 33 44 56 12 78 23 78 90
```

```
> dim(vendas) <- c(3, 4)
> vendas
```

```
     [,1] [,2] [,3] [,4]
[1,]   45   77   56   23
[2,]   23   33   12   78
[3,]   66   44   78   90
```

- Uma maneira mais simples de criar a matriz seria usando uma função específica para isso,

```
> (vendas <- matrix(c(45, 23, 66, 77, 33, 44, 56, 12, 78, 23, 78,
+     90), 3, 4))

     [,1] [,2] [,3] [,4]
[1,]   45   77   56   23
[2,]   23   33   12   78
[3,]   66   44   78   90
```

- Os números foram "espalhados" pela matriz por coluna. Caso não seja isto o que pretendemos, poderemos preencher a matriz por linhas da seguinte forma,

```
> (vendas <- matrix(c(45, 23, 66, 77, 33, 44, 56, 12, 78, 23, 78,
+     90), 3, 4, byrow = T))

     [,1] [,2] [,3] [,4]
[1,]   45   23   66   77
[2,]   33   44   56   12
[3,]   78   23   78   90
```

- Nas matrizes também é possível dar nomes aos elementos para tornar a leitura da informação mais legível.

```
> rownames(vendas) <- c("loja1", "loja2", "loja3")
> colnames(vendas) <- c("1.trim", "2.trim", "3.trim", "4.trim")
> vendas

      1.trim 2.trim 3.trim 4.trim
loja1     45     23     66     77
loja2     33     44     56     12
loja3     78     23     78     90
```

- Podemos aceder aos elementos individuais das matrizes usando um esquema de indexação semelhante ao dos vectores, mas desta vez com dois índices: um para a linha e outro para a coluna.

```
> vendas[2, 2]

[1] 44
```

- Ou então, tirando partido dos nomes,

```
> vendas["loja2", "2.trim"]

[1] 44
```

- Podemos tirar partido dos esquemas de indexação discutidos anteriormente para os vectores em matrizes,

```
> vendas[1, -c(2, 4)]

1.trim 3.trim
    45     66
```

- Podemos mesmo omitir uma das dimensões das matrizes para deste modo obter todos os elementos da mesma (um índice vazio),

```
> vendas[1, ]

1.trim 2.trim 3.trim 4.trim
    45     23     66     77


> vendas[, "4.trim"]

loja1 loja2 loja3
   77    12    90
```

- As funções `cbind()` e `rbind()` podem ser usadas para juntar dois ou mais vectores ou matrizes, por colunas ou por linhas, respectivamente. Os seguintes exemplos ilustram o seu uso,

```
> (m1 <- matrix(c(45, 23, 66, 77, 33, 44, 56, 12, 78, 23), 2, 5))
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]   45   66   33   56   78
[2,]   23   77   44   12   23
```

```
> cbind(c(4, 76), m1[, 4])
```

```
     [,1] [,2]
[1,]    4   56
[2,]   76   12
```

```
> (m2 <- matrix(rep(10, 50), 10, 5))
```

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

```
> (m3 <- rbind(m1[1, ], m2[5, ]))
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]   45   66   33   56   78
[2,]   10   10   10   10   10
```

- As regras aritméticas e de reciclagem que estudamos anteriormente, também se aplicam às matrizes,

```
> (m <- matrix(c(45, 23, 66, 77, 33, 44, 56, 12, 78, 23), 2, 5))

     [,1] [,2] [,3] [,4] [,5]
[1,]   45   66   33   56   78
[2,]   23   77   44   12   23


> m * 3

     [,1] [,2] [,3] [,4] [,5]
[1,]  135  198   99  168  234
[2,]   69  231  132   36   69


> (m1 <- matrix(c(45, 23, 66, 77, 33, 44), 2, 3))

     [,1] [,2] [,3]
[1,]   45   66   33
[2,]   23   77   44


> (m2 <- matrix(c(12, 65, 32, 7, 4, 78), 2, 3))

     [,1] [,2] [,3]
[1,]   12   32    4
[2,]   65    7   78


> m1 + m2

     [,1] [,2] [,3]
[1,]   57   98   37
[2,]   88   84  122
```

- O R também tem disponíveis operadores da álgebra matricial

```
> (m1 <- matrix(c(45, 23, 66, 77, 33, 44), 2, 3))

     [,1] [,2] [,3]
[1,]   45   66   33
[2,]   23   77   44


> (m2 <- matrix(c(5, 3, 466, 54.5, 3.2, -34), 3, 2))

     [,1]  [,2]
[1,]    5  54.5
[2,]    3   3.2
[3,]  466 -34.0


> m1 %*% m2

      [,1]   [,2]
[1,] 15801 1541.7
[2,] 20850    3.9
```

- O R tem muitas outras funções da álgebra matricial,

```
> t(m1)

     [,1] [,2]
[1,]   45   23
[2,]   66   77
[3,]   33   44


> (m <- matrix(c(34, -23, 43, 5), 2, 2))

     [,1] [,2]
[1,]   34   43
[2,]  -23    5


> det(m)

[1] 1159


> solve(m)

            [,1]        [,2]
[1,] 0.004314064 -0.03710095
[2,] 0.019844694  0.02933563
```

- A função `solve()` também pode ser usada para resolver sistemas de equações lineares. Imaginemos o seguinte sistema de equações,

$$\begin{cases} -4x + 0.3y &= 12.3 \\ 54.3x - 4y &= 45 \end{cases}$$

Podemos resolvê-lo em R da seguinte forma,

```
> coefs <- matrix(c(-4, 0.3, 54.3, -4), 2, 2, byrow = T)
> ys <- c(12.3, 45)
> solve(coefs, ys)

[1]  216.2069 2923.7586
```

5.1. Exemplos de Utilização de Matrizes

O Ministério da Saúde fez um estudo sobre a percentagem média de fumadores do sexo femenino e masculino em escolas de diversas cidades do país.

Os resultados obtidos são indicados na seguinte tabela:

	Aveiro	Braga	Bragança	Porto	Coimbra	Covilhã	Leiria	Lisboa	Setúbal	Faro
mulheres	50.3	60.6	71.5	82.3	59.9	79.3	41.4	80.9	72.1	59.1
homens	54.4	52.4	67.1	78.3	59.2	65.1	86.3	81.3	57.3	61.3

Para armazenar estes dados num objecto do R o mais adequado seria uma matriz,

```
> percFum <- matrix(c(50.3, 60.6, 71.5, 82.3, 59.9, 79.3, 41.4,
+     80.9, 72.1, 59.1, 54.4, 52.4, 67.1, 78.3, 59.2, 65.1, 86.3,
+     81.3, 57.3, 61.3), 2, 10, byrow = T)
> colnames(percFum) <- c("Aveiro", "Braga", "Bragança", "Porto",
+     "Coimbra", "Covilhã", "Leiria", "Lisboa", "Setúbal", "Faro")
> rownames(percFum) <- c("mulheres", "homens")
> percFum

         Aveiro Braga Bragança Porto Coimbra Covilhã Leiria Lisboa Setúbal Faro
mulheres   50.3  60.6     71.5  82.3    59.9    79.3   41.4   80.9    72.1 59.1
homens     54.4  52.4     67.1  78.3    59.2    65.1   86.3   81.3    57.3 61.3
```

Algumas questões sobre estes dados:

1. Quais as percentagens médias de fumadores por cidade e por sexo?

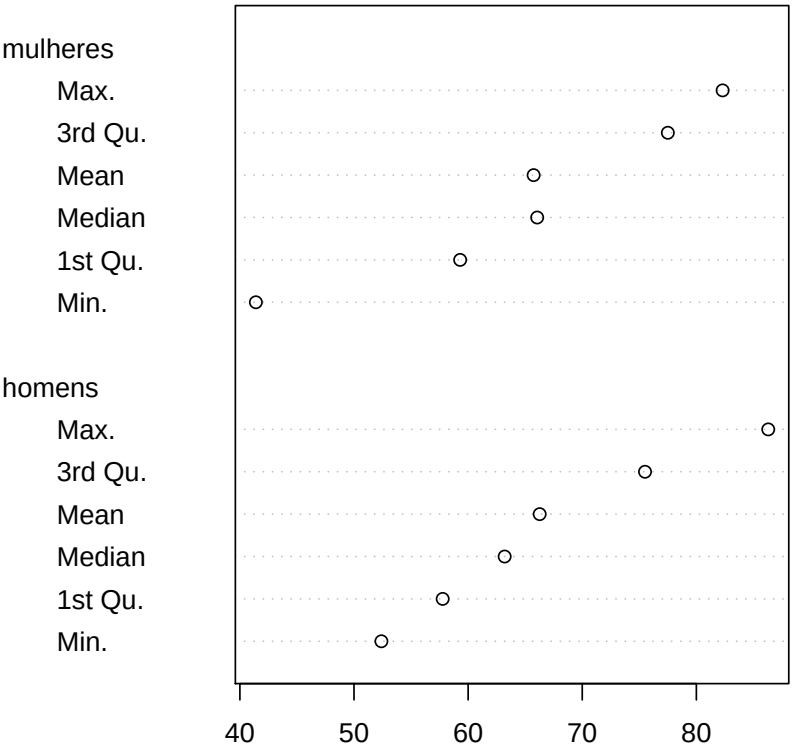
```
> apply(percFum, 2, mean)

  Aveiro    Braga Bragança    Porto  Coimbra  Covilhã   Leiria   Lisboa
   52.35    56.50    69.30    80.30    59.55    72.20    63.85    81.10
 Setúbal     Faro
   64.70    60.20


> apply(percFum, 1, mean)

mulheres   homens
   65.74    66.27
```

2. Obtenha as estatísticas descritivas básicas da percentagem de fumadores, por sexo.

