

Alberto Vicentini e Ricardo de Oliveira Perdiz

Curso básico de introdução à linguagem R

Disciplina BOT89 - PPGBOT INPA



Sumário

Lista de Figuras	xiii
Lista de Tabelas	xv
Prefácio	xvii
Sobre os autores	xxxv
I Parte I	1
1 Conceitos introdutórios	5
1.1 Console e scripts	5
1.2 Linguagem objeto-orientada	7
1.3 R-base e pacotes	10
1.4 Ajuda no R	11
1.5 Área de trabalho vs. Pasta de Trabalho	14
1.6 Dicas de organização do trabalho	17
1.7 Dicas de erros comuns de sintaxe da linguagem R . .	19
1.7.1 Parênteses	19
1.7.2 Vírgulas	20
1.7.3 Aspas	21
1.7.4 Números	21
1.7.5 Nomes de objetos	22

1.7.6	Atribuição de objetos	23
1.7.7	Busque entender as partes	24
1.8	O R como calculadora	28
1.8.1	Operadores	28
1.8.2	Precedência de parênteses	31
1.8.3	Funções e constantes matemáticas	32
1.8.4	NA e valores afins	36
1.9	Exercícios	38
2	Objeto I - Vetores	39
2.1	Vetores e operações vetoriais I	39
2.1.1	Criação de Vetores	39
2.1.2	Sequências Numéricas & Repetições	42
2.2	Operações Matemáticas com Vetores	45
2.3	Funções com Vetores	48
2.4	Classes de vetores e fatores	51
2.5	Indexação	60
2.5.1	Usando índices numéricos	61
2.5.2	Usando índices de nomes	62
2.6	Vetores e operadores lógicos	64
2.6.1	Fazendo perguntas à vetores	64
2.6.2	Filtrando dados com vetores lógicos	73
2.6.3	Perguntando por valores ausentes - NA	75
2.7	Para saber mais:	78
2.8	Exercícios	79

3 Objeto II - Matrizes e data.frames	81
3.1 Matriz vs. data.frame	81
3.1.1 Criando matrizes	81
3.1.2 Criando data.frames	86
3.1.3 Funções importantes na manipulação de matrizes e data.frames	90
3.2 Indexação de matriz e data.frame	98
3.2.1 Matrizes	98
3.2.2 data.frame	102
3.3 Filtrando e ordenando matrizes e data.frames	106
3.3.1 Filtragem de dados	106
3.3.2 Ordenação de dados	111
3.4 Importando e exportando dados no R	117
3.4.1 Arquivos de texto simples para estocar dados	118
3.4.2 Importando dados	120
3.4.3 Exportando dados	126
3.4.4 Outras funções úteis	128
3.5 Para saber mais:	130
3.6 Exercícios	130
4 Objeto III - Listas e objetos complexos	131
4.1 Listas	131
4.2 Objetos complexos	143
5 Funções gráficas	147
5.1 Dispositivos Gráficos	147
5.1.1 Função plot()	147
5.1.2 Dispositivos de Tela	148

5.1.3	Listar e controlar dispositivos	149
5.1.4	Dispositivos de arquivos	151
5.1.5	Figuras vetoriais em pdf ou postscript	151
5.1.6	Figuras raster	153
5.2	Parâmetros gráficos, parte I - Margem, fonte, proporções	154
5.2.1	Margem da figura	163
5.2.2	Aspecto dos eixos	164
5.2.3	Proporção dos eixos	166
5.2.4	Múltiplas figuras na mesma tela ou página com <code>mflow()</code> e <code>mfcol()</code>	167
5.2.5	Múltiplas figuras na mesma image usando a função <code>layout()</code>	170
5.3	Parâmetros gráficos, parte II - Símbolos e cores	171
5.3.1	Tipo de símbolo - <code>pch</code>	172
5.3.2	Tamanho dos pontos - <code>cex</code>	177
5.3.3	Linhas	180
5.3.4	Cores de símbolos - <code>col</code> e <code>bg</code>	183
5.3.5	Funções que definem cores	188
5.4	Funções gráficas de alto nível	201
5.4.1	<code>plot()</code> - uma função genérica	202
5.4.2	<code>hist()</code>	207
5.4.3	<code>boxplot()</code>	209
5.4.4	<code>barplot()</code>	211
5.4.5	<code>plot.phylo()</code>	215
5.4.6	<code>image()</code> e <code>contour()</code>	217
5.4.7	<code>map()</code>	221
5.4.8	<code>xlim</code> e <code>ylim</code>	223

5.5	Funções gráficas de baixo nível	224
5.5.1	legend()	225
5.5.2	points()	229
5.5.3	text() e mtext()	233
5.5.4	axis()	236
5.5.5	abline()	240
5.5.6	arrows(), rect(), polygon() e segments() . . .	241
5.5.7	symbols()	246
5.6	Funções gráficas interativas	249
5.6.1	locator()	249
5.6.2	identify()	250
5.7	Para saber mais:	251
5.8	Exercícios	252
6	Iteração e controle de fluxo	253
6.1	Funções da família apply()	253
6.1.1	Em uma matriz	253
6.1.2	Em um vetor ou lista	257
6.1.3	Por categoria de um fator	261
6.2	Condicionais	263
6.2.1	Condicional if ()	264
6.2.2	Condicional if () com o else	265
6.2.3	Condicional ifelse()	268
6.3	Iterações	277
6.3.1	Iteração com for(){}	277
6.3.2	Iteração com while(){}	283
6.3.3	Iteração com for(){} e a condicional if(){} .	284

6.4	Criando ou modificando funções	285
6.4.1	Exemplo I	286
6.4.2	Exemplo II	287
6.5	Para saber mais:	290
7	Sumarização de dados	291
7.1	Tabelas dinâmicas	291
7.2	Tabelas de contagem	296
7.3	Lógica da junção de tabelas	299
7.3.1	Maneira 1 - função <code>match()</code>	303
7.3.2	Maneira 2 - índices nominais	304
7.4	Junção de tabelas utilizando funções	306
7.4.1	Dados para nossa prática	306
7.4.2	Maneira 3 - função <code>merge()</code>	308
7.5	Para saber mais	327
7.6	Exercícios	327
8	Manipulação de textos, arquivos e pastas	329
8.1	Funções para manipulação e busca de textos	329
8.1.1	Colar ou concatenar textos	329
8.1.2	Quebrar ou desconectarar textos	331
8.1.3	Altera caixa do texto	335
8.1.4	Remove espaços em branco	337
8.1.5	Substitui ou pega parte de texto	338
8.1.6	Expressões Regulares	339
8.1.7	Remove Acentos	341
8.1.8	Metacaracteres	342
8.1.9	Classes de caracteres	346

8.1.10 Barra invertida \	349
8.2 Manipulação de pastas e arquivos	351
8.3 Para saber mais:	353
9 Amostragens aleatórias	355
9.1 Funções para gerar permutações e simular dados	355
9.2 Distribuições aleatórias	359
9.2.1 Distribuição Normal	360
9.3 Para saber mais:	367
9.4 Exercícios	367
10 Extraíndo dados de colunas descritivas	369
10.1 Dados categóricos	369
10.1.1 Passo 01 - Lista de referência	369
10.1.2 Passo 02 - Funções que fazem a busca	370
10.1.3 Passo 03 - Usando a função	372
10.2 Listas de referência	373
10.2.1 Hábito da planta	373
10.2.2 Estado de fertilidade da amostra	374
10.2.3 Classes de habitat	374
10.2.4 Textura do solo	375
10.2.5 Exsudato	375
10.3 Extraíndo altura e dap	376
10.3.1 Altura	376
10.3.2 DAP	379
10.4 Usando essas funções	381
10.4.1 Obtendo valores de altura	382
10.4.2 Obtendo valores de DAP	395

II Parte II	427
11 Checagem dos dados	431
11.1 Tem valores ausentes?	434
11.2 Colunas com fatores	445
12 AED de univariadas	451
12.1 Qual a distribuição dos valores numéricos?	451
12.1.1 Dados do tutorial	451
12.2 Tabelas de variáveis categóricas	467
12.2.1 Resumo de gráficos univariados	471
12.3 As variáveis têm distribuição normal?	478
12.4 Para saber mais:	492
13 AED de bivariadas	493
13.1 Dados do tutorial	493
13.2 Fatores e contagens	494
13.3 Variável numérica vs. fator	503
13.4 Variável numérica vs. numérica	506
13.5 Outros gráficos bivariados	516
14 AED de multivariadas	523
14.1 Matrizes de distância	523
14.2 Ordenação com matrizes de distância	526
14.3 Escalonamento Não-Métrico Multidimensional (NMDS)	526
14.3.1 Exemplo com dados morfológicos	530
14.4 Análises de agrupamento	535

<i>Contents</i>	<i>xi</i>
14.4.1 Agrupamento pelo método da mínima variância	537
14.4.2 Agrupamento por UPGMA	537
14.4.3 Agrupamento por centróides	538
14.4.4 Exemplo florístico	543
14.4.5 Análise de Coordenadas Principais (PCoA)	550
14.5 Componentes Principais (PCA)	559
Apêndice	565
A Base R vs. Tidyverse	567
A.1 O que é o Tidyverse?	567
A.1.1 dplyr e ggplot2, símbolos do Tidyverse	569
A.2 Usando o dplyr	570
A.2.1 Selecionando colunas com select()	570
A.2.2 Filtrando dados com filter()	573
A.2.3 Destrinchando as funções group_by e summarise	575
A.3 O operador %>% e o encadeamento de ações	578
A.3.1 Conjunto de comandos 1	582
A.3.2 Conjunto de comandos 2	582
A.3.3 Resumo do operador %>%:	586
A.4 Usando o ggplot2	586
A.5 dplyr e ggplot2 em conjunto	591
A.5.1 Gráfico de espalhamento	592
A.5.2 Diagrama de caixas	592
A.5.3 Histograma	593
A.6 Comparando o mesmo conjunto de ações entre base R e dplyr	594

A.6.1 E o uso do operador <code>%>%</code> com o pacote <code>base</code> ?	599
A.7 Tidyverse, um resumo	601
A.8 Para saber mais:	602
B Baixar e descomprimir um arquivo	605
C Aulas em vídeo	607
C.1 Importando e exportando dados	607
C.2 Criação de vetores	607
C.3 Classes de vetores e fatores	607
C.4 Sequências numéricas e repetições, operações e funções com vetores	608
C.5 Listas	608
C.6 Criando matrizes	608
C.7 Criando dataframes, e operações importantes em matrizes e dataframes	608
C.8 Indexação de matrizes e dataframes	608
C.9 Condicionais	609
C.10 Funções da família <code>apply()</code>	609
C.11 Funções de manipulação de pasta e arquivos	609
C.12 Funções de manipulação de texto - parte 01	609
C.13 Funções de manipulação de texto - parte 02	609
C.14 Objetos complexos	610
D Caras de Chernoff	611

Lista de Figuras

1	Imagen inicial do repositório hospedado no GitHub contendo o código fonte deste livro.	xxii
2	Destaque do botão "Code" (retângulo "azul" pontilhado circundando o botão, com uma seta "azul" apontando para o mesmo botão) presente na página inicial do repositório hospedado no GitHub contendo o código fonte deste livro. Há um outro retângulo de cor "vermelha" destacando a opção 'Download Zip.'	xxiii
3	Curvas de aprendizagem do aluno Sabichão (Geek) e de um aluno que tem medo de código (non-geek). . .	xxx
1.1	Imagen de uma sessão do R para Mac OSX com um janela de script aberta além do console.	5
1.2	Imagen de uma sessão do RStudio com três painéis abertos.	6
7.1	Fonte: www.sqlfromhell.com	309
7.2	Fonte: www.sqlfromhell.com	313
7.3	Fonte: www.sqlfromhell.com	316
7.4	Fonte: www.sqlfromhell.com	319
7.5	Fonte: www.sqlfromhell.com	321
7.6	Fonte: www.sqlfromhell.com	326

- A.1 Porcentagem de visitas às questões pertinentes a algumas linguagens de programação na plataforma *Stack Overflow*. Dados obtidos apenas de países desenvolvidos segundo o Banco Mundial. Nota-se o aumento quase linear referente à linguagem R a partir do ano 2014. Figura extraída desta postagem: <https://stackoverflow.blog/2017/10/10/impressive-growth-r/> 568

Lista de Tabelas

7.1	Tabela 1	307
7.2	Tabela 2	307
7.3	Tabela 3	308
7.4	Tabela 1	313
7.5	Tabela 2	318
7.6	Tabela 3	323
A.1	Principais funções do pacote R ‘dplyr’	571
A.2	Principais funções do pacote R ‘ggplot2’	586



Prefácio

Este livro foi criado inicialmente para servir como um material de apoio básico aos estudantes da disciplina Preparação de dados para Análises Estatísticas - Introdução ao R (BOT-89) do Programa de Pós-Graduação em Ciências Biológicas (Botânica)¹ do INPA, ofertada desde o ano de 2011.

Inicialmente, a disciplina consistiu de uma modificação da documentação do curso de Introdução à Linguagem R² oferecida pelo Programa de Pós-graduação em Ecologia³ do Instituto de Biociências da Universidade de São Paulo, atualmente ministrada pelo professor Dr. Alexandre Adalardo de Oliveira⁴, e que contem aulas de autoria dos professores Drs. A. A. de Oliveira, João Luís Ferreira Batista⁵, Paulo Inácio K. L. Prado⁶, e Rodrigo Augusto Santinelo Pereira ([Batista et al., 2009](#)). Como exemplo, parte dos dados (dados de avistamento de aves no Cerrado⁷ e Levantamento em Caixetais⁸) utilizados em nossas aulas são originários desta disciplina da Ecologia da USP.

Modificações nessa estrutura foram feitas por Alberto Vicentini ao longo dos últimos 10 anos em disciplinas anuais ofertadas no Programa de pós-graduação em Ciências Biológicas (Botânica) do Instituto Nacional de Pesquisas da Amazônia (INPA). Ricardo Perdiz transpôs as aulas originais formatadas em *dokuwiki* para o formato

¹<http://www.portais.atrio.scire.net.br/inpa-ppgbot/index.php/pt/>

²<http://ecor.ib.usp.br/doku.php>

³<https://poseecologia.ib.usp.br/>

⁴<http://labtrop.ib.usp.br/>

⁵<http://cmq.esalq.usp.br/>

⁶<http://ecologia.ib.usp.br/let/doku.php?id=prado:start>

⁷https://github.com/LABOTAM/IntroR/blob/main/dados/aves_cerrado.csv

⁸<https://github.com/LABOTAM/IntroR/blob/main/dados/caixeta.csv>

Rmarkdown⁹. Também fez correções e acrescentou (poucos) novos conteúdos.

Nossa meta é prover uma introdução básica da linguagem de programação R, que é extremamente poderosa para manipulação de dados, análises estatísticas, produção de gráficos e de documentos dinâmicos, e cujo domínio oferece um grande ganho de produtividade a todo cientista e facilita o entendimento de métodos analíticos. Ressaltamos que **isto não é um curso de estatística**.

Estrutura do livro



Este livro está em constante atualização. Você pode encontrar os arquivos de Rmarkdown que geraram este livro neste endereço: <https://github.com/LABOTAM/IntroR/tree/main>. Se você tiver alguma contribuição a fazer, seja na forma de correções, críticas, ou o desejo de contribuir ativamente com o projeto, sinta-se à vontade para abrir um problema no repositório do livro: <https://github.com/LABOTAM/IntroR/issues>. Grato!

O livro está dividido em três partes. A primeira parte contém 10 capítulos e aborda os ensinamentos básicos para o usuário aprender a usar a linguagem como uma ferramenta científica.

- **Capítulo 1** aborda os aspectos básicos da linguagem R, o software que utilizamos para usar a linguagem, pacotes com funções, como solicitar ajuda sobre funções, dicas de organização das pastas, e o R como uma calculadora.
- **Capítulo 2** aborda a criação, indexação, e operações matemáticas com vetores no R.

⁹<https://rmarkdown.rstudio.com/>

- **Capítulo 3** aborda a criação, indexação, filtragem, importação e exportação de matrizes e data.frames.
- **Capítulo 4** ensina a criar, indexar, e filtrar listas e objetos complexos.
- **Capítulo 5** aborda as funções gráficas do pacote `base` do R.
- **Capítulo 6** aborda dois conceitos chave em qualquer linguagem de programação: iteração e controles de fluxo.
- **Capítulo 7** abrange o tópico de sumarização de dados, explicando tabelas de contagem, tabelas dinâmicas e junção de tabelas.
- **Capítulo 8** ensina ferramentas básicas para a manipulação de textos, arquivos e pastas.
- **Capítulo 9** ensina de maneira breve funções para amostragens aleatórias e distribuições de valores.
- **Capítulo 10** abrange o uso de funções customizadas para extração de variáveis morfológicas de notas de texto obtidas de espécimes de plantas.

A segunda parte contem tutoriais curtos de análise exploratória de dado (AED). A AED **deve ser iniciada ainda durante a coleta de dados**, pois através dela podemos conhecer nossos dados e acompanhar como estamos coletando esses dados. AED baseia-se largamente em técnicas visuais (gráficos) e pode levar entre 20 a 50% do tempo das análises. Os capítulos abaixo exemplificam algumas ferramentas do R para fazer AED:

- **Capítulo 11** aborda a checagem inicial dos dados.
- **Capítulo 12** aborda a AED em dados univariados.
- **Capítulo 13** aborda a AED em dados bivariados.
- **Capítulo 14** aborda a AED em dados multivariados.

Por fim, temos o apêndice contendo materiais suplementares:

- **Apêndice A** inclui uma discussão breve sobre o universo de pacotes conhecidos como Tidyverse¹⁰, comparando-os com o pacote `base` do R.

¹⁰<https://www.tidyverse.org/>

- **Apêndice B** apresenta um tutorial para baixar e descomprimir arquivos .zip dentro R.
 - **Apêndice C** apresenta uma lista de vídeos gravados por nós, autores, abordando diversos tópicos deste livro e que podem auxiliar o aprendizado da linguagem.
 - **Apêndice D** apresenta um código para gerar as caras de Chernoff¹¹, que é uma maneira divertida de aprender a lidar com dados multivariados.
-

Agradecimentos

Agradecemos aos Drs. Alexandre Oliveira, João Luís Ferreira Batista, Paulo Inácio K. L. Prado, e Rodrigo Augusto Santinelo Pereira, por permitirem o uso de conteúdo parcial da disciplina introdutória ministrada por eles na USP como base para a criação deste livro.

Pré-requisitos

Devido ao modo como foi construído, este livro pode ser tanto lido quanto “executado”, uma vez que seus arquivos são o que chamamos de *Rmarkdown*, uma linguagem que mistura texto com código. Sendo assim, o livro pode ser visualizado em computadores, em celulares modernos, e em *tablets*. Não é necessário ter um computador próprio, apesar de que possuir um facilita o aprendizado. Entendemos que nosso país ainda carece de muito investimento na melhora da qualidade de vida da população em geral, o que inclui acesso facilitado aos bens de informática. Desta forma, providenciamos duas maneiras de o leitor deste livro fazer bem uso das informações aqui contidas.

¹¹http://www.fics.edu.br/index.php/augusto_guzzo/article/view/67/78

Caso você possua um computador

Instale o R

Baixe e instale o R¹², próprio para o seu sistema operacional.

Instale um ambiente de desenvolvimento integrado - IDE

Recomendamos que o software RStudio¹³ seja instalado no seu computador, pois ele foi criado facilitar a interação do usuário com os múltiplos recursos oferecidos pela linguagem, em especial os inúmeros pacotes desenvolvidos por funcionários desta empresa, entre os quais se inclui o pacote bookdown¹⁴, utilizado para construir este livro online. Ademais, ambientes de desenvolvimento integrado (do inglês *Integrated Development Environment*, **IDE**) como este software têm o objetivo de prover ao usuário mais ferramentas na interação entre o código e o resultado da execução do código, além de facilitar o uso de ferramentas alternativas para construção de texto mesclado com códigos (e.g., LaTeX, Markdown etc), o que permite uma dinamicidade na construção de textos acadêmicos.

Existem outras alternativas de software, tais como o Atom¹⁵ e o Visual Studio Code¹⁶. Fique à vontade para escolher.

Baixe os pacotes abaixo

Serão necessários o uso de alguns pacotes de R (veja seção 1.3 para entender o que são pacotes de R) para poder fazer uso tanto do repositório localmente quanto fazer uso das próprias aulas. Então, rode o comando abaixo para instalar esses pacotes em seu computador:

¹²<https://www.r-project.org/>

¹³<https://www.rstudio.com/products/rstudio/download/>

¹⁴<https://github.com/rstudio/bookdown>

¹⁵<https://atom.io/>

¹⁶<https://code.visualstudio.com/>

```
install.packages(c('rmarkdown', 'bookdown', 'knitr', 'kableExtra', 'ape', 'labdsv', 'vegan', 'gridExtra'))
```

Há uma lista de pacotes que serão utilizados apenas no apêndice A. Eles não são necessários para acompanhar o curso, mas caso você chegue a este apêndice, eles serão necessários. Portanto, caso queira já instalá-los, execute o comando abaixo:

```
install.packages(c('ggplot2', 'dplyr', 'tibble', 'tidyverse', 'purrr', 'magrittr'))
```

Baixe o repositório para seu computador

Acesse a página do repositório¹⁷ contendo os código fonte deste livro (Fig. 1).

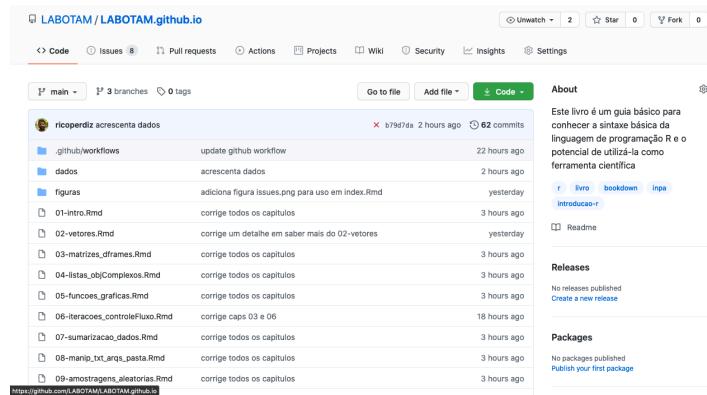


Figura 1: Imagem inicial do repositório hospedado no GitHub contendo o código fonte deste livro.

Busque um botão de cor **verde** chamado *Code* (Fig. 2). Clique neste botão e aparecerão algumas opções. Clique no botão **Download Zip**, destacado em vermelho na figura 2, e escolha onde você quer salvar o arquivo.

¹⁷<https://github.com/LABOTAM/IntroR>

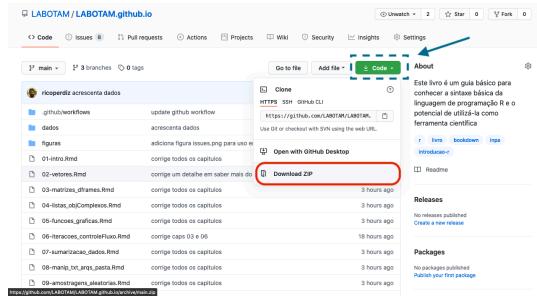


Figura 2: Destaque do botão "Code" (retângulo "azul" pontilhado circundando o botão, com uma seta "azul" apontando para o mesmo botão) presente na página inicial do repositório hospedado no GitHub contendo o código fonte deste livro. Há um outro retângulo de cor "vermelha" destacando a opção 'Download Zip.'

Após baixar, descomprima este arquivo .zip, escolha onde você deseja guardar a pasta, e comece a acessar os arquivos.

Os arquivos contendo cada capítulo deste livro são listados abaixo e podem ser encontrados na página inicial do repositório¹⁸:

```
## 00-autores.Rmd
## 01-intro.Rmd
## 02-vetores.Rmd
## 03-matrizess_dframes.Rmd
## 04-listas_objComplexos.Rmd
## 05-funcoes_graficas.Rmd
## 06-iteracoes_controleFluxo.Rmd
## 07-sumarizacao_dados.Rmd
## 08-manip_txt_arqs_pasta.Rmd
## 09-amostragens_aleatorias.Rmd
## 10-extracao_dados.Rmd
## 11-aed_checadados.Rmd
## 12-aed_checaunivar.Rmd
## 13-aed_checabivar.Rmd
## 14-aed_checamultivar.Rmd
## 15-tidyverse.Rmd
```

¹⁸<https://github.com/LABOTAM/IntroR>

```
## 16-baixa_descomprime.Rmd  
## 17-videos_html.Rmd  
## 17-videos_pdf.Rmd  
## 18-chernoff.Rmd  
## 19-referencias.Rmd
```

Você pode tanto utilizar os arquivos escritos em formato Rmarkdown¹⁹ e executar os códigos em cada célula de código (**code chunks**) ou usar os scripts contidos na pasta `codigo`, que nada mais são do que os capítulos do livro convertidos para um script de R, isto é, em formato `.r`. Nesses scripts, todos os pedaços de texto, incluindo cabeçalhos e comentários estão precedidos de `#'`; o que for código aparecerá sem `#'` precedentes. Clique no nome dos scripts abaixo para levá-lo direto ao arquivo dentro do repositório.

- 00-autores.R²⁰
- 01-intro.R²¹
- 02-vetores.R²²
- 03-matrices_dframes.R²³
- 04-listas_objComplexos.R²⁴
- 05-funcoes_graficas.R²⁵
- 06-iteracoes_controleFluxo.R²⁶
- 07-sumarizacao_dados.R²⁷

¹⁹<http://www.botanicaamazonica.wiki.br/labotam/doku.php?id=bot89:precurso:rmarkdown:inicio>

²⁰<https://github.com/LABOTAM/IntroR/tree/main/codigo/00-autores.R>

²¹<https://github.com/LABOTAM/IntroR/tree/main/codigo/01-intro.R>

²²<https://github.com/LABOTAM/IntroR/tree/main/codigo/02-vetores.R>

²³https://github.com/LABOTAM/IntroR/tree/main/codigo/03-matrices_dframes.R

²⁴https://github.com/LABOTAM/IntroR/tree/main/codigo/04-listas_objComplexos.R

²⁵https://github.com/LABOTAM/IntroR/tree/main/codigo/05-funcoes_graficas.R

²⁶https://github.com/LABOTAM/IntroR/tree/main/codigo/06-iteracoes_controleFluxo.R

²⁷https://github.com/LABOTAM/IntroR/tree/main/codigo/07-sumarizacao_dados.R

- 08-manip_txt_arqs_pasta.R²⁸
- 09-amostragens_aleatorias.R²⁹
- 10-extracao_dados.R³⁰
- 11-aed_checadados.R³¹
- 12-aed_checaunivar.R³²
- 13-aed_checabivar.R³³
- 14-aed_checamultivar.R³⁴
- 15-tidyverse.R³⁵
- 16-baixa_descomprime.R³⁶
- 17-videos_html.R³⁷
- 17-videos_pdf.R³⁸
- 18-chernoff.R³⁹
- 19-referencias.R⁴⁰

²⁸https://github.com/LABOTAM/IntroR/tree/main/codigo/08-manip_txt_arqs_pasta.R

²⁹https://github.com/LABOTAM/IntroR/tree/main/codigo/09-amostragens_aleatorias.R

³⁰https://github.com/LABOTAM/IntroR/tree/main/codigo/10-extracao_dados.R

³¹https://github.com/LABOTAM/IntroR/tree/main/codigo/11-aed_checadados.R

³²https://github.com/LABOTAM/IntroR/tree/main/codigo/12-aed_checaunivar.R

³³https://github.com/LABOTAM/IntroR/tree/main/codigo/13-aed_checabivar.R

³⁴https://github.com/LABOTAM/IntroR/tree/main/codigo/14-aed_checamultivar.R

³⁵<https://github.com/LABOTAM/IntroR/tree/main/codigo/15-tidyverse.R>

³⁶https://github.com/LABOTAM/IntroR/tree/main/codigo/16-baixa_descomprime.R

³⁷https://github.com/LABOTAM/IntroR/tree/main/codigo/17-videos_html.R

³⁸https://github.com/LABOTAM/IntroR/tree/main/codigo/17-videos_pdf.R

³⁹<https://github.com/LABOTAM/IntroR/tree/main/codigo/18-chernoff.R>

⁴⁰<https://github.com/LABOTAM/IntroR/tree/main/codigo/19-referencias.R>

Caso você não possua um computador

O Google oferece um serviço chamado Google Colab⁴¹ que pode ser utilizado em computadores, em celulares modernos com acesso à internet e possuídores de um navegador, e em tablets.

Em breve, apresentaremos aqui um endereço contendo todo o conteúdo deste livro já convertido para o formato de um Jupyter Notebook, que é o modelo de arquivo utilizado pelo Google Colab.

O que é o R e por qual razão você deve se preocupar em programar**O R em poucas palavras**

O R é um software livre e gratuito de desenvolvimento coletivo e é muito útil para quem trabalha com ciências e precisa manipular e analisar dados, gerar gráficos e publicações dinâmicas. Tem uma comunidade enorme de usuários e colaboradores e está disponível gratuitamente em diversos repositórios espalhados pelo mundo (<https://cran.r-project.org/>).

O R foi criado para permitir a melhor e mais ampla exploração de dados possível, (...) mas sempre explicando a natureza dos métodos utilizados, num formato aberto (livre) e comprehensível (Chambers, 2008).

⁴¹<https://research.google.com/colaboratory/faq.html>

O R é uma linguagem de programação; você conversa com o R através de um código.

Por ser uma linguagem de programação, **no R você pode fazer tudo**. A pergunta não é sobre **SE** o R faz alguma coisa, mas **COMO** fazer no R alguma coisa. E há várias maneiras de fazer a mesma coisa no R. É fundamental aprender a falar R, pois sabendo falar você pode dizer do seu jeito.

Todo estudante de ciências deveria aprender a programar, porque programar lhe ensina a pensar (Steve Jobs). Usar o R é a melhor forma de aprender estatística porque você precisa de fato entender o que está fazendo ao escrever um código.

Aprender a programar é como aprender um novo idioma; aprendendo o R fica mais fácil aprender outras linguagens de programação e lhe ajuda a aprender inglês, e o inglês lhe facilita falar R.

O R é uma fantástica ferramenta para fazer gráficos; veja exemplos em <http://www.r-graph-gallery.com>.

Com o desenvolvimento da linguagem na última década, você

pode gerar aplicativos (Shiny⁴²) e relatório dinâmicos usando Rmarkdown⁴³, facilitando a formatação gráfica e a reproduzibilidade dos seus artigos, relatórios e projetos científicos.

Por que programar?

A maioria dos programas de computador não é muito boa. O código no seu laptop, televisor, telefone, carro é geralmente mal documentado, inconsistente e pouco testado. Por que isso importa para a ciência? Porque transformar dados brutos (`raw data`) em artigos científicos geralmente requer um pouco de programação, o que significa que a maioria dos cientistas escreve software (Nick Barnes, 2010, Publish your computer code⁴⁴, Nature News).

É tentador tratar todas as coisas como se fossem pregos, se a única ferramenta que você tem é um martelo (Abraham Maslow, 1996, Lei do Instrumento⁴⁵)! Liberte-se da camisa de força dos softwares de estatística que limitam as análises que você pode fazer e como pode fazer.

Um dos objetivos da análise estatística é o de destilar um conjunto longo e complicado de dados em um número pequeno de estatísticas descritivas que façam sentido. Muitos dos pacotes estatísticos de computadores modernos, no entanto, fazem exatamente o oposto disso. Eles produzem automaticamente

⁴²<https://shiny.rstudio.com/>

⁴³<http://www.botanicaamazonica.wiki.br/labotam/doku.php?id=bot89:precurso:rmarkdown:inicio>

⁴⁴<http://www.nature.com/news/2010/101013/full/467753a.html>

⁴⁵https://en.wikipedia.org/wiki/Law_of_the_instrument

um número excessivo de resultados que termina sendo aceito sem crítica; pode levar à super interpretação dos dados; e incentiva o mal hábito de jogar dados e cuspir resultados (data trawling). O R, por outro lado, não lhe diz nada a não ser aquilo que você peça explicitamente (Crawley, 2007).

(...) aprender a programar é importante porque desenvolve habilidades analíticas e de resolução de problemas. É uma atividade criativa, um meio de expressar idéias abstratas. Assim, programar é divertido e é muito mais do que uma habilidade profissional. Ao projetar programas, aprendemos muitas habilidades que são importantes para todas as profissões. Essas habilidades podem ser resumidas como: (1) Leitura crítica, (2) Pensamento analítico, (3) Síntese criativa (Why programming is important⁴⁶).

Uma das coisas mais importantes que você (como cientista) pode fazer é dedicar um tempo para aprender uma linguagem de programação de verdade. Aprender a programar é como aprender outro idioma: exige tempo e treinamento, e não há resultados práticos imediatos. Mas se você supera essa primeira subida íngreme da curva de aprendizado, os ganhos como cientista são enormes. Programar não vai apenas livrar você da camisa de força dos pacotes estatísticos, mas também irá aguçar suas habilidades analíticas e ampliar os horizontes de modelagem (...) e estatística (Gotelli and Ellison, 2013).

⁴⁶<http://programmingstage.blogspot.com.br/2012/05/why-programming-is-important.html>

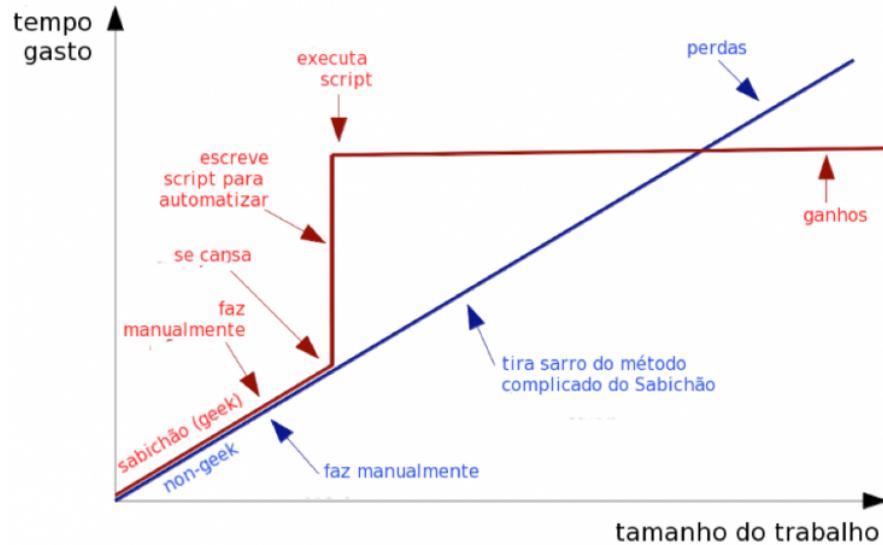


Figura 3: Curvas de aprendizagem do aluno Sabichão (Geek) e de um aluno que tem medo de código (non-geek).

Análise exploratória de dados (AED)

O que é AED e por que fazer AED?

Em estatística, análise exploratória de dados (AED) é uma abordagem para analisar conjuntos de dados visando resumir as suas principais características, particularmente com métodos gráficos e visuais. Um modelo estatístico pode ou não ser usado, mas AED objetiva principalmente ver o que os dados podem nos dizer para além da modelagem ou de testes de hipóteses. A AED foi promovida por John Tukey⁴⁷ para incentivar estatísticos a explorar dados e, eventualmente, formular novas hipóteses que possam levar à nova coleta de dados e a experimentos. AED é diferente de análise de dados inicial (AID), que foca mais estreitamente na verificação de suposições necessárias aos modelos de testes de

⁴⁷https://en.wikipedia.org/wiki/John_Tukey

hipóteses, manipulação de valores ausentes e transformações de variáveis, conforme necessário. AED abrange AID (Texto traduzido da WikiPedia⁴⁸).



Muitas vezes esquecemos como funciona a ciência e a engenharia. Idéias vêm mais frequentemente de exploração anterior do que de descargas atmosféricas. (...) **Encontrar a pergunta é muitas vezes mais importante do que encontrar a resposta.** Análise exploratória de dados é uma atitude, uma flexibilidade que depende de visualização gráfica, não é um conjunto de técnicas (Tukey, 1980).

Em sistemática e taxonomia, uma ciência histórica, a análise exploratória é fundamental, pois é através dela que podemos postular boas hipóteses e usar teste de hipóteses para confirmá-las.

Aplicar a análise exploratória de dados para criar hipóteses e então usar os mesmos dados para testar essas hipóteses deve ser evitado. Se alguém tem conhecimento *a priori* limitado, então uma abordagem válida é criar dois conjuntos de dados: aplicar exploração de dados sobre o primeiro conjunto para criar hipóteses e usar o segundo conjunto de dados para testar essas hipóteses . Tal processo, no entanto, só é prático para conjuntos de dados grandes.

⁴⁸https://en.wikipedia.org/wiki/Exploratory_data_analysis

Independentemente da situação específica, o uso constante e a estruturação de relatórios transparentes de exploração sistemática de dados melhorariam a qualidade da pesquisa ecológica e de quaisquer recomendações que ela produziria (Zuur et al., 2010).

A análise exploratória não é *dragagem* de dados! Assume-se que o pesquisador formulou hipóteses biológicas plausíveis *a priori* amparadas pela teoria.

Objetivos da AED

- Controlar a qualidade dos dados;
- Sugerir hipóteses para os padrões observados (novos estudos);
- Apoiar a escolha dos procedimentos estatísticos de testes de hipótese;
- Avaliar se os dados atendem às premissas dos procedimentos estatísticos escolhidos.

Para ler

Dois artigos que você deveria ler: Tukey (1980) e Zuur et al. (2010).

Exercícios no notaR

Ao fim de alguns capítulos, disponibilizaremos em seções chamadas **Exercícios** uma lista de endereços ligados ao notaR⁴⁹, um sistema criado para auxiliar no aprendizado da linguagem R desenvolvido pelo laboratório de Ecologia Teórica da Universidade de São Paulo (Chalom et al., 2012).

⁴⁹<http://www.lage.ib.usp.br/notar/index.php>

Conhecendo o notaR

Os exercícios do notaR obrigam o usuário a construir um script de R a partir de um enunciado, acabando por fugir do tradicional **copia-cola** de tutoriais de R. Muitos exercícios no sistema não obrigam o usuário a estar autenticado (estar logado usando login e senha) no sistema para poder fazê-los. Leia com atenção o enunciado de cada exercício antes de iniciar a construir o seu script.

Uma lista de todos os exercícios **notaR** incluídos neste livro está relacionada abaixo:

- Resolva o exercício 101.01 Bem vindo ao notaR⁵⁰.
- Resolva o exercício 101.05 Remoção Com Critério⁵¹.
- Resolva o exercício 102.04 Biomassa de Árvores⁵².
- Resolva o exercício 102.02 Sequências⁵³.
- Resolva o exercício 102.03 Conta de Luz⁵⁴.
- Resolva o exercício 102.01 Área Basal⁵⁵.
- Resolva o exercício 102.05 Variância na Unha⁵⁶.
- Resolva o exercício 101.03 Objetos de Data⁵⁷.
- Resolva o exercício 103.01 Distância entre cidades⁵⁸.
- Resolva o exercício 103.02 Criação de um data frame⁵⁹.
- Resolva o exercício 103.03 Criando uma Matriz⁶⁰.
- Resolva o exercício 302.02 Histogramas (frequência)⁶¹.
- Resolva o exercício 302.07 Gráficos com trechos selecionados do data.frame/matriz⁶².

⁵⁰<http://notar.ib.usp.br/exercicio/29>

⁵¹<http://notar.ib.usp.br/exercicio/15>

⁵²<http://notar.ib.usp.br/exercicio/19>

⁵³<http://notar.ib.usp.br/exercicio/12>

⁵⁴<http://notar.ib.usp.br/exercicio/4>

⁵⁵<http://notar.ib.usp.br/exercicio/17>

⁵⁶<http://notar.ib.usp.br/exercicio/5>

⁵⁷<http://notar.ib.usp.br/exercicio/18>

⁵⁸<http://notar.ib.usp.br/exercicio/23>

⁵⁹<http://notar.ib.usp.br/exercicio/2>

⁶⁰<http://notar.ib.usp.br/exercicio/13>

⁶¹<http://notar.ib.usp.br/exercicio/66>

⁶²<http://notar.ib.usp.br/exercicio/71>

- Resolva o exercício 302.04 Box-plots⁶³.
- Resolva o exercício 302.05 Gráficos de dispersão⁶⁴.
- Resolva o exercício 302.06 Matriz de dispersão⁶⁵.
- Resolva o exercício Editando parâmetros gráficos⁶⁶.
- Resolva o exercício 104.01 Sintetizando dados⁶⁷.
- Resolva o exercício 103.05 Classes de Objetos⁶⁸.
- Resolva o exercício 103.04 Lendo e salvando seus dados⁶⁹.
- Resolva o exercício 103.7 Cara ou coroa⁷⁰.
- Resolva o exercício 103.9 Lembrando matrizes e listas⁷¹.

Referências úteis

Listamos abaixo algumas referências importantes para complementar o seu estudo:

- Apostila USP-BIE-5782⁷².
- Introdução ao R no site do R⁷³.
- Introdução ao uso do programa R de Victor Landeiro e Fabricio Baccaro⁷⁴.
- Relatórios dinâmicos com R Markdown⁷⁵.

⁶³<http://notar.ib.usp.br/exercicio/67>

⁶⁴<http://notar.ib.usp.br/exercicio/69>

⁶⁵<http://notar.ib.usp.br/exercicio/70>

⁶⁶<http://notar.ib.usp.br/exercicio/75>

⁶⁷<http://notar.ib.usp.br/exercicio/14>

⁶⁸<http://notar.ib.usp.br/exercicio/21>

⁶⁹<http://notar.ib.usp.br/exercicio/22>

⁷⁰<http://notar.ib.usp.br/exercicio/34>

⁷¹<http://notar.ib.usp.br/exercicio/37>

⁷²http://ecologia.ib.usp.br/bie5782/doku.php?id=bie5782:02_tutoriais:tutorial4:start

⁷³<https://cran.r-project.org/doc/manuals/R-intro.html>

⁷⁴https://6e938257-ec28-4b42-9917-4ed33b2f1d0b.filesusr.com/ugd/e086b2_0d0de99bbb204ca68e079b4ae11b87d6.docx?dn=A_apostila_de_introdu%C3%A7%C3%A3o_ao_R_vers%C3%A3o_6.3.1_2018.docx

⁷⁵<http://www.botanicaamazonica.wiki.br/labotam/doku.php?id=bot89:precurso:rmarkdown:inicio>

Sobre os autores

Alberto é doutor em Ecologia, Evolução e Sistemática, com uma ênfase em Evolução pela Universidade do Missouri, Saint Louis (University of Missouri Saint Louis). É cientista do Instituto Nacional de Pesquisas da Amazônia (INPA) desde o ano de 2009. Ministra anualmente disciplinas ligadas ao Programa de pós-graduação em Ciências Biológicas do INPA, como **Preparação de dados para análises Estáticas - Introdução ao uso de linguagem R (BOT-89)**, **Delineamento Experimental e Amostral (BOT-90)**, e **Teoria e Prática de Sistemática Filogenética (BOT-99)**. É colíder do grupo de pesquisa Ecologia e Evolução de Plantas da Amazônia⁷⁶ do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

Ricardo⁷⁷ é doutor em Ciências Biológicas (Botânica) pelo INPA. Seus principais interesses são estar e se divertir com a família, trabalhar na floresta coletando e admirando a beleza da Natureza, trabalhar no herbário e laboratório, e também adora se divertir na frente do computador analisando dados e escrevendo. Ama correr e pedalar, e também se divertir e aprender com sua filha.

⁷⁶<http://dgp.cnpq.br/dgp/espelhogrupo/38425>

⁷⁷<https://www.ricardoperdiz.com>



Parte I

Parte I



Um passo a passo para conhecer a sintaxe básica da linguagem de programação e o potencial de usar o R como ferramenta científica.



1

Conceitos introdutórios

Recomendamos que o usuário siga a ordem dos capítulos, especialmente se é a primeira vez que o mesmo está lidando com o R. Use o R via RStudio, é mais simples. Busque executar cada script linha por linha, procurando entender o que cada linha faz e o que cada comando representa. E para cada função nova, busque a **ajuda do R**, leia o que significam cada um dos argumentos. Altere os argumentos das funções e veja o que acontece. Explore!

1.1 Console e scripts

As figuras abaixo mostram imagens relativamente similares ao que os usuários do R padrão (Figura 1.1) e RStudio (Figura 1.2) verão ao iniciarem esses programas.

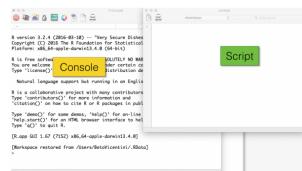


Figura 1.1: Imagem de uma sessão do R para Mac OS X com um janela de script aberta além do console.

Ao abrirem o R ou o RStudio você terão basicamente duas janelas principais:

- **Console** - corresponde à interface que interpreta o código da

linguagem. Os códigos digitados aqui serão interpretados pelo R (Texto destacado em **verde** nas figuras 1.1, 1.2);

- **Script** - digitar um código curto no console é simples, mas quando o código é longo, é importante guardá-lo em algum lugar para executá-lo em qualquer momento. Para isso utilizamos scripts, que são arquivos de textos simples que podem ser salvo (extensão .R) numa pasta no seu computador e reutilizado (Texto destacado em **amarelo** nas figuras 1.1, 1.2). Como um arquivo .R é um texto simples, ele é editável por qualquer editor de texto, como o Bloco de Notas ou Notepad++¹ para usuários Windows, TextWrangler² ou BBEdit³ para usuários macOS, ou gedit⁴ para usuários Linux. O editor de scripts do RStudio é excelente.

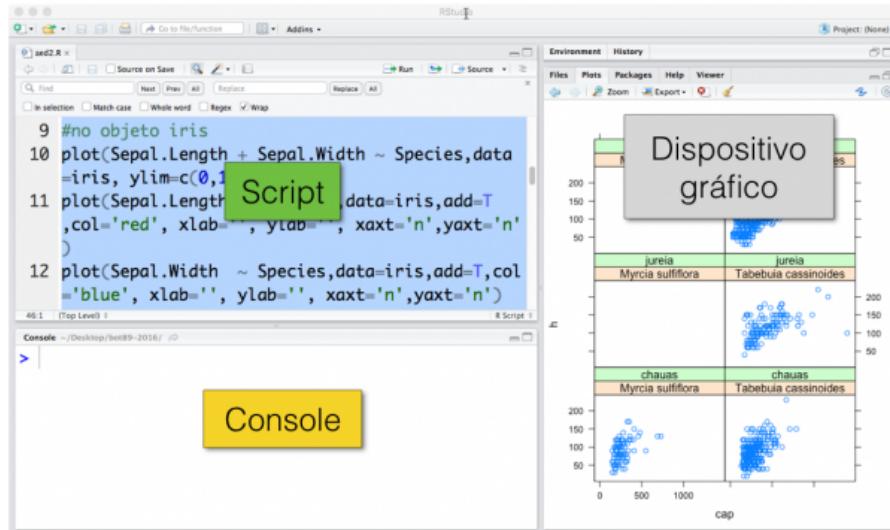


Figura 1.2: Imagem de uma sessão do RStudio com três painéis abertos.

¹<https://notepad-plus-plus.org/>

²<https://www.barebones.com/products/textwrangler/>

³<https://www.barebones.com/products/textwrangler/>

⁴https://help.gnome.org/users/gedit/stable/index.html.pt_BR

1.2 Linguagem objeto-orientada

A chave para entender o R é que trata-se de uma linguagem. Uma linguagem para manipular objetos (Venables and Ripley, 2002). Um objeto é identificado por uma palavra e veremos ao longo do curso que existem vários tipos de objetos. Este é o conceito mais importante para entender o R.

Para começar, digite algumas coisas no Console do seu R ou RStudio. Por exemplo, digite uma fórmula matemática simples no console e o execute:

```
3 + 5 + 10
```

```
## [1] 18
```

O console retornou 18, isto é, o R funcionou como uma calculadora. Portanto, números são interpretados como números pelo console.

Agora vamos criar um objeto simples chamado `objnum` e atribuir texto a ele:

```
objum <- "vou colocar um texto dentro do meu primeiro objeto" # enter  
objum # enter
```

```
## [1] "vou colocar um texto dentro do meu primeiro objeto"
```

Vejam que `objum` é o nome do meu primeiro objeto, e tem como conteúdo o texto que nós atribuímos a ele. Nós poderíamos ter chamado `objum` de qualquer coisa (sem espaços em branco):

```
banana <- "vou colocar um texto dentro do meu primeiro objeto"
banana
```

```
## [1] "vou colocar um texto dentro do meu primeiro objeto"
```

```
# mesma coisa né?
```

Um objeto também pode virar outro objeto. Vejamos abaixo:

```
objdois <- objum
objdois # pegou o conteúdo que eu coloquei em objum
```

```
## [1] "vou colocar um texto dentro do meu primeiro objeto"
```

Podemos guardar resultados de contas em um objeto:

```
objtres <- 3 + 5 + 10
objtres
```

```
## [1] 18
```

Note que o código exposto acima é geralmente o que está contido em um script⁵, ou seja, a sequência de códigos que você escreveu e que o R interpretará para realizar o que foi dito. Se você copiar e colar o trecho (ou os trechos) de código acima em um arquivo .R, você poderá executá-los tantas vezes quanto quiser. Copie então todos os trechos de código expostos até então para a janela do script e salve-o como script01.R em alguma pasta no seu computador.

⁵https://pt.wikipedia.org/wiki/Linguagem_de_script

Note também que dependendo do está incluso no script, o código apresenta cores diferentes para comentários, objetos, e símbolos de atribuição. Isso ajuda a entender os elementos da sintaxe (vocabulário e gramática do R). IDE's também mostram isso de forma colorida.

Note também que, à direita do símbolo #, o texto no código acima fica cinza. Este símbolo indica ao R que todo texto começando no # deve ser interpretado como texto apenas, e não como código. Comentar códigos é uma prática que deve ser tornar rotineira a fim de facilitar o entendimento de cada script.

Se você já está escrevendo um script num arquivo e não no console, selecione todo o texto na sua janela de script e digite `Control+R` (Windows), ou `Command+Enter` (Mac). O conteúdo do script será executado no Console. Note que o símbolo # não é interpretado pelo console.

O R vem com vários scripts prontos, que são funções que executam alguma coisa. Esses scripts são objetos de classe **função**, pois eles contêm um conjunto de códigos que utilizam parâmetros para executar um conjunto de passos. Parâmetros são objetos que a função utiliza segundo as suas especificações. Por exemplo, vamos listar os objetos criados no passo anterior:

```
# digite
ls() # note os parênteses (); a função ls() lista os objetos criados por você: [1] "banana"  " "

# adicione um parâmetro à função, especificando o que você quer listar, apenas aqueles que com
ls(pattern = "obj") # [1] "objdois" "objtres" "objum"

# agora veja o que a função ls é
ls # note que não digitiei os parênteses e ao digitar isso verei o script que está dentro da fu
# note que os argumentos da função são os objetos que vão dentro do () da função e são utilizad
ls(sorted = FALSE) # ele mostra os objetos na ordem que foram criados
```

1.3 R-base e pacotes

Como vimos acima, um script do tipo função é um objeto que executa um conjunto de comandos e recebe argumentos que modificam o que o script realiza. Existem dois tipos de funções:

- Funções do pacote `base` - funções que vêm junto com o R quando você faz o download do programa, como por exemplo as funções `length()`, `order()`, `sort()`, `sum()`, `mean()` etc;
- Funções de pacotes externos - pacotes (packages em inglês) são funções criadas por colaboradores e organizadas em pacotes que você pode baixar dos repositórios do R.

As funções do pacote `base` vêm junto com o programa e você não precisa se preocupar com isso. Pacotes, por outro lado, devem ser instalados pelo usuário conforme a necessidade. Por exemplo, para trabalhar com dados filogenéticos, você pode necessitar do pacote `ape`⁶ (Paradis et al., 2020), que já tem várias funções preparadas para análises filogenéticas. Se você necessitar deste pacote, por exemplo, basta executar o seguinte comando no console do R:

```
install.packages("ape")
```

Para trabalhar com pacotes você primeiro precisa definir um repositório, ou seja, um servidor, de vários disponíveis (espelhos do repositório oficial), de onde o R buscará o pacote desejado. Pode-se também o menu do R ou do RStudio para instalar pacotes e definir um repositório padrão para sua instalação.

Como tudo no R, você pode executar comandos que estão no menu usando funções. Por exemplo, ao invés de usarmos o menu do R para definir um repositório e instalar um pacote, vamos executar estas ações por meio de um script:

⁶<https://cran.r-project.org/web/packages/ape/index.html>

```
# isso pode não funcionar se estiver no INPA por causa do Proxy.  
chooseCRANmirror() # seleciona repositório  
install.packages("ape", dependencies = TRUE) # instala o pacote Ape
```

1.4 Ajuda no R

Toda função no R, oriunda tanto do pacote base ou dos pacotes extras, possui um pedaço de texto chamado de “ajuda” e que possui uma instrução sobre a utilização da mesma. Para obter esta instrução, é muito fácil. Basta executar uma das seguintes opções:

```
# como exemplo vamos usar a função ls()  
?ls # ou seja o comando é ?+"nome da função"  
# ou então  
help(topic = "ls") # nome da função como argumento da topic da função help
```

Ao digitar uma das opções acima, o R vai abrir uma janela (no RStudio, vai colocar esse resultado num dos painéis) contendo a página com a explicação da função:

```
?ls
```

```
## List Objects  
##  
## Description:  
##  
##   'ls' and 'objects' return a vector of character strings giving the  
##   names of the objects in the specified environment. When invoked  
##   with no argument at the top level prompt, 'ls' shows what data  
##   sets and functions a user has defined. When invoked with no
```

```
##      argument inside a function, 'ls' returns the names of the
##      function's local variables: this is useful in conjunction with
##      'browser'.
##
## Usage:
##
##      ls(name, pos = -1L, envir = as.environment(pos),
##          all.names = FALSE, pattern, sorted = TRUE)
##      objects(name, pos= -1L, envir = as.environment(pos),
##              all.names = FALSE, pattern, sorted = TRUE)
##
## Arguments:
##
##      name: which environment to use in listing the available objects.
##             Defaults to the _current_ environment. Although called
##             'name' for back compatibility, in fact this argument can
##             specify the environment in any form; see the 'Details'
##             section.
##
##      pos: an alternative argument to 'name' for specifying the
##            environment as a position in the search list. Mostly there
##            for back compatibility.
##
##      envir: an alternative argument to 'name' for specifying the
##             environment. Mostly there for back compatibility.
##
##      all.names: a logical value. If 'TRUE', all object names are returned.
##                 If 'FALSE', names which begin with a '.' are omitted.
##
##      pattern: an optional regular expression. Only names matching
##                'pattern' are returned. 'glob2rx' can be used to convert
##                wildcard patterns to regular expressions.
##
##      sorted: logical indicating if the resulting 'character' should be
##              sorted alphabetically. Note that this is part of 'ls()' may
##              take most of the time.
```

```
## Details:  
##  
##      The 'name' argument can specify the environment from which object  
##      names are taken in one of several forms: as an integer (the  
##      position in the 'search' list); as the character string name of an  
##      element in the search list; or as an explicit 'environment'  
##      (including using 'sys.frame' to access the currently active  
##      function calls). By default, the environment of the call to 'ls'  
##      or 'objects' is used. The 'pos' and 'envir' arguments are an  
##      alternative way to specify an environment, but are primarily there  
##      for back compatibility.  
##  
##      Note that the _order_ of strings for 'sorted = TRUE' is locale  
##      dependent, see 'Sys.getlocale'. If 'sorted = FALSE' the order is  
##      arbitrary, depending if the environment is hashed, the order of  
##      insertion of objects, ....  
##  
## References:  
##  
##      Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) _The New S  
##      Language_. Wadsworth & Brooks/Cole.  
##  
## See Also:  
##  
##      'glob2rx' for converting wildcard patterns to regular expressions.  
##  
##      'ls.str' for a long listing based on 'str'. 'apropos' (or 'find')  
##      for finding objects in the whole search path; 'grep' for more  
##      details on 'regular expressions'; 'class', 'methods', etc., for  
##      object-oriented programming.  
##  
## Examples:  
##  
##      .0b <- 1  
##      ls(pattern = "0")  
##      ls(pattern= "0", all.names = TRUE)      # also shows ".[foo]"  
##
```

```
##      # shows an empty list because inside myfunc no variables are defined
##      myfunc <- function() {ls()}
##      myfunc()

##      # define a local variable inside myfunc
##      myfunc <- function() {y <- 1; ls()}
##      myfunc()                      # shows "y"
```

Todos esses trechos de ajuda (?) do R têm a mesma estrutura e devem ser acessados constantemente para o bom entendimento do funcionamento das funções e seus argumentos.

1.5 Área de trabalho vs. Pasta de Trabalho

Dois conceitos são fundamentais para trabalhar no R:

- Área de Trabalho (environment no R) - é o local dentro do R onde você coloca os objetos criados durante a execução de scripts. Você pode visualizar/apagar os objetos no Console:

```
ls() # ls= listar - já vimos que isso lista os objetos criados por você e que estão na área de trabalho

?rm # veja o help desta função
rm(objum) # rm = remover - esta função apaga objetos da área de trabalho. No exemplo estamos criando um objeto
ls() # note agora que o objeto não existe mais

rm(list = ls()) # desse jeito apagamos todos os objetos que criamos anteriormente
ls() # note que não sobrou nenhum objeto
```

- Pasta de Trabalho (working directory- é o local (pasta) no seu computador que o R usa para salvar arquivos. É o caminho padrão

para o R encontrar arquivos de dados, scripts, etc. Você pode trabalhar no R sem definir uma **Pasta de Trabalho** mas é muito mais simples se você usa este recurso. Experimente a seguinte script:

```
getwd() # o nome desta função é abreviação de "get working directory" ou seja "pega a pasta de trabalho"
```

Toda vez que você usar o R, é uma boa prática primeiramente definir a *pasta de trabalho*, que é o local no seu computador onde você guarda os arquivos relacionados ao seu projeto (dados, scripts, resultados etc.).

Você pode definir a *pasta de trabalho* usando o menu do R (Arquivo ⇒ Diretório de Trabalho no Windows; ou Misc ⇒ Muda Pasta de Trabalho no Mac). Ou você pode usar uma função:

```
?setwd # veja o help da função que iremos utilizar  
minhapasta <- "/Users/BetoVicentini/Desktop/bot89-2016"  
# se estiver usando windows:  
# minhapasta = "c:/Users/BetoVicentini/Documents/bot89-2016"  
# minhapasta = "c:\\\\Users\\\\BetoVicentini\\\\Documents\\\\bot89-2016" #talvez precise usar backslash  
  
setwd(dir = minhapasta) # usa a função "set working directory" para especificar o diretório (pode ser necessário usar o comando anterior)  
# note que eu defini dir primeiramente como o objeto "minhapasta" usei esse objeto para especificar o diretório
```

Objetos criados no R por você podem ser salvos como um arquivo no seu computador. Você pode usar o menu do R ou RStudio para isso ou pode usar o comando abaixo:

```
?save # veja o help desta função que vamos utilizar  
?save.image # veja o help desta função que vamos utilizar  
  
# se sua área de trabalho estiver vazia  
ls() # se não retorna nada porque você apagou acima
```

```
# crie alguns objetos para o exercício
objum <- "um objeto de texto qualquer"
objum
objdois <- 18 # um objeto com um número qualquer
objdois
banana <- objum # um objeto idêntico a objum

# pronto, agora mostre todos os objetos na sua ÁREA DE TRABALHO
ls()

# agora salve esse objetos como um arquivo na sua PASTA DE TRABALHO
getwd() # veja onde o arquivo será salvo

# se voce quer salvar todos os objetos
save.image(file = "meusObjetos.Rdata")

# se voce quer salvar apenas alguns objetos
save(objum, banana, file = "meusObjetos2.Rdata")

# veja que na sua pasta de trabalho getwd() voce tem esses arquivos
dir() # lista arquivos na sua pasta de trabalho (análoga a função ls() que lista objetos)
dir(pattern = "Rdata") # lista apenas arquivos que tem no nome ".Rdata"

# apaga os objetos todos
rm(list = ls())
ls() # note que não tem mais objetos na sua área de trabalho, mas pode resgatar objetos que saíram
load(file = "meusObjetos2.Rdata") # puxa os objetos neste arquivo que está na sua pasta de trabalho
ls() # note que ele puxou os objetos objum e banana que salvamos acima neste arquivo

# puxe o outro arquivo gerado
load(file = "meusObjetos.Rdata") # puxa os objetos neste arquivo que está na sua pasta de trabalho
ls() # veja novamente os objetos na sua área de trabalho: Todos os objetos são listados. Se fizer
```

Portanto, *Área de Trabalho* e *Pasta de Trabalho* são dois conceitos fundamentais que você precisa entender bem, pois define como e onde você estará salvando informação quando usar o R, tanto em termos

de objetos (área de trabalho) como em termos de arquivos (pasta de trabalho).

Se você digita `q()`, que é a função para sair do R, ele geralmente lhe pergunta se você quer salvar os objetos numa área de trabalho padrão (que o R puxa quando você o inicia):

```
q() # sai do R  
# vai perguntar no Console algo do tipo "Save workspace image to ~/Desktop/bot89-2016/.RData"?  
# note que o arquivo não tem nome apenas extensão ".RData" e é salvo na pasta de trabalho definida
```

Códigos de scripts podem ser executados no console sem necessidade de abrir o script. Por exemplo, suponha que eu tenha salvo o seguinte script num arquivo `script01.R`:

```
objum <- "meu primeiro objeto de texto"  
objdois <- 18 # meu primeiro objeto numérico  
banana <- objum
```

Se este arquivo `Script01.R` está na minha pasta de trabalho, então posso executá-lo com a função `source()`:

```
dir(pattern = ".R") # mostra os arquivos .R que estão na pasta de trabalho  
rm(list = ls()) # apaga todos os objetos, para ver que ele irá criar os objetos indicados no script  
ls() # não tem nenhum objeto  
source("script01.R")  
ls() # os objetos foram criados como especificado no script
```

1.6 Dicas de organização do trabalho

Você irá manipular diversos arquivos durante a execução de qualquer projeto e precisa organizar os arquivos para não se perder e poder voltar a eles quando necessário.

A grande vantagem de realizar a análise de dados, produção de figuras etc, no R é criar um processo replicável, e é preciso ser bem organizado com os arquivos para poder reutilizar o processo com novos dados ou adaptá-lo para novas análises. Isso implica, entre outras coisas, em:

- Organizar o projeto em uma pasta no seu computador exclusiva para ele;
- Definir um critério lógico para dar nome aos arquivos do projeto;
- Organizar arquivos em subpastas segundo a necessidade, por exemplo, uma subpasta para dados, outra para figuras, outras para tabelas geradas e outros resultados. Pode inclusive colocar a criação dessas pastas nos scripts do R, que ele cria na pasta de trabalho ('getwd()') que deve ser a pasta do seu projeto através de (funções do R para manipular arquivos e pastas);
- O código dos seus scripts devem ser escritos de forma genérica para que ele possa ser usado com qualquer outro conjunto de dados que tenha a mesma estrutura que os seus (variáveis). Ou seja, o código deve ser escrito não em função do conteúdo dos seus dados mas apenas na estrutura (i.e. as colunas dos seus dados) deles. Esta é a chave para uma programação eficiente e para a replicabilidade. Isso significa que um resultado obtido a partir dos seus dados não deve ser manualmente atribuído a nenhum objeto no script, apenas atribuído a partir do objeto contendo o resultado.

A definição da pasta de trabalho (**working directory**) é fundamental simplificar a leitura de dados e salvar arquivos sem se preocupar com o caminho completo (**full path**) para cada arquivo, mas apenas o caminho relativo (**relative path**). Por exemplo, um arquivo numa pasta de projeto pode ter como caminho completo /home/usuario/Desktop/meuprojeto/dados/morfometricos.xlsx, e o caminho relativo à pasta de trabalho seria apenas dados/morfometricos.xlsx. Da mesma forma para salvar resultados a partir do R.

O RStudio⁷ (RStudio Team, 2020) facilita esse processo para você,

⁷<https://rstudio.com/>

permitindo que você crie um projeto. Um projeto basicamente consiste em definir uma pasta de trabalho para que, toda vez que você abrindo-o, a pasta que contém o arquivo de término `.Rproj` seja a sua pasta de trabalho. Alternativamente, você pode simplesmente indicar a pasta de trabalho toda vez que iniciar uma atividade utilizando as funções `setwd()` no console ou em um script.

1.7 Dicas de erros comuns de sintaxe da linguagem R

Abrir e esquecer de fechar parênteses, colchetes e aspas é um dos erros mais comuns no início da caminhada rumo ao aprendizado de qualquer linguagem de programação, e é necessário muita paciência⁸ por parte do usuário para aprender a lidar com a frustração dos sucessivos erros que aparecem no caminho. Mesmo usuários mais experientes têm que aprender a lidar com os erros, que são comuns, o que levou o programador Patrick Burns a publicar um curioso livro sob o título *The R Inferno*⁹ (Burns, 2012). Uma ferramenta útil para evitar tais problemas é o uso de uma IDE como o RStudio. Há opções na configuração que autorizam o software a checar seu código e informá-lo de possíveis erros, além de proverem ao usuário uma ferramenta de limpeza do código. Sugerimos fortemente a instalação de uma IDE para usar o R. Segue então uma brevíssima lista de erros super comuns que iniciantes enfrentam no aprendizado da linguagem R.

1.7.1 Parênteses

Lembre-se sempre de, ao abrir um parêntese, fechá-lo. Caso contrário, a ação não se completará e seu console ficará travado com um sinal de + aguardando que seu código seja completo. Neste caso,

⁸Lembrem-se sempre de que vocês não são os únicos a ter de lidar com a frustração durante o aprendizado de uma linguagem de programação. É importante estarem conscientes dessa verdade. Vejam esta postagem: <https://www.codingame.com/blog/dealing-with-programming-frustration-the-right-way/>

⁹https://www.burns-stat.com/pages/Tutor/R_inferno.pdf

ficar completo significa fechar o parêntese. Tente executar o código abaixo e verifique que o sinal de + ficará estagnado na tela de seu console. Para sair desta tela, clique no console e depois tecle Esc. Você verá que o sinal de > voltará a aparecer no console.

```
erro <- c(1, 2, 3)
```

Agora, execute o comando abaixo. Ele será executado perfeitamente.

```
erro <- c(1, 2, 3)
```

1.7.2 Vírgulas

Ao concatenar elementos ou ao adicionar valores em argumentos de funções, é necessário lembrar sempre de colocar as vírgulas em seus devidos lugares. Caso contrário ... mais um erro! Tentem executar o código abaixo:

```
numeros <- c(1, 5, 6 7, 8)
```

```
## Error: <text>:1:22: unexpected numeric constant
## 1: numeros <- c(1, 5, 6 7
##                                     ^
##                                     ^
```

O R dará o aviso `Error: unexpected numeric constant in "numeros <- c(1,5,6 7"` e encerrá a operação. Executem agora o comando abaixo. Será executado sem problemas.

```
numeros <- c(1, 5, 6, 7, 8)
```

1.7.3 Aspas

Textos são especificados dentro de aspas duplas "" ou aspas simples ''; tanto faz qual você usa, o importante é sempre que abrir aspas, fechar as aspas com o mesmo tipo. Se seu texto tem acentos, use aspas duplas para delimitá-lo. Aspas simples podem entrar num texto definido por aspas duplas, e vice-versa. Rode isso no seu console:

```
objum <- "um texto com 'aspas simples'"  
objum
```

```
## [1] "um texto com 'aspas simples'"
```

```
obj2 <- 'um texto com "aspas simples"'  
obj2
```

```
## [1] "um texto com \"aspas simples\""
```

```
# note que foi adicionada uma barra invertida, porque \" na sintaxe do Rsignifica aspas para o
```

1.7.4 Números

Números são sempre especificados sem aspas; se você colocar qualquer número entre aspas, ele será interpretado como texto:

```
obj1 <- 18  
obj1 + 1 # essa fórmula irá funcionar porque obj1 é um número
```

```
## [1] 19
```

```
obj2 <- "18"
```

Isso não vai funcionar porque `obj2` não é um número.

```
obj2 + 1
```

```
## Error in obj2 + 1: non-numeric argument to binary operator
```

Caso você insista em rodar, receberá a seguinte mensagem: `Error in obj2 + 1 : non-numeric argument to binary operator.`

1.7.5 Nomes de objetos

Nomes de objetos não podem ter espaço em branco e aspas são ignoradas:

```
obj 1 = "meutexto" #nao vai funcionar
```

```
## Error: <text>:1:5: unexpected numeric constant
## 1: obj 1
##           ^
```

```
obj1 <- "meutexto" # vai funcionar
obj1
```

```
## [1] "meutexto"
```

```
"obj1" <- "meu texto" # vai criar objeto obj1, ignorando as aspas  
obj1
```

```
## [1] "meu texto"
```

```
obj"1" = "meu texto" #nao vai funcionar
```

```
## Error: <text>:1:4: unexpected string constant  
## 1: obj"1"  
##           ^
```

1.7.6 Atribuição de objetos

A **atribuição de valores aos objetos** pode ser feita com dois operadores equivalentes, `=` ou `<-` (este pode ser utilizado no sentido inverso também `->`):

```
obj1 <- "meu texto"  
  
# ou pode escrever assim  
obj1 <- "meu texto" # atribui  
  
# ou assim  
"meu texto" -> obj1 # atribui  
obj1
```

```
## [1] "meu texto"
```

1.7.7 Busque entender as partes

Se você tiver dificuldade no entendimento de um script do R que está tentando rodar, separe os termos das linhas e expressões para entender o que cada parte está fazendo. Também pode alterar os valores dos argumentos para entender o funcionamento de uma função, ou simplesmente digitar a função sem os parênteses para ver o script que ela contém. Por exemplo, vejamos a expressão abaixo, que faz uso de um conjunto de dados chamado `iris`, que vem disponível com o R.

```
obj1 <- paste(levels(iris$Species), tapply(iris$Sepal.Length, INDEX = iris$Species, mean, na.rm = TRUE))
obj1
```

```
## [1] "setosa sépala média = 5.006"    "versicolor sépala média = 5.936"
## [3] "virginica sépala média = 6.588"
```

À primeira vista, a expressão parece complicada. Vamos separar as partes e entender pedaço por pedaço. Primeiro, vamos tentar entender quem é `iris`. Vamos checar a estrutura de `iris`:

```
str(iris) #veja a estrutura
```

```
## '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 ...
```

Agora, vamos obter um sumário estatístico de `iris`:

```
summary(iris) #veja o que é iris
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
	Min. :4.300	Min. :2.000	Min. :1.000	Min. :0.100	setosa :50
	1st Qu.:5.100	1st Qu.:2.800	1st Qu.:1.600	1st Qu.:0.300	versicolor:50
	Median :5.800	Median :3.000	Median :4.350	Median :1.300	virginica :50
	Mean :5.843	Mean :3.057	Mean :3.758	Mean :1.199	NA
	3rd Qu.:6.400	3rd Qu.:3.300	3rd Qu.:5.100	3rd Qu.:1.800	NA
	Max. :7.900	Max. :4.400	Max. :6.900	Max. :2.500	NA

Note que `iris$Species` contém 50 valores para três nomes e portanto eles estão sendo interpretados como categoria, isto é, é um fator. Vamos ver os níveis desse fator com a expressão abaixo, contida na função executada anteriormente e que desejamos destinar:

```
levels(iris$Species) #categorias especies do objeto factor iris$Species
## [1] "setosa"      "versicolor"   "virginica"
```

Partamos para o segundo elemento dessa função, que é a expressão colocada abaixo:

```
#o segundo elemento é o resultado de:
tapply(iris$Sepal.Length, INDEX = iris$Species, mean, na.rm = TRUE)
```

```
##      setosa versicolor virginica
##      5.006     5.936     6.588
```

Essa expressão basicamente calcula a média dos 50 comprimentos de sépala para cada espécie. Por fim, temos o último elemento da expressão que é o argumento `sep` pertencente à função `paste()`. Este argumento é responsável por informar

à função `paste()` qual separador nós vamos utilizar para separar os elementos contidos antes do argumento `sep`, isto é, separar o resultado de `levels(iris$Species)` da expressão `tapply(iris$Sepal.Length, INDEX=iris$Species, mean, na.rm=TRUE)`. Nesse caso, vamos separar esses dois elementos com o texto `sépala média =`.

```
paste(, sep ='sépala média = ')
```

Vamos agora ver o resultado da função que executamos lá em cima.

```
#veja o resultado
obj1
```

```
## [1] "setosa sépala média = 5.006"    "versicolor sépala média = 5.936"
## [3] "virginica sépala média = 6.588"
```

Lembre-se sempre de checar também o código de qualquer função, para poder entender como ela funciona. Vejamos os casos das funções `paste()` e `tapply()`, utilizadas neste exercício. Para checar o código de uma função, geralmente basta executar o nome da função sem parênteses. Vejamos:

```
tapply
```

```
## function (X, INDEX, FUN = NULL, ..., default = NA, simplify = TRUE)
## {
##     FUN <- if (!is.null(FUN))
##             match.fun(FUN)
##     if (inherits(INDEX, "formula")) {
##         if (is.data.frame(X))
##             INDEX <- .formula2varlist(INDEX, X)
```

```
##      else stop("'X' must be a data frame when 'INDEX' is a formula")
##    }
##    if (!is.list(INDEX))
##      INDEX <- list(INDEX)
##    INDEX <- lapply(INDEX, as.factor)
##    nI <- length(INDEX)
##    if (!nI)
##      stop("'INDEX' is of length zero")
##    if (!is.object(X) && !all(lengths(INDEX) == length(X)))
##      stop("arguments must have same length")
##    namelist <- lapply(INDEX, levels)
##    extent <- lengths(namelist, use.names = FALSE)
##    cumextent <- cumprod(extent)
##    if (cumextent[nI] > .Machine$integer.max)
##      stop("total number of levels >= 2^31")
##    storage.mode(cumextent) <- "integer"
##    ngroup <- cumextent[nI]
##    group <- as.integer(INDEX[[1L]])
##    if (nI > 1L)
##      for (i in 2L:nI) group <- group + cumextent[i - 1L] *
##        (as.integer(INDEX[[i]]) - 1L)
##    if (is.null(FUN))
##      return(group)
##    levels(group) <- as.character(seq_len(ngroup))
##    class(group) <- "factor"
##    ans <- split(X, group)
##    names(ans) <- NULL
##    index <- as.logical(lengths(ans))
##    ans <- lapply(X = ans[index], FUN = FUN, ...)
##    ansmat <- array(if (simplify && all(lengths(ans) == 1L)) {
##      ans <- unlist(ans, recursive = FALSE, use.names = FALSE)
##      if (!is.null(ans) && is.na(default) && is.atomic(ans))
##        vector(typeof(ans))
##      else default
##    }
##    else vector("list", prod(extent)), dim = extent, dimnames = namelist)
##    if (length(ans)) {
```

```
##           ansmat[index] <- ans
##       }
##   ansmat
## }
## <bytecode: 0x555cfa47ca70>
## <environment: namespace:base>
```

paste

```
## function (... , sep = " ", collapse = NULL, recycle0 = FALSE)
## .Internal(paste(list(...), sep, collapse, recycle0))
## <bytecode: 0x555cef20bff0>
## <environment: namespace:base>
```

1.8 O R como calculadora

O R é uma calculadora potente. Os exemplos abaixo mostram a informação básica necessária ao uso da ferramenta.

1.8.1 Operadores

- Operadores de atribuição:
 - use = ou <- ou -> para atribuir **valor** a **objeto** ou objeto a objeto
- Operadores matemáticos:
 - + e - são respectivamente adição e subtração;
 - * e / são respectivamente multiplicação e divisão;
 - ^ equivale à exponenciação.

Vamos testar os operadores abaixo. Execute cada operação, linha a linha:

```
3 + 3
```

```
## [1] 6
```

```
3 - 3
```

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

```
3 * 3
```

```
## [1] 9
```

```
3 / 3
```

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

```
3^3
```

```
## [1] 27
```

Objetos numéricos podem ser usados nas fórmulas aritméticas:

```
# atribua um valor a um ou mais objetos  
obj <- 3  
obj2 <- 3
```

```
# utilize objetos para fazer contas  
obj + obj2
```

```
## [1] 6
```

```
obj - obj2
```

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

```
obj * obj2
```

```
## [1] 9
```

```
obj / obj2
```

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

```
obj^obj2
```

```
## [1] 27
```

1.8.2 Precedência de parênteses

O uso de parênteses permite construir qualquer lógica de precedência no cálculo. Vamos atribuir valores a três objetos e fazer operações matemáticas com eles para exemplificar:

```
o1 <- 2  
o2 <- 4  
o3 <- 3
```

Os resultados não serão os mesmos para:

```
o1 * o2^o3 - 1 # =127
```

```
## [1] 127
```

```
o1 * o2^(o3 - 1) # =32
```

```
## [1] 32
```

```
(o1 * o2)^o3 - 1 # =511
```

```
## [1] 511
```

```
(o1 * o2)^(o3 - 1) # =64
```

```
## [1] 64
```

1.8.3 Funções e constantes matemáticas

Algumas operações aritméticas podem ser realizadas com algumas funções genéricas que apresentamos no exemplo de código abaixo. Veja o `?` para qualquer uma das funções abaixo e siga os links relacionados para ver todas as possibilidades dessas funções genéricas de uso genérico.

```
?srqt  
?abs  
?log  
?log10  
?sin  
?cos  
?asin  
?ceiling  
?floor  
?round
```

Veremos adiante que essas funções e operações matemáticas são aplicáveis à vetores.

```
# Raiz quadrada  
sqrt(9)
```

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

```
# valor absoluto  
abs(-12)
```

```
## [1] 12
```

```
abs(-66)
```

```
## [1] 66
```

```
abs(66)
```

```
## [1] 66
```

```
# logaritmo  
log(10) # Logaritmo natural
```

```
## [1] 2.302585
```

```
log(10, base = 10) # Logbase 10
```

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

```
log10(10) # Também log de base 10
```

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

```
# funções trigonométricas  
pi # é uma constante do R, um objeto chamado "pi" que tem o valor de 3.141593
```

```
## [1] 3.141593
```

```
sin(0.5 * pi) # Seno
```

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

```
cos(2 * pi) # Coseno
```

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

```
asin(1) # Arco seno (radianos)
```

```
## [1] 1.570796
```

```
asin(1) / pi * 180
```

```
## [1] 90
```

```
# arredondamentos  
# dois valores  
a <- 3.51  
b <- 3.49
```

```
# para o valor mais alto (ceiling = teto)  
ceiling(a)
```

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

```
ceiling(b)
```

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

```
# para o valor mais baixo (floor = chão)  
floor(a)
```

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

```
floor(b)
```

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

```
# arredonda dependendo se casa decimal é maior ou menor que 0.5  
round(a)
```

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

```
round(b)
```

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

1.8.4 NA e valores afins

Frequentemente no R, quando você realiza uma operação errada ou inválida, o R retorna um dos seguintes códigos (constantes lógicas), que **são entendidos pelo R como esses valores** e portanto podem ser usados na atribuição à objetos:

- `NA` = valores ausentes (faltando);
- `NAN` ou `not a number` = valores inválidos;
- `Inf` = infinito;
- `-Inf` = infinito negativo.

```
?NA # veja o help sobre isso
```

```
# um valor infinito negativo  
-5 / 0
```

```
## [1] -Inf
```

```
# um valor infinito positivo  
10 / 0
```

```
## [1] Inf
```

Note que o símbolo `Inf` serve como valor:

```
50000000000000000000 / Inf
```

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

Ele pode ser atribuído a um objeto:

```
# e que posso atribui-lo a um objeto
mf <- Inf
500 / mf
```

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

```
# valor inválido/inexistente
sqrt(-1)
```

```
## Warning in sqrt(-1): NaNs produced
```

```
## [1] NaN
```

```
# valores ausentes entram na matemática
2 * NA
```

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

```
2 * NaN
```

```
## [1] NaN
```

1.9 Exercícios

Como prática, vamos iniciar com dois exercícios do **notaR** para que você se familiarize tanto com a linguagem R quanto com o sistema **notaR**:

- Resolva o exercício 101.01 Bem vindo ao notaR¹⁰.
- Resolva o exercício 101.05 Remoção Com Critério¹¹.

¹⁰<http://notar.ib.usp.br/exercicio/29>

¹¹<http://notar.ib.usp.br/exercicio/15>

2

Objeto I - Vetores

2.1 Vetores e operações vetoriais I

Vetor no R é um tipo de objeto que concatena múltiplos valores de uma mesma classe. É fundamental que você entenda vetores para poder entender objetos mais complexos.

2.1.1 Criação de Vetores

A função `c()` é usada na criação de vetores, pois combina ou concatena elementos. Podemos concatenar números:

```
# um vetor de números
v1 <- c(3, 3.14, pi, 37.5, 38)
v1
```

```
## [1] 3.000000 3.140000 3.141593 37.500000 38.000000
```

Podemos concatenar textos:

```
v2 <- c("a", "banana", "maça", "pera", "jabuticaba")
v2
```

```
## [1] "a"          "banana"     "maça"       "pera"       "jabuticaba"
```

Podemos concatenar valores lógicos (veremos adiante como isso é importante):

```
v3 <- c(TRUE, TRUE, FALSE, FALSE)
v3
```

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

Podemos abreviar os valores lógicos TRUE como T e FALSE como F:

```
# c(TRUE, TRUE, FALSE, FALSE) e o mesmo que
v4 <- c(T, T, F, F)
v4
```

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

Vejamos se v3 é semelhante a v4:

```
v3 == v4
```

```
## [1] TRUE TRUE TRUE TRUE
```

Note que TRUE e FALSE são valores lógicos e essas palavras são entendidas apenas como tal quando em maiúsculas e sem aspas " pelo R. Tente executar o comando abaixo para ver o que acontece quando utilizamos esses valores em letras minúsculas:

```
v5 <- c(true, true, false, false)
```

O R retorna a mensagem de erro `Error: object 'true' not found` pois ele procura pelo primeiro objeto de nosso vetor `c(true, true, false, false)` na área de trabalho e, ao não encontrar, ele retorna esta mensagem de erro, justamente por não compreender `true` como um vetor lógico, e sim como um objeto! Se nós atribuímos valores a esses objetos, então a concatenação funciona, podendo assim atribuirmos este vetor ao objeto `v5` (ou com qualquer nome que desejamos):

```
true <- TRUE  
false <- FALSE  
v5 <- c(true, true, false, false)  
v5
```

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

Há no R valores constantes armazenados em objetos que podem ser chamados a qualquer momento por nós. São objetos que concatenam valores de texto, isto é, são vetores de texto. Vejamos abaixo alguns deles:

```
# essas constantes do R são vetores de texto  
LETTERS # letras maiusculas
```

```
## [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"  
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

```
letters # letras minusculas
```

```
## [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"  
## [20] "t" "u" "v" "w" "x" "y" "z"
```

```
month.abb # meses abreviados
```

```
## [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

```
month.name # meses por extenso
```

```
## [1] "January"   "February"  "March"     "April"      "May"       "June"  
## [7] "July"       "August"     "September" "October"    "November"   "December"
```

2.1.2 Sequências Numéricas & Repetições

É possível criar vetores numéricos usando a função `seq()` ou o operador `:`.

