- Tutorial
- Exercícios
- Apostila

3. Leitura e Manipulação de Dados

Leitura de Dados

Entrada de Dados Diretamente no R

Função "c()" (concatenate ou combine)

As funções de criação de vetores já foram detalhadas na seção Funções Matemáticas e Estatísticas desta apostila. Basta lembrar aqui que todas elas são usadas para entrar diretamente dados em vetores no R:

```
> meu.vetor <- c(10.5,11.3,12.4,5.7)
> meu.vetor
[1] 10.5 11.3 12.4 5.7
>
> vetor.vazio <- c()
> vetor.vazio
NULL
```

Função "matrix()"

A função `matrix` cria uma matriz com os valores do argumento `data`. O números de linhas e colunas são definidos pelos argumentos `nrow` e `ncol`:

```
> minha.matriz <- matrix(data=1:12,nrow=3,ncol=4)
> minha.matriz
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Como o *default* do argumento `data` é NA, se ele é omitido o resultado é uma matriz vazia:

```
> matriz.vazia <- matrix(nrow=3,ncol=4)
> matriz.vazia
     [,1] [,2] [,3] [,4]
[1,]   NA   NA   NA   NA
[2,]   NA   NA   NA   NA
[3,]   NA   NA   NA   NA
```

Também por *default*, os valores são preenchidos por coluna. Para preencher por linha basta o alterar o argumento `byrow` para TRUE:

```
> minha.matriz <- matrix(data=1:12,nrow=3,ncol=4,byrow=T)
> minha.matriz
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

Se o argumento `data` tem menos elementos do que a matriz, eles são repetidos até preenchê-la:

```
> elementos <- matrix(c("ar","água","terra","fogo","Leeloo"),ncol=4,nrow=4)
Warning message:
comprimento dos dados [5] não é um submúltiplo ou múltiplo do número de
linhas [4] na matrix
> elementos
     [,1]    [,2]     [,3]     [,4]
[1,] "ar"    "Leeloo" "fogo"   "terra"
[2,] "água"  "ar"     "Leeloo" "fogo"
[3,] "terra" "água"   "ar"     "Leeloo"
[4,] "fogo"  "terra"  "água"   "ar"
```

Função "data.frame()"

Com a função `data.frame` reunimos vetores de mesmo comprimento em um só objeto:

```
> nome <- c("Didi","Dedé","Mussum","Zacarias")
> ano.nasc <- c(1936,1936,1941,1934)
> vive <- c("V","V","F","F")
> trapalhoes <- data.frame(nomes,ano.nasc,vive)
> trapalhoes
     nomes ano.nasc vive
1     Didi     1936    V
2     Dedé     1936    V
3   Mussum     1941    F
4 Zacarias     1934    F
>
# O mesmo, em um só comando:
> trapalhoes <- data.frame(nomes=c("Didi","Dedé","Mussum","Zacarias"),
ano.nasc=c(1936,1936,1941,1934), vive=c("V","V","F","F"))
```

Função "edit()"

Esta função abre uma interface simples de edição de dados em formato planilha, e é útil para pequenas modificações. Mas para salvar as modificações atribua o resultado da função `edit` a um objeto:

```
trapalhoes.2<-edit(trapalhoes)
```

	nomes	ano.nasc	vive
1	Didi	1936	V
2	Dedé	1936	V
3	Mussum	1941	F
4	Zacarias	1934	F

Dados que já Estão em Arquivos

Leitura e Exportação de Arquivos-Texto: ''read.table()'' e ''write.table()''

Para conjuntos de dados grandes, é mais prático gerar um arquivo de texto (ASCII) a partir de uma planilha ou banco de dados, e usar a função `read.table` para ler os dados para um objeto no R.

Para criar um objeto com os dados do arquivo gbmam93.csv (apagar extensão .pdf), por exemplo, digitamos:

```
> gbmam93 <- read.table(file="gbmam93.csv",header=T,row.names=1,sep=",")
> gbmam93
   a b c d e f g h i j k l m n o p q r s
1  1 1 1 0 1 1 1 0 0 1 0 1 1 1 1 1 1 1 1
2  1 1 0 1 1 0 1 0 1 0 0 1 1 0 1 1 1 1 1
3  1 0 0 1 1 0 1 0 1 0 0 1 1 1 1 1 1 1 1
4  1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
5  1 0 0 1 1 0 1 0 1 0 0 1 1 0 0 0 1 0 0
6  1 1 1 0 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1
7  1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
8  1 0 1 1 1 0 0 0 1 0 0 1 1 0 0 1 1 0 1
9  1 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0
10 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 0
11 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1
12 1 0 0 1 1 0 0 0 1 0 0 1 0 0 0 0 0 0 1
13 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1
14 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
```

O argumento `header=T` indica que a primeira linha são os nomes das variáveis, assim como `row.names=1` indica que a primeira coluna deve ser usada para os nomes das linhas. O argumento `sep` indica qual é o sinal de separação de cada registro, no caso vírgulas.

Esses e os outros argumentos da função a tornam extremamente flexível para ler dados em arquivos texto. Consulte a ajuda para mais informações, e também para conhecer as variantes `read.csv` e `read.delim`.

Para exportar um objeto para um arquivo texto, use a função `write.table`, que tem a mesma

lógica.

Conjuntos de Dados Distribuídos com os Pacotes do R

Muitos pacotes do R incluem conjuntos de dados para exemplos, treinamento e verificação de análises. Se o pacote já está carregado (funções `library` ou `require`) todos os seus objetos estão disponíveis, inclusive os objetos de dados. Incluindo as séries temporais de número de peles de linces caçados no Canadá, analisadas pelo ecólogo Charles Elton obtém-se:

```
> lynx
Time Series:
Start = 1821
End = 1934
Frequency = 1
  [1]  269  321  585  871 1475 2821 3928 5943 4950 2577  523   98  184  279
409
 [16] 2285 2685 3409 1824  409  151   45   68  213  546 1033 2129 2536  957
361
 [31]  377  225  360  731 1638 2725 2871 2119  684  299  236  245  552 1623
3311
 [46] 6721 4254  687  255  473  358  784 1594 1676 2251 1426  756  299  201
229
 [61]  469  736 2042 2811 4431 2511  389   73   39   49   59  188  377 1292
4031
 [76] 3495  587  105  153  387  758 1307 3465 6991 6313 3794 1836  345  382
808
 [91] 1388 2713 3800 3091 2985 3790  674   81   80  108  229  399 1132 2432
3574
[106] 2935 1537  529  485  662 1000 1590 2657 3396
```