```
> apply(percFum, 1, summary)

        mulheres homens
Min.       41.40  52.40
1st Qu.    59.30  57.77
Median     66.05  63.20
Mean       65.74  66.27
3rd Qu.    77.50  75.50
Max.       82.30  86.30
```

3. Apresente a informação obtida anteriormente de forma gráfica

```
> dotchart(apply(percFum, 1, summary))
```



4. Qual a cidade em que a diferença entre a percentagem de homens e mulheres fumadoras é maior?

```
> colnames(percFum)[which.max(abs(percFum["homens", ] - percFum["mulheres",
+     ]))]

[1] "Leiria"
```

5. Para cada cidade indique qual o sexo que mais fuma.

```
> r <- rownames(percFum)[ifelse(percFum[1, ] - percFum[2, ] > 0,
+     1, 2)]
> names(r) <- colnames(percFum)
> r

    Aveiro      Braga   Bragança      Porto    Coimbra    Covilhã     Leiria
  "homens" "mulheres" "mulheres" "mulheres" "mulheres" "mulheres"   "homens"
    Lisboa    Setúbal       Faro
  "homens" "mulheres"   "homens"
```

6. Arrays

- Os *arrays* são extensões das matrizes para mais do que duas dimensões.
- Isto significa que podem ter vários índices. À parte esta pequena diferença, eles podem ser usados da mesma forma do que as matrizes.
- De modo semelhante à função `matrix`, existe uma função `array()` para facilitar a criação de *arrays*,

```
> (a <- array(1:50, dim = c(2, 5, 5)))

, , 1

     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10

, , 2

     [,1] [,2] [,3] [,4] [,5]
[1,]   11   13   15   17   19
[2,]   12   14   16   18   20

, , 3

     [,1] [,2] [,3] [,4] [,5]
[1,]   21   23   25   27   29
[2,]   22   24   26   28   30

, , 4

     [,1] [,2] [,3] [,4] [,5]
[1,]   31   33   35   37   39
[2,]   32   34   36   38   40

, , 5

     [,1] [,2] [,3] [,4] [,5]
[1,]   41   43   45   47   49
[2,]   42   44   46   48   50
```

- Os esquemas usuais de indexação que vimos para vectores e matrizes, podem também ser aplicados aos *arrays*.

6.1. Exemplos de Utilização de Arrays

Foi feito um estudo comparativo destinado a comparar a percentagem de fumadores em Espanha e Portugal, para cada sexo e dentro das faixas etárias < 18 anos, 18 a 40 anos, e mais de 40 anos.

Uma possível forma de guardar estas percentagens em R, seria através de um *array* de 3 dimensões: faixa etária, sexo e país. Vejamos uma hipótese usando percentagens "inventadas":

```
> percFum <- array(sample(0:100, 12), dim = c(3, 2, 2), dimnames = list(c("< 18",
+     "18-40", "> 40"), c("Mulheres", "Homens"), c("Portugal",
+     "Espanha")))
> percFum

, , Portugal

      Mulheres Homens
< 18        19     96
18-40       69     49
> 40        63      1

, , Espanha

      Mulheres Homens
< 18        77     28
18-40       74     27
> 40         6     88
```

Algumas questões sobre os dados:

1. Qual a percentagem média de fumadores em cada país?

```
> apply(percFum, 3, mean)

Portugal  Espanha
    49.5     50.0
```

2. Mostre as percentagens obtidas no estudo para mulheres.

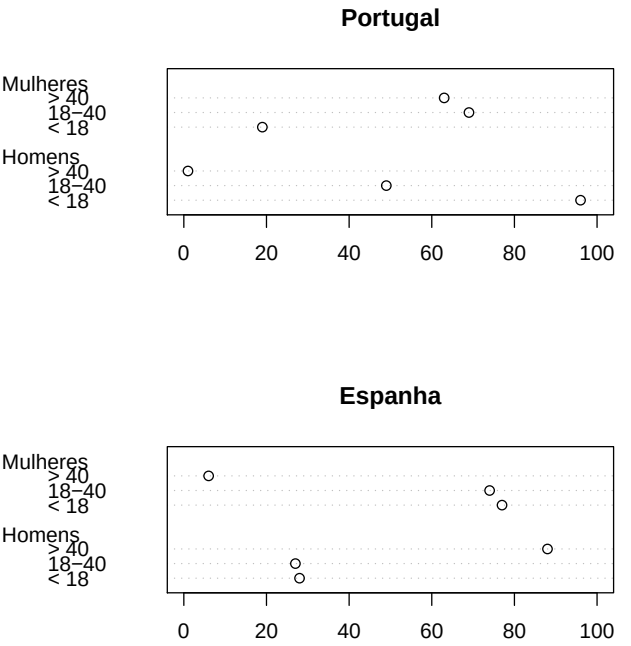
```
> percFum[, "Mulheres", ]

      Portugal Espanha
< 18        19      77
18-40       69      74
> 40        63       6
```

3. Qual a percentagem média de fumadores na faixa de menores de 18 anos, em cada país?

```
> apply(percFum[1, , ], 2, mean)

Portugal  Espanha
    57.5     52.5
```

4. Apresente um gráfico onde seja possível consultar os resultados do estudo em ambos os países.

```
> par(mfrow = c(2, 1))
> dotchart(percFum[, , "Portugal"], main = "Portugal", xlim = c(0,
+     100))
> dotchart(percFum[, , "Espanha"], main = "Espanha", xlim = c(0,
+     100))
> par(mfrow = c(1, 1))
```



X11 2

7. Listas

- Uma lista é uma colecção ordenada de objectos conhecidos como os componentes da lista.
- Esses componentes não necessitam de ser do mesmo tipo, modo ou tamanho.
- Os componentes de uma lista em R são sempre numerados e podem também ter um nome associados a eles.
- Vejamos um pequeno exemplo de criação de uma lista em R,

```
> (estudante <- list(nro = 34453, nome = "Carlos Silva", notas = c(14.3,
+     12, 15, 19)))

$nro
[1] 34453

$nome
[1] "Carlos Silva"

$notas
[1] 14.3 12.0 15.0 19.0
```

- Podemos extrair componentes específicos de uma lista através da seguinte sintaxe,

```
> estudante[[1]]

[1] 34453


> estudante[[3]]

[1] 14.3 12.0 15.0 19.0


> estudante[1]

$nro
[1] 34453
```

- No caso de listas com componentes com nomes, como o caso da lista **estudante**, podemos extrair os componentes da lista usando uma forma alternativa,

```
> estudante$nro

[1] 34453
```

- Os nomes dos componentes de uma lista podem ser manipulados de forma semelhante aos dos vectores,

```
> names(estudante) <- c("número", "nome", "notas")
> estudante

$número
[1] 34453

$nome
[1] "Carlos Silva"

$notas
[1] 14.3 12.0 15.0 19.0
```

- As listas podem ser extendidas acrescentando-lhes novos componentes,

```
> estudante$pais <- c("Ana Castro", "Miguel Silva")
> estudante

$número
[1] 34453

$nome
[1] "Carlos Silva"

$notas
[1] 14.3 12.0 15.0 19.0

$pais
[1] "Ana Castro"   "Miguel Silva"
```

- Podemos saber quantos componentes tem uma lista usando a função `length()`,

```
> length(estudante)

