Introdução ao Software R

Marcos F Silva

01 de agosto 2016

Contents

CAPÍTULO 1 - INTRODUÇÃO

Este curso foi idealizado para ser uma introdução ao R para pessoas com pouca ou nenhuma experiência anterior com o aplicativo ou mesmo com análise de dados de forma geral.

Foi pensado tendo em mente o técnico que necessita realizar a análise de dados colhidos durante o trabalho de campo, na internet ou mesmo nos sistemas de informação dos TCs.

A abordagem escolhida para fazer esta introdução ao R foi com a utilização do contexto de auditoria governamental, visando especificamente a implementação de técnicas básicas de análise de dados denominadas na literatura de auditoria **`Técnicas de Auditoria Assistidas por Computador - TAAC`**.

Dentro do ciclo proposto pela metodologia CRISP-DM estas técnicas estão inseridas nas etapas de **compreensão dos dados** e **preparação dos dados**. Veja a figura a seguir:

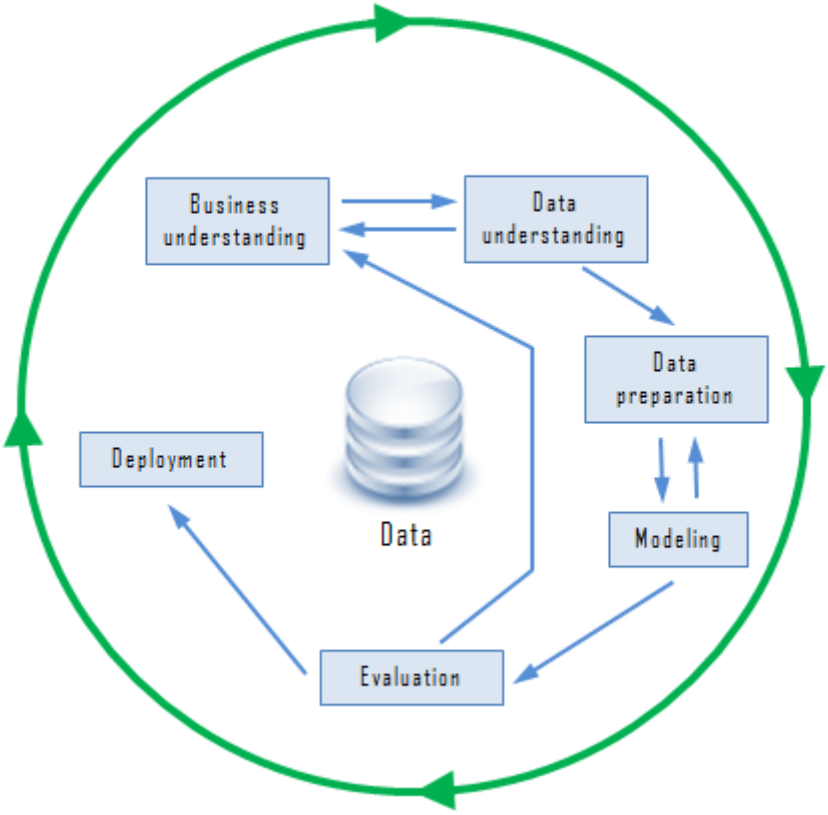


Figure 1:

Este curso atende, desta forma, a dois propósitos: ser uma introdução ao R e às técnicas de auditoria assistidas por computador, estando estruturado em 4 tópicos:

- Aspectos Básicos do R;
- Importação e exportação de dados;
- Produção de gráficos;

- Manipulação de dados;

No capítulo dedicado aos aspectos básicos do R, o objetivo é apresentar o conteúdo necessário a que o técnico tenha uma compreensão inicial de como o R funciona, onde poderá obter ajuda, como interagir com o ambiente, quais estrutruas de dados utiliza e quais as características dessas estruturas de dados.

Em razão das dificuldades normalmente experimentadas pelos usuários iniciantes durante a importação de arquivos de dados, optamos por dedicar um capítulo específico para essa tarefa no qual será visto como importar dados contidos em diversos formatos de arquivos usualmente encontrados na prática.

A elaboração de gráficos é um tópico usualmente não trabalhado na literatura sobre sobre TAAC a despeito de sua grande importância. Uma boa visualização de dados pode fornecer *insights* valiosos durante a exploração de um conjunto de dados que podem levar a identificação de anomalias, fraudes e erros.

Por fim temos a parte de manipulação de dados, onde será visto como implementar no R algumas das técnicas de análise de dados disponíveis em softwares de auditoria como ACL, IDEA, Arbutus, Lavastorm, ActiveData e TopCAATs.

A elaboração de modelos estatísticos, embora extremamente relevante, não será objeto deste curso. Acreditamos que este tópico deve ser objeto de curso específico.

R é usualmente descrito como um ambiente para análise de dados e produção de gráficos, possuindo também uma linguagem de programação. Neste curso, os aspectos do R relacionados à linguagem de programação não serão objeto de discussão.

1.1 - RStudio

Neste curso utilizaremos o RStudio para fazermos a interação com o R. Isto porque este aplicativo oferece um ambiente mais amigável e torna o uso do R muito mais produtivo.

Ao longo do curso os recursos deste aplicativo irão sendo apresentados à medida que se for fazendo necessário.

1.2 - Dados Utilizados

Estas notas de aula (`Notas_Aula_Curso_R.pdf`) e os conjuntos de dados utilizados (`Dados_Curso_R.zip`) podem ser baixados no link a seguir: `https://goo.gl/FaUG70`.

1.3 - Bibliografia Complementar

Existem disponíveis na internet uma grande quantidade de material para o aprendizado do R. Além da documentação oficial (`https://cran.r-project.org/manuals.html`) estão disponíveis no CRAN (*The Comprehensive R Archive Network*) diversos tutoriais em vários idiomas (`https://cran.r-project.org/other-docs.html`).

Uma compilação de tutoriais escritos em português pode ser obtida na página `http://goo.gl/wgmgFo`.

Outra fonte é a página: `http://www.ats.ucla.edu/stat/r/`.

CAPÍTULO 2 - ASPECTOS BÁSICOS DO R

2.1 - ONDE OBTER AJUDA

Algo com o qual o usuário do R deve se acostumar desde logo é consultar a ajuda das funções disponibilizadas pelo R. De fato, saber onde obter ajuda constitui uma das coisas mais importantes que se deve aprender a fazer.

O RStudio oferece diversas facilidades para acessar a documentação de funções. Uma delas é colocar o cursor sobre a função para a qual se deseja consultar a ajuda e pressionar a tecla `F1`. A ajuda para função aparecerá na aba `Help` do painel inferior direito. Nesta aba é possível pesquisar as funções cuja ajuda se deseja obter.

O Brasil possui uma lista de discussão bastante ativa chamada **R-Br** que pode ser acessada nos links a seguir:

- `http://r-br.2285057.n4.nabble.com/`
- `https://listas.inf.ufpr.br/cgi-bin/mailman/listinfo/r-br`

Além da lista de discussão nacional também existe a lista oficial do R que pode ser acessada a partir do *site* do R.

Stack Overflow também é uma importante fonte de informações. Consulte o link: `http://pt.stackoverflow.com/question`

2.2 - PRIMEIROS PASSOS

Nesta seção iniciaremos os primeiros passos rumo ao aprendizado do R.

2.2.1 Operações matemáticas

O uso mais elementar que se pode fazer do R é utilizá-lo como uma calculadora. Crie no RStudio um arquivo de *script* digite os seguintes comandos e submeta para execução:

```
12 + 48 - 178
```

```
[1] -118
```

```
3 * 7 + 10
```

```
[1] 31
```

```
45 / 7
```

```
[1] 6.428571
```

```
(4 + 6) / 2
```

```
[1] 5
```

```
2 ^ 3
```

```
[1] 8
```

As contas são feitas seguindo-se as regras de precedência usuais: potências, multiplicação e divisão, soma e subtração.

Os operadores matemáticos mais comuns são:

Operador	Significado
+	soma
-	subtração
*	multiplicação
/	divisão
^	exponenciação
%%	resto
%/%	divisão inteira

Operador de atribuição (<-)

Valores podem ser atribuídos a variáveis e estas podem ser utilizadas em cálculos. O símbolo `<-` é o operador de atribuição utilizado para atribuir valores a variáveis. No RStudio este operador pode ser criado utilizando-se a tecla de atalho **`Alt -`**.

O símbolo `=` também pode ser utilizado com esta finalidade embora não seja recomendado. Vejamos alguns exemplos:

```
# Atribui o valor 178 à variável 'num'
num <- 178

# Atribui à variável num seu valor anterior menos 100
num <- num - 100
num
```

```
[1] 78
```

```
# Outra forma de utilização do operador de atribuição
26 -> k
k
```

```
[1] 26
```

O simbolo `#` é utilizado pera inserir comentários nos *scripts*. O R irá ignorar tudo que venha depois desse símbolo

No R não é necessário definir com antecedência as variáveis a serem utilizadas como é feito em outras linguagens de programação.

2.2.2 Nomes de variáveis

Existem algumas restrições quanto aos nomes que podem ser utilizados para nomear variáveis. Não é possível criar variável que inicie por número ou por qualquer um dos seguintes caracteres:

> ^, !, $, @, +, -, *, /

Algumas palavras especiais utilizadas pelo R também não podem ser utilizadas como nomes de variáveis. São elas:

> `if`, `else`, `repeat`, `while`, `function`, `for`, `in`, `next`, `break`, `TRUE`, `FALSE`, `NULL`, `Inf`, `NaN`, `NA`, `NA_integer_`, `NA_real_`, `NA_complex_`, `NA_character_`.

Vamos testar algumas coisas:

```
8a <- 75
@_novo <- sqrt(49)
repeat <- 'repete'
```

Ao se escrever um *script* em R, ou em qualquer outra linguagem, é importante que se mantenha consistência na definição de nomes de varíaveis, nomes de funções, etc. Para ajudar neste quesito, existem alguns sites que dão boas dicas de como escolher bons nomes para variáveis, funções, etc.

Algumas destas dicas podem ser consultadas nos seguintes sites:

- http://google-styleguide.googlecode.com/svn/trunk/Rguide.xml
- http://r-pkgs.had.co.nz/style.html
- http://www.r-bloggers.com/consistent-naming-conventions-in-r/

Mas lembre-se: o que de fato importa é legibilidade do código. Assim, não tome estas orientações como verdade absoluta, mas como orientações gerais de como deixar o código mais legível.

2.2.3 Uso de funções

A maior parte do tempo nossa interação com o R será feita mediante a utilização de funções, as quais são utilizadas para se fazer tudo no R. Uma função recebe argumentos como *input* e devolvem algo ou realizam alguma tarefa como resultado do processamento dos *inputs* recebidos. Por exemplo, suponha que desejamos calcular a raiz quadrada de 146:

```
sqrt(146)
```

```
[1] 12.08305
```

No R as funções tem a seguinte estrutura: `nomefuncao(arg_1 = valor_1, arg_2 = valor_2, ..., arg_n = valor_n)`. Tem-se o nome da função e uma lista de argumentos e valores separadas por vírgula entre parênteses. Algumas funções podem não possuir argumentos mas deve-se sempre colocar os parênteses. Um exempo de função que não possui argumento é `getwd()` utilizada para se obter o diretório de trabalho.

Vejamos, por exemplo, a função `mean()` que retorna a média de um conjunto de números. Se consultarmos a ajuda desta função, veremos que ela é definida da seguinte forma: `mean(x, trim=0, na.rm=FALSE, ...)`.

A função tem 3 argumentos (*inputs*): `x`, `trim` e `na.rm`. Como pode ser visto, alguns argumentos já tem valores pré-definidos. No caso da função `mean()` os argumentos `trim` e `na.rm` já tem valores *default* atribuidos a eles que poderão ou não ser alterados pelo usuário.

O argumento `x` refere-se ao conjunto de dados para o qual se deseja calcular a média e o argumento `na.rm` informa à função se os valores faltantes, caso existam no conjunto de dados, devem ser excluídos. O argumento `trim` deve ser utilizado apenas se o usuário deseja calcular a média aparada.

Se os valores a serem fornecidos à função o forem na ordem em que os argumentos foram definidos, não há necessidade de escrever o nome do argumento, caso contrário sim.

Se passarmos os valores dos argumentos escrevendo os nomes dos mesmos não é necessário que os argumentos sejam passados à função na ordem em que foram definidos na função. Vejamos alguns exemplos para esclarecer o que foi dito.

Vamos criar um conjunto de dados para o qual desejamos calcular a média. Para isso, vamos utilizar a função `c()` que cria um vetor a partir dos valores inseridos como argumentos da função. (mais adiante explicaremos o que é um vetor):

```
# criar um pequeno conjunto de dados chamado dados
dados <- c(3, 5, 9, NA, 7, 16, 48)
dados
```

```
[1]  3  5  9 NA  7 16 48
```

Vamos agora usar a função `mean()` para calcular a média desses números. Para o caso em exame precisamos passar à função os valores para dois argumentos: `x` e `na.rm`. O argumento `trim` não precisa ter seu valor pré-definido modificado.

```
mean(x=dados, na.rm=TRUE)
```

```
[1] 14.66667
```

```
mean(na.rm=TRUE, x=dados)
```

```
[1] 14.66667
```

Como pode ser visto acima, nomeando-se os argumentos, estes podem ser passados à função em qualquer ordem. Sem nomeá-los devemos passá-los na ordem em que foram definidos na função. Exemplo:

```
mean(dados, , TRUE)
```

```
[1] 14.66667
```

Como não passamos valores para o argumento `trim` foi necessário deixar sua posição em branco. Mas, na prática, o que faríamos é:

```
mean(dados, na.rm=TRUE)
```

```
[1] 14.66667
```

Apenas relembrando: a lista de argumentos e valores são sempre separados por vírgula e deverão estar sempre entre de parênteses.

Mais alguns exemplos de uso de funções:

```
# Exibe o diretório de trabalho
getwd()
```

```
[1] "C:/Users/Marcos/Dropbox/1. Cursos ECG/Intro-R Treinamento TCE-MT/2.Rmd - Apostila"
```

```
# Exibe no console as variáveis existentes na área de trabalho
ls()
```

```
[1] "dados" "k"     "num"
```

```r
# Disponibiliza para uso o conjunto de dados interno do R chamado 'mtcars'
data(mtcars)

# Exibe os registros iniciais do conjunto de dados 'mtcars'
head(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
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
```

```r
# Retrna o número de observações existentes no conjunto de dados 'mtcars'
nrow(mtcars)
```

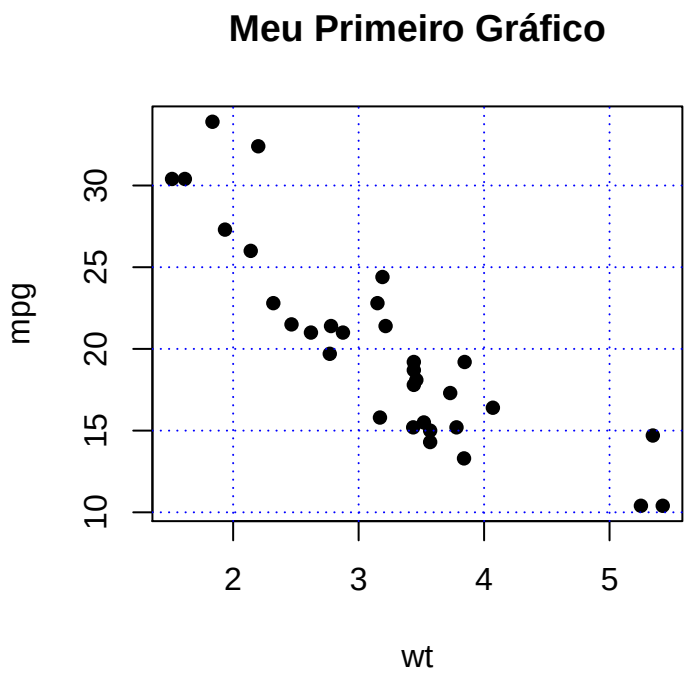
```
[1] 32
```

Mais adiante apresentaremos uma relação de funções para consulta.

2.2.4 Gráficos

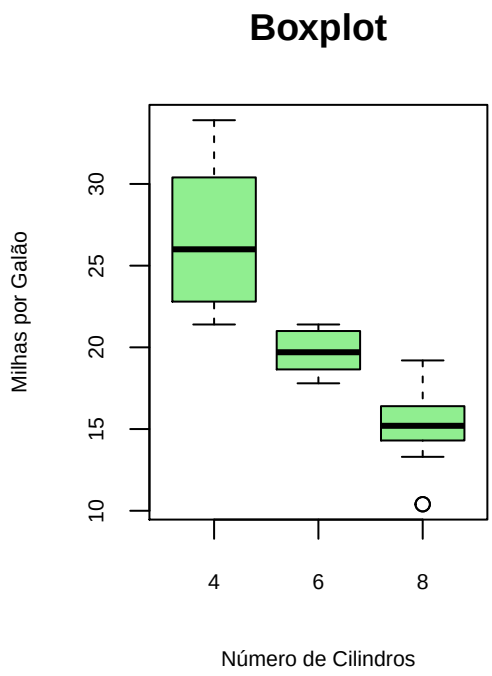
O R também permite a realização de gráficos de forma muito simples. Para ilustrar, apresentamos a seguir um diagrama de dispersão o qual é elaborado com a função `plot()`.

```r
plot(mpg ~ wt, data=mtcars, pch=16, main="Meu Primeiro Gráfico")
grid(col = 'blue') # Adiciona linhas de grade azuis ao gráfico
```



Com a função `boxplot()` podemos criar um boxplot da seguinte forma:

```
boxplot(mpg ~ cyl, data=mtcars,
        main='Boxplot',
        xlab='Número de Cilindros',
        ylab='Milhas por Galão',
        col='lightgreen',
        cex.lab=0.7, cex.axis=0.7)
```



No capítulo 3 estudaremos a construção de gráficos em maior detalhe.

2.2.5 Área de trabalho

A área de trabalho é o ambiente no qual as variáveis criadas durante o uso do R estarão armazenadas. Este ambiente é denominado `.GlobalEnv`.

Para consultarmos as variáveis existentes na área de trabalho utilizamos a função `ls()`

```
ls()
```

```
[1] "dados"  "k"      "mtcars" "num"
```

Para removermos uma ou mais variáveis da área de trabalho basta passar o nome da(s) variável(eis) a ser(em) removida(s) à função `rm()`. Exemplo:

```
rm(k, num)
ls()
```

```
[1] "dados"  "mtcars"
```

Para remover todas as variáveis da área de trabalho podemos utilizar o seguinte combinação de funções:

```
rm(list=ls())
```

Na aba `Environment` do painel superior direito do RStudio estarão evidenciados todos os objetos disponíveis na área de trabalho.

2.2.6 Pacotes

Dito de forma simples, pacotes são conjuntos de funções escritas para a realização de determinada tarefa. Quanto o R é instalado, ele já dispõe de um conjunto inicial de pacotes denominados *pacotes recomendados*. Outros pacotes podem ser adicionados pelo usuário a qualquer tempo.

Os pacotes estão disponíveis gratuitamente na internet, num repositório denominado CRAN (Compreensive R Archive Network) cujo endereço na internet é `https://cran.r-project.org/mirrors.html`. Além do CRAN, diversos pacotes estão hospedados em repositórios individuais no `GitHub`. Por exemplo o pacote `bookdown` está hospedado no seguinte repositório: `https://github.com/rstudio/bookdown`

Para instalar pacotes no R utiliza-se a função `install.packages()`.

Na aba `Packages` do painel inferior direito do RStudio o usuário poderá, além de consultar a relação dos pacotes já instalados em seu computador, instalar novos pacotes utilizando o botão `Install` ou atualizar os pacotes já instalados (botão `Update`).

Ao clicar sobre o nome do pacote o usuário é apresentado à página de ajuda do pacote, a qual contém a relação de todas as funções existentes no referido pacote. Clicando sobre a função, chaga-se à página de ajuda da função.

Para remover pacotes utiliza-se a função `remove.packages()`.

Todas as funções com as quais iremos trabalhar no R estarão armazenadas em pacotes. Como já dito anteriormente alguns pacotes já estão pré-instalados e alguns destes pacotes já são carregados automaticamente toda vez que o R é inicializado. Estes pacotes são: "stats", "graphics", "grDevices", "utils", "datasets", "methods", "base". As funções contidas nestes pacotes estão prontamente disponíveis para uso.

Por exemplo, o pacote `foreign` é um dos pacotes já pré-instalados mas suas funções não estão disponíveis para uso. Para que suas funções fiquem disponíveis é necessário "carregá-lo" para a memória do computador o que é feito utilizando-se a função `library()`:

```
library(foreign)
```

Agora as funções contidas neste pacote estão disponíveis para uso.

Um pacote é instalado apenas uma vez, mas para que seja possível utilizar suas funções será necessário carregá-lo todas as vezes que iniciarmos uma sessão no R.

2.3 - ESTRUTURAS DE DADOS

O R dispõe de algumas estruturas de dados nos quais estes podem ser armazenados. Estas estruturas são: *vetores*, *listas*, *data frames*, *matrizes* e *arrays*.

2.3.1 Vetores

Os vetores são a estrutura de dados mais simples do R. Podem ser construídos, além de outras, com as seguintes funções: `c()`, `rep()`, `seq()` ou com o operador `:`.

Em um vetor todos os valores devem ser do mesmo tipo, ou seja. todos os valores devem ser valores numéricos (`numeric`), caracteres (`character`), valores lógicos (`logical`) ou números complexos (`complex`).

Os vetores possuem 3 propriedades:

- seu **comprimento**, ou seja, a quantidade de elementos que ele possui;
- **tipo de dados** armazenados; e
- **atributos**, metadados adicionais que podem ser incluídos nos vetores como nomes.

Estes atributos podem ser consultados/obtidos com as seguintes funções: `length()` `typeof()` e `attributes()`. Vejamos alguns exemplos:

```
pessoas <- c('Alberto', 'Carlos', 'Francisco', 'José')
pessoas
```

```
[1] "Alberto"   "Carlos"    "Francisco" "José"
```

```
length(pessoas)
```

```
[1] 4
```

```
typeof(pessoas)
```

```
[1] "character"
```

```
attributes(pessoas)
```

```
NULL
```

Vamos atribuir nomes aos elementos do vetor acima e um comentário para o vetor:

```
pessoas2 <- c(nome1='Alberto', nome2='Carlos', nome3='Francisco', nome4='José')
comment(pessoas2) <- 'Turma 1'
attributes(pessoas2)
```

```
$names
[1] "nome1" "nome2" "nome3" "nome4"

$comment
[1] "Turma 1"
```

```r
names(pessoas2)
```

```
[1] "nome1" "nome2" "nome3" "nome4"
```

Outros vetores:

```r
x <- 20:1
x
```

```
 [1] 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

```r
typeof(x)
```

```
[1] "integer"
```

```r
y <- x + pi
typeof(y)
```

```
[1] "double"
```

```r
typeof(c(TRUE, FALSE, FALSE, TRUE))
```

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

Vamos tentar montar um vetor com dados de diferentes tipos:

```r
ff <- c('Maria', 8, TRUE)
ff
```

```
[1] "Maria" "8"     "TRUE"
```

```r
typeof(ff)
```

```
[1] "character"
```

Para acessar elementos de um vetor, podemos fazê-lo das seguintes formas:

1 - Fazendo referência à posição dos elementos no vetor (indexação por posição). Por exemplo:

```r
vec1 <- c(3, 5, 8, 7, 9, 16, 40)

## acessar o 2o elemento
vec1[2]
```

```
[1] 5
```

```
## acessar o 3o e 5o elementos
vec1[c(3, 5)]
```

```
[1] 8 9
```

```
## acessar o último elemento
vec1[length(vec1)]
```

```
[1] 40
```

```
## e se tentarmos o elemento 0 ou o elemento 10?
vec1[0]
```

```
numeric(0)
```

```
vec1[10]
```

```
[1] NA
```

Também é possível excluir elementos indicando os elementos a serem excluídos precedidos de um sinal de menos. Por exemplo, suponha que desejamos excluir o premeiro e o último elemento do vetor `vec1`:

```
vec1[-c(1, length(vec1))]
```

```
[1]  5  8  7  9 16
```

2 - Associando os valores lógicos `TRUE` ou `FALSE` aos elementos que devem permanecer ou sair do vetor, respectivamente. (indexação lógica). Exemplos:

```
vec1[c(TRUE, FALSE, FALSE, FALSE, FALSE, FALSE, TRUE)]
```

```
[1]  3 40
```

```
## Usando isso de forma mais interessante...
## Quais são elementos vec1 maiores que 10?
posicao <- vec1 > 10
posicao
```

```
[1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
```

```
vec1[posicao]
```

```
[1] 16 40
```

```
## De maneira mais abreviada...
vec1[vec1 > 10]
```

```
[1] 16 40
```

3 - Fazendo referência aos nomes dos elementos do vetor, caso os elementos estejam nomeados. (indexação por nomes)

```r
pessoas2[c('nome1', 'nome3')]
```

```
     nome1       nome3
  "Alberto" "Francisco"
```

Quando se utiliza a indexação por nomes não é possível utilizar a exclusão de valores utilizando o sinal de menos.

2.3.2 Vetorização

Uma característica importante do R é que ele opera de forma vetorizada, ou seja, as operações são realizadas em todos os elementos de um vetor simultaneamente.

```r
# Definição de um vetor
anos <- c(1800, 1850, 1900, 1950, 2000)

# Definição de outro vetor
carbono <- c(8, 54, 534, 1630, 6611)

# Realização de uma operação de subtração
anos - 1000
```

```
[1]  800  850  900  950 1000
```

```r
# Soma
anos + carbono
```

```
[1] 1808 1904 2434 3580 8611
```

O R utiliza a denominada **regra da reciclagem** que significa que em uma operação com dois vetores de tamanhos diferentes, o menor será *reciclado* até chegar ao tamanho do maior. A reciclagem é feita adicionando-se ao fim do menor vetor os valores iniciais deste mesmo vetor. Exemplo:

```r
a <- c(2, 3, 4, 5, 6)
b <- c(1, 2)

a + b
```

```
[1] 3 5 5 7 7
```

Nesta operação o vetor `b` foi reciclado para ficar da seguinte forma: `(1, 2, 1, 2, 1)` e então adicionado ao vetor `a`.

2.3.3 Matrizes

São estruturas de dados bidimensionais que, da mesma forma que os vetores, armazenam dados que sejam do mesmo tipo.

As matrizes são construídas com a função `matrix()`. Por exemplo, a matriz

$$\begin{bmatrix} 1 & 3 & 0 \\ 2 & 4 & -2 \end{bmatrix}$$

poderia ser construída no R da seguinte forma:

```
mat <- matrix(c(1, 2, 3, 4, 0, -2), ncol=3)
mat
```

```
     [,1] [,2] [,3]
[1,]    1    3    0
[2,]    2    4   -2
```

Por padrão, as células das matrizes são preenchidas por colunas.

Fica como tarefa verificar como fazer a função preencher a matriz por linhas.

Matrizes também podem ser criadas a partir de vetores, com a utilização das funções `cbind()` e `rbind()`. Os exemplos a seguir, ilustram esta possibilidade:

```
## rbind()
m1 <- rbind(c(1, 3, 0),
            c(2, 4, -2))
m1
```

```
     [,1] [,2] [,3]
[1,]    1    3    0
[2,]    2    4   -2
```

```
typeof(m1)
```

```
[1] "double"
```

```
class(m1)
```

```
[1] "matrix"
```

```
## cbind()
m2 <- cbind(c(1, 2), c(3, 4), c(0, -2))
m2
```

```
     [,1] [,2] [,3]
[1,]    1    3    0
[2,]    2    4   -2
```

Existem funções que permitem operações com matrizes (multiplicação de matrizes, cálculo do determinante, etc). Uma função muito útil é a função `t()` que realiza a transposição de uma matriz.

A função `class()` nos informa a **classe** de um objeto, não fornecendo informações sobre o tipo de dado, informação dada pela função `typeof()`.

Nota: A classe de um objeto define as operações que poderão ser realizadas nos mesmos bem como suas propriedades. A classe de um objeto tem relação com o paradigma de programação orientado a objetos.

