Deivison Venicio Souza
Thiago Wendling Goncalves de Oliveira
Luani Rosa de Oliveira Piva
Joielan Xipaia dos Santos
Carlos Roberto Sanquetta
Ana Paula Dalla Corte

Introdução ao R

Aplicações Florestais

Deivison Venicio Souza
Thiago Wendling Gonçalves de Oliveira
Luani Rosa de Oliveira Piva
Joielan Xipaia dos Santos
Carlos Roberto Sanquetta
Ana Paula Dalla Corte

Introdução ao R:

Aplicações Florestais

Curitiba, PR
2018

Ficha catalográfica elaborada pela
Biblioteca de Ciências Florestais e da Madeira - UFPR

Introdução ao R: aplicações florestais / Deivison Venicio Souza... [et al.].
– Curitiba: Ed. do Autor, 2018.
136 p. : il., color.

Inclui referências.
ISBN: 9788591735730

1. R (Linguagem de programação de computador). I. Souza, Deivison Venicio. II. Oliveira, Thiago Wendling Gonçalves de. III. Piva, Luani Rosa de Oliveira. IV. Santos, Joielan Xipaia dos. V. Sanquetta, Carlos Roberto. VI. Dalla Corte, Ana Paula. VII. Título.

CDD – 005.262
CDU – 004.43

Bibliotecária: Berenice Rodrigues Ferreira – CRB 9/1160

Prefácio

Vivemos uma era em que a tecnologia está presente em praticamente todas as atividades do nosso contidiano e, portanto, inevitavelmente vem se tornando essencial em nossas vidas. Diariamente, novos computadores, celulares, aplicativos e softwares, cada vez mais rápidos e sofisticados, têm sido criados. Ademais, o aprendizado de alguma linguagem de programação é imprescindível para o desenvolvimento da ciência e das sociedades modernas em geral. O conhecimento de linguagens de programação possibilita uma infinidade de aplicações às mais diversas áreas do conhecimento.

Nesse contexto, o R é uma linguagem de programação muito poderosa e uma das mais utilizadas no cenário mundial por cientistas de dados. Além de ser uma linguagem de livre acesso e código aberto, é extremamente dinâmica e possui inúmeros contribuidores que auxiliam de forma coletiva no seu desenvolvimento. Através do uso da linguagem R é possível realizar análises estatísticas que auxiliam na interpretação e descrição dos problemas, criar e editar gráficos, gerar relatórios dinâmicos, criar aplicativos para web, implementar modelos de aprendizagem de máquina, realizar simulações, dentre inúmeras outras possibilidades.

O livro "Introdução ao R: Aplicações Florestais" nasceu do sentimento de aproximar a linguagem R dos discentes de graduação em Engenharia Florestal, e mostrar o seu potencial para análise de problemas e busca de soluções, de modo a auxiliar nsa tomadas de decisões no gerenciamento dos recursos florestais.

Nessa primeira edição, demos enfoque a uma abordagem mais introdutória ao R, com alguns exemplos de aplicações na área florestal, procurando despertar seu interesse e curiosidade acerca do grandioso potencial de uso dessa ferramenta.

Esperamos que os usuários apreciem a nossa abordagem nesta primeira edição e extraiam o máximo proveito possível dos conhecimentos e conceitos iniciais aqui apresentados. Por fim, enquanto autores e Engenheiros Florestais, nos alegramos em saber que esse material poderá ser utilizado em universidades, empresas, órgãos públicos, ou onde quer que o profissional e/ou estudante da área florestal esteja.

Os autores.

Sumário

1 O ambiente R

1.1 Um software estatístico livre

R é um ambiente de software livre de análises estatísticas e edição de gráficos, capaz de compilar e executar em uma ampla variedade de plataformas UNIX, Windows e MacOS. Para fazer o download do R, é necessário escolher um espelho CRAN para que seja feito o download da versão mais atual. Os **espelhos CRAN** são servidores distribuídos em diversos países que armazenam o software R. Assim, ao deixar escolher qual servidor será feito o download, permite-se que o usuário defina o servidor mais próximo de sua localização, reduzindo o tempo de tráfego.

2 Instalação do RGui e Rstudio

2.2 O RGui

0 processo de download e instalação do RGui é detalhado a seguir:

Passo 1: Acessar a página do projeto R em https://www.r-project.org/;

Passo 2: Do lado esquerdo da página clique sobre o link **CRAN**;

Passo 3: Será aberta uma página com diversos links de **CRAN Mirrors**, isto é, espelhos CRAN. O ideal é selecionar o servidor mais próximo da sua localização para fazer o download do **R Development Core Team**. Veja na tabela os principais espelhos disponíveis no Brasil.

Link	Instituição
http://cran-r.c3sl.ufpr.br/	Universidade Federal do Paraná (UFPR)
http://nbcgib.uesc.br/mirrors/cran/	Center for Comp Biol at Universidade Estadual de Santa Cruz
https://cran.fiocruz.br/	Oswaldo Cruz Foundation, Rio de Janeiro
http://cran.fiocruz.br/	Oswaldo Cruz Foundation, Rio de Janeiro
https://vps.fmvz.usp.br/CRAN/	University of São Paulo, São Paulo
http://brieger.esalq.usp.br/CRAN/	University of São Paulo, Piracicaba

Passo 4: Na página http://cran-r.c3sl.ufpr.br/, na seção **Download and Install R**, clicar em um dos três links, conforme o sistema operacional (SO) do usuário:
- Download R for Windows;
- Download R for Linux; ou
- Download R for (Mac) OS X.

Passo 5: Clicar no link do **subdiretório base** ou em **install R for the first time**, para instalar o R pela primeira vez;

Passo 6: Clicar em **Download R 3.3.2 for (escolher SO)**. Assim, será iniciado o download do R Development Core Team para o respectivo sistema; e

Passo 7: Por fim, basta usar o setup para instalar o programa.

2.3 O editor Rstudio

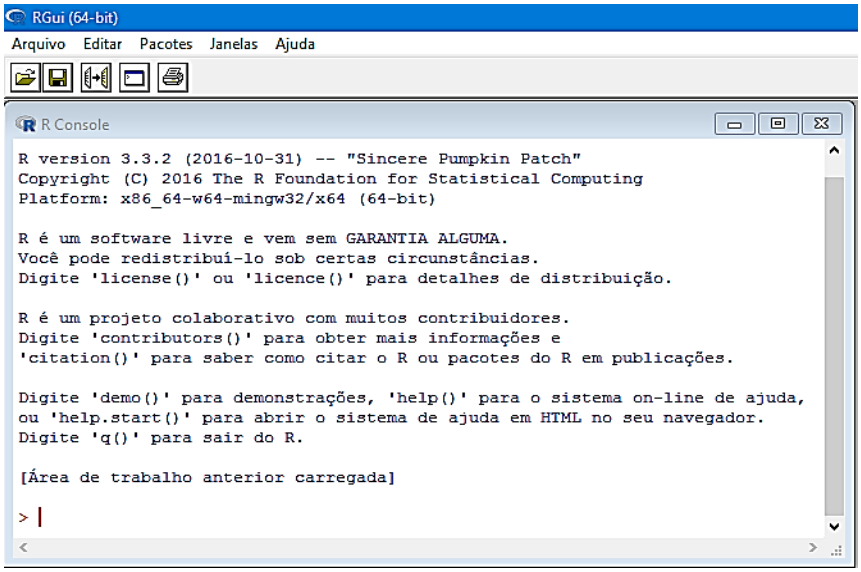
A seguir o processo de download e instalação do editor Rstudio:

Passo 1: Acessar a página do projeto RStudio: https://www.rstudio.com;
Passo 2: Products → RStudio;
Passo 3: Selecionar a versão do RStudio para Desktop;
Passo 4: Na edição Open source → Download Rstudio Desktop;
Passo 5: Installers for Supported Platforms → instalador RStudio; e
Passo 6: Por fim, basta usar o setup baixado para instalar o programa.

3 Iniciando no ambiente R

3.1 A janela inicial do RGui

No contato inicial do usuário com o **RGui** tem-se a visão inicial do **R Console** com a versão do R em uso e as condições de licenciamento. Algumas funções de teste são mostradas e, para saber o que cada função retorna basta digitá-las no **prompt de comando** do R Console simbolizado pelo sinal > (maior) em vermelho. Haverá um **cursor** piscando à direita do prompt, indicando o local para digitar os comandos para o R. Execute as funções demo(), help(), help.start() e q().



3.2 Operações e operadores aritméticos

A linguagem R permite executar operações aritméticas básicas (soma, subtração, multiplicação, divisão e potenciação):

Operadores	Nome	Operações
+	Somatório	2+3
*	Multiplicação	4*9
/	Divisão	20/5
-	Subtração	32-10
^ ou **	Potenciação	5^3
%%	Resto inteiro da divisão	10%%3
%/%	Parte inteira da divisão	10%/%3

```
2+3                          # Soma
## [1] 5
4*9                          # Multiplicação
## [1] 36
20/5                         # Divisão
## [1] 4
32-10                        # Subtração
## [1] 22
5^3                          # Potenciação
## [1] 125
10%%3                        # Resto inteiro da divisão
## [1] 1
10%/%3                       # Parte inteira da divisão
## [1] 3
```

3.3 Operadores lógicos

Alguns dos principais operadores lógicos estão descritos na tabela a seguir:

Operadores lógicos	Descrição
<	Menor que (…)
>	Maior que (…)
<=	Menor ou igual que (…)
>=	Maior ou igual que (…)
==	Igual à (…)
&	E (and)/para vetores
|	Ou (or)/para vetores
!=	Diferente de (…)
!	Não (…)
is.na()	Valor numérico ou faltante

3.4 Alguns comandos básicos

Comando	Ação
ls() ou objects()	Listar os objetos na janela de trabalho atual
rm()	Remover um objeto qualquer
help()	Solicitar ajuda sobre o uso de uma função
Ctrl + L	Limpar a tela do R console
history()	Listar os últimos comandos executados
getwd()	Mostrar o diretório de trabalho
setwd()	Mudar o diretório de trabalho
install.packages()	Instalar um pacote específico
q()	Fechar o console R
library()	Carregar um pacote específico
dir()	Listar os arquivos existentes no diretório
getOption("OutDec")	Verificar o separador decimal definido
options(OutDec=",")	Mudar o separador decimal para vírgula
round(x, digits=0)	Função para arredondamento de casas decimais
data()	Listar os conjuntos de dados dos pacotes atualmente carregados
?nomedodataset	Obter informações detalhadas sobre um conjunto de dados
class()	Verificar a classe de um objeto específico
search()	Listar todos os pacotes carregados

4 Estrutura de dados no R

Os objetos são criados no R com o objetivo de **armazenar dados**. Todo objeto em R tem uma classe, que pode ser descoberta usando a função class(). Os objetos-vetores podem ser do tipo **numeric**, **logical**, **character**, etc. Outros objetos incluem **matrizes**, **data frames**, **funções**, **listas**, **arrays**.

Antes de iniciar as seções que detalham os tipos de objetos existentes no R é importante saber como nomeá-los. Assim, podemos listar algumas condições para atribuição de nomes a objetos:

a) O nome do objeto deve, necessariamente, iniciar com uma letra;
b) Pode-se usar letras maiúsculas ou minúsculas no nome do objeto;
c) Os nomes dos objetos são sensíveis às letras maiúsculas ou minúsculas (ex: Ufpr/ufpr). Em outras palavras, diz-se que o R é *case sensitive*;
d) O nome não pode ter símbolos de funções ou operações matemáticas (+; /; -; *; ^);
e) Números (0 a 9) podem ser inseridos no nome do objeto, exceto na primeira posição;
f) Não se pode usar espaço entre os nomes dos objetos (alternativa = usar ponto (.) ou underline (_);
g) Para atribuir o nome a um objeto deve-se usar o comando <- (recebe); e
h) Pode-se usar também a função assign() para fazer atribuição (menos comum).

4.1 Vetores

O vetor é o tipo de objeto mais importante em R, constituindo a forma mais simples de armazenar dados. Os vetores são caracterizados por possuírem somente uma dimensão, sendo que todos os seus elementos constituintes devem ter, obrigatoriamente, a mesma natureza (classe).

Os vetores podem ser considerados como células contíguas que contém dados. Estas células podem ser acessadas por meio de operações de indexação (R Language Definition). As principais classes de vetores são: a) numeric; b) character; c) integer; d) logical; e) complex; e f) factor.

Quais funções podem ser utilizadas para criar um vetor?

1. **Função** c() (concatenate): Função genérica que permite concatenar (combinar) argumentos para formar um vetor.
2. **Função** seq(): Função genérica usada para gerar sequências de números em intervalos pré-definidos.

Função seq() - sequence

seq(**from** = ?, **to** = ?, **by** = ?, **length.out** = ?, **along.with** = ?)

from = define um valor inicial (start) para a sequência;
to = define um valor máximo para a sequência;
by = define um valor incremental para a sequência;
length.out = define o comprimento da sequência dentro de um intervalo; e
along.with = define uma sequência de inteiros do tamanho do argumento repassado.

3. **Função** rep(): Função genérica usada para replicar um valor **"x"**.

Função rep() - replicate

rep(**x** = ?, **times** = ?, **each** = ?)

x = escalar, vetor ou uma lista, entre outros;
times = um vetor de inteiro (não negativo) indicando quantas vezes repetir cada elemento; e
each = cada elemento de **x** é repetido **n** vezes.

d) Função scan(): Usada para criar vetores diretamente no **R Console**.

Existe uma sutil diferença ao utilizar a função concatenate para criar vetores de **números** ou de **caracteres**:

- **Vetor numérico:** Deve-se concatenar os números separando-os por **vírgula**; e
- **Vetor de caractere:** Deve-se escrever cada elemento entre **aspas** e separá-los por **vírgula**.

a) Função c() (concatenate):

```
# Criando objetos-vetores e atribuindo nomes.

c("Acapu", "Araucaria", "Mogno", "Cedro", "Ipe")
assign("especie", c("Acapu", "Araucaria", "Mogno", "Cedro", "Ipe"))
especie <- c("Acapu", "Araucaria", "Mogno", "Cedro", "Ipe")

c(23.0, 27.0, 33.6, 42.6, 52.1)
diametro <- c(23.0, 27.0, 33.6, 42.6, 52.1)

c(8.5, 9.2, 10.5, 13.4, 15.8)
altura <- c(8.5, 9.2, 10.5, 13.4, 15.8)
```

Obs.: A função assign() também pode ser usada para concatenar.

Elementos de diferentes naturezas no mesmo vetor?

Você já sabe que um vetor é constituído por elementos de uma mesma classe (natureza). Então, o que aconteceria caso tentássemos criar um vetor com elementos de diferentes naturezas? Neste caso, o R fará com que todos os elementos do vetor tenham a mesma natureza. Para exemplificar, considere a criação de um vetor **x** com dois tipos de elementos: **numérico** e **caractere.**

```
x <- c(42.6, 23.0, 27.0, 33.6, "Acapu")
print(x)

## [1] "42.6"  "23"    "27"    "33.6"  "Acapu"

class(x)

## [1] "character"
```

Perceba que, na existência de um elemento do tipo **caractere** (strings), todos os elementos numéricos foram transformados para caractere (elementos ficaram entre aspas!). Assim, a classe do vetor foi coagida de **numeric** para **character**. Isso significa, por exemplo, que o primeiro elemento do vetor **x** é o caractere "42.6" e não o número 42.6. Como consequência, operações matemáticas não poderiam ser aplicadas sobre este vetor.

b) Função seq() - (sequence):

seq(**from** = ?, **to** = ?, **by** = ?, **length.out** = ?, **along.with** = ?)

```
especie <- c("Acapu", "Araucaria", "Mogno", "Cedro", "Ipe")

seq(10)                                  # Sequência de 1 a 10, com intervalo 1.
##  [1]  1  2  3  4  5  6  7  8  9 10

seq(1:10)                                # Sequência de 1 a 10, com intervalo 1.
##  [1]  1  2  3  4  5  6  7  8  9 10

seq(from = 1, to = 10, by = 1)           # Sequência de 1 a 10, com intervalo 1.
##  [1]  1  2  3  4  5  6  7  8  9 10

seq(from = -2, to = 10, by = 2)          # Sequência de -2 a 10, com intervalo 2.
## [1] -2  0  2  4  6  8 10

seq(from=0, to=1, length.out = 11)       # Sequência de 11 números entre 0 e 1.
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

seq(along.with = especie)                # Sequência do tamanho do vetor-espécie.
## [1] 1 2 3 4 5
```

c) Função rep() (replicate):

rep(**x** = ?, **times** = ?, **each** = ?)

```
rep(x = 1:4, 2)                          # Repete a sequência de 1 a 4 (2x).
## [1] 1 2 3 4 1 2 3 4

rep(1:4, times = 3)                      # Repete a sequência de 1 a 4 (3x).
##  [1] 1 2 3 4 1 2 3 4 1 2 3 4

rep(x = 1:4, each = 2)                   # Repete cada valor em "x" (2x).
## [1] 1 1 2 2 3 3 4 4

rep(1:4, c(2,2,2,2))                     # Repete cada valor em "x" (2x).
## [1] 1 1 2 2 3 3 4 4

rep(1:4, c(2,1,2,1))                     # Repete cada valor em "x", com base em c().
## [1] 1 1 2 3 3 4

rep(1:4, each = 2, len = 4)              # Repete cada valor em "x", até 4 números.
## [1] 1 1 2 2

rep(1:4, each = 2, len = 10)             # Regra da reciclagem.
##  [1] 1 1 2 2 3 3 4 4 1 1

rep(1:4, each = 2, times = 3)            # Repete cada valor em "x" (2x), por 3x.
##  [1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
```

d) Função scan(): Crie os vetores abaixo usando a função scan().

```
especie <- c("Acapu", "Araucaria", "Mogno", "Cedro", "Ipe")

diametro <- c(23.0, 27.0, 33.6, 42.6, 52.1)

altura <- c(8.5, 9.2, 10.5, 13.4, 15.8)
```

Como identificar a classe de um objeto-vetor no R?

A identificação da classe de um objeto-vetor no R pode ser feita usando a função class(). Além disso, existem funções lógicas que testam a classe de um objeto-vetor: is.numeric(); is.character(); is.factor(); is.integer(); is.logical().

a) Objeto-vetor da classe "numeric"

```
a <- 45; class(a); is.numeric(a)
## [1] "numeric"
## [1] TRUE

b <- pi; class(b); is.numeric(b)
## [1] "numeric"
## [1] TRUE

c <- sqrt(2); class(c); is.numeric(c)
## [1] "numeric"
## [1] TRUE

diametro <- c(23.0, 27.0, 33.6, 42.6, 52.1)
class(diametro); is.numeric(diametro)
## [1] "numeric"
## [1] TRUE
```

b) Objeto-vetor da classe "character"

```
d <- "olá Mundo"; class(d); is.character(d)
## [1] "character"
## [1] TRUE

e <- c("Quem vai ao Pará", "Parou", "tomou açai, ficou!")
class(e); is.character(e)
## [1] "character"
## [1] TRUE

especie <- c("Acapu", "Araucaria", "Mogno", "Cedro", "Ipe")
class(especie); is.character(especie)
## [1] "character"
## [1] TRUE
```

*Observação: os comandos do R podem ser escritos em uma mesma linha, contudo devemos separá-los com o símbolo " ; " conforme feito nos exemplos acima.

c) Objeto-vetor da classe "factor"

A função factor() é usada para codificar um vetor como um "fator" (categorias). Outra alternativa é usar a função de conversão as.factor(). Um dos usos mais importantes dos fatores está na modelagem estatística. As variáveis categóricas entram em modelos estatísticos de maneira diferente das variáveis contínuas. Assim, o armazenamento de dados como fatores garante que as funções de modelagem tratem esses dados corretamente.

Os fatores em R são armazenados como um vetor de valores inteiros com um conjunto correspondente de valores de caractere a serem usados quando o fator é exibido. Os níveis (levels) de um fator sempre serão valores de caracteres.

No exemplo a seguir, observe que o objeto-vetor "cortar" ao ser impresso no console do R pertence à classe **character**, sendo que todos os seus valores são mostrados entre aspas. Iremos usar a função factor() para transformar o objeto-vetor para a classe **factor**. Feito isso, você perceberá que os elementos não serão impressos entre aspas e que, ainda, que serão transformados em níveis de fator, no caso **Não** e **Sim**.

```
cortar <- c("Não", "Não", "Não", "Não", "Sim", "Sim")
print(cortar); class(cortar)
## [1] "Não" "Não" "Não" "Não" "Sim" "Sim"
## [1] "character"

fcortar <- factor(cortar)
print(fcortar)
## [1] Não Não Não Não Sim Sim
## Levels: Não Sim
```

Os rótulos (nomes) dos níveis de fatores podem ser alterados com uso do argumento **labels**:

```
fuste <- c(1, 2, 2, 3, 1, 2, 3, 3, 1, 2, 3, 3, 1, 4, 4)
print (fuste); class(fuste)
##  [1] 1 2 2 3 1 2 3 3 1 2 3 3 1 4 4
## [1] "numeric"

ffuste <- factor(x = fuste,
            labels = c("I","II","III", "IV"))
print(ffuste)
##  [1] I   II  II  III I   II  III III I   II  III III I   IV  IV
## Levels: I II III IV

levels(ffuste)
## [1] "I"   "II"  "III" "IV"
```

Por default os fatores (levels) ficam dispostos em ordem alfabética:

```
# Fatores (levels) em ordem alfabética!

mes <- c ("Março", "Abril", "Janeiro", "Novembro",
        "Agosto", "Setembro", "Outubro", "Julho",
        "Dezembro", "Fevereiro", "Maio", "Junho")
class(mes)
## [1] "character"

fmes <- factor(mes)
class(fmes)
## [1] "factor"

print(fmes)
##  [1] Março    Abril    Janeiro  Novembro Agosto   Setembro Outubro
##  [8] Julho    Dezembro Fevereiro Maio    Junho
## 12 Levels: Abril Agosto Dezembro Fevereiro Janeiro Julho Junho ... Setembro
```

Pode-se declarar que existe uma hierarquia entre os fatores usando ordered = **TRUE**.

```
# Fatores ordenados
fmes2 <- factor(x = mes,
       levels = c("Janeiro", "Fevereiro", "Março", "Abril",
                  "Maio", "Junho", "Julho", "Agosto",
                  "Setembro", "Outubro", "Novembro", "Dezembro"),
           ordered = TRUE);
class(fmes2)
## [1] "ordered" "factor"
print(fmes2)
##  [1] Março    Abril    Janeiro  Novembro Agosto   Setembro Outubro
##  [8] Julho    Dezembro Fevereiro Maio    Junho
## 12 Levels: Janeiro < Fevereiro < Março < Abril < Maio < ... < Dezembro
```

Elementos especiais

Todo vetor pode possuir elementos especiais: **NA** (*Not Available*). Esta **string** dentro de um vetor indica um valor desconhecido. Por exemplo, imagine que você mediu a altura de algumas árvores em um povoamento florestal, porém 2 árvores não tiveram a altura medida. Poderíamos especificar um vetor de alturas sem desconsiderar a existência das duas árvores que faltaram:

```
# Not Available
altura <- c(5.6, 6.7, NA, 8.9, 2.3, 5.6, 7.8, 8.9, 6.1, NA)
print(altura)
##  [1] 5.6 6.7  NA 8.9 2.3 5.6 7.8 8.9 6.1  NA
length(altura)
## [1] 10
is.na(alturas)
##  [1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
```

4.2 Matrizes

As matrizes diferenciam-se dos vetores por admitirem duas dimensões, expressas por **linhas** e **colunas**. Uma matriz pode ser formada por elementos **numéricos** e/ou **caracteres** (strings) na sua estrutura.

A construção de matrizes no R pode ser feita com uso das funções rbind() (*row bind*) e cbind() (*colum bind*). As funções rbind() e cbind() organizam objetos-vetores em **linhas** ou **colunas**, respectivamente. No entanto, o mecanismo mais eficiente para construir matrizes é por meio da função matrix().

Quando utilizadas as funções rbind() e cbind() para construir matrizes a partir de vetores de diferentes classes (ex.: mat.2 e mat.4), todos os dados numéricos da matriz são convertidos em **caracteres**, fato evidenciado pelos números entre aspas na saída.

a) Função rbind()

```
especie <- c("Acapu", "Araucaria", "Mogno", "Cedro", "Ipe")
diametro <- c(23.0, 27.0, 33.6, 42.6, 52.1)
altura <- c(8.4, 8.7, 9.1, 13.2, 15.4)

mat.1 <- rbind(diametro, altura)
print(mat.1)
##           [,1]  [,2]  [,3]  [,4]  [,5]
## diametro 23.0  27.0  33.6  42.6  52.1
## altura    8.4   8.7   9.1  13.2  15.4

mat.2<-rbind(especie, diametro)
print(mat.2)
##                [,1]          [,2]          [,3]        [,4]        [,5]
## especie    "Acapu"    "Araucaria"    "Mogno"    "Cedro"    "Ipe"
## diametro    "23"          "27"          "33.6"      "42.6"      "52.1"
```

b) Função cbind()

```
mat.3 <- cbind(diametro, altura)
print(mat.3)
##      diametro  altura
## [1,]     23.0     8.4
## [2,]     27.0     8.7
## [3,]     33.6     9.1
## [4,]     42.6    13.2
## [5,]     52.1    15.4

mat.4 <- cbind(especie, diametro, altura)
print(mat.4)
##           especie      diametro     altura
## [1,]      "Acapu"        "23"       "8.4"
## [2,]      "Araucaria"    "27"       "8.7"
## [3,]      "Mogno"        "33.6"     "9.1"
## [4,]      "Cedro"        "42.6"     "13.2"
## [5,]      "Ipe"          "52.1"     "15.4"
```

c) Função matrix()

Uma matriz pode ser gerada no R, de maneira prática, por meio da função matrix(). Para facilitar a identificação, pode-se atribuir rótulos às linhas e colunas da matriz. Para tanto, pode-se usar as funções rownames() e colnames() ou o argumento **dimnames** disponível como argumento em matrix().

Função matrix()

matrix(**data** = NA, **nrow** = 1, **ncol** = 1, **byrow** = FALSE, **dimnames** = NULL)

data = um vetor de dados opcional;
nrow = número desejado de linhas;
ncol = número desejado de colunas;
byrow = argumento lógico. Se FALSE (default), a matriz é preenchida por colunas. Se TRUE, a matriz é preenchida por linhas; e
dimnames = Insere os rótulos das linhas e colunas da matriz com a função list().

```
# Cria uma matriz de 1 a 6, com 2 linhas e 3 colunas
mat.5 <- matrix(1:6, nrow=2, ncol=3)
print(mat.5)
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6

# Atribui nomes às linhas e colunas da "mat.5"
rownames(mat.5) <- c("L1","L2")
print(mat.5)
##    [,1] [,2] [,3]
## L1    1    3    5
## L2    2    4    6

colnames(mat.5) <- c("C1","C2","C3")
print(mat.5)
##    C1 C2 C3
## L1  1  3  5
## L2  2  4  6
```

Pode-se usar objetos-vetores que já existem na função matrix():

```
# Criar uma matriz a partir de objetos-vetores existente
especie <- c("Acapu", "Araucaria", "Mogno", "Cedro", "Ipe")
diametro <- c(23.0, 27.0, 33.6, 42.6, 52.1)

mat.6 <- matrix(diametro, nrow=5, ncol=1, byrow = TRUE,
          dimnames = list(especie))

print(mat.6)
##              [,1]
## Acapu        23.0
## Araucaria    27.0
## Mogno        33.6
## Cedro        42.6
## Ipe          52.1
```

Atribuindo um nome à única coluna da matriz 6 usando colnames():

```
colnames(mat.6) <- "diâmetro"

print(mat.6)
##              diâmetro
## Acapu            23.0
## Araucaria        27.0
## Mogno            33.6
## Cedro            42.6
## Ipe              52.1
```

Atribuindo nomes às linhas e colunas com o argumento **dimnames** da função matrix():

```
mat.7 <- matrix(1:6, nrow=2, ncol=3, byrow = TRUE,
          dimnames = list(c("L1", "L2"),
                          c("C1", "C2", "C3")))
print(mat.7)
##       C1 C2 C3
## L1     1  2  3
## L2     4  5  6
```

Situações especiais

Na criação de matrizes é interessante atentar-se para algumas situações especiais:

a) Descarte de elementos: Quando a quantidade de elementos (**n**) for maior que a quantidade de linhas e colunas (**ncol x nrow**); e

b) Regra da Reciclagem: Quando a quatidade de linhas e colunas (**ncol** x **nrow)** for maior do que a quantidade de elementos (**n**).

```
# Número de elementos > ncol x nrow (Descarte)
mat.8 <- matrix(1:9, nrow=2, ncol=3, byrow = TRUE,
          dimnames = list(c("L1", "L2"),
                          c("C1", "C2", "C3")))
## Warning in matrix(1:9, nrow = 2, ncol = 3, byrow = TRUE, dimnames =
## list(c("L1", : comprimento dos dados [9] não é um submúltiplo ou múltiplo
## do número de linhas [2]
print(mat.8)
##       C1  C2  C3
## L1     1   2   3
## L2     4   5   6

# ncol x nrow > número de elementos (Reciclagem)
mat.9 <- matrix(1:6, nrow=3, ncol=3, byrow = TRUE,
          dimnames = list(c("L1", "L2", "L3"),
                          c("C1", "C2", "C3")))
print(mat.9)
##       C1  C2  C3
## L1     1   2   3
## L2     4   5   6
## L3     1   2   3
```

4.3 Data Frames

A estrutura **data frame** é bastante similar à **matriz**. Entretanto, a principal diferença entre essas duas estruturas é que no **data frame** pode-se reunir vetores

de diferentes classes (naturezas), com a condição de possuírem **igual comprimento**.

Para criar **data frames** diretamente no R pode-se usar das funções data.frame() ou edit(). Em geral, quando se importa dados para o R, oriundos de um arquivo .CSV ou .txt pelos comandos **read.csv**() e **read.table**(), respectivamente, o R armazena-os no formato de um data frame.

Após criar um **data frame**, é interessante avaliar a sua estrutura por meio das funções str() e dim(). A função str() retorna as colunas com suas respectivas classes e número de observações por variável. A função dim() informar as dimensões do **data frame** (número de linhas e colunas).

Ainda, deve-se notar que a função data.frame() transforma, por *default*, as variáveis qualitativas em fatores (classe factor) com a especificação da quantidade de níveis (*levels*). Isso ocorre devido à argumentação stringsAsFactors = **TRUE** que por padrão transforma qualquer variável qualitativa em fator. Caso a transformação não seja desejável, basta modificar o comando para stringsAsFactors = **FALSE**. Assim, as variáveis qualitativas serão mantidas no formato **character**.

a) Função data.frame()

```
# Cria um data frame a partir de vetores existentes
especie <- c("Acapu", "Araucaria", "Mogno", "Cedro", "Ipe")
diametro <- c(23.0, 27.0, 33.6, 42.6, 52.1)
altura <- c(8.4, 8.7, 9.1, 13.2, 15.4)
cortar <- c("Não", "Não", "Não", "Não", "Sim")

invFlor.1 <- data.frame(especie, diametro, altura, cortar,
                        stringsAsFactors = TRUE)
print(invFlor.1)
```

especie	diametro	altura	cortar
Acapu	23.0	8.4	Não
Araucaria	27.0	8.7	Não
Mogno	33.6	9.1	Não
Cedro	42.6	13.2	Não
Ipe	52.1	15.4	Sim

```
# Cria um data frame com apenas um comando
invFlor.2 <- data.frame(
  especie = c("Acapu", "Araucaria", "Mogno", "Cedro", "Ipe"),
  diametro = c(23.0, 27.0, 33.6, 42.6, 52.1),
  altura = c(8.4, 8.7, 9.1, 13.2, 15.4),
  cortar = c("Não", "Não", "Não", "Não", "Sim"),
  stringsAsFactors = FALSE)

print(invFlor.2)
```

especie	diametro	altura	cortar
Acapu	23.0	8.4	Não
Araucaria	27.0	8.7	Não
Mogno	33.6	9.1	Não
Cedro	42.6	13.2	Não
Ipe	52.1	15.4	Sim

b) Função edit()

A função edit() pode ser usada para editar data frames já existentes, ou, ainda, para iniciar uma tabulação de dados diretamente no R. Assim, ao executar o comando edit(**data.frame()**) será aberto um **R Data Editor** "vazio" que permite a organização e tabulação de dados de modo similar ao Microsoft Office Excel. No entanto, se o data frame já existe, você terá a opção de editá-lo com o comando edit(**nome_do_df**), de tal modo que o data frame será aberto no **R Data Editor**, admitindo a alteração ou correção de quaisquer informações que desejar.

```
# Crie um data frame com o comando edit() e atribua a "invFlor.3"
invFlor.3 <- edit(data.frame())

# Edite as informações do data frame "invFlor.2" e atribua a "invFlor.4"
invFlor.4 <- edit(invFlor.2)
```

4.4 Listas

Uma **lista** é um objeto constituído de uma coleção ordenada de objetos conhecidos. Um aspecto importante da **lista** é que essa admite diferentes estruturas (vetores, matrizes, data frames e inclusive outras listas). A função list() é usada para criar uma lista. Se os objetos já existem pode-se simplesmente adicioná-los à função list().

Diferentemente do **data frame**, as **listas** admitem vetores de comprimentos distintos. O número de componentes de uma lista (nível superior) pode ser obtido usando a função length(**nome_do_objeto**).

Os componentes de uma lista podem ser nomeados. Fazendo-se isso, o acesso aos seus níveis superiores pode ser feito com uso de $ (dólar) ao invés de [[ ]] (colchetes duplos). Nomear os níveis superiores da lista é bastante útil, pois

facilita a identificação do componente a ser acessado por indexação (estudaremos os mecanismos de indexação de objetos mais adiante).