[1] 4
```

- Podemos juntar duas ou mais listas numa só usando a função `c()`,

```
> resto <- list(idade = 19, sexo = "masculino")
> (estudante <- c(estudante, resto))

$número
[1] 34453

$nome
[1] "Carlos Silva"

$notas
[1] 14.3 12.0 15.0 19.0

$pais
[1] "Ana Castro"   "Miguel Silva"

$idade
[1] 19

$sexo
[1] "masculino"
```

8. Data Frames

- Um *data frame* é um objecto do R que é normalmente usado para guardar tabelas de dados.
- Na sua forma, um *data frame*, é muito semelhante a uma matriz, mas as suas colunas têm nomes e podem conter dados de tipo diferente.
- Podemos criar um *data frame* da seguinte forma,

```
> notas.inform <- data.frame(nros = c(2355, 3456, 2334, 5456),
+     turma = c("tp1", "tp1", "tp2", "tp3"), notas = c(10.3, 9.3,
+         14.2, 15))
> notas.inform

  nros turma notas
1 2355   tp1  10.3
2 3456   tp1   9.3
3 2334   tp2  14.2
4 5456   tp3  15.0
```

- Os elementos de um *data frame* podem ser acedidos como uma matriz,

```
> notas.inform[2, 2]

[1] tp1
Levels: tp1 tp2 tp3
```

- Os esquemas de indexação descritos anteriormente também podem ser usados com os *data frames*.
- Adicionalmente, as colunas dos *data frames* podem ser acedidas na sua totalidade usando o seu nome,

```
> notas.inform$nros

[1] 2355 3456 2334 5456
```

- Usando os esquemas de indexação envolvendo condições lógicas, podemos fazer consultas mais complexas aos dados guardados num *data frame*.

```
> notas.inform[notas.inform$notas > 10, ]

  nros turma notas
1 2355   tp1  10.3
3 2334   tp2  14.2
4 5456   tp3  15.0


> notas.inform[notas.inform$notas > 14, "nros"]

[1] 2334 5456


> notas.inform[notas.inform$turma == "tp1", c("nros", "notas")]

  nros notas
1 2355  10.3
2 3456   9.3
```

- As consultas a *data frames* podem ser simplificadas através do uso da função `attach`,

```
> attach(notas.inform)
> notas.inform[notas > 14, ]

  nros turma notas
3 2334   tp2  14.2
4 5456   tp3  15.0


> turma

[1] tp1 tp1 tp2 tp3
Levels: tp1 tp2 tp3
```

- A função `detach()` tem o efeito inverso, e deve ser executada quando já não precisamos mais deste acesso "directo" às colunas,

```
> detach(notas.inform)
> turma
        Object "turma" not found
```

- É possível acrescentar novas colunas a um *data frame*,

```
> notas.inform$resultado <- c("aprovado", "oral", "aprovado", "aprovado")
> notas.inform

  nros turma notas resultado
1 2355   tp1  10.3  aprovado
2 3456   tp1   9.3      oral
3 2334   tp2  14.2  aprovado
4 5456   tp3  15.0  aprovado
```

- A única restrição a este tipo de acrescentos é a de que a nova coluna deverá ter tantos elementos quantas as linhas do *data frame*.
- Podemos saber o número de linhas e colunas de um *data frame* da seguinte forma,

```
> nrow(notas.inform)

[1] 4


> ncol(notas.inform)

[1] 4
```

- Para introduzir dados podemos usar um interface tipo folha de cálculo que o R possui,

```
> outras.notas <- data.frame()
> outras.notas <- edit(outras.notas)
```

- O interface da função `edit()` permite-nos acrescentar facilmente valores nas colunas do *data frame* bastando para isso *clicar* em cima de uma célula e começar a escrever o seu valor, bem como dar nomes às colunas do *data frame*, *clicando* em cima do seu nome. Experimente.

8.1. Exemplos de Utilização de Data Frames

Vamos usar na nossa ilustração os dados do Censos de 1977 nos USA, cujos dados estão guardados no dataset "state" do R (pode fazer ? state para saber mais informação sobre os dados).

Começemos por criar um *data frame* com os dados que nos interessam para a ilustração:

```
> censos77 <- cbind(as.data.frame(state.x77), state.region, state.division)
> colnames(censos77)[c(4, 6, 9, 10)] <- c("LifeExp", "HSGrad",
+     "Region", "Division")
> head(censos77)
```

```
           Population Income Illiteracy LifeExp Murder HSGrad Frost   Area
Alabama          3615   3624        2.1   69.05   15.1   41.3    20  50708
Alaska            365   6315        1.5   69.31   11.3   66.7   152 566432
Arizona          2212   4530        1.8   70.55    7.8   58.1    15 113417
Arkansas         2110   3378        1.9   70.66   10.1   39.9    65  51945
California      21198   5114        1.1   71.71   10.3   62.6    20 156361
Colorado         2541   4884        0.7   72.06    6.8   63.9   166 103766
           Region           Division
Alabama     South East South Central
Alaska       West            Pacific
Arizona      West           Mountain
Arkansas    South West South Central
California   West            Pacific
Colorado     West           Mountain
```

Para facilitar a resposta às questões que se seguem, façamos o seguinte:

```
> attach(censos77)
> estados <- rownames(censos77)
```

Vejamos agora algumas questões ilustrativas:

1. Propriedades estatísticas básicas dos dados

```
> summary(censos77)
```

```
   Population        Income       Illiteracy       LifeExp
Min.   :  365   Min.   :3098   Min.   :0.500   Min.   :67.96
1st Qu.: 1080   1st Qu.:3993   1st Qu.:0.625   1st Qu.:70.12
Median : 2838   Median :4519   Median :0.950   Median :70.67
Mean   : 4246   Mean   :4436   Mean   :1.170   Mean   :70.88
3rd Qu.: 4968   3rd Qu.:4814   3rd Qu.:1.575   3rd Qu.:71.89
Max.   :21198   Max.   :6315   Max.   :2.800   Max.   :73.60

     Murder          HSGrad          Frost             Area
Min.   : 1.400   Min.   :37.80   Min.   :  0.00   Min.   :  1049
1st Qu.: 4.350   1st Qu.:48.05   1st Qu.: 66.25   1st Qu.: 36985
Median : 6.850   Median :53.25   Median :114.50   Median : 54277
Mean   : 7.378   Mean   :53.11   Mean   :104.46   Mean   : 70736
3rd Qu.:10.675   3rd Qu.:59.15   3rd Qu.:139.75   3rd Qu.: 81162
Max.   :15.100   Max.   :67.30   Max.   :188.00   Max.   :566432

          Region                Division
Northeast     : 9   South Atlantic     : 8
South         :16   Mountain           : 8
North Central:12   West North Central: 7
West          :13   New England        : 6
                    East North Central: 5
                    Pacific            : 5
                    (Other)            :11
```

2. Maior e menor estados (em Área)

```
> estados[c(which.min(Area), which.max(Area))]

[1] "Rhode Island" "Alaska"
```

3. Os 5 maiores estados (em População)

```
> estados[order(Population, decreasing = T)[1:5]]

[1] "California"   "New York"     "Texas"        "Pennsylvania" "Illinois"
```

4. A diferença, relativamente à média de esperança de vida, para cada estado

```
> d <- LifeExp - mean(LifeExp)
> names(d) <- estados
> round(d, 2)
```

```
      Alabama        Alaska       Arizona      Arkansas    California
        -1.83         -1.57         -0.33         -0.22          0.83
     Colorado   Connecticut      Delaware       Florida       Georgia
         1.18          1.60         -0.82         -0.22         -2.34
      Hawaii         Idaho      Illinois       Indiana          Iowa
         2.72          0.99         -0.74          0.00          1.68
      Kansas      Kentucky     Louisiana         Maine      Maryland
         1.70         -0.78         -2.12         -0.49         -0.66
Massachusetts      Michigan     Minnesota   Mississippi      Missouri
         0.95         -0.25          2.08         -2.79         -0.19
     Montana      Nebraska        Nevada New Hampshire    New Jersey
        -0.32          1.72         -1.85          0.35          0.05
  New Mexico      New York North Carolina  North Dakota          Ohio
        -0.56         -0.33         -1.67          1.90         -0.06
    Oklahoma        Oregon  Pennsylvania  Rhode Island South Carolina
         0.54          1.25         -0.45          1.02         -2.92
South Dakota     Tennessee         Texas          Utah       Vermont
         1.20         -0.77          0.02          2.02          0.76
    Virginia    Washington West Virginia     Wisconsin       Wyoming
        -0.80          0.84         -1.40          1.60         -0.59
```

5. Os estados com salário per capita 20% superior à média

