Programa de pós-graduação de pesquisa clínica em doenças infecciosas - 2019.

"Computação: introdução a aplicativos de informática para auxílio à pesquisa"

R-project - Roteiro de aula

Coordenador: Pedro Emmanuel A. A. do Brasil

Leituras adicionais:

http://www.statmethods.net/

http://cran.r-project.org/doc/FAQ/R-FAQ.html

http://cran.r-project.org/doc/manuals/r-release/R-intro.pdf

http://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf

http://cran.r-project.org/other-docs.html

Há livros com tutoriais e exemplos com R mais genéricos tais como: "The R Book", "R For Dummies**",** "Data Analysis and Graphics Using R: An Example-Based Approach", "Software for Data Analysis: Programming with R" entre muitos outros. Há também livros de estratégias de análise de dados utilizando o R como: "Clinical Trial Data Analysis Using R", "Applied Meta-Analysis with R", "Longitudinal Data Analysis for the Behavioral Sciences Using R" etc.

Sumário

1. Objetivos do módulo

O curso é estritamente prático e o aluno deve manipular o programa durante as oficinas. Os objetivos deste curso são: permitir que o aluno de pós-graduação em saúde, não iniciado em "pilotagem de dados", seja apto a importar e editar os dados a serem utilizados na sua tese/dissertação e encontrar e entender as funções que o permitam conduzir o plano de análise de seu trabalho. Os conceitos aprendidos nesse curso permitirão ao aluno que chegar ao curso de estatística, que este possa ficar concentrado no conteúdo das aulas ao invés de como manipular o programa. Nenhuma estratégia de análise específica será abordada, tais como: testes diagnósticos, ensaios clínicos, modelos de regressão, análise de sobrevivência ou de dados longitudinais apesar de alguns exemplos serem mostrados ocasionalmente. Ao longo das oficinas, espera-se que os alunos devam tentar fazer tarefas sozinhos e à medida que conseguirem executarem algumas tarefas, possam buscar ajuda na própria documentação do R e em outros materiais didáticos mais complexos e sofisticados. Acho que para os não iniciados o sítio Quick-R é bem amigável (http://www.statmethods.net/). Junto com a apostila, no sítio há também um script para seguir os exemplos dessa apostila e alguns bancos de dados de exemplo.

2. Razões pelas quais R foi escolhido para este módulo:

- R é livre (www.opensource.org)
- R é capaz de realizar desde análises simples até análises muito sofisticadas
- O suporte para R é tão bom ou melhor do que softwares proprietários na lista de usuários, fóruns blogs e contato com autores, desde que o problema seja bem formulado. r-br@listas.c3sl.ufpr.br (cadastre-se antes)
- Multi-plataforma (Linux, Mac e Windows).
- Para os que estão iniciando e nunca experimentaram qualquer programa de análise, a princípio é tão difícil iniciar em R quanto em qualquer outro programa.

3. O que é o R?

R é uma linguagem e ambiente para computação estatística e gráfica. O R é um sistema desenvolvido a partir da linguagem S, que tem suas origens nos laboratórios da AT&T no final dos anos 1980. Posteriormente o S foi vendido

e deu origem a uma versão comercial, o S-Plus. Em 1995 dois professores de estatística da Universidade de Auckland, na Nova Zelândia, iniciaram o "Projeto R" (porque R vem depois de S), com o intuito de desenvolver um programa estatístico poderoso baseado na linguagem S, e de domínio público. O R pode ser baixado gratuitamente em http://www.r-project.org.

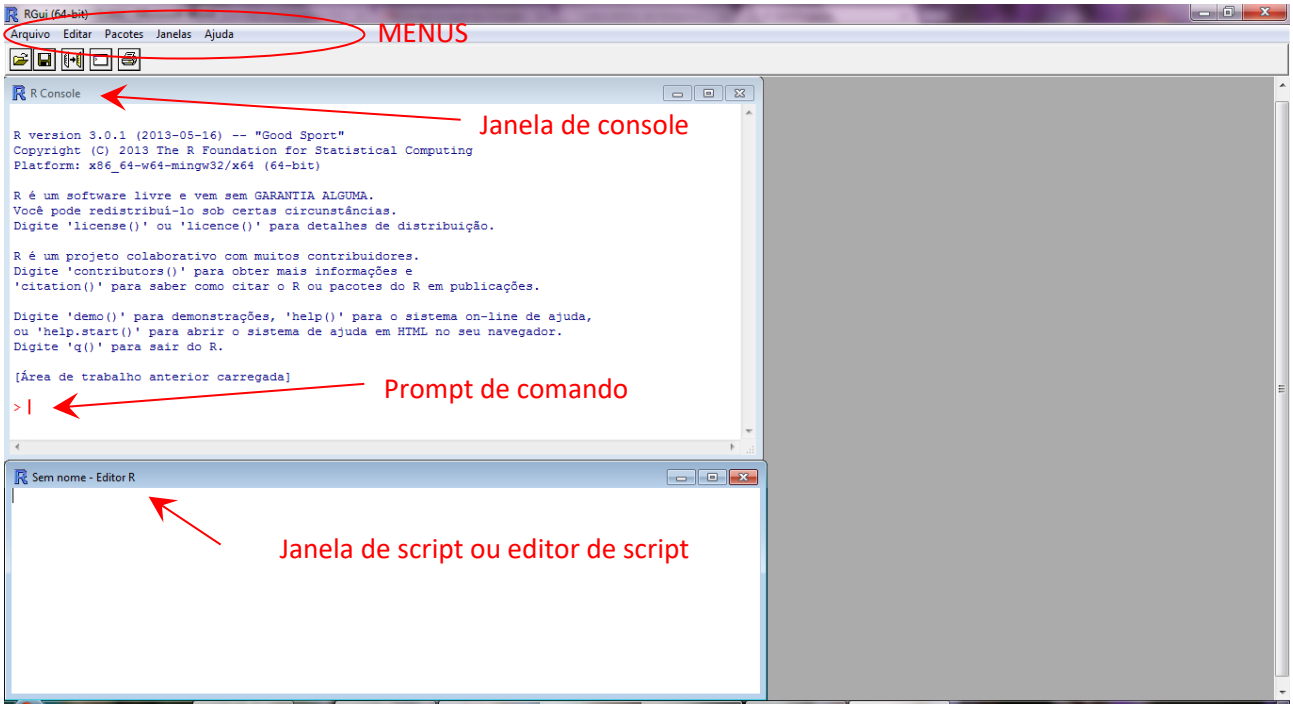
R é fornecido como um programa com interface de linhas de comando, que é o preferido por usuários experientes porque permite controle direto nos cálculos, é flexível e permite reprodutibilidade do plano de análise (a reprodutibilidade é o que faz com que essa abordagem seja considerada boa prática). No entanto, um bom conhecimento da linguagem é necessário. Portanto, a interface de linhas de comando pode ser intimidadora para os não iniciados. A curva de aprendizado para interface de linha de comandos é mais longa do que com interface gráfica, mas é reconhecida como um esforço recompensável e leva a melhores práticas (melhor compreensão do plano de análise; comandos facilmente salvos e substituídos e mantém uma rastreabilidade das edições realizadas). A interface com o usuário é a maior diferença entre o R e S-PLUS e outros programas de análise. Há iniciativas de interfaces gráficas para o R por parte de usuários como o R commander (Rcmdr), mas a maioria dos desenvolvedores dessas interfaces declara que essas são para familiarização da linguagem para o iniciante e provavelmente por isso nenhuma delas é completa (ou seja, nenhuma tem disponível todas as funções). Pessoalmente, a interface do RStudio é agradável, mas apenas porque torna a utilização da linha de comando mais agradável e amigável, não possuindo extensos menus com funções do tipo "point-and-click".

4. Instalação

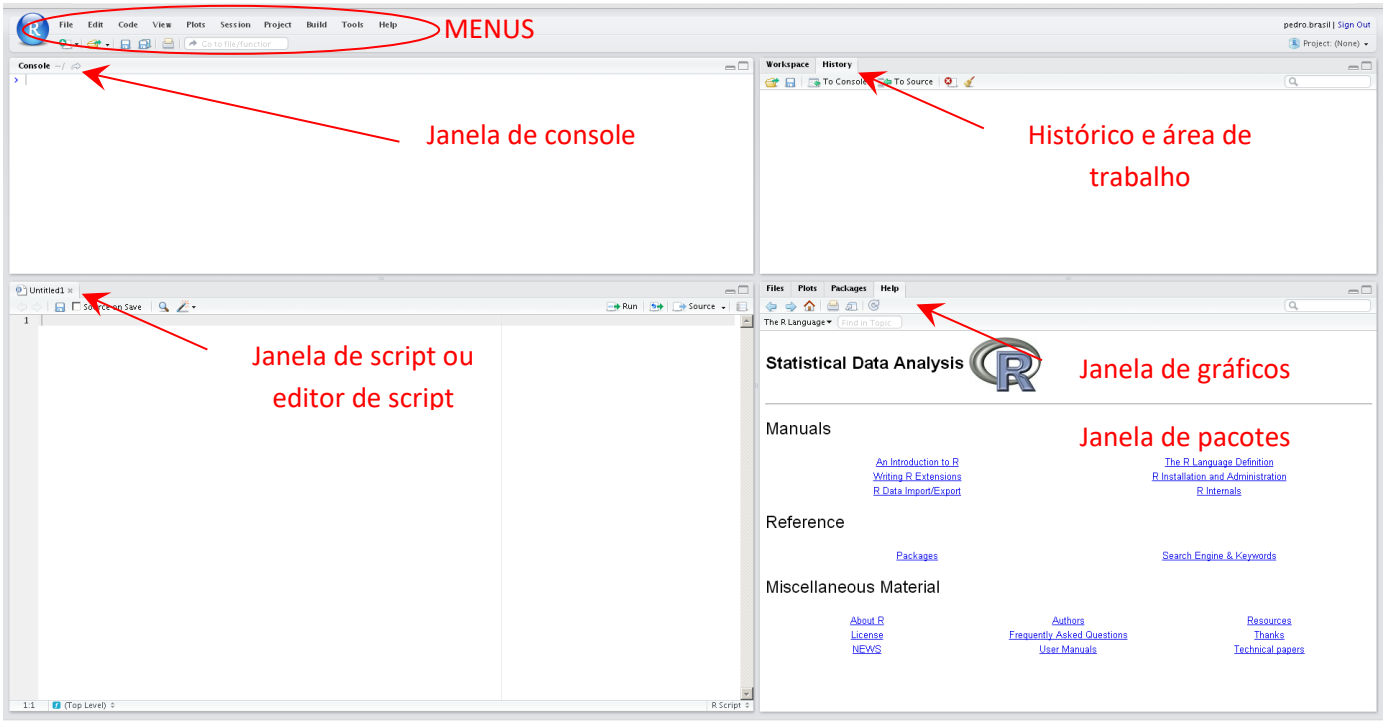
O aplicativo pode ser encontrado no espelho mais próximo do CRAN a partir de www.r-project.org. Há orientações claras e fáceis de como instalá-lo em diversos sistemas operacionais. Para instalá-lo em computadores com Windows, basta clicar duas vezes em cima do arquivo baixado do sitio e seguir as orientações da tela. Pessoalmente, a instalação padrão é boa. Para os que usam computadores multi-usuário e não são administradores da máquina (com privilégios de instalação) colar o diretório do R em qualquer pasta no computador ou num disco rígido externo também deve funcionar. Para usuários de Linux, há orientações no sitio para instalação, que são fáceis de seguir, no CRAN, sendo possível baixar os pacotes e instalá-los localmente ou adicionar o CRAN do r-project como repositório (por exemplo: http://cran.fiocruz.br/) e instalá-lo com o apt-get.

5. Iniciando

Se você baixou e instalou o R corretamente, quando você iniciar o programa clicando no ícone da área de trabalho, algo parecido com a imagem abaixo aparecerá! Repare que há alguns menus no topo da janela e há uma janela maior ("RGui"), e dentro desta uma menor chamada "R console" e um cursor piscando dentro dessa janela. Isso indica que o R está aguardando usuário escrever comandos. Se o usuário executar um gráfico, por exemplo, outra janela aparecerá dentro da janela maior. Outras janelas poderão aparecer e serem utilizadas de acordo os recursos desejados. Para executar qualquer tarefa é necessário que o usuário escreva um comando (função) no *prompt*. De forma alternativa, o usuário pode escrever o mesmo comando no editor de script e pedir para o R executar esse comando no *prompt* pelo atalho de teclado CTRL + R. A diferença é que pelo script o usuário edita e organiza as linhas de comando como lhe convêm, enquanto no console o histórico fica guardado pela ordem de execução. Pessoalmente, utilizo somente o editor de scripts. Para abrir o editor de scripts (com um novo ou já salvo script) utilize o menu "Arquivo". Todas as áreas de trabalho (olhar abaixo) possuem a extensão *.RData e os scripts possuem a extensão *.R.



Se você quiser experimentar o RStudio (para Linux, Windows ou Mac), baixe o programa de http://www.rstudio.com/. Se você instalou o RStudio, quando você abrir o RStudio, o R deverá parecer com algo como a imagem abaixo. Mais uma vez, o RStudio é uma interface que ajuda a organizar as janelas do R e a visualização dos seus recursos, mas não possui as funções em menus como por exemplo SPSS ou Stata. Lembrando que dentro do RStudio, para executar uma linha de comando do script o atalho é CTRL + Enter.



6. Entendendo o racional

Muito se fala que o R é um programa orientado a objetos e os iniciantes simplesmente não entendem. Isso quer dizer que o R possui uma área de trabalho na memória do computador, e o R é capaz de armazenar qualquer objeto nessa área de trabalho, inclusive banco de dados. Dessa forma, dá para perceber que o R pode trabalhar com vários objetos carregados simultaneamente, podendo estes ser bancos de dados, listas, vetores (sequência de valores), matrizes, tabelas etc, carregados simultaneamente, e pode carregá-los do disco ou salva-los no disco. Então qualquer modificação dos objetos na área de trabalho não modifica o que está gravado no disco até o usuário solicitar que isso seja feito. Isso não deve intimidar os iniciantes. Na verdade, não há necessidade de programar funções para execução de tarefas (apesar de ser possível dado a flexibilidade do programa), basta apenas que o usuário chame as funções

para que estas executem as tarefas desejadas. Essa flexibilidade permite que o usuário possa fazer as mesmas tarefas de diversas formas diferentes. Mas, devagar ...

O que é uma função? Aqui, função é sinônimo de comando. Função (usualmente aparece na documentação do R como FUN) é um conjunto de instruções que retorna um valor ou lista de valores, usualmente após uma sequência de operações ou cálculos, na forma mais simples apresenta-se com a seguinte estrutura:

```
>function(argument1,argument2,argument_n)
```

A função (ou comando) pode possuir um ou mais argumentos, ou se possuir argumentos padrão, pode não ser necessário inserir argumentos ou opções, porque os argumentos padrões serão utilizados (por exemplo `Sys.time()` retorna a hora do sistema operacional). Preste atenção que tanto as funções como seus argumentos devem ser escritos exatamente como esperados, ou seja, `Função` é diferente de `função`. Então argumentos são os elementos que a função pede para executar as operações necessárias, e opções são argumentos adicionais que podem interferir na execução das operações, modificando-as para que sejam feitas de forma diferente do padrão.

```
> log(base = 3, x = 2187) # é a mesma coisa que
[1] 7
> log(2187, 3)
[1] 7
> log(2187, 10) # mudando um dos argumentos - resultado diferente
[1] 3.339849
> log(2187) # o argumento base padrão é o log natural (2.718282)
[1] 7.690286
> -4:-7 # sequência de valores
[1] -4 -5 -6 -7
> abs(-4:-7)# muitas funções podem ter argumentos com diversos valores
```

```
[1] 4 5 6 7

> sqrt(abs(-4:-7))# muitas funções podem ser aninhadas em outras

[1] 2.000000 2.236068 2.449490 2.645751
```


Repare que se os argumentos entrarem na ordem esperada, o usuário não precisa indicar o que é cada argumento, mas o usuário pode modificar a ordem como achar conveniente, desde que indique que valor se aplica aos diferentes argumentos. O sinal de # no script indica que há quebra de linha no comando, assim o R não executa qualquer coisa após esse símbolo. Ainda, reparem que o número 1 sempre aparece entre colchetes na impressão no console. Esse número indica a posição do elemento dentro do objeto. Assim, supondo que o comando retorne 40 valores, e esses não caibam na mesma linha do console, quando R quebra a linha, ele retorna o número da ordem que inicia a linha. Por último, repare que as funções no R podem ser aninhadas umas às outras, desde que uma função retorne um valor aceitável por outra.

Para começar, chamemos uma função que demonstra alguns tipos de gráficos:

```
> demo(graphics)

> rm(list=ls()) # remove todos os objetos da área de trabalho
```

Repare que “graphics” é o nome de um pacote (conjunto de funções) que R possui para gerar gráficos. Esse é um dos pacotes mantidos pelo time do “core” do R. Há outros pacotes gráficos disponíveis. Há centenas e centenas de pacotes no R, que não vêm instalados como padrão, pois são na vasta maioria pacotes gerados por usuários.

O R pode funcionar como uma calculadora, de forma que o usuário digita as operações e a resposta retorna no console, ou o usuário pode pedir para armazenar a saída da função num objeto e depois pedir para imprimir no console, por exemplo:

```
> n <- 15

> n

[1] 15
```

```
> 5 -> n

> n

[1] 5

> 5 - n

[1] 0

> n

[1] 5

> x <- 1

> X <- 10

> x

[1] 1

> X

[1] 10

> 10 + 2

[1] 12

> n <- 10 + 2

> n

[1] 12

> n <- 3 + rnorm(1)

> n

[1] 2.208807

> pot <- 3^3

> pot

[1] 27

> div <- pot / 2

> div
```

```
[1] 13.5

> mult <- div * 10

> mult

[1] 135

> sub <- mult - 4

> sub

[1] 131

> seq <- 10:20

> seq

 [1] 10 11 12 13 14 15 16 17 18 19 20

> seq[4] # o [] fraciona o objeto

[1] 13

> seq[4:7]

[1] 13 14 15 16

> seq[c(4,7)]

[1] 13 16

> seq[-4:-7] # o - remove os elementos do objeto

[1] 10 11 12 17 18 19 20

> seq - 4

 [1]  6  7  8  9 10 11 12 13 14 15 16

> seq - c(4,2) # o R recicla os elementos para operações

 [1]  6  9  8 11 10 13 12 15 14 17 16

Mensagens de aviso perdidas:

In seq - c(4, 2) :

  comprimento do objeto maior não é múltiplo do comprimento do objeto menor

> c(-2,-9,-4,10) - c(4,6) # explicar melhor a reciclagem
```

```
[1]  -6 -15  -8   4

> n <- c("João","Maria","Carla") # objeto alfanumerico

> n

[1] "João"  "Maria" "Carla"

> seq <- c('1','2','3','20') # objeto alfanumerico

> seq

[1] "1"  "2"  "3"  "20"

> sort(seq) # ordenado alfabeticamente

[1] "1"  "2"  "20" "3"
```

As primeiras coisas que se deve aprender desse exemplo são os operadores: <- ou -> (atribuição); + (soma), - (subtração); * (multiplicação); / (divisão); == (igual), != (diferente); > (maior); < (menor); ^ (eleva a potência); : (define um intervalo). Estes são operadores fundamentais e algumas questões de precisão aritmética devem ser consideradas (`?Arithmetic`) . O sinal de = é equivalente ao <- na maioria das situações. Do exemplo acima vemos que o operador de atribuição deposita um valor qualquer num objeto. Há operadores que funcionam para extrair partes de objetos, os [ ]. Quando o objeto tem apenas uma dimensão (comprimento), funciona como o exemplo acima (haverá exemplos com mais dimensões abaixo). Então repare que para extrair o elemento na quarta posição deve-se seguir do objeto o "[4]", para extrair uma sequência basta inserir a sequência entre os colchetes e para extrair posições específicas não sequenciais é necessário concatenar as posições. Essa noção será muito importante para posteriormente podermos editar bancos dados. O R possui uma lógica de reutilização ou reciclagem. Repare que o vetor `seq` está sendo subtraído de um vetor com os valores 4 e 6. Como `seq()` e esse vetor possuem dimensões diferentes, o R subtrai 4 da primeira posição de `seq()`, 6 da segunda posição de `seq()` e então começa a reciclar o vetor c(4,6) até chegar na última posição de `seq()`.

Repare também que o R tem como comportamento padrão substituir os objetos sem aviso de substituição. No exemplo acima, o n recebeu o valor 15, posteriormente substituído por 5 e em seguida por 12 sem qualquer interrupção de avisos. Ainda, os objetos devem sempre ser iniciados com letras, podendo ser seguidas de números e outros caracteres (como . ou _). `rnorm()` é uma função que gera valores de uma distribuição normal e `c()` é uma função de concatenação ou de condições do objeto. Ainda, repare que se um objeto possui múltiplos valores, e operações são executadas com esses objetos, as operações são executadas para cada um dos valores no objeto. Mas depois de um tempo, como eu sei quais objetos estão carregados na área de trabalho?

```
> ls()
[1] "div"   "mult"  "n"     "nomes" "pot"   "seq"   "sub"   "x"     "X
> ls.str()
div :  num 13.5
mult :  num 135
n :  num 3.38
nomes :  chr [1:3] "João" "Maria" "Carla"
pot :  num 27
seq :  int [1:11] 10 11 12 13 14 15 16 17 18 19 ...
sub :  num 131
x :  num 1
X :  num 10
```

Repare que a função `ls()` lista os nomes dos objetos carregados na área de trabalho, mas não mostra o seu conteúdo, enquanto que a `ls.str()` mostra a classe (numérico, letras, datas etc) e uma breve descrição de seu conteúdo (dependendo do tamanho do conteúdo).

