Entrada e saída de dados no R

Fernando P. Mayer

2022-02-22

Introdução

A entrada de dados no R pode ser realizada de diferentes formas. O formato mais adequado vai depender do tamanho do conjunto de dados, e se os dados já existem em outro formato para serem importados ou se serão digitados diretamente no R.

A seguir são descritas as formas de entrada de dados com indicação de quando cada uma das formas deve ser usada. Os três primeiros casos são adequados para entrada de dados diretamente no R, os seguintes descrevem como importar dados já disponíveis

eletronicamentede um arquivo texto, em outro sistema ou no próprio R.

Posteriormente também será mostrado como fazer para exportar bases de dados geradas e/ou alteradas dentro do R.

Entrada de dados

Entrada de dados diretamente no R

Vetores

A forma mais básica de entrada de dados no R é através da função `c()` (como já vimos). A partir dela pode se criar os outros tipos de objetos como listas e data frames.

As funções básicas de entrada de dados são:

- `c()`
- `rep()`
- `seq()` ou `:`

A partir destas funções básicas podemos criar objetos de classes mais específicas com

- `matrix()`
- `list()`
- `data.frame()`

Entrada via teclado

Usando a função `scan()`

Esta função lê dados diretamento do console, isto é, coloca o R em modo *prompt* onde o usuário deve digitar cada dado seguido da tecla `Enter`. Para encerrar a entrada de dados basta digitar `Enter` duas vezes consecutivas.

Veja o seguinte resultado:

```
y <- scan()


1: 11
2: 24
3: 35
4: 29
5: 39
6: 47
7:
Read 6 items
```

```
y
```

```
# [1] 11 24 35 29 39 47
```

Os dados também podem ser digitados em sequência, desde que separados por um espaço,

```
y <- scan()
```

```
1: 11 24
3: 35 29
5: 39 47
7:
Read 6 items
```

```
y
```

```
# [1] 11 24 35 29 39 47
```

Este formato é mais ágil que o anterior (com `c()`, por exemplo) e é conveniente para digitar vetores longos. Esta função pode também ser usada para ler dados de um arquivo ou conexão, aceitando inclusive endereços de URLs (endereços da *web*) o que iremos mencionar em detalhes mais adiante.

Por padrão, a função `scan()` aceita apenas valores numéricos como entrada (lembre-se que vetores só podem ter elementos da mesma classe). Para alterar a classe de objeto de entrada, precisamos especificar o argumento `what` de `scan()`. Por exemplo, para entrar com um vetor de caracteres, fazemos

```
x <- scan(what = "character")
```

```
1: a
2: b
3: c
4:
Read 3 items
```

```
x
```

```
# [1] "a" "b" "c"
```

Outras classe possíveis para o argumento `what` são: `logical`, `integer`, `numeric`, `complex`, `character`, `raw` e `list`.

Exercícios 1

1. Usando a função `scan()` crie objetos para armazenar os seguintes valores:
    a. 19, 13, 19, 23, 18, 20, 25, 14, 20, 18, 22, 18, 23, 14, 19
    b. `joaquina`, `armação`, `praia brava`, `praia mole`, `morro das pedras`
    c. `TRUE`, `TRUE`, `FALSE`, `FALSE`, `TRUE`

Usando da função `readLines()`

Esta função é particularmente útil para ler entradas na forma de texto (*strings*). Por exemplo, para ler uma linha a ser digitada na tela do R, siga o comando abaixo e digite o texto indicado. Ao terminar pressione a tecla `Enter` e o texto será armazanado no objeto texto.

```
texto <- readLines(n = 1)


Estou digitando no console


texto

# [1] "Estou digitando no console"
```

Um possível uso é dentro que funções que solicitem que o usuário responda e/ou entre com informações na medida que são solicitadas. Experimente definir e rodar o função a seguir.

```
cor <- function() {
    cat("Digite sua cor favorita\n")
    cor <- readLines(n = 1)
    cat(paste("Sua cor favorita é", cor), "\n")
    return(invisible())
}


> cor()
Digite sua cor favorita
azul
Sua cor favorita é azul
```

Nesse exemplo, `readLines()` foi utilizada para efetuar a leitura via teclado, mas a função permite ainda entrada de dados por conexões com outros dispositivos de *input*. Por exemplo, pode ser utilizada para ler texto de um arquivo. Consulte a documentação da função para maiores detalhes e exemplos.