Como qualquer objeto de um pacote, `lynx` tem um arquivo de ajuda, que é exibido com o comando `help(lynx)` ou `?lynx`:

```
lynx {datasets}                                                   R
Documentation

Annual Canadian Lynx trappings 1821-1934

Description:

     Annual numbers of lynx trappings for 1821-1934 in Canada. Taken
     from Brockwell & Davis (1991), this appears to be the series
     considered by Campbell & Walker (1977).

Usage:

     lynx

Source:
```

```
     Brockwell, P. J. and Davis, R. A. (1991) Time Series and
     Forecasting Methods.  Second edition. Springer. Series G (page
     557).

References:

     Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) The New S
     Language. Wadsworth & Brooks/Cole.

     Campbell, M. J.and A. M.  Walker (1977).  A Survey of statistical
     work on the Mackenzie River series of annual Canadian lynx
     trappings for the years  1821-1934 and a 'new' analysis. Journal of
     the Royal Statistical Society series A, 140, 411-431.
```

A página de ajuda mostra que o objeto de dados `lynx` está no pacote `datasets` que contém uma grande quantidade de conjuntos de dados. Para ter mais informações, execute o comando:

```
help(datasets)
```

Esse pacote faz parte da distribuição básica do R, e é carregado automaticamente quando se executa o R:

```
> search()
[1] ".GlobalEnv"        "package:stats"     "package:graphics"
[4] "package:grDevices" "package:utils"     "package:datasets"
[7] "package:methods"   "Autoloads"         "package:base"
```

Para fazer uma cópia de um objeto de dados de um pacote em sua área de trabalho, use a função `data`:

```
> ls()
[1] "gbmam93"     "trapalhoes"  "vetor.vazio"
> data(lynx)
> ls()
[1] "gbmam93"     "lynx"        "trapalhoes"  "vetor.vazio"
> data(BCI,package="vegan")
> ls()
[1] "gbmam93"     "lynx"   "BCI"     "trapalhoes"  "vetor.vazio"
```

No segundo caso, o pacote *vegan*, que tem o conjunto de dados, não está carregado e por isso deve ser indicado no argumento `package`.

Importação de Arquivos de Outros Pacotes Estatísticos

O pacote *foreign* contém funções para importar e exportar diretamente arquivos de pacotes estatísticos.

Como é um pacote recomendado pelo *R Core Team*, faz parte da distribuição básica do R, e

provavelmente está disponível **mas precisa ser carregado** com o comando `library(foreign)`.

Para mais informações, digite `help(package=foreign)`.

Exercícios

Os exercícios **103.1** e **103.2** estão disponíveis no NotaR, um sistema automático de correção de exercícios. Caso esteja fazendo "o curso" do R é preciso postá-lo nesse sistema. Caso não esteja, poste o código no sistema para saber seu aproveitamento.

- notaR

Exercício 3.1. *Construir uma matriz de distâncias*

Abaixo as distâncias por estradas entre quatro cidades da Europa, em quilômetros:

- Atenas a Madri: 3949
- Atenas a Paris: 3000
- Atenas a Estocolmo: 3927
- Madri a Paris: 1273
- Madri a Estocolmo: 3188
- Paris a Estocolmo: 1827

1. Construa uma matriz de distâncias com esses valores.
2. Compare sua matriz com objeto `eurodist`, disponível no pacote `datasets`.

Exercício 3.2. *Criação de um data frame*

Imagine um experimento em que hamsters de dois fenótipos (claros e escuros) recebem três tipos diferentes de dieta, e no qual as diferenças dos pesos (g) entre o fim e o início do experimento sejam:

	DIETA A	**DIETA B**	**DIETA C**
CLAROS	0.1 1.1 3.7	5.7 -1.2 -1.5	3.0 -0.4 0.6
ESCUROS	1.5 -0.1 2.0	0.6 -3.0 -0.3	-0.2 0.3 1.5

1. Crie um *data frame* com esses dados, na qual cada hamster seja uma linha, e as colunas sejam as variáveis cor, dieta e variação do peso.

DICA: Use as funções de gerar repetições para criar os vetores dos tratamentos.

Exercício 3.3. *Leitura de Arquivo Texto*

Crie um objeto com os dados do arquivo Conjunto de Dados: Mamíferos na Great Basin (EUA)

Exercício 3.4. *Buscando um Arquivo de um Pacote*

1. Descubra um objeto de dados de um pacote já carregado no R, e carregue esse pacote em sua área de trabalho.
2. Faça o mesmo para um pacote que está disponível em sua instalação de R, mas **não** está carregado.
3. Por que dados de pacotes carregados não são exibidos pelo comando `ls()`?
4. Consulte a página de ajuda da função `ls` para descobrir como listar objetos de um pacote carregado.

DICAS:

- O comando `search()` retorna uma lista dos pacotes carregados e o comando `library()` lista os pacotes disponíveis em seu computador para serem carregados. O comando `library(nome do pacote)` carrega um pacote.
- Para listar os objetos de dados nos pacotes carregados, use `data()`. Na saída desse comando há instruções de como listar todos os objetos de dados, incluindo os de pacotes não carregados.

Tipos de Objetos de Dados

Vetores, matrizes e listas são objetos com características diferentes, formalmente definidas no R como *atributos*. Em uma linguagem orientada a objetos, são esses atributos que definem o contexto para a execução de um comando e, portanto, seu resultado. Fazendo uma analogia com o mundo físico, uma mesma ação tem resultados diferentes de acordo com as características do objeto em que é aplicada.

O que é importante reter aqui é que os resultados que obtemos de um comando no R (incluindo as mensagens de erro!) são em boa parte definidas pelo objeto de dados, e não apenas pela função. Por isso, quando enfrentar algum problema, verifique com cuidado na documentação se o seu comando se aplica à classe de objeto de dados que você está usando e, se sim, como.[1)]

Atributos de um Objeto de Dados

Todo objeto no R tem dois atributos básicos, que são o tipo de dado[2)] e o número de elementos que

contêm. As funções `mode` e `length` retornam esses atributos:

```
> pares
[1]  2  4  6  8 10
> mode(pares)
[1] "numeric"
> length(pares)
[1] 5
```

Objetos também podem ter uma ou mais classes. Um vetor numérico pode ser da classe dos inteiros ou da classe dos fatores. Um objeto da classe matriz pode ter dados do tipo numérico, lógicos[3] ou caracteres. O comando `class` retorna a classe de um objeto:

```
> matriz.letras
     [,1] [,2] [,3] [,4]
[1,] "a"  "b"  "c"  "d"
[2,] "a"  "b"  "c"  "d"
[3,] "a"  "b"  "c"  "d"
> mode(matriz.letras)
[1] "character"
> class(matriz.letras)
[1] "matrix"
> matriz.numeros
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    1    2    3    4
[3,]    1    2    3    4
> mode(matriz.numeros)
[1] "numeric"
> class(matriz.numeros)
[1] "matrix"
```

Vetores

São um conjunto de elementos do mesmo tipo, como números ou caracteres. Há várias classes de vetores. Os vetores que temos trabalhado até agora são numéricos:

```
> class( a )
[1] "numeric"
> class( b )
[1] "integer"
> class( c )
[1] "integer"
```

A classe '`numeric`' designa números reais enquanto que a classe '`integer`' designa números inteiros.