```
> estados[Income > 1.2 * mean(Income)]
```

```
[1] "Alaska"      "Connecticut"
```

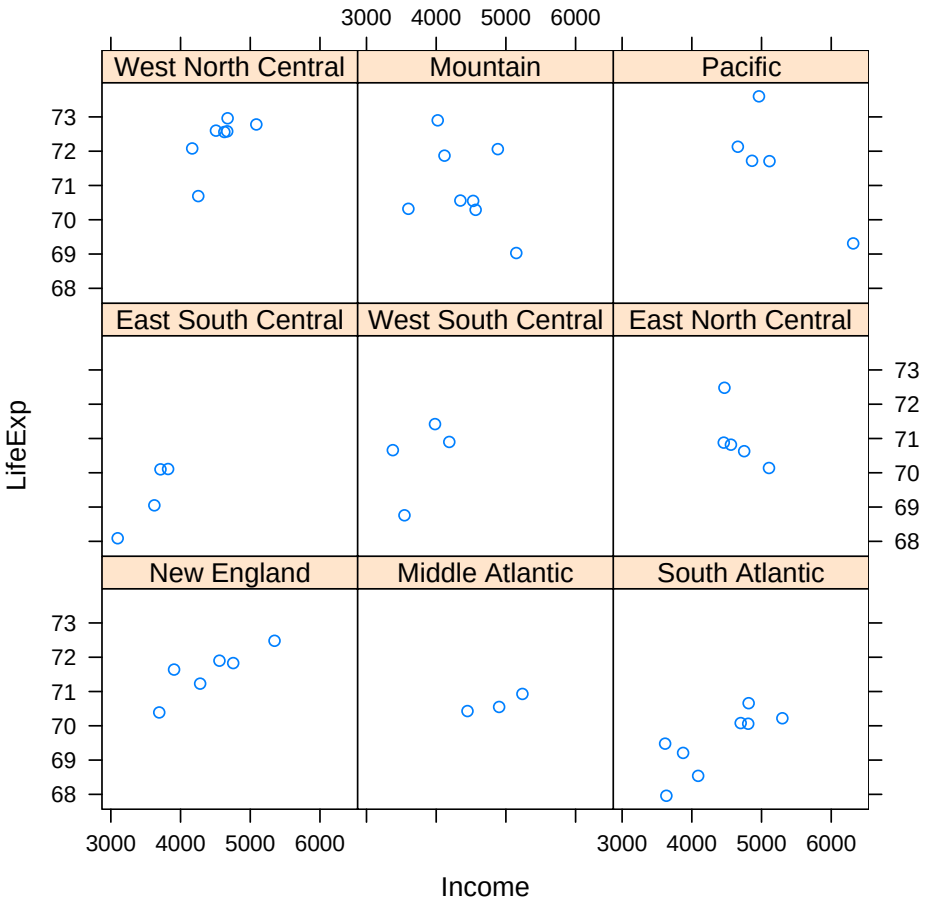
6. Quais os 5 estados com menor densidade populacional (População por unidade de Área) e qual o respectivo valor dessa densidade?

```
> dens <- 1000 * Population/Area
> names(dens) <- estados
> dens[order(dens)[1:5]]
```

```
    Alaska      Wyoming      Montana       Nevada South Dakota
 0.6443845    3.8681934    5.1240839    5.3690542    8.9658350
```

7. Um gráfico que mostre a relação entre a esperança média de vida e o salário para cada divisão dos USA

```
> library(lattice)
> xyplot(LifeExp ~ Income | Division, censos77)
```



9. Séries Temporais

Por vezes temos dados que estão "etiquetados" pelo tempo (por exemplo a evolução anual do número de turistas em Portugal). Este tipo de dados são normalmente conhecidos como séries temporais.

Uma série temporal (univariada) é pois um conjunto de observações de uma variável Y ao longo do tempo, $y_1, y_2, \ldots, y_{t-1}, y_t$.

O R tem várias packages dedicadas a este tipo de dados.

Vamos distinguir dois tipos principais de séries temporais: regulares e irregulares.

9.1. Séries Regulares

Numa série temporal regular as observações da variável são igualmente espaçadas ao long do tempo.

A função `ts()` é a principal forma de criar um objecto para armazenar este tipo de dados. Vejamos alguns exemplos da sua utilização:

```
> ts(rnorm(10), start = 1990, frequency = 1)

Time Series:
Start = 1990
End = 1999
Frequency = 1
 [1]  1.390075113  0.157727880  0.526614185  0.596235866  0.075847455
 [6]  1.763427630  0.121550527  0.006578956 -1.305921993 -1.159423937

> ts(rnorm(10), frequency = 4, start = c(1959, 2))

           Qtr1        Qtr2        Qtr3        Qtr4
1959             0.48893202 -0.47319738 -1.65099052
1960  0.09566026 -0.05086652  0.60318794  0.43908427
1961  0.27239684 -0.70511331 -0.15313474

> ts(rnorm(10), frequency = 12, start = c(1959, 2))

             Feb         Mar         Apr         May         Jun         Jul
1959  0.71590591  0.61741425 -0.77567548  0.78791410  1.27351617  0.81724662
             Aug         Sep         Oct         Nov
1959  1.68091804  0.06803156  1.18959282  1.24392528
```

Por vezes temos várias variáveis a serem "observadas" (medidas) em cada instante temporal. Neste caso temos o que se chama uma série temporal multivariada.

Vejamos como criar este tipo de séries em R:

```
> (m <- ts(matrix(rnorm(30), 10, 3), start = c(1961, 6), frequency = 12))

            Series 1   Series 2    Series 3
Jun 1961  0.73214490 -1.3039062 -0.02396328
Jul 1961 -0.06015466 -0.3002657  0.89175316
Aug 1961  0.40477147  2.4219848 -0.01558526
Sep 1961  0.72996049  0.4271499  0.70462471
Oct 1961 -0.97126039  0.7031937 -1.60086459
Nov 1961 -2.23985973  0.4120297 -1.26702486
Dec 1961 -2.03978697  0.5722928  0.76289340
Jan 1962 -0.13479678 -0.7532844 -0.35877907
Feb 1962  0.79642552  0.1881931 -0.59821972
Mar 1962  1.04784402 -0.8658050 -0.75636832
```

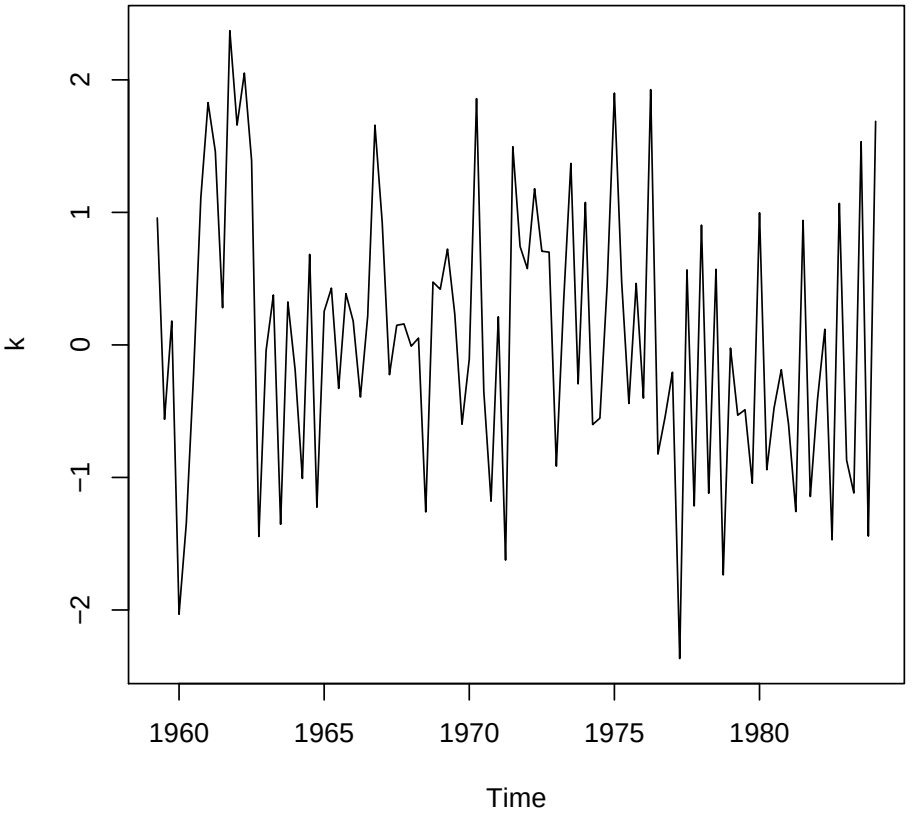
As seguintes funções são úteis para a manipulação temporal de objectos deste tipo:

```
> x <- ts(rnorm(10), frequency = 4, start = c(1959, 2))
> start(x)

[1] 1959    2

> end(x)

[1] 1961    3
```

```
> window(x, start = c(1959, 5))

           Qtr1       Qtr2       Qtr3       Qtr4
1960  0.2620206 -0.7530333 -0.6291260  0.4632187
1961 -1.4053335  0.6180812  0.3653492

> window(x, end = c(1959, 7))

           Qtr1       Qtr2       Qtr3       Qtr4
1959            -1.4667177 -0.2608822 -0.5518541
1960  0.2620206 -0.7530333 -0.6291260

> window(x, start = c(1959, 4), end = c(1959, 9))

           Qtr1       Qtr2       Qtr3       Qtr4
1959                                    -0.5518541
1960  0.2620206 -0.7530333 -0.6291260  0.4632187
1961 -1.4053335
```

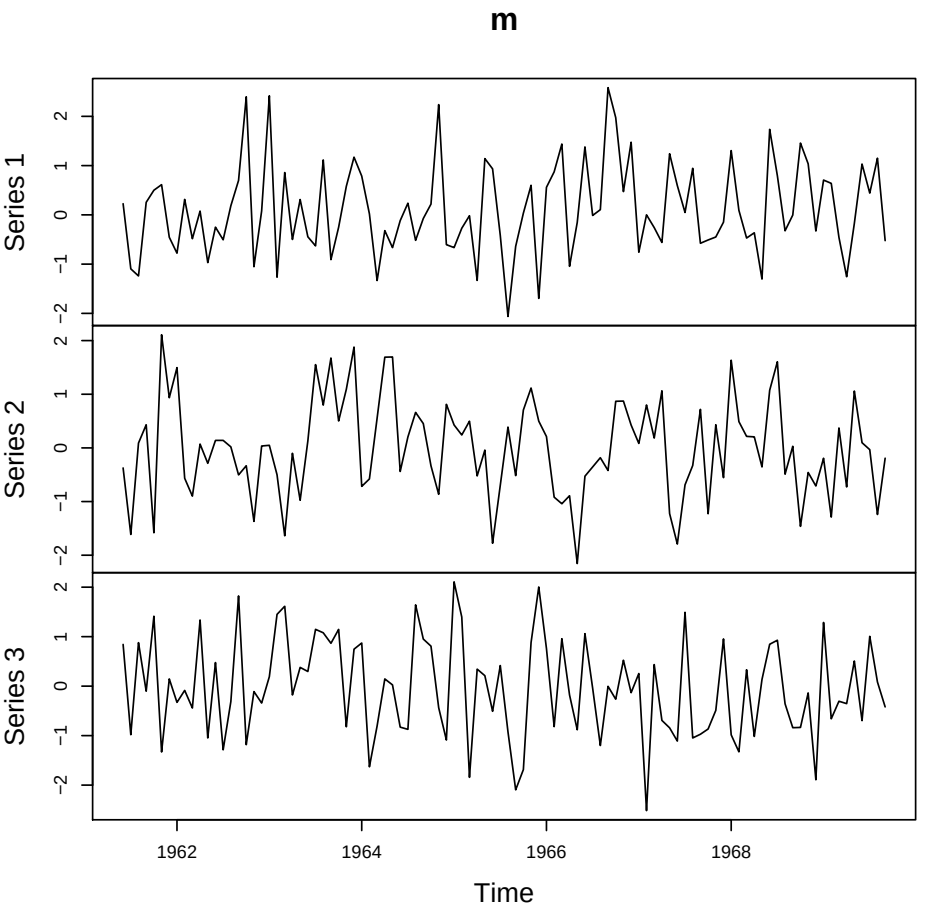
A função `plot()` também pode ser aplicada a este tipo de objectos:

```
> k <- ts(rnorm(100), frequency = 4, start = c(1959, 2))
> plot(k)
```



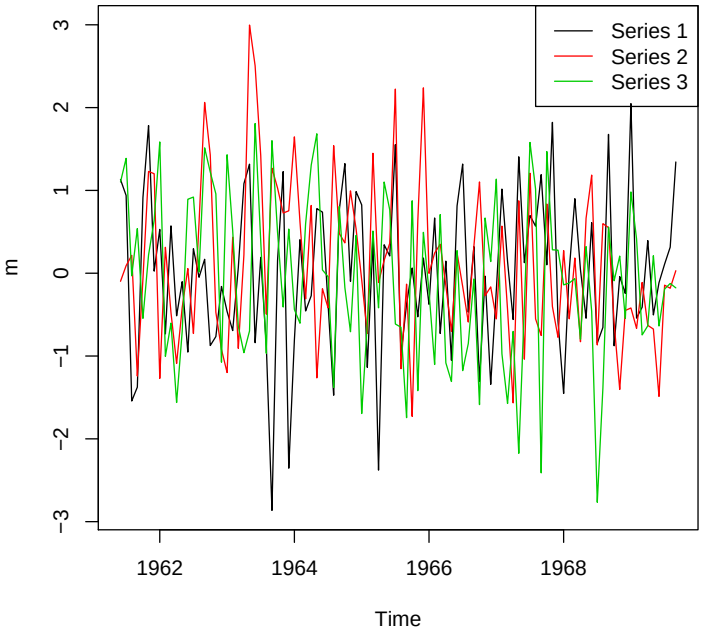
As séries multivariadas são por defeito desenhadas em gráficos separados...

```
> m <- ts(matrix(rnorm(300), 100, 3), start = c(1961, 6), frequency = 12)
> plot(m)
```



Mas também podem aparecer num só gráfico...

```
> m <- ts(matrix(rnorm(300), 100, 3), start = c(1961, 6), frequency = 12)
> plot(m, plot.type = "single", col = 1:3)
> legend("topright", legend = colnames(m), col = 1:3, lty = 1)
```



X11 2

Seguem-se exemplos de outras funções bastante úteis para lidar com séries temporais:

```
> (x <- ts(rnorm(10), frequency = 4, start = c(1959, 2)))

           Qtr1       Qtr2       Qtr3       Qtr4
1959             0.2053188  0.8096783  0.7985234
1960  0.2665182  0.6087678 -0.1042227  0.8458853
1961  0.8037273 -1.9851966  0.7499250

> lag(x)

           Qtr1       Qtr2       Qtr3       Qtr4
1959  0.2053188  0.8096783  0.7985234  0.2665182
1960  0.6087678 -0.1042227  0.8458853  0.8037273
1961 -1.9851966  0.7499250

> lag(x, 4)

           Qtr1       Qtr2       Qtr3       Qtr4
1958             0.2053188  0.8096783  0.7985234
1959  0.2665182  0.6087678 -0.1042227  0.8458853
1960  0.8037273 -1.9851966  0.7499250
```

```
> x
           Qtr1       Qtr2       Qtr3       Qtr4
1959             0.2053188  0.8096783  0.7985234
1960  0.2665182  0.6087678 -0.1042227  0.8458853
1961  0.8037273 -1.9851966  0.7499250

> diff(x)
            Qtr1        Qtr2        Qtr3        Qtr4
1959                           0.60435955 -0.01115496
1960 -0.53200520  0.34224962 -0.71299046  0.95010801
1961 -0.04215805 -2.78892387  2.73512157

> diff(x, differences = 2)
           Qtr1       Qtr2       Qtr3       Qtr4
1959                                    -0.6155145
1960 -0.5208502  0.8742548 -1.0552401  1.6630985
1961 -0.9922661 -2.7467658  5.5240454

> diff(x, lag = 2)
           Qtr1       Qtr2       Qtr3       Qtr4
1959                                     0.5932046
1960 -0.5431602 -0.1897556 -0.3707408  0.2371175
1961  0.9079500 -2.8310819 -0.0538023
```

Estas funções são também aplicáveis a séries multivariadas.

Por vezes estamos interessados em juntar várias séries num só objecto, isto é numa série multivariada. O R encarrega-se de fazer a junção certa de acordo com as etiquetas temporais de cada série!