Para remover um objeto da área de trabalho basta:

```
> rm(div,mult,n,pot,seq,sub,x,X) ; ls()# o ; indica que o comando terminou
[1] "nomes"
> rm(list = ls()) # remove todos os objetos da área de trabalho
> ls()
character(0)
```

Operações adicionais com vetores ou outros objetos

Agora alguns exemplos de como manipular vetores serão demonstrados. Essa compreensão é fundamental para que o usuário consiga entender os exemplos de edição de dados mais adiante.

```
> peso <- c(62.5, 70.3, 52.1, 98.0, 90.2, 70.6)
> peso
[1] 62.5 70.3 52.1 98.0 90.2 70.6
> altura <- c(1.70, 1.82, 1.75, 1.94, 1.84, 1.61)
> altura
[1] 1.70 1.82 1.75 1.94 1.84 1.61
> imc <- peso/altura^2
> imc # a operação é realizada com um elemento na mesma ordem do outro vetor
[1] 21.62630 21.22328 17.01224 26.03890 26.64225 27.23660
```

Repare que o R executou uma operação respeitando a ordem dos elementos, ou seja, o primeiro peso foi dividido pelo quadrado da primeira altura e assim por diante até o último elemento. No entanto, uma das condições para que isso ocorra de forma razoável é que peso e altura possuem o mesmo comprimento:

```
> length(peso)
[1] 6
> length(altura)
[1] 6
> length(peso) == length(altura) # retorna um valor lógico
[1] TRUE
```

```
> length(peso) != length(altura) # retorna um valor lógico
[1] FALSE
```


Já deu para perceber nos exemplos acima que os números entre os colchetes representam índices que denotam a posição do elemento dentro do objeto. Com esses índices é possível adicionar, remover, substituir ou condicionar operações. Segue mais alguns exemplos:

```
> peso[4] <- 64 # peso na posição 4 é substituído
> peso
[1] 62.5 70.3 52.1 64.0 90.2 70.6 94.2
> altura[4:5] <- c(1.84,1.94) # elementos de altura na posição 4 e 5 são substituídos
> # altura[4:5] <- altura[5:4] # mesma coisa
> altura
[1] 1.70 1.82 1.75 1.84 1.94 1.61
> peso[8] <- 94.2 # adiciona mais elemento na posicao 8
# o mesmo poderia ser dito com o comando peso <- c(peso,NA,94.2)
> peso # Foi adicionado um elemento vazio representado por NA (NOT ASSIGNED ou missing) automaticamente, porque a posição 7 não foi especificada
[1] 62.5 70.3 52.1 64.0 90.2 70.6   NA 94.2
> is.na(peso) # Retorna TRUE onde está vazio
[1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
> !is.na(peso) # Retorna TRUE onde NÃO está vazio
[1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
> peso <- peso[!is.na(peso)] # remove todos os elementos que estão vazios
> peso
[1] 62.5 70.3 52.1 64.0 90.2 70.6 94.2
```

```
> peso > 72 # retorna TRUE onde os elementos são > 72
[1] FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE
> peso[peso > 72]
[1] 90.2 94.2
> rm(list=ls())
```


Repare que em um dos exemplos acima um valor ausente (missing) foi gerado, o `NA`. Cuidado com os valores `NA`, pois todas as operações com `NA` retornam `NA` e muitas funções não funcionam ou retornam erro quando há `NA` nos dados. Muitas possuem argumentos para ignorar os `NA`.

Os exemplos acima realizaram operações com valores lógicos (`TRUE` ou `FALSE`) e índices numéricos que indicam as posições dos elementos nos objetos. Mas também é possível adicionar nomes aos elementos dos objetos e fazer as operações acima com os nomes dos elementos ao invés dos índices das posições (nomes são atributos de objetos e serão comentados também mais adiante). Quando se executa operações com banco de dados é possível especificar as colunas pela posição, pelos nomes ou com valores lógicos. Seguem exemplos de operações com nomes:

```
> taxa <- c(2.85,3.10,10.13,11.86,8.15)
> names(taxa)
NULL
> names(taxa) <- c("Norte","Nordeste","Sudeste","Sul","C.Oeste")
> taxa
   Norte Nordeste  Sudeste      Sul  C.Oeste
    2.85     3.10    10.13    11.86     8.15
> names(taxa)
[1] "Norte"    "Nordeste" "Sudeste"  "Sul"      "C.Oeste"
> taxa['Sul'] # aspas simples funcionam como aspas duplas quase sempre
  Sul
11.86
```

```
> taxa[c("Sudeste","Sul")] # não é possível usar o - p/ excluir
Sudeste     Sul
  10.13   11.86
> taxa[-c("Sudeste","Sul")] # não é possível usar o "-" p/ excluir
Erro em -c("Sudeste", "Sul") : argumento inválido para operador unário
> taxa[!c("Sudeste","Sul")] # não é possível usar o "!" p/ excluir ERRO
Erro em !c("Sudeste", "Sul") : argumento de tipo inválido
> match(c("Sudeste","Sul"),names(taxa)) # mas é possível saber qual o índice desses nomes
[1] 3 4
> taxa[-match(c("Sudeste","Sul"),names(taxa))] # o menos exclui elementos do objeto
   Norte Nordeste  C.Oeste
    2.85     3.10     8.15
> pie(taxa) # os nomes são passados para o gráfico
> barplot(taxa) # os nomes são passados para o gráfico
```

7. Acessando a documentação de ajuda

O R possui uma documentação com uma estrutura muito particular. Há ajuda que é instalada juntamente com os pacotes já disponíveis, há ajuda que está em pacotes não instalados, é há o suporte de outros usuários. A documentação do R está sempre em HTML, ou seja, será sempre disponibilizada através do navegador. Com o `help.start()`, toda a documentação do R, bem como uma ferramenta de procura dentro dessa documentação é disponível.

No entanto, para achar uma documentação a partir do console é necessário saber exatamente o nome da função e o pacote da função precisa estar carregado. Caso contrário, é necessário explicitar que se deseja procurar termos dentro de toda documentação. Para buscar ajuda, é possível fazer de diversas formas:

```
> help.start() # Chama a documentação no navegador

If nothing happens, you should open

'http://127.0.0.1:14715/doc/html/index.html' yourself

> ?boxplot

> help(read.spss) # o erro aparece porque o foreign não está carregado

No documentation for 'read.spss' in specified packages and libraries:

you could try '??read.spss'

> ??read.spss # mesma coisa que help.search('read.spss')

> RSiteSearch("COX model extensions") # no sítio do R
```

Repare que na ajuda da função `boxplot`, por exemplo, há uma breve descrição, em seguida a estrutura da função com os possíveis argumentos. Adiante a descrição de cada um dos argumentos, detalhes, os valores que a função retorna, referências, funções semelhantes e os exemplos. É possível chamar os exemplos (a maioria das funções possui exemplos) a partir do console para ver como a função funciona.

```
> example(boxplot)

> demo() # retorna todos demos disponíveis

> demo(plotmath) # retorna um demo desse conjunto de funções de operadores
```

Alguns pacotes possuem uma demonstração, que usualmente são um conjunto de exemplos. A função demo retorna a execução de funções disponíveis no pacote. Ainda, é possível chamar a ajuda de funções ou pacotes por documentação extra que o autor disponibiliza em pdf ou de funções que não estão instaladas, as vinhetas.

```
> vignette(all = T) # procura por todas as vinhetas de pacotes instalados

> vignette("timedep") # carrega a vinheta do pacote survival
```

A vinheta é uma espécie de tutorial de como o autor usaria as diferentes funções no pacote. Usualmente é um documento de poucas páginas com exemplos comuns em situações comuns. Nem todos os pacotes possuem vinhetas.

A quantidade de pacotes para o R já muito grande e é continuamente crescente. Assim, o time do R organiza documentação comuns para determinados tópicos no TaskViews (http://cran.r-project.org/web/views/). Assim, fica mais fácil comparar funções de pacotes diferentes que fazem coisas semelhantes ou que tem o mesmo propósito.

Por último, se o usuário não conseguir resolver o seu problema com essas ajudas ou documentações, o usuário pode recorrer a lista de usuários que se ajudam mutuamente. As listas de usuários do R possuem pessoas muito experientes e pessoas iniciantes. Há duas regras para o bom uso de listas de e-mails de usuários: (1) seja educado; (2) no email não mande somente perguntas como "como faço para executar um boxplot?", mas seja específico, mostre que você já procurou a documentação, coloque um pedaço do script que você fez e as saídas com erros, possivelmente um pedaço dos dados ou um exemplo que possa ser reproduzido por outros usuários, de forma que outros entendam onde você está empacado. A lista de usuários no Brasil é r-br@listas.c3sl.ufpr.br, necessitando cadastro antecipado. Essa lista é bastante diversificada com gente de todas as áreas. Por isso, a maioria das mensagens pode não ser interessante. Pelo mesmo motivo, é uma lista bem movimentada com dezenas de mensagens diárias.

8. Atributos

Os objetos armazenados na área de trabalho também podem ser caracterizados por seus atributos, além do nome e do seu conteúdo. Se um vetor recebe o conteúdo 1, 2 e 3 (por exemplo: `c(1,2,3)`), esse pode significar diferentes categorias, ou número de vezes em que um evento ocorre. Assim, o R pode fazer operações diferenciadas dado os diferentes atributos de um objeto. Se o usuário solicitar um resumo de um objeto numérico, receberá a média, mediana etc. Caso seja um fator, a frequência de cada categoria será impressa no console. Quatro são os modos básicos: *numeric*, *character*, *complex*, e *logical*. Para visualizar os atributos de um objeto:

```
> x <- 1
> mode(x)
[1] "numeric"
```

```
> length(x)
[1] 1
> A <- "Gomphotherium"; compar <- TRUE; z <- 1i
> mode(A); mode(compar); mode(z)
[1] "character"
[1] "logical"
[1] "complex"
> class(A); class(compar); class(z) # aqui class é equivalente a mode
[1] "character"
[1] "logical"
[1] "complex"
> N <- 2.1e23
> N
[1] 2.1e+23
> x <- 5/0
> x
[1] Inf
> exp(x)
[1] Inf
> x - x
[1] NaN
```

Repare ainda nos exemplos acima que o R trabalha de forma razoável e consistente com valores complexos infinitos ou inexistentes que podem também ser utilizados em operações.

Caso o objeto seja alfanumérico (“character”), o conteúdo deve estar entre aspas. Caso contrário, o R retornará o conteúdo de um objeto. Mais uma vez, as aspas podem ser simples ou duplas, desde que quando se inicia com aspas simples deve-se terminar com aspas simples, e o mesmo ocorre com as aspas duplas. Se aspas fazem parte da expressão, essas podem ser ignoradas sendo precedidas por contra-barra.


```
> nome <- c("João","Maria","Carla")
> a <- "nome"
> a
[1] "nome"
> a <- nome
> a
[1] "João"  "Maria" "Carla"
> x <- "Double quotes \" delimitate R's strings."
> x
[1] "Double quotes \" delimitate R's strings."
> cat(x)
Double quotes " delimitate R's strings.
> x <- 'Double quotes " delimitate R\'s strings.'
> x
[1] "Double quotes \" delimitate R's strings."
> cat(x)
Double quotes " delimitate R's strings.
```

Há classes de objetos que são comuns tais como *vector*, *factor*, *array, matrix, data.frame,ts* e *list. vector* são objetos genéricos que armazenam números, letras, elementos complexos ou elementos lógicos. *factors* são objetos

que armazenam números ou letras, mas esses representam classes fixas pré-determinadas (podendo ser ordenadas ou não). *array* são vetores com uma, duas ou mais dimensões que podem ter atributos adicionais. Matrizes são *array* de duas dimensões. *data.frame* é o banco de dados, que pode ser composto por vetores e fatores, de diferentes modos, sempre com a mesma dimensão, e que pode ser considerado um caso especial de uma lista. *ts* é um banco de dados para séries temporais e por isso possuem atributos adicionais como datas e frequências. *list* é um objeto que pode armazenar qualquer outro objeto incluindo outras listas. Muito do raciocínio que se aplica aos bancos de dados se aplica também as listas. É interessante tentar entender as listas, pois a maioria das funções do R retorna valores dentro de listas. Há funções que permitem o usuário verificar como o objeto está armazenado como `class()`, mas também outras sobre classes específicas para verificar ou transformar objetos como `is.factor()` ou `as.factor()`. Isso é importante porque muitas funções genéricas do R executam operações diferentes ou retornam valores diferentes dependendo da classe do objeto.


```
> data <- as.matrix(cbind(c(1:10),c(31:40)),ncol=2)
> data
      [,1] [,2]
 [1,]    1   31
 [2,]    2   32
 [3,]    3   33
 [4,]    4   34
 [5,]    5   35
 [6,]    6   36
 [7,]    7   37
 [8,]    8   38
 [9,]    9   39
[10,]   10   40
> is.matrix(data)
[1] TRUE
> is.factor(data)
```

```
[1] FALSE

> is.data.frame(data)

[1] FALSE

> class(data)

[1] "matrix"

> data2 <- as.data.frame(data)

> data2

   V1 V2

1   1 31

2   2 32

3   3 33

4   4 34

5   5 35

6   6 36

7   7 37

8   8 38

9   9 39

10 10 40

> is.matrix(data2)

[1] FALSE

> is.factor(data2)

[1] FALSE

> is.data.frame(data2)

[1] TRUE

> class(data2)

[1] "data.frame"
```

```
> data3 <- as.ts(data)

> data3

Time Series:

Start = 1

End = 10

Frequency = 1

   Series 1 Series 2

 1        1       31

 2        2       32

 3        3       33

 4        4       34

 5        5       35

 6        6       36

 7        7       37

 8        8       38

 9        9       39

10       10       40

> is.data.frame(data3)

[1] FALSE

> is.matrix(data3)

[1] TRUE

> class(data3)

[1] "mts"    "ts"     "matrix"

> plot(data2)

> plot(data3)

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

```
>

> l <- list(letras = LETTERS[1:12], Aviso = "Listas permitem elementos com dimensões e atributos
diferentes", M = matrix(12:1, ncol=3), F = as.formula("x~y"))

> l

$letras

 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L"


$Aviso

[1] "Listas permitem elementos com dimensões e atributos diferentes"


$M

     [,1] [,2] [,3]

[1,]   12    8    4

[2,]   11    7    3

[3,]   10    6    2

[4,]    9    5    1


$F

x ~ y


> names(l)

[1] "letras" "Aviso"  "M"      "F"

> data <- as.list(data)

> names(data)

NULL

> is.data.frame(data)
```

```
[1] FALSE

> class(data)

[1] "list"

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


9. Pacotes com funções adicionais e caminho de procura

R possui um caminho ordenado de procura para as coisas que estão carregadas na memória. Assim, toda vez que um pacote adicional é carregado ele fica na frente de pacotes básicos do R como o base, utils, stats e graphics. Ainda, se um ou mais bancos de dados são anexados no caminho, o R procura sucessivamente no caminho as informações anexadas. Então para saber o que está no caminho de procura:

```
> search()

[1] ".GlobalEnv"        "package:stats"     "package:graphics"  "package:grDevices"

[5] "package:utils"     "package:datasets"  "package:methods"   "Autoloads"

[9] "package:base"
```

Repare que até o momento somente pacotes base estão carregados. Se pacotes adicionais forem carregados, eles aparecerão nessa lista. Para carregar um pacote:

```
> library(survival) # ou library('survival')

Carregando pacotes exigidos: splines

> search()

 [1] ".GlobalEnv"        "package:survival"  "package:splines"   "package:stats"

 [5] "package:graphics"  "package:grDevices" "package:utils"     "package:datasets"

 [9] "package:methods"   "Autoloads"         "package:base"
```

Repare que agora o pacote survival está carregado e aparece em primeiro lugar na lista de procura. Ainda, quando se carrega o pacote survival, o pacote splines também é carregado, isso ocorre porque o splines é uma dependência do survival. O que significa que uma ou mais funções do survival chama uma ou mais funções do pacote splines e por isso esse precisa estar carregado para o correto funcionamento do survival.

Para instalar um pacote novo utiliza-se:

```
> install.packages('rms') # install.packages() chama uma lista de pacotes no CRAN
```

Para instalar um pacote é preciso especificar de onde ele será instalado. O padrão é do CRAN mas o R e o RStudio possuem comportamentos diferentes. No caso do R, o espelho do CRAN precisa ser especificado pelo menos uma vez durante cada sessão. Se for a primeira vez da sessão, o R pergunta de qual espelho do CRAN (o mais interessante é que seja o local geograficamente mais próximo) o usuário deseja baixar o pacote através de uma janela. Essa escolha valerá para todas as instalações dessa sessão do R, ou até o usuário pedir para modificar o repositório. Ainda, se o usuário não especificar qualquer pacote, o R retorna uma lista de pacotes disponíveis. Já no RStudio, um repositório padrão já é especificado na instalação (e pode ser facilmente modificado pelo usuário através das preferências globais) e por isso não precisa ser especificado. Porém se o usuário não especificar um pacote válido, a função retorna erro. No formato padrão, essa função instala a biblioteca no diretório padrão com todas as dependências necessárias.

```
> library(rms)
Carregando pacotes exigidos: Hmisc
Carregando pacotes exigidos: lattice
Carregando pacotes exigidos: survival
Carregando pacotes exigidos: Formula
Carregando pacotes exigidos: ggplot2
```

```
Attaching package: 'Hmisc'

The following objects are masked from 'package:base':

    format.pval, units

Carregando pacotes exigidos: SparseM

Attaching package: 'SparseM'

The following object is masked from 'package:base':

    backsolve
```

Perceba que ao carregar o pacote rms, vários avisos aparecem. Quando objetos estão mascarados, significa que funções que possuem o mesmo nome (mas não necessariamente fazem a mesma coisa) estão em posições diferentes no caminho de procura. Então, se a partir de agora o usuário utilizar uma função chamada `format.pval()`, o R chamará essa função do pacote Hmisc e não do pacote base. Para desanexar pacotes é necessário que seja na ordem do `search()`, pois o R retorna um erro em desanexar pacotes que são dependências:

```
> detach("package:rms"); detach("package:SparseM"); detach("package:Hmisc");
detach("package:ggplot2"); detach("package:Formula"); detach("package:lattice");
detach("package:survival")
```

Um bizú para desanexar todos os pacotes com uma linha só, independente do que esteja carregado (assumindo que o usuário está usando RStudio):

```
> while( search()[2] != "tools:rstudio" ) detach(2) # se for no R natural, trocar "tools:rstudio"
por "stats"
```

Agora um exemplo de má prática que caiu em desuso ao longo do tempo. É possível anexar bancos de dados para que não haja necessidade de dizer que um determinado fator ou vetor está dentro desse ou daquele banco. Por exemplo:


```
> data <- data.frame(ID=1:10,Sex=rep(c('F','M'),5),Crea=rnorm(10,.9,.3))
> data
   ID Sex      Crea
1   1   F 1.0991194
2   2   M 0.9058879
3   3   F 0.7423035
4   4   M 1.0184502
5   5   F 0.5805014
6   6   M 1.3270718
7   7   F 0.5130607
8   8   M 0.7684874
9   9   F 0.6287209
10 10   M 0.3538246
> Crea # aqui retornará um erro porque Crea não é um objeto da área de trabalho
Erro: objeto 'Crea' não encontrado
> data$Crea # aqui o $ significa que Crea é um elemento dentro de data
 [1] 1.0991194 0.9058879 0.7423035 1.0184502 0.5805014 1.3270718 0.5130607
 [8] 0.7684874 0.6287209 0.3538246
> attach(data)
> Crea # agora deu certo porque data está no caminho de procura do R
 [1] 1.0991194 0.9058879 0.7423035 1.0184502 0.5805014 1.3270718 0.5130607
```

```
 [8] 0.7684874 0.6287209 0.3538246

> search()

 [1] ".GlobalEnv"        "data"              "package:stats"

 [4] "package:graphics"  "package:grDevices" "package:utils"

 [7] "package:datasets"  "package:methods"   "Autoloads"

[10] "package:base"

> ls()

[1] "data"
```


Reparem que agora `data` está dentro do caminho de procura, por ter sido anexado com a função `attach()`. Quando solicitamos que o Crea (uma variável do banco) seja retornada, só funciona após anexar o banco no caminho. Para ter no console a mesma informação sem anexar o banco, seria necessário usar o `data$` para que o R entenda que Crea está dentro do banco `data`.

```
> data$DisfRenal <- ifelse(Crea > 0.9,'Sim','Não')

> DisfRenal # mas o banco estava carregado, não?

Erro: objeto 'DisfRenal' não encontrado

> attach(data) # Ops ... :(   seria necessário desanexar o banco antes

The following objects are masked from data (position 3):


    Crea, ID, Sex

> DisfRenal

 [1] "Não" "Não" "Não" "Não" "Sim" "Sim" "Não" "Não" "Não" "Não"

> search()

 [1] ".GlobalEnv"        "data"              "data"

 [4] "package:stats"     "package:graphics"  "package:grDevices"

 [7] "package:utils"     "package:datasets"  "package:methods"
```

```
[10] "Autoloads"         "package:base"

> rm(data) # remova da area de trabalho mas não do caminho

> Crea

 [1] 0.8039623 0.6598072 0.5015922 0.8818928 1.1264821 1.2544496 0.6608462

 [8] 0.8563744 0.6201631 0.6911771

> Sex

 [1] F M F M F M F M F M

Levels: F M

> DisfRenal

 [1] "Não" "Não" "Não" "Não" "Sim" "Sim" "Não" "Não" "Não" "Não"

> ls()

character(0)

> detach(data) ; detach(data) # necessidade de desanexar muitas vezes

> search()

[1] ".GlobalEnv"        "package:stats"     "package:graphics"

[4] "package:grDevices" "package:utils"     "package:datasets"

[7] "package:methods"   "Autoloads"         "package:base"

>
```

Se houver edições depois de anexar o banco, as edições posteriores não entram no caminho mesmo que o banco esteja já esteja anexado. Dessa forma haveria necessidade de desanexar e anexar todas as ocasiões de edições, mas essa é uma pratica ruim. A função `with()` é uma pratica alternativa nessas situações que poderia funcionar, porém a experiência diz que anexar bancos não é bom. Repare que mesmo removendo o banco da área de trabalho, o banco não é removido do caminho de procura podendo o usuário utilizar as variáveis que estão no caminho de procura. Se o usuário anexar o mesmo banco mais de uma vez, o banco aparecerá mais de uma vez no caminho de procura, podendo ocorrer que algumas edições estão em um banco e outras em outro banco.