É possível ter no R um vetor tipo '`character`' formado por palavras ou frases:

```
> sp = c( "Myrcia sulfiflora", "Syagrus romanzoffianus" , "Tabebuia
cassinoides", "Myrcia sulfiflora" )
> mode( sp )
[1] "character"
>
```

Vetores da Classe Fator e as Funções ''table'' e ''tapply''

Os fatores são uma classe especial de vetores, que definem variáveis categóricas de classificação, como os tratamentos em um experimento fatorial, ou categorias em uma tabela de contingência.

A função `factor` cria um fator, a partir de um vetor :

```
> sexo <- factor(rep(c("F","M"),each=9))
> sexo
 [1] M M M M M M M M M F F F F F F F F
Levels: F M
>
> numeros <- rep(1:3,each=3)
> numeros
[1] 1 1 1 2 2 2 3 3 3
> numeros.f <- factor(numeros)
> numeros.f
[1] 1 1 1 2 2 2 3 3 3
Levels: 1 2 3
```

Em muitos casos, indicar que um vetor é um fator é importante para a análise, e várias funções no R exigem variáveis dessa classe, ou têm respostas específicas para ela [4].

Note que fatores têm um atributo que especifica seus níveis ou categorias (`levels`) , que seguem ordem alfanumérica crescente, por *default*. Como essa ordem é importante para muitas análises, pode-se alterá-la com o argumento `levels`, por exemplo para colocar o controle antes dos tratamentos:

```
> tratamentos <- factor(rep(c("Controle","Adubo A","Adubo B"),each=4))
> tratamentos
 [1] Controle Controle Controle Controle Adubo A  Adubo A  Adubo A  Adubo A
 [9] Adubo B  Adubo B  Adubo B  Adubo B
Levels: Adubo A Adubo B Controle
>
> tratamentos <- factor(rep(c("Controle","Adubo A","Adubo
B"),each=4),levels=c("Controle","Adubo A","Adubo B"))
> tratamentos
 [1] Controle Controle Controle Controle Adubo A  Adubo A  Adubo A  Adubo A
 [9] Adubo B  Adubo B  Adubo B  Adubo B
Levels: Controle Adubo A Adubo B
```

Há ainda a função `levels`, que retorna os níveis de um fator:

```
> tratamentos <- factor(rep(1:3,each=4))
> tratamentos
 [1] 1 1 1 1 2 2 2 2 3 3 3 3
Levels: 1 2 3
>
> levels(tratamentos)
[1] "1" "2" "3"
```

Fatores podem conter níveis não usados (vazios):

```
> politicos <- factor(rep("corrupto",10),levels=c("corrupto","honesto"))
> politicos
 [1] corrupto corrupto corrupto corrupto corrupto corrupto corrupto corrupto
 [9] corrupto corrupto
Levels: corrupto honesto
```

Duas Coisa que Você Precisa Saber sobre Fatores

- Um fator pode ter níveis para os quais não há valores. Isso pode acontecer quando alguns valores são selecionados ou excluídos (ver item sobre indexação, nessa seção). Isso é muito importante, pois afeta o resultado de muitas funções (Veja os exemplos da função `tapply`, a seguir e também no item sobre indexação de fatores).
- Em versões mais antigas do R, a função `read.table` transformava em fatores todas as variáveis que tivessem caracteres por *default*. Em versões mais recentes, esse não é mais o padrão. Mas você pode decidir como você quer lidar com os caracteres durante a leitura usando o argumento `as.is`.

A função ''tapply''

Para aplicar uma função aos subconjuntos de um vetor definidos por um fator use a função `tapply`:

```
> pop.2007
Feira de Santana          Salvador        São Paulo          Niterói
          544113           2714119          11104712           476669
     Nova Iguaçu           Recife      Santo André   Rio de Janeiro
          858150           1528970           676846          6178762
        Sorocaba         Campinas           Osasco        Guarulhos
          590846           1073020           724368          1289047
        Jaboatão
          661901
```

```
> regiao
 [1] NE NE SE SE SE NE SE SE SE SE SE SE NE
Levels: NE SE

##Soma do número de habitantes das cidades por região
> tapply(X=pop.2007,INDEX=regiao,FUN=sum)
      NE       SE
 5449103 22972420

#Número de habitantes da cidade mais populosa de cada região
> tapply(X=pop.2007,INDEX=regiao,FUN=max)
      NE       SE
 2714119 11104712
```

Os argumentos básicos são o vetor de valores (X), o fator que será usado para definir os subconjuntos (INDEX), e a função que será aplicada (FUN). É possível usar mais de um fator para definir os subconjuntos:

```
> sexo <- factor(rep(c("F","M"),each=9))
> dieta <- factor(rep(rep(c("normal","light","diet"),each=3),2),
levels=c("normal","light","diet"))
> peso <- c(65,69,83,90,58,84,85,74,92,71,72,78,67,65,62,74,73,68)
> sexo
 [1] F F F F F F F F F M M M M M M M M M
Levels: F M
> dieta
 [1] normal normal normal light  light  light  diet   diet   diet   normal
[11] normal normal light  light  light  diet   diet   diet
Levels: normal light diet
> peso
 [1] 65 69 83 90 58 84 85 74 92 71 72 78 67 65 62 74 73 68
> ##Media de peso por sexo e dieta
> tapply(peso,list(sexo,dieta),mean)
      diet    light   normal
F 83.66667 77.33333 72.33333
M 71.66667 64.66667 73.66667
```

A função "table"

Para contar elementos em cada nível de um fator, use a função `table`:

```
> table(politicos)
politicos
corrupto  honesto
      10        0
```

A função pode fazer tabulações cruzadas, gerando uma tabela de contingência[5]:

```
> table(sexo,dieta)
```

```
    dieta
sexo normal light diet
   F      3     3    3
   M      3     3    3
```

A função `table` trata cada valor de um vetor como um nível de um fator. Portanto, é útil também para contar a frequência de valores em vetores de números inteiros e de caracteres:

```
> x <- rep(1:5,each=10)
> class(x)
[1] "integer"
> table(x)
x
 1  2  3  4  5
10 10 10 10 10

> y <- rep(c("a","b","c"),20)
> class(y)
[1] "character"
> table(y)
y
 a  b  c
20 20 20
```

Listas

Uma lista é um objeto composto de vetores que podem ser diferentes classes e tamanhos, e podem ser criadas com o comando `list`

```
> minha.lista <- list(um.vetor=1:5, uma.matriz=matrix(1:6,2,3),
um.dframe=data.frame(seculo=c("XIX","XX","XXI"),inicio=c(1801,1901,2001)))
> minha.lista
$um.vetor
[1] 1 2 3 4 5

$uma.matriz
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

$um.dframe
  seculo inicio
1    XIX   1801
2     XX   1901
3    XXI   2001
```