```
> (x <- ts(rnorm(10), frequency = 4, start = c(1959, 2)))

            Qtr1        Qtr2        Qtr3        Qtr4
1959              0.32729078 -0.03745572 -1.22029997
1960 -0.37219413 -0.24057365 -0.56578588 -1.00650471
1961 -1.11443709  0.54407663 -0.86480304

> (y <- ts(rnorm(10), frequency = 4, start = c(1960, 1)))

           Qtr1       Qtr2       Qtr3       Qtr4
1960 -0.2937353 -0.5087559 -0.3518185 -1.7756490
1961  0.5808027  0.9599895  1.4002038  2.4219120
1962  1.2703331  0.5430826

> cbind(x, y)

                  x          y
1959 Q2  0.32729078         NA
1959 Q3 -0.03745572         NA
1959 Q4 -1.22029997         NA
1960 Q1 -0.37219413 -0.2937353
1960 Q2 -0.24057365 -0.5087559
1960 Q3 -0.56578588 -0.3518185
1960 Q4 -1.00650471 -1.7756490
1961 Q1 -1.11443709  0.5808027
1961 Q2  0.54407663  0.9599895
1961 Q3 -0.86480304  1.4002038
1961 Q4          NA  2.4219120
1962 Q1          NA  1.2703331
1962 Q2          NA  0.5430826
```

Também relacionada é a junção de uma série com ela própria mas "atrasada" (lagged):

```
> (x <- ts(rnorm(10), frequency = 4, start = c(1959, 2)))

           Qtr1       Qtr2       Qtr3       Qtr4
1959             0.8401383  0.3323405 -0.4444015
1960 -0.2466285 -1.2624572  0.4725261  2.6851647
1961 -0.5561170 -0.8198095 -2.2332913

> embed(x, 3)

           [,1]       [,2]       [,3]
[1,] -0.4444015  0.3323405  0.8401383
[2,] -0.2466285 -0.4444015  0.3323405
[3,] -1.2624572 -0.2466285 -0.4444015
[4,]  0.4725261 -1.2624572 -0.2466285
[5,]  2.6851647  0.4725261 -1.2624572
[6,] -0.5561170  2.6851647  0.4725261
[7,] -0.8198095 -0.5561170  2.6851647
[8,] -2.2332913 -0.8198095 -0.5561170
```

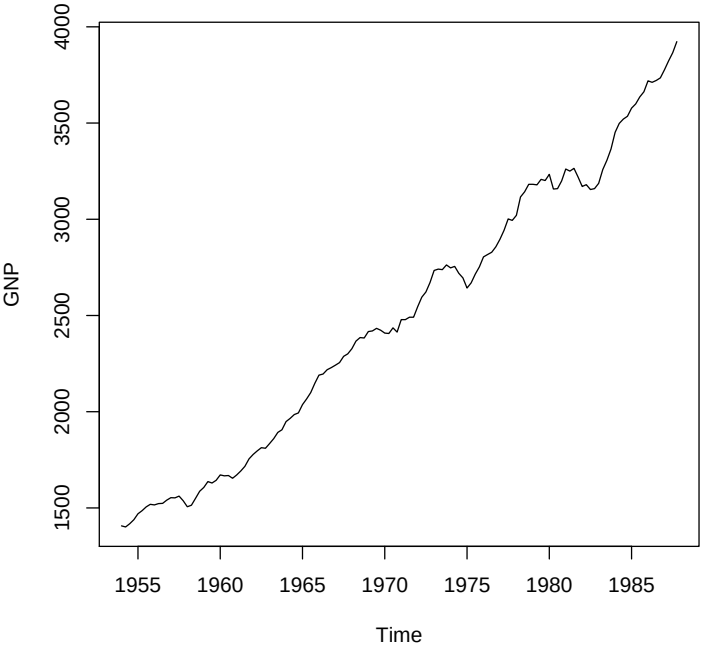
O mesmo efeito, mas talvez mais interessante uma vez que obtemos uma série multivariada, é conseguido com:

```
> x

           Qtr1       Qtr2       Qtr3       Qtr4
1959              0.8401383  0.3323405 -0.4444015
1960 -0.2466285 -1.2624572  0.4725261  2.6851647
1961 -0.5561170 -0.8198095 -2.2332913

> na.remove(cbind(lag(x, 2), lag(x), x))

         lag(x, 2)     lag(x)          x
1959 Q2 -0.4444015  0.3323405  0.8401383
1959 Q3 -0.2466285 -0.4444015  0.3323405
1959 Q4 -1.2624572 -0.2466285 -0.4444015
1960 Q1  0.4725261 -1.2624572 -0.2466285
1960 Q2  2.6851647  0.4725261 -1.2624572
1960 Q3 -0.5561170  2.6851647  0.4725261
1960 Q4 -0.8198095 -0.5561170  2.6851647
1961 Q1 -2.2332913 -0.8198095 -0.5561170
attr(,"na.removed")
[1]  1  2 11 12
```

9.2. Exemplos Ilustrativos

Vamos usar umas séries temporais sobre a economia dos USA que estão disponíveis na package "tseries".

```
> library(tseries)
> data(USeconomic)
> plot(GNP)
```



X11 2

- Mostrar os dados sobre a década de oitenta:

```
> plot(window(GNP, start = c(1980, 1)))
```

- Quais os valores médios em cada ano?

```
> ts(apply(matrix(GNP, ncol = 4, byrow = T), 1, mean), start = 1954)

Time Series:
Start = 1954
End = 1987
Frequency = 1
 [1] 1416.200 1494.875 1525.650 1551.125 1539.250 1629.075 1665.225 1708.650
 [9] 1799.375 1873.300 1973.250 2087.550 2208.350 2271.325 2365.625 2423.250
[17] 2416.175 2484.775 2608.525 2744.025 2729.325 2694.975 2826.675 2958.650
[25] 3115.150 3192.325 3187.175 3248.800 3166.025 3279.100 3501.375 3618.725
[33] 3721.725 3847.000
```

- Qual a variação percentual de trimestre para trimestre?

```
> pGNP <- window(GNP, end = c(1963, 4))
> pGNP

       Qtr1   Qtr2   Qtr3   Qtr4
1954 1406.8 1401.2 1418.0 1438.8
1955 1469.6 1485.7 1505.5 1518.7
1956 1515.7 1522.6 1523.7 1540.6
1957 1553.3 1552.4 1561.5 1537.3
1958 1506.1 1514.2 1550.0 1586.7
1959 1606.4 1637.0 1629.5 1643.4
1960 1671.6 1666.8 1668.4 1654.1
1961 1671.3 1692.1 1716.3 1754.9
1962 1777.9 1796.4 1813.1 1810.1
1963 1834.6 1860.0 1892.5 1906.1

> 100 * diff(pGNP)/lag(pGNP, -1)

            Qtr1        Qtr2        Qtr3        Qtr4
1954             -0.39806653  1.19897231  1.46685472
1955  2.14067278  1.09553620  1.33270512  0.87678512
1956 -0.19753737  0.45523520  0.07224484  1.10914222
1957  0.82435415 -0.05794116  0.58618913 -1.54979187
1958 -2.02953230  0.53781289  2.36428477  2.36774194
1959  1.24157056  1.90488048 -0.45815516  0.85302240
1960  1.71595473 -0.28715004  0.09599232 -0.85710861
1961  1.03984040  1.24454018  1.43017552  2.24902406
1962  1.31061599  1.04055346  0.92963705 -0.16546247
1963  1.35351638  1.38449798  1.74731183  0.71862616
```

9.3. Séries Irregulares

Nas séries irregulares os valores não têm necessariamente que ser espalhados de forma igual ao longo do tempo.

O R tem várias packages que definem objectos específicos para armazenar este tipo de dados.

No nosso estudo vamos usar a package "zoo".

Vejamos como criar um objecto "zoo" em R:

```
> library(zoo)
> (x <- zoo(rnorm(5), c("2006-3-21", "2006-3-24", "2006-6-21",
+     "2006-6-23", "2006-09-21")))

2006-09-21  2006-3-21  2006-3-24  2006-6-21  2006-6-23
-0.6318295  0.7720011 -0.9342685 -1.0330684  0.6087508

> index(x)

[1] "2006-09-21" "2006-3-21"  "2006-3-24"  "2006-6-21"  "2006-6-23"

> coredata(x)

[1] -0.6318295  0.7720011 -0.9342685 -1.0330684  0.6087508
```

O segundo argumento da função `zoo()`, as etiquetas temporais dos dados, pode ser de qualquer tipo (!), desde que faça sentido ordenar os seus valores (isto é desde que se possa aplicar a função `order()` aos valores). Fora isto, não há qualquer limitação o que torna este tipo de objectos bastante flexíveis.

9.3.1. Lidar com Datas e Horas em R

A maioria das vezes vamos usar indíces nos nossos objectos "zoo" que são tempo.
O R tem vários tipos de objectos para lidar com datas e tempo em geral. Vejamos alguns exemplos:

1. Os objectos do tipo "Date"

   Neste tipo de objectos as datas são representadas como o número de dias que passaram desde 1970-01-01.