Por isso, NÃO recomendo utilizar o `attach`, mas sempre definir os bancos (ou listas, ou matrizes) com uma única letra, de tal forma que minimiza a quantidade de digitação e facilita os comandos.

10. Importar dados de outros formatos para o R

Antes de importar ou exportar dados é necessário verificar se o R irá carregar ou salvar os bancos do diretório onde eles estão. Para isso, as funções `getwd()` e `setwd()` verifica e modifica o diretório de trabalho respectivamente. Prestar atenção que o endereço do diretório deve ser explicitado entre aspas e que as barras seguem a lógica do Linux e não as contra-barras do Windows. Especificar o de diretório com o `setwd()` é boa pratica, mesmo assim acredito que especificar o caminho completo e nome do arquivo é prática mais adequada.

```
> getwd() # um dos poucos comandos que há no menu
[1] "C:/Banco"
> setwd('c:/banco/curso_R') # o mesmo que setwd('c:\\banco\\curso_R')
> getwd()
[1] "c:/banco/curso_R"
```

A função original do R que salva a área de trabalho com todos os objetos é `save.image()` – essa é um envelope da função `save()` para salvar todos os objetos da área de trabalho de uma vez, já que com a `save()` é preciso especificar um ou mais objetos a serem salvos - e `load()` carrega a área de trabalho. Há diversas funções no pacote de utilidades que permite escrever e ler dados de arquivos texto em formatos como txt, csv, com diferentes separadores e diversas outras opções. O pacote do core do R que importa/exporta dados de/para diversos formatos (Stata, SPSS, SAS etc) é o foreign. Há outros pacotes que possuem a mesma funcionalidade de importação e exportação, inclusive para formatos adicionais, com detalhes diferentes como `epiDisplay::use` e `Hmisc::spss.get` (`sas.get` ou `stata.get`). As funções de importação do pacote 'Hmisc' geralmente são mais amigáveis em algumas situações, como, por exemplo, para importar bancos com datas do formato SPSS. Para importar/exportar de planilhas de xls ou xlsx há necessidade de instalar um pacote adicional (a semelhança do Hmisc ou do epiDisplay). Há diversos disponíveis, mas a sugestão é instalar o pacote 'readxl', já que diferente de outros pacotes que fazem essa tarefa, esse pacote não possui dependência e nem instalação de programas adicionais como o Java - JDK (Java Development Kit) e o JRE (Java Runtime Enviroment). Uma solução alternativa, seria abrir a planilha no programa original, salvar os dados como csv, e posteriormente importar para o R utilizando-se o `read.csv()` ou

`read.csv2()`. Atenção com os bancos em csv, já que há os formatos latinos e norte-americano onde um é separado por ";" e o outro por ",". que correspondem ao `read.csv()` ou `read.csv2()`, respectivamente. Só dá para saber como as caselas estão separadas abrindo o arquivo com bloco de notas e olhando ou na tentativa e erro. Se a fonte que salvou o arquivo em primeiro lugar informou qual o separador também serve.

A seguir alguns exemplos de importar os dados 'MS' nos diferentes formatos: RData, csv, xls e sav. Lembre que toda vez que executar uma operação verifique se o objeto ficou como planejado. Importe os dados para objetos de apenas uma letra, pois facilitará a edição de dados posteriormente.

```
> # Exemplo genérico

> setwd("C:\banco\projeto\")

a <- read.csv2("meusdados.csv") # os dados precisam ser depositados num objeto

> a <- read.csv2("C:\banco\projeto\meusdados.csv") # igual ao anterior especificando o caminho completo
```

A função `View()` permite visualizar os dados como em um uma planilha, porém não permite edições. Para editar dados, a sugestão é fazê-lo pelo script, nunca por planilha. Outras funções que podem ajudar a olhar o banco depois da importação e verificar se deu certo são `head()`, `tail()`, `summary()`, `str()` e `Hmisc::describe()`.

```
>setwd('c:/banco/curso_R')

>a <- read.csv2('MS.csv') # Banco em formato texto separado por vírgula.

>View(a)

>head(a) # mostra as seis primeiras linhas

>tail(a,10) # mostra as cinco últimas linhas

>summary(a) # summary funciona pra quase todo tipo de objeto

>str(a) # descrive inclusive os atributos

>#  No pacote Hmisc

>library('Hmisc') # somente se estiver instalado

>page(describe(a),'print') # no Rstudio page() possui um bug
```

```
>#  Pode-se salvar a saída do page num arquivo texto para posterior visualização

>
```

A partir de outros formatos


```
>library('foreign') # carrega pacote

>?read.spss

>b <- read.spss('MS.sav', to.data.frame = TRUE)

>View(b)

>

>library('Hmisc') # carrega pacote

>d <- spss.get('MS.sav')

>#  install.packages('readxl') # instala um novo pacote

>library('readxl') #

>?read_excel

>z <- read_excel('MS.xls')

>View(z)

>

>#  save.image('c:/banco/curso_R/MS.RData')

>rm(list = ls()) ; ls()

>load('c:/banco/curso_R/MS.RData') ; ls()
```

Muitos pacotes possuem bancos de dados como exemplos, e há um pacote somente com banco de dados de exemplos, o 'dataset'. Para carregar um banco de um pacote já carregado:

```
> data('iris') # dados no pacote dataset

>?iris
```

As etiquetas das variáveis não são passadas automaticamente para as tabelas e gráficos na maioria dos pacotes. Mesmo assim, é interessante mantermos as etiquetas para que as descrições das variáveis sejam uniformes nas tabelas e gráficos quando for possível passa-las. As funções do foreign importam etiquetas de formatos como o SAS, SPSS e Stata e as deposita como atributos do banco. Algumas funções do pacote epiDisplay usam as etiquetas importadas por essas funções. O pacote Hmisc (`?Hmisc::spss.get, ?Hmisc::sas.get, ?Hmisc::stata.get`) importa as etiquetas e as deposita como atributos da variável, e muitas funções do Hmsic e rms usam essas as etiquetas. Algumas funções de gráficos e tabelas de outros pacotes permitem entradas de expressões para fins de descrição que poderiam receber o valor das etiquetas. Assim, o usuário deve prestar atenção ao importar os dados com funções desses pacotes e utilizar as funções desses pacotes (respectivamente) para conduzir suas análises, considerando que as etiquetas aparecerão nas tabelas e gráficos se estiverem no lugar esperado pelas respectivas funções. Se os dados não possuírem etiquetas, há diferentes formas de inserir etiquetas no banco, sendo a mais fácil a função `Hmisc::label` que insere uma etiqueta como atributo da coluna do banco. As etiquetas podem ser armazenadas diretamente como atributos do banco. Uma prática razoável é manter um script que define as etiquetas ou manter um objeto que mantenha as etiquetas relacionadas com os nomes das respetivas colunas. Em seções abaixo haverá alguns exemplos de tabelas que usarão etiquetas.

```
> # Mostrando atributos do banco importado com o foreign::read.spss

> attributes(b)

$names

[1] "PATIENT"  "SEX"      "AGE_AT_O" "ONSET_TI" "GROUP"


$class

[1] "data.frame"


$row.names

 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
32


$variable.labels

       PATIENT            SEX       AGE_AT_O       ONSET_TI          GROUP
```

```
           ""             "" "Age at onset"   "Onset time"             ""
```



```
> attr(b, "variable.labels")
       PATIENT            SEX       AGE_AT_O       ONSET_TI          GROUP
           ""             "" "Age at onset"   "Onset time"             ""
> # Manipulação e edição de dados podem descartar as etiquetas.
> b2 <- subset(b, select = c("SEX","GROUP"))
> attributes(b2)
$names
[1] "SEX"   "GROUP"


$row.names
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
32


$class
[1] "data.frame"


> rm(b2)
>
> # Mostrando atributos do banco importado com o Hmisc::spss.get
> attributes(e)
$names
[1] "PATIENT"  "SEX"      "AGE.AT.O" "ONSET.TI" "GROUP"


$row.names
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
32
```



```
$class

[1] "data.frame"


> label(e)

       PATIENT            SEX       AGE.AT.O       ONSET.TI          GROUP

            ""             "" "Age at onset"   "Onset time"             ""

> # Armazenado como atributo da coluna é menos sucetível a perdas

> attributes(e$AGE.AT.O)

$label

      AGE_AT_O

"Age at onset"


$class

[1] "labelled" "numeric"


> label(e$AGE.AT.O)

      AGE_AT_O

"Age at onset"

> attr(e$AGE.AT.O, "label")

      AGE_AT_O

"Age at onset"

>

> # Adicionanando etiquetas com a função do Hmisc
```

```
> label(e$PATIENT) <- "Patient ID"
> # Mesma coisa que
> # attr(e$PATIENT, "label") <- "Patient ID"
> label(e$SEX) <- "Sex"
> label(e$GROUP) <- "Comparison group"
> label(e)
           PATIENT                SEX           AGE.AT.O           ONSET.TI              GROUP
      "Patient ID"              "Sex"     "Age at onset"       "Onset time" "Comparison group"
>
> # Adicionando etiquetas como atributos do banco
> attr(b, "variable.labels")
       PATIENT            SEX       AGE_AT_O       ONSET_TI          GROUP
            ""             "" "Age at onset"   "Onset time"             ""
>  attr(b,  "variable.labels")[c("PATIENT","SEX","GROUP")]  <-  c("Patient  ID","Sex","Comparison
group")
> attr(b, "variable.labels")
           PATIENT                SEX           AGE_AT_O           ONSET_TI              GROUP
      "Patient ID"              "Sex"     "Age at onset"       "Onset time" "Comparison group"
>
> # Amazenando as etiquetas em um objeto com Hmisc::label (atributos da coluna)
> etiquetas <- label(e)
> etiquetas
           PATIENT                SEX           AGE.AT.O           ONSET.TI              GROUP
      "Patient ID"              "Sex"     "Age at onset"       "Onset time" "Comparison group"
>
> # Amazenando as etiquetas em um objeto a partir dos atributos do banco
```

```
> etiquetas <- attr(b, "variable.labels")

> etiquetas

           PATIENT                SEX            AGE_AT_O           ONSET_TI                GROUP

      "Patient ID"              "Sex"     "Age at onset"       "Onset time" "Comparison group"
```


11. Tipos de variáveis

Como comentado anteriormente, os bancos de dados são objetos de duas dimensões (linhas e colunas) e podem ser considerados casos especiais de listas, pois a semelhança das listas cada uma de suas colunas pode ter atributos e formatos diferentes, porém diferente das listas, todas as colunas (objetos) dentro do banco possuem o mesmo comprimento. As matrizes possuem todos os elementos com os mesmos atributos, e só podem ser editadas e manipuladas indicando os índices de linhas e colunas, já no banco de dados, além do usuário poder se referir ao índice da linha e da coluna para edição, adição ou extração, também pode se referir ao nome da linha ou, principalmente, da coluna. As classes ou formatos das variáveis em banco de dados podem ser os mesmos de um vetor, como comentado acima.

Uma questão sempre comentada em outros programas de análise são as etiquetas (ou rótulos, ou labels em Inglês) das variáveis e dos valores. Os rótulos dos valores simplesmente não fazem sentido no R, por conta da utilização de fatores e fatores ordenados. Na verdade, o R substitui os valores numéricos das variáveis pelas etiquetas e os mantém ordenadas com a mesma ordem numérica original como um atributo (levels) do objeto fator. Assim, não há no R valores numéricos que representariam classes ou classes ordenadas, mas as próprias classes são representadas.

Os principais tipos de variáveis:

character é um formato de dados em que o conteúdo é armazenado somente como letras. Assim mesmo que haja números ou datas na informação da variável, essas serão tratadas como se fossem palavras. *numeric* são variáveis tratadas como números. Há algumas variações como *double* (objetos numéricos com maior precisão numérica), *integer* (números inteiros) etc. *factor* são variáveis que apresentam classes que podem ser ordenadas ou não. A principal diferença entre esse formato e *character* é que a classes são fixas, então mesmo que não haja nenhum paciente com grupo sanguíneo A, a classe estará presente nas tabelas. Além disso, quando uma variável está no formato *factor*, para

inserir um novo valor na variável é preciso definir uma nova classe (ou nível), e há sempre uma classe de referência mesmo que as classes não sejam ordenadas. Se uma tabela é gerada a partir de uma variável *character*, a saída é em ordem alfabética, e se for de uma variável *factor* a saída é na ordem das classes. O mesmo ocorre quando variáveis entram em modelos de regressão e etc. *Date* obviamente é uma variável que armazena datas. No entanto, como todos os programas de estatística, esse tipo variável possui uma informação numérica para permitir operações, e por isso ela pode ser formatada de diversas formas, inclusive com a presença de horas e minutos. Gerando um banco de dados simulados para demonstração:


```
> ID <- 1:20

> Nome <-
paste(sample(c("Roberto","Mauro","Carla","Paula","Raquel"),20,T),sample(c("Ferreira","Silva","Amo
rim","Prado","Figueira"),20,T))

> Peso <- sample(seq(60.0,110.0,0.1),20,F)

> Altura <- sample(seq(1.60,2.05,0.01),20,F)

> IMC <- as.factor(Peso / as.numeric(Altura)^2)

> dia <- sample(1:30,20,T)

> mes <- sample(1:12,20,T)

> ano <- sample(2006:2009,20,T)

> data <- paste0(ano,'-',mes,'-',dia)

> data2 <- as.Date(paste0(ano,'-',mes,'-',dia))

> data3 <- as.double(data2)

> abo <- sample(c('A','B','O','AB'),20,T)

> abo2 <- as.factor(abo)

> b <- data.frame(ID,Nome,Peso,Altura,IMC,data,data2,data3,abo,abo2)

> b$Altura <- as.character(b$Altura)

> b$abo <- as.character(b$abo)

> b$abo2 <- relevel(b$abo2 ,ref='O')

> b[1:7,]
```

```
  ID           Nome  Peso Altura               IMC       data      data2 data3 abo abo2
1  1    Carla Silva  84.2   1.82 25.4196353097452 2008-11-30 2008-11-30 14213   A    A
2  2 Carla Figueira  73.4   1.98 18.7225793286399  2006-4-17 2006-04-17 13255   A    A
3  3    Mauro Prado  98.8      2             24.7  2007-6-25 2007-06-25 13689  AB   AB
4  4   Raquel Silva  97.1   1.97 25.0199695946816  2009-6-22 2009-06-22 14417  AB   AB
5  5    Mauro Silva 109.1   1.94  28.988202784568  2009-2-11 2009-02-11 14286   B    B
6  6  Raquel Amorim  70.1   1.61 27.0437097334208   2007-3-5 2007-03-05 13577  AB   AB
7  7    Carla Prado  81.6   1.78 25.7543239489963 2006-10-20 2006-10-20 13441  AB   AB
>
> class(b$Peso)
[1] "numeric"
> b$Peso[1:10]
 [1]  84.2  73.4  98.8  97.1 109.1  70.1  81.6  72.5  92.2 102.7
>
> class(b$Altura)
[1] "character"
> b$Altura[1:10]
 [1] "1.82" "1.98" "2"    "1.97" "1.94" "1.61" "1.78" "1.76" "2.01" "1.77"
>
> class(IMC)
[1] "factor"
> b$IMC[1:10]
 [1] 25.4196353097452 18.7225793286399 24.7             25.0199695946816 28.988202784568
 [6] 27.0437097334208 25.7543239489963 23.4052169421488 22.8212172966016 32.7811293051167
20 Levels: 16.3398692810458 18.7225793286399 22.4880642361111 ... 37.8892733564014
>
```

```
> class(b$data)
[1] "factor"
> b$data[1:5]
[1] 2008-11-30 2006-4-17  2007-6-25  2009-6-22  2009-2-11
20 Levels: 2006-1-2 2006-10-20 2006-2-23 2006-4-17 2007-3-5 2007-6-16 2007-6-25 ... 2009-
7-3
> b$data2[1:5]
[1] "2008-11-30" "2006-04-17" "2007-06-25" "2009-06-22" "2009-02-11"
>
> class(b$data2)
[1] "Date"
> b$data3[1:5]
[1] 14213 13255 13689 14417 14286
>
> class(b$data3)
[1] "numeric"
> class(b$abo)
[1] "character"
> table(b$abo)

 A AB  B  O
 8  6  3  3
> levels(b$abo)
NULL
>
> class(b$abo2)
```

```
[1] "factor"

> table(b$abo2)


 O  A AB  B

 3  8  6  3

> levels(b$abo2)

[1] "O"  "A"  "AB" "B"

>
```


12. Conteúdo do Exercício 1 foi coberto. Proceder para a avaliação.

Lembrar que os exercícios devem ser executados na plataforma moodle até a data da sessão presencial para que este mesmo exercício possa ser corrigido em aula

13. Edição de dados

A principal dica para editar dados antes de uma análise é: mantenha scripts separados, um para edição e outro para análise. Preferencialmente mantenha um script para cada tipo de análise ou para cada objetivo do projeto. Outra dica é manter os scripts, os dados e os resultados em pastas diferentes dentro da pasta do mesmo projeto. Talvez não seja o caso dos exemplos na apostila, no entanto à medida que o script vai aumentando de tamanho e a quantidade de arquivos do mesmo projeto vai aumentando as coisas podem ficar confusas. Assim, também é interessante manter comentários no script com as coisas que foram pensadas no momento da edição/análise ou separando blocos da edição. O RStudio possui um recurso interessante pra quem analisa dados de mais de um projeto que é a pasta de projetos (botão no alto a direita). Quando se salva um projeto, o RStudio já abre a sessão seguinte na pasta do projeto e os arquivos do projeto abertos na última sessão.

Todas as variáveis de bancos de dados podem ser manipuladas como os vetores e fatores dos exemplos anteriores. Um dos problemas mais comuns que quase sempre ocorre e que faz a edição necessária é que as variáveis podem não estar no formato necessário, por exemplo, números ou datas que são importados como letras, letras que

são importadas como fatores etc. Outra coisa que quase sempre necessita de edição do banco é necessidade de recodificação de variáveis ou remoção de dados ausentes. Sempre que for editar variáveis, teste antes para ver se o comando retorna a saída desejada (possivelmente em uma parte menor dos dados se os dados originais forem muito grandes) e teste depois das edições para verificar que a função depositou os valores esperados. Nessa seção haverá demonstrações de como executar edições comuns necessárias para posterior análise. Para isso vamos importar os dados MS2.csv para um objeto chamado `a`:


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

>

> # Importar o MS2 para o a

> setwd('c:/banco/curso_R')

> a <- read.csv2('MS2.csv') # csv2 é relacionado com a pontuação latina vs inglesa

> head(a)

  X Patient Sex Age.at.onset Onset.time Group      Data        bebidas morte                tabaco

1 1       1   M           30          2     A   6/7/2009 cachaça,cerveja     0     Sim

2 2       2   M           15          3     A  27/5/2011 cerveja,cachaça     0 Não

3 3       3   M        24.00          3     A   4/9/2009          cerveja     1     Sim

4 4       4   M           48          3     A 28/12/2010          cachaça     0 Não

5 5       5   F           19          4     A  3/11/2009 cachaça,cerveja     1     Sim

6 6       6   F           44          4     B  3/12/2009 cachaça,cerveja     0 Não

> tail(a)

    X Patient Sex Age.at.onset Onset.time Group      Data        bebidas morte                tabaco

27 27      27   M           40         20     ?  25/8/2011 cachaça,cerveja     0

28 28      28   F           27         24     B  24/9/2011          cachaça     1 Não

29 29      29   F           21         27     B 24/10/2011 cerveja,cachaça     0 Não

30 30      30   F           17         28     A 23/11/2011          cerveja     0 Não

31 31      31   F           20         29     A 23/12/2011 cerveja,cachaça     1 Não
```

```
32 32      32    F           33          32      B  22/1/2012          cachaça      0      Sim

> str(a)

'data.frame':32 obs. of  10 variables:

 $ X           : int  1 2 3 4 5 6 7 8 9 10 ...

 $ Patient     : Factor w/ 32 levels "1","10","11",..: 1 12 23 27 28 29 30 31 32 2 ...

 $ Sex         : Factor w/ 6 levels "","  M","f","F",..: 6 6 6 6 4 4 4 6 6 4 ...

 $ Age.at.onset: Factor w/ 25 levels "","14","15","16",..: 16 3 11 25 7 24 21 10 20 13 ...

 $ Onset.time  : int  2 3 3 3 4 4 4 5 5 5 ...