Nas palavras de Bill Venables, listas são como “varais” onde se pode pendurar qualquer outro objeto, inclusive outras listas, o que as torna **objetos recursivos**:

```
super.lista <- list(lista.velha=minha.lista, um.numero=1)
```

Alguns objetos têm como atributo o nome de seus elementos, como é o caso das listas. A função `names` retorna esses nomes:

```
> names(minha.lista)
[1] "um.vetor"   "uma.matriz" "um.dframe"
>
> names(super.lista)
[1] "lista.velha" "um.numero"
```

Seleção Rápida de um Objeto em uma Lista

O operador $ permite selecionar rapidamente um objeto de uma lista:

```
> minha.lista$um.vetor
[1] 1 2 3 4 5
>
> names(super.lista)
[1] "lista.velha" "um.numero"
>
> names(super.lista$lista.velha)
[1] "um.vetor"   "uma.matriz" "um.dframe"
>
> names(super.lista$lista.velha$um.dframe)
[1] "seculo" "inicio"
```

Data Frames

A tabela de dados (*data frame*) é um tipo especial de lista, composta por vetores de mesmo tamanho, mas que podem ser de classes diferentes:

```
> names(trapalhoes)
[1] "nomes"    "ano.nasc" "vive"
>
> trapalhoes
     nomes ano.nasc  vive
1     Didi     1936  TRUE
2     Dedé     1936  TRUE
3   Mussum     1941 FALSE
4 Zacarias     1934 FALSE
>
> class(trapalhoes$nomes)
[1] "character"
> class(trapalhoes$ano.nasc)
[1] "numeric"
> class(trapalhoes$vive)
```

```
[1] "logical"
```

Seleção Rápida de Variáveis em Data Frames

Todas as operações já descritas para listas são válidas para os *data frames*.

Assim como para listas, o operador $ pode ser usado selecionar um dos vetores que compõem um *data frame*, como no exemplo acima.

Esse operador também pode ser usado para criar novas variáveis (vetores) e acrescentá-las ao objeto. Para isso, basta acrescentar após o operador o nome da nova variável, e atribuir a ela um valor:

```
> trapalhoes$idade.2008 <- 2008 - trapalhoes$ano.nasc
> trapalhoes
     nomes ano.nasc  vive idade.2007
1     Didi     1936  TRUE         72
2     Dedé     1936  TRUE         72
3   Mussum     1941 FALSE         67
4 Zacarias     1934 FALSE         74
```

A Função "aggregate"

A função `aggregate` gera subconjuntos de cada um dos vetores de um *data frame*, executa uma função para cada um desses subconjuntos, e retorna um novo *data frame* com os resultados.

Como seu resultado é sempre um *data frame*, a função `aggregate` é mais adequada que `tapply` para fazer estatísticas de muitos casos por uma ou muitas combinações de critérios:

```
> carros.marcas
 [1] Chevrolet Chevrolet Chevrolet Chevrolet Chevrolet Chevrolet Chevrolet
 [8] Chevrolet Ford      Ford      Ford      Ford      Ford      Ford
[15] Ford      Ford      Toyota    Toyota    Toyota    Toyota
Levels: Chevrolet Ford Toyota
> carros.numeros
   Price Horsepower Weight
12  13.4        110   2490
13  11.4        110   2785
14  15.1        160   3240
15  15.9        110   3195
16  16.3        170   3715
17  16.6        165   4025
18  18.8        170   3910
19  38.0        300   3380
31   7.4         63   1845
32  10.1        127   2530
33  11.3         96   2690
34  15.9        105   2850
35  14.0        115   2710
```

```
36  19.9        145   3735
37  20.2        140   3325
38  20.9        190   3950
84   9.8         82   2055
85  18.4        135   2950
86  18.2        130   3030
87  22.7        138   3785

> aggregate(x=carros.numeros,by=list(carros.marcas),FUN=mean)
    Group.1   Price Horsepower   Weight
1 Chevrolet 18.1875    161.875 3342.500
2      Ford 14.9625    122.625 2954.375
3    Toyota 17.2750    121.250 2955.000
```

Os argumentos básicos da função são o objeto com os valores (x), o(s) fator(es) que definem os subgrupos (by, que deve ser uma lista), e a função a ser aplicada a cada vetor em x (FUN).

Matrizes e "Arrays"

Matrizes são vetores cujos valores são referenciados por dois índices, o número da linha e o número da coluna.

```
> my.matrix
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
[2,]    2    2    2    2
[3,]    3    3    3    3
```

Os índices entre colchetes são a referência do par [linha,coluna][6]. Esses índices são exibidos quando as dimensões não têm nomes, que são controlados pelas funções `rownames` e `colnames`:

```
> rownames(my.matrix) <- c("R1","R2","R3")
> colnames(my.matrix) <- c("C1","C2","C3","C4")
>
> my.matrix
   C1 C2 C3 C4
R1  1  1  1  1
R2  2  2  2  2
R3  3  3  3  3
```

A função `dimnames` retorna uma lista com os nomes de cada dimensão de uma matriz

```
> dimnames(my.matrix)
[[1]]
[1] "R1" "R2" "R3"

[[2]]
[1] "C1" "C2" "C3" "C4"
```

A função `dim` retorna o comprimento de cada dimensão de uma matriz, no caso três linhas e quatro colunas:

```
> dim(my.matrix)
[1] 3 4
```

Totais Marginais: a função ''apply''

Para aplicar qualquer função a uma das dimensões de uma matriz, use a função `apply`:

```
##Soma dos valores de cada linha:
> apply(X=my.matrix,MARGIN=1,FUN=sum)
R1 R2 R3
 4  8 12
>
##Máximo de cada coluna
> apply(X=my.matrix,MARGIN=2,FUN=max)
C1 C2 C3 C4
 3  3  3  3
```

Os argumentos básicos da função são a matriz (X), a dimensão (MARGIN, valor 1 para linhas, valor 2 para colunas) e a função a aplicar (FUN).

Álgebra Matricial

Todas as operações matriciais podem ser realizadas com as matrizes numéricas. O R possui funções para estas operações, como %*%, para multiplicação, entre outras:

```
> m
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
[3,]    1    1    1
[4,]    1    1    1
> n
[1] 1 2 3
##Multiplicação usual: NÃO É MULTIPLICAÇÃO MATRICIAL
> m*n
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
[3,]    3    3    3
##Diagonal da matriz resultante
> diag(m*n)
[1] 1 2 3
>
##Multiplicação matricial
> m%*%n
```

```
     [,1]
[1,]    6
[2,]    6
[3,]    6
```

Consulte a ajuda para detalhes.