```r
t(mat)
```

```
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    0   -2
```

Em essência, as matrizes são vetores com um atributo dimensão associado.

```r
x <- 1:10
class(x)
```

```
[1] "integer"
```

```r
attr(x,"dim") <- c(2, 5)
class(x)
```

```
[1] "matrix"
```

Para se acessar os elementos de uma matriz faz-se referência aos indices do item que se queira acessar. Por exemplo, supondo que na matriz `x` $2x5$ criada anteriormente, quiséssemos acessar o elemento que esteja na 2a linha e 3a coluna, faz-se:

```r
x[2, 3]
```

```
[1] 6
```

2.3.4 Array

Os *arrays* são estruturas semelhantes às matrizes mas podem possuir mais de duas dimensões. Em essência é um vetor multidimensional. A construção de *arrays* dá-se com a utilização da função `array()`. Exemplo:

```r
array(1:18,
      dim = c(3, 3, 2),
      dimnames = list(c('linha1', 'linha2', 'linha3'),
                      c('coluna1', 'coluna2', 'coluna3'),
                      c('planilha1', 'planilha2')))
```

```
, , planilha1

       coluna1 coluna2 coluna3
linha1       1       4       7
linha2       2       5       8
linha3       3       6       9

, , planilha2

       coluna1 coluna2 coluna3
linha1      10      13      16
linha2      11      14      17
linha3      12      15      18
```

Em teoria, os *arrays* podem ter qualquer número de dimensões, mas na prática trabalhar com *arrays* com mais de 3 dimensões é muito difícil.

2.3.5 Lista

A lista é a estrutura de dados mais flexível do R. Pode armazenar dados de tipos diferentes, contidos em qualquer uma das estruturas de dados do R.

Exemplos:

```
list(1, 2, 3)
```

```
[[1]]
[1] 1

[[2]]
[1] 2

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

```
list(c(1, 2, 3))
```

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

```
escola <- list(
  alunos = c('João', 'Maria', 'Ana'),
  idades = c(13, 14, 12),
  escola = 'Colégio ECG-TCERJ',
  notas =  array(c(7, 2, 9,   6, 4, 9,   6, 7, 8,  7, 5, 4,
                   8, 7, 8,   8, 7, 5,   9, 8, 8,  6, 4, 9,
                   6, 7, 5,   8, 7, 8,   4, 8, 7,  9, 6, 8),
                 dim = c(3, 4, 3),
                 dimnames = list(
                           c('João', 'Maria', 'Ana'),
                           c('Português', 'Matemática', 'Ciências', 'Artes'),
                           c('B1', 'B2', 'B3'))))


escola
```

```
$alunos
[1] "João"  "Maria" "Ana"

$idades
[1] 13 14 12

$escola
[1] "Colégio ECG-TCERJ"

$notas
, , B1

      Português Matemática Ciências Artes
João          7          6        6     7
```

```
Maria         2          4        7     5
Ana           9          9        8     4

, , B2

      Português Matemática Ciências Artes
João          8          8        9     6
Maria         7          7        8     4
Ana           8          5        8     9

, , B3

      Português Matemática Ciências Artes
João          6          8        4     9
Maria         7          7        8     6
Ana           5          8        7     8
```

Os componentes de uma lista podem ser acessados da seguinte forma:

```
## acesso pelo nome do componente...
escola[['alunos']]
```

```
[1] "João"  "Maria" "Ana"
```

```
escola['alunos']
```

```
$alunos
[1] "João"  "Maria" "Ana"
```

```
class(escola[['alunos']]) ## acessa o conteúdo do componente...
```

```
[1] "character"
```

```
class(escola['alunos'])   ## retorna o componente..
```

```
[1] "list"
```

```
## acesso pela posição do componente na lista...
escola[[1]]
```

```
[1] "João"  "Maria" "Ana"
```

```
## acesso usando o operador $
escola$idades
```

```
[1] 13 14 12
```

```
escola$notas[,,'B1'] ## acessa o primeiro bimestre...
```

```
      Português Matemática Ciências Artes
João          7          6        6     7
Maria         2          4        7     5
Ana           9          9        8     4
```

```
escola$notas['João', , ]
```

```
           B1 B2 B3
Português   7  8  6
Matemática  6  8  8
Ciências    6  9  4
Artes       7  6  9
```

2.3.5 Data frame

O `data frame` é uma lista especial onde cada componente é um vetor e todos tem o mesmo tamanho. As colunas do `data frame`podem ser vetores de quaisquer tipos.

Esta estrutura de dados é adequada para armazenar dados que se apresentem em uma estrutura retangular com linhas e colunas.

O `data frame` pode ser criado com a função `data.frame()`. As funções de importação de dados em geral retornam `data frame`.

Exemplo:

```
dados_alunos <- data.frame(alunos = c('João', 'Maria', 'Ana'),
                           idades = c(13, 14, 12),
                           sexo   = c('M', 'F', 'F'),
                           serie  = c('5o ano', '6o ano', '5o ano'),
                           stringsAsFactors = FALSE)

dados_alunos
```

```
  alunos idades sexo  serie
1   João     13    M 5o ano
2  Maria     14    F 6o ano
3    Ana     12    F 5o ano
```

```
class(dados_alunos)
```

```
[1] "data.frame"
```

```
str(dados_alunos)
```

```
'data.frame':   3 obs. of  4 variables:
 $ alunos: chr  "João" "Maria" "Ana"
 $ idades: num  13 14 12
 $ sexo  : chr  "M" "F" "F"
 $ serie : chr  "5o ano" "6o ano" "5o ano"
```

O acesso aos dados contidos em um data frame podem ser feitos de forma análoga a das matrizes. Por exemplo, para acessar o elemento que está na 3a linha e 2a coluna do data frame `dados_alunos`, faz-se da seguinte forma:

```
dados_alunos[3, 2]
```

```
[1] 12
```

Para acessar toda uma linha ou coluna, pode-se fazer:

```
dados_alunos[1,] # primeira linha
```

```
  alunos idades sexo  serie
1   João     13    M 5o ano
```

```
dados_alunos[,3] # terceira coluna
```

```
[1] "M" "F" "F"
```

```
dados_alunos[c(1, 2), 3] # primeira e segunda linha da terceira coluna
```

```
[1] "M" "F"
```

Como visto para os vetores, a indexação por nomes e lógica também são aplicáveis. Exemplo:

```
# indexação lógica
dados_alunos[dados_alunos$sexo == 'F',]
```

```
  alunos idades sexo  serie
2  Maria     14    F 6o ano
3    Ana     12    F 5o ano
```

```
dados_alunos[, c(TRUE, FALSE, TRUE, FALSE)]
```

```
  alunos sexo
1   João    M
2  Maria    F
3    Ana    F
```

```
dados_alunos
```

```
  alunos idades sexo  serie
1   João     13    M 5o ano
2  Maria     14    F 6o ano
3    Ana     12    F 5o ano
```

```r
# indexação por nomes
dados_alunos[c(1, 3), c('alunos', 'serie')]
```

```
  alunos  serie
1   João 5o ano
3    Ana 5o ano
```

2.4 - TIPOS DE DADOS

Os dados utilizados pelo R podem ser dos seguintes tipos: `numeric`, `logical`, `character`, `complex` e `raw`. O tipo numérico pode ser inteiro (`integer`) ou ponto flutuante (`double`). A função `typeof()` nos informam o tipo do dado.

Exemplos:

```r
typeof(mat)
```

```
[1] "double"
```

```r
typeof(c(8L, 76L, 49L, 26L))
```

```
[1] "integer"
```

```r
typeof(c(8, 76, 49, 26))
```

```
[1] "double"
```

```r
sexo <-  factor(c('Masculino', 'Feminino', 'Feminino', 'Masculino', 'Masculino'))
typeof(sexo)
```

```
[1] "integer"
```

```r
class(sexo)
```

```
[1] "factor"
```

Alguns tipos de dados podem ser convertidos para um outro tipo de dados. Por exemplo, um vetor numérico pode ser convertido para caracteres, um vetor lógico para numérico. Esta conversão é feita utilizando as seguintes funções: `as.character()`, `as.numeric()`, `as.logic()`, `as.complex()`, etc.

Exemplo:

```r
## conversão de número para caractere
num <- c(5, 10, 20, 40.18)
num
```

```
[1]  5.00 10.00 20.00 40.18
```

```r
as.character(num)
```

```
[1] "5"     "10"    "20"    "40.18"
```

```r
## conversão de lógigo para numérico
logico <- c(TRUE, FALSE, FALSE, TRUE, TRUE)
logico
```

```
[1]  TRUE FALSE FALSE  TRUE  TRUE
```

```r
as.numeric(logico)
```

```
[1] 1 0 0 1 1
```

Para uma relação completa das funções de conversão digite `apropos('^as\\.')` no console.

2.5 - FUNÇÕES DIVERSAS E OPERADORES

O R dispõe de um amplo conjunto de funções matemáticas, estatísticas e outras. Algumas destas funções estão elencadas a seguir:

2.5.1 Funções Matemáticas

Função	Descrição
`abs()`	calcula o valor absoluto
`sqrt()`	calcula a raiz quadrada
`exp()`	calcula o valor da função exponencial
`log()`	calcula o logaritmo
`log10()`	calcula o logaritmo na base 10
`log2()`	calcula o logaritmo na base 2
`cos()`	calcula o cosseno
`sin()`	calcula o seno
`tan()`	calcula a tangente
`sum()`	calcula a soma dos elementos de um vetor
`prod()`	calcula o produto dos elementos de um vetor

Exemplos:

```r
abs(3 * 5 - 55)
```

```
[1] 40
```

```r
sqrt(144)
```

```
[1] 12
```

```r
log(sqrt(49))
```

```
[1] 1.94591
```

```r
round(sin(pi),0)
```

```
[1] 0
```

```r
round(cos(pi),0)
```

```
[1] -1
```

```r
prod(c(2, 5, 2, 3))
```

```
[1] 60
```

```r
sum(c(2, 5, 2, 3))
```

```
[1] 12
```

2.5.2 Funções Estatísticas

Função	Descrição
`range()`	retorna os valores mínimo e máximo
`mean()`	retorna a média
`median()`	retorna a mediana
`min()`	retorna o valor mínimo
`max()`	retorna o valor máximo
`var()`	retorna a variância
`sd()`	retorna o desvio padrão
`quantile()`	retorna separatrizes (quartis, decis, percentis, etc.)
`IQR()`	retorna o intervalo interquartil
`summary()`	retorna um conjunto de estatísticas descritivas

Exemplos:

```r
# Cria um vetor e números
idades <- c(27, 45, 36, 19, 32, 55, 26, 31, 65, 27, 46)
min(idades)
```

```
[1] 19
```

```r
max(idades)
```

```
[1] 65
```

```r
range(idades)
```

```
[1] 19 65
```

```r
mean(idades)
```

```
[1] 37.18182
```

```r
summary(idades)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  19.00   27.00   32.00   37.18   45.50   65.00
```

```r
quantile(idades, probs = 0.75)
```

```
 75%
45.5
```

```r
quantile(idades, probs = seq(0, 1, 0.1))
```

```
  0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
  19   26   27   27   31   32   36   45   46   55   65
```

2.5.3 Funções para Operações com Conjuntos

Função	Descrição
`union(x, y)`	união dos elementos contidos objetos `x` e `y`
`intersect(x, y)`	interseção dos elementos contidos em `x`e `y`
`setdiff(x, y)`	elementos que estão em `x` e não estão em `y`
`setequal(x, y)`	retorna verdadeiro se `x` e `y` são iguais
`is.element(el, set)`	retorna verdadeiro se um elemento `el` pertence ao conjunto `set`

Exemplos:

```r
# Cria dois vetores de caracteres
v1 <- c('João', 'Carlos', 'Leandro', 'Pedro', 'Francisco', 'Joana', 'Karine', 'Renata')
v2 <- c('Josyanne', 'Renata', 'Pedro', 'Talita', 'Aline', 'Mariana')
```

```r
union(v1, v2)
```

```
 [1] "João"      "Carlos"    "Leandro"   "Pedro"     "Francisco"
 [6] "Joana"     "Karine"    "Renata"    "Josyanne"  "Talita"
[11] "Aline"     "Mariana"
```

```r
intersect(v1, v2)
```

```
[1] "Pedro"  "Renata"
```

```r
is.element(c('Leandro', 'Marcos'), v1)
```

```
[1]  TRUE FALSE
```

```r
is.element('Leandro', v2)
```

```
[1] FALSE
```

```r
setequal(v1, v2)
```

```
[1] FALSE
```

```r
setequal(c(5, 12, 17), c(17, 5, 12))
```

```
[1] TRUE
```

2.5.4 Funções Diversas

Função	Descrição
`floor()`	retorna o inteiro imediatamente inferior
`ceiling()`	retorna o inteiro imediatamente superior
`trunc()`	retora a parte não fracionária do número
`round()`	arredonda para o número de casas decimais especificada
`signif()`	arredonda para o número de dígitos significativos especificado
`cumsum()`	soma acumulada dos valores
`cumprod()`	proudto acumulado dos valores
`cummax()`	acumula preservando os valores máximos
`cummin()`	acumula preservando os valores mínimos
`diff()`	retorna a diferença entre valores
`sign()`	retorna um vetor contendo o sinal dos elementos de um vetor
`sort()`	ordena os elementos do objeto fornecido como argumento

Exemplos:

```r
tt <- c(2, 5, 7, 9, 3, 4, 4)
cumsum(tt)
```

```
[1]  2  7 14 23 26 30 34
```

```r
cummax(tt)
```

```
[1] 2 5 7 9 9 9 9
```

```r
cummin(tt)
```

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

```r
diff(tt)
```

```
[1]  3  2  2 -6  1  0
```

```r
sort(tt, decreasing = TRUE)
```

```
[1] 9 7 5 4 4 3 2
```

2.5.5 Operadores para Comparação

Operador	Significado
>	maior que
<	menor que
>=	maior ou igual
<=	menor ou igual
!=	diferente
==	igual

Exemplos:

```r
'A' < 'B'
```

```
[1] TRUE
```

```r
'Z' > 'Q'
```

```
[1] TRUE
```

```r
c(4, 7, 6, 12) == c(2, 7, 4, 8)
```

```
[1] FALSE  TRUE FALSE FALSE
```

2.5.6 Operadores Lógicos

Operador	Significado
`&`	e
`|`	ou
`!`	negação
`xor()`	retorna TRUE se exatamente um valor é verdadeiro
`any()`	retrona TRUE se ao menos um valor é verdadeiro
`all()`	retorna TRUE se todos os valores são verdadeiros
`isTRUE()`	testa se é `TRUE` (não vetorizado)

Exemplos:

```
7 > 5 & 12 < pi
```

```
[1] FALSE
```

```
1:5 > 2
```

```
[1] FALSE FALSE  TRUE  TRUE  TRUE
```

```
!(1:5 > 2)
```

```
[1]  TRUE  TRUE FALSE FALSE FALSE
```

```
isTRUE(1:5 > 2)
```

```
[1] FALSE
```

```
isTRUE(5 > 2)
```

```
[1] TRUE
```

Para mais informações consultar a ajuda para `Syntax`, `Arithmetic` e `Logic`

Outras funções de interesse são: `is.factor()`, `is.character()`, `is.numeric()`, etc. Estas funções testam um objeto para verificar se os valores são, respectivamente, fatores, caracteres ou números. Para uma lista completa deste tipo de função digite `apropos('^is\\.')` no console.

2.6 - VALORES ESPECIAS

O R possui alguns valores que são considerados especiais. São eles: `NULL`, `NA`, `Inf` e `NaN`.

2.6.1 `NA`

`NA` (*not available*) é o valor utilizado pelo R para indicar que um valor está faltando.

Representa um valor *missing*. Para testar a existência de `NA` utiliza-se a função `is.na()`.

É um erro muito comum tentar testar a existência de `NA` utilizando o operador lógico `==`. Veja o exemplo a seguir:

```r
z <- c(1:3,NA)
is.na(z)
```

```
[1] FALSE FALSE FALSE  TRUE
```

```r
z == NA    # Não funciona
```

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

```r
z[z == NA] # Não funciona
```

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

2.6.2 NaN

`NaN` (*not a number*) é um tipo de valor faltante que se origina na operação com valores numéricos. A função `is.na()` também retorna `TRUE` para `NaN` ao passo que `is.nan()` testa apenas para `NaN`.

```r
# Exemplo
0/0
```

```
[1] NaN
```

```r
Inf - Inf
```

```
[1] NaN
```

```r
log(-3)
```

```
[1] NaN
```

2.6.3 NULL

`NULL` representa o objeto nulo. Pode ser usado para remover componentes em listas ou colunas em dataframes.

`Inf` e `-Inf` representam infinito.

```r
# Exemplo
x <- list("a", 1:5, c("p", "q", "r"))
x[[2]] <- NULL # Remove o segundo componente da lista.
x
```

```
[[1]]
[1] "a"

[[2]]
[1] "p" "q" "r"
```

```
5/0
```

```
[1] Inf
```

2.7 - EXERCÍCIOS

2.7.1 - Execute as seguintes tarefas:

(a) Calcule a raiz quadrada de 729.

(b) Obtenha o resto da divisão de 150 por 4.

(c) Crie uma variável `b` contendo o valor 1947.

(d) Converta a variável `b` para o tipo caractere.

(e) Defina o diretório de trabalho para `C:\Temp`

(f) Crie um vetor numérico contendo os valores de 1 a 6 e mostre a sua classe.

(g) Inicialize um vetor de caracteres de tamanho 26.

(h) Exiba todos os objetos criados na área de trabalho.

(i) Remova, de uma única vez, todos os objetos da área de trabalho.

2.7.2 - Considere o vetor `x`, definido como `x <- c(5,9,2,3,4,6,7,0,8,12,2,9)`. Verifique se você compreendeu como funciona o mecanismo de *subseting* em vetores tentando antecipar quais os resultados dos comandos a seguir.

```
x[2]
x[2:4]
x[c(2,3,6)]
x[c(1:5,10:12)]
x[-(10:12)]
```

Use o R para conferir suas respostas.

2.7.3 - Dadas as matrizes `X` e `Y` definidas a seguir:

$$X = \begin{bmatrix} 3 & 2 \\ 1 & 1 \end{bmatrix} \quad Y = \begin{bmatrix} 1 & 4 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Verifique se você compreendeu corretamente o mecanismo de *subseting* em matrizes tentando antecipara o resultado das seguintes operações:

```
X[1,]
X[2,]
X[,2]
Y[1,2]
Y[,2:3]
```

Use o R para conferir suas respostas.

2.7.4 - Obtenha as seguintes somas:

$\sum_{i=1}^{100}(i^3 + 4i^2)$ e

$\sum_{i=1}^{25}(\frac{2^i}{i} + \frac{3^i}{i^2})$

2.7.5 - Utilizando a função `c()` e o operador `:`, obtenha a seguinte sequência: `1,2,3,...,19,20,19,18,...,2,1`

2.7.6 - Considere o vetor `x` definido a seguir:

```
set.seed(50)
x <- sample(0:999, 250, replace=TRUE)
```

(a) • Selecione os valores de `x` > 600

(b) • Selecione o último elemento do vetor

(c) • Obtenha os elementos de `x` que estejam nas posições 1, 4, 12 e 17

2.7.7 - A função `rep()` é utilizada para replicar elementos de vetores. Por exemplo:

```
rep(c('A', 'B', 'C'), c(3, 3, 3))
```

```
[1] "A" "A" "A" "B" "B" "B" "C" "C" "C"
```

```
rep(1:4, c(2,3,4,5))
```

```
 [1] 1 1 2 2 2 3 3 3 3 4 4 4 4 4
```

Como poderia ser gerada a sequência `4,6,3, 4,6,3,...,4,6,3` onde cada um dos dígitos `4`, `6`, `3` ocorrem 10 vezes

2.7.8 - No exemplo da escola, como seria possível obter, para cada aluno e matéria, a soma das notas nos 3 bimestres?

CAPÍTULO 3 - ENTRADA E SAÍDA DE DADOS

3.1 - TIPOS DE ARQUIVOS DE DADOS

Antes de ilustrar como os dados podem ser importados pelo R, convém fazer algumas considerações sobre os tipos de arquivos nos quais os dados (numéricos e textuais) são armazenados, já que as funções utilizadas para realizar a importação desses dados dependerão desse fato.

Os arquivos de dados mais comumente encontrados na prática são os seguintes:

- arquivos texto com valores separados por delimitadores (ex. `.csv`);
- arquivos texto com formato fixo;

- arquivos texto com com marcação de tags (`.html`, `.xml`, `.kml`);

- arquivos texto *JavaScript Object Notation* (`.json`);
- arquivos do excel (`.xls`, `.xlsx`);
- arquivos de bancos de dados (Access, SQLite, etc.);

- arquivos de aplicativos estatísticos (Stata, SPSS, etc.).

3.2 - IMPORTAÇÃO E EXPORTAÇÃO DE DADOS

Neste tópico serão vistas as funções disponíveis para a importação e exportação de dados.

Mostraremos como importar os tipos mais comuns de arquivos. Arquivos `.json`, `.html`, `.xml` e `kml` não serão tratados neste curso.

3.2.1 Arquivos texto com delimitadores

Estes arquivos tem como característica a fato de que os dados são separados por um caractere delimitador, usualmente a vírgula (`,`), o ponto e vírgula (`;`), o caractere `\t` que representa a tabulação e o pipe `|`.

A principal função nativa do R para a importação de arquivos com esta estrutura á a função `read.table()`. Esta função tem uma grande quantidade de parâmetros com valores *default* que devem ser modificados pelo usuário dependendo da estrutura do arquivo de dados a ser importado.

Além dessa função, existem as funções `read.csv()`, `read.csv2()` e `read.delim()` que são versões da função `read.table()` com alterações no valor `default` de alguns parâmetros.

Para ilustrar o uso da função `read.table()` utilizaremos o arquivo `Receita_Municipios_RJ_2013.txt` que é um arquivo cujo delimitador é o caractere `\t` (tabulação). Sua importação pode ser feita da seguinte forma:

```
# Definição do diretório de trabalho
diretorio <- "C:\\Users\\Marcos\\Dropbox\\1. Cursos ECG\\Intro-R Treinamento TCE-MT\\3.dados"
setwd(diretorio)

# Importação dos dados
receitas <- read.table('Receita_Municipios_RJ_2013.txt',
                       sep='\t', header = TRUE,  comment.char = '')
head(receitas[,c(1, 2, 4)])
```

```
             Munic.pio C.d.Item.Receita.TCE Valor.Arrecadado
1              CORDEIRO             17219900        222697.81
2             PINHEIRAL             97220104        -74109.74
3 SANTA MARIA MADALENA              24710100         94849.59
4             VARRE-SAI             97210102      -1035237.11
5         NOVA FRIBURGO             13281000       1067541.46
6     SAO JOAO DA BARRA             97210105        -96862.76
```

Vamos apresentar a seguir uma tabela com os principais argumentos desta função e os valores a serem passados à mesma:

Argumento	Valor
`file`	string contendo o caminho até o arquivo a ser importado
`header`	informa à função se a primeira linha do arquivo contém os nomes das colunas. O valor *default* é `FA`

Argumento	Valor
`sep`	informa à função o caractere a ser usado como delimitador. O valor `default` e `" "`
`quote`	informa à função o caractere utilizado para delimitar strings que contenham valores especiais.
`dec`	informa à função o caractere separador de decimal
`row.names`	informa à função qual variável deve ser considerada para dar nome aos registros
`col.names`	passa á função os nomes das variáveis
`as.is`	informa à função para realizar a importação dos dados no formato que estão
`na.strings`	informa à função os caracteres que devem ser considerados valores faltantes
`colClasses`	informa à função as classes das variáveis a serem importadsas
`nrows`	informa a quantidade de linhas a serem importadas
`skip`	informa à função a quantidadade de registros iniciais do arquivo devem se ignorados.
`strip.white`	informa à funãção que os caracteres brancos devem ser removidos dos dados
`comment.char`	informa à função quais caracteres estão sendo utilizado para designar valores faltantes. O valor *defa*

Arquivos com outros delimitadores também podem ser importados com a função `read.table()`, bastando definir o parâmetro `sep=` para o separador adequado. Delimitadores usuais são: `;`, `,`, `' '` e `|`.

Para exportarmos dados, utilizamos a função `write.table()`. Para exportarmos o conjunto de dados `receitas` que acabamos de importar, pode-se proceder da seguinte forma:

```
setwd(diretorio)
write.table(receitas, file='receitas.csv', sep=';', row.names=FALSE)
```

O conjunto de dados foi exportado no formato `.csv`. Também estão disponíveis as funções `write.csv()`e `write.csv2()`.

Estas funções podem importar dados que estejam hospedados em sites na internet com protocolo `http`.

No exemplo a seguir será visto como importar dados do *site* do Senado Federal. Especificamente, serão importados dados relativos aos contratos celebrados pelo Senado disponíveis em seu portal de transparência.

```
arquivo <- 'http://www.senado.gov.br/transparencia/LAI/licitacoes/contratos.csv'
contratos <- read.csv2(arquivo, skip=1, as.is = TRUE)
head(contratos)
dim(contratos)
```

3.2.2 Importação de arquivos do Excel

O R não possui funções nativas para a importação ou exportação de dados para o foramto de arquivo do Excel. Não obstante, existem pacotes que realizam esta tarefa. Nestas notas serão utilizados dois pacotes: `readxl` e `openxlsx`. O pacote `readxl` somente tem a funcionalidade de leitura de dados e o `openxlsx` possui as funcionalidades de leitura e escrita de dados. No Anexo IV apresentamos com maiore datalhes como utilizar o pacote `openxlsx`.

Como estes pacotes não são pré-instalados, será necessário instalá-los, o que pode ser feito da seguinte forma:

```
install.packages(c('readxl', 'openxlsx'))
```

Uma outra forma de instalar estes pacotes é com a utilização do botão `Install` existente na aba `Packages` do painel inferior direito do RStudio.