```
# usando o :
```

```
1:10 # cria uma sequencia de números inteiros 1 a 10
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
20:0 # cria uma sequencia de números inteiros 20 a 0
```

```
## [1] 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

```
0:-20 # cria uma sequencia de números inteiros 0 a -20
```

```
## [1] 0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -
14 -15 -16 -17 -18
## [20] -19 -20
```

usando a função `seq()` temos maior controle das sequências:

```
?seq # veja o help da função
```

```
seq(from = 1, to = 10, by = 0.5) # de 1 a 0 a cada 0.5
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0
## [16] 8.5 9.0 9.5 10.0
```

```
seq(from = 10, to = 0, by = -0.5) # de 10 a 0 a cada 0.5
```

```
## [1] 10.0 9.5 9.0 8.5 8.0 7.5 7.0 6.5 6.0 5.5 5.0 4.5 4.0 3.5 3.0
## [16] 2.5 2.0 1.5 1.0 0.5 0.0
```

```
seq(from = 100, to = 0, length.out = 10) # 10 valores igualmente espaçados de 100 a 0
```

```
## [1] 100.00000 88.88889 77.77778 66.66667 55.55556 44.44444 33.33333
## [8] 22.22222 11.11111 0.00000
```

É possível criar vetores repetindo valores através da função `rep()`:

```
# para números  
rep(5, times = 3) # cria um vetor com três elementos de valor 5
```

```
## [1] 5 5 5
```

```
rep(1:5, times = 3) # cria um vetor com três repetições da sequência de 1 a 5
```

```
## [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
```

```
rep(1:5, each = 3) # cria um vetor repetindo três vezes cada elemento da sequência de 1 a 5
```

```
## [1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5
```

Podemos também utilizar a função `rep()` com vetores de texto:

```
# para textos  
obj <- c("banana", "maça", "pera")  
rep(obj, times = 3)
```

```
## [1] "banana" "maça"   "pera"    "banana" "maça"   "pera"    "banana" "maça"  
## [9] "pera"
```

```
rep(obj, each = 3)
```

```
## [1] "banana" "banana" "banana" "maça"   "maça"   "maça"   "pera"   "pera"  
## [9] "pera"
```

2.2 Operações Matemáticas com Vetores

Todas as operações aplicadas a um vetor são aplicadas a cada um de seus elementos:

```
meuvetor <- 1:5 # uma sequencia de 1 a 5
mv2 <- meuvetor * 3 # uma sequencia onde cada valor de meuvetor foi multiplicado por 3
mv2
```

```
## [1] 3 6 9 12 15
```

```
mv2 <- meuvetor / 3 # uma sequencia onde cada valor de meuvetor foi dividido por 3
mv2
```

```
## [1] 0.3333333 0.6666667 1.0000000 1.3333333 1.6666667
```

```
# se usar uma função matemática com um vetor ela afetará cada elemento individualmente
meuvetor <- c(49, 25, 16, 4, 1)
sqrt(meuvetor) # raiz quadrada de cada elemento em meuvetor
```

```
## [1] 7 5 4 2 1
```

Operações com dois ou mais vetores são pareadas. Se os vetores têm o mesmo comprimento (mesmo número de elementos), então a operação é feita par a par, na ordem em que os elementos aparecem no vetor:

```
v1 <- c(1, 5, 10, 15)
v2 <- c(2, 4, 8, 16)
v1 + v2 # soma dos valores individuais e pareados
```

```
## [1] 3 9 18 31
```

```
v1 * v2
```

```
## [1] 2 20 80 240
```

```
v1^v2
```

```
## [1] 1.000000e+00 6.250000e+02 1.000000e+08 6.568408e+18
```



REGRA DA RECICLAGEM - se os vetores não têm o mesmo comprimento (mesmo número de elementos), então a operação é feita par a par, mas o vetor mais curto é reciclado, i.e. os elementos do vetor mais curto são repetidos sequencialmente até que a operação seja aplicada a todos os elementos do vetor mais longo (o R dará uma aviso quando a operação envolver vetores de tamanhos diferentes, pois às vezes não é isso que queremos).

A mais simples operação para entender a regra da reciclagem é a operação entre um vetor longo e um vetor atômico de um único valor:

```
v1 <- c(1, 5, 10, 15) # vetor com 4 elementos  
v2 <- 2 # vetor com 1 elemento  
v1 * v2 # cada elemento de v1 é multiplicado pelo único valor do vetor2
```

```
## [1] 2 10 20 30
```

Mas a reciclagem se aplica em todos os casos de operação entre vetores de tamanhos diferentes:

```
v1 <- c(1, 5, 10, 15) # vetor com 4 elementos  
v2 <- c(3, 2) # vetor com 2 elementos  
v1 * v2 # os valores de v1 são multiplicados par a par pelos valores de v2. Como v2 tem apenas
```

```
## [1] 3 10 30 30
```

Quanto temos vetores de tamanhos não múltiplos entre si, como por exemplo o objeto `ob` de tamanho 10 e o objeto `oc` de tamanho 3, o R executa a operação, porém retorna uma mensagem de alerta em que diz que o vetor de tamanho maior (`ob`) não é múltiplo do vetor de tamanho menor (`oc`):

```
ob <- rep(c(0, 1), each = 5)  
oc <- 1:3  
ob * oc
```

```
## Warning in ob * oc: longer object length is not a multiple of shorter object  
## length
```

```
## [1] 0 0 0 0 0 3 1 2 3 1
```

2.3 Funções com Vetores

Algumas funções operam sobre todo o vetor e não sobre cada elemento individualmente. Essas funções são utilizadas constantemente no R e, portanto, devemos conhecer as principais:

- `length()` e `sort()` - comprimento e ordenação de vetores

```
meuvetor <- 10:1
mv2 <- seq(30, 99, by = 3)
length(meuvetor) # quantos elementos tem meu vetor1
```

```
## [1] 10
```

```
length(mv2) # quantos elementos tem meu vetor2
```

```
## [1] 24
```

```
length(meuvetor) / length(mv2) # operação com os resultados
```

```
## [1] 0.4166667
```

```
mvord <- sort(meuvetor) # ordena os elementos em ordem crescente
mvord
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
sort(mvord, decreasing = TRUE) # ordena os elementos em ordem decrescentes
```

```
## [1] 10 9 8 7 6 5 4 3 2 1
```

- `mean()`, `sd()`, `min()`, `sum()` etc. - funções de estatística descritiva:

```
?mean # veja ajuda de uma dessas funções e navegue por outras
```

```
v1 <- c(2, 4, 6, 8, 4, 3, 5, 7)
sum(v1) # soma de todos os valores
```

```
## [1] 39
```

```
mean(v1) # média aritmética dos valores
```

```
## [1] 4.875
```

```
median(v1) # valor da mediana
```

```
## [1] 4.5
```

```
sd(v1) # desvio padrão
```

```
## [1] 2.03101
```

```
var(v1) # variância
```

```
## [1] 4.125
```

```
sqrt(var(v1)) # desvio padrão, pois este é a raíz quadrada da variância
```

```
## [1] 2.03101
```

```
min(v1) # valor mínimo
```

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

```
max(v1) # valor máximo
```

```
## [1] 8
```

```
range(v1) # mínimo e máximo
```

```
## [1] 2 8
```

```
diff(v1) # intervalos (diferenças entre valores consecutivos) entre os valores do vetor
```

```
## [1] 2 2 2 -4 -1 2 2
```

```
cumsum(v1) # soma cumulativa dos valores  
## [1] 2 6 12 20 24 27 32 39
```

2.4 Classes de vetores e fatores

Para entender os conceitos, vamos primeiro conhecer algumas funções úteis no entendimento das classes de objetos do R e algumas funções importantes: `class()`, `is.[class]()` e `as.[class]()`.

Vetores têm classes diferentes e todos os elementos de um vetor pertencem à mesma classe.

As principais classes são:

- `numeric` (=double, i.e. inclui casas decimais);
- `integer` (numérico mas de valor inteiro);
- `character` (texto);
- `logical` (verdadeiro ou falso);
- `date` (para datas).

A função `class()` nos permite saber a classe de um objeto do R.

```
?class # veja o help dessa função
```

```
v1 <- 1:20 # um vetor de números inteiros  
class(v1)
```

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

```
v2 <- seq(1, 10, by = 0.5) # um vetor de números  
class(v2)
```

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

```
v3 <- rep(c("A", "B"), each = 10) # um vetor de palavras (character)  
class(v3)
```

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

```
v4 <- c(T, T, F, F) # um vetor lógico  
class(v4)
```

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

```
v4 <- c(10, "A", 20, "B") # um vetor com misto de números e letras será convertido para texto  
class(v4)
```

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

```
v4
```

```
## [1] "10" "A"   "20" "B"
```

```
# veja que em v4 os elementos 10 e 20 viraram palavras, porque vetor só aceita elementos da mesma classe
```

As funções genéricas `is.[class]()` permitem você perguntar se um vetor é de uma determinada classe (`is?`). Ao utilizar essas funções, o R retornará um vetor lógico, ou seja, verdadeiro ou falso dependendo da classe do objeto:

```
v1 <- 1:20 # um vetor de números inteiros  
is.integer(v1) # verdadeiro
```

```
## [1] TRUE
```

```
is.numeric(v1) # também verdadeiro porque números inteiros também são números
```

```
## [1] TRUE
```

```
v3 <- rep(c("A", "B"), each = 10) # um vetor de palavras  
is.character(v3) # verdadeiro
```

```
## [1] TRUE
```

```
is.numeric(v3) # falso, porque o vetor contém palavras
```

```
## [1] FALSE
```

```
v4 <- c(10, "A", 20, "B") # um vetor com misto de números e letras  
is.numeric(v4) # falso, porque o vetor contém apenas palavras
```

```
## [1] FALSE
```

```
v4 <- c(T, T, F, F) # um vetor lógico  
is.logical(v4) # verdadeiro
```

```
## [1] TRUE
```

```
is.numeric(v4) # falso
```

```
## [1] FALSE
```

```
is.character(v4) # falso
```

```
## [1] FALSE
```

As funções genéricas `as.[class]()` (`as` = ‘como uma?’) permitem você converter um vetor de uma classe para outra. Em alguns casos, isso faz sentido; em outros, o retorno será de valores inexistentes (`NA`) ou não numéricos (`NaN`).

```
# conversão total  
v1 <- 1:20 # um vetor de números inteiros  
as.character(v1) # converte para texto um vetor numérico
```

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

```
# conversão parcial  
v4 <- c(10, "A", 20, "B") # um vetor com misto de números e letras  
as.numeric(v4) # converte cada elemento separadamente (regra da reciclagem é aplicada), o R va
```

```
## Warning: NAs introduced by coercion
```

```
## [1] 10 NA 20 NA
```

```
# um vetor de texto não pode virar número  
v3 <- rep(c("A", "B"), each = 10) # um vetor de palavras  
as.numeric(v3) # todos viram NA pois a conversão é inválida
```

```
## Warning: NAs introduced by coercion
```

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

```
# mas um vetor lógico pode virar número  
v4 <- c(T, T, F, F) # um vetor lógico  
as.numeric(v4) # verdadeiro vira 1 e falso vira 0 - isso é muito útil e é por isso que operaç
```

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

```
sum(v4)
```

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

```
mean(v4)
```

```
## [1] 0.5
```

```
min(v4)
```

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

O `factor` (=fator) na linguagem do R é um tipo especial de vetor com elementos de texto (classe `character`), em que os valores de texto são categorias. Isso tem algumas vantagens operacionais e sempre que o R precisa de um vetor de texto no formato de `factor`, ele converte automaticamente (se possível). No entanto, é muito importante que você entenda a diferença entre um vetor de classe `character` e um vetor de classe `factor`. Isso vai aparecer o tempo todo enquanto você usa o R e algumas vezes você precisará converter de um para outro. Funções importantes a serem memorizadas são:

- `levels()` - para ver (ou modificar) os níveis ou categorias de um fator;
- `as.factor()` e `as.vector()` - para converter entre fator e vetor.

```
# um exemplo de um vetor de palavras
mvv <- c("abacate", "banana", "mamão", "uva")
# repetindo cada fruta 3 vezes
mvv <- rep(mvv, each = 3)
# veja conteúdo
mvv
```

```
## [1] "abacate" "abacate" "abacate" "banana" "banana" "banana" "mamão"
## [8] "mamão"   "mamão"   "uva"     "uva"     "uva"
```

```
# qual a classe desse vetor?  
class(mvv)
```

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

```
# vamos converter esse vetor de character para um fator  
mvv2 <- as.factor(mvv)  
class(mvv2) # de fato mudou para factor
```

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

```
mvv2 # compare a estrutura deste objeto com mvv (apareceu a a palavra Levels:, que mostra as c
```

```
## [1] abacate abacate abacate banana banana banana mamão mamão mamão  
## [10] uva      uva      uva  
## Levels: abacate banana mamão uva
```

```
# por ser um fator você pode
```

```
levels(mvv2) # você pode ver os níveis do fator, ou seja as categorias que ele contém)
```

```
## [1] "abacate" "banana"   "mamão"    "uva"
```

```
levels(mvv2) <- c("abacate", "banana", "mamão", "uva") # você pode mudar/corrigir os níveis,  
mvv2 # veja como mudaram as categorias e os valores
```

```
## [1] abacate abacate abacate banana banana banana mamão mamão mamão  
## [10] uva      uva      uva  
## Levels: abacate banana mamão uva
```

```
as.numeric(mvv2) # você pode converter o fator em numérico, de forma que cada categoria vire u
```

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

```
as.numeric(mvv) # não pode fazer a mesma coisa com um vetor de palavras
```

```
## Warning: NAs introduced by coercion
```

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

A função `as.Date()` converte um vetor de trecho em um objeto de classe `date`. Datas são uma classe especial, que permite operações aritméticas para calcular distâncias temporais.

```
# muitas vezes queremos calcular tempo entre duas observações, como por exemplo, entre duas me
```

```
# Qual a diferença em dias entre duas datas?
```

```
data1 <- "31/03/1964"
```

```
data2 <- "17/04/2016"
```

```
# eu não posso simplesmente subtrair esses valores
```

```
data2 - data1
```

O R retorna uma mensagem de erro (`Error in data2 - data1 : non-numeric argument to binary operator`) porque esses objetos são de classe texto, e operações matemáticas só são permitidas com números ou datas. Vejam que a classe dos objetos criados acima são do tipo “texto” (`character`):

```
class(data1)  
## [1] "character"
```

```
class(data2)
```

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

Porém, se convertermos esses objetos para a classe `Date`, então poderemos fazer operações matemáticas com eles:

```
# mas o R tem um classe para datas  
# entao fazemos a conversao  
?as.Date # veja o help dessa função
```

```
data1 <- as.Date(data1, format = "%d/%m/%Y")  
data2 <- as.Date(data2, format = "%d/%m/%Y")  
# agora a classe mudou  
class(data1)
```

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

```
class(data2)
```

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

```
# posso fazer matemática com data  
data2 - data1
```

```
## Time difference of 19010 days
```

```
# note o argumento format, ele importa para o R possa entender o formato de sua data  
data3 <- "2016-04-21" # formato americano  
as.Date(data3, format = "%d/%m/%Y") # se eu usasse isso com o mesmo formato acima, o resultado
```

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

```
# mas mudando a simbologia do argumento format  
as.Date(data3, format = "%Y-%m-%d")
```

```
## [1] "2016-04-21"
```

```
# ele reconhece
```

2.5 Indexação

Já vimos que vetores são conjuntos de valores da mesma classe (seção 2). Esses valores tem uma posição dentro do vetor, ou seja, possuem um **índice**.

Já vimos também que podemos alterar a ordem dos valores, utilizando a função `sort()`, ou seja, alterar a posição dos elementos no vetor.

O índice identifica os elementos do vetor individualmente:

- pode ser um número equivalente à posição do elemento no vetor,
- ou pode ser um nome, quando os elementos do vetor tiverem um nome.

Entender indexação é fundamental para escrever bons códigos no R, pois isso se aplica também às matrizes e às outras classes de objetos do R.

Aqui vamos ver indexação de vetores, que é dada pelo operador `[]`.

2.5.1 Usando índices numéricos

```
# um vetor simples
v1 <- 1:10
v1[1] # valor na posição/índice 1
```

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

```
v1[8] # valor na posição/índice 8
```

```
## [1] 8
```

```
# em outra ordem
v1 <- 1:10
v1 <- sort(v1, decreasing = TRUE) # ordena decrescente
v1[1] # valor na posição/índice 1
```

```
## [1] 10
```

```
v1[8] # valor na posição/índice 8
```

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

2.5.2 Usando índices de nomes

Índices de nomes são elementos essenciais na manipulação de dados reais, pois nomes de linhas (seus registros) e nomes de colunas (suas variáveis) são nomes dos elementos que compõem a sua matriz. Índice de nomes preservam o identificador dos seus objetos (registros). Podemos atribuir nomes aos elementos do vetor usando a função `names()`. Uma outra função útil se chama `paste()`, muito utilizada na manipulação de textos.

```
?paste # veja o help da função paste
?names # veja o help da função names
```

```
# um vetor simples
v1 <- 1:10
# criando um vetor para usar como nomes
v2 <- paste("nome", v1, sep = "") # significa = use a regra da reciclagem e cole (paste) a pa
v2 # é portanto um conjunto de textos
```

```
## [1] "nome1" "nome2" "nome3" "nome4" "nome5" "nome6" "nome7" "nome8"
## [9] "nome9"  "nome10"
```

```
# note que é muito mais rápido fazer isso do que escrever nome1, nome2 ... nome10, certo?

# agora vamos atribuir v2 como nome dos elementos de v1
# para isso é importante que v1 e v2 tenham o mesmo comprimento
length(v1)
```

```
## [1] 10
```

```
length(v2)
```

```
## [1] 10
```

```
names(v1) # deve ser nulo, pois os elementos não tem nome
```

```
## NULL
```

```
names(v1) <- v2 # atribuimos os nomes  
v2 # pronto agora os elementos tem nome
```

```
## [1] "nome1" "nome2" "nome3" "nome4" "nome5" "nome6" "nome7" "nome8"  
## [9] "nome9"  "nome10"
```

```
# posso usar o nome como índice para pegar elementos  
v1["nome8"] # valor do elemento que tem nome = nome8
```

```
## nome8  
##     8
```

```
v1[8] # isso deve ser equivalente, pois criamos os nomes assim
```

```
## nome8  
##     8
```

```
# mas note a diferença quando reordenamos o vetor e mudamos os valores de posição
v3 <- sort(v1, decreasing = T)
v3[8] # o índice numérico pega outro valor
```

```
## nome3
##      3
```

```
v3["nome8"] # o indice de nome pega o mesmo valor (PRESERVA)
```

```
## nome8
##      8
```

2.6 Vetores e operadores lógicos

Para manipular dados no R, entender vetores lógicos e operadores lógicos é fundamental. Vetores lógicos são vetores de verdadeiros (`TRUE` ou apenas `T`, sempre em letras maiúsculas) ou falsos (`FALSE` ou `F`). Eles podem ser convertidos em vetores numéricos e, portanto, operados matematicamente ($T = 1$, e $F = 0$).

2.6.1 Fazendo perguntas à vetores

Vetores lógicos podem ser respostas às perguntas feitas por **operadores lógicos**:

- `>` - é maior que?
- `<` - é menor que?
- `>=` - é maior igual a?

- `<=` - é menor igual a?
- `==` - é igual a?
- `!=` - é diferente de?
- `%in%` - compara conteúdo de vetores

Há ainda a função `duplicated()` que busca valores repetidos em um vetor. O resultado desta função é um vetor contendo `TRUE` ou `FALSE`. Valores que possuam o valor `TRUE` são duplicados. Para checar os duplicados, devemos filtrar o resultado desta ação (veja na seção [2.6.2](#)).

```
# um vetor numerico
v1 <- 1:20
# quais valores de v1 são maiores ou iguais a 10
p1 <- v1 >= 10 # vai retornar um vetor lógico
p1
## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE
## [13] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

# soma dos verdadeiros responde "quantos valores de v1 são maiores ou iguais a 10, pois apenas
sum(p1)
## [1] 11

# experimente os demais operadores

# a regra da reciclagem também se aplica neste conceito
v1 <- 1:20
v2 <- 1:20
p2 <- v1 == v2 # compara cada par dos vetores que são idênticos
p2 # é o vetor lógico resultando, todos os valores são verdadeiros
```

```
## [1] TRUE  
## [16] TRUE TRUE TRUE TRUE TRUE
```

```
# portanto, as seguintes expressões também são verdadeiras  
sum(v1 == v2) == length(v1)
```

```
## [1] TRUE
```

```
# ou então  
sum(v1 == v2) == length(v2)
```

```
## [1] TRUE
```

```
# valores duplicados  
vv <- c(2, 2, 2, 3, 4, 5)  
vv # apenas o dois é duplicado
```

```
## [1] 2 2 2 3 4 5
```

```
duplicated(vv) # note que esta função retorna TRUE apenas para dois dos três valores 2 (o pri
```

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

```
# comparando vetores
v1 <- c(1, 2, 3, 4)
v2 <- c(4, 4, 5, 6)
v1 %in% v2 # quantos elementos de v1 existem em v2
```

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

```
sum(v1 %in% v2) # apenas 1
```

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

```
v2 %in% v1 # quais elementos de v2 estão em v1
```

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

```
sum(v2 %in% v1) # os dois quatro
```

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

```
notas.dos.alunos <- c(6.0, 5.1, 6.8, 2.8, 6.1, 9.0, 4.3, 10.4, 6.0, 7.9, 8.9, 6.8, 9.8, 4.6, 1)
```

```
## Quantos aprovados?
```

```
sum(notas.dos.alunos >= 5)
```

```
## [1] 14
```

```
# Qual a proporção de aprovados?
prop <- sum(notas.dos.alunos >= 5) / length(notas.dos.alunos)
prop
```

```
## [1] 0.7777778
```

```
# ou em texto
paste(round(prop * 100), "%", sep = "")
```

```
## [1] "78%"
```

Podemos usar também vetores de texto e fatores em conjunto com operadores lógicos.

```
# E VETORES DE TEXTO?
v1 <- rep(c("banana", "pera", "laranja", "limão"), 10)
v1 # um vetor de palavras
```

```
## [1] "banana" "pera"   "laranja" "limão"   "banana" "pera"   "laranja"
## [8] "limão"   "banana" "pera"   "laranja" "limão"   "banana" "pera"
## [15] "laranja" "limão"   "banana" "pera"   "laranja" "limão"   "banana"
## [22] "pera"   "laranja" "limão"   "banana" "pera"   "laranja" "limão"
## [29] "banana" "pera"   "laranja" "limão"   "banana" "pera"   "laranja"
## [36] "limão"   "banana" "pera"   "laranja" "limão"
```

```
# quantos elementos são iguais a banana
v1 == "banana"
```

```
## [1] TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE  
## [13] TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE  
## [25] TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE  
## [37] TRUE FALSE FALSE FALSE
```

```
sum(v1 == "banana")
```

```
## [1] 10
```

também poderia perguntar: quantos elementos de v1 contém banana

```
sum(v1 %in% "banana")
```

```
## [1] 10
```

```
v1 %in% "banana"
```

```
## [1] TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE  
## [13] TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE  
## [25] TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE  
## [37] TRUE FALSE FALSE FALSE
```

no caso acima == e %in% funcionam igual, mas o operador %in% é útil quando quisermos comparar

```
v2 <- c("banana", "pera", "abacate")  
v1 %in% v2 # quais elementos de v1 correspondem a elementos de v2
```

```
## [1] TRUE TRUE FALSE FALSE TRUE TRUE FALSE FALSE TRUE TRUE FALSE FALSE  
## [13] TRUE TRUE FALSE FALSE TRUE TRUE FALSE FALSE TRUE TRUE FALSE FALSE  
## [25] TRUE TRUE FALSE FALSE TRUE TRUE FALSE FALSE TRUE TRUE FALSE FALSE  
## [37] TRUE TRUE FALSE FALSE
```

```
sum(v1 %in% v2) # quantos são? 10 laranjas e 10 peras
```

```
## [1] 20
```

```
v2 %in% v1 # quais elementos de v2 estão em v1
```

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

```
sum(v2 %in% v1) # quantos são (apenas laranja e pera, abacate não está)
```

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

Operadores auxiliares permitem combinar perguntas:

- & equivale a E - essa condição E essa outra;
- | equivale a ou - essa condição OU essa outra;
- ! - inverte os valores da pergunta

```
# um vetor  
v1 <- 1:20  
v1
```

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

```
p1 <- v1 > 5 & v1 <= 15 # quais elementos de v1 são maiores que 5 E menores ou iguais a 15
sum(p1) # quantos são?
```

```
## [1] 10
```

```
p1 <- v1 > 5 | v1 <= 15 # quais elementos de v1 são maiores que 5 OU menores ou iguais a 15
sum(p1) # quantos são?
```

```
## [1] 20
```

```
# !exclamação NEGA ou INVERTE verdadeiros e falsos
v1 <- 1:20
sum(v1 == 5) # quantos v1 são iguais a 5?
```

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

```
sum(!v1 == 5) # quantos v1 são diferentes de 5?
```

```
## [1] 19
```

```
sum(v1 > 5) # quantos v1 são maiores que 5?
```

```
## [1] 15
```

```
sum(!v1 > 5) # quantos v1 são menores que 5?
```

```
## [1] 5
```

```
# texto
v1 <- rep(c("banana", "pera", "laranja", "limão"), 10)
v1 # um vetor de palavras
```

```
## [1] "banana" "pera"   "laranja" "limão"   "banana" "pera"   "laranja"
## [8] "limão"   "banana" "pera"   "laranja" "limão"   "banana" "pera"
## [15] "laranja" "limão"   "banana" "pera"   "laranja" "limão"   "banana"
## [22] "pera"   "laranja" "limão"   "banana" "pera"   "laranja" "limão"
## [29] "banana" "pera"   "laranja" "limão"   "banana" "pera"   "laranja"
## [36] "limão"   "banana" "pera"   "laranja" "limão"
```

```
vl <- v1 == "banana" & v1 == "pera" # quantos elementos de v1 sao banana E sao pera
vl
```

```
## [1] FALSE FALSE
## [13] FALSE FALSE
## [25] FALSE FALSE
## [37] FALSE FALSE FALSE FALSE
```

```
sum(vl) # nenhum valor satisfaz as duas condicoes
```

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

```
vl <- v1 == "banana" | v1 == "pera" # quantos elementos de v1 sao banana ou sao pera  
vl
```

```
## [1] TRUE TRUE FALSE FALSE TRUE TRUE FALSE FALSE TRUE TRUE FALSE FALSE  
## [13] TRUE TRUE FALSE FALSE TRUE TRUE FALSE FALSE TRUE TRUE FALSE FALSE  
## [25] TRUE TRUE FALSE FALSE TRUE TRUE FALSE FALSE TRUE TRUE FALSE FALSE  
## [37] TRUE TRUE FALSE FALSE
```

```
sum(vl) # tem 20 valores que satisfazem 1 das condições
```

```
## [1] 20
```

```
# isso é o mesmo que pergunta desse outro jeito:  
sum(v1 %in% c("banana", "pera"))
```

```
## [1] 20
```

2.6.2 Filtrando dados com vetores lógicos

Vetores lógicos podem ser usados como índices (Seção 2.5) para filtrar elementos de um vetor. É através deste conceito que podemos filtrar dados de matrizes e criar subconjunto de dados.

```
# um vetor com sequencia de 1 a 100  
v1 <- 1:100  
  
p1 <- v1 > 15 # Pergunta 1 quantos são maiores que 15  
v1[p1] # valores que satisfazem a pergunta 1
```

```
## [1] 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
## [20] 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
## [39] 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
## [58] 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91
## [77] 92 93 94 95 96 97 98 99 100
```

```
p2 <- v1 <= 20 # Pergunta 2 quantos são menores ou iguais a 20
v1[p2] # valores que satisfazem a pergunta 2
```

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

```
# quantos satisfazem as duas perguntas
p3 <- p1 & p2
v1[p2] # valores que satisfazem as duas perguntas
```

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

A função `grep()` permite a busca de uma palavra (ou pedaço dela) em um vetor de palavras. Mais de uma palavra pode ser buscada ao mesmo tempo.

```
?grep # veja o help dessa função e seus argumentos
```

```
# um vetor de palavras
v1 <- rep(c("banana", "pera", "laranja", "limão"), 5)
grep("an", v1) # quais elementos tem a palavra 'an' no nome?
```

```
## [1] 1 3 5 7 9 11 13 15 17 19
```

```
# note que é case.sensitive (depende se é maiusculo ou minúsculo)
grep("An", v1) # não encontra nada
```

```
## integer(0)
```

```
grep("An", v1, ignore.case = T) # mas eu posso dizer para ele ignorar se é minusculo ou maiúsculo
```

```
## [1] 1 3 5 7 9 11 13 15 17 19
```

```
# quem são esses elementos
vl <- grep("An", v1, ignore.case = T) # pega os índices desses elementos
v1[vl]
```

```
## [1] "banana" "laranja" "banana" "laranja" "banana" "laranja" "banana"
## [8] "laranja" "banana" "laranja"
```

```
unique(v1[vl]) # valores únicos desse vetor
```

```
## [1] "banana" "laranja"
```

2.6.3 Perguntando por valores ausentes - NA

Vimos anteriormente como o R codifica valores ausentes (seção 1.8.4): converte em uma classe lógica definida pela palavra `NA` em maiúsculo. E nossos dados frequentemente têm valores ausentes. Isso vai gerar avisos indesejáveis e impedir certas análises. Então, muitas vezes precisamos tirar registros com valores ausentes ou colunas com muitos valores ausentes.

Perguntar por valores ausentes no R é feito por uma função especial chamada `is.na()`. A resposta da função é um vetor lógico indicando quem é e quem não é NA. Há uma outra função chamada `na.omit()` que elimina valores NA de um vetor.

```
?is.na # veja o help
```

```
# um vetor com NAs
v1 <- c(NA, NA, 1, 2, 3, 4, 5, 6)
is.na(v1) # quem é NA?
```

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

```
v2 <- v1[!is.na(v1)] # criar um vetor novo com quem não é NA (note o !)
v2
```

```
## [1] 1 2 3 4 5 6
```

```
# isso também pode ser feito com na.omit()
?na.omit # veja o help dessa função
```

```
v3 <- na.omit(v1)
v3 # a diferença é que criou um objeto de classe na.omit
```

```
## [1] 1 2 3 4 5 6
## attr(),"na.action"
## [1] 1 2
## attr(),"class"
## [1] "omit"
```

```
v3 <- as.vector(v3) # isso elimina a diferença, convertendo em vetor  
v3 # agora é idêntico a v2
```

```
## [1] 1 2 3 4 5 6
```

```
# agora suponha o seguinte vetor  
v4 <- c("NA", "NA", "pera", "banana", "mamão")  
is.na(v4) # ops todos são falsos
```

```
## [1] FALSE FALSE FALSE FALSE FALSE
```

```
# isso porque "NA" é texto e não um objeto de classe lógica  
class(NA)
```

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

```
class("NA")
```

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

```
# mas eu poderia corrigir isso  
v4[v4 == "NA"] # vejo
```

```
## [1] "NA" "NA"
```

```
v4[v4 == "NA"] <- NA # corrojo  
v4  
  
## [1] NA NA "pera" "banana" "mamão"
```

```
is.na(v4) # agora dois são NAs  
  
## [1] TRUE TRUE FALSE FALSE FALSE
```

```
# note que agora todos são diferentes de "NA" como texto  
v4[!v4 == "NA"]
```

```
## [1] NA NA "pera" "banana" "mamão"
```

```
# mas isso de mostra quem não é corretamente  
v5 <- v4[!is.na(v4)]  
v5
```

```
## [1] "pera" "banana" "mamão"
```

2.7 Para saber mais:

Veja nossas vídeoaulas com parte do conteúdo deste capítulo:

- Criação de vetores¹.

¹<https://youtu.be/qXSZkGoDkIY>

- Sequências numéricas e repetições².
 - Indexação³.
 - Vetores e Operadores Lógicos⁴.
-

2.8 Exercícios

- Resolva o exercício 102.04 Biomassa de Árvores⁵.
- Resolva o exercício 102.02 Sequências⁶.
- Resolva o exercício 102.03 Conta de Luz⁷.
- Resolva o exercício 102.01 Área Basal⁸.
- Resolva o exercício 102.05 Variância na Unha⁹.
- Resolva o exercício 101.03 Objetos de Data¹⁰.

²<https://youtu.be/PJ02yj0gnWc>

³http://www.botanicaamazonica.wiki.br/labotam/lib/exe/fetch.php?media=bot89:precurso:5vetores:video01_bot89-2020-04-07_07.38.30.mp4

⁴http://www.botanicaamazonica.wiki.br/labotam/lib/exe/fetch.php?media=bot89:precurso:6vetores:video01_bot89-2020-04-07_07.52.00.mp4

⁵<http://notar.ib.usp.br/exercicio/19>

⁶<http://notar.ib.usp.br/exercicio/12>

⁷<http://notar.ib.usp.br/exercicio/4>

⁸<http://notar.ib.usp.br/exercicio/17>

⁹<http://notar.ib.usp.br/exercicio/5>

¹⁰<http://notar.ib.usp.br/exercicio/18>



3

Objeto II - Matrizes e data.frames

3.1 Matriz vs. data.frame

Objetos de classe `matrix` ou `data.frame` são objetos bidimensionais (tem linhas e colunas), e constituem a forma como nossos dados estão organizados. Precisamos entender a diferença entre essas classes e suas propriedades.

Objetos de classe `matrix` contêm linhas e colunas, mas os valores de toda a matriz são da mesma classe (`numeric`, `character`, ou `logical`, por exemplo). Operações matemáticas com matrizes utilizam matrizes numéricas, portanto, de uma única classe, `matrix`.

Objetos de classe `data.frame` também contém linhas e colunas, mas podem misturar colunas de classes diferentes (`numeric` e `character`, `factor`, e `logical`, por exemplo). Quando importamos dados ao R, geralmente atribuímos os dados a um objeto de classe `data.frame`.

Podemos converter um objeto de classe `matrix` para `data.frame` e vice-versa, usando as funções `as.data.frame()` ou `as.matrix()`. Porém, quando convertemos os dados para um objeto de classe `matrix`, todos os dados passam a ser da mesma classe, geralmente havendo perda de dados.

3.1.1 Criando matrizes

Poder criar uma matriz no R é muito útil para várias finalidades como, por exemplo, simular dados em testes de permutação ou preencher uma tabela com resultados de uma análise. Matrizes podem ser criadas de diferentes formas (e.g., pode juntar matrizes pelas linhas

e colunas, ou pode extrair sub-matrizes de uma matriz). Para criar matrizes, a função básica se chama `matrix()`:

```
# veja o help da função
?matrix

# a função se usa assim: matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
# onde:
# data = NA #um vetor de comprimento igual ao número de células desejadas que é nrow*ncol.
# byrow = FALSE #A forma de preenchimento da planilha pelos dados em data. Se byrow=TRUE, então
# colunas
# nrow = número de linhas
# ncol = número de colunas
# dimnames = um objeto do tipo lista (que ainda não vimos), com dois vetores, um com os nomes
```

```
# exemplo 1 - matriz de 3x3 com zeros
mm <- matrix(data = 0, nrow = 3, ncol = 3, byrow = F, dimnames = NULL)
mm
```

0	0	0
0	0	0
0	0	0

```
# note que data tem comprimento 1, apenas 1 valor. Pela regra da reciclagem ele é repetido até
# exemplo2 - matriz de 3x3 com valores
dd <- 1:9 # nove valores
mm <- matrix(data = dd, nrow = 3, ncol = 3, byrow = F, dimnames = NULL)
mm
```

1	4	7
2	5	8
3	6	9

```
# mudando byrow para TRUE preenchemos pelas linhas
mm2 <- matrix(data = dd, nrow = 3, ncol = 3, byrow = TRUE, dimnames = NULL)
mm2
```

1	2	3
4	5	6
7	8	9

```
# exemplo3 - matriz de 3x3 com valores e nomes de colunas e linhas
# define dimensao
nrow <- 3
ncol <- 3
# define data
dd <- 1:9 # nove valores
# define nome de colunas
cln <- paste("coluna", 1:ncol, sep = "")
# define nome de linhas
lln <- paste("linha", 1:nrow, sep = "")

mm <- matrix(data = dd, nrow = nrow, ncol = ncol, byrow = F, dimnames = list(lln, cln))
```

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

Para unir ou criar matrizes (e *data.frames*) temos duas funções úteis:

- `rbind()`, que vem do inglês *row bind*, ou seja, cole linhas;
- `cbind()`, que vem do inglês *column bind*, ou seja, cole colunas.

```
# vetores numéricos de mesmo comprimento
v1 <- 1:10
v2 <- 10:1
v3 <- 11:20
# essas duas condições devem ser verdadeiras
length(v1) == length(v2)
```

```
## [1] TRUE
```

```
length(v1) == length(v3)
```

```
## [1] TRUE
```

```
# entao posso criar uma matriz juntando esses vetores em linhas ou colunas
mml <- rbind(v1, v2, v3)
class(mml) # criou um matrix
```

```
## [1] "matrix" "array"
```

```
mml
```

v1	1	2	3	4	5	6	7	8	9	10
v2	10	9	8	7	6	5	4	3	2	1
v3	11	12	13	14	15	16	17	18	19	20

```
# ou
mmc <- cbind(v1, v2, v3)
class(mmc)
```

```
## [1] "matrix" "array"
```

```
# ou se eu já tenho uma matriz, posso usar essas funções para adicionar novas linhas ou colunas  
novovetor <- 31:40
```

```
# por linha
```

```
ncol(mml) == length(novovetor) # neste caso o número de colunas da matrix precisa ser igual ao
```

```
## [1] TRUE
```

```
mml <- rbind(mml, novovetor) # junto a matrix existente com o novo vetor, adicionando uma nova  
mml
```

v1	1	2	3	4	5	6	7	8	9	10
v2	10	9	8	7	6	5	4	3	2	1
v3	11	12	13	14	15	16	17	18	19	20
novovetor	31	32	33	34	35	36	37	38	39	40

```
# note que a nova linha recebeu como nome o nome do objeto que continha o dado
```

```
# por coluna
```

```
nrow(mmc) == length(novovetor) # neste caso o número de linhas da matrix precisa ser igual ao
```

```
## [1] TRUE
```

```
mmc <- cbind(mmc, novovetor) # junto a matrix existente com o novo vetor, adicionando uma nova  
mmc
```

v1	v2	v3	novovetor
1	10	11	31
2	9	12	32
3	8	13	33
4	7	14	34
5	6	15	35
6	5	16	36
7	4	17	37
8	3	18	38
9	2	19	39
10	1	20	40

```
# note que a nova coluna recebeu como nome o nome do objeto que continha o dado
```

3.1.2 Criando data.frames

Objetos de classe `data.frame` são tabelas de dados, apresentam duas dimensões, e permitem misturar dados de classes diferentes, numéricos, texto (character ou factor) e lógicos. Quando importamos nossos dados ao R, em geral criamos objetos de classe `data.frame`. Para criar ou converter dados em `data.frames`, podemos usar as funções `data.frame()` e `as.data.frame()`.

```
?data.frame # veja o help das funções acima
```

```
# a função que cria o objeto é
data.frame(..., row.names = NULL, check.rows = FALSE, check.names = TRUE, stringsAsFactors = FALSE)

# de todos os argumentos os mais importantes são:
# ... #que pode ser vetores ou tag = vetor (os dados da tabela)
# stringsAsFactors #que especifica se queremos os textos como vetores ou fatores
```

```
# exemplo 1 -  
# Primeiro criamos alguns dados  
# um vetor numerico  
v1 <- 1:10  
# um vetor de letras do mesmo comprimento usando a constante LETTERS  
v2 <- LETTERS[1:10]  
# um vetor de palavras de mesmo comprimento  
v3 <- rep(c("fulano", "jose", "joaquim", "martin"), length.out = length(v1))
```

```
# Juntamos num data.frame com fatores  
dd <- data.frame(v1, v2, v3, stringsAsFactors = T)  
class(dd) # é um data frame
```

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

```
dim(dd) # dimensoes, linhas e colunas
```

```
## [1] 10 3
```

```
ncol(dd) # numero de colunas
```

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

```
nrow(dd) # numero de linhas
```

```
## [1] 10
```

```
str(dd) # estrutura do objeto (veja as classes das colunas)
```

```
## 'data.frame':   10 obs. of  3 variables:  
## $ v1: int  1 2 3 4 5 6 7 8 9 10  
## $ v2: Factor w/ 10 levels "A","B","C","D",...: 1 2 3 4 5 6 7 8 9 10  
## $ v3: Factor w/ 4 levels "fulano","joaquim",...: 1 3 2 4 1 3 2 4 1 3
```

```
# JUNTAMOS SEM FATORES  
dd2 <- data.frame(v1, v2, v3, stringsAsFactors = FALSE)  
class(dd2) # é um data frame
```

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

```
str(dd2) # estrutura do objeto (veja as classes das colunas)
```

```
## 'data.frame':   10 obs. of  3 variables:  
## $ v1: int  1 2 3 4 5 6 7 8 9 10  
## $ v2: chr  "A" "B" "C" "D" ...  
## $ v3: chr  "fulano" "jose" "joaquim" "martin" ...
```

```
# juntamos com nome de colunas (tag = vetor)  
dd2 <- data.frame(RegisterID = v1, CódigoZ = v2, Pessoa = v3, stringsAsFactors = FALSE)  
dd2
```

RegistorID	CodigoZ	Pessoa
1	A	fulano
2	B	jose
3	C	joaquim
4	D	martin
5	E	fulano
6	F	jose
7	G	joaquim
8	H	martin
9	I	fulano
10	J	jose

```
# agora vamos usar o cbind que vimos acima
dz <- cbind(v1, v2, v3)
# ou entao usando tag=vetor para ter nomes das colunas de acordo
dz <- cbind(RegistorID = v1, CodigoZ = v2, Pessoa = v3)
class(dz) # isso cria uma matriz
```

```
## [1] "matrix" "array"
```

```
str(dz) # todos os dados são da mesma classe (texto)
```

```
## chr [1:10, 1:3] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "A" "B" "C" "D" ...
## - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:3] "RegistorID" "CodigoZ" "Pessoa"
```

```
dz <- as.data.frame(dz) # convertemos num data.frame
class(dz) # é um data.frame
```

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

```
str(dz) # converte numeros para numerico e texto para fator
```

```
## 'data.frame':   10 obs. of  3 variables:
## $ RegisterID: chr  "1" "2" "3" "4" ...
## $ CódigoZ    : chr  "A" "B" "C" "D" ...
## $ Pessoa      : chr  "fulano" "jose" "joaquim" "martin" ...
```

```
dz <- as.data.frame(as.matrix(dz), stringsAsFactors = FALSE) # convertemos num data.frame sem
str(dz) # converte numeros para numerico e texto para character
```

```
## 'data.frame':   10 obs. of  3 variables:
## $ RegisterID: chr  "1" "2" "3" "4" ...
## $ CódigoZ    : chr  "A" "B" "C" "D" ...
## $ Pessoa      : chr  "fulano" "jose" "joaquim" "martin" ...
```

3.1.3 Funções importantes na manipulação de matrizes e data.frames

As funções `head()` e `tail()` mostram o cabeçalho e rodapé tanto para matrizes como para `data.frames`, respectivamente. Vejam o `?` dessas duas funções:

```
?head
?tail
```

```
# Primeiro criamos alguns dados
# um vetor numerico
v1 <- 1:10
# um vetor de letras do mesmo comprimento usando a constante LETTERS
v2 <- LETTERS[1:10]
```

```
# um vetor de palavras de mesmo comprimento  
v3 <- rep(c("fulano", "jose", "joaquim", "martin"), length.out = length(v1))  
# juntamos com nome de colunas (tag = vetor) e com  
dd2 <- data.frame(RegisterID = v1, CódigoZ = v2, Pessoa = v3, stringsAsFactors = TRUE)  
# cabeçalho  
head(dd2) # primeiras 6 linhas
```

RegisterID	CódigoZ	Pessoa
1	A	fulano
2	B	jose
3	C	joaquim
4	D	martin
5	E	fulano
6	F	jose

```
head(dd2, 3) # três primeiras linhas
```

RegisterID	CódigoZ	Pessoa
1	A	fulano
2	B	jose
3	C	joaquim

```
# rodapé  
tail(dd2) # seis últimas linhas
```

	RegisterID	CódigoZ	Pessoa
5	5	E	fulano
6	6	F	jose
7	7	G	joaquim
8	8	H	martin
9	9	I	fulano
10	10	J	jose

```
tail(dd2, 3) # três últimas linhas
```

	RegistorID	CodigoZ	Pessoa
8	8	H	martin
9	9	I	fulano
10	10	J	jose

As funções `dim()`, `nrow()` e `ncol()` informam as dimensões de matrizes e data.frames, número de linhas, e número de colunas, respectivamente.

```
dim(dd2) # vetor com dois valores, número de linhas e número de colunas
```

```
## [1] 10 3
```

```
nrow(dd2) # número de linhas do data.frame ou matrix
```

```
## [1] 10
```

```
ncol(dd2) # número de colunas do data.frame ou matrix
```

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

```
nrow(as.matrix(dd2))
```

```
## [1] 10
```

```
ncol(as.matrix(dd2))
```

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

As funções `str()` e `summary()` informam a estrutura dos `data.frames` e o resumo dos dados, respectivamente.

```
str(dd2) # mostra a estrutura do objeto, quais colunas, classes de colunas e total de valores
```

```
## 'data.frame':    10 obs. of  3 variables:
## $ RegisterID: int  1 2 3 4 5 6 7 8 9 10
## $ CódigoZ   : Factor w/ 10 levels "A","B","C","D",...: 1 2 3 4 5 6 7 8 9 10
## $ Pessoa     : Factor w/ 4 levels "fulano","joaquim",...: 1 3 2 4 1 3 2 4 1 3
```

```
summary(dd2) # mostra para cada coluna a variação encontrada: estatística descritiva de variáv
```

	RegisterID	CódigoZ	Pessoa
Min. :1.00	A :1	fulano :3	
1st Qu.: 3.25	B :1	joaquim:2	
Median : 5.50	C :1	jose :3	
Mean : 5.50	D :1	martin :2	
3rd Qu.: 7.75	E :1	NA	
Max. :10.00	F :1	NA	
NA	(Other):4	NA	

As funções `colnames()` e `rownames()` permitem VER e ATRIBUIR valores de nomes de linhas e colunas em `data.frames` e matrizes. Em um `data.frame`, os nomes de linhas DEVEM SER ÚNICOS e não podem ter duas linhas com o mesmo nome. São códigos que identificam registros únicos. Isso é muito importante para o entendimento dos identificadores dos seus dados.

```
# vamos criar uma matriz com nomes de linhas e colunas
mm <- matrix(1:9, nrow = 3, ncol = 3, dimnames = list(paste("linha", 1:3, sep = ""), paste("coluna", 1:3, sep = "")))
# e converter essa matriz para um data.frame
dd <- as.data.frame(mm)

# vamos também criar outra matriz SEM nomes de linhas e colunas
mm2 <- matrix(1:9, nrow = 3, ncol = 3)
# e converter essa matriz para um data.frame
dd2 <- as.data.frame(mm2)
dd2
```

V1	V2	V3
1	4	7
2	5	8
3	6	9

```
# para os objetos com nomes podemos ver os nomes
rownames(mm)
```

```
## [1] "linha1" "linha2" "linha3"
```

```
rownames(dd)
```

```
## [1] "linha1" "linha2" "linha3"
```

```
colnames(mm)
```

```
## [1] "coluna1" "coluna2" "coluna3"
```

```
colnames(dd)
```

```
## [1] "coluna1" "coluna2" "coluna3"
```

```
# para os objetos sem nomes  
rownames(mm2) # nulo, não tem nome
```

```
## NULL
```

```
rownames(dd2) # números em formato de texto
```

```
## [1] "1" "2" "3"
```

```
colnames(mm2) # nulo, não tem nome
```

```
## NULL
```

```
colnames(dd2) # V1 a Vncol(dd) - ele cria nomes das colunas
```

```
## [1] "V1" "V2" "V3"
```

```
# note que no caso do data.frame dd2, apesar de não ter nome de linha e coluna, o R criou uma
```

```
# essas funções permitem VER mas também permitem ATRIBUIR (modificar) nomes
# modificando quem já tem nome (matriz, mas funciona igual para dd)
colnames(mm) # nomes atuais
```

```
## [1] "coluna1" "coluna2" "coluna3"
```

```
colnames(mm) <- c("novonome1", "novonome2", "novonome3")
mm # veja como o nome das colunas mudou
```

	novonome1	novonome2	novonome3
linha1	1	4	7
linha2	2	5	8
linha3	3	6	9

```
# mudando apenas o nome da coluna2
colnames(mm)[2] <- "colunaDOIS"
colnames(mm) # nomes atuais
```

```
## [1] "novonome1"  "colunaDOIS" "novonome3"
```

```
# atribuindo quando não tem nome
colnames(mm2) # está vazio ou não existe (NULL)
```

```
## NULL
```

```
colnames(mm2) <- paste("banana", 1:ncol(mm2), sep = "-")
mm2 # agora tem nome de coluna
```

banana-1	banana-2	banana-3
1	4	7
2	5	8
3	6	9

```
rownames(mm2) # nomes de linhas também está vazio
```

```
## NULL
```

```
rownames(mm2) <- paste("chuchu", 1:nrow(mm2), sep = ".")  
mm2 # agora tem nomes de linha e coluna
```

	banana-1	banana-2	banana-3
chuchu.1	1	4	7
chuchu.2	2	5	8
chuchu.3	3	6	9

Vamos tentar atribuir um mesmo nome de linha `teste1` a duas linhas de nossa matriz `mm2` e ver o que acontece:

```
rownames(mm2)[1:2] <- "teste1" # coloque o nome teste1 para as linhas 1 e 2 FUNCIONA PARA MATRIZ
```

Reparam que um mesmo nome de linha pode ser utilizado em mais de uma linha de uma matriz. Será que isso pode ser feito em um *data.frame*? Vejamos:

```
rownames(dd)[1:2] <- "teste1" # nao funciona, porque ele não aceita nomes repetidos de linhas
```

3.2 Indexação de matriz e data.frame

Entender indexação é fundamental para manipular dados no R. Em indexação de vetores (seção 2.5), vimos que é possível usar números, códigos/nomes ou valores de verdadeiro ou falso (lógico), como elementos para visualizar, filtrar e mudar dados em vetores unidimensionais.

O mesmo operador, `[]`, pode ser usado para indexação de uma matriz ou um `data.frame`. A única diferença é que, por matrizes e `data.frames` serem bidimensionais, precisamos indicar a qual dimensão estamos nos referindo. Portanto o operador de indexação para matrizes e `data.frame` tem a seguinte estrutura, `[índiceDeLinha, índiceDeColuna]`, em que a vírgula separa os índices de linha e coluna.

3.2.1 Matrizes

```
# vamos criar uma matriz
mm <- matrix(1:9, nrow = 3, ncol = 3, dimnames = list(paste("linha", 1:3, sep = ""), paste("co
```

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

```
# USANDO INDICE NUMÉRICO
mm[1, 2] # mostra o elemento da linha 1 e coluna 2
```

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

```
mm[1, ncol(mm)] # mostra o elemento da linha 1 e última coluna
```

```
## [1] 7
```

```
mm[nrow(mm), ncol(mm)] # mostra o elemento da última linha e última coluna
```

```
## [1] 9
```

```
mm[, 1] # mostra a coluna 1
```

```
## linha1 linha2 linha3  
##      1      2      3
```

```
# eu posso juntar indices de matrizes e vetores na mesma linha  
mm[, 1][2] # mostra o segundo elemento do vetor correspondente a primeira coluna
```

```
## linha2  
##      2
```

```
mm[1, ] # mostra a linha 1
```

```
## coluna1 coluna2 coluna3  
##      1      4      7
```

```
mm[nrow(mm), ] # mostra a ultima linha
```

```
## coluna1 coluna2 coluna3
##      3      6      9
```

```
mm[, 1:2] # mostra as duas primeiras colunas
```

	coluna1	coluna2
linha1	1	4
linha2	2	5
linha3	3	6

```
mm[1:2, 1:3] # mostra as duas primeiras linhas e duas primeiras colunas
```

	coluna1	coluna2	coluna3
linha1	1	4	7
linha2	2	5	8

```
mm[3:nrow(mm), ] # mostra da linha tres a ultima linha
```

```
## coluna1 coluna2 coluna3
##      3      6      9
```

```
mm[c(3, 1), c(3, 2)] # mostra as linhas 3 e 1 e colunas 3 e 2 (nessa ordem)
```

	coluna3	coluna2
linha3	9	6
linha1	7	4

```
# USANDO INDICES DE NOMES
```

```
mm["linha1", ] # mostra a linha 1 - note que poderia ser outro nome, poderia ter chamado no i
```

```
## coluna1 coluna2 coluna3
```

```
##      1      4      7
```

```
mm[, "coluna1"] # mostra a coluna 1
```

```
## linha1 linha2 linha3
```

```
##      1      2      3
```

```
mm[c("linha3", "linha1"), c("coluna3", "coluna1")] # mostra a linhas 3 e 1 e colunas 3 e 1. NO
```

	coluna3	coluna1
linha3	9	3
linha1	7	1

```
# SE EU POSSO VER EU POSSO MUDAR
```

```
mm
```

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

```
mm[1, 3] # elemento da linha 1 coluna 3
```

```
## [1] 7
```

```
mm[1, 3] <- 33 # mudei o elemento
mm[2, 2:3]
```

```
## coluna2 coluna3
##      5      8
```

```
mm[2, 2:3] <- mm[2, 2:3] * 10 # mudei os valores das colunas 2 e 3 para a linha 2, multiplicar
mm[2, 2:3]
```

```
## coluna2 coluna3
##      50     80
```

```
mm
```

	coluna1	coluna2	coluna3
linha1	1	4	33
linha2	2	50	80
linha3	3	6	9

3.2.2 data.frame

O operador [índiceDeLinha , índiceDeColuna] também funciona para data.frames. Outro operador útil na manipulação de data.frames é o \$. Ele permite a visualização e atribuição de valores a qualquer coluna.

```
# vamos criar uma matriz com nomes de linhas e colunas
mm <- matrix(1:9, nrow = 3, ncol = 3, dimnames = list(paste("linha", 1:3, sep = ""), paste("co
```

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

```
# convertemos para um data.frame
dd <- as.data.frame(mm)
dd$coluna1 # pego a coluna 1 (note que o nome da coluna vai sem "aspas")
```

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

Veja que o uso do operador `$` não funciona em matrizes:

```
mm$coluna1 # veja como não funciona para o objeto matrix
```

```
dd$coluna1[2] # vejo o segundo elemento da coluna1
```

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

```
# isso é o mesmo que
dd[2, "coluna1"]
```

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

```
# se eu vejo eu posso mudar
dd[2, "coluna1"] <- 10
dd$coluna1[3] <- 20
dd$coluna3 # pego a coluna tres
```

```
## [1] 7 8 9
```

```
# também posso adicionar uma nova coluna
dd$novacoluna <- LETTERS[1:nrow(dd)]
dd # agora tenho uma nova coluna
```

	coluna1	coluna2	coluna3	novacoluna
linha1	1	4	7	A
linha2	10	5	8	B
linha3	20	6	9	C

```
# ou poderia usar outra forma
dd[, "nova2"] <- LETTERS # não vai funcionar por estou atribuindo um vetor muito mais longo da
```

```
length(LETTERS) > nrow(dd) # essa expressão é verdadeira
```

```
## [1] TRUE
```

```
dd[, "nova2"] <- LETTERS[1:nrow(dd)] # isso tem o mesmo comprimento e funciona
dd
```

	coluna1	coluna2	coluna3	novacoluna	nova2
linha1	1	4	7	A	A
linha2	10	5	8	B	B
linha3	20	6	9	C	C

```
# posso adicionar uma coluna vazia
dd$outracoluna <- NA
dd
```

	coluna1	coluna2	coluna3	novacoluna	nova2	outracoluna
linha1	1	4	7	A	A	NA
linha2	10	5	8	B	B	NA
linha3	20	6	9	C	C	NA

```
# e ainda outra (lógica)
dd$maisuma <- TRUE
dd
```

	coluna1	coluna2	coluna3	novacoluna	nova2	outracoluna	maisuma
linha1	1	4	7	A	A	NA	TRUE
linha2	10	5	8	B	B	NA	TRUE
linha3	20	6	9	C	C	NA	TRUE

Adicionar colunas em uma matriz é um pouco diferente do que se faz com um `data.frame`:

```
# primeiro não posso usar $ porque matrix não entende isso
class(mm) # é uma matrix
mm$colun3 # isso não funciona
```

```
mm[, "coluna3"] # isso funciona
```

```
## linha1 linha2 linha3
##      7      8      9
```

```
# adicionando uma coluna
```

```
mm[, 4] # isso não existe  
mm[, 4] <- log(mm[, "coluna3"]) # isso não funciona
```

```
# poderia usar a função cbind que vimos anteriormente  
mm <- cbind(mm, LOGCOLUNA3 = log(mm[, "coluna3"]))) # assim eu posso
```

3.3 Filtrando e ordenando matrizes e `data.frames`

3.3.1 Filtragem de dados

Já vimos como fazer perguntas sobre vetores (Seção 2.6.1) e obter vetores lógicos ou valores de índices que nos permitem extrair ou filtrar de vetores os dados que satisfazem às condições das perguntas feitas. Aqui vamos estender isso para objetos de classe `matrix` e `data.frame`, porque é através de vetores lógicos ou de matrizes lógicas que podemos filtrar dados de objetos bidimensionais.

```
?iris # veja o help do R sobre Edgar Anderson's Iris Data que explica esses dados que vem com
```

```
class(iris)
```

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

```
str(iris) # estrutura, veja as colunas
```

```
## '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 ...
```

```
# vamos filtrar os dados de uma das espécies  
unique(iris$Species) # vemos os valores únicos
```

```
## [1] setosa      versicolor virginica  
## Levels: setosa versicolor virginica
```

```
# ou, tendo em vista que é um fator  
levels(iris$Species)
```

```
## [1] "setosa"      "versicolor"  "virginica"
```

```
sp1 <- levels(iris$Species)[1]  
# quais linhas correspondem a essa espécie  
vl <- iris$Species == sp1  
sum(vl) # numero de linhas que satisfazem a pergunta
```

```
## [1] 50
```

```
nrow(iris) # numero total de linhas no data.frame
```

```
## [1] 150
```

```
# filtrando os dados eu simplesmente uso o vetor lógico como índice de linha. O novo objeto criado é só uma matriz com linhas filtradas
iris.sp1 <- iris[vl, ]
nrow(iris.sp1) == sum(vl) # então esta condição é verdadeira
```

```
## [1] TRUE
```

```
# filtrar segundo duas colunas
vl <- iris$Species == sp1 # seja da especie em sp1
sum(vl) # quantas são?
```

```
## [1] 50
```

```
vl2 <- iris$Sepal.Length <= 5 # tenha sepala menor ou igual a 5
sum(vl2) # quantas são?
```

```
## [1] 32
```

```
# combinando as duas perguntas
vll <- vl & vl2
sum(vll) # quantas são?
```

```
## [1] 28
```

```
# filtrando
ff <- iris[vll, ]
class(ff) # novo data.frame resultando do filtro realizado
```

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

```
nrow(ff) == sum(vll) # isso deve ser verdadeiro  
## [1] TRUE
```

3.3.1.1 Dados com valores ausentes

As funções `is.na()` e `na.omit()` vistas anteriormente (Seção 2.6.3) permitem eliminar linhas e colunas que tenham valores ausentes. A presença de valores às vezes impede certas análises de serem executadas.

```
# vamos fazer uma cópia do objeto iris e modificar ele acrescentando alguns NAs  
dd <- iris  
# tem algum NA originalmente?  
sum(is.na(dd)) # não tem  
## [1] 0
```

```
# qual a dimensão?  
dim(dd)
```

```
## [1] 150    5
```

```
# pega 10 valores aleatórios entre 1:150 (linhas)  
v1 <- sample(1:nrow(dd), size = 10, replace = F)  
# nessas linhas acrescenta NAs na coluna 2  
dd[v1, 2] <- NA  
# pega outros 10 valores aleatórios entre 1:150 (linhas)  
v1 <- sample(1:nrow(dd), size = 10, replace = F)
```

```
# nessas linhas acrescenta NAs na coluna 3
dd[v1, 3] <- NA

# pronto agora temos um data.frame com NAs
sum(is.na(dd)) # tem 20 NAs na tabela
```

```
## [1] 20
```

```
# quais linhas tem NA
vl <- is.na(dd[, 2]) | is.na(dd[, 3]) # ou é NA em 2 ou em 3 que foi onde mudei
dd[vl, ]
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
3	4.7	NA	1.3	0.2	setosa
15	5.8	NA	1.2	0.2	setosa
16	5.7	4.4	NA	0.4	setosa
17	5.4	NA	1.3	0.4	setosa
18	5.1	NA	1.4	0.3	setosa
25	4.8	NA	1.9	0.2	setosa
33	5.2	4.1	NA	0.1	setosa
44	5.0	3.5	NA	0.6	setosa
60	5.2	2.7	NA	1.4	versicolor
76	6.6	3.0	NA	1.4	versicolor
90	5.5	NA	4.0	1.3	versicolor
91	5.5	NA	4.4	1.2	versicolor
93	5.8	2.6	NA	1.2	versicolor
95	5.6	NA	4.2	1.3	versicolor
98	6.2	2.9	NA	1.3	versicolor
126	7.2	3.2	NA	1.8	virginica
142	6.9	NA	5.1	2.3	virginica
143	5.8	2.7	NA	1.9	virginica
149	6.2	3.4	NA	2.3	virginica
150	5.9	NA	5.1	1.8	virginica

```
# use na.omit() para eliminar todas as linhas que NA em alguma coluna
sum(is.na(dd)) # tem 20 valores
```

```
## [1] 20
```

```
dd2 <- na.omit(dd)
sum(is.na(dd2)) # nao tem mais nenhum
```

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

3.3.2 Ordenação de dados

Para ordenar matrizes e `data.frames`, é preciso entender a diferença entre duas funções:

- `sort()` ordena um vetor e retorna os valores ordenados;
- `order()` ordena um vetor e retorna os **índices dos valores ordenados**. É isso que deve ser utilizado para ordenar matrizes e `data.frames`.

```
?sort
?order
```

```
# ordenação
str(iris)
```

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

```
# função SORT()
o1 <- sort(iris$Sepal.Length) # pega os valores ordenados da coluna comprimento de sépala
o1 # sao valores de sepalias do menor para o maior
```

```
## [1] 4.3 4.4 4.4 4.4 4.5 4.6 4.6 4.6 4.6 4.7 4.7 4.8 4.8 4.8 4.8 4.8 4.9 4.9
## [19] 4.9 4.9 4.9 4.9 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.1 5.1 5.1 5.1
## [37] 5.1 5.1 5.1 5.1 5.2 5.2 5.2 5.2 5.3 5.4 5.4 5.4 5.4 5.4 5.4 5.5 5.5
## [55] 5.5 5.5 5.5 5.5 5.5 5.6 5.6 5.6 5.6 5.6 5.6 5.7 5.7 5.7 5.7 5.7 5.7 5.7
## [73] 5.7 5.8 5.8 5.8 5.8 5.8 5.8 5.9 5.9 5.9 5.9 6.0 6.0 6.0 6.0 6.0 6.0 6.1
## [91] 6.1 6.1 6.1 6.1 6.1 6.2 6.2 6.2 6.2 6.3 6.3 6.3 6.3 6.3 6.3 6.3 6.3 6.3
## [109] 6.4 6.4 6.4 6.4 6.4 6.4 6.4 6.5 6.5 6.5 6.5 6.6 6.6 6.7 6.7 6.7 6.7 6.7
## [127] 6.7 6.7 6.7 6.7 6.8 6.8 6.8 6.9 6.9 6.9 6.9 7.0 7.1 7.2 7.2 7.2 7.3 7.4
## [145] 7.6 7.7 7.7 7.7 7.7 7.7 7.9
```

```
# em ordem decrescente
o2 <- sort(iris$Sepal.Length, decreasing = T)
o2 # sao valores de sepalias do maior para o menor
```

```
## [1] 7.9 7.7 7.7 7.7 7.7 7.6 7.4 7.3 7.2 7.2 7.2 7.1 7.0 6.9 6.9 6.9 6.9 6.8
## [19] 6.8 6.8 6.7 6.7 6.7 6.7 6.7 6.7 6.7 6.7 6.7 6.6 6.6 6.5 6.5 6.5 6.5 6.4
## [37] 6.4 6.4 6.4 6.4 6.4 6.4 6.3 6.3 6.3 6.3 6.3 6.3 6.3 6.3 6.3 6.2 6.2 6.2
## [55] 6.2 6.1 6.1 6.1 6.1 6.1 6.0 6.0 6.0 6.0 6.0 6.0 6.0 6.0 5.9 5.9 5.9 5.8 5.8
## [73] 5.8 5.8 5.8 5.8 5.8 5.7 5.7 5.7 5.7 5.7 5.7 5.7 5.7 5.6 5.6 5.6 5.6 5.6 5.6
## [91] 5.6 5.5 5.5 5.5 5.5 5.5 5.5 5.5 5.4 5.4 5.4 5.4 5.4 5.4 5.4 5.3 5.2 5.2 5.2
## [109] 5.2 5.1 5.1 5.1 5.1 5.1 5.1 5.1 5.1 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.0
## [127] 5.0 5.0 4.9 4.9 4.9 4.9 4.9 4.9 4.8 4.8 4.8 4.8 4.8 4.7 4.7 4.6 4.6 4.6 4.6
## [145] 4.6 4.5 4.4 4.4 4.4 4.3
```

```
# FUNCAO order()
# qual o indice dos valores ordenados em ordem crescente?
o3 <- order(iris$Sepal.Length)
o3 # esses valores correspondem aos INDICES dos valores ordenados

## [1] 14 9 39 43 42 4 7 23 48 3 30 12 13 25 31 46 2 10
## [19] 35 38 58 107 5 8 26 27 36 41 44 50 61 94 1 18 20 22
## [37] 24 40 45 47 99 28 29 33 60 49 6 11 17 21 32 85 34 37
## [55] 54 81 82 90 91 65 67 70 89 95 122 16 19 56 80 96 97 100
## [73] 114 15 68 83 93 102 115 143 62 71 150 63 79 84 86 120 139 64
## [91] 72 74 92 128 135 69 98 127 149 57 73 88 101 104 124 134 137 147
## [109] 52 75 112 116 129 133 138 55 105 111 117 148 59 76 66 78 87 109
## [127] 125 141 145 146 77 113 144 53 121 140 142 51 103 110 126 130 108 131
## [145] 106 118 119 123 136 132
```

```
# entao para ver os valores ordenados
iris$Sepal.Length[o3]
```

```
## [1] 4.3 4.4 4.4 4.4 4.5 4.6 4.6 4.6 4.6 4.7 4.7 4.8 4.8 4.8 4.8 4.8 4.9 4.9
## [19] 4.9 4.9 4.9 4.9 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.0 5.1 5.1 5.1
## [37] 5.1 5.1 5.1 5.1 5.1 5.2 5.2 5.2 5.2 5.3 5.4 5.4 5.4 5.4 5.4 5.4 5.5 5.5
## [55] 5.5 5.5 5.5 5.5 5.5 5.6 5.6 5.6 5.6 5.6 5.6 5.7 5.7 5.7 5.7 5.7 5.7 5.7
## [73] 5.7 5.8 5.8 5.8 5.8 5.8 5.8 5.9 5.9 5.9 6.0 6.0 6.0 6.0 6.0 6.0 6.1
## [91] 6.1 6.1 6.1 6.1 6.1 6.2 6.2 6.2 6.2 6.3 6.3 6.3 6.3 6.3 6.3 6.3 6.3 6.3
## [109] 6.4 6.4 6.4 6.4 6.4 6.4 6.4 6.4 6.5 6.5 6.5 6.5 6.6 6.6 6.7 6.7 6.7 6.7
## [127] 6.7 6.7 6.7 6.7 6.8 6.8 6.8 6.9 6.9 6.9 6.9 7.0 7.1 7.2 7.2 7.2 7.3 7.4
## [145] 7.6 7.7 7.7 7.7 7.7 7.7 7.9
```

```
# entao isso deve ser totalmente verdadeiro:
iris$Sepal.Length[o3] == sort(iris$Sepal.Length) # as comparacoes para a par sao identicas
```

```
## [1] TRUE TRUE
## [16] TRUE TRUE
## [31] TRUE TRUE
## [46] TRUE TRUE
## [61] TRUE TRUE
## [76] TRUE TRUE
## [91] TRUE TRUE
## [106] TRUE TRUE
## [121] TRUE TRUE
## [136] TRUE TRUE
```

então esta expressão também é verdadeira:

```
sum(iris$Sepal.Length[03] == sort(iris$Sepal.Length)) == nrow(iris)
```

```
## [1] TRUE
```

portanto a função sort ordena os valores e função order mostra apenas os índices dos valores

idx <- order(iris\$Sepal.Length) # índice das linhas ordenadas segundo o comprimento das sepals

compara com o original:

```
sum(iris$Sepal.Length[idx] == iris$Sepal.Length) == nrow(iris) # é FALSO porque em iris as linhas
```

```
## [1] FALSE
```

vamos mudar isso

novo.iris <- iris[idx,] # pego o vetor de índices dos valores ordenados e uso na indexação

idx2 <- order(novo.iris\$Sepal.Length) # índice das linhas ordenadas segundo o comprimento das

```
# note que agora essa expressão é verdadeira, porque o original já está ordenado segundo essa
sum(novo.iris$Sepal.Length[idx2] == novo.iris$Sepal.Length) == nrow(novo.iris) # é VERDADEIRO
```

```
## [1] TRUE
```

```
# AGORA ORDENANDO POR MULTIPHAS COLUNAS
```

```
idx <- order(iris$Species, iris$Sepal.Length, decreasing = TRUE) # por especie e por sepala em
```

```
# ordena segundo essas duas colunas
```

```
novo.iris <- iris[idx, ]
```

```
novo.iris[, c("Species", "Sepal.Length")]
```

	Species	Sepal.Length
132	virginica	7.9
118	virginica	7.7
119	virginica	7.7
123	virginica	7.7
136	virginica	7.7
106	virginica	7.6
131	virginica	7.4
108	virginica	7.3
110	virginica	7.2
126	virginica	7.2
130	virginica	7.2
103	virginica	7.1
121	virginica	6.9
140	virginica	6.9
142	virginica	6.9
113	virginica	6.8
144	virginica	6.8
109	virginica	6.7
125	virginica	6.7
141	virginica	6.7
145	virginica	6.7
146	virginica	6.7
105	virginica	6.5
111	virginica	6.5
117	virginica	6.5
148	virginica	6.5
112	virginica	6.4
116	virginica	6.4
129	virginica	6.4
133	virginica	6.4
138	virginica	6.4
101	virginica	6.3
104	virginica	6.3
124	virginica	6.3
134	virginica	6.3
137	virginica	6.3
147	virginica	6.3
127	virginica	6.2
149	virginica	6.2
128	virginica	6.1
135	virginica	6.1
120	virginica	6.0
139	virginica	6.0

```
# para cada especie esta ordenado por sepala:  
novo.iris[novo.iris$Species == "versicolor", ]$Sepal.Length  
  
## [1] 7.0 6.9 6.8 6.7 6.7 6.6 6.6 6.5 6.4 6.4 6.3 6.3 6.3 6.2 6.2 6.1 6.1 6.1  
## [20] 6.1 6.0 6.0 6.0 5.9 5.9 5.8 5.8 5.8 5.7 5.7 5.7 5.7 5.7 5.6 5.6 5.6 5.6  
## [39] 5.6 5.5 5.5 5.5 5.5 5.4 5.2 5.1 5.0 5.0 4.9  
  
novo.iris[novo.iris$Species == "virginica", ]$Sepal.Length  
  
## [1] 7.9 7.7 7.7 7.7 7.7 7.6 7.4 7.3 7.2 7.2 7.2 7.1 6.9 6.9 6.9 6.8 6.8 6.7 6.7  
## [20] 6.7 6.7 6.7 6.5 6.5 6.5 6.5 6.4 6.4 6.4 6.4 6.3 6.3 6.3 6.3 6.3 6.3 6.2  
## [39] 6.2 6.1 6.1 6.0 6.0 5.9 5.8 5.8 5.8 5.7 5.6 4.9
```

3.4 Importando e exportando dados no R



ATENÇÃO! Se você utiliza Windows, e no seu gerenciador de arquivos os arquivos aparecem sem **extensão** (.csv, .txt, .doc etc.), mude nas suas preferências para não **ocultar extensões de arquivos conhecidos**. Dessa forma você consegue ver os arquivos pelo tipo (extensão).

Existem diversas funções para importar dados para objetos do R, incluindo funções para ler arquivos do Excel (.xls, ou .xlsx), arquivos XML, arquivos *.DBF etc. O R também tem pacotes que interagem diretamente com bancos de dados (mysql, postgres, sql etc.). Não cobriremos a importação desses tipos aqui, mas você pode pesquisar sozinho no rede.

É frequente encontrarmos problemas de acentuação e na transferibilidade entre sistemas operacionais diferentes (Mac, Linux, Windows). A palavra chave aqui é codificação de caracteres¹ (em inglês, “character encoding”).

3.4.1 Arquivos de texto simples para estocar dados

Muitos dados que obtemos online e os próprios scripts do R são do formato mais simples que existe, que são arquivos de texto, geralmente arquivos salvos com extensões .csv ou .txt. Arquivos desse tipo podem ser abertos em qualquer editor de texto, em qualquer sistema operacional e em qualquer versão. Isso garante arquivamento, longevidade e transferibilidade. Portanto, é a melhor forma de salvar seus dados e compartilhá-los.

Qualquer arquivo desse tipo pode ser lido pelos editores de script do R ou RStudio. Pode também exportar planilhas do Excel ou LibreOffice (e afins) para esse formato. Vamos nos concentrar neste curso em lidar com arquivos deste tipo.

É importante atentar em arquivos de texto contendo dados tabulados para:

- O **separador** das colunas pode ser ;, tabulação (no R = ”), , ou qualquer símbolo que indique a separação das colunas (ou seja não está nas células);
- Casas decimais podem ser separadas por . ou ,.



DICA: Procure saber como seus dados estão antes de tentar importá-los, de forma a indicar corretamente o delimitador e o separador das casas decimais adequados. Você evitará assim muita dor de cabeça!

¹https://en.wikipedia.org/wiki/Character_encoding

- Datas - colunas com datas constituem um objeto de classe `date` no R, que a converte em número que pode ser usado em operações matemáticas. Dependendo de como seus dados estão formatados no original, é comum a inversão de mês com dia entre, por exemplo, o sistema inglês (MM-DD-YYYY) e o sistema português (DD-MM-YYYY). Tenha controle disso!

Por isso, recomenda-se que:

- Defina um padrão que você sempre usará para formatar seus dados ANTES de importá-los ao R. Dessa forma você irá memorizar rapidamente como importar os dados do jeito que você sempre prepara;
- Padronize a codificação dos caracteres (UTF8 é padrão Mac e Linux; Latin1 é padrão Windows) em arquivos `.txt`;
- Padronize o separador de casa decimal (ponto ou vírgula?);
- Padronize a quebra de linha, i.e., o que indica no texto o início de uma nova linha (novamente, isso é diferente entre Mac, Linux e Windows);
- Padronize se colunas de texto vão entre aspas;
- Padronize como você dá nome às colunas; nome de colunas e de linhas não devem ser muito longos, e deve-se **evitar acentos ou espaços em branco em nomes de colunas**. Isso é muito importante!
- Se você usa planilhas, recomendamos usar uma versão de software livre da família LibreOffice/OpenOffice pois eles permitem um maior controle da exportação dos dados, o que inclui controlar o tipo de codificação de caracter dos dados de saída e também separadores das colunas, tanto para ler como para salvar arquivos de planilhas.

3.4.2 Importando dados

3.4.2.1 Pacote base do R

A principal função para importar dados no R é `read.table()`. Ela funciona para importar arquivos em formato de texto simples (`.csv`, `.txt`).

Vamos utilizar um conjunto de dados contendo as coordenadas geográficas dos municípios brasileiros² para praticar a importação dos dados. Baixe-o para a sua pasta de trabalho.

Em seguida, abra o arquivo com um editor de texto simples (Bloco de Notas, Notepad++, TextWrangler, gedit, etc.) e veja como ele está formatado. Verifique:

- Qual é o separador de colunas?;
- Qual é a codificação dos caracteres? (consegue ver e editar isso no seu editor?);
- Qual é a quebra de linha? (consegue ver e editar isso no seu editor?)
- Aspas duplas ou simples definem colunas? (este arquivo não tem nenhuma aspas!)

Vamos agora abrir este arquivo no LibreOffice (ou similar, como o Excel). Busque os comandos de importação para poder importar o arquivo de texto³. Veja os controles na importação quanto aos elementos acima:

- Salve o arquivo como `*.ods`;
- Salve novamente como `*.csv` - veja como você tem controle na exportação quanto aos elementos acima.

²<https://github.com/LABOTAM/IntroR/blob/main/dados/municipiosbrasil.csv>

³Isso é mais fácil de ser feito no LibreOffice/OpenOffice

```
# se você colocou o arquivo na sua pasta de trabalho, ele deve estar visível por  
dir(pattern = "csv")
```

```
# então posso ler sem precisar especificar o caminho até o arquivo  
# veja o help da função antes de começar  
?read.table  
# os seguintes argumentos são mais importantes:  
# sep = " " #o código que separa as colunas, o padrão é espaço  
# quote = "\"" #o que define células de texto - o padrão é interpretar tanto aspa simples como dupla  
# dec = "." #ponto é a casa decimal padrão  
# header = FALSE #a primeira linha não tem o nome de colunas  
# as.is = FALSE #o padrão é converter texto em fatores, se usar T não fará isso  
# na.strings #se definir, pode informar aqui que símbolos em células inteiras que sejam interpretados como strings  
# encoding #codificação da acentuação. o padrão é 'unknown' (desconhecido), na qual ele reconhece
```

```
# o arquivo original tem os seguintes formatos:  
# colunas separadas por tabulação (no R isso é definido pela expressão regular "\t")  
# decimal com ponto  
# não tem aspas definindo as colunas de texto.  
# a primeira linha é o nome das colunas.  
# Então, para ler posso usar:  
dd <- read.table(file = "municipiosbrasil.csv", sep = "\t", header = T)  
class(dd) # data.frame  
dim(dd) # dimensão do objeto  
head(dd) # cabeçalho do data.frame  
  
# veja o que aconteceria se eu achasse que no meu arquivo as colunas são separadas por vírgula  
dd2 <- read.table(file = "municipiosbrasil.csv", sep = ",", header = T)  
head(dd2)  
dim(dd2) # apenas 1 coluna, porque o separador informado não é o mesmo dos dados  
  
# e se o encoding do meu arquivo estiver errado?
```

```
dd3 <- read.table(file = "municipiosbrasil.csv", sep = "\t", header = T, encoding = "latin1")
dd3[5562, ] # veja o que aconteceu com os acentos nessa linha
dd[5562, ] # no original o encoding não é "latin1"

# veja a estrutura do objeto correto
str(dd)
# Poxa, todas as colunas são fatores, mesmo as colunas Latitude e Longitude que são numéricas.
# Deve ter algum valor nessas colunas que não são numéricos.
# Quais são?
vl <- is.na(as.numeric(as.vector(dd$Latitude))) # quais são NA quando eu converto para numérico
sum(vl)
dd[vl, ] # essas linhas tem a palavra "NULL" para Latitude e Longitude no arquivo original (v)
```

Podemos usar o argumento `na.strings` para corrigir isso durante a importação:

```
dd4 <- read.table(file = "municipiosbrasil.csv", sep = "\t", header = T, dec = ".", na.strings = "NULL")
# qualquer CELULA INTEIRA que contenha NULL ou NA ou esteja vazia SERÁ INTERPRETADA COMO VALOR NA
str(dd4)
# note que agora as colunas Latitude e Longitude foram interpretadas como número

# mas o que acontece se informamos mal a casa decimal?
dd5 <- read.table(file = "municipiosbrasil.csv", sep = "\t", header = T, dec = ",", na.strings = "NULL")
str(dd5)
# como tem ponto como definição de casa decimal no arquivo de dados, as colunas numéricas foram interpretadas como strings

# Texto como vetores ou fatores?
dd4 <- read.table(file = "municipiosbrasil.csv", sep = "\t", header = T, dec = ".", na.strings = "NULL")
str(dd4) # todas as colunas de texto neste objeto foram interpretadas como fatores
# o argumento as.is permite corrigir isso. "as is" significa "como está" nos dados originais,
dd6 <- read.table(
  file = "municipiosbrasil.csv",
  sep = "\t",
  header = T,
  dec = ".")
```

```
na.strings = c("NULL", "NA", ""),
as.is = TRUE
)
str(dd6) # diferentemente do objeto anterior, não há mais fatores

# Lembram que colocamos o endereço do arquivo mais acima?
# podemos usar um endereço da internet para baixar um arquivo
dd7 <-
read.table(
  "https://github.com/LABOTAM/IntroR/blob/main/dados/municipiosbrasil.csv",
  sep = "\t",
  header = TRUE
)
dd7
head(dd7)
# a função read.table tem vários outros argumentos. Veja o help e entenda isso bem.
```

3.4.2.2 Pacote `readr`

A principal função para ler arquivos do pacote `readr` (Wickham and Hester, 2020) se chama `read_delim()`. Funciona de maneira parecida com o `read.table()` com algumas pequenas diferenças: não converte colunas de texto que possam ser categorizadas em fatores (`read.table()` faz isso por padrão), retorna um `tibble` no lugar de um `data.frame` (`tibbles` são `data.frames` diferentes na maneira como aparecem no console; além de mostrar apenas uma porção dos dados, para cada coluna há a indicação do tipo de variável presente), assume por padrão que o dado importado possui cabeçalho. Existem outras diferenças que podem ser melhor entendidas na página do pacote (<https://github.com/tidyverse/readr>). Os argumentos possuem nomes diferentes do que os utilizados em `read.table()` e, como este, importa arquivos em formato de texto simples (`.csv`, `.txt`).