Arrays

Os *arrays* são a generalização das matrizes para mais de duas dimensões. Um exemplo é o objeto `Titanic`, com as seguintes dimensões:

```
> dim(Titanic)
[1] 4 2 2 2
```

Com os seguintes nomes:

```
> dimnames(Titanic)
$Class
[1] "1st"  "2nd"  "3rd"  "Crew"

$Sex
[1] "Male"   "Female"

$Age
[1] "Child" "Adult"

$Survived
[1] "No"  "Yes"
```

Todas as operações aplicáveis a matrizes também o são para *arrays*:

```
> adultos.por.sexo <- apply(Titanic, c(2, 4), sum)
##Mulheres primeiro?
> adultos.por.sexo
        Survived
Sex        No Yes
  Male   1364 367
  Female  126 344

##Como o objeto resultante é uma matriz, pode-se aplicar apply novamente:
> adultos.por.sexo/apply(adultos.por.sexo,1,sum)
        Survived
Sex              No       Yes
  Male   0.7879838 0.2120162
  Female 0.2680851 0.7319149
```

Exercícios

Exercício 3.5. *Classes de Objetos*

O pacote "datasets" contém vários conjuntos de dados para uso em treinamento com a linguagem R.

O conjunto "iris" é distribuído de duas formas diferentes, nos objetos `iris` e `iris3`. São quatro medidas de flores de três espécies de *Iris* (Iridaceae).

1. Quais são as classes desses dois objetos?
2. Calcule a média de cada uma das quatro medidas por espécie, dos dois objetos.
3. Os nomes das variáveis estão em inglês. Mude-os para português no objeto `iris`. (*DICA: Como tudo mais no R, os resultados da função `names` podem ser armazenados em um objeto*)

Exercício 3.6. *Importação e modificação de um arquivo texto*

1. Crie um objeto com os dados do arquivo-texto esaligna.csv.
2. Verifique o conteúdo do objeto resultante, com a função `summary`.
3. Acrescente uma nova coluna ao data frame resultante, com a soma das biomassas de folhas e do tronco de cada árvore.
4. Acrescente outra coluna, com o valor da área basal de cada árvore.

Exercício 3.7. *Alteração de Atributos de um Objeto*

1. Crie o seguinte objeto da classe matriz:

```
> minha.matriz <-
matrix(seq(from=2,to=10,by=2),ncol=5,nrow=3,byrow=T)
```

1. Mude os nomes das linhas para "L1" a "L3" e das colunas para "C1" a "C5". **Dica**: A função `paste` pode poupar trabalho.
2. O que acontece com a matriz após o comando:

```
> dim(minha.matriz) <- NULL
```

3. Como reverter este resultado?
4. Se você respondeu ao item anterior, percebeu que os nomes da matriz foram perdidos. Como restituí-los sem ter que refazer o passo 2? **Dica**: o resultado da função `dimnames` é uma lista com os nomes de cada dimensão.

Exercício 3.8. *Agregação*

1. Crie um *data frame* com os dados do arquivo itirapina.csv
2. A partir desse objeto, crie um novo *data frame* com o número de ordens, famílias, gêneros e espécies de insetos por tribo de planta.

DICA: para calcular a riqueza, crie esta função, digitando o comando abaixo:

```
riqueza <- function(x) { length(na.omit(unique(x))) }
```

Detalhes sobre construção de funções estão no tópico "Noções de Programação em Linguagem S". Por ora basta saber que essa função conta quantos elementos diferentes há num vetor, excluindo os valores faltantes, e.g.:

```
> letras <- rep( c(letters[1:3],NA), each=2)
> letras
[1] "a" "a" "b" "b" "c" "c" NA  NA
> riqueza(letras)
[1] 3
```

Exercício 3.9. *Operações com Matrizes*

As matrizes de transição são uma maneira conveniente de modelar o crescimento de uma população dividida em faixas etárias, ou estágios de desenvolvimento. Para uma população de *Coryphanta robinsorum* (Cactaceae) no deserto do Arizona, divida em três estágios, a matriz de transição foi:

```
0,43    0    0,56
0,33  0,61     0
   0  0,30   0,96
```

Os elementos da matriz são as probabilidades de transição, num intervalo de tempo, do estágio correspondente ao número da coluna para o estágio correspondente ao número da linha. Por exemplo, a chance de um indivíduo passar do estágio 1 para o 2 é 0,33, e de permanecer em 1 é de 0,43.

1. Crie um objeto da classe matriz com esses valores. Isso permite realizar as operações matriciais a seguir.
2. Para calcular o número de indivíduos em cada estágio após um intervalo de tempo, basta multiplicar a matriz de transição pelas abundâncias dos indivíduos em cada estágio. Começando com 50 indivíduos do estágio 1, 25 do estágio 2 e 10 no estágio 3, qual será o número de plantas em cada estágio após três intervalos?

3. **Opcional**: A taxa de crescimento geométrico da população é o primeiro autovalor da matriz de transição, que pode ser calculado com a função `eigen`[7]. Se a taxa é maior que um a população está crescendo. É o caso dessa população?

O R como Ambiente de Operações Vetoriais

Na verdade, o R é muito mais que uma simples calculadora. O R é um **ambiente** onde podemos realizar operações vetoriais e matriciais.

Além das regras básicas para operações com vetores numéricos (ver Vetores: Operações Matemáticas), há operações aplicáveis a outros tipos de dados, e as importantíssimas **operações lógicas**, aplicáveis a qualquer classe.

Operações com Caracteres

Para vetores do tipo '`character`' operações matemáticas não fazem sentido e retornam uma mensagem de erro e o valor NA:

```
> mean( sp )
[1] NA
Warning message:
argument 'is' not numeric or logical: returning NA in: mean.default(sp)
>
```

Mas existem muitas operações que funcionam ou são próprias desse tipo de vetores:

```
> sort( sp ) # ordenação de caracteres em ordem crescente
[1] "Myrcia sulfiflora"      "Myrcia sulfiflora"      "Syagrus
romanzoffianus"
[4] "Tabebuia cassinoides"
>
> grep("Myrcia", sp) # busca por elementos do vetor de caracteres contendo
"Myrcia"
[1] 1 4
>
> table( sp ) # contagem do numero de elementos para cada classe de elemento
sp
     Myrcia sulfiflora Syagrus romanzoffianus   Tabebuia cassinoides
                     2                       1                       1
>
> sub("Myrcia", "M.", sp ) # substituição de caracteres
[1] "M. sulfiflora"          "Syagrus romanzoffianus" "Tabebuia cassinoides"
"M. sulfiflora"
>
```

```
> strsplit(sp, " ") # divisão de cada elemento do vetor por um dado
caractere ou símbolo (no caso, um espaço)
[[1]]
[1] "Myrcia"     "sulfiflora"

[[2]]
[1] "Syagrus"        "romanzoffianus"

[[3]]
[1] "Tabebuia"    "cassinoides"

[[4]]
[1] "Myrcia"     "sulfiflora"
>
```