```
> Sys.Date()

[1] "2006-12-13"

> hoje <- Sys.Date()
> format(hoje, "%d %b %Y")

[1] "13 Dec 2006"

> (dezsemanas <- seq(hoje, len = 10, by = "1 week"))

 [1] "2006-12-13" "2006-12-20" "2006-12-27" "2007-01-03" "2007-01-10"
 [6] "2007-01-17" "2007-01-24" "2007-01-31" "2007-02-07" "2007-02-14"

> weekdays(hoje)

[1] "Wednesday"

> months(hoje)

[1] "December"
```

```
> as.Date("2006-9-23") - as.Date("2003-04-30")

Time difference of 1242 days

> ISOdate(2001, 1, 1) - ISOdate(2000, 6, 14)

Time difference of 201 days

> cut(ISOdate(2001, 1, 1) + 70 * 86400 * runif(10), "weeks")

 [1] 2001-03-05 2001-02-05 2001-01-15 2001-01-01 2001-02-05 2001-01-15
 [7] 2001-01-22 2001-02-26 2001-02-12 2001-01-01
10 Levels: 2001-01-01 2001-01-08 2001-01-15 2001-01-22 ... 2001-03-05

> table(cut(seq(ISOdate(2006, 1, 1), to = ISOdate(2006, 12, 31),
+     by = "day"), "month"))

2006-01-01 2006-02-01 2006-03-01 2006-04-01 2006-05-01 2006-06-01 2006-07-01
        31         28         31         30         31         30         31
2006-08-01 2006-09-01 2006-10-01 2006-11-01 2006-12-01
        31         30         31         30         31
```

2. Objectos do tipo POSIXt

Este tipo de objectos pode ser usado para guardar tempo até ao segundo.

Na realidade são 2 tipos de objectos:

(a) POSIXct que representa as datas como o número de segundos que passaram desde 1970.

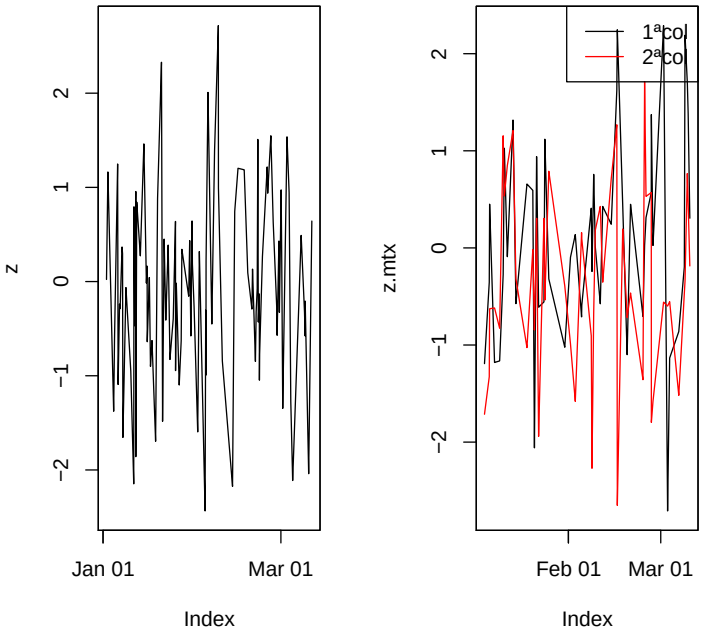
(b) POSIXlt que representa as datas como uma lista com várias componentes, como: "sec"; "min"; "hour"; "mday"; "mon"; "year"; etc.

```
> (z <- Sys.time())

[1] "2006-12-13 10:46:10 WET"

> as.POSIXlt(Sys.time(), "GMT")

[1] "2006-12-13 10:46:10 GMT"

> as.POSIXct("2006-12-23 12:45") - as.POSIXct("2006-12-21 21:54")

Time difference of 1.61875 days

> format(Sys.time(), "%a %b %d %X %Y %Z")

[1] "Wed Dec 13 10:46:10 AM 2006 WET"
```

```
> x <- c("1jan1960", "2jan1960", "31mar1960", "30jul1960")
> strptime(x, "%d%b%Y")

[1] "1960-01-01" "1960-01-02" "1960-03-31" "1960-07-30"

> dates <- c("02/27/92", "02/27/92", "01/14/92", "02/28/92", "02/01/92")
> times <- c("23:03:20", "22:29:56", "01:03:30", "18:21:03", "16:56:26")
> x <- paste(dates, times)
> strptime(x, "%m/%d/%y %H:%M:%S")

[1] "1992-02-27 23:03:20" "1992-02-27 22:29:56" "1992-01-14 01:03:30"
[4] "1992-02-28 18:21:03" "1992-02-01 16:56:26"
```

Voltemos aos objectos "zoo" !

```
> z <- zoo(rnorm(100), sort(ISOdate(2001, 1, 1) + 70 * 86400 *
+     runif(100)))
> z.mtx <- zoo(matrix(rnorm(100), 50, 2), sort(ISOdate(2001, 1,
+     1) + 70 * 86400 * runif(50)))
> par(mfrow = c(1, 2))
> plot(z)
> plot(z.mtx, plot.type = "single", col = 1:2)
> legend("topright", c("1ªcol", "2ªcol"), lty = 1, col = 1:2)
> par(mfrow = c(1, 1))
```



X11 2

Algumas funções úteis...

```
> (x <- zoo(rnorm(15), seq(as.Date("2006-09-01"), length = 15,
+     by = "days")))

 2006-09-01  2006-09-02  2006-09-03  2006-09-04  2006-09-05  2006-09-06
-0.11851869 -0.09824664 -0.72986259 -1.16233797  1.46792759  0.44086308
 2006-09-07  2006-09-08  2006-09-09  2006-09-10  2006-09-11  2006-09-12
 0.50146449 -0.42248998  0.32156780  1.20518896  0.28709799 -1.70507345
 2006-09-13  2006-09-14  2006-09-15
 0.35541530  0.35173200 -0.41374864

> start(x)

[1] "2006-09-01"

> end(x)

[1] "2006-09-15"

> x[3:6]

2006-09-03 2006-09-04 2006-09-05 2006-09-06
-0.7298626 -1.1623380  1.4679276  0.4408631

> x[coredata(x) > 0]

2006-09-05 2006-09-06 2006-09-07 2006-09-09 2006-09-10 2006-09-11 2006-09-13
 1.4679276  0.4408631  0.5014645  0.3215678  1.2051890  0.2870980  0.3554153
2006-09-14
 0.3517320
```

```
> x[as.Date("2006-09-14")]

2006-09-14
  0.351732

> window(x, start = as.Date("2006-09-10"), end = as.Date("2006-09-15"))

2006-09-10 2006-09-11 2006-09-12 2006-09-13 2006-09-14 2006-09-15
 1.2051890  0.2870980 -1.7050735  0.3554153  0.3517320 -0.4137486

> (y <- zoo(rnorm(10), seq(as.Date("2006-09-10"), length = 10,
+     by = "days")))

 2006-09-10  2006-09-11  2006-09-12  2006-09-13  2006-09-14  2006-09-15
-0.51304997 -0.20885203  0.42885074 -0.41474225  0.52181225  1.02155063
 2006-09-16  2006-09-17  2006-09-18  2006-09-19
-0.06395362  1.67940845 -1.66325803 -0.77636365

> x - y

2006-09-10 2006-09-11 2006-09-12 2006-09-13 2006-09-14 2006-09-15
 1.7182389  0.4959500 -2.1339242  0.7701576 -0.1700802 -1.4352993
```

```
> x <- zoo(rnorm(7), seq(as.Date("2006-09-01"), length = 7, by = "days"))
> y <- zoo(rnorm(7), seq(as.Date("2006-09-06"), length = 7, by = "days"))
> cbind(x, y)

                   x          y
2006-09-01  0.6452115         NA
2006-09-02 -0.6895073         NA
2006-09-03 -2.0397397         NA
2006-09-04 -0.8475332         NA
2006-09-05 -0.4511278         NA
2006-09-06 -0.8233070  0.5969922
2006-09-07 -0.5617405  1.4211448
2006-09-08         NA -0.4237792
2006-09-09         NA -0.4507535
2006-09-10         NA -0.2462414
2006-09-11         NA -0.6183404
2006-09-12         NA -0.4605929