Uma vez instalados os pacotes, para disponibilizar as funções para uso é necessário carregar os pacotes. Isto é feito da seguinte forma:

```
library(readxl)
library(openxlsx)
```

Para realizar a importação dos dados contidos no arquivo `precos_acoes_petrobras.xlsx` utilizando-se a função `read_excel()` do pacote `readxl` pode-se proceder da seguinte forma:

```
setwd(diretorio)
acoes_petrobras <- read_excel('precos_acoes_petrobras.xlsx')
tail(acoes_petrobras)
```

```
          Data Cotacao Mínima Máxima Variação Variação (%)   Volume
241 2014-01-09   15.70  15.65  16.29    -0.49        -3.03 25579200
242 2014-01-08   16.19  16.15  16.39     0.03         0.19 15558400
243 2014-01-07   16.16  16.16  16.83    -0.46        -2.77 18785300
244 2014-01-06   16.62  16.16  16.64     0.20         1.22 20474600
245 2014-01-03   16.42  16.42  16.78    -0.33        -1.97 17598400
246 2014-01-02   16.75  16.65  17.20    -0.33        -1.93 17111300
```

Com o pacote `openxlsx` a importação dos dados pode ser feita da seguinte forma:

```
setwd(diretorio)
bolsa_valores <- read.xlsx("tui.xlsx", sheet = 1)
```

A exportação de dados para arquivo do excel com este pacote pode ser feita da seguinte forma.

```
setwd(diretorio)
write.xlsx(mtcars, file="mtcars.xlsx",  sheet="mtcars")
```

Outro pacote que pode ser utilizado para trabalhar com arquivos do Excel é o `XLConnect` para o qual o usuário poderá obter informações no site `http://www.mirai-solutions.com/site/index.cfm?id_art=66328`. O tutorial deste pacote pode ser baixado no seguinte site: `http://cran.r-project.org/web/packages/XLConnect/vignettes/`

Nota: O pacote `openxlsx` não possui dependência do java como o `XLConnect` mas exige que se tenha um aplicativo para compactação de arquivos na `search path` do windows ou o aplicativo `Rtools` instalado.

3.2.3 Importar da área de transferência

Uma possibilidade é a importação de dados que foram copiados para a área de transferência do Windows. Esta possibilidade é utilizada para importar dados contidos em planilhas do excel que tenham sido copiados para a área de transferência.

Este procedimento será ilustrado com o conjunto de dados `tui.xls`. Após copiar todo o conteúdo da planilha, executa-se o código a seguir:

```
acoes <- read.delim2("clipboard")
```

3.2.4 Arquivos texto de formato fixo

Arquivos texto de formato fixo são arquivos nos quais os dados ocupam tamanho fixo em cada linha do arquivo.

Para a importação de arquivos desta natureza é necessário que se disponha do dicionário de variáveis, que é o documento onde estarão especificadas quais variáveis estão representadas no arquivo, a posição de início de cada variável no arquivo e o comprimento de cada variável.

O arquivo `Arfile.ASC` constante do rol de arquivos que serão utilizados no curso é deste tipo. O dicionário deste arquivo de dados consta do documento `Descricao Arquivos Dados_v2.doc` também disponibilizado.

Este tipo de arquivo é utilizado, por exemplo, para armazenar os microdados da PNAD, do Censo Escolar, etc.

O arquivo `Arfile.ASC` pode ser importado da seguinte forma:

```
## Definir o diretório de trabalho
setwd(diretorio)

contas_receber <- read.fwf('Arfile.ASC',
                           widths=c(11, 4, 4, 15, 8),
                           col.names=c('account', 'division', 'store', 'balance', 'duedate'))

head(contas_receber)
```

```
      account division store  balance  duedate
1 S0000309077      246    20 13192.42 20010101
2 S0000041943       87     3   260.97 20010103
3 S0000143191       87    20  9541.28 20010106
4 S0000459709     9045    20  2254.19 20010110
5 S0000030187      139     4  2286.84 20010110
6 S0000002624       28     9  3993.90 20010111
```

3.2.5 Arquivos `.RData` e `.R`

Dados existentes no R podem ser salvos no formato `.RData`. O conteúdo destes arquivos podem ser obtidos com a função `load()`.

```
setwd(diretorio)
load('sigrh.RData')
dim(sigrh)
```

```
[1] 114806     18
```

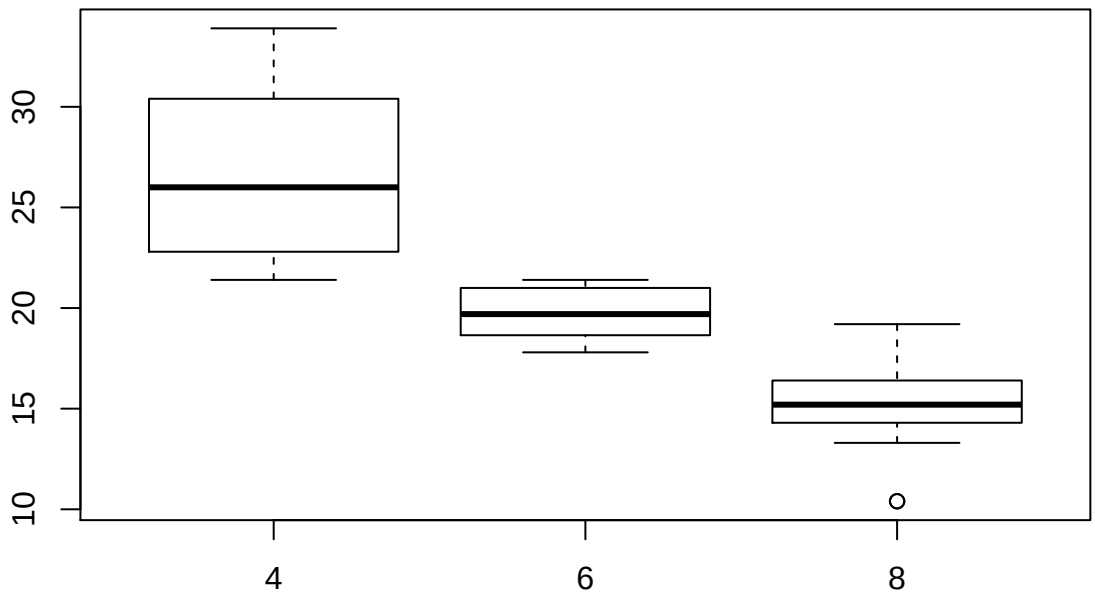
Para salvar qualquer objeto (vetor, lista, data frame, matriz, array, função, etc.) existente na área de trabalho, utiliza-se a função `save()`. Exemplo:

```
save(contas_receber, file='contas_receber.RData')
```

Os comandos existentes em um arquivo de *script* do R (`.R`) podem ser executados ao se carregar o arquivo no R. Isto pode ser feito com a função `source()`. Exemplo:

```
setwd(diretorio)
source('hello_world.R')
```

```
[1] "Hello world!"
```



3.2.6 Arquivos de pacotes estatísticos

O R possui o pacote `foreign` que possui funções que permitem importar arquivos de dados de alguns pacotes estatísticos, como `Stata`, `epiinfo`, `SPSS` e `SYSTAT`.

Os exemplos a seguir ilustram como importar arquivos do Stata (`.dta`) e do SPSS (`.sav`).

```
library(foreign)
setwd(diretorio)

## Stata
stata_file <- read.dta('regsim.dta')
str(stata_file)
```

```
'data.frame':   500 obs. of  12 variables:
 $ b1   : num  1.93 2.15 2.06 2.13 1.98 ...
 $ b1r  : num  1.93 2.17 2.05 2.14 1.93 ...
 $ se1r : num  0.1086 0.1302 0.1056 0.0939 0.1254 ...
 $ b1q  : num  1.97 2.1 2.06 2.13 1.87 ...
 $ se1q : num  0.135 0.159 0.132 0.137 0.153 ...
 $ se1qb: num  0.1568 0.1954 0.1387 0.0887 0.1469 ...
 $ b2   : num  2.03 2.05 1.9 2.45 2.13 ...
 $ b2r  : num  1.9 2.18 2.07 2.11 1.92 ...
 $ se2r : num  0.112 0.1292 0.1066 0.0944 0.1244 ...
 $ b2q  : num  1.88 2.1 2.06 2.13 1.87 ...
```

```
 $ se2q : num  0.146 0.159 0.132 0.137 0.153 ...
 $ se2qb: num  0.1529 0.1886 0.1486 0.0962 0.1533 ...
 - attr(*, "datalabel")= chr "Monte Carlo estimates of b in 500 samples of n=100"
 - attr(*, "time.stamp")= chr " 2 Jul 2012 06:11"
 - attr(*, "formats")= chr  "%9.0g" "%9.0g" "%9.0g" "%9.0g" ...
 - attr(*, "types")= int  254 254 254 254 254 254 254 254 254 254 ...
 - attr(*, "val.labels")= chr  "" "" "" "" ...
 - attr(*, "var.labels")= chr  "r(B1)" "r(B1R)" "r(SE1R)" "r(B1Q)" ...
 - attr(*, "expansion.fields")=List of 38
  ..$ : chr  "_dta" "command" "regsim"
  ..$ : chr  "_dta" "seed" "X16f16f11385e59bdd74e06a06a630271000438a1"
  ..$ : chr  "b1" "expression" "r(B1)"
  ..$ : chr  "b1" "coleq" "_"
  ..$ : chr  "b1" "colname" "b1"
  ..$ : chr  "b1r" "expression" "r(B1R)"
  ..$ : chr  "b1r" "coleq" "_"
  ..$ : chr  "b1r" "colname" "b1r"
  ..$ : chr  "se1r" "expression" "r(SE1R)"
  ..$ : chr  "se1r" "coleq" "_"
  ..$ : chr  "se1r" "colname" "se1r"
  ..$ : chr  "b1q" "expression" "r(B1Q)"
  ..$ : chr  "b1q" "coleq" "_"
  ..$ : chr  "b1q" "colname" "b1q"
  ..$ : chr  "se1q" "expression" "r(SE1Q)"
  ..$ : chr  "se1q" "coleq" "_"
  ..$ : chr  "se1q" "colname" "se1q"
  ..$ : chr  "se1qb" "expression" "r(SE1QB)"
  ..$ : chr  "se1qb" "coleq" "_"
  ..$ : chr  "se1qb" "colname" "se1qb"
  ..$ : chr  "b2" "expression" "r(B2)"
  ..$ : chr  "b2" "coleq" "_"
  ..$ : chr  "b2" "colname" "b2"
  ..$ : chr  "b2r" "expression" "r(B2R)"
  ..$ : chr  "b2r" "coleq" "_"
  ..$ : chr  "b2r" "colname" "b2r"
  ..$ : chr  "se2r" "expression" "r(SE2R)"
  ..$ : chr  "se2r" "coleq" "_"
  ..$ : chr  "se2r" "colname" "se2r"
  ..$ : chr  "b2q" "expression" "r(B2Q)"
  ..$ : chr  "b2q" "coleq" "_"
  ..$ : chr  "b2q" "colname" "b2q"
  ..$ : chr  "se2q" "expression" "r(SE2Q)"
  ..$ : chr  "se2q" "coleq" "_"
  ..$ : chr  "se2q" "colname" "se2q"
  ..$ : chr  "se2qb" "expression" "r(SE2QB)"
  ..$ : chr  "se2qb" "coleq" "_"
  ..$ : chr  "se2qb" "colname" "se2qb"
 - attr(*, "version")= int 12
```

```
## SPSS
spss_file <- read.spss('p004.sav', to.data.frame = TRUE)
str(spss_file)
```

```
'data.frame':   199 obs. of  7 variables:
```

```
 $ CURRENTM: num  45 86 50 42 61 93 91 90 53 84 ...
 $ PREVIOUS: num  45 86 50 42 61 93 91 90 53 84 ...
 $ FAT     : num  5.5 4.4 6.5 7.4 3.8 ...
 $ PROTEIN : num  8.9 4.1 4 4.1 3.8 ...
 $ DAYS    : num  21 25 25 25 33 45 46 46 46 50 ...
 $ LACTATIO: num  5 4 7 2 2 3 2 5 2 7 ...
 $ I79     : num  0 0 0 0 0 0 0 0 0 0 ...
 - attr(*, "variable.labels")= Named chr  "CurrentMilk" "Previous" "Fat" "Protein" ...
  ..- attr(*, "names")= chr  "CURRENTM" "PREVIOUS" "FAT" "PROTEIN" ...
```

Arquivos `.dbf` também podem ser importados pelo R. Exemplo:

```
setwd(diretorio)
obito <- read.dbf("DOPRJ2014.dbf", as.is = TRUE)
head(obito)[,1:4]
```

```
  NUMERODO       CODINST ORIGEM NUMERODV
1 16785720 RRJ3304550000      1        7
2 19090442 RRJ3304550000      1        9
3 19090450 MRJ3302700001      1        0
4 19090452 MRJ3302700001      1        6
5 19091363 MRJ3303300001      1        0
6 19103500 RRJ3304550000      1        9
```

O arquivo acima refere-se às declarações de óbitos emitidas no Estado do Rio de Janeiro, apuração preliminar, para o ano de 2014 baixado o site do DATASUS.

Para a importação de arquivos `.dbf` compactados (`.dbc`) pode-se utilizar o pacote `read.dbc` que dispõe da função `read.dbc()` para a importação deste tipo de arquivo, muito utilizado pelo DATASUS.

3.2.7 Importação de dados de Sistemas Gerenciadores de Bancos de Dados - SGBDs

Muitas vezes os dados necessários à análise estão armazenados em bancos de dados. Existem diversos desses aplicativos no mercado sendo bastante conhecidos o `SQLServer`, `ORACLE`, `MySQL`, `PostgreSQL`, `Access`, `SQLite`, etc.

O R dispõe de um conjunto de pacotes que possibilitam o acesso aos dados armazenados nestes aplicativos. Nestas notas será visto como acessar dados contidos em arquivo do aplicativo `Access`.

Para a recuperação de dados contidos em arquivos do `Access`, será utilizado o pacote `RODBC`, que permite a realização de conexões via ODBC com diversos bancos de dados.

```
library(RODBC)
odbcDataSources() # consulta os drives instalados
```

```
                                        Excel Files
"Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb)"
                                    MS Access Database
             "Microsoft Access Driver (*.mdb, *.accdb)"
                                           access_32
                  "Driver do Microsoft Access (*.mdb)"
```

```r
setwd(diretorio)
conn <- odbcDriverConnect(paste("DRIVER=Driver do Microsoft Access (*.mdb)",
                                "DBQ=C:\\Users\\Marcos\\Dropbox\\1. Cursos ECG\\Intro-R Treinamento TCE

sqlTables(conn) # lista as tabelas existentes na base...
```

```
                                                                                TABLE_CAT
1  C:\\Users\\Marcos\\Dropbox\\1. Cursos ECG\\Intro-R Treinamento TCE-MT\\3.dados\\Microdados_Set15
2  C:\\Users\\Marcos\\Dropbox\\1. Cursos ECG\\Intro-R Treinamento TCE-MT\\3.dados\\Microdados_Set15
3  C:\\Users\\Marcos\\Dropbox\\1. Cursos ECG\\Intro-R Treinamento TCE-MT\\3.dados\\Microdados_Set15
4  C:\\Users\\Marcos\\Dropbox\\1. Cursos ECG\\Intro-R Treinamento TCE-MT\\3.dados\\Microdados_Set15
5  C:\\Users\\Marcos\\Dropbox\\1. Cursos ECG\\Intro-R Treinamento TCE-MT\\3.dados\\Microdados_Set15
6  C:\\Users\\Marcos\\Dropbox\\1. Cursos ECG\\Intro-R Treinamento TCE-MT\\3.dados\\Microdados_Set15
7  C:\\Users\\Marcos\\Dropbox\\1. Cursos ECG\\Intro-R Treinamento TCE-MT\\3.dados\\Microdados_Set15
8  C:\\Users\\Marcos\\Dropbox\\1. Cursos ECG\\Intro-R Treinamento TCE-MT\\3.dados\\Microdados_Set15
9  C:\\Users\\Marcos\\Dropbox\\1. Cursos ECG\\Intro-R Treinamento TCE-MT\\3.dados\\Microdados_Set15
10 C:\\Users\\Marcos\\Dropbox\\1. Cursos ECG\\Intro-R Treinamento TCE-MT\\3.dados\\Microdados_Set15
   TABLE_SCHEM                TABLE_NAME   TABLE_TYPE REMARKS
1         <NA>          MSysAccessStorage SYSTEM TABLE    <NA>
2         <NA>                  MSysACEs SYSTEM TABLE    <NA>
3         <NA> MSysNavPaneGroupCategories SYSTEM TABLE    <NA>
4         <NA>          MSysNavPaneGroups SYSTEM TABLE    <NA>
5         <NA>  MSysNavPaneGroupToObjects SYSTEM TABLE    <NA>
6         <NA>       MSysNavPaneObjectIDs SYSTEM TABLE    <NA>
7         <NA>               MSysObjects SYSTEM TABLE    <NA>
8         <NA>               MSysQueries SYSTEM TABLE    <NA>
9         <NA>          MSysRelationships SYSTEM TABLE    <NA>
10        <NA>           Microdados_Set15        TABLE    <NA>
```

```r
sqlColumns(conn, "Microdados_Set15")[, c('COLUMN_NAME', 'TYPE_NAME')] # identifica as colunas...
```

```
   COLUMN_NAME TYPE_NAME
1     rgocronu   VARCHAR
2         nvpi   VARCHAR
3         vori   VARCHAR
4         vano   VARCHAR
5         etit   VARCHAR
6         eten   VARCHAR
7         enas   VARCHAR
8         eida   VARCHAR
9         emai   VARCHAR
10        esex   VARCHAR
11        ecor   VARCHAR
12        epro   VARCHAR
13        eesc   VARCHAR
14        eeci   VARCHAR
15        enat   VARCHAR
16        ebai   VARCHAR
17        emun   VARCHAR
18        datc   VARCHAR
19        DSCR   VARCHAR
20        locf   VARCHAR
```

```
21        situ   VARCHAR
22        circ   VARCHAR
23        inst   VARCHAR
24        ftlo   VARCHAR
25        flog   VARCHAR
26        fnum   VARCHAR
27        fcom   VARCHAR
28        fref   VARCHAR
29        fbai   VARCHAR
30        ftlc   VARCHAR
31        rela   VARCHAR
32        datf   VARCHAR
33        horf   VARCHAR
34        horc   VARCHAR
35        fmun   VARCHAR
36        fufe   VARCHAR
37           x   VARCHAR
38           y   VARCHAR
39    falecido   VARCHAR
40       preso   VARCHAR
41       datro   VARCHAR
42  rgoc9099fl   VARCHAR
43  reautuacao   VARCHAR
44  desmembrad   VARCHAR
45      qlfcid   VARCHAR
46  guia_prisa   VARCHAR
47        eseq   VARCHAR
48        aisp   VARCHAR
49        hora   VARCHAR
50      diasem    DOUBLE
51       faixa    DOUBLE
52         mes    DOUBLE
53      regiao   VARCHAR
54       naisp    DOUBLE
55        risp    DOUBLE
56        base   VARCHAR
57         upj    DOUBLE
58       doerj    DOUBLE
59    etenresu   VARCHAR
60   reautuado    DOUBLE
61 desmembrado    DOUBLE
62   delito_DO    DOUBLE
63  total_rbft    DOUBLE
64   delegacia    DOUBLE
65        etenx   VARCHAR
66     nascido    DOUBLE
67   duplicado    DOUBLE
```

```
## Extrair todos os dados existentes na tabela Microdados_Set15
dados_mdb <- sqlFetch(conn, "Microdados_Set15")
dim(dados_mdb)
```

```
[1] 205730     67
```

```
head(dados_mdb)
```

```
        rgocronu nvpi vori vano etit       eten        enas eida
1 004-07156/2015 7156    4 2015 1040 Testemunha          NA
2 004-07156/2015 7156    4 2015 1040     Vítima 25-APR-1943   72
3 004-07157/2015 7157    4 2015   92      Autor          NA
4 004-07157/2015 7157    4 2015   92     Vítima 28-JUL-1988   27
5 004-07158/2015 7158    4 2015    7      Autor          NA
6 004-07158/2015 7158    4 2015    7      Autor          NA
            emai      esex   ecor                epro             eesc
1                MASCULINO Branca       Policial civil
2 MAIOR DE IDADE  FEMININO Branca               Do lar  Alfabetizado(a)
3
4 MAIOR DE IDADE MASCULINO Branca Analista de sistemas 3º Grau completo
5                MASCULINO  Parda
6                MASCULINO  Parda
       eeci           enat         ebai           emun        datc
1                                                     01-SEP-2015
2  Ignorado RIO DE JANEIRO       CENTRO RIO DE JANEIRO 01-SEP-2015
3                                                     01-SEP-2015
4 Casado(a) RIO DE JANEIRO CAMPO GRANDE RIO DE JANEIRO 01-SEP-2015
5                                                     01-SEP-2015
6                                                     01-SEP-2015
                              DSCR                  locf
1 Remoção para Verificação de Óbito        RUA DO LAVRADIO
2 Remoção para Verificação de Óbito        RUA DO LAVRADIO
3         Furto de Telefone Celular PRAÇA CRISTIANO OTTONI
4         Furto de Telefone Celular PRAÇA CRISTIANO OTTONI
5               Abuso de Autoridade       RUA MIGUEL COUTO
6               Abuso de Autoridade       RUA MIGUEL COUTO
                              situ circ inst  ftlo             flog fnum
1                RO TRANSF.OUTRA DP    5        RUA      DO LAVRADIO  178
2                RO TRANSF.OUTRA DP    5        RUA      DO LAVRADIO  178
3                       RO SUSPENSO    4      PRAÇA CRISTIANO OTTONI   01
4                       RO SUSPENSO    4      PRAÇA CRISTIANO OTTONI   01
5 TERMO CIRCUNSTANCIADO EM ANDAMENTO    4        RUA     MIGUEL COUTO   00
6 TERMO CIRCUNSTANCIADO EM ANDAMENTO    4        RUA     MIGUEL COUTO   00
           fcom                                    fref   fbai
1       APTº 507                                     CENTRO
2       APTº 507                                     CENTRO
3 CENTRAL BRASIL                                     CENTRO
4 CENTRAL BRASIL                                     CENTRO
5                ESQUINA DA AVENIDA PRESIDENTE VARGAS CENTRO
6                ESQUINA DA AVENIDA PRESIDENTE VARGAS CENTRO
                 ftlc    rela        datf  horf     horc
1          RESIDÊNCIA NENHUMA 10-JUL-2015 09:23 01:05:40
2          RESIDÊNCIA NENHUMA 10-JUL-2015 09:23 01:05:40
3 ESTAÇÃO FERROVIÁRIA NENHUMA 31-AUG-2015 17:30 07:24:20
4 ESTAÇÃO FERROVIÁRIA NENHUMA 31-AUG-2015 17:30 07:24:20
5         VIA PÚBLICA NENHUMA 01-SEP-2015 08:40 09:30:02
6         VIA PÚBLICA NENHUMA 01-SEP-2015 08:40 09:30:02
                     fmun fufe x y falecido preso    datro rgoc9099fl
1            RIO DE JANEIRO   RJ 0 0        0     0 1/9/2015          0
```

```
2            RIO DE JANEIRO   RJ 0 0        1     0 1/9/2015          0
3 COMENDADOR LEVY GASPARIAN   RJ 0 0        0     0 1/9/2015          0
4 COMENDADOR LEVY GASPARIAN   RJ 0 0        0     0 1/9/2015          0
5            RIO DE JANEIRO   RJ 0 0        0     0 1/9/2015          1
6            RIO DE JANEIRO   RJ 0 0        0     0 1/9/2015          1
  reautuacao desmembrad    qlfcid guia_prisa eseq aisp hora diasem faixa
1         NA         NA 229811814               1    5    9      6     2
2         NA         NA 229811837               2    5    9      6     2
3         NA         NA 229812308               1    5   17      2     3
4         NA         NA 229812305               2    5   17      2     3
5         NA         NA 229813190               1    5    8      3     2
6         NA         NA 229813173               2    5    8      3     2
  mes regiao naisp risp base upj  doerj   etenresu reautuado desmembrado
1   9    CAP     5    1        0  11040 TESTEMUNHA         1           1
2   9    CAP     5    1        1  11040     VÍTIMA         1           1
3   9    CAP     5    1        0 210092      AUTOR         1           1
4   9    CAP     5    1        1 210092     VÍTIMA         1           1
5   9    CAP     5    1        0  10007      AUTOR         1           1
6   9    CAP     5    1        0  10007      AUTOR         1           1
  delito_DO total_rbft delegacia      etenx nascido duplicado
1        NA         NA         5 Testemunha       2         0
2        NA         NA         5     Vítima       1         0
3        NA          2         4      Autor       2         0
4        NA         NA         4     Vítima       1         0
5        NA         NA         4      Autor       2         0
6        NA         NA         4      Autor       2         0
```

```
## Realizar uma consulta SQL...
consulta <- "select *
             from Microdados_Set15
             where aisp = '18' and regiao = 'CAP'"

dados_mdb <- sqlQuery(conn, consulta, max=30) # submete a query e retorna 30 linhas...
dim(dados_mdb)
```

```
[1] 30 67
```

```
## Echar a conexão com a base de dados.
odbcClose(conn)
```

Com o pacote `sqldf` é possível recuperar dados de arquivos `.csv` utilizando consultas SQL como se o arquivo fosse uma tabela de um banco de dados.

```
library(sqldf)
```

```
Loading required package: gsubfn

Loading required package: proto

Loading required package: RSQLite

Loading required package: DBI
```

```r
setwd(diretorio)
conexao <- file('ESCOLAS.CSV')
escolas_estaduais <- sqldf('select * from conexao', file.format = list(sep='|'))
```

```
Loading required package: tcltk
```

```r
dim(escolas_estaduais)
```

```
[1] 276331    141
```

```r
head(escolas_estaduais)[,1:4]
```

```
  ANO_CENSO PK_COD_ENTIDADE                    NO_ENTIDADE
1      2014        35925378                  PARQUE TAMARI
2      2014        22127046             CRECHE TIA REMEDIOS
3      2014        22127062      INST EDUC MACHADO DE ASSIS
4      2014        35007237 LANDIA SANTOS BATISTA PROFESSORA
5      2014        35082156              JORACY CRUZ DR EMEF
6      2014        22012745         ESC MUL DE SAO JERONIMO
  COD_ORGAO_REGIONAL_INEP
1                   10318
2                      13
3                       0
4                   10502
5                   10502
6                       0
```

4. EXERCÍCIOS

1. Importe o conjunto de dados `Address.ASC`. Quantos registros tem este conjunto de dados?

Nota: O dicionário de dados está no arquivo `Descricao Arquivos Dados_v2.doc`.

2. Tente realizar a importação do conjunto de dados `Employees.txt`. O arquivo foi importado corretamente? Caso não tenha sido, você sabe dizer qual a razão?

3. Realize a importação do conjunto de dados `balanco.csv` utilizando o mecanismo de importar da área de transferência.

4. Relize a importação do conjunto de dados `despesas_candidatos_2014_RJ.txt`. Quantos registros e quantas variáveis tem este conjunto de dados?

5. Importe o conjunto de dados `tui.RData`. Quais as variáveis existentes neste conjunto de dados?

6. Importe o conjunto de dados contido no arquivo `carsdata.dta` e `Child Aggression.sav`. Quantas observações tem cada um dos arquivos?

7. Importe do conjunto de dados existente no seguinte endereço:

`http://dominios.governoeletronico.gov.br/dados-abertos/Dominios_GovBR_basico.csv`

Quantos registros tem este conjunto de dados? Quantas variáveis?

8. Utilizando o pacote `sqldf` importe os 50 primeiros registros do conjunto de dados `Microdados 2013 - 2015.csv` contido no arquivo `Microdados 2013 - 2015.zip`.

CAPÍTULO 4 - MANIPULAÇÃO DE DADOS

Neste capítulo serão vistos alguns recursos disponíveis no R para a manipulação de bases de dados.

4.1 - STRINGS E DATAS

4.1.1 Datas e horas

Datas e horas são tipos de dados frequentemente encontrados na prática e o R dispõe de algumas funções para lidar com eles.