Entrada de dados em arquivos texto

Se os dados já estão disponíveis em formato eletrônico, isto é, já foram digitados em outro programa, voce pode importar os dados para o R sem a necessidade de digitá-los novamente.

A forma mais fácil de fazer isto é usar dados em formato texto (arquivo do tipo ASCII). Por exemplo, se seus dados estão disponíveis em uma planilha eletrônica como LibreOffice Calc, MS Excel ou similar, você pode escolher a opção `Salvar como...` e gravar os dados em um arquivo em formato texto. Os dois prncipais formatos de texto são:

- `txt` : arquivo de texto puro, onde as colunas são separadas geralmente por uma tabulação (`Tab`) ou espaço (`Spc`)
- `csv` : arquivo de texto, onde as colunas são geralmente separadas por vírgula (*comma separated value*), ou ponto-e-vírgula.

No R usa-se `scan()` mencionada anteriormente, ou então a função mais flexível `read.table()` para ler os dados de um arquivo texto e armazenar no formato de um data frame.

Antes de importar para o R:

- Se houverem valores perdidos, preencha com `NA`
- A matriz de dados deve formar um bloco só. Se houverem colunas de diferentes comprimentos, preencha com `NA`
- Salve o arquivo como "valores separados por vírgula" (`.csv`), mas atenção:
    - Se o separador de decimal for `,` , o separador de campos será `;` automaticamente (o que é mais comum nos sistemas em português).

A função `read.table()`

O método mais comum de importação de dados para o R, é utilizando a função `read.table()` . Como exemplo, baixe o arquivo `crabs.csv` disponível aqui (http://leg.ufpr.br/~fernandomayer/data/crabs.csv), e salve em um diretório chamado `dados` no seu diretório de trabalho.

Para importar um arquivo `.csv` faça:

```
dados <- read.table("dados/crabs.csv", header = TRUE,
                    sep = ";", dec = ",")
```

Argumentos:

- `"crabs.csv"` : nome do arquivo. (Considerando que o arquivo `crabs.csv` está dentro do diretório `dados`).
- `header = TRUE` : significa que a primeira linha do arquivo deve ser interpretada como os nomes das colunas
- `sep = ";"` : o separador de colunas (também pode ser `","` , `"\t"` para tabulação e `""` para espaços)
- `dec = ","` : o separador de decimais (também pode ser `"."`)

As funções `read.csv()` e `read.csv2()` são chamadas de *wrappers* (envelopes) que tornam o usa da função `read.table()` um pouco mais direta, alterando alguns argumentos. Por exemplo, o comando acima poderia ser substituído por

```
dados <- read.csv2("dados/crabs.csv")
```

O objeto criado com as funções `read.*()` sempre serão da classe `data.frame`, e quando houverem colunas com caracteres, estas colunas sempre serão da classe `character`. Você pode alterar esse padrão usando o argumento `stringAsFactors = TRUE`

```
dados2 <- read.csv2("dados/crabs.csv", stringsAsFactors = TRUE)
```

Para conferir a estrutura dos dados importados, usamos a função `str()` que serve para demonstrar a estrutura de um objeto, como o nome das colunas e suas classes:

```
str(dados)
```

```
# 'data.frame': 156 obs. of  7 variables:
#  $ especie: chr  "azul" "azul" "azul" "azul" ...
#  $ sexo   : chr  "M" "M" "M" "M" ...
#  $ FL     : num  8.1 8.8 9.2 9.6 10.8 11.6 11.8 12.3 12.6 12.8 ...
#  $ RW     : num  6.7 7.7 7.8 7.9 9 9.1 10.5 11 10 10.9 ...
#  $ CL     : num  16.1 18.1 19 20.1 23 24.5 25.2 26.8 27.7 27.4 ...
#  $ CW     : num  19 20.8 22.4 23.1 26.5 28.4 29.3 31.5 31.7 31.5
...
#  $ BD     : num  7 7.4 7.7 8.2 9.8 10.4 10.3 11.4 11.4 11 ...
```

```
str(dados2)
```

```
# 'data.frame': 156 obs. of  7 variables:
#  $ especie: Factor w/ 2 levels "azul","laranja": 1 1 1 1 1 1 1 1 1
1 ...
#  $ sexo   : Factor w/ 2 levels "F","M": 2 2 2 2 2 2 2 2 2 2 ...
#  $ FL     : num  8.1 8.8 9.2 9.6 10.8 11.6 11.8 12.3 12.6 12.8 ...
#  $ RW     : num  6.7 7.7 7.8 7.9 9 9.1 10.5 11 10 10.9 ...
#  $ CL     : num  16.1 18.1 19 20.1 23 24.5 25.2 26.8 27.7 27.4 ...
#  $ CW     : num  19 20.8 22.4 23.1 26.5 28.4 29.3 31.5 31.7 31.5
...
#  $ BD     : num  7 7.4 7.7 8.2 9.8 10.4 10.3 11.4 11.4 11 ...
```