 $ Group       : Factor w/ 5 levels " B","?","A","aa",..: 3 3 3 3 3 5 1 3 3 3 ...

 $ Data        : Factor w/ 26 levels "","1/2/2010",..: 26 16 25 18 21 22 18 2 23 7 ...

 $ bebidas     : Factor w/ 5 levels "","cachaça","cachaça,cerveja",..: 3 5 4 2 3 3 4 5 2 5 ...

 $ morte       : int  0 0 1 0 1 0 1 1 0 1 ...

 $ tabaco      : Factor w/ 3 levels "","Não                      ",..: 3 2 3 2 3 2 3 3 2 3 ...

>

> a$Patient[1:6]

[1] 1 2 3 4 5 6

Levels: 1 10 11 12 13 14 15 16. 17 18 19 2 20 21 22 23 24 25 26 27 28 29 3 30 31 32 4 5 6 7 8 9

> a$Patient <- as.numeric(as.character(a$Patient))

> a$Patient[1:6]

[1] 1 2 3 4 5 6

>

> a$Sex[1:6]

[1] M M M M F F

Levels:   M f F m M

> levels(a$Sex) # verifica os níeveis da variável

[1] ""   " M" "f"  "F"  "m"  "M"
```

```
> levels(a$Sex) <- c(NA,"Masculino","Feminino","Feminino","Masculino","Masculino")
> table(a$Sex)

Masculino  Feminino
       14        16
>
> a$morte[1:6]
[1] 0 0 1 0 1 0
> a$morte <- factor(a$morte, label = c('Censura','Obito'))
> table(a$morte)

Censura   Obito
     17      15
>
> a$tabaco[1:6]
[1] Sim                     Não                     Sim                     Não
[5] Sim                     Não
Levels:  Não                    Sim
> str(a$tabaco)
 Factor w/ 3 levels "","Não                    ",..: 3 2 3 2 3 2 3 3 2 3 ...
> levels(a$tabaco) <- trimws(levels(a$tabaco)) # Se fizer direto na variável não retorna fator
> table(a$tabaco)

    Não Sim
  3  16  13
> # levels(a$tabaco) <- c(NA,'Não','Sim') # mesma coisa que
```

```
> # levels(a$tabaco)[1] <- NA # mema coisa que
> a$tabaco <- droplevels(a$tabaco, exclude = "") # Prefiro essa
> table(a$tabaco, useNA = "ifany")

 Não  Sim <NA>
  16   13    3
>
> colnames(a)
 [1] "X"           "Patient"     "Sex"         "Age.at.onset" "Onset.time"   "Group"       "Data"
 [8] "bebidas"      "morte"        "tabaco"
> names(a)[3] # o mesmo que colnames(a)[3]
[1] "Sex"
> names(a)[4] <- 'Age' # modifica o nome da variável para algo mais curto
> a$Age[1:6]
[1] 30    15    24.00 48    19    44
Levels:  14 15 16 17 18 19 20 21 24 24.00 25 26 27 29 30 32 33 37 38 39 40 42 44 48
> a$Age <- as.character(a$Age) # trnaforma para character
> a$Age <- as.numeric(gsub(',','.',a$Age)) # substitui as virgulas por pontos e trnasforma para
numeros
> head(a$Age)
[1] 30 15 24 48 19 44
>
> a$Onset.time[1:6]
[1] 2 3 3 3 4 4
> a$tempo <- as.factor(ifelse(a$Onset.time > 10,'Tardio','Precoce')) # cria uma nova variável
> levels(a$tempo) # Nesse tipo de transformação o níveis sempre entram em ordem alfabetica
```

```
[1] "Precoce" "Tardio"

> table(a$tempo)


Precoce  Tardio

     15      16

> a$Onset.time[1:6]

[1] 2 3 3 3 4 4

>

> a$Group[1:6]

[1] A A A A A B

Levels:  B ? A aa B

> levels(a$Group)

[1] " B" "?"  "A"  "aa" "B"

# Opção mais simples

# levels(a$Group) <- c('B',NA,'A','A','B')*

# edição de forma condicional é sempre mais seguro

levels(a$Group)[levels(a$Group) == "aa"] <- "A"

levels(a$Group)[levels(a$Group) == "?"] <- NA

levels(a$Group) <- trimws(levels(a$Group))

a$Group <- relevel(a$Group,ref='A')

> levels(a$Group)

[1] "A" "B"

> table(a$Group)


 A  B

24  7
```

```
>
> a$Data[1:6]
[1] 6/7/2009   27/5/2011  4/9/2009   28/12/2010 3/11/2009  3/12/2009
26 Levels:  1/2/2010 1/6/2010 1/7/2010 2/5/2010 22/1/2012 23/11/2011 23/12/2011 24/10/2011 ...
6/7/2009
> a$Data <- as.Date(a$Data,format = '%d/%m/%Y')
> a$Data[1:6]
[1] "2009-07-06" "2011-05-27" "2009-09-04" "2010-12-28" "2009-11-03" "2009-12-03"
>
> a$bebida[1:6]
[1] cachaça,cerveja cerveja,cachaça cerveja        cachaça        cachaça,cerveja cachaça,cerveja
Levels:  cachaça cachaça,cerveja cerveja cerveja,cachaça
> a$bebida <- as.character(a$bebida)
> table(a$bebida)

                        cachaça cachaça,cerveja         cerveja cerveja,cachaça
             1                5               8               7              11
> a$bebida <- trimws(a$bebida)
> a$bebida[a$bebida == ''] <- NA # transforma os espaços em branco em NA
> a$cachaca <- ifelse(grepl('cachaça',a$bebida),'Sim','Não') # nova variável
>
> # identifica condicionalmente quais linhas possuem uma ou outra palavra
> a$cachaca[1:6]
[1] "Sim" "Sim" "Não" "Sim" "Sim" "Sim"
> a$cerveja <- ifelse(grepl('cerveja',a$bebida),'Sim','Não') # nova variável
> a$cerveja[1:6]
```

```
[1] "Sim" "Sim" "Sim" "Não" "Sim" "Sim"

> table(a$bebida,a$cachaca)


                  Não Sim
  cachaça           0   5
  cachaça,cerveja   0   8
  cerveja           7   0
  cerveja,cachaça   0  11
> table(a$bebida,a$cerveja)


                  Não Sim
  cachaça           5   0
  cachaça,cerveja   0   8
  cerveja           0   7
  cerveja,cachaça   0  11
>
```

Nas edições exemplificadas acima, diversas operações foram executadas: os níveis de diversas variáveis formatadas como *factors* foram especificados; espaços em brancos excedentes de variáveis *character* foram removidos; o nome de uma variável foi substituído; transformações para fatores; recodificação de variáveis para binárias a partir de contínuas; substituição de elementos alfanuméricos dentro de variáveis *character*; transformação de datas de *character* para *Date*; substituição de espaços em branco por `NA` e a recodificação condicional pelo conteúdo de outra variável.

Agora vamos fazer algumas outras operações com o mesmo banco que são comuns antes ou durante a análise de dados como: criar variáveis com mais de uma condicional, apagar variáveis, categorizar variáveis, ordenar os dados, fracionar os dados, juntar com outros dados.

```
> a$Age[1:6] # é a mesma coisa que
```

```
[1] 30 15 24 48 19 44

> a[1:6,'Age'] # a tem duas dimensões

[1] 30 15 24 48 19 44

> dim(a) # número de linhas e colunas, linhas sempre vem antes de colunas

[1] 32 14

>

> a$Grupos <- NA #cria variável vazia

> a$Age > 40 & a$Sex=='Feminino' # retorna TRUE ou FALSE

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

[13] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

[25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

> which(a$Age > 40 & a$Sex=='Feminino') # retorna os índices onde estão TRUE

[1]  6 17

> a[which(a$Age > 40 & a$Sex=='Feminino'),'Grupos'] # Variável vazia

[1] NA NA

> a[which(a$Age > 40 & a$Sex=='Feminino'),'Grupos'] <- 'Grupo 1'

> a[which(a$Age < 40 & a$Sex=='Feminino'),'Grupos'] <- 'Grupo 2'

> a[which(a$Age > 40 & a$Sex=='Masculino'),'Grupos'] <- 'Grupo 3'

> a[which(a$Age < 40 & a$Sex=='Masculino'),'Grupos'] <- 'Grupo 4'

> table(a$Grupos)


Grupo 1 Grupo 2 Grupo 3 Grupo 4

      2      14       1      11

> addmargins(table(a$Grupos, useNA = "ifany")) # Variáveis condição com NA retornam NA


Grupo 1 Grupo 2 Grupo 3 Grupo 4    <NA>     Sum
```

```
      2      14       1      11       4      32

> # apagando uma variável

> a$bebida <- NULL

> names(a) # bebida (sem o s) não está mais no banco

 [1] "Patient"    "Sex"        "Age"        "Onset.time" "Group"

 [6] "Data"       "bebidas"    "morte"      "tabaco"     "tempo"

[11] "cachaca"    "cerveja"    "Grupos"

> # a <- a[,-which(names(a)=='bebida')] # mesma coisa

> # a <- a[,-macth('bebida',names(a))] # mesma coisa

> # a['bebida'] <- NULL # mesma coisa

> # a <- subset(a, select = names(a)!='bebida') # mesma coisa

> # a[, c("X","bebidas")] <- list(NULL) # apagando diversas variáveis

>

> #categorizando uma variável

> a$Age_cat <- cut(a$Age,c(10,20,30,40,50),labels=c('10-19','20-29','30-39','40-49'), ,
include.lowest = TRUE)

> table(a$Age_cat)


10-19 20-29 30-39 40-49

   13     9     6     3

>

> # Organizando por Data (crescente)

> order(a$Data)

 [1]  1  3  5  6  8  9 11 12 13 15 16 18  4  7 19 20 21 22 23  2 14 24 25 27 28

[26] 29 10 17 30 31 32 26

> a <- a[order(a$Data),]
```

```
> order(a$Data)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32
>
> # olhar somente parte dos dados ou fazendo um subset
# Função subset retorna de forma mais consistente se uma ou mais condições não são atendidas
# subset(a, select = c('Data','morte'), subset = a$Sex == 'Masculino')
# O mesmo que
> a[which(a$Sex=='Masculino'),c('Data','morte')] # imprime Data e morte para o Homens
         Data   morte
1  2009-07-06 Censura
3  2009-09-04   Obito
8  2010-02-01   Obito
9  2010-03-03 Censura
11 2010-05-02   Obito
16 2010-09-29   Obito
4  2010-12-28 Censura
22 2011-03-28   Obito
23 2011-04-27   Obito
2  2011-05-27 Censura
14 2011-05-27   Obito
25 2011-06-26 Censura
27 2011-08-25 Censura
26       <NA>   Obito
>
> # simulando um dado
```

```
> b <- data.frame(ID=1:22,Crea=rnorm(22,.9,.3),Hg=rnorm(22,18,4))
> head(b)
  ID      Crea        Hg
1  1 0.5450512 19.151989
2  2 1.0282234  9.813840
3  3 1.2882576 22.255019
4  4 0.8578820 10.788968
5  5 0.7878459  8.959633
6  6 1.0200976 13.451268
>
> # juntando com outros dados
> a <- merge(a,b,by.x='Patient',by.y='ID',all=T, sort=F)
> View(a)
```

14. Descrevendo e estatísticas sumárias de bancos

Estatísticas sumárias de bancos de dados são interessantes para termos alguma familiaridade com o banco de dados, de tal forma que saibamos as variações e as características de variáveis e sabermos o que esperar quando as análises forem realizadas. É possível fazer essa inspeção visualmente e através de estatísticas. Uma dica que pode ajudar é manter uma descrição ou mapa dos dados em uma janela separada. O comando `page()` é bom para manter a descrição do banco sempre visível. Uma alternativa é salvar a descrição do banco num arquivo texto simples e manter o arquivo aberto durante o trabalho. A maioria das abordagens mais comuns já foi comentada ou demonstrada acima. Vamos importar o banco oswego.rec e demostrar algumas abordagens.

```
> rm(list = ls())
> setwd('c:/banco/curso_R')
> library('foreign')
```

```
>

> o <- read.epiinfo('oswego.rec')

> # summary(o)


> str(o)

'data.frame':                                   75 obs. of  20 variables:

 $ AGE       : num  11 52 65 59 13 63 70 40 15 33 ...

 $ SEX       : 'AsIs' chr  "M" "F" "M" "F" ...

 $ TIMESUPPER: num  NA 2000 1830 1830 NA 1930 1930 1930 2200 1900 ...

 $ ILL       : logi  FALSE TRUE TRUE TRUE FALSE TRUE ...

 $ ONSETDATE : 'AsIs' chr  NA "04/19" "04/19" "04/19" ...

 $ ONSETTIME : num  NA 30 30 30 NA 2230 2230 200 100 2300 ...

 $ BAKEDHAM  : logi  FALSE TRUE TRUE TRUE FALSE TRUE ...

 $ SPINACH   : logi  FALSE TRUE TRUE TRUE FALSE TRUE ...

 $ MASHEDPOTA: logi  FALSE TRUE TRUE FALSE FALSE FALSE ...

 $ CABBAGESAL: logi  FALSE FALSE TRUE FALSE FALSE TRUE ...

 $ JELLO     : logi  FALSE FALSE FALSE FALSE FALSE TRUE ...

 $ ROLLS     : logi  FALSE TRUE FALSE FALSE FALSE FALSE ...

 $ BROWNBREAD: logi  FALSE FALSE FALSE FALSE FALSE FALSE ...

 $ MILK      : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...

 $ COFFEE    : logi  FALSE TRUE TRUE TRUE FALSE FALSE ...

 $ WATER     : logi  FALSE FALSE FALSE FALSE FALSE TRUE ...

 $ CAKES     : logi  FALSE FALSE FALSE TRUE FALSE FALSE ...

 $ VANILLA   : logi  FALSE TRUE TRUE TRUE FALSE TRUE ...

 $ CHOCOLATE : logi  TRUE FALSE TRUE TRUE TRUE FALSE ...

 $ FRUITSALAD: logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
```

```
 - attr(*, "prompts")= Named chr  "Age:" "Sex:" "Time of Supper(24 hour):" "Ill?" ...
  ..- attr(*, "names")= chr  "AGE" "SEX" "TIMESUPPER" "ILL" ...
>
> o <- read.epiinfo('oswego.rec', guess.broken.dates = T, thisyear = "1972",lower.case.names = T)
> View(o)
> # summary(o)
> str(o)
'data.frame':                                   75 obs. of  20 variables:
 $ age       : num  11 52 65 59 13 63 70 40 15 33 ...
 $ sex       : 'AsIs' chr  "M" "F" "M" "F" ...
 $ timesupper: num  NA 2000 1830 1830 NA 1930 1930 1930 2200 1900 ...
 $ ill       : logi  FALSE TRUE TRUE TRUE FALSE TRUE ...
 $ onsetdate : Date, format: NA "1972-04-19" "1972-04-19" ...
 $ onsettime : num  NA 30 30 30 NA 2230 2230 200 100 2300 ...
 $ bakedham  : logi  FALSE TRUE TRUE TRUE FALSE TRUE ...
 $ spinach   : logi  FALSE TRUE TRUE TRUE FALSE TRUE ...
 $ mashedpota: logi  FALSE TRUE TRUE FALSE FALSE FALSE ...
 $ cabbagesal: logi  FALSE FALSE TRUE FALSE FALSE TRUE ...
 $ jello     : logi  FALSE FALSE FALSE FALSE FALSE TRUE ...
 $ rolls     : logi  FALSE TRUE FALSE FALSE FALSE FALSE ...
 $ brownbread: logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
 $ milk      : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
 $ coffee    : logi  FALSE TRUE TRUE TRUE FALSE FALSE ...
 $ water     : logi  FALSE FALSE FALSE FALSE FALSE TRUE ...
 $ cakes     : logi  FALSE FALSE FALSE TRUE FALSE FALSE ...
 $ vanilla   : logi  FALSE TRUE TRUE TRUE FALSE TRUE ...
```

```
 $ chocolate : logi  TRUE FALSE TRUE TRUE TRUE FALSE ...
 $ fruitsalad: logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
 - attr(*, "prompts")= Named chr  "Age:" "Sex:" "Time of Supper(24 hour):" "Ill?" ...
  ..- attr(*, "names")= chr  "AGE" "SEX" "TIMESUPPER" "ILL" ...
> head(o)
  age sex timesupper   ill  onsetdate onsettime bakedham spinach mashedpota cabbagesal jello rolls
1  11   M         NA FALSE       <NA>        NA    FALSE   FALSE      FALSE      FALSE FALSE FALSE
2  52   F       2000  TRUE 1972-04-19        30     TRUE    TRUE       TRUE      FALSE FALSE  TRUE
3  65   M       1830  TRUE 1972-04-19        30     TRUE    TRUE       TRUE       TRUE FALSE FALSE
4  59   F       1830  TRUE 1972-04-19        30     TRUE    TRUE      FALSE      FALSE FALSE FALSE
5  13   F         NA FALSE       <NA>        NA    FALSE   FALSE      FALSE      FALSE FALSE FALSE
6  63   F       1930  TRUE 1972-04-18      2230     TRUE    TRUE      FALSE       TRUE  TRUE FALSE
  brownbread  milk coffee water cakes vanilla chocolate fruitsalad
1      FALSE FALSE  FALSE FALSE FALSE   FALSE      TRUE      FALSE
2      FALSE FALSE   TRUE FALSE FALSE    TRUE     FALSE      FALSE
3      FALSE FALSE   TRUE FALSE FALSE    TRUE      TRUE      FALSE
4      FALSE FALSE   TRUE FALSE  TRUE    TRUE      TRUE      FALSE
5      FALSE FALSE  FALSE FALSE FALSE   FALSE      TRUE      FALSE
6      FALSE FALSE  FALSE  TRUE FALSE    TRUE     FALSE      FALSE
> tail(o)
   age sex timesupper   ill  onsetdate onsettime bakedham spinach mashedpota cabbagesal jello rolls
70  21   F         NA  TRUE 1972-04-19        30     TRUE   FALSE      FALSE       TRUE  TRUE FALSE
71  60   M       1930  TRUE 1972-04-19       100    FALSE   FALSE      FALSE      FALSE FALSE FALSE
72  18   F       1930  TRUE 1972-04-18      2400     TRUE    TRUE       TRUE       TRUE  TRUE FALSE
73  14   F       2200 FALSE       <NA>        NA    FALSE   FALSE      FALSE      FALSE FALSE FALSE
74  52   M         NA  TRUE 1972-04-19       215     TRUE   FALSE       TRUE      FALSE  TRUE  TRUE
```

```
75  45   F         NA  TRUE 1972-04-18      2300     TRUE    TRUE       TRUE       TRUE  TRUE  TRUE
   brownbread  milk coffee water cakes vanilla chocolate fruitsalad
70      FALSE FALSE  FALSE FALSE  TRUE    TRUE     FALSE      FALSE
71      FALSE FALSE  FALSE FALSE FALSE    TRUE      TRUE      FALSE
72      FALSE FALSE  FALSE  TRUE  TRUE    TRUE      TRUE      FALSE
73      FALSE FALSE  FALSE FALSE  TRUE    TRUE     FALSE      FALSE
74       TRUE FALSE   TRUE  TRUE  TRUE    TRUE      TRUE      FALSE
75       TRUE FALSE   TRUE FALSE  TRUE    TRUE     FALSE       TRUE
>
> library('Hmisc')
Carregando pacotes exigidos: lattice

Attaching package: 'lattice'

The following object is masked from 'package:epiDisplay':

    dotplot

Carregando pacotes exigidos: Formula
Carregando pacotes exigidos: ggplot2

Attaching package: 'ggplot2'

The following object is masked from 'package:epiDisplay':

    alpha
```

```
Attaching package: 'Hmisc'


The following objects are masked from 'package:base':


    format.pval, units


> o <- stata.get('oswego.dta') # repare que Hmisc importa corretamente as horas mas não importa do
mesmo jeito as datas.
> label(o$id) <- 'Unique ID' # adiciona uma etiqueta, aparece na descrição
> # mesma coisa que
> # attr(o$id,'label') <- "Unique ID"
> attr(o$id,"label") # mesma coisa que Hmisc::label(o$id)
[1] "Unique ID"
> attributes(o$id)
$label
[1] "Unique ID"


$class
[1] "labelled" "integer"


$format
[1] "%9.0g"


> # mesma coisa que Hmisc::label(o$age) <- "Idade ao evento"
```

```
> attr(o$age,'label') <- "Idade ao evento"

> attributes(o$age)

$label

[1] "Idade ao evento"


$class

[1] "labelled" "integer"


$format

[1] "%9.0g"


> attr(o$sex,'label') <- "Sexo"

>label(o$meal.time) <- "Hora da refeição"

>label(o$ill) <- "Doente"

>label(o$onset.date) <- "Data incidente"

>label(o$onset.time)  <- "Hora incidente"

>label(o$baked.ham)  <- "Presunto"

>label(o$spinach)  <- "Espinafre"

>label(o$mashed.potato) <- "Purê de batata"

>label(o$cabbage.salad)  <- "Salada"

>label(o$jello) <- "Gelatina"

>label(o$rolls)  <- "Rolinho"

>label(o$brown.bread)  <- "Pão integral"

>label(o$milk)  <- "Leite"

>label(o$coffee) <- "Café"

>label(o$water) <- "Água"
```

```
>label(o$cakes) <- "Bolos"

>label(o$vanilla.ice.cream) <- "Sorvete de creme"

>label(o$chocolate.ice.cream) <- "Sorvete de chocolate"