Também é possível concatenar vetores de caracteres usando a função `paste`:

```
> bicho <- c("pato","gato","boi")
> cor <- c("branco","preto","vermeio")
> paste(bicho,cor)
[1] "pato branco" "gato preto"  "boi vermeio"
```

Operações Lógicas

Algumas operações são válidas para qualquer tipo de vetor. Essas operações envolvem comparações e são chamadas de operações lógicas:

```
> "Tabebuia cassinoides" == sp
[1] FALSE FALSE  TRUE FALSE
>
> a <= 7
[1]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
>
> b
[1] 1 2 3 4 5 6 7 8
> b >= 4
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
>
> c
 [1] 20 21 22 23 24 25 26 27 28 29 30 31 32
> (c %% 2) != 0
 [1] FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE
[13] FALSE
>
```

Como o R é um **ambiente vetorial** o resultado de operações lógicas também podem ser guardadas em vetores. Assim, surgem os vetores de classe `'logical'`:

```
> b
```

```
[1] 1 2 3 4 5 6 7 8
> f <- b <= 5
> f
[1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
> class( f )
[1] "logical"
>
```

Teste lógico para valores faltantes: função ''is.na()''

O indicador de valor faltante (“*missing values*) no R é NA, e o de valores não-numéricos (em geral resultantes de operações que não têm um valor definido) é NaN. A operação lógica para testar esses valores é feita com a função `is.na`, e não com os operadores == ou !=

```
> a <- seq(-100,100,50)
> a
[1] -100  -50    0   50  100
> b <- sqrt(a)
Warning message:
NaNs produzidos in: sqrt(a)
> b
[1]       NaN       NaN  0.000000  7.071068 10.000000
> b==NA
[1] NA NA NA NA NA
> b=="NA"
[1] FALSE FALSE FALSE FALSE FALSE
> is.na(b)
[1]  TRUE  TRUE FALSE FALSE FALSE
> !is.na(b)
[1] FALSE FALSE  TRUE  TRUE  TRUE
```

Operadores Lógicos no R [8)]

```
=========================================
 OPERADOR         DESCRIÇÃO
=========================================

   ==             igual
   !=             diferente
   >              maior
   <              menor
   >=             maior ou igual
   <=             menor ou igual
   &              e (and)
   |              ou (or)
   !              não
   is.na()        valor faltante ou
                  não numérico

=========================================
```

Sinais de Atribuição e de Igualdade

Já tratamos dos sinais de atribuição no item sobre criação de objetos da seção de Introdução ao R, onde vimos que um dos sinais de atribuição é um sinal de igual (**=**):

```
> a = log(2)
> a
[1] 0.6931472
>
```

Um **detalhe importantíssimo** é diferenciar um **sinal de igualdade** de um **sinal de atribuição**.

O sinal de igualdade faz uma comparação entre dois elementos. No R o sinal de igualdade são dois sinais de igual seguidos (**==**). Este operador retorna o resultado do teste lógico *"a igual b?"*, que só pode ter dois valores, T (verdadeiro), ou F (falso):

```
> a = 2 + 2
> a == 4
[1] TRUE
> a == 2+2
[1] TRUE
> a == 2
[1] FALSE
>
```

Uma maneira simples de quantificar frequências

Os vetores lógicos (`logical`) podem participar de operações matemáticas. Nesse caso o valor TRUE assume o valor 1, e valor FALSE assume o valor 0:

```
> f
[1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
> f * 7
[1] 7 7 7 7 7 0 0 0
>
> f * (1:length(f))
[1] 1 2 3 4 5 0 0 0
>
```

Para ter frequência de dados que satisfaçam uma certa condição basta somar o vetor lógico resultante:

```
> notas.dos.alunos
 [1]  6.0  5.1  6.8  2.8  6.1  9.0  4.3 10.4  6.0  7.9  8.9  6.8  9.8  4.6
11.3
[16]  8.0  6.7  4.5
##Quantos valores iguais ou maiores que cinco?
> sum(notas.dos.alunos>=5)
[1] 14
```

```
##Qual a proporção deste valor em relação ao total?
> sum(notas.dos.alunos>=5)/length(notas.dos.alunos)
[1] 0.7777778
```

Exercícios

Exercício 4.10. *Tabela de Cores*

Considere o seguinte vetor:

```
>  cores = c("amarelo","vermelho","azul","laranja")
>
```

Para gerar uma amostra, com reposição, dessas cores execute o comando:

```
> muitas.cores = sample(cores, 20, TRUE)
> muitas.cores
 [1] "amarelo"  "azul"     "amarelo"  "amarelo"  "vermelho" "laranja"
 [7] "laranja"  "azul"     "amarelo"  "vermelho" "amarelo"  "laranja"
[13] "azul"     "amarelo"  "amarelo"  "amarelo"  "amarelo"  "vermelho"
[19] "azul"     "laranja"
>
```

Como podemos obter uma tabela de freqüência das cores?

Exercício 4.11. *Vetor Normal*

Para gerar uma amostra de 1000 números de uma distribuição Normal com média 23 e desvio padrão 5, utilize o comando:

```
> vnormal = rnorm(1000, 23, 5)
```

Quantas observações no vetor '`vnormal`' são maiores que 28? E maiores que 33?

Exercício 4.12. *R Colors I*

A função '`colors()`' lista todas as cores que o R é capaz de gerar.

Quantas dessas cores são variantes da cor salmão (*salmon*)?

Quantas são variantes de verde?

Subconjuntos e Indexação

Frequentemente teremos que trabalhar não com um vetor inteiro, mas com um *subconjunto* dele. Para obter subconjuntos de um vetor temos que realizar operações de **indexação**, isto é, associar ao

vetor um outro vetor de mesmo tamanho com os **índices** dos elementos selecionados.