Função list()

Pode-se criar uma lista a partir de **objetos já existentes**:

```
# Cria objetos-vetores
especie <- c("Acapu", "Araucaria", "Mogno", "Cedro", "Ipe")
diametro <- c(23.0, 27.0, 33.6, 42.6, 52.1)
altura <- c(8.4, 8.7, 9.1, 13.2, 15.4)

# lista sem atribuir nomes aos os objetos
list.1 <- list(especie, diametro, altura)                    # cria uma lista
print(list.1)                                                # imprime a lista
## [[1]]
## [1] "Acapu"     "Araucaria" "Mogno"     "Cedro"     "Ipe"
##
## [[2]]
## [1] 23.0 27.0 33.6 42.6 52.1
##
## [[3]]
## [1]  8.4  8.7  9.1 13.2 15.4

length(list.1)                                               # comprimento da lista
## [1] 3
```

Pode-se criar uma lista com **diferentes classes de objetos**. A seguir, uma lista com três componentes: **um vetor**, **uma matriz** e **um data frame**:

```
# Uma lista com diferentes classes de objetos
list.2 <- list(diametro, mat.7, invFlor.1)                    # cria uma lista
print(list.2)                                                 # imprime a lista
## [[1]]
## [1] 23.0 27.0 33.6 42.6 52.1
##
## [[2]]
##       C1  C2  C3
## L1    1   2   3
## L2    4   5   6
##
## [[3]]
##       especie      diametro   altura     cortar
## 1     Acapu          23.0       8.4       Não
## 2     Araucaria      27.0       8.7       Não
## 3     Mogno          33.6       9.1       Não
## 4     Cedro          42.6       13.2      Não
## 5     Ipe            52.1       15.4      Sim

length(list.2)                                                # comprimento da lista
## [1] 3

str(list.2)                                                   # estrutura da lista
## List of 3
##  $ : num [1:5] 23 27 33.6 42.6 52.1
##  $ : int [1:2, 1:3] 1 4 2 5 3 6
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:2] "L1" "L2"
##   .. ..$ : chr [1:3] "C1" "C2" "C3"
##  $ :'data.frame':    5 obs. of  4 variables:
##   ..$ especie : Factor w/ 5 levels "Acapu","Araucaria",..: 1 2 5 3 4
##   ..$ diametro: num [1:5] 23 27 33.6 42.6 52.1
##   ..$ altura  : num [1:5] 8.4 8.7 9.1 13.2 15.4
##   ..$ cortar  : Factor w/ 2 levels "Não","Sim": 1 1 1 1 2
```

Os componentes de uma lista podem ser nomeados. Fazendo isso, a identificação e a indexação dos componentes são facilitadas.

```
# Uma lista de objetos-vetores (nomeados)
list.2 <- list(vetor = diametro, matriz = mat.7, DF = invFlor.1)        # cria uma lista
print(list.2)                                                           # imprime a lista
## $vetor
## [1] 23.0 27.0 33.6 42.6 52.1
##
## $matriz
##    C1 C2 C3
## L1  1  2  3
## L2  4  5  6
##
## $DF
##     especie diametro altura cortar
## 1     Acapu     23.0    8.4    Não
## 2 Araucaria     27.0    8.7    Não
## 3     Mogno     33.6    9.1    Não
## 4     Cedro     42.6   13.2    Não
## 5       Ipe     52.1   15.4    Sim

length(list.2)                                                          # comprimento da lista
## [1] 3

list.3 <- list(Especie = especie, Diametro = diametro, Altura = altura)   # cria uma lista
print(list.3)                                                           # imprime a lista
## $Especie
## [1] "Acapu"     "Araucaria" "Mogno"     "Cedro"     "Ipe"
##
## $Diametro
## [1] 23.0 27.0 33.6 42.6 52.1
##
## $Altura
## [1]  8.4  8.7  9.1 13.2 15.4

length(list.3)                                                          # comprimento da lista
## [1] 3
```

Concatenando listas

Quando a intenção for unir diferentes **listas** em um único objeto pode-se usar a função c(). O novo objeto criado também será uma **lista**, cujos componentes serão os mesmos dos objetos-listas concatenados dispostos na ordem dos argumentos estabelecidos.

```
list.4 <- list(Especie = especie, Diametro = diametro)                    # cria uma lista
list.5 <- list(Altura = altura, Cortar = cortar)                          # cria uma lista
list.6 <- c(list.4, list.5)                                               # cria uma lista de lista

print(list.6)                                                             # imprime a lista de lista
## $Especie
## [1] "Acapu"     "Araucaria" "Mogno"     "Cedro"     "Ipe"
##
## $Diametro
## [1] 23.0 27.0 33.6 42.6 52.1
##
## $Altura
## [1]  8.4  8.7  9.1 13.2 15.4
##
## $Cortar
## [1] "Não" "Não" "Não" "Não" "Sim"
```

Um objeto-lista pode ser coagido (transformado) em um data frame usando a função as.data.frame(), desde que as restrições de criação de data frames sejam satisfeitas. Isto é, a lista a ser coagida deve possuir vetores de igual comprimento.

```
as.data.frame(list.6)
```

Especie	Diametro	Altura	Cortar
Acapu	23.000	8.400	Não
Araucaria	27.000	8.700	Não
Mogno	33.600	9.100	Não
Cedro	42.600	13.200	Não
Ipe	52.100	15.400	Sim

5 Indexação no R

Quando o interesse é **extrair**, **excluir** ou **substituir** elementos de objetos é possível fazê-lo por meio de algum **mecanismo de indexação**, que dependerá do tipo de objeto manejado. Para isso, utilizam-se operadores básicos para localizar a posição do elemento. De modo geral, os seguintes elementos de um objeto do R podem ser indexados: a) nomes; b) valores lógicos; c) inteiros positivos; d) inteiros negativos; e) espaço em branco; e f) zero.

Para indexar elementos ou subconjuntos de objetos no R existem três operadores básicos: [ ], [[ ]] e $.

1. O operador [ ] permite extrair múltiplos elementos de um objeto, retornando um novo objeto de mesma classe.
2. O operador [[ ]] permite extrair elementos de objetos do tipo lista ou data frames. A classe do objeto extraído não será, necessariamente, uma lista ou data frame.

3. O operador $ permite extrair componentes nomeados de uma lista ou data frame.

5.1 Indexação de vetores

Para extrair, excluir ou substituir elementos no objeto-vetor usa-se o comando [*i*]. O índice *i* indica a posição do elemento no objeto e inicia-se no valor 1. A função c() pode ser usada para concatenar as posições desejadas dentro de colchetes.

1. **Extração por indexação positiva**: especifica (entre colchetes) os elementos a serem extraídos.

```
# cria os objetos-vetores
especie <- c("Acapu", "Araucaria", "Mogno", "Cedro", "Ipe")
diametro <- c(23.0, 27.0, 33.6, 42.6, 52.1)

# Extraindo elementos (usando valores inteiros)
especie[2]                                  # Um elemento.
## [1] "Araucaria"

diametro[1:3]                               # Múltiplos elementos (sequenciais).
## [1] 23.0 27.0 33.6

especie[c(1,3,5)]                           # Múltiplos elementos (alternados).
## [1] "Acapu" "Mogno" "Ipe"

diametro[seq(from = 1, to = 5, by = 2)]     # Múltiplos elementos usando seq ().
## [1] 23.0 33.6 52.1
```

2. **Extração por indexação negativa**: especifica (entre colchetes) os elementos a serem excluídos, retornando os demais. Deve-se usar o sinal negativo (-) para um ou mais elementos do objeto-vetor que se deseja excluir [*-i*].

```
# Excluindo elementos (usando valores inteiros)
especie[-2]                                 # Um elemento.
## [1] "Acapu" "Mogno" "Cedro" "Ipe"

diametro[-(1:3)]                            # Múltiplos elementos (sequenciais).
## [1] 42.6 52.1

especie[-c(1,3,5)]                          # Múltiplos elementos (alternados).
## [1] "Araucaria" "Cedro"
```

3. **Extração por indexação de strings** (vetores nomeados): este mecanismo apenas é possível quando o objeto-vetor possui um atributo de nomes para identificar seus elementos. Pode-se usar a função names() para atribuir nomes (rótulos) para cada elemento do objeto-vetor. A vantagem dessa função consiste na facilidade de identificação dos elementos do objeto-vetor.

```
altura <- c(1.52, 1.83, 1.74, 1.67, 1.98)
names(altura)
## NULL

# Atribuindo nomes (labels) aos elementos do objeto-vetor
names(altura) <- c("João", "Maria", "José", "Pedro", "Tomé")
names(altura)
## [1] "João"  "Maria" "José"  "Pedro" "Tomé"

# Extraindo elementos (usando nomes)
altura[c("Maria", "José")]    # Altura de pessoas específicas?
## Maria  José
##  1.83  1.74
```

4. **Extração por indexação lógica**: a extração de elementos no objeto-vetor pode ser feita com uso de **operadores lógicos**.

```
Especie <- c(NA, "Araucaria", "Mogno", "Cedro", "Ipe",
          "Tauari", "Jatoba", "Araucaria", "Acapu", NA)
Diametro <- c(23.0, 27.0, 33.6, 42.6, 52.1,
           65.8, 79.6, 45.2, 50.0, 89.8)

# Extrai árvores que não sejam Araucaria
Especie[Especie != "Araucaria"]
## [1] NA      "Mogno"  "Cedro"  "Ipe"    "Tauari" "Jatoba" "Acapu"  NA

# Diâmetros > 50cm.
Diametro[Diametro >= 50]
## [1] 52.1 65.8 79.6 50.0 89.8
```

5. **Substituição por indexação**: algumas vezes deseja-se substituir um ou mais elementos do objeto-vetor por outro.

```
Especie <- c(NA, "Araucaria", "Mogno", "Cedro", "Ipe", NA)
Diametro <- c(23.0, 27.0, 33.6, 42.6, 52.1)

is.na(Especie)                                    # verificar se existe valor faltante
## [1]  TRUE FALSE FALSE FALSE FALSE  TRUE

Especie[is.na(Especie)] <- "Não Identificada"     # Substitui os NAs
print(Especie)
## [1] "Não Identificada" "Araucaria"       "Mogno"
## [4] "Cedro"            "Ipe"             "Não Identificada"

# Altera o diâmetro da posição 3, e atribui 33,5cm.
Diametro[3] <- 33.5

# Altera os diâmetros das posições 4 e 5, e atribui 55,3cm e 63,4cm
Diametro[c(4, 5)] <- c(55.3, 63.4)
```

Comandos especiais para vetores

Função which(): Qual?

which(x = ?, arr.ind = FALSE, useNames = TRUE)

A função which() recebe e avalia **expressões lógicas** (objeto-vetor lógico) e fornece a posição dos elementos nos quais a expressão lógica é verdadeira (TRUE). Os **NAs** são admitidos pela função, porém são tratados como FALSE.

```
alfa <- LETTERS
Especie <- c(NA, "Araucaria", "Mogno", "Cedro", "Ipe",
         "Tauari", "Jatoba", "Araucaria", "Acapu", NA)
Diametro <- c(23.0, 27.0, 33.6, 42.6, 52.1,
          65.8, 79.6, 45.2, 50.0, 89.8)

which(alfa == "H")                                # posição da letra "H" no alfabeto
## [1] 8

which(Especie == "Mogno")                         # posição do "Mogno"
## [1] 3

which(Diametro >= 50 | Diametro < 25)             # posição de árvores com 50 <= d < 25
## [1]  1  5  6  7  9 10
```

Função which.min(): Qual a posição do valor mínimo?
Função which.max(): Qual a posição do valor máximo?
Função unique(): Recebe um objeto-vetor e, retorna um vetor sem repetições.

```
which.max(Diametro)                    # qual a posição da árvore de maior diâmetro?
## [1] 10

which.min(Diametro)                    # qual a posição da árvore de menor diâmetro?
## [1] 1

unique(Especie)                        # gere um vetor sem repetições do objeto 'Especie'.
## [1] NA         "Araucaria" "Mogno"    "Cedro"    "Ipe"      "Tauari"
## [7] "Jatoba"   "Acapu"
```

Comando %in%

É um operador binário que retorna um vetor booleano (lógico) de tamanho sempre igual ao vetor-esquerdo, com indicativo das correspondências verdadeiras (TRUE). No exemplo (Esp.1 %in% c("Mogno", "Cedro")) a pergunta feita é: As espécies "Mogno" e "Cedro" estão contidas no vetor 'Esp.1'? O comando %in% faz com que cada elemento do vetor 'Esp.1' seja avaliado e retorna um vetor de respostas booleanas de comprimento igual à de 'Esp.1'.

```
# cria os vetores
Esp.1 <- c("Acapu", "Araucaria", "Mogno", "Cedro", "Tauari")
Esp.2 <- c("Angelim", "Araucaria", "Tauari")
Diametro <- c(23.0, 27.0, 33.6, 42.6, 52.1)

# vetor direito está contido no esquerdo?
Esp.1 %in% c("Mogno", "Cedro")
## [1] FALSE FALSE  TRUE  TRUE FALSE
Esp.1 %in% Esp.2
## [1] FALSE  TRUE FALSE FALSE  TRUE
Esp.2 %in% Esp.1
## [1] FALSE  TRUE  TRUE
```

5.2 Indexação de matrizes

Para extrair, excluir ou substituir elementos de uma matriz usa-se o comando [*i*, *j*]. O índice *i* indica as linhas e o índice *j* indica as colunas da matriz.

- Se o argumento for do tipo [*i*,] ter-se-á acesso a todos os elementos da linha *i* especificada.
- Se o argumento for do tipo [, *j*] ter-se-á acesso a todos os elementos da coluna *j* especificada.
- Se nem o número da linha e nem o número da coluna é informado [,]: a matriz é acessada por completa.

Por exemplo, em uma matriz **3x3** o elemento situado na 1ª linha da 1ª coluna representado pela letra *x*, pode ser acessado fazendo: *x*[1, 1].

Matriz 3x3

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}$$

1. **Extraindo elementos**: usando indexação positiva.

```
# Cria uma matriz
mat <- matrix(1:6, nrow=2, ncol=3, byrow = TRUE,
          dimnames = list(c("L1", "L2"),
                          c("C1", "C2", "C3")))

mat[2, 2]                                   # extrai o elemento da linha 2 e coluna 2.
## [1] 5

mat[2, ]                                    # extrai todos elementos da linha 2.
## C1 C2 C3
##  4  5  6

mat[, 3]                                    # extrai todos elementos da coluna 3.
## L1 L2
##  3  6

mat[c(1, 2),  c(2, 3)]                      # extrai os elementos de L1 e L2, C2 e C3.
##    C2 C3
## L1  2  3
## L2  5  6
```

2. **Excluindo elementos**: usando indexação negativa.

```
mat <- matrix(1:6, nrow=2, ncol=3, byrow = TRUE,
          dimnames = list(c("L1", "L2"),
                          c("C1", "C2", "C3")))
mat[, c(-1, -3)]                             # exclui as colunas 1 e 3.
## L1 L2
##  2  5

mat[-2, -1]                                  # exclui a linha 2 e a coluna 1.
## C2 C3
##  2  3
```

3. Substituindo elementos

```
mat <- matrix(1:6, nrow=2, ncol=3, byrow = TRUE,
              dimnames = list(c("L1", "L2"),
                              c("C1", "C2", "C3")))
print(mat)
##    C1 C2 C3
## L1  1  2  3
## L2  4  5  6

# substitui o elemento da posição 1 por zero.
mat[1, 1] <- 0; print(mat)
##    C1 C2 C3
## L1  0  2  3
## L2  4  5  6
# substitui os elementos das posições 1 e 5 por zero.
# Obs.: observe a forma de contagem das posições!
mat[c(1,5)] <- c(0,0)
print(mat)
##    C1 C2 C3
## L1  0  2  0
## L2  4  5  6
```

5.3 Indexação de data frames

O acesso a um determinado vetor em um data frame pode ser realizado utilizando-se do comando [ ] (similar às matrizes) ou dos comandos [[ ]] e $ (similar às listas). As funções attach() (*attachment*) e with() podem ser usadas para facilitar o acesso a esses vetores.

```
invFlor.2 <- data.frame(
  especie = c("Acapu", "Araucaria", "Mogno", "Cedro", "Ipe"),
  diametro = c(23.0, 27.0, 33.6, 42.6, 52.1),
  altura = c(8.4, 8.7, 9.1, 13.2, 15.4),
  cortar = c("Não", "Não", "Não", "Não", "Sim"),
  stringsAsFactors = TRUE)

print(invFlor.2)
```

especie	diametro	altura	cortar
Acapu	23.000	8.400	Não
Araucaria	27.000	8.700	Não
Mogno	33.600	9.100	Não
Cedro	42.600	13.200	Não
Ipe	52.100	15.400	Sim

Comandos [ ] e [[ ]]

```
invFlor.2[2, 1]                                    # similar às matrizes
## [1] Araucaria
## Levels: Acapu Araucaria Cedro Ipe Mogno

invFlor.2[[2, 1]]                                  # similar às listas
## [1] Araucaria
## Levels: Acapu Araucaria Cedro Ipe Mogno

invFlor.2[, c(1, 2, 4), drop=FALSE]                # drop = FALSE
```

especie	diametro	cortar
Acapu	23.000	Não
Araucaria	27.000	Não
Mogno	33.600	Não
Cedro	42.600	Não
Ipe	52.100	Sim

Comando $

```
invFlor.2$diametro                 # acessa a coluna "diâmetro"
## [1] 23.0 27.0 33.6 42.6 52.1

invFlor.2$cortar                   # acessa a coluna "cortar"
## [1] Não Não Não Não Sim
## Levels: Não Sim

invFlor.2$cortar[c(4,5)]           # acessa a coluna "cortar" e extrai os elementos da posição [4, 5]
## [1] Não Sim
## Levels: Não Sim
```

Usando operadores lógicos

Os operadores lógicos podem ser usados para extrair informações específicas do data frame. Primeiramente, usaremos os operadores lógicos para obter retornos booleanos (TRUE ou FALSE):

```
# Quais valores de altura são maiores do que 10 m?
invFlor.2$altura > 10
## [1] FALSE FALSE FALSE  TRUE  TRUE

# Quais valores de diâmetro são maiores ou iguais à 50cm?
invFlor.2$diametro >= 50
## [1] FALSE FALSE FALSE FALSE  TRUE

# Quais árvores possuem "Sim" para "Corte"?
invFlor.2$cortar == "Sim"
## [1] FALSE FALSE FALSE FALSE  TRUE
```

No entanto, na maioria das vezes deseja-se extrair os elementos:

```
# Quais valores de altura são maiores do que 10m?
invFlor.2[invFlor.2$altura > 10, ]
##   especie diametro altura cortar
## 4   Cedro     42.6   13.2    Não
## 5     Ipe     52.1   15.4    Sim

# Quais valores de diâmetro são maiores ou iguais a 50cm?
invFlor.2[invFlor.2$diametro >= 50,]
##   especie diametro altura cortar
## 5     Ipe     52.1   15.4    Sim

# Quais árvores possuem "Sim" para "Corte"?
invFlor.2[invFlor.2$cortar == "Sim",]
##   especie diametro altura cortar
## 5     Ipe     52.1   15.4    Sim

# Extrai árvores com diâmetros < 50cm e não disponíveis p/ corte
invFlor.2[invFlor.2$diametro < 50 & invFlor.2$cortar == "Não", ]
```

especie	diametro	altura	cortar
Acapu	23.000	8.400	Não
Araucaria	27.000	8.700	Não
Mogno	33.600	9.100	Não
Cedro	42.600	13.200	Não

Adicionar linhas e colunas ao data frame

```
# Adicionando a coluna "Protegida"
protegida <- c("Sim", "Sim", "Sim", "Não", "Não")
invFlor.2$protegida <- protegida
print(invFlor.2)
```

especie	diametro	altura	cortar	protegida
Acapu	23.000	8.400	Não	Sim
Araucaria	27.000	8.700	Não	Sim
Mogno	33.600	9.100	Não	Sim
Cedro	42.600	13.200	Não	Não
Ipe	52.100	15.400	Sim	Não

Para adicionar linhas a um data frame há uma restrição: **não é possível adicionar novos níveis de fator para vetores da classe factor**. Para exemplificar, tentar-se-á inserir o seguinte vetor-linha: ("Castanha", 150.9, 25.6,"Não","Sim") na linha 1. No entanto, antes estudaremos alguns pontos:

I. A coluna "espécie" pertence à classe **factor** e, originalmente, **não possui** o nível de fator "Castanha"; e

II. A coluna "cortar" pertence à classe **factor** e, originalmente, **já possui** o nível de fator "Não".

Quando adicionamos um novo vetor-linha, o que acontece?

Surgirá uma **mensagem de erro** atestando que o nível de fator "Castanha" é inválido. Isso ocorre porque a espécie "Castanha" não existe no data frame original, sendo então considerada um novo nível de fator. Assim, um **NA** será impresso no lugar do nome da "Castanha". Idealmente, para evitar problemas deste tipo, pode-se manter todas as colunas qualitativas como **character**. Para tanto, deve-se usar o argumento stringsAsFactors = FALSE.

```
str(invFlor.2)
## 'data.frame':    5 obs. of  5 variables:
##  $ especie  : Factor w/ 5 levels "Acapu","Araucaria",..: 1 2 5 3 4
##  $ diametro : num  23 27 33.6 42.6 52.1
##  $ altura   : num  8.4 8.7 9.1 13.2 15.4
##  $ cortar   : Factor w/ 2 levels "Não","Sim": 1 1 1 1 2
##  $ protegida: chr  "Sim" "Sim" "Sim" "Não" ...

invFlor.2[1,] <- c("Castanha", 150.9, 25.6, "Não", "Sim")
## Warning in `[<-.factor`(`*tmp*`, iseq, value = "Castanha"): invalid factor
## level, NA generated
print(invFlor.2)
```

especie	diametro	altura	cortar	protegida
NA	150.9	25.6	Não	Sim
Araucaria	27	8.7	Não	Sim
Mogno	33.6	9.1	Não	Sim
Cedro	42.6	13.2	Não	Não
Ipe	52.1	15.4	Sim	Não

```
invFlor.2$especie <- as.character(invFlor.2$especie)
invFlor.2[1,] <- c("Castanha", 150.9, 25.6, "Não", "Sim")
print(invFlor.2)
```

especie	diametro	altura	cortar	protegida
Castanha	150.9	25.6	Não	Sim
Araucaria	27	8.7	Não	Sim
Mogno	33.6	9.1	Não	Sim
Cedro	42.6	13.2	Não	Não
Ipe	52.1	15.4	Sim	Não

Função attach()

A função attach() oferece acesso direto aos vetores do data frame. Para finalizar seu uso deve-se utilizar a função detach().

```
attach(invFlor.2)                    # inicia a função attach
diametro                             # acessa o vetor "diâmetro" diretamente
## [1] 23.0 27.0 33.6 42.6 52.1

cortar                               # acessa o vetor "cortar" diretamente
## [1] "Não" "Não" "Não" "Não" "Sim"
detach(invFlor.2)                    # finaliza a função attach
```

Função with()

```
 # acessa a coluna "espécie"
with(invFlor.2, especie)
## [1] "Castanha"  "Araucaria" "Mogno"     "Cedro"     "Ipe"

# acessa elementos específicos na coluna
with(invFlor.2, especie[4:5])
## [1] "Cedro" "Ipe"
```

5.3.1 Função subset()

A função subset() pode ser usada para extrair um subconjunto de vetores, matrizes e data frames que atendem às condições especificadas. Ainda, pode-se utilizar de operadores lógicos no processo de obtenção dos subconjuntos. Para

estudar a função, usar-se-á o famoso conjunto de dados de **flores de iris** (*iris data set*). O conjunto de dados é composto de 150 observações (instâncias) e 5 variáveis (Sepal.Length, Sepal.Width, Petal.Length e Petal.Width, Species), medidas de 50 flores pertencentes a três espécies de íris (*Iris setosa*, *Iris versicolor* e *Iris virginica*).

```
data("iris")                                    # chama o conjunto iris
str(iris)                                       # inspeciona a estrutura do conjunto
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
dim(iris)                                       # verifica a dimensão do conjunto
## [1] 150   5
```

```
head(iris)                                      # imprime as 6 primeira linhas do conjunto
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.100	3.500	1.400	0.200	setosa
4.900	3.000	1.400	0.200	setosa
4.700	3.200	1.300	0.200	setosa
4.600	3.100	1.500	0.200	setosa
5.000	3.600	1.400	0.200	setosa
5.400	3.900	1.700	0.400	setosa

```
tail(iris)                                      # imprime as 6 últimas linhas do conjunto
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
6.700	3.300	5.700	2.500	virginica
6.700	3.000	5.200	2.300	virginica
6.300	2.500	5.000	1.900	virginica
6.500	3.000	5.200	2.000	virginica
6.200	3.400	5.400	2.300	virginica
5.900	3.000	5.100	1.800	virginica

```
# flores com Sepal.Length > 7,5, mantendo as colunas Species e Sepal.Length
subset(iris, Sepal.Length > 7.5, select = c(Species, Sepal.Length))
```

Species	Sepal.Length
virginica	7.600
virginica	7.700
virginica	7.700

Species	Sepal.Length
virginica	7.700
virginica	7.900
virginica	7.700

```
# flores de setosa com Sepal.Width < 3, mantendo as colunas Species e Sepal.Width.
subset(iris, Species == "setosa" & Sepal.Width < 3,
    select = c(Species, Sepal.Width)))
```

Species	Sepal.Width
setosa	2.900
setosa	2.300

5.3.2 Função split()

A divisão de data frames pode ser realizada por meio da função split(). Para tanto, especifique dois argumentos básicos: a) data frame que se almeja dividir; b) fator a ser considerado para a divisão.

```
protegida <- split(invFlor.2, invFlor.2$protegida)

print(protegida)
## $Não
##     especie  diametro   altura   cortar   protegida
## 4   Cedro      42.6       13.2     Não        Não
## 5   Ipe        52.1       15.4     Sim        Não
##
## $Sim
##       especie    diametro    altura   cortar   protegida
## 1   Castanha      150.9       25.6     Não        Sim
## 2   Araucaria      27         8.7      Não        Sim
## 3   Mogno         33.6        9.1      Não        Sim
```

5.4 Indexação de listas

A indexação de lista pode ser feita com uso dos comandos [[ ]] e $. Ainda, para acessar subíndices dos componentes da lista pode-se fazer: [[ ]][ ]. Para exemplificar, considere a lista “list.2” que contém 3 componentes (vetor, matriz e DF):

```
list.2 <- list(diametro, mat.8, invFlor.1)          # uma lista cujos componentes não estão nomeados

list.2[[1]]                                         # extrai o componente da posição 1 (diâmetro)
## [1] 23.0 27.0 33.6 42.6 52.1

list.2[[2]][1,]                                     # extrai a linha 1 do componente 2 (matriz)
## C1 C2 C3
##  1  2  3



# extrai as árvores com d > 50 e cortar ==Sim do componente 3 (data frame)
list.2[[3]][diametro > 50 & cortar == "Sim", ]

##      especie    diametro   altura    cortar
## 5      Ipe          52.1       15.4       Sim
```

Se a lista tiver seus componentes nomeados o acesso é facilitado pelo uso do comando $.

```
# uma lista cujos componentes estão nomeados
list.2 <- list(diametro=diametro, matriz=mat.8, DF=invFlor.1)

print(list.2)
## $diametro
## [1] 23.0 27.0 33.6 42.6 52.1
##
## $matriz
##      C1 C2 C3
## L1   1   2   3
## L2   4   5   6
##
## $DF
##       especie      diametro    altura     cortar
## 1      Acapu          23.0        8.4        Não
## 2      Araucaria      27.0        8.7        Não
## 3      Mogno          33.6        9.1        Não
## 4      Cedro          42.6        13.2       Não
## 5      Ipe            52.1        15.4       Sim

list.2$diametro
## [1] 23.0 27.0 33.6 42.6 52.1

list.2$matriz[1,]
## C1 C2 C3
##  1   2   3

list.2$DF[diametro > 50 & cortar == "Sim", ]
##   especie diametro altura cortar
## 5     Ipe     52.1   15.4    Sim
```

Substituindo elementos de componentes da lista:

```
# substitui os elementos das posições 3 e 5 por 50 e 60 cm de diâmetro
list.2$diametro[c(3,5)] <- c(50, 60)

print(list.2)
## $diametro
## [1] 23.0 27.0 50.0 42.6 60.0
##
## $matriz
##      C1 C2 C3
## L1   1    2   3
## L2   4    5   6
##
## $DF
##       especie      diametro    altura     cortar
## 1     Acapu          23.0        8.4       Não
## 2     Araucaria      27.0        8.7       Não
## 3     Mogno          33.6        9.1       Não
## 4     Cedro          42.6        13.2      Não
## 5     Ipe            52.1        15.4      Sim
```

Excluindo componentes da lista:

```
list.2$DF <- NULL                          # excluindo todo os componentes do DF
print(list.2)                              # imprime a lista resultante
## $diametro
## [1] 23.0 27.0 50.0 42.6 60.0
##
## $matriz
##    C1 C2 C3
## L1  1  2  3
## L2  4  5  6

list.2$matriz <- list.2$matriz[,-c(2,3)]   # excluindo as colunas 2 e 3 do componente matriz
print(list.2)                              # imprime a lista resultante
## $diametro
## [1] 23.0 27.0 50.0 42.6 60.0
##
## $matriz
## L1 L2
##  1  4
```

6. Criando funções no R

O ambiente R possui muitas funções **interativas** e **prontas** disponíveis em inúmeros pacotes, para o uso fácil e prático pelos usuários. Não obstante, em algumas situações pode-se desejar criar a própria função. Para tanto, deve-se utilizar a função function(){} disponível no R-base (R CORE TEAM, 2017). Entre parênteses devem ser inseridos o(s) argumento(s) da função, sobre os quais irá

trabalhar o **código** descrito entre chaves {}. Opcionalmente, no escopo da função pode-se usar a função return(), e especificar entre os parênteses a(s) saída(s) desejada(s) ao se aplicar a função.

Função function(){}

function(arglist = arg_1, arg_2, …, arg_n){ expressão return(valor) }

arglist = um argumento ou lista de argumentos sobre o(s) qual(is) irá atuar o código em **expressão**;
expressão = uma expressão ou código a ser usado sobre os argumentos da função;
return = especifica o retorno da função; e
valor = uma expressão que representa a saída desejada.

1. Criando uma função para obter a média aritmética de um vetor **x** qualquer:

```
Media <- function(x){
  n = length(x)
  Soma=sum(x)
  Media=Soma/n

  return(Media)
}
```

Dado um objeto-vetor de diâmetros de árvores, pode-se calcular a média utilizando a função criada:

```
diametro <- c(23.4, 54.3, 45.1, 67.1, 34.7)
Media(diametro)
## [1] 44.92
```

2. Criando uma função para obter o coeficiente de variação de um vetor **x** qualquer:

```
CV <- function(x){
  Media = sum(x)/length(x)
  DP = sqrt(sum((x-mean(x))^2)/(length(x)-1))
  CV = (DP/Media)*100

  return(CV)
}
```

Para o mesmo vetor de diâmetros de árvores pode-se calcular o CV:

```
CV(diametro)
## [1] 37.70612
```

3. Criando uma função para obter várias estatísticas descritivas:

```
descritivas.1 <-
 function(x){
 n <- length(x)
 Soma <- sum(x)
 Media <- round(sum(x)/n, digits=2)
 Variancia <- var(x)
 DP <- round(sd(x), digits=2)
 CV <- round((DP/Media)*100,digits=2)
 Minimo <- min(x)
 Quartil1 <- quantile(x,0.25)
 Mediana <- median(x)
 Quartil3 <- quantile(x,0.75)
 Interquatil <- Quartil3-Quartil1
 Maximo <- max(x)

 return(list(n=n, Media=Media, DP=DP, CV=CV, Min=Minimo,
         Q1=Quartil1, Md=Mediana, Q3=Quartil3,
         Interquatil=Interquatil, Max=Maximo))
}
```

Agora, usaremos a função 'descritivas.1' sobre o conjunto de dados '**trees**' disponível no pacote **datasets**. Este conjunto de dados fornece medições do diâmetro, altura e volume de madeira de 31 cerejeiras (*Prunus serotina*). Assim, pode-se aplicar, por exemplo, a função para obter as estatísticas descritivas para a variável **volume de madeira**. A função recebe o vetor de **volume** no argumento **x** e retorna uma lista com as estatísticas descritivas. Se você desejar obter as estatísticas para diâmetro e altura, deverá realizar o procedimento para cada variável.

```
data("trees")                                  # chama o conjunto 'trees'
descritivas.1(trees$Volume)
## $n
## [1] 31
##
## $Media
## [1] 30.17
##
## $DP
## [1] 16.44
##
## $CV
## [1] 54.49
##
## $Min
## [1] 10.2
##
## $Q1
##  25%
## 19.4
```

```
##
## $Md
## [1] 24.2
##
## $Q3
##  75%
## 37.3
##
## $Interquatil
##  75%
## 17.9
##
## $Max
## [1] 77
```

Mas, seria possível criar uma função para retornar várias estatísticas para diversas variáveis simultaneamente? A seguir é apresentada uma função que faz exatamente isso. Neste caso, iremos usar uma função própria no argumento **FUN** da função apply(). Além disso, o argumento **MARGIN** será definido com inteiro igual a 2, para que a função em **FUN** seja aplicada às colunas do data frame.

```
descritivas.2 <-
 apply(X = trees, MARGIN = 2, FUN =
       function(x){
         n <- length(x)
         Soma <- sum(x)
         Media <- round(sum(x)/n, digits=2)
         Variancia <- var(x)
         DP <- round(sd(x), digits=2)
         CV <- round((DP/Media)*100,digits=2)
         Minimo <- min(x)
         Quartil1 <- quantile(x,0.25)
         Mediana <- median(x)
         Quartil3 <- quantile(x,0.75)
         Interquatil <- Quartil3-Quartil1
         Maximo <- max(x)

         return(list(n=n, Media=Media, DP=DP, CV=CV, Min=Minimo,
                Q1=Quartil1, Md=Mediana, Q3=Quartil3,
                Interquatil=Interquatil, Max=Maximo))
})
```

Pode-se melhorar a saída das estatísticas utilizando-se a função call():

```
do.call(rbind, descritivas.2)
##                n      Media     DP     CV    Min     Q1    Md     Q3    Interquatil    Max
## Girth         31      13.25   3.14   23.7    8.3  11.05  12.9  15.25        4.2        20.6
## Height        31       76     6.37   8.38   63     72     76    80           8          87
## Volume        31      30.17  16.44  54.49   10.2   19.4   24.2  37.3        17.9        77
```

7 Estruturas de controle

7.1 Instrução condicional

De modo geral, pode-se identificar três principais variações de instruções condicionais: **if**, **if-else** e **if-elseif-else**. A estrutura if(){} condiciona a execução de uma lista de instruções (código) em função de uma condição booleana (verdadeira ou falsa). Assim, a aplicação da condicional if(){} requer duas argumentações básicas:

a) A condição a ser satisfeita, descrita entre parênteses; e
b) Uma ou mais instrução(ões), expressas entre chaves {}, que deverão ser executadas caso a condição booleana seja verdadeira. Para mais informações faça **?"if"**.

Alternativamente, a condição if(){}pode ser usada em conjunto com else{}. Neste caso, tem-se a seguinte estrutura: if(){} else{}. Assim, a condição será examinada a cada passagem pela estrutura if() e, caso seja satisfeita (VERDADEIRA) a lista de instruções entre chaves será executada. Do contrário, se for FALSA, será executado o código atribuído ao comando else{}.

A instrução if(){} elseif(){} else{} é outra variação da estrutura condicional if(), sendo usada quando se deseja executar mais de uma condicional através de instruções aninhadas. Deve-se atentar para o cuidado de saber a qual if() está atrelado cada else{}.

Como usar a instrução condicional?