```
# pacote readr  
# usando como exemplo o mesmo arquivo municipiosbrasil.csv  
library("readr")  
rr1 <- read_delim("municipiosbrasil.csv", delim = "\t")  
rr1  
dd7  
dim(rr1)  
dim(dd7)
```

Como no pacote base, também podemos ler arquivos diretamente da rede:

```
rr2 <- read_delim("https://github.com/LABOTAM/IntroR/blob/main/dados/municipiosbrasil.csv", de  
rr2
```

3.4.2.3 Pacote data.table

A principal função para ler arquivos do pacote `data.table` ([Dowle and Srinivasan, 2020](#)) se chama `fread()`. Este pacote é muito conhecido devido à velocidade de suas ações, funcionando perfeitamente para dados grandes. Esta função possui uma particularidade: o usuário não precisa indicar o separador; automaticamente ele descobre o separador e lê o arquivo. Em casos especiais, é necessário a indicação do separador com o argumento `sep`, igual ao `read.table()`. Ao ler um arquivo, a função retorna também um `data.frame`, porém com certas particularidades quanto à impressão do resultado na tela do console, como acontece com a função `read_delim()` do pacote `readr`. Mais informações, leiam atentamente o site do pacote: <https://github.com/Rdatatable/data.table>. O pacote como um todo é uma excelente ferramenta na manipulação de dados. Como os pacotes citados acima, esta função é capaz de importar arquivos em formato de texto simples (`.csv`, `.txt`).

```
# pacote data.table
# usando como exemplo o mesmo arquivo municipiosbrasil.csv
library("data.table")
dt1 <- fread("municipiosbrasil.csv")
dt1
```

Também podemos ler arquivos diretamente da rede, providenciando um endereço que contenha um arquivo de texto simples:

```
dt2 <- fread("https://github.com/LABOTAM/IntroR/blob/main/dados/municipiosbrasil.csv")
dt2
```

3.4.2.4 Importando do Excel diretamente

Utilizamos o pacote `readxl`([Wickham and Bryan, 2019](#)) para ler dados de arquivos Excel, isto é, arquivos `.xlsx` ou `.xls`. A principal função para importar dados deste pacote se chama `read_excel()`. Os principais erros aqui podem ser por células unidas, cabeçalhos no topo da planilha, e acentos. Veja o `?readxl` das funções usadas para conhecer parâmetros opcionais para resolver esses possíveis problemas.

```
# instale o pacote
library("readxl")

# se o arquivo for xls
# Salve o arquivo municipiosbrasil.csv como xlsx ou xls
meuxlsx <- "municipiosbrasil.xlsx"
dd <- read_excel(path = meuxlsx, sheet = 1)
dd
dd <- as.data.frame(dd)
dd

# se o arquivo for xlsx
```

```
meuxls <- "municipiosbrasil.xls"
dd <- read_excel(path = meuxls, sheet = 1)
dd
dd <- as.data.frame(dd)
dd
```

3.4.3 Exportando dados

3.4.3.1 Pacote base do R

A principal função do pacote `base` do R para exportar dados se chama `write.table()`. Ela funciona para exportar arquivos em formato de texto simples (.csv, .txt) e usa basicamente os mesmos argumentos da função `read.table()`.

```
?write.table # veja o help - recomendo usar essa função genérica e evitar de usar atalhos tipo
```

Por se tratar de uma função do pacote `base`, não é necessário recorrer a função `library()` para chamar nenhum pacote, pois a função encontra-se disponível a qualquer momento para ser utilizada no R:

```
# vamos usar o mesmo arquivo
dir(pattern = "csv")
# ler o arquivo para o R para ter algo a exportar
dd <- read.table(file = "municipiosbrasil.csv", sep = "\t", header = T, dec = ".", na.strings = "")
str(dd) # diferentemente do objeto anterior, não há mais fatores

# filtrando apenas para municípios do Amazonas:
vl <- dd$Province %in% "Amazonas"
sum(vl) # quantos são?
# ou, desse jeito que é identico:
vl <- dd$Province == "Amazonas"
sum(vl)
```

```
dd.am <- dd[vl, ]  
nrow(dd.am) == sum(vl) # deve ser verdadeiro, certo?  
  
# salvando esses dados num novo arquivo com diferentes formatações:  
# separado por tabulação e textos sem aspas e células NA sem nada  
write.table(dd.am, file = "muni-am1.csv", sep = "\t", na = "", quote = FALSE)  
# separado por tabulação e textos com aspas e células NA sem nada  
write.table(dd.am, file = "muni-am2.csv", sep = "\t", na = "", quote = TRUE)  
  
# separado por vírgula e textos com aspas e células NA com a palavra valor.ausente  
write.table(dd.am, file = "muni-am3.csv", sep = ",", na = "valor.ausente", quote = TRUE)  
  
# separado por vírgula e textos com aspas e células NA vazios e não adiciona nomes das linhas  
write.table(dd.am, file = "muni-am4.csv", sep = ",", na = "", quote = TRUE, row.names = FALSE)  
  
# separado por tabulação e textos sem aspas e células NA vazias, sem nomes das linhas, e quebra  
write.table(dd.am, file = "muni-am5.csv", sep = ",", na = "", quote = TRUE, row.names = FALSE)  
  
# ABRA OS ARQUIVOS GERADOS NO SEU EDITOR DE TEXTO E COMPARE AS FORMATAÇÕES GERADAS
```

3.4.3.2 Pacote `readr`

A principal função do pacote `readr` para exportar dados se chama `write_delim()`. Ela exporta `data.frames` em formato de texto simples (`.csv`, `.txt`), utilizando basicamente os mesmos argumentos da função `read_delim()`, pertencente ao mesmo pacote.

```
# exportando dados com pacote readr  
# utilizando mesmo objeto criado com pacote base  
write_delim(dd.am, "muni-am6.csv", delim = "\t")  
write_delim(dd.am, "muni-am7.csv", delim = ";")
```

3.4.3.3 Pacote `data.table`

A principal função do pacote `data.table` para exportar `data.frames` em formato de texto simples (`.csv`, `.txt`) se chama `fwrite()` e usa basicamente os mesmos argumentos da função `read_delim()`, pertencente ao pacote `readr`.

```
# exportando dados com pacote data.table
# utilizando mesmo objeto criado com pacote base
fwrite(dd.am, "muni-am8.csv", sep = "\t")
```

3.4.4 Outras funções úteis

A função `scan()` lê um arquivo de texto em qualquer formato para um vetor ou lista no R. Trata-se de uma função genérica que é bom memorizar. Vamos usar o mesmo arquivo `municipiosbrasil.csv` para demonstrar sua utilidade:

```
# esta função é muito útil para ler linha por linha um arquivo de texto que você quer explorar
dd <- scan(file = "municipiosbrasil.csv", what = "complex", sep = "\n")
```

```
class(dd)
```

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

```
length(dd) # cada linha é um elemento do vetor
```

```
## [1] 5568
```

```
dd[1]
```

```
## [1] "Country\tProvince\tRegiao\tMunicipio\tLatitude\tLongitude"
```

```
# usando tabulação
dd2 <- scan(file = "municipiosbrasil.csv", what = "complex", sep = "\t")
```

```
class(dd2)
```

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

```
length(dd2) # cada célula é um elemento deste vetor
```

```
## [1] 33408
```

```
dd2[1:5]
```

```
## [1] "Country"    "Province"   "Regiao"     "Municipio"  "Latitude"
```

```
# não faz muito sentido com esses dados que tem formato de tabela, mas essa função pode ser usada
# de um artigo no qual você quer buscar palavras e tabular palavras-chaves?
# num log de uma análise feita por outro software (não no R) do qual você quer extrair resultados
# etc.
```

3.5 Para saber mais:

- Vídeoaula de Indexação de matrizes e data.frames⁴.
 - Vídeoaula de filtragem e ordenação de matrizes⁵.
-

3.6 Exercícios

- Resolva o exercício 103.01 Distância entre cidades⁶.
- Resolva o exercício 103.02 Criação de um data frame⁷.
- Resolva o exercício 103.03 Criando uma Matriz⁸.

⁴<https://youtu.be/CJILnDzVviQ>

⁵http://www.botanicaamazonica.wiki.br/labotam/lib/exe/fetch.php?media=bot89:precurso:3matrizes:video01_bot89-2020-04-08_11.41.08.mp4

⁶<http://notar.ib.usp.br/exercicio/23>

⁷<http://notar.ib.usp.br/exercicio/2>

⁸<http://notar.ib.usp.br/exercicio/13>

4

Objeto III - Listas e objetos complexos

4.1 Listas

O resultado de muitas análises são objetos de classe `list` e você precisa entender o que isso significa. Listas permitem organizar diferentes classes de objetos numa estrutura hierárquica organizada. Uma mesma lista pode incluir elementos que são vetores de qualquer classe, matrizes, data.frames etc. Listas são criadas pela função `list()`. A indexação de listas é dado pelo operador `[[indice_ou_nome]]` ou `lista$seguida do nome` (se os elementos da lista tem nome; e.g. `lista$nomeDeUmElementoDaLista`).

```
?list # veja o help
```

```
# um vetor simples
v1 <- 1:10
class(v1)
```

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

```
# outro vetor simples
v2 <- LETTERS
class(v2)
```

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

```
# uma matriz simples
mm <- matrix(1:9, nrow = 3, ncol = 3)
class(mm)
```

```
## [1] "matrix" "array"
```

```
# um data.frame
dd <- iris
dd
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa
5.4	3.4	1.7	0.2	setosa
5.1	3.7	1.5	0.4	setosa
4.6	3.6	1.0	0.2	setosa
5.1	3.3	1.7	0.5	setosa
4.8	3.4	1.9	0.2	setosa
5.0	3.0	1.6	0.2	setosa
5.0	3.4	1.6	0.4	setosa
5.2	3.5	1.5	0.2	setosa
5.2	3.4	1.4	0.2	setosa
4.7	3.2	1.6	0.2	setosa
4.8	3.1	1.6	0.2	setosa
5.4	3.4	1.5	0.4	setosa
5.2	4.1	1.5	0.1	setosa
5.5	4.2	1.4	0.2	setosa
4.9	3.1	1.5	0.2	setosa
5.0	3.2	1.2	0.2	setosa
5.5	3.5	1.3	0.2	setosa
4.9	3.6	1.4	0.1	setosa
4.4	3.0	1.3	0.2	setosa
5.1	3.4	1.5	0.2	setosa
5.0	3.5	1.3	0.3	setosa
4.5	2.3	1.3	0.3	setosa
4.4	3.2	1.3	0.2	setosa

```
class(dd)
```

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

```
# criamos uma lista simples
```

```
v1
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
v2
```

```
## [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"  
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

```
mm
```

1	4	7
2	5	8
3	6	9

```
dd
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa
5.4	3.4	1.7	0.2	setosa
5.1	3.7	1.5	0.4	setosa
4.6	3.6	1.0	0.2	setosa
5.1	3.3	1.7	0.5	setosa
4.8	3.4	1.9	0.2	setosa
5.0	3.0	1.6	0.2	setosa
5.0	3.4	1.6	0.4	setosa
5.2	3.5	1.5	0.2	setosa
5.2	3.4	1.4	0.2	setosa
4.7	3.2	1.6	0.2	setosa
4.8	3.1	1.6	0.2	setosa
5.4	3.4	1.5	0.4	setosa
5.2	4.1	1.5	0.1	setosa
5.5	4.2	1.4	0.2	setosa
4.9	3.1	1.5	0.2	setosa
5.0	3.2	1.2	0.2	setosa
5.5	3.5	1.3	0.2	setosa
4.9	3.6	1.4	0.1	setosa
4.4	3.0	1.3	0.2	setosa
5.1	3.4	1.5	0.2	setosa
5.0	3.5	1.3	0.3	setosa
4.5	2.3	1.3	0.3	setosa
4.4	3.2	1.3	0.2	setosa

```
ml <- list(v1, v2, mm, dd)
class(ml) # deve ser lista
```

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

```
length(ml) # número de elementos
```

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

```
names(ml) # os elementos dessa lista não tem nome
```

```
## NULL
```

```
str(ml) # veja a estrutura do objeto
```

```
## List of 4
## $ : int [1:10] 1 2 3 4 5 6 7 8 9 10
## $ : chr [1:26] "A" "B" "C" "D" ...
## $ : int [1:3, 1:3] 1 2 3 4 5 6 7 8 9
## $ :'data.frame':   150 obs. of  5 variables:
##   ..$ Sepal.Length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##   ..$ Sepal.Width : num [1:150] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##   ..$ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##   ..$ Petal.Width : num [1:150] 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 ...
```

```
ml[[1]] # o elemento 1 é o vetor
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
ml[[2]] # o segundo também
```

```
## [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"  
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

```
ml[[1:2]] # note que retorna apenas o segundo... não funciona como vetor para pegar mais de um
```

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

```
ml[[1]][3] # terceiro elemento do vetor que está no elemento 1 da lista
```

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

```
ml[[3]] # é uma matrix
```

1	4	7
2	5	8
3	6	9

```
ml[[3]][1, 3] # valor da primeira linha e da terceira coluna da matrix que o objeto 2 da lista
```

```
## [1] 7
```

```
ml[[4]] # é uma data.frame
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa
5.4	3.4	1.7	0.2	setosa
5.1	3.7	1.5	0.4	setosa
4.6	3.6	1.0	0.2	setosa
5.1	3.3	1.7	0.5	setosa
4.8	3.4	1.9	0.2	setosa
5.0	3.0	1.6	0.2	setosa
5.0	3.4	1.6	0.4	setosa
5.2	3.5	1.5	0.2	setosa
5.2	3.4	1.4	0.2	setosa
4.7	3.2	1.6	0.2	setosa
4.8	3.1	1.6	0.2	setosa
5.4	3.4	1.5	0.4	setosa
5.2	4.1	1.5	0.1	setosa
5.5	4.2	1.4	0.2	setosa
4.9	3.1	1.5	0.2	setosa
5.0	3.2	1.2	0.2	setosa
5.5	3.5	1.3	0.2	setosa
4.9	3.6	1.4	0.1	setosa
4.4	3.0	1.3	0.2	setosa
5.1	3.4	1.5	0.2	setosa
5.0	3.5	1.3	0.3	setosa
4.5	2.3	1.3	0.3	setosa
4.4	3.2	1.3	0.2	setosa

```
ml[[4]][, 1] # a primeira coluna deste data.frame
```

```
## [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
## [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
## [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
## [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
## [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
## [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

```
ml[[4]]$Sepal.Length # mesma coisa
```

```
## [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
## [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
## [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
## [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
## [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
## [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

```
# uma lista pode conter listas
mll <- list(list(v1, v2), mm, dd) # o primeiro elemento virou uma lista com dois vetores
class(mll[[1]]) # é uma lista agora
```

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

```
mll[[1]][[1]] # a sublista 1 do elemento 1 da lista
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
mll[[1]][[2]] # a sublista 2 do elemento 1 da lista
```

```
## [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"  
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

```
# criamos uma lista com nomes  
ml <- list(VETOR1 = v1, MATRIZ = mm, TABELA = dd)  
class(ml)
```

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

```
str(ml)
```

```
## List of 3  
## $ VETOR1: int [1:10] 1 2 3 4 5 6 7 8 9 10  
## $ MATRIZ: int [1:3, 1:3] 1 2 3 4 5 6 7 8 9  
## $ TABELA:'data.frame': 150 obs. of 5 variables:  
##   ..$ Sepal.Length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...  
##   ..$ Sepal.Width : num [1:150] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...  
##   ..$ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...  
##   ..$ Petal.Width : num [1:150] 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 ...
```

```
names(ml) # nome dos elementos
```

```
## [1] "VETOR1" "MATRIZ" "TABELA"
```

```
ml[["VETOR1"]]
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
ml$VETOR1 # ou assim, da mesma forma que uma coluna de um data.frame
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
ml[["TABELA"]][, "Sepal.Length"] # coluna do data.frame em TABELA
```

```
## [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
## [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
## [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
## [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
## [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
## [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

```
ml[["TABELA"]]$Sepal.Length # idem
```

```
## [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
## [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
## [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
## [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
## [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
## [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

```
ml$TABELA$Sepal.Length # idem também
```

```
## [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
## [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
## [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
## [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
## [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
## [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

4.2 Objetos complexos

Em alguns casos como, por exemplo, em arquivos de dados espaciais (shapefiles), que apresentam estruturas complexas que incluem as especificações dos polígonos, pontos ou linhas, a projeção espacial e os dados associados, exige-se um objeto que possua uma estrutura de complexidade similar.

Nestes casos, é importante que você conheça o operador `@`, que permite extrair elementos desses objetos. Em alguns casos você

terá que usá-lo para entender o objeto ou para pegar elementos dos mesmos objetos.

A função `slotNames()` permite ver os elementos que podem ser extraídos com o operador `@`. Abaixo mostramos um exemplo através de um mapa dos municípios brasileiros. Para isso, vamos precisar baixar um arquivo (<https://github.com/LABOTAM/IntroR/blob/main/dados/municipiosshape.zip>). Baixe este arquivo comprimido que contem os arquivos de um único *shapefile*. Descomprima-o na sua pasta de trabalho e você perceberá que haverá vários arquivos associados. Para trabalhar com esses arquivos, vamos utilizar os pacotes `maps` (Brownrigg, 2018) e `rgdal` (Bivand et al., 2020).

```
# vamos precisar de dois pacotes para dados espaciais
# se não tiver instalado, instale com as dependencias
# install.packages(c("maps", "rgdal"), dependencies = TRUE)
library("maps")
library("rgdal")

# agora mudem o diretório para pasta que contem os arquivos shape
# lembre que voce pode tanto utilizar a função `setwd()` quanto clicar em Ferramentas/Tools
dir(pattern = "shp") # lista arquivos shape na pasta
# veja o help da função para ler shapefiles
?readOGR

# le o shape file com objeto espacial no R
mp <- readOGR(dsn = "MUNICIPIOS.shp", layer = "MUNICIPIOS", encoding = "UTF-8")
plot(mp) # veja o mapa

class(mp) # é um objeto de classe SpatialPolygonsDataFrame
str(mp) # a estrutura é complexa

# tem elementos definidos por $, que é interpretado diretamente, é correspondente a um data.frame
names(mp)
mp$NOME_MUNI
dim(mp)
```

```
# tem elementos definidos por @
# ver o help da função
# ?slot
# slotNames(mp) #lista quais são esses elementos

mp@data # é o mesmo data.frame que é automaticamente reconhecido na expressão acima
dim(mp) == dim(mp@data)
names(mp) == names(mp@data)

# área do mapa dos municípios (os limites em latitude e longitude da área)
mp@bbox
plot(mp@bbox)
# adicionamos o mapa mundi sobre isso
map(add = T)

# mp@polygons define cada polígono individualmente numa lista
class(mp@polygons)
mp@polygons[[1]] # um elemento qualquer
class(mp@polygons[[1]])
str(mp@polygons[[1]])

# veja que este objeto tem vários elementos definidos por @
# slotNames(mp@polygons[[1]]) #slots desse objeto
mp@polygons[[1]]@labpt # o centroid do polígono 1 que é o município de:
mp@data$NOME_MUNI[1] # Chuí

# qual elemento é manaus?
gp <- grep("Manaus", mp@data$NOME_MUNI)
# pega o polígono de manaus
manaus <- mp@polygons[[gp]]
class(manaus)
str(manaus)

# plota manaus
# dev.off() #fecha dispositivos gráficos podes precisar disso
```

```
map(xlim = mp@bbox["x", ], ylim = mp@bbox["y", ])
polygon(manaus@Polygons[[1]]@coords, col = "red")
# centroides do polígono de manaus
ctro <- manaus@Polygons[[1]]@labpt
ctro[2] <- ctro[2] + 1.5 # adiciona 1.5 graus na latitude para não plotar sobre o polígono do
# plota no nome
text(x = ctro[1], y = ctro[2], labels = "Manaus", cex = 0.8)
```

5

Funções gráficas

5.1 Dispositivos Gráficos

Existem dois principais tipos de **dispositivos (Devices) gráficos** no R que basicamente significam onde você imprimirá um gráfico ou figura. Isso pode ocorrer:

- Na tela do computador (monitor), ou seja, em janelas do R ou do RStudio onde você visualiza gráficos;
- Em um arquivo em formato .pdf, .jpeg, .tiff, .png, .eps etc.

5.1.1 Função `plot()`

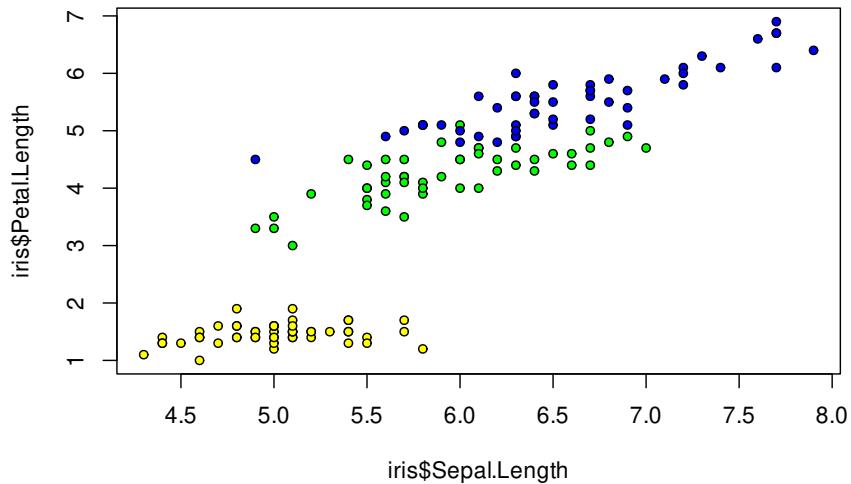
A função `plot()` (ou `plot.default()`) é a principal função genérica para gerar gráficos no R. Veremos isso com mais detalhes na seção **Funções gráficas de alto nível**. Aqui demonstraremos o uso da função simplificadamente para facilitar as demais explicações. Entenda que geramos gráficos com essa função. Vamos a um exemplo muito simples:

```
# o objeto R do iris como exemplo  
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:  
## $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...  
## $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
```

```
## $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
## $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
## $ Species     : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 ...
```

```
# plotando comprimento de sépala vs. comprimento de pétalas e colorindo os pontos de acordo com a espécie
vcl <- c("yellow", "green", "blue")[as.numeric(iris$Species)] # cria um vetor de cores para cada classe
# plota a figura
plot(iris$Sepal.Length, iris$Petal.Length, type = "p", pch = 21, bg = vcl, cex = 0.8)
```



Muitos argumentos da função `plot()` são parâmetros gráficos (Seções [5.2](#) e [5.3](#)).

5.1.2 Dispositivos de Tela

Quando você usa funções para gerar um gráfico, o R ou RStudio usa automaticamente um dispositivo de tela. No entanto, você pode abrir novas janelas com as seguintes funções:

- `x11()` ou `x11()` funcionam no Mac, Linux e Windows;

- `quartz()` funciona apenas no Mac;
- `windows()` funciona apenas no Windows.

```
?device # veja o help da função e as opções de devices
X11() # irá abrir uma janela nova

# vamos plotar o mesmo gráfico do exercício anterior
vcl <- c("yellow", "green", "blue")[as.numeric(iris$Species)] # cria um vetor de cores para cada classe
# plota a figura
plot(iris$Sepal.Length, iris$Petal.Length, type = "p", pch = 21, bg = vcl, cex = 0.8)

# irá fechar essa nova janela
dev.off()

quartz() # num mac abre uma janela sem dar nenhum aviso (funciona melhor que X11() no Mac)
vcl <- c("yellow", "green", "blue")[as.numeric(iris$Species)] # cria um vetor de cores para cada classe
# plota a figura
plot(iris$Sepal.Length, iris$Petal.Length, type = "p", pch = 21, bg = vcl, cex = 0.8)

dev.off() # fecha
```

5.1.3 Listar e controlar dispositivos

Existem funções que permitem trabalhar com vários dispositivos ao mesmo tempo, controlando o uso dos dispositivos abertos. Pode haver, por exemplo, várias janelas diferentes mostrando gráficos diferentes durante um trabalho. As principais funções para trabalhar com dispositivos são:

- A função `dev.list()` lista todos os dispositivos abertos no momento (geralmente o dispositivo padrão, que é sempre o número 1, é ignorado na lista), retornando o número (ordem de abertura) e o nome de cada um;

- `dev.cur()` mostra qual o dispositivo que está ativo (em inglês, *current*). Se há vários abertos, haverá sempre um que estará ativo naquele momento e se você enviar uma figura, ela sairá neste dispositivo;
- `dev.set()` torna ativo um determinado dispositivo;
- `dev.off()` fecha o dispositivo atual ou vários. Esta é a função dessa lista que é a mais usada na prática.

```
# vamos primeiro fechar todos os dispositivos
dev.off(which = dev.list())
dev.list() # vai retornar NULL por não há dispositivos abertos
# vamos abrir vários dispositivos
X11() # primeira janela extra
X11() # segunda janela e mesma figura com cores diferentes
X11() # terceira janela e mesma figura com cores diferentes

# coloque e redimensione as janelas para ter as tres visiveis na tua tela e volte aqui.

# veja os dispositivos abertos
dev.list()

# qual o atual
dev.cur() # o ultimo que abrimos, né

dev.set(2)
# vamos mudar para o segundo e plotar algo
plot(iris$Sepal.Length, iris$Petal.Length, type = "p", cex = 0.8)

# vamos mudar para o terceiro e plotar outra coisa
dev.set(3)
dev.cur() # deve responder 3
vcl <- c("yellow", "green", "blue")[as.numeric(iris$Species)]
plot(iris$Sepal.Length, iris$Petal.Length, type = "p", pch = 21, bg = vcl, cex = 0.8)
```

```
# vamos mudar para o quarto e plotar a mesma coisa com outras cores
dev.set(4)
dev.cur() # deve responder 4
vcl <- c("red", "blue", "white")[as.numeric(iris$Species)]
plot(iris$Sepal.Length, iris$Petal.Length, type = "p", pch = 21, bg = vcl, cex = 0.8)

# vamos fechar o dispositivo atual
dev.cur()
dev.off()
dev.cur() # mudou automaticamente porque voce fechou o 4
dev.off() # mesma coisa fechou o ativo
dev.cur() # sobrou o 3
# fechar este também
dev.off()
dev.list() # nao tem mais nenhum aberto
```

5.1.4 Dispositivos de arquivos

Há vários dispositivos para gerar arquivos com imagens. As funções em geral têm o nome do tipo de arquivo gerado. Vamos ver dois exemplos apenas, mas a mesma lógica se aplica a qualquer um dos dispositivos listados no `? devices` (execute o comando `help('device')` e veja a explicação sobre os dispositivos (*devices*)).

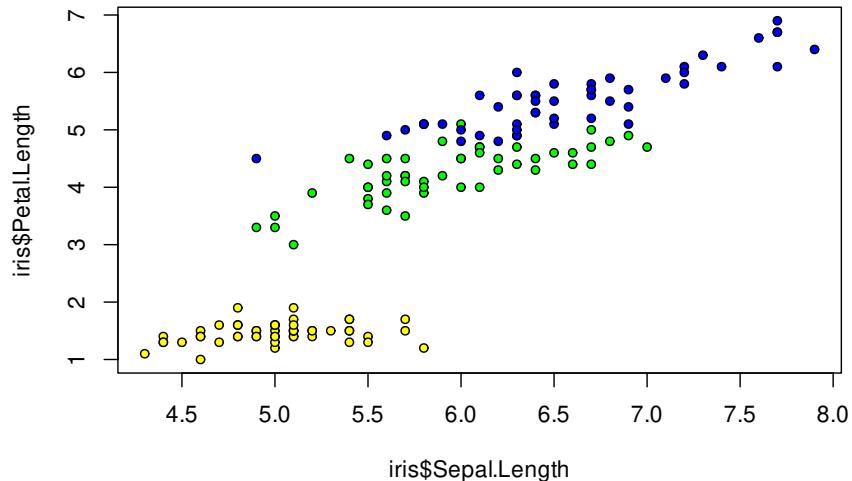
Essas funções são ótimas ferramentas para incluir nos seus scripts visando a produção de figuras para uma publicação. Procure na revista em que você deseja publicar seu artigo as especificações técnicas das figuras. Em seguida, cheque os parâmetros das funções gráficas (Seções 5.2 e 5.3) para você ser capaz de gerar figuras na especificação exata da revista selecionada.

5.1.5 Figuras vetoriais em pdf ou postscript

Em nossa opinião, as funções `pdf()` e `postscript()` são as mais importantes para a geração de figuras, porque elas geram arquivos de

excelente qualidade por serem vetoriais. Não há a necessidade de se definir a resolução, porque nesse tipo de imagem isso não existe.

```
# plotando a figura na tela é o que fazemos normalmente
vcl <- c("yellow", "green", "blue")[as.numeric(iris$Species)]
plot(iris$Sepal.Length, iris$Petal.Length, type = "p", pch = 21, bg = vcl, cex = 0.8)
```



```
# salvando a figura num PDF
?pdf # veja o help dessa função e seus argumentos, especialmente width e height
# abre o dispositivo para pdf
pdf(file = "meupdf.pdf", width = lcm(15), height = lcm(10))
# lcm() apenas pega valores em cm e converte em polegadas que é a especificação padrão da função
# plota a figura
plot(iris$Sepal.Length, iris$Petal.Length, type = "p", pch = 21, bg = vcl, cex = 0.8)
dev.off() # fecha o pdf
# só será possível ver o pdf se tiver fechado ele.
getwd() # o pdf foi gerado nessa pasta
```

```
# se voce incluir vários gráficos. O pdf irá gerar várias páginas ao mesmo tempo.
# abre o dispositivo para pdf
pdf(file = "meupdf2.pdf", paper = "a4")
# tamanho papel A4
# plota a figura 10 vezes
for (i in 1:10) {
  plot(iris$Sepal.Length, iris$Petal.Length, type = "p", pch = 21, bg = vcl, cex = 0.8)
}
dev.off() # fecha o pdf
# veja que o arquivo 2 tem várias páginas
```

5.1.6 Figuras raster

As funções `jpeg()`, `png()` e `tiff()` geram imagens em pixels, cuja qualidade depende muito da definição da resolução.

```
# formato jpeg sem controlar a resolucao (usando units='px' ou pixel)
# abre o dispositivo no formato desejado
jpeg(filename = "meujpeg.jpg", width = 600, height = 400, units = "px")

# plota alguma coisa
vcl <- c("yellow", "green", "blue")[as.numeric(iris$Species)]
plot(iris$Sepal.Length, iris$Petal.Length, type = "p", pch = 21, bg = vcl, cex = 0.8)

# fecha o jpeg
dev.off()

# ABRA O ARQUIVO E FAÇA UM ZOOM GRANDE
# note que os pixels são super visíveis

# AGORA MELHORANDO A RESOLUCAO
# formato jpeg com 300dpi
# abre o dispositivo no formato desejado
jpeg(filename = "meujpeg2.jpg", width = 15, height = 10, units = "cm", res = 300)
```

```
# plota alguma coisa
vcl <- c("yellow", "green", "blue")[as.numeric(iris$Species)]
plot(iris$Sepal.Length, iris$Petal.Length, type = "p", pch = 21, bg = vcl, cex = 0.8)

# fecha o jpeg
dev.off()
```



DICA: Abra os três arquivos e compare a resolução deles, ampliando a imagem até visualizar o pixel. Note que em um pdf você nunca verá o pixel. Por isso, recomendamos que você trabalhe sempre com a função `pdf()`.

5.2 Parâmetros gráficos, parte I - Margem, fonte, proporções

Para fazer boas figuras no R, você precisa muitas vezes controlar **parâmetros gráficos dos dispositivos**, como margem da figura, tamanho de fonte, tipo da fonte, distância das legendas dos eixos x e y, se essas legendas são horizontais ou verticais, o tipo símbolos dos pontos, as cores dos simbolos etc. Você pode definir esses parâmetros diretamente nas funções gráficas de alto-nível `highlevel plot` (... , `<tag> = <value>`), onde um valor (`value`) de um parâmetro (`tag`) pode ser adicionado diretamente como argumento de uma função de alto-nível.

Alguns desses parâmetros, no entanto, só podem ser definidos através da função `par()`, que também define **parâmetros gráficos globais**, isto é, se você alterar os parâmetros através dessa função, isso será alterado para todos os gráficos que forem abertos posteriormente durante a mesma sessão do R. Esta mesma função também permite visualizar os parâmetros gráficos.

Leia atentamente o ? da função `par()`¹. Você pode salvar os parâmetros globais antes de alterá-los.

```
?par # veja o help dessa função
```

```
# você pode ver os parâmetros padrão:  
par() # vejo todos os parâmetros
```

```
## $xlog  
## [1] FALSE  
##  
## $ylog  
## [1] FALSE  
##  
## $adj  
## [1] 0.5  
##  
## $ann  
## [1] TRUE  
##  
## $ask  
## [1] FALSE  
##  
## $bg  
## [1] "white"  
##  
## $bty  
## [1] "o"  
##  
## $cex  
## [1] 1  
##
```

¹<http://stat.ethz.ch/R-manual/R-devel/library/graphics/html/par.html>

```
## $cex.axis
## [1] 1
##
## $cex.lab
## [1] 1
##
## $cex.main
## [1] 1.2
##
## $cex.sub
## [1] 1
##
## $cin
## [1] 0.15 0.20
##
## $col
## [1] "black"
##
## $col.axis
## [1] "black"
##
## $col.lab
## [1] "black"
##
## $col.main
## [1] "black"
##
## $col.sub
## [1] "black"
##
## $cra
## [1] 10.8 14.4
##
## $crt
## [1] 0
##
## $csi
```

```
## [1] 0.2
##
## $cxy
## [1] 0.02851711 0.07518797
##
## $din
## [1] 6.5 4.5
##
## $err
## [1] 0
##
## $family
## [1] ""
##
## $fg
## [1] "black"
##
## $fig
## [1] 0 1 0 1
##
## $fin
## [1] 6.5 4.5
##
## $font
## [1] 1
##
## $font.axis
## [1] 1
##
## $font.lab
## [1] 1
##
## $font.main
## [1] 2
##
## $font.sub
## [1] 1
```

```
##  
## $lab  
## [1] 5 5 7  
##  
## $las  
## [1] 0  
##  
## $lend  
## [1] "round"  
##  
## $lheight  
## [1] 1  
##  
## $ljoin  
## [1] "round"  
##  
## $lmitre  
## [1] 10  
##  
## $lty  
## [1] "solid"  
##  
## $lwd  
## [1] 1  
##  
## $mai  
## [1] 1.02 0.82 0.82 0.42  
##  
## $mar  
## [1] 5.1 4.1 4.1 2.1  
##  
## $mex  
## [1] 1  
##  
## $mfcol  
## [1] 1 1  
##
```

```
## $mfg
## [1] 1 1 1 1
##
## $mfrow
## [1] 1 1
##
## $mgp
## [1] 3 1 0
##
## $mkh
## [1] 0.001
##
## $new
## [1] FALSE
##
## $oma
## [1] 0 0 0 0
##
## $omd
## [1] 0 1 0 1
##
## $omi
## [1] 0 0 0 0
##
## $page
## [1] TRUE
##
## $pch
## [1] 1
##
## $pin
## [1] 5.26 2.66
##
## $plt
## [1] 0.1261538 0.9353846 0.2266667 0.8177778
##
## $ps
```

```
## [1] 12
##
## $pty
## [1] "m"
##
## $smo
## [1] 1
##
## $srt
## [1] 0
##
## $tck
## [1] NA
##
## $tcl
## [1] -0.5
##
## $usr
## [1] 0 1 0 1
##
## $xaxp
## [1] 0 1 5
##
## $xaxs
## [1] "r"
##
## $xaxt
## [1] "s"
##
## $xpd
## [1] FALSE
##
## $yaxp
## [1] 0 1 5
##
## $yaxs
## [1] "r"
```

```
##  
## $yaxt  
## [1] "s"  
##  
## $ylbias  
## [1] 0.2
```

```
op <- par() # pego todos os parâmetros  
class(op) # isso é uma lista
```

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

```
names(op) # esses são os nomes dos parâmetros
```

```
## [1] "xlog"      "ylog"      "adj"       "ann"       "ask"       "bg"  
## [7] "bty"        "cex"        "cex.axis"   "cex.lab"    "cex.main"   "cex.sub"  
## [13] "cin"        "col"        "col.axis"   "col.lab"    "col.main"   "col.sub"  
## [19] "cra"        "crt"        "csi"        "cxy"        "din"        "err"  
## [25] "family"     "fg"         "fig"        "fin"        "font"       "font.axis"  
## [31] "font.lab"    "font.main"   "font.sub"    "lab"        "las"        "lend"  
## [37] "lheight"    "ljoin"      "lmitre"    "lty"        "lwd"        "mai"  
## [43] "mar"         "mex"        "mfcol"     "mfg"        "mfrow"     "mgp"  
## [49] "mkh"         "new"        "oma"       "omd"        "omi"       "page"  
## [55] "pch"         "pin"        "plt"       "ps"         "pty"       "smo"  
## [61] "srt"         "tck"        "tcl"       "usr"        "xaxp"      "xaxs"  
## [67] "xaxt"       "xpd"       "yaxp"      "yaxs"      "yaxt"      "ylbias"
```

```
# a função par permite ver os valores atualmente definidos  
par("family") # tipo de fonte não tem
```

```
## [1] ""
```

```
par("mar") # margens da figura em número de linhas
```

```
## [1] 5.1 4.1 4.1 2.1
```

```
# a função para tem apenas 1 argumento além dos parâmetros gráficos  
op2 <- par(no.readonly = TRUE)  
class(op2)
```

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

```
# a diferença entre especificar no.readonly como verdadeiro (na primeira opção não especificada) é que a lista gerada pode ser usada para refazer alterações, ou seja, para resgatar os parâmetros
```

```
# ou seja, posso fazer:  
par(op2) # para resgatar valores originais
```

```
# mas não posso fazer o mesmo com a primeira opção  
par(op) # porque op é uma lista diferente
```

```
length(op) == length(op2) # tem comprimentos diferentes
```

```
## [1] FALSE
```

```
identical(op, op2) # não são identicas
```

```
## [1] FALSE
```

5.2.1 Margem da figura

```
# Usando novamente o exemplo de iris
Sepalas <- iris$Sepal.Length
Petalas <- iris$Petal.Length

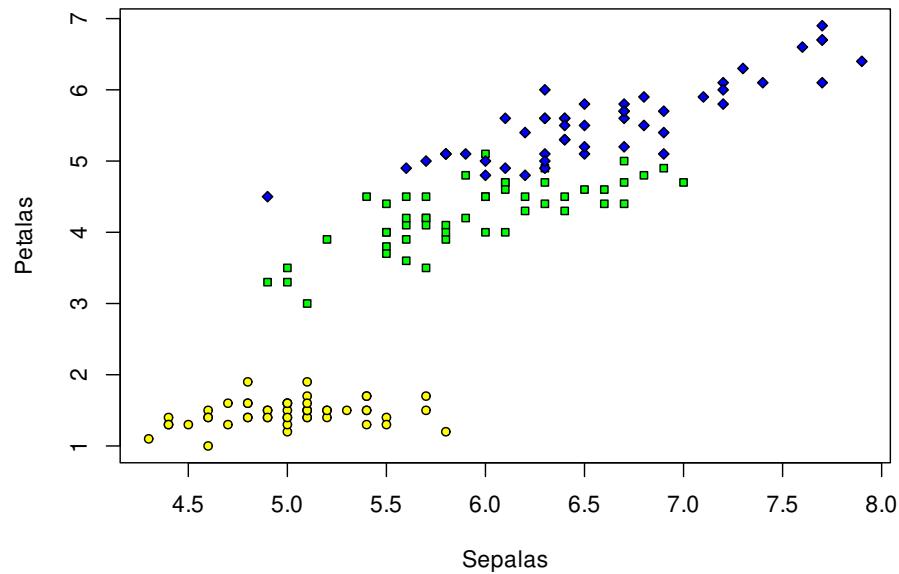
# especies como números
spp <- as.numeric(as.factor(iris$Species))
# uma cor para cada espécie
vcl <- c("yellow", "green", "blue")[spp]
# um simbolo para cada espécie
sbs <- c(21, 22, 23)[spp]

# plota a figura no dispositivo padrão
plot(Sepalas,
      Petalas,
      pch = sbs,
      bg = vcl,
      cex = 0.8
)

# ALTERANDO A MARGEM em NUMEROS DE LINHAS
# salva o valor padrao para resgatar ao final
op <- par(no.readonly = TRUE)
op$mar # esses são os valor atuais para Margem Inferior, Esquerda, Superior e Direita, respect
```

```
## [1] 5.1 4.1 4.1 2.1
```

```
# reduzindo as margem direita e superior em número de linhas
par(mar = c(5, 4, 1, 0.5))
plot(Sepalas,
     Petalas,
     pch = sbs,
     bg = vcl,
     cex = 0.8
)
```



```
# voltando ao original
par(mar = op$mar)
```

5.2.2 Aspecto dos eixos

```
op <- par(no.readonly = TRUE)

# POSICAO DAS LEGENDAS DOS EIXOS
par("mgp") # três valores que correspondem ao número de linhas para: (1) Titulo dos Eixos; (2)

# TAMANHO DE FONTE DOS EIXOS É RELATIVO AO VALOR DE FONTE PADRÃO
par("cex") # tamanho padrão
par("cex.lab") # número que multiplicado por op$cex indica o valor da fonte dos títulos dos eixos
par("cex.axis") # dos valores

# TAMANHO DAS BARRAS DE CADA VALOR
par("tck") # geralmente não tem padrão definido pois é extraído de outros valores automaticamente

# DIMINUINDO A FONTE DISSO
par(cex.lab = 0.8, cex.axis = 0.7)
plot(Sepalas, Petalas, pch = sbs, bg = vcl, cex = 0.8)

# note que isso muda se eu alterar o tamanho de fonte padrão par(cex)
par(cex = 2, cex.lab = 0.8, cex.axis = 0.7)
plot(Sepalas, Petalas, pch = sbs, bg = vcl, cex = 0.8)

# APROXIMANDO
par(mgp = c(1.5, 0.5, 0))
plot(Sepalas, Petalas, pch = sbs, bg = vcl, cex = 0.8)

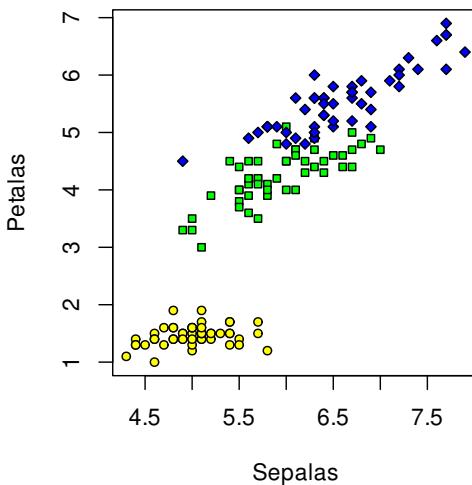
# e agora
par(tck = -0.01) # note o valor negativo
plot(Sepalas, Petalas, pch = sbs, bg = vcl, cex = 0.8)

par(tck = 0.01) # se colocar positivo
plot(Sepalas, Petalas, pch = sbs, bg = vcl, cex = 0.8)
```

```
par(op) # restaurando valores originais  
plot(Sepalas, Petalas, pch = sbs, bg = vcl, cex = 0.8)
```

5.2.3 Proporção dos eixos

```
op <- par(no.readonly = TRUE)  
  
par("pty") # valor "m" maximiza a área disponível  
  
## [1] "m"  
  
plot(Sepalas, Petalas, pch = sbs, bg = vcl, cex = 0.8)  
  
# agora mantendo a proporção dos eixos (sem esticar nenhum dos dois)  
par(pty = "s")  
plot(Sepalas, Petalas, pch = sbs, bg = vcl, cex = 0.8)
```



```
# se não notou diferença, expandir o dispositivo onde está desenhando a figura
```

5.2.4 Múltiplas figuras na mesma tela ou página com `mfrow()` e `mfcoll()`

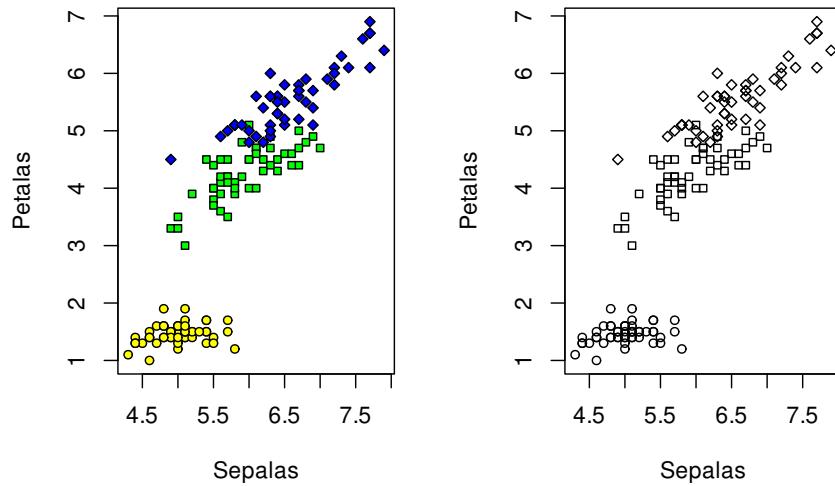
Você pode colocar diferentes gráficos na mesma tela ou na mesma página de um pdf, por exemplo. Temos duas formas de fazer isso. Os parâmetros `mfrow` e `mfcoll` dividem os dispositivo em células de tamanhos idênticos. Todas as figuras terão o mesmo tamanho. A diferença entre os dois parâmetros é que `mfrow()` preenche o espaço por linhas, enquanto `mfcoll()` preenche o espaço pelas colunas.

```
par(op)
par("mfrow") # o dispositivo não está dividido: tem 1 linha e 1 coluna
```

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

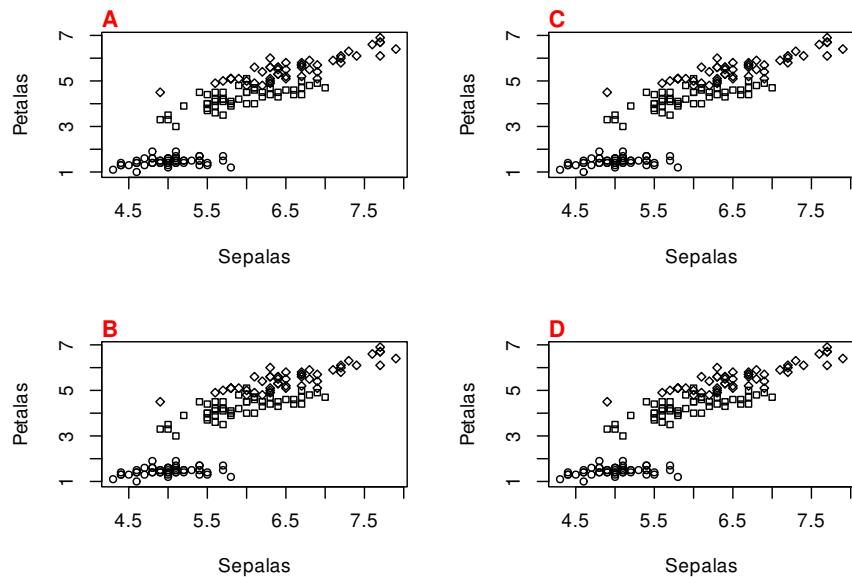
```
plot(Sepalas, Petalas, pch = sbs, bg = vcl, cex = 0.8)

# EXEMPLO 1
# duas figuras no mesmo dispositivo
par(mfrow = c(1, 2)) # dividir o dispositivo em 1 linha e duas colunas
# plota a primeira figura
plot(Sepalas, Petalas, pch = sbs, bg = vcl, cex = 0.8)
# plota a segunda sem cores
plot(Sepalas, Petalas, pch = sbs, bg = NULL, cex = 0.8)
```



```
# EXEMPLO 2
par(mar = c(5.1, 4.1, 2.1, 2.1))
par(mfrow = c(2, 2)) # dividir o dispositivo em 2 linhas e duas colunas
# plota quatro figuras identificadas
for (f in 1:4) {
  plot(Sepalas, Petalas, pch = sbs, bg = NULL, cex = 0.8)
  # adiciona uma letra para você ver a ordem e comparar com o exemplo 3
  mtext(LETTERS[f], side = 3, line = 0, adj = 0, font = 2, col = "red") # esta é uma função de R
}

# EXEMPLO 3
par(mfcol = c(2, 2)) # mesma coisa mas preenchendo por colunas (compare com a figura acima)
for (f in 1:4) {
  plot(Sepalas, Petalas, pch = sbs, bg = NULL, cex = 0.8)
  mtext(LETTERS[f], side = 3, line = 0, adj = 0, font = 2, col = "red")
}
```



```
# colocando exemplo 2 e 3 num pdf
pdf(file = "meuPDFmfrow.pdf", paper = "a4")
# EXEMPLO 2 expandido
par(mfrow = c(3, 2)) # tres linhas e duas colunas
for (f in 1:6) {
  plot(Sepalas, Petalas, pch = sbs, bg = NULL, cex = 0.8)
  mtext(LETTERS[f], side = 3, line = 0, adj = 0, font = 2, col = "red")
}
# EXEMPLO 3 expandido
par(mfcol = c(3, 2))
for (f in 1:6) {
  plot(Sepalas, Petalas, pch = sbs, bg = NULL, cex = 0.8)
  mtext(LETTERS[f], side = 3, line = 0, adj = 0, font = 2, col = "red")
}
dev.off()
```

5.2.5 Múltiplas figuras na mesma image usando a função `layout()`

A função `layout()` também permite dividir um dispositivo para múltiplas figuras, mas de uma forma muito mais complexa.

```
?layout # veja o help dessa função

# voce precisa definir uma matriz que indica:
# 1) o numero de figuras (valores da matriz)
# 2) a posição das figuras (numero de linhas e colunas)
# por exemplo, suponha que queremos plotar 3 figuras:
# 1 preenchendo a largura da página e metade da altura
# 2 outras figuras preenchendo a outra metade da altura
```

```
# neste caso a figura 1 ira ocupar dois espacos
mm <- matrix(c(1, 1, 2, 3), nrow = 2, ncol = 2, byrow = T)
mm # a figura 1 irá ocupar a posicao do numero 1 na matriz
# a figura 2 irá ocupar a posicao do numero 2 nessa matriz
# a figura 3 irá ocupar a posicao do numero 3 dessa matriz
# a largura de cada coluna e linha é especificada pelos argumentos widths e heights e usaremos
# divide o dispositivo
ml <- layout(mm, widths = rep(lcm(5), ncol(mm)), heights = rep(lcm(5), nrow(mm)))
# mostra a divisao feita
layout.show(ml)

# plota a primeira figura
plot(Sepalas, Petalas, pch = sbs, bg = vcl, cex = 0.8)
# a segunda
plot(Sepalas, Petalas, pch = sbs, bg = vcl, cex = 0.8)
# a terceira
plot(Sepalas, Petalas, pch = sbs, bg = vcl, cex = 0.8)
```

Podemos salvar essas figuras em um pdf:

```
pdf(file = "meuPDFLayout.pdf", paper = "letter")
layout(mm, widths = rep(lcm(8), ncol(mm)), heights = rep(lcm(8), nrow(mm)))
# plota as tres figuras fazendo uma iteracao
for (f in 1:3) {
  plot(Sepalas, Petalas, pch = sbs, bg = NULL, cex = 0.8)
  mtext(LETTERS[f], side = 3, line = 0, adj = 0, font = 2, col = "red")
}
dev.off() # fecha o pdf
```

5.3 Parâmetros gráficos, parte II - Símbolos e cores

Você pode fazer o que quiser na produção de um gráfico no R. Isso requer conhecer bem os parâmetros gráficos que já apresentamos de forma geral anteriormente (ver seção 5.2). Aqui apresentamos alguns parâmetros de uso comum pelas funções gráficas de alto (Seção 5.4) e baixo nível (seção 5.5), que definem símbolos e cores. Você viu isso se leu o ? da função `par()` e entendeu o que ela faz.

Em gráficos de dispersão e/ou na necessidade de colocar qualquer símbolo em um gráfico qualquer, precisamos saber como definir símbolos, suas cores e seus tamanhos. Os parâmetros gráficos que fazem isso são principalmente os seguintes:

- `pch` - define o tipo de símbolo para pontos;
- `lty` - define o tipo de símbolo para linhas;
- `col` e `bg` - respectivamente definem a cor de linhas e o “recheio” do ponto de símbolos, gráficos, polígonos etc;
- `cex` - define o tamanho de símbolos e texto;
- `lwd` - espessura das linhas.

5.3.1 Tipo de símbolo - `pch`

Nós queremos símbolos, em geral, para adicionar em um gráfico os pontos referentes às nossas unidades amostrais. Portanto, esse tipo de símbolo no R é chamado de `points`, e já vimos a função de baixo-nível `points()`, que depende desse argumento ou parâmetro. O argumento `pch` pode ser um número², que especifica um símbolo, ou símbolos de um único caractere [e.g. `c("A", "*", "&", "?")`].

```
# veja o definido como parâmetro global
par("pch")

# o que isso quer dizer?
# vamos usar a função example()
?example # se quiser saber o que isso faz
# quando digitar o comando abaixo,
# vai precisar RESPONDER NO CONSOLE para prosseguir.
example("points")
# pare na terceira figura que esse exemplo gera.
par("pch") # este valor corresponde ao símbolo nessa figura.
# note que os símbolos de 21:25 permitem definir cor de linha e de recheiro, os demais apenas
```

```
# vamos gerar alguns exemplos com os dados de iris
class(iris)
```

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

```
dim(iris)
```

²Valores do argumento `pch` variam de 0 a 25; veja o ? da função `points()` para saber quais são esses símbolos

```
## [1] 150    5
```

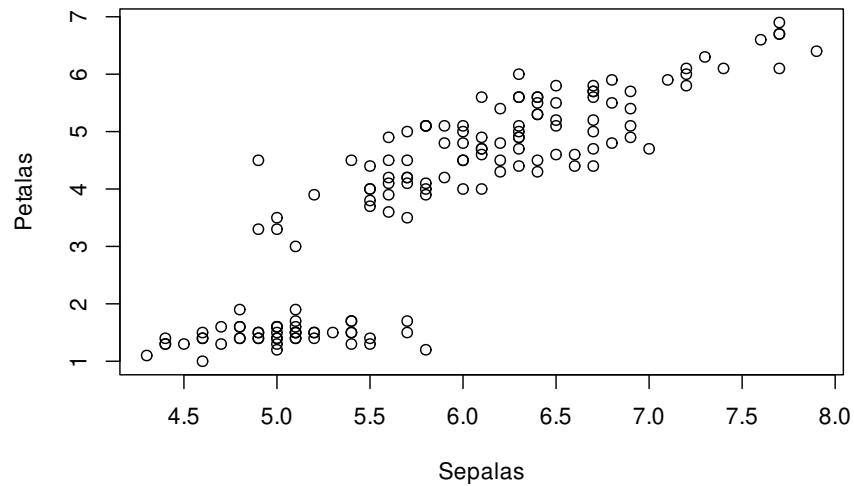
```
colnames(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

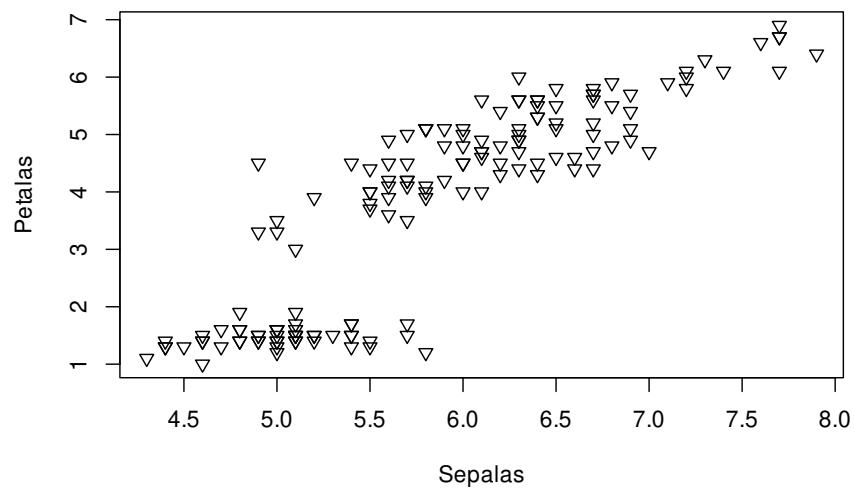
```
# com a definição padrão de pch  
par("pch")
```

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

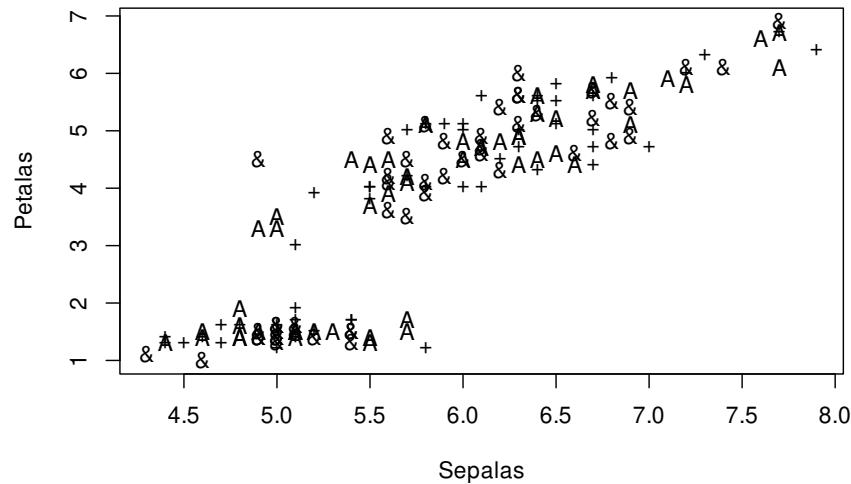
```
Sepalas <- iris$Sepal.Length  
Petalas <- iris$Petal.Length  
plot(Sepalas, Petalas)
```



```
# colocando todas com um único símbolo diferente  
plot(Sepalas, Petalas, pch = 25)
```



```
# colocando cada espécie com um único símbolo  
# neste caso o argumento pch deve ter um vetor do número de linhas em iris, se for menor, pelo  
# então se eu definir apenas três simbolos, um para cada espécie:  
plot(Sepalas, Petalas, pch = c("A", "&", "+"))
```

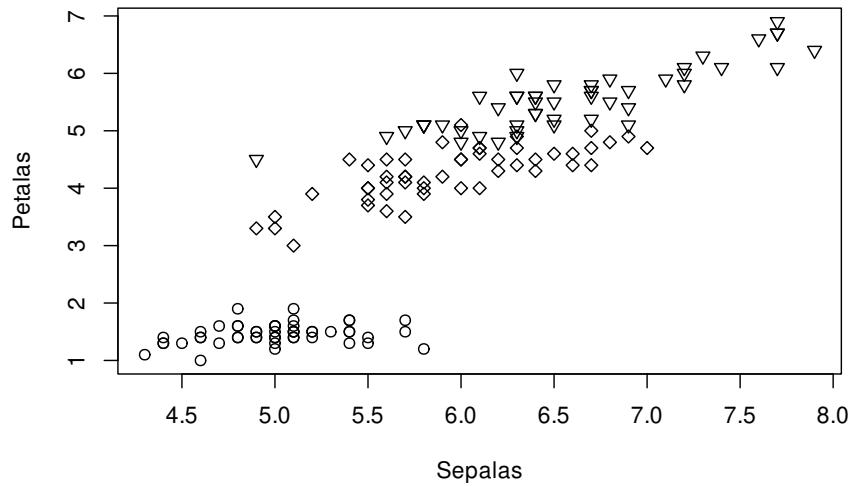


```
# eles vão aparecer misturados, isso não está de acordo com as espécies
# uma forma rápida de fazer isso, tendo em vista que o pch pode ser um valor numéricico,
# é transformar o nome das minhas espécies em números.
# Eu posso fazer isso se convertendo um fator para número, pois nele as categorias estão expli-
spp <- iris$Species
class(spp) # já é um fator
```

```
# mas isso gera números por spp é um fator  
as.numeric(spp)
```

```
# então se eu quero os símbolos 22,23,25 para representar minhas espécies, eu uso esses números
pch.das.spp <- c(21, 23, 25)[as.numeric(spp)]
pch.das.spp
```

```
# posso usar isso como argumento em plot:  
plot(Sepalas, Petalas, pch = pch.das.spp)
```

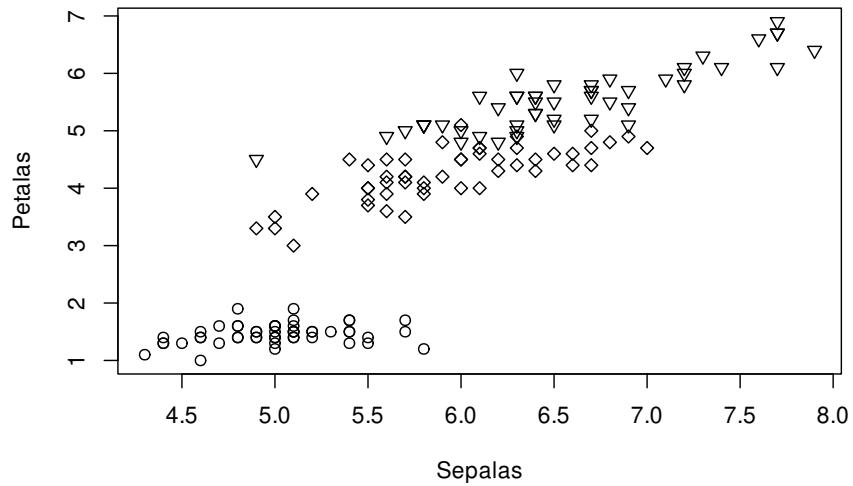


```
# pronto cada espécie é um símbolo diferente
```

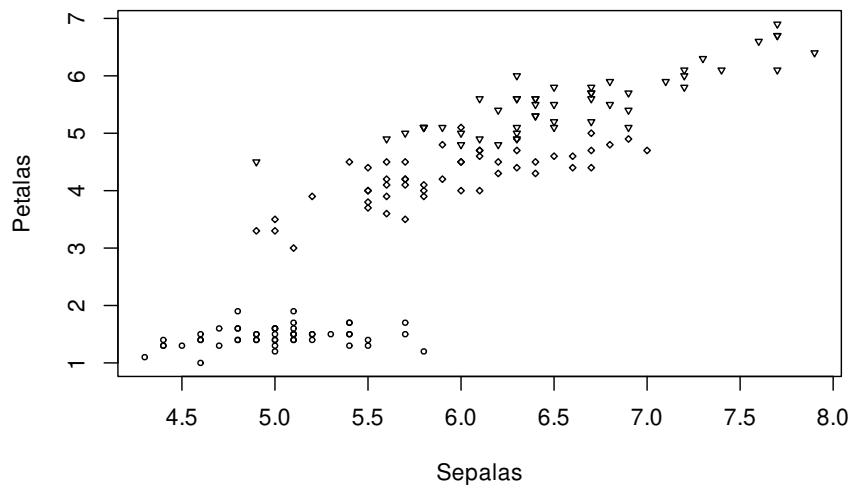
5.3.2 Tamanho dos pontos - cex

O argumento genérico `cex` especifica o tamanho dos pontos relativo ao padrão do dispositivo. Aceita um valor numérico que é multiplicado pelo valor do seu dispositivo. O padrão geral é 1, isto é, 100% do tamanho. Se colocar 0.5 teremos 50%; se 1.5, teremos 150% do tamanho padrão.

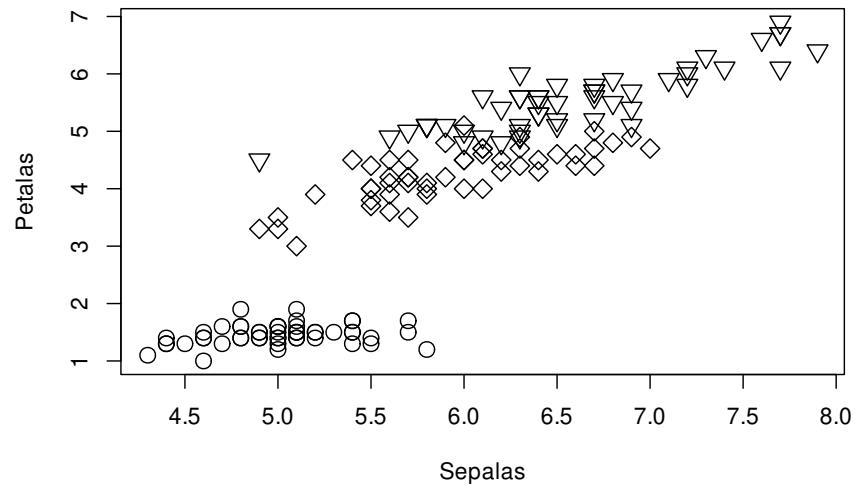
```
# da etapa anterior
pch.das.spp <- c(21, 23, 25)[as.numeric(spp)]
# temos nosso gráfico com símbolos
plot(Sepalas, Petalas, pch = pch.das.spp)
```



```
# mudando o tamanho de todos os simbolos  
plot(Sepalas, Petalas, pch = pch.das.spp, cex = 0.5)
```

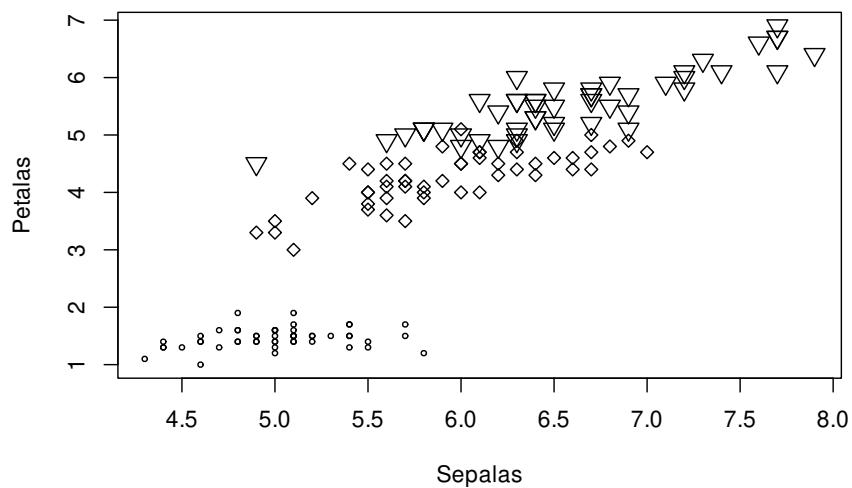


```
# para maior  
plot(Sepalas, Petalas, pch = pch.das.spp, cex = 1.5)
```



```
# um tamanho diferente por ponto (especie)
tm <- c(0.5, 1, 1.5)[as.numeric(spp)]
tm
```

```
plot(Sepalas, Petalas, pch = pch.das.spp, cex = tm)
```



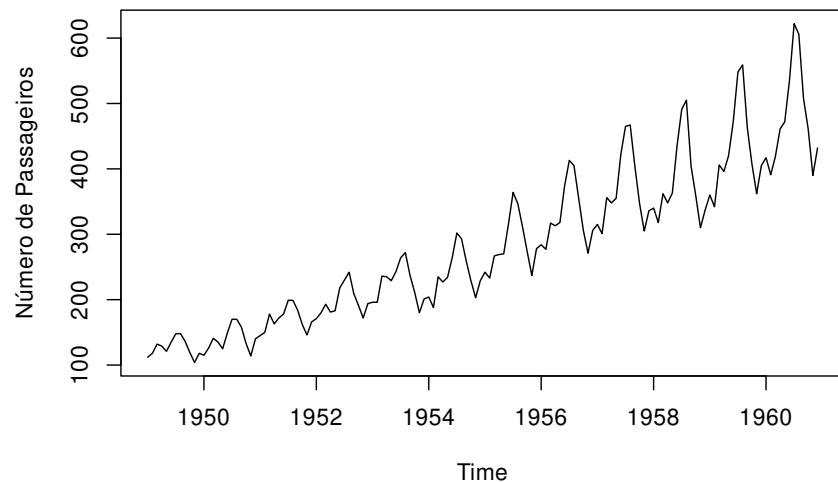
5.3.3 Linhas

Os argumentos `lwd` e `lty` controlam a espessura e o tipo das linhas, respectivamente.

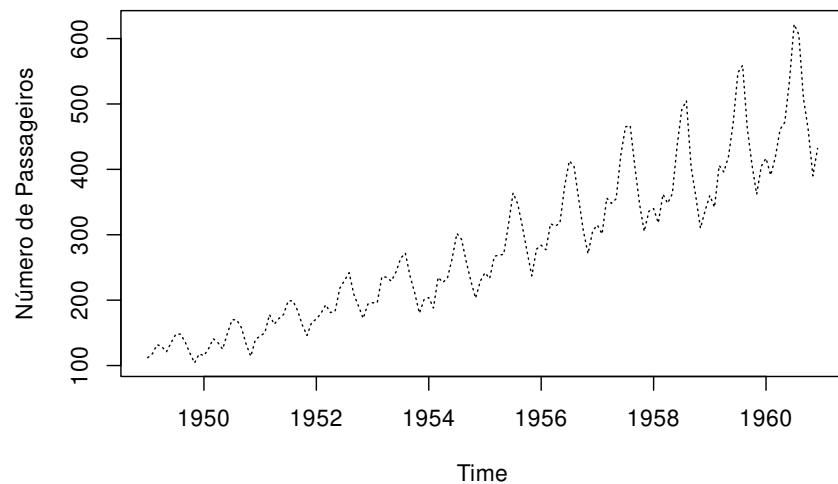
```
data("AirPassengers") # veja esse conjunto de dados com ?datasets  
class(AirPassengers)
```

```
## [1] "ts"
```

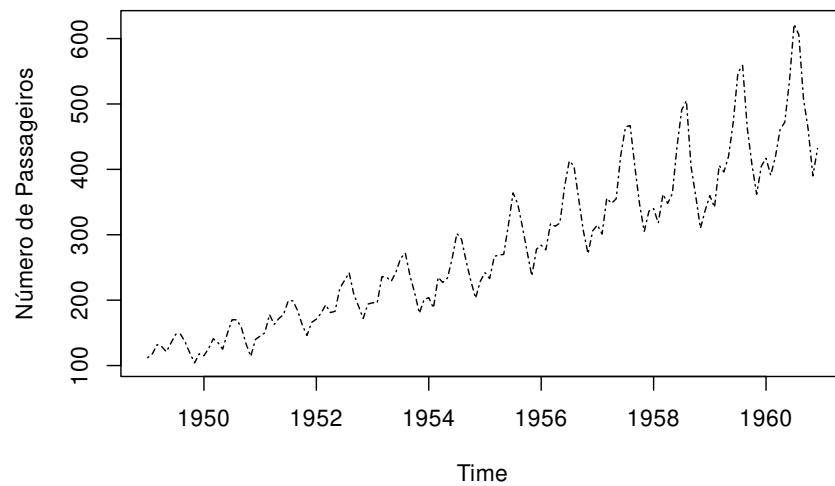
```
plot(AirPassengers, ylab = "Número de Passageiros")
```



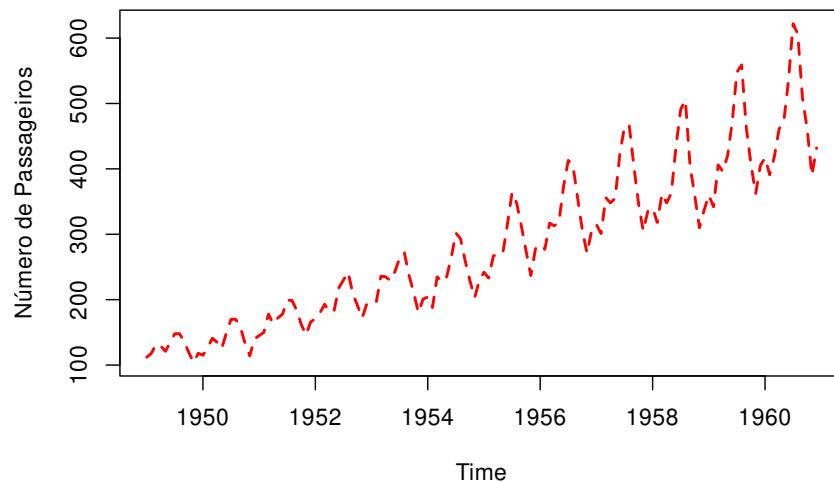
```
# mudando o tipo de linha  
plot(AirPassengers, ylab = "Número de Passageiros", lty = "dotted")
```



```
# outro tipo  
plot(AirPassengers, ylab = "Número de Passageiros", lty = "dotdash")
```



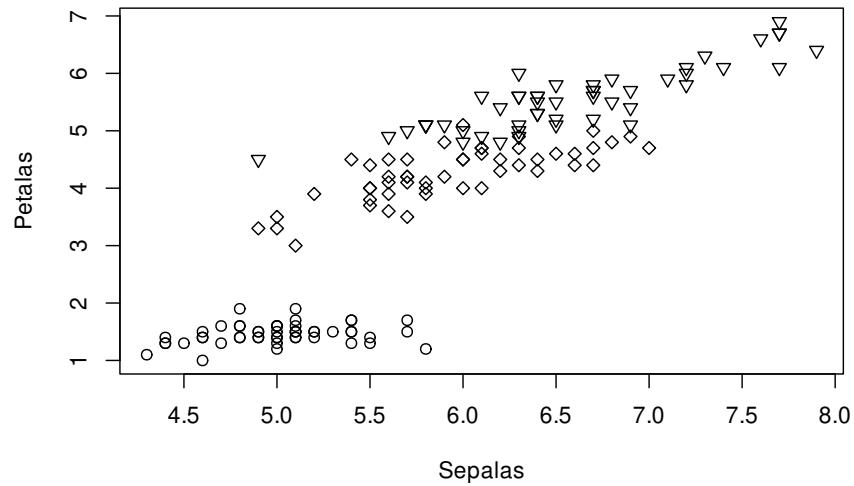
```
# mudando a cor e espessura  
plot(AirPassengers, ylab = "Número de Passageiros", lty = "dashed", col = "red", lwd = 2)
```



5.3.4 Cores de símbolos - `col` e `bg`

O argumento `col` define cores para os símbolos. No caso de símbolos de uma única cor, utilizamos apenas este argumento. Já o argumento `bg` define as cores do preenchimento dos símbolos.

```
# o que definimos antes são símbolos que permitem definir cores para preenchimento e linha:  
pch.das.spp <- c(21, 23, 25)[as.numeric(spp)]  
# temos nosso gráfico com símbolos  
plot(Sepalas, Petalas, pch = pch.das.spp)
```

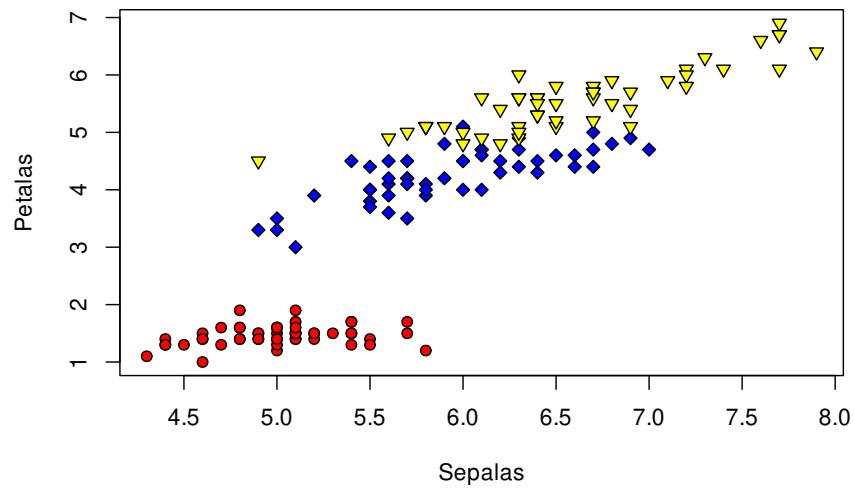


```
# vamos adicionar as cores seguindo o mesmo raciocínio
cores.spp <- c("red", "blue", "yellow")[as.numeric(spp)]
cores.spp
```

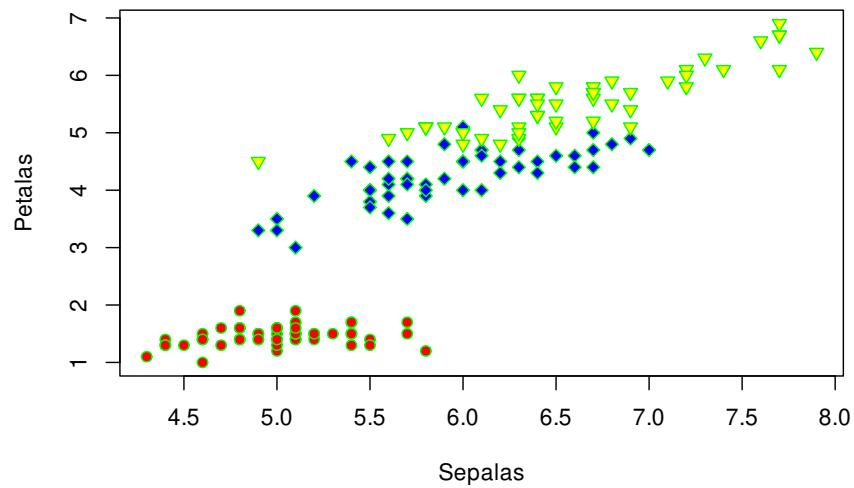
```
## [1] "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"
## [9] "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"
## [17] "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"
## [25] "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"
## [33] "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"
## [41] "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"
## [49] "red"   "red"   "blue"   "blue"   "blue"   "blue"   "blue"   "blue"   "blue"
## [57] "blue"   "blue"   "blue"   "blue"   "blue"   "blue"   "blue"   "blue"   "blue"
## [65] "blue"   "blue"   "blue"   "blue"   "blue"   "blue"   "blue"   "blue"   "blue"
## [73] "blue"   "blue"   "blue"   "blue"   "blue"   "blue"   "blue"   "blue"   "blue"
## [81] "blue"   "blue"   "blue"   "blue"   "blue"   "blue"   "blue"   "blue"   "blue"
## [89] "blue"   "blue"   "blue"   "blue"   "blue"   "blue"   "blue"   "blue"   "blue"
## [97] "blue"   "blue"   "blue"   "blue"   "yellow" "yellow" "yellow" "yellow" "yellow"
## [105] "yellow" "yellow" "yellow" "yellow" "yellow" "yellow" "yellow" "yellow" "yellow"
## [113] "yellow" "yellow" "yellow" "yellow" "yellow" "yellow" "yellow" "yellow" "yellow"
```

```
## [121] "yellow" "yellow" "yellow" "yellow" "yellow" "yellow" "yellow" "yellow"  
## [129] "yellow" "yellow" "yellow" "yellow" "yellow" "yellow" "yellow" "yellow"  
## [137] "yellow" "yellow" "yellow" "yellow" "yellow" "yellow" "yellow" "yellow"  
## [145] "yellow" "yellow" "yellow" "yellow" "yellow" "yellow" "yellow"
```

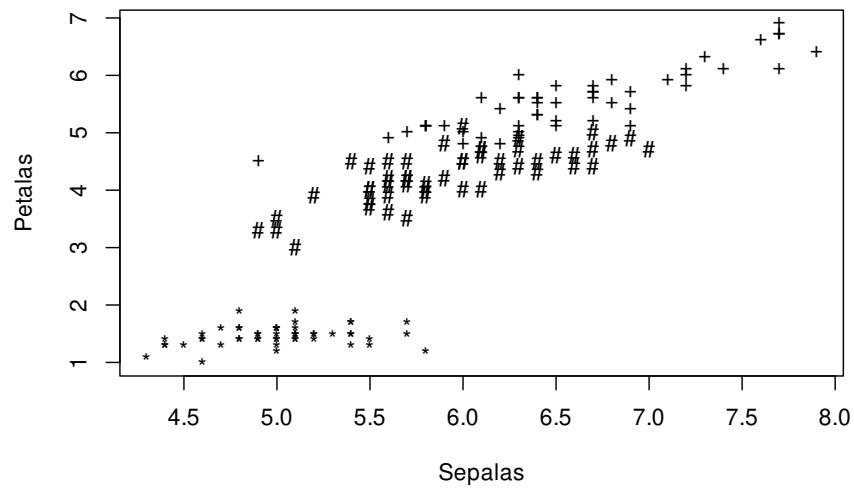
```
plot(Sepalas, Petalas, pch = pch.das.spp, bg = cores.spp)
```



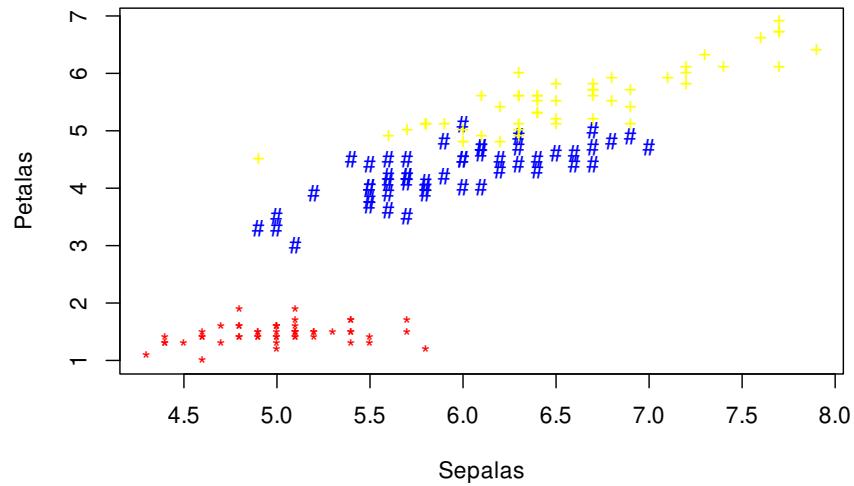
```
# podemos mudar a cor da linha desses símbolos (todos para verde pela regra da reciclagem espelhada)  
plot(Sepalas, Petalas, pch = pch.das.spp, bg = cores.spp, col = "green")
```



```
# mudando simbolos
nsb <- c("*", "#", "+")[as.numeric(spp)]
plot(Sepalas, Petalas, pch = nsb, bg = cores.spp)
```



```
# note que apesar de bg estar especificando cores as cores nao foram adicionadas, porque coloca
# Mudando:
plot(Sepalas, Petalas, pch = nsb, col = cores.spp)
```



5.3.5 Funções que definem cores

Cores podem ser definidas por palavras ou por códigos. As funções `colors()` ou `colours()` listam cores pelo nome.