>label(o$fruit.salad)  <- "Salada de frutas"

>etiquetas <- label(o)

>etiquetas

                     id                     age                     sex               meal.time
            "Unique ID"      "Idade ao evento"                 "Sexo"     "Hora da refeição"
                    ill             onset.date             onset.time               baked.ham
               "Doente"       "Data incidente"       "Hora incidente"              "Presunto"
                spinach          mashed.potato          cabbage.salad                  jello
            "Espinafre"       "Pure de batata"               "Salada"              "Gelatina"
                 rolls            brown.bread                   milk                 coffee
              "Rolinho"         "Pão integral"               "Leite"                 "Café"
                 water                   cakes      vanilla.ice.cream    chocolate.ice.cream
                 "Água"                "Bolos"     "Sorvete de creme" "Sorvete de chocolate"
            fruit.salad
    "Salada de frutas"

> describe(o) # Esse banco possui os rótulos inseridos

o


 21  Variables      75  Observations

--------------------------------------------------------------------------------

id : Unique ID  Format:%9.0g

       n  missing distinct     Info     Mean      Gmd      .05      .10

      75        0       75        1       38    25.33      4.7      8.4
```

```
     .25       .50       .75       .90       .95
    19.5      38.0      56.5      67.6      71.3

lowest :  1  2  3  4  5, highest: 71 72 73 74 75
---------------------------------------------------------------------------
age : Idade ao evento  Format:%9.0g
       n  missing distinct     Info     Mean      Gmd      .05      .10
      75        0       45    0.999    36.81    24.75      8.0     11.0
     .25       .50       .75       .90       .95
    16.5      36.0      57.5      65.0      69.3

lowest :  3  7  8  9 10, highest: 69 70 72 74 77
---------------------------------------------------------------------------
sex : Sexo  Format:%1s
       n  missing distinct
      75        0        2

Value          F     M
Frequency     44    31
Proportion 0.587 0.413
---------------------------------------------------------------------------
meal.time : Hora da refeição  Format:%8s
       n  missing distinct
      75        0        7

Value      10:00 PM 11:00 AM  6:30 PM  7:00 PM  7:30 PM  8:00 PM       NA
```

```
Frequency        11        1        2        2       10        1       48
Proportion    0.147    0.013    0.027    0.027    0.133    0.013    0.640
--------------------------------------------------------------------------------
ill : Doente  Format:%1s
       n  missing distinct
      75        0        2

Value          N     Y
Frequency     29    46
Proportion 0.387 0.613
--------------------------------------------------------------------------------
onset.date : Data incidente  Format:%4s
       n  missing distinct
      75        0        3

Value       4/18  4/19    NA
Frequency     20    26    29
Proportion 0.267 0.347 0.387
--------------------------------------------------------------------------------
onset.time : Hora incidente  Format:%8s
       n  missing distinct
      75        0       18

1:00 AM (10, 0.133), 10:00 PM (1, 0.013), 10:15 PM (1, 0.013), 10:30 AM (1,
0.013), 10:30 PM (4, 0.053), 11:00 PM (5, 0.067), 11:30 PM (2, 0.027), 12:00
AM (2, 0.027), 12:30 AM (7, 0.093), 2:00 AM (3, 0.040), 2:15 AM (1, 0.013),
```

```
2:30 AM (2, 0.027), 3:00 PM (1, 0.013), 9:00 PM (1, 0.013), 9:15 PM (1,

0.013), 9:30 PM (2, 0.027), 9:45 PM (2, 0.027), NA (29, 0.387)

-------------------------------------------------------------------------------

baked.ham : Presunto  Format:%1s

       n  missing distinct

      75        0        2


Value          N     Y

Frequency     29    46

Proportion 0.387 0.613

-------------------------------------------------------------------------------

spinach : Espinafre  Format:%1s

       n  missing distinct

      75        0        2


Value          N     Y

Frequency     32    43

Proportion 0.427 0.573

-------------------------------------------------------------------------------

mashed.potato : Purê de batata  Format:%2s

       n  missing distinct

      75        0        3


Value          N    NA     Y

Frequency     37     1    37

Proportion 0.493 0.013 0.493
```

```
---------------------------------------------------------------------------

cabbage.salad : Salada  Format:%1s

       n  missing distinct

      75        0        2


Value          N     Y

Frequency     47    28

Proportion 0.627 0.373

---------------------------------------------------------------------------

jello : Gelatina  Format:%1s

       n  missing distinct

      75        0        2


Value          N     Y

Frequency     52    23

Proportion 0.693 0.307

---------------------------------------------------------------------------

rolls : Rolinho  Format:%1s

       n  missing distinct

      75        0        2


Value          N     Y

Frequency     38    37

Proportion 0.507 0.493

---------------------------------------------------------------------------

brown.bread : Pão integral  Format:%1s
```

```
       n  missing distinct

      75        0        2


Value         N    Y

Frequency    48   27

Proportion 0.64 0.36

---------------------------------------------------------------------------

milk : Leite  Format:%1s

       n  missing distinct

      75        0        2


Value          N     Y

Frequency     71     4

Proportion 0.947 0.053

---------------------------------------------------------------------------

coffee : Café  Format:%1s

       n  missing distinct

      75        0        2


Value          N     Y

Frequency     44    31

Proportion 0.587 0.413

---------------------------------------------------------------------------

water : Água  Format:%1s

       n  missing distinct

      75        0        2
```

```
Value          N    Y

Frequency     51   24

Proportion 0.68 0.32

-------------------------------------------------------------------------------

cakes : Bolos  Format:%1s

       n  missing distinct

      75        0        2


Value          N     Y

Frequency     35    40

Proportion 0.467 0.533

-------------------------------------------------------------------------------

vanilla.ice.cream : Sorvete de creme  Format:%1s

       n  missing distinct

      75        0        2


Value          N    Y

Frequency     21   54

Proportion 0.28 0.72

-------------------------------------------------------------------------------

chocolate.ice.cream : Sorvete de chocolate  Format:%2s

       n  missing distinct

      75        0        3


Value          N    NA     Y
```

```
Frequency      27      1     47

Proportion 0.360 0.013 0.627

-------------------------------------------------------------------------------

fruit.salad : Salada de frutas  Format:%1s

       n  missing distinct

      75        0        2


Value          N     Y

Frequency     69     6

Proportion 0.92 0.08

-------------------------------------------------------------------------------

> # Essa lógica de etiquetas funciona se importados com sas.get, stata.get ou spss.get e houver etiquetas no banco origianal

> page(describe(o),'print')

> table(o$sex)


 F  M

44 31

> table(o$sex, dnn = attr(o$sex,'label')) # table(o$sex, dnn = label(o$sex))

Sexo

 F  M

44 31

>

>

> # Alternativa ao page é salvar a descrição no disco e abrir o arquivo

> sink("descricao_oswego.txt") # A impressão do console vai pro diretório padrão\arquivo
```

```
> describe(o)

> sink()

> # file.show("descricao_oswego.txt")

>

> while(search()[2] != "tools:rstudio") detach(2) # desanexa todos os pacotes

>

> # detach("package:Hmisc");detach("package:Formula")

> # detach("package:survival");detach("package:splines")

>

> library('epiDisplay')

Carregando pacotes exigidos: foreign

Carregando pacotes exigidos: survival

Carregando pacotes exigidos: MASS

Carregando pacotes exigidos: nnet

> o <- read.epiinfo('oswego.rec', guess.broken.dates = T, thisyear = "1972",lower.case.names = T)

> des(o) # esse banco não possui qualquer rótulo

> attr(o, "var.labels") <- etiquetas # etiquetas, criado acima, passado para o banco como atributo
permite a utilização das etiquetas nessa descrição


 No. of observations =  75

   Variable      Class           Description

1  age           numeric

2  sex           AsIs

3  timesupper    numeric

4  ill           logical

5  onsetdate     Date
```

```
6  onsettime     numeric

7  bakedham      logical

8  spinach       logical

9  mashedpota    logical

10 cabbagesal    logical

11 jello         logical

12 rolls         logical

13 brownbread    logical

14 milk          logical

15 coffee        logical

16 water         logical

17 cakes         logical

18 vanilla       logical

19 chocolate     logical

20 fruitsalad    logical

> page(des(o),'print')

> summ(o)


No. of observations = 75


   Var. name  obs. mean         median  s.d.    min.   max.

1  age        75   36.81        36      21.45   3      77

2  sex

3  timesupper 28   1930.71      1930    440.64 0      2200

4  ill        75   0.61         1       0.49    0      1

5  onsetdate  46   838.52173913 839     <NA>    838    839
```

```
6  onsettime  46   1132.07      630      1057.9 30      2400

7  bakedham   75   0.61         1        0.49   0       1

8  spinach    75   0.57         1        0.5    0       1

9  mashedpota 74   0.5          0.5      0.5    0       1

10 cabbagesal 75   0.37         0        0.49   0       1

11 jello      75   0.31         0        0.46   0       1

12 rolls      75   0.49         0        0.5    0       1

13 brownbread 75   0.36         0        0.48   0       1

14 milk       75   0.05         0        0.23   0       1

15 coffee     75   0.41         0        0.5    0       1

16 water      75   0.32         0        0.47   0       1

17 cakes      75   0.53         1        0.5    0       1

18 vanilla    75   0.72         1        0.45   0       1

19 chocolate  74   0.64         1        0.48   0       1

20 fruitsalad 75   0.08         0        0.27   0       1

> # codebook(o)

> > attributes(o)

$names

 [1] "age"        "sex"        "timesupper" "ill"        "onsetdate"  "onsettime"

 [7] "bakedham"   "spinach"    "mashedpota" "cabbagesal" "jello"      "rolls"

[13] "brownbread" "milk"       "coffee"     "water"      "cakes"      "vanilla"

[19] "chocolate"  "fruitsalad"


$row.names

 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

[28] 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
```

```
[55] 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
```



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

$prompts
                      AGE                       SEX                TIMESUPPER
                "Age:"                  "Sex:" "Time of Supper(24 hour):"
                      ILL                ONSETDATE                ONSETTIME
                "Ill?"      "Onset Date (MM/DD):"    "Onset Time (24 hour):"
                 BAKEDHAM                  SPINACH                MASHEDPOTA
            "Baked Ham:"              "Spinach:"      "Mashed Potatoes:"
                CABBAGESAL                   JELLO                    ROLLS
         "Cabbage Salad:"               "Jello:"                 "Rolls:"
                BROWNBREAD                    MILK                   COFFEE
          "Brown Bread:"                 "Milk:"                "Coffee:"
                     WATER                   CAKES                  VANILLA
                 "Water:"                 "Cakes:"  "Ice Cream:    Vanilla:"
               CHOCOLATE               FRUITSALAD
            "Chocolate:"          "Fruit Salad:"

$var.labels
                    id                   age                   sex
            "Unique ID"  "Idade ao evento"              "Sexo"
              meal.time                   ill            onset.date
  "Hora da refeição"            "Doente"   "Data incidente"
```

```
          onset.time              baked.ham                spinach
"Hora incidente"             "Presunto"            "Espinafre"
       mashed.potato         cabbage.salad                jello
"Purê de batata"               "Salada"             "Gelatina"
               rolls           brown.bread                 milk
           "Rolinho"        "Pão integral"              "Leite"
              coffee                  water                cakes
              "Café"                 "Água"              "Bolos"
    vanilla.ice.cream   chocolate.ice.cream          fruit.salad
"Sorvete de creme" "Sorvete de chocolate"    "Salada de frutas"
> attr(o, "variable.labels") # não é a mesma coisa que attr(o, "var.labels")
NULL
> etiquetas_en <- attr(o, "prompts")
> etiquetas_en
                      AGE                      SEX                TIMESUPPER
                "Age:"                  "Sex:" "Time of Supper(24 hour):"
                      ILL                ONSETDATE                 ONSETTIME
                "Ill?"      "Onset Date (MM/DD):"    "Onset Time (24 hour):"
                 BAKEDHAM                 SPINACH                 MASHEDPOTA
            "Baked Ham:"               "Spinach:"       "Mashed Potatoes:"
              CABBAGESAL                   JELLO                      ROLLS
        "Cabbage Salad:"                "Jello:"                  "Rolls:"
              BROWNBREAD                    MILK                     COFFEE
          "Brown Bread:"                  "Milk:"                  "Coffee:"
                 WATER                   CAKES                    VANILLA
              "Water:"                 "Cakes:"   "Ice Cream:    Vanilla:"
```

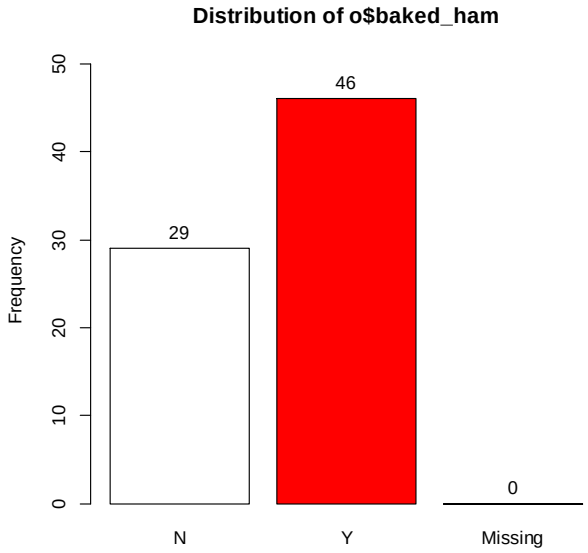
```
                CHOCOLATE                 FRUITSALAD

             "Chocolate:"              "Fruit Salad:"

> etiquetas_en <- gsub(":","",etiquetas_en) # Remove os ":" das etiquetas

> names(etiquetas_en) <- names(o)

> names(etiquetas) # Objetos não são idênticos nos conteúdo nem nos nomes. Cuidado.

 [1] "id"                  "age"                 "sex"

 [4] "meal.time"           "ill"                 "onset.date"

 [7] "onset.time"          "baked.ham"           "spinach"

[10] "mashed.potato"       "cabbage.salad"       "jello"

[13] "rolls"               "brown.bread"         "milk"

[16] "coffee"              "water"               "cakes"

[19] "vanilla.ice.cream"   "chocolate.ice.cream" "fruit.salad"


> epiDisplay::tab1(o$sex, main = attr(o,'var.labels')["sex"])

o$sex :

        Frequency Percent Cum. percent

F              44    58.7         58.7

M              31    41.3        100.0

  Total        75   100.0        100.0

>

> # Infelizmente as etiquetas guardadas como atributos do banco e as guardadas como atributos da
variável não são intercambiáveis

> # Mas as etiquetas das variáveis do banco importada com qualquer função pode ser acessado se
disponível

>

> # ATENÇÃO COM OS . OU _ DAS VARIAVEIS QUE DEPENDE DA FUNÇÃO QUE IMPORTOU O BANCO

>
```

Algumas vezes é preciso importar o banco mais de uma vez, para conhecer o seu conteúdo e acertar opções mais convenientes de importação de dados. Isso depende também do formato original do banco e das opções oferecidas na função que importa o banco. No exemplo acima, o ano das datas armazenadas no banco original estavam com apenas dois dígitos e o R não conseguiu identificar automaticamente, assim argumentos adicionais fazem com que as datas sejam importadas com um formato mais amigável. Ainda, repare que com o pacote Hmisc, nenhum argumento adicional é dado, porém as horas são importadas no formato adequado, mas a data não veio com o ano. A função `View()` permite visualizar os dados (mas não editar) como em uma planilha. A função `summary()` é um função genérica. Isso quer dizer que se o argumento de entrada é um vetor numérico, essa retorna valores como o máximo, mínimo, mediana, média, desvio padrão, mas se o argumento de entrada é um fator, a função retorna frequências das categorias e a quantidade de `NAs` na variável. A vantagem de chamar essa função para um banco de dados, é que esta faz para todas as variáveis simultaneamente. A função `str()` mostra a estrutura do banco (ou de qualquer outro objeto, pois também é genérica) de forma compacta. As funções `head() e tail()` mostram as primeiras linhas ou ultimas linhas dos dados. Já no pacote Hmisc, a função `describe()` fornece o nome, o formato, sumário dos valores e o rótulo se houver, numa forma mais amigável e completa. No pacote epiDisplay, as funções `des(), summ() e codebook()` fazem em conjunto o que a função `Hmisc::describe()` faz sozinha, com uma aparência um tanto diferente. Uma vez que a importação deu certo e foram identificadas as etiquetas do banco, é conveniente guardar as etiquetas das variáveis num objeto separado.

Abaixo, algumas explorações dos dados são realizadas com tabelas individuais ou com gráficos, pois muitos gostam de visualizar dessa forma para melhor compreender distribuições e quantidades de NAs.

```
> > rm(o)
> library('Hmisc')
> o <- stata.get('oswego.dta')
>
> while(search()[2] != "tools:rstudio") detach(2)
>
> # "ifany" retorna someste se houver NA
> table(o$baked.ham, useNA = "always", dnn = etiquetas["baked.ham"]) # Sempre mostra os NA
```

```
Presunto

   N    Y <NA>

  29   46    0

> table(o$baked.ham, useNA = "ifany", dnn = etiquetas["baked.ham"]) # Mostra NA se presente

Presunto

 N  Y

29 46

> addmargins(table(o$baked.ham, useNA = "always")) # adiciona margens - padrão = soma


   N    Y <NA>  Sum

  29   46    0   75

> addmargins(prop.table(table(o$baked.ham, useNA = "always"))) # adiciona margens - padrão = soma


        N         Y      <NA>       Sum

0.3866667 0.6133333 0.0000000 1.0000000

>

> library(epiDisplay)

> > tab1(o$baked_ham) # do pacote epiDisplay, retorna um gráfico também.

o$baked_ham :

        Frequency Percent Cum. percent

N              29    38.7         38.7

Y              46    61.3        100.0

  Total        75   100.0        100.0

>
```




```
> tab1(o$chocolate_ice_cream,graph=F)

o$chocolate_ice_cream :

        Frequency   %(NA+)   %(NA-)

N              27     36.0     36.5

Y              47     62.7     63.5

<NA>            1      1.3      0.0

  Total        75    100.0    100.0

>
```

A função original `base::table()` permite explorar os `NAs`, colocar nomes nas dimensões e com funções adicionais é possível adicionar marginais de tal forma que a tabela fique mais completa, e fazer uma tabela de percentuais. No entanto, para tabelas de apenas uma dimensão a função `epiDisplay::tab1()` faz as mesmas coisas, além de oferecer a frequência com e sem os `NAs` (omitindo a coluna dos `NAs` se ausente nos dados) e possui uma saída mais amigável. Ao mesmo tempo é possível que esta gere um gráfico de barras para visualização da informação da tabela.