Podemos também visualizar algumas linhas iniciais e finais do objeto importado através de duas funções auxiliares:

```
head(dados)
```

```
#   especie sexo   FL  RW   CL   CW   BD
# 1    azul    M  8.1 6.7 16.1 19.0  7.0
# 2    azul    M  8.8 7.7 18.1 20.8  7.4
# 3    azul    M  9.2 7.8 19.0 22.4  7.7
# 4    azul    M  9.6 7.9 20.1 23.1  8.2
# 5    azul    M 10.8 9.0 23.0 26.5  9.8
# 6    azul    M 11.6 9.1 24.5 28.4 10.4
```

```
tail(dados)
```

```
#     especie sexo   FL   RW   CL   CW   BD
# 151 laranja    F 21.3 18.4 43.8 48.4 20.0
# 152 laranja    F 21.4 18.0 41.2 46.2 18.7
# 153 laranja    F 21.7 17.1 41.7 47.2 19.6
# 154 laranja    F 21.9 17.2 42.6 47.4 19.5
# 155 laranja    F 22.5 17.2 43.0 48.7 19.8
# 156 laranja    F 23.1 20.2 46.2 52.5 21.1
```

Exercícios 2

1. Baixe os arquivos abaixo e coloque os arquivos em um local apropriado (de preferência no mesmo diretório de trabalho que voce definiu no início da sessão), faça a importação usando a função `read.table()`, e confira a estrutura dos dados com `str()`.
    a. prb0519.dat (http://leg.ufpr.br/~fernandomayer/data/BHH2/prb0519.dat)
    b. tab0303.dat (http://leg.ufpr.br/~fernandomayer/data/BHH2/tab0303.dat)
    c. tab1208.dat (http://leg.ufpr.br/~fernandomayer/data/BHH2/tab1208.dat)
    d. ReadMe.txt (http://leg.ufpr.br/~fernandomayer/data/BHH2/ReadMe.txt)
    e. montgomery_6-26.csv (http://leg.ufpr.br/~fernandomayer/data/montgomery_6-26.csv)
    f. montgomery_14-12.txt (http://leg.ufpr.br/~fernandomayer/data/montgomery_14-12.txt)
    g. montgomery_ex6-2.csv (http://leg.ufpr.br/~fernandomayer/data/montgomery_ex6-2.csv)
    h. ipea_habitacao.csv (http://www.leg.ufpr.br/~fernandomayer/data/ipea_habitacao.csv)
    i. stratford.csv (http://www.leg.ufpr.br/~fernandomayer/data/stratford.csv)

As funções permitem ainda ler dados diretamente disponíveis na *web*. Por exemplo, os dados do exemplo poderiam ser lidos diretamente com o comando a seguir, sem a necessidade de copiar primeiro os dados para algum local no computador do usuário:

```
dados <- read.csv2("http://www.leg.ufpr.br/~fernandomayer/data/crabs.
csv")
```

Para maiores informações consulte a documentação desta função com `?read.table()`. Embora `read.table()` seja provavelmente a função mais utilizada existem outras que podem ser úteis e determinadas situações:

- `read.fwf()` é conveniente para ler *fixed width formats*
- `read.fortran()` é semelhante à anterior porém usando o estilo Fortran de especificação das colunas
- `read.csv()`, `read.csv2()`, `read.delim()` e `read.delim2()`: estas funções são praticamente iguais a `read.table()` porém com diferentes opções padrão. Em geral (mas não sempre) dados em formato `csv` usado no Brasil são lidos diretamente com `read.csv2()`.