Variação 1 - if(){}

```
if(condição booleana){

código que será executado, caso a condição booleana seja TRUE.
}
```

Variação 2 - if(){} else{}

```
if(condição booleana){
código que será executado, caso a condição booleana seja TRUE.
}
else{
código que será executado, caso a condição booleana seja FALSE.
}
```

Variação 3 - if(){} elseif(){} else{}

```
if(condição booleana 1){
código que será executado, caso a condição booleana 1 seja TRUE.
}
else if(condição booleana 2){
código que será executado, caso a condição booleana 2 seja TRUE.
}
else{
código que será executado, caso as condições 1 a n sejam FALSE.
}
```

Uso da instrução if(){}

```
x <- 9                                  # um escalar  "x" = 9
if (x < 10){                            # Se =  "x" < 10
  print(x)                              # Imprima = "x"
  print("É menor do que 10")            # Faça = imprima "É menor do que 10"
}                                       # fim Se
## [1] 9
## [1] "É menor do que 10"
```

Uso da instrução if(){} else{}

```
x <- 11                                 # um escalar  "x" = 11
if (x <= 10){                           # Se =  "x" <= 10
  cat("É menor ou igual a 10")          # Imprima = "É menor ou igual a 10"
}else{                                  # do contrário
  cat("É maior do que 10")              # Imprima = "É maior do que 10"
}                                       # fim Se
## É maior do que 10
```

Uso da instrução if(){} elseif(){} else{} - ifs aninhados

```
x <- 9                                  # um escalar  "x" = 9
if (x == 10){                           # Se =  "x" == 10
  print("É igual a 10")                 # Imprima = "É igual a 10"
} else if (x > 10) {                    # Se =  "x" > 10
  print("É maior do que 10")            # Imprima = "É maior do que 10"
} else {                                # do contrário
  print("É menor do que 10")            # Imprima = "É menor do que 10"
}                                       # fim Se
## [1] "É menor do que 10"
```

As instruções condicionais podem ser usadas dentro de funções:

```
dap1 <- 64
dap2 <- 43
```

```
categoria <- function (x)
if (x >= 50){
  print(paste("Cortar =", x))
} else {
  print(paste("Protegida =", x))
}

categoria(dap1)
## [1] "Cortar = 64"
categoria(dap2)
## [1] "Protegida = 43"
```

Função ifelse() – instrução condicional para vetores

Em outras situações, pode-se requerer usar alguma das 3 variações de instrução condicional a um vetor numérico. Porém, nesse caso, ocorrerá uma **mensagem de erro**. Para fins ilustrativos, a seguir criou-se um vetor **x** qualquer com seis elementos e, em seguida foi elaborada uma estrutura condicional com a seguinte condição booleana if(x == 10), que se verdadeira retorna "**É igual a 10**". De fato, o desejável seria que a condição booleana fosse avaliada para cada elemento do vetor **x**. Porém, isso não acontece. Ao executar o comando surge uma mensagem de erro informando que a condição tem comprimento maior do que 1 e somente o primeiro elemento do vetor **x** será usado. Ou seja, a condição booleana foi avaliada apenas para o número 10, cujo retorno é "É igual a 10".

```
x <- c(10, 9, 8, 10, 15, 16)                    # um vetor numérico
if (x == 10){                                   # Se =  "x" == 10
  print("É igual a 10")                         # Imprima = "É igual a 10"
}else{                                          # do contrário
  print("Não é igual a 10")                     # Imprima = "Não é igual a 10"
}                                               # fim Se
## Warning in if (x == 10) {: a condição tem comprimento > 1 e somente o
## primeiro elemento será usado
## [1] "É igual a 10"
```

Caso a intenção seja aplicar uma condicional a vetores de comprimento maior do que 1, a alternativa é usar a função vetorizada ifelse(). Assim, a condição booleana opera sobre cada elemento do vetor:

Função ifelse()

ifelse(**test** = ?, **yes** = ?, **no** = ?)

test = um objeto que pode ser coagido para o modo lógico;

yes = instrução a ser retornada quando a condição for VERDADEIRA; e

no = instrução a ser executada quando a condição for FALSA.

Retornando ao exemplo do vetor numérico **x** usaremos a função ifelse():

```
x <- c(10, 9, 8, 10, 15, 16)
ifelse(test=(x == 10), yes="É igual a 10", no="Não é igual a 10")
## [1] "É igual a 10"     "Não é igual a 10" "Não é igual a 10"
## [4] "É igual a 10"     "Não é igual a 10" "Não é igual a 10"
```

O exemplo a seguir simula uma situação de análise de um censo florestal para fins de licenciamento de Plano de Manejo Florestal Sustentável (PMFS). A ideia é identificar as árvores passíveis de exploração legal, considerando os seguintes critérios:

- O atendimento ao diâmetro mínimo de corte (DMC ≥ 50 cm) (BRASIL, 2009); e
- Se a espécie é protegida por lei.

Agora, ao invés de um simples vetor tem-se um data frame com quatro variáveis.

```
invFlor.3 <- data.frame(
  especie = c("Acapu", "Araucaria", "Mogno", "Cedro", "Ipe"),
  diametro = c(23.0, 27.0, 33.6, 42.6, 52.1),
  altura = c(8.4, 8.7, 9.1, 13.2, 15.4),
  protegida = c("Sim", "Sim", "Sim", "Não", "Não"),
  stringsAsFactors = FALSE)

# solução
ifelse(test=(invFlor.3$diametro >= 50 & invFlor.3$protegida == "Não"),
       yes="Pode Explorar", no="Não Pode Explorar")
## [1] "Não Pode Explorar" "Não Pode Explorar" "Não Pode Explorar"
## [4] "Não Pode Explorar" "Pode Explorar"
```

7.2 Instrução de repetição

Em algumas situações você pode se deparar com a necessidade de executar um mesmo código ou parte de um código repetidamente até que uma determinada

condição seja satisfeita. No entanto, reescrever um código várias vezes pode ser bastante trabalhoso e em certas circunstâncias impraticável.

Assim, pode-se usar de estruturas de controle de fluxo disponíveis no R-base. Tais estruturas possibilitam ao usuário executar repetidas vezes algum trecho do cógico ou, até mesmo, o código inteiro de um algoritmo, sob determinada(s) condição(ões). Para tanto, cria-se um **looping** (laços de repetição) para que as repetições sejam executadas tantas vezes quanto forem necessárias. Os principais comandos de controle de fluxo no R são: a) while (enquanto); b) for (para); e c) repeat (repita). Para saber mais sobre controle de fluxo no R, digite: **?Control**.

7.2.1 Comando while()

O comando while (enquanto) é uma estrutura de controle de fluxo no R que tem a seguinte estrutura: while(){}. O comando inicia com a inserção de uma condição booleana entre parênteses. Em seguida, o código é descrito entre chaves, o qual será executado repetidamente **ENQUANTO** a condição booleana for **TRUE**.

Comando while(){}

while(condição booleana){

Código que será executado repetidamente, ENQUANTO a condição booleana forVERDADEIRA

}

```
x <- 2                          # um escalar "x" (valor inicial das sucessivas iterações),
while(x < 10){                  # enquanto = "x" for < 10 (condição booleana),
  x <- x+2                      # faça = x+2 enquanto a condição x < 10 seja verdadeira,
  print(x)                      # imprima = reporte os resultados das iterações de x+2,
}                               # fim Enquanto
## [1] 4
## [1] 6
## [1] 8
## [1] 10
```

Entendendo as iterações:

Enquanto (x <10)			
Iteração	**Para cada "x"(...)**	**Avalia**	**Faça: x+2**
It1	x == 2,	x <10 = TRUE	faça: x <- 2+2 = 4
It2	x == 4,	x <10 = TRUE	faça: x <- 4+2 = 6
It3	x == 6,	x <10 = TRUE	faça: x <- 6+2 = 8
It4	x == 8,	x <10 = TRUE	faça: x <- 8+2 = 10
It5	x == 10,	x <10 = FALSE	...
fim Enquanto			

Pulando um laço do loop while(){} – usando o argumento **next** na condicional if(){}:

```
x <- 2                                  # um escalar "x" (valor inicial das sucessivas iterações),
while(x < 10){                          # enquanto = "x" for < 10 (condição booleana),
 x <- x+2                               # faça = x+2 enquanto a condição x < 10 seja verdadeira,
 if(x==8){next}                         # se, próximo = pule o laço se x == 8, ainda sim faça x+2,
 print(x)                               # imprima = reporte os resultados das iterações de x+2,
}                                       # fim Enquanto
## [1] 4
## [1] 6
## [1] 10
```

Entendendo as iterações:

Enquanto (x <10)				
Iteração	**Para cada "x"(...)**	**Avalia**	**Faça: x+2**	**if (x == 8){next}**
it1	x == 2,	x <10 = TRUE	faça: x <- 2+2 = 4	FALSE
It2	x == 4,	x <10 = TRUE	faça: x <- 4+2 = 6	FALSE
It3	x == 6,	x <10 = TRUE	faça: x <- 6+2 = 8	TRUE
It4	x == 10,	x <10 = FALSE	...	...
fim Enquanto				

Quebrando um laço do loop while(){} – usando o argumento **break** na condicional if(){}:

```
x <- 2                                  # um escalar "x" (valor inicial das sucessivas iterações),
while(x < 10){                          # enquanto = "x" for < 10 (condição booleana),
 if(x==8){break}                        # se, quebre = Pare de executar x+2, se x == 8,
 x <- x+2                               # faça = x+2 enquanto x < 10 e, se x == 8 seja verdade,
 print(x)                               # imprima = reporte os resultados das iterações de x+2,
}                                       # fim Enquanto
## [1] 4
## [1] 6
## [1] 8
```

Entendendo as iterações:

Enquanto (x <10)				
Iteração	**Para cada "x"(...)**	**Avalia**	**Faça: x+2**	**if {x == 8}(break)**
it1	x == 2,	x <10 = TRUE	faça: x <- 2+2 = 4	FALSE
It2	x == 4,	x <10 = TRUE	faça: x <- 4+2 = 6	FALSE
It3	x == 6,	x <10 = TRUE	faça: x <- 6+2 = 8	TRUE
fim Enquanto				

7.2.2 Comando for()

O comando for() (para) é uma estrutura de controle de fluxo no R inicia com a especificação, entre parênteses, do nome da variável de iteração e do vetor ou lista de elementos. Em seguida, entre chaves descreve-se o bloco de códigos que será executado a cada iteração (loop).

Comando for()

for(variável in sequência){código}

variável = nome da variável de iteração;
sequência = vetor ou lista de valores com *i* elementos; e
código = código a ser repetido sob cada elemento *i* da sequência, a cada iteração.

O pacote **magicfor** (MAKIYAMA, 2016) dispõe de funções interessantes para armazenar os resultados das iterações dos loops for(). A seguir a implementação de um loop e criação de um data frame com a função magic_result_as_dataframe(). No data frame é adicionada a variável de iteração *i* e os resultados da operação i^3:

```
for(i in 1:4){                          # para = cada elemento "i" do vetor 1:4,
  cub <- i^3                            # calcule: "i^3" e add ao objeto "cub",
  print(cub)                            # faça = imprima o objeto "cub",
}                                       # fim Para
magic_result_as_dataframe()             # um df com os resultados das iterações.
##   i cub
## 1 1   1
## 2 2   8
## 3 3  27
## 4 4  64
```

Entendendo as iterações:

Para (i in 1:4)		
Iteração	**Para cada "i"(...)**	**faça: i^3**
It1	i == 1,	faça: $1^3 = 1$
It2	i == 2,	faça: $2^3 = 8$
It3	i == 3,	faça: $3^3 = 27$
It4	i == 4,	faça: $4^3 = 64$
fim Para		

O comando a seguir produz o mesmo efeito:

```
x <- c(1, 2, 3, 4)                          # cria um vetor "x" para ser usado em seq,

for(i in x){                                # para = cada elemento “i” do vetor "x",
  cub <- i^3                                # calcule: “i^3” e add ao objeto "cub",
  print(cub)                                # faça = imprima o objeto "cub",
}                                           # fim Para
magic_result_as_dataframe()                 # um df com os resultados das iterações.
##   i cub
## 1 1   1
## 2 2   8
## 3 3  27
## 4 4  64
```

Cria um vetor **x** que será usado no loop, que executará i^3 e *i*i*:

```
x <- c(1, 2, 3, 4)                          # cria um vetor "x" para ser usado em seq,

for(i in x){                                # para: cada elemento “i” do vetor "x",
  cub <- i^3                                # faça: calcule i^3 e add ao objeto "cub",
  prod <- i*i                               # faça: calcule i*i e add ao objeto "prod",
  put(cub, prod)                            # armazene: os valores da iteração,
}                                           # fim Para
## cub: 1, prod: 1
## cub: 8, prod: 4
## cub: 27, prod: 9
## cub: 64, prod: 16
magic_result_as_dataframe()                 # um df com os resultados das iterações.
##   i
## 1 1
## 2 2
## 3 3
## 4 4
```

Cria um vetor **x** vazio e atribui-se o resultado do produto (*i*i*) às posições *i* do vetor. É interessante observar o que acontece quando se executa a função dentro e fora do comando. Para o exemplo, quando usada a função print() dentro do loop obtêm-se o retorno do vetor **x** após cada iteração *i*. Do contrário, ao imprimir o objeto fora do loop obtêm-se apenas o vetor **x** final.

```
x <- numeric(0)                             # cria um vetor numérico “x” vazio,

for(i in 1:6){                              # para: cada elemento "i" na seq. de 1 a 6,
  x[i] <- i*i                               # faça: calcule i*i e adicione ao vetor "x",
  print(x)                                  # imprima = o vetor "x" ao fim de cada iteração,
}                                           # fim Para

print(x)                                    # imprima o vetor "x" final.
```

Usando os elementos do vetor **x** para obter i^2, e adicionando os resultados ao vetor **y**:

```
x <- c(1, 2, 3, 4)                    # cria um vetor numérico "x" qualquer,
y <- numeric(0)                       # cria um vetor numérico "y" vazio,

for(i in x){                          # para: cada elemento "i" em "x",
  y[i] <- x[i]^2                      # faça: calcule x[i]^2 e adicione ao vetor "y",
  print(y)                            # imprima: o vetor "y" ao fim de cada iteração,
}                                     # fim Para

print(y)                              # imprima: o vetor "y" final.
```

O mesmo resultado é alcançado usando-se a função append(). Para isso, basta informar os parâmetros **x** (vetor que receberá os elementos de **values**) e **values** (valores que devem ser anexados ao vetor **x**).

```
y <- c()                              # cria: um vetor "y" vazio,
for(i in 1:4){                        # para: cada elemento "i" na seq. 1:4,
  y <- append(x = y,
              values = i^2)           # faça: calcule i^2 e anexe a "y",
  print(y)                            # imprima: o vetor "y" ao fim de cada iteração,
}                                     # fim Para

print(y)                              # imprima: o vetor "y" final.
```

Condicional if(){} dentro do comando for() - para caso de vetores:

```
x <- c(10, 9, 15, 16)                 # cria: um vetor numérico "x" qualquer,

for(i in x){                          # para: cada elemento "i" em "x",
  if (i == 10){                       # se: "i" for = 10,
    print("É igual a 10")             # imprima: "É igual a 10",
  }else{                              # do contrário,
    print("É diferente de 10")        # imprima: "É diferente de 10",
  }                                   # fim Se
}                                     # fim Para
```

Aplicando loops aninhados (consecutivos) com o comando for(): no exemplo, é criada uma matriz a partir do produto de duas sequências: (1, 2, 3) e (1, 2), cujos elementos constituintes são representados por *i* e *j*, respectivamente. O resultado do produto é adicionado a uma matriz:

```
matrix <- matrix(nrow = 3, ncol = 2)

for(i in 1:3){
  for(j in 1:2){
matrix[i,j] = i*j
  }
}
```

```
matrix
##      [,1] [,2]
## [1,]    1    2
## [2,]    2    4
## [3,]    3    6
```

8 Gerando de amostras aleatórias

8.1 Função sample()

A função sample() permite gerar amostras aleatórias de um determinado vetor ou conjunto de dados, com ou sem reposição dos elementos sorteados. Ao usar a função é necessário usar conjuntamente a função set.seed() para garantir a reprodutibilidade dos números aleatorizados. Caso a função set.seed() não seja inserida, a cada execução da função sample() um novo conjunto de amostras aleatórias será gerado.

Função sample()

sample(**x** = ?, **size** = ?, **replace** = ?, **prob** = ?)

x = um vetor de um ou mais elementos de onde deverá ser retirada a amostra ou um inteiro positivo;
size = número não negativo e inteiro, referente ao tamanho da amostragem a ser gerada;
replace = argumento informando se a amostragem deve ser feita com reposição ou não. O default é FALSE (sem reposição); e
prob = vetor de probabilidades para obtenção dos elementos do vetor **x** que será amostrado. O default é NULL.

Usando a função sample()

a) Gerar uma amostra aleatória de 10 números inteiros, sem reposição, no intervalo de 0 e 100.

```
set.seed(1)
sample(x=0:100, size=10)
##  [1] 26 37 56 89 19 86 97 62 58  5
```

b) Gerar uma amostra aleatória de 15 números inteiros, sem reposição, do vetor **x**.

```
set.seed(2)
x <- seq(from=1, to=50, by=1)
sample(x=x, size=15)
##  [1] 10 35 28  8 44 43  6 36 20 23 41 50 29  7 15
```

c) Gerar uma amostra aleatória de 5 números, com reposição, entre os possíveis resultados do lançamento de um dado.

```
set.seed(3)
sample(x=1:6, size=5, replace = TRUE)
## [1] 2 5 3 2 4
```

d) Gerar uma amostra aleatória de 6 números, sem reposição, entre os possíveis resultados do lançamento de um dado.

```
set.seed(4)
sample(x=1:6, size=6, replace = FALSE)
## [1] 4 1 2 5 6 3
```

e) Gerar uma amostra aleatória, sem reposição, a partir do vetor de caractere de 6 espécies florestais.

```
set.seed(5)
Especie <- c("Acapu", "Angelim Pedra", "Timborana", "Maparajuba",
         "Ipê Amarelo", "Jatobá")
sample(x=Especie, size=6, replace = FALSE)
## [1] "Angelim Pedra" "Maparajuba"    "Ipê Amarelo"   "Acapu"
## [5] "Timborana"     "Jatobá"
```

f) **Gera um erro**: quando **replace** = FALSE, o tamanho da amostra (size) não pode ser maior do que a população em **x**.

```
set.seed(6)
Especie <- c("Acapu", "Angelim Pedra", "Timborana", "Maparajuba",
         "Ipê Amarelo", "Jatobá")
sample(x=Especie, size=7, replace = FALSE)
```

g) Gerar uma amostra aleatória de 50 árvores, com reposição, a partir do vetor **diâmetro**.

```
set.seed(7)
diametro <- c(23.0, 27.0, 33.6, 42.6, 52.1, 63.2)
sample(x=diametro, size=50, replace = TRUE)
##  [1] 63.2 33.6 23.0 23.0 27.0 52.1 33.6 63.2 23.0 33.6 27.0 27.0 52.1 23.0
## [15] 33.6 23.0 42.6 23.0 63.2 27.0 42.6 27.0 63.2 63.2 63.2 23.0 42.6 33.6
## [29] 63.2 33.6 52.1 27.0 27.0 27.0 33.6 63.2 33.6 52.1 63.2 33.6 52.1 33.6
## [43] 52.1 33.6 63.2 27.0 23.0 52.1 63.2 63.2
```

h) Gerar uma amostra aleatória dos elementos de um data frame e, em seguida, separar em dados de treino e validação (procedimento comum em **aprendizado de máquina**).

Para fins didáticos, usar-se-á o conjunto de dados **trees** que contém informações de circunferência, altura e volume de 31 árvores de cerejeira. Assim, a função sample() será escrita com a seguinte sintaxe:

x = receberá os inteiros que representam os dois conjuntos de dados (1 e 2);

size = receberá o número de linhas (nrow) do conjunto de dados;

replace = TRUE, atribuirá os valores 1 (um) ou 2 (dois) a uma determinada linha do conjunto; e

prob = receberá um vetor de probabilidade (0,7 e 0,3).

Definindo-se **replace** = TRUE, cada novo sorteio atribui 1 ou 2 às linhas do data frame, até que o vetor de probabilidades de 0,7 e 0,3 sejam satisfeitos ou fiquem próximo do desejado.

```
set.seed(8)
data("trees")

amostra.aleatoria <- sample(x=1:2, size=nrow(trees),
                            replace=TRUE, prob=c(0.7, 0.3))
amostra.aleatoria
##  [1] 1 1 2 1 1 2 1 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 2 1

table(amostra.aleatoria)
## amostra.aleatoria
##  1  2
## 24  7

round(prop.table(table(amostra.aleatoria)), digits = 2)
## amostra.aleatoria
##    1    2
## 0.77 0.23
```

Ao imprimir o objeto **amostra.aleatoria** o sorteio realizado será evidenciado pelos números inteiros (1 e 2) definidos no argumento **x**. É possível perceber também a ação do argumento **prob** no processo de amostragem. Cada valor **1** e **2** receberá uma linha do conjunto de dados **trees**, seguindo a ordem sequencial de sua ocorrência no objeto **amostra.aleatoria**. Por exemplo, as árvores situadas nas linhas 3, 6, 8, 9, 16, 26 e 30 constituirão o conjunto de validação.

```
which(amostra.aleatoria %in% 1)          # linhas sorteadas e que irão compor a amostra de treino

##  [1]  1  2  4  5  7 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 27 28 29
## [24] 31

which(amostra.aleatoria %in% 2)          # linhas sorteadas e que irão compor a amostra de validação

## [1]  3  6  8  9 16 26 30
```

```
treino <- trees[amostra.aleatoria==1,]          # árvores da amostra de treino

validacao <- trees[amostra.aleatoria==2,]     # árvores da amostra de validação
```

9 Visualização gráfica no R

O R é um poderoso software para edição de gráficos. Nesse livro iremos trabalhar com o pacote "graphics" já instalado no R base. A lista de argumentos utilizados na construção de gráficos pode ser acessada fazedo **?par()** no console do R:

```
?par()
```

Demonstrativo de gráficos no R

Para a visualização dos possíveis gráficos que podem ser realizados digitamos o comando demo (graphics) no console do R e clicamos repetidamente no comando de execução **Enter + Ctrl.** Os gráficos são mostrados no item *Plots* do RStudio.

```
demo(graphics)
```

Tipos de comandos: alto nível, baixo nível e interativos

- Comandos de alto nível (criam gráficos completos)
plot(); boxplot(); e hist().

- Comandos de baixo nível (adicionam informações a algum gráfico já existente)
points(); lines(); e title().

- Comandos interativos (permitem que o usuário interaja com a janela gráfica)
identify()

9.1 A função plot()

É uma função genérica para plotagem de objetos R. Para saber mais detalhes sobre os argumentos dos parâmetros gráficos, consulte a função **par**. O uso da função plot() é um dos mecanismos mais simples para a criação dos gráficos no R. Para estudar alguns detalhes da função usar-se-á de dados biométricos de 30 árvores de um povoamento de *Tectona grandis* situado nas coordenadas geográficas Latitude -03°16'10,96''e Longitude -52°23'42,58'', no Campo Experimental da

Embrapa Amazônia Oriental, km 23 da Rodovia BR 230 (Transamazônica). O volume do fuste de *T. grandis* pelo obtido pelo método de *Huber*.

Sintaxe

plot(**x**, **y**, **type** = "", **main** = "", **sub** = "", **xlab** = "", **ylab** = "", **asp** = "" ….)

```
install.packages("data.table")      #Instalamos o pacote data.table para facilitar a chamada dos dados
library(data.table)
teca <- fread("Tectona.csv")
print(teca)
```

Arvore	DAP	H	Volume
1	6.600	8.900	0.017
2	6.800	7.950	0.017
3	7.000	6.870	0.014
4	7.100	12.700	0.027
5	7.500	11.740	0.028
6	9.900	13.840	0.066
7	10.800	15.250	0.049
8	11.200	13.000	0.058
9	11.300	14.600	0.066
10	11.700	15.600	0.083
11	12.600	17.200	0.100
12	14.000	16.600	0.127
13	14.300	17.870	0.158
14	14.600	17.900	0.141
15	14.900	16.500	0.132
16	15.000	18.300	0.153
17	15.300	17.400	0.162
18	15.400	17.550	0.139
19	16.200	16.990	0.192
20	17.100	17.720	0.158
21	17.400	16.200	0.196
22	18.000	16.100	0.197
23	18.100	15.340	0.250
24	19.200	16.150	0.203
25	19.800	14.380	0.262
26	20.600	22.000	0.317
27	22.500	20.600	0.377
28	23.500	16.700	0.263
29	23.800	22.800	0.395
30	32.600	21.550	0.603

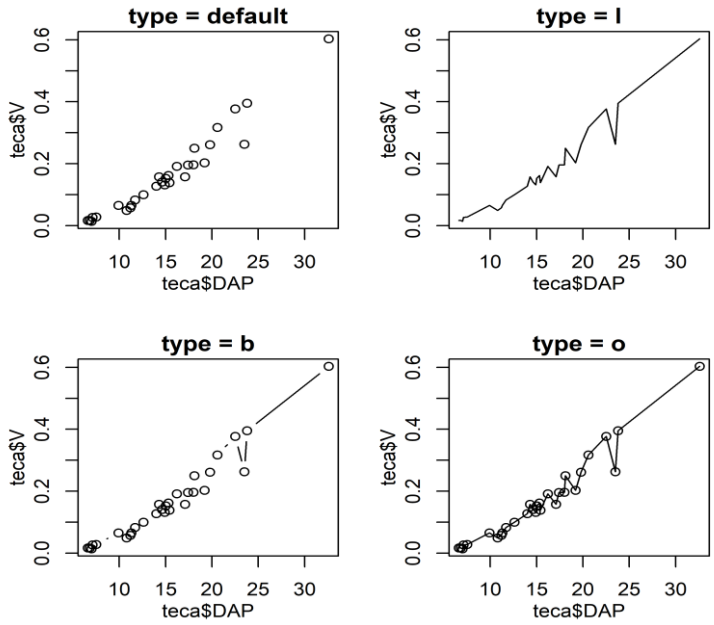
Inicialmente, uma análise exploratória do conjunto de dados:

```
str(teca)                                   # estrutura do conjunto de dados
dim(teca)                                   # dimensão do conjunto de dados
summary(teca)                               # um rápido sumário estatístico
##     Arvore           DAP              H               Volume
## Min.   : 1.00   Min.   : 6.60   Min.   : 6.87   Min.   :0.01419
## 1st Qu.: 8.25   1st Qu.:11.22   1st Qu.:14.44   1st Qu.:0.06555
## Median :15.50   Median :14.95   Median :16.35   Median :0.14713
## Mean   :15.50   Mean   :15.16   Mean   :15.88   Mean   :0.16496
## 3rd Qu.:22.75   3rd Qu.:18.07   3rd Qu.:17.68   3rd Qu.:0.20165
## Max.   :30.00   Max.   :32.60   Max.   :22.80   Max.   :0.60309
```

Criando um gráfico de dispersão entre diâmetro (cm) e altura (m). O default da função plot() é usar **type = "p"**. No entanto outros tipos de plotagem (type) estão disponíveis. O argumento type pode assumir: ("p" = points, "l" = lines, "b" = both, "c" = lines part alone of "b", "o" = 'overplotted', "h" = 'histogram', "s" = stair steps, "S" = other steps, "n" = no plotting).

9.1.1 Modificando o tipo de gráfico (type)

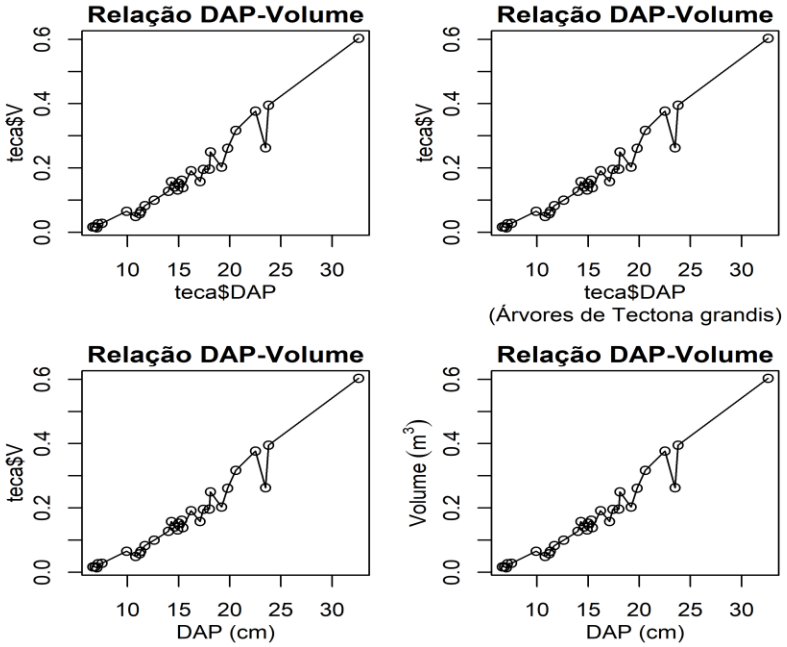
```
plot(teca$DAP, teca$V, main = "type = default")
plot(teca$DAP, teca$V, type="l", main = "type = l")
plot(teca$DAP, teca$V, type="b", main = "type = b")
plot(teca$DAP, teca$V, type="o", main = "type = o")
```



9.1.2 Adicionando título, subtítulo e rótulos

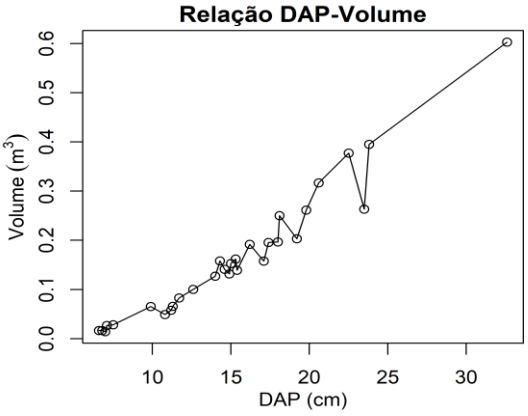
Usar os comandos **main** e **sub** para adicionar um título e subtítulo ao gráfico. Para modificar os títulos dos eixos usar os comandos **xlab** = "Título do eixo x" e **ylab** = "Título do eixo y".

```
plot(teca$DAP, teca$V, type = "o", main = "Relação DAP-Volume")

plot(teca$DAP, teca$V, type = "o", main = "Relação DAP-Volume",
    sub = "(Árvores de Tectona grandis)")

plot(teca$DAP, teca$V, type = "o", main = "Relação DAP-Volume",
    xlab = "DAP (cm)")

plot(teca$DAP, teca$V, type = "o", main = "Relação DAP-Volume",
     xlab = "DAP (cm)", ylab = expression(Volume ~ (m^3)))
```



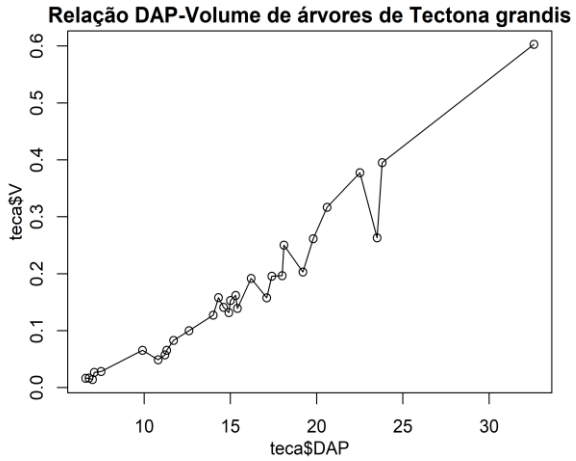
O comando **title()** (comando de baixo nível) constitui outro mecanismo de inserção de título e rótulos de eixos em gráficos da função plot().

```
plot(teca$DAP, teca$V, type = "o", xlab = "", ylab = "")
title("Relação DAP-Volume", xlab = "DAP (cm)", ylab = expression(Volume ~ (m^3)))
```

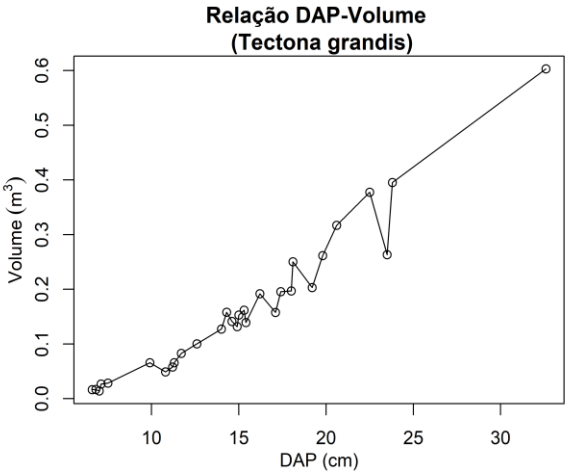


Se o título for demasiadamente grande pode-se fazer a quebra de linha inserindo "\n" no título:

```
plot(teca$DAP, teca$V, type="o", main="Relação DAP-Volume de árvores de Tectona grandis")
```



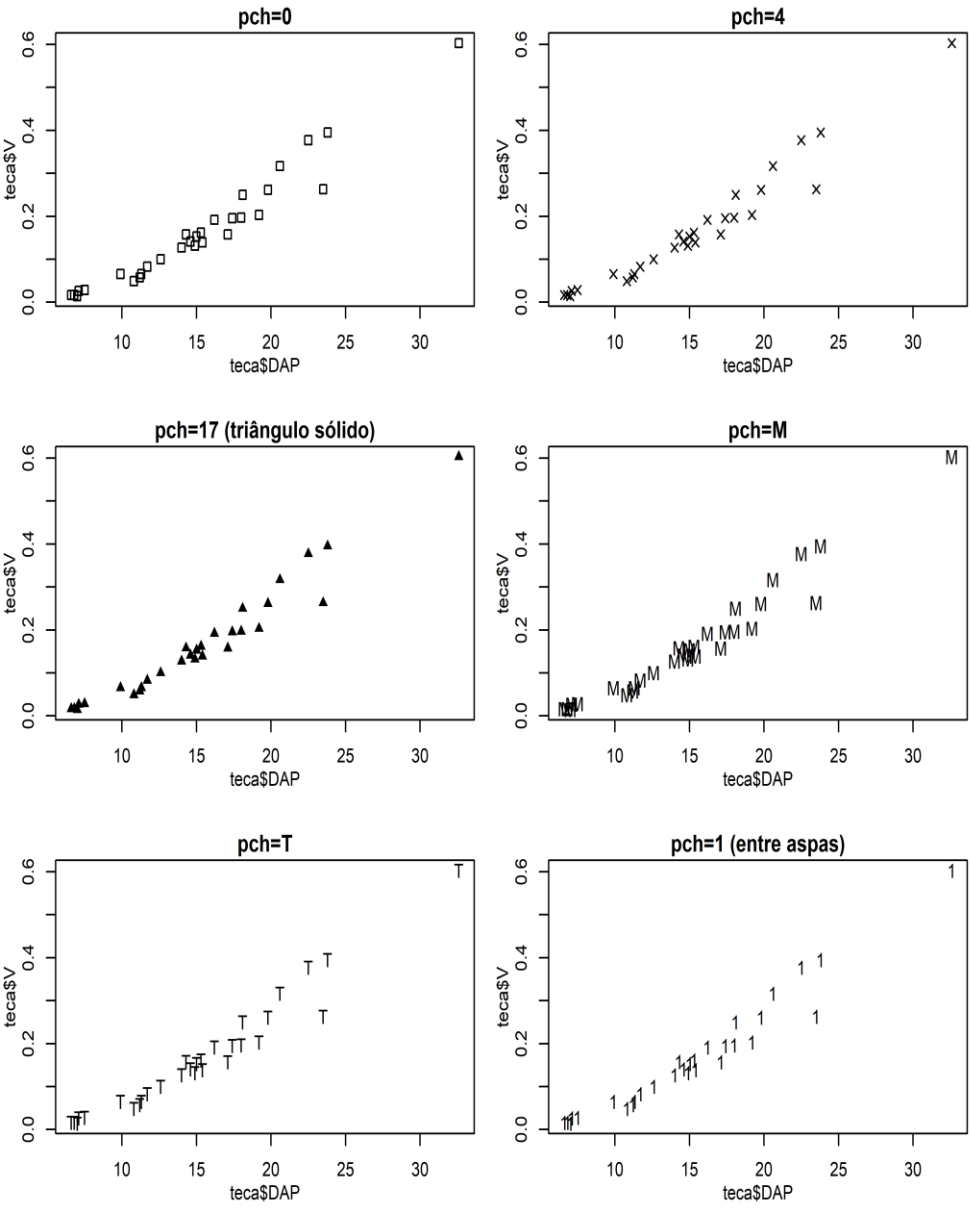

```
plot(teca$DAP, teca$V, type="o",  main="Relação DAP-Volume \n (Tectona grandis)",
   xlab = "DAP (cm)", ylab = expression(Volume ~ (m^3)))
```



9.1.3 Modificando tipos de pontos (pch)

Pode-se alterar o tipo de ponto usando o comando **pch**, que recebe um valor numérico que define o tipo de ponto plotado no gráfico.