É possível fazer essa exploração de forma visualmente mais amigável com a função `epiDisplay::tableStack()` ou preferencialmente com `ems::tableStack()` (clone da primeira com algumas

modificações) que consegue fazer uma tabela para todas as variáveis simultaneamente empilhando-as. Essa função do epiDisplay possui dois inconvenientes: (1) mostra o total de linhas do banco e não do total para cada variável; (2) e não mostra os `NAs`, omitindo-os todos automaticamente. Além disso, não arredonda as casas decimais de forma uniforme no caso de zeros depois das vírgulas. No caso do pacote ems,, esses inconvenientes são superados por opções na função. Por isso, logo abaixo há outro exemplo de fazer a mesma coisa mas combinando as funções `lapply()` e `tab1()` (mas poderia ser `lapply()` e `table()`). É um pouco mais complexo para os iniciantes, mas contorna os inconvenientes da `epiDisplay::tableStack()`.


```
> # Com apenas parte dos dados para exemplo
> vars <- c("age", "sex", "ill", "baked.ham", "spinach", "mashed.potato")
> tab.des <- lapply(vars, function(i) epiDisplay::tab1(o[,i],graph = F))
> names(tab.des) <- vars
> tab.des
$age
o[, i] :
        Frequency Percent Cum. percent
3               1     1.3          1.3
7               2     2.7          4.0
8               2     2.7          6.7
9               1     1.3          8.0
10              1     1.3          9.3
11              4     5.3         14.7
12              1     1.3         16.0
13              2     2.7         18.7
14              1     1.3         20.0
15              3     4.0         24.0
16              1     1.3         25.3
```

17	4	5.3	30.7
18	1	1.3	32.0
20	2	2.7	34.7
21	1	1.3	36.0
24	1	1.3	37.3
25	2	2.7	40.0
32	1	1.3	41.3
33	2	2.7	44.0
35	4	5.3	49.3
36	3	4.0	53.3
37	2	2.7	56.0
38	1	1.3	57.3
40	2	2.7	60.0
44	1	1.3	61.3
45	1	1.3	62.7
48	1	1.3	64.0
50	2	2.7	66.7
52	3	4.0	70.7
53	1	1.3	72.0
54	1	1.3	73.3
57	1	1.3	74.7
58	1	1.3	76.0
59	2	2.7	78.7
60	1	1.3	80.0
62	4	5.3	85.3
63	1	1.3	86.7

```
64              1     1.3         88.0

65              3     4.0         92.0

68              1     1.3         93.3

69              1     1.3         94.7

70              1     1.3         96.0

72              1     1.3         97.3

74              1     1.3         98.7

77              1     1.3        100.0

  Total        75   100.0        100.0


$sex

o[, i] :

        Frequency Percent Cum. percent

F              44    58.7         58.7

M              31    41.3        100.0

  Total        75   100.0        100.0


$ill

o[, i] :

        Frequency Percent Cum. percent

N              29    38.7         38.7

Y              46    61.3        100.0

  Total        75   100.0        100.0


$baked.ham

o[, i] :
```

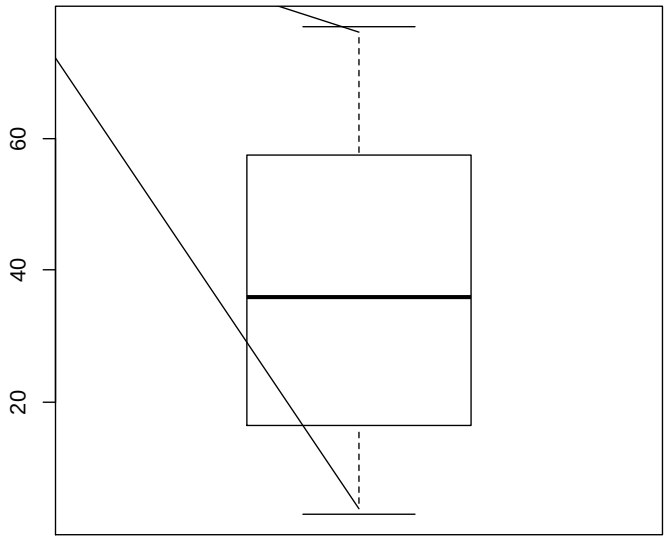
```
        Frequency Percent Cum. percent
N              29    38.7         38.7
Y              46    61.3        100.0
  Total        75   100.0        100.0


$spinach
o[, i] :
        Frequency Percent Cum. percent
N              32    42.7         42.7
Y              43    57.3        100.0
  Total        75   100.0        100.0


$mashed.potato
o[, i] :
        Frequency   %(NA+)   %(NA-)
N              37     49.3       50
Y              37     49.3       50
<NA>            1      1.3        0
  Total        75    100.0      100


>
> # Exploração parecida com tableStack
> ems::tableStack(vars, dataFrame = o, by = '', NAcol = F, NArow = T)
              Total
Total         75
```

```
age

  mean (SD)   36.81 (21.45)

  NA          0 (0.00)


sex

   F          44 (58.67)

   M          31 (41.33)

   NA         0 (0.00)


ill

   N          29 (38.67)

   Y          46 (61.33)

   NA         0 (0.00)


baked.ham

   N          29 (38.67)

   Y          46 (61.33)

   NA         0 (0.00)


spinach

   N          32 (42.67)

   Y          43 (57.33)

   NA         0 (0.00)


mashed.potato

   N          37 (49.33)
```

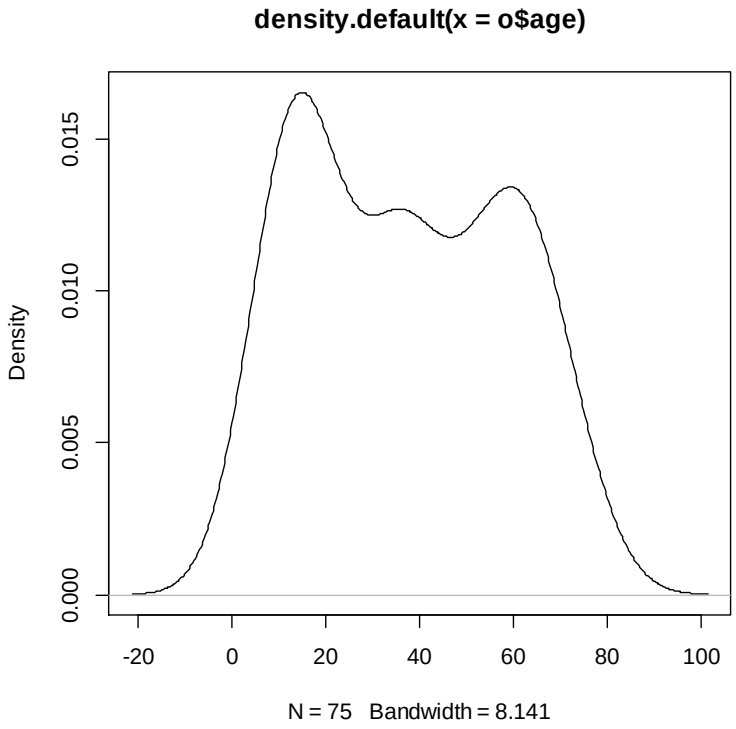
```
Y           37 (49.33)

NA          1 (1.33)
```


Algumas vezes as variáveis contínuas são mais bem exploradas através de gráficos (funções gráficas serão exemplificadas melhor em seção abaixo). Usualmente, gráficos de densidade, boxplot ou pontos, histogramas são razoáveis. Aqui nos exemplos, mostraremos apenas uma variável no gráfico, mas, a semelhança das tabelas, é possível fazer gráficos com diversas variáveis na mesma janela.

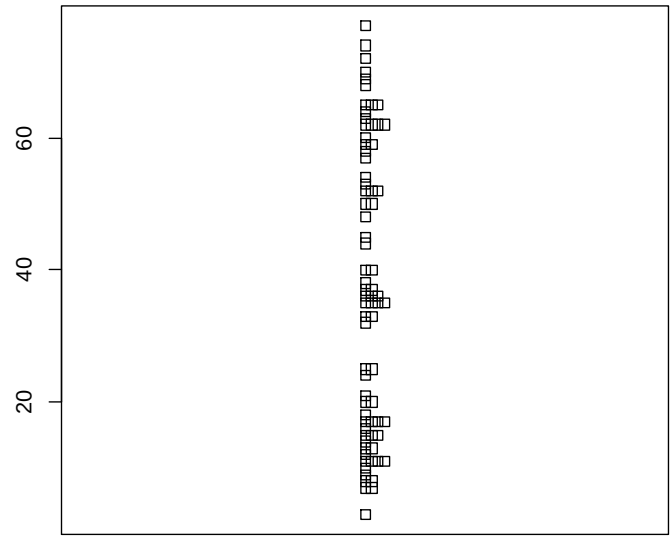
```
> boxplot(o$age)
```



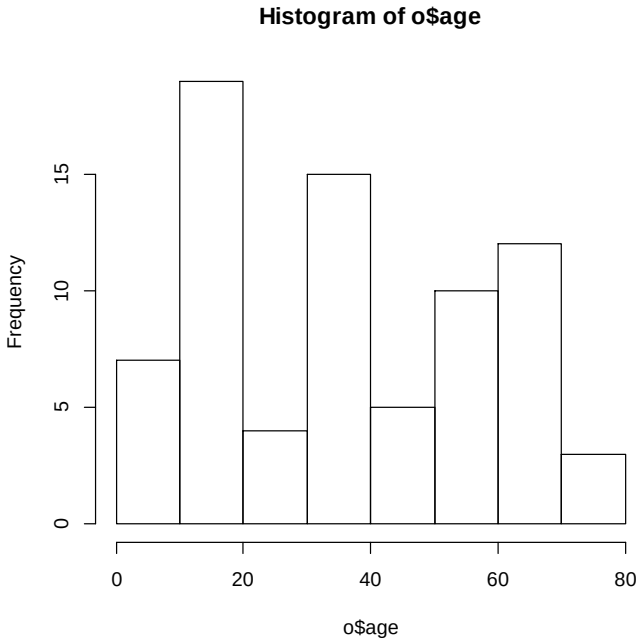

```
> plot(density(o$age))
```




```
> stripchart(o$age,'stack',vertical = T)
```




```
> hist(o$age)
```



15. Conteúdo do Exercício 2 foi coberto. Proceder para avaliação.

16. Salvar resultados no disco

Abaixo na seção de gráficos, haverá exemplos de como salvar gráficos. Portanto, nesta seção abordaremos somente como exportar tabelas, de forma a ser possível inseri-las posteriormente no manuscrito. Como tudo no R, sempre há mais de uma maneira. É possível exportar para pdf, para HTML, para excel ou word dentre outros, o que usualmente exige pacotes adicionais. No entanto, a forma mais simples de trabalhar com tabelas é exportá-las para planilhas de cálculo com as funções `write.csv()` ou `write.table()`. Posteriormente será possível abrir a planilha *.csv e editá-la com o programa original de forma a inserir bordas, alinhar etc, mas não editar o seu conteúdo. Essa prática parte do princípio que os objetos são de fato tabelas, matrizes ou bancos de dados que podem ser formatados como tabelas no manuscrito. E para isso, o conteúdo da tabela deverá estar todo pronto, para que a edição posterior seja somente na aparência e não no conteúdo. Para exportar tabelas para pdf, HTML, word ou LibreOffice, recomendo os pacotes rtf (para word), rapport (para LibreOffice), sweave (para pdf), knitr (pdf ou HTML) ou markdown (HTML) (http://cran.r-project.org/web/views/ReproducibleResearch.html).

Para isso mostraremos algumas formas de análise simples em tabelas, como estimativa de riscos relativos ou tabelas com testes de hipótese que poderão ser mostrados em tabelas e posteriormente exportados. Aqui vamos mostrar somente exportações para planilhas e documentos em word. As demais opções são mais sofisticadas e não estão no escopo do curso.

```
> # setwd('c:/banco/curso_R')

> rm(list = ls())

> library('Hmisc')

Carregando pacotes exigidos: lattice

Carregando pacotes exigidos: survival

Carregando pacotes exigidos: Formula

Carregando pacotes exigidos: ggplot2
```

```
Attaching package: 'Hmisc'


The following objects are masked from 'package:base':

    format.pval, units


> o <- stata.get('oswego.dta')
# Acrescentando algumas etiquetas ao banco de dados
> label(o$age) <- 'Idade'
> label(o$sex) <- 'Sexo'
> label(o$baked.ham) <- 'Presunto'
> label(o$spinach) <- 'Espinafre'
> label(o$ill) <- 'Doente'
> label(o$vanilla.ice.cream) <- 'Sorvete creme'
> label(o$mashed.potato) <- 'Purê de batata'
> label(o$cabbage.salad) <- 'Salada'
> label(o$jello) <- 'Gelatina'
# Armazenando as etiquetas em um objeto separado
> etiquetas <- label(o)
>
> # Desanexa todos os pacotes
> while(search()[2] != "tools:rstudio") detach(2)
>
> # Substitui os "NA" por NA
> o[o=='NA'] <- NA
```

```
>
> table(o$sex,o$ill) # tabela simples


     N  Y
  F 14 30
  M 15 16
> table(o$sex,o$ill, dnn = etiquetas[c('sex','ill')]) # tabela com nomes nas margens
    Doente
Sexo  N  Y
   F 14 30
   M 15 16
> tabela1 <- addmargins(table(o$sex,o$ill,dnn = etiquetas[c('sex','ill')]))
> tabela1 # armazenada num objeto com as somas das margens
     Doente
Sexo   N  Y Sum
  F   14 30  44
  M   15 16  31
  Sum 29 46  75
> write.csv2(tabela1,'Tabela_1.csv')
> file.show('Tabela_1.csv')
>
> library(ems)
> # Fazendo as etiquetas
> attr(o, "var.labels") <- etiquetas
> # Fazendo as tabelas
```

```
> tabela2 <- tableStack(c(age,sex,baked.ham,spinach),dataFrame=o,
by=ill,decimal=2,total.column=T, var.labels = FALSE)
```



```
> tabela2
```

	N	Y	Total	Test stat.	P.Value
Total	29	46	75		
2 : age				Ranksum test	0.2837
median (IQR)	35.00 (14.00 - 50.00)	38.50 (17.25 - 58.75)	36.00 (16.50 - 57.50)		
3 : sex				Chisq. (1 df) = 1.465	0.2262
F	14 (48.28)	30 (65.22)	44 (58.67)		
M	15 (51.72)	16 (34.78)	31 (41.33)		
8 : baked.ham				Chisq. (1 df) = 0.019	0.8890
N	12 (41.38)	17 (36.96)	29 (38.67)		
Y	17 (58.62)	29 (63.04)	46 (61.33)		
9 : spinach				Chisq. (1 df) = 0.000	1.0000
N	12 (41.38)	20 (43.48)	32 (42.67)		
Y	17 (58.62)	26 (56.52)	43 (57.33)		

```
> tabela2 <- tableStack(c(age,sex,baked.ham,spinach),dataFrame=o,by=ill,decimal=2,total.column=T)
> tabela2
```

	N	Y	Total	Test stat.	P.Value
Total	29	46	75		

```
Idade                                                                    Ranksum test          0.2837

  median (IQR) 35.00 (14.00 - 50.00) 38.50 (17.25 - 58.75) 36.00 (16.50 - 57.50)
```



```
Sexo                                                                     Chisq. (1 df) = 1.465 0.2262

   F           14 (48.28)            30 (65.22)            44 (58.67)

   M           15 (51.72)            16 (34.78)            31 (41.33)


Presunto                                                                 Chisq. (1 df) = 0.019 0.8890

   N           12 (41.38)            17 (36.96)            29 (38.67)

   Y           17 (58.62)            29 (63.04)            46 (61.33)


Espinafre                                                                Chisq. (1 df) = 0.000 1.0000

   N           12 (41.38)            20 (43.48)            32 (42.67)

   Y           17 (58.62)            26 (56.52)            43 (57.33)


> write.csv2(tabela2,'Tabela_2.csv')

> # file.show('Tabela_2.csv') # executa um arquivo

>

> # Desanexa todos os pacotes

> while(search()[2] != "tools:rstudio") detach(2)

>

> library('epitools')

> riskratio(table(o$sex,o$ill)) # estimando o risco relativo

$data


         N  Y Total

  F     14 30    44
```

```
  M     15 16    31
  Total 29 46    75


$measure
   risk ratio with 95% C.I.
     estimate     lower    upper
  F 1.0000000        NA       NA
  M 0.7569892 0.5093946 1.124929


$p.value
   two-sided
    midp.exact fisher.exact chi.square
  F         NA           NA         NA
  M  0.1587331    0.1588193  0.1467924


$correction
[1] FALSE


attr(,"method")
[1] "Unconditional MLE & normal approximation (Wald) CI"
> oddsratio(table(o$sex,o$ill))
$data


         N  Y Total
  F     14 30    44
  M     15 16    31
```

```
  Total 29 46    75


$measure

   odds ratio with 95% C.I.

     estimate     lower    upper

  F 1.0000000        NA       NA

  M 0.5037182 0.1904794 1.306541


$p.value

   two-sided

    midp.exact fisher.exact chi.square

  F         NA           NA         NA

  M  0.1587331    0.1588193  0.1467924


$correction

[1] FALSE


attr(,"method")

[1] "median-unbiased estimate & mid-p exact CI"

>

> source('rrStack.R') # source lê um aquivo no diretório padrão

> # rrStack empilha numa tabela a saída do epitools::riskratio

> # nesta função há necessidade de informar as exposições como fatores, o desfecho e o banco

> for(i in c(3,8:21)){o[,i] <- as.factor(o[,i])} # transforma as colunas 3 e 8 a 21 em fatores

>
```

```
> tabela3 <-
rrStack(outcome='ill',exposure=c('sex',"milk","vanilla.ice.cream","fruit.salad"),data=o)

          variables categories  N risk lower upper   RR lower upper midp.exact fisher.exact chi.square

1               sex          F 44 0.68  0.52  0.81    1                                                

2                            M 31 0.52  0.33   0.7 0.76  0.51  1.12       0.16         0.16       0.15

3              milk          N 71 0.62   0.5  0.73    1                                                

4                            Y  4  0.5  0.07  0.93 0.81   0.3  2.19       0.66         0.64       0.63

5 vanilla.ice.cream          N 21 0.14  0.03  0.36    1                                                

6                            Y 54  0.8  0.66  0.89 5.57  1.94 16.03          0            0          0

7       fruit.salad          N 69 0.61  0.48  0.72    1                                                

8                            Y  6 0.67  0.22  0.96  1.1   0.6  1.99       0.82            1       0.78

Warning messages:

1: In chisq.test(xx, correct = correction) :

  Chi-squared approximation may be incorrect

2: In chisq.test(xx, correct = correction) :

  Chi-squared approximation may be incorrect

> write.csv2(tabela3,'Tabela_3.csv', na = "-")

> file.show('Tabela_3.csv')
```

O exemplo a seguir utilizará o pacote ‘rtf’.

```
# install.packages('rtf')

# vignette('rtf') # Procurar a vinheta do pacote rtf

library('rtf')


rtf <- RTF('Tabelas.doc',width=8.5,height=11,font.size=12,omi=c(1,1,1,1))
```

```
# Cria um objeto com os códigos para o arquivo, com nome e formatação

addTable(rtf,tabela1,col.justify='C',header.col.justify='C') # adiciona o objeto tabela1

addPageBreak(rtf) # quebra a página

addTable(rtf,tabela2,font.size=10,NA.string="") # adiciona o objeto tabela2

addPageBreak(rtf) # quebra a página

addTable(rtf,tabela3[,1:9],font.size=10,NA.string="") # adiciona o objeto tabela3

done(rtf) # encerra a sessão de inserção de objetos no arquivo texto

file.show('Tabelas.doc') # abrir o arquivo no diretório de trabalho

detach("package:rtf")
```


17. Gráficos

Há muitos pacotes gráficos no R. Mais recentemente o R tem sido cada vez mais utilizado como ferramenta interativa na web para usuários, ou seja, demonstrações de análises de forma que o usuário possa configurar opções e ver diferentes resultados em formas de gráficos e tabelas. Há pacotes gráficos que são especificamente desenvolvidos para interação na web como o plotly. Os pacotes que são elaborados e suportados pelo time do R são o 'graphics', o 'grid' e o 'lattice'. Aqui há alguns exemplos somente com o graphics que é o que já vem instalado e carregado toda vez que o R abre. A quantidade de comandos e opções é imensa, e nem de longe essa introdução irá mencionar muitas possibilidades. Novamente, a semelhança das demais tarefas, é possível chegar nos mesmos resultados de diferentes formas. Diferente da lógica orientada a objetos, a maioria dos comandos gráficos não retornam valores que possam depositados em objetos, mas abrem dispositivos gráficos (dispositivo é sinônimo de janela), que podem ser visualizados no monitor ou salvos no disco. Há os comandos denominados *high level* (mais próximos do usuário) que criam gráficos, e comandos denominados *low levels* (mais próximos da linguagem fundamental) que adicionam elementos aos dispositivos já abertos. Todos os gráficos são criados a partir de parâmetros que podem ser modificados pela função `par()`. No entanto, muitas das funções gráficas permitem que muitos dos argumentos sejam passados para `par()` sem que esta seja chamada com antecedência através dos argumentos '...' . É possível abrir mais de uma janela simultaneamente (`?x11`), porém isso cria a necessidade de escolher em qual janela deve-se fazer o gráfico (`?dev.curr ; ?dev.set`). Também é possível dividir uma janela permitindo assim que mais de um gráfico seja demonstrado na mesma janela. Isso pode ser feito de formas variadas

(`?split.screen` ou com os argumentos `mfrow` ou `mfcol` do `par()`). A flexibilidade é tanta que provavelmente não há gráfico que o R não faça, é apenas uma questão de saber como fazê-lo. No entanto, abaixo faremos alguns exemplos comuns a partir de um banco de dados.

A função `plot()` é a função mais básica para o gráficos. A semelhança das funções `summary()` e `print()`, `plot()` é uma função genérica e retorna coisas diferentes dependendo da classe ou conteúdo do objeto. A função `plot()` geralmente possui uma extensão oculta que se refere à classe do objeto. Por exemplo, se o usuário conduzir uma análise de sobrevivência e conduzir uma análise de resíduos com a função `cox.zph()`, um `plot()` desse objeto evoca a função `plot.cox.zph()` que gera um gráfico específico de resíduos oriundos dessa análise. Assim, se a ajuda do `plot()` for chamada, não haverá muita ajuda, pois há várias funções `plot()`, e a mais básica possui uma ajuda pobre. A ajuda do `plot.default()` será mais informativa. A função `plot()` abre um dispositivo e executa um gráfico. Os gráficos possuem parâmetros pré-estabelecidos, como margens, tamanho da janela, área da janela que será utilizada pelo gráfico, fontes, cores etc. Esses são utilizadas pela `plot()` mas são definidas pela função `par()`. Muitos dos parâmetros de `par()` podem ser passados da função utilizada (como `plot()`, `hist()` ou `legend()`) para `par()` através de argumentos '...'. A definição desses argumentos sempre são coisas que são passadas adiante para outras funções que são chamadas por uma função. Muito ocasionalmente há necessidade de modificar esses parâmetros com antecedência porque não há como passar adiante esses argumentos, assim é necessário nesses casos que `par()` seja evocada com antecedência ao gráfico. Para pedir ajuda para o pacote gráfico:

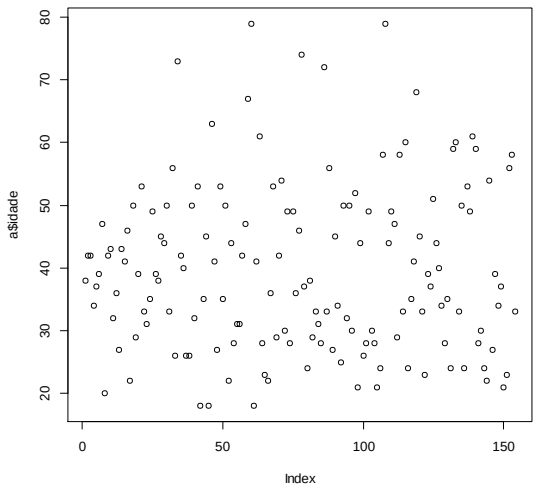
```
> help(package='graphics')
```

Começando com alguns demos de gráficos complexos, cores que os gráficos aceitam, como inserir algarismos e caracteres especiais e anotações matemáticas.