```
# lista todas as cores disponíveis pelo nome
colors()
# nossa tem 657 cores. Difícil, né?
# reduz para as mais distintas
colors(distinct = TRUE)
# ainda 502
```

```
# eu posso pegar ou buscar por cores
ascores <- colors(distinct = TRUE)
vl <- grep("blue", ascores)
ascores[vl]
```

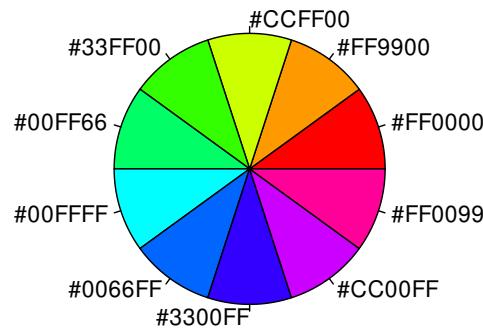
```
# 60 nomes que contém a palavra azul  
  
# vamos ver todas as cores em um pdf longo  
# vamos plotar pizzas coloridas com 10 cores cada  
# 4 pizzas por página
```

```
?pie # veja o help dessa função
```

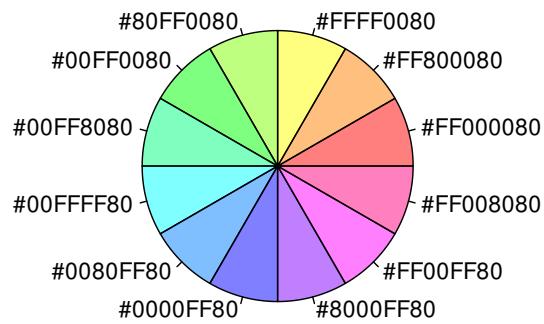
```
# abre um PDF  
pdf("cores.pdf", width = lcm(29), height = lcm(21))  
  
# vamos dividir o dispositivo em 2 colunas e 3 linhas (e diminuir a margem)  
par(mfrow = c(2, 2), mar = c(3, 3, 3, 3), cex = 0.7)  
ln <- length(ascores)  
ln # sao 502 cores  
ceiling(ln / 10) # , entao serao 51 pizzas  
  
# plota cada pizza fazendo uma iteração:  
idx <- 0 # o objeto nullo para usar de indice na iteracao para fazer de 10 em 10  
for (p in 1:ceiling(ln / 10)) {  
  de <- idx + 1 # cor do indice idx  
  ate <- idx + 10 # ate cor do indice idx+10  
  cls <- ascores[de:ate] # cores da pizza da iteracao p  
  pie(rep(1, 10), col = cls, labels = cls)  
  idx <- ate  
}  
dev.off() # fecha o pdf
```

As funções `rainbow()` ou `terrain.colors()` geram gradientes de cores em um padrão definido.

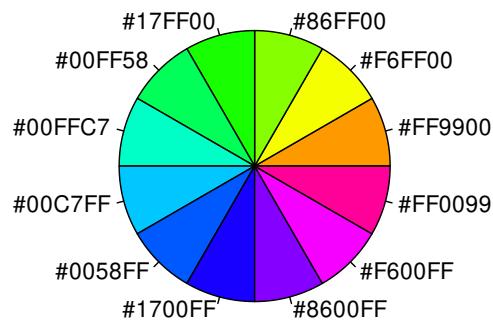
```
# 10 cores do arco-iris  
cls <- rainbow(n = 10)  
pie(rep(1, 10), col = cls, labels = cls)
```



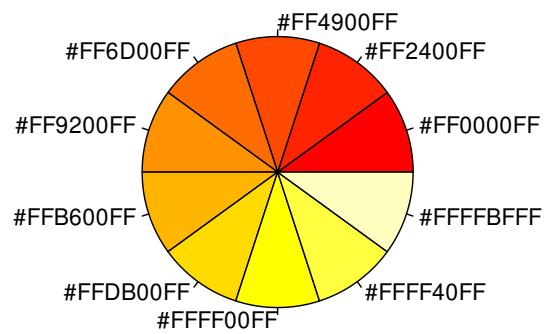
```
# 12 cores do arco-iris com 50% de transparência  
cls <- rainbow(n = 12, alpha = 0.5)  
pie(rep(1, 12), col = cls, labels = cls)
```



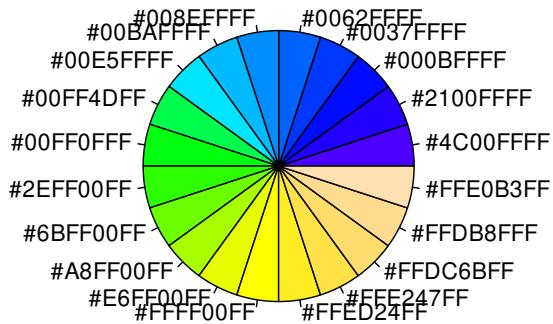
```
# 12 cores do arco-iris, limitando o espectro
cls <- rainbow(n = 12, start = 0.1, end = 0.9)
pie(rep(1, 12), col = cls, labels = cls)
```



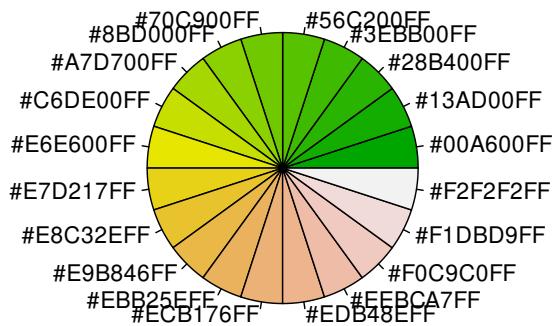
```
# 10 cores quentes
cls <- heat.colors(n = 10, alpha = 1)
pie(rep(1, 10), col = cls, labels = cls)
```



```
# 20 cores topográficas
cls <- topo.colors(n = 20, alpha = 1)
pie(rep(1, 20), col = cls, labels = cls)
```



```
# ou melhor
cls <- terrain.colors(n = 20, alpha = 1)
pie(rep(1, 20), col = cls, labels = cls)
```



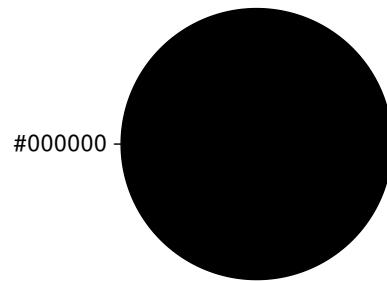
As funções `rgb()` ou `hsv()` geram qualquer tipo de cor para ser utilizados nos gráficos.

```
?rgb # veja o help dessa função e veja as funções sob See Also
```

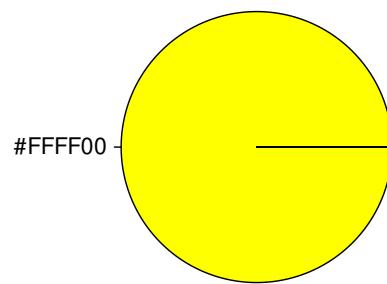
```
# podemos entender melhor fazendo o caminho inverso
defs <- col2rgb("yellow") # extraímos as especificações de amarelo (cores primárias) pela paleta
defs
```

red	255
green	255
blue	0

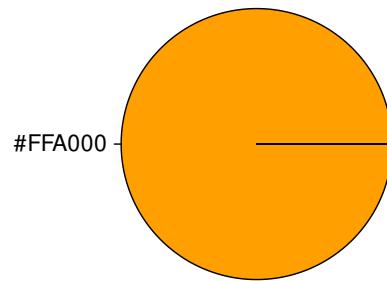
```
# sem cor temos preto
cls <- rgb(red = 0, green = 0, blue = 0) # fazemos amarelo
pie(1, col = cls, labels = cls)
```



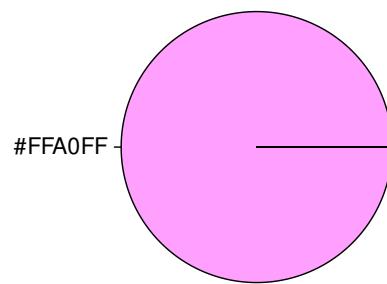
```
# fazemos amarelo
cls <- rgb(red = 255, green = 255, blue = 0, maxColorValue = 255)
pie(1, col = cls, labels = cls)
```



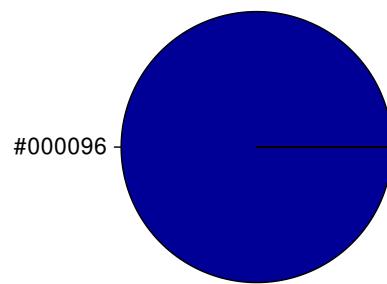
```
# tiramos verde
cls <- rgb(red = 255, green = 160, blue = 0, maxColorValue = 255)
pie(1, col = cls, labels = cls)
```



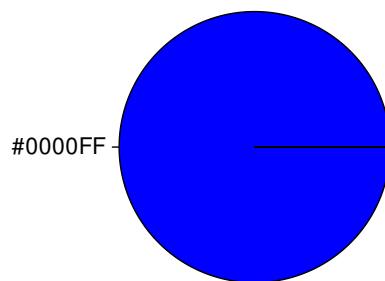
```
# tiramos verde e adicionamos azul
cls <- rgb(red = 255, green = 160, blue = 255, maxColorValue = 255)
pie(1, col = cls, labels = cls)
```



```
# tiramos azul-escuro (mais perto de 0)
cls <- rgb(red = 0, green = 0, blue = 150, maxColorValue = 255)
pie(1, col = cls, labels = cls)
```



```
# tiramos azul
cls <- rgb(red = 0, green = 0, blue = 255, maxColorValue = 255)
pie(1, col = cls, labels = cls)
```



5.4 Funções gráficas de alto nível

Funções gráficas de alto-nível (*highlevel plots*) são as funções que usamos para gerar os gráficos. Além da função genérica `plot()`, veremos apenas alguns poucos exemplos, mas existem diversas funções gráficas de alto nível, muitas das quais dependem de pacotes específicos. Existem inúmeras outras funções de alto-nível, algumas do R-base, outras geradas por colaboradores na forma de pacotes do R³. A função `plot()` entende algumas dessas outras funções dos pacotes automaticamente, dependendo da classe do objeto, e você sequer precisa usar o nome específico (veja o exemplo em `plot.phylo()`, abaixo).

No `? do pacote graphics`, você encontra ajuda para funções gráficas em geral:

³<https://cran.r-project.org/web/packages/>

```
demo("graphics") # execute este demonstrativo - lembre-se de interagir no console do R para as
?graphics # veja o help dessa função para alguns gráficos
```

5.4.1 `plot()` - uma função genérica

A função `plot()` é a principal para produção de gráficos porque é uma **função genérica** que irá gerar um gráfico dependendo da **classe do objeto**. Em muitos casos, não precisamos usar outras funções, porque a depender da classe do objeto, esta função automaticamente entenderá o que você precisa.

```
?plot # veja o help dessa função
```

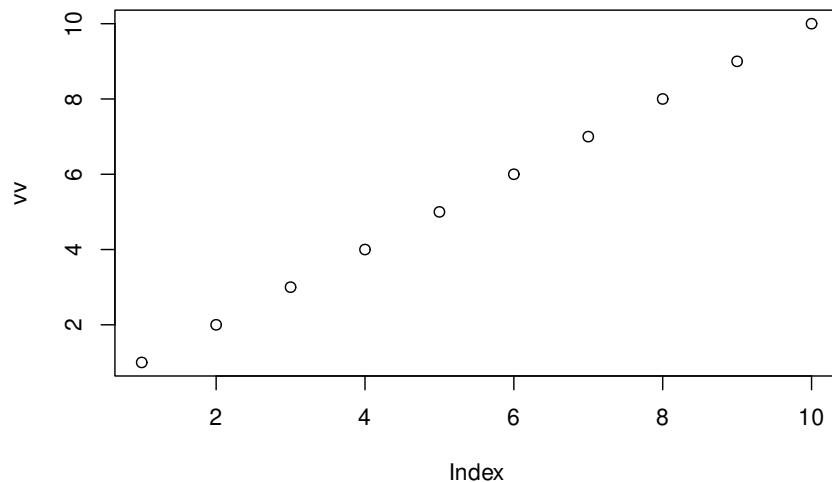
Suponha que tenhamos um vetor simples e numérico chamado `vv`:

```
vv <- 1:10
# Qual é a classe desse vetor?
class(vv)

## [1] "integer"
```

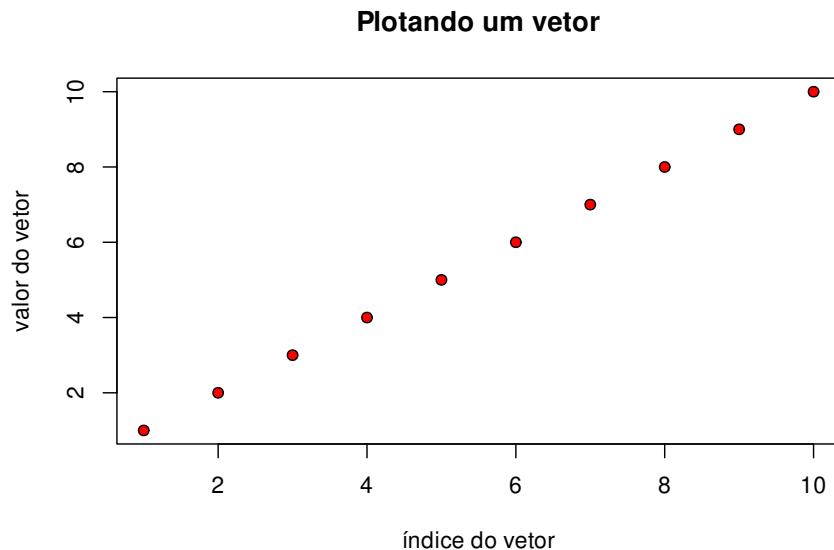
Vamos plotá-lo:

```
plot(vv)
```



Vamos mudar alguns argumentos da função `plot()`. Acrescentaremos um tipo diferente de ponto (argumento `pch`), uma cor vermelha para o fundo do ponto (argumento `bg`), um novo texto para o eixo x (argumento `xlab`), um novo texto para o eixo y (argumento `ylab`), e um título para o plot (argumento `main`):

```
plot(vv, pch = 21, bg = "red", xlab = "índice do vetor", ylab = "valor do vetor", main = "Plot")
```



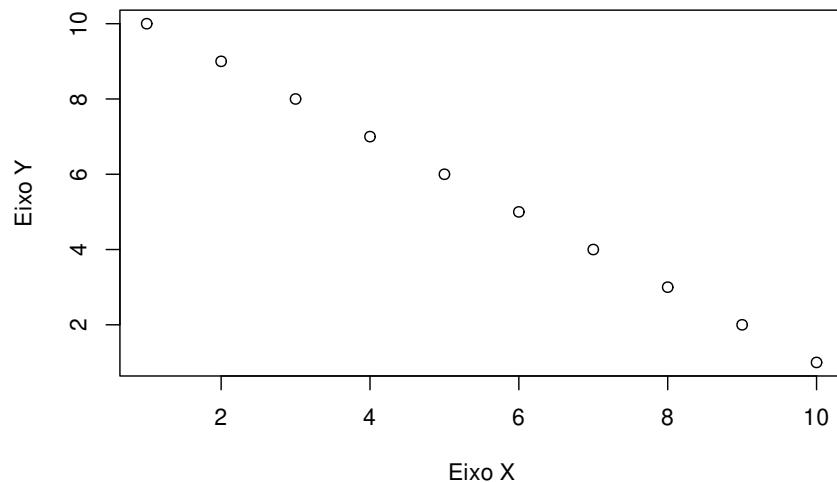
Vamos trabalhar agora com uma matriz de duas colunas:

```
xx <- 1:10 # um vetor
yy <- 10:1 # o mesmo vetor invertido
mm <- cbind(xx, yy) # junta em uma matrix
# Qual a classe dessa matriz?
class(mm)
```

```
## [1] "matrix" "array"
```

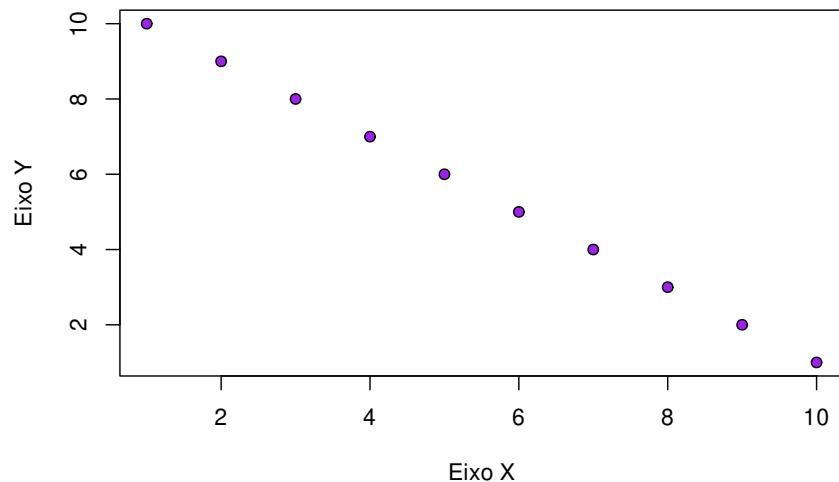
Vamos utilizar a matriz de duas colunas `mm` diretamente como primeiro argumento da função `plot()`:

```
plot(mm, xlab = "Eixo X", ylab = "Eixo Y", main = "Plotando uma matriz de 2 colunas")
```

Plotando uma matriz de 2 colunas

Podemos também especificar separadamente os eixos x e y, utilizando os vetores `xx` e `yy`, utilizados para compor a matriz `mm`:

```
# ou poderia especificar diretamente os eixos xx e yy sem usar a matriz
plot(xx, yy, xlab = "Eixo X", ylab = "Eixo Y", main = "Plotando uma matriz de 2 colunas", pch = 1)
```

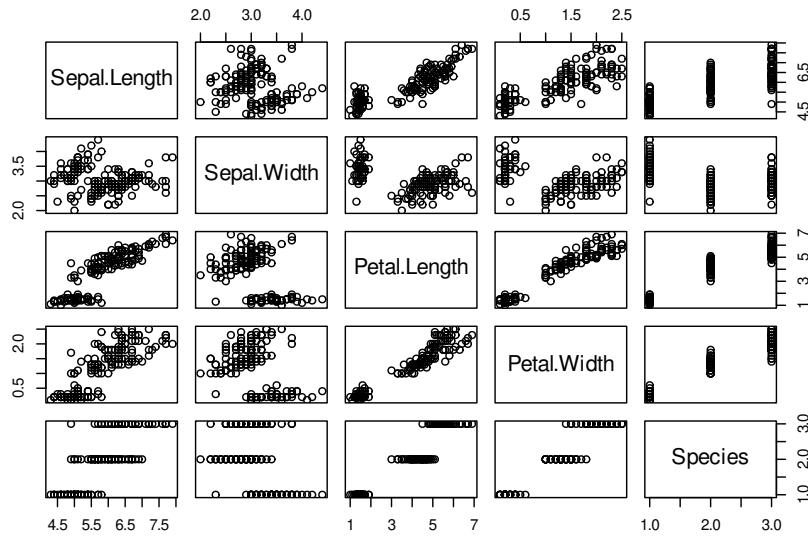
Plotando uma matriz de 2 colunas

Se temos um conjunto de dados e queremos ver a relação entre si de todas as colunas do conjunto, podemos simplesmente usar:

```
class(iris) # um conjunto de dados no formato data.frame
```

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

```
plot(iris)
```



```
# o nome dos eixos está na diagonal dessa figura
```

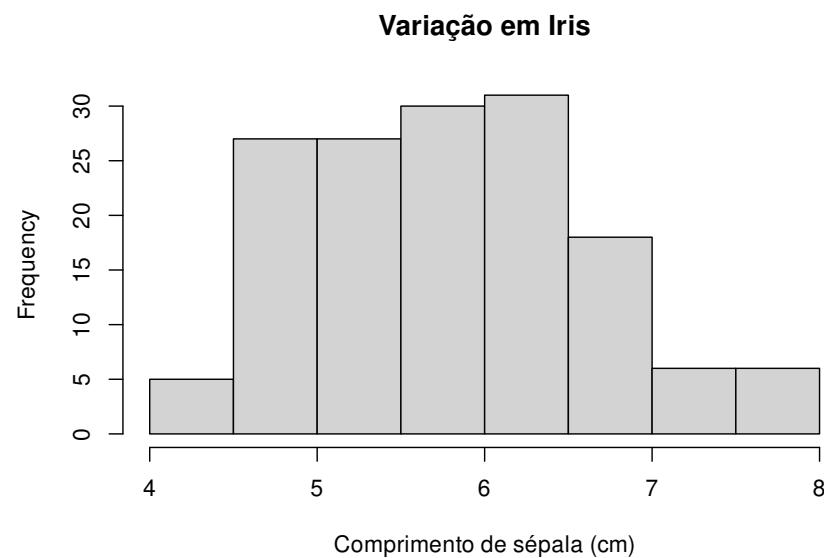
5.4.2 hist()

Esta importante função serve para visualizar a variação de uma variável apenas.

```
?hist # veja o help
```

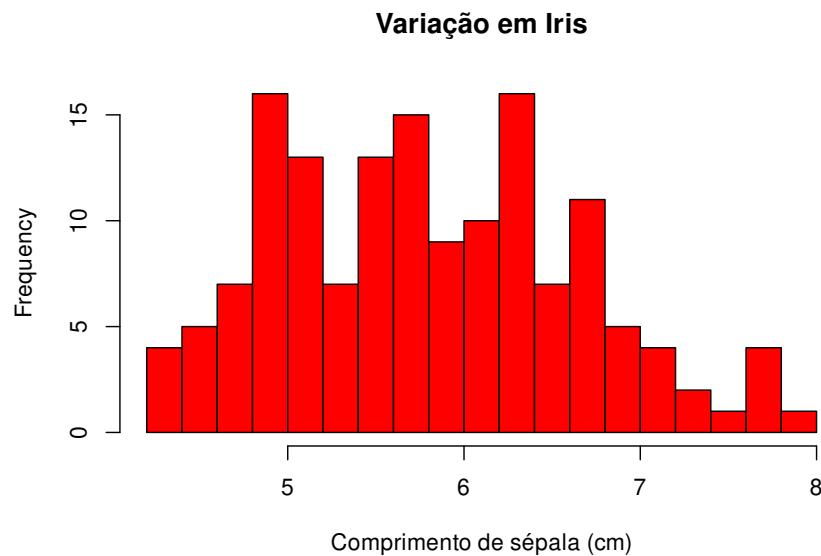
Vamos utilizar apenas a variável Sepal.Length do conjunto de dados `iris`:

```
class(iris$Sepal.Length) # é um vetor  
## [1] "numeric"  
  
hist(iris$Sepal.Length, xlab = "Comprimento de sépala (cm)", main = "Variação em Iris")
```



Vamos diminuir os espaçamentos entre as barras e colorí-las:

```
# diminuindo os espaçamentos entre as barras e colorindo  
hist(iris$Sepal.Length, xlab = "Comprimento de sépala (cm)", main = "Variação em Iris", breaks
```



```
# note que o eixo y é a frequencia que o valor ocorre, ou seja, corresponde ao número de linhas
```

5.4.3 `boxplot()`

Esta função é importante por mostrar a variação em uma variável qualquer em relação às categorias de um fator. É uma das melhores formas gráficas para mostrar a distribuição de valores de uma variável em relação às categorias.

```
?boxplot
```

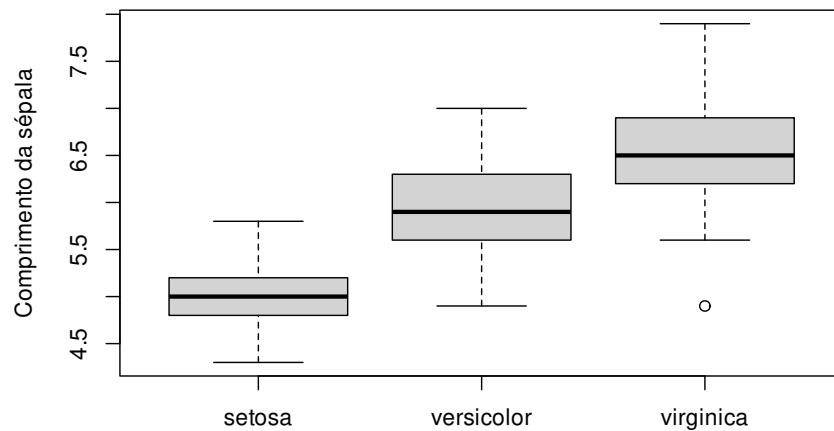
```
# vamos usar o objeto iris
class(iris$Sepal.Length) # é um vetor
```

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

```
class(iris$Species) # é um fator  
## [1] "factor"  
  
boxplot(iris$Sepal.Length ~ iris$Species, ylab = "Comprimento da sépala")
```



```
# você pode fazer a mesma coisa com a função plot() porque ela é genérica que irá reconhecer o tipo de dado  
plot(iris$Sepal.Length ~ iris$Species, ylab = "Comprimento da sépala", xlab = "") # note que o xlab é vazio
```



5.4.4 barplot()

Esta função gera um gráfico de barras simples.

```
?barplot # veja o help dessa função  
?VADeaths # veja o help desse conjunto de dados
```

```
class(VADeaths)
```

```
## [1] "matrix" "array"
```

```
VADeaths # taxas de mortalidade/1000 habitantes no estado da Virgínia em 1940
```

	Rural Male	Rural Female	Urban Male	Urban Female
50-54	11.7	8.7	15.4	8.4
55-59	18.1	11.7	24.3	13.6
60-64	26.9	20.3	37.0	19.3
65-69	41.0	30.9	54.6	35.1
70-74	66.0	54.3	71.1	50.0

```
rownames(VADeaths) # cada linha é uma classe de idade
```

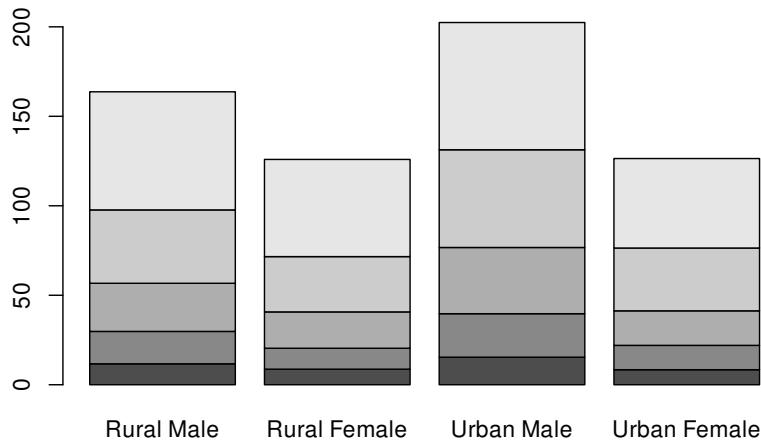
```
## [1] "50-54" "55-59" "60-64" "65-69" "70-74"
```

```
colnames(VADeaths) # cada coluna é sexo e cidade ou rural
```

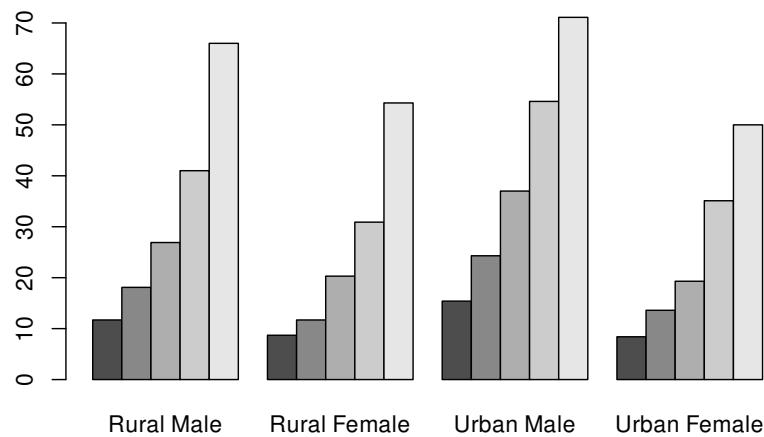
```
## [1] "Rural Male"    "Rural Female"   "Urban Male"     "Urban Female"
```

```
# com as informações padrão
```

```
barplot(VADeaths) # cada cor é uma linha
```



```
barplot(VADeaths, beside = TRUE) # lado a lado (cada barra é uma linha)
```



```
# vamos colorir diferente, uma para cada linha  
rownames(VADeaths)
```

```
## [1] "50-54" "55-59" "60-64" "65-69" "70-74"
```

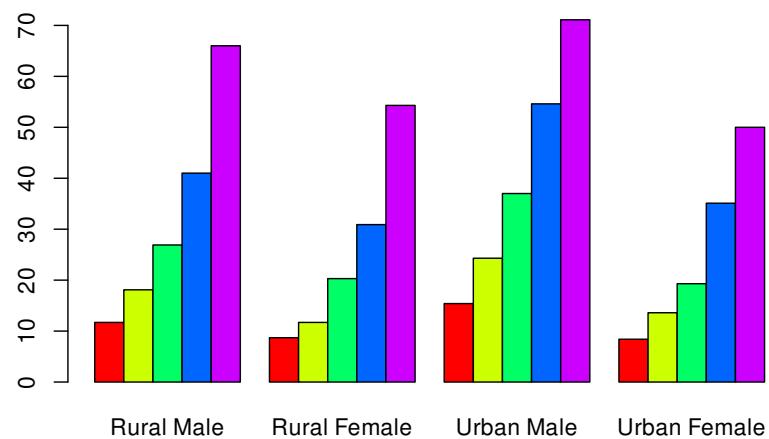
```
cores <- rainbow(n = nrow(VADeaths))  
cores # uma cor por linha
```

```
## [1] "#FF0000" "#CCFF00" "#00FF66" "#0066FF" "#CC00FF"
```

```
# essa é a correspondencia:  
cbind(rownames(VADeaths), cores)
```

	cores
50-54	#FF0000
55-59	#CCFF00
60-64	#00FF66
65-69	#0066FF
70-74	#CC00FF

```
# plota com essas cores  
barplot(VADeaths, beside = TRUE, col = cores)
```



5.4.5 `plot.phylo()`

O pacote Ape (Analyses of Phylogenetics and Evolution)⁴ (Paradis et al., 2020) possui uma função para desenhar uma filogenia. Caso você não tenha instalado o pacote `ape` ainda, faça-o assim:

```
install.packages("ape")
```

Depois, siga o exemplo abaixo:

```
library(ape) # chama o pacote

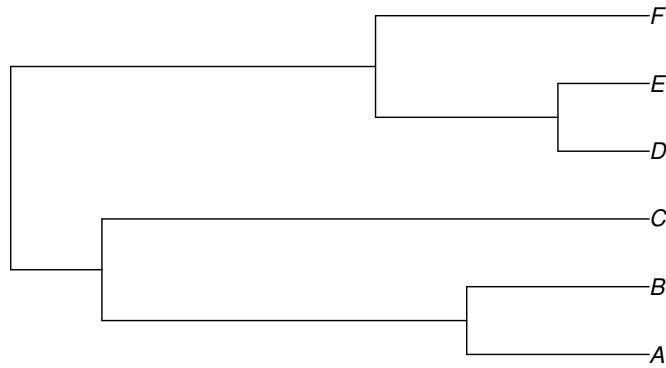
# uma filogenia hipotética para exemplo
arv <- "(((A:1,B:1):2,C:3):0.5,((D:0.5,E:0.5):1,F:1.5):2);"

# lê essa filogenia para um objeto de classe phylo, que é uma classe para dendrogramas.
phy <- read.tree(text = arv)
class(phy)

## [1] "phylo"
```

```
# podemos plotar essa filogenia com
plot(phy)
```

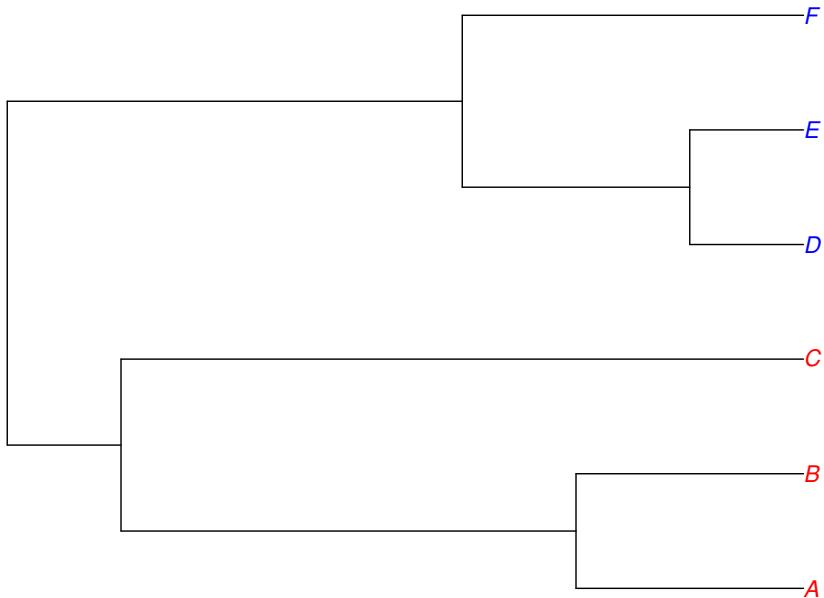
⁴<https://cran.r-project.org/web/packages/ape/index.html>



```
# porque a função plot reconhece  
# que objetos de classe "phylo" são árvores  
# de fato ela usa a função plot.phylo() para gerar essa figura, que contém argumentos específicos
```

```
?plot.phylo # veja o help e os argumentos
```

```
# portanto posso incluir argumentos de plot.phylo() quando uso a função plot() para plotar um  
plot(phy, no.margin = TRUE, tip.color = c(rep("red", 3), rep("blue", 3)))  
  
# seria o mesmo que dizer  
plot.phylo(phy, no.margin = TRUE, tip.color = c(rep("red", 3), rep("blue", 3)))
```

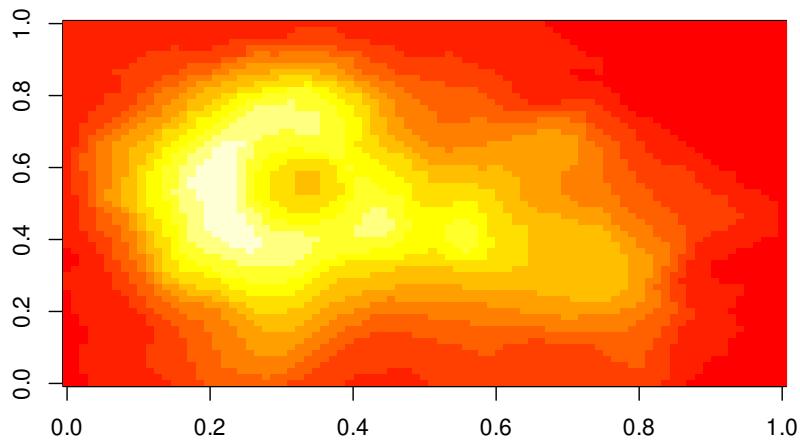


5.4.6 `image()` e `contour()`

```
# existe um conjunto de dados topográficos na base do R que vamos usar como exemplo
class(volcano) # é uma matriz
dim(volcano) # com 87 linhas e 61 uma colunas
volcano[1:5, 1:5] # os valores são altitude
hist(volcano, breaks = 20) # essa é a distribuição de valores de elevação
range(volcano) # amplitude de variação
```

```
# podemos usar algumas funções para visualizar um conjunto de dados que tem essa estrutura (m)
?image # veja o help
```

```
image(volcano, col = heat.colors(12))
```

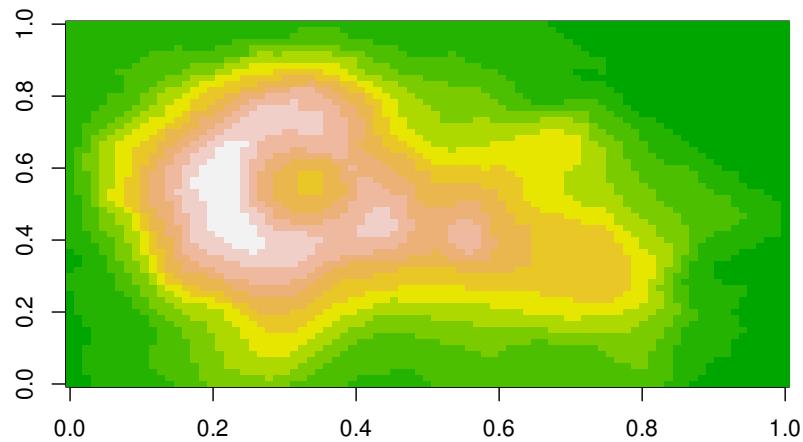


```
?heat.colors # veja opções de gradientes continuos de cores
```

```
# vamos mudar isso,  
cores <- terrain.colors(12) # 12 categorias de cores  
cores # o código é uma cor em HTML
```

```
## [1] "#00A600" "#24B300" "#4CBF00" "#7ACC00" "#ADD900" "#E6E600" "#E8C727"  
## [8] "#EAB64E" "#ECB176" "#EEB99F" "#F0FCF8" "#F2F2F2"
```

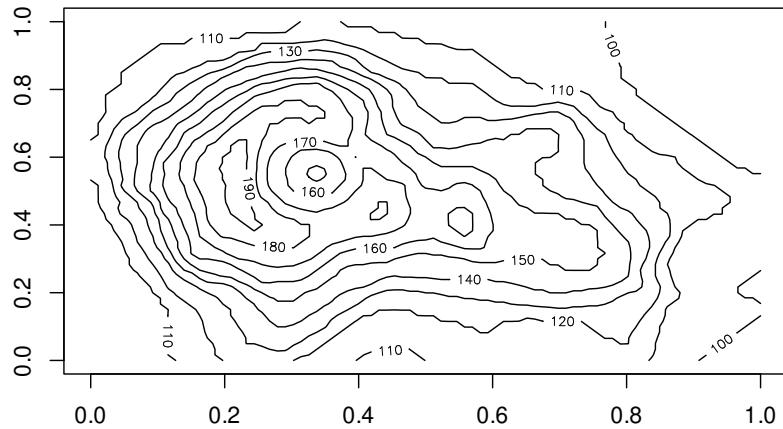
```
image(volcano, col = cores)
```



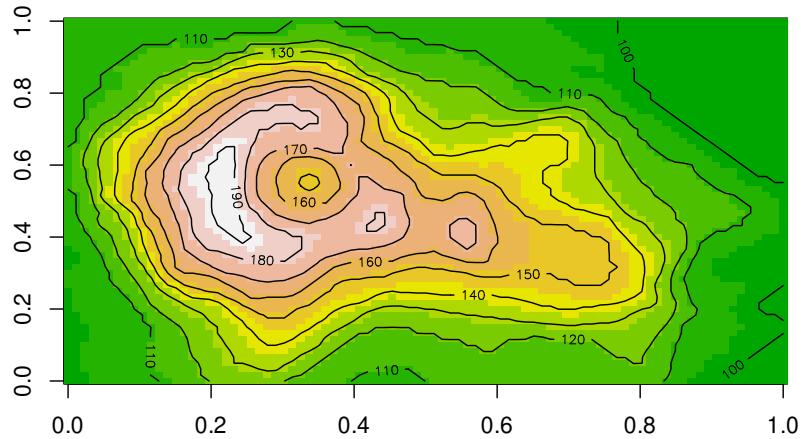
```
# notem a cratera
```

```
?contour # veja o help
```

```
# outra opção é fazermos os contornos (ou curvas de nível neste caso do vulcão)
contour(volcano)
# vamos mudar o número de níveis para o mesmo usado para as cores
contour(volcano, nlevels = 12)
```



```
# vamos juntar essas figuras em uma só
image(volcano, col = cores)
contour(volcano, nlevels = 12, add = TRUE)
```



```
# note o argumento add=TRUE que indica para a função de alto-nível
# que ela não deve começar uma nova figura,
# apenas adiciona à uma já criada por outra função de alto-nível
# muitas funções de alto-nível tem esse argumento add, que, portanto, mimetiza o que função gr...
```

5.4.7 `map()`

O pacote `maps`⁵ (Brownrigg, 2018) fornece funções para desenhar mapas com divisões administrativas. A função `map()` é uma de alto-nível. Veremos depois como adicionar sobre esses mapas informação através de funções de baixo-nível.

```
?maps:::map # veja o help
```

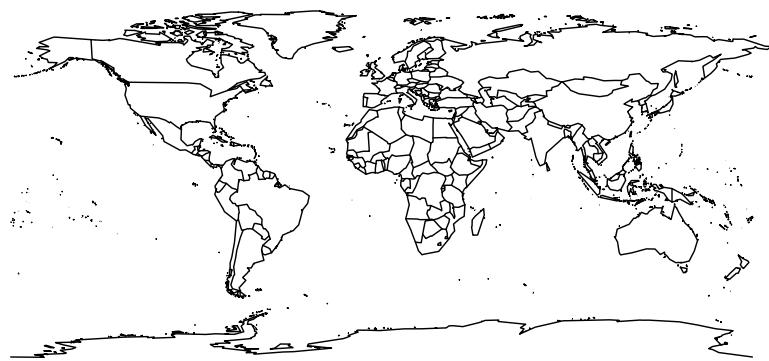
Caso você não tenha instalado o pacote `maps` ainda, faça-o assim:

⁵<https://cran.r-project.org/web/packages/maps/index.html>

```
install.packages("maps")
```

Agora, siga o exemplo abaixo:

```
library(maps) # instalar se não tiver instalado  
map() # mapa do mundo
```



```
map(region = "Brazil") # Brazil
```



```
# dev.off() #pode precisar fechar o dispositivo se estiver avisar que a figura é muito grande  
dev.off()  
  
## null device  
##  
sa <- c("Brazil", "Guyana", "French Guiana", "Suriname", "Venezuela", "Colombia", "Ecuador", "  
map(region = sa, col = "red", lty = "dashed")
```

5.4.8 xlim e ylim

Esses dois parâmetros gráficos permitem ajustar o tamanho dos eixos de um gráfico, impondo limites.

```
# CONTROLANDO UM XLIM
plot(iris$Sepal.Length, iris$Sepal.Width, pch = 21, col = "red")
# vamos aumentar um pouco o eixo X
rg <- range(iris$Sepal.Length) # variacao atual
rg
# aumentar
rg2 <- rg + c(-1, +1) # adicionamos 1 unidade de cada lado
rg2
plot(iris$Sepal.Length, iris$Sepal.Width, pch = 21, col = "red", xlim = rg2)

# LIMITANDO UM MAPA POR SUAS COORDENADAS
dev.off()
library(maps) # instalar se não tiver instalado
map() # mapa do mundo
# alguns limites em latitude e longitude
lat0 <- -22
lat1 <- 5
long0 <- -80
long1 <- -30
dev.off()
map(xlim = c(long0, long1), ylim = c(lat0, lat1))
```

5.5 Funções gráficas de baixo nível

Funções de baixo nível são funções que permitem **ADICIONAR** elementos em um gráfico já aberto com as funções `plot()` ou `plot.new()`. Muitas vezes você pode usar uma função de alto nível para fazer a mesma coisa com o argumento `add = TRUE`. Vamos ver alguns exemplos mais comumente utilizados de funções de baixo-nível.

5.5.1 `legend()`

A função `legend()` permite colocar uma legenda sobre um gráfico aberto por uma função de alto nível.

```
?legend # veja o help dessa função
# vamos usar iris novamente

Sepalas <- iris$Sepal.Length
Petalas <- iris$Petal.Length
nlevels(iris$Species) # contém três espécies

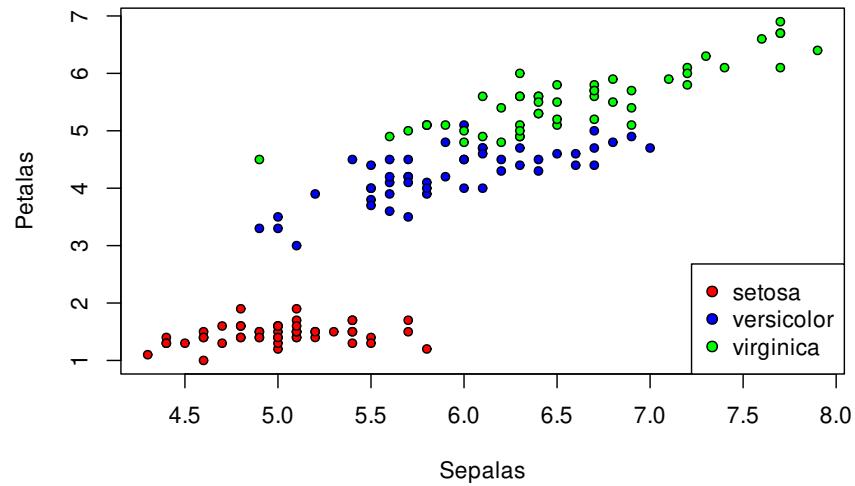
## [1] 3
```

```
levels(iris$Species)

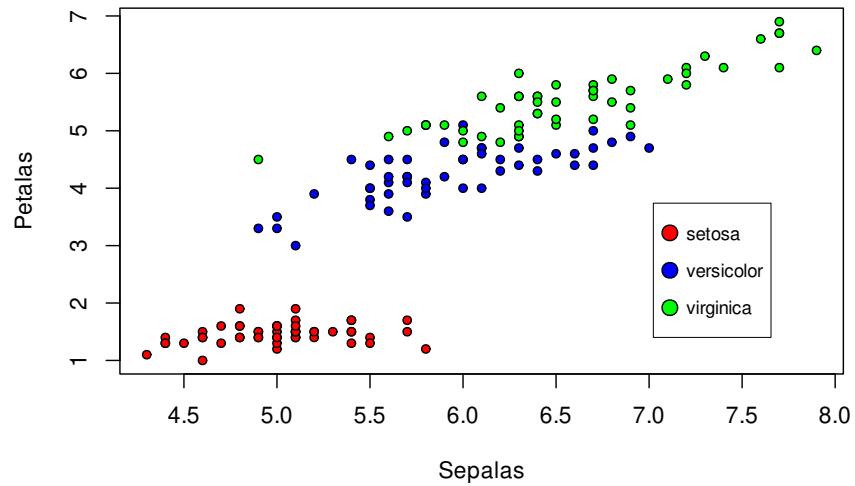
## [1] "setosa"      "versicolor"   "virginica"

ascores <- c("red", "blue", "green")[as.numeric(iris$Species)]
# plota a figura
plot(Sepalas, Petalas, pch = 21, bg = ascores, cex = 0.8)

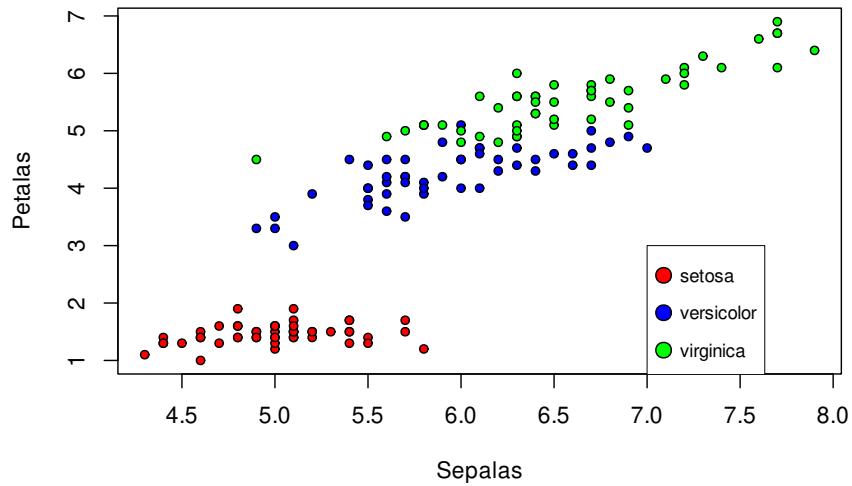
# vamos colocar uma legenda no canto direito inferior
# definimos o que vai na legenda
cores <- c("red", "blue", "green")
texto <- levels(iris$Species)
legend("bottomright", legend = texto, pt.bg = cores, pch = 21)
```



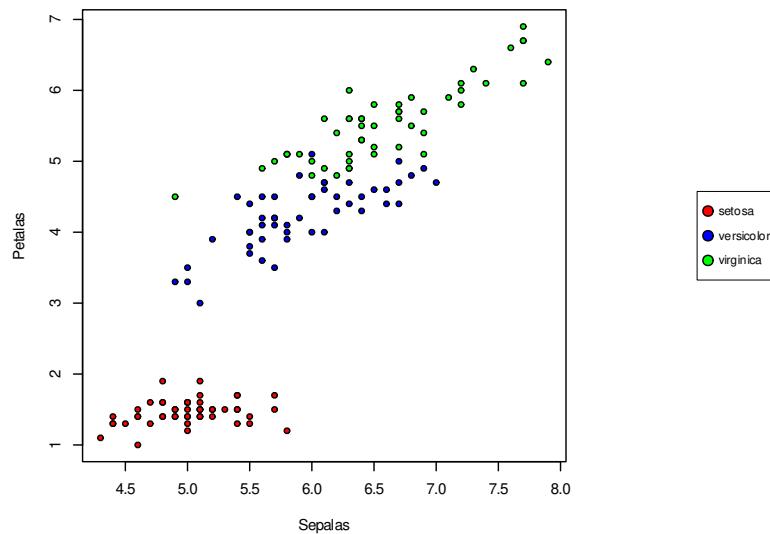
```
# ficou apertado né...
# aumentar o espacamento, colocar mais para dentro e tirar o box, diminuir o texto, aumentar o
# plota a figura
plot(Sepalas, Petalas, pch = 21, bg = ascores, cex = 0.8)
legend("bottomright", legend = texto, pt.bg = cores, pch = 21, y.intersp = 1.7, inset = 0.1, b
```



```
# podemos colocar legendas também pela coordenada
plot(Sepalas, Petalas, pch = 21, bg = ascores, cex = 0.8)
legend(x = 7, y = 3, legend = texto, pt.bg = cores, pch = 21, y.intersp = 1.7, inset = 0.1, b
```



```
# ou colocar fora da figura
# neste caso, primeiro dividimos o dispositivo em tres partes e plotamos a figura em 2 partes
layout(matrix(c(1, 1, 2), nrow = 1, ncol = 3))
plot(Sepalas, Petalas, pch = 21, bg = ascores, cex = 0.8)
# usamos a função plot.new() para mudar de parte no dispositivo cirando um plot vazio
plot.new() # nao vai ver nada acontecendo
# agora plotamos a legenda nesse espaço
legend("left", legend = texto, pt.bg = cores, pch = 21, y.intersp = 1.7, inset = 0.1, box.lwd
```



5.5.2 `points()`

Esta função genérica serve para adicionar pontos, linhas, símbolos etc, segundo coordenadas x ou x+y em um gráfico aberto por uma função de alto nível.

```
layout(1) # restaurando para 1 se fez o script acima  
# vamos usar um conjunto de dados de crescimento de laranja  
data("Orange")  
colnames(Orange) # número da árvore, idade, circunferencia nessa idade
```

```
## [1] "Tree"           "age"            "circumference"
```

```
head(Orange)
```

Tree	age	circumference
1	118	30
1	484	58
1	664	87
1	1004	115
1	1231	120
1	1372	142

```
unique(Orange$Tree) # tem cinco árvores
```

```
## [1] 1 2 3 4 5
## Levels: 3 < 1 < 5 < 2 < 4
```

```
# vamos plotar idade por circunferencia
# uma linha por árvore
# cada linha de um tipo e cor diferente

# definir tipos de linhas e cores
ascores <- c("red", "blue", "green", "yellow", "purple")
linhatipos <- c("solid", "dashed", "dotted", "dotdash", "longdash")

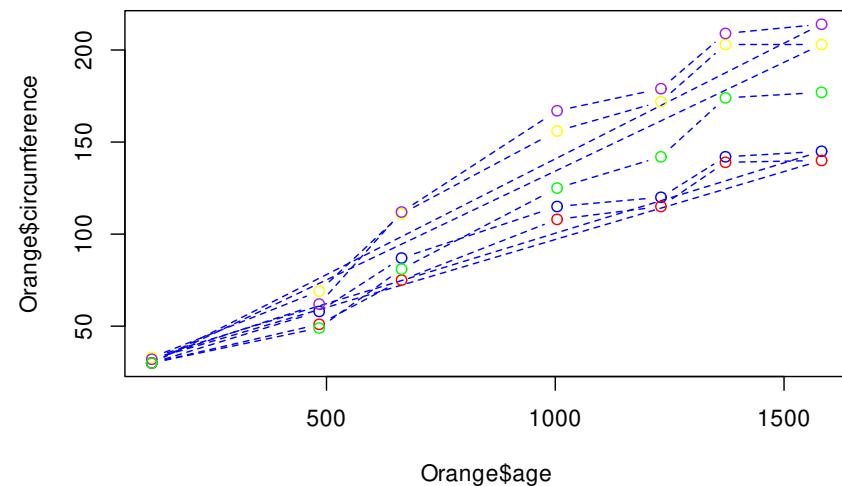
# uma cor por árvore
treeidx <- as.numeric(Orange$Tree)
cls <- ascores[treeidx]
cls
```

```
## [1] "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "yellow"
## [9] "yellow" "yellow" "yellow" "yellow" "yellow" "yellow" "red"   "red"
## [17] "red"    "red"    "red"    "red"    "red"    "purple" "purple" "purple"
## [25] "purple" "purple" "purple" "purple" "green"  "green"  "green"  "green"
## [33] "green"  "green"  "green"
```

```
# um tipo de linha por arvore  
tps <- linhatipos[treeidx]  
tps
```

```
## [1] "dashed"   "dashed"   "dashed"   "dashed"   "dashed"   "dashed"  
## [7] "dashed"   "dotdash"  "dotdash"  "dotdash"  "dotdash"  "dotdash"  
## [13] "dotdash"  "dotdash"  "solid"    "solid"    "solid"    "solid"  
## [19] "solid"    "solid"    "solid"    "longdash" "longdash" "longdash"  
## [25] "longdash" "longdash" "longdash" "longdash" "dotted"   "dotted"  
## [31] "dotted"   "dotted"   "dotted"   "dotted"   "dotted"   "dotted"
```

```
# fazemos uma figura  
plot(Orange$age, Orange$circumference, lwd = 1, type = "b", col = cls, lty = tps)
```



```
# note que todas as linhas sairam com primeira cor e com o primeiro tipo  
cls[1]
```

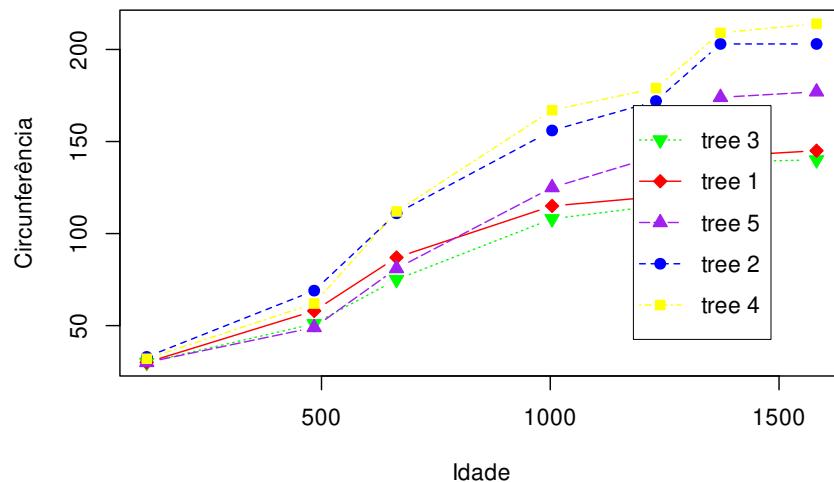
```
## [1] "blue"
```

```
tps[1]
```

```
## [1] "dashed"
```

```
# não dá para fazer assim, porque não especificamos como os pontos se ligam
```

```
# podemos no entanto colocar linha por linha  
# desenhamos o gráfico com uma função de alto nível com os dados, mas vazia  
plot(Orange$age, Orange$circumference, type = "n", xlab = "Idade", ylab = "Circunferência")  
# adicionamo para cada árvore os pontos e as linhas  
arvs <- as.numeric(levels(Orange$Tree))  
pts <- arvs + 20 # simbolos para pontos  
i <- 1  
for (i in 1:length(arvs)) { # para cada árvore  
  # filtra os dados  
  dd <- Orange[Orange$Tree == arvs[i], ]  
  # adiciona no gráfico aberto a linha e os pontos e uma vez (type='b', veja o argumento type)  
  points(dd$age, dd$circumference, type = "b", pch = pts[arvs[i]], lty = linhatipos[arvs[i]],  
}  
  
# vamos adicionar uma legenda  
txt <- paste("tree", arvs)  
legend("bottomright", inset = 0.1, box.lwd = 0, legend = txt, pch = pts[arvs], pt.bg = ascores
```



5.5.3 `text()` e `mtext()`

`text()` e `mtext()`- funções para plotar textos (≥ 1 caractere de comprimento) sobre uma figura de alto nível. `text()` plota dentro da área da figura através de coordendas, `mtext()` plota fora da área da figura com indicação do lado (side) e da distância da linha dos eixos (line).

```
?text # veja o help dessa função
# vamos usar iris novamente
```

```
Sepalas <- iris$Sepal.Length
Petalas <- iris$Petal.Length
nlevels(iris$Species) # contém três espécies
```

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

```
levels(iris$Species)

## [1] "setosa"      "versicolor"   "virginica"

ascores <- c("red", "blue", "green")[as.numeric(iris$Species)]

# vamos dar um código de texto para cada especie (suas primeiras letras maiusculo)
levels(iris$Species)

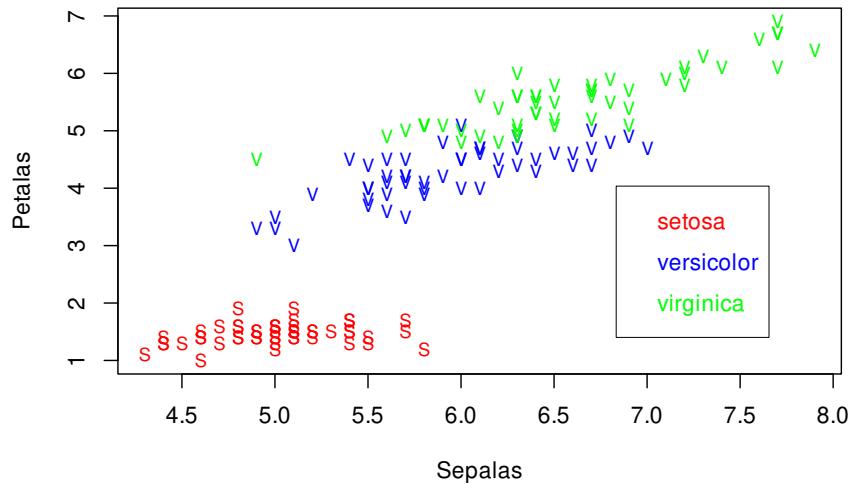
## [1] "setosa"      "versicolor"   "virginica"

spp <- toupper(substr(iris$Species, 1, 2))
spp

## [1] "SE" "SE"
## [16] "SE" "SE"
## [31] "SE" "SE"
## [46] "SE" "SE" "SE" "SE" "SE" "VE" "VE" "VE" "VE" "VE" "VE" "VE" "VE" "VE"
## [61] "VE" "VE"
## [76] "VE" "VE"
## [91] "VE" "VE" "VE" "VE" "VE" "VE" "VE" "VE" "VE" "VI" "VI" "VI" "VI" "VI"
## [106] "VI" "VI"
## [121] "VI" "VI"
## [136] "VI" "VI"
```



```
# plota a figura, mas ao inves de simbolos adicionamos text
# poderiamos fazer isso se spp fosse apenas 1 caractere
plot(Sepalas, Petalas, pch = spp, col = ascores, cex = 0.8)
legend("bottomright", legend = levels(iris$Species), text.col = c("red", "blue", "green"), ins...
```



```
# mas note que versicolor e virginica não se diferenciam, pois pch plota apenas 1 caractere que

# podemos, no entanto, usar a função text para isso
plot(Sepalas, Petalas, type = "n")
text(Sepalas, Petalas, labels = spp, col = ascores, cex = 0.8)
txt <- paste("Iris", levels(iris$Species))
legend("bottomright", legend = txt, text.col = c("red", "blue", "green"), inset = 0.1, box.lwd = 1)

# posso usar essa função para adicionar qualquer texto, em qualquer coordenada
text(4.5, 6, labels = "Exemplo Iris", adj = 0, col = "yellow")
# note o argumento adj que define o ajuste do texto em relação à coordenada x y especificada
# apenas o x
text(x = 4.5, y = 6, labels = "Exemplo Iris", adj = 1, col = "black")

# o x e o y do ajuste (dois valores em adj)
text(x = 4.5, y = 6, labels = "Exemplo Iris", adj = c(0.3, 1), col = "purple")

# a função mtext plot for da área das coordenadas da figura
# lado superior
```

```

mtext(side = 3, text = "A.", line = 1, col = "red", font = 2, cex = 2)
# lado superior esquerdo

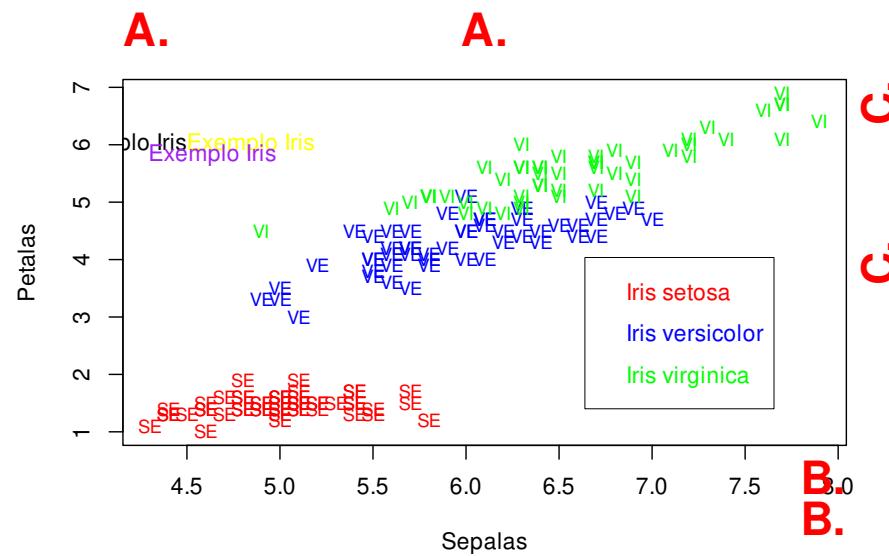
mtext(side = 3, text = "A.", line = 1, col = "red", font = 2, cex = 2, adj = 0)

# lado inferior direito
mtext(side = 1, text = "B.", line = 1, col = "red", font = 2, cex = 2, adj = 1)
# mais para baixo
mtext(side = 1, text = "B.", line = 2.5, col = "red", font = 2, cex = 2, adj = 1)

# lado direito superior
mtext(side = 4, text = "C.", line = 1, col = "red", font = 2, cex = 2, adj = 1)

# lado direito no meio
mtext(side = 4, text = "C.", line = 1, col = "red", font = 2, cex = 2, adj = 0.5)

```



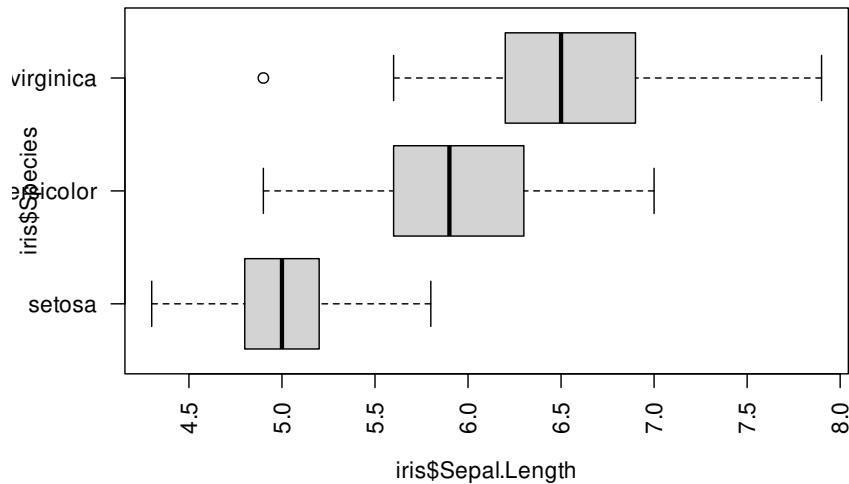
5.5.4 axis()

A função `axis()` adiciona eixos individualmente. É importante quando desejamos combinar dois gráficos em um só.

```
?axis
```

```
# algumas vezes queremos adicionar eixos, ou formatar o eixo de uma forma individualizada
# por exemplo, na seguinte figura, queremos colocar o nome das espécies no eixo y em posição
plot(iris$Sepal.Length ~ iris$Species, horizontal = TRUE)

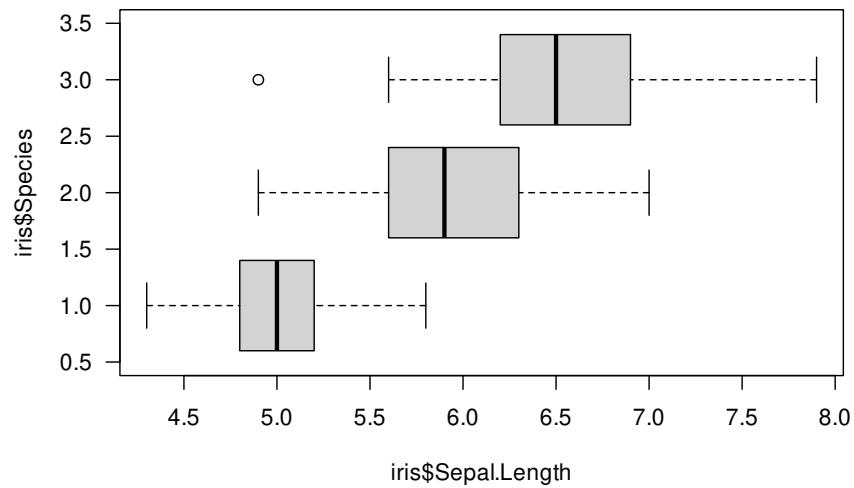
# o argumento para isso é par(las=2)
# se mudar isso eu mudo para todos os eixos
par(las = 2)
plot(iris$Sepal.Length ~ iris$Species, horizontal = TRUE)
```



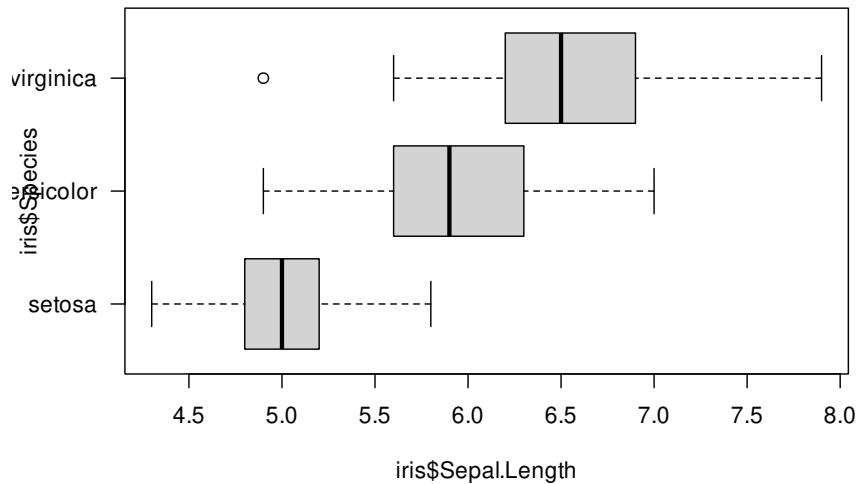
```
# mas eu quero apenas o eixo y
par(las = 1) # voltamos ao padrão 1

# plotamos a figura sem esse eixo
```

```
plot(iris$Sepal.Length ~ iris$Species, horizontal = TRUE, yaxt = "n")
# adicionamos o eixo y na horizontal
axis(side = 2, las = 2)
```

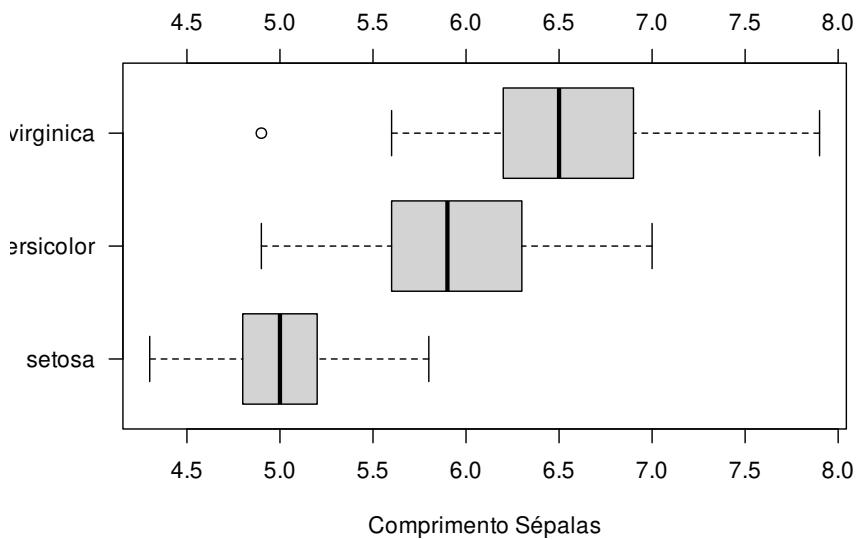


```
# mas ele plotou números não é isso que eu quero
plot(iris$Sepal.Length ~ iris$Species, horizontal = TRUE, yaxt = "n")
# entao dizemos onde e o que queremos desenhar
axis(side = 2, at = 1:3, labels = levels(iris$Species))
```



```
# vamos mudar o titulo do eixo da variavel dependente (y) e apagar das especies
plot(iris$Sepal.Length ~ iris$Species, horizontal = TRUE, yaxt = "n", ylab = "Comprimento Sépala")
# entao dizemos onde e o que queremos desenhar
axis(side = 2, at = 1:3, labels = levels(iris$Species))

# também posso adicionar eixos em outras posicoes
axis(side = 3)
```



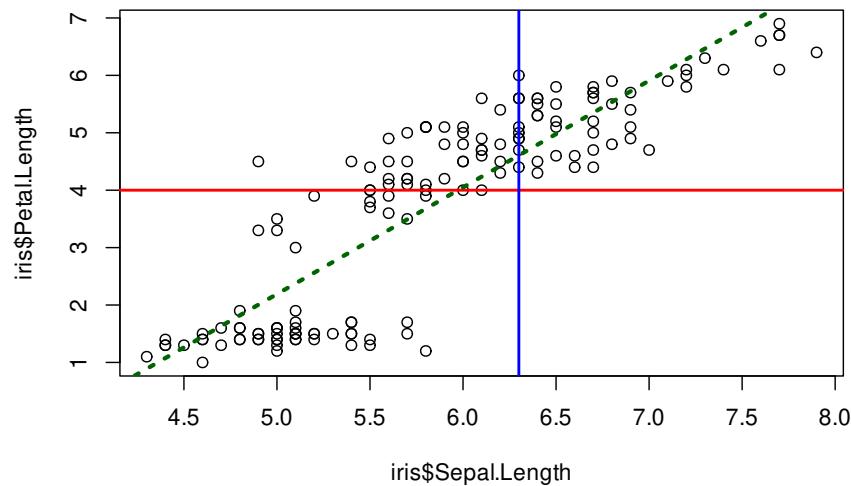
5.5.5 abline()

Esta função plota uma linha reta sobre um gráfico, ou uma linha de ajuste de uma correlação ou regressão.