> merge(x, y, all = FALSE)

                   x         y
2006-09-06 -0.8233070 0.5969922
2006-09-07 -0.5617405 1.4211448
```

```
> xx <- x[1:6]
> lag(xx)

2006-09-01 2006-09-02 2006-09-03 2006-09-04 2006-09-05
-0.6895073 -2.0397397 -0.8475332 -0.4511278 -0.8233070

> merge(xx, lag(xx))

                  xx    lag(xx)
2006-09-01  0.6452115 -0.6895073
2006-09-02 -0.6895073 -2.0397397
2006-09-03 -2.0397397 -0.8475332
2006-09-04 -0.8475332 -0.4511278
2006-09-05 -0.4511278 -0.8233070
2006-09-06 -0.8233070         NA

> diff(xx)

2006-09-02 2006-09-03 2006-09-04 2006-09-05 2006-09-06
-1.3347189 -1.3502323  1.1922064  0.3964055 -0.3721793
```

Valores desconhecidos

```
> x

2006-09-01 2006-09-02 2006-09-03 2006-09-04 2006-09-05 2006-09-06 2006-09-07
 0.6452115 -0.6895073 -2.0397397 -0.8475332 -0.4511278 -0.8233070 -0.5617405

> x[sample(1:length(x), 3)] <- NA
> x

2006-09-01 2006-09-02 2006-09-03 2006-09-04 2006-09-05 2006-09-06 2006-09-07
        NA -0.6895073         NA -0.8475332         NA -0.8233070 -0.5617405

> na.omit(x)

2006-09-02 2006-09-04 2006-09-06 2006-09-07
-0.6895073 -0.8475332 -0.8233070 -0.5617405

> na.contiguous(x)

2006-09-06 2006-09-07
-0.8233070 -0.5617405

> na.approx(x)

2006-09-02 2006-09-03 2006-09-04 2006-09-05 2006-09-06 2006-09-07
-0.6895073 -0.7685203 -0.8475332 -0.8354201 -0.8233070 -0.5617405

> na.locf(x)

2006-09-02 2006-09-03 2006-09-04 2006-09-05 2006-09-06 2006-09-07
-0.6895073 -0.6895073 -0.8475332 -0.8475332 -0.8233070 -0.5617405
```

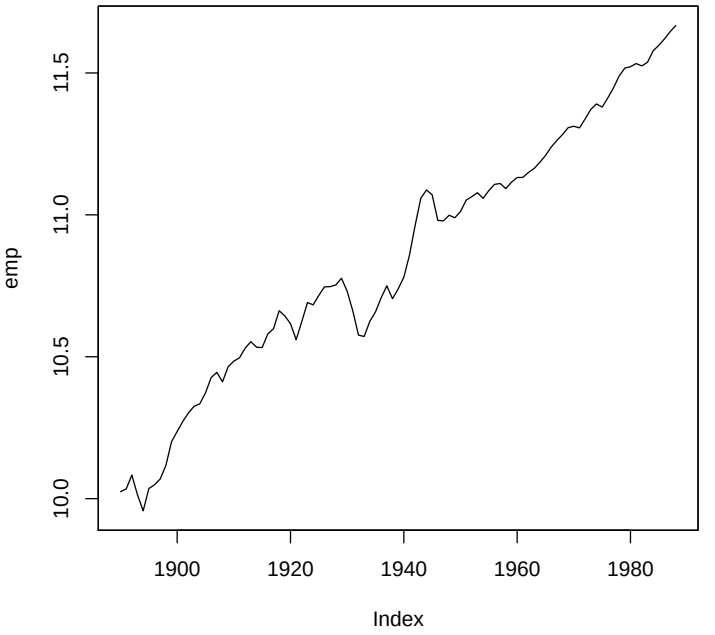
Funções "deslizantes"...

```
> x

2006-09-01 2006-09-02 2006-09-03 2006-09-04 2006-09-05 2006-09-06 2006-09-07
        NA -0.6895073         NA -0.8475332         NA -0.8233070 -0.5617405

> rapply(x, 4, mean)

2006-09-02 2006-09-03 2006-09-04 2006-09-05
        NA         NA         NA         NA

> rapply(x, 4, mean, align = "right")

2006-09-04 2006-09-05 2006-09-06 2006-09-07
        NA         NA         NA         NA

> rollmean(x, 3)

2006-09-02 2006-09-03 2006-09-04 2006-09-05 2006-09-06
        NA         NA         NA         NA         NA

> rollmean(x, 3, na.pad = T)

2006-09-01 2006-09-02 2006-09-03 2006-09-04 2006-09-05 2006-09-06 2006-09-07
        NA         NA         NA         NA         NA         NA         NA
```

9.4. Exemplos Ilustrativos

Vamos usar uma série temporal que vem com a package "tseries".

```
> library(tseries)
> data(NelPlo)
> emp <- as.zoo(emp)
> plot(emp)
```



X11 2

1. Qual o ano com maior taxa de desemprego?

```
> index(emp)[which.max(emp)]

[1] 1988
```

2. Quais os anos em que houve um decréscimo da taxa de desemprego?

```
> index(emp)[which(diff(emp) < 0)]

 [1] 1892 1893 1907 1913 1914 1918 1919 1920 1923 1929 1930 1931 1932 1937 1944
[16] 1945 1946 1948 1953 1957 1970 1974 1981
```

3. Quais os valores para a década de 1940?

```
> window(emp, start = 1940, end = 1949)

    1940     1941     1942     1943     1944     1945     1946     1947
10.78021 10.85842 10.96336 11.05864 11.08782 11.07069 10.98020 10.97900
    1948     1949
10.99881 10.98979
```

4. Quais as variações percentuais da taxa de desemprego na década de 1970?

```
> e70 <- window(emp, start = 1970, end = 1979)
> 100 * diff(e70)/lag(e70, -1)

       1971        1972        1973        1974        1975        1976
-0.05147533  0.28102685  0.29663525  0.16819677 -0.09884182  0.28606283
       1977        1978        1979
 0.30970308  0.36581579  0.24096035
```

5. Qual a variância da taxa de desemprego, calculada nos 10 anos anteriores, ao longo do tempo?

```
> rapply(emp, 10, var, align = "right")
```

```
        1899         1900         1901         1902         1903         1904
0.0043164053 0.0072301097 0.0105064830 0.0143260764 0.0163156148 0.0138364058
        1905         1906         1907         1908         1909         1910
0.0133219136 0.0125663261 0.0102792798 0.0067389970 0.0059566240 0.0053514401
        1911         1912         1913         1914         1915         1916
0.0048066204 0.0047277121 0.0046669007 0.0034282596 0.0024581737 0.0027341644
        1917         1918         1919         1920         1921         1922
0.0031059528 0.0034275996 0.0035031205 0.0029861848 0.0022791049 0.0021025168
        1923         1924         1925         1926         1927         1928
0.0028543544 0.0027377919 0.0025671279 0.0032136650 0.0035843655 0.0042710028
        1929         1930         1931         1932         1933         1934
0.0050224383 0.0044227507 0.0022415504 0.0034268368 0.0053077916 0.0058145934
        1935         1936         1937         1938         1939         1940
0.0058195926 0.0054386936 0.0054786945 0.0049369967 0.0042375919 0.0051239749
        1941         1942         1943         1944         1945         1946
0.0083054614 0.0128951684 0.0188448027 0.0238606512 0.0250455107 0.0225296557
        1947         1948         1949         1950         1951         1952
0.0201693089 0.0145986154 0.0090390329 0.0042896667 0.0019530632 0.0017194214
        1953         1954         1955         1956         1957         1958
0.0018824586 0.0015972135 0.0017591428 0.0019768914 0.0018531014 0.0015486516
        1959         1960         1961         1962         1963         1964
0.0010269118 0.0007090677 0.0006810974 0.0007783365 0.0009751658 0.0009950459
        1965         1966         1967         1968         1969         1970
0.0013791465 0.0021631961 0.0030389876 0.0034730088 0.0040974517 0.0043656594
        1971         1972         1973         1974         1975         1976
0.0038307450 0.0035258168 0.0033690620 0.0031995350 0.0025923573 0.0025067004
        1977         1978         1979         1980         1981         1982
0.0028860283 0.0038476572 0.0050660125 0.0055342408 0.0050758415 0.0043271895
        1983         1984         1985         1986         1987         1988
0.0038699324 0.0038517439 0.0030605487 0.0025883021 0.0025841365 0.0030932399
```