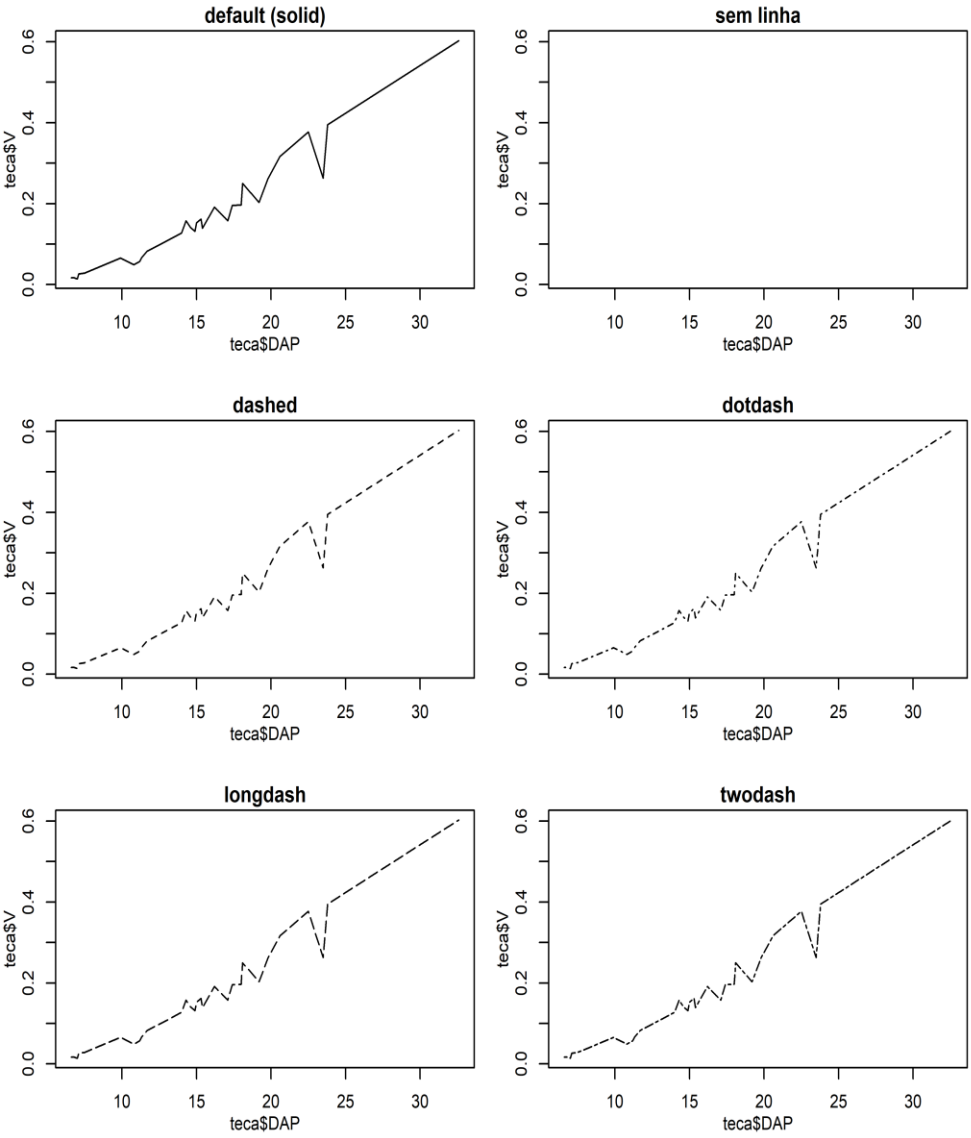
```
plot(teca$DAP, teca$V, pch=0, main = "pch=0")
plot(teca$DAP, teca$V, pch=4, main = "pch=4")
plot(teca$DAP, teca$V, pch=17, main = "pch=17 (triângulo sólido)")
plot(teca$DAP, teca$V, pch="M", main = "pch=M")
plot(teca$DAP, teca$V, pch="T", main = "pch=T")
plot(teca$DAP, teca$V, pch="1", main = "pch=1 (entre aspas)")
```



9.1.4 Modificando tipos de linhas (lty)

Pode-se alterar o tipo de linha usando o comando **lty**, que assume um valor numérico que varia de 0 a 6.

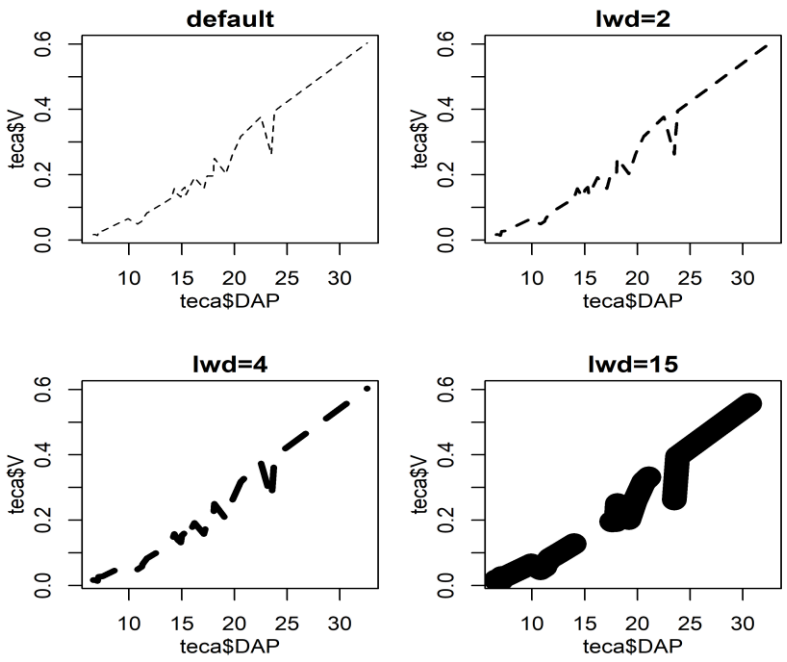
```
plot(teca$DAP, teca$V, type="l", main = "default (solid)")
plot(teca$DAP, teca$V, type="l", lty=0, main = "sem linha")
plot(teca$DAP, teca$V, type="l", lty=2, main = "dashed")
plot(teca$DAP, teca$V, type="l", lty=4, main = "dotdash")
plot(teca$DAP, teca$V, type="l", lty=5, main = "longdash")
plot(teca$DAP, teca$V, type="l", lty=6, main = "twodash")
```



9.1.5 Modificando a largura das linhas (lwd)

Pode-se alterar a largura da linha usando o comando **lwd**, que assume sempre um valor numérico positivo maior ou igual a 1.

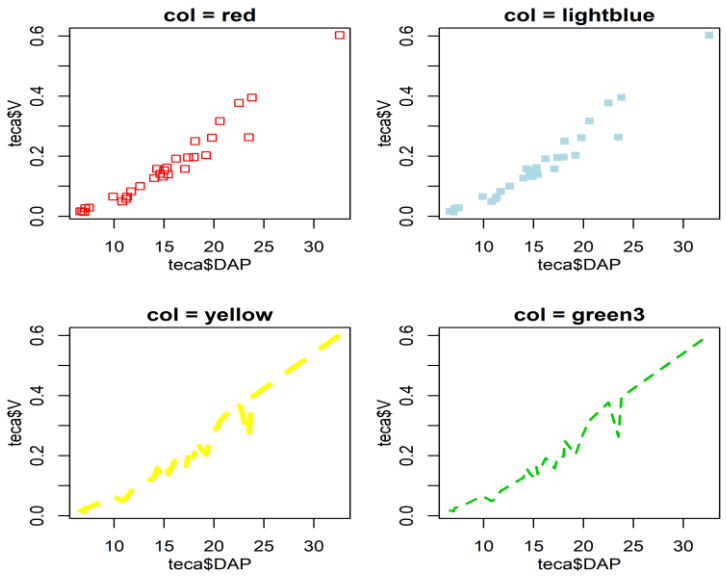
```
plot(teca$DAP, teca$V, type="l", lty=2, main = "default")
plot(teca$DAP, teca$V, type="l", lty=2, lwd=2, main = "lwd=2")
plot(teca$DAP, teca$V, type="l", lty=2, lwd=4, main = "lwd=4")
plot(teca$DAP, teca$V, type="l", lty=2, lwd=15, main = "lwd=15")
```



9.1.6 Modificando cores de pontos e linhas (col)

Pode-se alterar a cor das linhas e pontos. Basta usar o comando **col** e especificar entre aspas a cor desejada. A função colors() fornece uma variedade de cores disponíveis para uso. Utilize a função demo("colors") para visualizar uma demostração de cores.

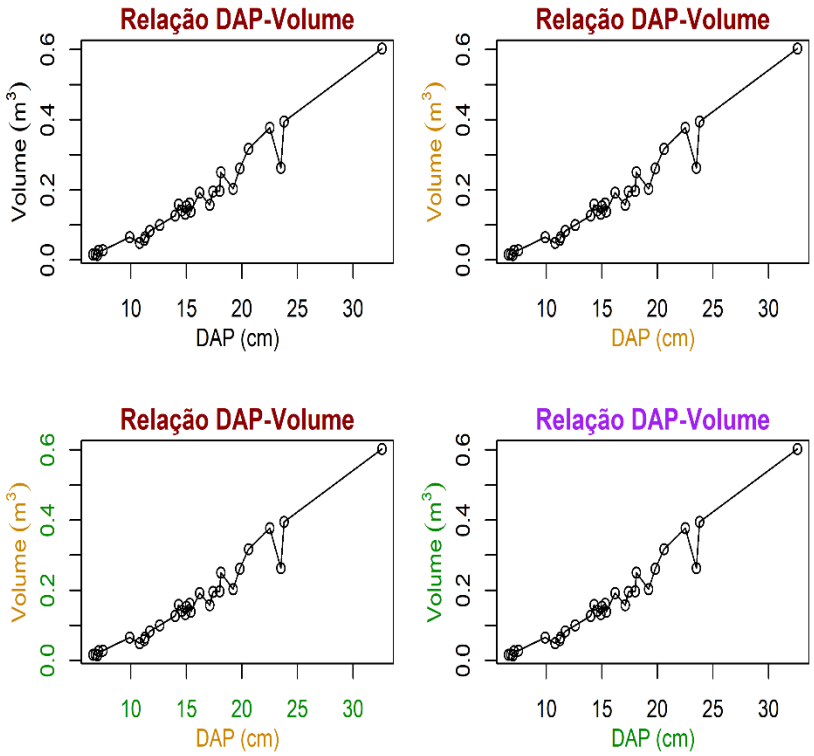
```
head(colors(), 40)
length(colours())
plot(teca$DAP, teca$V, pch=0, col="red", main = "col = red")
plot(teca$DAP, teca$V, pch=15, col="lightblue", main = "col = lightblue")
plot(teca$DAP, teca$V, type="l", lty=2, lwd=4, col="yellow", main = "col = yellow")
plot(teca$DAP, teca$V, type="l", lty=2, lwd=2, col="green3", main = "col = green3")
```



9.1.7 Modificando cores do título e eixos (col.main, col.lab e col.axis)

Podem-se alterar as cores do título e labels dos eixos **x** e **y** usando os comandos **col.main** e **col.lab**. O comando **col.axis** pode ser usado para modificar as cores dos valores dos eixos. Além disso, existem comandos de baixo nível que realizam as mesmas tarefas: **title()**.

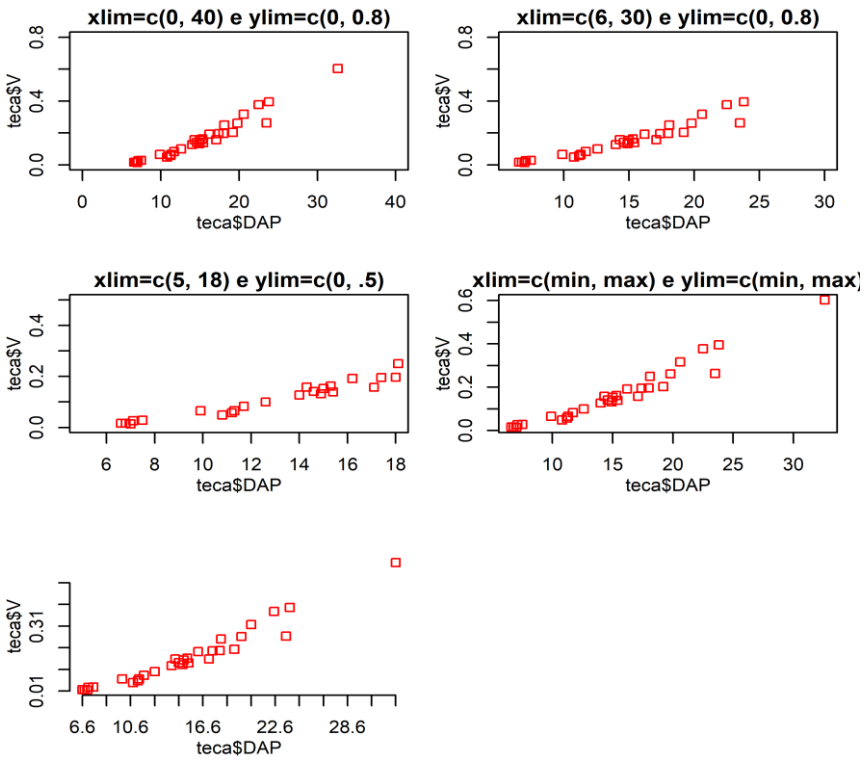
```
plot(teca$DAP, teca$V, type="o", main="Relação DAP-Volume",
    xlab="DAP (cm)", ylab=expression(Volume ~ (m^3)),
    col.main="red4")

plot(teca$DAP, teca$V, type="o", main="Relação DAP-Volume",
    xlab="DAP (cm)", ylab=expression(Volume ~ (m^3)),
    col.main="red4", col.lab="orange3")

plot(teca$DAP, teca$V, type="o", main="Relação DAP-Volume",
    xlab="DAP (cm)", ylab=expression(Volume ~ (m^3)),
    col.main="red4", col.lab="orange3",
    col.axis="green4")

plot(teca$DAP, teca$V, type="o", xlab="", ylab="")
title("Relação DAP-Volume", col.main="purple",
    xlab="DAP (cm)", ylab=expression(Volume ~ (m^3)), col.lab="green4")
```



9.1.8 Modificando os limites dos eixos (xlim, ylim e axis)

Podem-se alterar os limites dos eixos (mínimos e máximos) usando os comandos **xlim** e **ylim**. Além disso, o comando **axis** pode ser usado para obter uma maior personalização dos eixos. Porém, no plot original deve constar **axis = FALSE**.

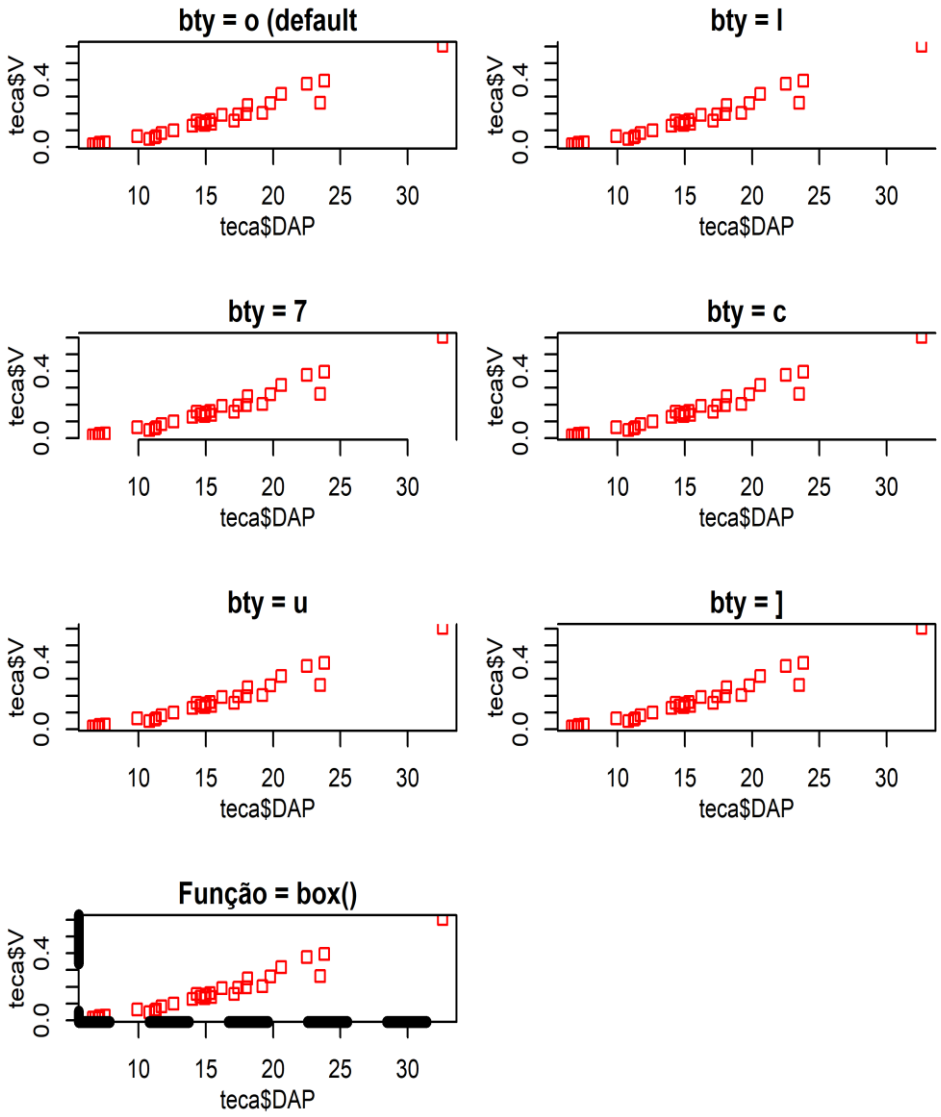
```
plot(teca$DAP, teca$V, pch=0, col="red", xlim=c(0, 40), ylim=c(0, 0.8),
   main="xlim=c(0, 40) e ylim=c(0, 0.8)")

plot(teca$DAP, teca$V, pch=0, col="red", xlim=c(6, 30), ylim=c(0, 0.8),
    main="xlim=c(6, 30) e ylim=c(0, 0.8)")

plot(teca$DAP, teca$V, pch=0, col="red", xlim=c(5, 18), ylim=c(0, 0.5),
   main="xlim=c(5, 18) e ylim=c(0, .5)")

plot(teca$DAP, teca$V, pch=0, col="red", xlim=c(min(teca$DAP), max(teca$DAP)),
   ylim=c(min(teca$V), max(teca$V)), main="xlim=c(min, max) e ylim=c(min, max)")

plot(teca$DAP, teca$V, pch=0, col="red", axes=F)
axis(1, seq(from = round(min(teca$DAP),2),
        to = round(max(teca$DAP),2),
        by = 2))
axis(2, seq(from = round(min(teca$V),2),
        to = round(max(teca$V),2),
        by = 0.1))
```



9.1.9 Modificando as bordas dos gráficos (bty)

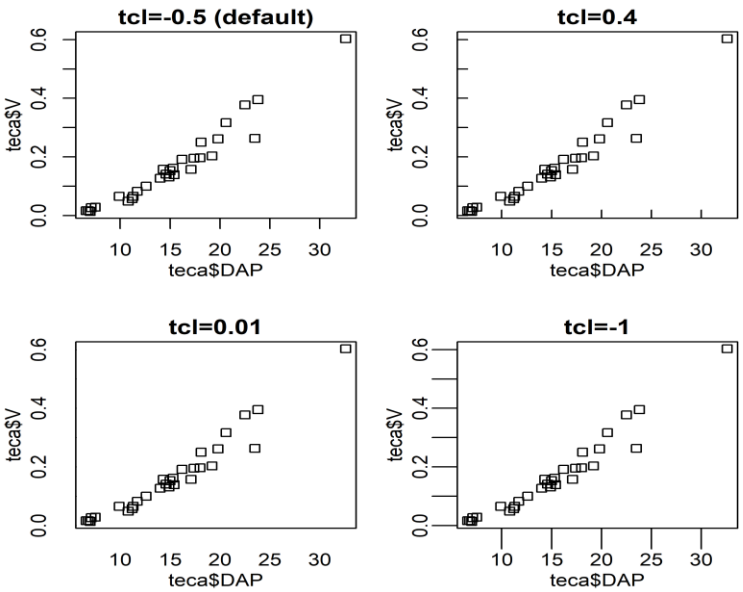
Pode-se alterar o estilo de borda do gráfico usando o comando **bty**. As opções disponíveis são: **bty** =“o”, “l”, “7”, “c”, “u”, “]”. Também pode ser o usado o comando **box( )** após a chamada do gráfico.

```
plot(teca$DAP, teca$V, pch=0, col="red", bty="o", main="bty = o (default")
plot(teca$DAP, teca$V, pch=0, col="red", bty="l", main="bty = l")
plot(teca$DAP, teca$V, pch=0, col="red", bty="7", main="bty = 7")
plot(teca$DAP, teca$V, pch=0, col="red", bty="c", main="bty = c")
plot(teca$DAP, teca$V, pch=0, col="red", bty="u", main="bty = u")
plot(teca$DAP, teca$V, pch=0, col="red", bty="]", main="bty = ]")

plot(teca$DAP, teca$V, pch=0, col="red", main="Função = box()")
box(which="plot", lty=2, bty ="l", lwd=5)
```



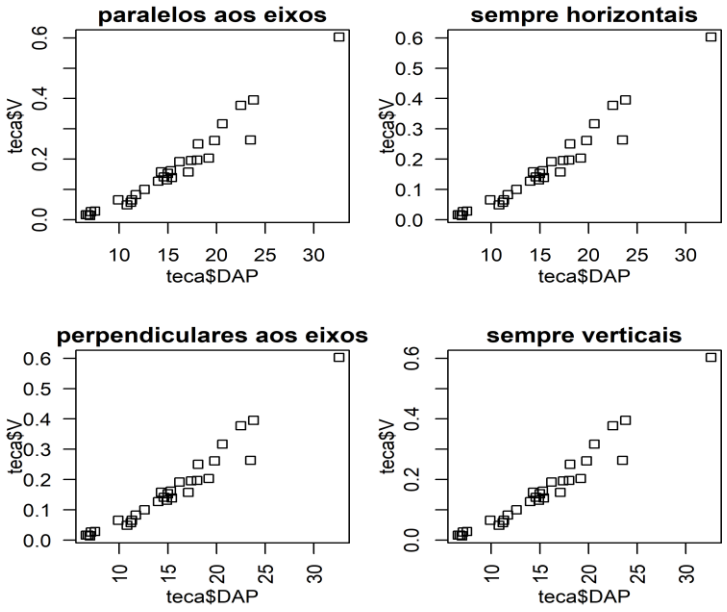
9.1.10 Modificando o comprimento e a direção dos marcadores de eixos (tcl)

```
plot(teca$DAP, teca$V, pch=0, main="tcl=-0.5 (default)")
plot(teca$DAP, teca$V, pch=0, tcl=0.4, main="tcl=0.4")
plot(teca$DAP, teca$V, pch=0, tcl=0.01, main="tcl=0.01")
plot(teca$DAP, teca$V, pch=0, tcl=-1, main="tcl=-1")
```



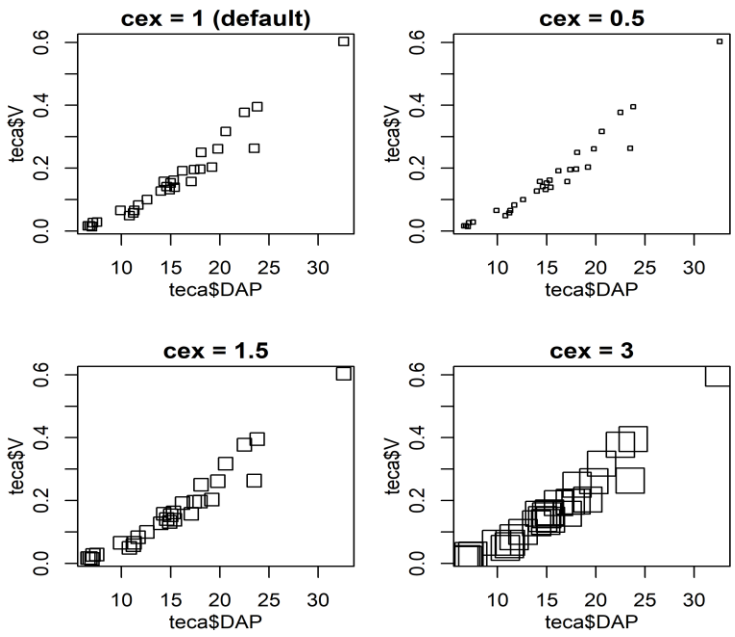
9.1.11 Modificando a orientação dos valores nos eixos (las)

```
plot(teca$DAP, teca$V, pch=0, las=0, main="paralelos aos eixos")
plot(teca$DAP, teca$V, pch=0, las=1, main="sempre horizontais")
plot(teca$DAP, teca$V, pch=0, las=2, main="perpendiculares aos eixos")
plot(teca$DAP, teca$V, pch=0, las=3, main="sempre verticais")
```



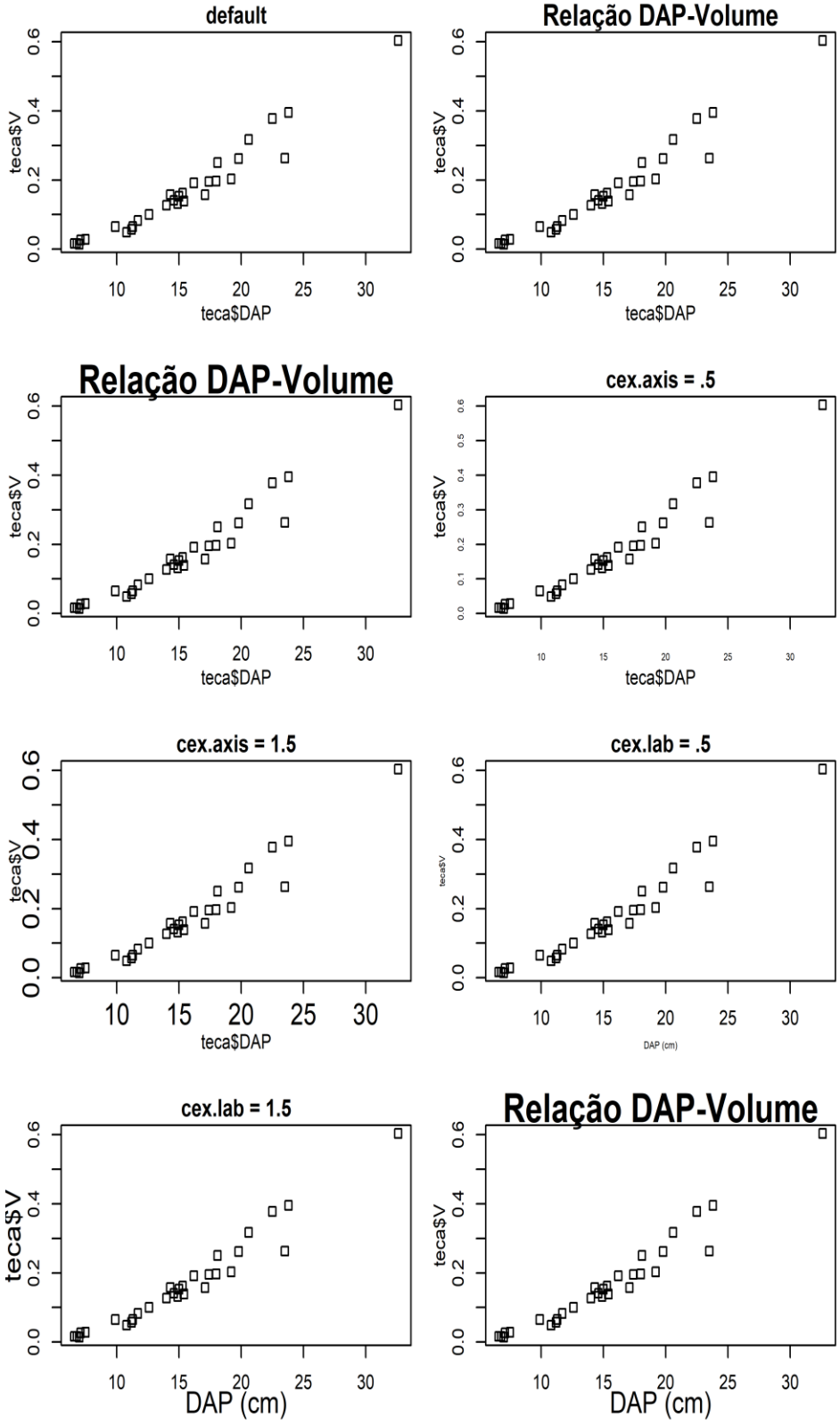
9.1.12 Modificando o tamanho dos pontos (cex)

```
plot(teca$DAP, teca$V, pch=0,  main="cex = 1 (default)")
plot(teca$DAP, teca$V, pch=0, cex=0.5, main="cex = 0.5")
plot(teca$DAP, teca$V, pch=0, cex=1.5, main="cex = 1.5")
plot(teca$DAP, teca$V, pch=0, cex=3, main="cex = 3")
```



9.1.13 Modificando o tamanho do título, valores e labels dos eixos (cex.main, cex.axis e cex.lab)

```
plot(teca$DAP, teca$V, pch=0, main="default")
plot(teca$DAP, teca$V, pch=0, main="Relação DAP-Volume", cex.main=1.5)
plot(teca$DAP, teca$V, pch=0, main="Relação DAP-Volume", cex.main=2)
plot(teca$DAP, teca$V, pch=0, cex.axis=.5, main="cex.axis = .5")
plot(teca$DAP, teca$V, pch=0, cex.axis=1.5, main="cex.axis = 1.5")
plot(teca$DAP, teca$V, pch=0, xlab="DAP (cm)",cex.lab=.5, main="cex.lab = .5")
plot(teca$DAP, teca$V, pch=0, xlab="DAP (cm)",cex.lab=1.5, main="cex.lab = 1.5")
#Usando o comando title( )
plot(teca$DAP, teca$V, pch=0, xlab="")
title(main="Relação DAP-Volume", cex.main=2, xlab="DAP (cm)", cex.lab=1.5)
```


default
Relação DAP-Volume
Relação DAP-Volume
cex.axis = .5
cex.axis = 1.5
cex.lab = .5
cex.lab = 1.5
Relação DAP-Volume
teca$V
teca$DAP
DAP (cm)

9.2 A função hist()

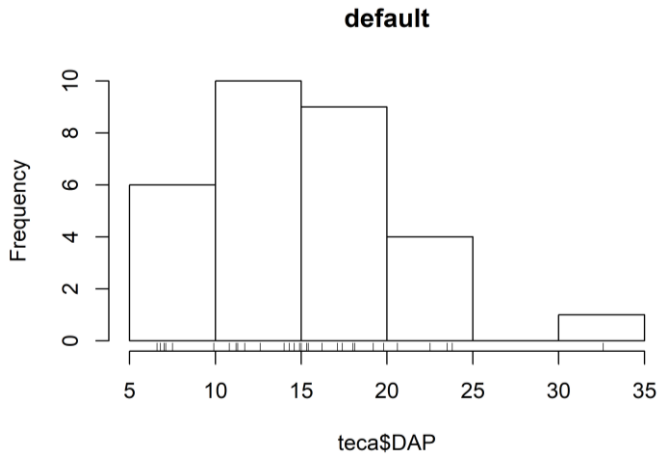
A função genérica hist() gera um histograma para o conjunto de dados fornecidos. Caso a especificação seja **plot = TRUE** (default), um histograma é plotado. Do contrário, se **plot = FALSE** a função apenas calcula internamente todos os parâmetros de hist(). Você pode acessar os argumentos da função fazendo: **?hist** ou **?plot.histogram**.

Função hist()

hist(**x**, **breaks** = "Sturges", **nrow**, **freq** = NULL, **include.lowest** = TRUE, **right** = TRUE, **col** = NULL, **border** = NULL, **main** = "Histogram of", **xlim** = range(breaks), **ylim** = NULL, **xlab**, **ylab**, **axes** = TRUE, **right** = TRUE)

x = conjunto de dados;
breaks = define os intervalos da classe - pode ser uma função, vetor, número;
freq = se TRUE = frequência (contagem), FALSE = densidade (probabilidade);
include.lowest = incluir o menor valor no intervalo definido pelo break;
col e border = cores das barras e das bordas; **main** = Título do gráfico;
xlim e ylim = limites dos eixos;
xlab e ylab = título dos eixos;
axes = se axes = TRUE os eixos são desenhados; e
right = se right = TRUE intervalo aberto a esquerda.