O **operador** de indexação é o colchetes `[]`, e um vetor pode ser indexado de três formas principais:

A) **Vetor de números inteiros positivos**: os números se referem às posições desejadas do vetor indexado.

```
> a
[1]  1.0000000 10.0000000  3.4000000  3.1415927  0.7853982  0.3678794
0.8020016
[8]  0.4338837
> a[ 2:4 ]                              # subconjuntos dos elementos nas
posições 2 a 4
[1] 10.000000  3.400000  3.141593
> b
[1] 1 2 3 4 5 6 7 8
> b[ c(2,5,8) ]                         # subconjuntos dos elementos nas
posições 2, 5 e 8
[1] 2 5 8
>
```

B) **Vetor de números inteiros negativos**: os números se referem às posições **não** desejadas do vetor indexado.

```
 a
[1]  1.0000000 10.0000000  3.4000000  3.1415927  0.7853982  0.3678794
0.8020016
[8]  0.4338837
> a[ -(2:4) ]                                          # Exclui as
posições de 2 a 4
[1] 1.0000000 0.7853982 0.3678794 0.8020016 0.4338837
> b
[1] 1 2 3 4 5 6 7 8
> b[ -c(2,5,8) ]                                       # Exclui as
posições de 2, 5 e 8
[1] 1 3 4 6 7
>
```

C) **Vetor lógico**: os elementos do vetor lógico correspondentes a TRUE são selecionados, os elementos correspondentes a FALSE são excluídos.

```
> b
[1] 1 2 3 4 5 6 7 8
> b[ c(TRUE, TRUE, FALSE, FALSE, FALSE, TRUE, TRUE) ]
[1] 1 2 6 7 8
> b[ b< 5 ]
[1] 1 2 3 4
> b[ b >= 7 ]
[1] 7 8
> b[ b == max(b) ]
[1] 8
```

```
> b[ b == min(b) ]
[1] 1
>
```

Na indexação por vetores lógicos, esses vetores devem ter **o mesmo comprimento** do vetor indexado. Caso contrário a operação será defeituosa:

```
> b
[1] 1 2 3 4 5 6 7 8
> b[ c(TRUE, TRUE) ]
[1] 1 2 3 4 5 6 7 8
> b[ c(FALSE, FALSE) ]
integer(0)
>
```

As operações por vetores lógicos podem combinar vários critérios, por meio dos operadores “E”, “OU” e “NÃO”:

Por questão de segurança do wiki contra spam algumas palavras são proibidas. Nos exemplos a seguir a palavra “estupro” em inglês foi substituída por “Abuso”, por esse motivo para rodar as linhas de código deve retornar a palavra para o idioma inglesa.

```
## Primeiras 5 linhas do data frame USArrests (crimes/1000 habitantes em
cada estado dos EUA, em 1973):
> USArrests[1:5,]
           Murder Assault UrbanPop Abusos
Alabama      13.2     236       58 21.2
Alaska       10.0     263       48 44.5
Arizona       8.1     294       80 31.0
Arkansas      8.8     190       50 19.5
California    9.0     276       91 40.6

##População Urbana dos estados em que a razão entre assassinatos e assaltos
foi maior que 20
> USArrests$UrbanPop[USArrests$Murder>USArrests$Assault/20]
 [1] 58 60 83 65 66 52 66 44 70 53 75 72 48 59 80 63 39

##Mesma condição acima, apenas estados com população menor do 55 milhões
> USArrests$UrbanPop[USArrests$Murder>USArrests$Assault/20 &
USArrests$UrbanPop<55]
[1] 52 44 53 48 39
```

D) **Vetor caracter**: nesse caso o vetor deve ser *nomeado* (função `names`) por um vetor `character`:

```
> zoo = c(4, 10, 2, 45)
```

```
> names(zoo) = c("onça", "anta", "tatu", "guará")
> zoo[ c("anta", "guará") ]
 anta guará
   10    45
> zoo[ grep("ç", names(zoo)) ]
onça
   4
>
```

Indexação de Fatores

A indexação de um fator pode resultar em níveis não usados. Caso você queira excluir esses níveis, use o argumento `drop`, do operador `[]`:

```
> tratamentos
 [1] Controle Controle Controle Controle Adubo A  Adubo A  Adubo A  Adubo A
 [9] Adubo B  Adubo B  Adubo B  Adubo B
Levels: Controle Adubo A Adubo B
> resposta
 [1] 1.8 3.0 0.9 1.7 2.4 2.7 2.6 1.5 3.0 2.7 0.8 3.0
> resp.sem.controle <- resposta[tratamentos!="Controle"]
> trat.sem.controle <- tratamentos[tratamentos!="Controle"]
> tapply(resp.sem.controle,trat.sem.controle,mean)
Controle  Adubo A  Adubo B
      NA    2.300    2.375
>
> ## Para eliminar níveis vazios do fator:
> trat.sem.controle <- tratamentos[tratamentos!="Controle",drop=T]
> tapply(resp.sem.controle,trat.sem.controle,mean)
Adubo A Adubo B
  2.300   2.375
```

Indexação de Matrizes e Data Frames

O modo de indexação de matrizes é `[linhas,colunas]`:

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

```
[2,]    2    5
```

A mesma notação é válida para *data frames*:

```
> USArrests[1:5,c(2,4)]
           Assault abuso
Alabama        236 21.2
Alaska         263 44.5
Arizona        294 31.0
Arkansas       190 19.5
California     276 40.6
>
> USArrests[1:5,c("Assault","Abuso")]
           Assault Abuso
Alabama        236 21.2
Alaska         263 44.5
Arizona        294 31.0
Arkansas       190 19.5
California     276 40.6
>
> USArrests[USArrests$UrbanPop>80,c("Assault","Abuso")]
              Assault Abuso
California        276 40.6
Hawaii             46 20.2
Illinois          249 24.0
Massachusetts     149 16.3
Nevada            252 46.0
New Jersey        159 18.8
```

Para incluir todas as linhas ou colunas, omita o valor (mas mantenha a vírgula!):

```
> matriz[,1]
[1] 1 2 3
> matriz[,c(1,4)]
     [,1] [,2]
[1,]    1   10
[2,]    2   11
[3,]    3   12
>
> USArrests[1:5,]
           Murder Assault UrbanPop Abuso
Alabama      13.2     236       58 21.2
Alaska       10.0     263       48 44.5
Arizona       8.1     294       80 31.0
Arkansas      8.8     190       50 19.5
California    9.0     276       91 40.6
>
> USArrests[grep("C",row.names(USArrests)),]
              Murder Assault UrbanPop Abuso
California       9.0     276       91 40.6
Colorado         7.9     204       78 38.7
```

```
Connecticut       3.3     110       77 11.1
North Carolina   13.0     337       45 16.1
South Carolina   14.4     279       48 22.5
>
```

A notação é estendida para um *array* de qualquer dimensão, como o objeto `Titanic`, que tem quatro dimensões:

```
> dimnames(Titanic)
$Class
[1] "1st"  "2nd"  "3rd"  "Crew"

$Sex
[1] "Male"   "Female"

$Age
[1] "Child" "Adult"

$Survived
[1] "No"  "Yes"

## Adultos sobreviventes
> Titanic[,,2,2]
      Sex
Class  Male Female
  1st    57    140
  2nd    14     80
  3rd    75     76
  Crew  192     20

## O mesmo, usando os nomes
> Titanic[,,"Adult","Yes"]
      Sex
Class  Male Female
  1st    57    140
  2nd    14     80
  3rd    75     76
  Crew  192     20
```