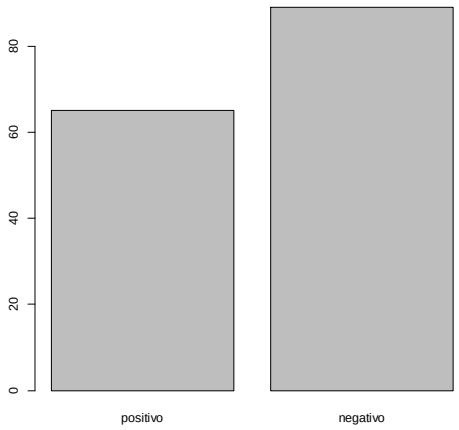
```
> demo('persp') # gráficos complexos em 3D
> demo('colors') # cores nos gráficos
> demo('Hershey') # como inserir algumas algarismos e símbolos
> demo('plotmath') # como inserir algumas anotações nos gráficos
> rm(list=ls())
```

Os exemplos de gráficos serão realizados utilizando-se o banco 'toxicidade'. Primeiro, algumas formas bem simples de gráfico serão apresentadas, passando por gráficos para uma única variável categórica, para uma única variável contínua, gráficos combinando 2 ou mais fatores, gráficos combinando fatores com variáveis numéricas, gráficos com duas ou mais variáveis contínuas, adicionando parâmetros comuns aos gráficos, adicionando pontos linhas e segmentos, adicionando legendas e finalmente salvando os gráficos no disco.

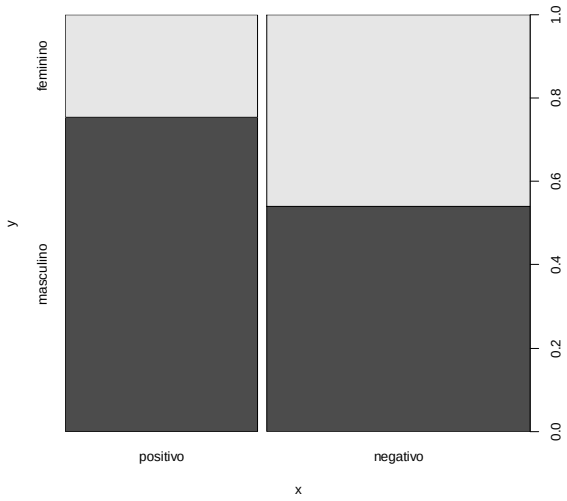
```
 > # setwd('c:/banco/curso_r')

> load('toxicidade.RData');ls()

[1] "a"

>

> > plot(a$idade) # uma variável contínua; eixo horizontal é o n da linha do banco
```



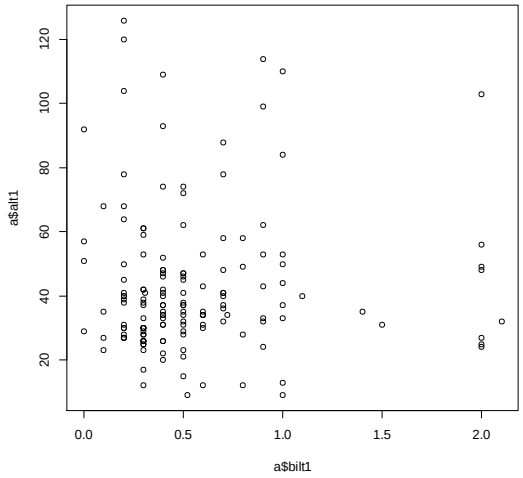

```
> plot(a$hiv) # uma variável categórica
```



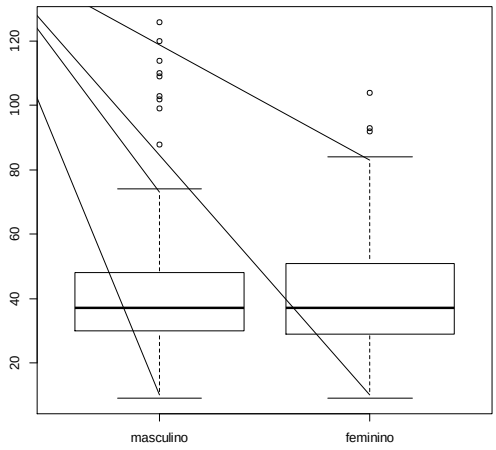

```
> plot(a$hiv,a$sexo) # fator com fator
```



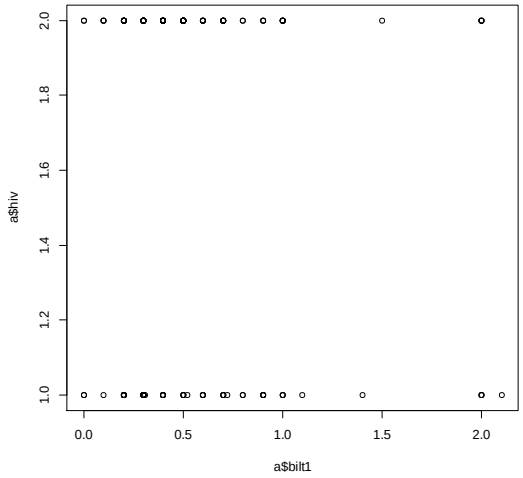

```
> plot(a$bilt1,a$alt1) # numérica com numérica
```



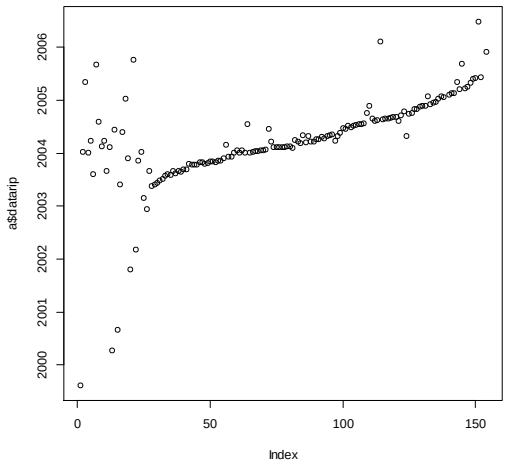

```
> plot(a$sexo,a$alt1) # fator com numérica
```



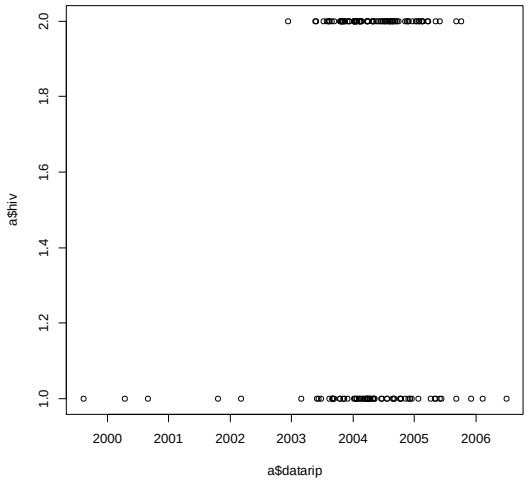

```
> plot(a$bilt1,a$hiv) # esse não faz muito sentido
```



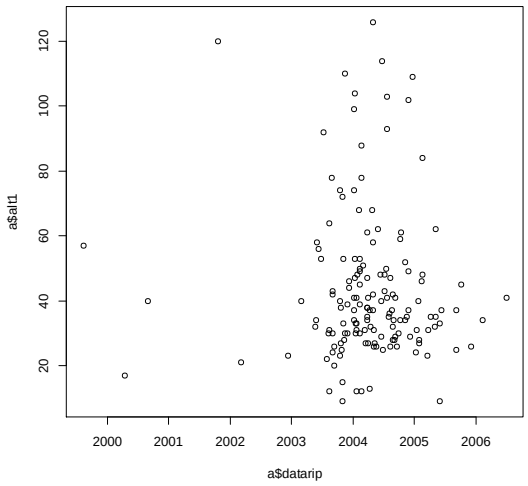

```
> plot(a$datarip)# data pelo índice da linha do banco de dados
```



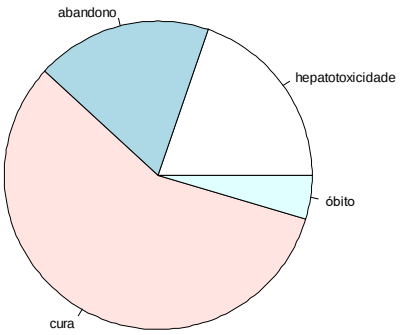

```
> plot(a$datarip,a$hiv) # data com fator não faz muito sentido
```



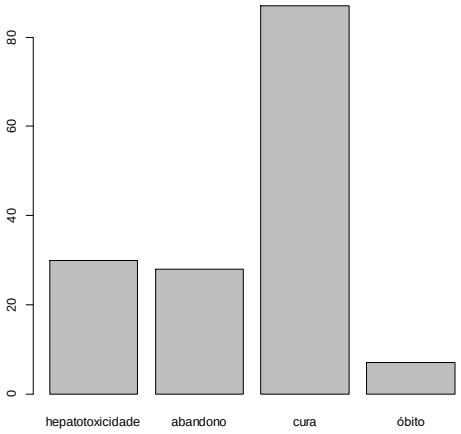

```
> plot(a$datarip,a$alt1) # data com numérico
```



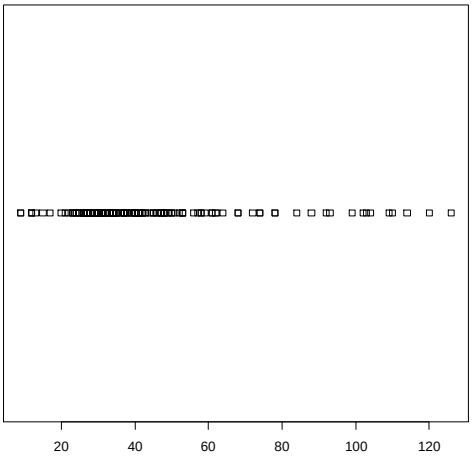

```
> # Gráficos para uma única variável fator

> pie(table(a$desfecho))
```



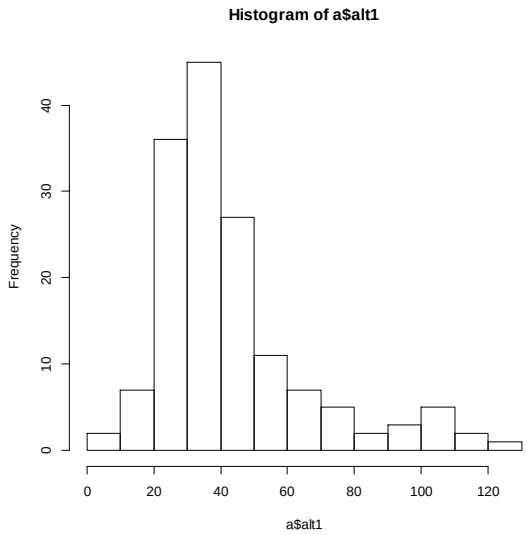

```
> barplot(table(a$desfecho)) # mesma coisa que plot(a$desfecho)
```



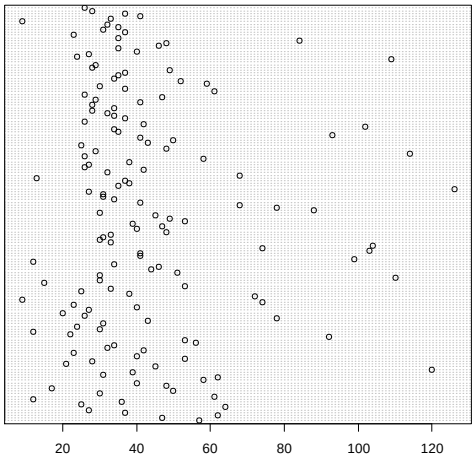

```
> # Graficos para uma única variável continua

> stripchart(a$alt1)# valores observados
```



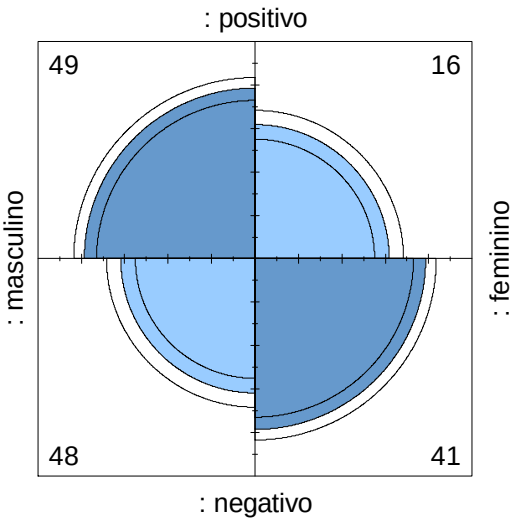

```
> hist(a$alt1) # coloca um título
```



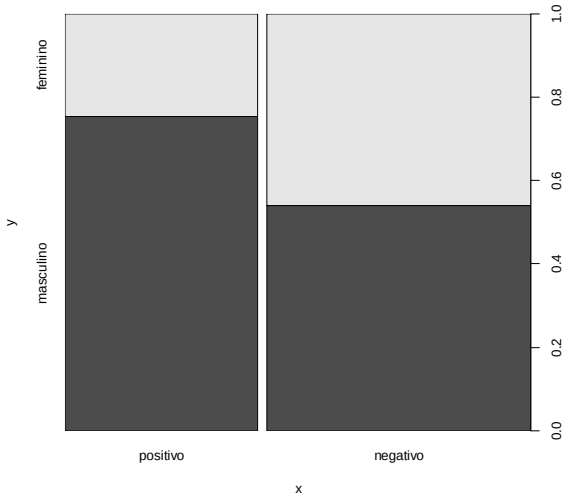

```
> dotchart(a$alt1) # o eixo vertical é a n da linha do banco de dados
```



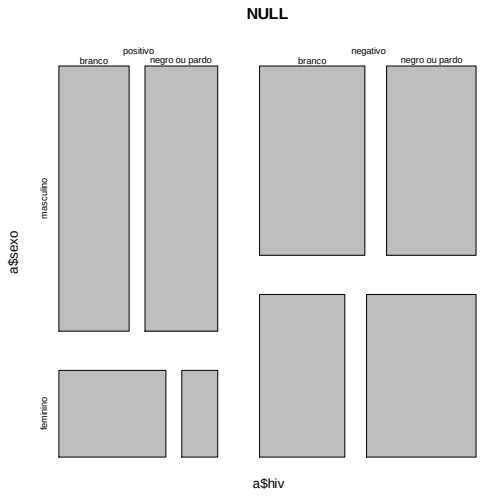

```
> # Gráficos de fatores ou categorias

> fourfoldplot(table(a$hiv,a$sexo))
```



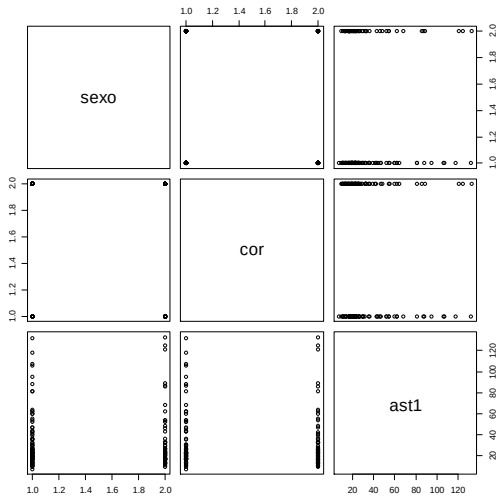

```
> plot(a$hiv,a$sexo)
```



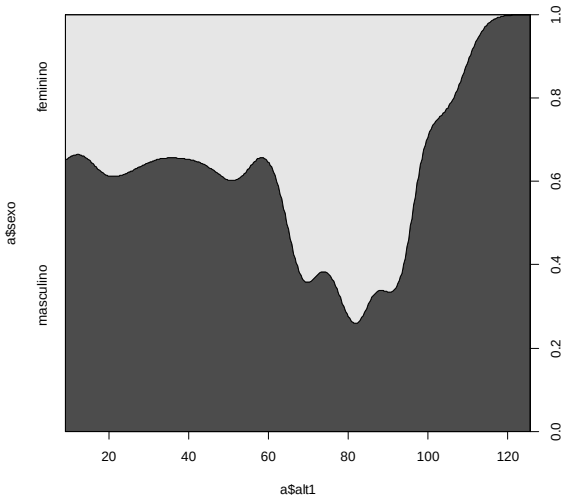

```
> mosaicplot(a$hiv ~ a$sexo + a$cor)
```



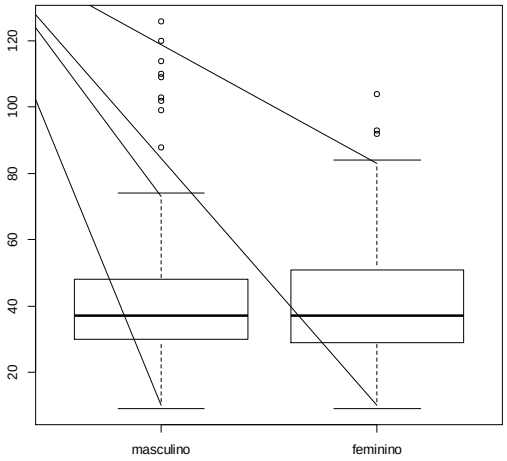

```
> plot(a[,c('sexo','cor','ast1')]) # plot.data.frame não faz sentido para fatores
```



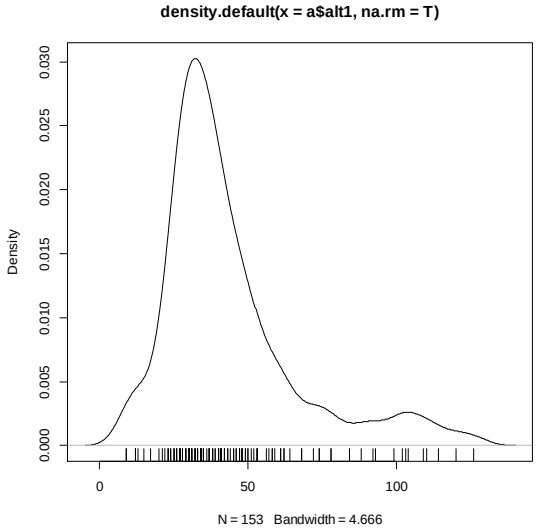

```
> # Gráficos de variáveis fatores com numéricas

> cdplot(a$sexo ~ a$alt1)
```



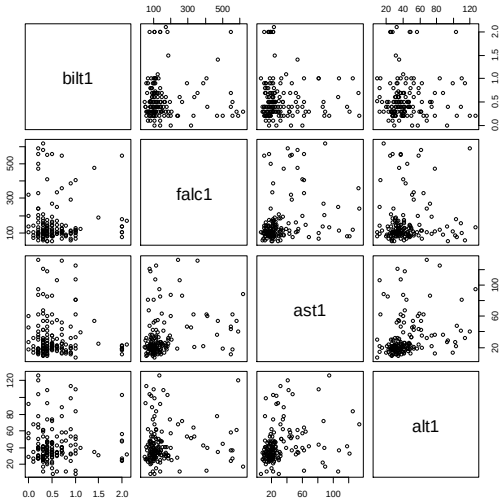

```
> boxplot(a$alt1 ~ a$sexo) # mesma coisa que plot(a$alt1 ~ a$sexo)
```



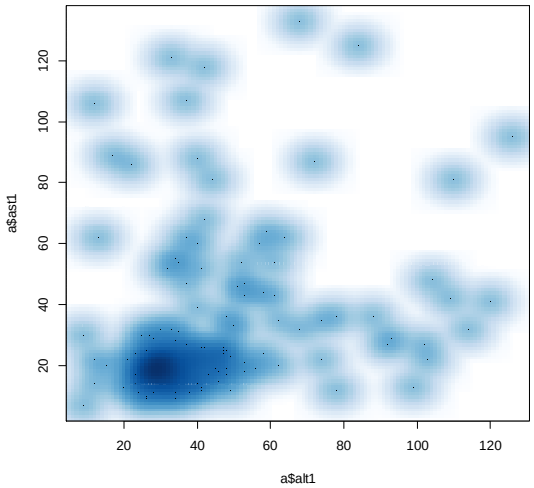

```
> plot(density(a$alt1,na.rm=T)) # um plot da função de densidade

> rug(a$alt1) # adiciona segmentos em cada observação
```



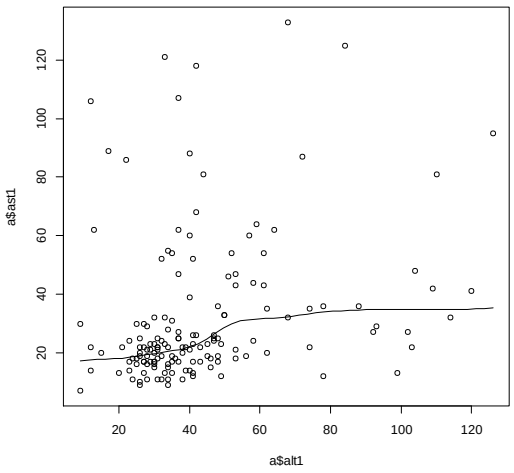

```
> # Gráficos para variáveis contínuas

> plot(a[,c('bilt1','falc1','ast1','alt1')]) # plot.data.frame() ou pairs()
```