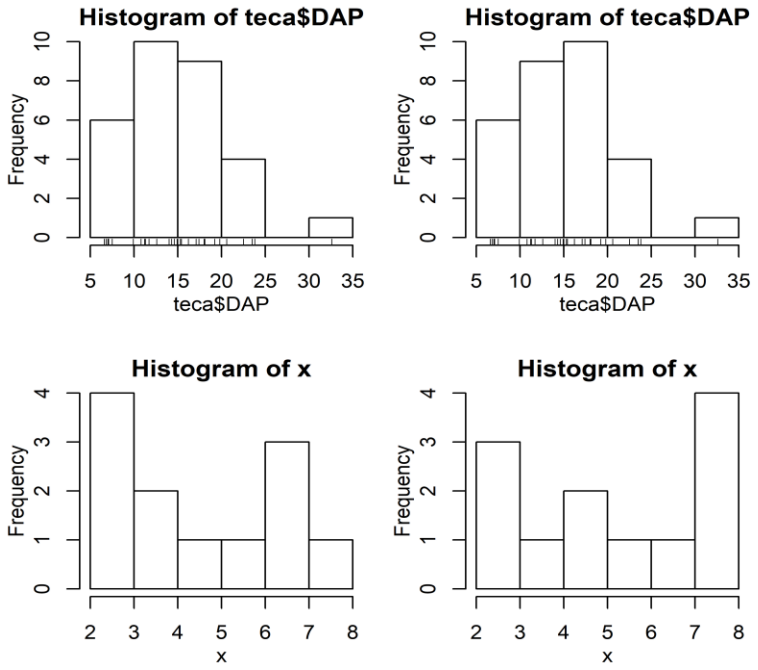
```
hist(teca$DAP, main="default")
rug(teca$DAP)  #Adiciona a quantidade de observações dentro do intervalo de classes, representado pelas lin
has dispostas abaixo das barras do histograma
```



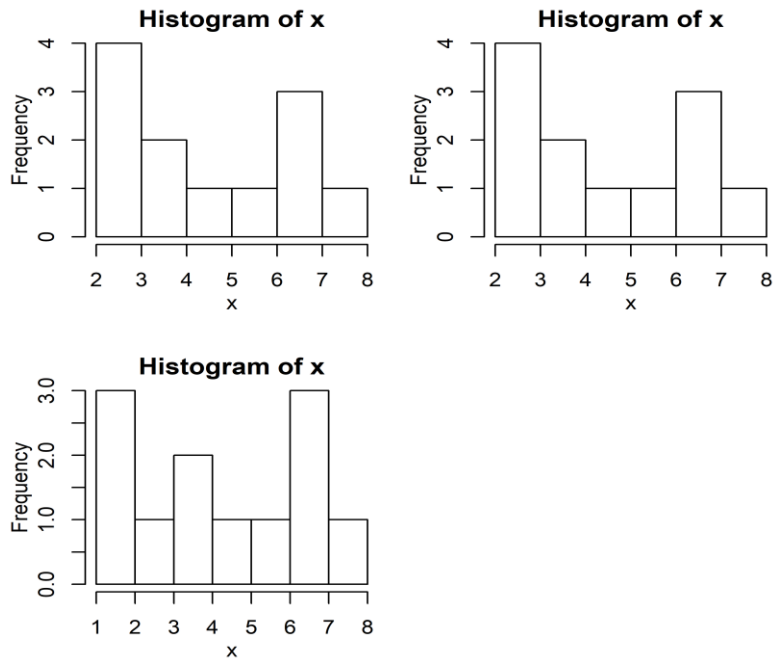
9.2.1 Modificando o intervalo das classes

```
hist(teca$DAP, right = TRUE); rug(teca$DAP)
hist(teca$DAP, right = FALSE);  rug(teca$DAP)          #Intervalo aberto a direita
```

```
#Outro exemplo
x <- c(2,2,2,3,4,4,5,6,7,7,7,8)
hist(x, right=TRUE)
hist(x, right=FALSE)
```

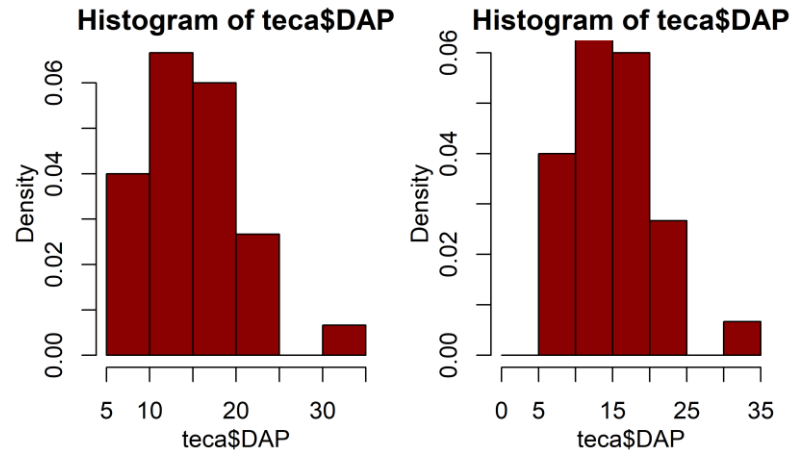



```
hist(x, right = TRUE, include.lowest = TRUE)

#Mensagem de aviso, pois falta o argumento breaks
hist(x, right = TRUE,  include.lowest = FALSE)

## Warning in hist.default(x, right = TRUE, include.lowest = FALSE):
## 'include.lowest' ignored as 'breaks' is not a vector

hist(x, right = TRUE, include.lowest = FALSE, breaks=1:8)          #Intervalo entre as classes
```



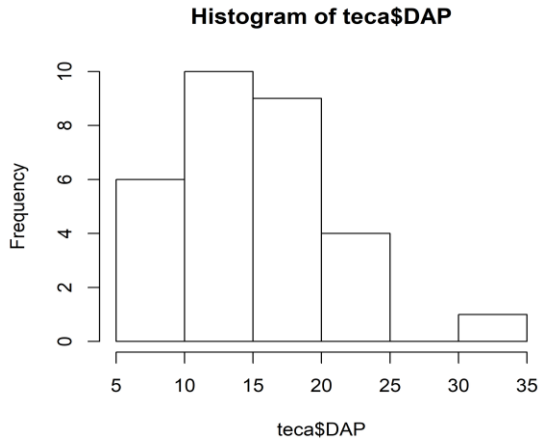
9.2.2 Modificando os breaks (intervalos de classe)

```
hist(teca$DAP, col="red4", probability = TRUE)
hist(teca$DAP, col="red4", probability = TRUE,
    breaks = c(c(0,5), c(5,10), c(10,15), c(15,20), c(20,25), c(25,30), c(30,35)),
    ylim=c(0,0.06))

#Os breaks indicam que o gráfico deve ter classes de 0 até 5, 5 até 10, 10 até 15 e assim por diante
```



9.2.3 Parâmetros do histograma

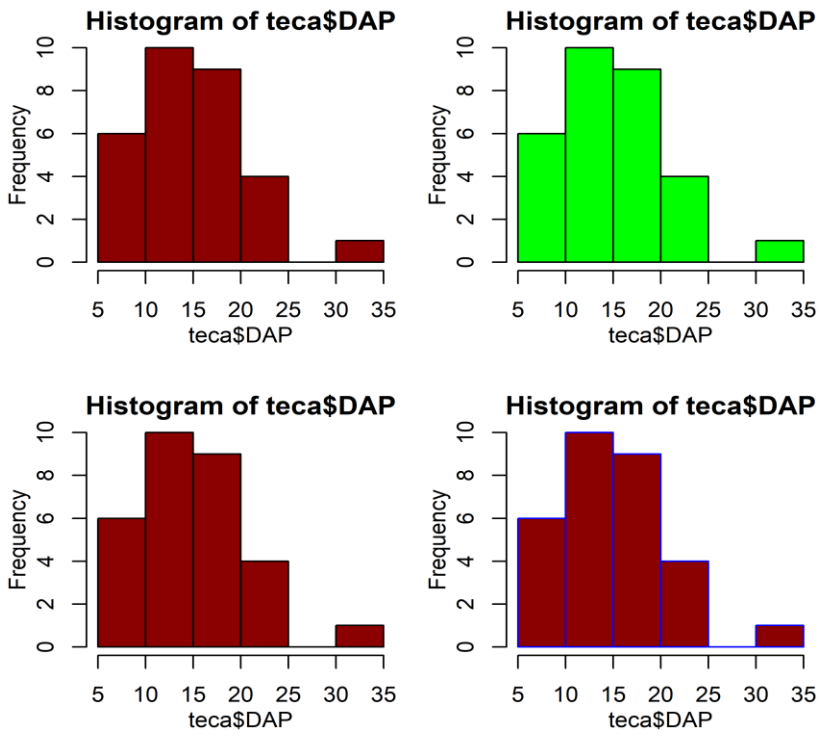
```
hist <- hist(teca$DAP)
```




```
hist$breaks                          # breakpoints (Regra de Sturges).
## [1]  5 10 15 20 25 30 35
hist$counts                          # frequência absoluta em cada classe.
## [1]  6 10  9  4  0  1
hist$mids                            # ponto médio das classes.
## [1]  7.5 12.5 17.5 22.5 27.5 32.5
hist$density
## [1] 0.040000000 0.066666667 0.060000000 0.026666667 0.000000000 0.006666667
```

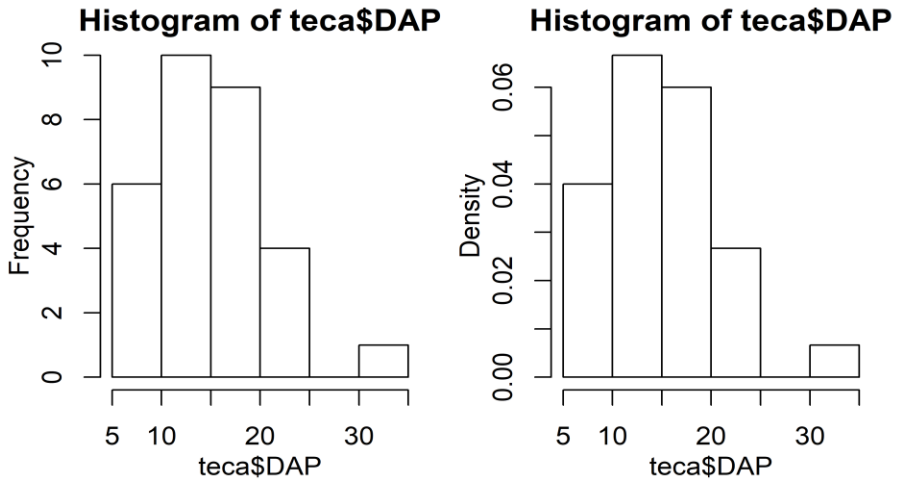
9.2.4 Modificando as cores das colunas e bordas (col e border)

```
hist(teca$DAP, col="red4")
hist(teca$DAP, col="green")
hist(teca$DAP, col="red4")
hist(teca$DAP, col="red4", border="blue")
```



9.2.5 Mostrando as probabilidades (density)

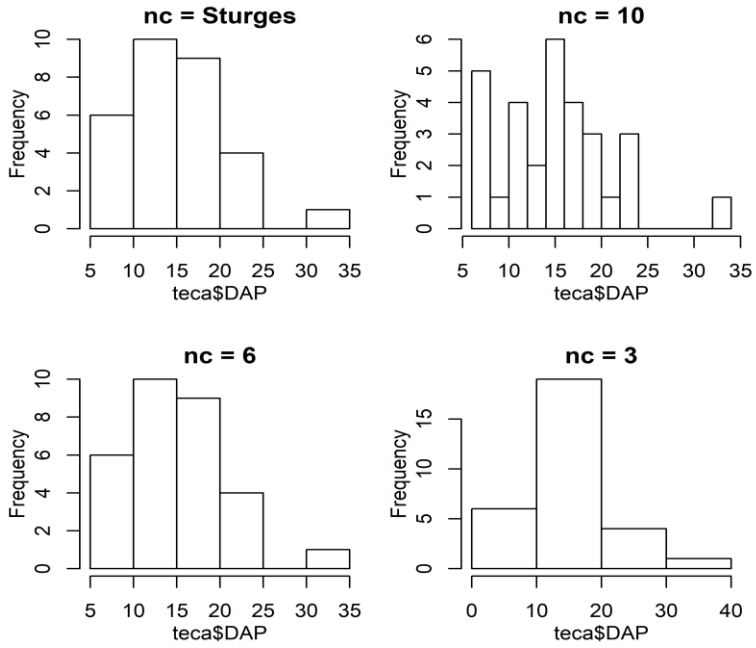
```
hist(teca$DAP)
hist(teca$DAP, probability = TRUE)          # density = 0 a 1
```



9.2.6 Número de classes (nc)

```
hist(teca$DAP, main = "nc = Sturges")
hist(teca$DAP, nc = 10, main = "nc = 10")
```

```
hist(teca$DAP, nc = 6, main = "nc = 6")
hist(teca$DAP, nc = 3, main = "nc = 3")
```



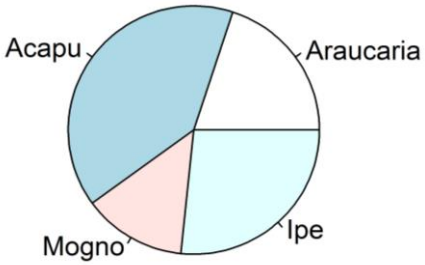
9.3 A função pie() (setores)

A função genérica pie() pode ser usada para construir um gráfico de setores. Para saber mais sobre a função faça: **?pie**. Os principais argumentos da função são: **labels** = vetor contendo os rótulos de cada fatia; **radius** = raio da circunferência do gráfico (*default*=1); e **col** = vetor contendo as cores de cada fatia. A seguir um gráfico de setores é construído a partir de um data frame com as contagens já disponíveis:

```
df <- data.frame(
  especie = c("Araucaria", "Acapu", "Mogno", "Ipe"),
  contagem = c(15, 30, 10, 20))
print(df)
```

especie	contagem
Araucaria	15.000
Acapu	30.000
Mogno	10.000
Ipe	20.000

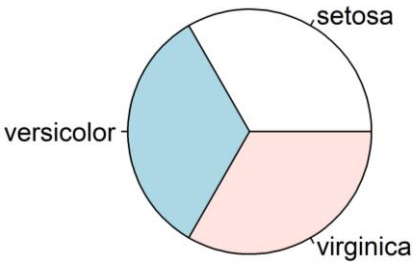
```
pie(df$contagem, labels = df$especie, main="")
```



Usando a função table()

Para obter contagens de fatores pode-se usar a função table(). Vamos obter uma tabela de contagem para o **conjunto de dados iris**. Quantas flores existem de cada espécie (setosa, virginica e versicolor)?

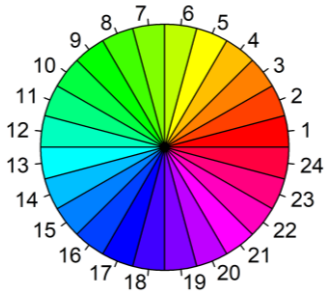
```
table(iris$Species)
##     setosa versicolor  virginica
##         50         50         50

pie(table(iris$Species))
```



9.3.1 Modificando cores (col)

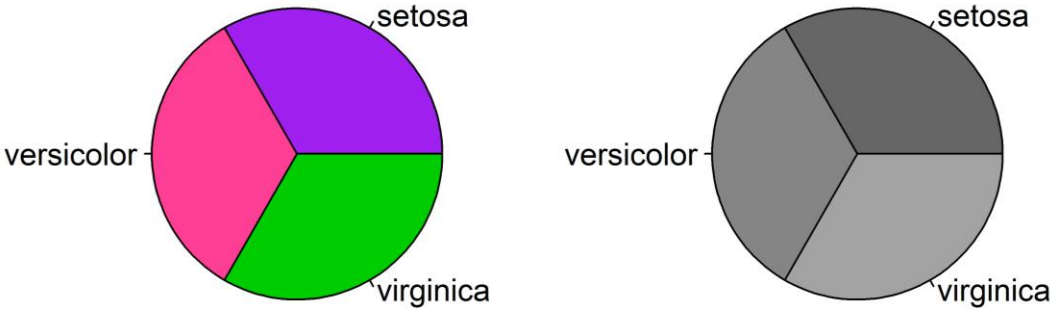
Um esquema de cores

```
pie(rep(1, 24), col = rainbow(24), radius = 1)
```



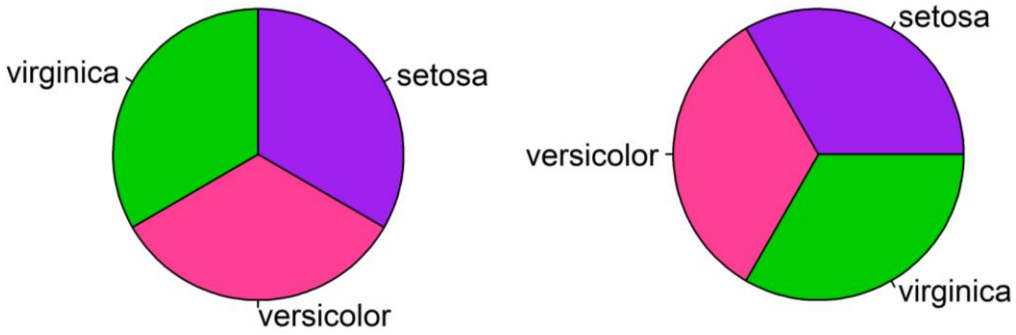
Modificando a cor das fatias

```
pie(table(iris$Species),col = c("purple", "violetred1", "green3"))
pie(table(iris$Species),col = gray(seq(0.4, 1.0, length = 6)))        #Níveis de cinza
```



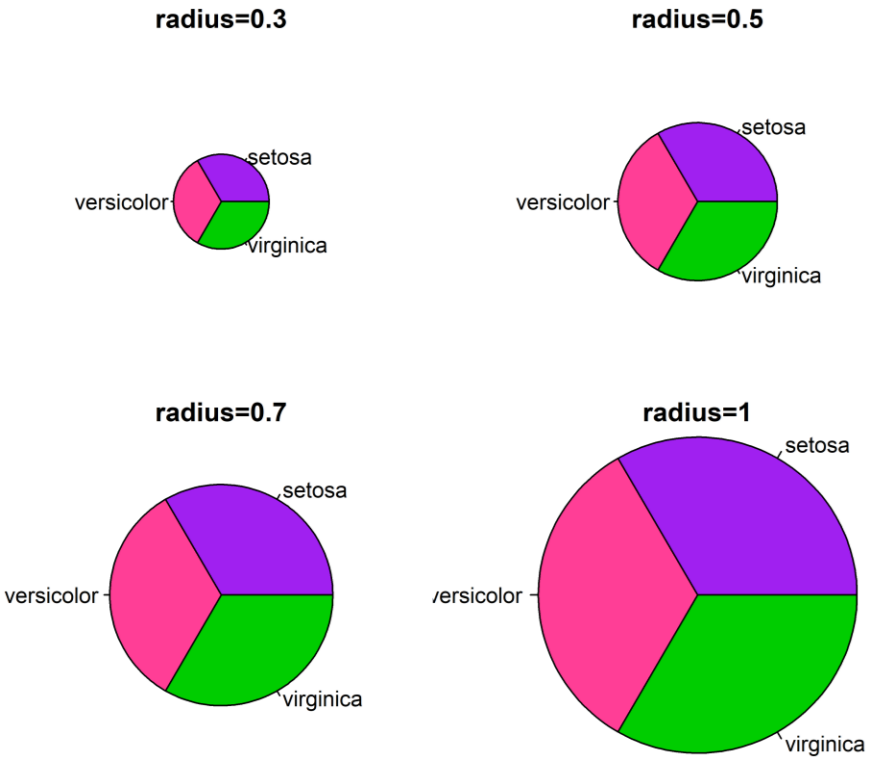
9.3.2 Modificando o sentido do desenho das fatias (horário e anti-horário)

```
pie(table(iris$Species), col = c("purple", "violetred1", "green3"), clockwise = TRUE)
pie(table(iris$Species), col = c("purple", "violetred1", "green3"), clockwise = FALSE)
```



9.3.3 Modificando o tamanho do gráfico (radius)

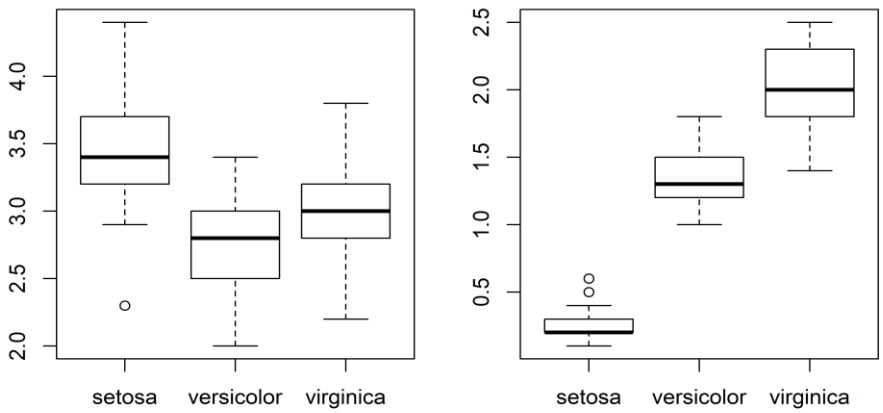
```
pie(table(iris$Species), radius=0.3, col = c("purple", "violetred1", "green3"), main = "radius=0.3")
pie(table(iris$Species), radius=0.5, col = c("purple", "violetred1", "green3"), main = "radius=0.5")
pie(table(iris$Species), radius=0.7, col = c("purple", "violetred1", "green3"), main = "radius=0.7")
pie(table(iris$Species), radius=1, col = c("purple", "violetred1", "green3"), main = "radius=1")
```



9.4 A função boxplot()

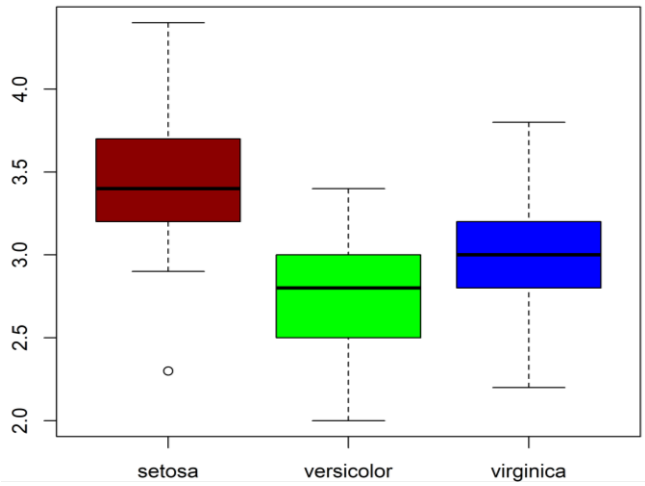
A função boxplot() pode ser usada para obter um gráfico de boxplot, também chamado de "gráfico de caixa". A seguir serão feitos gráficos boxplot para cada espécie de iris, para as variáveis 'Sepal.Width' e 'Petal.Width'. Os argumentos da função boxplot são inseridos y~x, diferente dos gráficos mostrados anteriormente, sendo x uma variável categórica.

```
boxplot(Sepal.Width ~ Species, data=iris)
boxplot(Petal.Width ~ Species, data=iris)
```



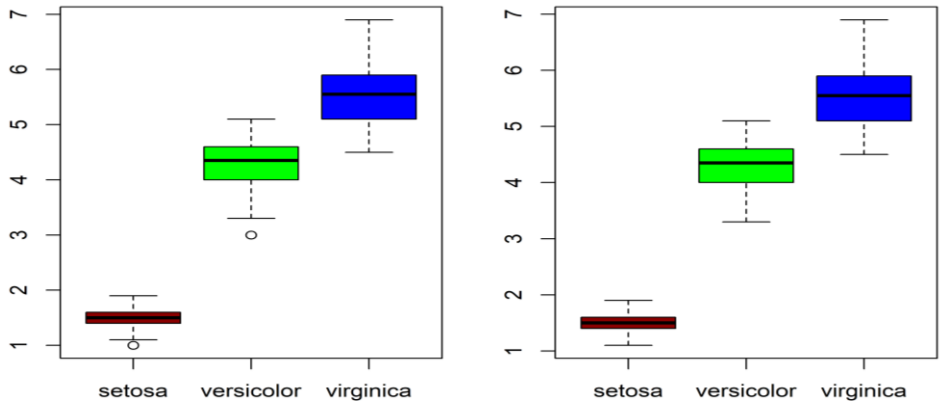
9.4.1 Modificando as cores das caixas (col)

```
boxplot(Sepal.Width~Species, data=iris, col=c("red4", "green", "blue"))
```



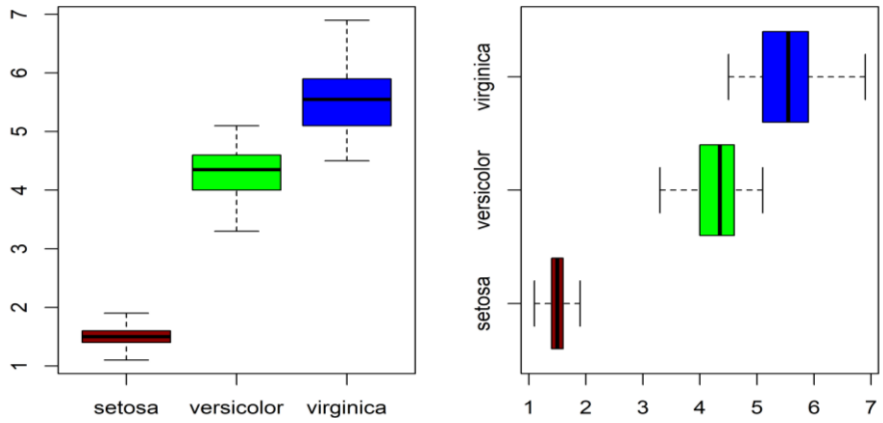
9.4.2 Plotando os outliers (outline)

```
boxplot(Petal.Length ~ Species, data=iris, col=c("red4", "green", "blue"), outline=TRUE)
boxplot(Petal.Length ~ Species, data=iris, col=c("red4", "green", "blue"), outline=FALSE)
```



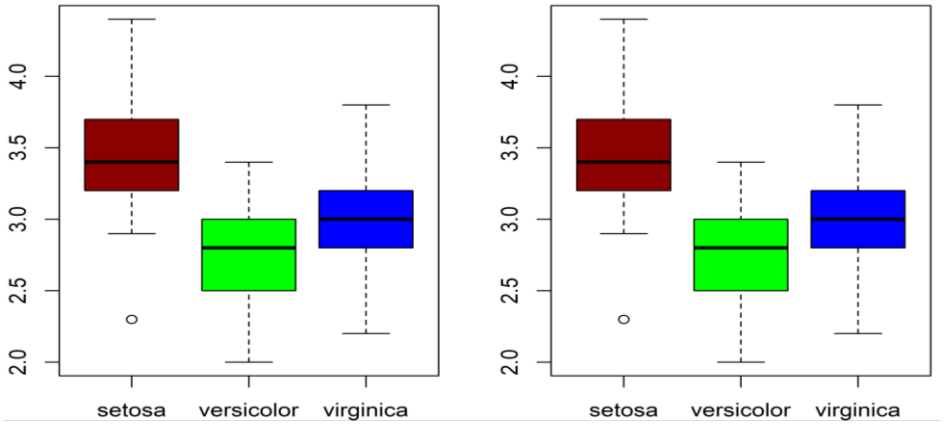
9.4.3 Alterando a orientação (horizontal)

```
boxplot(Petal.Length~Species, data=iris, col=c("red4", "green", "blue"),
      outline=FALSE, horizontal=FALSE)
boxplot(Petal.Length~Species, data=iris, col=c("red4", "green", "blue"),
      outline=FALSE, horizontal=TRUE)
```



9.4.4 Inserindo a variação dentro da caixa do boxplot (varwidth)

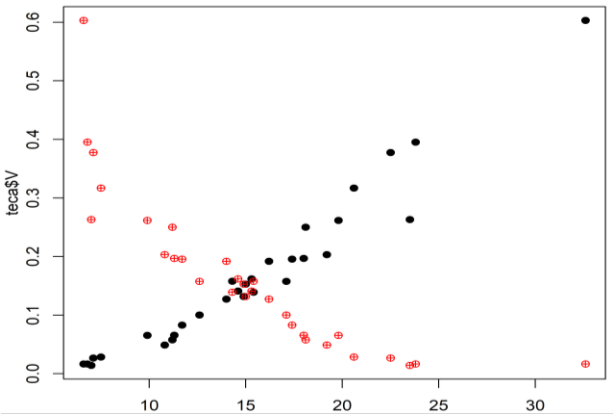
```
#O commando varwith( ) dimensiona o tamanho das caixas de acordo com o número de observações em cada f
ator. É um bom argumento para ser usado demostrar a variação de observações entre tratamentos, locais ou es
pécies....

boxplot(Sepal.Width~Species, data=iris, col=c("red4", "green", "blue"),
      outline=TRUE, varwidth=FALSE)
boxplot(Sepal.Width~Species, data=iris, col=c("red4", "green", "blue"),
      outline=TRUE, varwidth=TRUE)
```



9.5 Adicionando "elementos" a um gráfico existente

9.5.1 Adicionando pontos (points())

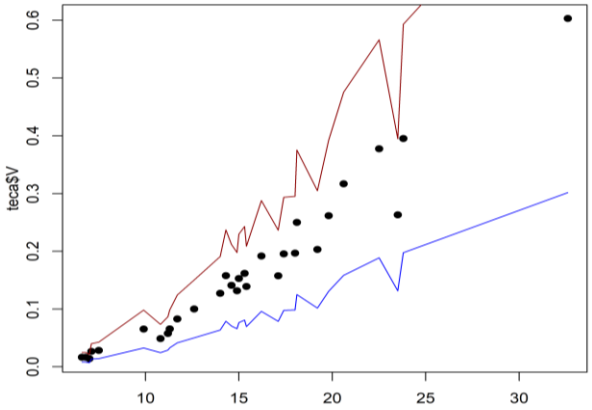
```
plot(teca$DAP, teca$V, pch=19)
points(rev(teca$DAP), teca$V, col="red", pch=10)                    # Reverso da variável DAP com o V
```



9.5.2 Adicionando linhas (lines())

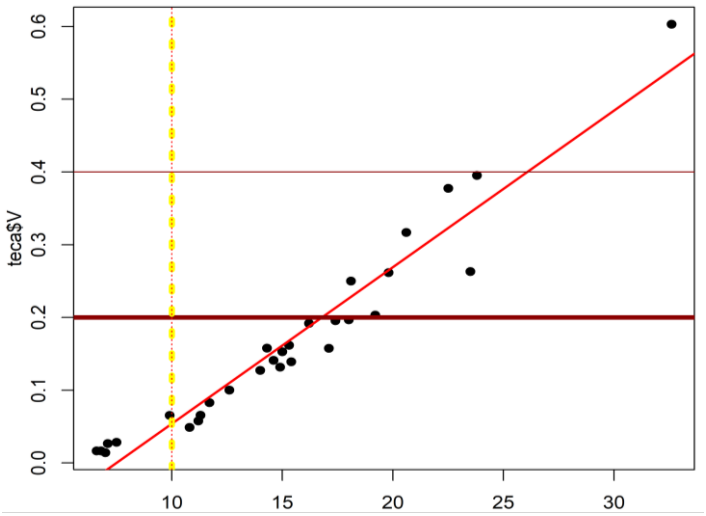
O comando lines() é muito usado para adicionar a linha de ajuste de regressão linear simples.

```
plot(teca$DAP, teca$V, pch=19)
lines(teca$DAP, teca$V*1.5, col="red4")
lines(teca$DAP, teca$V/2, col="blue")
```



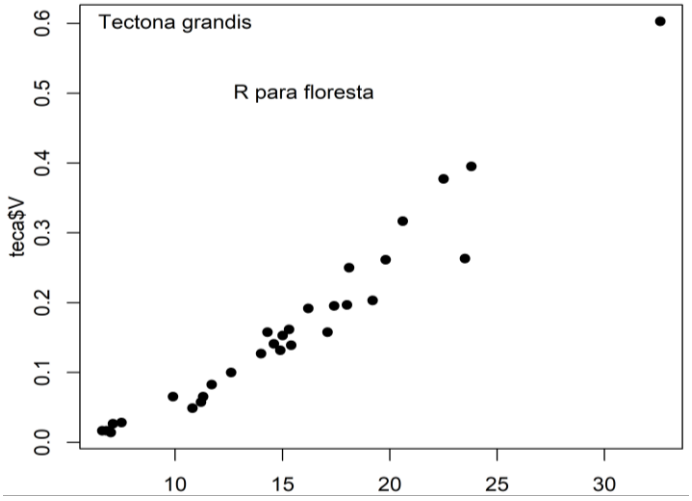
9.5.3 Adicionando retas (ablines())

```
plot(teca$DAP, teca$V, pch=19)

# x = intercepto e y = beta 1
abline(lm(teca$V ~ teca$DAP)$coefficients[1], lm(teca$V ~ teca$DAP)$coefficients[2],
      col="red", lwd=2, lty=1)

#A função "lm" será trabalhada mais adiante aqui ela foi usada apenas como ilustração

abline(h=.4, col="red4")
abline(h=.2, col="red4", lwd=4)   # Reta horizontal passando por y=200 na cor vermelha
abline(v=10, col="yellow", lty=3, lwd=5) # Reta vertical amarela x=10,  pontilhada e largura de 5
abline(v=10, col="red", lty=3, lwd=1)  # Ficou na frente da linha anterior
```