Para lidar com datas, a principal função é `as.Date()` que possibilita a conversão de strings representando datas em um objeto da classe `Date`. Esta função recebe como argumento um vetor de caracteres que representem data (argumento `x`) e uma indicação de como as datas estão representadas pelos caracteres (argumento `format=`).

Por exemplo, as *strings* "02102016", "02-10-2016", "02/10/2016", "02.10.2016", "02OUT2016" representam, todas, a mesma data: "dois de outubro de 2016".

Os exemplos a seguir ilustram a utilização desta função:

```
# Criação de um vetor de strings representando datas
data1 <- c("2014-11-18", "2007-12-14", "2015-06-01")
class(data1)
```

```
[1] "character"
```

```
# Conversão do vetor de 'strings' para um vetor de datas
data1 <- as.Date(data1)
data1
```

```
[1] "2014-11-18" "2007-12-14" "2015-06-01"
```

```
class(data1)
```

```
[1] "Date"
```

Quando a data está representada no formato "YYYY-MM-DD" não é necessário passar à função um valor para o argumento `format=`. Os exemplos a seguir ilustram a conversão de caracteres em data para outras representações.

```
## Criação de vetor de strings representando datas. Outro formato.
data2 <- c("28/01/1972", "14/12/1942", "17/08/2014")
as.Date(data2, format="%d/%m/%Y")
```

```
[1] "1972-01-28" "1942-12-14" "2014-08-17"
```

```
data3 <- c("180504", "291214", "260875")
as.Date(data3, format="%d%m%y")
```

```
[1] "2004-05-18" "2014-12-29" "1975-08-26"
```

```r
data4 <- c("12 agosto 2013", "17 fevereiro 2011", "18 julho 2013")
as.Date(data4, "%d %B %Y")
```

```
[1] "2013-08-12" "2011-02-17" "2013-07-18"
```

Como pode ser visto, em alguns casos a conversão de *string* para datas exigiu que se fornecesse como argumento para a função um código de formatação indicando como as datas estão representadas.

O quadro a seguir fornece alguns destes códigos de formatação. Para maiores detalhes consultar o help da função `strptime()`

Código	Descrição
%d	Dia (decimal)
%m	Mês (decimal)
%b	Mês (abreviação)
%B	Mês (nome completo)
%y	Ano (dois dígitos)
%Y	Ano (quatro dígitos)

Uma vez que se diponha de um objeto que represente data, funções específicas para trabalhar com datas podem ser utilizadas, como por exemplo `weekdays()`, `quarters()`, `months()`.

```r
# Converter o vetor de caracteres 'data4' criado anteriormente para
# para um vetor de datas.
dias <- as.Date(data4, "%d %B %Y")
dias
```

```
[1] "2013-08-12" "2011-02-17" "2013-07-18"
```

```r
# Obter os dias da semana
weekdays(dias)
```

```
[1] "segunda-feira" "quinta-feira"  "quinta-feira"
```

```r
weekdays(dias, abbrev=TRUE)
```

```
[1] "seg" "qui" "qui"
```

```r
# Trimestres
quarters(dias)
```

```
[1] "Q3" "Q1" "Q3"
```

```r
## Meses
months(dias, abbreviate = TRUE)
```

```
[1] "ago" "fev" "jul"
```

Sequências de datas podem ser geradas com a função `seq()`:

```r
## Gerar um vetor de datas de 01/11/2014 a 09/11/2014
seq_data <- seq(as.Date("2014-11-01"), as.Date("2014-11-09"), "days")
seq_data
```

```
[1] "2014-11-01" "2014-11-02" "2014-11-03" "2014-11-04" "2014-11-05"
[6] "2014-11-06" "2014-11-07" "2014-11-08" "2014-11-09"
```

```r
## Identifica os finais de semana.
weekdays(seq_data, abbrev=TRUE) %in% c("sáb", "dom")
```

```
[1]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
```

A quantidade de dias entre duas datas pode ser calculada com a função `difftime()` da seguinte forma:

```r
x2 <- difftime(as.Date("2014-11-09"), as.Date("2014-11-01") , units="days")
x2
```

```
Time difference of 8 days
```

```r
x2 <- as.numeric(x2)
class(x2)
```

```
[1] "numeric"
```

```r
x2
```

```
[1] 8
```

Outras funções de interesse são: `strptime()` e `strftime()`.

A função `strptime` é utilizada para converter strings representando data para objetos da classe `POSIXlt` e `POSIXt`.

```r
datas <- c("28/01/1972", "14/12/1942", "17/08/2014")
datas_posix <- strptime(datas, "%d/%m/%Y")
datas_posix
```

```
[1] "1972-01-28 BRT" "1942-12-14 BRT" "2014-08-17 BRT"
```

```r
class(datas_posix)
```

```
[1] "POSIXlt" "POSIXt"
```

```r
data_hora <- c('2002-06-09 12:45:40',
               '2003-02-23 09:30:40',
               '2002-09-04 16:45:40',
               '2002-11-13 20:00:40',
               '2002-07-07 17:30:40')

data_hora <- strptime(data_hora, "%Y-%m-%d %H:%M:%S")
data_hora
```

```
[1] "2002-06-09 12:45:40 BRT"  "2003-02-23 09:30:40 BRT"
[3] "2002-09-04 16:45:40 BRT"  "2002-11-13 20:00:40 BRST"
[5] "2002-07-07 17:30:40 BRT"
```

```
## Obtem data e hora atual
Sys.time()
```

```
[1] "2016-07-31 23:02:44 BRT"
```

A função `strftime()` é utilizada para formatar a apresentação de datas. Exemplo:

```
strftime(datas_posix, '%Y') ## ano (4 dígitos)
```

```
[1] "1972" "1942" "2014"
```

```
strftime(datas_posix, '%w') ## dia da semana- 0 = domingo
```

```
[1] "5" "1" "0"
```

```
strftime(datas_posix, '%a') ## dia da semana abreviado
```

```
[1] "sex" "seg" "dom"
```

A função `format()` também pode ser utilizada para apresentar datas em formatos definidos pelo usuário, da mesma forma que a função `strftime()`.

```
format(datas_posix, '%a')
```

```
[1] "sex" "seg" "dom"
```

É muito comum recebermos dados onde as datas estejam expressas em inglês. Quando isso ocorre, a conversão dos dados para o formato de data necessita de cuidado adicional que consiste em alterar a configuração do R para que ele entenda corretamente os meses. O exemplo a seguir ilustra esta situação:

```
data_ingles <- c('09-Apr-14', '15-Apr-14', '14-May-14', '14-May-14',
                 '15-Apr-14', '10-Apr-14', '29-Oct-14', '29-Oct-14',
                 '30-Oct-14', '17-Dec-14', '17-Dec-14', '18-Dec-14')

## Tentativa 1 - erro... (desconsidera o fato de estar em inglês...)
as.Date(data_ingles, '%d-%b-%y')
```

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

```
## Tentativa 2 - altera a configuração...
#  Captura e guarda a configuração original...
minha_configuracao <-  Sys.getlocale("LC_TIME")
minha_configuracao
```

```
[1] "Portuguese_Brazil.1252"
```

```r
# altera a configuração...
Sys.setlocale("LC_TIME", "C")
```

```
[1] "C"
```

```r
# realiza a conversão... o mesmo feito anteriormente...
as.Date(data_ingles, '%d-%b-%y')
```

```
 [1] "2014-04-09" "2014-04-15" "2014-05-14" "2014-05-14" "2014-04-15"
 [6] "2014-04-10" "2014-10-29" "2014-10-29" "2014-10-30" "2014-12-17"
[11] "2014-12-17" "2014-12-18"
```

```r
# volta para a cofiguração original...
Sys.setlocale("LC_TIME", minha_configuracao)
```

```
[1] "Portuguese_Brazil.1252"
```

O pacote `lubridate` fornece várias funções para lidar com datas. Mais informações: `?DateTimeClasses`

4.1.2 Manipulação de Strings

Existem diversas funções que operam sobre *strings* de caracteres. As funções `tolower()` e `toupper()` são utilizadas para converter letras para caixa baixa e caixa alta respectivamente.

```r
toupper('curso de r')
```

```
[1] "CURSO DE R"
```

```r
tolower('CURSO DE R')
```

```
[1] "curso de r"
```

Uma função muito utilizada na prática é função `paste()`, que concatena as *strings* fornecidas como argumento.

```r
paste("A", 1:10, sep="|")
```

```
 [1] "A|1"  "A|2"  "A|3"  "A|4"  "A|5"  "A|6"  "A|7"  "A|8"  "A|9"  "A|10"
```

```r
paste("A", 1:10, sep="", collapse="-")
```

```
[1] "A1-A2-A3-A4-A5-A6-A7-A8-A9-A10"
```

A função `nchar()` retorna o número de caracteres em uma *string*. Exemplo:

```r
nchar(c("Marcos", "João", "Paulo"))
```

```
[1] 6 4 5
```

A função `strsplit()` realiza a separação de strings de acordo com algum padrão fornecido à função. Exemplos:

```r
strsplit(c("Marcos", "Marcio", "Sandra"), '')
```

```
[[1]]
[1] "M" "a" "r" "c" "o" "s"

[[2]]
[1] "M" "a" "r" "c" "i" "o"

[[3]]
[1] "S" "a" "n" "d" "r" "a"
```

```r
aa <- strsplit(c("28/01/1972", "14/12/1942", "17/08/2014"), "/")
aa
```

```
[[1]]
[1] "28"   "01"   "1972"

[[2]]
[1] "14"   "12"   "1942"

[[3]]
[1] "17"   "08"   "2014"
```

Para extrair partes de uma *string*, utiliza-se a função `substr()`.

```r
substr("calculo", 1, 3) # retorna um subconjunto de uma string
```

```
[1] "cal"
```

```r
substr("calculo", 3, 5)
```

```
[1] "lcu"
```

A função `strtrim()` também pode ser utilizada para extrair partes de uma string. Exemplo:

```r
strtrim(c("Marcos", "Marcio", "Sandra"), 3)
```

```
[1] "Mar" "Mar" "San"
```

```r
strtrim(rev(c("Marcos", "Marcio", "Sandra")), 5)
```

```
[1] "Sandr" "Marci" "Marco"
```

```r
strtrim(c("Marcos", "Marcio", "Sandra"), c(1,5,10))
```

```
[1] "M"      "Marci"  "Sandra"
```

A função `trimws()` remove espaços em branco antes, depois ou de ambos os lados de uma palavra. Exemplo:

```r
trimws(c('   Marcos', '   Marcio   ', 'Sandra   '))
```

```
[1] "Marcos" "Marcio" "Sandra"
```

4.2 - BÁSICO DE EXPRESSÕES REGULARES

Uma expressão regular é um padrão que descreve um conjunto de caracteres. Por exemplo, o padrão `sor` vai 'casar' com as seguintes *strings*: `sorriso`, `opressor`, `soro`, `sorte` e `resorte` já que todas contém a string `sor`, mas não casa com as *strings* `solar`, `senhor`, embora ambas contenham as strings `s`, `o` e `r`.

A principal função do R para fazer buscas utilizando expressões regulares é a função `grep()`. O exemplo a seguir ilustra sua utilização:

```r
# Define um vetor de caracteres
palavras <- c('sorriso', 'opressor', 'soro', 'sorte', 'solar','senhor', 'resorte')

# Retorna as strings no vetor 'palavras' que casam com a string 'sor'
grep('sor', palavras, value=TRUE)
```

```
[1] "sorriso"  "opressor" "soro"     "sorte"    "resorte"
```

Na composição de expressões regulares existem caracteres especiais que permitem uma grande generalização na busca por 'casamento' de *strings*. Por exemplo, supondo que se deseje obter as palavras que terminem com a *string* `or` e as que iniciem com a *string* `o` podemos utilizar, respectivamente, os caracteres especiais `$` e `^` conforme mostrado a seguir:

```r
# Palavras que terminem com 'or'
grep('or$', palavras, value=TRUE)
```

```
[1] "opressor" "senhor"
```

```r
# Palavras que iniciem com 'o'
grep('^o', palavras, value=TRUE)
```

```
[1] "opressor"
```

No quadro a seguir apresenta-se uma relação destes caracteres especiais, também conhecidos por metacaracteres:

Caractere	Descrição
^	casa caracteres no início da *string*
$	casa caracteres no fim da *string*
.	casa qualquer caractere
|	separa padrões alternativos
()	define um grupo
[]	define uma lista
	casa 0 ou mais ocorrências do caractere precedente
?	indica a ocorrência ou não do caractere precedente
+	uma ou mais ocorrências do caractere precedente
{n}	casa `n` ocorrências do caractere precedente

Caractere	Descrição
{n,}	casa no mínimo `n` ocorrências do caractere precedente
{n,m}	casa de `n` até `m` ocorrências do caractere precedente
\d \D	casa dígito e casa não dígitos
\s \S	casa espaço e casa não espaço
\w \W	casa palavras e casa não palavras
\b	casa o caractere branco no início ou fim de uma *string*

A construção de `classes de caracteres` pode ser feita colocando-se um conjunto de caracteres entre colchetes (lista). Por exemplo, a expressão regular "Jos[ée]" irá casar tanto com a *string* "José" como "Jose".

Uma lista negada pode ser construída colocando-se o caractere "^" como primeiro elemento de uma `classe de caracteres`, fazendo com que a expressão regular case qualquer coisa menos a *string* dentro dos colchetes.

A combinação destes caracteres especiais permitem o casamento de padrões de *string* bem complexos.

Assim como a função `grep()`, outras funções do R tem por argumento expressões regulares, como por exemplo, as funções `grepl()`, `strsplit()`, `list.files()`, `sub()`, `gsub()`, `regexpr()`, `gregexpr()`, `regexec()` e `apropos()`.

As funções `sub()` e `gsub()` são utilizadas para realizar a substituição de *strings*. Estas funções recebem como argumento uma expressão regular que represente a *string* a ser substituída e a *string* a ser utilizada para realizar a substituição. A difernça entre as duas funções reside no fato de que `sub()` realizará a substituição apenas na primeira ocorrência de casamento da expresssão regular com as *strings* fornecidas, enquanto `gsub()`realizará a substituição em todas as ocorrências de casamentos da expressão regular com as *strings* fornecidas.

No exemplo a seguir o objetivo é realizar a substituição de toda a *string* contida em cada elemento do vetor `txt` por uma que indique o número do processo:

```
# Criação de um vetor de strings
txt <- c(
"A licitação constitui o processo TCE-RJ 104.308-4/12 que foi cadastrado no dia 02/08/2014.",
"O processo 203.475-8/14 refere-se a uma inexigibilidade de licitação",
"A denúncia foi formalizada no processo 108703-2/09",
"103428-5/15",
"Deu entrada o processo TCE-RJ no 1034268/12")

txt
```

```
[1] "A licitação constitui o processo TCE-RJ 104.308-4/12 que foi cadastrado no dia 02/08/2014."
[2] "O processo 203.475-8/14 refere-se a uma inexigibilidade de licitação"
[3] "A denúncia foi formalizada no processo 108703-2/09"
[4] "103428-5/15"
[5] "Deu entrada o processo TCE-RJ no 1034268/12"
```

```
sub(".*(\\d{3}\\.?\\d{3}-?\\d/\\d{2}).*", "\\1", txt)
```

```
[1] "104.308-4/12" "203.475-8/14" "108703-2/09"  "103428-5/15"
[5] "1034268/12"
```

A expressão regular fornecida à função `sub()` casa com todas as *strings* existentes no vetor `txt`. Utilizamos um retrovisor ("\1") que indica que se deve utilizar o trecho da expressão regular entre parênteses, no caso

`\\d{3}\\.?\\d{3}-?\\d/\\d{2}` que casa com os números de processos, para substituir a *string* casada. Assim, o que se fez foi substituir toda a *string* por um trecho dela que representa o número de processo.

Outros exemplos de uso de expressões regulares:

```
vetor <- c("Abril", "Agosto", "Julho", "Maio", "abril")
vetor
```

```
[1] "Abril"  "Agosto" "Julho"  "Maio"   "abril"
```

```
grep("^A", vetor, ignore.case=TRUE)
```

```
[1] 1 2 5
```

```
grep("^A", vetor, ignore.case=TRUE, value=TRUE)
```

```
[1] "Abril"  "Agosto" "abril"
```

```
grep("il$", vetor, ignore.case=TRUE, value=TRUE)
```

```
[1] "Abril" "abril"
```

```
txt <- c("O meu cpf é 012.486/00 e só", "o cpf dele é 048-762/01 e ele não sabe", "o cpf da galera é 05

sub(".*(\\d{3}[\\.-]?\\d{3}/\\d{2}).*", "\\1", txt)
```

```
[1] "012.486/00" "048-762/01" "052796/01"
```

A função `grepl()` retorna `TRUE` ou `FALSE` caso tenha havido ou não casamento da expressão regular fornecida à função com as *strings* passadas à função. Isso faz com que esta função seja muito conveniente para a uitlização em filtros.

Para mais informações `?regex`

4.3 - MANIPULAÇÃO DE DADOS

Nos exemplos que se seguem serão utilizados diversos conjuntos de dados. O primeiro a ser utilizado é o conjunto de dados `RH.csv` que contém informações relativas aos funcionários de uma empresa.

A importação da base de dados é feita da seguinte forma:

```
diretorio <- "C:\\Users\\Marcos\\Dropbox\\1. Cursos ECG\\Intro-R Treinamento TCE-MT\\3.dados"
setwd(diretorio)

rh <- read.csv2("RH.csv", as.is=TRUE)
```

É boa prática sempre conferir se a importação dos dados se deu corretamente. Checagens possíveis são:

```
head(rh, 2)
```

```
       Sexo Estado.Civil Anos.de.estudo         Formação Tempo.de.empresa
1 Masculino       Casado             14 Sócio-econômicas               19
2 Masculino        Viúvo             19 Sócio-econômicas               31
    Unidade Departamento      Cargo Salário Bônus
1  Curitiba     Produção Assistente   16.67 28.02
2 São Paulo       Vendas Assistente   29.13 41.24
```

```
str(rh)
```

```
'data.frame':   5000 obs. of  10 variables:
 $ Sexo            : chr  "Masculino" "Masculino" "Feminino" "Feminino" ...
 $ Estado.Civil    : chr  "Casado" "Viúvo" "Casado" "Casado" ...
 $ Anos.de.estudo  : int  14 19 18 16 15 18 18 12 14 12 ...
 $ Formação        : chr  "Sócio-econômicas" "Sócio-econômicas" "Sócio-econômicas" "Sócio-econômicas" .
 $ Tempo.de.empresa: int  19 31 28 20 15 23 27 20 11 16 ...
 $ Unidade         : chr  "Curitiba" "São Paulo" "Rio de Janeiro" "Rio de Janeiro" ...
 $ Departamento    : chr  "Produção" "Vendas" "Financeiro" "Vendas" ...
 $ Cargo           : chr  "Assistente" "Assistente" "Assistente" "Assistente" ...
 $ Salário         : num  16.7 29.1 21.8 22.6 16.7 ...
 $ Bônus           : num  28.02 41.24 16.88 13.5 8.44 ...
```

Aparentemente, não houve qualque problema com a importação dos dados. Inicia-se, a seguir a apresentação de como implementar algumas técnicas de análise de dados no R.

‘

4.3.1 Inclusão e exclusão de variáveis

O cálculo de novos campos em um `data frame` a partir de campos já existentes no conjunto de dados pode ser feito utilizando-se as funções `transform()` ou `within()` ou ainda utilizando-se o operador `$`.

No exemplo a seguir duas novas variáveis serão incluídas no data frame `rh`.

```
rh <- transform(rh, SalTot = Salário + Bônus,
                    SalLiq = 0.95 * Salário)
```

Com a função `within()`, a criação de uma nova variável dá-se da seguinte forma:

```
rh <- within(rh, Desconto <- Salário * (1 - 0.08))
```

Utilizando-se o operador `$`:

```
rh$NovaColuna <- rh$SalTot + 1000
head(rh, 2)
```

```
       Sexo Estado.Civil Anos.de.estudo         Formação Tempo.de.empresa
1 Masculino       Casado             14 Sócio-econômicas               19
2 Masculino        Viúvo             19 Sócio-econômicas               31
    Unidade Departamento      Cargo Salário Bônus SalTot  SalLiq Desconto
```

```
1  Curitiba     Produção Assistente   16.67 28.02  44.69 15.8365  15.3364
2 São Paulo       Vendas Assistente   29.13 41.24  70.37 27.6735  26.7996
  NovaColuna
1    1044.69
2    1070.37
```

A função `with()` permite acessar diretamente as colunas (variáveis) de um `data frame` e realizar operações com elas. Exemplo:

```
with(rh, summary(Salário))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   1.00    3.39    8.44   12.84   17.52  151.50       8
```

A eliminação de variáveis pode ser feita utilizando-se `NULL`. A eliminação da variável `SalLiq` pode ser feita da seguinte forma:

```
rh$SalLiq <- NULL
head(rh, 2)
```

```
       Sexo Estado.Civil Anos.de.estudo         Formação Tempo.de.empresa
1 Masculino       Casado             14 Sócio-econômicas               19
2 Masculino        Viúvo             19 Sócio-econômicas               31
    Unidade Departamento      Cargo Salário Bônus SalTot Desconto
1  Curitiba     Produção Assistente   16.67 28.02  44.69  15.3364
2 São Paulo       Vendas Assistente   29.13 41.24  70.37  26.7996
  NovaColuna
1    1044.69
2    1070.37
```

Outras formas de se excluir variáveis são:

```
## Selecionar apenas as variáveis de interesse
head(rh[, c("Anos.de.estudo", "Unidade")], 2)
```

```
  Anos.de.estudo   Unidade
1             14  Curitiba
2             19 São Paulo
```

```
## Descartar as variáveis indesejadas
head(rh[, -c(1, 5)], 2)
```

```
  Estado.Civil Anos.de.estudo         Formação   Unidade Departamento
1       Casado             14 Sócio-econômicas  Curitiba     Produção
2        Viúvo             19 Sócio-econômicas São Paulo       Vendas
       Cargo Salário Bônus SalTot Desconto NovaColuna
1 Assistente   16.67 28.02  44.69  15.3364    1044.69
2 Assistente   29.13 41.24  70.37  26.7996    1070.37
```

4.3.2 Recodificação de variáveis

Às vezes é necessário corrigir os valores de variáveis em uma base de dados. No exemplo a seguir, a variável `Formação` será recodificada de forma que as inconsistências existentes sejam corrigidas.

```
unique(rh$Formação)
```

```
[1] "Sócio-econômicas" "Exatas"           "Humanas"
[4] "Biológicas"       "Exat "            "Huma "
[7] " "                "Sóci "            "Biol "
```

Para corrigir as inconsistências identificadas, pode-se proceder da seguinte forma:

```
rh$Formação[rh$Formação %in% c("Biológicas", "Biol ")] <- "BIO"
rh$Formação[rh$Formação %in% c("Exatas", "Exat ")] <- "EXA"
rh$Formação[rh$Formação %in% c("Humanas", "Huma ")] <- "HUM"
rh$Formação[rh$Formação %in% c("Sócio-econômicas", "Sóci ")] <- "SEC"
rh$Formação[rh$Formação == " "] <- NA

unique(rh$Formação)
```

```
[1] "SEC" "EXA" "HUM" "BIO" NA
```

4.3.3 Categorização de variáveis

É muito frequente a situação em que se deseja categorizar uma variável numérica. No R, isso pode ser feito com a função `cut()`. A variável `Salário` será categorizada em 4 classes de acordo com a faixa salarial, conforme mostrado a seguir:

- Junior: `]0, 6.17]`
- Pleno: `]6.17, 15]`
- Senior: `]15, Inf[`

```
summary(rh$Salário)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   1.00    3.39    8.44   12.84   17.52  151.50       8
```

```
rh <- transform(rh,
                ClasseSalarial = cut(Salário,
                                     breaks=c(0, 6.17, 15, Inf),
                                     labels=c("Junior", "Pleno", "Senior")))

head(rh, 2)
```

```
       Sexo Estado.Civil Anos.de.estudo Formação Tempo.de.empresa
1 Masculino       Casado             14      SEC               19
2 Masculino        Viúvo             19      SEC               31
    Unidade Departamento      Cargo Salário Bônus SalTot Desconto
1  Curitiba     Produção Assistente   16.67 28.02  44.69  15.3364
2 São Paulo       Vendas Assistente   29.13 41.24  70.37  26.7996
  NovaColuna ClasseSalarial
1    1044.69         Senior
2    1070.37         Senior
```

4.3.4 Filtro

Uma operação que se realiza com bastente frequência em um conjunto de dados é a aplicação de filtros; que consiste em selecionar os registros do conjunto de dados que atendam a um determinado critério.

No exemplo a seguir, será aplicado um filtro à base de dados com o objetivo de selecionar apenas os registros para os quais a variável `Anos.de.estudo` é maior ou igual a 17 e `Sexo` é igual a masculino.

```
unique(rh$Sexo)
```

```
[1] "Masculino" "Feminino"  NA
```

```
escolaridade <- subset(rh, Anos.de.estudo >= 17 & Sexo == "Masculino")
head(escolaridade)
```

```
        Sexo Estado.Civil Anos.de.estudo Formação Tempo.de.empresa
2  Masculino        Viúvo             19      SEC               31
6  Masculino       Casado             18      SEC               23
7  Masculino       Casado             18      EXA               27
12 Masculino       Casado             18      SEC               17
18 Masculino        Viúvo             19      EXA               14
25 Masculino       Casado             18      SEC               27
          Unidade Departamento      Cargo Salário Bônus SalTot Desconto
2       São Paulo       Vendas Assistente   29.13 41.24  70.37  26.7996
6       São Paulo      Pessoal Assistente    8.34 86.88  95.22   7.6728
7        Curitiba       Vendas   Auxiliar   11.15  8.80  19.95  10.2580
12 Rio de Janeiro       Vendas    Gerente   12.45 34.00  46.45  11.4540
18       Curitiba     Produção Assistente   11.28 28.02  39.30  10.3776
25 Rio de Janeiro       Vendas    Gerente   34.30 69.14 103.44  31.5560
   NovaColuna ClasseSalarial
2     1070.37         Senior
6     1095.22          Pleno
7     1019.95          Pleno
12    1046.45          Pleno
18    1039.30          Pleno
25    1103.44         Senior
```

4.3.5 Ordenação

A ordenação de um conjunto pode ser feita utilizando-se a função `order()`. A seguir será feita a ordenação do conjunto de dados em ordem crescente de `Salário`.

```
idx <- order(rh$Salário,  decreasing=TRUE)
rh <- rh[idx, ]
head(rh)
```

```
          Sexo Estado.Civil Anos.de.estudo Formação Tempo.de.empresa
3521 Masculino       Casado             25      SEC               38
409  Masculino       Casado             28      SEC               37
1863  Feminino       Casado             18      SEC               28
3697 Masculino       Casado             23      SEC               39
4267  Feminino       Casado             24      SEC               53
```

```
4166  Feminino       Casado             23      SEC               30
            Unidade Departamento           Cargo Salário  Bônus SalTot
3521 Rio de Janeiro       Vendas Vice-Presidente  151.48  90.10 241.58
409  Rio de Janeiro       Vendas Vice-Presidente  141.42 341.18 482.60
1863       Curitiba       Vendas         Gerente  127.47  77.10 204.57
3697 Rio de Janeiro       Vendas Vice-Presidente  110.23  76.50 186.73
4267 Rio de Janeiro       Vendas Vice-Presidente  109.10 251.12 360.22
4166 Rio de Janeiro       Vendas Vice-Presidente  103.55 225.02 328.57
     Desconto NovaColuna ClasseSalarial
3521 139.3616    1241.58         Senior
409  130.1064    1482.60         Senior
1863 117.2724    1204.57         Senior
3697 101.4116    1186.73         Senior
4267 100.3720    1360.22         Senior
4166  95.2660    1328.57         Senior
```

4.3.6 *Merging*

Para reunir em uma única base de dados campos de duas ou mais bases de dados, pode-se utilizar a função `merge()`. Com vistas a facilitar o entendimento de como a função opera, será utilizado um conjunto de dados bem pequeno, que será definido a seguir:

```r
Pdiv <- data.frame(Time=c("Flamengo", "Vasco", "Fluminense"),
                   Torcida=c(100, 50, 60))

Sdiv <- data.frame(Time=c("Vasco", "Fluminense", "Bangu"),
                   QtdTit = c(22, 33, 40))

Tdiv <- data.frame(Time = c("Vasco", "Bangu", "Olaria"),
                   Derrotas = c(30, 15, 14))
```

```r
Pdiv
```

```
        Time Torcida
1   Flamengo     100
2      Vasco      50
3 Fluminense      60
```

```r
Sdiv
```

```
        Time QtdTit
1      Vasco     22
2 Fluminense     33
3      Bangu     40
```

```r
Tdiv
```

```
    Time Derrotas
1  Vasco       30
2  Bangu       15
3 Olaria       14
```

```
merge(Pdiv, Sdiv, by="Time")
```

```
        Time Torcida QtdTit
1 Fluminense      60     33
2      Vasco      50     22
```

```
merge(Pdiv, Sdiv, by="Time", all.x=TRUE)
```

```
        Time Torcida QtdTit
1   Flamengo     100     NA
2 Fluminense      60     33
3      Vasco      50     22
```

```
merge(Pdiv, Sdiv, by="Time", all.y=TRUE)
```

```
        Time Torcida QtdTit
1 Fluminense      60     33
2      Vasco      50     22
3      Bangu      NA     40
```

```
merge(Pdiv, Sdiv, by="Time", all=TRUE)
```

```
        Time Torcida QtdTit
1   Flamengo     100     NA
2 Fluminense      60     33
3      Vasco      50     22
4      Bangu      NA     40
```

Feita esta pequena ilustração do funcionamento da função `merge()`, será feita a seguir sua aplicação utilizando-se os conjuntos de dados `Arfile.ASC` e `Address.ASC`.