Exercícios 3

1. Faça a leitura dos dados do Exercício 2, mas agora utilize o endereço *web* dos arquivos.

Entrada de dados através da área de transferência

Um mecanismos comum para copiar dados de um programa para o outro é usando a **área de transferência** (ou *clipboard*). Tipicamente isto é feito com o mecanismo de copia-e-cola, ou seja-se, marca-se os dados desejados em algum aplicativo (editor, planilha, página web, etc), usa-se o mecanismo de COPIAR (opção no menu do programa que muitas vezes corresponde o teclar `Ctrl + c`), o que transfere os dados para a área de transferência. Funções como `scan()`, `read.table()` e outras podem ser usades apra ler os dados diretamente da área de transferência passando-se a opção `"clipboard"` ao primeiro argumento. Por exemplo, os seguintes dados:

```
ID  Grupo  Gasto  Ano
23      A  25,4   11
12      B  12,3   09
23      A  19,8   07
```

podem ser marcados e copiados para área de transferência e lidos diretamente com

```
dados.clip <- read.table("clipboard", header = TRUE, dec = ",")
```

```
str(dados.clip)
```

```
# 'data.frame': 3 obs. of  4 variables:
#  $ ID   : int  23 12 23
#  $ Grupo: chr  "A" "B" "A"
#  $ Gasto: num  25.4 12.3 19.8
#  $ Ano  : int  11 9 7
```

Importando dados diretamente de planilhas

Existem alguns pacotes disponíveis que podem ler dados diretamente de planilhas do MS Excel. No entanto, estes pacotes geralmente possuem particularidades quanto ao sistema operacional e demais dependências para funcionar corretamente.

Um destes pacotes, é o **gdata**, que funciona em diversos sistemas operacionais mas depende da linguagem Perl estar instalada. Por exemplo, para ler o conjuto de dados `crabs` armazenado em uma planilha do Excel (disponível aqui (http://leg.ufpr.br/~fernandomayer/data/crabs.xls)), podemos usar

```
## Carrega o pacote
library(gdata)
## Leitura diretamente do Excel
dados.xls <- read.xls("dados/crabs.xls", sheet = "Plan1",
                      header = TRUE, dec = ",")
## Estrutura
str(dados.xls)
```

```
# 'data.frame': 156 obs. of  7 variables:
#  $ especie: chr  "azul" "azul" "azul" "azul" ...
#  $ sexo   : chr  "M" "M" "M" "M" ...
#  $ FL     : num  8.1 8.8 9.2 9.6 10.8 11.6 11.8 12.3 12.6 12.8 ...
#  $ RW     : num  6.7 7.7 7.8 7.9 9 9.1 10.5 11 10 10.9 ...
#  $ CL     : num  16.1 18.1 19 20.1 23 24.5 25.2 26.8 27.7 27.4 ...
#  $ CW     : num  19 20.8 22.4 23.1 26.5 28.4 29.3 31.5 31.7 31.5
...
#  $ BD     : num  7 7.4 7.7 8.2 9.8 10.4 10.3 11.4 11.4 11 ...
```

Outros pacotes que possuem funções similares são: **openxlsx**, **xlsx**, e **XLConnect**.

Estruturas de dados mais complexas são tipicamente armazenadas nos chamados DBMS (*database management system*) ou RDBMS (*relational database management system*). Aguns exemplos são Oracle, Microsoft SQL server, MySQL, PostgreSQL, Microsoft Access, dentre outros. O R possui ferramentas implementadas em pacotes para acesso a estes sistemas gerenciadores.

Para mais detalhes consulte o manual R Data Import/Export (http://cran-r.c3sl.ufpr.br/doc/manuals/r-release/R-data.html) e a documentação dos pacotes que implementam tal funcionalidade. Alguns destes pacotes disponíveis são: **RODBC**, **DBI**, **RMySQL**, **RPostgreSQL**, **ROracle**, **RNetCDF**, **RSQLite**, dentre outros.

Carregando dados já disponíveis no R

O R já possui alguns conjuntos de dados que estão disponíeis logo após a instalação. Estes dados são também objetos que precisam ser carregados para ficarem disponiveis para o usuário. Normalmente, estes conjuntos de dados são para uso de exemplo de funções.