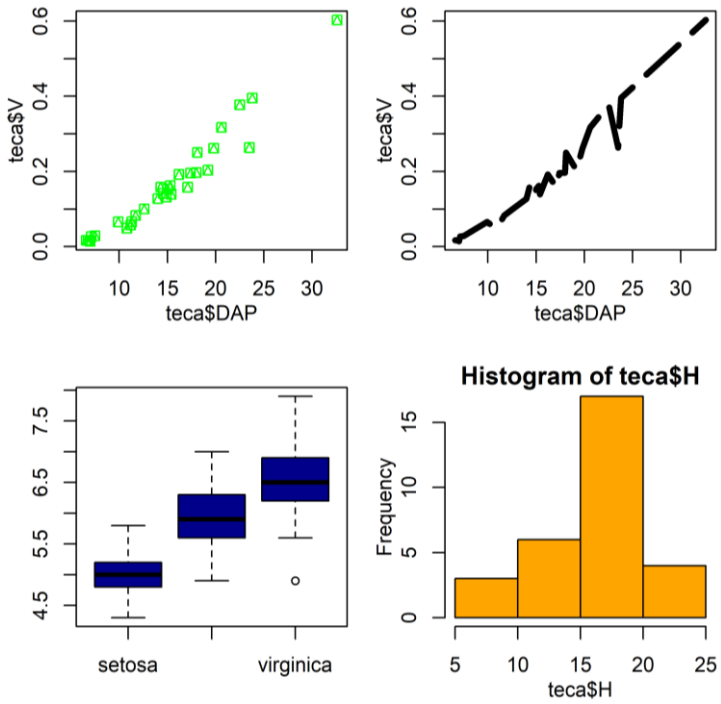
9.5.4 Adicionando texto (text())

```
plot(teca$DAP, teca$V, pch=19)
text(10,0.6, "Tectona grandis")
text(16,0.5, "R para floresta")
```

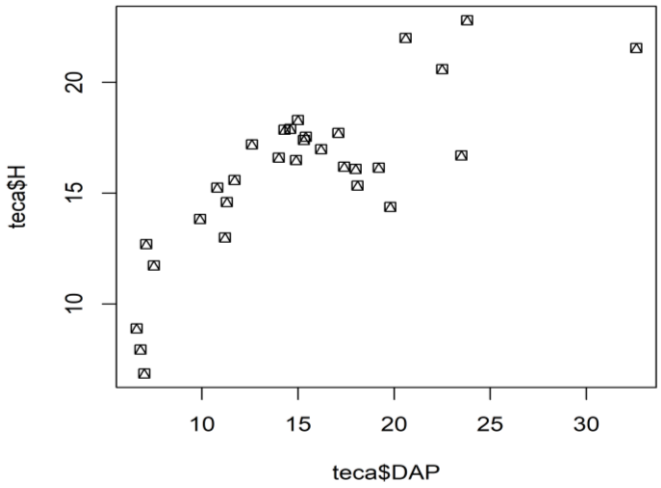


9.5.5 Múltiplos gráficos na mesma janela

```
# Duas linhas e duas colunas
op <- par(mfrow = c(2,2))  #Preenchimento por linha
plot(teca$DAP, teca$V, pch=14, col="green")
plot(teca$DAP, teca$V, type="l", lty=5, lwd=4)
boxplot(Sepal.Length ~ Species, data=iris, col="blue4")
hist(teca$H, nc=5, col= "orange")
par(op)  #Utiliza esse argumento para finalizar o comando par, ou seja, para que os próximos gráficos a sere
m feitos não apareçam em uma janela gráfica com vários gráficos
```

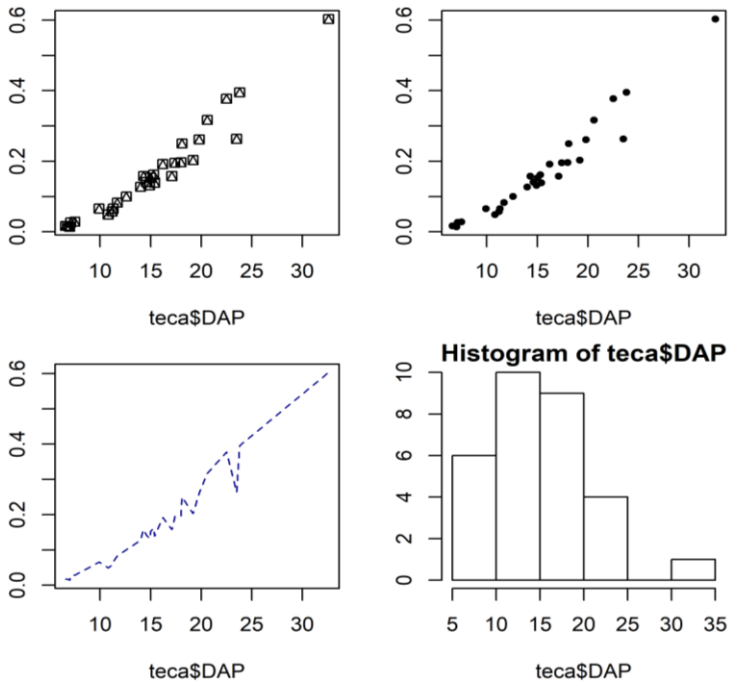



```
plot(teca$DAP, teca$H, pch=14)      #Janela gráfica com um gráfico apenas
```



9.5.6 Configurando as margens da janela gráfica

```
# Margens - mar(bottom, left, top, right)
op <- par(mfrow=c(2,2), mar = c(4.5,3,1,1))
plot(teca$DAP, teca$V, pch=14)
plot(teca$DAP, teca$V, pch=20)
plot(teca$DAP, teca$V, type="l", lty=2, col="blue4")
hist(teca$DAP, nc=5)
par(op)

#A unidade das margens são polegadas (inches)
```



10 Análise exploratória de dados

10.1 Configurando o diretório de trabalho

Um detalhe importante no início de cada seção de trabalho no ambiente R é inspecionar o seu diretório de trabalho atual, ou seja, o local onde serão salvos os arquivos produzidos pelo R (.R, etc.). Para tanto, basta usar a função getwd(). Caso não esteja no diretório desejado, use a função setwd(**choose.dir()**) para selecioná-lo. A função dir() é usada para listar os arquivos existentes no diretório corrente.

```
getwd()
setwd(choose.dir())
dir()
```

10.2 Importando um conjunto de dados

10.2.1 Funções read.csv() e read.table()

O R permite importar dados disponíveis em outros formatos, por exemplo, **.csv** (Comma-separated values) ou **.txt** (arquivo de texto). Ainda, é possível importar os dados de arquivos de outros programas estatísticos (SPSS, SAS, Stata). Duas funções extremamente úteis para realizar essa tarefa são read.csv() (importa arquivos no formato **.csv**) e read.table() (importar arquivos no formato **.txt**). Se os dados estiverem em planilha do Microsoft Office Excel, pode-se salvá-los com a

extensão **.txt** fazendo: salvar como → texto (separado por tabulações) (*.txt). A função read.table() possui diversos argumentos que devem ser cuidadosamente compreendidos. Alguns dos argumentos mais importantes são tratados a seguir:

Função read.table()

read.table(**file**, **header** = TRUE, **sep** = ";", **quote** = "", **dec** ="." , **as.is** = !stringsAsFactors, **na.strings** = c("NA"," ","NULL"), **encoding** ="unknown")

file = nome do arquivo que contém o conjunto de dados;

header = um valor lógico indicando se o arquivo contém os nomes das variáveis na primeira **linha**. Se header = TRUE informa que contém nomes;

quote = define células de texto. O *default* é interpretar tanto aspa simples como dupla presentes;

sep = separador de colunas usado no arquivo. O *default* é espaço;

dec = separador decimal usado no arquivo. O *default* é ponto (.);

as.is = o *default* da read.table() é converter variáveis de caracteres em fatores. Usar TRUE não fará a conversão;

na.strings = se definir, pode informar que símbolos em células inteiras sejam interpretados como valores ausentes (NA); e

encoding = codificação da acentuação. O *default* é 'unknown' (desconhecido), na qual ele reconhece segundo o sistema operacional. As opções mais usadas são Latin-1 ou UTF-8.

Os dados importados são coagidos a um data frame. Para entender melhor os argumentos da função read.table(), iremos importar o arquivo nomeado **arvores.txt** (deve estar no diretório corrente) e atribuiremos o nome **nativas** ao objeto. Do arquivo original **arvores.txt** tem-se os seguintes formatos:

1. As colunas estão separadas por tabulação (no R, isso é definido pela expressão regular "\t");
2. O separador decimal usado no arquivo é ponto (.);
3. Não tem aspas definindo as colunas de texto; e
4. O arquivo contém os nomes das variáveis na primeira linha.

O que aconteceria se especificasse errôneamente o separador de colunas usado no arquivo, cujo default é espaço? No exemplo, usa-se o separador vírgula (,) para fins de exemplificação. Fazendo isso, as duas colunas existentes são interpretadas como uma só.

```
nativas <- read.table(file = "arvores.txt", sep=",", header=T)
dim(nativas)                                  # apenas 1 coluna, porque o separador não está correto.
## [1] 25  1
```

Então, para ler o arquivo **arvores.txt** corretamente deve-se fazer:

```
nativas <- read.table("arvores.txt", header = T, sep="")      # Carrega corretamente
print(nativas)                                                # Imprime o data frame
```

Arvores	Altura
1	18.000
2	15.000
3	19.000
4	12.000
5	14.000
6	14.000
7	17.000
8	21.000
9	20.500
10	23.000
11	19.000
12	21.000
13	18.000
14	16.000
15	12.000
16	14.000
17	15.500
18	19.500
19	22.000
20	20.300
21	24.000
22	17.000
23	14.000
24	18.000
25	21.000

```
summary(nativas)                                  # Resumo estatístico dos dados

##     Arvores          Altura
##  Min.   : 1       Min.   :12.00
##  1st Qu.: 7       1st Qu.:15.00
##  Median :13       Median :18.00
##  Mean   :13       Mean   :17.79
##  3rd Qu.:19       3rd Qu.:20.50
##  Max.   :25       Max.   :24.00

class(nativas)                                    # classe do objeto
## [1] "data.frame"
dim(nativas)                                      # dimensão do objeto
## [1] 25  2
```

Um aspecto importante na análise exploratória é verificar se o conjunto de dados possui valores faltantes. Para tanto, pode-se usar a função lógica is.na():

```
is.na(nativas)                    # possui valores faltantes?
```

10.2.2 Função read.xlsx()

Existem diversos pacotes com funções dedicadas à importação de conjuntos de dados para o ambiente R. Um exemplo é o pacote **xlsx** (DRAGULESCU, 2014) desenvolvido para permitir a leitura de dados diretamente de planilhas do Microsoft Office Excel usando a função read.xlsx(). Inicialmente é preciso realizar a instalação: install.packages(**"xlsx"**). Em seguida, usar-se-á a função para importar o arquivo **guanandi.xlsx** que contém quatro variáveis: número da muda, dcolo (diâmetro do colo), h (altura) e número de folhas.

```
library(xlsx)                                    # Carrega o pacote
# Usando read.xlsx(): sheetName = Nome arquivo
guanandi <- read.xlsx(file = "guanandi.xlsx", sheetName = "guanandi", head = TRUE)
print(guanandi)
```

muda	dcolo	h	folhas
1.000	1.700	20.000	6.000
2.000	2.400	18.000	9.000
3.000	3.500	17.500	10.000
4.000	3.000	17.000	11.000
5.000	3.200	19.500	4.000
6.000	2.900	20.100	5.000
7.000	1.400	19.000	6.000
8.000	0.900	20.200	7.000
9.000	3.100	18.600	9.000
10.000	3.600	17.400	10.000
11.000	2.200	16.000	8.000
12.000	1.400	17.900	11.000
13.000	1.600	18.100	12.000
14.000	1.000	19.000	6.000
15.000	2.800	21.000	4.000
16.000	1.600	20.600	9.000
17.000	1.900	22.000	8.000
18.000	2.300	17.400	11.000
19.000	2.700	16.900	10.000
20.000	3.000	16.300	10.000
21.000	3.100	19.500	5.000
22.000	2.700	18.400	7.000
23.000	1.900	20.600	9.000
24.000	2.200	21.100	10.000
25.000	3.000	20.000	9.000

10.3 Estatística descritiva

É a parte da estatística que descreve e avalia um conjunto de dados, sejam populacionais ou oriundos de uma amostra representativa da população. Vale destacar que o simples uso de estatísticas descritivas não nos permite conclusões e/ou inferências robustas sobre uma determinada população. No entanto, nos fornece informações iniciais relevantes sobre a estrutura dos dados e auxilia na tomada de decisão a respeito dos tratamentos e métodos estatísticos adequados. Na estatística descritiva, existem dois métodos comuns de representação dos dados:

a) método gráfico (gráficos e tabelas); e

b) método numérico (medidas de tendência central, dispersão, entre outros).

As medidas de tendência central incluem: **média aritmética, moda, mediana**, entre outras. As medidas de dispersão incluem: **amplitude total, variância, desvio padrão e coeficiente de variação**.

10.3.1 Medidas de tendência central

1. Média aritmética: Matematicamente, é a soma de todos os elementos de uma amostra ou população dividida pela quantidade de elementos

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

A média aritmética é a medida de tendência central mais conhecida e pode ser facilmente obtida no R usando a função mean():

```
x <- c(1, 5, 7, 18, 32, 10, 6)                    # criando um vetor
mean(x)                                           # obtendo a média
## [1] 11.28571
```

Há situações em que, no conjunto de dados, constam valores ausentes (**NA**). Quando isso ocorre, deve-se usar o argumento deve-se usar o argumento **na.rm = TRUE** para que o cálculo da média seja realizado desconsiderando o valor desconhecido **NA**.

```
w <- 1:5                                          # cria um vetor de sequência de 1 a 5
w[2] <- NA                                        # substitui o elemento da posição 2 por NA
print(w)                                          # imprime o vetor com NA
## [1]  1 NA  3  4  5

mean(w)                                           # cálculo inapropriado da média
## [1] NA
```

```
mean(w, na.rm = TRUE)              # cálculo apropriado da média
## [1] 3.25
```

2. Mediana: corresponde ao valor intermediário de um conjunto de dados, quando os **n** valores são dispostos de maneira ordenada (rank). De outro modo, é o valor que divide um conjunto de dados em duas partes iguais, em que 50% dos dados ordenados estão acima da mediana e 50% estão abaixo da mediana. O uso da mediana é indicado quando o conjunto de dados possui valores extremos (outliers), pois o uso da média, nesses casos, pode ser viesado. Existem duas regras básicas para descobrir a posição da mediana nos dados ordenados a partir do número de observações (n) do conjunto de dados:

- **Quando n for ímpar:** a mediana será igual à (n+1)/2;
- **Quando n for par:** a mediana será igual à média aritmética dos valores que ocupam as posições n/2 e n/2+1.

O ambiente R possui a função median() que retorna a mediana de um conjunto de dados sem a necessidade de realizar o procedimento de ordenamento para encontrar a posição da medida:

```
x <- c(1, 5, 7, 18, 32, 10, 6)                  # vetor x não ordenado (n=7)
y <- c(1, 5, 6, 7, 10, 18, 32)                  # vetor y ordenado (n=7)

median(x)                                       # a mediana de x?
## [1] 7
median(y)                                       # a mediana de y?
## [1] 7
```

3. Moda: Refere-se ao valor mais frequente dentro do conjunto de dados. Um conjunto de dados pode ser classificado quanto à moda em: a) amodal; b) unimodal; c) bimodal; e d) multimodal.

a) **Amodal:** quando não possui moda. Isso ocorre quando não existe um valor mais frequente;

b) **Unimodal:** quando possui apenas uma moda (um valor mais frequente);

c) **Bimodal:** quando possui duas modas; e

d) **Multimodal:** quando possui três ou mais modas.

Por não ser uma medida de posição muito usual, no R-base não existe uma função específica para calcular a **moda**. Então, para obter a moda utilizaremos a

função mfv() (*Most frequent value*) do pacote **modeest** (PONCET, 2012). No entanto, a função mfv() retorna a moda apenas para vetores numéricos. Quando o interesse é saber a moda para um **vetor de character** (variável nominal) pode-se usar a função modeOf() do pacote **lsr** (NAVARRO, 2015). De modo contrário da função mfv(), a função modeOf() funciona para vetores numéricos e vetores de caracteres.

```
install.packages("modeest")
library(modeest)
v <- seq(from = 1, to = 10, by = 1)                    # amodal (todos valores)
x <- c(0, 1, 2, 3, 4, 5, 5, 5, 5, 6, 6, 7, 7, 8)       # unimodal
y <- c(0, 1, 2, 3, 4, 5, 5, 5, 6, 6, 7, 7, 7)          # bimodal
z <- c(0, 1, 2, 3, 4, 5, 5, 5, 6, 6, 6, 7, 7, 7)       # multimodal
mfv(v)                                                 # moda do vetor v?
##  [1]  1  2  3  4  5  6  7  8  9 10
mfv(x)                                                 # moda do vetor x?
## [1] 5
mfv(y)                                                 # moda do vetor y?
## [1] 5 7
mfv(z)                                                 # moda do vetor z?
## [1] 5 6 7
```

A moda também pode ser obtida a partir da função table(), já apresentada na construção de gráficos de setores.

```
x <- c(0, 1, 2, 3, 4, 5, 5, 5, 5, 6, 6, 7, 7, 8)
table(x)
## x
## 0 1 2 3 4 5 6 7 8
## 1 1 1 1 1 4 2 2 1
```

Veja uma situação para um vetor de caracteres (variável nominal):

```
install.packages("lsr")
library(lsr)
especie <- c(rep("Araucaria", 20), rep("Acapu", 10),  rep("Mogno", 5), rep("Ipe", 2))

class(especie)
## [1] "character"

modeOf(especie)
## [1] "Araucaria"

table(especie)
## especie
##    Acapu   Araucaria      Ipe    Mogno
##       10          20        2        5
```

Para finalizar, apresentamos um exemplo com todas as medidas de tendência central estudadas:

```
x <- c(20, 7, 5, 9, 6, 21, 24, 10, 12, 22, 21, 16, 13, 6, 6, 2, 19, 3, 10, 7, 2, 18, 4, 6, 18)
mean(x)                                          # obtendo a média
## [1] 11.48

median(x)                                        # obtendo a mediana
## [1] 10

mfv(x)                                           # obtendo a moda (pacote modeest)
## [1] 6

table(x)                                         # frequências de cada valor
## x
##  2  3  4  5  6  7  9 10 12 13 16 18 19 20 21 22 24
##  2  1  1  1  4  2  1  2  1  1  1  2  1  1  2  1  1
```

10.3.2 Medidas de dispersão

São medidas que descrevem a dispersão ou variabilidade dos dados em relação à média ($\bar{x}$), ou seja, indicam o quanto os dados estão dispersos em torno da região central. As principais medidas de dispersão incluem: **amplitude total (AT), variância (s^2), desvio padrão (s) e coeficiente de variação (CV)**.

1. Amplitude total: Refere-se à diferença entre o maior (máximo) e o menor (mínimo) valor de um conjunto de dados. A Amplitude total (AT) não é uma medida robusta para verificar a variabilidade de um conjunto de dados, pois considera apenas os valores extremos na distibuição amostral. No entanto, serve para indicar se existe ou não dispersão no conjunto, dando uma ideia inicial da exixtência ou não de variabilidade. No ambiente R pode-se obter a amplitude total indiretamente (AT = máximo - mínimo). Entretanto, a função range() retorna os valores mínimos e máximos, permitindo, em seguida, o cálculo da AT.

```
t <- c(2, 4, 5, 6, 10)                           # um vetor de inteiros

max(t)-min(t)                                    # cálculo indireto da AT

## [1] 8

range(t)                                         # mostra o min(t) e o max(t)

## [1]  2 10

range(t)[2]-range(t)[1]                          # outra maneira

## [1] 8
```

2. Variância: A variância de uma amostra de **n** elementos é definida como a soma ao quadrado dos desvios dos elementos em relação à média, dividido pelo grau de liberdade. No ambiente R pode-se usar a função var() para calcular a variância. A variância amostral é dada por:

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1}$$

```
z <- c(1, 2, 3, 4, 5)                    # um vetor de inteiros
var(z)                                   # variância de z
## [1] 2.5
```

No ambiente R, também é possível obter o valor da variância amostral a partir da fórmula dessa medida de dispersão:

```
#Soma de Quadrados dos Desvios, dividido pelo grau de liberdade (n-1)

sum((z-mean(z))^2)/(length(z)-1)
## [1] 2.5
```

3. Desvio Padrão: corresponde ao valor da raiz quadrada da variância amostral. O valor do desvio-padrão diferencia-se da variância uma vez que é representado na mesma unidade de medida dos dados originais. No ambiente R pode-se usar a função sd() (*standard deviation*) para calcular o desvio padrão ou, simplesmente, extrair a a raiz quadrada da variância. O desvio padrão amostral é dado por:

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1}}$$

```
x <- c(1, 2, 3, 4, 5)                    # um vetor de inteiros

sqrt(var(x))                             # cálculo da raiz quadrada da variância
## [1] 1.581139
sd(x)                                    # usando a função sd()
## [1] 1.581139
```

4. Coeficiente de variação: corresponde a uma medida relativa da dispersão, obtida em função do desvio padrão e da média aritmética amostral. No R-base não existe uma função específica para calcular o CV. Porém, seu valor pode ser facilmente obtido dividindo o **desvio padrão pela média aritmética amostral** (desvio padrão/média). Em seguida, deve-se multiplicar por 100 para obter a medida em percentual (%). Ainda, pode-se usar a função coefficient.variation() do pacote **FinCal** (FAN, 2016).

```
library(FinCal)
z <- c(1, 2, 3, 4, 5)                    # um vetor de inteiros
```

```
sd(z)/mean(z)*100                                    # (desvio padrão/média)*100
## [1] 52.70463

coefficient.variation(sd=sd(z), avg=mean(z))*100   # usando a função coefficient.variation()
## [1] 52.70463
```

Outras medidas de dispersão podem ser calculadas no R, como é o caso do erro-padrão da média (precisão da média). Sua fórmula é a seguinte: desvio padrão/raiz (n); ou raiz quadrada (variância/n). Médias com menor erro-padrão são consideradas mais precisas.

10.3.3 Exercício prático: análise descritiva dos dados

Para aprimorar os conhecimentos sobre análise descritiva de dados, utilizar-se-á o conjunto de dados **iris**. Inicialmente, use as funções do R-base detalhadas anteriormente e, somente depois use o pacote **fBasics** para obter as estatísticas descritivas para cada variável. Responda as questões a seguir:

a) Qual a média aritmética das quatro variáveis numéricas?

b) Qual a mediana das quatro variáveis numéricas?

c) Qual a moda das quatro variáveis numéricas?

d) Qual a amplitude total das quatro variáveis numéricas?

e) Qual a variância amostral das quatro variáveis numéricas?

f) Qual a desvio padrão amostral das quatro variáveis numéricas?

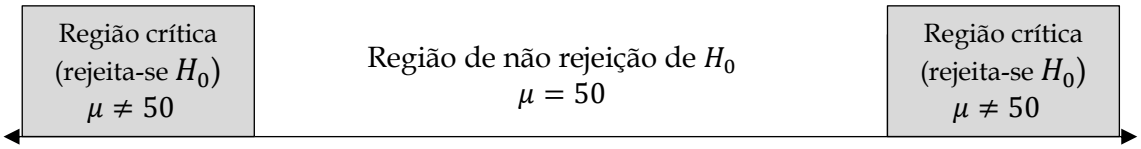
g) Qual o coeficiente de variação das quatro variáveis numéricas?

h) Use o pacote **fBasics** e obtenha as estatísticas descritivas para cada variável do conjunto iris.

```
# Exemplo de aplicação do pacote fbasics

install.packages("FinCal")
library(FinCal)
iris                                                 # carrega o conjunto "íris"
iris[1:4]                                            # seleciona apenas as colunas 1 à 4 (numéricas)

colStats(iris[1:4], FUN=median)                      # cálculo das medianas por colunas
colStats(iris[1:4], FUN=mean)                        # cálculo das médias por colunas

# Para saber mais sobre o pacote, digite o comando ?fBasics
```

11 Teste de hipóteses

Na análise de experimentos, a realização de **testes de hipótese** é fundamental para a tomada de decisão acerca da aceitação ou rejeição de alguma **hipótese experimental** (MONTGOMERY & RUNGER, 2003). Em outras palavras, os testes de hipótese são aplicados para investigar a **nulidade da hipótese estatística formulada**. A chamada **região crítica** para o teste consiste no intervalo da curva de distribuição de probabilidade cujos valores estão fora da região de aceitação de H_0 (MONTGOMERY & RUNGER, 2003). Ou seja, caso μ corresponda a um valor que ocorre dentro da região crítica, a hipótese nula será rejeitada. A maioria dos testes considera 5% de probabilidade para a região crítica, sendo esse valor conhecido como nível de significância do teste (α). Para esse nível de significância, valores de $p < 0{,}05$ conduzem à rejeição da hipótese nula H_0. A seguir, buscou-se representar as regiões de rejeição e não rejeição para testes bilaterais, nos quais se considera regiões críticas nos dois extremos da distribuição (para H_0: $\mu = 50$ e H_1: $\mu \neq 50$).



Em trabalhos científicos atuais têm sido mais usual relatar o valor de α (probabilidade) em complemento à constatação de rejeição ou não de H_0, de acordo com o nível de significância fixado (α). Para a aplicação de um teste de hipóteses, os procedimentos gerais consistem em:

- Formulação das hipóteses H_0 e H_1 (hipóteses do experimento) e definição do nível de significância (α);
- Determinação do valor da estatística do teste (por exemplo: t, Z, F), de acordo com o nível de significância;
- Cálculo do *p-valor* correspondente; e
- Comparação do *p-valor* com α (alfa).

Interpretação dos resultados:

- Se *p-valor* for menor ou igual à α, rejeita-se H_0.
- Se *p-valor* for maior que α, não rejeita-se H_0.

11.1 Teste t-Student

Teste de hipóteses utilizado para rejeitar ou não uma hipótese nula quando os dados seguem uma distribuição t de Student (STUDENT, 1908). Esse teste é comumente aplicado quando se deseja comparar, estatisticamente, as médias de

duas amostras. Por exemplo, deseja-se testar a hipótese nula H_0: $\bar{x}_1 = \bar{x}_2$ versus uma hipótese alternativa $\bar{x}_1 > \bar{x}_2$, ou H_0: $\bar{x}_1 = \bar{x}_2$ versus a hipótese alternativa $\bar{x}_1 \neq \bar{x}_2$. Nesse caso, a estatística do teste deve ser obtida com base na média amostral. Caso **x** apresente uma distribuição normal com valores de $\bar{x}$ (média) e s^2 (variância) conhecidos, o valor de t será:

$$t = \frac{\bar{x} - \mu}{s/\sqrt{n}}$$

Regra de decisão:

- Se $t_{calculado} \geq t_{tabelado}$: rejeita-se a hipótese H_0.
- Se $t_{calculado} < t_{tabelado}$: não rejeita-se a hipótese H_0.

O valor de t ($t_{tabelado}$) pode ser consultado na tabela de Distribuições t de Student, variando de acordo com o nível de significância e graus de liberdade. Entretanto, esse valor também pode ser obtido no próprio R, usando a função pt(), de acordo com o nível de significância (α) definido. Para a aplicação do teste t bilateral, deve-se dividir o valor de α por 2: $t_{tabelado} = t(\alpha/2; n-1)$.

A seguir, considere os diâmetros de árvores oriundos de diferentes amostras:

```
dap1 <- c(30.5, 35.3, 33.2, 40.8, 42.3, 41.5, 36.3, 43.2, 34.6, 38.5)

dap2 <- c(28.2, 35.1, 33.2, 35.6, 40.2, 37.4, 34.2, 42.1, 30.5, 38.4)
```

11.1.1 Teste t para uma média

Aqui, testar-se-á se "dap1" tem média igual à 35. Então, para um *p-valor* = 0,05, as hipóteses do teste são:

- H_0: $\overline{dap} = 35$
- H_1: $\overline{dap} \neq 35$

O ambiente R dispõe da função t.test() para aplicação do teste t-Student.

```
t.test(dap1,                          # amostra a ser testada
    mu=35,                            # hipótese de nulidade
    alternativa = "greater")          # teste unilateral à direita
##
##  One Sample t-test
##
## data:  dap1
## t = 1.9323, df = 9, p-value = 0.08535
## alternative hypothesis: true mean is not equal to 35
## 95 percent confidence interval:
##  34.5528 40.6872
```

```
## sample estimates:
## mean of x
##    37.62
```

Interpretação do resultado (valor de *t* e *p-valor*): $t_{calculado} < t_{tabelado}$; *p-valor* > 0,05. Não rejeita-se H_0, ou seja, $\overline{dap} = 35$.

11.1.2 Teste t para as médias de duas amostras independentes

Agora, presumir-se-á que as amostras são oriundas de duas populações diferentes e que ambas possuem variâncias homogêneas e distribuição normal. Suponha que o interesse seja testar se as amostras (dap1 e dap2) possuem médias estatisticamente iguais (α = 1%), considerando que ambas são independentes e oriundas de distribuição normal.

As hipóteses do teste são:

- H_0: $\overline{dap1} = \overline{dap2}$
- H_1: $\overline{dap1} \neq \overline{dap2}$

A função t.test() pode ser usada novamente:

```
t.test(dap1, dap2,                          # amostras a serem comparadas
    conf.level = 0.99)                      # nível de significância
##  Welch Two Sample t-test
## data:  dap1 and dap2
## t = 1.1148, df = 17.999, p-value = 0.2796
## alternative hypothesis: true difference in means is not equal to 0
## 99 percent confidence interval:
##  -3.369829  7.629829
## sample estimates:
## mean of x mean of y
##    37.62     35.49
```

Interpretação do resultado: p-*valor* > 0,01, não se rejeita a hipótese H_0. Ou seja, as amostras pertencem a populações com mesma média.

11.1.3 Teste t para médias de duas amostras dependentes

É aplicado quando os **n** valores de duas amostras são coletados dos mesmos indivíduos (por exemplo, das mesmas árvores), caracterizando uma dependência entre as amostras. Nesses casos, dizemos que os dados são pareados e, para testar as hipóteses, utiliza-se o **teste *t*-pareado**. Assim, para o mesmo conjunto de dados (dap1 e dap2), com nível de significância de 1%, considere as seguintes hipóteses:

- H_0: $\overline{dap1} = \overline{dap2}$

- H_1: $\overline{dap1} \neq \overline{dap2}$

Novamente pode-se usar a função t.test(), com adição do argumento **paired=TRUE** para informar que as amostras são pareadas (dependentes).

```
t.test(dap1, dap2,
       conf.level = 0.99,                    # nível de significância
       paired = TRUE)                        # afirma dependência entre as amostras
##
##  Paired t-test
##
## data:  dap1 and dap2
## t = 3.6493, df = 9, p-value = 0.005323
## alternative hypothesis: true difference in means is not equal to 0
## 99 percent confidence interval:
##  0.2331487 4.0268513
## sample estimates:
## mean of the differences
##                    2.13
```

Interpretação do resultado: p-*valor* < 0,01. Rejeita-se H_0, ou seja, $\overline{dap1} \neq \overline{dap2}$.

12 Aplicações Florestais

12.1 Análise de Variância (ANOVA)

A Análise de Variância (ANOVA) é o procedimento que permite decompor a variação total existente no experimento em variação devido à diferença entre efeitos dos tratamentos (fatores controlados) e em variação devido ao acaso (erro experimental ou resíduo) (FISHER, 1921). De outro modo, é o procedimento estatístico utilizado para testar a hipótese de nulidade para **mais de duas médias**, baseado na análise das variâncias.

VARIAÇÃO TOTAL = **EFEITO DOS TRATAMENTOS** (fatores controlados) + **EFEITO DO ERRO EXPERIMENTAL OU RESÍDUO** (fatores não-controlados)

Apesar do nome, é um teste de comparação de médias onde os dados amostrais são separados em grupos, segundo características específicas (fatores). Variam de acordo com os diferentes tipos de delineamentos experimentais (GOMES, 2000), sendo os mais usuais:

a) DIC (Delineamento Inteiramente Casualizado);
b) DBC (Delineamento em Blocos Casualizados); e
c) DQL (Delineamento em Quadrado Latino).

O Teste F (Fisher-Snedecor) é usado para testar a ANOVA de um DIC. As hipóteses H_0 e H_1 são:

- H_0: todas as médias dos tratamentos são iguais entre si
- H_1: há pelo menos dois tratamentos cujas médias são diferentes entre si

Regra de decisão:

- Rejeita-se H_0: Se $F_{calculado} > F_{tabelado}$ (há pelo menos dois tratamentos com médias diferentes).
- Não rejeita-se H_0: Se $F_{calculado} \leq F_{tabelado}$ (todas as médias são estatisticamente iguais entre si).

12.1.1 Delineamento Inteiramente Casualizado (DIC)

O DIC é um dos delineamentos experimentais mais utilizados nas ciências florestais, sendo que, para ser aplicado, devem-se seguir os princípios básicos de repetição e casualização (SILVA & SILVA, 1982). Além disso, no DIC os tratamentos experimentais devem ser distribuídos às unidades experimentais de forma completamente aleatória (por sorteio), possuindo flexibilidade quanto ao número de repetições por tratamento. É recomendado quando podem ser dadas condições homogêneas ou uniformes para todas as unidades experimentais (SILVA & SILVA, 1982).