Estes dois conjuntos de dados são arquivos texto de formato fixo e, dessa forma, sua importação depende de conhecermos o *layout* dos arquivos, o que consta do documento `Descricao Arquivos Dados_v2.doc` que acompanha os conjuntos de dados utilizados neste curso.

Com base nas informações contidas naquele documento, a importação dos dados pode ser feita da seguinte forma:

```
setwd(diretorio)

contas_receber <- read.fwf('Arfile.ASC',
                           widths = c(11, 4, 4, 15, 8),
                           col.names = c('account', 'division', 'store', 'balance', 'duedate'),

endereco <- read.fwf('Address.ASC',
                     widths = c(11, 33, 33, 30, 25, 5),
                     col.names = c('account', 'name1', 'name2', 'street', 'cityst', 'zip'),
                     comment.char='',
                     strip.white = TRUE)
```

A operação de *merging* depende de que as bases de dados a serem reunidas tenham ao menos um campo em comum.

Nas duas bases de dados importadas, o campo `account`, que identifica cada cliente, é comum às duas bases de dados e pode ser utilizada como `chave` para a execução desta operação.

```
nova_base <- merge(contas_receber, endereco)
head(nova_base, 3)
```

```
      account division store balance  duedate
1 S0000000003       28     5 9609.75 20010227
2 S0000000036       28    40  120.98 20011129
3 S0000000140       28     8 3092.04 20011226
                            name1                              name2
1 ITF CHURCH CEMETERY PERPETUITY AS TRUSTEES BERKSHIRE VALLEY PRESB.
2                      JONES/SHEPARD
3           LESLIE E OR ROY S LARSEN
             street           cityst   zip
1   10 HOLLY DRIVE     DENVILLE, AZ 72134
2       81 MAIN ST HACKETTSTOWN, AZ 72140
3 3 ROSEVILLE ROAD     STANHOPE, AZ 72174
```

4.3.7 Cruzamento de dados

O que é normalmente denominado de `cruzamento de dados` consiste em identificar quais valores de uma variável constam de uma outra variável uma outra base de dados.

Retomando o exemplo dos times de futebol, uma questão de interesse poderia ser: quais times da 1a divisão também estão na 2a?. Isto pode ser feito da seguinte forma:

```
Pdiv <- transform(Pdiv, EstaNa2aDiv = ifelse(Time %in% Sdiv$Time, 1, 0))
Pdiv
```

```
        Time Torcida EstaNa2aDiv
1   Flamengo     100           0
2      Vasco      50           1
3 Fluminense      60           1
```

No exemplo a seguir utilizaremos os conjuntos de dados `contratos.csv` e `receitas_candidatos_2014_RJ.txt` com o objetivo de verificar se, dentre as empresas contratadas por orgãos estaduais, alguma figura como doadora de campanha.

```
setwd(diretorio)

## Importa a a base de contratos
contratos <- read.csv2('contratos.csv', as.is=TRUE)
names(contratos) <- gsub('\\.', '', names(contratos))
contratos$Contratado <- gsub('[[:punct:]]', '', contratos$Contratado)

## Importa a base de doadores de campanha
doadores <- read.csv2('receitas_candidatos_2014_RJ.txt', as.is=TRUE, na.strings = '#NULO')
names(doadores) <- gsub('\\.', '', names(doadores))
```

```
## Cruzamento dos dados - marcar os contratos onde os fornecedores são doadores de campanha
contratos <- transform(contratos,
                       EhDoador = ifelse(Contratado %in% doadores$CPFCNPJdodoador, 1, 0))

contratos_com_doadores_campanha <- subset(contratos, EhDoador == 1)

## Obter a lista de contratados doadores de campanha
lista_doadores <- subset(contratos_com_doadores_campanha,
                         !duplicated(Contratado),
                         select=c('Contratado', 'DescriçãoContratado'))

nrow(lista_doadores)
```

```
[1] 62
```

4.3.8 *Reshaping*

As vezes os conjuntos de dados que se tem à mão precisam ter sua estrutura modificada para seja possível realizar a análise de interesse ou para se passar o conjunto de dados como argumento de uma função.

Especificamente considera-se o caso de passar os dados do `formato longo` para o `formato amplo` e vice-versa.

Para mais informações sobre a organização de dados, sugere-se a leitura do artigo `**Tidy Data**` do Hadley Wickham que consta do material de leitura disponibilizado no Moodle. Este artigo também pode ser baixado aqui.

No R, a função básica para realizar esta tarefa é a função `reshape()`. Porém o uso desta função é um pouco complicado, visto que envolve a especificação de diversos argumentos de forma não muito intuitiva.

Assim, será utilizado o pacote `reshape2` que facilita bastante a realização do procedimento de *reshaping*. Este pacote disponibiliza as funções `melt()` e `dcast()` que possibilitam, respectivamente, a conversão do formato amplo para o longo, e do longo para o amplo.

O exemplo a seguir ilustra como passar um conjundo de dados do formato amplo para longo. Será utilizado o conjunto de dados `IDH1991_2000.csv`:

```
library(reshape2)
setwd(diretorio)
idh <- read.csv2('IDH1991_2000.csv', as.is = TRUE)
head(idh, 2)
```

```
     Cod                 Mun   idh2000 idhRenda2000 idhEduc2000
1 330010 Angra dos Reis (RJ) 0.7887205    0.7105522   0.8689171
2 330015        Aperibé (RJ) 0.7525647    0.6607060   0.8665584
  idhLongev2000   idh1991 idhRenda1991 idhEduc1991 idhLongev1991
1     0.7866921 0.7208758    0.6583721   0.7977447     0.7065105
2     0.7304296 0.6762482    0.5728085   0.7560621     0.6998742
```

```
idh2 <- melt(idh, id.vars=c('Cod', 'Mun'), variable.name='IDH', value.name = 'VlrIDH')
head(idh2, 10)
```

```
      Cod                    Mun     IDH    VlrIDH
1  330010    Angra dos Reis (RJ) idh2000 0.7887205
2  330015           Aperibé (RJ) idh2000 0.7525647
```

```
3  330020          Araruama (RJ) idh2000 0.7519513
4  330022             Areal (RJ) idh2000 0.7750303
5  330023 Armação de Búzios (RJ) idh2000 0.7703591
6  330025   Arraial do Cabo (RJ) idh2000 0.7900429
7  330030    Barra do Piraí (RJ) idh2000 0.7900376
8  330040       Barra Mansa (RJ) idh2000 0.7998771
9  330045      Belford Roxo (RJ) idh2000 0.7432436
10 330050        Bom Jardim (RJ) idh2000 0.7323627
```

O exemplo a seguir fará o oposto. O conjunto de dados está no formato longo e será modificado para o formato amplo. Será utilizado o conjunto de dados `Receita_Municipios_RJ_2013.txt`

Como visto, a função `dcast()` permite a realização desta tarefa. Exemplo:

```
setwd(diretorio)
receita <- read.table('Receita_Municipios_RJ_2013.txt', header=TRUE, sep='\t', as.is = TRUE)
names(receita) <- c('municipio', 'codigo', 'descricao', 'vlr_arrecadado')

## Transformando os dados...
receita2 <- dcast(receita, codigo + descricao ~ municipio,
                  value.var = 'vlr_arrecadado')

receita2[1:10, c(1,3:5)]
```

```
     codigo ANGRA DOS REIS   APERIBE   ARARUAMA
1  11120101             NA        NA         NA
2  11120200    37362220.82 165978.48 14490195.4
3  11120431    31754253.63 223199.35   777812.6
4  11120434     1571335.29        NA   638080.2
5  11120800     8645431.69  92110.28  4535156.7
6  11130500    78735541.84 504547.26 10512722.7
7  11210000       14484.72  92181.55  1574512.4
8  11220000      620542.61  46579.93  7517184.3
9  11300000             NA        NA         NA
10 12102901       15241.69        NA         NA
```

Outras funções de interesse são: `stack()` e `unstack()`.

4.3.9 Tabulação cruzada

A tabulação cruzada consiste normalmente em se determinar a distribuição de freqüências com referência a uma, duas ou mais variáveis categóricas. Para executar esta tarefa o R dispõe das funções `table()` e `xtabs()`. Será utilizado o conjunto de dados `rh` para ilustrar o procedimento.

Caso se queira calcular a quantidade de funcionários em cada categoria da variável `Formação2`:

```
with(rh, table(Formação))
```

```
Formação
 BIO  EXA  HUM  SEC
 553 1174  971 2292
```

Caso fosse necessário saber a quantidade de funcionários por sexo e estado civil:

```
with(rh, table(Sexo, Estado.Civil))
```

```
           Estado.Civil
Sexo        Casado Divorciado Solteiro Viúvo
  Feminino    1367        181      371   135
  Masculino   2080        238      419   198
```

Caso o interesse esteja em somar os valores de uma varíável quantitativa com base nos valoes de uma ou mais variáveis categóricas, pode-se utilizar a função `xtabs()`.

Por exemplo, suponha que se deseja obter o total dos salários pagos por Unidade e, também, o total dos salários pagos por Unidade e Departamento:

```
## Total dos salários pagos por Unidade
xtabs(Salário ~ Unidade , data=rh)
```

```
Unidade
      Curitiba  Florianópolis Rio de Janeiro      São Paulo
      11126.91        2687.78       38559.46       11640.33
```

```
## Total dos salários pagos por Unidade e Departamento
xtabs(Salário ~ Unidade + Departamento , data=rh)
```

```
               Departamento
Unidade          Financeiro   Pessoal  Produção    Vendas
  Curitiba         1789.820  1064.105  1804.130  6414.560
  Florianópolis     474.950   355.260   811.345  1046.225
  Rio de Janeiro   3190.325  1946.480  1948.685 31383.210
  São Paulo        1173.500  1004.420  1405.980  8053.580
```

```
## Total dos salários e Bônus pagos por Unidade
xtabs(cbind(Salário, Bônus) ~ Unidade, data=rh)
```

```
Unidade           Salário    Bônus
  Curitiba       11098.98 24356.94
  Florianópolis   2683.68  6974.62
  Rio de Janeiro 38559.46 73121.81
  São Paulo      11640.33 24252.31
```

A função `xtabs()` também pode ser utilizada para se obter a distribuição de frequências, da mesma forma como a função `table()`. Exemplo:

```
xtabs( ~ Estado.Civil, data=rh)
```

```
Estado.Civil
    Casado Divorciado   Solteiro      Viúvo
      3449        419        790        333
```

Às vezes pode ser conveniente adicionar à tabela as totalizações de linha e coluna. A função `addmargins()` pode ser utilizada com esta finalidade.

Também pode ser de interesse obter uma tabela com as proporções de observações em cada célula. Isto pode ser feito com a função `prop.table()`.

Fica como tarefa ler a ajuda destas funções para aprender o seu funcionamento.

4.3.10 Sumarização de dados

A sumarização de dados, ou agregação, consiste em calcular uma medida resumo, usuamente a média ou total, de acordo com os valores de uma variável qualitativa. No exemplo que se segue, utiliza-se o conjunto de dados `rh` para calcular a média de salários, bônus, anos de estudo e tempo de empresa por sexo.

```
aggregate(rh[, c("Salário", "Bônus", "Anos.de.estudo", "Tempo.de.empresa")],
          by=list(Sexo=rh$Sexo),
          FUN=mean, na.rm=TRUE)
```

```
       Sexo   Salário    Bônus Anos.de.estudo Tempo.de.empresa
1  Feminino  8.685849 19.69167       14.15459         13.74562
2 Masculino 15.748613 30.13476       15.58288         16.67382
```

No exemplo acima utilizou-se uma função pré-definida do R (`mean()`) para se realizar a agregação. Mas é possível também utilizar funções definidas pelo usuário. No exemplo a seguir, será definida uma função chamada `estdesc()` que calcula algumas estatísticas descritivas para um vetor numérico. Depois esta função será utilizada para sumarizar a base de dados.

```
## Definição da função estatdesc
estdesc <- function(x, na.rm=TRUE){
  media  <- mean(x, na.rm=na.rm)
  maximo <- max(x, na.rm=na.rm)
  minimo <- min(x, na.rm=na.rm)
  return(c(Media=media, Max=maximo, Min=minimo))
}

## Definição das variáveis a serem utilizadas
colunas <- names(rh)[c(9, 10)]

## Cálculo da agregação
ed <-  aggregate(rh[, colunas],
                 by=list(Sexo=rh$Sexo, ECivil=rh$Estado.Civil),
                 FUN=estdesc)

ed
```

```
       Sexo     ECivil Salário.Media Salário.Max Salário.Min Bônus.Media
1  Feminino     Casado     10.859993  127.470000    1.020000   24.540300
2 Masculino     Casado     18.682147  151.480000    1.020000   35.960944
3  Feminino Divorciado      4.754088   27.220000    1.000000   11.790276
4 Masculino Divorciado     10.300903   44.720000    1.000000   18.455798
5  Feminino   Solteiro      3.532197   30.630000    1.000000    8.029191
6 Masculino   Solteiro      7.041098   60.080000    1.000000   12.932363
7  Feminino      Viúvo      6.267687   45.370000    1.020000   13.643704
```

```
8 Masculino      Viúvo      9.905888   77.470000    1.140000   18.927475
   Bônus.Max  Bônus.Min
1 251.120000   2.000000
2 341.180000   2.000000
3  49.480000   2.000000
4 120.160000   2.120000
5 112.680000   2.000000
6 160.480000   2.000000
7 120.160000   2.000000
8 111.740000   2.060000
```

4.3.11 Sorteio de amostras aleatórias

Uma operação comum em trabalhos de auditoria é a seleção de amostras. No R esta operação pode ser realizada com a função `sample()`. No exemplo a seguir, será selecionada uma amostra aleatória simples sem reposição de 30 elementos do conjunto de dados `rh`.

```
set.seed(1)
linhas_sorteadas <- sample(row.names(rh), 30)
amostra <- rh[linhas_sorteadas,]
dim(amostra)
```

```
[1] 30 14
```

4.3.12 Duplicidades

Às vezes é necessário testar se existe duplicidade em valores de uma variável. Por exemplo, ao se examinar uma relação de empenhos, não se espera encontrar o mesmo número de empenho emitido mais de uma vez.

A identificação de duplicidades será ilustrada com o arquivo `Invoices.csv`. O objetivo é identificar eventuais faturas emitidas em duplicidade.

```
setwd(diretorio)
faturamento <- read.csv2("Invoices.csv")

## Identifica eventuais faturas duplicadass
repetidos <- faturamento$InvoiceNo[duplicated(faturamento$InvoiceNo)]
repetidos
```

```
[1] 20010
```

```
subset(faturamento, InvoiceNo %in% repetidos)
```

```
         Date InvoiceNo CustomerNo SalesPerson ProductNo UnitPrice
11 14/01/2003     20010      10439          19        38      7.45
12 14/01/2003     20010      10439          99        38      7.45
   Quantity Amount
11       28  208.6
12       28  208.6
```

4.3.13 Apensar bases de dados

Às vezes pode ser necessário reunir em um único conjunto de dados um ou mais conjuntos de dados. Usualmente o que se deseja é 'colocar uma base de dados embaixo da outra'. Isso pode ser feito no R com a função `rbind()` desde que as bases de dados contenham as mesmas variáveis, nas mesmas posições.

Para ilustrar este procedimento serão utilizados os conjuntos de dados `Trans_Abril.xls` e `Trans_Maio.xls`.

```
library(readxl)
setwd(diretorio)

abril <- read_excel('Trans_Abril.xls')
maio1 <- read_excel('Trans_Maio.xls', sheet='Trans1_Maio')
maio2 <- read_excel('Trans_Maio.xls', sheet='Trans2_Maio')
```

```
dim(abril)
```

```
[1] 285   6
```

```
dim(maio1)
```

```
[1] 86  6
```

```
dim(maio2)
```

```
[1] 114   6
```

```
head(abril, 2)
```

```
  NÃºmcartÃ£o  Valor Data_Trans CÃ³digos NÃºmclien
1 8.59012e+15 270.63 2003-04-02     1731    001000
2 8.59012e+15 899.76 2003-04-02     1731    002000
                DescriÃ§Ã£o
1 Contratos de eletricidade
2 Contratos de eletricidade
```

```
head(maio1, 2)
```

```
          NÃºmcartÃ£o CÃ³digos Data_Trans NÃºmclien
1 8590 1252 7244 7003     4131 2003-05-27    925007
2    8590128346463420     4214 2003-05-28    051593
                                          DescriÃ§Ã£o   Valor
1 Linhas de ônibus, incluindo charters, ônibus de turismo $108.01
2                           Serviços de entrega - Local  $71.57
```

```
head(maio2, 2)
```

```
          NÃºmcartÃ£o CÃ³digos Data_Trans NÃºmclien
1 8590-1224-9766-3807     2741 2003-05-04    962353
2    8590122281964011     5021 2003-05-01    812465
                      DescriÃ§Ã£o   Valor
1 Publicações e impressões diversas $510.43
2 Móveis de escritório e comerciais $178.96
```

```
dados <- rbind(abril[, c(1, 4, 3, 5, 6, 2)],    # reordenamento das colunas...
               maio1,
               maio2)

## Correção de encoding nos nomes das variáveis
names(dados) <- iconv(names(dados), from='utf-8', to='latin1')
head(dados)
```

```
         Númcartão Códigos Data_Trans Númclien
1 8590120032047834    1731 2003-04-02   001000
2 8590120092563655    1731 2003-04-02   002000
3 8590120233319873    1750 2003-04-04   250402
4 8590120534914664    1750 2003-04-08   003000
5 8590120674263418    2741 2003-04-08   001000
6 8590120716753180    2741 2003-04-15   002000
                          Descrição  Valor
1        Contratos de eletricidade  270.63
2        Contratos de eletricidade  899.76
3          Contratos de carpintaria 730.46
4          Contratos de carpintaria 106.01
5 Publicações e impressões diversas 309.37
6 Publicações e impressões diversas 534.14
```

```
dim(dados)
```

```
[1] 485   6
```

4.3.14 Família apply

O que usualmente se denomina `família apply` consiste em um conjunto de funções cujos integrantes mais conhecidos são: `apply()`, `tapply()`, `lapply()`, `sapply()` e `mapply()`.

A forma de utilização de cada uma delas será apresentada seguir.

4.3.14.1 apply()

A função `apply()` é utilizada para aplicar uma função às linhas ou colunas de um `data frame` ou `matriz` ou `array`. No exemplo a seguir, calcula-se a média dos salários e bônus dos funcionários.

```
media_salarial <- apply(rh[,c("Salário", "Bônus")], 2, mean, na.rm=TRUE)
media_salarial
```

```
 Salário    Bônus
12.83608 25.83507
```

É possível substituir as colunas de um data frame por valores modificados. No exemplo a seguir, as colunas `Sexo`, `Estado.Civil` e `Formação` serão substituídas por seus valoes em caixa alta.

```
rh[,c("Sexo", "Estado.Civil", "Formação")] <- apply(rh[, c("Sexo", "Estado.Civil", "Formação")], 2, tou

head(rh)
```

```
          Sexo Estado.Civil Anos.de.estudo Formação Tempo.de.empresa
3521 MASCULINO       CASADO             25      SEC               38
409  MASCULINO       CASADO             28      SEC               37
1863  FEMININO       CASADO             18      SEC               28
3697 MASCULINO       CASADO             23      SEC               39
4267  FEMININO       CASADO             24      SEC               53
4166  FEMININO       CASADO             23      SEC               30
            Unidade Departamento           Cargo Salário  Bônus SalTot
3521 Rio de Janeiro       Vendas Vice-Presidente  151.48  90.10 241.58
409  Rio de Janeiro       Vendas Vice-Presidente  141.42 341.18 482.60
1863       Curitiba       Vendas         Gerente  127.47  77.10 204.57
3697 Rio de Janeiro       Vendas Vice-Presidente  110.23  76.50 186.73
4267 Rio de Janeiro       Vendas Vice-Presidente  109.10 251.12 360.22
4166 Rio de Janeiro       Vendas Vice-Presidente  103.55 225.02 328.57
     Desconto NovaColuna ClasseSalarial
3521 139.3616    1241.58         Senior
409  130.1064    1482.60         Senior
1863 117.2724    1204.57         Senior
3697 101.4116    1186.73         Senior
4267 100.3720    1360.22         Senior
4166  95.2660    1328.57         Senior
```

Estas funções aceitam também funções definidas pelo usuários ou as denominadas `funções anônimas`, que são definidas dentro das funções `apply()`.

4.3.14.2 tapply()

Aplica uma função em um grupo de observações definidas pelos valores de uma variável qualitativa. No exemplo a seguir, será calculada a soma dos anos de estudo por Unidade e depois a soma dos anos de estudo por Sexo e Unidade.

```
## Soma dos anos de estudo por unidade
anos_estudo_unidade <- tapply(rh[["Anos.de.estudo"]], list(rh[["Unidade"]]), sum, na.rm=TRUE)
anos_estudo_unidade
```

```
      Curitiba  Florianópolis Rio de Janeiro      São Paulo
         18363           7392          34278          14729
```

```
## Soma dos anos de estudo por Sexo e Unidade
anos_estudo_sexo_unid <- tapply(rh[["Anos.de.estudo"]], list(rh$Sexo, rh$Unidade), sum, na.rm=TRUE)
anos_estudo_sexo_unid
```

```
          Curitiba Florianópolis Rio de Janeiro São Paulo
FEMININO      7896          4157          11419      5625
MASCULINO    10467          3235          22833      9104
```

4.3.14.3 lapply()

Esta função aplica uma função aos elementos de uma `lista` ou `vetor`. O resultado desta função é sempre uma lista. Exemplo:

```
## Definição de uma lista.
rr <- list(a=c(1, 5, 9), b=c(10, 15, 25, 48, 72, 29), c=c(12, 12))

## Extrai os elementos da segunda posição em cada componente da lista
lapply(rr, "[", 2)
```

```
$a
[1] 5

$b
[1] 15

$c
[1] 12
```

```
## Aplicar a função range() a cada componente da lista e convertendo para data frame
qq <- as.data.frame(lapply(rr, range))
qq
```

```
  a  b  c
1 1 10 12
2 9 72 12
```

```
row.names(qq) <- c("Min", "Max")
qq
```

```
    a  b  c
Min 1 10 12
Max 9 72 12
```

4.3.14.4 sapply()

A função `sapply()` é uma versão da função `lapply()` no sentido de que ela tenta retornar um vetor ou matriz, em vez de lista. Seu uso é igual ao da função `lapply()`:

```
maximo <- sapply(rh[, c("Salário", "Bônus")], max, na.rm=TRUE)
maximo
```

```
Salário   Bônus
 151.48  341.18
```

4.3.14.5 mapply()

A função `mapply()` é uma versão multivariada da função `sapply()` e é utilizada quando se deseja aplicar uma função a cada componente de multiplas listas ou vetores.

Dependendo da função a ser utilizada e da quantidade de listas passadas à funçao, este tomará os elementos das listas como argumentos.

Os exemplos a seguir ilustram sua utilização.

```
mapply(rep, 1:4, 4:1)
```

```
[[1]]
[1] 1 1 1 1

[[2]]
[1] 2 2 2

[[3]]
[1] 3 3

[[4]]
[1] 4
```

```
mapply(rep, times = 1:4, MoreArgs = list(x = 42))
```

```
[[1]]
[1] 42

[[2]]
[1] 42 42

[[3]]
[1] 42 42 42

[[4]]
[1] 42 42 42 42
```

Outros membros da família são: `rapply()`, `eapply()` e `vapply()`

4.3.15 Lei de Benford

O uso da Lei de Benford será ilustrado usando o conjunto de dados `despesas_candidatos_2014_RJ.txt` para examinar as despesas declaradas por um determinado candidato a deputado estadual. Esta técnica é utilizada para verificar a possibilidade de que as despesas informadas não sejam legítimas.

O pacote `benford.analysis` disponibiliza funções que nos permitem realizar esta análise.