Para carregar conjuntos de dados que são disponibilizados com o R, use o comando `data()`. Por exemplo, abaixo mostramos como carregar o conjunto `mtcars` que está no pacote **datasets**.

```
## Objetos criados até o momento nesta seção
ls()
```

```
# [1] "dados"      "dados.clip" "dados.xls"  "dados2"     "texto"
# [6] "x"          "y"
```

```
## Carrega a base de dados mtcars
data(mtcars)
## Note como agora o objeto mtcars fica disponível na sua área de
## trabalho
ls()
```

```
# [1] "dados"      "dados.clip" "dados.xls"  "dados2"     "mtcars"
# [6] "texto"      "x"          "y"
```

```
## Estrutura e visualização do objeto
str(mtcars)
```

```
# 'data.frame': 32 obs. of  11 variables:
#  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
#  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
#  $ disp: num  160 160 108 258 360 ...
#  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
#  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
#  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
#  $ qsec: num  16.5 17 18.6 19.4 17 ...
#  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
#  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
#  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
#  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...

head(mtcars)

#                   mpg cyl disp  hp drat    wt  qsec vs am gear car
b
# Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4
4
# Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4
4
# Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4
1
# Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3
1
# Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3
2
# Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3
1
```

As bases de dados também possuem páginas de documentação para explicar o que são os dados e as colunas correspondentes. Para ver o que são os dados do `mtcars` por exemplo, veja `?mtcars`.

O conjunto `mtcars` é disponibilizado prontamente pois faz parte do pacote **datasets**, que por padrão é sempre carregado na inicialização do R. No entanto, existem outros conjuntos de dados, disponibilizados por outros pacotes, que precisam ser carregados para que os dados possam ser disponibilizados. Por exemplo, os dados do objeto `topo` são do pacote **MASS**. Se tentarmos fazer

```
data(topo)

# Warning in data(topo): data set 'topo' not found
```

Portanto, precisamos primeiro carregar o pacote **MASS** com

```
library(MASS)
```

e agora podemos carregar o objeto `topo` com

```
data(topo)
## O objeto fica disponível na sua área de trabalho
ls()

# [1] "dados"      "dados.clip" "dados.xls"  "dados2"     "mtcars"
# [6] "texto"      "topo"       "x"          "y"
```

```
## Confere a estrutura
str(topo)

# 'data.frame': 52 obs. of  3 variables:
#  $ x: num  0.3 1.4 2.4 3.6 5.7 1.6 2.9 3.4 3.4 4.8 ...
#  $ y: num  6.1 6.2 6.1 6.2 6.2 5.2 5.1 5.3 5.7 5.6 ...
#  $ z: int  870 793 755 690 800 800 730 728 710 780 ...
```

A função `data()` pode ainda ser usada para listar os conjutos de dados disponíveis.

```
data()
```

e também pode ser útil para listar os conjuntos de dados disponíveis para um pacote específico, por exemplo

```
data(package = "nlme")
```

Importando dados de outros programas

É possível ler dados diretamente de outros formatos que não seja texto (ASCII). Isto em geral é mais eficiente e requer menos memória do que converter para formato texto. Há funções para importar dados diretamente de EpiInfo, Minitab, S-PLUS, SAS, SPSS, Stata, Systat e Octave. Além disto é comum surgir a necessidade de importar dados de planilhas eletrônicas. Muitas funções que permitem a importação de dados de outros programas são implementadas no pacote **foreign**.

A seguir listamos algumas (não todas!) destas funções:

- `read.dbf()` para arquivos DBASE
- `read.epiinfo()` para arquivos .REC do Epi-Info
- `read.mtp()` para arquivos "Minitab Portable Worksheet"
- `read.S()` para arquivos do S-PLUS, e `restore.data()` para "dumps" do S-PLUS
- `read.spss()` para dados do SPSS
- `read.systat()` para dados do SYSTAT
- `read.dta()` para dados do STATA
- `read.octave()` para dados do OCTAVE (um clone do MATLAB)
- Para dados do SAS há ao menos duas alternativas:
  - O pacote **foreign** disponibiliza `read.xport()` para ler do formato TRANSPORT do SAS e `read.ssd()` pode escrever dados permanentes do SAS (`.ssd` ou `.sas7bdat`) no formato TRANSPORT, se o SAS estiver disponível no seu sistema e depois usa internamente `read.xport()` para ler os dados no R.
  - O pacote **Hmisc** disponibiliza `sas.get()` que também requer o SAS no sistema.

Para mais detalhes consulte a documentação de cada função e/ou o manual R Data Import/Export (http://cran-r.c3sl.ufpr.br/doc/manuals/r-release/R-data.html).

Saída de dados do R

Usando a função `write.table()`

Para exportar objetos do R, usamos a função `write.table()` , que possui argumentos parecidos com aqueles da função `read.table()` .