```
# linha sobre o eixo
?abline # permite colocar linha em regressões ou linhas retas simples
```

```
plot(iris$Sepal.Length, iris$Petal.Length)
# um linha horizontal
abline(h = 4, col = "red", lwd = 2)
# uma linha vertical
abline(v = 6.3, col = "blue", lwd = 2)

# uma linha de uma regressão
rg <- lm(iris$Petal.Length ~ iris$Sepal.Length) # faço a regressão entre essas variáveis
abline(coef(rg), col = "darkgreen", lwd = 3, lty = "dotted") # plot pelos coeficientes
```



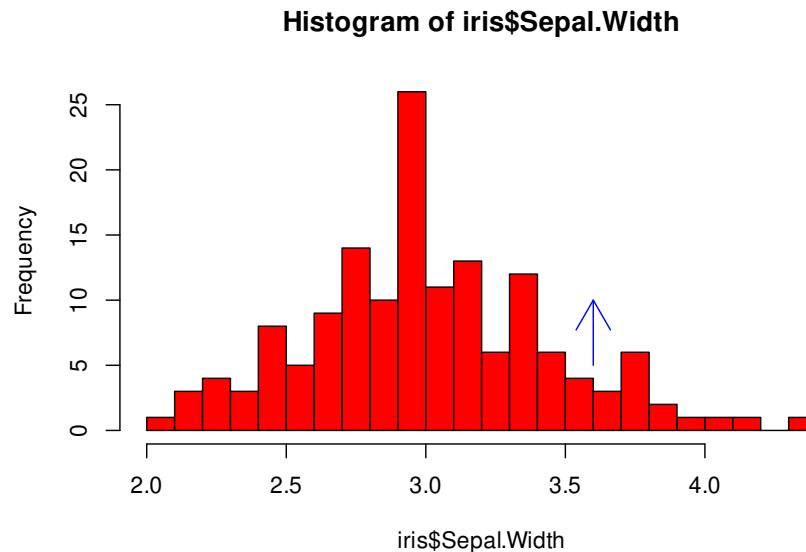
5.5.6 `arrows()`, `rect()`, `polygon()` e `segments()`

Estas funções plotam flechas, retângulos, polígonos, ou segmentos de linhas sobre sobre gráficos, respectivamente.

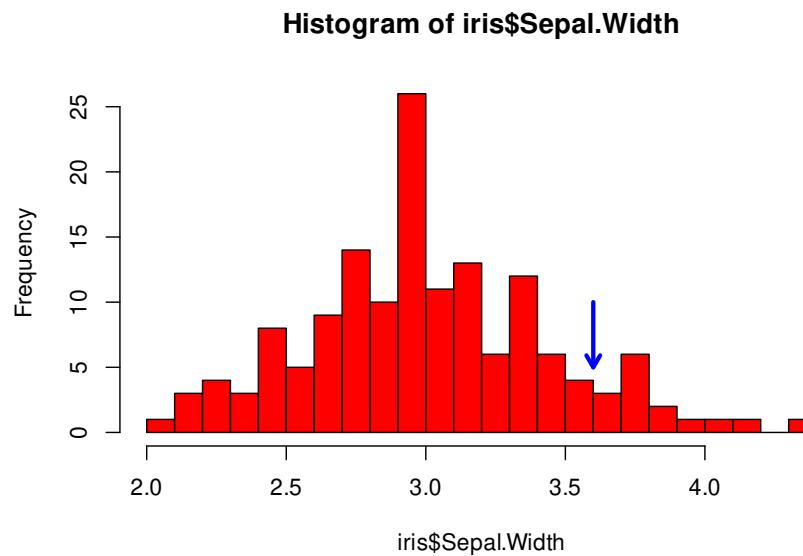
5.5.6.1 Flechas

```
?arrows # veja o help
```

```
# vamos criar um histograma
hist(iris$Sepal.Width, breaks = 20, col = "red")
# adicionar uma seta
arrows(x0 = 3.6, y0 = 5, x1 = 3.6, y1 = 10, lty = "solid", col = "blue")
```



```
# invertendo y0 e y1, diminuindo o tamanho da seta e colocando mais grosso
hist(iris$Sepal.Width, breaks = 20, col = "red")
arrows(x0 = 3.6, y0 = 10, x1 = 3.6, y1 = 5, lty = "solid", lwd = 3, col = "blue", length = 0.1)
```

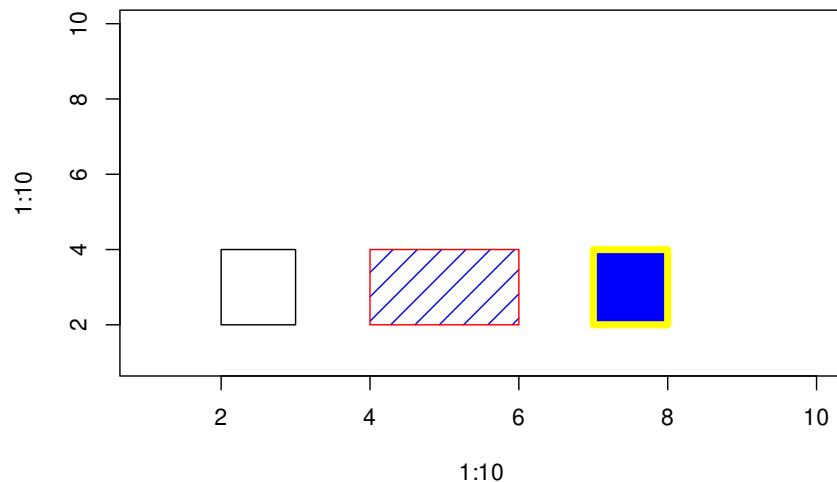


5.5.6.2 Retângulos

```
?rect # veja o help
```

```
plot(1:10, 1:10, type = "n")
rect(xleft = 2, ybottom = 2, xright = 3, ytop = 4)
# com preenchimento
rect(xleft = 4, ybottom = 2, xright = 6, ytop = 4, density = 8, border = "red", col = "blue")

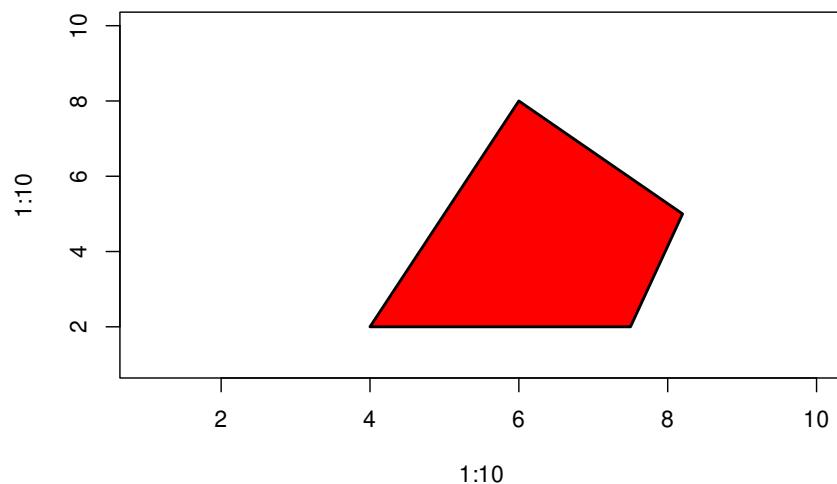
# com preenchimento total (density tem valor negativo (um qualquer))
rect(xleft = 7, ybottom = 2, xright = 8, ytop = 4, density = -1, border = "yellow", col = "blue")
```



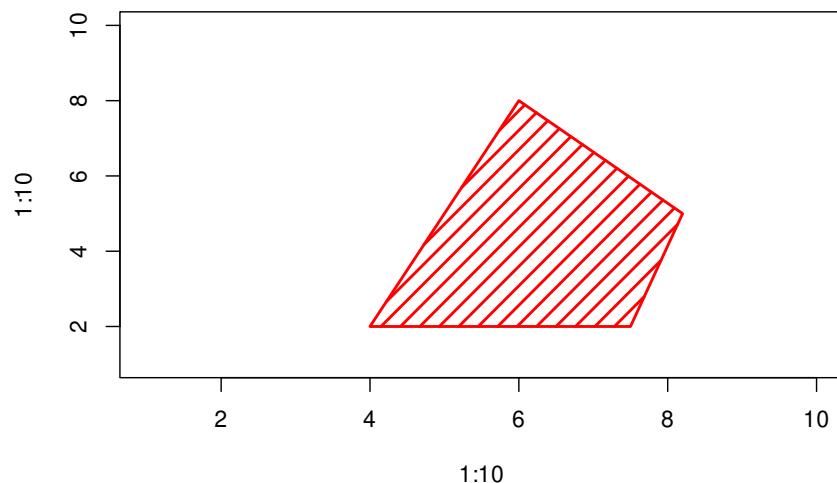
5.5.6.3 Polígonos

```
?polygon # veja o help
```

```
plot(1:10, 1:10, type = "n")
polygon(x = c(4, 7.5, 8.2, 6, 4), y = c(2, 2, 5, 8, 2), lwd = 2, col = "red")
```



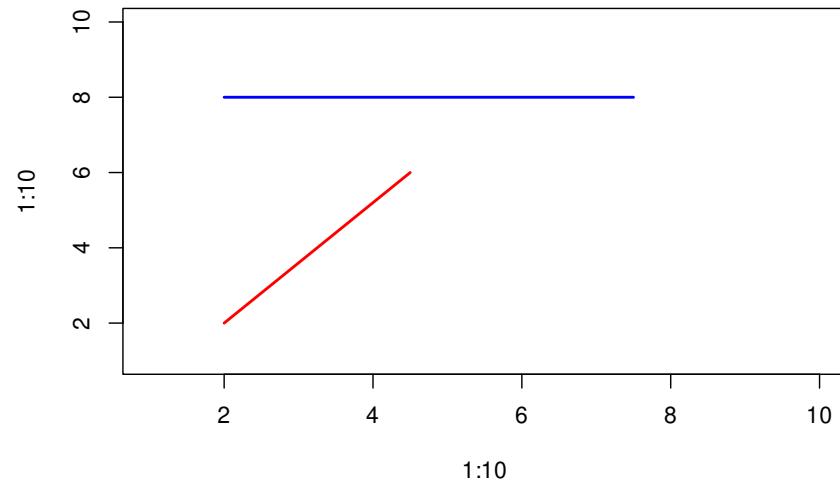
```
plot(1:10, 1:10, type = "n")
polygon(x = c(4, 7.5, 8.2, 6, 4), y = c(2, 2, 5, 8, 2), lwd = 2, col = "red", density = 10)
```



5.5.6.4 Segmentos de linhas

```
?segments
```

```
plot(1:10, 1:10, type = "n")
segments(x0 = 2, y0 = 2, x1 = 4.5, y1 = 6, lwd = 2, col = "red")
segments(x0 = 2, y0 = 8, x1 = 7.5, y1 = 8, lwd = 2, col = "blue")
```

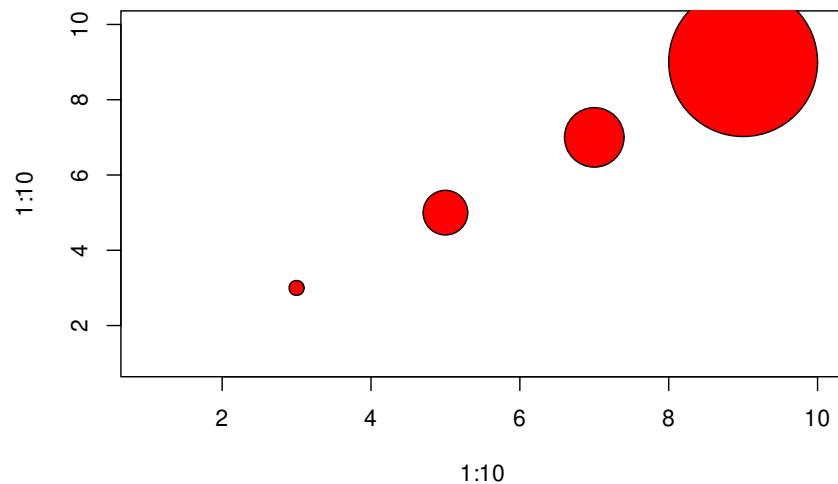


5.5.7 symbols()

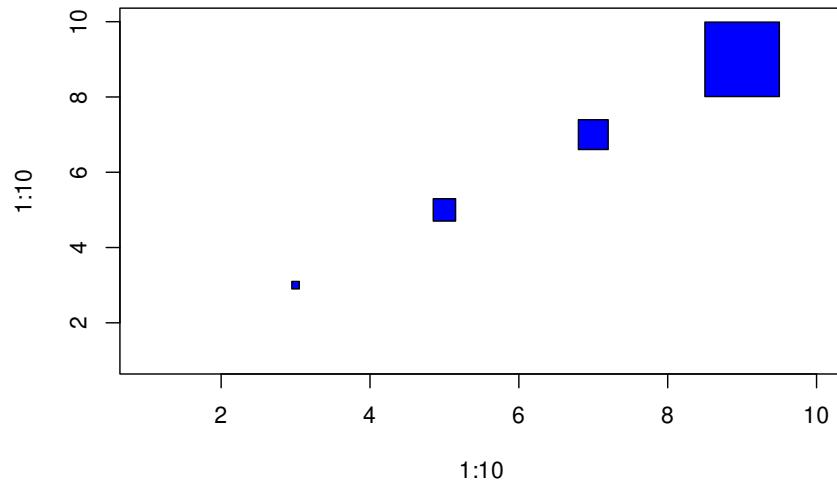
A função `symbols()` é uma de alto nível, mas com a adição do argumento `add=TRUE`, ela pode ser utilizada para adicionar sobre gráficos já plotados símbolos que expressam quantidades.

```
?symbols # ver o help
```

```
plot(1:10, 1:10, type = "n")
xx <- c(3, 5, 7, 9)
yy <- c(3, 5, 7, 9)
zz <- c(0.1, 0.3, 0.4, 1) # tamanho em unidade gráfica
symbols(xx, yy, circles = zz, bg = "red", add = T, inches = FALSE)
```

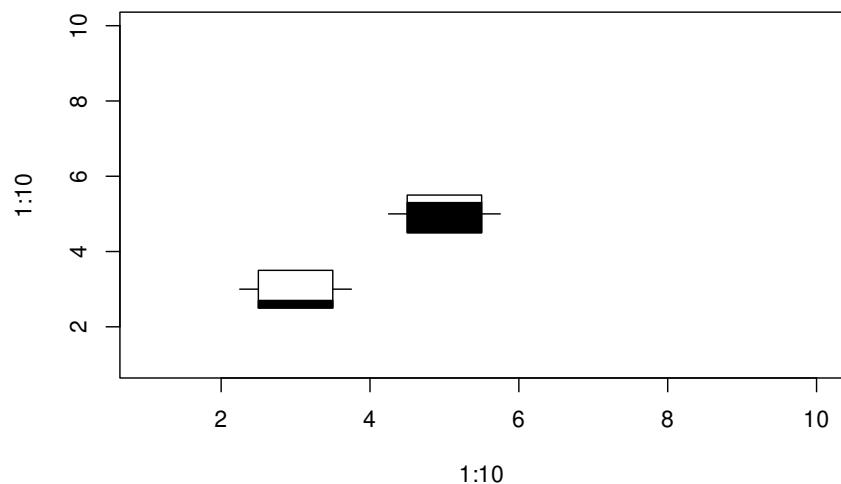


```
plot(1:10, 1:10, type = "n")
symbols(xx, yy, squares = zz, bg = "blue", add = T, inches = FALSE)
```



```
# agora mais complexo, expressando proporção
plot(1:10, 1:10, type = "n")

# uma matrix que com tres colunas: largura do simbolo, altura do simbolo e proporcao que simbolo
# ocupa da largura e altura do simbolo
v1 <- matrix(c(1, 1, 0.2), nrow = 1, ncol = 3) # 20% cheio
# ll = rep(v1,length(xx))
symbols(xx[1], yy[1], thermometers = v1, add = T, inches = FALSE)
v2 <- matrix(c(1, 1, 0.8), nrow = 1, ncol = 3) # 80% cheio
# ll = rep(v1,length(xx))
symbols(xx[2], yy[2], thermometers = v2, add = T, inches = FALSE)
```



5.6 Funções gráficas interativas

5.6.1 locator()

Esta função extrai coordenadas de um gráfico.

```
?locator # veja o help
# fazemos um gráfico
?trees # se quiser entender esses dados
data("trees")
class(trees)
colnames(trees)
Circunferencia <- trees$Girth
Altura <- trees$Height
plot(Circunferencia, Altura, type = "p", pch = 21, bg = "red")
```

```

locator(n = 1)
# clique na figura dentro do gráfico
# veja que no console há uma lista com os valores das coordenadas x e y

# pode fazer a mesma coisa já salvando as coordendas num objeto. Colete 2 pontos agora, traçar
cds <- locator(n = 2, type = "l")
cds # dois valores para cada coordenada
# posso usar os valores obtidos, por exemplo para plotar uma flexa
x0 <- cds$x[1]
x1 <- cds$x[2]
y0 <- cds$y[1]
y1 <- cds$y[2]
arrows(x0, y0, x1, y1, lwd = 2, col = "red")

```

5.6.2 identify()

Esta função identifica os pontos de um gráfico.

```

?identify # veja o help

# fazemos um gráfico
data("trees")
Circunferencia <- trees$Girth
Altura <- trees$Height
# plotamos vazio
plot(Circunferencia, Altura, type = "n")
ss <- sample(1:nrow(trees), 1) # um valor de indice aleatorio
# um ponto qualquer em vermelho
points(Circunferencia[ss], Altura[ss], pch = 21, cex = 1.5, bg = "red")
# o resto dos pontos em branco
points(Circunferencia[-ss], Altura[-ss], pch = 21, cex = 1.5, bg = "white")
# identificar pontos. Execute a função e selecione no ponto vermelho
identify(Circunferencia, Altura, n = 1, tolerance = 1)
# o numero que aparece no gráfico é o indice

```

```
# portanto, deve ser igual ao valor de  
ss  
trees[ss, ]
```

5.7 Para saber mais:

(I). Vídeoaulas com conteúdos parciais deste capítulo:

- Dispositivos gráficos⁶;
- Parâmetros gráficos, parte I⁷;
- Parâmetros gráficos, parte II⁸;
- Parâmetros gráficos parte II⁹;
- Funções gráficas de alto nível¹⁰;
- Funções gráficas de baixo nível, parte I¹¹;
- Funções gráficas de baixo nível, parte II¹²;

⁶http://www.botanicaamazonica.wiki.br/labotam/lib/exe/fetch.php?media=bot89:precurso:1plot:video01_bot89-2020-04-09_18.22.40.mp4

⁷http://www.botanicaamazonica.wiki.br/labotam/lib/exe/fetch.php?media=bot89:precurso:3plot:video01_bot89-2020-04-13_08.47.11.mp4

⁸http://www.botanicaamazonica.wiki.br/labotam/lib/exe/fetch.php?media=bot89:precurso:3plot:video01_bot89-2020-04-13_09.17.45.mp4

⁹http://www.botanicaamazonica.wiki.br/labotam/lib/exe/fetch.php?media=bot89:precurso:4plot:video01_bot89-2020-04-13_10.15.19.mp4

¹⁰http://www.botanicaamazonica.wiki.br/labotam/lib/exe/fetch.php?media=bot89:precurso:2plot:video01_bot89-2020-04-13_11.26.19.mp4

¹¹http://www.botanicaamazonica.wiki.br/labotam/lib/exe/fetch.php?media=bot89:precurso:5plot:video01_bot89-2020-04-13_12.05.18.mp4

¹²http://www.botanicaamazonica.wiki.br/labotam/lib/exe/fetch.php?media=bot89:precurso:5plot:video01_bot89-2020-04-13_13.07.07.mp4

(2). R Gallery¹³ - página que tem exemplos de gráficos do R e os códigos correspondentes.

5.8 Exercícios

Esta série de exercícios exemplificam algumas funções que serão retomadas no tutorial de AED (Capítulos 11, 12, 13 e 14). Procure exercitar o que aprenderam até aqui. O conjunto de dados alunos2018.txt¹⁴ será utilizado na maioria dos exercícios listados abaixo:

- Resolva o exercício 302.02 Histogramas (frequência)¹⁵.
- Resolva o exercício 302.07 Gráficos com trechos selecionados do data.frame/matriz¹⁶.
- Resolva o exercício 302.04 Box-plots¹⁷.
- Resolva o exercício 302.05 Gráficos de dispersão¹⁸.
- Resolva o exercício 302.06 Matriz de dispersão¹⁹.
- Resolva o exercício Editando parâmetros gráficos²⁰.

¹³<http://www.r-graph-gallery.com/>

¹⁴<http://www.lage.ib.usp.br/notar/files/alunos2018.txt>

¹⁵<http://notar.ib.usp.br/exercicio/66>

¹⁶<http://notar.ib.usp.br/exercicio/71>

¹⁷<http://notar.ib.usp.br/exercicio/67>

¹⁸<http://notar.ib.usp.br/exercicio/69>

¹⁹<http://notar.ib.usp.br/exercicio/70>

²⁰<http://notar.ib.usp.br/exercicio/75>

6

Iteração e controle de fluxo

6.1 Funções da família `apply()`

Algumas funções da família `apply()` são muito úteis na manipulação de dados e descrição de dados. Essas funções são muito poderosas, porque permitem você fazer iterações de forma simples, ou seja, você pode aplicar uma função a vários objetos simultaneamente com funções dessa família. O que você faz com essas funções você também faz com as iterações que fazem uso da expressão `for() {}` (veja seção 6.3), mas essas funções simplificam e aceleram o processo.

6.1.1 Em uma matriz

A função `apply()` poderia ser traduzida como:

aplique uma função (FUN) a todas as linhas ou colunas (MARGIN) de uma matriz (X):

```
?apply # Veja o help  
# os argumentos dessa função são:  
# X = matrix  
# MAGRIN = 1 indica linha, 2 indica colunas  
# FUN = a função que você deseja aplicar
```

```
# ... ARGUMENTOS DESSA FUNCAO SE FOR O CASO

# TOTAIS MARGINAIS
# crie uma matriz
```

```
X <- matrix(1:36, nrow = 4, ncol = 9)
colnames(X) <- paste("col", 1:ncol(X))
rownames(X) <- paste("ln", 1:nrow(X))
head(X) # cabeça da matriz criada
```

	col 1	col 2	col 3	col 4	col 5	col 6	col 7	col 8	col 9
ln 1	1	5	9	13	17	21	25	29	33
ln 2	2	6	10	14	18	22	26	30	34
ln 3	3	7	11	15	19	23	27	31	35
ln 4	4	8	12	16	20	24	28	32	36

```
# calcule para todas as linhas:
# a soma dos valores
apply(X, MARGIN = 1, FUN = sum)
```

```
## ln 1 ln 2 ln 3 ln 4
## 153 162 171 180
```

```
# o valor máximo
apply(X, MARGIN = 1, FUN = max)
```

```
## ln 1 ln 2 ln 3 ln 4
## 33 34 35 36
```

```
# a média  
apply(X, MARGIN = 1, FUN = mean)
```

```
## ln 1 ln 2 ln 3 ln 4  
## 17 18 19 20
```

```
# o desvio padrão  
apply(X, MARGIN = 1, FUN = sd)
```

```
##      ln 1      ln 2      ln 3      ln 4  
## 10.95445 10.95445 10.95445 10.95445
```

```
# para todas as colunas  
# a soma dos valores  
apply(X, MARGIN = 2, FUN = sum)
```

```
## col 1 col 2 col 3 col 4 col 5 col 6 col 7 col 8 col 9  
## 10 26 42 58 74 90 106 122 138
```

```
# o valor máximo  
apply(X, MARGIN = 2, FUN = max)
```

```
## col 1 col 2 col 3 col 4 col 5 col 6 col 7 col 8 col 9  
## 4 8 12 16 20 24 28 32 36
```

```
# a média  
apply(X, MARGIN = 2, FUN = mean)
```

```
## col 1 col 2 col 3 col 4 col 5 col 6 col 7 col 8 col 9  
## 2.5 6.5 10.5 14.5 18.5 22.5 26.5 30.5 34.5
```

```
# o desvio padrão  
apply(X, MARGIN = 2, FUN = sd)
```

```
## col 1 col 2 col 3 col 4 col 5 col 6 col 7 col 8  
## 1.290994 1.290994 1.290994 1.290994 1.290994 1.290994 1.290994 1.290994  
## col 9  
## 1.290994
```

As funções `rowSums()`, `rowMeans()`, `colSums()`, ou `colMeans()` são equivalentes à função `apply()`. Elas simplificam o uso para somas (em inglês, *sum*) e médias (em inglês, *mean*) de linhas (em inglês, *rows*) e colunas (em inglês, *columns*). Se você conhece bem a função `apply()`, você pode fazer o que essas funções fazem e muito mais. Portanto, ao dominar a função `apply()`, você acaba por não precisar se preocupar em aprender essas funções mais específicas.

```
?rowSums
```

```
rowSums(X) # soma de cada linha
```

```
## ln 1 ln 2 ln 3 ln 4  
## 153 162 171 180
```

```
rowMeans(X) # media de cada linha
```

```
## ln 1 ln 2 ln 3 ln 4  
##   17   18   19   20
```

```
colSums(X) # soma de cada coluna
```

```
## col 1 col 2 col 3 col 4 col 5 col 6 col 7 col 8 col 9  
##   10    26    42    58    74    90   106   122   138
```

```
colMeans(X) # média de cada coluna
```

```
## col 1 col 2 col 3 col 4 col 5 col 6 col 7 col 8 col 9  
##   2.5   6.5  10.5  14.5  18.5  22.5  26.5  30.5  34.5
```

6.1.2 Em um vetor ou lista

A função `lapply()` aplica uma função `FUN` para cada elemento de um vetor ou de uma lista, e retorna um objeto de classe `list`.

```
# muito simples, imprime algo linha por linha  
ll <- lapply(LETTERS, print)
```

```
## [1] "A"  
## [1] "B"  
## [1] "C"  
## [1] "D"  
## [1] "E"  
## [1] "F"
```

```
## [1] "G"  
## [1] "H"  
## [1] "I"  
## [1] "J"  
## [1] "K"  
## [1] "L"  
## [1] "M"  
## [1] "N"  
## [1] "O"  
## [1] "P"  
## [1] "Q"  
## [1] "R"  
## [1] "S"  
## [1] "T"  
## [1] "U"  
## [1] "V"  
## [1] "W"  
## [1] "X"  
## [1] "Y"  
## [1] "Z"
```

```
class(ll)
```

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

```
str(ll)
```

```
## List of 26  
## $ : chr "A"  
## $ : chr "B"  
## $ : chr "C"  
## $ : chr "D"
```

```
## $ : chr "E"  
## $ : chr "F"  
## $ : chr "G"  
## $ : chr "H"  
## $ : chr "I"  
## $ : chr "J"  
## $ : chr "K"  
## $ : chr "L"  
## $ : chr "M"  
## $ : chr "N"  
## $ : chr "O"  
## $ : chr "P"  
## $ : chr "Q"  
## $ : chr "R"  
## $ : chr "S"  
## $ : chr "T"  
## $ : chr "U"  
## $ : chr "V"  
## $ : chr "W"  
## $ : chr "X"  
## $ : chr "Y"  
## $ : chr "Z"
```

```
# suponha três vetores de tamanhos diferentes  
v1 <- sample(1:1000, 50)  
length(v1)
```

```
## [1] 50
```

```
v2 <- sample(1:100, 30)  
length(v2)
```

```
## [1] 30
```

```
v3 <- sample(1000:2000, 90)
length(v3)
```

```
## [1] 90
```

```
# imagina que isso esteja numa lista
ml <- list(v1, v2, v3)
class(ml)
```

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

```
length(ml)
```

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

```
# posso usar lapply para calcular a media desses vetores
lapply(ml, mean) # note que retorna uma lista
```

```
## [[1]]
## [1] 523.4
##
## [[2]]
## [1] 46.3
##
## [[3]]
## [1] 1534.989
```

```
lt <- lapply(ml, mean) # podemos guardar  
lt <- as.vector(lt, mode = "numeric") # e transformar num vetor. MODE neste caso é fundamental  
lt  
  
## [1] 523.400 46.300 1534.989
```

6.1.3 Por categoria de um fator

A função `tapply()` aplica uma função `FUN` em uma coluna numérica individualizando os resultados para cada categoria de um determinado fator.

```
# vamos usar os dados de Iris novamente  
?iris  
  
# calculando o comprimento médio de sepálas pelas categorias de um fator (especies)  
class(iris$Sepal.Length) # variável numérica  
  
## [1] "numeric"  
  
class(iris$Species) # fator com categorias que correspondem a espécies  
  
## [1] "factor"  
  
tapply(iris$Sepal.Length, iris$Species, FUN = mean)  
  
##      setosa versicolor virginica  
##      5.006     5.936     6.588
```

```
# calculando o comprimento máximo por especie
tapply(iris$Sepal.Length, iris$Species, FUN = max)
```

```
##      setosa versicolor virginica
##      5.8       7.0       7.9
```

```
# a amplitude de variacao
tapply(iris$Sepal.Length, iris$Species, FUN = range)
```

```
## $setosa
## [1] 4.3 5.8
##
## $versicolor
## [1] 4.9 7.0
##
## $virginica
## [1] 4.9 7.9
```

```
# note que essa função sempre retorna um array (que é uma lista de fato, neste caso)
tm <- tapply(iris$Sepal.Length, iris$Species, FUN = min)
class(tm)
```

```
## [1] "array"
```

```
names(tm)
```

```
## [1] "setosa"      "versicolor"   "virginica"
```

```
tm[["setosa"]] # indexadores de lista  
  
## [1] 4.3  
  
tr <- tapply(iris$Sepal.Length, iris$Species, FUN = range)  
class(tr)  
  
## [1] "array"
```

```
names(tr)
```

```
## [1] "setosa"      "versicolor"   "virginica"
```

```
tr[["setosa"]]
```

```
## [1] 4.3 5.8
```

6.2 Condicionais

Condicionais são expressões que permitem a um programa a tomada de decisões. Vamos tratar aqui das condicionais `if ()`, `if () else {}`, e `ifelse ()` (veja a seção [Para saber mais](#) para mais informações).

6.2.1 Condicional `if ()`

A expressão `if ()` avalia um vetor atômico (ou de índice 1) lógico e executa o que estiver entre {} se o valor do vetor for verdadeiro (TRUE).

A estrutura básica de um `if ()` é:

```
if (condicao) {  
  ação a ser realizada caso a condição seja VERDADEIRA  
}
```

Dentro do par de parênteses, deve haver uma **condição**. Condições em R são feitas com operadores lógicos: ==, !=, >, < etc (veja a seção @ref(#vetor-operador-logico) para relembrar; veja também a seção **Para saber mais** pois apresenta *links* para vídeoaulas importantes).

6.2.1.1 Exemplo 01

```
perdiz_estuda_breu <- TRUE  
perdiz_estuda_grama <- FALSE  
perdiz_estuda_breu
```

```
## [1] TRUE
```

```
perdiz_estuda_grama
```

```
## [1] FALSE
```

```
if (perdiz_estuda_breu) {  
    print("Breu pode ser Protium, Dacryodes, Trattinnickia, e é da família Burseraceae")  
}
```

```
## [1] "Breu pode ser Protium, Dacryodes, Trattinnickia, e é da família Burseraceae"
```

```
if (perdiz_estuda_grama) {  
    print("Isso não vai dar imprimir")  
}
```

Reparam na condição que deve ser satisfeita dentro dos parênteses: TRUE, para efetuar a ação; se FALSE, não executa a função:

```
if (TRUE) {  
    print("Eu executo!")  
}
```

```
## [1] "Eu executo!"
```

```
if (FALSE) {  
    print("Eu não executo")  
}
```

6.2.2 Condicional `if ()` com `o else`

A condicional `if ()` pode ser expandida para `if () {} else {}`: execute em {} se `if ()` for VERDADEIRO, caso contrário (== `else`), execute o que estiver entre o segundo par {}.

6.2.2.1 Exemplo 01

```
meunumero <- 77
meunumero
```

```
## [1] 77
```

```
if (meunumero == 76) {
  print("Meu número é o 3")
} else {

  # meu_else <- paste0("Mas pode ser o ", meunumero)
  # print(meu_else)
  # 1+1
  print(paste("Se eu somar o meu número ", meunumero, "com 3, eu vou obter ", meunumero + 3))
}
```

```
## [1] "Se eu somar o meu número 77 com 3, eu vou obter 80"
```

6.2.2.2 Exemplo 02

```
familia <- c("Burseraceae", "Solanaceae", "Sapindaceae", "Rubiaceae")
clado <- c("Malvids", "Lamiids", "Malvids", "Lamiids")
apg <- data.frame(familia = familia, clado = clado, stringsAsFactors = FALSE)
apg
```

familia	clado
Burseraceae	Malvids
Solanaceae	Lamiids
Sapindaceae	Malvids
Rubiaceae	Lamiids

```
str(apg)
```

```
## 'data.frame':    4 obs. of  2 variables:  
## $ familia: chr "Burseraceae" "Solanaceae" "Sapindaceae" "Rubiaceae"  
## $ clado  : chr "Malvids" "Lamiids" "Malvids" "Lamiids"
```

```
dim(apg)
```

```
## [1] 4 2
```

```
malvids <- c("Burseraceae", "Sapindaceae")
```

```
meunumero <- 4  
apg$familia[meunumero]
```

```
## [1] "Rubiaceae"
```

```
apg$familia[meunumero] %in% malvids
```

```
## [1] FALSE
```

```
if (apg$familia[meunumero] %in% malvids) {
  malv_fam <- paste(malvids, collapse = " e ")
  malv_fam
  clado <- "Malvids"
  frase <- paste(malv_fam, clado, sep = " pertencem ao clado das ")
  frase
  print(frase)
} else {
  paste("A família", apg$familia[meunumero], "pertence ao clado das", apg$clado[meunumero])
}
```

```
## [1] "A família Rubiaceae pertence ao clado das Lamiids"
```

6.2.3 Condicional **ifelse()**

```
ifelse(condicao, executa se VERDADEIRO, executa se FALSO)
```

```
set.seed(333)
familia
```

```
## [1] "Burseraceae" "Solanaceae"  "Sapindaceae" "Rubiaceae"
```

```
familias <- sample(familia, 100, replace = TRUE)
familias
```

```
##  [1] "Solanaceae" "Burseraceae" "Sapindaceae" "Solanaceae" "Sapindaceae"
##  [6] "Rubiaceae"   "Solanaceae"  "Solanaceae"  "Sapindaceae" "Rubiaceae"
## [11] "Sapindaceae" "Sapindaceae" "Rubiaceae"   "Sapindaceae" "Burseraceae"
## [16] "Sapindaceae" "Sapindaceae" "Solanaceae"  "Sapindaceae" "Solanaceae"
## [21] "Burseraceae" "Solanaceae"  "Rubiaceae"   "Solanaceae" "Sapindaceae"
```

```
## [26] "Burseraceae" "Solanaceae" "Sapindaceae" "Solanaceae" "Rubiaceae"
## [31] "Solanaceae" "Sapindaceae" "Burseraceae" "Solanaceae" "Rubiaceae"
## [36] "Burseraceae" "Solanaceae" "Solanaceae" "Rubiaceae" "Rubiaceae"
## [41] "Burseraceae" "Burseraceae" "Burseraceae" "Burseraceae" "Sapindaceae"
## [46] "Burseraceae" "Solanaceae" "Burseraceae" "Rubiaceae" "Burseraceae"
## [51] "Sapindaceae" "Burseraceae" "Sapindaceae" "Burseraceae" "Sapindaceae"
## [56] "Rubiaceae" "Rubiaceae" "Rubiaceae" "Rubiaceae" "Solanaceae"
## [61] "Burseraceae" "Solanaceae" "Burseraceae" "Sapindaceae" "Burseraceae"
## [66] "Burseraceae" "Rubiaceae" "Sapindaceae" "Solanaceae" "Solanaceae"
## [71] "Rubiaceae" "Solanaceae" "Sapindaceae" "Burseraceae" "Rubiaceae"
## [76] "Solanaceae" "Solanaceae" "Burseraceae" "Burseraceae" "Sapindaceae"
## [81] "Solanaceae" "Sapindaceae" "Sapindaceae" "Burseraceae" "Sapindaceae"
## [86] "Rubiaceae" "Burseraceae" "Sapindaceae" "Solanaceae" "Rubiaceae"
## [91] "Solanaceae" "Solanaceae" "Burseraceae" "Solanaceae" "Burseraceae"
## [96] "Burseraceae" "Solanaceae" "Solanaceae" "Sapindaceae" "Solanaceae"
```

```
table(familias)
```

Burseraceae	Rubiaceae	Sapindaceae	Solanaceae
27	18	25	30

familias

```
## [1] "Solanaceae" "Burseraceae" "Sapindaceae" "Solanaceae" "Sapindaceae"
## [6] "Rubiaceae" "Solanaceae" "Solanaceae" "Sapindaceae" "Rubiaceae"
## [11] "Sapindaceae" "Sapindaceae" "Rubiaceae" "Sapindaceae" "Burseraceae"
## [16] "Sapindaceae" "Sapindaceae" "Solanaceae" "Sapindaceae" "Solanaceae"
## [21] "Burseraceae" "Solanaceae" "Rubiaceae" "Solanaceae" "Sapindaceae"
## [26] "Burseraceae" "Solanaceae" "Sapindaceae" "Solanaceae" "Rubiaceae"
## [31] "Solanaceae" "Sapindaceae" "Burseraceae" "Solanaceae" "Rubiaceae"
## [36] "Burseraceae" "Solanaceae" "Solanaceae" "Rubiaceae" "Rubiaceae"
## [41] "Burseraceae" "Burseraceae" "Burseraceae" "Burseraceae" "Sapindaceae"
## [46] "Burseraceae" "Solanaceae" "Burseraceae" "Rubiaceae" "Burseraceae"
## [51] "Sapindaceae" "Burseraceae" "Sapindaceae" "Burseraceae" "Sapindaceae"
```

```
## [56] "Rubiaceae" "Rubiaceae" "Rubiaceae" "Rubiaceae" "Solanaceae"
## [61] "Burseraceae" "Solanaceae" "Burseraceae" "Sapindaceae" "Burseraceae"
## [66] "Burseraceae" "Rubiaceae" "Sapindaceae" "Solanaceae" "Solanaceae"
## [71] "Rubiaceae" "Solanaceae" "Sapindaceae" "Burseraceae" "Rubiaceae"
## [76] "Solanaceae" "Solanaceae" "Burseraceae" "Burseraceae" "Sapindaceae"
## [81] "Solanaceae" "Sapindaceae" "Sapindaceae" "Burseraceae" "Sapindaceae"
## [86] "Rubiaceae" "Burseraceae" "Sapindaceae" "Solanaceae" "Rubiaceae"
## [91] "Solanaceae" "Solanaceae" "Burseraceae" "Solanaceae" "Burseraceae"
## [96] "Burseraceae" "Solanaceae" "Solanaceae" "Sapindaceae" "Solanaceae"
```

```
ifelse(familias == "Burseraceae", TRUE, FALSE)
```

```
## [1] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [13] FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE
## [25] FALSE TRUE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE TRUE
## [37] FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE FALSE TRUE FALSE TRUE
## [49] FALSE TRUE FALSE TRUE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## [61] TRUE FALSE TRUE FALSE TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## [73] FALSE TRUE FALSE FALSE FALSE TRUE TRUE FALSE FALSE FALSE FALSE TRUE
## [85] FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE TRUE FALSE TRUE TRUE
## [97] FALSE FALSE FALSE FALSE
```

```
ifelse(familias == "Sapindaceae", "Sapindaceae é a família do guaraná", "Não é Sapindaceae")
```

```
## [1] "Não é Sapindaceae" "Não é Sapindaceae"
## [3] "Sapindaceae é a família do guaraná" "Não é Sapindaceae"
## [5] "Sapindaceae é a família do guaraná" "Não é Sapindaceae"
## [7] "Não é Sapindaceae" "Não é Sapindaceae"
## [9] "Sapindaceae é a família do guaraná" "Não é Sapindaceae"
## [11] "Sapindaceae é a família do guaraná" "Sapindaceae é a família do guaraná"
## [13] "Não é Sapindaceae" "Sapindaceae é a família do guaraná"
## [15] "Não é Sapindaceae" "Sapindaceae é a família do guaraná"
```

```
## [17] "Sapindaceae é a família do guaraná" "Não é Sapindaceae"  
## [19] "Sapindaceae é a família do guaraná" "Não é Sapindaceae"  
## [21] "Não é Sapindaceae" "Não é Sapindaceae"  
## [23] "Não é Sapindaceae" "Não é Sapindaceae"  
## [25] "Sapindaceae é a família do guaraná" "Não é Sapindaceae"  
## [27] "Não é Sapindaceae" "Sapindaceae é a família do guaraná"  
## [29] "Não é Sapindaceae" "Não é Sapindaceae"  
## [31] "Não é Sapindaceae" "Sapindaceae é a família do guaraná"  
## [33] "Não é Sapindaceae" "Não é Sapindaceae"  
## [35] "Não é Sapindaceae" "Não é Sapindaceae"  
## [37] "Não é Sapindaceae" "Não é Sapindaceae"  
## [39] "Não é Sapindaceae" "Não é Sapindaceae"  
## [41] "Não é Sapindaceae" "Não é Sapindaceae"  
## [43] "Não é Sapindaceae" "Não é Sapindaceae"  
## [45] "Sapindaceae é a família do guaraná" "Não é Sapindaceae"  
## [47] "Não é Sapindaceae" "Não é Sapindaceae"  
## [49] "Não é Sapindaceae" "Não é Sapindaceae"  
## [51] "Sapindaceae é a família do guaraná" "Não é Sapindaceae"  
## [53] "Sapindaceae é a família do guaraná" "Não é Sapindaceae"  
## [55] "Sapindaceae é a família do guaraná" "Não é Sapindaceae"  
## [57] "Não é Sapindaceae" "Não é Sapindaceae"  
## [59] "Não é Sapindaceae" "Não é Sapindaceae"  
## [61] "Não é Sapindaceae" "Não é Sapindaceae"  
## [63] "Não é Sapindaceae" "Sapindaceae é a família do guaraná"  
## [65] "Não é Sapindaceae" "Não é Sapindaceae"  
## [67] "Não é Sapindaceae" "Sapindaceae é a família do guaraná"  
## [69] "Não é Sapindaceae" "Não é Sapindaceae"  
## [71] "Não é Sapindaceae" "Não é Sapindaceae"  
## [73] "Sapindaceae é a família do guaraná" "Não é Sapindaceae"  
## [75] "Não é Sapindaceae" "Não é Sapindaceae"  
## [77] "Não é Sapindaceae" "Não é Sapindaceae"  
## [79] "Não é Sapindaceae" "Sapindaceae é a família do guaraná"  
## [81] "Não é Sapindaceae" "Sapindaceae é a família do guaraná"  
## [83] "Sapindaceae é a família do guaraná" "Não é Sapindaceae"  
## [85] "Sapindaceae é a família do guaraná" "Não é Sapindaceae"  
## [87] "Não é Sapindaceae" "Sapindaceae é a família do guaraná"  
## [89] "Não é Sapindaceae" "Não é Sapindaceae"
```

```

## [91] "Não é Sapindaceae"           "Não é Sapindaceae"
## [93] "Não é Sapindaceae"           "Não é Sapindaceae"
## [95] "Não é Sapindaceae"           "Não é Sapindaceae"
## [97] "Não é Sapindaceae"           "Não é Sapindaceae"
## [99] "Sapindaceae é a família do guaraná" "Não é Sapindaceae"

```

familias[3]

```
## [1] "Sapindaceae"
```

familias[1:10]

```

## [1] "Solanaceae" "Burseraceae" "Sapindaceae" "Solanaceae" "Sapindaceae"
## [6] "Rubiaceae"   "Solanaceae"  "Solanaceae"  "Sapindaceae" "Rubiaceae"

```

```

ifelse(familias == "Sapindaceae", "Sapindaceae da ordem Sapindales",
       ifelse(familias == "Burseraceae", "Burseraceae também é da ordem Sapindales", "Nenhuma dessas famílias é da ordem Sapindales")
)

```

```

## [1] "Nenhuma dessas famílias é uma Sapindales"
## [2] "Burseraceae também é da ordem Sapindales"
## [3] "Sapindaceae da ordem Sapindales"
## [4] "Nenhuma dessas famílias é uma Sapindales"
## [5] "Sapindaceae da ordem Sapindales"
## [6] "Nenhuma dessas famílias é uma Sapindales"
## [7] "Nenhuma dessas famílias é uma Sapindales"
## [8] "Nenhuma dessas famílias é uma Sapindales"
## [9] "Sapindaceae da ordem Sapindales"
## [10] "Nenhuma dessas famílias é uma Sapindales"
## [11] "Sapindaceae da ordem Sapindales"

```

```
## [12] "Sapindaceae da ordem Sapindales"  
## [13] "Nenhuma dessas famílias é uma Sapindales"  
## [14] "Sapindaceae da ordem Sapindales"  
## [15] "Burseraceae também é da ordem Sapindales"  
## [16] "Sapindaceae da ordem Sapindales"  
## [17] "Sapindaceae da ordem Sapindales"  
## [18] "Nenhuma dessas famílias é uma Sapindales"  
## [19] "Sapindaceae da ordem Sapindales"  
## [20] "Nenhuma dessas famílias é uma Sapindales"  
## [21] "Burseraceae também é da ordem Sapindales"  
## [22] "Nenhuma dessas famílias é uma Sapindales"  
## [23] "Nenhuma dessas famílias é uma Sapindales"  
## [24] "Nenhuma dessas famílias é uma Sapindales"  
## [25] "Sapindaceae da ordem Sapindales"  
## [26] "Burseraceae também é da ordem Sapindales"  
## [27] "Nenhuma dessas famílias é uma Sapindales"  
## [28] "Sapindaceae da ordem Sapindales"  
## [29] "Nenhuma dessas famílias é uma Sapindales"  
## [30] "Nenhuma dessas famílias é uma Sapindales"  
## [31] "Nenhuma dessas famílias é uma Sapindales"  
## [32] "Sapindaceae da ordem Sapindales"  
## [33] "Burseraceae também é da ordem Sapindales"  
## [34] "Nenhuma dessas famílias é uma Sapindales"  
## [35] "Nenhuma dessas famílias é uma Sapindales"  
## [36] "Burseraceae também é da ordem Sapindales"  
## [37] "Nenhuma dessas famílias é uma Sapindales"  
## [38] "Nenhuma dessas famílias é uma Sapindales"  
## [39] "Nenhuma dessas famílias é uma Sapindales"  
## [40] "Nenhuma dessas famílias é uma Sapindales"  
## [41] "Burseraceae também é da ordem Sapindales"  
## [42] "Burseraceae também é da ordem Sapindales"  
## [43] "Burseraceae também é da ordem Sapindales"  
## [44] "Burseraceae também é da ordem Sapindales"  
## [45] "Sapindaceae da ordem Sapindales"  
## [46] "Burseraceae também é da ordem Sapindales"  
## [47] "Nenhuma dessas famílias é uma Sapindales"  
## [48] "Burseraceae também é da ordem Sapindales"
```

```
## [49] "Nenhuma dessas famílias é uma Sapindales"
## [50] "Burseraceae também é da ordem Sapindales"
## [51] "Sapindaceae da ordem Sapindales"
## [52] "Burseraceae também é da ordem Sapindales"
## [53] "Sapindaceae da ordem Sapindales"
## [54] "Burseraceae também é da ordem Sapindales"
## [55] "Sapindaceae da ordem Sapindales"
## [56] "Nenhuma dessas famílias é uma Sapindales"
## [57] "Nenhuma dessas famílias é uma Sapindales"
## [58] "Nenhuma dessas famílias é uma Sapindales"
## [59] "Nenhuma dessas famílias é uma Sapindales"
## [60] "Nenhuma dessas famílias é uma Sapindales"
## [61] "Burseraceae também é da ordem Sapindales"
## [62] "Nenhuma dessas famílias é uma Sapindales"
## [63] "Burseraceae também é da ordem Sapindales"
## [64] "Sapindaceae da ordem Sapindales"
## [65] "Burseraceae também é da ordem Sapindales"
## [66] "Burseraceae também é da ordem Sapindales"
## [67] "Nenhuma dessas famílias é uma Sapindales"
## [68] "Sapindaceae da ordem Sapindales"
## [69] "Nenhuma dessas famílias é uma Sapindales"
## [70] "Nenhuma dessas famílias é uma Sapindales"
## [71] "Nenhuma dessas famílias é uma Sapindales"
## [72] "Nenhuma dessas famílias é uma Sapindales"
## [73] "Sapindaceae da ordem Sapindales"
## [74] "Burseraceae também é da ordem Sapindales"
## [75] "Nenhuma dessas famílias é uma Sapindales"
## [76] "Nenhuma dessas famílias é uma Sapindales"
## [77] "Nenhuma dessas famílias é uma Sapindales"
## [78] "Burseraceae também é da ordem Sapindales"
## [79] "Burseraceae também é da ordem Sapindales"
## [80] "Sapindaceae da ordem Sapindales"
## [81] "Nenhuma dessas famílias é uma Sapindales"
## [82] "Sapindaceae da ordem Sapindales"
## [83] "Sapindaceae da ordem Sapindales"
## [84] "Burseraceae também é da ordem Sapindales"
## [85] "Sapindaceae da ordem Sapindales"
```

```
## [86] "Nenhuma dessas famílias é uma Sapindales"
## [87] "Burseraceae também é da ordem Sapindales"
## [88] "Sapindaceae da ordem Sapindales"
## [89] "Nenhuma dessas famílias é uma Sapindales"
## [90] "Nenhuma dessas famílias é uma Sapindales"
## [91] "Nenhuma dessas famílias é uma Sapindales"
## [92] "Nenhuma dessas famílias é uma Sapindales"
## [93] "Burseraceae também é da ordem Sapindales"
## [94] "Nenhuma dessas famílias é uma Sapindales"
## [95] "Burseraceae também é da ordem Sapindales"
## [96] "Burseraceae também é da ordem Sapindales"
## [97] "Nenhuma dessas famílias é uma Sapindales"
## [98] "Nenhuma dessas famílias é uma Sapindales"
## [99] "Sapindaceae da ordem Sapindales"
## [100] "Nenhuma dessas famílias é uma Sapindales"
```

```
sei_qual_ordem <- ifelse(familias == "Sapindaceae", "Sapindaceae da ordem Sapindales",
  ifelse(familias == "Burseraceae", "Burseraceae também é da ordem Sapindales", "Nenhuma dessa")
)

meudf <- data.frame(familias, sei_qual_ordem)
meudf
```

familias	sei_qual_ordem
Solanaceae	Nenhuma dessas famílias é uma Sapindales
Burseraceae	Burseraceae também é da ordem Sapindales
Sapindaceae	Sapindaceae da ordem Sapindales
Solanaceae	Nenhuma dessas famílias é uma Sapindales
Sapindaceae	Sapindaceae da ordem Sapindales
Rubiaceae	Nenhuma dessas famílias é uma Sapindales
Solanaceae	Nenhuma dessas famílias é uma Sapindales
Solanaceae	Nenhuma dessas famílias é uma Sapindales
Sapindaceae	Sapindaceae da ordem Sapindales
Rubiaceae	Nenhuma dessas famílias é uma Sapindales
Sapindaceae	Sapindaceae da ordem Sapindales
Sapindaceae	Sapindaceae da ordem Sapindales
Rubiaceae	Nenhuma dessas famílias é uma Sapindales
Sapindaceae	Sapindaceae da ordem Sapindales
Burseraceae	Burseraceae também é da ordem Sapindales
Sapindaceae	Sapindaceae da ordem Sapindales
Sapindaceae	Sapindaceae da ordem Sapindales
Solanaceae	Nenhuma dessas famílias é uma Sapindales
Sapindaceae	Sapindaceae da ordem Sapindales
Solanaceae	Nenhuma dessas famílias é uma Sapindales
Burseraceae	Burseraceae também é da ordem Sapindales
Solanaceae	Nenhuma dessas famílias é uma Sapindales
Rubiaceae	Nenhuma dessas famílias é uma Sapindales
Solanaceae	Nenhuma dessas famílias é uma Sapindales
Sapindaceae	Sapindaceae da ordem Sapindales
Burseraceae	Burseraceae também é da ordem Sapindales
Solanaceae	Nenhuma dessas famílias é uma Sapindales
Sapindaceae	Sapindaceae da ordem Sapindales
Burseraceae	Burseraceae também é da ordem Sapindales
Solanaceae	Nenhuma dessas famílias é uma Sapindales
Rubiaceae	Nenhuma dessas famílias é uma Sapindales
Burseraceae	Burseraceae também é da ordem Sapindales
Solanaceae	Nenhuma dessas famílias é uma Sapindales
Solanaceae	Nenhuma dessas famílias é uma Sapindales
Rubiaceae	Nenhuma dessas famílias é uma Sapindales
Rubiaceae	Nenhuma dessas famílias é uma Sapindales
Burseraceae	Burseraceae também é da ordem Sapindales
Burseraceae	Burseraceae também é da ordem Sapindales
Burseraceae	Burseraceae também é da ordem Sapindales

6.3 Iterações

Vocês usaram várias funções da família `apply`, especialmente `apply()` e `tapply()`, que são funções especiais que repetem uma mesma função `FUN` para cada objeto de um conjunto (vetores, matrizes, valores atômicos). Essas funções utilizam portanto a lógica de iterações (em inglês, *loops*), ou seja, fazem a mesma ação repetidas vezes; em outras palavras, fazem *LOOPS*, dão voltas, realizam o mesmo percurso várias vezes, percorrem um circuito.

As expressões `for(){}` e `while(){}` permitem fazer *LOOPS*, e *LOOPS* dentro de *LOOPS* com muita liberdade. Aprendê-las é o mesmo que aprender todas as funções da família `apply` juntas (`apply()`, `tapply()`, `sapply()`, `lapply()`, `mapply()` etc). Se você compreende os *LOOPS*, pode fazer o que essas funções fazem sem precisar delas (embora elas possam executar a tarefa mais rapidamente).

6.3.1 Iteração com `for(){}`

6.3.1.1 Exemplo 1

Um exemplo simples do que seria uma iteração com o comando `for(){}`:

```
# vamos imprimir na tela as letras do objeto LETTERS
for (let in 1:length(LETTERS)) {
  paraimprimir <- paste(LETTERS[let], " é a letra de índice ", let)
  print(paraimprimir)
}
```

```
## [1] "A é a letra de índice 1"
## [1] "B é a letra de índice 2"
## [1] "C é a letra de índice 3"
## [1] "D é a letra de índice 4"
```

```

## [1] "E é a letra de índice 5"
## [1] "F é a letra de índice 6"
## [1] "G é a letra de índice 7"
## [1] "H é a letra de índice 8"
## [1] "I é a letra de índice 9"
## [1] "J é a letra de índice 10"
## [1] "K é a letra de índice 11"
## [1] "L é a letra de índice 12"
## [1] "M é a letra de índice 13"
## [1] "N é a letra de índice 14"
## [1] "O é a letra de índice 15"
## [1] "P é a letra de índice 16"
## [1] "Q é a letra de índice 17"
## [1] "R é a letra de índice 18"
## [1] "S é a letra de índice 19"
## [1] "T é a letra de índice 20"
## [1] "U é a letra de índice 21"
## [1] "V é a letra de índice 22"
## [1] "W é a letra de índice 23"
## [1] "X é a letra de índice 24"
## [1] "Y é a letra de índice 25"
## [1] "Z é a letra de índice 26"

```

Ou seja, para cada elemento do objeto `let`, assumindo os valores na sequência do elemento 1 ao elemento correspondente ao comprimento (`== length(LETTERS)`) do objeto `LETTERS`, execute o que está dentro de {}.

6.3.1.2 Exemplo 2

```

# Fazendo um loop com for(){}
# criamos uma matriz
vetn <- rnorm(100, 30, 1)
mvtn <- matrix(vetn, ncol = 10, dimnames = list(paste("linha", 1:10), paste("coluna", 1:10)))
head(mvtn)

```

	coluna 1	coluna 2	coluna 3	coluna 4	coluna 5	coluna 6	coluna 7	coluna 8
linha 1	29.83293	29.14018	29.23649	30.41905	30.94899	29.84389	29.97366	30.15412
linha 2	31.24129	31.05689	29.45685	30.92010	30.28868	30.51557	29.74122	30.46949
linha 3	30.29010	31.34928	29.17843	28.80200	30.94568	29.34699	29.88953	28.76698
linha 4	29.94628	30.14674	30.23371	28.56894	31.47962	32.13595	29.62404	28.93148
linha 5	30.07430	30.81179	29.26558	29.58111	29.59002	29.91044	32.09775	31.96293
linha 6	29.88561	28.47641	29.62197	30.23359	29.96315	28.71014	30.48448	31.47933

```
# Fazendo algo == apply(mvetn, 2, mean)
# cria um objeto para salvar o resultado
resultado <- NULL
for (coluna in 1:ncol(mvetn)) {
  # pega a coluna
  cl <- mvetn[, coluna]
  # calcula a media
  mcl <- mean(cl)
  print(paste("Média da coluna", colnames(mvetn)[cl], "=?", round(mcl, 2)))
  # salva o resultado com a media do item anterior
  resultado <- c(resultado, mcl)
}
```

```
## [1] "Média da coluna NA = 30.02" "Média da coluna NA = 30.02"
## [3] "Média da coluna NA = 30.02" "Média da coluna NA = 30.02"
## [5] "Média da coluna NA = 30.02" "Média da coluna NA = 30.02"
## [7] "Média da coluna NA = 30.02" "Média da coluna NA = 30.02"
## [9] "Média da coluna NA = 30.02" "Média da coluna NA = 30.02"
## [1] "Média da coluna NA = 29.77" "Média da coluna NA = 29.77"
## [3] "Média da coluna NA = 29.77" "Média da coluna NA = 29.77"
## [5] "Média da coluna NA = 29.77" "Média da coluna NA = 29.77"
## [7] "Média da coluna NA = 29.77" "Média da coluna NA = 29.77"
## [9] "Média da coluna NA = 29.77" "Média da coluna NA = 29.77"
## [1] "Média da coluna NA = 29.92" "Média da coluna NA = 29.92"
## [3] "Média da coluna NA = 29.92" "Média da coluna NA = 29.92"
## [5] "Média da coluna NA = 29.92" "Média da coluna NA = 29.92"
## [7] "Média da coluna NA = 29.92" "Média da coluna NA = 29.92"
## [9] "Média da coluna NA = 29.92" "Média da coluna NA = 29.92"
```

```
## [1] "Média da coluna NA = 29.8" "Média da coluna NA = 29.8"
## [3] "Média da coluna NA = 29.8" "Média da coluna NA = 29.8"
## [5] "Média da coluna NA = 29.8" "Média da coluna NA = 29.8"
## [7] "Média da coluna NA = 29.8" "Média da coluna NA = 29.8"
## [9] "Média da coluna NA = 29.8" "Média da coluna NA = 29.8"
## [1] "Média da coluna NA = 30.21" "Média da coluna NA = 30.21"
## [3] "Média da coluna NA = 30.21" "Média da coluna NA = 30.21"
## [5] "Média da coluna NA = 30.21" "Média da coluna NA = 30.21"
## [7] "Média da coluna NA = 30.21" "Média da coluna NA = 30.21"
## [9] "Média da coluna NA = 30.21" "Média da coluna NA = 30.21"
## [1] "Média da coluna NA = 29.85" "Média da coluna NA = 29.85"
## [3] "Média da coluna NA = 29.85" "Média da coluna NA = 29.85"
## [5] "Média da coluna NA = 29.85" "Média da coluna NA = 29.85"
## [7] "Média da coluna NA = 29.85" "Média da coluna NA = 29.85"
## [9] "Média da coluna NA = 29.85" "Média da coluna NA = 29.85"
## [1] "Média da coluna NA = 30.34" "Média da coluna NA = 30.34"
## [3] "Média da coluna NA = 30.34" "Média da coluna NA = 30.34"
## [5] "Média da coluna NA = 30.34" "Média da coluna NA = 30.34"
## [7] "Média da coluna NA = 30.34" "Média da coluna NA = 30.34"
## [9] "Média da coluna NA = 30.34" "Média da coluna NA = 30.34"
## [1] "Média da coluna NA = 30.33" "Média da coluna NA = 30.33"
## [3] "Média da coluna NA = 30.33" "Média da coluna NA = 30.33"
## [5] "Média da coluna NA = 30.33" "Média da coluna NA = 30.33"
## [7] "Média da coluna NA = 30.33" "Média da coluna NA = 30.33"
## [9] "Média da coluna NA = 30.33" "Média da coluna NA = 30.33"
## [1] "Média da coluna NA = 29.95" "Média da coluna NA = 29.95"
## [3] "Média da coluna NA = 29.95" "Média da coluna NA = 29.95"
## [5] "Média da coluna NA = 29.95" "Média da coluna NA = 29.95"
## [7] "Média da coluna NA = 29.95" "Média da coluna NA = 29.95"
## [9] "Média da coluna NA = 29.95" "Média da coluna NA = 29.95"
## [1] "Média da coluna NA = 30.69" "Média da coluna NA = 30.69"
## [3] "Média da coluna NA = 30.69" "Média da coluna NA = 30.69"
## [5] "Média da coluna NA = 30.69" "Média da coluna NA = 30.69"
## [7] "Média da coluna NA = 30.69" "Média da coluna NA = 30.69"
## [9] "Média da coluna NA = 30.69" "Média da coluna NA = 30.69"
```

```
# adiciona o nome das colunas  
names(resultado) <- colnames(mvetn)  
# ver o resultado  
resultado
```

```
## coluna 1 coluna 2 coluna 3 coluna 4 coluna 5 coluna 6 coluna 7 coluna 8  
## 30.02231 29.76714 29.91783 29.79591 30.21034 29.85470 30.33654 30.32650  
## coluna 9 coluna 10  
## 29.94533 30.69054
```

```
# identico ao apply, maior controle de como a média é aplicada  
resultado == apply(mvetn, 2, mean)
```

```
## coluna 1 coluna 2 coluna 3 coluna 4 coluna 5 coluna 6 coluna 7 coluna 8  
## TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE  
## coluna 9 coluna 10  
## TRUE TRUE
```

6.3.1.3 Exemplo 3

Agora usando `for(){}` para fazer algo como o `tapply()`.

```
# Fazendo algo == tapply(mvetn[,2], mvetn$classe, sum)  
# criamos uma matriz  
vetn <- rnorm(100, 30, 1)  
mvetn <- matrix(vetn, ncol = 10, dimnames = list(paste("linha", 1:10), paste("coluna", 1:10)))  
# transformamos num data.frame adicionando uma coluna categorica  
mvetn <- data.frame(classe = sample(paste("categ", 1:3, sep = ""), size = nrow(mvetn), replace = TRUE))  
head(mvetn[, 1:5])  
# cria um objeto para salvar o resultado  
resultado <- NULL  
# para cada categoria
```

```

for (ct in 1:length(levels(mvetn$classe))) {
  # pega a categoria
  cl <- levels(mvetn$classe)[ct]
  # filtra os dados (vetor lógico)
  vl <- mvetn$classe == cl
  # calcula a soma dos dados da categoria
  soma <- sum(mvetn[vl, 2], na.rm = T)
  # imprime o passo
  print(paste("A soma da categoria", cl, "é igual a", soma))
  # junta os resultados
  resultado <- c(resultado, soma)
}
# atribui nomes aos elementos do vetor de somas
names(resultado) <- levels(mvetn$classe)
# confere
resultado == tapply(mvetn[, 2], mvetn$classe, sum)

```

6.3.1.4 Exemplo 4

Calculando somas e médias de linhas, similar ao que podemos fazer com a função `apply()`:

```

somas <- NULL # objeto vazio para salvar soma de cada linha
medias <- NULL # objeto vazio para salvar medias de cada linha
for (i in 1:nrow(X)) { # para cada linha
  somai <- sum(X[i, ]) # soma dos valores na linha i
  somas <- c(somas, somai) # junta a somai com o resto (que estará vazio na primeira vez)
  mediai <- mean(X[i, ]) # média dos valores na linha i
  medias <- c(medias, mediai) # junta as medias
}
# como a matriz tem nomes, acrescenta esses nomes aos vetores com resultados
names(medias) <- rownames(X)
medias

## ln 1 ln 2 ln 3 ln 4

```

```
##   17   18   19   20
```

```
names(somas) <- rownames(X)
somas
```

```
## ln 1 ln 2 ln 3 ln 4
## 153 162 171 180
```

6.3.2 Iteração com `while(){}`

O comando `while(){} funciona de forma parecida, mas faz algo ENQUANTO (em inglês, while) a condição em while(){} seja verdadeira. Um exemplo simples:`

6.3.2.1 Exemplo 1

```
# cria um vetor de valores aleatorizados
vet <- sample(10:100)
# amostra um valor do vetor até que este valor seja 10
conta <- 1
valor <- 0
while (valor != 10) {
  valor <- sample(vet, 1)
  print(paste("o valor selecionado na iteração", conta, "foi de ", valor))
  conta <- conta + 1
}
```

```
## [1] "o valor selecionado na iteração 1 foi de 81"
## [1] "o valor selecionado na iteração 2 foi de 23"
## [1] "o valor selecionado na iteração 3 foi de 30"
## [1] "o valor selecionado na iteração 4 foi de 32"
## [1] "o valor selecionado na iteração 5 foi de 39"
```

```
## [1] "o valor selecionado na iteração 6 foi de 74"
## [1] "o valor selecionado na iteração 7 foi de 88"
## [1] "o valor selecionado na iteração 8 foi de 16"
## [1] "o valor selecionado na iteração 9 foi de 60"
## [1] "o valor selecionado na iteração 10 foi de 61"
## [1] "o valor selecionado na iteração 11 foi de 62"
## [1] "o valor selecionado na iteração 12 foi de 74"
## [1] "o valor selecionado na iteração 13 foi de 10"
```

6.3.3 Iteração com `for(){} e a condicional if(){}`

6.3.3.1 Exemplo 1

```
# cria um vetor de valores aleatorizados
vet <- sample(10:100, 60, replace = T)
# amostra um valor do vetor até que este valor seja 10
for (v in 1:10000) {
  valor <- sample(vet, 1)
  if (valor == 10) {
    # se o valor selecionado aleatoriamente for 10, ou seja se a expressão valor==10 for TRUE
    # imprima isso
    print(paste("A primeira vez que o valor 10 foi selecionado aleatoriamente foi quando o obje")
    # interrompa (quebre) o loop
    break # note este argumento
  } else {
    # caso contrario, valor!=10, imprime o valor selecionado e continua o loop
    print(paste("O valor selecionado foi ", valor, "no índice", v))
  }
}
```

É possível que o script acima repita as 10000 vezes do `for(){} sem encontrar o valor 10, até porque o 10 pode não estar em vet se não for amostrado.`

6.4 Criando ou modificando funções

Funções são objetos que contêm um script que usa os argumentos (também objetos) para executar alguma coisa. A expressão `function(){}` é utilizada para criar funções. Criar funções é útil pois podem executar algo várias vezes (podendo ser de forma diferente) sem precisar reescrever o código todas às vezes. Isso nos auxilia em práticas rotineiras como, por exemplo, na manipulação de um conjunto de dados de localização geográfica de espécimes botânicos. Podemos gerar um mapa personalizado de distribuição geográfica de cada espécie criando uma função para plotar um mapa, e depois utilizamos a aplicamos a função sobre a categoria, neste caso, a variável contendo o nome da espécie. É muito útil também poder modificar uma função criada por outra pessoa, seja uma função de um determinado **pacote** ou uma função que você encontrou em uma página qualquer navegando pela internet.

É muito simples construir uma função. Há um bloco que deve ser sempre repetido:

```
function(meu_argumento1, meu_argumento2, ...) {  
  
    # AQUI FICAM AS AÇÕES DE SUA FUNÇÃO, COMO POR EXEMPLO  
  
    return("resultado da função") #  
  
}
```

É costume sempre utilizar a função `return()` como último elemento da função para que algum objeto seja retornado ao usuário.

```
?return # veja o help
```

6.4.1 Exemplo I

Vamos fazer a nossa versão da função `mean()`, que tira a média aritmética dos valores de um vetor.

```
# Vamos fazer essa função na unha:
amedia <- function(x) {
  # x será um vetor de comprimento >=1 e todos os valores devem ser numéricos, senão precisamos tratar
  # essa função irá retornar o valor do 'am' que definimos como nulo inicialmente
  am <- NULL
  # condicao 1
  c1 <- is.vector(x)
  # condicao 2
  xx <- as.numeric(x) # convertemos em numérico
  xx <- xx[!is.na(xx)] # tira o que não é número (o que não foi convertido ou está em branco)
  c2 <- length(xx) == length(x) # os comprimentos são iguais? e tem algum valor?
  if (c1 & length(xx) > 0) { # se for um vetor e houver algum valor numérico, pode calcular a média
    am <- sum(xx) / length(xx)
  } else {
    print(paste(length(x) - length(xx), "valores do vetor não são numéricos e foram excluídos"))
  }
  return(am)
}
```

Vamos agora utilizar a função:

```
v1 <- c(1, 2, 3, 4, 5, 6)
amedia(v1)
```

```
## [1] 3.5
```

```
v1 <- c(3, 3, 3, 3, 3, 3, 3)
amedia(v1)
```

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

```
v1 <- c(3, 3, 3, 3, 3, 3, "A")
amedia(v1)
```

```
## Warning in amedia(v1): NAs introduced by coercion
## [1] "1 valores do vetor não são numéricos e foram excluídos"
## [1] 3
```

```
v1 <- LETTERS
amedia(v1)
```

```
## Warning in amedia(v1): NAs introduced by coercion
## [1] "0 objeto não é um vetor ou não há valores numéricos"
## NULL
```

6.4.2 Exemplo II

Os *loops* por meio da expressão `for(){}` e a condicional `if` são muito úteis dentro de funções. Sua função pode, por exemplo, ser construída para que um argumento possa assumir diferentes valores e, a depender do valor, executar uma coisa diferente. Como exemplo, vamos criar uma função que contém o script de exemplo de `if`:

```
# CRIA uma função com os seguintes argumentos:  
# vet = um vetor de valores  
# busca.valor = um valor para busca em vet  
# nrun = numero de vezes da iteracao  
minhafuncao <- function(vet, busca.valor, nrun) {  
  # cria um loop do número de vezes (note o argumento nrun abaixo)  
  for (v in 1:nrun) {  
    # pega um valor aleatorio  
    # amostra o indice aleatoriamente  
    idx <- sample(1:length(vet), 1)  
    valor <- vet[idx]  
    # se o valor amostrado for igual ao valor procurado para  
    if (valor == busca.valor) {  
      # se o valor selecionado aleatoriamente for 10, ou seja se a expressão valor==10 for TRUE  
      # imprima isso  
      # print(paste("A primeira vez que o valor 10 foi selecionado aleatoriamente foi quando o  
      # interrompa (quebre) o loop  
      # break # note este argumento  
    } else {  
      # caso contrario, valor!=10, imprime o valor selecionado e continua o loop  
      # print(paste("O valor selecionado foi ",valor,"no indice",v))  
    }  
  }  
  # o valor do objeto v será o último valor assumido na execucao do for()[], se tiver encontrado  
  # se encontrou retorna o indice do valor (que é o último valor assumido por idx)  
  # caso o último valor seja == ao valor buscado, ou que o ultimo v é menor que o especificado  
  if (v < nrun | valor == busca.valor) {  
    print(paste("Encontrei o valor", busca.valor, "no indice, ", idx, "do vetor indicado"))  
    return(idx)  
  } else {  
    # caso contrário retorna NA  
    print(paste("Não encontrei o valor", busca.valor, "no vetor indicado"))  
    return(NA)  
  }  
}
```

Vamos utilizar a função recém-criada:

```
umvetor <- sample(10:100, 60, replace = T)
minhafuncao(vet = umvetor, busca.valor = 22, nrun = 1000)

## [1] "Não encontrei o valor 22 no vetor indicado"

## [1] NA

retornou.isso <- minhafuncao(vet = umvetor, busca.valor = 22, nrun = 1000)

## [1] "Não encontrei o valor 22 no vetor indicado"

# e usar de novo com outros valores
umvetor <- sample(200:300, 50, replace = T)
sort(table(umvetor), decreasing = T)[1:10]



|     |     |     |     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 255 | 210 | 219 | 228 | 260 | 261 | 264 | 269 | 201 | 202 |
| 3   | 2   | 2   | 2   | 2   | 2   | 2   | 2   | 1   | 1   |



minhafuncao(vet = umvetor, busca.valor = 250, nrun = 10000)

## [1] "Encontrei o valor 250 no indice, 36 do vetor indicado"

## [1] 36
```

```
retornou.isso <- minhafuncao(vet = umvetor, busca.valor = 250, nrun = 10000)
```

```
## [1] "Encontrei o valor 250 no indice, 36 do vetor indicado"
```

6.5 Para saber mais:

- (1) Acesse vídeoaulas com conteúdo parcial deste capítulo clicando nos tópicos abaixo:

- Vetores e Operadores Lógicos¹;
- Condicionais no R²

- (2) Auxílio do R - executem a expressão `?control` no R e vejam a explicação sobre condicionais.

¹http://www.botanicaamazonica.wiki.br/labotam/lib/exe/fetch.php?media=bot89:precurso:6vetores:video01_bot89-2020-04-07_07.52.00.mp4

²https://youtu.be/8Z_kO2PwQZc

7

Sumarização de dados

7.1 Tabelas dinâmicas

A função `tapply()` calcula alguma função sobre um vetor numérico para cada categoria de um fator. Já a função `aggregate()` faz o mesmo, mas permite múltiplos fatores e sempre retorna um `data.frame`.

Vamos usar dados de parcelas em caixetas¹, formações dominadas por *Tabebuia cassinoides* (Lam.) D.C. (Bignoniaceae), espécie comum da Mata Atlântica nos estados de São Paulo, Minas Gerais, Rio de Janeiro e Espírito Santo. Baixe o arquivo para seu computador e instale-o na sua pasta de trabalho antes de seguir com os comandos abaixo.

```
caixeta <- read.table("caixeta.csv", sep = ",", header = T)
```

```
names(caixeta)
```

```
## [1] "local"    "parcela"   "arvore"    "fuste"     "cap"       "h"         "especie"
```

```
## tapply: resumo de uma variável numérica, separada por níveis de um ou mais fatores
?tapply # veja o help dessa função
```

¹<https://github.com/LABOTAM/IntroR/blob/main/dados/caixeta.csv>

```
# altura máxima de cada espécie
tapply(caixeta$h, INDEX = caixeta$especie, FUN = max)
```

##	<i>Alchornea triplinervia</i>	<i>Andira fraxinifolia</i>	bombacaceae
##	140	90	150
##	<i>Cabralea canjerana</i>	<i>Calophyllum brasiliensis</i>	<i>Calophyllum brasiliensis</i>
##	150	200	160
##	<i>Cecropia</i> sp	<i>Coussapoa macrocarpa</i>	<i>Coussapoa micropoda</i>
##	70	100	110
##	<i>Cryptocaria moschata</i>	<i>Cyathea</i> sp	<i>Eugenia oblongata</i>
##	140	30	100
##	eugenias3	fabaceae1	<i>Ficus</i> sp
##	110	70	130
##	<i>Gomidesia</i> sp	<i>Ilex durosa</i>	<i>Ilex</i> sp
##	110	130	90
##	indet.1	indet.2	indet.3
##	170	80	80
##	<i>Inga</i> sp	<i>Jacaranda puberula</i>	jussara
##	110	50	160
##	<i>Matayba</i> sp	Mela 1	Mela 2
##	130	90	50
##	<i>Myrcia sulfiflora</i>	Myrtaceae 3	myrtaceae1
##	170	40	160
##	myrtaceae2	myrtaceae4	<i>Pera glabrata</i>
##	30	140	100
##	<i>Persea</i> sp	<i>Pisonia</i> sp	<i>Psidium</i> sp
##	130	110	120
##	<i>Simplocos</i> sp	Solanum sp1	Solanum sp2
##	90	60	70
##	<i>Syagrus romanzoffianus</i>	Tabebuia 1	<i>Tabebuia cassinoides</i>
##	70	60	480
##	<i>Tibouchina nutticeps</i>		
##		50	

```
# circunferencia media por localidade
tapply(caixeta$cap, INDEX = caixeta$local, FUN = mean)
```

```
##    chauas     jureia     retiro
## 293.6385 404.4813 236.5972
```

```
## "Tabelas dinâmicas": função aggregate
## Criar data.frame com altura média dos fustes por espécie e por local
```

```
?aggregate # veja o help dessa função
```

```
names(caixeta)
```

```
## [1] "local"   "parcela" "arvore"  "fuste"   "cap"      "h"       "especie"
```

```
# circunferencia máxima por espécie
ob1 <- aggregate(caixeta$cap, by = list(especie = caixeta$especie), FUN = max)
class(ob1) # obtenho um data frame
```

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

```
head(ob1)
```

especie	x
Alchornea triplinervia	840
Andira fraxinifolia	340
bombacaceae	380
Cabralea canjerana	720
Calophyllum brasiliensis	1130
Calophyllum brasiliensis	2100

```
# neste caso também poderia fazer assim
ob2 <- tapply(caixeta$h, caixeta$especie, max)
class(ob2) # mas neste caso nos temos um array (um vetor unidimensional)
```

```
## [1] "array"
```

```
ob2[1:10]
```

```
##   Alchornea triplinervia     Andira fraxinifolia      bombacaceae
##                 140                  90                  150
##   Cabralea canjerana Calophyllum brasiliensis Calophyllum brasiliensis
##                 150                  200                  160
##   Cecropia sp     Coussapoa macrocarpa    Coussapoa micropoda
##                 70                  100                  110
##   Cryptocaria moschata
##                 140
```

```
# mas se eu quiser por localidade e por especie, preciso usar aggregate
caixeta.alt <- aggregate(caixeta$h, by = list(local = caixeta$local, especie = caixeta$especie))
```

```
head(caixeta.alt)
```

local	especie	x
jureia	Alchornea triplinervia	140
retiro	Alchornea triplinervia	100
jureia	Andira fraxinifolia	90
jureia	bombacaceae	150
jureia	Cabralea canjerana	150
chauas	Callophyllum brasiliensis	200

Vamos calcular a área basal (soma da área de todo os fustes)

calculando a área basal de cada fuste, considerando o fuste um círculo perfeito, poderíamos
 caixeta\$ab <- caixeta\$cap^2 / 4 * pi
 ## e agora criamos a planilha, com aggregate, somando as áreas basais dos fustes
 caixeta.2 <- aggregate(caixeta\$ab, by = list(local = caixeta\$local, parcela = caixeta\$parcela),
 class(caixeta.2)

[1] "data.frame"

```
head(caixeta.2)
```

local	parcela	especie	x
retiro	1	Alchornea triplinervia	53092.92
jureia	2	Alchornea triplinervia	554176.94
retiro	2	Alchornea triplinervia	90949.11
retiro	3	Alchornea triplinervia	230121.66
jureia	4	Alchornea triplinervia	292246.66
jureia	5	Alchornea triplinervia	273397.10

7.2 Tabelas de contagem

Vamos usar utilizar os mesmos dados de caixetas utilizados na seção [7.1](#). A função `table()` permite contar valores em fatores e vetores.

```
caixeta <- read.table("caixeta.csv", sep = ",", header = T)
```

```
names(caixeta)
```

```
## [1] "local"   "parcela" "arvore"  "fuste"   "cap"     "h"       "especie"  
## [8] "ab"
```

```
# tem a coluna especie
```

```
# podemos resumir quantos individuos tem de cada espécie (considerando que cada linha é um in  
table(caixeta$especie)
```

Var1	Freq
Alchornea triplinervia	15
Andira fraxinifolia	4
bombacaceae	1
Cabralea canjerana	4
Callophyllum brasiliensis	7
Callophyllum brasiliensis	4
Cecropia sp	1
Coussapoa macrocarpa	3
Coussapoa micropoda	9
Cryptocaria moschata	2
Cyathea sp	2
Eugenia oblongata	2
eugenia3	1
fabaceae1	1
Ficus sp	2
Gomidesia sp	1
Ilex duxosa	8
Ilex sp	2
indet.1	1
indet.2	1
indet.3	1
Inga sp	4
Jacaranda puberula	2
jussara	37
Matayba sp	1
Mela 1	63
Mela 2	2
Myrcia sulfiflora	96
Myrtaceae 3	3
myrtaceae1	4
myrtaceae2	1
myrtaceae4	1
Pera glabrata	1
Persea sp	3
Pisonia sp	2
Psidium sp	20
Simplocos sp	2
Solanum sp1	1
Solanum sp2	1
Syagrus romanzoffianus	1
Tabebuia 1	10
Tabebuia cassinoides	698
Tibouchina nutticeps	2

```
# mostra as tres especies mais abundantes
sort(table(caixeta$especie), decreasing = T)[1:3]
```

Tabebuia cassinoides	Myrcia sulfiflora	Mela 1
698	96	63

```
# quantos individuos por localidade?
table(caixeta$local)
```

chauas	jureia	retiro
426	241	360

```
# especie por localidade
tb <- table(caixeta$especie, caixeta$local)
```

```
head(tb, 3) # mostra as tres primeiras linhas
```

	chauas	jureia	retiro
Alchornea triplinervia	0	3	12
Andira fraxinifolia	0	4	0
bombacaceae	0	1	0

```
# tabela de presenca e ausencia de especie por localidade
tb <- table(caixeta$especie, caixeta$local)
# quem tem mais de 0 individuos esta presente
# portanto, substituo por 1
tb[tb > 0] <- 1
```

```
head(tb)
```

	chauas	jureia	retiro
Alchornea triplinervia	0	1	1
Andira fraxinifolia	0	1	0
bombacaceae	0	1	0
Cabralea canjerana	0	1	0
Callophyllum brasiliensis	1	0	0
Callophyllum brasiliensis	0	1	0

```
# sendo assim, posso ver o numero de especie por localidade aplicando a soma das linha que tem
apply(tb, 2, sum)
```

```
## chauas jureia retiro
##      13      22      13
```

7.3 Lógica da junção de tabelas

Unir tabelas é uma prática corriqueira com bases de dados. É comum termos dados relacionados em tabelas diferentes, recurso que minimiza a entrada de redundância e portanto de erros nos nossos dados.

É frequente também a necessidade de ter esses dados reunidos em uma só tabela. Para unir tabelas, é necessário que duas tabelas diferentes possuam uma coluna em comum, a quem vamos chamar de **identificador**. Vamos criar aqui uma situação artificial com os dados *iris*, mas imagine uma situação mais complexa com muitos dados.

```
# uma tabela com os nomes das espécies
spp <- unique(data.frame(
  GENUS = "Iris",
  SPECIES = iris$Species,
  stringsAsFactors = F
))
spp$fullname <- paste(spp$GENUS, spp$SPECIES)
spp
```

	GENUS	SPECIES	fullname
1	Iris	setosa	Iris setosa
51	Iris	versicolor	Iris versicolor
101	Iris	virginica	Iris virginica

Vamos adicionar uns dados ao objeto `spp`. Para isso, utilizaremos o pacote `taxize` (Chamberlain et al., 2020) para buscar nomes de espécies na rede. Para fazer uso da função `tp_search()`, é necessário ter uma chave API², que nada mais é que uma senha para que você possa acessar serviços na rede sem a necessidade de um navegador. Nós utilizamos uma chave obtida junto ao Tropicos.org³, base de dados do Jardim Botânico do Missouri (Missouri Botanical Garden). O pacote `taxize` oferece uma função chamada `use_tropicos()` que abre o navegador na página de solicitação da chave API. Você pode executar o comando, preencher o formulário e aguardar por sua chave:

```
use_tropicos()
```

Para este exemplo, guardamos nossa chave API em um objeto chamado `tropicos_key` que, por motivos óbvios, não mostraremos aqui o que ele guarda:

²https://en.wikipedia.org/wiki/Application_programming_interface_key

³<https://www.tropicos.org/>

```
# install.packages("taxize")
library("taxize") # instale se nao tiver
sppinfo <- sapply(spp$fullname, tp_search, key = tropicos_key, type = "exact")
```

O resultado de nossa pesquisa foi estocado no objeto `sppinfo`. Vamos pegar as colunas obtidas para todos os nomes:

```
keys <- table(unlist(lapply(sppinfo, names)))
keys <- names(keys[keys == length(sppinfo)])
keys
```

```
## [1] "author"                  "displaydate"
## [3] "displayreference"        "family"
## [5] "nameid"                  "nomenclaturestatusname"
## [7] "rankabbreviation"        "scientificname"
## [9] "scientificnamewithauthors" "totalrows"
```

Juntemos agora tudo em um único `data.frame`:

```
sppinfo <- as.data.frame(do.call(mapply, c(FUN = c, lapply(sppinfo, `[,`, keys))), stringsAsFactors = FALSE))
sppinfo
```

	author	displaydate	displayreference	family	nameid
Iris setosa	Pall. ex Link	1820	Jahrb. Gewächsk. 1(3): 71	Iridaceae	16600262
Iris versicolor1	L.	1753	Sp. Pl. 1: 39	Iridaceae	16600268
Iris versicolor2	Thunb.	1784	Fl. Jap. 34	Iridaceae	100206951
Iris virginica	L.	1753	Sp. Pl. 1: 39	Iridaceae	16600544

Vamos excluir os nomes ilegítimos:

```
sppinfo <- sppinfo[-grep("illeg", sppinfo$nomenclaturestatusname, ignore.case = T), ]
sppinfo
```

	author	displaydate	displayreference	family	nameid	n
Iris setosa	Pall. ex Link	1820	Jahrb. Gewächsk. 1(3): 71	Iridaceae	16600262	N
Iris versicolor	L.	1753	Sp. Pl. 1: 39	Iridaceae	16600268	N
Iris virginica	L.	1753	Sp. Pl. 1: 39	Iridaceae	16600544	N

Vamos agora criar um identificador compartilhado entre as tabelas:

```
sppinfo$Species <- gsub("Iris ", "", sppinfo$scientificname)
```

Vamos bagunçar a ordem dos dados em `sppinfo` para mostrar como se procede a junção de tabelas:

```
set.seed(4857)
sppinfo <- sppinfo[sample(1:3), ]
rownames(sppinfo) <- sppinfo$Species
```

Agora temos dois conjuntos de dados que em comum possuem a coluna `Species`, mas apresentam linhas diferentes:

```
sppinfo
```

	author	displaydate	displayreference	family	nameid	nomer
versicolor	L.	1753	Sp. Pl. 1: 39	Iridaceae	16600268	No op
virginica	L.	1753	Sp. Pl. 1: 39	Iridaceae	16600544	No op
setosa	Pall. ex Link	1820	Jahrb. Gewächsk. 1(3): 71	Iridaceae	16600262	No op

```
head(iris)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

Suponhamos que você queira adicionar à tabela `iris` uma coluna com informação que está na tabela `sppinfo`. Podemos pensar em duas maneiras de executar esta ação.