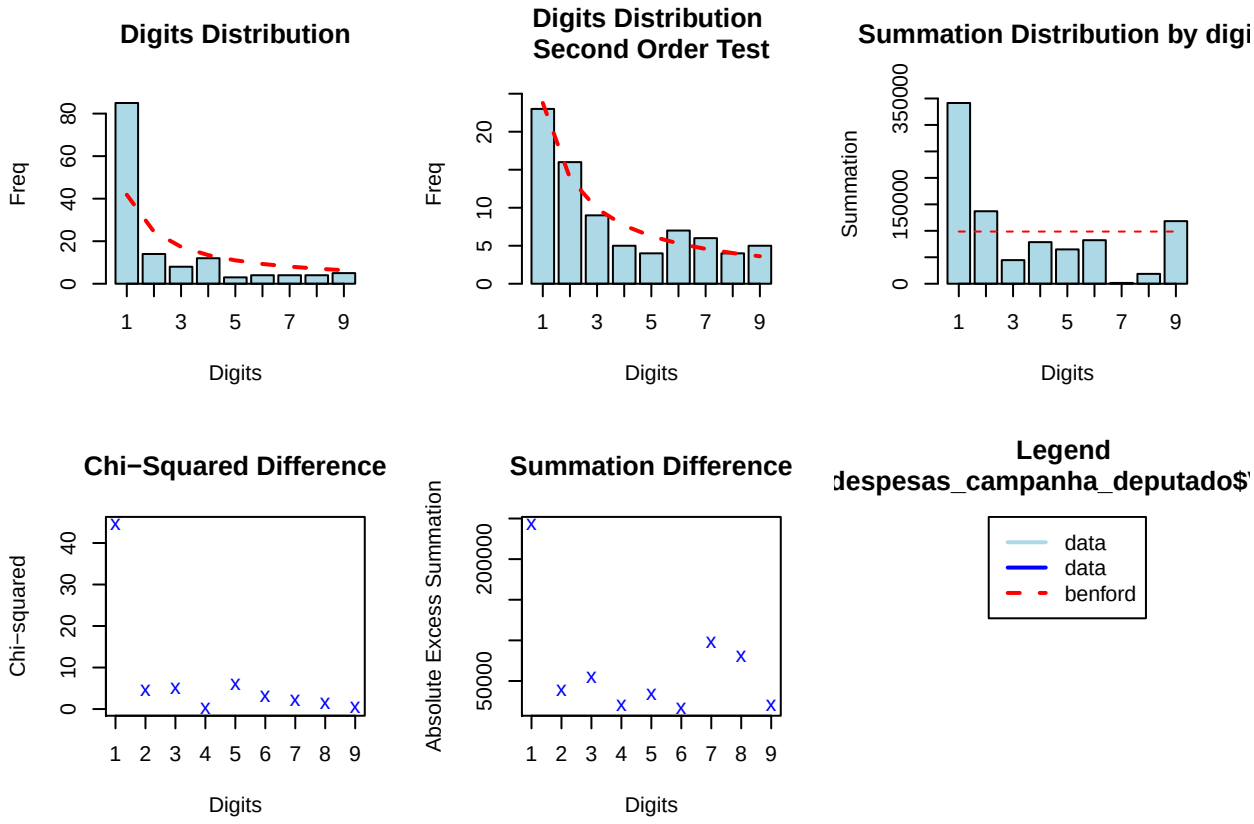
Uma vez instalado este pacote, sua utilização é feita da seguinte forma:

```
library(benford.analysis)
setwd(diretorio)

# Importação dos dados
despesas_campanha <- read.csv2("despesas_candidatos_2014_RJ.txt", as.is = TRUE)

## Filtrar candidados a deputado estadual
despesas_campanha_deputado <- subset(despesas_campanha,
                                     Cargo == "Deputado Estadual" &
                                     Nome.candidato == "SAMUEL LIMA MALAFAIA")

## Análise dos primeiros dígitos
```

```r
benf_despesas <- benford(despesas_campanha_deputado$Valor.despesa,
                         number.of.digits = 1,
                         discrete = FALSE)

benf_despesas
```

```
Benford object:

Data: despesas_campanha_deputado$Valor.despesa
Number of observations used = 139
Number of obs. for second order = 79
First digits analysed = 1

Mantissa:

[1]  0.35  0.08 -0.41  0.89

The 5 largest deviations:

  digits absolute.diff
1      1         43.16
2      2         10.48
3      3          9.37
4      5          8.01
5      6          5.31

Stats:

    Pearson's Chi-squared test

data:  despesas_campanha_deputado$Valor.despesa
X-squared = 66.754, df = 8, p-value = 2.169e-11


    Mantissa Arc Test

data:  despesas_campanha_deputado$Valor.despesa
L2 = 0.16599, df = 2, p-value = 9.549e-11

Mean Absolute Deviation: 0.06899573
Distortion Factor: -27.41896

Remember: Real data will never conform perfectly to Benford's Law. You should not focus on p-values!
```

```r
# Gráfico
plot(benf_despesas)
```



4.3.16 Fuzzy matching

Em diversas situações faz-se necessário identificar *strings* que sejam semelhantes entre si; e uma das formas de se executar esta tarefa é com a utilização da técnica de `fuzzy matching`, que consiste na aplicação de algoritmos que conseguem determinar o grau de semelhança entre *strings*.

As funções nativas do R que podem ser utilizadas para esta finalidade são `agrep()` e `adist()`.

O pacote `stringdist` fornece um conjunto adicional de funções para o cálculo da distância entre *strings*. A vantagem de se utilizar este pacote é a possibilidade de escolher o algoritmo de comparação de *strings* a ser utilizado.

No exemplo a seguir, será utilizado o conjunto de dados `Address.ASC` com o objetivo de tentar identificar endereços que, embora sejam o mesmo, tenham sido grafados de formas diferentes.

```
library(stringdist)
setwd(diretorio)

endereco <- read.fwf('Address.ASC',
                     widths = c(11, 33, 33, 30, 25, 5),
                     col.names = c('account', 'name1', 'name2', 'street', 'cityst', 'zip'),
                     comment.char='',
                     strip.white = TRUE, as.is=TRUE)

## Obtenção do vetor de endereços
end <- endereco$street
```

```r
## Exclusão de dígitos
end <- gsub('(.*?)\\d+(.*?)', '\\1\\2', end)

## Comparação
end <- unique(end)
end2 <- expand.grid(end, end, stringsAsFactors = FALSE)

## Calculo da distância
end2 <- transform(end2, dist = stringdist(end2[,1], end2[,2]))

## Exibir os dez primeiros.
head(subset(end2, dist != 0 & dist <= 2), 10)
```

```
                  Var1               Var2 dist
11127       COLBY DRIVE        HOLLY DRIVE    2
18137   EAST SHORE ROAD    WEST SHORE ROAD    2
20854         MAYNE AVE          MAPLE AVE    2
23075            JAY ST             OAK ST    2
23247          OAK ST.              OAK ST    1
24492        JAY STREET         OAK STREET    2
27577     VREELAND AVE.       VREELAND AVE    1
28898     WOODSIDE AVE.       WOODSIDE AVE    1
32261   MT PLEASANT AVE  MT. PLEASANT AVE.    2
32812  MT. PLEASANT AVE  MT. PLEASANT AVE.    1
```

5 - EXERCÍCIOS

1. Importe o conjunto de dados `RH.csv`. Corrija os problemas existentes nas variáveis `Sexo` e `Unidade` (mesmo valor escrito de formas distintas). Calcule quantos funcionários existem de cada sexo.

2. Categorize a variável `Tempo.de.empresa` da seguinte forma:

- `JUNIOR`: de 0 a 5 anos de empresa
- `PLENO`: de 5 a 20 anos de empresa
- `SENIOR`: mais de 20 anos de empresa

Calcule quantos funcionários existem em cada categoria.
Calcule quantos funcionários existem em cada categoria por sexo.

3. Interrogue o arquivo `despesas_candidatos_2014_RJ.txt` e obtenha as respostas para as seguintes questões:

(a) Quantas observações tem a base de dados?

(b) Quais são as variáveis desta base de dados?

(c) Quanto cada candidato a deputado estadual gastou?

(d) Quanto cada partido gastou?

(e) Para cada cargo, quantos candidatos existem na base de dados?

(f) Quantos candidatos cada partido teve para cada um dos cargos?

(g) Qual o tipo de despesa mais frequente?

(h) Com que tipo de despesa houve mais gasto?

(i) Em média, quanto foi gasto por cada candidato para cada um dos cargos? Liste apenas os 10 maiores valores médios.

(j) Os valores das despesas seguem a lei de benford?

(k) Quanto foi gasto em campanha para cada cargo?

(l) Quantos partidos existem na base de dados?

4. Aplique um filtro ao conjunto de dados `rh` de forma a obter os registros que atendam ao seguinte critério: funcionários do sexo Feminino que trabalhem na Unidade do Rio de Janeiro com tempo de serviço entre 5 e 15 anos inclusive.

5. Use a função `lapply()` para calcular a soma dos últimos elementos das componentes da lista definida como:

```
lista1 <- list(c(15, 29, 36),
               c(75, 13.75, 19, 76, 127, 48),
               c(12.2, 23, 79, 107),
               c(43))
```

Dica: após obter a lista contendo os últimos elementos dos vetores que compõem a lista, utilize as funções `unlist()` e `sum()`.

6. Utilizando o conjunto de dados `rh`, sorteie uma amostra aleatória simples de 50 observações sem reposição de forma que 25 observações sejam de funcionários do sexo feminino e 25 do sexo masculino (estratificação por sexo).

Repita este procedimento para sortear uma amostra aleatória simples de forma que ela contenha 10% das observações de cada Unidade.

7. Importe o conjunto de dados `despesas_candidatos_2014_RJ.txt` e aplique um filtro de forma que o conjunto de dados resultante contenha apenas candidatos a deputado estadual cuja quantidade de registros na base seja superior ou igual a 500.

8. Após extraír os arquivos contidos em `specdata.zip`, em um diretório, reúna os arquivos em uma única base de dados. Quantos registros tem a base de dados resultante?

Dicas: (1) Utilize a função `list.files()` para obter um vetor com os nomes dos arquivos. **(2)** Utilize a função `lapply()` para aplicar a função `read.csv()` aos elementos do vetor criado em (1). A conversão da lista resultante em um data frame pode ser feita da seguinte forma: `df <- do.call("rbind", lista)` onde `lista` é a lista resultante da execução da dica (2) e `df` é o nome do data frame que se deseja criar.

9. O arquivo `cobertura de vacina.csv` foi obtido a partir de tabulação realizada no DATASUS e contém dados de cobertura de vacinação para os municípios do ERJ para os anos de 2010, 2011 e 2012. A partir da coluna `Municípios` crie duas novas colunas contendo o código do município e o nome do município.

10. Importe o conjunto de dados `Arfile.ASC` e converta a coluna `DUEDATE` para o formato de data.

11. Para o conjunto de dados importado acima, crie uma nova coluna chamada `TRIM` contendo informação sobre a que trimestre pertence o valor devido. Calcule o total devido por trimeste. Agora calcule o valor acumulado por trimestre.

12. Ainda usando a base dados acima, crie um novo campo denominado `DIAS_ATRASO` contendo a quantidade de dias decorridos entre a data de vencimento (variável `DUEDATE`) e a data de `31/12/2001`, data do encerramento do exercício. Crie mais uma variável, com o nome `AGE`, contendo a categorização da vaiável `DIAS_ATRASO` da seguinte forma:

- "0-30" - se o valor de `DIAS_ATRASO` for inferior ou igual a 30 dias
- "30-60" - se o valor de `DIAS_ATRASO` for superior a 30 dias e inferior a 60 dias
- "60-90" - se o valor de `DIAS_ATRASO` for superior a 60 dias e inferior a 90 dias
- "> 90" - se o valor de `DIAS_ATRASO` for superior a 90 dias

Qual o valor devido em cada uma das categorias criadas?

13. Importe o conjunto de dados `ucad_cap13_dados_crus.csv` e faça a sua limpeza, que consistirá em remover as strings do tipo `(ID 127)` presentes nos sobrenomes e a string `^` que antecede o nome.

14. Faça com que o vetor `nomes <- c('jOÃo', 'MarIA', 'caRLoS', 'fErNanDo', 'ClAuDio', 'frAnCIScO', 'RoDRigO')` apresente apenas o primeiro e último caracteres em caixa alta.

15. Importe o conjunto de dados `contratos.csv` e aplique um filtro, utilizando apenas o campo `Objeto Resumido` de forma a identificar, caso existam, os contratos relativos a fornecimento de quentinhas.

CAPÍTULO 5 - GRÁFICOS ESTATÍSTICOS

5.1 - CONSIDERAÇÕES GERAIS SOBRE A ELABORAÇÃO DE GRÁFICOS

Gráficos devem ser utilizados com o objetivo de comunicar informação, não devendo ser elaborados de forma a distrair ou enganar o leitor.

Não é boa prática elaborar gráficos com efeito tridimensional que acabam por distorcer a informação que se deseja comunicar.

Durante uma análise exploratória, alguns gráficos são produzidos meramente com o objetivo de explorar os dados. São produzidos diversos gráficos até que se obtenha alguns que de fato se mostrem mais adequados para os dados em análise.

Outras vezes, o objetivo é produzir gráficos que irão compor relatórios, os quais terão um acabamento mais bem elalaborado.

Assim, distingue-se entre gráficos utilizados principalmente para fins exploratórios e gráficos para apresentação.

5.2 - PARÂMETROS GRÁFICOS

Parâmetros gráficos são argumentos que permitem alterar diversos aspectos dos gráficos produzidos com o sistema básico de gráficos do R, tais como o símbolo usado para marcar pontos em gráficos, o tipo da linha, a cor, o tamanho do gráfico, o tamanho das margens, etc.

Estes parâmetros podem ser modificados utilizando a função `par()` ou como agumentos das funções que produzem gráficos. Apresentaremos a seguir os principais parâmetros gráficos. A seguir, apresenta-se um elenco de parâmetros gráficos e uma rápida descrição de sua função.

5.2.1 Simbolos e linhas

Parâmetro	Descrição
`pch`	Especifica o tipo de marcador a ser utilizado para representar pontos nos gráficos
`cex`	Especifica o tamanho do ponto relativamente ao tamanho padrão. 1='default', 1.5='50% maior', 0.5='metade do default', etc.
`lty`	Especifica o tipo de linha a ser desenhada. Assume valores de 1 a 6
`lwd`	Especifica a espessura da linha. Os valores são relativos ao default 1

Os códigos para os marcadores são apresentados a seguir:

Figure 2:

5.2.2 Cores

Parâmetro	Descrição
`col`	Especifica as cores a serem utilizadas no gráfico. Aceita valores de cores especificados por indice, hexadecimal, valor rgb e hsv.
`col.axis`	Especifica as cores dos eixos
`col.lab`	Especifica as cores dos labels dos eixos
`col.main`	Especifica as cores para os títulos
`col.sub`	Especifica as cores para os subtítulos
`fg`	Especifica a cor para o foreground do gráfico
`bg`	Especifica a cor para o background do gráfico

5.2.3 Elementos textuais

Parâmetro	Descrição
`cex`	Especifica o tamanho do texto.
`cex.axis`	Especifica o tamanho do texto dos eixos em relação ao a `cex`
`cex.lab`	Especifica o tamanho do texto dos labels dos eixos em relação ao valor de `cex`
`cex.main`	Especifica o tamanho do título em relação ao valor de `cex`

Parâmetro	Descrição
`cex.sub`	Especifica o tamanho do subtítulo em relação ao valor de `cex`
`font`	Especifica a fonte a ser utilizada no texto: 1=normal, 2=negrito, 3=italico, 4=italico negrito, 5=simbolo (em Adobe encoding symbol)
`font.axis`	Especifica a fonte para o texto dos eixos
`font.lab`	Especifica a fonte para os rótulos dos eixos
`font.main`	Especifica a fonte para os titulos
`font.sub`	Especifica a fonte para os subtitulos
`family`	Especifica a família da fonte utilizada para o texto. Valores padrão são: serif, sans e mono

5.2.4 Dimensões do gráfico e das margens

Parâmetro	Descrição
`pin`	Especifica as dimensões do gráfico (largura e comprimento) em polegadas
`mai`	Especifica os tamanhos das margens. Os valores são especificados em polegadas em um vetor da forma `c(bottom, left, top, right)` onde o primeiro elemento indica o tamanho da margem inforior, o segundo o tamanho da margem esquerda, o terceiro o tamanho da margem superior e o quarto o tamanho da margem direita
`mar`	Vetor numérico indicando o tamanho das margens em linhas

Além desses parâmetros, algumas funções podem ser utilizadas para adicionar elementos ao gráfico. Algumas dessas funções são:

Função	Descrição
`title()`	Adiciona elementos textuais ao gráfico, como título, labels para os eixos `x`e `y` bem como outros parâmetros gráficos disponíveis em `par()`
`axis()`	Adiciona eixo ao gráfico corrente, permitindo a especificação do lado do gráfico onde se dejesa adicionar o eixo, a posição, dentre outras opções.
`text()`	Permite adicionar elemento textual ao gráfico
`mtext()`	Adiciona elemento textual às margens do gráfico
`abline()`	Permite adicionar uma linha ao gráfico
`legend()`	Permite adicionar legenda ao gráfico
`plotmath()`	Permite adicionar anotações matemáticas nos gráficos

5.3 - CORES

Diversas funções podem ser utilizadas para definir as cores a serem utilizadas em gráficos. Pode-se definir paletas de cores específicas ou utilizando paletas pré-definidas.

As cores podem ser especificadas pelo seu nomes (exemplo `black`) e por valores hexadecimais (exemplo `#FF0000` - vermelho). Nos subtópicos a seguir serão apresentados estes recursos.

5.3.1 Obtendo os Nomes das Cores

Para acessar os nomes das funções existentes no R, pode-se utilizar a função `colors()`.

```
## Quantidade de cores definidas no R
length(colors())
```

```
[1] 657
```

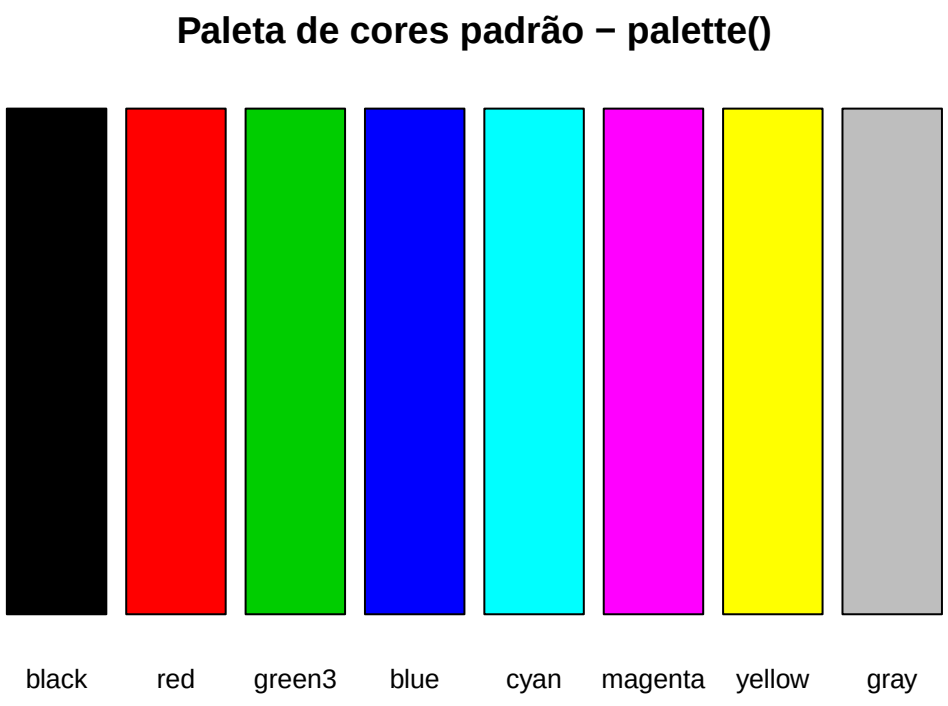
```
## 10 primeiras cores
colors()[1:10]
```

```
 [1] "white"         "aliceblue"     "antiquewhite"  "antiquewhite1"
 [5] "antiquewhite2" "antiquewhite3" "antiquewhite4" "aquamarine"
 [9] "aquamarine1"   "aquamarine2"
```

Arquivos contendo os nomes das cores e sua aparência encontram-se no repositório do curso.

5.3.2 Usando Paletas Pré-definidas

Algumas paletas pré-definidas estão disoníveis para uso. A função `palette()` define uma paleta de cores padrão contendo 8 cores: “black”, “red”, “green3”, “blue”, “cyan”, “magenta”, “yellow”, “gray”.



A paleta de cores *default* (mostrada acima) pode ser modificada passando-se como argumento para a função `pallete()` as cores desejadas . Exemplo:

```
palette(c('red', 'orange', 'blue'))
palette()
```

```
[1] "red"    "orange" "blue"
```

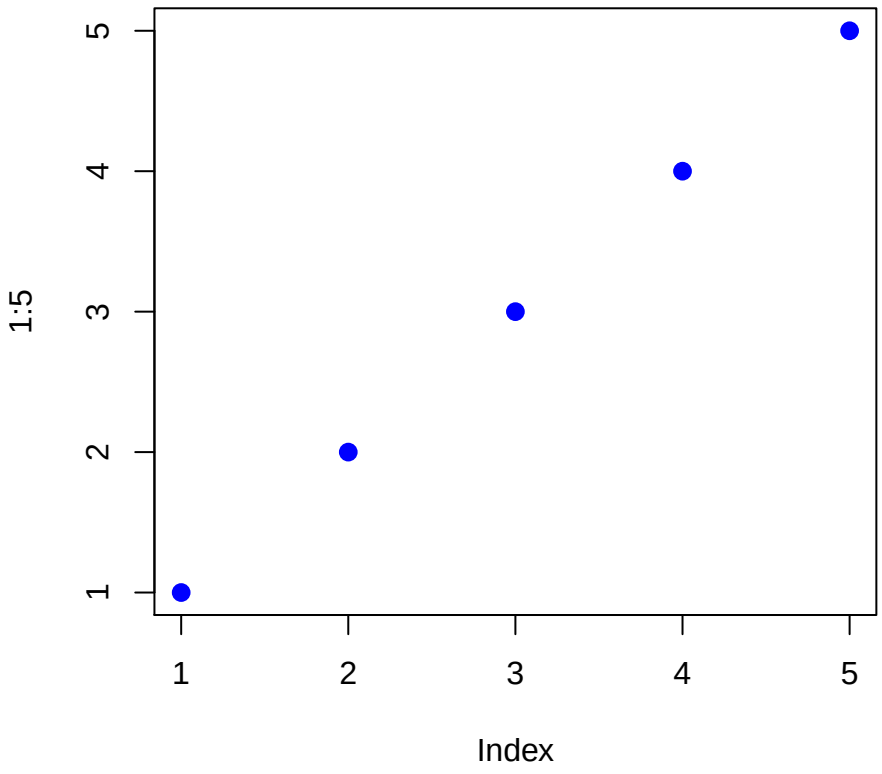
Para retornar à paleta original:

```
palette('default')
palette()
```

```
[1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta" "yellow"
[8] "gray"
```

A função `palette()`permite que se especifique cores por números representando a localização da cor na paleta. Caso a paleta seja modificada, os números farão referência às posições das novas cores na paleta. Por exemplo:

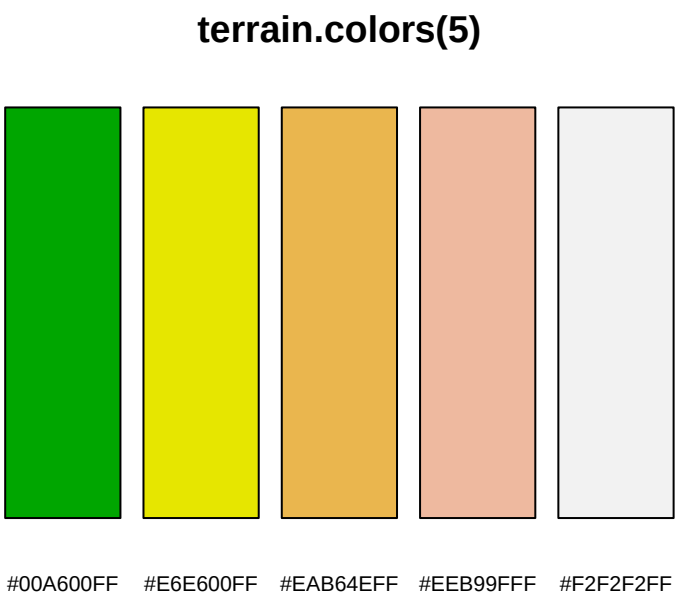
```
plot(1:5, pch=16, cex=1.3, col=4)
```



Como a cor azul é a quarta cor na paleta, ela é utilizada para colorir os pontos. `palette(colors())` permite que se especifique todas as cores por números.

Funções contedo paletas pré-definidas são: `rainbow()`, `gray.colors()`, `terrain.colors()`, `topo.colors()`, `heat.colors()`, `cm.colors()`.

Exemplo da peleta `terrain.colors()` com cinco cores:



Fica como tarefa de casa verificar as outras paletas.

5.3.3 Definindo Paletas Personalizadas

No tópico anterior foram vistas algumas funções que disponibilizam paletas de cores pré-definidas. Neste tópico será visto como especificar paletas de cores personalizadas.

Duas funções que permitem a definição de paletas de cores são as funções `collorRamp()` e `colorRampPalette()`. Estas funções permitem criar cores intermediárias entre cores definidas pelo usuário.

A função `collorRamp()` retorna **uma função** cujo argumento será um vetor de valores entre 0 e 1 que serão mapeados para uma matriz de cores especificadas no formato RGB com uma linha por cor e outras 3 ou 4 colunas, especificando as componentes RGB e alpha.

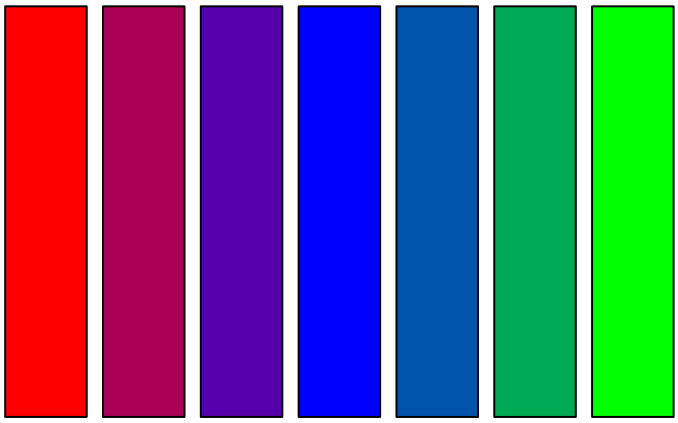
A função `colorRampPalette()` também retorna **uma função** que aceitará como argumento um valor inteiro indicando a quantidade de cores desejadas e retorna um vetor de caracteres contendo cores interpolando entre as cores definidas.

Exemplo:

```
## Criar uma paleta de cores variando do vermelho ao verde passando pelo azul

# Define a função que irá gerar a paleta
minha_paleta <- colorRampPalette(c("red", "blue", "green"))

# Cria uma paleta com 7 cores variando do vermelho ao verde
paleta_vermelho_verde <- minha_paleta(7)
paleta_vermelho_verde
```

```
[1] "#FF0000" "#AA0055" "#5500AA" "#0000FF" "#0054AA" "#00AA54" "#00FF00"
```

O resultado é uma paleta de cores que vai do vermelho ao verde passando pelo azul.



Além destas funções, uma opção frequentemente utilizada é a utilização de pacotes que disponibilizem paletas de cores bastante elaboradas.

Um pacote bastante utilizado é o `RColorBrewer`.

As paletas de cores são construidas de acordo com a seguinte classificação: `sequenciais`, `divergentes` e `qualitativa`.

As paletas **sequenciais** são úteis para destacar dados ordenados que variem de valores baixo para alto. Normalmente escolhe-se as cores mais claras para valores baixos e as cores mais escuras para os altos.

As paletas **divergentes** colocam igual ênfase valores medianso e mais ênfase nos valores extramos.

As paletas **qualitativas** não enfatizam os valores representados. São úteis para representar dados nominais ou categóricos.

As paletas existentes no pacote, em cada uma das categorias acima definidas, são listadas a seguir:

Paletas sequenciais: `Blues`, `BuGn`, `BuPu`, `GnBu`, `Greens`, `Greys`, `Oranges`, `OrRd`, `PuBu`, `PuBuGn`, `PuRd`, `Purples`, `RdPu`, `Reds`, `YlGn`, `YlGnBu`, `YlOrBr`, `YlOrRd`

Paletas divergentes: `BrBG`, `PiYG`, `PRGn`, `PuOr`, `RdBu`, `RdGy`, `RdYlBu`, `RdYlGn`, `Spectral`

Paletas qualitativas: `Accent`, `Dark2`, `Paired`, `Pastel1`, `Pastel2`, `Set1`, `Set2`, `Set3`

Para consultar uma paleta de cores, utiliza-se a função `display.brewer.pal()`, que tem por argumentos a quantidade de cores desejada e o nome da paleta.