A função `write.table()` é capaz de criar um arquivo de texto no formato `txt` ou `csv` , com as especificações definidas pelos argumentos.

Para ilustrar o uso desta função, considerer o conjunto de dados `iris`

```
data(iris)
str(iris)

# 'data.frame': 150 obs. of  5 variables:
#  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
#  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
#  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
#  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
#  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1
1 1 1 1 1 1 1 ...
```

Podemos exportar esse data frame com

```
write.table(iris, file = "dados/iris.csv")
```

Por padrão, o arquivo resultante tem colunas separadas por espaço, o separador de decimal é ponto, e os nomes das linhas são também incluídos (o que geralmente é desnecessário). Para alterar essa configuração podemos fazer

```
write.table(iris, file = "dados/iris.csv", row.names = FALSE,
            sep = ";", dec = ",")
```

Os argumentos são

- `iris` : o nome do objeto a ser exportado (matriz ou data frame)
- `"iris.csv"` : nome do arquivo a ser gerado. (Considerando que o arquivo `iris.csv` será criado dentro do diretório `dados`).
- `row.names = FALSE` : para eliminar o nome das linhas do objeto (geralmente desnecessário), como retornado por `row.names()`
- `sep = ";"` : o separador de colunas (também pode ser `","` , `"\t"` para tabulação e `""` para espaços)
- `dec = ","` : o separador de decimais (também pode ser `"."`)

Note que o objeto a ser exportado (nesse caso `iris`) deve ser em formato tabular, ou seja, uma matriz ou data frame. Outras classes de objetos podem ser exportadas, mas haverá uma coerção para data frame, o que pode fazer com que o resultado final não seja o esperado.

Assim como `read.table()` possui as funções `read.csv()` e `read.csv2()` , a função `write.table()` possui as funções `write.table()` e `write.table2()` como *wrappers*. O comando acima também poderia ser executado como

```
write.csv2(iris, file = "dados/iris.csv", row.names = FALSE)
```

Note que `row.names = FALSE` ainda é necessário para eliminar os nomes das linhas.

O pacote **foreign** também possui funções para exportar para uma variedade de formatos. Veja a documentação em `help(package = "foreign")` . Os pacotes para ler dados diretamente de arquivos do MS Excel mencionados acima também possuem funções para exportar diretamente para esse formato.

Exercícios 4

1. Considere a tabela abaixo com o resultado de uma pesquisa que avaliou o número de fumates e não fumantes por sexo.

	Sexo	
Condição	**Masculino**	**Feminino**

Fumante	49	54
	64	61
	37	79
	52	64
	68	29
Não fumante	27	40
	58	39
	52	44
	41	34
	30	44

2. Digite estes dados em uma planilha eletrônica em um formato apropriado para um data frame do R, e salve em um arquivo `csv`.
3. Importe esse arquivo para o R com `read.table()`.
4. Crie uma nova coluna no objeto que contém estes dados, sendo a coluna com o número de pessoas multiplicada por 2.
5. Exporte esse novo objeto usando a função `write.table()`.
6. Tente criar esse mesmo conjunto de dados usando comandos do R (ex.: `c()`, `rep()`, `data.frame()`, etc.)

Usando os formatos textual e binário para ler/escrever dados

As formas mais comuns de entrada de dados no R são através da entrada direta pelo teclado (e.g. `c()` ou `scan()`), ou pela importação de arquivos de texto (e.g. `read.table()`). No etanto, ainda existem mais dois formatos par armazenar dados para leitura no R: o textual e o binário.

O **formato binário** é aquele armazenado em um arquivo binário, ou seja, um arquivo que contém apenas 0s e 1s, e possui um formato específico que só pode ser lido por determinado *software* ou função. É o oposto de um arquivo de texto, por exemplo, que podemos abrir e editar em qualquer prorama que edite texto puro.

O **formato textual** é o intermediário entre o texto puro e o binário. Os dados em formato textual são apresentados como texto puro, mas contém informações adicionais, chamados de **metadados**, que preservam toda a estrutura dos dados, como as classes de cada coluna de um data frame.

Formato textual

O formato textual é muito útil para compartilhar conjuntos de dados que não são muito grandes, e onde a formatação (leia-se: classes de objetos) precisa ser mantida.