Usando Indexação para Alterar Valores

Combinando as operações de indexação e de atribuição é possível alterar os valores de qualquer parte de um objeto:

```
> zoo
 onça  anta  tatu guará
    4    10     2    45
> names(zoo)[4] <- "lobo-guará"
> zoo
```

```
      onça       anta       tatu lobo-guará
         4         10          2         45
>
> matriz
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> matriz[,c(1,4)] <- NA
> matriz
     [,1] [,2] [,3] [,4]
[1,]   NA    4    7   NA
[2,]   NA    5    8   NA
[3,]   NA    6    9   NA
> matriz[is.na(matriz)==T] <- 0
> matriz
     [,1] [,2] [,3] [,4]
[1,]    0    4    7    0
[2,]    0    5    8    0
[3,]    0    6    9    0
```

Ordenação por Indexação: Função ''order()''

Com a indexação é possível mudar a ordem de um vetor:

```
> zoo
      onça       anta       tatu lobo-guará
         4         10          2         45
>##Invertendo a ordem
> zoo[4:1]
lobo-guará       tatu       anta       onça
        45          2         10          4
##Uma ordem arbitrária
> zoo[c(3,4,2,1)]
      tatu lobo-guará       anta       onça
         2         45         10          4
```

A função `order` retorna os **índices** dos elementos de um vetor:

```
##Vetor de nomes do vetor zoo:
> names(zoo)
[1] "onça"       "anta"       "tatu"       "lobo-guará"

## O nome "anta", que tem o índice 2, é o primeiro na ordem alfabética:
> order(names(zoo))
[1] 2 4 1 3
```

E com isso podemos usar seu resultado para ordenar um vetor em função de quaisquer outros:

```
> zoo[order(names(zoo))]
      anta lobo-guará       onça       tatu
        10         45          4          2
```

O argumento da função comporta múltiplos vetores de critério. Em caso de empate pelo(s) primeiro(s) critério(s), os seguintes são usados:

```
> cidades
                 regiao estado pop.2007
Feira de Santana     NE     BA   544113
Salvador             NE     BA  2714119
São Paulo            SE     SP 11104712
Niterói              SE     RJ   476669
Nova Iguaçu          SE     RJ   858150
Recife               NE     PE  1528970
Santo André          SE     SP   676846
Rio de Janeiro       SE     RJ  6178762
Sorocaba             SE     SP   590846
Campinas             SE     SP  1073020
Osasco               SE     SP   724368
Guarulhos            SE     SP  1289047
Jaboatão             NE     PE   661901
>
>
cidades[order(cidades$regiao,cidades$estado,cidades$pop.2007,decreasing=T),]
                 regiao estado pop.2007
São Paulo            SE     SP 11104712
Guarulhos            SE     SP  1289047
Campinas             SE     SP  1073020
Osasco               SE     SP   724368
Santo André          SE     SP   676846
Sorocaba             SE     SP   590846
Rio de Janeiro       SE     RJ  6178762
Nova Iguaçu          SE     RJ   858150
Niterói              SE     RJ   476669
Recife               NE     PE  1528970
Jaboatão             NE     PE   661901
Salvador             NE     BA  2714119
Feira de Santana     NE     BA   544113
```

Prefira Ordenar por Indexação

A função `sort` é limitada porque ordena apenas o vetor ao qual é aplicada.

A função `order` permite uma ordenação mais flexível, pois pode usar critérios múltiplos, e os índices resultantes podem ser aplicados a qualquer objeto de igual comprimento.

Exercícios

Exercício 4.13. *Comando Curto, Resultado nem Tanto*

Verifique o resultado do comando:

```
> ?"["
```

Exercício 4.14 *Indexação de Listas*

Crie uma lista com o comando:

```
minha.lista <- list(um.vetor=1:5, uma.matriz=matrix(1:6,2,3),
                    um.dframe=data.frame(seculo=c("XIX","XX","XXI"),
                    inicio=c(1801,1901,2001)))
```

Qual a diferença entre os subconjuntos obtidos com os três comandos a seguir:

```
minha.lista[1]
minha.lista[[1]]
minha.lista$um.vetor
```

Exercício 4.15. *Vetor Normal II*

Para gerar uma amostra de 10.000 números de uma distribuição Normal com média 30 e desvio padrão 7, utilize o comando:

```
> vnormal = rnorm(10000, 30, 7)
```

Qual o somatório das observações no vetor '`vnormal`' que são maiores que 44? E maiores que 51?

Como você excluiria a *maior* observação do vetor '`vnormal`'?

Exercício 4.16. *R Colors II*

Das cores que o R pode gerar, quais são as cores variantes de salmão?

Quais são as variantes de rosa?

Exercício 4.17. *Modificação de Data Frame*

1. Ops! Há um erro no arquivo criado no exercício 4.3, no nome do *Diplodocus*. Como corrigir?
2. Os valores de massa cerebral das três espécies de dinossauros agora estão disponíveis, mas no objeto estão como valores faltantes (NA). Substitua-os, usando o operador de indexação.

Diplodocus	50
Triceratops	70
Brachiosaurus	154,5

Exercício 4.18. Aninhamento de comunidades

O termo “aninhamento” (*nesting*) é usado para a situação em que comunidades mais pobres em espécies são um subconjunto das comunidades mais ricas. Uma análise exploratória rápida de aninhamento é ordenar as linhas e as colunas de uma matriz binária de ocorrência das espécies por comunidades.

1. Crie um objeto da classe matriz com a matriz de ocorrência de mamíferos em topos de montanhas. (*DICA: a função `read.table` retorna um* data frame*. Use a função `as.matrix` para mudar a classe para matriz.*)
2. Use o ordenamento por indexação para criar uma matriz com as comunidades por ordem decrescente de espécies, e as espécies por ordem decrescente de frequência de ocorrência. (OUTRA DICA: lembre-se da função `apply`!).
3. A matriz resultante tem sinais de aninhamento? Por que?

1)
em termos técnicos: verifique se e como a função usada define um método para o objeto de dados usado
2)
em termos técnicos, trata-se do modo de armazenamento, e.g., apenas números, apenas caracteres, ou uma mistura, que é uma lista
3)
V= verdadeiro ou F = Falso
4)
em termos técnicos, dizemos que há métodos para cada classe de objeto, e que algumas funções têm métodos específicos para fatores, ou só têm para essa classe. Veja a seção sobre programação para detalhes
5)
em termos técnicos, a função table retorna um “array” (veja abaixo) com o mesmo número de dimensões que os fatores fornecidos como argumentos
6)
a notação [,1] significa “todas as linhas da coluna 1”, mais detalhes no item sobre indexação, mais abaixo
7)
consulte a ajuda para interpretar o resultados dessa função
8)
Incluímos aqui a função `is.na` para lembrar que para testar a ocorrência de valores faltantes ou não numéricos (Na e NaN) ela deve ser usada, e não os operadores == ou `!=`