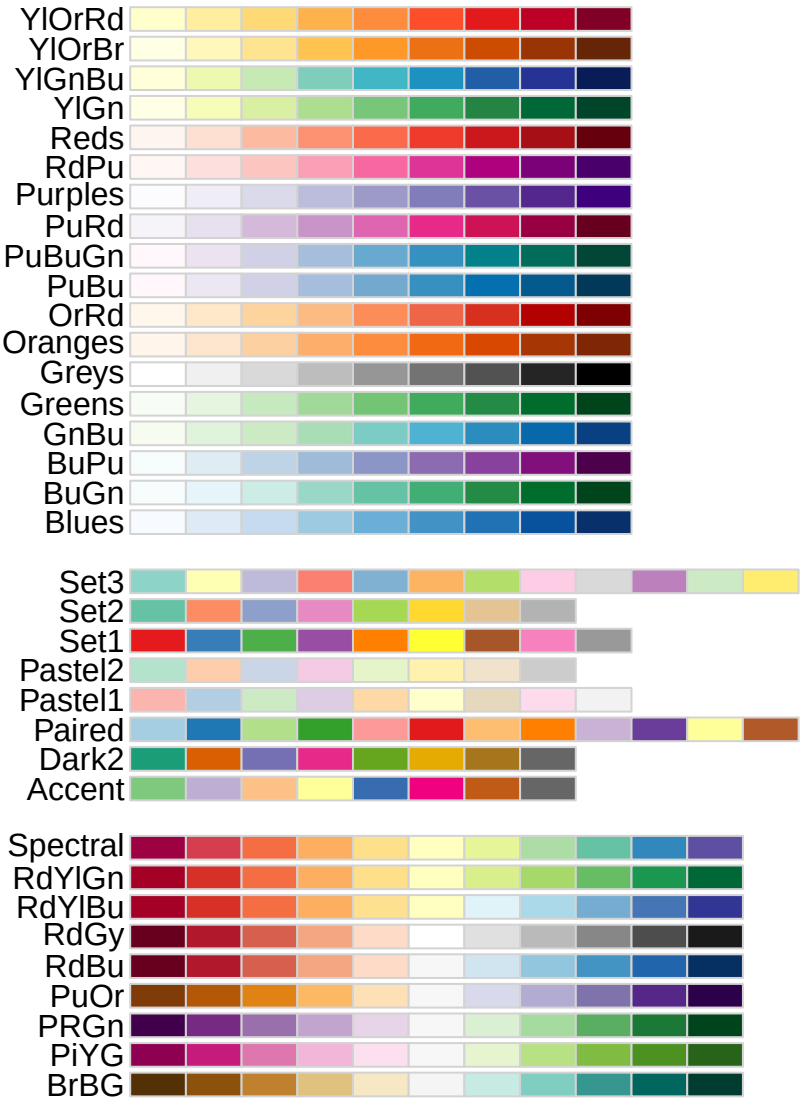
```
library(RColorBrewer)
display.brewer.pal(7, 'BuGn')
```

BuGn (sequential)

Deve ser notado que cada paleta de cores disponibiliza uma quantidade distinta de cores. Para usar estas paletas de cores nos gráficos faz-se uso da função `brewer.pal()`, que retorna um vetor com as cores escolhidas.

Para visualizar todas as paletas disponibilizadas pelo pacote, utilize a função `display.brewer.all()`. Exemplo:

```
display.brewer.all()
```



O pacote `colorspace` fornece uma interface gráfica conveniente para a seleção de cores. É um excelente recurso para simplificar a escolha de cores.

5.3.4 Definindo Vetores de Cores

Existe um conjunto de funções que permitem ao usuário definir cores. Exemplos de tais funções são:

Função	Descrição
`hsv()`	Cria vetores de cores a partir da especificação de Hue (tonalidade), Saturation (Saturação) e Value (Valor)
`gray()`	Cria vetores de cores em tonalidade de cinza
`hcl()`	Cria vetores de cores a partir da especificação de Hue (tonalidade), Chroma (cromaticidade) e Luminance (luminância)
`rgb()`	Cria vetores de cores a partir da especificação das intensidades de Red, Green e Blue
`col2rgb()`	Esta função permite a conversão de cores especificadas pelo nome, hexadecimal para o formato RGB.

Deixamos a cargo do leitor consultar a ajuda das referidas funções e verificar como utilizá-las.

5.4 - GRÁFICOS ESTATÍSTICOS

O R dispõe de 4 sistemas gráficos: `base`, `lattice` , `ggplot2` e `grid`. Neste curso serão abordados apenas os gráficos do sistema básico do R e o pacote `lattice` para gráficos condicionados.

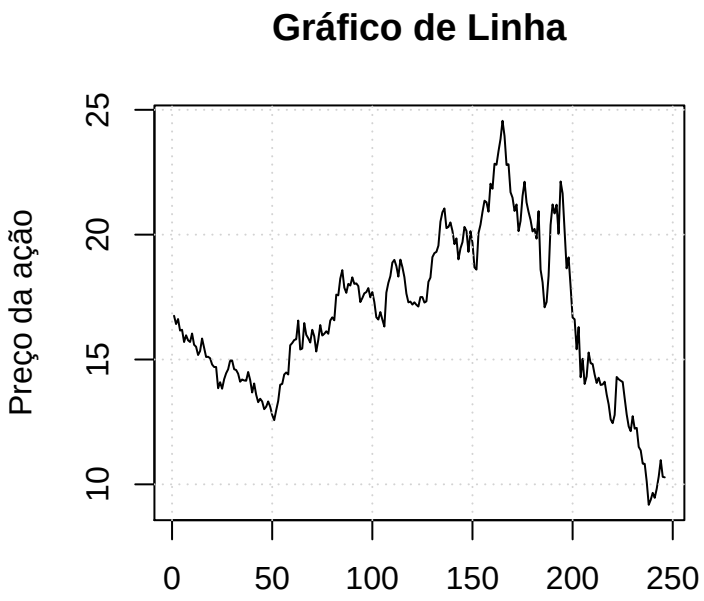
O sistema `base` é o sistema gráfico nativo do R. Serão apresentados a seguir os principais gráficos disponibilizados por este sistema.

Gráficos podem ser uma excelente ferramenta para auxiliar nos trabalhos de auditoria mas, infelizmente, são subutilizados.

5.4.1 Gráfico de linhas

Gráficos de linha são indicados para comunicar a evolução temporal de uma variável. Estes gráficos permitem observar tendências e sazonalidades na evolução temporal da variável. Seu uso será exemplificado com o conjunto de dados `precos_acoes_petrobras.xlsx`.

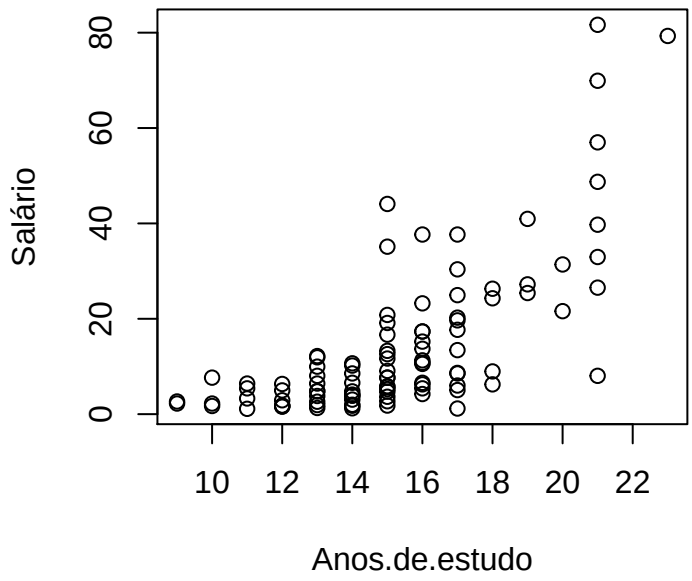
```
library(readxl)
diretorio <- 'C:\\Users\\Marcos\\Dropbox\\1. Cursos ECG\\Intro-R Treinamento TCE-MT\\3.dados'

setwd(diretorio)
acoes <- read_excel('precos_acoes_petrobras.xlsx')

acoes <- acoes[order(acoes$Data),]
plot(acoes$Cotacao, type = 'l',
     main='Gráfico de Linha',
     xlab='', ylab='Preço da ação')
grid()
```



5.4.2 Diagrama de dispersão

O diagrama de dispersão é um gráfico muito utilizado para evidenciar o relacionamento entre duas variáveis quantitativas. Por exemplo, como é relação entre as variáveis `Salário` e `Anos.de.estudo` no conjunto de dados `rh`?

```
setwd(diretorio)
load('rh_limpo.RData')

## Retirar uma amostra aleatória de 100 elementos
set.seed(10)
rh_amostra <- rh[sample(row.names(rh), 100),]
plot(Salário ~ Anos.de.estudo, data=rh_amostra)
```



Os padrões normalmente buscados em um diagrama de dispersão (`scatterplot`), podem ser ilustrados com as figuras a seguir:

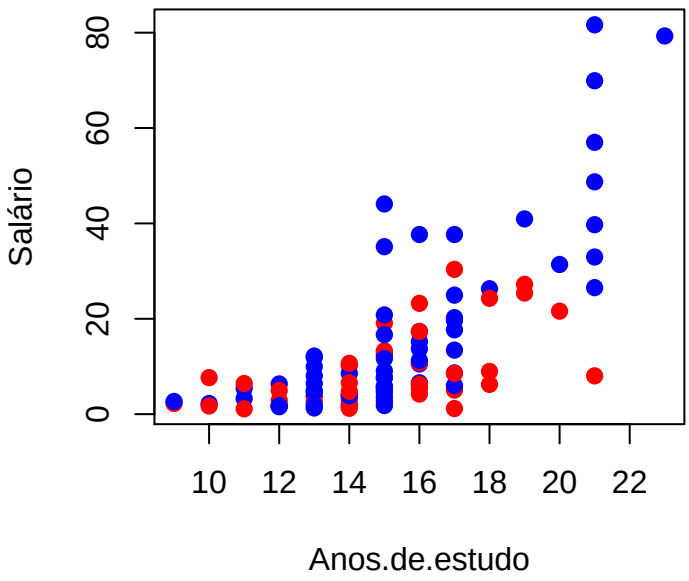
Fonte:Livro `Data Points` de Natan Yau

O gráfico acima mostra a relação entre `Salário` e `Anos.de.estudo` para uma amostra dos funcionários da empresa.

Mas será que o padrão evidenciado acima se repete se fossem considerados separadamente os dados dos funconários do sexo masculino e feminino?

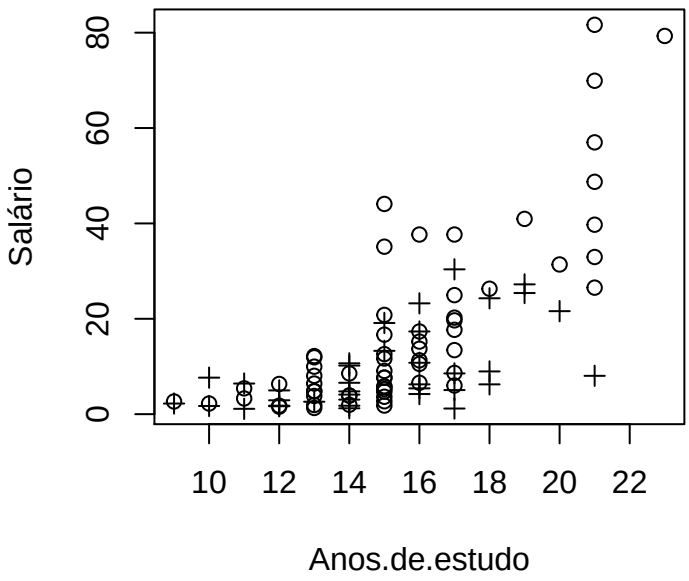
Uma forma de tentar visualizar estas duas categorias no gráfico é atribuir cores diferentes aos pontos do gráfico que representam dados de funcionários do sexo masculino e feminino. Pro exemplo, o gráfico a seguir utiliza cores (azul = homens, verde = mulheres) para evidenciar os sexos.

```
cores <- ifelse(rh_amostra$Sexo == 'Masculino', 'blue', 'red')
plot(Salário ~ Anos.de.estudo, data=rh_amostra, col=cores, pch=16, cex=1.2)
```



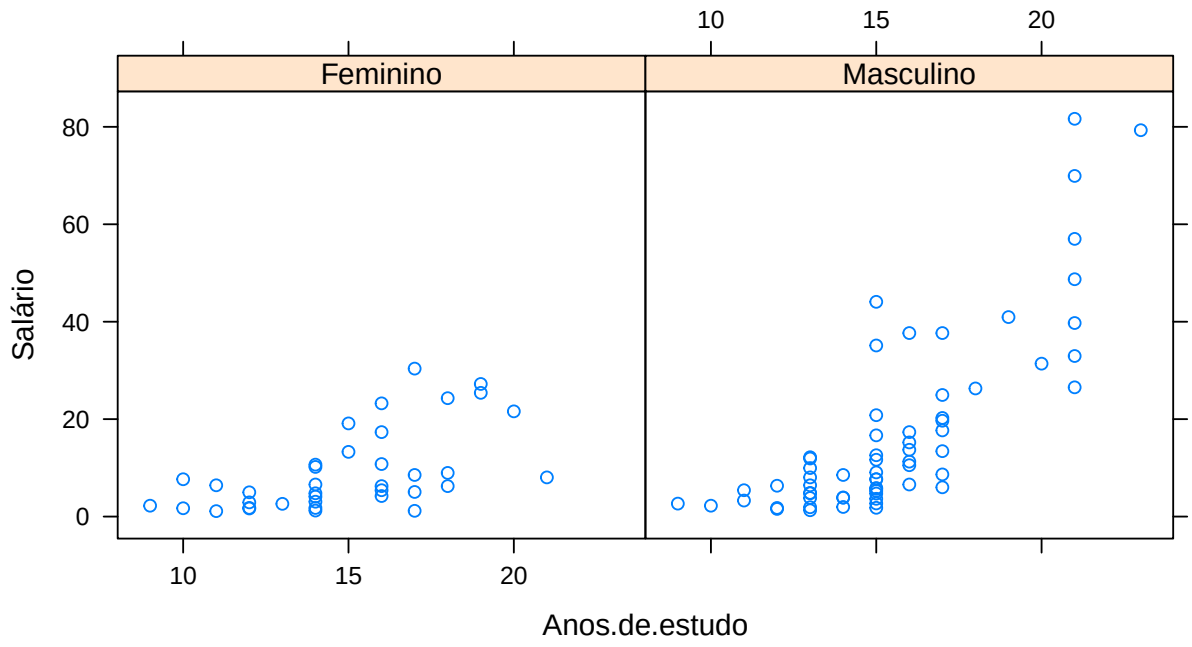
Também seria possível utilizar o formato do marcador (parâmetro `pch=`) para distinguir entre homens e mulheres.

```
simbolo <- ifelse(rh_amostra$Sexo == 'Masculino', 1, 3)
plot(Salário ~ Anos.de.estudo, data=rh_amostra, pch=simbolo)
```



A análise dos gráficos não permitem tirar grandes conclusões. Em situações como essa, os gráficos condicionados podem ser bastante úteis. No exemplo a seguir utilizaremos o pacote `lattice` para realizar diagramas de dispersão condiciondos à variável `Sexo`.

```
library(lattice)
xyplot(Salário ~ Anos.de.estudo | Sexo, data=rh_amostra)
```



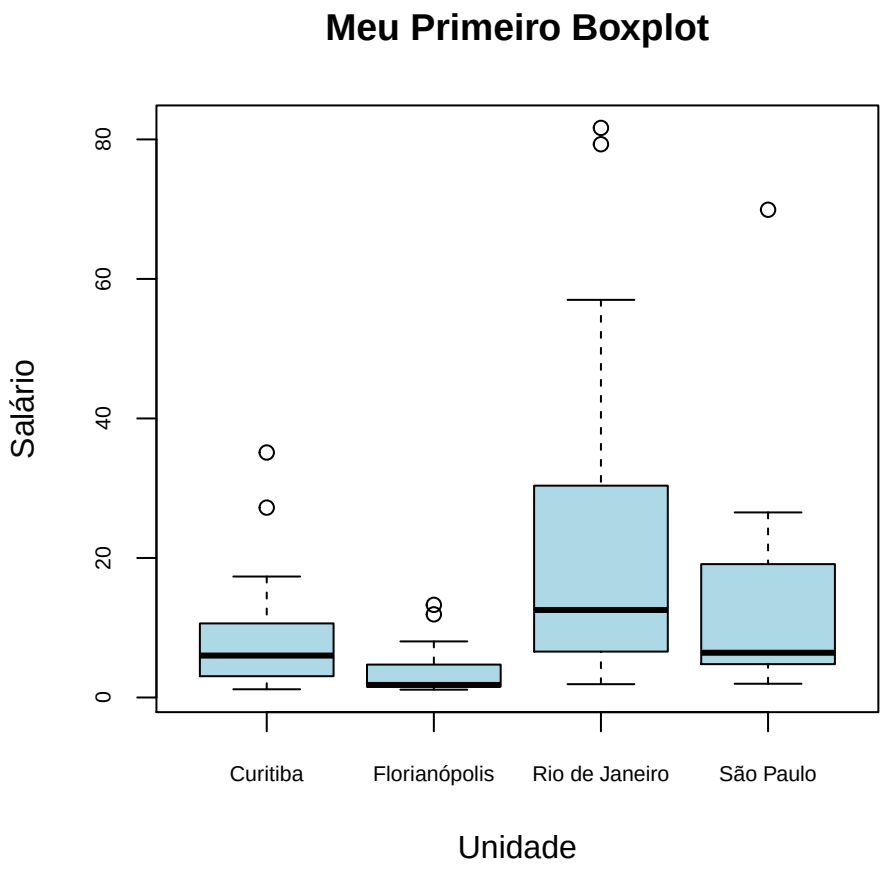
A abordagem acima é mais efetiva do que as que feitas anteriormente.

5.4.3 Boxplot

O boxplot é um gráfico que representa a distribuição de um conjunto de dados com base em alguns de seus parâmetros descritivos, quais sejam: a mediana ($q2$), o quartil inferior ($q1$), o quartil superior ($q3$) e do intervalo interquartil ($IQR = q3 - q1$).

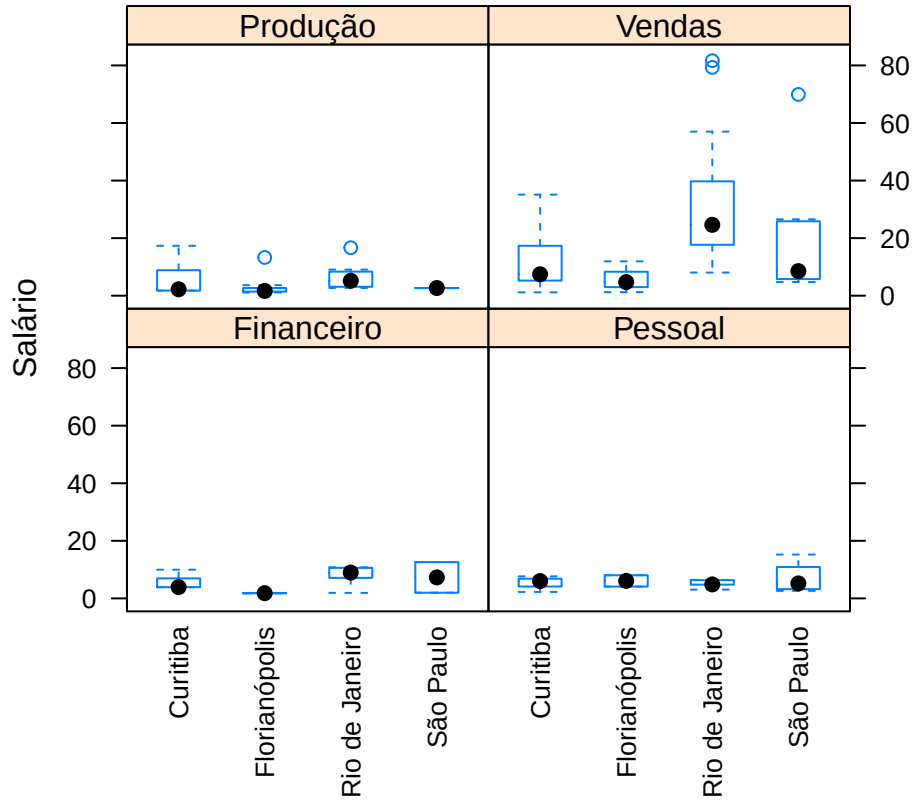
No gráfico a seguir faremos boxplots da variável `Salário` por `unidade`.

```
setwd(diretorio)

boxplot(Salário ~ Unidade, data=rh_amostra,
        main='Meu Primeiro Boxplot', # Definição do título do gráfico
        xlab='Unidade',              # Label para o eixo x
        ylab='Salário',              # Label para o eixo y
        col='lightblue',
        cex.axis=0.7)                # Definição da cor a ser usada
```

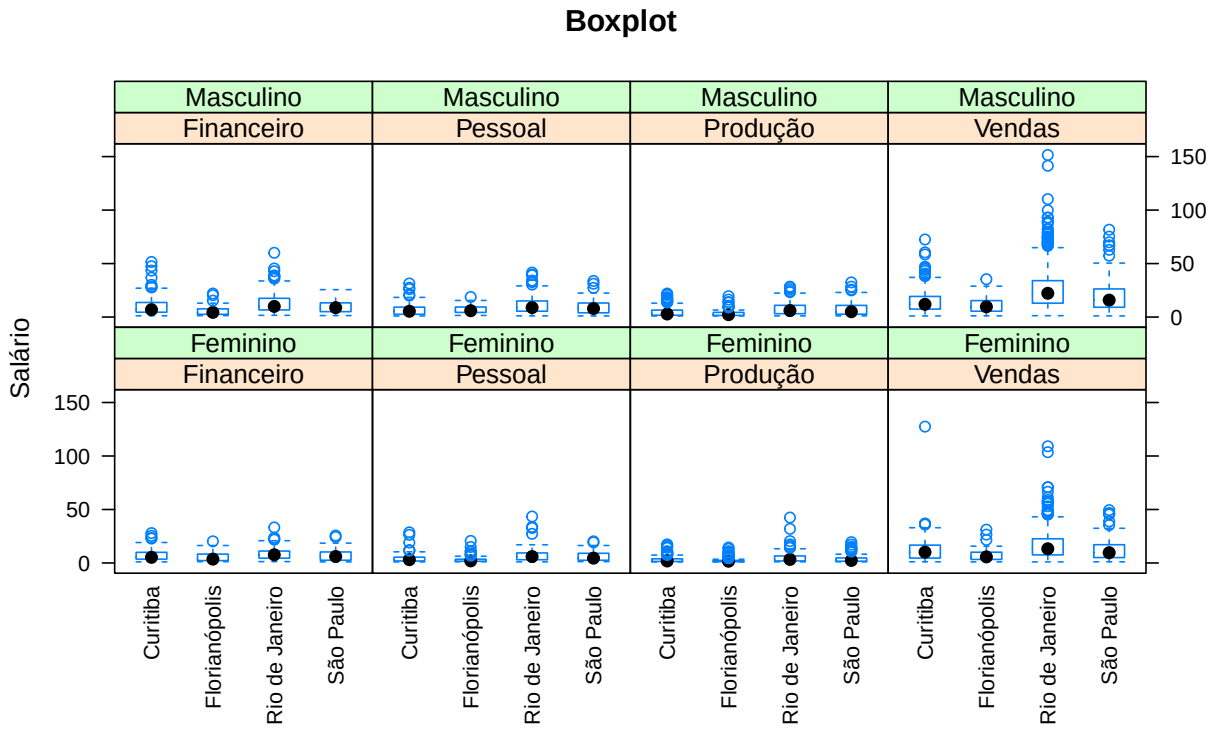


Nos exemplos a seguir, faremos os mesmos boxplots, mas agora condicionados à variável `Departamento`, e `Departamento` e `Sexo`.

```
bwplot(Salário ~ Unidade | Departamento, data=rh_amostra,
       main='Boxplot', scales = list(x = list(rot = 90)))
```



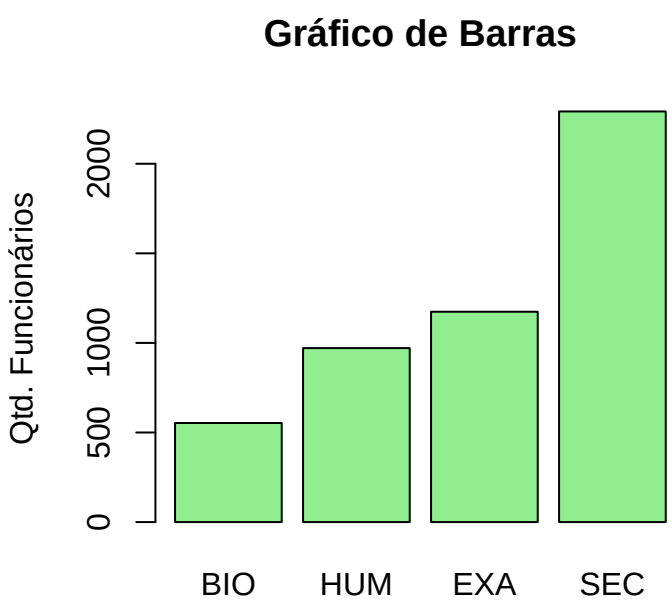

```
bwplot(Salário ~ Unidade | Departamento + Sexo, data=rh,
       main='Boxplot', scales = list(x = list(rot = 90)))
```



5.4.4 Gráfico de barras

Gráficos de barras são muito comuns.