7.3.1 Maneira 1 - função `match()`

Pegaremos as linhas da tabela `sppinfo` com correspondência a cada linha da tabela `iris`. Para isso, devemos ter o **índice** da tabela `sppinfo` segundo o valor da coluna `Species`, que é o identificador em comum entre as duas tabelas:

```
idxinfo <- match(iris$Species, sppinfo$Species)
```

Guardamos esta correspondência no vetor `idxinfo`, que possui o mesmo comprimento que o número de linhas que `iris` e contém o número das linhas (os **índices!**) da tabela `sppinfo`:

```
# assim, seguindo indexacao numerica eu posso pegar informacoes da tabela sppinfo e colocar na
iris$speciesComAutor <- sppinfo$scientificnamewithauthors[idxinfo]
head(iris)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	speciesComAutor
5.1	3.5	1.4	0.2	setosa	Iris setosa Pall. ex Link
4.9	3.0	1.4	0.2	setosa	Iris setosa Pall. ex Link
4.7	3.2	1.3	0.2	setosa	Iris setosa Pall. ex Link
4.6	3.1	1.5	0.2	setosa	Iris setosa Pall. ex Link
5.0	3.6	1.4	0.2	setosa	Iris setosa Pall. ex Link
5.4	3.9	1.7	0.4	setosa	Iris setosa Pall. ex Link

Agora, vamos unir as duas tabelas:

```
novoiris <- cbind(iris, sppinfo[idxinfo, ])
head(novoiris)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	speciesComAutor
setosa	5.1	3.5	1.4	0.2	setosa	Iris setosa Pall. ex
setosa.1	4.9	3.0	1.4	0.2	setosa	Iris setosa Pall. ex
setosa.2	4.7	3.2	1.3	0.2	setosa	Iris setosa Pall. ex
setosa.3	4.6	3.1	1.5	0.2	setosa	Iris setosa Pall. ex
setosa.4	5.0	3.6	1.4	0.2	setosa	Iris setosa Pall. ex
setosa.5	5.4	3.9	1.7	0.4	setosa	Iris setosa Pall. ex

7.3.2 Maneira 2 - índices nominais

A tabela sppinfo contém nomes de linhas que correspondem aos valores que estão na coluna iris\$Species. Portanto, para fazer a mesma coisa que fizemos na [maneira 1](#), nós poderíamos simplesmente filtrar através dos nomes das linhas da tabela sppinfo:

```
iris$speciesComAutor <- sppinfo[iris$Species, ]$scientificnamewithauthors
# juntando as duas tabelas completas
novoiris <- cbind(iris, sppinfo[iris$Species, ])
novoiris
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	speciesCo
versicolor	5.1	3.5	1.4	0.2	setosa	Iris versico
versicolor.1	4.9	3.0	1.4	0.2	setosa	Iris versico
versicolor.2	4.7	3.2	1.3	0.2	setosa	Iris versico
versicolor.3	4.6	3.1	1.5	0.2	setosa	Iris versico
versicolor.4	5.0	3.6	1.4	0.2	setosa	Iris versico
versicolor.5	5.4	3.9	1.7	0.4	setosa	Iris versico
versicolor.6	4.6	3.4	1.4	0.3	setosa	Iris versico
versicolor.7	5.0	3.4	1.5	0.2	setosa	Iris versico
versicolor.8	4.4	2.9	1.4	0.2	setosa	Iris versico
versicolor.9	4.9	3.1	1.5	0.1	setosa	Iris versico
versicolor.10	5.4	3.7	1.5	0.2	setosa	Iris versico
versicolor.11	4.8	3.4	1.6	0.2	setosa	Iris versico
versicolor.12	4.8	3.0	1.4	0.1	setosa	Iris versico
versicolor.13	4.3	3.0	1.1	0.1	setosa	Iris versico
versicolor.14	5.8	4.0	1.2	0.2	setosa	Iris versico
versicolor.15	5.7	4.4	1.5	0.4	setosa	Iris versico
versicolor.16	5.4	3.9	1.3	0.4	setosa	Iris versico
versicolor.17	5.1	3.5	1.4	0.3	setosa	Iris versico
versicolor.18	5.7	3.8	1.7	0.3	setosa	Iris versico
versicolor.19	5.1	3.8	1.5	0.3	setosa	Iris versico
versicolor.20	5.4	3.4	1.7	0.2	setosa	Iris versico
versicolor.21	5.1	3.7	1.5	0.4	setosa	Iris versico
versicolor.22	4.6	3.6	1.0	0.2	setosa	Iris versico
versicolor.23	5.1	3.3	1.7	0.5	setosa	Iris versico
versicolor.24	4.8	3.4	1.9	0.2	setosa	Iris versico
versicolor.25	5.0	3.0	1.6	0.2	setosa	Iris versico
versicolor.26	5.0	3.4	1.6	0.4	setosa	Iris versico
versicolor.27	5.2	3.5	1.5	0.2	setosa	Iris versico
versicolor.28	5.2	3.4	1.4	0.2	setosa	Iris versico
versicolor.29	4.7	3.2	1.6	0.2	setosa	Iris versico
versicolor.30	4.8	3.1	1.6	0.2	setosa	Iris versico
versicolor.31	5.4	3.4	1.5	0.4	setosa	Iris versico
versicolor.32	5.2	4.1	1.5	0.1	setosa	Iris versico
versicolor.33	5.5	4.2	1.4	0.2	setosa	Iris versico
versicolor.34	4.9	3.1	1.5	0.2	setosa	Iris versico
versicolor.35	5.0	3.2	1.2	0.2	setosa	Iris versico
versicolor.36	5.5	3.5	1.3	0.2	setosa	Iris versico
versicolor.37	4.9	3.6	1.4	0.1	setosa	Iris versico
versicolor.38	4.4	3.0	1.3	0.2	setosa	Iris versico
versicolor.39	5.1	3.4	1.5	0.2	setosa	Iris versico
versicolor.40	5.0	3.5	1.3	0.3	setosa	Iris versico
versicolor.41	4.5	2.3	1.3	0.3	setosa	Iris versico
versicolor.42	4.4	3.2	1.3	0.2	setosa	Iris versico

7.4 Junção de tabelas utilizando funções⁴

O pacote base do R fornece uma função que executa essa ação, chamada `merge()`. Porém, há alguns tipos de junções não podem ser executados com esta função, o que nos levará ao uso de vetores lógicos em conjunto com a função `interaction()`. Daremos exemplos com essas duas novas maneiras.

7.4.1 Dados para nossa prática

Utilizaremos três tabelas para esta prática:

- (1) O `data.frame tab1` possui nomes de famílias, gêneros e epítetos específicos de algumas angiospermas:

```
familia <- c("Burseraceae", "Solanaceae", "Sapindaceae", "Rubiaceae", "Lauraceae")
generos <- c("Protium", "Trattinnickia", "Dacryodes", "Duckeodendron", "Markea", "Solanum", "A
epitetos <- c("aracouchini", "burserifolia", "edilsonii", "cestroides", "ulei", "cyathophorum"
tab1 <- data.frame(familia = rep(familia, each = 3), "Anisophylleaceae", "Pentaphylacaceae")
```

- (2) O `data.frame tab2` contem um conjunto pequeno com alguns nomes de famílias, gêneros, e o nome de seus respectivos clados acima dos nomes de ordens segundo o [APG \(2016\)](#):

```
familia2 <- c("Burseraceae", "Solanaceae", "Sapindaceae", "Rubiaceae", "Annonaceae")
generos2 <- c("Protium", "Duckeodendron", "Thinouia", "Psychotria", "Guatteria")
clado <- c("Malvids", "Lamiids", "Malvids", "Lamiids", "Magnoliids")
tab2 <- data.frame(familia = familia2, genero = generos2, clado = clado, stringsAsFactors = F
```

⁴Texto publicado originalmente no blog de R.O.Perdiz (<https://www.ricardoperdiz.com/blog/2020-04-juncao-tbl/>)

Tabela 7.1: Tabela 1

familia	genero	epiteto
Burseraceae	Protium	aracouchini
Burseraceae	Trattinnickia	burserifolia
Burseraceae	Dacryodes	edilsonii
Solanaceae	Duckeodendron	cestroides
Solanaceae	Markea	ulei
Solanaceae	Solanum	cyathophorum
Sapindaceae	Allophylastrum	frutescens
Sapindaceae	Cupania	rubiginosa
Sapindaceae	Thinouia	myriantha
Rubiaceae	Psychotria	viridis
Rubiaceae	Duroia	eriopila
Rubiaceae	Cinchona	amazonica
Lauraceae	Ocotea	delicata
Lauraceae	Licaria	aureosericea
Lauraceae	Rhodostemonodaphne	recurva
Anisophylleaceae	Anisophyllea	manausensis
Pentaphylacaceae	Freziera	carinata

Tabela 7.2: Tabela 2

familia	genero	clado
Burseraceae	Protium	Malvids
Solanaceae	Duckeodendron	Lamiids
Sapindaceae	Thinouia	Malvids
Rubiaceae	Psychotria	Lamiids
Annonaceae	Guatteria	Magnoliids

Tabela 7.3: Tabela 3

	familia	genero	epiteto
1	Burseraceae	Protium	aracouchini
2	Burseraceae	Trattinnickia	burserifolia
3	Burseraceae	Dacryodes	edilsonii
7	Sapindaceae	Allophylastrum	frutescens
8	Sapindaceae	Cupania	rubiginosa
9	Sapindaceae	Thinouia	myriantha

- (3) O `data.frame` `tab3` corresponde à tabela 2, `tab2`, sem as famílias Solanaceae e Rubiaceae:

```
tab3 <- subset(tab1, familia %in% c("Burseraceae", "Sapindaceae"))
```

7.4.2 Maneira 3 - função `merge()`

O básico para entender a função `merge()` é saber que existem dois argumentos, `x` e `y`, que correspondem aos `data.frames` de entrada. Quando unimos tabelas, existem junções que adicionam variáveis, e junções que filtram variáveis. Vamos ver abaixo 4 tipos da primeira (**junção interna**, **junção à esquerda**, **junção à direita**, **junção total**), e dois tipos desta última (**semijunção** e **antijunção**).

7.4.2.1 Junção interna



Ao juntarmos tabelas `x` e `y`, temos todas as linhas de `x` em que há valores em comum com `y`, e todas as colunas de `x` e `y`. Se houver múltiplas *correspondências* entre `x` e `y`, todas as combinações retornam.

Em nosso exemplo, vamos unir as tabelas 1 e 2. Ambas possuem

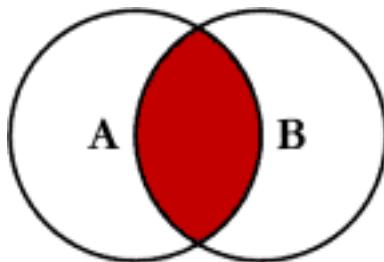


Figura 7.1: Fonte: www.sqlfromhell.com

em comum os identificadores `familia` e `genero`. Para facilitar o entendimento, vamos verificar primeiro cada tabela com cores para checar as correspondências entre `x` e `y` nas variáveis em comum:

Tabela 1

familia	
genero	
epiteto	
Burseraceae	
Protium	
aracouchini	
Burseraceae	
Trattinnickia	
burserifolia	
Burseraceae	
Dacryodes	
edilsonii	
Solanaceae	
Duckeodendron	
cestroides	
Solanaceae	

Markea
ulei
Solanaceae
Solanum
cyathophorum
Sapindaceae
Allophylastrum
frutescens
Sapindaceae
Cupania
rubiginosa
Sapindaceae
Thinouia
myriantha
Rubiaceae
Psychotria
viridis
Rubiaceae
Duroia
eriopila
Rubiaceae
Cinchona
amazonica
Lauraceae
Ocotea
delicata
Lauraceae

Licaria
aureosericea
Lauraceae
Rhodostemonodaphne
recurva
Anisophylleaceae
Anisophyllea
manausensis
Pentaphylacaceae
Freziera
carinata
Tabela 2
familia
genero
clado
Burseraceae
Protium
Malvids
Solanaceae
Duckeodendron
Lamiids
Sapindaceae
Thinouia
Malvids
Rubiaceae
Psychotria
Lamiids

Annonaceae

Guatteria

Magnoliids

Reparam que os valores em que há correspondência entre x e y estão coloridos de amarelo; para os em que não há correspondência, estão coloridos de vermelho. Agora, executemos a junção das duas tabelas:

```
merge(x = tab1, y = tab2)
```

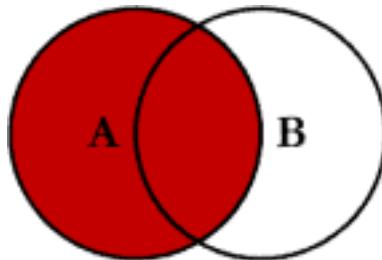
familia	genero	epiteto	clado
Burseraceae	Protium	aracouchini	Malvids
Rubiaceae	Psychotria	viridis	Lamiids
Sapindaceae	Thinouia	myriantha	Malvids
Solanaceae	Duckeodendron	cestroides	Lamiids

Vejam que houve a incorporação dos valores da coluna `epiteto`, presente apenas na tabela 2, em que há correspondência entre as tabelas 1 e 2. É importante notar que as famílias *Lauraceae*, *Anisophylleaceae*, e *Pentaphylacaceae* ficaram de fora, pois não são encontradas na tabela y , isto é, a tabela 2, assim como seus respectivos gêneros e epítetos associados a estes. Gêneros presentes na tabela 1 de famílias em comum entre ambas as tabelas também não foram incorporados nessa junção, pois não encontram correspondência na tabela 2: *Dacryodes*, *Trattinnickia*, *Markea*, *Solanum*, *Allophylastrum*, *Cupania*, *Duroia*, *Cinchona*. Revejam o conceito de *junção interna* para entender o porquê desse acontecimento.

7.4.2.2 Junção à esquerda



Ao juntarmos tabelas x e y , temos todas as linhas de x , e todas as colunas de x e y . Linhas em x sem correspondência em y terão valores NA adicionados nas novas colunas. Se houver múltiplas *correspondências* entre x e y , todas as combinações retornam.

**Figura 7.2:** Fonte: www.sqlfromhell.com

Continuaremos utilizando as tabelas 1 e 2. Como mostrado anteriormente, ambas possuem em comum os identificadores `familia` e `genero`. Chequemos novamente as cores das correspondências dentro de cada identificador, coloridas em amarelo:

Tabela 2

Tabela 7.4: Tabela 1

familia	genero	epiteto
Burseraceae	Protium	aracouchini
Burseraceae	Trattinnickia	burserifolia
Burseraceae	Dacryodes	edilsonii
Solanaceae	Duckeodendron	cestroides
Solanaceae	Markea	ulei
Solanaceae	Solanum	cyathophorum
Sapindaceae	Allophylastrum	frutescens
Sapindaceae	Cupania	rubiginosa
Sapindaceae	Thinouia	myriantha
Rubiaceae	Psychotria	viridis
Rubiaceae	Duroia	eriopila
Rubiaceae	Cinchona	amazonica
Lauraceae	Ocotea	delicata
Lauraceae	Licaria	aureosericea
Lauraceae	Rhodostemonodaphne	recurva
Anisophylleaceae	Anisophyllea	manausensis
Pentaphylacaceae	Freziera	carinata

familia
genero
clado
Burseraceae
Protium
Malvids
Solanaceae
Duckeodendron
Lamiids
Sapindaceae
Thinouia
Malvids
Rubiaceae
Psychotria
Lamiids
Annonaceae
Guatteria
Magnoliids

Em uma junção à esquerda, todas as linhas de `x` retornam após a junção. Para executar este tipo de junção, acrescentaremos um novo argumento, `all.x = TRUE`, indicando que manteremos todas as linhas de `x`, isto é, o `data.frame` à esquerda, que é a tabela 1.

```
merge(x = tab1, y = tab2, all.x = TRUE)
```

familia	genero	epiteto	clado
Anisophylleaceae	Anisophyllea	manausensis	NA
Burseraceae	Dacryodes	edilsonii	NA
Burseraceae	Protium	aracouchini	Malvids
Burseraceae	Trattinnickia	burserifolia	NA
Lauraceae	Licaria	aureosericea	NA
Lauraceae	Ocotea	delicata	NA
Lauraceae	Rhodostemonodaphne	recurva	NA
Pentaphylacaceae	Freziera	carinata	NA
Rubiaceae	Cinchona	amazonica	NA
Rubiaceae	Duroia	eriopila	NA
Rubiaceae	Psychotria	viridis	Lamiids
Sapindaceae	Allophylastrum	frutescens	NA
Sapindaceae	Cupania	rubiginosa	NA
Sapindaceae	Thinouia	myriantha	Malvids
Solanaceae	Duckeodendron	cestroides	Lamiids
Solanaceae	Markea	ulei	NA
Solanaceae	Solanum	cyathophorum	NA

Agora, temos uma nova situação. Para os valores de `x` sem correspondência em `y`, valores `NA` são acrescentados. Reparem na coluna `clado` e vejam que isso ocorreu apenas nesta variável. Por exemplo, vejam a família `Anisophylleaceae`. Ela ocorre apenas na tabela 1 e, portanto, não possui nenhum valor de `clado` associado a ela, pois esta variável ocorre apenas na tabela 2. Com a junção das tabelas, essa variável é retida, porém sem a existência de um valor para a família, é inserido então o valor `NA`. Temos também o caso de `Annonaceae`, presente na tabela 2. A família não é recuperada na junção interna, pois ela não existe na tabela 1 dentro da variável `familia` e, portanto, não apresenta correspondência com nenhum dado da tabela 1. Revejam o conceito de **junção à esquerda** para entender o porquê desse acontecimento.

7.4.2.3 Junção à direita



Ao juntarmos tabelas x e y , temos todas as linhas de y , e todas as colunas de x e y . Linhas em y sem correspondência em x terão valores NA adicionados nas novas colunas. Se houver múltiplas *correspondências* entre x e y , todas as combinações retornam.

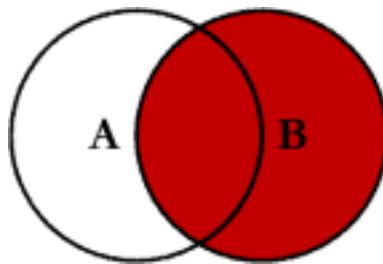


Figura 7.3: Fonte: www.sqlfromhell.com

De maneira oposta à junção à esquerda, na junção à direita são mantidas todas as linhas de y . Desta vez, o argumento a ser utilizado é `all.y = TRUE`. Antes de executar a junção, vamos checar novamente as variáveis em comum e correspondências entre as tabelas x e y :

Tabela 1
familia
genero
epiteto
Burseraceae
Protium
aracouchini
Burseraceae
Trattinnickia
burserifolia

Burseraceae

Dacryodes

edilsonii

Solanaceae

Duckeodendron

cestroides

Solanaceae

Markea

ulei

Solanaceae

Solanum

cyathophorum

Sapindaceae

Allophylastrum

frutescens

Sapindaceae

Cupania

rubiginosa

Sapindaceae

Thinouia

myriantha

Rubiaceae

Psychotria

viridis

Rubiaceae

Duroia

eriopila

Tabela 7.5: Tabela 2

familia	genero	clado
Burseraceae	Protium	Malvids
Solanaceae	Duckeodendron	Lamiids
Sapindaceae	Thinouia	Malvids
Rubiaceae	Psychotria	Lamiids
Annonaceae	Guatteria	Magnoliids

Rubiaceae
 Cinchona
 amazonica
 Lauraceae
 Ocotea
 delicata
 Lauraceae
 Licaria
 aureosericea
 Lauraceae
 Rhodostemonodaphne
 recurva
 Anisophylleaceae
 Anisophyllea
 manausensis
 Pentaphylacaceae
 Freziera
 carinata

Agora executaremos a junção com o comando abaixo. Não deixem de reparar no uso do argumento `all.y = TRUE`, pois ele é o responsável por agora manter todas as linhas da tabela 2 (`== y`):

```
merge(x = tab1, y = tab2, all.y = TRUE)
```

familia	genero	epiteto	clado
Annonaceae	Guatteria	NA	Magnoliids
Burseraceae	Protium	aracouchini	Malvids
Rubiaceae	Psychotria	viridis	Lamiids
Sapindaceae	Thinouia	myriantha	Malvids
Solanaceae	Duckeodendron	cestroides	Lamiids

Notem que agora todos os dados da tabela 2 foram mantidos. Houve a inserção de um valor `NA` para a família Annonaceae na variável `epiteto`, pois esta variável não está presente na tabela 2. Revejam o conceito de **junção à direita** para entender o porquê desse acontecimento.

7.4.2.4 Junção total



Ao juntarmos tabelas `x` e `y`, temos todas as linhas e colunas de `x` e `y`. Onde não houver valores correspondentes, valores `NA` serão colocados nesses lugares.

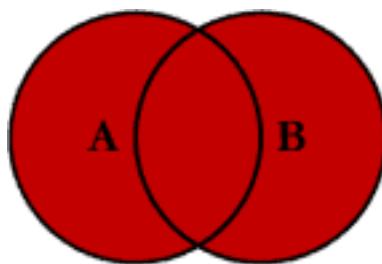


Figura 7.4: Fonte: www.sqlfromhell.com

Em uma junção total, uniremos todas as linhas de `x` e `y` utilizando o argumento `all = TRUE`.

```
merge(x = tab1, y = tab2, all = TRUE)
```

familia	genero	epiteto	clado
Anisophylleaceae	Anisophyllea	manausensis	NA
Annonaceae	Guatteria	NA	Magnoliids
Burseraceae	Dacryodes	edilsonii	NA
Burseraceae	Protium	aracouchini	Malvids
Burseraceae	Trattinnickia	burserifolia	NA
Lauraceae	Licaria	aureosericea	NA
Lauraceae	Ocotea	delicata	NA
Lauraceae	Rhodostemonodaphne	recurva	NA
Pentaphylacaceae	Freziera	carinata	NA
Rubiaceae	Cinchona	amazonica	NA
Rubiaceae	Duroia	eriopila	NA
Rubiaceae	Psychotria	viridis	Lamiids
Sapindaceae	Allophylastrum	frutescens	NA
Sapindaceae	Cupania	rubiginosa	NA
Sapindaceae	Thinouia	myriantha	Malvids
Solanaceae	Duckeodendron	cestroides	Lamiids
Solanaceae	Markea	ulei	NA
Solanaceae	Solanum	cyathophorum	NA

Reparem que valores `NA` são colocados nos valores da tabela 2 referentes à coluna `epiteto`, ausente na tabela 1. O mesmo se passa com valores da coluna `clado`, presente na tabela 2 e ausente na tabela 1. Revejam o conceito de **junção total** para entender o porquê desse acontecimento.

7.4.2.5 Semijunção



Ao juntarmos tabelas `x` e `y`, temos todas as linhas de `x` onde houver valores correspondentes em `y`, mantendo apenas colunas de `x`. É parecida com a junção interna, porém difere desta por nunca

duplicar valores de x , retornando sempre apenas valores de x que houver uma correspondência em y .

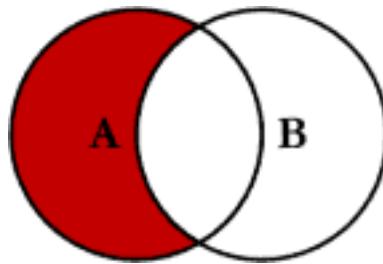


Figura 7.5: Fonte: www.sqlfromhell.com

A semijunção é muito similar à junção interna, diferindo desta por não incorporar as colunas de y , pois apenas utiliza esta tabela para filtrar os dados de x , constituindo-se então em um tipo de junção que filtra variáveis. Neste exemplo, utilizaremos as tabelas 1 e 3. Ambas compartilham as colunas `familia` e `genero`. Vamos checar primeiramente cada tabela e ver o que é compartilhado entre cada uma:

Tabela 1
familia
genero
epiteto
Burseraceae
Protium
aracouchini
Burseraceae
Trattinnickia
burserifolia
Burseraceae

Dacryodes
edilsonii
Solanaceae
Duckeodendron
cestroides
Solanaceae
Markea
ulei
Solanaceae
Solanum
cyathophorum
Sapindaceae
Allophylastrum
frutescens
Sapindaceae
Cupania
rubiginosa
Sapindaceae
Thinouia
myriantha
Rubiaceae
Psychotria
viridis
Rubiaceae
Duroia
eriopila
Rubiaceae

Tabela 7.6: Tabela 3

	familia	genero	epiteto
1	Burseraceae	Protium	aracouchini
2	Burseraceae	Trattinnickia	burserifolia
3	Burseraceae	Dacryodes	edilsonii
7	Sapindaceae	Allophylastrum	frutescens
8	Sapindaceae	Cupania	rubiginosa
9	Sapindaceae	Thinouia	myriantha

Cinchona

amazonica

Lauraceae

Ocotea

delicata

Lauraceae

Licaria

aureosericea

Lauraceae

Rhodostemonodaphne

recurva

Anisophylleaceae

Anisophyllea

manausensis

Pentaphylacaceae

Freziera

carinata

7.4.2.5.1 Maneira 4 - vetores lógicos e a função `interaction()`

Para executar uma semijunção com o pacote `base` do R, devemos fazer uso de vetores lógicos e da função `interaction()`, pois a função `merge()` não fornece uma maneira de se obter o que desejamos.

Vamos então à prática⁵. As colunas compartilhadas por ambas as tabelas serão nossas chaves:

```
chaves <- c("familia", "genero")
```

Partimos então para filtrar na tabela 1 a combinação de linhas para esse conjunto de colunas utilizando a função `interaction()` do pacote `base` do R:

```
interaction(tab1[, chaves])
```

```
## [1] Burseraceae.Protium           Burseraceae.Trattinnickia
## [3] Burseraceae.Dacryodes         Solanaceae.Duckeodendron
## [5] Solanaceae.Markea            Solanaceae.Solanum
## [7] Sapindaceae.Allophylastrum   Sapindaceae.Cupania
## [9] Sapindaceae.Thinouia          Rubiaceae.Psychotria
## [11] Rubiaceae.Duroia             Rubiaceae.Cinchona
## [13] Lauraceae.Ocotea             Lauraceae.Licaria
## [15] Lauraceae.Rhodostemonodaphne Anisophylleaceae.Anisophyllea
## [17] Pentaphylacaceae.Freziera
## 119 Levels: Anisophylleaceae.Allophylastrum ... Solanaceae.Trattinnickia
```

Essa função computa um vetor de fatores que representa a interação das colunas fornecidas na tabela 1. Se fizermos isso com a tabela 3, poderemos saber quais combinações ocorrem em ambas as tabelas.

⁵ Esta solução de semijunção é baseada no tutorial do pacote `poorman`⁶, recém-criado para emular as funções do pacote `dplyr`⁷.

```
interaction(tab3[, chaves])  
  
## [1] Burseraceae.Protium      Burseraceae.Trattinnickia  
## [3] Burseraceae.Dacryodes    Sapindaceae.Allophylastrum  
## [5] Sapindaceae.Cupania      Sapindaceae.Thinouia  
## 12 Levels: Burseraceae.Allophylastrum ... Sapindaceae.Trattinnickia
```

Agora utilizamos a mesma função `interaction` e o operador `%in%` para retornar um vetor lógico que utilizaremos para filtrar os valores da tabela 1 com correspondência na tabela 3.

```
linhas <- interaction(tab1[, chaves]) %in% interaction(tab3[, chaves])  
linhas
```

```
## [1] TRUE TRUE TRUE FALSE FALSE FALSE TRUE TRUE TRUE FALSE FALSE FALSE  
## [13] FALSE FALSE FALSE FALSE FALSE
```

```
tab1[linhas, ]
```

	familia	genero	epiteto
1	Burseraceae	Protium	aracouchini
2	Burseraceae	Trattinnickia	burserifolia
3	Burseraceae	Dacryodes	edilsonii
7	Sapindaceae	Allophylastrum	frutescens
8	Sapindaceae	Cupania	rubiginosa
9	Sapindaceae	Thinouia	myriantha

7.4.2.6 Antijunção



Retorna todas as linhas de x em que não há correspondência em y , mantendo apenas colunas de x .

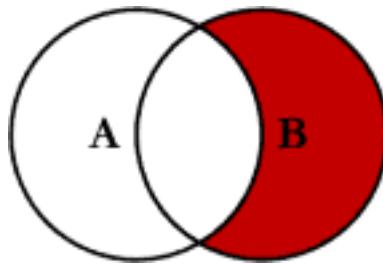


Figura 7.6: Fonte: www.sqlfromhell.com

Uma antijunção é ligeiramente diferente de uma semijunção pois ela retorna todas as linhas de x que não aparecem em y . Portanto, podemos utilizar o inverso de nosso vetor lógico `linhas` e utilizar este inverso para filtrar as linhas da tabela 1 e ter nossa tabela antijunção entre x e y :

```
antilinhas <- !linhas
tab1[antilinhas, ]
```

	familia	genero	epiteto
4	Solanaceae	Duckeodendron	cestroides
5	Solanaceae	Markea	ulei
6	Solanaceae	Solanum	cyathophorum
10	Rubiaceae	Psychotria	viridis
11	Rubiaceae	Duroia	eriopila
12	Rubiaceae	Cinchona	amazonica
13	Lauraceae	Ocotea	delicata
14	Lauraceae	Licaria	aureosericea
15	Lauraceae	Rhodostemonodaphne	recurva
16	Anisophylleaceae	Anisophyllea	manausensis
17	Pentaphylacaceae	Freziera	carinata

7.5 Para saber mais

- Join in R⁸;
 - Tudo sobre Joins (merge) em R⁹;
 - Join com dplyr¹⁰;
 - Entendendo o JOIN do SQL (ou Junções)¹¹ - Obtive as imagens aqui apresentadas desta página. Há uma boa explicação com SQL como pano de fundo para operações de junções de tabela.
-

7.6 Exercícios

- Resolva o exercício 104.01 Sintetizando dados¹².
- Resolva o exercício 103.05 Classes de Objetos¹³.
- Resolva o exercício 103.04 Lendo e salvando seus dados¹⁴.
- Resolva o exercício 103.06 Acrescentando dados¹⁵.
- Resolva o exercício 103.8 De vetor a data.frame ordenado¹⁶.

⁸<http://www.datasciencemakesimple.com/join-in-r-merge-in-r/>

⁹<https://www.fulljoin.com.br/posts/2016-05-12-tudo-sobre-joins/>

¹⁰<https://rpubs.com/CristianaFreitas/311735>

¹¹<https://www.sqlfromhell.com/entendendo-join-sql/>

¹²<http://notar.ib.usp.br/exercicio/14>

¹³<http://notar.ib.usp.br/exercicio/21>

¹⁴<http://notar.ib.usp.br/exercicio/22>

¹⁵<http://notar.ib.usp.br/exercicio/24>

¹⁶<http://notar.ib.usp.br/exercicio/35>



8

Manipulação de textos, arquivos e pastas

8.1 Funções para manipulação e busca de textos

8.1.1 Colar ou concatenar textos

A função `paste()` concatena textos.

```
?paste # veja o help dessa função
```

Vamos criar dois vetores de texto, `txt` e `txt2`, para, em seguida, concatená-los com esta função:

```
# um vetor de texto
txt <- c("banana", "maça", "pera")
# outro vetor
txt2 <- LETTERS[1:3]
```

```
# concatenamos os textos par a par
paste(txt, txt2)
```

```
## [1] "banana A" "maça B"   "pera C"
```

```
# mudamos o separador  
paste(txt, txt2, sep = "-")
```

```
## [1] "banana-A" "maça-B"    "pera-C"
```

Podemos também unir os elementos contidos em um vetor em um único elemento, separados por algum símbolo. Basta usar o argumento `collapse` e indicar qual será o elemento que unirá os elementos contidos em um vetor:

```
# imagine que queremos juntar os elementos de um vetor  
# num único texto, separados por ";"  
paste(txt, collapse = ";")
```

```
## [1] "banana;maça;pera"
```

```
paste(txt, collapse = "/")
```

```
## [1] "banana/maça/pera"
```

```
paste(txt, collapse = " e ")
```

```
## [1] "banana e maça e pera"
```

```
paste(txt, collapse = " mais ")
```

```
## [1] "banana mais maça mais pera"
```

8.1.2 Quebrar ou desconcatenar textos

A função `strsplit()` quebra um vetor de texto segundo um separador determinado pelo argumento `split`, e retorna uma lista como resultado.

```
?strsplit
```

```
txt <- "Este é um texto com várias palavras"  
strsplit(txt, split = " ")
```

```
## [[1]]  
## [1] "Este"     "é"       "um"      "texto"    "com"      "várias"   "palavras"
```

Note que o resultado da ação é um objeto de classe lista:

```
class(strsplit(txt, split = " "))
```

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

Então, para pegar o resultado da ação da função `strsplit()`, devemos usar o que aprendemos em **indexação de listas**. Para indexar listas, devemos usar duplo colchetes `[[n]]`, em que `n` é o número do elemento que desejamos reter. Vejamos abaixo que retemos o resultado de `strsplit()` no vetor `txt` em um objeto `pl`:

```
pl <- strsplit(txt, split = " ")  
pl
```

```
## [[1]]  
## [1] "Este"     "é"       "um"      "texto"    "com"      "várias"   "palavras"
```

Para pegar o tamanho do objeto `pl`, usamos `length(pl)` que nos dá o resultado de `1`. Por isso, se desejamos reter essa ação como um vetor de texto, nosso `n` é `1`, logo, `[[1]]`:

```
pl <- strsplit(txt, split = " ")[[1]]  
pl
```

```
## [1] "Este"     "é"       "um"      "texto"    "com"     "várias"   "palavras"
```

Agora o objeto `pl` é um vetor de texto e de tamanho corresponde ao número de palavras contidas no vetor:

```
class(pl)
```

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

```
length(pl) # numero de palavras no texto
```

```
## [1] 7
```

8.1.2.1 Um exemplo prático

Suponha que você tenha uma tabela em que o nome da espécie esteja em uma única coluna e você queira separar o gênero do epíteto. Podemos usar a função `strsplit()` para resolver essa situação. Vejamos:

```
# um dado qualquer  
spp <- c("Ocotea guianensis", "Ocotea longifolia", "Licaria tenuicarpa")  
dap <- c(10, 20, 30)  
dd <- data.frame(ESPECIE = spp, DAP = dap, stringsAsFactors = F)  
dd
```

ESPECIE	DAP
Ocotea guianensis	10
Ocotea longifolia	20
Licaria tenuicarpa	30

Vamos criar uma função para receber um vetor de texto e quebrá-lo segundo o separador `sep = " "`, isto é, vai quebrar a palavra onde houver um espaço " ".

```
# criamos um função
peganome <- function(x, qual = 1, sep = " ") {
  # pega o texto e quebra segundo o separador informado
  xx <- strsplit(x, split = sep)[[1]]
  # retorna o valor
  return(xx[qual])
}
```

Com a função criada acima, `peganome()`, vamos utilizar a função `lapply()` para aplicar esta função sobre cada elemento do vetor `dd$ESPECIE`:

```
# usamos essa função com o lapply para pegar o gênero
lapply(dd$ESPECIE, peganome, qual = 1)
```

```
## [[1]]
## [1] "Ocotea"
##
## [[2]]
## [1] "Ocotea"
##
## [[3]]
## [1] "Licaria"
```

Repare que o resultado da ação é uma lista, pois `lapply()` sempre

retorna uma lista! Podemos converter esta lista em um vetor usando o procedimento abaixo:

```
# como é uma lista transformamos num vetor
gg <- as.vector(lapply(X = dd$ESPECIE, FUN = peganome, qual = 1), mode = "character")
gg
```

```
## [1] "Ocotea"  "Ocotea"  "Licaria"
```

Podemos também gerar um vetor de texto simplesmente utilizando a função `vapply()` para aplicar a função `peganome()` em cada elemento de `dd$ESPECIE`, informando à função `vapply()` que esperamos como retorno um vetor de texto de tamanho 1 (`FUN.VALUE = as.character(1)`). O resultado é o mesmo obtido no objeto `gg`:

```
vapply(X = dd$ESPECIE, FUN = peganome, FUN.VALUE = as.character(1), qual = 1)

## Ocotea guianensis  Ocotea longifolia  Licaria tenuicarpa
##          "Ocotea"           "Ocotea"         "Licaria"
```

Agora que temos o nome do gênero separado do epíteto, vamos guardar o nome do gênero estocado em `gg` no `data.frame dd`:

```
# ADICIONAMOS NO OBJETO ORIGINAL
dd$GENERO <- gg
dd
```

ESPECIE	DAP	GENERO
Ocotea guianensis	10	Ocotea
Ocotea longifolia	20	Ocotea
Licaria tenuicarpa	30	Licaria

Agora vamos pegar o epíteto da espécie, mudando o argumento `qual` da função `peganome()`:

```
# usamos essa função com o lapply para pegar o epíteto
spp <- as.vector(lapply(dd$ESPECIE, peganome, qual = 2), mode = "character")
spp
```

```
## [1] "guianensis" "longifolia" "tenuicarpa"
```

Repetimos então o procedimento de guardar o epíteto spp em nosso data.frame dd:

```
# adiciona
dd$SPP <- spp
# mostra
dd
```

ESPECIE	DAP	GENERO	SPP
Ocotea guianensis	10	Ocotea	guianensis
Ocotea longifolia	20	Ocotea	longifolia
Licaria tenuicarpa	30	Licaria	tenuicarpa

```
str(dd)
```

```
## 'data.frame':   3 obs. of  4 variables:
## $ ESPECIE: chr  "Ocotea guianensis" "Ocotea longifolia" "Licaria tenuicarpa"
## $ DAP    : num  10 20 30
## $ GENERO : chr  "Ocotea" "Ocotea" "Licaria"
## $ SPP    : chr  "guianensis" "longifolia" "tenuicarpa"
```

8.1.3 Altera caixa do texto

As funções toupper() e tolower() mudam a caixa do texto, isto é, convertem o texto para totalmente maiúsculas ou minúsculas, respectivamente.

```
txt <- "Um texto QUALQUER"
# transforma para caixa baixa
tolower(txt)
```

```
## [1] "um texto qualquer"
```

```
# caixa alta
toupper(txt)
```

```
## [1] "UM TEXTO QUALQUER"
```

```
# apenas a primeira letra em maiusculo
# temos que construir nossa função para isso
mudatexto <- function(x) {
  xx <- strsplit(x, split = "")[[1]]
  xx <- tolower(xx)
  xx[1] <- toupper(xx[1])
  xx <- paste(xx, collapse = "")
  return(xx)
}
txt
```

```
## [1] "Um texto QUALQUER"
```

```
mudatexto(txt)
```

```
## [1] "Um texto qualquer"
```

8.1.4 Remove espaços em branco

A função `trimws()` remove espaços em branco no início e final do texto. Repare que isso ocorre no objeto `txt2` criado abaixo:

```
txt2 <- "    Outro texto com espacos em branco no inicio e fim    "
```

Podemos removê-los usando a função `trimws()` e informando para o argumento `which` em qual(is) posição(ões) queremos remover os espaços em branco. Vejamos:

```
# Remove dos dois lados
trimws(txt2, which = "both")
```

```
## [1] "Outro texto com espacos em branco no inicio e fim"
```

```
# remove do lado direito
trimws(txt2, which = "right")
```

```
## [1] "    Outro texto com espacos em branco no inicio e fim"
```

```
# Remove do lado esquerdo
trimws(txt2, which = "left")
```

```
## [1] "Outro texto com espacos em branco no inicio e fim    "
```

8.1.5 Substitui ou pega parte de texto

A função `gsub()` busca padrões de um texto para substituí-los por outro em um vetor de caracteres.

Já a função `substr()` extrai ou substitui pedaços de um texto em um vetor de caracteres.

Por fim, a função `nchar()` conta o número de caracteres contidos em um vetor de caracteres.

```
# substitui palavras ou parte de palavras
txt <- "Um texto que contém muita COISA"
gsub("m", "M", txt)
```

```
## [1] "UM texto que contéM Muita COISA"
```

```
gsub("que", "o qual", txt)
```

```
## [1] "Um texto o qual contém muita COISA"
```

```
# pega uma parte do texto
substr(txt, 1, 10)
```

```
## [1] "Um texto q"
```

```
substr(txt, 11, 20)
```

```
## [1] "ue contém "
```

```
# qual o numero de caracteres num texto  
nchar(txt)  
  
## [1] 31  
  
substr(txt, nchar(txt) - 10, nchar(txt)) # dez ultimos caracteres  
  
## [1] "muita COISA"
```

8.1.6 Expressões Regulares

Expressões regulares¹ são modelos lógicos para buscar caracteres em textos. É uma ferramenta muito poderosa usada em computação. Não é fácil entender e usar expressões regulares, mas é muito poderoso e importante conhecer os recursos. Aqui apresentamos apenas um exemplo muito simples do uso de expressões regulares.

```
txt <- "Um texto com varios numeros 1 2 9 no meio e 5 7 8 20 3456"  
txt  
  
## [1] "Um texto com varios numeros 1 2 9 no meio e 5 7 8 20 3456"  
  
# imagine que eu queira apagar todos os números desse texto  
gsub("[0-9]", "", txt) # paga números de 0 a 9  
  
## [1] "Um texto com varios numeros      no meio e      "
```

¹https://pt.wikipedia.org/wiki/Express%C3%A3o_regular

```
# todas as letras minusculas  
gsub("[a-z]", "", txt)
```

```
## [1] "U      1 2 9      5 7 8 20 3456"
```

```
# todas as letras  
gsub("[a-z|A-Z]", "", txt)
```

```
## [1] "      1 2 9      5 7 8 20 3456"
```

```
# todas as minúculas e numeros  
gsub("[a-z|0-9]", "", txt)
```

```
## [1] "U      "
```

Entender expressões regulares (*regular expressions*) é muito importante porque é fonte frequente de erros quando escrevemos códigos. Algumas funções que buscam por padrões interpretam expressões regulares para buscar por padrões em objetos de texto. Por exemplo, `gsub()` e `grep()` são duas funções parecidas, que buscam por um padrão (argumento `pattern`) em textos. Enquanto `gsub()` busca e permite substituir o padrão por outro texto, `grep()` retorna o índice do elemento que contém o padrão de interesse. O argumento `pattern`, citado anteriormente, pode ser um texto simples ou uma expressão regular, então é importante que você entenda um pouco sobre isso. Expressões regulares podem ser bem complexas e até existem sites que lhe permitem entender e gerar uma expressão regular para uma busca específica (veja abaixo na seção **Para saber mais**).

8.1.7 Remove Acentos

Remover acentos e padronizar textos podem ser importantes quando estamos comparando dados. Por exemplo, quando queremos verificar se os nomes que aparecem em uma coluna estão padronizados, devemos checar se há duplicidade de palavras devido à presença de um acento em uma palavra e ausência em outra (e.g., odores “cítrico” e “citrico” são computados como duas palavras diferentes). Não há uma função específica, mas como tudo pode ser feito no R, uma solução é mostrada aqui.

```
txt <- "Palavrão ou bobalhão têm acentos"  
# convertemos quebrando acentos  
xx <- iconv(txt, to = "ASCII//TRANSLIT")  
xx
```

```
## [1] "Palavraq ou bobalhaq tem acentos"
```

```
# apagamos os acentos sozinhos  
xx <- gsub("[~|^|~|\\"|'|`]", "", xx)  
xx
```

```
## [1] "Palavrao ou bobalhao tem acentos"
```

```
# podemos colocar isso numa função  
removeacentos <- function(x) {  
  xx <- iconv(x, to = "ASCII//TRANSLIT")  
  xx <- gsub("[~|^|~|\\"|'|`]", "", xx)  
  return(xx)  
}  
  
# usando a função criada  
removeacentos(txt)
```

```
## [1] "Palavrao ou bobalhao tem acentos"
```

```
txt2 <- "macarrão também tem acentos, né?"  
removeacentos(txt2)
```

```
## [1] "macarrao tambem tem acentos, ne?"
```

8.1.8 Metacaracteres

Metacaracteres são caracteres que em uma expressão regular apresentam um significado especial e por isso não são interpretados como tal. Alguns metacaracteres são:

- . encontra qualquer coisa que exista;
- + encontra o item que precede esse metacaractere uma ou mais vezes;
- * encontra o item que precede esse metacaractere zero ou mais vezes;
- ^ encontra qualquer espaço vazio no início da linha. Quando usado numa expressão de classe de caractere (veja abaixo), encontra qualquer caractere exceto aqueles que seguem esse metacaractere;
- \\$ encontra qualquer espaço vazio no final da linha;
- | operador que significa ou e é utilizado em buscas do tipo `busque isso OU aquilo`;
- (e) para agrupamentos;
- [e] definem classes de caracteres em expressões regulares (veja abaixo na seção **Classes de caracteres**).

Suponha que você importe ao R um conjunto de dados de uma planilha usando, por exemplo, a função `read.table()`, e que os nomes das colunas no original tinham muitos espaços em branco. Como o R não

aceita espaços em branco, ele substitui espaços por pontos ao utilizar esta função, então os nomes das colunas do `data.frame` dos dados terão muitos pontos, algumas vezes em sequência. Suponha também que isso acontece toda vez que você importa dados ao R e já está cansado desse comportamento e você quer colocar no seu script um comando que elimina esses pontos, de forma a não se preocupar com isso no editor de planilhas onde estão os seus dados.

```
# vamos criar um data.frame que simule esta situação
Número.....da.....árvore <- c(1, 10, 15)
Diâmetro..cm. <- c(10, 15, 20)
Altura.....m.... <- c(15, 20, 39)
dados <- data.frame(Número.....da.....árvore, Diâmetro..cm., Altura.....m....)
class(dados)
```

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

```
dim(dados)
```

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

```
colnames(dados)
```

```
## [1] "Número.....da.....árvore" "Diâmetro..cm."
## [3] "Altura.....m...."
```

```
# nome de colunas, que também neste caso pode ser obtido com
names(dados)
```

```
## [1] "Número.....da.....árvore" "Diâmetro..cm."
## [3] "Altura.....m...."
```

```
nn <- names(dados)
# então digamos que eu queira substituir pontos multiplos por apenas 1 ponto. como eu quero m
gsub(pattern = "...", replacement = ".", nn)
```

```
## [1] "....." "....." "....."
```

```
# estranho né? tudo virou ponto
```

isso é porque . é um metacaractere e o R não interpreta isso como ponto. Ponto significa pegar

pelo fato de ponto ser um metacaractere, precisamos indicar que queremos o caractere ponto e
gsub(pattern = "\\.\\.", replacement = ".", nn)

```
## [1] "Número...da....árvore" "Diâmetro.cm." "Altura...m.."
```

diminuiu o número de pontos, mas preciso fazer isso várias vezes para ficar com um único ponto
nn2 <- gsub(pattern = "\\\\.", replacement = ".", nn)
nn2 <- gsub(pattern = "\\\\\.", replacement = ".", nn2)
nn2

```
## [1] "Número..da..árvore" "Diâmetro.cm." "Altura..m."
```

```
nn2 <- gsub(pattern = "\\\\.", replacement = ".", nn2)
nn2
```

```
## [1] "Número.da.árvore" "Diâmetro.cm." "Altura.m."
```

```
# pronto consegui
```

note que isso fica restrito a quantos pontos tem nos meus dados, e portanto, não é um método

seria mais fácil adicionar um metacaractere (+) e fazer isso uma única vez, sem necessidade de
nn

```
## [1] "Número.....da.....árvore" "Diâmetro..cm."  
## [3] "Altura.....m...."
```

```
nn2 <- gsub(pattern = "\\\.+", replacement = ".", nn)  
nn2
```

```
## [1] "Número.da.árvore" "Diâmetro.cm."      "Altura.m."
```

```
# procurar por qualquer letra+  
txt <- "qualqueeeeeer palavra"  
gsub("e+", "E", txt)
```

```
## [1] "qualquEr palavra"
```

```
# da mesma forma se eu quiser buscar pelo caractere + ao invés de usar o metacaractere, preciso  
txt <- c("um texto com simbolo +", "sem o simbolo")  
gsub("+", "MAIS", txt) # nao funciona
```

```
## [1] "MAISuMAISmMAIS MAIStMAISEMAISxMAIStMAISOmais MAIScMAISOmais MAISsMAISiMAISmMAISbMAISlMAISOmais"  
## [2] "MAISsMAISEMAISmMAIS MAISOmais MAISsMAISiMAISmMAISbMAISOmaislMAISOmais"
```

```
gsub("\\\+", "MAIS", txt) # funciona  
  
## [1] "um texto com simbolo MAIS" "sem o simbolo"
```

8.1.9 Classes de caracteres

Algumas classes de caracteres podem ser utilizadas em buscas com expressões regulares:

- [0-9] - busca números no vetor de caracteres;
- [a-z] - busca apenas caracteres minúsculos no vetor de caracteres;
- [A-Z] - busca apenas caracteres maiúsculos no vetor de caracteres;
- [a-zA-Z] - busca caracteres do alfabeto no vetor de caracteres;
- [^a-zA-Z] - busca não alfábéticos no vetor de caracteres;
- [a-zA-Z0-9] - busca elementos alfa-numéricos no vetor de caracteres;
- [\t\n\r\f\v] - busca espaçamentos no vetor de caracteres. Espaçamentos podem ser quebras de linha, tabulações etc;
- []\$*+.?[^{|(\#%&~_/.!,:;\"}@-]} - busca caracteres de pontuação no vetor de caracteres.

Percebe-se que estas classes permitem fazer buscas complexas. Suponha que você precisa substituir todos os valores que não contenham letras ou números por NA:

```
# suponha um vetor de palavras (nomes de cores neste exemplo)  
vt <- colors()  
# vejamos os 30 primeiros valores  
head(vt, 30)
```

```
## [1] "white"      "aliceblue"    "antiquewhite" "antiquewhite1"  
## [5] "antiquewhite2" "antiquewhite3" "antiquewhite4" "aquamarine"  
## [9] "aquamarine1"   "aquamarine2"   "aquamarine3"   "aquamarine4"  
## [13] "azure"        "azure1"       "azure2"       "azure3"  
## [17] "azure4"        "beige"        "bisque"       "bisque1"  
## [21] "bisque2"       "bisque3"       "bisque4"       "black"  
## [25] "blanchedalmond" "blue"        "blue1"        "blue2"  
## [29] "blue3"        "blue4"
```

podemos mostrar os valores que contém branco:

```
grep("white", vt, value = TRUE)
```

```
## [1] "white"      "antiquewhite" "antiquewhite1" "antiquewhite2"  
## [5] "antiquewhite3" "antiquewhite4" "floralwhite"   "ghostwhite"  
## [9] "navajowhite"  "navajowhite1" "navajowhite2" "navajowhite3"  
## [13] "navajowhite4" "whitesmoke"
```

se eu coloco white numa expressão regular

vejamos os 30 primeiros valores

```
grep("[white]", vt, value = TRUE)[1:30]
```

```
## [1] "white"      "aliceblue"    "antiquewhite" "antiquewhite1"  
## [5] "antiquewhite2" "antiquewhite3" "antiquewhite4" "aquamarine"  
## [9] "aquamarine1"   "aquamarine2"   "aquamarine3"   "aquamarine4"  
## [13] "azure"        "azure1"       "azure2"       "azure3"  
## [17] "azure4"        "beige"        "bisque"       "bisque1"  
## [21] "bisque2"       "bisque3"       "bisque4"       "blanchedalmond"  
## [25] "blue"         "blue1"        "blue2"        "blue3"  
## [29] "blue4"        "blueviolet"
```

```
# neste caso significa pegue qualquer elemento que contenha uma das letras indicadas: w, h, i,
```

```
# pega só cores que tenham número  
# vejamos os 30 primeiros valores  
grep("[0-9]", vt, value = TRUE)[1:30]
```

```
## [1] "antiquewhite1" "antiquewhite2" "antiquewhite3" "antiquewhite4"  
## [5] "aquamarine1"   "aquamarine2"   "aquamarine3"   "aquamarine4"  
## [9] "azure1"        "azure2"       "azure3"       "azure4"  
## [13] "bisque1"       "bisque2"      "bisque3"      "bisque4"  
## [17] "blue1"         "blue2"        "blue3"        "blue4"  
## [21] "brown1"        "brown2"       "brown3"       "brown4"  
## [25] "burlywood1"   "burlywood2"   "burlywood3"   "burlywood4"  
## [29] "cadetblue1"   "cadetblue2"
```

```
# neste caso é o mesmo que nao-alfabeticos  
# vejamos os 30 primeiros valores  
grep("[^a-zA-Z]", vt, value = TRUE)[1:30]
```

```
## [1] "antiquewhite1" "antiquewhite2" "antiquewhite3" "antiquewhite4"  
## [5] "aquamarine1"   "aquamarine2"   "aquamarine3"   "aquamarine4"  
## [9] "azure1"        "azure2"       "azure3"       "azure4"  
## [13] "bisque1"       "bisque2"      "bisque3"      "bisque4"  
## [17] "blue1"         "blue2"        "blue3"        "blue4"  
## [21] "brown1"        "brown2"       "brown3"       "brown4"  
## [25] "burlywood1"   "burlywood2"   "burlywood3"   "burlywood4"  
## [29] "cadetblue1"   "cadetblue2"
```

```
# que tenham caracteres maiusculos  
grep("[A-Z]", vt, value = TRUE) # nao tem nenhum neste vetor
```

```
## character(0)
```

```
# minusculos  
# vejamos os 30 primeiros valores  
grep("[a-z]", vt, value = TRUE)[1:30]
```

```
## [1] "white"        "aliceblue"     "antiquewhite"  "antiquewhite1"  
## [5] "antiquewhite2" "antiquewhite3" "antiquewhite4" "aquamarine"  
## [9] "aquamarine1"   "aquamarine2"   "aquamarine3"   "aquamarine4"  
## [13] "azure"         "azure1"       "azure2"       "azure3"  
## [17] "azure4"        "beige"        "bisque"       "bisque1"  
## [21] "bisque2"       "bisque3"      "bisque4"      "black"  
## [25] "blanchedalmond" "blue"        "blue1"       "blue2"  
## [29] "blue3"         "blue4"
```

8.1.10 Barra invertida \

O uso da barra invertida \ (em inglês, *backslash*) indica ao R que a expressão regular espera lidar com tabulações, quebra de linhas e outros símbolos especiais. Isso é fundamental se você quer incluir/buscar num texto por aspas, parênteses, colchetes, barras, metacaracteres, etc.

```
?Quotes # leia atentamente esse help  
# criar um texto com aspas
```

Se você tentar executar o comando abaixo, perceberá que ele não funciona, porque as aspas são utilizadas para abrir e fechar textos.

```
txt = "Um texto que tem "aspas" no meio"
```

Porém, podemos combinar aspas simples e duplas para fazer isso:

```
txt <- 'Um texto que tem "aspas" no meio'  
txt
```

```
## [1] "Um texto que tem \"aspas\" no meio"
```

```
# veja que o R adicionou uma \ antes das aspas do texto, criando um escape character, que indica  
# entao eu posso fazer isso  
txt <- "Um texto que tem \"aspas\" "  
txt
```

```
## [1] "Um texto que tem \"aspas\" "
```

```
# também funciona  
  
# isso é importante quando buscamos valores  
txt <- c("com \"aspas\"", "outro objeto")  
grep("\\"", txt) # qual elemento tem aspas
```

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

```
txt[grep("\\"", txt)]
```

```
## [1] "com \"aspas\""
```

```
# barras  
txt <- c("com \\ barras", "sem barras")  
txt
```

```
## [1] "com \\ barras" "sem barras"
```

```
grep("\\\\\", txt) # isso é mais complicado, veja o número de barras que preciso para buscar pa
```

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

```
# parentesis  
txt <- c("sem parenteses", "com ()")
```

```
grep("(", txt) # isso não funciona por parentesis é usado para abrir e fechar funcoes, o R se  
grep("\\\\(", txt) # assim funciona  
grep("\\\\)", txt)
```

8.2 Manipulação de pastas e arquivos

As funções abaixo auxiliam a checagem de arquivos dentro de pastas, possibilitam a criação de pastas, e a cópia de arquivos dentro do sistema. Dominar essas funções e algumas outras correlatas são importantíssimas, por exemplo, no ramo da bioinformática. No dia a dia, dominá-las também possibilitará que muito tempo seja pouparado em atividades maçantes, como copiar ou mover muitos arquivos de uma pasta a outra.

Vamos analisar o uso das funções `dir()`, `dir.create()` e `file.copy()` (vejam a seção [Para saber mais](#) para mais informações). Não esqueçam de checar o `?` de cada função (e.g., `?dir`) para auxiliar no entendimento do funcionamento de cada uma das funções.

```
# meu caminho
caminho <- "~/Documents/DOC/PROJETO_DOC/R/pkgs/BOT89-introR/tutorial/"
# lista todos os arquivos no caminho que sejam pdfs
arqs <- dir(caminho, pattern = ".pdf")

# renomeia os arquivos adicionando a data no final do nome
# cria função para gerar novos nomes
novonome <- function(x) {
  # separa as palavras
  xx <- strsplit(x, ".pdf")[[1]]
  # cola a data
  xx1 <- paste(xx, Sys.Date(), sep = "_")
  # junta novamente tudo
  xx <- paste(xx1, ".pdf", sep = "")
  # retorna o novo nome
  return(xx)
}

# agora copia cada arquivo para uma subpasta em caminho
novapasta <- paste(caminho, "/pdfs", sep = "")
dir.create(novapasta)

# salva os arquivos com mesmo nome na nova pasta
# cria uma função para isso
salvaarq <- function(arq, origem, destino) {
  from <- paste(origem, "/", arq, sep = "")
  to <- paste(destino, "/", novonome(arq), sep = "")
  file.copy(from, to)
}

# aplica a função a todos os arquivos
```

```
sapply(arqs, salvaarq, origem = caminho, destino = novapasta)

# pronto os arquivos devem ter sido copiados
dir(novapasta)
```

8.3 Para saber mais:

- (1) RegExr is an online tool to learn, build, & test Regular Expressions (RegEx / RegExp)² - Página que ajuda o usuário a construir uma expressão regular.
- (2) Vídeoaulas - temos vídeosaulas disponíveis, gravadas com o material desta aula. Acesse-as nos *links* abaixo:
 - parte I³;
 - parte II⁴;
 - Manipulação de pastas e arquivos⁵.
- (3) Pacote fs⁶ - Um pacote que agrupa funções para manipulações de arquivos e pastas, e que apresenta algumas melhorias sobre o conjunto de funções do pacote base do R. Além disso, há alguns tutoriais disponíveis.

²<http://regexr.com>

³<https://youtu.be/at5XpBAT1sI>

⁴<https://youtu.be/j6O6e96qmEc>

⁵<https://youtu.be/TbY5nEQTwLw>

⁶<https://github.com/r-lib/fs>



9

Amostragens aleatórias

9.1 Funções para gerar permutações e simular dados

A função `sample()` aleatoriza valores em um vetor, com ou sem repetição, ou amostra um certo número de valores aleatoriamente de um vetor de valores.

```
# ALEATORIZANDO DADOS SEM AMOSTRAGEM  
?sample
```

```
# suponha um vetor de numeros  
v1 <- 1:20  
v1 # os numeros sao sequenciais
```

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

```
# se quiser embaralhar aleatoriamente  
sample(v1)
```

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

```
sample(v1) # cada vez será diferente
```

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

```
sample(v1) # diferente de novo
```

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

```
# no exemplo acima, os numeros sao apenas embaralhados
table(v1) # cada valor só aparece uma vez
```

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

```
table(sample(v1)) # aqui tambem
```

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

```
# se eu quiser amostras com repeticao
table(sample(v1, replace = TRUE))
```

1	2	3	4	5	6	7	8	9	10	13	15	17	18	20					
1	1	1	1	1	3	2	1	1	1	1	1	1	1	1	1	2	2		

```
# ele repetiu alguns valores aleatoriamente
table(sample(v1, replace = TRUE)) # vai ser diferente toda vez que voce executa
```

1	2	4	5	6	9	12	14	16	17	18	20
3	2	1	1	2	2	1	1	1	2	2	2

```
# mas o comprimento é sempre o mesmo  
length(v1)
```

```
## [1] 20
```

```
length(sample(v1))
```

```
## [1] 20
```

```
length(sample(v1, replace = T))
```

```
## [1] 20
```

```
# agora com textos  
table(LETTERS)
```

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

```
table(sample(LETTERS, replace = T))
```

A	B	D	E	F	G	I	J	K	M	Q	R	S	X	Y	Z
1	1	2	1	1	1	1	1	2	1	1	5	3	1	1	3

Amostraremos n valores de um vetor qualquer:

```
# agora amostra um número qualquer de valores de um vetor qualquer:  
umvetor <- seq(from = 0, to = 1000, by = 0.1)  
length(umvetor)
```

```
## [1] 10001
```

```
head(umvetor, 10) # primeiros 10 elementos
```

```
## [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
```

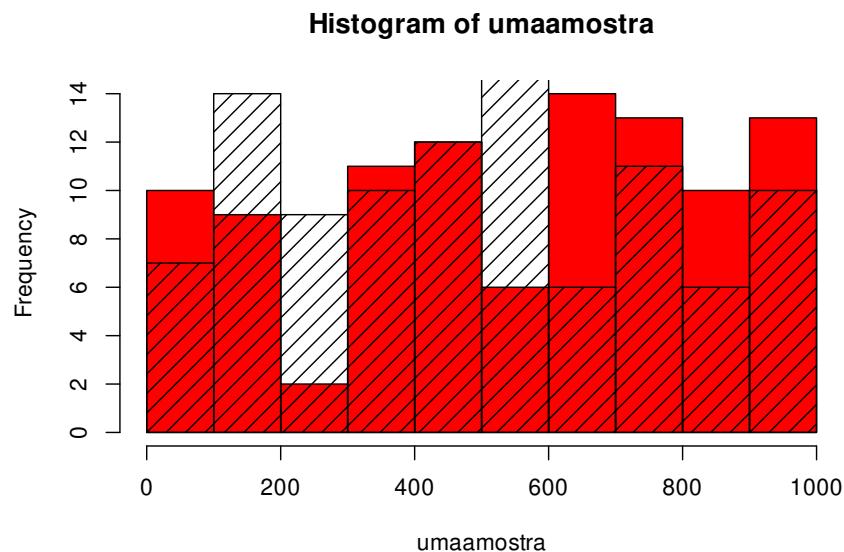
```
# agora amostra 100 valores desse vetor que tem 9991 valores  
umaamostra <- sample(umvetor, size = 100, replace = T)  
# entao isso deve ser verdadeiro  
length(umaamostra) == 100
```

```
## [1] TRUE
```

```
# primeiros 10 valores amostrados  
head(umaamostra, 10)
```

```
## [1] 144.8 886.8 993.6 200.0 746.7 904.8 688.3 848.2 97.6 732.1
```

```
# histograma da amostra  
hist(umaamostra, col = "red")  
# note que a distribuição é uniforme, qualquer valor tem a mesma chance de ser amostrado  
# faco outra vez e adiciono  
outraamostra <- sample(umvetor, size = 100, replace = T)  
# adiciono ao histograma anterior  
hist(outraamostra, density = 10, add = T, col = "black")
```



```
# cada vez é diferente a sequencia de valores amostrados
```

9.2 Distribuições aleatórias

O R disponibiliza um conjunto um conjunto de funções para gerar densidade, distribuição cumulativa, quantis e geração aleatória de variáveis com diferentes distribuições de probabilidade (veja a seção [Para saber mais](#) para mais informações).

Aqui vamos conhecer as funções que geram variáveis aleatórias para duas distribuições probabilísticas:

- Distribuição normal¹;
- Distribuição log-normal².

¹https://en.wikipedia.org/wiki/Normal_distribution

²https://en.wikipedia.org/wiki/Log-normal_distribution

```
# veja no R o help para as funções para as diferentes distribuições disponíveis no R.  
?Distributions
```

9.2.1 Distribuição Normal

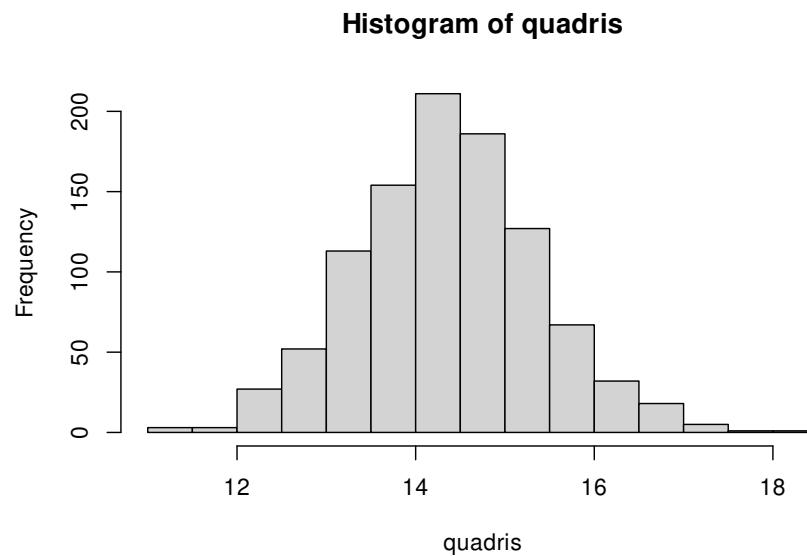
Segundo as companhias aéreas dos EUA, a largura dos quadris dos homens norte-americanos segue uma curva normal, com média de 14.4 polegadas, e desvio-padrão de 1 polegada.

9.2.1.1 rnorm()

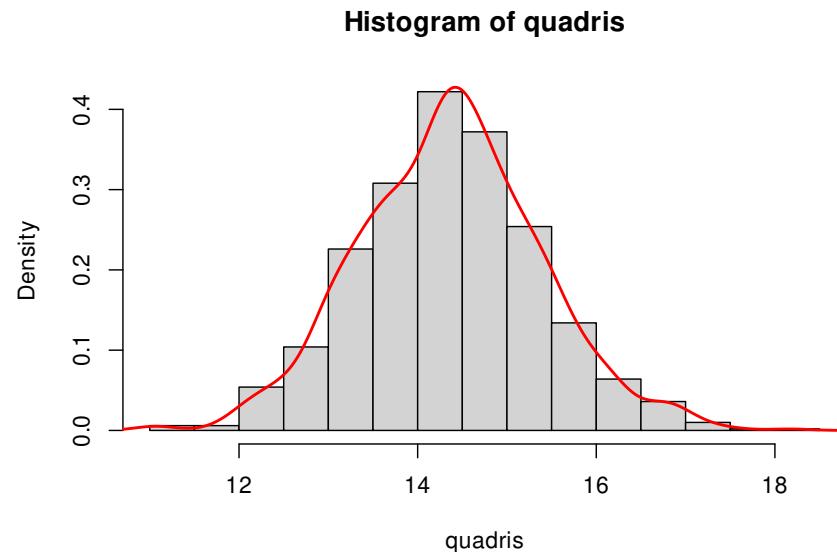
Esta função permite gerar essa distribuição de valores aleatórios que seguem uma distribuição normal se soubermos a média e o desvio padrão.

```
?rnorm # veja o help da função
```

```
amedia <- 14.4  
osd <- 1  
# vamos gerar 1000 valores aleatórios com essa distribuição  
quadris <- rnorm(n = 1000, mean = amedia, sd = osd)  
# visualizando essa distribuição de valores  
hist(quadris)
```



```
# ou visualizando a densidade probabilística dessa distribuição
hist(quadris, probability = T)
# a linha da densidade
lines(stats::density(quadris), col = "red", lwd = 2)
```



9.2.1.2 `pnorm()`

Esta função permite responder à pergunta:



Qual percentual de pessoas não cabe em um assento de 15 polegadas?

```
# veja o help dessa função
?pnorm
```

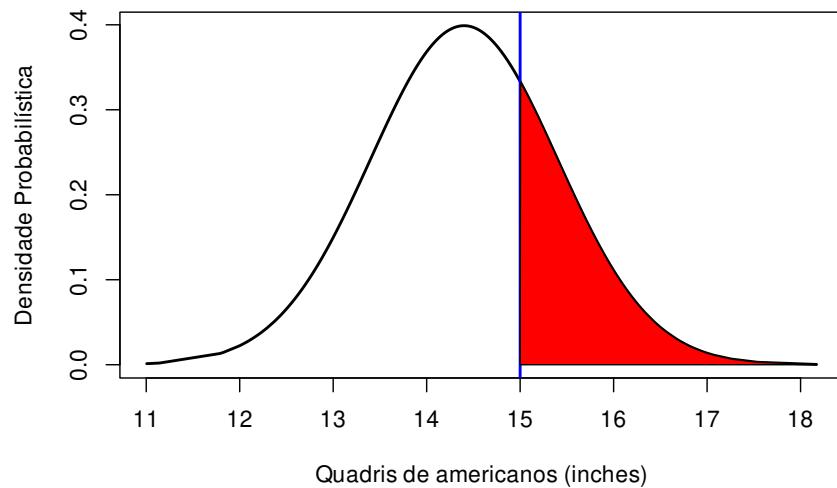
```
# qual o percentil/quantil da distribuição que é maior que 15 polegadas
# quem não cabe no assento?
qMaior15in <- pnorm(q = 15, mean = 14.4, sd = 1, lower.tail = F)
# note o argumento lower.tail=F (você quer apenas o percentil que está acima desse valor)
# ca. de 27%
qMaior15in
```

```
## [1] 0.2742531
```

```
print(paste(round(qMaior15in * 100, 0), "% dos americanos não cabem em um assento de 15 polegadas"))
```

```
## [1] "27% dos americanos não cabem em um assento de 15 polegadas"
```

```
# vamos visualizar isso
x <- sort(quadris) # ordenamos os quadris
y <- dnorm(x, mean = amedia, sd = 1) # calculamos a densidade probabilística desses valores
plot(x, y, type = "l", lwd = 2, col = "black", ylab = "Densidade Probabilística", xlab = "Quadris")
# agora fazemos o mesmo para os quadris que satisfazem a condição
abline(v = 15, lwd = 2, col = "blue") # o quadril da pergunta
xx <- sort(quadris[quadris >= 15])
yy <- dnorm(xx, mean = amedia, sd = 1)
# adicionamos isso na forma de um polígono
# coordenadas X do polígono (note que adicionei valores para poder fechar o polígono)
xxp <- c(15, xx, max(xx), 15)
# e adiciono também no eixo y
ypy <- c(0, yy, 0, 0)
# desenho o polígono correspondente
polygon(xxp, yyp, col = "red")
```



```
## a area desse polígono representa esse quantil
# (i.e. a porcentagem da população que tem quadril >= 15)
```

9.2.1.3 qnorm()

Esta função permite responder à pergunta:

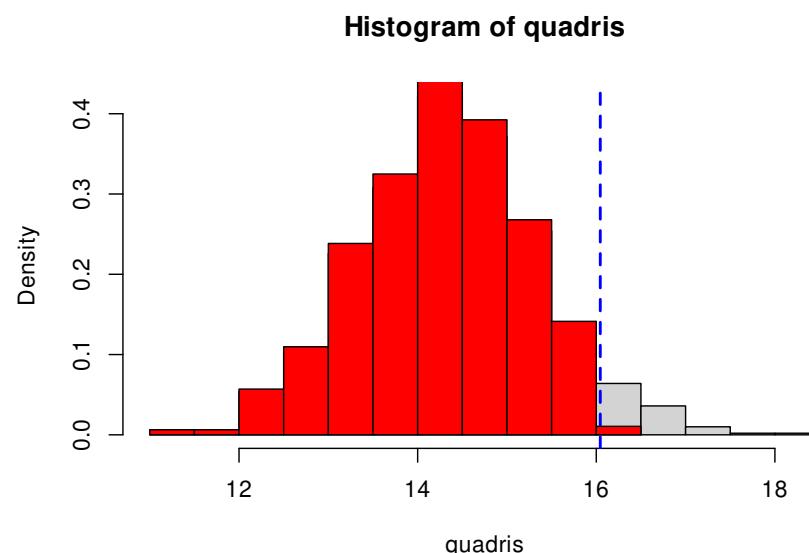


Qual largura de assento garante que 95% da população caberá?

```
# agora fazemos o inverso da função pnorm, ao invés de informar o quantil,
# informamos o percentil
qnorm(p = 0.95, mean = 14.4, sd = 1)
```

```
## [1] 16.04485
```

```
# este é o tamanho do assento que inclui 95% dos valores
# numa distribuição normal com essa média e esse desvio padrão
tamanho95 <- qnorm(p = 0.95, mean = 14.4, sd = 1)
# graficando isso?
hist(quadrис, prob = T)
# limite de tamanho
abline(v = tamanho95, col = "blue", lwd = 2, lty = "dashed")
# plota valores que estão dentro dessa distribuição
hist(quadrис[quadrис < tamanho95], prob = T, add = T, col = "red")
```



9.2.1.4 dnorm()

A função `dnorm()` gera a densidade probabilística para um conjunto de valores, tendo em vista uma distribuição normal de mesma média e mesmo desvio padrão. Podemos responder à pergunta:



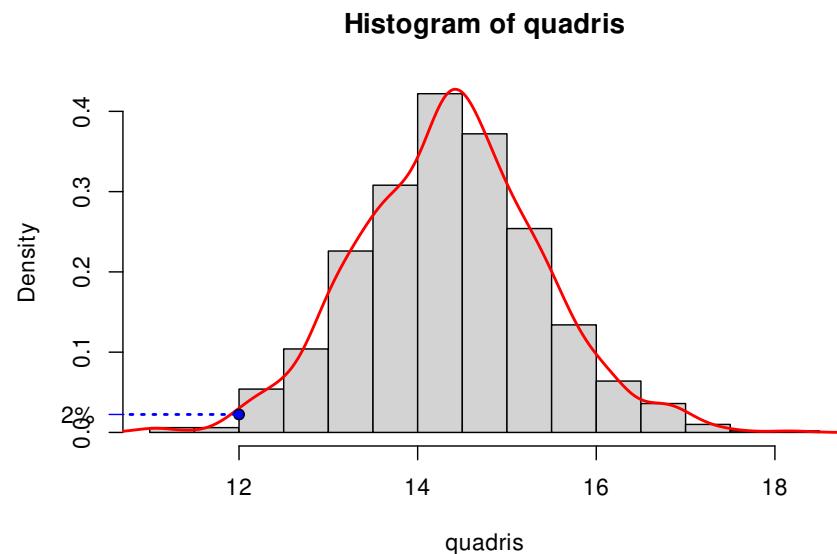
Qual a densidade probabilística para um valor de quadril de 12

polegadas? Ou seja, qual a probabilidade de amostrar na população uma pessoa com quadril de 12 polegadas?

```
dq12 <- dnorm(x = 12, mean = 14.4, sd = 1)  
dq12
```

```
## [1] 0.02239453
```

```
# mostra na figura  
# plota a densidade probabilística  
hist(quadris, prob = T)  
# coloca a linha  
lines(stats::density(quadris), col = "red", lwd = 2)  
# adiciona um ponto que mostra a probabilidade do valor  
points(x = 12, y = dq12, pch = 21, bg = "blue")  
segments(0, dq12, 12, dq12, lty = "dotted", col = "blue", lwd = 2)  
axis(side = 2, at = dq12, labels = paste(round(dq12 * 100, 0), "%", sep = ""), line = NULL, la
```



9.3 Para saber mais:

- (I) Probability distributions³.

9.4 Exercícios

- Resolva o exercício 103.7 Cara ou coroa⁴.
- Resolva o exercício 103.9 Lembrando matrizes e listas⁵.

³<http://www.r-tutor.com/elementary-statistics/probability-distributions>

⁴<http://notar.ib.usp.br/exercicio/34>

⁵<http://notar.ib.usp.br/exercicio/37>



10

Extraindo dados de colunas descritivas

Na compilação de dados de espécimes botânicos, informações sobre a planta e o habitat estão em geral anotadas na forma de textos (notas), num formato pouco útil para entender os atributos dos organismos. Os scripts aqui relacionados mostram exemplos de como extrair informações dessas colunas descritivas, categorizando informações de **habito**, **fertilidade**, **textura do solo**, **habitat**, **dap**, **altura**.

10.1 Dados categóricos

A lógica básica é simples. Para cada tipo de informação (variável), buscaremos por um conjunto de palavras e suas variantes, e substituirmos todas elas por uma única palavra, criando estados de variação (**categorias**). É importante pensar na ordem em que a busca é feita, pois em algumas notas descritivas duas palavras podem ser encontradas e você quer apenas uma delas. Por exemplo, na busca por hábito você deveria buscar a palavra *epífita* antes da palavra *árvore*, porque se a coleta for de uma planta *epífita*, a palavra *árvore* pode aparecer como hospedeiro. Não precisa se preocupar com acentos nem se a palavra está em caixa alta ou baixa. Criaremos uma função para ignorar isso nas comparações.

10.1.1 Passo 01 - Lista de referência

Você cria uma lista de referencia (veja a seção 10.2 para exemplos de listas de referência) para cada variável que quer buscar, composta de vetores com palavras ou parte de palavras a serem buscadas.

Cada vetor tem um nome, que é a palavra que define a categoria (estados de variação da variável). A construção dessa lista deve ser baseada nas palavras que existem nos seus dados. As listas de referência abaixo contêm alguns conjuntos de palavras que você pode ajustar para seu uso. Vamos tomar como exemplo a variável hábito. Teremos duas categorias que desejamos separar: arvoreta e árvore. Vamos primeiro criar uma lista de referência usando a construção `obj[[nome.da.categoria]] = c(palavras.a.buscar)`.

```
referencias <- list()
referencias[["arvoreta"]] <- c("arvoreta", "treelet", "arvore pequena", "arbolito", "small tree")
referencias[["árvore"]] <- c("arvore", "tree", "rvore", "arvo", "arbol", "avore")
referencias

## $arvoreta
## [1] "arvoreta"      "treelet"       "arvore pequena" "arbolito"
## [5] "small tree"
##
## $árvore
## [1] "arvore" "tree"   "rvore"   "arvo"   "arbol"  "avore"
```

Vejam que em nossa lista de referência `referencias` incluímos primeiramente arvoreta e depois árvore, porque **árvore** faz parte da palavra **arvoreta**, e nós desejamos diferenciar as duas categorias. Note que erros de grafia e palavras em diferentes idiomas podem ser inseridos nesses vetores como palavras a serem buscadas.

10.1.2 Passo 02 - Funções que fazem a busca

Primeiro criaremos uma função `removeacentos()`, que pega um texto ou vetor de textos e remove acentos. Isso é útil porque é frequente a mesma palavra aparecer com e sem acentos, por isso é melhor ignorar acentos nas comparações.

```
removeacentos <- function(x) {  
  # remove acentos  
  xx <- iconv(x, to = "ASCII//TRANSLIT")  
  xx <- gsub("[~|^~|\"|'|`]", "", xx)  
  return(xx)  
}
```

E uma segunda função `pegavalor()` que usa a lista de referência para extrair os dados de um texto. Nesta função, o argumento `x` é um vetor de classe `character` de comprimento igual a 1 contendo o texto original a ser explorado. O segundo argumento da função, `referencias`, é uma lista de comprimento maior ou igual a 1 referente as categorias a serem buscadas no elemento `x`:

```
pegavalor <- function(x, referencias) {  
  acategoria <- NA # objeto onde armazena a categoria caso encontre em x  
  x <- x[!is.na(x)] # elimina elementos NA em x  
  x <- removeacentos(x)  
  if (length(x) == 1 & is.character(x)[1]) { # se x ainda for um texto  
    for (t in 1:length(referencias)) {  
      # se ainda não encontrou procura em outra referencia  
      if (is.na(acategoria)) {  
        words <- referencias[[t]] # pega as palavras a serem buscadas  
        words <- words[!is.na(words)] # limpa caso haja NAs  
        hab <- names(referencias)[t] # pega a categoria correspondente  
        if (!is.null(hab) & !is.na(hab) & length(words) >= 1) {  
          words <- removeacentos(words)  
          found <- 0  
          # para cada palavra chave busca em x e se encontrar anota  
          for (g in 1:length(words)) {  
            gp <- grep(words[g], x, ignore.case = T)  
            if (length(gp) > 0) {  
              found <- found + 1  
            }  
          }  
        }  
      }  
    }  
  }  
}
```

```

# se encontrou, atribui
if (found > 0) {
  acategoria <- hab
}
}
}
}
}
# retorna o resultado
return(acategoria)
}

```

10.1.3 Passo 03 - Usando a função

Vamos utilizar dados de exemplo para testar a função. Baixe o arquivo presente neste endereço (https://github.com/LABOTAM/IntroR/blob/main/dados/pegados_dados_exemplo.csv). Ele apresenta uma coluna com notas de exsicatas para ilustrar nosso exemplo.