Para criar um conjunto de dados no formato textual, usamos a função `dput()`. Vamos criar um data frame de exemplo e ver o resultado da chamada dessa função:

```
da <- data.frame(A = c(1, 2), B = c("a", "b"))
dput(da)

# structure(list(A = c(1, 2), B = c("a", "b")), class = "data.frame",
row.names = c(NA,
# -2L))
```

Note que o resultado de `dput()` é no formato do R, e preserva metadados como as classes do objeto e de cada coluna, e os nomes das linhas e colunas.

Outas classes de objetos são facilmente preservadas quando armazenadas com o resultado de `dput()`. Por exemplo, uma matriz:

```
ma <- matrix(1:9, ncol = 3)
dput(ma)

# structure(1:9, .Dim = c(3L, 3L))
```

E uma lista:

```
la <- list(da, ma)
dput(la)

# list(structure(list(A = c(1, 2), B = c("a", "b")), class = "data.fr
ame", row.names = c(NA,
# -2L)), structure(1:9, .Dim = c(3L, 3L)))
```

A saída da função `dput()` pode ser copiada para um script do R, para garantir que qualquer pessoa que venha usar o código (incluindo você no futuro), usará os dados no formato correto (esperado). Isso é muito importante para a **pesquisa reproduzível**!

A saída de `dput()` também pode ser salva diretamente em um arquivo de script do R, por exemplo,

```
dput(da, file = "da.R")
```

irá criar o arquivo `da.R` com o resultado da função. Para importar os dados salvos dessa forma, usamos a função `dget()`,

```
da2 <- dget(file = "da.R")
da2

#   A B
# 1 1 a
# 2 2 b
```

Múltiplos objetos podem ser armazenados em formato textual usando a função `dump()`.

```
dump(c("da", "ma", "la"), file = "dados.R")
```

Note que os objetos são passados como um vetor de caracteres, e um arquivo chamado `dados.R` é criado com todos os objetos no formato textual. Para importar estes objetos para uma sessão do R, usamos a função `source()`,

```
source("dados.R")
```

que já cria os objetos na sua área de trabalho com os mesmos nomes e atributos como foram armazenados.

Formato binário

Armazenar dados em formato binário é vantajoso quando não há uma forma "fácil" de armazenar os dados em formato de texto puro ou textual. Além disso, algumas vezes o formato binário possui maior eficiência em termos de velocidade de leitura/escrita, dependendo dos dados. Outra vantagem é que valores numéricos geralmente perdem precisão quando armazenados em texto ou textual, enquanto que o formato binário preserva toda a precisão (embora essa perda de precisão seja desprezível na maioria dos casos).

Para salvar um objeto contendo dados no R, usamos a função `save()`. Por exemplo, para armazenar o objeto `da` criado acima, fazemos

```
save(da, file = "dados.rda")
```

Esse comando irá criar o arquivo (binário) `dados.rda`. Note que a extensão `.rda` é comumente utilizada para dados binários do R, mas não é única.

Paa salvar mais de um objeto no mesmo arquivo, basta passar os nomes na mesma função

```
save(da, ma, file = "dados.rda")
```

A função `save.image()` pode ser utilizada se a intenção é salvar **todos** os objetos criados na sua área de trabalho (isso inclui qualquer objeto, não só os conjuntos de dados). Nesse caso, podemos fazer

```
save.image(file = "workspace.RData")
```

Note que quando foi utilizada a função `save()`, a extensão do arquivo foi `rda`, e com `save.image()` foi `RData`. Isso é uma convenção comum de arquivos binários do R, mas não é obrigatório. Qualquer uma das extensões funciona em ambas as funções.

Para carregar os conjuntos de dados (ou de forma mais geral, os objetos) armazenados em formato binário, usamos a função `load()`

```
load("dados.rda")
load("workspace.RData")
```

Dessa forma, os objetos já estarão disponíveis na sua área de trabalho.

Informações sobre diretórios e arquivos

O R possui uma variedade de funções para mostrar informações sobre arquivos e diretórios. Alguns exemplos são:

- `file.info()` mostra o tamanho do arquivo, data de criação, …
- `dir()` mostra todos os arquivos presentes em um diretório (tente com `recursive = TRUE`)
- `file.exists()` retorna `TRUE` ou `FALSE` para a presença de um arquivo
- `getwd()` e `setwd()` para verificar e altear o diretório de trabalho

Veja `?files` para uma lista copleta de funções úteis para manipular arquivos de dentro do R.