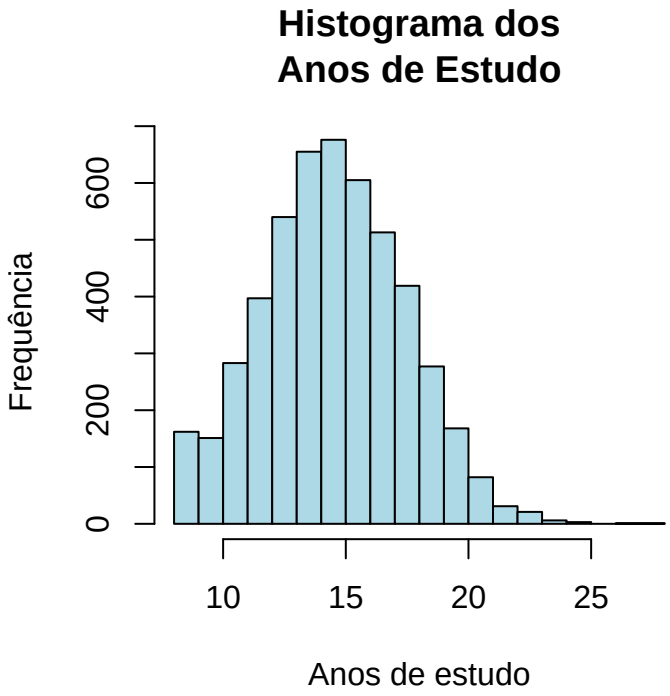
```
dd <- sort(with(rh, table(Formação)))
barplot(dd, main="Gráfico de Barras",
            col='lightgreen',
            ylab='Qtd. Funcionários')
```



5.4.5 Histograma

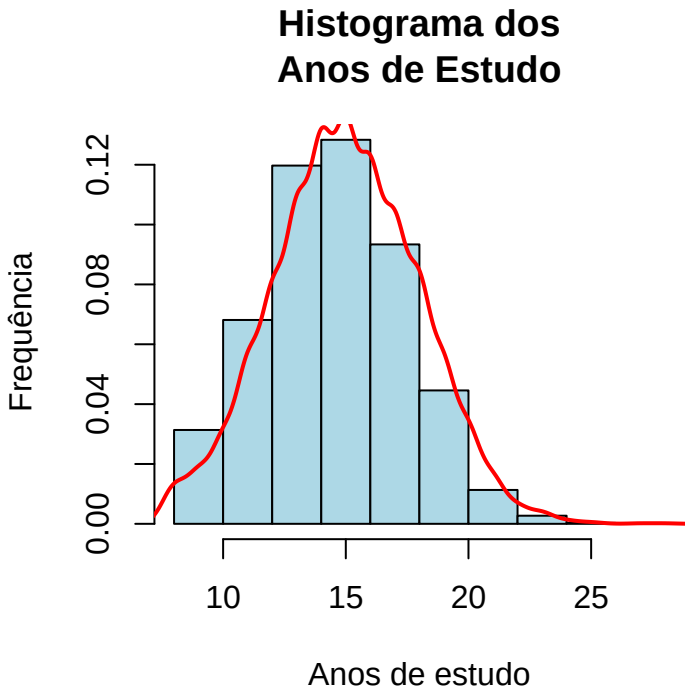
Os histogramas são gráficos muito utilizados para se avaliar a distribuição de uma variável. No exemplo a seguir, utiliza-se um histograma para avaliar a distruibuição da variável `Anos.de.estudo`.

```
hist(rh$Anos.de.estudo,
     col="lightblue",
     breaks=15,
     main='Histograma dos\nAnos de Estudo',
     xlab='Anos de estudo',
     ylab='Frequência')
```



Histograma com adição de densidade:

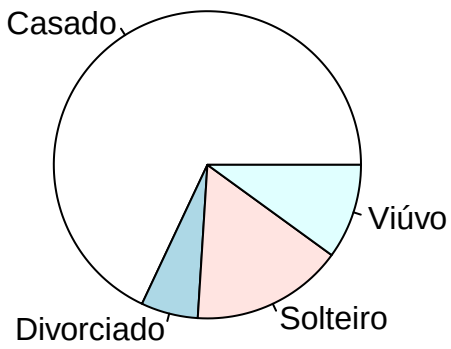
```
hist(rh$Anos.de.estudo,
     col="lightblue",
     main='Histograma dos\nAnos de Estudo',
     xlab='Anos de estudo',
     ylab='Frequência',
     probability = TRUE)

d <- density(na.omit(rh$Anos.de.estudo))
lines(d, col='red', lwd=2)
```



5.4.6 Gráfico de pizza

Gráficos de pizza são muito utilizados na prática embora não sejam as visualizações mais efetivas, em razão da dificuldadeo ser humano em avaliar ângulos. No R este gráfico é feito com a função `pie()`. Exemplo:

```
estcivil <- table(rh_amostra$Estado.Civil)
pie(estcivil)
```



Com frequência, um gráfico de barras será mais efetivo que um gráfico de pizza.

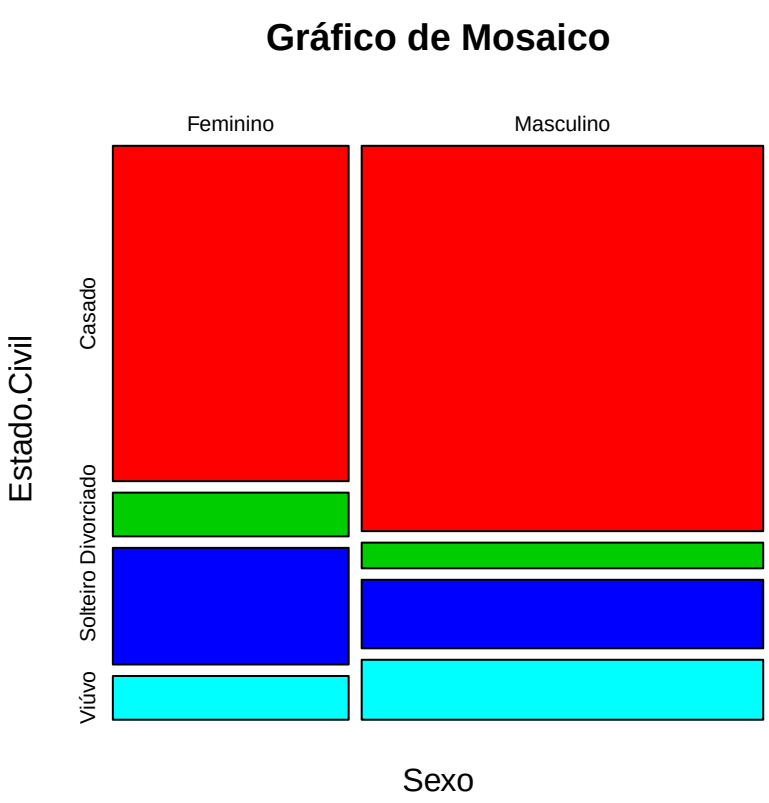
5.4.7 Gráfico de mosaico

O gráfico de mosaico é um gráfico utilizado para a visuação de tabelas de frequências. Permite uma visualização das frequências contidas nas células.

```
pp <- with(rh_amostra, table(Sexo, Estado.Civil))
prop.table(pp, 1)
```

```
           Estado.Civil
Sexo            Casado Divorciado   Solteiro      Viúvo
  Feminino  0.62162162 0.08108108 0.21621622 0.08108108
  Masculino 0.71428571 0.04761905 0.12698413 0.11111111
```

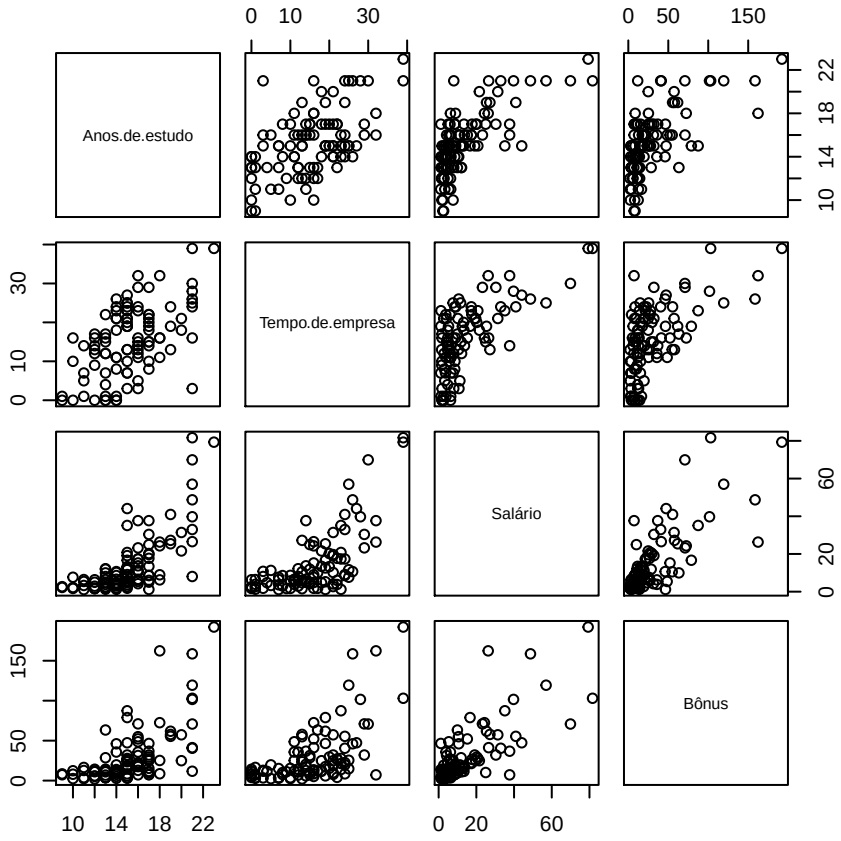
```
mosaicplot(pp, main='Gráfico de Mosaico', color=c(2, 3, 4, 5))
```



5.4.8 Scatterplot matrix

Um gráfico de natureza exploratória muito útil é a matriz de diagramas de dispersão, que nos possibilita visualizar em um só gráfico diversos diagramas de dispersão.

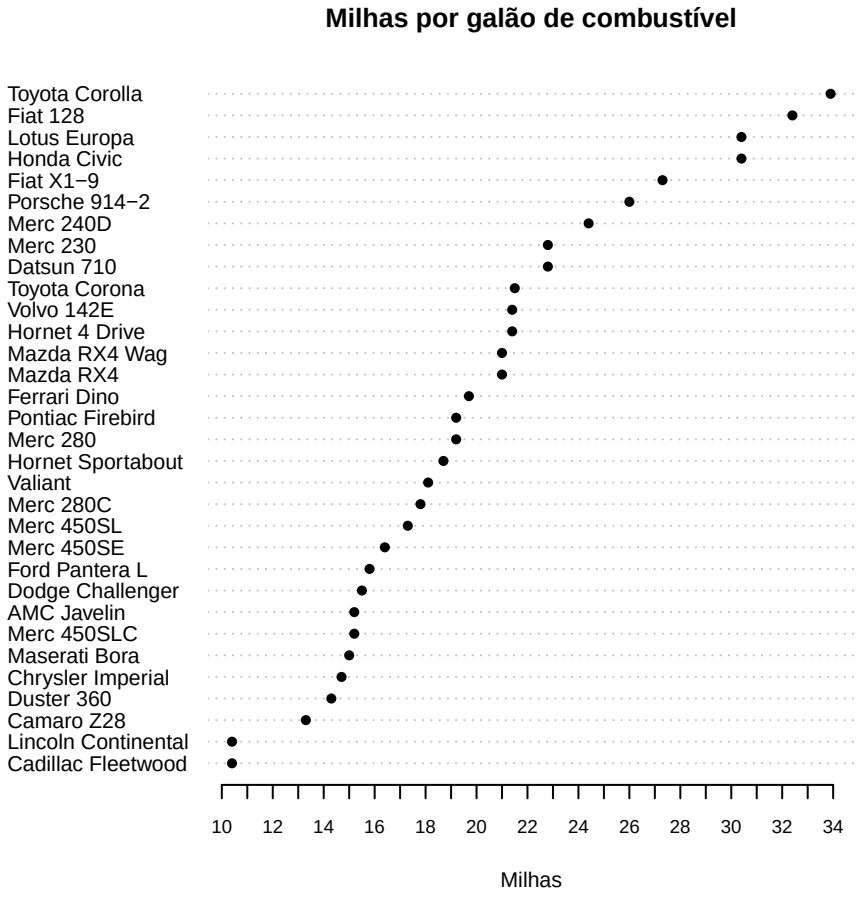
```
pairs(rh_amostra[,c('Anos.de.estudo', 'Tempo.de.empresa', 'Salário', 'Bônus')])
```



5.4.9 Gráfico de pontos

Às vezes o gráfico de pontos pode ser um bom substituto para o gráfico de barras. O conjunto dedados `mtcars` será utilizado para ilustrar sua construção.

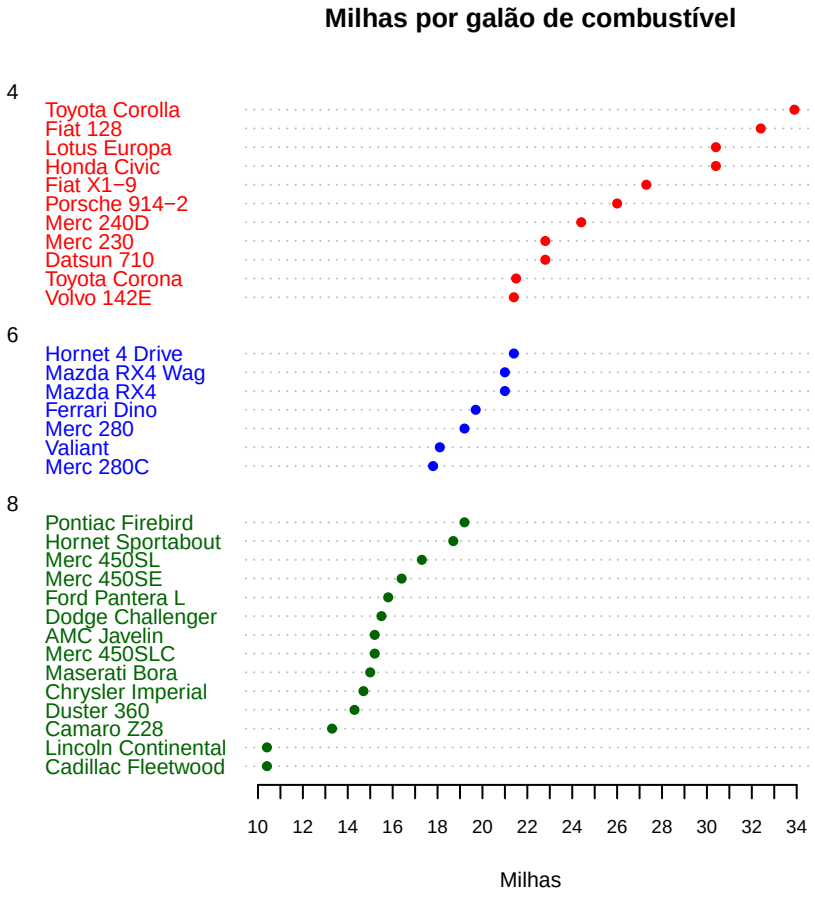
```
mtcars <- mtcars[order(mtcars$mpg),]

par(bty='n', xaxt='n')
dotchart(mtcars$mpg, labels=row.names(mtcars),
         cex=0.7, main='Milhas por galão de combustível',
         xlab='Milhas', pch = 16)
par(xaxt='s')
axis(1, at=10:34, cex.axis=0.6)
```




```
## Gráfico de pontos agrupado por um fator...
unique(mtcars$cyl)
```

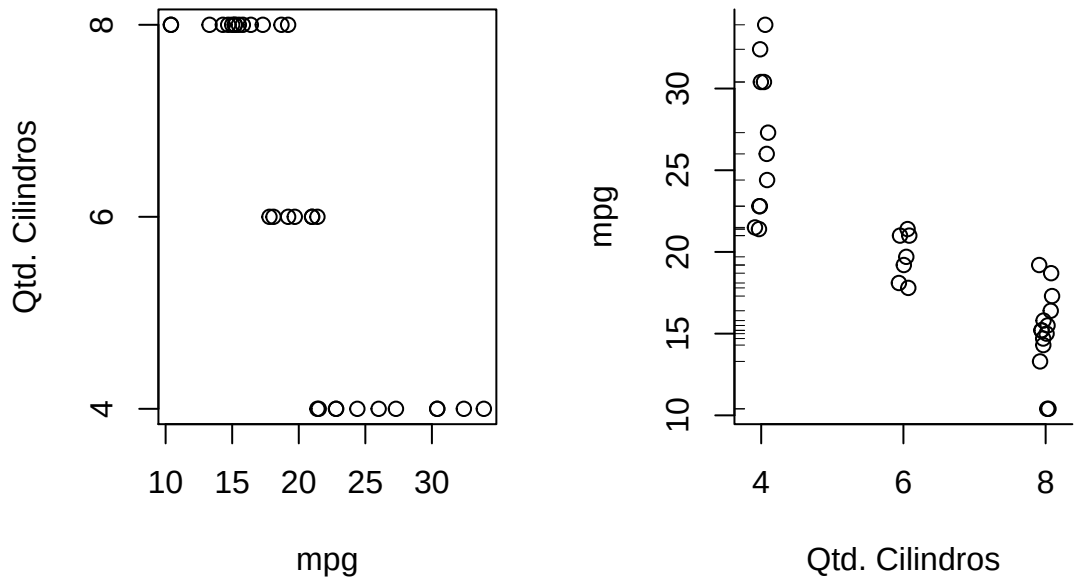
```
[1] 8 6 4
```

```
cores <- ifelse(mtcars$cyl == 4, 'red',
         ifelse(mtcars$cyl == 6, 'blue', 'darkgreen'))

par(xaxt='n')
dotchart(mtcars$mpg, labels=row.names(mtcars), cex=0.7,
         groups=factor(mtcars$cyl), gcolor='black', color=cores,
         pch=16, main='Milhas por galão de combustível', xlab='Milhas')
par(xaxt='s')
axis(1, at=10:34, cex.axis=0.6)
```



Outra função que pode ser utilizada para produzir gráficos semelhantes aos gráficos de pontos é `stripchart()` Seu uso é ilustrado a seguir.

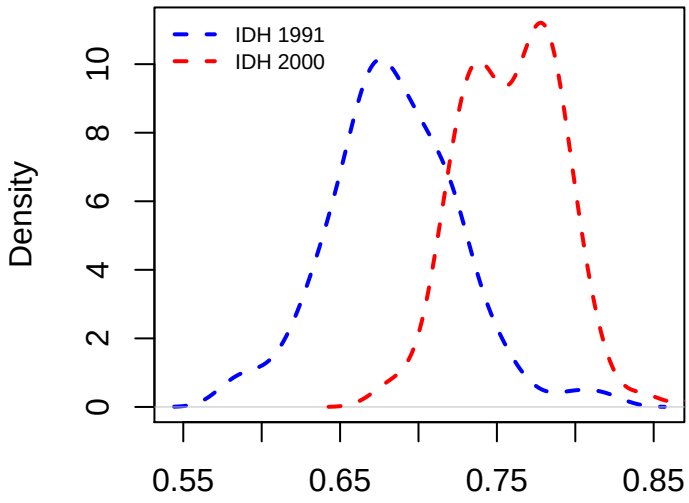
```
par(bty='o', mfrow=c(1, 2))
stripchart(mpg ~ cyl, data=mtcars, pch=21, ylab='Qtd. Cilindros')

par(bty='l')
stripchart(mpg ~ cyl, data=mtcars, method='jitter',
           jitter=0.05, pch=21, vertical=TRUE, xlab='Qtd. Cilindros')

rug(mtcars$mpg, side=2)
```



5.4.10 Gráficos de densidade

Estes gráficos podem ser muito úteis quando queremos comparar as distribuições de dois ou mais conjuntos de dados. O conjunto de dados `IDH1991_2000.csv` será utilizado para compararmos a do IDH dos municípios nos de 1991 e 2000.

```
setwd(diretorio)
idh <- read.csv2('IDH1991_2000.csv')
idh <- idh[, c('idh1991', 'idh2000')]

## Desenhando os gráficos
idh1991 <- density(idh$idh1991)
idh2000 <- density(idh$idh2000)

y.limites <- range(c(idh1991$y,idh2000$y))

plot(idh1991, main='', xlab='', lty=2, lwd=2, col='blue', ylim=y.limites)
lines(idh2000, lty=2, lwd=2, col='red')

legend('topleft', legend=c('IDH 1991', 'IDH 2000'),
       lwd=2, lty=2, col=c('blue', 'red'), bty='n', cex=0.7, y.intersp=1)
```



5.4.11 Calendar Plot

Para produzir este gráficos, utilizaremos a função `calendarHeat()`. Esta fução está definida no arquivo `funcao_calendarHeat.R` e pode ser utilizada da seguintes forma:

```
## Carregar a função calendarHeat()
source('C:\\Users\\Marcos\\Dropbox\\1. Cursos ECG\\Intro-R Treinamento TCE-MT\\5.scripts\\6. funcao_cal

setwd(diretorio)
isp <- read.csv2('microdados_isp_2013_2015Abr.csv', as.is = TRUE)
isp <- subset(isp, ETIT == 1 & ETEN == 'Vítima')$DATF
isp <- table(isp)
isp <- as.data.frame(isp)
isp <- subset(isp, isp != '')

## Converão das datas...
minha_configuracao <-  Sys.getlocale("LC_TIME")
Sys.setlocale("LC_TIME", "C")
```

```
[1] "C"
```

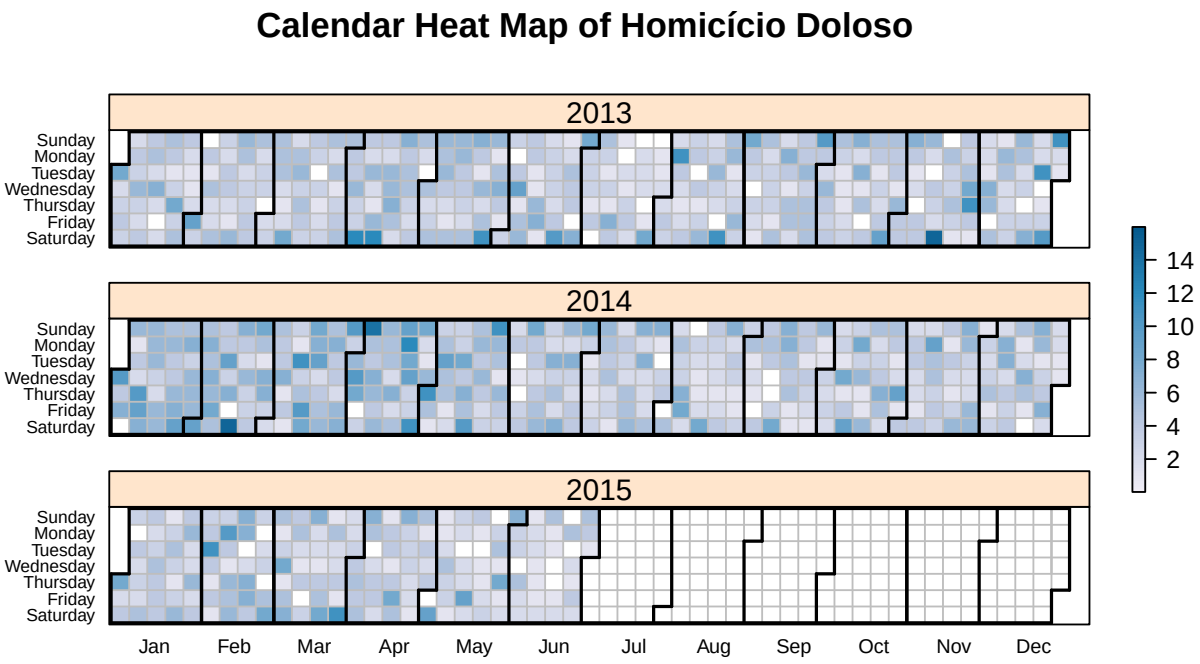
```
isp$isp <- as.Date(isp$isp, '%d-%b-%Y')
Sys.setlocale("LC_TIME", minha_configuracao)
```

```
[1] "Portuguese_Brazil.1252"
```

```
## Apenas registros relativos aos anaos de 2013, 2014 e 2015.
isp <- subset(isp, as.integer(format(isp, '%Y')) >= 2013)

## PRODUZ O GRÁFICO DE CALENDÁRIO
calendarHeat(dates=isp$isp, values=isp$Freq,
             varname='Homicício Doloso', color='w2b')
```

```
Loading required package: grid
```

```
Loading required package: chron
```



5.5 - OUTROS GRÁFICOS

Além dos gráficos apresentados acima, diversos outros estão disponíveis aos usuários do R. Sugiro pesquisar na internet sobre os seguintes gráficos:

- *violin plot* - pacote{vioplot}
- *qqplot* - base R
- *parallel coordinates* - pacote{GGally}
- *calendar plot* - pacote{googleVis} / calendarHeat()

- *heatmap* - base R
- *sankey plot* - pacotes{riverplot, googleVis}
- *spider plot / radar chart* base R
- *bubble chart* - base R
- *treemap* pacote{treemap, googleVis}

Além desses gráficos, pesquise os seguintes pacotes: `ggplot2`, `ggmap`, `rCharts`, `googleVis`, `ggvis`, `rgl`, `htmlwidgets`, `plotrix`, `manipulate`, `Rggobi`, `iPlots`, `igraph`, `GrapheR`.

5.6 - DISPOSITIVOS GRÁFICOS

Até o momento os gráficos produzidos foram exibidos na tela do computador e, a menos que fossem salvos manualmente com o uso das facilidades oferecidas pelo RStudio, eram perdidos.

O R dispõe de conjunto de funções para auxiliar na construção de gráficos produzidos no sistema básico e com o pacote `grid`. Algumas dessas funções permitirão salvar os gráficos produzidos em formatos diversos. Por exemplos, gráficos podem ser produzidos e salvos nos seguintes formatos: .svg, .png, .pdf, .bmp, .jpeg e .tiff, utilizando-se para tanto as seguintes funções, respectivamente: `svg()`, `png()`, `pdf()`, `bmp()`, `jpeg()`, `tiff()`. Exemplo. Criar um gráfico em .pdf:

```
setwd(diretorio)
pdf('meu_primeiro_grafico.pdf') ## Abre o dispositivo gráfico
plot(1:10)                      ## Cria o gráfico propriamente dito
dev.off()                       ## Desliga o dispositivo gráficos (fecha a conexão com o arquivo)
```

Nota: as dimensões do gráfico são fornecidas em polegadas. A opção default é 7 x 7.

Lembre-se: `1 in = 2.54cm` e `1 cm = 0.39370079 in`

Para converter de cm para polegadas podemos escrever uma funçãozinha bem simples:

```
to.pol <- function(cm){cm * 0.39370079}
```

Agora podemos escrever nossa função pensando nas medidas em termos de centímetros. Um gráfico de 7 cm de altura, por 10 cm de largura:

```
setwd(diretorio)
pdf('meu_segundo_grafico.pdf', height = to.pol(7), width = to.pol(10))
plot(1:10, 10:1)
dev.off()
```

A função `dev.off()` tem a função de fechar o dispositivo gráfico, para outros gráficos que venham a ser produzidos não sejam encaminhados para o mesmo arquivo.

5.7 - EXERCÍCIOS

1. Utilizando o conjnto de dados `rh_limpo.RData` elabore um gráfico de barras que indique a frequência de pessoas em cada cada categoria da variável `Estado.Civil`. Pesquise na internet como colocar o valor da frequência absoluta em cima de cada barra.

2. Faça um gráfico de dispersão das variáveis `Anos.de.estudo` e `Tempo.de.empresa`. Existe alguma relação entre estas variáveis?

3. Faça boxplots da variável `Anos.de.estudo`, por `Sexo`, condicionados à variável `Departamento`. Comente a respeito.

4. Crie uma paleta de cores, contendo as seguintes cores: `white`, `chartreuse3`, `darkred`, `gray47`, `grey39`.

5. Crie uma paleta de 7 cores indo do vermelho ao laranja. Crie um gráfico de pizza com 7 fatias e pinte com as cores da paleta criada.

6. Crie um gráfico de pizza com cinco fatias e pinte as fatias das seguintes cores: `verde`, `amarelo`, `azul`, `preto`, `vermelho`.

7. Utilizando o conjundo de dados `cobertura de vacina.csv`, elabore um grafico boxplot que permita comparar a cobertura de vacinação nos anos 2010, 2011 e 2012. Coloque título e rótulos no eixo y. Pinte os boxplot de amarelo.

8. Produza um histograma da variável `x <- rnorm(500)` no formato `.png`, com o tamanho 5cm x 5cm.