```

# voce já deve ter os seguinte objetos: pegavvalor(), removeacentos(), referencias
# le o arquivo de exemplo
dados <- read.table(file = "pegados_dados_exemplo.csv", sep = "\t", as.is = TRUE, na.strings =
txt.org <- dados$NOTAS_ORIGINAL # coluna com dados de notas (cada linha é um registro)
head(txt.org)

## [1] "; HABITAT: Floresta Ombrófila Densa Submontana.; DESC: Árvore de 12m; frutos imaturos verdes
## [2] "; Hábito: Árvore; Cor da flor: ; Cor do fruto: verdes árvore de 8 m de altura, frutos verdes
## [3] "; Hábito: Árvore; Cor da flor: brancas (botoes brancos); Cor do fruto: árvore de 10 m de altura
## [4] "?muena?, 15m, fls. Yellow"
## [5] "[BARCODE = *MIRR 00705*]Tall terra firme forest with trees to 35m; Transecto de tatu, c. 4km
## [6] "10m x 15cm DAP. Frutos imaturos verdes. 10m x 15cm dap. frutos imaturos verdes. ; de árvore

```

```
habito <- sapply(txt.org, pegavvalor, referencias = referencias)
table(habito) # o que estiver NA significa que ele não encontrou as palavras em referencias.
```

árvore	arvoreta
542	113

```
# adiciona com uma nova coluna em dados
dados$HABITO <- habito
```

10.2 Listas de referência

As listas de referência abaixo podem ser úteis e você pode acrescentar termos de busca a cada categoria. A ordem das categorias na lista deve ser pensada de forma que minimize erros. Por exemplo, é importante primeiro encontrar `liana herbacea` e apenas depois definir `liana lenhosa` porque muitas vezes aparece apenas a palavra 'liana' que se refere a plantas lenhosas quando não diferenciada.

O formato básico é:

```
lista[["categoria a definir"]] = c(vetor com as palavras a serem
encontradas que indiquem a categoria)
```

10.2.1 Hábito da planta

```

habito.refs <- list()
habito.refs[["liana herbácea"]] <- c("erva trepadeira", "trepadeira", "liana herbácea", "liana")
habito.refs[["liana lenhosa"]] <- c("cipó", "liana", "liana sublenhosa", "liana lenhosa")
habito.refs[["hemiepífita"]] <- c("hemiepifita", "hemi-epifita", "hemiepiphyte", "hemi-epiphyte")
habito.refs[["epífita"]] <- c("epífita", "epiphyte")
habito.refs[["arvoreta"]] <- c("arvoreta", "treelet", "Trellet", "arvore pequena", "arbólito");
habito.refs[["árvore"]] <- c("m de altura", "DAP:", "arvore", "tree", "rvore", "arvo", "arbólito")
habito.refs[["arbusto"]] <- c("arbust", "shrub", "scrub", "subarbust", "arbust", "Shruv")
habito.refs[["hemiparasita"]] <- c("hemiparasit", "hemi-parasit")
habito.refs[["saprófita"]] <- c("saprofit", "saprophy")
habito.refs[["parasita"]] <- c("parasit")
habito.refs[["erva aquática"]] <- c("erva aquática", "erva aquatica", "Macrófita aquática", "macrofita")
habito.refs[["erva"]] <- c("erva", "herbace", "terrestre", "herbacia", "herb")

```

10.2.2 Estado de fertilidade da amostra

```

fertilidade.refs <- list()
fertilidade.refs[["flores"]] <- c(" flor", " petal", " flôr", " pétala", " estigma", " sépal", "estigmas", "sépalos")
fertilidade.refs[["botões"]] <- c(" botão", " botões", " botaõ", " botaõ", " bud")
fertilidade.refs[["frutos"]] <- c(" fruto", " fruit", " futo", " fruto", "frutescencia", "frutescencia")
fertilidade.refs[["estéril"]] <- c("Estéril", "Sterile")

```

10.2.3 Classes de hábitat

```

habitat.refs <- list()
habitat.refs[["Floresta ciliar"]] <- c("Riverbank", "beria de rio", "beira de rio", "berra de rio")
habitat.refs[["Floresta de igapó"]] <- c("igapo", "margin of black water igarapé")
habitat.refs[["Floresta Inundável"]] <- c("varzea", "flooded forest", "floresta indundavel", "floresta inundável")
habitat.refs[["Campinarana"]] <- c("campinarana", "varillal", "chamisal", "wallaba", "curuni", "curuní")
habitat.refs[["Campina"]] <- c("campina", "campinarana gramíneo lenhosa", "savannis areno")

```

```
habitat.refs[["Aquático"]] <- c("aquatic")
habitat.refs[["Vegetação secundária"]] <- c("Mata secundária", "Secondary forest", "capoeira",
habitat.refs[["Floresta de Baixio"]] <- c("Nomflooded moist forest", "baixio", "area encharcada")
habitat.refs[["Floresta de Vertente"]] <- c("vertente", "encosta", "slope", "Solo areno-argiloso")
habitat.refs[["Floresta de Platô"]] <- c("plato", "plateau")
habitat.refs[["Floresta de Terra Firme"]] <- c("Solo argiloso. Dossel", "Solo argiloso, dossel")
habitat.refs[["Cerradão"]] <- c("cerradao")
habitat.refs[["Cerrado"]] <- c("cerrado", "savanna", "savana")
habitat.refs[["Campo rupestre"]] <- c("Campo rupestre")
```

10.2.4 Textura do solo

```
solo.refs <- list()
solo.refs[["pedregoso"]] <- c("pedregos", "pedral", "rock")
solo.refs[["areno-argiloso"]] <- c("areno-argilos", "areno argilos", "arenoso-argiloso", "arenoso")
solo.refs[["argiloso"]] <- c("argilos", "agilos", "clay", "clayish", "loam", "arcill")
solo.refs[["arenoso"]] <- c("arenoso", "aenoso", "areia branca", "sand", "arrenoso")
solo.refs[["siltoso"]] <- c("silte", "silt")
```

10.2.5 Exsudato

```
exsudato.refs <- list()
exsudato.refs[["látex"]] <- c("latex", "látex")
exsudato.refs[["resina"]] <- c("resina")
exsudato.refs[["seiva"]] <- c("seiva", "sap")
exsudato.refs[["exsudato"]] <- c("exsudato", "exsudação", "exsudate")
```

10.3 Extraindo altura e dap

Funções para extrair altura e diâmetro à altura do peito (DAP) de colunas descritivas.

10.3.1 Altura

```
pegaaltura <- function(x) {  
  x <- gsub("\\+/-", "", x)  
  x <- gsub(" ", " ", x)  
  x <- gsub(" ", " ", x)  
  # print(x)  
  pt1 <- "[+]?[0-9]*[.,.]?[0-9] m x"  
  pt2 <- "[+]?[0-9]*[.,.]?[0-9]m x"  
  pt3 <- "[+]?[0-9]*[.,.]?[0-9] m\\. x"  
  pt4 <- "[+]?[0-9]*[.,.]?[0-9] m de altura"  
  pt5 <- "[+]?[0-9]*[.,.]?[0-9] m\\. de altura"  
  pt6 <- "[+]?[0-9]*[.,.]?[0-9]m d ealtura"  
  pt7 <- "[+]?[0-9]*[.,.]?[0-9]m de alto"  
  pt8 <- "[+]?[0-9]*[.,.]?[0-9] m de alto"  
  pt9 <- "[+]?[0-9]*[.,.]?[0-9]m alto"  
  pt10 <- "[+]?[0-9]*[.,.]?[0-9]m de altura"  
  pt11 <- "[+]?[0-9]*[.,.]?[0-9] m tall"  
  pt12 <- "[+]?[0-9]*[.,.]?[0-9]m tall"  
  pt13 <- "[+]?[0-9]*[.,.]?[0-9] m\\. tall"  
  pt14 <- "[+]?[0-9]*[.,.]?[0-9] m\\. Tall"  
  pt15 <- "[+]?[0-9]*[.,.]?[0-9]m\\. tall"  
  pt16 <- "Tree, [+]?[0-9]*[.,.]?[0-9]m"  
  pt17 <- "Tree, [+]?[0-9]*[.,.]?[0-9] m"  
  pt18 <- "Tree [+]?[0-9]*[.,.]?[0-9] m"  
  pt19 <- "Tree [+]?[0-9]*[.,.]?[0-9]m"  
  pt20 <- "to [+]?[0-9]*v?[0-9] m"  
  pt21 <- "Tree [0-9]*-[0-9]*m"
```

```
pt22 <- "de [-+]?[0-9]*[,.]?[0-9] m"
pt23 <- "de [-+]?[0-9]*v?[0-9]m"
pt24 <- "de [0-9]*-[0-9]* m"
pt25 <- "Tree [0-9]*-[0-9]* m"
pt26 <- "de [0-9]*-[0-9]*m"
pt27 <- "alt. [-+]?[0-9]*[,.]?[0-9] m"
pt28 <- "[+-]?[0-9]*[,.]?[0-9] feet high"
pt29 <- "[+-]?[0-9]*[,.]?[0-9] ft. high"

pt30 <- "rvore [-+]?[0-9]*[,.]?[0-9]m"
pt31 <- "rvore [-+]?[0-9]*[,.]?[0-9] m"
pt32 <- "rvore, [-+]?[0-9]*[,.]?[0-9] m"
pt33 <- "rvore, [-+]?[0-9]*[,.]?[0-9]m"

pt34 <- "rbusto [-+]?[0-9]*[,.]?[0-9]m"
pt35 <- "rbusto [-+]?[0-9]*[,.]?[0-9] m"
pt36 <- "rbusto, [-+]?[0-9]*[,.]?[0-9] m"
pt37 <- "rbusto, [-+]?[0-9]*[,.].?[0-9]m"
pt38 <- "rvore +/- [-+]?[0-9]*[,.].?[0-9]m"
pt39 <- "rvore ca. [-+]?[0-9]*[,.].?[0-9]m"
pt40 <- "de [-+]?[0-9]*[,.].?[0-9] de altura"
pt41 <- "altura [-+]?[0-9]*[,.].?[0-9]m"
pt42 <- "altura [-+]?[0-9]*[,.].?[0-9] m"
pt43 <- "of [-+]?[0-9]*[,.].?[0-9]m"
pt44 <- "to [-+]?[0-9]*[,.].?[0-9]m"
pt45 <- "Treelet [-+]?[0-9]*[,.].?[0-9]m"
pt46 <- "rvore [-+]?[0-9]*[,.].?[0-9] m"
pt47 <- "rvore, [-+]?[0-9]*[,.].?[0-9] m"
pt48 <- "rvoreta, [-+]?[0-9]*[,.].?[0-9] m"
pt49 <- "Fuste= [-+]?[0-9]*[,.].?[0-9] m"
pt50 <- "Fuste= [-+]?[0-9]*[,.].?[0-9]m"
pt51 <- "com [-+]?[0-9]*[,.].?[0-9] m. alt."
pt52 <- "Height: [-+]?[0-9]*[,.].?[0-9] m"
pt53 <- "Arbol [-+]?[0-9]*[,.].?[0-9]m"
pt54 <- "Treelet, [-+]?[0-9]*[,.].?[0-9]m"
pt55 <- "altura = [-+]?[0-9]*[,.].?[0-9]m"
```

```
pt56 <- "Fuste = [-+]?[0-9]*[,.].?[0-9]m"
pt57 <- "Fuste = [-+]?[0-9]*[,.].?[0-9] m"

altura <- NA
for (p in 1:57) {
  pt <- get(paste("pt", p, sep = ""))
  gp <- grep(pt, x, ignore.case = F)
  if (length(gp) > 0 & is.na(altura)) {
    # print(p)
    rmm <- strsplit(x, pt)[[1]]
    rmm <- rmm[rmm != "" & rmm != "." & !is.na(rmm)]
    xx <- x
    if (length(rmm) > 0) {
      for (r in length(rmm):1) {
        xx <- gsub(rmm[r], "", xx, fixed = T, useBytes = T)
      }
    }
    xx <- trimws(gsub("[A-Z]|\\(|\\)|:|=|", "", xx, ignore.case = T), which = "both")
    xx <- gsub(",.", ".", xx)
    tt <- grep("-", xx)
    if (length(tt) > 0) {
      xxx <- strsplit(xx, "-")[[1]]
      xxx <- xxx[xxx != ""]
      xxx <- gsub("\\.", "", xxx)
      xx <- mean(as.numeric(trimws(xxx, which = "both")), na.rm = T)
    } else {
      xx <- strsplit(xx, " ")[[1]]
      xx <- trimws(xx, which = "both")
      xx <- xx[xx != "."]
      xx <- xx[1]
    }
    xx <- as.numeric(xx)
    if (!is.na(xx) && xx > 0) {
      altura <- xx
    } else {
      altura <- NA
    }
  }
}
```

```
        }
    }
}
return(altura)
}
```

10.3.2 DAP

```
pegadap <- function(x) {
  x <- gsub("\\" + "/-", "", x)
  x <- gsub(" ", " ", x)
  x <- gsub(" ", " ", x)
  # print(x)
  pt1 <- "x [-+]?[0-9]*[.,.]?[0-9] cm de circ"
  pt2 <- "x [-+]?[0-9]*[.,.]?[0-9] cm de di"
  pt3 <- "x [-+]?[0-9]*[.,.]?[0-9]cm de di"
  pt4 <- "x [-+]?[0-9]*[.,.]?[0-9]cm di"
  pt5 <- "m x [-+]?[0-9]*[.,.]?[0-9] cm DAP"
  pt6 <- "m x [-+]?[0-9]*[.,.]?[0-9] cm"
  pt7 <- "de [-+]?[0-9]*[.,.]?[0-9] m de DAP"
  pt8 <- "[+]?[0-9]*[.,.]?[0-9] cm de DAP"
  pt9 <- "[+]?[0-9]*[.,.]?[0-9] cm (DAP)"
  pt10 <- "[+]?[0-9]*[.,.]?[0-9] cm dbh"
  pt11 <- "DAP [-+]?[0-9]*[.,.]?[0-9] cm"
  pt12 <- "[+]?[0-9]*[.,.]?[0-9] cm D.A.P."
  pt13 <- "D.A.P. = [-+]?[0-9]*[.,.]?[0-9] cm"
  pt14 <- "[+]?[0-9]*[.,.]?[0-9]cm de di"
  pt15 <- "[+]?[0-9]*[.,.]?[0-9]cm dap"
  pt16 <- "[+]?[0-9]*[.,.]?[0-9]cm. dia"
  pt17 <- "dbh. [-+]?[0-9]*[.,.]?[0-9]cm"
  pt18 <- "[+]?[0-9]*[.,.]?[0-9]cm. in dia"
  pt19 <- "[+]?[0-9]*[.,.]?[0-9] cm de di"
  pt20 <- "[+]?[0-9]*[.,.]?[0-9] cm (DAP)"
```

```
pt21 <- "DBH [-+]?[0-9]*[.,]?[0-9] cm"

# x="Árvore de 13m de altura x 11cm de diâmetro do fuste."
altura <- NA
for (p in 1:21) {
  pt <- get(paste("pt", p, sep = ""))
  gp <- grep(pt, x, ignore.case = F)
  if (length(gp) == 1 & is.na(altura)) {
    # print(p)
    rmm <- strsplit(x, pt)[[1]]
    rmm <- rmm[rmm != "" & rmm != "." & !is.na(rmm)]
    xx <- x
    if (length(rmm) == 1) {
      for (r in length(rmm):1) {
        xx <- gsub(rmm[r], "", xx, fixed = T, useBytes = T)
      }
    } else {
      if (length(rmm) == 2) {
        n1 <- nchar(rmm[1])
        n2 <- nchar(rmm[2])
        n0 <- nchar(x)
        ns <- n1 + 1
        nt <- n0 - n2
        xx <- substr(x, ns, nt)
      } else {
        if (length(rmm) > 2) {
          xx <- NA
        }
      }
    }
    if (!is.na(xx)) {
      xx <- trimws(gsub("[A-Z]|\\"|\\(|\\)|:|=|\"", "", xx, ignore.case = T), which = "both")
      xx <- gsub(",.", ".", xx)
      tt <- grep("-", xx)
      if (length(tt) > 0) {
        xxx <- strsplit(xx, "-")[[1]]
      }
    }
  }
}
```

```
xxx <- xxx[xxx != ""]
xxx <- gsub("\ \.", "", xxx)
xx <- mean(as.numeric(trimws(xxx, which = "both")), na.rm = T)
} else {
  xx <- strsplit(xx, " ")[[1]]
  xx <- trimws(xx, which = "both")
  xx <- xx[xx != "." & xx != "" & xx != "..."]
  xx <- xx[1]
  if (substr(xx, nchar(xx), nchar(xx)) == ".") {
    xx <- substr(xx, 1, nchar(xx) - 1)
  }
}
xx <- as.numeric(xx)
if (xx > 0) {
  altura <- xx
}
}
return(altura)
}
```

10.4 Usando essas funções

Vamos utilizar o conjunto `dados` carregado acima para utilizar as funções que criamos anteriormente:

```
txt.org <- as.vector(dados$NOTAS) # sua coluna com dados descritivos
```

10.4.1 Obtendo valores de altura

```
# aplica a funcao  
alts <- sapply(txt.org, pegaaltura)
```

```
## Warning in FUN(X[[i]], ...): NAs introduced by coercion  
  
## Warning in mean(as.numeric(trimws(xxx, which = "both")), na.rm = T): NAs  
## introduced by coercion  
  
## Warning in mean(as.numeric(trimws(xxx, which = "both")), na.rm = T): NAs  
## introduced by coercion  
  
## Warning in FUN(X[[i]], ...): NAs introduced by coercion  
  
## Warning in FUN(X[[i]], ...): NAs introduced by coercion  
  
## Warning in mean(as.numeric(trimws(xxx, which = "both")), na.rm = T): NAs  
## introduced by coercion  
  
## Warning in mean(as.numeric(trimws(xxx, which = "both")), na.rm = T): NAs  
## introduced by coercion
```

```
head(alts)
```

```
## ; HABITAT: Floresta Ombrófila Densa Submontana.; DESC: Árvore  
## ; Hábito: Árvore; Cor da folha: Verde; Altura: 10-15m  
##
```

```

## ; Hábito: Árvore; Cor da flor: brancas (botoes brancos); Cor do fruto: árvore de 10 m de altura,
##
##
##
## [BARCODE = *MIRR 00
##
##
##
## names(alts) <- NULL
# quais valores viraram NA (ou seja, não encontrou um valor de altura)
txt.org[is.na(alts)]

## [1] "?muena?, 15m, fls. Yellow"
## [2] "[BARCODE = *MIRR 00705*]Tall terra firme forest with trees to 35m; Transecto de tatu, c. 4
## [3] "1st specimen: Tree, flowers cream. 2nd specimen: Shrub 20 ft. tall; flowers cream-
colored."
## [4] "3m tree. Flowers tan color"
## [5] "6 in. diam, 30 ft high. Phenology of specimen: Fruit."
## [6] "7.7 cm DAP"
## [7] "8 m tree; flowers yellowish."
## [8] "Altura 13m, PAP 63cm, caule interno branco, externo róseo."
## [9] "Ambiente: Floresta perenifolia. Habito: Árvore menor que 5m"
## [10] "Ambiente: Mata Atlântica. Habito: Árvore, 7-8m."
## [11] "Ambiente: Mata Estacional Semidecidual com Mata ciliar. Habito: Árvore, ca 14m. floresta
## [12] "Ambiente: Mata higrofila, úmida. Habito: Árvore maior que 5m e menor 20m"
## [13] "Ambiente: Mata junto a plantaçao de coco. Habito: Arvoreta, 3m. arvoreta. 3. metros. flor
argiloso ; ha"
## [14] "Ambiente: Mata Ombrófila Densa Montana. Habito: Arvoreta, ca. 4m."
## [15] "Ambiente: Mata. Borda com canavial. Habito: Árvore, ca. 6m."
## [16] "Ambiente: Mata. Borda da mata. Habito: Árvore, 10 a 12m."
## [17] "Ambiente: Mata. Habito: Árvore, ca 15m. árvore ca. 15 m alt. folahs com indumento castanho
amareladas.; Árvore ca. 15 m alt. Folahs com indumento castanho. Flores creme-
amareladas."

```

```

## [18] "Ambiente: Mata. Habito: Árvore, ca. 9m"
## [19] "Ambiente: Mata. Habito: Árvore, DAP 37,2cm."
## [20] "Ambiente: Mata. Habito: Árvorezinha, 7m."
## [21] "Ambiente: Mata. Na borda da mata. Habito: Árvore, 5 a 6m."
## [22] "Ambiente: Mata. Na borda da mata. Habito: Árvore, c.a 9m."
## [23] "Ambiente: Mata. Na borda da mata. Habito: Árvore, ca. 4 a 5m."
## [24] "Ambiente: Mata. Na borda da mata. Habito: Árvore, ca. 6,5m."
## [25] "Ambiente: Mata. No interior da mata. Habito: Árvore, 8 a 10m."
## [26] "Arbol 7 m, DAP=8 cm, simpódico, aceites etéreos en las hojas. Frutos verde-amarillentos."
## [27] "Arbol 8 m, DAP: 11.1 cm, madera aromática, estéril."
## [28] "Arbol de aproximadamente 10 m alto. Frutos verdes en el exterior, morados internamente y ...
## [29] "Arbol, 20 metres de altura. Frutos drupas verdes, dentro de capsula negra +/- leñosa."
## [30] "Arbóreo, terrestre, fuste cilíndrico, casca áspera, altura total 1,5m, fuste 10m, cap fuste ...
## [31] "Arbóreo, terrestre, fuste cilíndrico, casca áspera, altura total 12m, fuste 6m, cap fuste ...
## [32] "Arbóreo, terrestre, fuste cilíndrico, casca áspera, altura total 13m, fuste 9m, cap fuste ...
## [33] "Arbóreo, terrestre, fuste cilíndrico, casca áspera, altura total 15m, fuste 10m, cap fuste ...
## [34] "Arbóreo, terrestre, fuste cilíndrico, casca áspera, altura total 15m, fuste 6m, cap fuste ...
## [35] "Arbóreo, terrestre, fuste cilíndrico, casca áspera, altura total 15m, fuste 8m, cap fuste ...
## [36] "Arbóreo, terrestre, fuste cilíndrico, casca áspera, altura total 16m, fuste 7m, cap fuste ...
## [37] "Arbóreo, terrestre, fuste cilíndrico, casca áspera, altura total 4m, fuste 2m, cap fuste ...
## [38] "Arbóreo. Inflorescencia patente, botoes verdes.; DESC: Arbóreo. Inflorescencia patente, ...
## [39] "arbre, 11 m, 9 cm DBH. fr. vert r points blancs, clair r lapex, plus foncé r la base, claire ...
## [40] "arbre, 8 m, 10 cm DBH; fleur trimcres, 3+3 tép. jaunes recouverts de duvet roux, 3+3+3 éta ...
## [41] "Arbustivo, fruto verde e preto; Costeiro"
## [42] "Arbusto ca. 1 m alt.; caule amarronzado; frutos verdes."
## [43] "Arbusto ca. 1,50m alt.; caule amarronzado; frutos imaturos verdes."
## [44] "Arbusto ca. 1,5m alt e 1cm de diâmetro. Caule acinzentado. Folhas coriáceas. Flores em bo ...
## [45] "Arbusto ca. 2 m alt.; inflorescencia em botoes amarronzados; frutos imaturos verdes"
## [46] "Arbusto ca. 2 m alt.; ramos marrom-escuro; folhas adaxialmente verde brilhante e abaxialmente verde claro; flores amareladas."
## [47] "Arbusto ca. 2 m. Botoes ferrugíneos.HABITAT:Mata Atlântica.;"
## [48] "Arbusto ca. 3 m alt.; flores alvacentas."
## [49] "Arbusto ca. 3,0m, frutos imaturos. Subst. arenosos.; Cerrado (Típico). coleta: xls: spsf ...
## [50] "Arbusto ca. 3.0 m. alt. Folhas verdes concolores. Flores em botoes amarelados"
## [51] "Arbusto cerca de 1,30 m."

```

```
## [52] "Arbusto ramoso; ramos acinzentados; folhas coriáceas, abaxialmente verde mais claro; fru
## [53] "Arbusto, flor branca esverdeada"
## [54] "Arbusto, folhas simples alternas. Flores pequenas, pouco vistosas."
##      [55] "Arbusto, ocasional, ca. 1,60m, caule verde-amarronzado, folha levemente papirácea, adaxialmente verde brilhante, infloresc. jovem verde-alvascenta; Restinga"
## [56] "Arbusto; flores amarelas; femea. arbusto; flores amarelas; fêmea."
## [57] "Arbusto; ramos amarronzados; folhas coriáceas, adaxialmente verde-escuro brilhante e abaxialmente verde-claro; flores amareladas."
## [58] "Área de influencia direta (canteiro)."
## [59] "Área de influencia indireta."
## [60] "Árvore"
## [61] "ÁRVORE ."
## [62] "Árvore 15m, frutos pequenos um pouco ovais, caule cor de kaki escuro, mata t.f.; DESC: Árv
## [63] "Árvore até 11m, frutos verdes."
## [64] "Árvore até 20 m alt. Flores brancas."
## [65] "Árvore c/ca. 4 m alt.; Botoes creme; Fruto com cálice persistente.HABITAT:Mata degradada"
## [66] "Árvore ca 5 m. Inflorescencia em botoes."
## [67] "Árvore ca 5m de alt, botoes florais ferruginosos, arvore ca 5m de alt, botões florais ferr
## [68] "Árvore ca. 10 m de alt., flores alvo-amareladas"
## [69] "Árvore ca. 11m, com frutos verdes."
## [70] "Árvore ca. 12 m alt.; frutos imaturos verdes."
## [71] "Árvore ca. 15 m alt., copada; folhas e lenho aromáticos; folhas coriáceas; flores amarron
## [72] "Árvore ca. 15 m alt.; flores amareladas. arvore ca. 15m alt.; flores amareladas."
## [73] "Árvore ca. 18m; folhas discolores verdes; coriáceas; frutos imaturos verdes. árvore. 8. m
## [74] "Árvore ca. 4 m. altura. Frutos imaturos verdes com receptáculo avermelhado"
## [75] "Árvore ca. 5.0 m, frutos imaturos.; Campo Cerrado. Substrato arenop
pedregoso. coleta: xls: spsfantigo; geografia: cerrado&nf;; Árvore ca. 5.0 m, frutos imaturos.; C
pedregoso."
## [76] "Árvore ca. 6 m alt.; caule marrom; folhas levemente papiráceas; frutos imaturos verdes. o
## [77] "Árvore ca. 6 m alt.; folhas brilhantes na face superior, flores alvas."
## [78] "Árvore ca. 6m, flores creme."
## [79] "Árvore ca. 7, 0 m, botoes florais esverdeadas. Subst. arenop
pedregoso.; Cerrado (Típico). coleta: xls: spsfantigo; geografia: cerrado&nf;;"
## [80] "Árvore ca. 7m, frutos pretos em forma de sino"
## [81] "Árvore ca. 8 m alt. e 52 cm CAP; tronco castanho com bastante lenticelas bem desenvolvidas"
## [82] "Árvore ca. 8 m alt.; folhas coriáceas; frutos imaturos."
```

[83] "Árvore ca. 8 m alt.; frutos em desenvolvimento. Indivíduo 75. coordenadas do material: 3° 22,7s; 49°58-24,5w, indivíduo 75 (na ident. de pedro l. r. moraes consta que pode ser glomerata)"
[84] "Árvore ca. 8 m alt.; frutos imaturos verdes. árvore. 8. metros. frutos imaturos verdes.; "
[85] "Árvore ca. 8 m, com inflorescencia de coloração amarela-esverdeada. árvore ca. 8 m, com inflorescência de coloração amarela-esverdeada. árvore ca. 8m, com inflorescencia de coloração amarela-esverdeada.; Árvore ca. 8 m, com inflorescencia de coloração amarela-esverdeada."
[86] "Árvore ca. de 5,0m de alt. folhas discolores, ramos castanhos, botoes verdes e flores crema
[87] "Árvore com ± 8 m alt. Flores branco-amareladas, com leve perfume agradável."
[88] "Árvore com 10 a 12 m de alt. Folhas rígidas coriáceas. Flores de coloração creme."
[89] "Árvore com 15m. Inflorescencia verde."
[90] "Árvore com 5 a 6 metros de altura; Ramos lenticelados, folhas coriáceas, frutos com cúpula
[91] "Árvore com 5 metros de altura; flor branca muito perfumada."
[92] "Árvore com 8 a 10 m de alt. Ramos cilíndricos tomentosos, folhas coriáceas. Flores de color
[93] "Árvore com 8,0 metros de altura; inflorescencia esverdeada."
[94] "ÁRVORE COM 8M DE ALTURA, DIÂMETRO DA COPA 6M; FANERÓFITA. FOLHA ESPATIFORME; FLOR PEQUENA
[95] "Árvore com até 14 m; ramos aromáticos; inflorescencia alvacenta."
[96] "Árvore com ca. 6,5m alt.; Inflorescencia em panículas creme tomentosas; presença de dom
[97] "Árvore com ca. 7m de altur; eixo floral verde-creme; cálice, corola e androceu creme-
amarelados. Anteras quando maduras marrons; estigma preto; flores e folhas aromáticas."
[98] "Árvore com ca. de 4,0m. Frutos esverdeados quando imaturos."
[99] "Árvore com ca. de 5 a 6 metros de altura; caule escuro; folhas coriáceas discolor; inflores
[100] "Árvore com cerca de 4 a 5 m de alt. Caule acinzentado, com muitas lenticelas. Folhas coria
[101] "Árvore com flores amarelas, botoes amarelo-
esverdeados. árvore com flores amarelas; botoes amarelo-esverdeado."
[102] "Árvore com folhas e botoes florais. árvore com folhas e botões florais.; Árvore com folhas
[103] "Árvore com frutos maduros.; DESC: Árvore com frutos maduros."
[104] "Árvore com fuste tortuoso, com aproximadamente 5m alt., por 10cm de diâmetro. Flores dim
pardo, corola amarela. Madeireira."
[105] "árvore com mais ou menos 15m de alt."
[106] "ÁRVORE DE 10 M DE ALT., E 40 CM DE DIÂM, RESQUICIO DA MATA PRIMITIVA"
[107] "Árvore de 12 - 15 m, frutos ainda jovens."
[108] "ARVORE DE 3-4 MTR., FL. ALVA."
[109] "Árvore de 31,0 cm de circunferencia. árvore de 31,0 cm de circunferência.; Árvore de 31,0

```
## [110] "Árvore de 37,0 cm de circunferencia. árvore de 37,0 cm de circunferência.; Árvore de 37,0
## [111] "ÁRVORE DE 8M DE ALTURA."
## [112] "Árvore de apróx. 8 m de alt. Copa plana."
## [113] "Árvore de aproximadamente 6 metros de altura. Copa ampla. ; DE"
## [114] "Árvore de aproximadamente 6,5m folhas simples, subcrassas. Exudado incolor pegajoso e cheiro amargoso. habitat:terrestre no interior da mata. exudado incolor pegajoso e cheiro amargoso. parco 6,5 m. folhas simples, subcrassas. material estéril.; floresta estacional (mata d; ha"
## [115] "árvore de até 5m de alt., flor alva"
## [116] "Árvore de até 60 m. fl. alvacenta."
## [117] "ÁRVORE DE CA. 8 M DE ALT., FOLHA SIMPLES, INTEIRA, SOLTANDO-SE COM FACILIDADE"
## [118] "Árvore de frequencia relativa baixa com flores alvas e frutos ausentes."
## [119] "Árvore de grande porte, flores alvas esbranquiçadas, sem perfume, pequenas."
## [120] "Árvore de grande porte, flores pequenas e amarelo com pouco perfume. árvore de grande porte"
## [121] "Árvore de grande porte, sem látex ou resina. Flor alva ou marrom, sem perfume. árvore de grande porte"
## [122] "Árvore de grande porte. Flores amarelas pouco perfumadas, pequenas. árvore de grande porte"
## [123] "Árvore de grande porte. Flores mais ou menos roxas."
## [124] "Árvore de grande porte. Flores pequenas de cor alvas esverdeada."
## [125] "Árvore de mais ou menos 8 metros de altura."
## [126] "Árvore de pequeno porte. Inflorescencia terminal com pequenas flores esverdeadas.; Solo"
## [127] "Árvore de porte regular."
## [128] "Árvore em moita."
## [129] "Árvore flor amarela"
## [130] "Árvore frondosa, isolada no pasto. Flores creme."
## [131] "Árvore grande . Ref no O Genero Ocotea Aulb, no Nordeste do Brasil, Lauraceas). de Ida de"
## [132] "Árvore mais ou menos 10m alt. Flores amarelas pouco perfumadas."
## [133] "Árvore mais ou menos 3m de 5cm de diâmetro na base. Flores amareladas. Os ramos tem sulcos"
## [134] "Árvore media, fl. esbranquiçada."
## [135] "Árvore mediana."
## [136] "Árvore Ns 126.; Floresta de terra-firme. ; HA"
## [137] "ÁRVORE PEQ. FL ALVACENTA"
## [138] "Árvore pequena, fl verde - pardacenta."
## [139] "Árvore pequena. (Arb.) Flôres amarelo pardo."
## [140] "Árvore pequena. Flor amarelo pardo. árvore pequena. flor amarelo-pardo."
```

```

## [141] "Árvore regular; flores creme; folhas com cheiro de canela sassafraz. Na mata."
## [142] "àrvore rrvore;àrvore"
## [143] "Árvore tombada, folhas lisas, frutos verdes com cúpula marrom levemente avermelhada"
## [144] "Árvore, 7-10m, Comum na encosta e níveis superiores. Inflorescencia abundante, terminal, amareladas."
## [145] "Árvore, abundante, flor branca árvore, flor branca, abundante."
## [146] "Árvore, construçao e carpintaria."
## [147] "Árvore, construçao e carpintaria. folha larga.;Árvore, construçao e carpintaria"
## [148] "Árvore, flor esbranquiçada."
## [149] "Árvore, folhas semicrassas, levemente discolor com face adaxial verde brilhante, frutos
## [150] "Árvore; 10 m; tronco com lenticelas redondas e odor forte.; Terra firme, floresta secundária"
## [151] "Árvore; 11 m .; Terra firme, floresta secundária sobre solo antropogenico (Terra Preta de Índio)
## [152] "Árvore; 4,0 m; botoes florais; flores creme; folhas face adaxial verde-escuro e face abaxial verde-esbranquiçado.; Borda da capoeira. ; HA árvore; 4,0 m; botoes florais; escuro e face abaxial verde-esbranquiçado.; borda da capoeira.; habitat: borda da capoeira.; descricao: "
## [153] "Árvore; 4,0 m; flores e botoes cremes. ; DE"
## [154] "Árvore; 8 m; pecíolo com cera branca.; Terra firme, floresta secundária sobre solo antropogenico"
## [155] "Árvore; 8 m.; Terra firme, floresta secundária sobre solo antropogenico (Terra Preta de Índio)
## [156] "Árvore; flores brancas. Capoeiras e matas..Habitat: Terrestre"
## [157] "Árvore: 5,0m. Folhas coriáceas, discolors, face abaxial pubescente. Frutos imaturos vermelhos"
## [158] "Árvore."
## [159] "Árvore."
## [160] "Árvore."
## [161] "Árvore."
## [162] "Árvore."
## [163] "Árvore."
## [164] "árvore. 10. m. folas alternas, flor amarela.; interior da mata ; HA"
## [165] "árvore. 12. m. folhas alternas.; Mata ; HA"
## [166] "árvore. 8. m. presença de frutos verdes e maduros, associaçao com formigas e ninhas de h
## [167] "árvore. 9. m. presença de botoes florais e flores, floraçao intensa.; Mata (borda) ; HA"
## [168] "Árvore. Amostra 283 - Parcela 6 -
Sao Joao. árvore. amostra 619 - parcela 4 - quizanga."
## [169] "Árvore. Botoes e flores acastanhados. Solo argiloso."
## [170] "Árvore. Botoes florais verdes-claro. árvore. botões florais verdes-claro.;Árvore. Botoes florais verdes-claro."
## [171] "Árvore. Flor creme; .; HABITAT: .; DESC: Árvore. Flor creme;Árvore. Flor creme."

```

```
## [172] "Árvore. Flores com sépalas persistente verde-claras. Fruto imaturo verde escuro."
## [173] "Árvore. Folhas coriáceas, discolores, face abaxial pubescente.; Floresta Estacional Semidecidual; Floresta Estacional Semidecidual"
## [174] "Árvore. Folhas simples, alternas, face abaxial discolor, simples, crassa.&nf;; Floresta Estacional Semidecidual; Floresta Estacional Semidecidual"
## [175] "Árvore. Frequencia alta. Flores cremes."
## [176] "Árvore. Mata de terra firme. árvore.; mata de terra firme."
## [177] "Árvore.; Mata úmida"
## [178] "Árvore.; Mata úmida"
## [179] "Árvore.; Mata úmida árvore.; mata úmida."
## [180] "Árvore.; Orla da mata"
## [181] "Árvore.Habitat: Terrestre"
## [182] "Árvore&nf;Nome vulgar: Louro"
## [183] "Arvoreta , 4m alt., flores amareladas. Coleta de folhas para análises"
## [184] "Arvoreta 3-4m alt., frutos imaturos verdes.; Floresta Atlântica; HABITAT: Floresta Atlântica; DESCRITIVO: Floresta Atlântica; HABITAT: Floresta Atlântica"
## [185] "Arvoreta 4m, flor creme. Encosta rochosa do morro"
## [186] "Arvoreta 5m alt., cálice vermelho; Floresta Atlântica; HABITAT: Floresta Atlântica; DESCRIÇÃO: Floresta Atlântica; HABITAT: Floresta Atlântica"
## [187] "Arvoreta ca. 3,0 m, botoes florais esverdeados.; Cerrado (típico). Substrato arenoso-pedregoso. coleta: xls: spsfantigo; geografia: cerrado&nf;;"
## [188] "Arvoreta ca. 4,0 m, flores amareladas.; Mata ciliar. Afloramento rochoso."
## [189] "Arvoreta ca. 4,5 m, botoes florais esverdeados.; Mata ciliar.&nf;Substrato arenoso-pedregoso."
## [190] "Arvoreta ca. 4.0 m, frutos imaturos.; Cerrado Sujo. Substrato arenoso. coleta: cerrado sujo"
## [191] "Arvoreta ca. 4m alt., flores cremes"
## [192] "Arvoreta ca. 7m alt. Frutos verdes."
## [193] "Arvoreta com +/- 3,5 de alt. Ramos acinzentado.Folhas cartáceas. Inflorescencia ferrugínea. frutos esverdeados .Planta aromática .Frequente."
## [194] "Arvoreta com 2 m; (inflorescencia) frutescencia de cor creme amarelada retirada com uma pinça"
## [195] "Arvoreta em touceira densa; flores cremes. Frequentemente na capoeira de solo arenoso, perto de rios"
## [196] "Arvoreta lenhosa com 3,5m. Indivíduo nº 2467. Folhas verdes discolores, material vegetativo"
## [197] "Arvoreta pouco ramificada 3m. Flores alvas."
## [198] "Arvoreta, ca. 4m, flor creme, inflorescencia e botoes florais cremes, aromáticas."
## [199] "Arvoreta, ca. 5m. Folhas verde discolores, brilhosas. Inflorescencias verde, flores amarelas"
## [200] "Arvoreta, flor branca aromática"
## [201] "Arvoreta; 3,0 m; frutos imaturos; base do revestimento em forma de cálice de cor marrom-escuro; na proximidade do ápice do ramo presença de fungos (cochonilhas). ; DE"
```

```

## [202] "Arvoreta; 3,0 m; pequenos frutos imaturos verde-cremes. PIBIC Jr. 2009/2010.; Vertente. ; HA"
## [203] "Arvoreta; 5 m; frutos com cálice amarronzados e frutos imaturos verdes; tronco cilíndrico"
## [204] "Arvoreta; 7,0 m; frutos imaturos verdes com cúpula verde amarronzada. ; DE"
## [205] "Arvoreta; flor branca."
## [206] "Arvoreta; frutos imaturos verdes e maduros roxos; material conservado com álcool.; Sub-bosque. ; HA"
## [207] "Arvoreta; Tabuleiro litorâneo"
## [208] "Arvoreta."
## [209] "Arvoreta."
## [210] "arvoreta. 2.5. m. botoes verdes, cálice castanho-claro, flores com pétalas brancas.; Mata (borda) ; HA"
## [211] "arvoreta. 2.5. m. flores com pétalas creme.; borda da mata ; HA"
## [212] "arvoreta. 6. m. botoes florais e flores com pétalas creme.; Mata (borda) ; HA"
## [213] "ARVOREZINHA COM RAMOS ESCANDENTED . FRUTOS COM RECEPTÁBULOS AVERMELHADOS , IMATUROS VERD
## [214] "Árvore 14 m. Frutos verdes. Interior da mata."
## [215] "Beira de estrada, antropizada. Contato Savana Floresta/ Floresta Estacional/ Floresta Ombrófila"
## [216] "Bl. creme. Schwarzwasser."
## [217] "Botoes florais creme. Área de influencia direta da UHE Jirau."
## [218] "Botoes florais verdes."
## [219] "Broto de árvore; flores creme; frutos pretos. Mata de terra firme, argilosa."
## [220] "Caatinga arbórea."
## [221] "Cálice verde."
## [222] "Cerrado"
## [223] "Cerrado / floresta ombrófila."
## [224] "Cerrado sensu stricu. Solo arenoso."
## [225] "Cheiro forte na casca e folhas. Verdes descoloridos. Botoes amarelos. Ausentes"
## [226] "Coleta de material estérial do projeto de doutorado do professor Aldenir.; Arvoreta; 6 m"
## [227] "COLETA: Cerradao&nbsp;Liana semi-ciófila; cálice avinosado; frutos imaturos verdes; GEOGRAFIA: Área da Sede Santa Luzia; solo argiloso.&nbsp;Relevo plano.; folhas cor verde, cartáceas, presença de galhas escuras na superfície abaxial da folha;"
## [228] "COLETA: Cerrado arenoso &nbsp;Arbusto 80cm, flores bege.; GEOGRAFIA: A 13km do povoado Chapéu"
## [229] "COLETA: Cerrado com solo arenoso.&nbsp;Semi arbusto formando moitas; flores cor amarelo clara"
## [230] "COLETA: Cerrado&nbsp;Solo com textura arenosa/argilosa; relevo plano.&nbsp;Subarbusto helicônio"
## [231] "COLETA: Flores brancas com glândulas vermelhas.&nbsp;Árvores 11m.; GEOGRAFIA: (R.23, Praça das Rosas"
## [232] "COLETA: Mata Atlântica.&nbsp;Árvore 8m;"

```

```
## [233] "COLETA: Mata Atlântica.&nf;Árvore, parte inferior do fruto vermelha.; GEOGRAFIA: Arvore  
## [234] "COLETA: XLS: SPSFAntigo;"  
## [235] "COLETA: XLS: SPSFAntigo; GEOGRAFIA: cerrado&nf;;"  
## [236] "COLETA: XLS: SPSFAntigo&nf;Árvore; GEOGRAFIA: trilha do vinhatico;"  
## [237] "COLETA: XLS: SPSFAntigo&nf;Árvore. Exemplar ns 19374; GEOGRAFIA: Mata;"  
## [238] "COLETA: XLS: SPSFAntigo&nf;Floresta secundária&nf;Árvore flores creme;"  
## [239] "Cúpulas imaturas verde-marrons, ramos da inflorescencia verdes. Cheiro típico de Lauraceae."  
## [240] "DESC:Arvoreta      2,2m;      flor      creme-  
esverdeada; cupulos persistentes negros; botoes florais creme-  
esverdeados."  
## [241] "DESC:Subarbusto 40cm; fruto imaturo verde. Rara no local."  
## [242] "Erva ereta. Folhas papiráceas. Flores com corola branca."  
## [243] "Espécie ameaçada de extinção (Em perigo). material sem dados da planta. solo pobre, arenoso."  
## [244] "Flores alvas. Floresta preservada, dossel médio 20m."  
## [245] "Flores alvas. Mata de terra firme."  
## [246] "Flores amarelas com perfume agradável."  
## [247] "Flores amarelas, pedicelo verde"  
## [248] "Flores      pequenas,      amarelo-  
claro,      com      perfume.      flores      pequenas,      de      cor      amarelo-  
claro com perfume."  
## [249] "Floresta Atlântica. Arbóreo. Coletada no Porto Capim.; DESC: Floresta Atlântica. Arbórea."  
## [250] "Floresta estacional"  
## [251] "Floresta estacional decidual."  
## [252] "Floresta estacional semidecidual submontana."  
## [253] "Floresta Estacional Semidecidual Submontana. ; HA"  
## [254] "Floresta Estacional Semidecidual Submontana. ; HA"  
## [255] "Floresta estacional troca de material.; árvore ca 8,0 m. folhas cartáceas, discoloração comum."  
## [256] "Floresta Ombrófila Densa"  
## [257] "Floresta Ombrófila Densa"  
## [258] "Floresta Ombrófila Densa"  
## [259] "Floresta Ombrófila Densa"  
## [260] "Floresta Ombrófila Densa"  
## [261] "Floresta ombrófila densa em regeneração."  
## [262] "Floresta Ombrófila Montana"  
## [263] "Floresta Ombrófila Montana"  
## [264] "Floresta ombrófila."  
## [265] "Floresta ombrófila."
```

```
## [266] "Floresta ombrófila."
## [267] "Floresta ombrófila."
## [268] "Floresta ombrófila."
## [269] "Floresta ombrófila."
## [270] "Floresta ombrófila."
## [271] "Flws. yellow. Male."
## [272] "Folhas alternas, glabras."
## [273] "Folhas coriáceas, discolores, inflorescencia axilar, ferruginosa, raque esverdeadao, bot ALCB (48550). Projeto Financiado pelo CNPq. mata."
## [274] "Folhas simples, alternas, alongadas, oblongas, cheiro agradável. P8 I36"
## [275] "fruto maduro"
## [276] "Frutos ainda jovens"
## [277] "frutos verdes"
## [278] "Grand arbor."
## [279] "HABITAT:Cerrado.; DESC:Árvore, frutos verdes.;"
## [280] "Ident. ant.: o. organensis (Meis.) Mez"
## [281] "Inflorescencia com pedúnculo verde-limao, flores com pedicelos, cálice e corola branco-amarelados, androceu e gineceu amarelados, cheiro doce agradável inflorescencia com pedúnculo verde-limao. flores com pedicelos, cálice e corola branco-amarelados. androceu e gineceu amarelados, cheiro doce agradável. inflorescência com pedúnculo verde-limão. flores com pedicelos, cálice e corola branco-amarelados. androceu e gineceu amarelados, cheiro doce agradável.; Inflorescencia com pedúnculo verde-limao. Flores com pedicelos, cálice e corola branco-amarelados. Androceu e gineceu amarelados, cheiro doce agradável."
## [282] "Linha de transmissao Jirau/Sts Antonio."
## [283] "Linha de transmissao Jirau/Sts Antonio."
## [284] "Mata Atlântica"
## [285] "Mata Atlântica"
## [286] "Mata Atlântica"
## [287] "Mata Atlântica (Floresta Estacional Semidecidual); HABITAT: Mata Atlântica (Floresta Es"
## [288] "Mata Atlântica de encosta."
## [289] "Mata Atlântica em vale de tabuleiro."
## [290] "Mata Atlântico, beira da lagoa."
## [291] "Mata Ciliar"
## [292] "Mata ciliar."
## [293] "Mata de Chaves.Borda com canavial.; HABITAT: Mata de Chaves.Borda com canavial."
```

```
## [294] "Mata de terra firme."
## [295] "Mata em leito de córrego."
## [296] "Mata ombrófila secundária. Solo arenoso."
## [297] "Mata perenifólia sob latosolos"
## [298] "Material sem dados da planta e sem data de coleta."
## [299] "Material sem dados da planta."
## [300] "Material sem dados da planta."
## [301] "Município de Humaitá, Pixuna, Km 40 da Rodovia Transamazônica. Campina, flores alvas."
## [302] "Muy alto y ranudo; hoja verde clara; flor amarilla; fruto ovalado, verde claro."
## [303] "Numa capoeira 6m alt., e 20 cm de diam."
## [304] "old secondary forest with few primary remnants"
## [305] "open uplands, \\\muena\\, 3 m, cupule green, fr. green."
## [306] "pequena árvore de formaçao rupestre em frutos. pequena árvore de formaçao rupestre em fru
## [307] "Pétalas creme, anteras amarelada. Área de influencia direta (canteiro)."
## [308] "Planta com 6,0 metros de altura. Fruto e cáile verdes."
## [309] "Planta com fruto."
## [310] "Planta com grossa casca fissurada de coloraçao cinza. A espécie possuiindivíduos de grande
Reserva Ecológica do IBGE. Plantas da Bahia."
## [311] "Pole ca. 1.3 m. Receptacle and fruits green. Mata de terra firme."
## [312] "Pole ca. 1.5-2 m; fruits and receptacle green. Disturbed roadside margins in tall forest, 2 m;
fruits and receptacle green. disturbed roadside margins in tall forest, terra firme. laterized
2 m; fruits and receptacle green. Disturbed roadside margins in tall forest, terra firme. Laterized
## [313] "Possui acúleos."
## [314] "Restinga. Floresta."
## [315] "Sem dados. sem dados.; sem dados."
## [316] "Shrub 3 1/2 tall. Brushy clearing and secondary rain forest, sandy soil."
## [317] "Shrub 3 m. Immature fruit (includng cupule) green."
## [318] "Shrub. Brushy clearings and secondary rain forest, sandy soil."
## [319] "Shrubs and tree of indeterminable size pendent or leaning out over cliff; fruit green; fl
white. Common pole along the river. N to NE bank-
shale cliffs to terra firme."
## [320] "Small tree 5 m; flowers cream."
## [321] "Small tree 5-6 m; flowers yellowish; receptacle red. Riverbank and cliff vegetation."
## [322] "Small tree, 5 m high. Leaves glossy green above, greyish sheen beneath. Terminal infloresc
## [323] "Small tree; 4 cm. dbh."
## [324] "Término coleta 02.04.1979"
## [325] "Terrestre, arbórea, fruto maduro e cor roxa, fruto imaturo de cor verde, exsudaçao incold
```

```

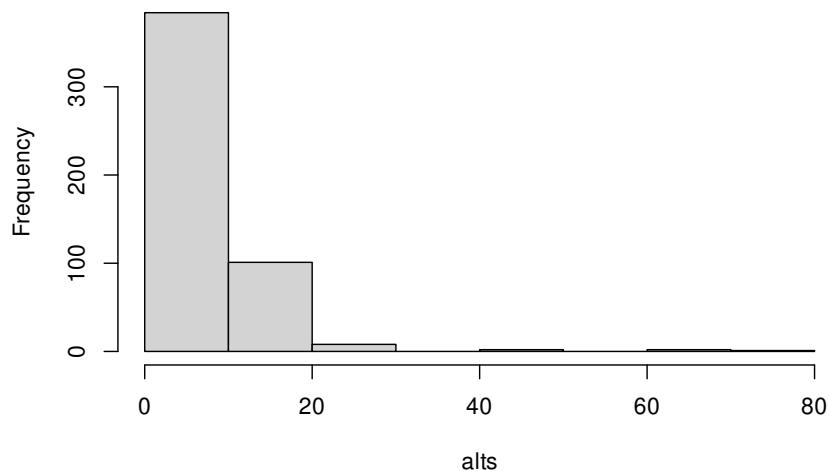
## [326] "Trabalho de Conclusao de Curso"
## [327] "Tree # 2685, 14-15m; fruit lighter green in darker green cup."
## [328] "Tree +- tall, leaves coriaceous, glossy and dark green above, lustrous pale brown beneath"
## [329] "Tree 18. bole 12m, straight, cylindrical. Fruits, flower small"
## [330] "TREE 30 M X 80 CM DIAM. CUPULE DARK GREEN, FRUIT PALE GREEN."
## [331] "TREE 6 M X 10 CM DIAM, BUDS GREEN."
## [332] "TREE CA 5 M. FRUITS LIGHT GREEN, RECEPTACLE DARKER GREEN. FLOWERS CREAM WHITE."
## [333] "Tree ca. 2.5-3 m; flowers cream-white. Disturbed roadside margins in tall forest, terra firme; laterized clay with sand deposits. 3 m; flowers cream-white. disturbed roadside margins in tall forest, terra firme; laterized clay w
## [334] "Tree ca. 3 m; fruits green, becoming red-tinged; receptacle green. Disturbed roadside margins in tall forest, terra firme. Laterized clay w
tinged; receptacle green. disturbed roadside margins in tall forest, terra firme. laterized clay w
tinged; receptacle green.;Tree ca. 3 m; fruits green, becoming red-tinged; receptacle green. Disturbed roadside margins in tall forest, terra firme. Laterized clay w
## [335] "Tree shrub juvenile... revisar;Tree shrub juvenile"
## [336] "tree shurb juvenile"
## [337] "Tree to 25m.tall, 70cm. DBH;wood medium soft. Bark smooth;lvs. glaucous beneath, not aron
## [338] "Tree, 60cm dbh. Leaves collected from forest floor. Transect tree 2:7/4a. Vegetative eco
## [339] "Treelet, 15 m; buds pale green. Primary forest."
## [340] "Treelet, 4 m DBH; 3 cm, bark gray, brts grayed green with slight bloom; lvs yellow-green, shining above, paler, more or less glaucous beneath; fls orange-yellow. Same as no. 328."
## [341] "Tronco com a base reta. Casca externa cinza. Ramos jovens com quinas."
## [342] "UHE Teles Pires"
## [343] "UHE Teles Pires"
## [344] "UHE Teles Pires"
## [345] "VINY SHRUB, CA 4.5M TALL. WOOD IN ALCOHOL"

```

```

# histograma
hist(alts)

```

Histogram of alts

```
# adiciona ao conjunto de dados  
dados$ALTURA_m <- alts
```

10.4.2 Obtendo valores de DAP

```
# aplica a função  
daps <- sapply(txt.org, pegadap)  
head(daps)
```

```
## ; HABITAT: Floresta Ombrófila Densa Submontana.; DESC: Árvore  
## ; Hábito: Árvore; Cor da flor: brancas (botoes brancos); Cor do fruto: árvore de 10 m de altura,  
##  
##
```



```
## [17] "Ambiente: Mata Atlântica Costeira (Mata de Tabuleiro). Habito: Árvore, 18m. tree, 18 m; f
## [18] "Ambiente: Mata Atlântica. Habito: Árvore, 7-8m."
## [19] "Ambiente: Mata Ciliar. Habito: Árvore, ca. 4m. coleta: mata ciliar, sub bosque.&nf;árvor
242 a ca.19km.&nf;ex huefs 18526.;""
## [20] "Ambiente: Mata de Cipó com . Habito: Árvore, 5m. tree to 5 m. leaves coriaceous dark green
green.;Tree to 5 m. Leaves coriaceous dark green above, pale beneath. Flowers pale grey-
green."
## [21] "Ambiente: Mata Estacional Semidecidual com Mata ciliar. Habito: Árvore, ca 14m. floresta
## [22] "Ambiente: Mata higrofila, úmida. Habito: Árvore maior que 5m e menor 20m"
## [23] "Ambiente: Mata junto a plantação de coco. Habito: Arvoreta, 3m. arvoreta. 3. metros. flor
argiloso ; ha"
## [24] "Ambiente: Mata junto aplantação de coco. Solo sílico-
argiloso. Habito: Árvore, 6m X 15cm. árvore. 6. metros. 6 m x 15cm, flores creme, recobertas por p
argiloso ; ha"
## [25] "Ambiente: Mata Ombrófila Densa Montana. Habito: Arvoreta, ca. 4m."
## [26] "Ambiente: Mata Pluvial. Habito: Árvore, 8m."
## [27] "Ambiente: Mata. Borda com canavial. Habito: Árvore, ca. 6m."
## [28] "Ambiente: Mata. Borda da mata. Habito: Árvore, 10 a 12m."
## [29] "Ambiente: Mata. Habito: Árvore, 6m."
## [30] "Ambiente: Mata. Habito: Árvore, ca 15m. árvore ca. 15 m alt. folahs com indumento castanh
amareladas.;Árvore ca. 15 m alt. Folahs com indumento castanho. Flores creme-
amareladas."
## [31] "Ambiente: Mata. Habito: Árvore, ca. 9m"
## [32] "Ambiente: Mata. Habito: Árvore, DAP 37,2cm."
## [33] "Ambiente: Mata. Habito: Árvorezinha, 7m."
## [34] "Ambiente: Mata. Interiro da mata. Habito: Árvore, 10m."
## [35] "Ambiente: Mata. Na borda da mata. Habito: Árvore, 5 a 6m."
## [36] "Ambiente: Mata. Na borda da mata. Habito: Árvore, c.a 9m."
## [37] "Ambiente: Mata. Na borda da mata. Habito: Árvore, ca. 4 a 5m."
## [38] "Ambiente: Mata. Na borda da mata. Habito: Árvore, ca. 6,5m."
## [39] "Ambiente: Mata. No interior da mata. Habito: Árvore, 8 a 10m."
## [40] "Ambiente: Restinga. Habito: Árvore 8m X 0,30m. árvore com 8m alt., 30m de diâmetro. flor b
## [41] "AQrvore de 16m"
## [42] "Arbol 10 m de altura, flores blancas, frutos verdes."
## [43] "Arbol 7 m, DAP=8 cm, simpódico, aceites etéreos en las hojas. Frutos verde-
amarillentos."
## [44] "Arbol 8 m, DAP: 11.1 cm, madera aromática, estéril."
```

```
## [45] "ARbol de 10 m. Botones y raquis amarillos"
## [46] "Arbol de 10 m. Ovario verde. Estigma persistent café. Fruit café. Creciendo al lado del ca
## [47] "ARbol de 10-12 mts. de altura, frutos verdes. Ocasional."
## [48] "Arbol de 12 m. Flores verdosas. Común."
## [49] "Arbol de 14-16 m, DAP: 25.1 cm, frutos inmaduros verdes, morados al madurar."
## [50] "Arbol de 15 m. Botones florales verdes."
##      [51] "árbol de 17m; corteza externa lisa-
ploma oscura; flores en botones"
## [52] "arbol de 20 m de altura, fruto ovalados verdes; receptaculo cupular rojo. Olor aromático"
## [53] "Arbol de 20 m de altura, fruto verde, semilla rosada. Abundante."
## [54] "Árbol de 4,5 m. Flores cremosas"
## [55] "Arbol de 5 m. Frutos verde pálidos, cáliz carnoso, verde oscuro."
## [56] "Arbol de 6 m, ramitas muy angulosas."
## [57] "Arbol de 8 m. Flor amarilla. Fruto con cupula verde oscuro y semilla verde-
claro."
## [58] "Arbol de 8 m. Frutos verdes con apics amarillo. Botones florales blancos."
## [59] "Arbol de 8 m. Frutos verdes. Creciendo en bosque secundario."
## [60] "Arbol de 8-10 mts. de altura, frutos verdes, receptáculo rojo. Ocasional."
## [61] "Arbol de 8.0 m de alto. Ramificado basalmente. DAP de 12.0 cm. Tronco de corteza gris. Ram
## [62] "Arbol de 9 m. Esteril."
## [63] "Arbol de 9 metros, ramitas angulosas."
## [64] "Arbol de 9-10 mts, frutos verdes, follage denso na extendido, zona al borde del camino."
## [65] "Arbol de aproximadamente 10 m alto. Frutos verdes en el exterior, morados internamente y
## [66] "Árbol joven de 8m alt. Esteril"
## [67] "Arbol, 20 metres de altura. Frutos drupas verdes, dentro de capsula negra +/--
leñosa."
## [68] "Arbolito de 5 metros, glabro, con las ramitas angulosas. Muy escaso, se observó solo un in
## [69] "Arbórea com ca. 1,2 m de alt.; infl. com flores amarelo-
esbranquiçado arbórea com ca. 1,2m de altura; inf. com flores amarelo-
esbranquiçado."
## [70] "Arbóreo, terrestre, fuste cilíndrico, casca áspera, altura total 1,5m, fuste 10m, cap fus
## [71] "Arbóreo, terrestre, fuste cilíndrico, casca áspera, altura total 12m, fuste 6m, cap fuste
## [72] "Arbóreo, terrestre, fuste cilíndrico, casca áspera, altura total 13m, fuste 9m, cap fuste
## [73] "Arbóreo, terrestre, fuste cilíndrico, casca áspera, altura total 15m, fuste 10m, cap fus
## [74] "Arbóreo, terrestre, fuste cilíndrico, casca áspera, altura total 15m, fuste 6m, cap fuste
## [75] "Arbóreo, terrestre, fuste cilíndrico, casca áspera, altura total 15m, fuste 8m, cap fuste
## [76] "Arbóreo, terrestre, fuste cilíndrico, casca áspera, altura total 16m, fuste 7m, cap fuste
```

[77] "Arbóreo, terrestre, fuste cilíndrico, casca áspera, altura total 4m, fuste 2m, cap fuste
[78] "Arbóreo. Altura de 5,0 m e com fruto imaturo verde."
[79] "Arbóreo. Inflorescencia patente, botoes verdes.; DESC: Arbóreo. Inflorescencia patente,
[80] "arbre, 11 m, 9 cm DBH. fr. vert r points blancs, clair r lapex, plus foncé r la base, clair
[81] "arbre, 8 m, 10 cm DBH; fleur trimcres, 3+3 tép. jaunes recouverts de duvet roux, 3+3+3 éta
[82] "Arbustivo, fruto verde e preto; Costeiro"
[83] "Arbusto 3m. campo rupestre"
[84] "Arbusto 8 m alto. Frutos verdes."
[85] "arbusto até 3m de altura. Flores e botões amarelos"
[86] "Arbusto ca. 1 m alt.; caule amarronzado; frutos verdes."
[87] "Arbusto ca. 1-2m de altura; caules alados, folhas subcorláceas. arbusto.; floresta secunda
[88] "Arbusto ca. 1,50m alt.; caule amarronzado; frutos imaturos verdes."
[89] "Arbusto ca. 2 m alt.; inflorescencia em botoes amarronzados; frutos imaturos verdes"
[90] "Arbusto ca. 2 m alt.; ramos marrom-
escuro; folhas adaxialmente verde brilhante e abaxialmente verde claro; flores amareladas."
[91] "Arbusto ca. 2 m. Botoes ferrugíneos.HABITAT:Mata Atlântica.;"
[92] "Arbusto ca. 3 m alt.; flores alvacentas."
[93] "Arbusto ca. 3,0m, frutos imaturos. Subst. arenosos.; Cerrado (Típico). coleta: xls: spsf
[94] "Arbusto ca. 3.0 m. alt. Folhas verdes concolores. Flores em botoes amarelados"
[95] "Arbusto ca. de 1m de altura. Botoes cremes.HABITAT:Caatinga com solo argilo-
arenoso.; arbusto, ca. de 1,0m de altura, botoes creme, perianto, estames creme. arbusto, ca. de 1m
[96] "Arbusto cerca de 1,30 m."
[97] "Arbusto cerca de 2m de altura, ramos acinzentados. Botoes florais ferrugíneos. Planta arc
[98] "Arbusto com ca. de 3,5m. de altura; frutos imaturos verdes e os maduros venoso-
arroxeados; cálice cupuliforme nigrescente. arbusto de 3,5m de altura. frutos imaturos verdes e o
arroxeados. cálice cupuliforme nigrescente."
[99] "Arbusto de 2 m de altura; inflorescencia creme. arbusto de 2 m de altura; inflorescência c
[100] "Arbusto de 2,5 m de altura; inflorescencia esbranquiçada. Capoeira de solo arenoso. arb
[101] "Arbusto de 2,50 m de altura; botoes florais amarelos. Mata secundária; solo arenoso. arb
[102] "ARBUSTO DE 3 M DE ALT., INFLOR. AMARELADA. município de oriximiná, rio trombetas, margem
[103] "Arbusto de 3 m de altura; flores amarelas. Terreno arenoso, capoeira."
[104] "Arbusto de 3 m de altura; flores cremes. Capoeira aberta, terreno arenoso."
[105] "Arbusto de 3 m de altura; frutos verdes, parte apical amarela. Terra firme."
[106] "Arbusto de 3 m de altura; flores creme. Mata virgem, solo arenoso."
[107] "Arbusto de 3m de altura. Frutos verdes e cálice marrom. Terra firme e solo arenoso."
[108] "Arbusto de 3m, inflorescencia creme, fruto vermelho ainda jovens"
[109] "Arbusto de 4 m de altura; flores amareladas, botoes esverdeadas. Mata de igapó, solo arenoso."

```
## [110] "Arbusto de 4 m de altura; flores amareladas. Terra firme; solo arenoso."
## [111] "Arbusto de 4 m de altura; flores amarelas. Capoeira; solo arenoso."
## [112] "Arbusto de 4 m de altura; flores e estames amarelos; frutos imaturos de cor verde. Campin
## [113] "Arbusto de 4 m de altura; frutos imaturos verdes. Mata de terra firme; solo argilo-
arenoso."
## [114] "Arbusto de 4 m. de altura, frutos imaturos verdes."
## [115] "Arbusto de 4m de altura, frutos imaturos verdes. Mata na beira da estrada."
## [116] "Arbusto de 5 m de altura; botoes florais amarelo esverdeados. Mata de terra firme, solo a
argiloso."
## [117] "Arbusto de 5 m de altura; botoes florais amarelos. Mata de terra firme; solo argilo-
pedregoso."
## [118] "Arbusto de 5 m de altura; frutos verdes. Capoeira."
## [119] "Arbusto de 5 m de altura; inflorescencia em botoes e flores verde-
amareladas. Capoeira de solo argiloso. arbusto de 5 m de altura; inflorescência em botões e flores
amareladas. capoeira de solo argiloso.;Arbusto de 5 m de altura; inflorescencia em botoes e flores
amareladas. Capoeira de solo argiloso."
## [120] "Arbusto de 5m de altura. Flores e botoes amarelos."
## [121] "Arbusto de 6 m de altura. arbusto de 6 m. de altura."
## [122] "Arbusto ramoso; ramos acinzentados; folhas coriáceas, abaxialmente verde mais claro; fru
## [123] "Arbusto, 1,30 m de altura, com frutos jovens de coloração verde, frequencia mediana. arb
## [124] "Arbusto, 3 m., 15 cm."
## [125] "Arbusto, 3 m., 15 cm."
## [126] "Arbusto, ca. de 2,5m de altura, frutos pequenos de cor verde"
## [127] "Arbusto, ca. de 3m de altura. Botoes amarelos. arbusto, ca. de 3m de altura. botoes amare
## [128] "Arbusto, flor branca esverdeada"
## [129] "Arbusto, folhas simples alternas. Flores pequenas, pouco vistosas."
## [130] "Arbusto, ocasional, ca. 1,60m, caule verde-
amarronzado, folha levemente papirácea, adaxialmente verde brilhante, infloresc. jovem verde-
alvascente; Restinga"
## [131] "Arbusto; flores amarelas; femea. arbusto; flores amarelas; fêmea."
## [132] "Arbusto; ramos amarronzados; folhas coriáceas, adaxialmente verde-
escuro brilhante e abaxialmente verde-claro; flores amareladas."
## [133] "Área de influencia direta (canteiro)."
## [134] "Área de influencia indireta."
## [135] "Árvore com cerca de 5m. Flores alvas."
## [136] "Àrv. 10m alt. árvore de 10m de altura. folhas simples, alternas, espatulada a lanceolado.
2mm. inflorescencia racemos. floresta secundária contígua a floresta primária. árvore de 10m de al
```

2mm. inflorescência racemos. floresta secundária contígua a floresta primária. rrv. 10m alt.; Árvore
2mm. Inflorescencia racemos. floresta secundária contígua a floresta primária."
[137] "Árv. de 6m x 10cm de diâm; flores brancas, estames amarelas. árvore de 6m x 10cm de diâm.,
[138] "Àrv. de aprox. 6m de alt. fol. alternas, botões florais amarelos, fl. de corola ccreme. á-
[139] "Árvore"
[140] "ÁRVORE ."
[141] "Árvore 10 m 20 cm de diam.; frutos imaturos verdes-
pálidos. receptáculos verde-escuro. Transição entre praia arenosa de igapó e capoeira arenosa e ve-
pálidos. receptáculos verde-escuro. transição entre praia arenosa de igapó e capoeira arenosa e ve-
[142] "Árvore 10 m; flores ferrugíneas; muito frequente, tanto na orla como no interior da mata."
[143] "árvore 10 m., flores brancas coordinate uncertainty: approx. in a 1000 m. radius image un-
[144] "Árvore 10,5m, botoes florais amarelos, ramos com odor cítrico agradável. Mata Atlântica."
[145] "Árvore 10m alt. Folhas alternas, coriáceas, fruto seco, grande, verde."
[146] "Árvore 10m x 20cm de diâm., flores brancas; Mata costeira; HABITAT: Mata costeira; DESC:
[147] "Árvore 15m, frutos pequenos um pouco ovais, caule cor de kaki escuro, mata t.f.; DESC: Ár-
[148] "Árvore 17 m, flores brancas. coleta: árvore 17 m, flores brancas.;"
[149] "Árvore 18m, frutos imaturos de cor verde. Mata."
[150] "Árvore 20m, fuste 15m X 35cm, casca parda inteira com marcas amarelas, odor forte de cane-
[151] "Árvore 25 m. Casca lisa. Cória aromático. Espécie abundante no entorno."
[152] "Árvore 3m. Frutos velhos. mata ciliar"
[153] "Árvore 4m, flores amarelas pálidas (cor de creme). Mata Atlântica. árvore, 4 m de altitud-
[154] "Árvore 4m. Frutos imaturos verdes. mata ciliar"
[155] "Árvore 4m. Frutos imaturos verdes. mata ciliar"
[156] "Arvore 5 m, fruto preto"
[157] "Árvore 5m altura; folhas coriáceas, discolores, botoes e inflorescencias ferrugíneas. á-
6757; mata."
[158] "Árvore 5m de altura, flores em botoes cremen, coleta de folhas para análises, nome vulgar-
[159] "Árvore 5m, tronco ramificado. Frutos deiscendo, cúpula vermelha."
[160] "Árvore 5m. Botoes esverdeados."
[161] "Árvore 6 m de altura, flores alvo-esverdeadas"
[162] "Árvore 7 m alt., folhas discolores, verdes com nervuras verde-
claras na face adaxial, verde-claras na abaxial, perianto alvo-
hialino, glandulas amarelas, estames verde-claros."
[163] "Arvore 7 m alt., folhas opacas, discolores. Infrutescências pendentes com frutos verde-
esveralda cheios de pontos brancos, e cúpula rugulosa verde-opaca"
[164] "Árvore 7m, frutos imaturos verdes"
[165] "Árvore 7m. Frutos verdes."

```

## [166] "Árvore 8 m de altura. Folhas coriáceas, alternas, glabras, fruto verde com cúpula."
## [167] "Árvore 8 m., flor creme."
## [168] "árvore 8 m., frutos amarelo esverdeado coordinate uncertainty: approx. in a 1000 m. radius
## [169] "Árvore 8m. Frutos imaturos."
## [170] "Árvore aprox. 6 m de altura, flores alvas.; Mata Atlântica; HABITAT: Mata Atlântica; DESCRIÇÃO: folhas
## [171] "Árvore aproximadamente de 4 m de altura; fruto apreciado pelos passarinhos; flores esverdeadas"
## [172] "Árvore até 11m, frutos verdes."
## [173] "Árvore até 20 m alt. Flores brancas."
## [174] "Árvore c/ca. 4 m alt.; Botoes creme; Fruto com cálice persistente.HABITAT:Mata degradada"
## [175] "Árvore ca 5 m. Inflorescencia em botoes."
## [176] "Arvore ca 5m de alt, botoes florais ferruginosos, arvore ca 5m de alt, botões florais ferruginosos"
## [177] "Árvore ca. 10 m alt.; caule amarronzado; flores amarronzadas. árvore ca. 10 m alt.; caule amarronzado"
## [178] "Árvore ca. 10 m de alt., flores alvo-amareladas"
## [179] "Árvore ca. 11m, com frutos verdes."
## [180] "Árvore ca. 12 m alt.; frutos imaturos verdes."
## [181] "Árvore ca. 12 m de altura com 0,3m de diâmetro na base. Flores amarelas sem perfume. Sépala amarela"
## [182] "Árvore ca. 15 m alt., copada; folhas e lenho aromáticos; folhas coriáceas; flores amarronzadas"
## [183] "Árvore ca. 15 m alt.; flores amareladas. arvore ca. 15m alt.; flores amareladas."
## [184] "Árvore ca. 15 m alt.; folhas coriáceas com face abaxial verde-acinzentada; flores com tépalas marrom-claras. coleta: mata higrófila.&nbsp;árvore 15m; flores com tépalas marrom claras.; geografia: cedro-vermelho"
## [185] "Árvore ca. 15m de altura. Ramos enegrecidos. Folhas com face adaxial verde e abaxial glauca"
## [186] "Árvore ca. 18m; folhas discolors verdes; coriáceas; frutos imaturos verdes. árvore. 8. m de altura"
## [187] "Árvore ca. 2 m de altura. Botoes cremes árvore ca. 2 m de altura. botões cremes;Árvore ca. 2 m de altura"
## [188] "Árvore ca. 2,5 m alt.; folhas simples alternadas, levemente discolors abaxialmente verde-claro e tomentosos, adaxialmente verde-escuro. árvore ca. 2,5m de altura, folhas simples alternadas, levemente discolors abaxialmente verde-claro"
## [189] "Árvore ca. 20m de altura, nao marcada. Folhas verdes discolors com rede de venação amarela"
## [190] "Árvore ca. 3 m de altura. Frutos verdes."
## [191] "Árvore ca. 3m de altura com 5cm de diâmetro; flores levemente perfumadas, diminutas, verdes pálidas, quase amarelas. madeireira."
## [192] "Arvore ca. 4 m. altura. Frutos imaturos verdes com receptáculo avermelhado"
## [193] "Árvore ca. 5 m de altura. Botoes florais cremes. árvore ca. 5 m de altura. botões florais cremes"
## [194] "Árvore ca. 5,0m, frutos imaturos.; Cerrado, Areno-pedregoso."
## [195] "Árvore ca. 5.0 m, frutos imaturos.; Campo Cerrado. Substrato arenoso"

```

pedregoso. coleta: xls: spsfantigo; geografia: cerrado&nf;;; Árvore ca. 5.0 m, frutos imaturos.; C
pedregoso."

[196] "Árvore ca. 6 m alt.; caule marrom; folhas levemente papiráceas; frutos imaturos verdes. c
[197] "Árvore ca. 6 m alt.; folhas brilhantes na face superior, flores alvas."

[198] "Árvore ca. 6m de altura; flores diminutas sem perfume, cálice e corola amarela; frutos ve
[199] "Árvore ca. 6m, flores creme."

[200] "Árvore ca. 7 m alt.; caule marrom; folhas levemente papiráceas; flores creme-
amarronzadas. árvore ca. 7 m alt.; caule marrom; folhas levemente papiráceas; flores creme-
amarronzadas.; orla da mata. árvore ca. 7 m de altura; caule marrom; folhas levemente papiráceas; f
amarronzadas. árvore ca. 7m alt; caule marrom; folhas levemente papiráceas; flores creme-
amarronzadas orla da mata"

[201] "Árvore ca. 7, 0 m, botoes florais esverdeadas. Subst. aren-
pedregoso.; Cerrado (Típico). coleta: xls: spsfantigo; geografia: cerrado&nf;;;"

[202] "Árvore ca. 7m de altura com 20cm de diâmetro. Folhas simples, alternas, grandes, lisas, g

[203] "Árvore ca. 7m, frutos pretos em forma de sino"

[204] "Árvore ca. 8 m alt. e 52 cm CAP; tronco castanho com bastante lenticelas bem desenvolvida

[205] "Árvore ca. 8 m alt.; em frutificaçao dentro da mata. árvore com 8m de altura, em frutifica

[206] "Árvore ca. 8 m alt.; folhas coriáceas; frutos imaturos."

[207] "Árvore ca. 8 m alt.; frutos em desenvolvimento. Individuo 75. coordenadas do material: 3
22,7s; 49°58-24,5w, indivíduo 75 (na ident. de pedro l. r. moraes consta que pode ser glomerata)"

[208] "Árvore ca. 8 m alt.; frutos imaturos verdes. árvore. 8. metros. frutos imaturos verdes. ;

[209] "Árvore ca. 8 m, com inflorescencia de coloração amarela-
esverdeada. árvore ca. 8 m, com inflorescência de coloração amarela-
esverdeada. árvore ca. 8m, com inflorescencia de coloração amarela-
esverdeada.; Árvore ca. 8 m, com inflorescencia de coloração amarela-
esverdeada."

[210] "Árvore ca. 8m de altura com 4cm de diâmetro; flores diminutas com coloração pardo; frutos
claro. madeira sem látex madeireira."

[211] "Árvore ca. 8m de altura, DAP: 3,3, tronco cilíndrico. Ramos com fissuras verdes, folhas s

[212] "Árvore ca. 8m de altura; frutos imaturos verdes. árvore com ca. de 8m de alt. frutos imatu
branco.habitat:mata atlântica; árvore com cerca de 8m de altura.; mata atlântica. ; ha"

[213] "Árvore ca. de 10m de altura. Botoes cremes."

[214] "Árvore ca. de 12m de altura. Botoes creme-amarelados."

[215] "Árvore ca. de 12m de altura. Frutos imaturos verdes e maduros marrons."

[216] "Árvore ca. de 18m de altura. Botoes imaturos alvos. Ocorrencia frequente."

[217] "Árvore ca. de 20m de altura. Frutos imaturos verdes."

[218] "Árvore ca. de 2m de alt. Botoes florais creme.HABITAT:Mata Atlântica em encosta; árvore c

[219] "Árvore ca. de 3m de altura. Botoes cremes.HABITAT:Mata Atlântica.; herbácea. flores pequenas amarelas."

[220] "Árvore ca. de 4 a 5 m de altura; caule marrom-escuro, lenticelados; folhas coriáceras; fruto com cúpula que recobre cerca de metade do fruto. árvore escuro, lenticelados. folhas coriáceas. frutos com cupula que recobre cerca de metade do fruto."

[221] "Árvore ca. de 4 a 5 m de altura; caule marrom-escuro, lenticelados; folhas coriáceras; fruto com cúpula que recobre cerca de metade do fruto. árvore escuro, lenticelados. folhas coriáceas. frutos com cupula que recobre cerca de metade do fruto."

[222] "Árvore ca. de 4m de altura. Botoes florais cremes e flores amarelas.HABITAT:Mata Atlântica."

[223] "Árvore ca. de 5 metros de altura. Flores com botoes amarelos."

[224] "Árvore ca. de 5,0m de alt. folhas descoloradas, ramos castanhos, botoes verdes e flores creme."

[225] "Árvore ca. de 5m de altura. Flores creme. Ocasional."

[226] "Árvore ca. de 6 m de altura; caule marrom-escuro, lenticulado; frutos imaturos de coloração verde. árvore ca. de 6 m de altura; caule marrom-escuro, lenticulado; frutos imaturos de coloração verde. árvore cerca de 6 metros de altura; caule marrom-escuro, lenticulado; frutos imaturos de coloração verde.; borda da mata.; habitat: borda da mata.; escuro, lenticulado; frutos imaturos de coloração ver"

[227] "Árvore ca. de 6m de alt., folhas com margem lisa, inflorescências em início de floração, folhas descoloradas."

[228] "Árvore ca. de 7,5m de altura, visitada por formigas. Flores e botões florais creme. Frutos secos."

[229] "Árvore ca. de 7m de altura. Botoes florais verdes. Frutos secos.HABITAT:Floresta Estacional."

[230] "Árvore ca. de 8 a 9 m de altura; Córtez que sofre oxidação ficando avermelhado; frutos com apicalidades."

[231] "Árvore ca. de 8 a 9 m de altura; Córtez que sofre oxidação ficando avermelhado; frutos com apicalidades."

[232] "Árvore ca. de 8m de altura. Botoes creme-ferrugíneos."

[233] "Árvore ca. de 8m de altura. Frutos verdes.HABITAT:Floresta Estacional.; árvore. cerca de 8m de altura; frutos verdes."

[234] "Árvore cerca de 15m de altura; DAP cerca de 25cm; folhas mais claras nas terminações dos ramos."

[235] "Árvore cerca de 4 metros; caule lenticulado. Folhas coriáceas, apiculadas. Frutos com a forma de bala."

[236] "Árvore cerca de 4,5m de altura, receptáculo vermelho. Espécie comum no local."

[237] "Árvore cerca de 7 metros de altura; tronco lenticulado, folhas cartáceas. Inflorescências amarelas."

[238] "Árvore cerca de 8m de altura; caule enegrecido; botoes florais amarelos. Frequentes."

[239] "Árvore cerca de 9 metros, caule acinzentado lenticulado, folhas pilosas na face abaxial."

[240] "Árvore com ± 8 m alt. Flores brancas-amareladas, com leve perfume agradável."

[241] "Árvore com 10 a 12 m de alt. Folhas rígidas coriáceas. Flores de coloração creme."

[242] "Árvore com 10 m alt. Folhas coriáceas. Flores creme, com tépalas eretas. árvore, 10m; folhas descoloradas."

[243] "Árvore com 15 a 16 m de altura;Caule marrom-escuro, lenticulado; Ramos cilíndricos, escuros, pilosos, folhas coriáceas; frutos com a cúpula envolvidos parcialmente."

[244] "Árvore com 15 m de alt. Folhas coriáceas; Frutos de coloração verde envolvidos parcialmente."

[245] "Árvore com 15m. Inflorescência verde."