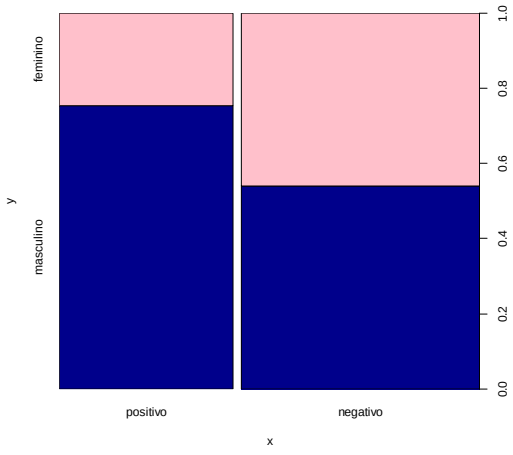

```
> smoothScatter(a$alt1,a$ast1)

KernSmooth 2.23 loaded

Copyright M. P. Wand 1997-2009
```




```
> scatter.smooth(a$alt1,a$ast1) # do pacote stats
```

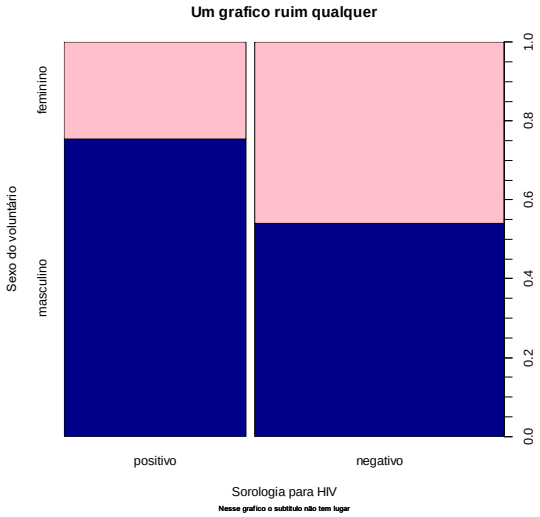



```
>
> # Adicionando parâmetros comuns
> # col= receber os argumentos de cores
> plot(a$hiv,a$sexo,col=c('blue4','pink'))
```

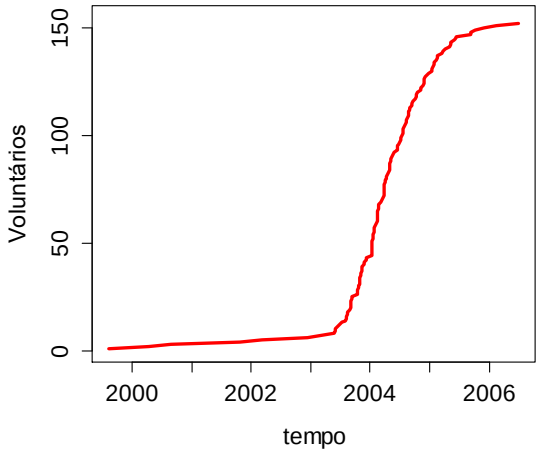



```
> # xlab= e ylab= recebem os nomes dos eixos e main o título principal
>  plot(a$hiv,a$sexo,col=c('blue4','pink'),xlab='Sorologia para HIV',ylab='Sexo do voluntário',main='Um grafico ruim qualquer')
> # adicionando um subtitulo e cex= o tamanho da fonte, no caso do subtitulo
> title(sub='Nesse grafico o subtitulo não tem lugar',cex.sub=.6)
> # adicionando eixos extras com a função axis
```

```
> axis(4,seq(0,1,.05),F)
```

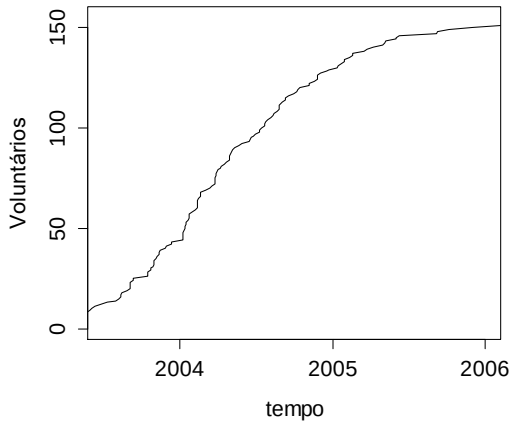



```
>

> a <- a[order(a$datarip),]

> # type= indica o tipo de grafico (linhas vs pontos)

> plot(a$datarip,1:nrow(a),type='l',xlab='tempo',ylab='Voluntários')

> # lwd= indica a espessura da linha e col a cor da linha

> plot(a$datarip,1:nrow(a),type='l',xlab='tempo',ylab='Voluntários',lwd=3,col=2)
```

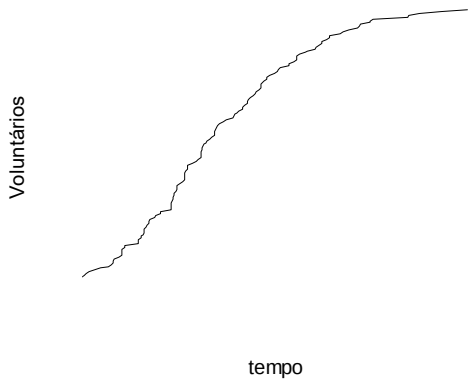



```
> # xlim= e ylim= indicam os limites do grafico nas escalas dos eixos

> plot(a$datarip,1:nrow(a),xlab='tempo',ylab='Voluntários',type='l',
```

```
+       xlim=c(as.Date('2003-07-01'),as.Date('2006-01-01')))
```

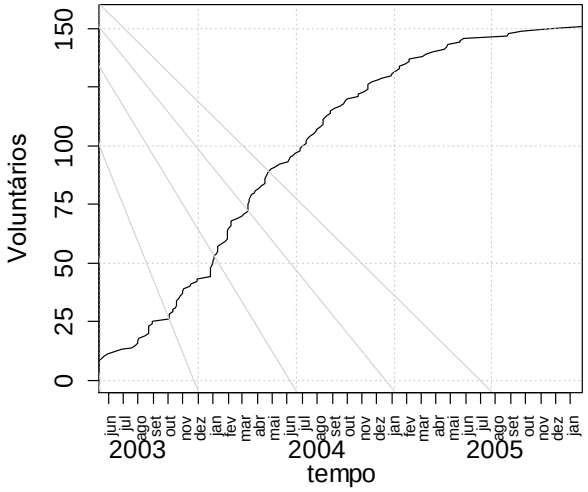



```
> # frame= retira/coloca a caixa em torno do grafico e axes os graficos
> plot(a$datarip,1:nrow(a),frame=F,axes=F,
+   xlab='tempo',ylab='Voluntários',type='l',
+   xlim=c(as.Date('2003-07-01'),as.Date('2006-01-01')))
> # acrescenta um eixo em outro formato, no caso o mes
> # las= orientação do texto, format= formato da data, line= quantas alinhas a parir da margem
```

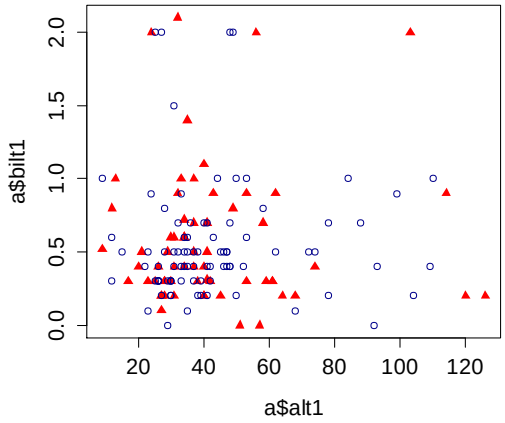



```
> axis.Date(1, at = seq(min(a$datarip,na.rm=T),max(a$datarip,na.rm=T), "months"),
+ format='%b', cex.axis=.6,las=2)
> axis.Date(1, at = seq(min(a$datarip,na.rm=T),max(a$datarip,na.rm=T), "years"),
```

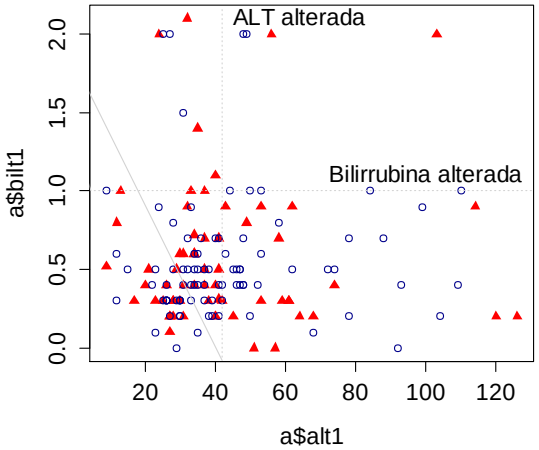
```
  + format='%Y',line=1,tick=F)
  > axis(2,seq(0,150,25))
  > # acrescenta a caixa ao redor do grafico e grade
  > box(); grid()
```

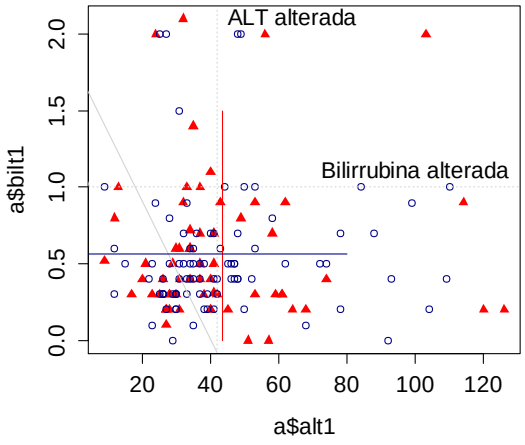

```
> # Adicionando pontos, linhas e segmentos
> # Faz um grafico como tamanho dos pontos = 0
> plot(a$alt1,a$bilt1,cex=0)
> # adiciona pontos somente para o grupo hiv positivo com cor vermelha
> points(a$alt1[which(a$hiv=='positivo')],a$bilt1[which(a$hiv=='positivo')],
+ pch=17,col='red')
> # adiciona pontos para o grupo HIV negativo da cor azul
> points(a$alt1[which(a$hiv=='negativo')],a$bilt1[which(a$hiv=='negativo')],
+ pch=1,col='darkblue')
```



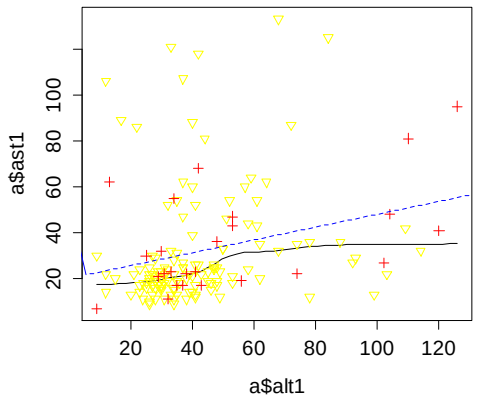

```
> # adiciona linhas representando os valores normais de alt e bilirrubina
> abline(h=1,col='lightgray',lty='dotted')
> abline(v=42,col='lightgray',lty='dotted')
> # Adiciona texto nas posições especificadas
> text(42,2.1,'Alt alterada',pos=4)
> text(100,1,Bilirrubina alterada',pos=3)
```




```
> # adiciona segmentos representando as médias das variáveis
> segments(mean(a$alt1,na.rm=T),0,mean(a$alt1,na.rm=T),1.5,col='red')
> segments(0,mean(a$bilt1,na.rm=T),80,mean(a$bilt1,na.rm=T),col='blue4')
```

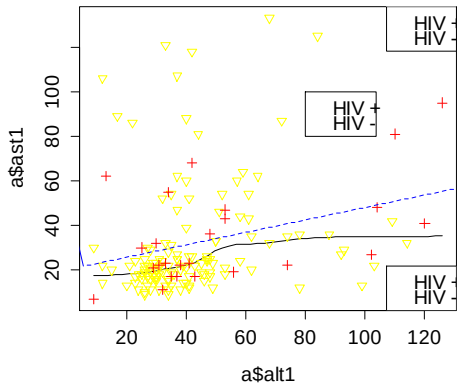



```
> # abre o grafico somente com a função de alisamento
> scatter.smooth(a$alt1,a$ast1,cex=0)
> # adiciona pontos como no exemplo anterior
> points(a$alt1[which(a$hepatoto=='negativo')],a$ast1[which(a$hepatoto=='negativo')],
+ pch=6,col='yellow')
> points(a$alt1[which(a$hepatoto=='positivo')],a$ast1[which(a$hepatoto=='positivo')],
+ pch=3,col='red')
> # adiciona uma linha com a relação linear entre as variáveis
> abline(lm(a$ast1 ~ a$alt1),lty=2,col=4)
```

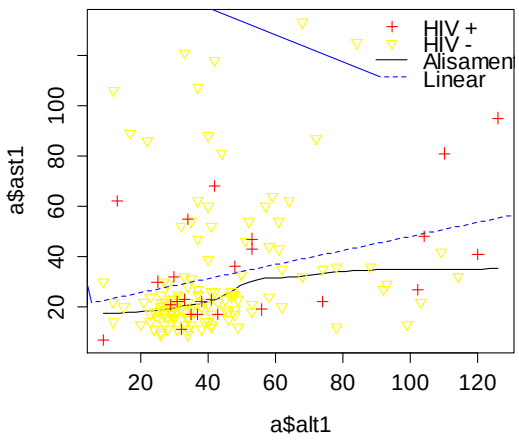



```
> # Adicionando legenda em diferentes posições
```

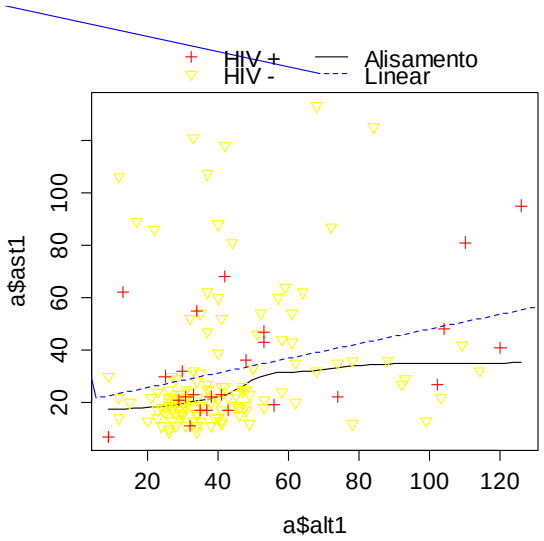
```
> legend('topright',c('HIV +','HIV -')) # no canto superior direito

> legend('bottomright',c('HIV +','HIV -')) # no canto inferior direito

> legend(80,100,c('HIV +','HIV -')) # numa posição específica
```


```
> # o argumento pch adiciona pontos

> # o argumento lty adiciona linhas

> # para ambos, NA não insere valores

> # repare que os pontos e as linhas são adicionados na mesma

> # ordem que o conteúdo da legenda

> legend('topright',c('HIV +','HIV -','Alisamento','Linear'),pch=c(3,6,NA,NA),

+ lty=c(NA,NA,1,2),col=c('red','yellow','black','blue'),bty='n')
```

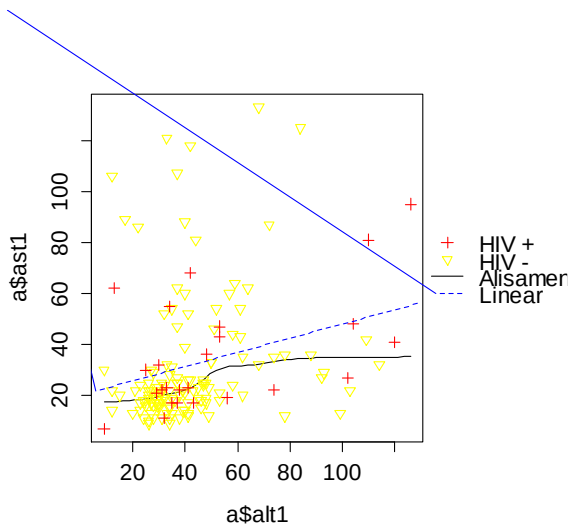


```
> # A mesma legenda acima do grafico
> scatter.smooth(a$alt1,a$ast1,cex=0)
> points(a$alt1[which(a$hepatoto=='negativo')],a$ast1[which(a$hepatoto=='negativo')],
+ pch=6,col='yellow')
> points(a$alt1[which(a$hepatoto=='positivo')],a$ast1[which(a$hepatoto=='positivo')],
+ pch=3,col='red')
> abline(lm(a$ast1 ~ a$alt1),lty=2,col=4)
> # xpd permite a inserção fora da área do grafico
> # inset é a distância da posição original
> # ncol é o número de colunas da legenda
> legend('top',c('HIV +','HIV -','Alisamento','Linear'),pch=c(3,6,NA,NA),
+ lty=c(NA,NA,1,2),col=c('red','yellow','black','blue'),bty='n',xpd=NA,inset=-.15,
+ ncol = 2)
```




```
> # A mesma legenda a direita do gráfico
> # necessário modificar a margem do gráfico
> par(mar = c(5.1, 4.1, 4.1, 7))
> scatter.smooth(a$alt1,a$ast1,cex=0)
```

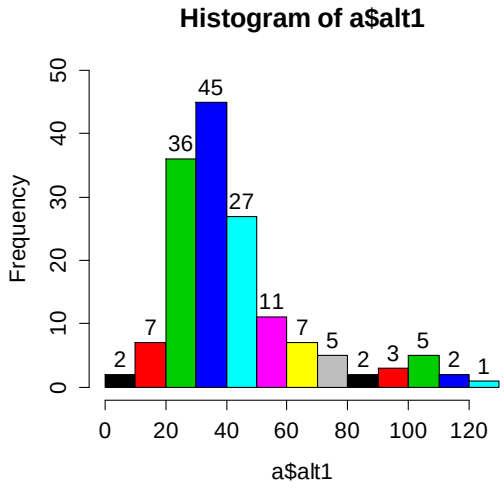
```
> points(a$alt1[which(a$hepatoto=='negativo')],a$ast1[which(a$hepatoto=='negativo')],
+ pch=6,col='yellow')
> points(a$alt1[which(a$hepatoto=='positivo')],a$ast1[which(a$hepatoto=='positivo')],
+ pch=3,col='red')
> abline(lm(a$ast1 ~ a$alt1),lty=2,col=4)
> # ncol é o número de colunas
> legend('right',c('HIV +','HIV -','Alisamento','Linear'),pch=c(3,6,NA,NA),
+ lty=c(NA,NA,1,2),col=c('red','yellow','black','blue'),bty='n',xpd=NA,inset=-.45)
> par(mar = c(5.1, 4.1, 4.1, 2.1)) # retornando a margem original
```



Depois de fazer algum gráfico conveniente para demonstrar um resultado, é necessário salva-lo. Assim como as outras tarefas salvar os gráficos pode ser realizado de diversas formas. Claro que o usuário pode clicar em cima do gráfico e pedir para copiar e colar num documento texto. Como comentado anteriormente, fazer por script é pratica melhor porque facilita os o incremento progressivo de parâmetros gráficos, e mantém a memória de como o gráfico foi feito. É possível salvar os gráficos em documentos com o realizado com as tabelas acima, apenas chamando as funções apropriadas para esse fim. É pratica comum de relatórios de pesquisa as figuras serem enviadas em arquivos separados para os periódicos científicos. Funções do pacote de 'grDevices' possivelmente utilizadas para esse fim são `dev.print()`, `dev.copy()`, `savePlot()`. No entanto aqui, vamos nos ater as funções que abrem dispositivos em formatos de fotografias. A lógica pode parecer um pouco estranha a princípio, mas na verdade é bastante simples. Ao invés de evocar uma função que abre um dispositivo em janela, o dispositivo é aberto no disco, em seguida os

comandos gráficos são inseridos nesse dispositivo, e posteriormente há necessidade de o usuário solicitar que o dispositivo seja fechado. Assim, os gráficos podem ser salvos em `bmp()`, `bitmap()`, `jpeg()`, `png()`, `tiff()`, e `pdf()`. Algumas demonstrações serão realizadas somente com uma dessas funções, já que todas seguem padrões semelhantes e os exemplos são equivalentes, apenas gerando arquivos com extensões diferentes. Há diversos parâmetros que podem ser modificados nessas funções, no entanto a maioria dos argumentos padrões é razoável. Por último o usuário deve entender que o padrão dessas funções é ter 480 pixels de uma altura por 480 pixels de largura, com uma resolução de 72 pontos por pixel. Assim, se houver necessidade de melhorar a resolução dessas imagens, na maioria das vezes, é necessário aumentar essas três dimensões proporcionalmente para que a figura não fique distorcida. Quando se salva um gráfico com essas funções acima, o gráfico não aparece em uma janela visível para o usuário, por isso é sempre interessante que depois do gráfico salvo, que usuário abra o arquivo para verificar se o gráfico ficou como esperado. Há também a alternativa de exportar os gráficos


```
> # Salvando os gráficos
> tiff('Histograma.tiff') # abre o dipositivo em tiff
> hist(a$alt1,col=1:16,labels=T,ylim=c(0,50)) # deposita o gráfico
> dev.off() # fecha o dispositivo
RStudioGD
        2
> file.show('Histograma.tiff')
>
> # melhora a definição em 5 vezes e insere compressão para reduzir o tamanho do arquivo
> tiff('Histograma.tiff',heigh=480*5,width=480*5,res=72*5, compression = "lzw")
> hist(a$alt1,col=1:16,labels=T,ylim=c(0,50)) # deposita o gráfico
> dev.off() # fecha o dispositivo
RStudioGD
        2
> file.show('Histograma.tiff')
```



18. Conteúdo do Exercício 3 foi coberto. Proceder para a avaliação.