Para fins de aplicação prática no R, considere a necessidade de estudos relativos ao comportamento da espécie *Schizolobium parahyba* var. *amazonicum* em viveiro. Assim, procedeu-se a instalação de um experimento sob o DIC com intuito de avaliar o crescimento em altura da espécie sob diferentes condições de sombreamento. Para tanto, foram estabelecidos **quatro tratamentos** (pleno sol, 20%, 50% e 70% de sombreamento) com **cinco repetições** de 25 sementes. Os dados estão no arquivo **parica.txt** e correspondem aos valores médios da altura, em centímetros, obtidos após trinta dias de germinadas as sementes.

```
parica <- read.table(file = "parica.txt", sep="", header=T)
print(parica)
```

Trat	Rep
T1	20.000
T1	22.500
T1	25.000
T1	23.500
T1	17.500
T2	23.500
T2	27.500
T2	25.000
T2	26.000
T2	23.500

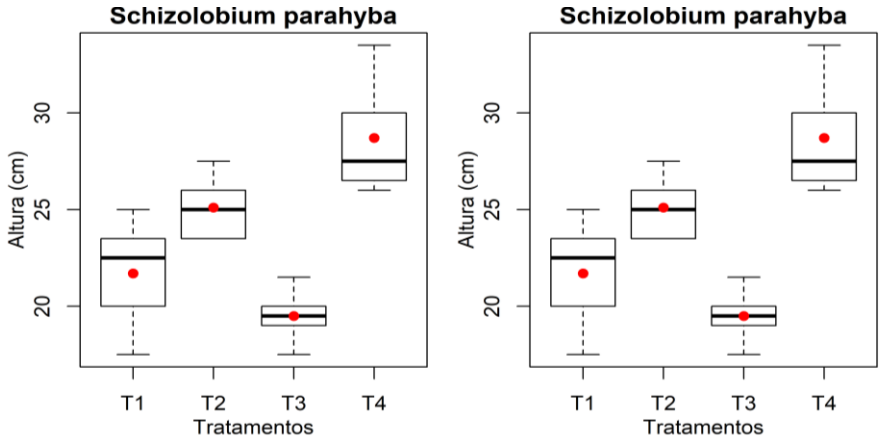
Trat	Rep
T3	17.500
T3	19.000
T3	19.500
T3	20.000
T3	21.500
T4	30.000
T4	33.500
T4	27.500
T4	26.000
T4	26.500

Inicialmente, vamos fazer uma análise exploratória dos dados. Dentre outras coisas, é importante ter conhecimento da média e da variância de cada tratamento:

```
summary(parica)                                  # um resumo estatístico
##  Trat       Rep
##  T1:5   Min.   :17.50
##  T2:5   1st Qu.:20.00
##  T3:5   Median :23.50
##  T4:5   Mean   :23.75
##         3rd Qu.:26.12
##         Max.   :33.50

media <- tapply(parica$Rep, parica$Trat, mean)   # média de altura em cada tratamento?
##   T1   T2   T3   T4
## 21.7 25.1 19.5 28.7

var <- tapply(parica$Rep, parica$Trat, var)      # variância da altura em cada tratamento?
print(var)
##    T1    T2     T3     T4
## 8.825 2.925  2.125  9.575

bartlett.test(parica$Rep, parica$Trat)           # teste de homogeneidade de variâncias
##
##  Bartlett test of homogeneity of variances
##
## data:  parica$Rep and parica$Trat
## Bartlett's K-squared = 2.9361, df = 3, p-value = 0.4016
```

Pode-se plotar a relação do Tratamento versus Altura da plântula em um gráfico do tipo boxplot, no qual as linhas horizontais pretas dentro das caixas representam as medianas e, os pontos em vermelho, as médias:

```
# plot: Rep x Trat
plot(parica$Trat, parica$Rep, type="o", main="Schizolobium parahyba",
    xlab="Tratamentos", ylab="Altura (cm)")
points(media, pch=20, col=2, cex=1.5)                    # Adiciona médias/tratamento

# Usando a função boxplot()
boxplot(parica$Rep ~ parica$Trat, main="Schizolobium parahyba",
    xlab="Tratamentos", ylab="Altura (cm)")
points(media, pch=20, col=2, cex=1.5)
```

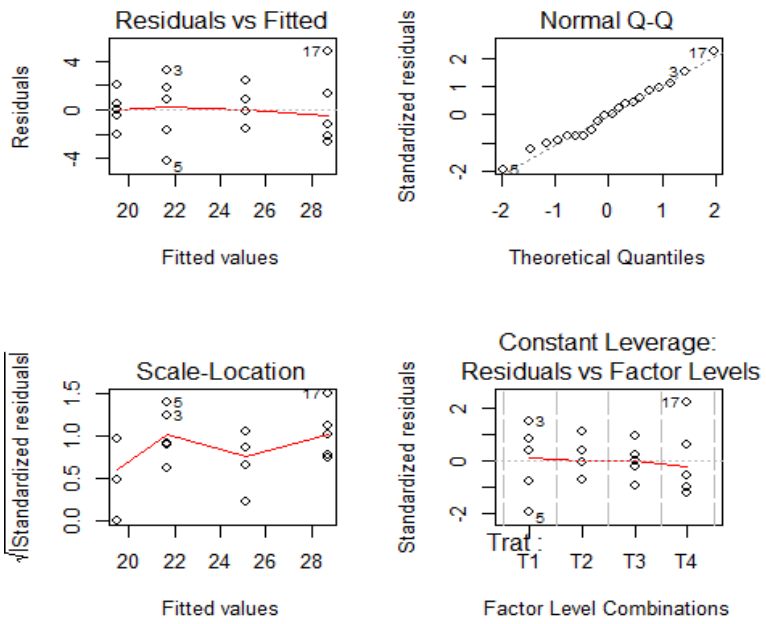


Para obter a ANOVA em um experimento sob Delineamento Inteiramente Casualizado (DIC), pode-se usar a função aov() do pacote **stats** do R-base (R CORE TEAM, 2017). Em seguida, pode-se usar a função anova() ou a função summary() para obter o quadro de ANOVA.

```
anova.DIC <- aov(Rep ~ Trat, data=parica)

anova(anova.DIC)
## Analysis of Variance Table
## Response: Rep
##           Df Sum Sq Mean Sq F value    Pr(>F)
## Trat       3 242.95  80.983  13.814 0.0001049 ***
## Residuals 16  93.80   5.863
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Uma análise gráfica dos resultados da ANOVA para o DIC:

```
plot(anova.DIC)                                          # gráficos para análise dos resíduos
```

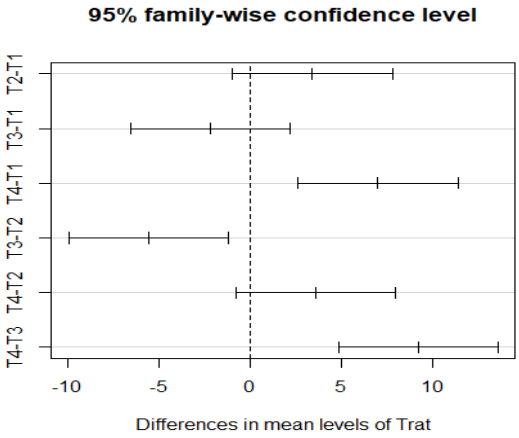


Interpretação: Há diferença entre, pelo menos, 2 tratamentos (rejeita-se H_0).

Quando há rejeição de H_0 ($F_{calculado} > F_{tabelado}$), constata-se que as médias dos tratamentos não são todas iguais entre si. Entretanto, o teste F não permite identificar quais tratamentos diferem entre si. Para identificar essas diferenças, é necessário realizar testes **post-hoc** ou **testes de comparações múltiplas**. Algun dos testes de comparações múltiplas mais conhecidos são o **teste de Tukey** e o **teste de Duncan**. Assim, para o exemplo do Paricá aplicaremos o teste de comparações múltiplas de **Tukey** usando da função TukeyHSD().

```
Tukey <- TukeyHSD(anova.DIC)
Tukey                                   # gera uma tabela com os resultados do teste de Tukey

## Tukey multiple comparisons of means
## 95% family-wise confidence level

## Fit: aov(formula = Rep ~ Trat, data = parica)

## $`Trat`
##           diff       lwr       upr     p adj
##  T2-T1    3.4 -0.981192  7.781192 0.1598946
##  T3-T1   -2.2 -6.581192  2.181192 0.4960496
##  T4-T1    7.0  2.618808 11.381192 0.0016074
##  T3-T2   -5.6 -9.981192 -1.218808 0.0102943
##  T4-T2    3.6 -0.781192  7.981192 0.1277882
##  T4-T3    9.2  4.818808 13.581192 0.0000976

# Na tabela acima, observa-se que a comparação entre os tratamentos T4-T1 e T4-T3 foram os únicos com
p-valor (coluna "p adj") menores que 0,05. Ou seja, foram os tratamentos que diferiram estatisticamente.
O resultado da tabela também pode ser observado graficamente:
```

```
plot(Tukey)          # gera um gráfico que indica diferenças estatísticas entre T4-T1 e T4-T3 (únicas
                       linhas horizontais que não tocam a linha tracejada)
```



12.1.2 Pressupostos para a realização de testes lineares

Para atender as premissas de modelos lineares é necessário verificar previamente às análises, a normalidade dos resíduos e a homocedasticidade das variâncias.

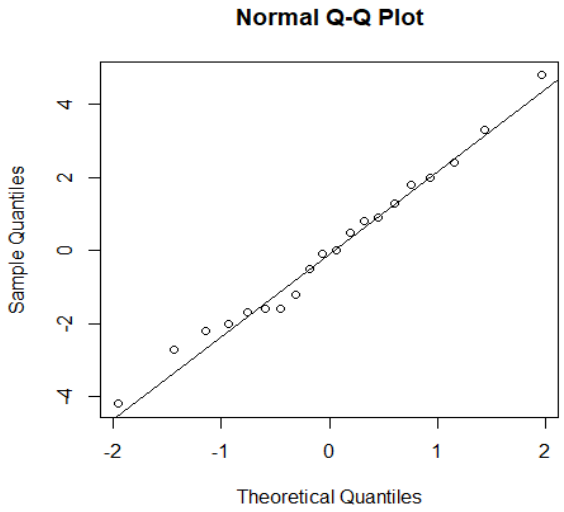
- **Normalidade dos resíduos**: os resíduos, ou erros experimentais, devem ser normalmente distribuídos.
- **Homocedasticidade das variâncias**: os erros experimentais devem possuir homogeneidade de variâncias. Importante para validar os testes de comparação de médias.

Teste de normalidade mais comum: Shapiro-Wilk. A normalidade é admitida quando $p > 0{,}05$ (H_0= existem evidências para admitir a normalidade dos resíduos).

```
shapiro.test(resid(anova.DIC))

##  Shapiro-Wilk normality test
##
## data:  resid(anova.DIC)
## W = 0.98613, p-value = 0.9876
```

O gráfico **Normal Q-Q plot** pode ser obtido usando as funções qqnorm() e qqline().

```
qqnorm(resid(anova.DIC))                    # obtendo o papel de probabilidade normal.
qqline(resid(anova.DIC))                    # inserindo uma linha auxiliar (linear).
```



Homogeneidade de variâncias: **teste de Levene** para testar a Homogeneidade de variâncias. Para aplica-lo pode-se usar o pacote **car** (FOX; WEISBERG, 2011). Execute install.packages("car"), caso não tenha o pacote instalado. A homogeneidade é admitida quando p > 0,05.

```
install.packages("car")
library(car)
leveneTest(Rep~Trat, parica)

## Levene's Test for Homogeneity of Variance (center = median)
##       Df       F value       Pr(>F)
##  group     3  0.7217        0.5536
##       16
```

12.2 Regressão Linear

A regressão linear simples é utilizada para analisar relações entre variáveis contínuas (KENNEY & KEEPING, 1962). No R, para especificar um modelo é preciso usar a notação de fórmulas. Por exemplo, para um modelo de regressão com uma variável resposta **Y** (ou dependente) e uma variável preditora **X** (ou independente), a notação de fórmula a ser usada no R é: **Y ~ X**. O argumento ~ (**til**) significa **em relação à. . .** ou **modelado por**. . . (ou seja, **Y** em relação à **X,** ou **Y** modelado por **X)**. Caso haja duas variáveis preditoras no modelo, a fórmula será: **Y~X1+X2**.

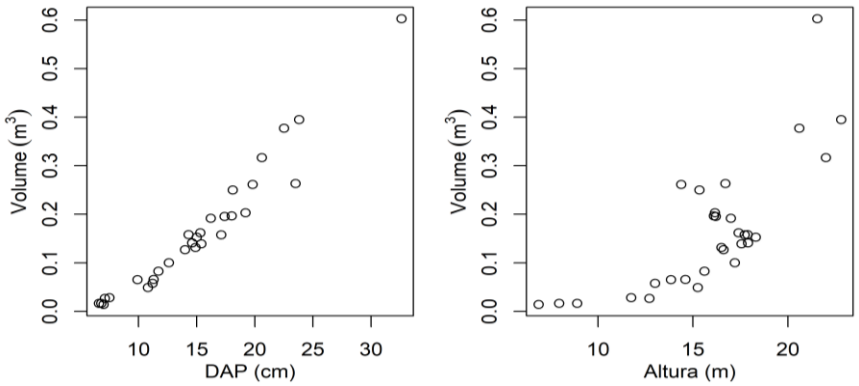
Para implementar uma regressão linear no ambiente R pode-se usar a função lm(), disponível no R-base. A função retorna os valores dos coeficientes betas estimados por meio do Método de Mínimos Quadrados. Para fins de exemplificação, considere os dados de DAP (cm), H (m) e Volume (m^3) de 30 árvores de *Tectona grandis* (Teca):

```
library(data.table)
teca <- fread("Tectona.csv")
print(teca)
```

Arvore	DAP	H	Volume
1	6.600	8.900	0.017
2	6.800	7.950	0.017
3	7.000	6.870	0.014
4	7.100	12.700	0.027
5	7.500	11.740	0.028
6	9.900	13.840	0.066
7	10.800	15.250	0.049
8	11.200	13.000	0.058
9	11.300	14.600	0.066
10	11.700	15.600	0.083
11	12.600	17.200	0.100
12	14.000	16.600	0.127
13	14.300	17.870	0.158
14	14.600	17.900	0.141
15	14.900	16.500	0.132
16	15.000	18.300	0.153
17	15.300	17.400	0.162
18	15.400	17.550	0.139
19	16.200	16.990	0.192
20	17.100	17.720	0.158
21	17.400	16.200	0.196
22	18.000	16.100	0.197
23	18.100	15.340	0.250
24	19.200	16.150	0.203
25	19.800	14.380	0.262
26	20.600	22.000	0.317
27	22.500	20.600	0.377
28	23.500	16.700	0.263
29	23.800	22.800	0.395
30	32.600	21.550	0.603

12.2.1 Gráficos de dispersão - plot()

```
attach(teca)

plot(DAP, Volume, type = "p", main=NULL, font.main=NULL, col.main=NULL,
    xlab="DAP (cm)", ylab=expression(Volume~(m^3)), font.lab=1, col.lab="black",
    font.axis=1, col.axis = "black")

plot(H, Volume, type = "p", main=NULL, font.main=NULL, col.main=NULL,
    xlab="Altura (m)", ylab=expression(Volume~(m^3)), font.lab=1, col.lab="black",
    font.axis=1, col.axis = "black")

detach(teca)
```



12.2.2 Ajuste de modelos

Ajustar os seguintes modelos volumétricos:

- Berkhout: $V = \beta_0 + \beta_1\ (DAP) V = \beta_0 + \beta_1\ (DAP)$
- Kopezky-Gerhardt: $V = \beta_0 + \beta_1\ (DAP^2)$
- Schumacher-Hall: $\ln V = \beta_0 + \beta_1\ (\ln DAP) + \beta_2\ (\ln H)$

```
Berkhout <- lm(Volume ~ DAP, data=teca)                  # Ajuste do modelo de Berkhout
KGerhardt <- lm(Volume ~ I(DAP^2), data=teca)            # Ajuste do modelo de Kopezky-Gerhardt
SHall <- lm(log(Volume) ~ log(DAP) + log(H), data=teca)  # Ajuste do modelo de Schumacher-Hall
```

Para obter informações mais detalhadas do ajuste pode-se usar a função summary(). A partir dessa função, os seguintes valores podem ser observados: Erro-Padrão ($s_{\bar{x}}$), Estimativas dos Coeficientes de Regressão (Valores de β), Erro-Padrão de Estimativa (s_{yx}), Coeficiente de Determinação (R^2) e Coeficiente de Determinação Ajustado ($R^2{}_{aj.}$). Além disso, pode-se visualizar o teste F de significância global e o do(s) teste(s) t-Student para significância do(s) coeficiente(s) da regressão.

```
summary(Berkhout)
```

```
## Call:
## lm(formula = Volume ~ DAP, data = teca)
##
## Residuals:
##       Min         1Q         Median       3Q            Max
## -0.081155  -0.020937     -0.006857    0.027424    0.062904
##
## Coefficients:
##                        Estimate     Std. Error    t value     Pr(>|t|)
## (Intercept)         -0.161216       0.017116    -9.419       3.55e-10 ***
## DAP                   0.021516       0.001053     20.428       < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.03377 on 28 degrees of freedom
## Multiple R-squared:  0.9371, Adjusted R-squared:  0.9349
## F-statistic: 417.3 on 1 and 28 DF,  p-value: < 2.2e-16
```

```
summary(KGehrardt)

## Call:
## lm(formula = Volume ~ I(DAP^2), data = teca)
##
## Residuals:
##       Min          1Q          Median       3Q           Max
## -0.077437    -0.015987    -0.004587   0.014409    0.064796
##
## Coefficients:
##                        Estimate     Std. Error    t value     Pr(>|t|)
## (Intercept)        3.913e-03       9.023e-03     0.434         0.668
## I(DAP^2)           6.098e-04       2.685e-05    22.710        <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.03056 on 28 degrees of freedom
## Multiple R-squared:  0.9485, Adjusted R-squared:  0.9467
## F-statistic: 515.8 on 1 and 28 DF,  p-value: < 2.2e-16
```

```
summary(SHall)

## Call:
## lm(formula = log(Volume) ~ log(DAP) + log(H), data = teca)
##
## Residuals:
##      Min         1Q         Median        3Q        Max
## -0.33671    -0.08354    0.00374      0.05991   0.30833
##
## Coefficients:
```

```
##                 Estimate     Std. Error    t value      Pr(>|t|)
## (Intercept)     -9.4574        0.2793      -33.862      < 2e-16 ***
## log(DAP)         1.8982        0.1183       16.052     2.47e-15 ***
## log(H)           0.8301        0.1760        4.717      6.51e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1428 on 27 degrees of freedom
## Multiple R-squared:  0.9805, Adjusted R-squared:  0.9791
## F-statistic: 679.1 on 2 and 27 DF,  p-value: < 2.2e-16
```

A função anova() permite obter a tabela de ANOVA da Regressão:

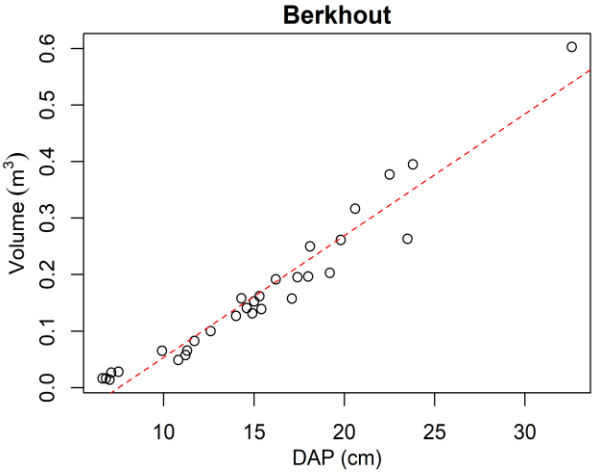
```
anova(Berkhout)

## Analysis of Variance Table
##
## Response: Volume
##                 Df      Sum Sq      Mean Sq     F value      Pr(>F)
## DAP              1      0.47588      0.47588      417.32      < 2.2e-16 ***
## Residuals       28      0.03193      0.00114
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(KGehrardt)

## Analysis of Variance Table
##
## Response: Volume
##                 Df      Sum Sq       Mean Sq      F value      Pr(>F)
## I(DAP^2)         1      0.48166       0.48166      515.77      < 2.2e-16 ***
## Residuals       28      0.02615       0.00093
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(SHall)

## Analysis of Variance Table
##
## Response: log(Volume)
##                 Df      Sum Sq       Mean Sq      F value      Pr(>F)
## log(DAP)         1      27.2316       27.2316      335.981      < 2.2e-16 ***
## log(H)           1      0.4536        0.4536       22.255      6.508e-05 ***
## Residuals       27      0.5503        0.0204
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Pode-se visualizar a reta ajustada para o modelo de Berkhout ($V = \beta_0 + \beta_1\ (DAP) V = \beta_0 + \beta_1\ (DAP)$):

```
plot(teca$Volume~teca$DAP, main="Berkhout",
   xlab="DAP (cm)", ylab = expression(Volume~(m^3)))
abline(Berkhout, lty=2, col="red")                    #abline da regressão linear
```



Outras informações importantes podem ser obtidas. Por exemplo, os valores preditos pelo modelo de regressão podem ser extraídos com uso da função predict(). Já os valores dos resíduos podem ser obtidos com a função residuals().

```
predict(Berkhout)
residuals(Berkhout)
```

12.2.3 Análise de Resíduos

Os resíduos da regressão linear devem ser avaliados quanto às pressuposições de normalidade, autocorrelação e homocedasticidade. O teste para normalidade dos resíduos pode ser realizado usado a função shapiro.test(). A autocorrelação pode ser detectada com uso da função durbinWatsonTest() e a heterocedasticidade dos resíduos com a função Breusch-Pagan().

12.2.3.1 Normalidade dos resíduos

Realizando o teste de normalidade dos resíduos (α = 0,05) para os modelos de Berkhout, Kopezky-Gehrardt e Shumacher-Hall:

Teste de Shapiro-Wilk para normalidade (valores de p > 0,05 indicam resíduos com distribuição normal).

```
shapiro.test(Berkhout$residuals)

##  Shapiro-Wilk normality test
## data:  Berkhout$residuals
## W = 0.97824, p-value = 0.7772
```

```
shapiro.test(KGehrardt$residuals)
##  Shapiro-Wilk normality test
## data:  KGehrardt$residuals
## W = 0.96446, p-value = 0.4005

shapiro.test(SHall$residuals)
##  Shapiro-Wilk normality test
## data:  SHall$residuals
## W = 0.98224, p-value = 0.8815
```

12.2.3.2 Autocorrelação dos resíduos

O teste **Durbin-Watson** (α = 0,05) pode ser utilizado para apontar indícios de autocorrelação dos resíduos em modelos de regressão linear. Quando $p\text{-}value \geq$ 0,05, não existem evidências de que os resíduos sejam autocorrelacionados. Para tanto, pode-se usar a função durbinWatsonTest() disponível no pacote **car** (FOX; WEISBERG, 2011).

```
library(car)
durbinWatsonTest(Berkhout)

##  lag   Autocorrelation   D-W Statistic    p-value
##    1       0.0490722           1.73776      0.364
##  Alternative hypothesis: rho != 0

durbinWatsonTest(KGehrardt)

##  lag   Autocorrelation    D-W Statistic     p-value
##    1       -0.2825325            2.466242       0.276
##  Alternative hypothesis: rho != 0

durbinWatsonTest(SHall)

##  lag   Autocorrelation     D-W Statistic     p-value
##    1       -0.1096209            2.135885       0.942
##  Alternative hypothesis: rho != 0
```

12.2.3.3 Heterocedasticidade dos resíduos

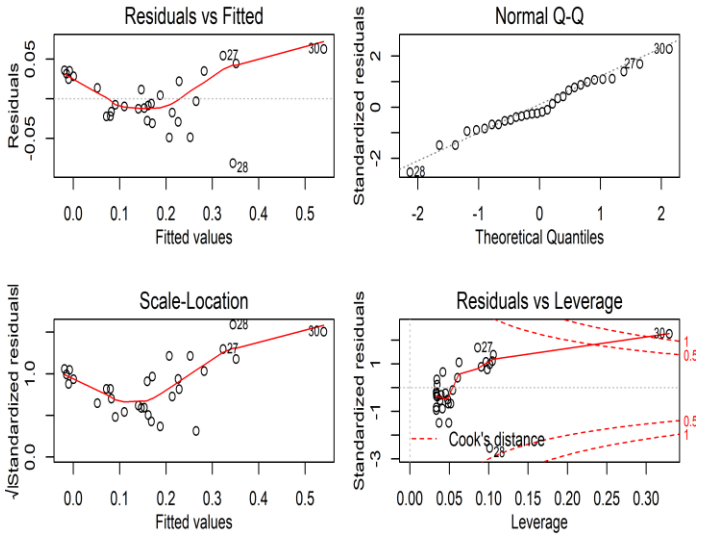
O teste **Breusch-Pagan** (α =0,05) pode ser usado para detectar homocedasticidade de variâncias dos resíduos. Quando p-$value$ < 0,05 a hipótese nula (H_0) é rejeitada, isto é, há presença de heterocedasticidade. Para aplicação do teste, pode-se usar a função bptest() disponível no pacote **lmtest** (ZEILEIS; HOTHORN, 2002).

```
install.packages("lmtest")
library(lmtest)
bptest(Berkhout)

##  studentized Breusch-Pagan test
```

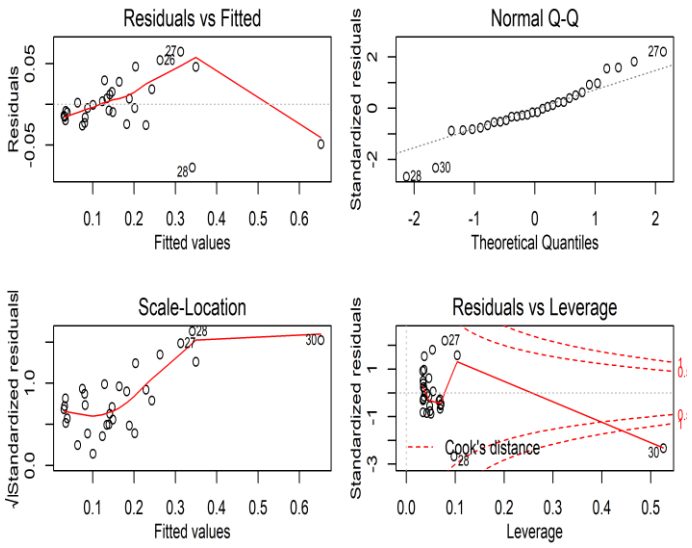
```
## data:  Berkhout
## BP = 9.8249, df = 1, p-value = 0.001722

bptest(KGehrardt)

##  studentized Breusch-Pagan test
## data:  KGehrardt
## BP = 12.322, df = 1, p-value = 0.0004476

bptest(SHall)
##  studentized Breusch-Pagan test
## data:  SHall
## BP = 2.018, df = 2, p-value = 0.3646
```

A função plot() default gera quatro gráficos úteis para detecção da normalidade, homocedasticidade e valores influentes:

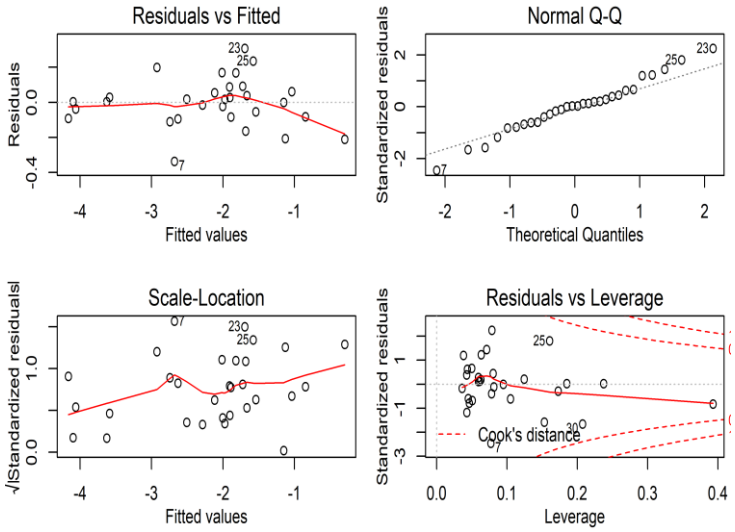
```
plot(Berkhout)
```




```
plot(KGehrardt)
```




```
plot(SHall)
```



12.2.4 Multicolinearidade (modelos múltiplos)

A inexistência de multicolinearidade é outra importante pressuposição que deve ser analisada em modelos de regressão linear múltipla. O grau de confundimento entre variáveis preditoras pode ser verificado por meio da estatística *Variance Inflation Factor* – (*VIF*), que detecta problemas de multicolinearidade. É recorrente o uso de VIF > 10 como critério para afirmar a existência de multicolinearidade no modelo de regressão linear múltipla. Para obter o valor desse índice, pode-se usar a função vif() do pacote **faraway** (FARAWAY, 2016).

```
install.packages("faraway")
```

```
library(faraway)
vif(SHall)

##   log(DAP)    log(H)
##    3.339837   3.339837
```

13 Amostragem Aleatória Simples (AAS)

É o procedimento fundamental de seleção a partir do qual derivam os demais processos de amostragem (PÉLLICO NETO; BRENA, 1997; SANQUETTA et al., 2009). A Amostragem Aleatória Simples (AAS) baseia-se num processo estritamente aleatório, onde as unidades amostrais (UAs) são selecionadas com igual probabilidade (1/N), em que N é o nº total de unidades de amostras que compõem o espaço amostral, ou seja, a população amostrada (QUEIROZ, 1998).

A seleção das unidades amostrais pode ser realizada com ou sem reposição. No sorteio com reposição uma determinada unidade de amostra pode ser sorteada mais de uma vez para compor a amostra. No sorteio sem reposição uma unidade de amostra poderá ser sorteada somente uma única vez para compor a amostra (PÉLLICO NETO; BRENA, 1997).

Em inventários florestais, a AAS é geralmente conduzida através de um sorteio sem reposição. Além disso, expõe que a inclusão de uma unidade amostral mais de uma vez na amostra refletirá uma homogeneização da variância entre as UAs (QUEIROZ, 1998).

13.1 Estimadores da AAS

As principais estimativas a serem obtidas para a AAS são:

a) Média aritmética ($\overline{x}$);
b) Variância (s_x^2);
c) Desvio Padrão (s_x);
d) Coeficiente de variação (CV%);
e) Intensidade amostral (n);
f) Variância da média ($s_{\overline{x}}^2$);
g) Erro padrão da média ($s_{\overline{x}}$);
h) Erro de amostragem: absoluto (E_a) e relativo (E_r);
i) Intervalo de confiança para média ($IC_{\overline{x}}$);
j) Total da população ($\hat{X}$); e
k) Intervalo de confiança para o total ($IC_{\hat{X}}$).

13.2 Estimando os parâmetros da AAS

A seguir as principais estimativas da AAS serão obtidas usando a linguagem R. Os dados utilizados foram extraídos do livro **Inventário Florestal: Planejamento e execução**, com o seguinte enunciado: "Em um talhão de *Pinus taeda* plantado em uma área de 40 hectares foi realizado um inventário, cujo objetivo é estimar o volume de madeira da população em questão. Para a realização do inventário foi utilizado o processo de amostragem aleatória simples, onde se deseja saber quantas parcelas de 600m² devem ser usadas para atingir a precisão desejada. A definição do número ideal de parcelas depende da variabilidade da população. Para isto foi realizado um inventário piloto, onde foram medidas 16 parcelas, com a finalidade de obter a variância da população e, assim, estimar a intensidade amostral para o inventário definitivo. Para os cálculos das estimativas foi considerado que o erro máximo admissível é de 10% e o nível de probabilidade é de 95% (SANQUETTA et al., 2014, p. 110). "

```
library("data.table")
IF <- fread("AAS.csv")                    # Carrega o conjunto de dados
print(IF)
```

Parcela	Volume
1	20.850
2	19.470
3	24.130
4	24.340
5	25.130
6	22.370
7	22.510
8	19.780
9	25.050
10	28.840
11	23.700
12	24.780
13	22.580
14	23.700
15	36.160
16	17.830

Obtendo as estatísticas básicas para as unidades amostrais:

1. Média aritmética $(\bar{x})$ - m^3/parcela:

```
mean(IF$Volume)
## [1] 23.82625
```

2. Variância (s_x^2) - $(m^3/parcela)^2$

```
var(IF$Volume)
## [1] 17.8215
```

3. Desvio Padrão (s_x) - m^3/parcela:

```
sd(IF$Volume)
## [1] 4.221552
```

4. Coeficiente de variação (CV%):

```
coefficient.variation(sd=sd(IF$Volume),avg=mean(IF$Volume))*100
## [1] 17.71807
```

5. Intensidade amostral (n): A intensidade de amostragem é função: i) da variabilidade da variável de interesse (volume da floresta); ii) do erro de amostragem máximo admissível; e iii) do nível de confiança fixado (PÉLLICO NETO; BRENA, 1997). Assim, para obter a estimativa de n é necessário obter as seguintes informações:

a) O valor de E (expectância do erro) - m^3/parcela: Em inventário florestal é usual estabelecer um erro máximo admissível de 10% para estimativa da média. Então, para obter o valor de E basta fazer:

$$E = (LE \,.\, \overline{x})$$

Uma função para obter o valor de E:

```
E <- function(x){
  media = mean(x)
  E = signif(0.1*media, 4)
  return(E)
}

E(IF$Volume)
## [1] 2.383
```

b) O número de unidades amostrais possíveis na população (N): Quantas unidades de amostra de tamanho de 600 m² é possível estabelecer na população, cuja área total é de 400.000m² (40 hectares)? Para obter essa informação basta dividir a área da população pelo tamanho da parcela. Assim, obtêm-se que são possíveis 667 parcelas de 600m² na população.

$$N = \frac{A}{a}$$

Uma função para obter o valor de N:

```
N <- function(A, a){
  N <- ceiling(A/a)
  return(N)
}

N = N(400000,600)
N
## [1] 667
```

c) Determinar se a população é finita ou infinita: Para determinar o número de unidades de amostras necessárias para atender determinada precisão e nível de probabilidade é necessário saber, antecipadamente, se a população é **Finita** ou **Infinita**. Para tanto, inicialmente é preciso obter a fração de amostragem *f* e, em seguida, observar a convenção (PÉLLICO NETO; BRENA, 1997):

$$\begin{cases} Se \quad 1 - f \geq 0{,}98 & \text{População Infinita} \\ Se \quad 1 - f < 0{,}98 & \text{População Finita} \end{cases}$$

A fração de amostragem (*f*) é dada pela razão entre o número de unidades de amostra (*n*) e o número total de unidades de amostra possíveis na população (N) (PÉLLICO NETO; BRENA, 1997; SANQUETTA et al., 2014):

$$f = \frac{n}{N}$$

Uma função para obter o Fator de Correção (FC) e constatar a natureza da população. O cálculo intermediário da fração de amostragem (*f*) é realizado e também retornado. A função FC recebe três parâmetros: x = vetor da variável de interesse; **A** = area da população; e **a** = tamanho da unidade de amostra. O valor de **A** e **a** devem estar na mesma unidade de medida.

```
FC <- function(x, A, a){
  n <- length(x)
  N <- ceiling(A/a)
  f <- n/N
  FC <- 1-f

  if(FC >= 0.98){
    cat("A população é Infinita. Portanto, despreze o FC na fórmula da n.\n")
  }else{
    cat("A população é Finita. Portanto, use o FC para corrigir n.\n")
  }
  return(list(f=f, FC=FC))
}

FC <- FC(x=IF$Volume, A=400000, a = 600)
## A população é Finita. Portanto, use o FC para corrigir n.
```

Determinados os valores de *E* e *N*, e constatada a natureza finita da população, pode-se obter a intensidade de amostragem ideal em função da variância:

```
n <- function(x, A, a){
  N <- ceiling(A/a)
  E = 0.1*mean(x)
  t = qt(1-.05/2, df=length(x)-1)
  n <- ceiling((N*t^2*var(x))/(N*E^2 + t^2*var(x)))
  cat(paste("Para atender ao erro estabelecido você deve amostrar", n,
            "parcelas.\n"))

  if(n <= length(x)){
    cat("Esforço amostral satisfatório. O IF é definitivo!")
  }else{
    cat(paste("Retorne a campo e meça mais", abs(length(x)-n), "parcelas."))
  }
}

n(x = IF$Volume, A = 400000, a = 600)
## Para atender ao erro estabelecido você deve amostrar 14 parcelas.
## Esforço amostral satisfatório. O IF é definitivo!
```

6. Variância da média ($s_{\bar{x}}^2$) - $(m^3/parcela)^2$: A variância da média para populações finitas é dada por:

$$s_{\bar{x}}^2 = \frac{s_x^2}{n} \cdot \left(\frac{N-n}{N}\right)$$

```
var(IF$Volume)/length(IF$Volume)*(FC$FC)
## [1] 1.087125
```

7. Erro padrão da média ($s_{\bar{x}}$) - $(m^3/parcela)$: O erro padrão da média para populações finitas é dado por:

$$s_{\bar{x}} = \pm \frac{s_x}{\sqrt{n}} \cdot \sqrt{(1-f)}$$

```
sbarx <- sd(IF$Volume)/sqrt(length(IF$Volume))*(sqrt(FC$FC))
sbarx
## [1] 1.042653
```

8. Erro de amostragem:

a) Erro de amostragem absoluto (E_a) - (m^3/parcela): O erro de amostragem absoluto é dado por:

$$E_a = \pm t \, . \, s_{\overline{x}}$$

```
Ea <- qt(1-.05/2, df=length(IF$Volume)-1)*sbarx
Ea
## [1] 2.222362
```

b) Erro de amostragem relativo (E_r) - %: O erro de amostragem relativo é dado por:

$$E_r = \pm \frac{t \, . \, s_{\overline{x}}}{\overline{x}} \, . \, 100$$

```
Er <- Ea/mean(IF$Volume)*100
Er
## [1] 9.327369
```

9. Intervalo de confiança para média ($IC_{\overline{x}}$): O IC para média é dado por:

$$IC_{\overline{x}} = [\overline{x} - (t \, . \, s_{\overline{x}}) \leq \overline{X} \leq \overline{x} + (t \, . \, s_{\overline{x}})] = P$$

```
# Limite inferior para média (LI)
LIbarx <- mean(IF$Volume)-Ea
LIbarx
## [1] 21.60389

# Limite superior para média (LS)
LSbarx <- mean(IF$Volume)+Ea
LSbarx
## [1] 26.04861
```

Portanto, o IC para média é: $IC_{\overline{x}} = [21{,}61 \, m^3/parcela \leq \overline{X} \leq 26{,}05 \, m^3/parcela] = 95\%$.

10. Total da população ($\hat{X}$) - m^3: O volume estimado para o total da população é dado por:

$$\hat{X} = N \, . \, \overline{x}$$

```
hatX <- N*mean(IF$Volume)
hatX
## [1] 15892.11
```

11. Intervalo de confiança para o total ($IC_{\hat{X}}$): O IC para a estimativa do volume total da população é dado por:

$$IC_{\hat{X}} = [\hat{X} - N(t\,.\,s_{\overline{x}}) \leq X \leq \hat{X} + N(t\,.\,s_{\overline{x}})] = P$$

```
# Limite inferior para total da população (LI)
LIhatx <- hatX - N*Ea
LIhatx
## [1] 14409.79

# Limite superior para total da população (LS)
LShatx <- hatX + N*Ea
LShatx
## [1] 17374.42
```

Portanto, o IC para o total da população é: $IC_{\hat{X}} = [14.409{,}8\,m^3 \leq \overline{X} \leq 17.374{,}4\,m^3] = 95\%$.

Observe que para melhor compreensão dos cálculos os parâmetros da AAS foram obtidos passo a passo. Porém, é possível obter todas as estimativas criando apenas uma única função no ambiente R. A seguir é apresentada uma função genérica para obter as estimativas da AAS. A função **AAS** recebe três parâmetros: **x** = vetor da variável de interesse; **A** = area da população; e **a** = tamanho da unidade de amostra. Internamente a função faz o cálculo do Fator de Proporcionalidade (FP) para obter os parâmetros estimados em hectare (ha):

```
AAS <- function(x, A, a){
 FP <- 10000/a
 x <- x*FP
 Soma <- sum(x)
 Media <- mean(x, na.rm = TRUE)
 Variancia <- var(x)
 N <- ceiling(A/a)
 f <- length(x)/N
 FC <- 1-f
 E = 0.1*mean(x)
 t = qt(1-.05/2, df=length(x)-1)

 if(FC >= 0.98){
  cat("\n-------------------------------------------------------\n
     A população é Infinita.\n")
  n <- ceiling((t^2*var(x))/E^2)
  cat("Para atender ao erro estabelecido você deve amostrar", n, "parcelas.\n")
  VarM <- var(x)/length(x)
  SdM <- sqrt(VarM)
  Ea <- t*SdM
  Er <- (Ea/Media)*100
  ICI <- Media - Ea
```

```
  ICS <- Media + Ea
  TotPop <- N*Media
  ICIP <- ICI*A
  ICSP <- ICS*A

  if(n <= length(x)){
  cat("Esforço amostral satisfatório. O IF é definitivo!")
  }else{
  cat("Retorne a campo e meça mais", abs(length(x)-n), "parcelas.")
  }

}else{
  cat("\n---------------------------------------------------------------\n")
  cat("A população é Finita -", "FC =",round(FC,3),"\n")
  n <- ceiling((N*t^2*var(x))/(N*E^2 + t^2*var(x)))
  cat("Para atender ao erro estabelecido você deve amostrar", n, "parcelas.\n")
  VarM <- var(x)/length(x)*FC
  SdM <- (sd(x)/sqrt(length(x)))*sqrt(FC)
  Ea <- t*SdM
  Er <- (Ea/Media)*100
  ICI <- Media - Ea
  ICS <- Media + Ea
  TotPop <- N*Media
  ICIP <- ICI*A
  ICSP <- ICS*A

  if(n <= length(x)){
  cat("Esforço amostral satisfatório. O IF é definitivo!")
  }else{
  cat("ATENÇÃO: Retorne a campo e meça mais", abs(length(x)-n), "parcelas.")
  }

}
  cat("\n---------------------------------------------------------------\n")

 df <- format(data.frame(Parametros=
                    c("Soma", "Média", "Número de amostras possíveis",
                      "Fração de amostragem", "Erro máximo admissível",
                      "t-student", "Intensidade amostral",
                      "Variância da média", "Erro padrão da Média",
                      "Erro de amostragem absoluto",
                      "Erro de amostragem relativo",
                      "IC inferior para média",
                      "IC superior para média", "Total da população",
                      "IC inferior para total da população",
                      "IC superior para total da população"),
              Estimativas=c(Soma, Media, N, f, E, t, n,
                            VarM, SdM, Ea, Er, ICI, ICS,
                            TotPop, ICIP, ICSP)), justify = "right",
          digits = 6, nsmall=3, scientific=FALSE)
 return(df)
}
```

```
AAS(x = IF$Volume, A = 400000, a = 600)

## ------------------------------------------------------------
## A população é Finita - FC = 0.976
## Para atender ao erro estabelecido você deve amostrar 14 parcelas.
## Esforço amostral satisfatório. O IF é definitivo!
## ------------------------------------------------------------

##       Parametros                          Estimativas
## 1     Soma                                6353.666667
## 2     Média                               397.104167
## 3     Número de amostras possíveis        667.000000
## 4     Fração de amostragem                0.023988
## 5     Erro máximo admissível              39.710417
## 6     t-student                           2.131450
## 7     Intensidade amostral                14.000000
## 8     Variância da média                  301.979212
## 9     Erro padrão da Média                17.377549
## 10    Erro de amostragem absoluto         37.039369
## 11    Erro de amostragem relativo         9.327369
## 12    IC inferior para média              360.064798
## 13    IC superior para média              434.143536
## 14    Total da população                  264868.479167
## 15    IC inferior para total da população 144025919.023034
## 16    IC superior para total da população 173657414.310299
```

14 Análise fitossociológica

14.1 Estrutura Horizontal

A análise fitossociológica da floresta abrange a estimativa de diversos parâmetros. Os parâmetros fitossociológicos da estrutura horizontal podem ser expressos, em valores absolutos e relativos (MUELLER-DOMBOIS; ELLENBERG, 1974). Os principais parâmetros fitossociológicos da estrutura horizontal são (MUELLER-DOMBOIS; ELLENBERG, 1974; SANQUETTA et al., 2014; SOUZA; SOARES, 2013):

1. Densidade absoluta (DA_i) da *i*-ésima espécie, em número de indivíduos por hectare, por espécie;
2. Densidade relativa (DR_i) da *i*-ésima espécie, em porcentagem;
3. Dominância absoluta (DoA_i) da *i*-ésima espécie ($m^2.ha^{-1}$);
4. Dominância relativa (DoR_i) da *i*-ésima espécie, em porcentagem;
5. Frequência absoluta (FA_i) da *i*-ésima espécie, em porcentagem;
6. Frequência relativa (FR_i) da *i*-ésima espécie, em porcentagem;
7. Valor de cobertura (VC_i) da *i*-ésima espécie, em porcentagem;
8. Porcentagem de cobertura (PC_i) da *i*-ésima espécie;
9. Valor de importância (VI_i) da *i*-ésima espécie, em porcentagem; e
10. Porcentagem de importância (PI_i) da *i*-ésima espécie.

14.2 Estimando os parâmetros da estrutura horizontal

A seguir os principais parâmetros fitossociológicos da estrutura horizontal serão obtidos usando a linguagem R. Os dados foram obtidos de parcelas permanentes instaladas em Floresta Ombrófila Mista (FOM), com nível de inclusão de 10cm (DAP ≥ 10 cm). Foram estabelecidas 3 parcelas de 1000 m² (0.1 ha), com área total amostra (A) de 0.3 ha. Para fins didáticos, serão apresentados os códigos para obter os parâmetros da estrutura horizontal da espécie *Araucaria angustifolia* e, em seguida, mostrar-se-á um código para obter os parâmetros de todas as espécies de uma só vez usando o pacote **data.table** para manipulação de dados (DOWLE; SRINIVASAN, 2017):

```
library("data.table")                    # Pacote para manipulação de dados
FOM <- fread("Fito.csv")                 # Carrega o conjunto de dados
print(FOM)
```

Parcela	Especie	DAP
1	Araucaria angustifolia	34.500
1	Ocotea porosa	56.200
1	Ocotea pulchella	13.400
1	Maytenus ilicifolia	10.200
1	Ilex paraguariensis	10.900
1	Campomanesia xanthocarpa	24.700
1	Drymis brasiliensis	23.800
1	Allophyllus edulis	13.100
1	Matayba eleagnoides	31.000
1	Cupania vernalis	10.000
1	Capsicodendron dinnisii	21.900
1	Araucaria angustifolia	76.600
1	Araucaria angustifolia	45.900
1	Ilex paraguariensis	23.000
1	Matayba eleagnoides	17.900
1	Araucaria angustifolia	29.900
1	Campomanesia xanthocarpa	17.600
1	Ilex paraguariensis	13.900
1	Allophyllus edulis	12.100
1	Ilex paraguariensis	11.300
1	Vernonia discolor	23.800
2	Ilex paraguariensis	14.500
2	Nectandra grandiflora	13.500
2	Eugenia uniflora	13.200
2	Campomanesia xanthocarpa	22.000
2	Araucaria angustifolia	34.600
2	Matayba eleagnoides	32.000
2	Mimosa scabrella	28.000
2	Araucaria angustifolia	134.000
2	Ilex paraguariensis	19.500
2	Sapium glandulatum	12.900
2	Schinus terebinthifolius	23.500
2	Ilex paraguariensis	35.000
2	Araucaria angustifolia	19.000
2	Ilex paraguariensis	27.000
2	Ilex paraguariensis	17.600
2	Araucaria angustifolia	50.700
2	Nectandra grandiflora	16.900
2	Nectandra megapotamica	32.700
2	Araucaria angustifolia	44.800
2	Ilex paraguariensis	21.000
2	Luehea divaricata	34.500
2	Cedrela fissilis	38.600

Parcela	Especie	DAP
2	Matayba eleagnoides	23.800
3	Piptocarpha angustifolia	27.900
3	Ilex paraguariensis	17.500
3	Ilex paraguariensis	26.000
3	Campomanesia xanthocarpa	22.000
3	Araucaria angustifolia	45.000
3	Matayba eleagnoides	28.700
3	Matayba eleagnoides	19.700
3	Araucaria angustifolia	48.000
3	Nectandra megapotamica	15.600
3	Ocotea porosa	78.900
3	Ilex paraguariensis	23.000
3	Ilex paraguariensis	16.000
3	Araucaria angustifolia	56.000
3	Ilex paraguariensis	23.600
3	Ilex paraguariensis	19.900
3	Araucaria angustifolia	25.600
3	Matayba eleagnoides	27.600
3	Nectandra grandiflora	19.800
3	Araucaria angustifolia	18.900
3	Ilex paraguariensis	10.100
3	Mimosa scabrella	27.000
3	Schinus terebinthifolius	21.000
3	Allophyllus edulis	18.400
3	Ilex dumosa	35.700
3	Tabebuia avellanedae	29.600
3	Sapium glandulatum	23.600

Inicialmente, pode-se fazer uma breve inspeção dos dados:

```
nrow(FOM)
## [1] 70
names(FOM)
## [1] "Parcela" "Especie" "DAP"
dim(FOM)
## [1] 70  3
```

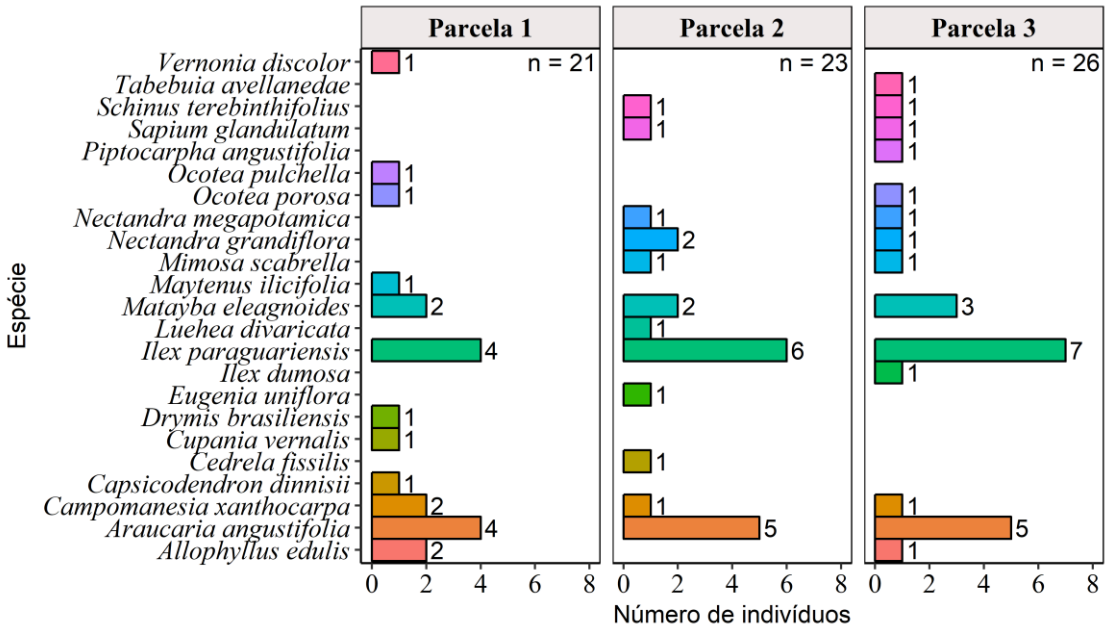
```
# n = Número total de indivíduos amostrados na j-ésima parcela
FOM[, .(n=.N), by=Parcela][]
```

Parcela	n
1	21
2	23
3	26

```
# Número de indivíduos amostrados da i-ésima espécie na j-ésima parcela
FOM[, .(ni=.N), by=c("Parcela", "Especie")]
```

Parcela	Especie	ni
1	Araucaria angustifolia	4
1	Ocotea porosa	1
1	Ocotea pulchella	1
1	Maytenus ilicifolia	1
1	Ilex paraguariensis	4
1	Campomanesia xanthocarpa	2
1	Drymis brasiliensis	1
1	Allophyllus edulis	2
1	Matayba eleagnoides	2
1	Cupania vernalis	1
1	Capsicodendron dinnisii	1
1	Vernonia discolor	1
2	Ilex paraguariensis	6
2	Nectandra grandiflora	2
2	Eugenia uniflora	1
2	Campomanesia xanthocarpa	1
2	Araucaria angustifolia	5
2	Matayba eleagnoides	2
2	Mimosa scabrella	1
2	Sapium glandulatum	1
2	Schinus terebinthifolius	1
2	Nectandra megapotamica	1
2	Luehea divaricata	1
2	Cedrela fissilis	1
3	Piptocarpha angustifolia	1
3	Ilex paraguariensis	7
3	Campomanesia xanthocarpa	1
3	Araucaria angustifolia	5
3	Matayba eleagnoides	3
3	Nectandra megapotamica	1
3	Ocotea porosa	1
3	Nectandra grandiflora	1
3	Mimosa scabrella	1
3	Schinus terebinthifolius	1
3	Allophyllus edulis	1
3	Ilex dumosa	1
3	Tabebuia avellanedae	1
3	Sapium glandulatum	1

```
# Pode-se gerar um gráfico do número de indivíduos por parcela a partir da função ggplot (), presente no
 pacote "ggplot2":

install.packages("ggplot2")                                    # instalação do pacote
library(ggplot2)                                               # carrega o pacote
ggplot(FOM[, .(ni=.N), by=c("Parcela", "Especie")],            # gráfico a partir da função ggplot()
      aes(x=Especie, y=ni, fill=Especie)) +
 geom_bar(stat="identity",position="dodge",width = 1,colour="black")+
 geom_text(aes(label=ni,hjust=-.3, vjust=0.5),
          position=position_dodge(width = 0.7))+
 facet_grid(~ Parcela, labeller=labeller(
   Parcela = Parcela<-as_labeller(
    c(`1`="Parcela 1",`2`="Parcela 2",`3`="Parcela 3"))))+
    coord_flip()+
 geom_text(data=ddply(.data=FOM, .(Parcela), summarize,
                 n=paste("n =", length(Especie))),
          aes(x=23, y=7, label=n), colour="black",
          inherit.aes=FALSE, parse=FALSE)+
    theme_bw()+
    theme(axis.line.x=element_line(size=0.5,colour="black"),
      axis.line.y=element_line(size=0.5,colour="black"),
      axis.line=element_line(size=1,colour="black"),
      strip.text.x=element_text(colour=1,size=12,family="serif",face="bold"),
      strip.background = element_rect(colour="black", fill="snow2"),
      panel.grid.major=element_blank(),
      panel.grid.minor=element_blank(),
      panel.border=element_rect(color="black"),
      panel.background=element_blank(),
      axis.text.x=element_text(colour="black",size=12,family="serif",angle=0),
      axis.text.y=element_text(colour=1,size=12,family="serif",face="italic"),
      legend.position="none")+
 scale_x_discrete(name="Espécie")+
 scale_y_continuous(name="Número de indivíduos",
                 limits=c(0,8))
```



Obtendo os parâmetros fitossociológicos da estrutura horizontal:

1. Densidade absoluta (DA_i): Para obter a DA_i é necessário o conhecimento do número de individuos amostrados da *i*-ésima espécie (n_i) e da área total amostra (A):

$$DA_i = \frac{n_i}{A}$$

- *Araucaria angustifolia*

```
DAi <- function(x, A){
  ni <- nrow(subset(FOM, Especie=="Araucaria angustifolia"))
  DAi <- ni/A
  return(DAi)
}

DAi(x = FOM$Especie, A = 0.3)
## [1] 46.66667
```

2. Densidade relativa (DR_i): Para obter a densidade relativa de cada espécie basta usar a fórmula abaixo. Em que: s = número total de espécies observadas.

$$DR_i = \frac{DA_i}{\sum_{i=1}^{s}(DA_i)}$$

- *Araucaria angustifolia*

```
DRi <- function(x, A){
  ni <- nrow(subset(FOM, Especie=="Araucaria angustifolia"))
  DAi <- ni/A
  DTA <- length(x)/A
  DRi <- (DAi/DTA)*100
  return(DRi)
}

DRi(x = FOM$Especie, A = 0.3)
## [1] 20
```

3. Dominância absoluta (DoA_i) - $m^2.ha^{-1}$: Para obter a dominância absoluta é necessário o cálculo prévio das áreas transversais (g_j) da *j*-ésima árvore. Em seguida, a área basal da espécie (G_i) é obtida pela soma das g_j para cada espécie. Por fim, a DoA_i da *i*-ésima espécie é calculada pela razão da G_i pela área total amostrada (A):

$$\begin{cases} DoA_i = \frac{\sum_{j=1}^{n_i} g_j}{A} = \frac{G_i}{A} \\ g_j = \frac{\pi}{40000}.DAP_j^2 \end{cases}$$

- *Araucaria angustifolia*

```
DoAi <- function(data, A, ...){
  data <- data[Especie=="Araucaria angustifolia"]
  gi <- data[, .(gi=pi*DAP^2/40000)]
  Gi <- sum(gi)
  DoAi <- Gi/A
  return(DoAi)
}

DoAi(data=FOM, A=0.3)
## [1] 11.15996
```

4. Dominância relativa (DoR_i): A dominância relativa da *i*-ésima espécie é calculada pela razão da dominância absoluta de cada espécie (DoA_i) pela dominância total (soma dos valores de DoA_i para cada espécie). Alternativamente, pode-se fazer a razão da área basal da *i*-ésima espécie (G_i) pela soma das áreas basais de todas as espécies amostradas (G_T):

$$DoR_i = \frac{DoA_i}{\sum_{i=1}^{s}(DoA_i)}.100 = \frac{G_i}{G_T}.100$$

- *Araucaria angustifolia*

```
DoRi <- function(data, A, ...){
 Gt <- data[, .(gi=pi*DAP^2/40000)]
 data <- data[Especie=="Araucaria angustifolia"]
 gi <- data[, .(gi=pi*DAP^2/40000)]
 Gi <- sum(gi)
 DoAi <- Gi/A
 DoRi <- (Gi/sum(Gt))*100
 return(DoRi)
}

DoRi(data=FOM, A=0.3)
## [1] 53.50602
```

5. Frequência absoluta (FA_i): A frequência absoluta de cada espécie é calculada pela razão do número de unidades de amostras (U_i) onde foram encontradas a *i*-ésima espécie e o número total de unidades de amostras (U_T). Pode-se multiplicar por 100 para obter o parâmetro relativizado:

$$FA_i = \frac{U_i}{U_T}.100$$

- *Araucaria angustifolia*

```
FAi <- function(data, ...){
 Ut <- length(unique(data$Parcela))
 Ui <- unique(data, by=c("Especie", "Parcela"))[, .(Ui=.N), by="Especie"]
 Ui <- Ui[Especie=="Araucaria angustifolia", Ui]
 FAi <- (Ui/Ut)*100
 return(FAi)
}

FAi(data=FOM)
## [1] 100
```

6. Frequência relativa (FR_i): A frequência relativa da *i*-ésima espécie é calculada pela razão da frequência absoluta de cada espécie (FA_i) pela frequência total (soma dos valores de FA_i para cada espécie):

$$FR_i = \frac{FA_i}{\sum_{i=1}^{s}(FA_i)}.100$$

- *Araucaria angustifolia*

```
FRi <- function(data, ...){
 Ut <- length(unique(FOM$Parcela))
 Ui <- unique(FOM, by=c("Especie", "Parcela"))[, .(Ui=.N), by="Especie"]
 FAi <- Ui[, .(FAi=(Ui/length(unique(FOM$Parcela)))*100)]
```

```
 Ui_AA <- Ui[Especie=="Araucaria angustifolia", Ui]
 FAi_AA <- (Ui_AA/Ut)*100
 FRi <- (FAi_AA /sum(FAi))*100
 return(FRi)
}

FRi(data=FOM)
## [1] 7.894737
```

7. Valor de cobertura (VC_i): O valor de cobertura integra os parâmetros de densidade e dominância relativa, isto é, o VC_i é soma de DR_i e DoR_i de cada espécie:

$$VC_i = DR_i + DoR_i$$

- *Araucaria angustifolia*

```
VCi <- DRi(x = FOM$Especie, A = 0.3) + DoRi(data=FOM, A=0.3)
VCi
## [1] 73.50602
```

8. Porcentagem de cobertura (PC_i): Por extensão, pode-se obter a porcentagem de cobertura da *i*-ésima espécie fazendo-se a média de DR_i e DoR_i:

$$PC_i = \frac{DR_i + DoR_i}{2}$$

- *Araucaria angustifolia*

```
PCi <- (DRi(x = FOM$Especie, A = 0.3) + DoRi(data=FOM, A=0.3))/2
PCi
## [1] 36.75301
```

9. Valor de importância (VI_i): O valor de importância integra os parâmetros de densidade, dominância e frequência relativa, isto é, o VI_i é soma de DR_i, DoR_i e FR_i de cada espécie:

$$VI_i = DR_i + DoR_i + FR_i$$

- *Araucaria angustifolia*

```
VIi <- DRi(x = FOM$Especie, A = 0.3) + DoRi(data=FOM, A=0.3) + FRi(data=FOM)
VIi
## [1] 81.40076
```

10. Porcentagem de importância (PI_i): Por extensão, pode-se obter a porcentagem de importância da *i*-ésima espécie fazendo-se a média de DR_i, DoR_i e FR_i:

$$PI_i = \frac{DR_i + DoR_i + FR_i}{3}$$

- *Araucaria angustifolia*

```
PIi <- (DRi(x = FOM$Especie, A = 0.3) + DoRi(data=FOM, A=0.3) + FRi(data=FOM))/3
PIi
## [1] 27.13359
```

Inicialmente, para melhor compreensão dos cálculos os parâmetros da estrutura horizontal foram obtidos apenas para a espécie *Araucaria angustifolia*. Porém, é possível obter todas as estimativas criando-se apenas uma única função no ambiente R. A seguir é apresentada uma função genérica para obter os parâmetros fitossociológicos da estrutura horizontal. A função **EH** recebe quatro parâmetros: **species** = vetor contendo as espécies inventariadas; **sample** = vetor indicando as parcelas de ocorrência das espécies; **d** = vetor com diâmetro das árvores; e **A** = escalar indicando a área total amostrada:

```
# Uma função genérica
EH <- function(species, sample, d, A,...){
  DT <- data.table(species=species, sample=sample, d=d)
  DT <- DT[,`:=`(gi=pi*d^2/40000)]
  Ui <- unique(DT, by=c("species", "sample"))[, .(Ui=.N), by="species"][order(species)]
  ni <- DT[, .(ni=.N, Gi = sum(gi)), by="species"]
  ni <- ni[Ui, on="species"]
  EH <- ni[,DAi := ni/A,
         ][,DRi := (DAi/sum(DAi))*100,
           ][,DoAi := Gi/A,
             ][,DoRi := (DoAi/sum(DoAi))*100,
               ][,VCi := DRi + DoRi,
                 ][,PCi := VCi/2,
                   ][,FAi := (Ui/length(unique(DT$sample)))*100,
                     ][,FRi := (FAi/sum(FAi))*100,
                       ][,VIi := DRi + DoRi + FRi,
                         ][,PIi := VIi/3][order(-VIi)]
  return(EH)
}

EH <- EH(species=FOM$Especie, sample=FOM$Parcela, d=FOM$DAP, A=0.3)
```

Species	ni	Gi	Ui	DAi	DRi	DoAi	DoRi	VCi	PCi	FAi	FRi	VIi	PIi
Araucaria angustifolia	14	3.3479867	3	46.666667	20.000000	11.1599555	53.5060230	73.506023	36.7530115	100.00000	7.894737	81.400760	27.133587
Ilex paraguariensis	17	0.5565488	3	56.666667	24.285714	1.8551628	8.8945144	33.180229	16.5901143	100.00000	7.894737	41.074965	13.691655
Matayba eleagnoides	7	0.3805560	3	23.333333	10.000000	1.2685201	6.0818762	16.081876	8.0409381	100.00000	7.894737	23.976613	7.992204
Ocotea porosa	2	0.7369901	2	6.666667	2.857143	2.4566338	11.7782464	14.635389	7.3176946	66.66667	5.263158	19.898547	6.632849
Campomanesia xanthocarpa	4	0.1482714	3	13.333333	5.714286	0.4942380	2.3696070	8.083893	4.0419463	100.00000	7.894737	15.978630	5.326210
Nectandra grandiflora	3	0.0675364	2	10.000000	4.285714	0.2251213	1.0793363	5.365051	2.6825253	66.66667	5.263158	10.628208	3.542736
Allophyllus edulis	3	0.0515677	2	10.000000	4.285714	0.1718922	0.8241314	5.109846	2.5549229	66.66667	5.263158	10.373004	3.457668
Mimosa scabrella	2	0.1188307	2	6.666667	2.857143	0.3961025	1.8990997	4.756242	2.3781213	66.66667	5.263158	10.019400	3.339800
Nectandra megapotamica	2	0.1030953	2	6.666667	2.857143	0.3436510	1.6476227	4.504766	2.2523828	66.66667	5.263158	9.767924	3.255975
Schinus terebinthifolius	2	0.0780097	2	6.666667	2.857143	0.2600322	1.2467156	4.103859	2.0519293	66.66667	5.263158	9.367016	3.122339
Sapium glandulatum	2	0.0568133	2	6.666667	2.857143	0.1893778	0.9079655	3.765108	1.8825542	66.66667	5.263158	9.028266	3.009422
Cedrela fissilis	1	0.1170212	1	3.333333	1.428571	0.3900706	1.8701802	3.298752	1.6493758	33.33333	2.631579	5.930331	1.976777
Ilex dumosa	1	0.1000982	1	3.333333	1.428571	0.3336607	1.5997248	3.028296	1.5141481	33.33333	2.631579	5.659875	1.886625
Luehea divaricata	1	0.0934820	1	3.333333	1.428571	0.3116067	1.4939877	2.922559	1.4612796	33.33333	2.631579	5.554138	1.851379
Tabebuia avellanedae	1	0.0688134	1	3.333333	1.428571	0.2293782	1.0997457	2.528317	1.2641585	33.33333	2.631579	5.159896	1.719965
Piptocarpha angustifolia	1	0.0611362	1	3.333333	1.428571	0.2037873	0.9770510	2.405622	1.2028112	33.33333	2.631579	5.037201	1.679067
Drymis brasiliensis	1	0.0444881	1	3.333333	1.428571	0.1482936	0.7109888	2.139560	1.0697801	33.33333	2.631579	4.771139	1.590380
Vernonia discolor	1	0.0444881	1	3.333333	1.428571	0.1482936	0.7109888	2.139560	1.0697801	33.33333	2.631579	4.771139	1.590380
Capsicodendron dinnisii	1	0.0376685	1	3.333333	1.428571	0.1255616	0.6020008	2.030572	1.0152861	33.33333	2.631579	4.662151	1.554050
Ocotea pulchella	1	0.0141026	1	3.333333	1.428571	0.0470087	0.2253816	1.653953	0.8269765	33.33333	2.631579	4.285532	1.428511
Eugenia uniflora	1	0.0136848	1	3.333333	1.428571	0.0456159	0.2187040	1.647275	0.8236377	33.33333	2.631579	4.278854	1.426285
Maytenus ilicifolia	1	0.0081713	1	3.333333	1.428571	0.0272376	0.1305898	1.559161	0.7795806	33.33333	2.631579	4.190740	1.396913
Cupania vernalis	1	0.0078540	1	3.333333	1.428571	0.0261799	0.1255188	1.554090	0.7770451	33.33333	2.631579	4.185669	1.395223

Referencial teórico

1. Manuais técnicos

R Language Definition

An Introduction to R - is based on the former "Notes on R", gives an introduction to the language and how to use R for doing statistical analysis and graphics.

2. Legislação

BRASIL. Resolução nº 406, de 2 de fevereiro de 2009. Estabelece parâmetros técnicos a serem adotados na elaboração, apresentação, avaliação técnica e execução de Plano de Manejo Florestal Sustentável - PMFS com fins madeireiros, para florestas nativas e suas formas de sucessão no bioma Amazônia. Diário Oficial [da República Federativa do Brasil], Brasília, nº 26, p. 100, 6 de fevereiro de 2009. Seção 1.

3. Pacotes

DOWLE, M.; SRINIVASAN, A. **data.table: Extension of 'data.frame'**. R package version 1.10.4-3, 2017.

DRAGULESCU, A. A. **xlsx: Read, write, format Excel 2007 and Excel 97/2000/XP/2003 files**. R package version 0.5.7, 2014.

FAN, F. Y. **FinCal: Time Value of Money, Time Series Analysis and Computational Finance**. R package version 0.6.3., 2016.

FARAWAY, J. **faraway: Functions and Datasets for Books by Julian Faraway**. R package version 1.0.7, 2016.

FOX, J.; WEISBERG, S. **An R Companion to Applied Regression**. Second ed. Thousand Oaks CA: Sage, 2011.

WICKHAM, H. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2009.

MAKIYAMA, K. **magicfor: Magic Functions to Obtain Results from for Loops**. R package version 0.1.0, 2016.

NAVARRO, D. **Learning statistics with R: A tutorial for psychology students and other beginners. (Version 0.5)**. Adelaide, Australia: University of Adelaide, 2015.

PONCET, P. **modeest: Mode Estimation**. R package version 2.1, 2012.

R CORE TEAM. **R: A Language and Environment for Statistical Computing**. Vienna, Austria: R Foundation for Statistical Computing, 2017.

ZEILEIS, A.; HOTHORN, T. Diagnostic Checking in Regression Relationships. **R News**, v. 2, n. 3, p. 7–10, 2002.

4. Livros

MONTGOMERY, D. C.; RUNGER, G. C. **Inferência estatística para uma única amostra. Estatística aplicada e probabilidade para engenheiros**. Rio de Janeiro. 2003.

MUELLER-DOMBOIS, D.; ELLENBERG, H. **Aims and methods of vegetation ecology**. 1974.

PÉLLICO NETO, S.; BRENA, D. **Inventário florestal**. Curitiba. 1997.

GOMES, F. P. **Curso de estatística experimental**. Nobel, 1987.

QUEIROZ, W. T. DE. **Técnicas de amostragem em inventário florestal nos trópicos**. Ministério da Educação e do Desporto, Faculdade de Ciências Agrárias de Pará, Serviço de Documentação e Informação, 1998.

SANQUETTA, C. et al. **Inventários florestais: planejamento e execução**. 3. ed. Curitiba PR: Curitiba: Multi-Graphic, 2014.

SILVA, J. A. A.; SILVA, I. P. **Estatística experimental aplicada à ciência florestal**. Recife. 1982.

SOUZA, A. L.; SOARES, C. P. B. **Florestas Nativas: estrutura, dinâmica e manejo**. Viçosa MG: UFV, 2013.

5. Artigos

FISHER, R. A. **Studies in Crop Variation. I. An examination of the yield of dressed grain from Broadbalk.** The Journal of Agricultural Science, v. 11, n. 2, p. 107-135, 1921.

KENNEY, J. F.; KEEPING, E. S. **Linear regression and correlation**. Mathematics of statistics, v. 1, p. 252-285, 1962.

STUDENT. **The probable error of a mean**. Biometrika, p. 1-25, 1908.

Agência Brasileira do ISBN
ISBN 978-85-917357-3-0