```
## [246] "Árvore com 16 m de altura. cálice verde."
## [247] "Árvore com 5 a 6 metros de altura; Ramos lenticelados, folhas coriáceas, frutos com cúpula
## [248] "Árvore com 5 metros de altura; flor branca muito perfumada."
## [249] "Árvore com 6,0m caído ao chao; flores passadas; frutos imaturos verdes; cálice persistente"
## [250] "Árvore com 7 m de altura. Fruto com cálice esverdeado."
## [251] "Árvore com 8 a 10 m de alt. Ramos cilíndricos tomentosos, folhas coriáceas. Flores de color
## [252] "Árvore com 8,0 metros de altura; inflorescencia esverdeada."
## [253] "Árvore com 8m de alt., flores com cálice e corola cremes, cheiro agradável árvore com 8m de
## [254] "ÁRVORE COM 8M DE ALTURA, DIÂMETRO DA COPA 6M; FANERÓFITA. FOLHA ESPATIFORME; FLOR PEQUENA
## [255] "Árvore com aprox. 6 m de altura, flores creme, grande presença de formigas, casca acinzentada
## [256] "Árvore com aproximadamente 15m de altura, cap. 57cm."
## [257] "Árvore com aproximadamente 3 a 4m de altura. Flores alvas, frutos novos, folhas alternas. Árvore
## [258] "Árvore com aproximadamente 3m de altura. Flores alvas, folhas alternas. Planta vivendo dentro de
## [259] "Árvore com até 14 m; ramos aromáticos; inflorescencia alvacenta."
## [260] "Árvore com ca. 10m de altura; frutos imaturos verdes."
## [261] "Árvore com ca. 12m de altura, fuste de ca. 6m de altura, com flor de cor amarelo e botões florais
## [262] "Árvore com ca. 15 m de altura. Flores amarelas. Flores em álcool."
## [263] "Árvore com ca. 3 m de altura; tronco com DAP de 22 cm; frutos maduros com uma coloração de
## [264] "Árvore com ca. 4m de altura; infl. com flores creme esverdeados com botões abertos, sem flor
## [265] "Árvore com ca. 6,5m alt.; Inflorescência em panículas creme tomentosas; presença de domo
## [266] "Arvore com ca. 7 m de altura. Folhas descoloridas com face superior mais escura, lucida. Inflor
## [267] "Árvore com ca. 7m de altura; eixo floral verde-creme; cálice, corola e androceu creme-amarelados. Anteras quando maduras marrons; estigma preto; flores e folhas aromáticas."
## [268] "Árvore com ca. de 4,0m. Frutos esverdeados quando imaturos."
## [269] "Árvore com ca. de 4m de altura. Frutos maduros marronsHABITAT:Mata Atlântica - Borda da Mata - Encosta.;" 
## [270] "Árvore com ca. de 5 a 6 metros de altura; caule escuro; folhas coriáceas descoloridas; inflorescência
## [271] "Árvore com ca. de 5 m de altura. Folhas descoloridas. Flor: inflorescência amarronzada. árvore
## [272] "Árvore com ca. de 5m de alt. Botoes amarelos.HABITAT:Mata Atlântica.;" árvore. 5. m. botoes amarelos
## [273] "Árvore com ca. de 5m de alt. Botoes florais ferrugíneos.HABITAT:Mata Atlântica; árvore, flor
## [274] "Árvore com ca. de 5m de alt. Botoes imaturos amarelo-esverdeados.HABITAT:Mata Atlântica;" 
## [275] "Arvore com ca. de 5m. Flores em racemo diminutas."
## [276] "Árvore com ca. de 6 m de altura. árvore com ca. de 6m de alt. frutos verdes imaturos.habit
## [277] "Árvore com ca. de 7m de altura. Flores amarelasHABITAT:Mata Atlântica - Grotas (borda da mata).;" árvore com cerca de 7m de altura; coletada na mata atlântica -
```

grota (borda da mata); flores amarelas; n.v. louro-branco.habitat:mata atlântica;"
[278] "Árvore com ca. de 7m de altura. Flores cremes.HABITAT:Mata Atlântica secundária.;"
[279] "Árvore com ca. de 8m alt., cheiro de pitanga, flores em botaõ árvore com cerca de 8m de al
[280] "Árvore com ca. de 8m de altura. Caule enegrecido, botoes florais amarelos. Frequentе.HAB
[281] "Árvore com cerca de 10 m. de altura. Folhas maduras. Flores alvo-
amarelas."
[282] "Árvore com cerca de 10m de altura; frutos maduros negros."
[283] "ÁRVORE COM CERCA DE 12M DE ALTURA. FLORES COM PÉTALAS ALVAS E ESTAMES MARRONS. FREQUENTE.
[284] "Árvore com cerca de 15m. Fruto imaturo verde."
[285] "Árvore com cerca de 18m de altura. Entrecasca com cheiro de pitanga. Frutos maduros amrel
[286] "Árvore com cerca de 2m de altura, indivíduo jovem. Fruto verde."
[287] "Árvore com cerca de 4 a 5 m de alt. Caule acinzentado, com muitas lenticelas. Folhas coria
[288] "árvore com cerca de 5 m. de altura; coletado na mat atlântico, cerrado, em área de transi
[289] "Árvore com cerca de 5 metros de altura; tronco marrom escuro, lenticulado. Ramos e folhas
[290] "Árvore com cerca de 5 metros de altura. Folhas cartáceas, fortemente discolores. Botoes e
[291] "Árvore com cerca de 5 metros; caule com lenticelas; frutos maduros, com a cúpula envolven
[292] "Árvore com cerca de 6 m. Flores com pétalas e estames creme. UTM 0671199S8739231; Solo ar
[293] "Arvore com cerca de 6 metros de altura, presença de acúleos, frutos secos"
[294] "árvore com cerca de 8 m. de altura; coletado no interior da mata atlântica; botoes ferrug
[295] "Árvore com cerca de 9 metros de altura; caule acinzentado, lenticulado; Folhas apiculada
[296] "Árvore com flores amarelas, botoes amarelo-
esverdeados. árvore com flores amarelas; botoes amarelo-esverdeado."
[297] "Árvore com folhas e botoes florais. árvore com folhas e botões florais.; Árvore com folhas
[298] "Árvore com frutos maduros.; DESC: Árvore com frutos maduros."
[299] "árvore com mais ou menos 15m de alt."
[300] "ÁRVORE DE 10 M DE ALT., E 40 CM DE DIÂM, RESQUICIO DA MATA PRIMITIVA"
[301] "Árvore de 10 m de altura x 20 cm de diâmetro; botoes florais. Mata de terra firme; solo ar
arenoso. árvore de 10 m de altura x 20 cm de diâmetro; botões florais. mata de terra firme; solo arg
arenoso."
[302] "Árvore de 10 m de altura x 50 cm de diâmetro; frutos imaturos verdes. árvore de 10 m de alt
[303] "Árvore de 10 m., flor amarela.; Árvore de 10 m."
[304] "Árvore de 10-15 m. de altura, flores pequenas de cor verde-
escuro a marrom. Solo argiloso. árvore de 10-
15 m. de altura, flores pequenas de cor verde-
escuro a marrom.; solo areno-argiloso, vegetaçao primaria.; Árvore de 10-

15 m. de altura, flores pequenas de cor verde-escuro a marrom. Solo argiloso."

[305] "Árvore de 10-15 metros. Material frutífero."

[306] "Árvore de 10,5m de altura e cap. 49cm. Folhas pequenas, creme. Módulo 8, indivíduo 428."

[307] "Árvore de 10m alt., 10 cm, inflorescencia em botoes"

[308] "Árvore de 10m de alt., com frutos."

[309] "Árvore de 10m de altura 20cm diametro, flor amarela, cálice creme."

[310] "Árvore de 10m de altura x 12cm de diâmetro, frutos jovens. árvore de 10m de altura x 12cm

[311] "Árvore de 10m de altura, DAP 9.3cm. Infrutescência verde-escura, frutos com cálice vermelho e frutos verdes. árvore de 10m de altura, dap 9.3cm. infrutescência escura, frutos com cálice vermelho e frutos verdes. árvore de 10m de altura, dap 9.3cm. infrutescência escura, frutos com cálice vermelho e frutos verdes. ; de"

[312] "Árvore de 10m de altura, esgalhada desda a base, frutos imaturos. Ocasional na mata de te

[313] "Árvore de 10m de altura, flor em botão creme."

[314] "Árvore de 10m de altura; inflorescência em panícula, com flores pequenas de cor amarelo cl

[315] "Árvore de 10m, com ramos flexuosos e flores brancacentas, frutos verdes com cupula verde"

[316] "Árvore de 12 - 15 m, frutos ainda jovens."

[317] "Árvore de 12 e 15cm diâm; frutos verdes. Col. de madeira. árvore de 12m e 15cm diâm., frut

[318] "Árvore de 12 m, flores claras"

[319] "Árvore de 12 m. de altra e 1 m. de circunferencia. (Amost. de madeira). árvore de 12m de a

[320] "ÁRVORE DE 12-15M DE ALTURA, FLORES EM BOTOES ESVERDEADOS. árvore de 12-15m de altura, flores em botões esverdeados."

[321] "Árvore de 12,30m de altura e 53cm de CAP. Módulo 2, indivíduo 57."

[322] "Árvore de 12m de altura com frutos jovens esverdeados. Mata de terra firme e solo argiloso"

[323] "Árvore de 12m de altura x 18cm de diâmetro, frutos jovens. Ocasional no capoeirao de terra"

[324] "Árvore de 12m e 25cm diâm., entre blocos de manganes, infl. em botoes."

[325] "Árvore de 12m, flor branco-sujo"

[326] "Árvore de 13,30m de altura, com 44 cm de cap., frutos quando maduros de cor marrom escuro."

[327] "Árvore de 13m de altura. Frutos imaturos. Mata de terra firme."

[328] "Árvore de 14m de altura com botoes florais esverdeado"

[329] "Árvore de 14m de altura, com 165cm de cap., módulo 2, indivíduo 83."

[330] "Árvore de 14m de altura, DAP 5.4cm. Botoes florais verdeclaro. árvore de 14m de altura, dap 5.4cm. botões florais verdeclaro. árvore de 14m de altura, dap 5.4cm. botoes florais verdeclaro. ; de"

[331] "Árvore de 15 m., flor creme - sujo."

[332] "Árvore de 15 m., flores creme em inflorescencia."

[333] "Árvore de 15,70m de altura e 65cm de CAP. Módulo 1, indivíduo 33."

[334] "Árvore de 15m de altura, DAP 15.5cm. Inflorescencia axilar em panículas, pedúnculo verdeclaro, flores amarelo-claro, botoes florais amarelo-claro. árvore de 15m de altura, dap 15.5cm. inflorescência axilar em panículas, pedúnculo verdeclaro, flores amarelo-claro, botões florais amarelo-claro. árvore de 15m de altura, dap 15.5cm. inflorescência axilar em panículas, pedúnculo verdeclaro, flores amarelo-claro, botoes florais amarelo-claro. ; de"

[335] "Árvore de 15m de altura, flores e estames amarelos, botoes florais amarelo-claros. Mata de terra firme alta e solo arenoso. árvore de 15m de altura, flores e estames amarelos claros. mata de terra firme alta e solo arenoso.; Árvore de 15m de altura, flores e estames amarelos claros. Mata de terra firme alta e solo arenoso."

[336] "Árvore de 16m de altura e 65cm de CAP. Folhas com a face abaxial tomentosa. Material estéril."

[337] "Árvore de 18 m. x 25 cm. de diâm., frutos imaturos verdes."

[338] "Árvore de 20 m de altura x 15 cm de diâmetro; inflorescencia jovem esverdeada. Mata de terra firme."

[339] "Árvore de 20m de altura x 25cm de diâmetro, madeira dura, pesada, creme, cheiro agradável."

[340] "Árvore de 20m de altura x 30cm de diâmetro, flores amareladas. Madeira com casca creme, cor amarela."

[341] "Arvore de 20m. Flores em botao."

[342] "Árvore de 21m de altura x 32cm de diâmetro de fuste, flores velhas colhidas no chao, ao pé da árvore."

[343] "Árvore de 25m de altura x 25cm de diâmetro, folhas verde-claras por baixo, madeira amareladas, pouco pesada, cúpula do fruto avermelhada. Ocasional na mata de terra firme."

[344] "Árvore de 3 m de altura; flores alvo-amareladas. Terra firme. arenoso, capoeira fechada."

[345] "ARVORE DE 3-4 MTR., FL. ALVA."

[346] "Árvore de 3-4m, frutos verdes. Receptaculos vermelhos."

[347] "Árvore de 3,0m de altura, caule cinza escuro, fissurado, folhas verdes, frutos verdes (já maduros)."

[348] "Árvore de 31,0 cm de circunferencia. árvore de 31,0 cm de circunferência.; Árvore de 31,0 cm de circunferência."

[349] "Árvore de 37,0 cm de circunferencia. árvore de 37,0 cm de circunferência.; Árvore de 37,0 cm de circunferência."

[350] "Árvore de 3m de alt., frutos imaturos verde"

[351] "Árvore de 4 m de altura, fina; flor branca. Capoeira na beira do rio."

[352] "Árvore de 4 m de altura; flores esverdeadas; botoes florais. Bosque, solo argilo-arenoso. árvore de 4 m de altura; flores esverdeadas; botões florais. bosque, solo argilo-arenoso."

[353] "Árvore de 4 m de altura."

[354] "Árvore de 4 metros árvore de 4m de alt.; árvore de 4m de alt."

[355] "Arvore de 4-6 m."

[356] "Árvore de 4,5 m de altura; frutos imaturos verdes. Capoeira, solo argiloso."

```
## [357] "Árvore de 4m de altura, frutos imaturos verdes. Capoeira e solo argiloso."  
## [358] "Árvore de 4m de altura. Frutos imaturos verdes. ; DE"  
## [359] "Árvore de 4m de altura. Inflorescencia com flores amareladas. Terreno firme e argiloso. O  
## [360] "Árvore de 5 m de altura x 10 cm de diâmetro. Flor creme. árvore de 5 m de altura x 10 cm de  
## [361] "Árvore de 5 m de altura x 6 cm de diâmetro; botoes florais amarelados; frutos imaturos ve  
esbranquiçados, passando a róseos e evermelhos quando amadurecem. Capoeira alta, solo argilo-  
silicoso. árvore de 5 m de altura x 6 cm de diâmetro; botões florais amarelados; frutos imaturos ve  
esbranquiçados, passando a róseos e evermelhos quando amadurecem. capoeira alta, solo argilo-  
silicoso."  
## [362] "Árvore de 5 m de altura; flores amarelas. Capoeira; solo argilo-  
arenoso."  
## [363] "Árvore de 5 m de altura; frutos verdes. Terreno; firme, argiloso, capoeira fechada. X = 1  
## [364] "Árvore de 5 m de altura; frutos vermelhos, cálice marron. Terra firme; solo argiloso; mat  
## [365] "Árvore de 5 m. de altura, com frutos."  
## [366] "Árvore de 5-7m, flor branca.; Cerrado"  
## [367] "Árvore de 5-8 m, botoes esverdeados."  
## [368] "Arvore de 5.0 m. alt. Folhas verdes concolores. Flores com botoes cremes"  
## [369] "Árvore de 5m de altura e botoes florais amarelados. Capoeira alta, solo argilo-  
silicoso. árvore de 5m de altura e botões florais amarelados. capoeira alta, solo argilo-  
silicoso."  
## [370] "Árvore de 5m de altura e flores amarelo-  
pálido, folhas alternas e coriáceas. Madeira castanho-  
claro, casca grossa, persistente, serve para vigas. Terra firme, argilosa. Mata virgem."  
## [371] "Árvore de 5m de altura x 30cm de circunferencia, frutos imaturos verdes. Mata de terra fi  
## [372] "Árvore de 5m de altura, flores cremes. Capoeira em frente da Reserva Biológica de Campina  
arenoso."  
## [373] "Árvore de 5m de altura. Flores amarelas. Terreno arenoso, capoeira fechada."  
## [374] "Árvore de 5m."  
## [375] "àrvore de 5m. rrvore de 5m.;àrvore de 5m."  
## [376] "Árvore de 6 m, flores amarelas."  
## [377] "Árvore de 6 m. por 30 cm. de circunferencia, frutos verdes, imaturos."  
## [378] "Árvore de 6m de altura. Terra firme e solo argiloso."  
## [379] "Arvore de 6m, botoes amarelosa, folhas discolor arvore de 6m, botões amarelosa, folhas d  
## [380] "Árvore de 7 m de altura; flores laranjas. Capoeira, terra firme, solo arenoso. árvore de  
## [381] "Árvore de 7 metros."  
## [382] "Árvore de 7 mts, frutos um pouco oval, caules verdes, amendoas marrom. árvore de 7m, frut  
## [383] "Árvore de 7m de altura, DAP 2.3cm. Inflorescencia e infrutescencia com frutos jovens. árv
```

[384] "Árvore de 8 m. Frutos imaturos verdes com cúpula vermelha.; Floresta Ombrófila Aberta Alu
[385] "Árvore de 8 metros, frutos novos."
[386] "Arvore de 8m"
[387] "Árvore de 8m de altura com inflorescencia esverdeada"
[388] "ÁRVORE DE 8M DE ALTURA."
[389] "Árvore de 8m x 15cm de circunferencia. Fruto imaturo, verde. árvore de 8m x 15cm de circun
[390] "Árvore de 9 m de altura. Mata de terra firme, solo argiloso."
[391] "Árvore de aprox. 6 m alt. Copa plana, muitos ramos jovens. Flores alvas. Inflorescencia e
[392] "Árvore de apróx. 8 m de alt. Copa plana."
[393] "Árvore de aproximadamente 6 metros de altura. Copa ampla. ; DE"
[394] "Árvore de aproximadamente 6,5m folhas simples, subcrassas. Exudado incolor pegajoso e ch
pedregoso. habitat:terrestre no interior da mata. exudado incolor pegajoso e cheiro amargoso. par
6,5 m. folhas simples, subcrassas. material estéril.; floresta estacional (mata d; ha"
[395] "Árvore de aproximadamente 6m de altura. Botoes amarelados."
[396] "Árvore de aproximadamente 8 m de altura. copa ampla ; DE"
[397] "árvore de até 5m de alt., flor alva"
[398] "Árvore de até 60 m. fl. alvacenta."
[399] "ÁRVORE DE CA. 8 M DE ALT., FOLHA SIMPLES, INTEIRA, SOLTANDO-
SE COM FACILIDADE"
[400] "Árvore de frequencia relativa baixa com flores alvas e frutos ausentes."
[401] "Árvore de fuste linheiro, ca. 8 m de altura e 10 cm de diâmetro. Copa ampla e esgalhada. F
esverdeado. Madeireira. árvore de fuste linheiro, cerca de 8 m alt. e 10 cm de diâmetro. copa ampla
esverdeado. madeireira."
[402] "Árvore de grande porte, flores alvo-
esbranquiçadas, sem perfume, pequenas."
[403] "Árvore de grande porte, flores pequenas e amarelo com pouco perfume. árvore de grande por
[404] "Árvore de grande porte, sem látex ou resina. Flor alva ou marrom, sem perfume. árvore de g
[405] "Árvore de grande porte. Flores amarelas pouco perfumadas, pequenas. árvore de grande por
[406] "Árvore de grande porte. Flores mais ou menos roxas."
[407] "Árvore de grande porte. Flores pequenas de cor alva-
esverdeada."
[408] "Árvore de mais ou menos 8 metros de altura."
[409] "Árvore de pequeno porte. Inflorescencia terminal com pequenas flores esverdeadas.; Solo
[410] "Árvore de porte regular."
[411] "Árvore em moita."
[412] "Árvore flor amarela"
[413] "Árvore florida (flores alvas) com 10 m de altura. Terra firme."

```
## [414] "Árvore frondosa, isolada no pasto. Flores creme."
```

```
## [415] "Árvore grande . Ref no O Genero Ocotea Aulb, no Nordeste do Brasil, Lauraceas). de Ida de
```

```
## [416] "Árvore mais ou menos 10m alt. Flores amarelas pouco perfumadas."
```

```
## [417] "Árvore media, fl. esbranquiçada."
```

```
## [418] "Árvore mediana de 10 a 15 m de altura, com flores esverdeadas aromáticas, na mata."
```

```
## [419] "Árvore mediana."
```

```
## [420] "Árvore na borda da mata, ca. de 5 m alt. Frutos verdes, cupula verde avermelhada.; Mata AT
```

```
## [421] "Árvore Ns 126.; Floresta de terra-firme. ; HA"
```

```
## [422] "ÁRVORE PEQ. FL ALVACENTA"
```

```
## [423] "Árvore pequena de 4 m de altura; flores esverdeadas; frutos novos com o cálice marron."
```

```
## [424] "Árvore pequena de 4 m de altura; flores esverdeadas; frutos novos com o cálice marron. Te
```

```
## [425] "Árvore pequena, fl verde - pardacenta."
```

```
## [426] "Árvore pequena. (Arb.) Flôres amarelo pardo."
```

```
## [427] "Árvore pequena. Flor amarelo pardo. árvore pequena. flor amarelo-pardo."
```

```
## [428] "Árvore planta com ca. de 3m de alt. Botoes florais esverdeados.HABITAT:Mata Atlântica em
```

```
## [429] "Árvore regular; flores creme; folhas com cheiro de canela sassafraz. Na mata."
```

```
## [430] "àrvore rrvore;àrvore"
```

```
## [431] "Árvore tombada, folhas lisas, frutos verdes com cúpula marrom levemente avermelhada"
```

```
## [432] "Árvore, 1,5 m.. Flor amarela. arvore. 1.5. m. flor amarela.; borda da mata ; ha doação do
```

```
## [433] "Árvore, 10 m, frutos imaturos verde claros. coleta: árvore 10 m, frutos verdes.; geografi
```

```
## [434] "Árvore, 10 metros; frutos imaturos verdes. Acima da pousada águas verdes, beira da estrada"
```

```
## [435] "Árvore, 10m, fruto imaturo verde; Mata subperenifolia"
```

```
## [436] "arvore, 14 m, 14 cm DBH. fl. jaunes en forme de coupe r 3+3 tép. jaune recouvert de duvet b
```

```
## [437] "Árvore, 6m. Flores cremes árvore, 6m. flores cremes.; mata atlântica. ; ha árvore, 6m. fl
```

```
## [438] "Árvore, 7-10m, Comum na encosta e níveis superiores. Inflorescencia abundante, terminal, amareladas."
```

```
## [439] "Árvore, 8 m."
```

```
## [440] "Árvore, abundante, flor branca árvore, flor branca, abundante."
```

```
## [441] "Árvore, ca de 10m de altura.Frutos verdes.; Mata Atlântica árvore, ca. de 10m de altura."
```

```
## [442] "Árvore, ca. de 4 m alt. Frutos verdes, cúpula vermelha.; Mata Atlântica. ; HA"
```

```
## [443] "Árvore, cerca de 5,0 m de altura; botoers florais verde-amarelos. Espécie comum no local. árvore, cerca de 5,0 m de altura; botões florais verde-amarelos. espécie comum no local."
```

```
## [444] "Árvore, construcao e carpintaria."
```

```
## [445] "Árvore, construcao e carpintaria. folha larga.; Árvore, construcao e carpintaria"
```

```
## [446] "Árvore, em floração, ca. de 5 m de altura, botoes florais cremen, frutos ausentes. árvore"
```

```

## [447] "Árvore, flor esbranquiçada."
## [448] "Árvore, folhas semicrassas, levemente discolor com face adaxial verde brilhante, frutos
## [449] "Árvore; 10 m; tronco com lenticelas redondas e odor forte.; Terra firme, floresta secundária
## [450] "Árvore; 11 m .; Terra firme, floresta secundária sobre solo antropogenico (Terra Preta de Índio); florais; flores creme; folhas face adaxial verde-brilhante e face abaxial verde-esbranquiçado.; Borda da capoeira. ; HA árvore; 4,0 m; botoes florais; florais; flores creme; folhas face adaxial verde-brilhante e face abaxial verde-esbranquiçado.; borda da capoeira.; habitat: borda da capoeira.; descritivas"
## [451] "Árvore; 4,0 m; botoes florais; flores creme; folhas face adaxial verde-brilhante e face abaxial verde-esbranquiçado.; Borda da capoeira. ; HA árvore; 4,0 m; botoes florais; florais; flores creme; folhas face adaxial verde-brilhante e face abaxial verde-esbranquiçado.; borda da capoeira.; habitat: borda da capoeira.; descritivas"
## [452] "Árvore; 4,0 m; flores e botoes creme. ; DE"
## [453] "Árvore; 8 m; pecíolo com cera branca.; Terra firme, floresta secundária sobre solo antropogenico (Terra Preta de Índio); florais; flores creme; folhas face adaxial verde-brilhante e face abaxial verde-esbranquiçado.; borda da capoeira.; habitat: borda da capoeira.; descritivas"
## [454] "Árvore; 8 m.; Terra firme, floresta secundária sobre solo antropogenico (Terra Preta de Índio); florais; flores creme; folhas face adaxial verde-brilhante e face abaxial verde-esbranquiçado.; borda da capoeira.; habitat: borda da capoeira.; descritivas"
## [455] "Árvore; flores brancas. Capoeiras e matas..Habitat: Terrestre"
## [456] "Árvore: 5,0m. Folhas coriáceas, discolores, face abaxial pubescente. Frutos imaturos verdes"
## [457] "Árvore."
## [458] "Árvore."
## [459] "Árvore."
## [460] "Árvore."
## [461] "Árvore."
## [462] "Árvore."
## [463] "árvore. 10. m. folhas alternas, flor amarela.; interior da mata ; HA"
## [464] "árvore. 12. m. folhas alternas.; Mata ; HA"
## [465] "árvore. 8. m. presença de frutos verdes e maduros, associação com formigas e ninhas de hastes"
## [466] "árvore. 9. m. presença de botoes florais e flores, floração intensa.; Mata (borda) ; HA"
## [467] "Árvore. Amostra 283 - Parcelsa 6 - "
Sao Joao. árvore. amostra 619 - parcela 4 - quizanga."
## [468] "Árvore. Botoes e flores acastanhados. Solo argiloso."
## [469] "Árvore. Botoes florais verdes-claro. árvore. botões florais verdes-claro.; Árvore. Botoes florais verdes-claro."
## [470] "Árvore. ca. de 8 m de altura. Inflorescência com flores amarelas,botoes florais amarelados"
## [471] "Árvore. Flor creme; .; HABITAT: .; DESC: Árvore. Flor creme;Árvore. Flor creme."
## [472] "Árvore. Flores com sépalas persistente verde-clara. Fruto imaturo verde escuro."
## [473] "Árvore. Folhas coriáceas, discolores, face abaxial pubescente.; Floresta Estacional Semidecidual"
## [474] "Árvore. Folhas simples, alternas, face abaxial discolor, simples, crassa.&nbsp;; Floresta Estacional Semidecidual"
## [475] "Árvore. Frequencia alta. Flores creme."
## [476] "Árvore. Mata de terra firme. árvore.; mata de terra firme."
## [477] "Árvore. Tronco circular de base dilatada. Ritidoma ocre e sob o ocre bordô, rígido, microestruturado."

```

escamoso, verrugoso. Lenticelas salientes, circulares, espocadas, dispersas, de ca. 5 mm, homogeneamente escamas. Casca morta fina. Casca viva marrom-claro e bordô mais externamente. Oxidação lenta para marrom-claro, de 4 mm, odor forte. Alburno amarelado. Ráquis da inflorescência amarelo-esverdeado; botões florais verde-amarelados; flores com pétalas brancas, estames amarelos. árvore escamoso, verrugoso. lenticelas salientes, circulares, espocadas, dispersas, de ca. 5 mm, homogeneamente escamas. casca morta fina. casca viva marrom-claro e bordô mais externamente. oxidação lenta para marrom-claro, de 4 mm, odor forte. alburno amarelado. ráquis da inflorescência amarelo-esverdeado; botões florais verde-amarelados; flores com pétalas brancas, estames amarelos. ráquis esverdeado, botões florais verde-amarelados, flores com pétalas brancas, estames amarelos solo arredondados. escamoso, verrugoso. lenticelas salientes, circulares, espocadas, dispersas, de +/- 5 mm, homogeneamente distribuída solo argiloso, dossel, 18 cm dap. tronco circular de base dilatada. escamoso, verrugoso. lenticelas salientes, circulares, espocadas, dispersas, de +/- 5 mm, homogeneamente distribuídas. alteração por estrias. desprendimento por micro-escamas. casca morta fina. casca viva marrom-claro e bordô mais externamente. oxidação lenta para marrom-claro, de 4 mm, odor forte. alburno amarelado. ráquis da inflorescência amarelo-esverdeado; botões florais verde-amarelados; flores com pétalas brancas, estames amarelos."

[478] "Árvore.; Mata úmida"

[479] "Árvore.; Mata úmida"

[480] "Árvore.; Mata úmida árvore.; mata úmida."

[481] "Árvore.; Orla da mata"

[482] "Árvore.Habitat: Terrestre"

[483] "Árvore&nf;Nome vulgar: Louro"

[484] "Árvores com cercas de 12m altura na borda da mata, porém, comum no interior desta. Folhas lanceoladas, velutinas, um tanto ferrugíneas, discolores. Inflorescências terminais e axilares. Flores amarelas."

[485] "Arvoreta , 4m alt., flores amareladas. Coleta de folhas para análises"

[486] "Arvoreta ± 3.0m alt. Caule acinzentado. Frutos imaturos esverdeados. Freqüente. habitat: Floresta Atlântica"

[487] "Arvoreta 3-4m alt., frutos imaturos verdes.; Floresta Atlântica; HABITAT: Floresta Atlântica 4m alt., frutos imaturos verdes."

[488] "Arvoreta 4m, flor creme. Encosta rochosa do morro"

[489] "Arvoreta 5m alt., cálice vermelho; Floresta Atlântica; HABITAT: Floresta Atlântica; DESCRIÇÃO: folhas lanceoladas"

[490] "Arvoreta 9m. Dap 20cm. Fuste 3m. Flores esbranquiçadas. arvoreto 9 m. dap 20 cm. fuste 3 m."

[491] "Arvoreta ca. 3 m de altura. Botoes cremes.HABITAT:Mata Atlântica em encosta.;"

[492] "Arvoreta ca. 3,0 m, botoes florais esverdeados.; Cerrado (típico). Substrato arenopedregoso. coleta: xls: spsfantigo; geografia: cerrado&nf;;"

[493] "Arvoreta ca. 4,0 m, flores amareladas.; Mata ciliar. Afloramento rochoso."

[494] "Arvoreta ca. 4,5 m, botoes florais esverdeados.; Mata ciliar.&nf;Substrato arenopedregoso."

[495] "Arvoreta ca. 4.0 m, frutos imaturos.; Cerrado Sujo. Substrato arenoso. coleta: cerrado s

[496] "Arvoreta ca. 4m alt., flores cremes"

[497] "Arvoreta ca. 7m alt. Frutos verdes."

[498] "Arvoreta ca. de 3,5m de altura. Flores amarelas.HABITAT:Caatinga; arvoreta, ca. 3,5m. fo
amarelado. arvoreta, ca. de 3,5m de altura, folhas discolares aromáticas, botoes amarelados, peria
amarelado. arvoreta, ca. de 3,5m de altura, folhas discolares aromáticas, botoes amarelados, peria
amarelo. ambiente: caatinga arvoreta, ca. de3,5m de altura, folhas discolores aromáticas, botoes a
amarelado.; desc: arvoreta, ca. de3,5m de altura, folhas discolores aromáticas, botoes amarelos, p
amarelado."

[499] "Arvoreta com +/- 3,5 de alt. Ramos acinzentado.Folhas cartáceas. Inflorescencia ferrugí
frutos esverdeados .Planta aromática .Frequente."

[500] "Arvoreta com 1,80m de altura. Indivíduo n 44. Folhas discolores. Frequente."

[501] "Arvoreta com 2 m; (inflorescencia) frutescencia de cor creme amarelada retirada com uma p

[502] "Arvoreta de 2m de altura, flores cremes e aromáticas em terra firme e solo arenoso."

[503] "Arvoreta de 3m de altura com frutos imaturos verdes. Levantamento da campina e flora apí
5)."

[504] "Arvoreta de 4 m de altura; flores amarelas. Capoeira de terra firme de solo arenoso."

[505] "Arvoreta de 4 m de altura; flores cremes. Capoeira; solo arenoso."

[506] "Arvoreta de 4m de altura com botoes florais creme esverdeados e flores creme.; Mata de be

[507] "Arvoreta de 4m de altura, frutos imaturos. Capoeira de solo argiloso."

[508] "Arvoreta de 4m de altura. Frutos verdes."

[509] "Arvoreta de 4m de altura. Frutos verdes. Terra firme e solo arenoso."

[510] "Arvoreta de 5 m de altura, flores cremes, botoes florais esverdeados."

[511] "Arvoreta de 5 m de altura; frutos imaturos, verdes. Mata de terra firme, solo argilo-
arenoso."

[512] "Arvoreta de 5 m de altura; frutos verdes; folhas alternas."

[513] "Arvoreta de 5 m x 10 cm de diamâmetro; flores amarelo-
esverdeado. Madeira creme, pouco pesada; casca esverdeada. Solo arenoso. X = 5426 arvoreta de 5 m x
esverdeado. madeira creme, pouco pesada; casca esverdeada. solo arenoso.&nf;x = 5426"

[514] "Arvoreta de 5m de altura com botoes florais creme amarelados.; Mata de beira de rio."

[515] "Arvoreta de 5m de altura, inflorescencia axilares brancas sem perfume. Folhas alternas.
clara perfumada. Terra firme, solo argilo-
arenoso. Capoeira alta de terra firme e mata derrubada. arvoreta de 5m de altura, inflorescência a

clara perfumada. terra firme, solo argilo-arenoso. capoeira alta de terra firme e mata derrubada."

[516] "Arvoreta de 6 m de altura; botoes florais, amarelo-esverdeado. arvoreta de 6 m de altura; botões florais, amarelo-esverdeado.;Arvoreta de 6 m de altura; botoes florais, amarelo-esverdeado."

[517] "Arvoreta de 6 m de altura; flores esverdeadas; botoes floral esverdeados. Capoeira; solo rg. 95429). arvoreta de 6m de altura; flores esverdeadas e botoes florais esverdeados. capoeira de

[518] "Arvoreta de 6 m de altura; inflorescencia jovem; frutos imaturos, verde claros; folhas v... argiloso. arvoreta de 6 m de altura; inflorescência jovem; frutos imaturos, verde claros; folhas v... argiloso."

[519] "Arvoreta de 6m de altura no igapó. Frutos verdes."

[520] "Arvoreta de 6m de altura, frutos imaturos verdes. Mata alta de terra firme e solo argilo-arenoso."

[521] "Arvoreta de 6m de altura, inflorescência jovem amarela, frutos imaturos jovens verdes. M...

[522] "Arvoreta de 6m de altura; com botoes jovens amarelados. Adjacencia."

[523] "Arvoreta de 6m de altura; flores cremes. Solo argiloso com afloramento rochoso."

[524] "Arvoreta de 6m de altura; folhas alternas grandes, inflorescencia terminal, botoes brancos esverdeados. Capoeira alta de terra firme e solo argilo-arenoso. arvoreta de 6m de altura; folhas alternas grandes, inflorescência terminal, botões brancos esverdeados. capoeira alta de terra firme e solo argilo-arenoso."

[525] "Arvoreta de 6m de altura. Frutos verdes. Mata de terra firme e solo arenoso."

[526] "Arvoreta de 7 m de altura; flores amareladas. Terra firme."

[527] "Arvoreta de 7 m de altura; folhas alternas; inflorescencia em panícula, corola gamopétala...

[528] "Arvoreta de 7 m de altura; folhas alternas; inflorescência em panícula, corola gamopétala...

[529] "Arvoreta de 7 m de altura; frutos imaturos verdes. Mata de terra firme, solo argiloso."

[530] "Arvoreta de 7 m de altura; frutos imaturos verdes. Mata de terra firme, solo argiloso."

[531] "Arvoreta de 7m de altura; frutos imaturos verdes."

[532] "Arvoreta de 8 m de altura com flores amareladas."

[533] "Arvoreta de 8 m de altura; flores amareladas; frutos imaturos verdes. Capoeira de terra f...

[534] "Arvoreta de 8 m de altura; flores em botoes cremes; frutos imaturos verdes. Capoeira de t... arenoso."

[535] "Arvoreta de 8m de altura. Botoes florais esverdeados. Mata de terra firme e solo argilosos...

[536] "arvoreta de 8m de alt, flores amareladas, botoes esverdeados arvoreta de 8m de alt, flores ..."

[537] "Arvoreta de mais ou menos 4m de altura, flores cremes, inflorescencia e botoes florais cr...

[538] "Arvoreta de mais ou menos 4m de altura, flores cremes, inflorescência e botões florais cr...

[539] "Arvoreta em touceira densa; flores cremes. Frequente na capoeira de solo arenoso, perto o...

[540] "Arvoreta lenhosa com 3,5m. Individuo n°2467. Folhas verdes descoloradas, material vegetativo"
[541] "Arvoreta pouco ramificada 3m. Flores alvas."
[542] "Arvoreta, ca. 4m, flor creme, inflorescência e botões florais cremes, aromáticas."
[543] "Arvoreta, ca. 5m. Folhas verde descoloradas, brilhosas. Inflorescências verde, flores amarelas"
[544] "Arvoreta, em floração, ca. 2,5 m. de altura, flores cremes, frutos ausentes"
[545] "Arvoreta, flor branca aromática"
[546] "Arvoreta; 3,0 m; frutos imaturos; base do revestimento em forma de cálice de cor marrom-escuro; na proximidade do ápice do ramo presença de fungos (cochonilhas). ; DE"
[547] "Arvoreta; 3,0 m; pequenos frutos imaturos verde-cremes. PIBIC Jr. 2009/2010.; Vertente. ; HA"
[548] "Arvoreta; 5 m; frutos com cálice amarronzados e frutos imaturos verdes; tronco cilíndrico"
[549] "Arvoreta; 7,0 m; frutos imaturos verdes com cúpula verde amarronzada. ; DE"
[550] "Arvoreta; flor branca."
[551] "Arvoreta; frutos imaturos verdes e maduros roxos; material conservado com álcool.; Sub-bosque. ; HA"
[552] "Arvoreta; Tabuleiro litorâneo"
[553] "Arvoreta."
[554] "Arvoreta."
[555] "arvoreta. 2,5. m. botões verdes, cálice castanho-claro, flores com pétalas brancas.; Mata (borda) ; HA"
[556] "arvoreta. 2,5. m. flores com pétalas creme.; borda da mata ; HA"
[557] "arvoreta. 6. m. botões florais e flores com pétalas creme.; Mata (borda) ; HA"
[558] "Arvoreto 5 m. Frutos verdes."
[559] "ARVOREZINHA COM RAMOS ESCANDENTES . FRUTOS COM RECEPTÁULOS AVERMELHADOS , IMATUROS VERDES"
[560] "Arvorezinha de 4-5m. Flores cremes. amostra química"
[561] "Arvorezinha de 5m alt., 5 cm diam., flor branca"
[562] "Árvore 14 m. Frutos verdes. Interior da mata."
[563] "Beira de estrada, antropizada. Contato Savana Floresta/ Floresta Estacional/ Floresta Ombronal"
[564] "Bl. creme. Schwarzwasser."
[565] "Botões florais creme. Área de influência direta da UHE Jirau."
[566] "Botões florais puberulentos. árvore de 5-6m. Solo arenoso. Floresta semidecidual botões florais puberulentos. Árvore de 5-6m. solo arenoso. floresta semidecidual"
[567] "Botões florais verdes."
[568] "Broto de árvore; flores cremes; frutos pretos. Mata de terra firme, argilosa."
[569] "ca. 12m de altura e 40cm de CAP."
[570] "Caatinga arbórea."

```
## [571] "Cálice verde."
## [572] "Cerca de 10 m de altura e 137 cm de CAP."
## [573] "Cerrado"
## [574] "Cerrado / floresta ombrófila."
## [575] "Cerrado sensu strictu. Solo arenoso."
## [576] "Cheiro forte na casca e folhas. Verdes discolores. Botoes amarelos. Ausentes"
## [577] "Coleta de material estérial do projeto de doutorado do professor Aldenir.; Arvoreta; 6 m"
## [578] "COLETA: Árvore 11m; corola amarelo escuro; estames passados escurecidos.;"
##      [579] "COLETA: Cerradao&nf;Liana semi-
ciófila; cálice avinosado; frutos imaturos verdes; GEOGRAFIA: Área da Sede Santa Luzia; solo argiloso.&nf;Relevo plano.; folhas cor verde, cartáceas, presença de galhas escuras na superfície a
ciófita; folhas cor verde, cartáceas: presença de galhas escuras na superfície abaxial da folha; c
## [580] "COLETA: Cerrado arenoso &nf;Arbusto 80cm, flores bege.; GEOGRAFIA: A 13km do povoado Ch
## [581] "COLETA: Cerrado com solo arenoso.&nf;Semi arbusto formando moitas; flores cor amarelo cl
##      [582] "COLETA: Cerrado stricto sensu&nf;Ex VIC-
24000&nf;Árvore 12m; flores de corola cor creme.;"
## [583] "COLETA: Cerrado&nf;Solo com textura arenosa/argilosa; relevo plano.&nf;Subarbusto helic
## [584] "COLETA: Flores brancas com glândulas vermelhas.&nf;Árvore 11m.; GEOGRAFIA: (R.23, Praç
## [585] "COLETA: Flores pequenas, esbranquiçadas a creme.&nf;Árvore 10m; GEOGRAFIA: (R.23, praç
## [586] "COLETA: Fragmento Florestal, interior da mata.&nf;Árvore 14m altura; frutos; GEOGRAFIA:
## [587] "COLETA: Fragmento Florestal, interior da mata.&nf;Árvore 20m altura; frutos; GEOGRAFIA:
## [588] "COLETA: Mata Atlântica.&nf;Árvore 8m;"
## [589] "COLETA: Mata Atlântica.&nf;Árvore, parte inferior do fruto vermelha.; GEOGRAFIA: Arvore
## [590] "COLETA: Mata de Altitude&nf;Árvore 12m; frutos imaturos verdes.; GEOGRAFIA: Beira da mat
## [591] "COLETA: Mata de Galeria nao inundável&nf;Árvore 12m altura; frutos imaturos verdes.;"
## [592] "COLETA: Mata estacional antropizada.&nf;Arbusto 1m altura; flores amarelo esverdeadas.;"
## [593] "COLETA: Mata secundária.&nf;Árvore 3m; frutos verde escuros, cúpula avermelhada.; GEOGR
##      [594] "COLETA: Vegetação arbustivo-
arbórea aberta.&nf;Árvore 5m, corola branca, frutos imaturos verdes.; GEOGRAFIA: A ca. De 28km oes
430.&nf;Beira de estrada.; vegetação arbustivo-
arbórea aberta. beira de estrada."
## [595] "COLETA: XLS: SPSFAntigo;"
## [596] "COLETA: XLS: SPSFAntigo; GEOGRAFIA: cerrado&nf;;"
## [597] "COLETA: XLS: SPSFAntigo&nf;Árvore; GEOGRAFIA: trilha do vinhatico;"
## [598] "COLETA: XLS: SPSFAntigo&nf;Árvore. Exemplar ns 19374; GEOGRAFIA: Mata;"
## [599] "COLETA: XLS: SPSFAntigo&nf;Floresta secundária&nf;Árvore flores creme;"
## [600] "Cúpulas imaturas verde-marrons, ramos da inflorescencia verdes. Cheiro típico de Laurace
```

```
## [601] "DESC:Arvoreta      2,2m;      flor      creme-
esverdeada; cupulos persistentes negros; botoes florais creme-
esverdeados.;"
## [602] "DESC:Subarbusto 40cm; fruto imaturo verde. Rara no local.;"
## [603] "Erva ereta. Folhas papiráceas. Flores com corola branca."
## [604] "Espécie ameaçada de extinção (Em perigo). material sem dados da planta. solo pobre, areno-
## [605] "Estrada Manaus - Carcaraí Km 39, Reserva Experimental de Silvicultura Tropical. Terra firme
## [606] "Estrada Manaus - Carcaraí Km 44. Estação Experimental de Silvicultura Tropical. Terra firme
carcaraí km 44. estação experimental de silvicultura tropical. terra firme, solo arenoso, campina.
Carcaraí Km 44. Estação Experimental de Silvicultura Tropical. Terra firme, solo arenoso, campina.
## [607] "Flores alvas. Floresta preservada, dossel médio 20m."
## [608] "Flores alvas. Mata de terra firme."
## [609] "Flores amarelas com perfume agradável."
## [610] "Flores amarelas solo arenoso. árvore. ritidoma verrucoso, por lenticelas elípticas horiz-
## [611] "Flores amarelas, pedicelo verde"
## [612] "Flores pequenas, amarelo-
claro, com perfume. flores pequenas, de cor amarelo-
claro com perfume."
## [613] "Floresta Atlântica. Arbóreo. Coletada no Porto Capim.; DESC: Floresta Atlântica. Arbórea
## [614] "Floresta estacional"
## [615] "Floresta estacional decidual."
## [616] "Floresta estacional semidecidual submontana."
## [617] "Floresta Estacional Semidecidual Submontana. ; HA"
## [618] "Floresta Estacional Semidecidual Submontana. ; HA"
## [619] "Floresta estacional troca de material.; árvore ca 8,0 m. folhas cartáceas, discoloração
## [620] "Floresta Ombrófila Densa"
## [621] "Floresta Ombrófila Densa"
## [622] "Floresta Ombrófila Densa"
## [623] "Floresta Ombrófila Densa"
## [624] "Floresta Ombrófila Densa"
## [625] "Floresta ombrófila densa em regeneração."
## [626] "Floresta Ombrófila Montana"
## [627] "Floresta Ombrófila Montana"
## [628] "Floresta ombrófila."
## [629] "Floresta ombrófila."
## [630] "Floresta ombrófila."
## [631] "Floresta ombrófila."
```

```
## [632] "Floresta ombrófila."
## [633] "Floresta ombrófila."
## [634] "Floresta ombrófila."
## [635] "Flws. yellow. Male."
## [636] "Folhas alternas, glabras."
## [637] "Folhas coriáceas, discolores, inflorescencia axilar, ferruginosa, rafe esverdeada, bot ALCB (48550). Projeto Financiado pelo CNPq. mata."
## [638] "Folhas simples, alternas, alongadas, oblongas, cheiro agradável. P8 I36"
## [639] "fruto maduro"
## [640] "Frutos ainda jovens"
## [641] "frutos verdes"
## [642] "Grand arbor."
## [643]      "HABITAT:Caatinga.      Cerrado,      arbóreo-
arbustivo.; DESC:Árvore apro. 4m de altura. Caule cinzento. Frutos enegrecidos, com receptáculo do
## [644] "HABITAT:Cerrado.; DESC:Árvore, frutos verdes.;"
## [645] "Ident. ant.: o. organensis (Meis.) Mez"
## [646]      "Inflorescencia com pedúnculo verde-
limao, flores com pedicelos, cálice e corola branco-
amarelados, androceu e gineceu amarelados, cheiro doce agradável inflorescencia com pedúnculo verde-
limao. flores com pedicelos, cálice e corola branco-
amarelados. androceu e gineceu amarelados, cheiro doce agradável. inflorescência com pedúnculo verde-
limão. Flores com pedicelos, cálice e corola branco-
amarelados. Androceu e gineceu amarelados, cheiro doce agradável."
## [647] "Lago do Castanho - Mirim. arvoreta de 3m de altura acima do nível da água do igapó, frutos
## [648] "Linha de transmissao Jirau/Sts Antonio."
## [649] "Linha de transmissao Jirau/Sts Antonio."
## [650]      "Localizado em solo areno argiloso-
pedregoso. Habitat:Terrestre no interior da mata. Cálice persistente vináceo. Ind. 2216. Parcela 4
## [651]      "Localizado em solo areno argiloso-
pedregoso. Habitat:Terrestre no interior da mata. Exudado incolor pegajoso, cheiro amargoso. Ind.
## [652]      "Localizado em solo areno argiloso-
pedregoso. Habitat:Terrestre no interior da mata. Ind. 1290. Parcela 26.; Árvore de 7,0 m. Folhas a
## [653]      "Localizado em solo areno argiloso-
pedregoso. Habitat:Terrestre no interior da mata. Ind. 1580. Parcela 32.; Árvore perfilhada de 5,0
```

```
## [654] "Localizado em solo arenoso argiloso-  
pedregoso. Habitat:Terrestre no interior da mata. Ind. 2208. UTM: 8457360.; Árvore de 8,0 m. Folha  
## [655] "Localizado em solo arenoso argiloso-  
pedregoso. Habitat:Terrestre no interior da mata. Ind. 2360. Parcela 50.; Árvore de 8,0 m. Folhas a  
## [656] "Mata Atlântica"  
## [657] "Mata Atlântica"  
## [658] "Mata Atlântica"  
## [659] "Mata Atlântica (Floresta Estacional Semidecidual); HABITAT: Mata Atlântica (Floresta Es-  
## [660] "Mata Atlântica de encosta."  
## [661] "Mata Atlântica em vale de tabuleiro."  
## [662] "Mata Atlântico, beira da lagoa."  
## [663] "Mata Ciliar"  
## [664] "Mata ciliar."  
## [665] "Mata de Chaves.Borda com canavial.; HABITAT: Mata de Chaves.Borda com canavial."  
## [666] "Mata de terra firme."  
## [667] "Mata em leito de córrego."  
## [668] "Mata ombrófila secundária. Solo arenoso."  
## [669] "Mata perenifólia sob latosolos"  
## [670] "Material sem dados da planta e sem data de coleta."  
## [671] "Material sem dados da planta."  
## [672] "Material sem dados da planta."  
## [673] "Município de Humaitá, Pixuna, Km 40 da Rodovia Transamazônica. Campina, flores alvas."  
## [674] "Município de Oriximiná, rio Paru do Oete, entre cachoeira pancada e rio Trombetas. Campin  
## [675] "Município de Oriximiná, rio Paru do Oete, entre cachoeira pancada e rio Trombetas. Campin  
## [676] "Município de Oriximiná, rio Trombetas, margem esquerda, a 5Km abaixo da cachoeira Porte  
## [677] "Muy alto y ranudo; hoja verde clara; flor amarilla; fruto ovalado, verde claro."  
## [678] "old secondary forest with few primary remnants"  
## [679] "open uplands, \\muena\\, 3 m, cupule green, fr. green."  
## [680] "pequena árvore de formaçao rupestre em frutos. pequena árvore de formação rupestre em fru  
## [681] "Pétalas creme, anteras amarelada. Área de influencia direta (canteiro)."  
## [682] "Planta c/ 6m de altura. N. V - Canela Sabiá."  
## [683] "Planta com 6,0 metros de altura. Fruto e cáile verdes."  
## [684] "Planta com fruto."  
## [685] "Planta com grossa casca fissurada de coloraçao cinza. A espécie possuiindivíduos de gran  
Reserva Ecológica do IBGE. Plantas da Bahia."  
## [686] "Pole ca. 1.3 m. Receptacle and fruits green. Mata de terra firme."  
## [687] "Pole ca. 1.5-2 m; fruits and receptacle green. Disturbed roadside margins in tall forest,"
```

2 m; fruits and receptacle green. disturbed roadside margins in tall forest, terra firme. laterized
2 m; fruits and receptacle green. Disturbed roadside margins in tall forest, terra firme. Laterized
[688] "Pole-like tree 1.5-2.5m tall. Receptacle green, fruits light green. Pickled fruits & flow-
Coumarouma forest."
[689] "Possui acúleos."
[690] "Restinga. Floresta."
[691] "Rio Trombetas, Monte Branco, jazida de Bauxita de Alcoa -
Mineraçao, mata de campina, solo arenoso. Árvore de 3-
6m, flores alvas na campina. rio trombetas, monte branco, jazida de bauxita de alcoa -
mineração, mata de campina, solo arenoso. árvore de 3-
6m, flores alvas na campina."
[692] "Secondary vegetation on road to airstrip. Tree to 10m tall; cupule green, immature fruits
yellow."
[693] "Sem dados. sem dados.; sem dados."
[694] "Shrub 2,5 m tall in disturbed white sand near gravel pits; flowers pale yellow. shrub 2,5
[695] "Shrub 20 ft. high, 3 inches in dia."
[696] "Shrub 3 1/2 tall. Brushy clearing and secondary rain forest, sandy soil."
[697] "Shrub 3 m tall; flowers cream."
[698] "Shrub 3 m. Immature fruit (includng cupule) green."
[699] "Shrub 3m tall to tree 10m tall in forest margin. Flowers dirty yellow. shrub 3m, to tree 1
[700] "Shrub 5 m tall; brances and leaves ascendeing; stems carinate; midveins yellowish; infl-
[701] "Shrub ca. 2m tall. Receptacle dark green. Fruits light green. Flowers cream-
white. Flowers in alcohol."
[702] "Shrub to 2 m. high."
[703] "Shrub, 2 m tall; corolla green. Capoeira."
[704] "Shrub, 3 m tall; cupule dark green, fruit light green."
[705] "Shrub, 3 m tall; cupule dark green, fruit light green. Capoeira on white sand."
[706] "Shrub. Brushy clearings and secondary rain forest, sandy soil."
[707] "Shrubs and tree of indeterminable size pendent or leaning out over cliff; fruit green; fl-
white. Common pole along the river. N to NE bank-
shale cliffs to terra firme."
[708] "Shruv, 2 m tall; fruit green."
[709] "Small tree 5 m; flowers cream."
[710] "Small tree 5-6 m; flowers yellowish; receptacle red. Riverbank and cliff vegetation."
[711] "Small tree to 10 m. tall, young stems strongly alate."
[712] "Small tree to 4 m. tall. Flowers white, fragrant."
[713] "Small tree, 5 m high. Leaves glossy green above, greyish sheen beneath. Terminal inflores-

```

## [714] "Small tree; 4 cm. dbh."
## [715] "Término coleta 02.04.1979"
## [716] "Terrestre, arbórea, fruto maduro e cor roxa, fruto imaturo de cor verde, exsudação incold
##      [717]      "Tierra      firma;      lomas      de      30-
50 m, de arcilla rojiza. Árbol de 20 m. Flores amarillentas, pétalos erectos. Corteza oscura, en es
## [718] "Trabalho de Conclusao de Curso"
##      [719]      "Track      from      Km      63,      Manaus      -
Itacoatiara Road. Savanna forest on white sand. Treelet, 4m tall. Flowers white."
## [720] "Tree # 2685, 14-15m; fruit lighter green in darker green cup."
## [721] "Tree +- tall, leaves coriaceous, glossy and dark green above, lustrous pale brown beneath"
## [722] "Tree 10 m, flowers cream."
## [723] "Tree 10 m., flowers white, fruits black with red cupule."
## [724] "Tree 10m tall x 15cm daiemter. Fruit green. Primary forest."
## [725] "Tree 10m tall. Flowers cream. Fruit ( in cups ) green. Terra firme; high forest near black
## [726] "Tree 15 m, diam. 20 cm. Bark dark brown warty lenticles and looping pattern of fine ridges
## [727] "Tree 15 m, flowers white"
## [728] "Tree 15 m. tall, 80 cm. circum. fls. minute; tepals yellow; stamens and stigma brown."
## [729] "Tree 15-25m, fls cream"
## [730] "Tree 16 ft. high, 4 inches in diameter."
## [731] "Tree 18. bole 12m, straight, cylindrical. Fruits, flower small"
## [732] "Tree 18m tall, flowers cream."
##      [733]      "Tree      18m      tall,      flowers      greenish-
white. Occasional. Edge of forest. Hilly forest on terra firme."
## [734] "Tree 2.5m tall. Buds cream."
## [735] "Tree 20m tall x 40cm diameter, cupule dark green, fruit pale green. tree 20m tall x 40cm d
## [736] "Tree 30 ft. high, 5 inches in dia. tree 50 ft. high, 6 inches in dia."
## [737] "TREE 30 M X 80 CM DIAM. CUPULE DARK GREEN, FRUIT PALE GREEN."
## [738] "Tree 4 meters"
## [739] "Tree 5 m. tall; fruit green."
## [740] "Tree 5m tall, cupule red, fruit green turning red."
## [741] "TREE 6 M X 10 CM DIAM, BUDS GREEN."
## [742] "Tree 6 m. Flowers greenish-white."
## [743] "Tree 70 ft. high, 8 inches in diameter."
## [744] "Tree 8 m tall; inflorescence yellow, very fragrant. Margins of forest on terra firme."
## [745] "Tree 8 m, flowers whitish."
##      [746]      "Tree      8m      tall,      perianth      greenish-
white, outer whorl of stamens pale green, inner whorl of stamens orange, stigma black. Occasional.

```

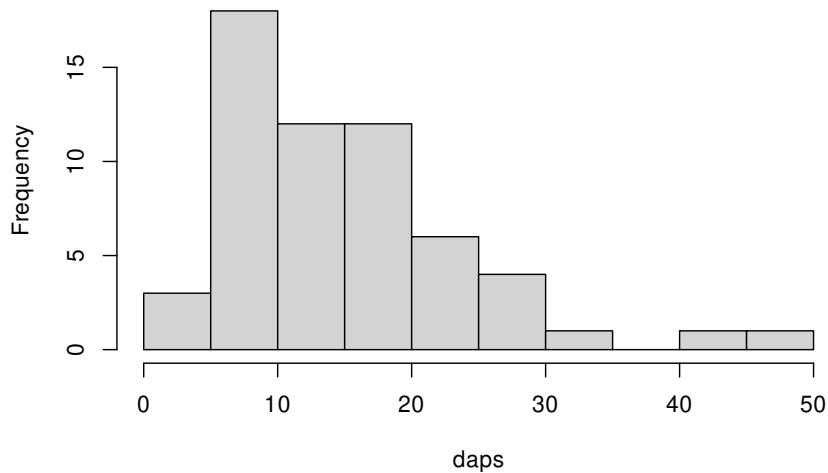
```
## [747] "TREE CA 5 M. FRUITS LIGHT GREEN, RECEPTACLE DARKER GREEN. FLOWERS CREAM WHITE."
## [748] "Tree ca. 2.5-3 m; flowers cream-white. Disturbed roadside margins in tall forest, terra firme; laterized clay with sand deposite. 3 m; flowers cream-white. disturbed roadside margins in tall forest, terra firme; laterized clay w
## [749] "Tree ca. 3 m; fruits green, becoming red-tinged; receptacle green. Disturbed roadside margins in tall forest, terra firme. Laterized clay w
tinged; receptacle green. disturbed roadside margins in tall forest, terra firme. laterized clay w
tinged; receptacle green.;Tree ca. 3 m; fruits green, becoming red-tinged; receptacle green. Disturbed roadside margins in tall forest, terra firme. Laterized clay w
## [750] "Tree ca. 6m tall. Buds cream-white; fruits green. Terra firme ( Assoc.: Dipteryx, Bertholletia, Couratari, a few bamboos )."
## [751] "Tree formerly 15 m. tall or more. Now broken off at top and resprouting. Petals cream."
## [752] "Tree shrub juvenile... revisar;Tree shrub juvenile"
## [753] "tree shurb juvenile"
## [754] "Tree to 25m.tall, 70cm. DBH;wood medium soft. Bark smooth;lvs. glaucous beneath, not aro
## [755] "Tree to 4 m. Leaves coriaceous, mid green, paler beneath. Fruit green with paler elongate
## [756] "tree to 6m tall: cupula green with white scales; youngs fruits light green."
## [757] "Tree, 10 m; flowers greenish-brown; leaves sub-coriaceous, lustrous above; leaf rachis and young stem greenish-tan"
## [758] "Tree, 18 m. x 20 cm. diameter. Flowers green."
## [759] "Tree, 18 m. x 25 cm. diameter. Flowers cream."
## [760] "Tree, 5 m tall; cupule dark green; fruit light green. Swamp beside stream."
## [761] "Tree, 5 m. x 8 cm. diameter. Buds green."
## [762] "Tree, 6 m tall; fruit green. Secondary forest by airstrip."
## [763] "Tree, 6 m tall; fruit pale green, cupule dark green."
## [764] "Tree, 6 m tall; petals yellow, stames yellow; fruit green. Capoeira."
## [765] "Tree, 6 m; inflorescence pale green, flowers pale green"
## [766] "Tree, 60cm dbh. Leaves collected from forest floor. Transect tree 2:7/4a. Vegetative eco
## [767] "Tree, 8-10 m tall; cupule greenish brown; fruit ellipsoid, green."
## [768] "Tree, fruits immature. Height: 10.0 m; DBH: 10.0 cm."
## [769] "Treelet 2m tall. Cupule dark green, fruit pale green."
## [770] "Treelet 3m tall. Buds and flowers yellow. Low forest on terra firme."
## [771] "Treelet 4m tall. Fruit green. Forest on terra firme."
## [772] "Treelet, 15 m; buds pale green. Primary forest."
## [773] "Treelet, 3 m tall; flowers cream, cupule dark green, fruit maturing red. Disturbed forest
## [774] "Treelet, 4 m DBH; 3 cm, bark gray, brts grayed green with slight bloom; lvs yellow-
```

green, shining above, paler, more or less glaucous beneath; fls orange-yellow. Same as no. 328."

```
## [775] "Treelet, 4 m tall; flowers yellow-
green, young leaves sericeous, silver. White sand capoeira."
## [776] "Treelet, 4 m. tall. Buds green."
## [777] "Treelet, 4 m. tall. Corolla cream."
## [778] "Treelet, 4 m. tall. Fruit red. Cupule green."
## [779] "Treelet, 5 m tall. Forest on terra firme."
## [780] "Tronco com a base reta. Casca externa cinza. Ramos jovens com quinas."
## [781] "UHE Teles Pires"
## [782] "UHE Teles Pires"
## [783] "UHE Teles Pires"
## [784] "Very common treelet, 1-
5 m tall, fls. white, fruit black. Secondary forest."
## [785] "VINY SHRUB, CA 4.5M TALL. WOOD IN ALCOHOL"
```

```
# histograma
hist(daps)
```

Histogram of daps



```
# adiciona ao conjunto de dados  
dados$DAPcm <- daps
```

Reparam que ambas as funções ainda precisam de alguns ajustes. Retirar variáveis de vetores de texto é algo complexo, especialmente quando não se há um padrão, fato observado comumente nas etiquetas de espécimes coletados e depositados em herbários. Ainda assim, conseguimos extrair bastante informação da coluna de notas de texto estocada em `txt.org`.



Parte II

Parte II



Tutorial para orientar a AED a ser realizada com dados de interesse do aluno.



II

Checagem dos dados

No tutorial abaixo vamos usar dados de avistamento de aves em fisionomias de cerrado. Baixem o arquivo contendo esses dados (https://github.com/LABOTAM/IntroR/blob/main/dados/aves_cerrado.csv) para a sua pasta de trabalho.

Vamos praticar neste tutorial o uso de funções que nos mostram a estrutura e resumo dos dados. Já vimos como utilizar essas funções na parte I deste livro.



- `str()` - mostra a estrutura do objeto dos dados;
- `head()` e `tail()` - mostra a cabeça ou a cauda da sua tabela de dados, respectivamente;
- `summary()` - faz um resumo de todas as variáveis nos seus dados.

Vamos começar importando os dados ao R:

```
## Lendo a planilha com read.table
avesc <- read.table("aves_cerrado.csv", row.names = 1, header = T, sep = ";", dec = ",", as.is = TRUE)
```

Repare no argumento `na.strings` da função `read.table()`. Ele é importante na importação de dados pois garante a codificação de valores ausentes usando a constante lógica `NA` do R. Se você não definir isso, o padrão é reconhecer apenas células que tenham o texto **NA** como valor faltante. Veja também o uso do argumento `as.is = TRUE`, que indica que não se deve converter colunas de texto em fatores (ou você pode usar o argumento `stringsAsFactors` para isso; este mesmo

argumento é utilizado também na função `data.frame()`). Vamos ver as primeiras 10 linhas do objeto `avesc`:

```
head(avesc, 10)
```

	fisionomia	urubu	carcara	seriema
Ce1	Ce	5	18	6
Ce2	Ce	7	7	6
Ce3	Ce	5	14	4
Ce4	Ce	3	12	5
Ce5	Ce	4	16	4
Ce6	Ce	NA	20	NA
Ce7	Ce	6	19	4
Ce8	Ce	4	21	10
Ce9	Ce	2	11	6
Ce10	Ce	5	9	7

Vamos criar uma cópia para usarmos depois.

```
aves2 <- avesc
```

É sempre bom verificar se os dados foram importados corretamente. É sempre um bom procedimento checarmos as dimensões do objeto com a função `dim()`, primeiras e últimas linhas e colunas do objeto com funções `head()` e `tail()` respectivamente

```
dim(avesc) # a dimensao do objeto (linhas e colunas)
```

```
## [1] 60 4
```

```
head(avesc, 3) # a cabeca do objeto (tres primeiras linhas)
```

	fisionomia	urubu	carcara	seriema
Ce1	Ce	5	18	6
Ce2	Ce	7	7	6
Ce3	Ce	5	14	4

```
tail(avesc, 3) # a cauda do objeto (tres ultimas linhas)
```

	fisionomia	urubu	carcara	seriema
CL18	CL	12	6	7
CL19	CL	18	5	4
CL20	CL	19	4	4

```
avesc[nrow(avesc), ] # ultima linha
```

	fisionomia	urubu	carcara	seriema
CL20	CL	19	4	4

Parece que está tudo certo! Vamos checar a estrutura do objeto:

```
# mostra a estrutura do data.frame
str(avesc)
```

```
## 'data.frame':   60 obs. of  4 variables:
## $ fisionomia: chr  "Ce" "Ce" "Ce" "Ce" ...
## $ urubu      : int  5 7 5 3 4 NA 6 4 2 5 ...
## $ carcara    : int  18 7 14 12 16 20 19 21 11 9 ...
## $ seriema    : int  6 6 4 5 4 NA 4 10 6 7 ...
```

Reparam que há uma variável de texto (`chr`) e três variáveis de números inteiros (`int`). Próximo passo é sempre checar um sumário estatístico das variáveis presentes no objeto usando a função `summary()`:

```
# mostra um resumo da variacao nas colunas
summary(avesc)
```

	fisionomia	urubu	carcara	seriema
Length:60	Min. : 2.00	Min. : 3.00	Min. : 2.000	
Class :character	1st Qu.: 7.00	1st Qu.: 5.50	1st Qu.: 4.000	
Mode :character	Median :12.00	Median : 9.00	Median : 5.000	
NA	Mean :11.93	Mean :10.25	Mean : 5.345	
NA	3rd Qu.:16.50	3rd Qu.:14.00	3rd Qu.: 6.000	
NA	Max. :22.00	Max. :24.00	Max. :12.000	
NA	NA's :1	NA's :1	NA's :2	

Há indicação de presença de valores `NA` nas variáveis numéricas, que são valores faltantes.

11.1 Tem valores ausentes?



Há valores ausentes em nossos dados? Eles são mesmo faltantes? Ou seja, o que significam valores ausentes no seu conjunto de dados?

Podemos utilizar a função `is.na()` para encontrar a constante lógica `NA`, ou seja, a constante que indica valores ausentes (reveja o uso da função `is.na()` na seção 2.6.3). Vejam o `? NA` da constante lógica `NA` para entender o significado dela no R:

```
?NA
```

Continuemos de onde paramos na seção anterior. Vimos que há a presença de `NA` no `data.frame avesc`. Chequemos quais registros

(linhas) têm valores `NA`. Vamos checar primeiramente a variável `avesc$urubu`:

```
avesc$urubu == NA ## erro: não retorna verdadeiro ou falso  
avesc[avesc$urubu == NA, ] ## também não funciona
```

Reparam que os comandos acima, apesar de funcionarem, não respondem à nossa pergunta que é saber quais linhas possuem `NA`. Para isso, devemos nos valer da função `is.na()`:

```
is.na(avesc) # pergunta em todo o data frame: quem é NA?
```

	fisionomia	urubu	carcara	seriema
Ce1	FALSE	FALSE	FALSE	FALSE
Ce2	FALSE	FALSE	FALSE	FALSE
Ce3	FALSE	FALSE	FALSE	FALSE
Ce4	FALSE	FALSE	FALSE	FALSE
Ce5	FALSE	FALSE	FALSE	FALSE
Ce6	FALSE	TRUE	FALSE	TRUE
Ce7	FALSE	FALSE	FALSE	FALSE
Ce8	FALSE	FALSE	FALSE	FALSE
Ce9	FALSE	FALSE	FALSE	FALSE
Ce10	FALSE	FALSE	FALSE	FALSE
Ce11	FALSE	FALSE	FALSE	FALSE
Ce12	FALSE	FALSE	FALSE	FALSE
Ce13	FALSE	FALSE	FALSE	FALSE
Ce14	FALSE	FALSE	FALSE	FALSE
Ce15	FALSE	FALSE	FALSE	FALSE
Ce16	FALSE	FALSE	FALSE	FALSE
Ce17	FALSE	FALSE	FALSE	FALSE
Ce18	FALSE	FALSE	FALSE	FALSE
Ce19	FALSE	FALSE	FALSE	FALSE
Ce20	FALSE	FALSE	FALSE	FALSE
CC1	FALSE	FALSE	FALSE	FALSE
CC2	FALSE	FALSE	FALSE	FALSE
CC3	FALSE	FALSE	FALSE	FALSE
CC4	FALSE	FALSE	FALSE	FALSE
CC5	FALSE	FALSE	FALSE	TRUE
CC6	FALSE	FALSE	FALSE	FALSE
CC7	FALSE	FALSE	FALSE	FALSE
CC8	FALSE	FALSE	FALSE	FALSE
CC9	FALSE	FALSE	FALSE	FALSE
CC10	FALSE	FALSE	FALSE	FALSE
CC11	FALSE	FALSE	FALSE	FALSE
CC12	FALSE	FALSE	FALSE	FALSE
CC13	FALSE	FALSE	FALSE	FALSE
CC14	FALSE	FALSE	FALSE	FALSE
CC15	FALSE	FALSE	FALSE	FALSE
CC16	FALSE	FALSE	FALSE	FALSE
CC17	FALSE	FALSE	FALSE	FALSE
CC18	FALSE	FALSE	FALSE	FALSE
CC19	FALSE	FALSE	FALSE	FALSE
CC20	FALSE	FALSE	FALSE	FALSE
CL1	FALSE	FALSE	FALSE	FALSE
CL2	FALSE	FALSE	FALSE	FALSE
CL3	FALSE	FALSE	FALSE	FALSE

```
!is.na(avesc) # inverte: quem não é NA?
```

	fisionomia	urubu	carcara	seriema
Ce1	TRUE	TRUE	TRUE	TRUE
Ce2	TRUE	TRUE	TRUE	TRUE
Ce3	TRUE	TRUE	TRUE	TRUE
Ce4	TRUE	TRUE	TRUE	TRUE
Ce5	TRUE	TRUE	TRUE	TRUE
Ce6	TRUE	FALSE	TRUE	FALSE
Ce7	TRUE	TRUE	TRUE	TRUE
Ce8	TRUE	TRUE	TRUE	TRUE
Ce9	TRUE	TRUE	TRUE	TRUE
Ce10	TRUE	TRUE	TRUE	TRUE
Ce11	TRUE	TRUE	TRUE	TRUE
Ce12	TRUE	TRUE	TRUE	TRUE
Ce13	TRUE	TRUE	TRUE	TRUE
Ce14	TRUE	TRUE	TRUE	TRUE
Ce15	TRUE	TRUE	TRUE	TRUE
Ce16	TRUE	TRUE	TRUE	TRUE
Ce17	TRUE	TRUE	TRUE	TRUE
Ce18	TRUE	TRUE	TRUE	TRUE
Ce19	TRUE	TRUE	TRUE	TRUE
Ce20	TRUE	TRUE	TRUE	TRUE
CC1	TRUE	TRUE	TRUE	TRUE
CC2	TRUE	TRUE	TRUE	TRUE
CC3	TRUE	TRUE	TRUE	TRUE
CC4	TRUE	TRUE	TRUE	TRUE
CC5	TRUE	TRUE	TRUE	FALSE
CC6	TRUE	TRUE	TRUE	TRUE
CC7	TRUE	TRUE	TRUE	TRUE
CC8	TRUE	TRUE	TRUE	TRUE
CC9	TRUE	TRUE	TRUE	TRUE
CC10	TRUE	TRUE	TRUE	TRUE
CC11	TRUE	TRUE	TRUE	TRUE
CC12	TRUE	TRUE	TRUE	TRUE
CC13	TRUE	TRUE	TRUE	TRUE
CC14	TRUE	TRUE	TRUE	TRUE
CC15	TRUE	TRUE	TRUE	TRUE
CC16	TRUE	TRUE	TRUE	TRUE
CC17	TRUE	TRUE	TRUE	TRUE
CC18	TRUE	TRUE	TRUE	TRUE
CC19	TRUE	TRUE	TRUE	TRUE
CC20	TRUE	TRUE	TRUE	TRUE
CL1	TRUE	TRUE	TRUE	TRUE
CL2	TRUE	TRUE	TRUE	TRUE
CL3	TRUE	TRUE	TRUE	TRUE

```
avesc[!is.na(avesc)]
```

```
## [1] "Ce" "Ce"
## [16] "Ce" "Ce" "ce" "Ce" "Ce" "CC" "CC" "CC" "CC" "CC" "CC" "CC" "CC" "CC" "CC"
## [31] "CC" "CC" "CC" "CC" "CC" "CC" "CC" "CC" "CC" "CL" "CL" "CL" "CL" "CL"
## [46] "CL" "CL"
## [61] "5" "7" "5" "3" "4" "6" "4" "2" "5" "6" "6" "7" "6" "5" "5"
## [76] "3" "13" "8" "7" "22" "10" "17" "16" "20" "18" "16" "14" "12" "15" "9"
## [91] "11" "20" "18" "8" "15" "17" "17" "12" "12" "11" "10" "19" "15" "13" "12"
## [106] "16" "13" "19" "19" "11" "18" "15" "19" "13" "16" "10" "12" "18" "19" "18"
## [121] "7" "14" "12" "16" "20" "19" "21" "11" "9" "12" "24" "17" "16" "15" "12"
## [136] "14" "14" "21" "13" "8" "16" "14" "16" "10" "7" "8" "8" "7" "14" "11"
## [151] "12" "11" "11" "7" "14" "8" "14" "7" "9" "3" "5" "5" "8" "6" "5"
## [166] "5" "3" "5" "3" "3" "6" "4" "4" "5" "3" "6" "5" "4" "6" "6"
## [181] "4" "5" "4" "4" "10" "6" "7" "8" "5" "6" "4" "6" "8" "3" "4"
## [196] "2" "4" "5" "3" "3" "6" "3" "5" "4" "4" "4" "8" "4" "4" "3"
## [211] "6" "5" "5" "6" "6" "12" "9" "6" "5" "4" "6" "4" "5" "6" "5"
## [226] "4" "9" "7" "5" "4" "6" "2" "10" "7" "4" "4"
```

Proceder com o último comando gera um resultado confuso, pois o `data.frame` é convertido em um vetor de dimensão de 236 valores. Podemos investigar da maneira abaixo. Checamos qual é o tamanho total de valores presentes no objeto `avesc`, multiplicando o número de linhas pelo número de colunas através da expressão `(nrow(avesc) * ncol(avesc))`. Temos então um número total de 240 valores possíveis em `avesc`. E comparamos essa valor com o número de valores não faltantes em `avesc` através da expressão `length(avesc[!is.na(avesc)])`, que retorna 236. Portanto, se não houver valores faltantes, a primeira expressão abaixo deve retornar verdadeiro (`TRUE`), e falso (`FALSE`) se houver valores faltantes:

```
(nrow(avesc) * ncol(avesc)) == length(avesc[!is.na(avesc)])
```

```
## [1] FALSE
```

Então quantos valores faltantes existem em nossos dados?

```
# ou então, o número de valores NA no data.frame é de:  
(nrow(avesc) * ncol(avesc)) - length(avesc[!is.na(avesc)])
```

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

O procedimento adotado acima pode ser difícil de entender. Fazer essa pergunta por colunas torna o entendimento mais fácil:

```
is.na(avesc$urubu) # quais são NA, vetor lógico
```

```
## [1] FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE  
## [13] FALSE  
## [25] FALSE  
## [37] FALSE  
## [49] FALSE FALSE
```

```
# mesmo que  
is.na(avesc$urubu) == T
```

```
## [1] FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE  
## [13] FALSE  
## [25] FALSE  
## [37] FALSE  
## [49] FALSE FALSE
```

```
# e o contrário é (quem não é NA)  
is.na(avesc$urubu) == F
```

```
## [1] TRUE TRUE TRUE TRUE FALSE TRUE TRUE TRUE TRUE TRUE TRUE  
## [13] TRUE  
## [25] TRUE  
## [37] TRUE  
## [49] TRUE TRUE
```

```
# ou simplesmente  
!is.na(avesc$urubu)
```

```
## [1] TRUE TRUE TRUE TRUE FALSE TRUE TRUE TRUE TRUE TRUE TRUE  
## [13] TRUE  
## [25] TRUE  
## [37] TRUE  
## [49] TRUE TRUE
```

Vetores lógicos TRUE e FALSE podem ser somados. TRUE corresponde a 1, e FALSE a 0. Usando o resultado de `is.na(avesc$urubu)` (ou qualquer outra variável de `avesc`) junto à função `sum()`, teremos então o número de valores faltantes na variável escolhida:

```
sum(is.na(avesc$urubu)) # quantos são?
```

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

```
sum(!is.na(avesc$urubu)) # quantos não são?
```

```
## [1] 59
```

```
# e, isso é verdadeiro, né?
(sum(is.na(avesc$urubu)) + sum(!is.na(avesc$urubu))) == nrow(avesc)
```

```
## [1] TRUE
```

Podemos perguntar quais posições do vetor lógico oriundo de `is.na(avesc$urubu)` são correspondentes a `NA` por meio da função `which()`. Teremos como resposta um vetor de números inteiros indicando o número das linhas com valores `NA` na `urubu`:

```
which(is.na(avesc$urubu)) # vetor com indices das posições que são NA
```

```
## [1] 6
```

Vamos utilizar agora este resultado para filtrar o `data.frame` `avesc` e checar que linha é essa:

```
avesc[which(is.na(avesc$urubu)), ] # mesma coisa, mas precisa de uma segunda função, então me
```

	fisionomia	urubu	carcara	seriema
Ce6	Ce	NA	20	NA

Podemos filtrar também sem a função `which()`, usando apenas os vetores lógicos `TRUE` e `FALSE` oriundos da função `is.na()`:

```
avesc[is.na(avesc$urubu), ] # mostra as linhas completas para os registros com NA na coluna ur
```

	fisionomia	urubu	carcara	seriema
Ce6	Ce	NA	20	NA

```
## para ver se tem NA em uma das tres colunas com nomes de aves: usamos o operador | (quer dizer que pelo menos uma das tres colunas tem NA)
meufiltro <- is.na(avesc$urubu) | is.na(avesc$carcara) | is.na(avesc$serieme)
sum(is.na(avesc)) # soma dos valores NA nas tres colunas
```

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

```
avesc[meufiltro, ] # mostra todas as linhas que tem algum valor NA
```

	fisionomia	urubu	carcara	serieme
Ce6	Ce	NA	20	NA
CC5	CC	20	10	NA
CL9	CL	19	NA	5

Esses valores NA, na verdade são AUSENCIA da ave (não avistamento) numa determinada localidade

```
## Então, vamos corrigir estes valores
vl <- is.na(avesc$urubu)
## Podemos ver os valores de vários jeitos
avesc$urubu[vl] # assim
```

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

```
avesc[vl, "urubu"] # ou assim
```

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

```
avesc[, "urubu"][vl] # ou assim...
## [1] NA

# se podemos ver, podemos atribuir 0 para esse valor ausentes
avesc$urubu[vl] <- 0

## Continuando, para as outras aves, mostrando variações de códigos
avesc$carcara[is.na(avesc$carcara)] <- 0
avesc$seriema[is.na(avesc$seriema) == T] <- 0

## Verificando se substituimos corretamente
avesc[meufiltro, ]
```

	fisionomia	urubu	carcara	seriema
Ce6	Ce	0	20	0
CC5	CC	20	10	0
CL9	CL	19	0	5

```
# poderíamos ter feito a mudança de uma vez
aves2[meufiltro, ] # a cópia que fiz no início
```

	fisionomia	urubu	carcara	seriema
Ce6	Ce	NA	20	NA
CC5	CC	20	10	NA
CL9	CL	19	NA	5

```
aves2[meufiltro, ][is.na(aves2[meufiltro, ])] # visualizo só os NAs
```

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

```
aves2[meufiltro, ][is.na(aves2[meufiltro, ])] <- 0 # atribuo 0  
  
## Agora esses valores são zero, certo?  
avesc[avesc$urubu == 0 | avesc$carcara == 0 | avesc$seriema == 0, ]
```

	fisionomia	urubu	carcara	seriema
Ce6	Ce	0	20	0
CC5	CC	20	10	0
CL9	CL	19	0	5

11.2 Colunas com fatores



As colunas com fatores estão codificadas corretamente?

Temos algumas funções úteis para se trabalhar com fatores. A primeira delas se chama `table()` e é responsável por fazer contagens de valores em fatores ou vetores de texto. Já as funções `factor()` e `as.factor()` permitem criar ou definir fatores.

```
# agora vamos ver a nossa coluna fisionomia, que não importamos como fator  
str(avesc)
```

```
## 'data.frame':   60 obs. of  4 variables:  
## $ fisionomia: chr  "Ce" "Ce" "Ce" "Ce" ...  
## $ urubu     : num  5 7 5 3 4 0 6 4 2 5 ...  
## $ carcara   : num  18 7 14 12 16 20 19 21 11 9 ...  
## $ seriema   : num  6 6 4 5 4 0 4 10 6 7 ...
```

```
avesc$fisionomia
```

```
## [1] "Ce" "Ce"
## [16] "Ce" "Ce" "ce" "Ce" "Ce" "CC" "CC" "CC" "CC" "CC" "CC" "CC" "CC" "CC"
## [31] "CC" "CC" "CC" "CC" "CC" "CC" "CC" "CC" "CC" "CL" "CL" "CL" "CL" "CL"
## [46] "CL" "CL"
```

```
class(avesc$fisionomia)
```

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

As categorias da variável `avesc$fisionomia` significam:

- “CL” = campo limpo;
- “CC” = campo cerrado;
- “Ce” = cerrado.

Vamos tabular essa coluna e verificar quantos valores temos para cada categoria:

```
table(avesc$fisionomia)
```

CC	ce	Ce	CL
20	1	19	20

Reparam que a categoria `ce` e `ce` são tratadas como diferentes pois o R¹ interpreta letras minúsculas e maiúsculas diferentemente. A falta de padronização em dados biológicos tabulados é muito frequente,

¹Isso ocorre na maioria das linguagens de programação. Vejam esta postagem: <https://softwareengineering.stackexchange.com/questions/9965/why-is-there-still-case-sensitivity-in-some-programming-languages>.

e trabalhar com scripts permite ao usuário documentar todas as mudanças em etapas que precedem o momento da análise. Por isso, fique sempre atento à padronização e a checagem de dados durante a AED.

Antes de proceder com a correção, vamos fazer uma cópia da variável `avesc$fisionomia` para fins deste exercício:

```
fisionomia.copia <- avesc$fisionomia
```

Digamos que o padrão deve ser `ce`, então vamos filtrar os valores presentes em `avesc` que não correspondem a `ce`, isto é, o valor `ce`:

```
vl <- avesc$fisionomia == "ce" # quem tem esse valor
avesc$fisionomia[vl] <- "Ce" # corrigindo
table(avesc$fisionomia)
```

CC	Ce	CL
20	20	20

Tendo em vista que a diferença é apenas de capitalização entre `ce` e `ce`, poderíamos ter feito simplesmente o exposto abaixo para efeito de correção:

```
avesc$fisionomia <- fisionomia.copia # volto ao valor original
```

Primeiro, usamos a cópia dos valores originais e o atribuímos aos valores modificados. Em seguida, mudamos a capitalização das palavras para caixa alta com a função `toupper()`. Em seguida, tabulamos as categorias:

```
# corrijo, simplesmente mudando tudo para maiúsculo:
avesc$fisionomia <- toupper(avesc$fisionomia)
table(avesc$fisionomia)
```

CC	CE	CL
20	20	20

Porém, se nós tivéssemos importado os dados transformando vetores de texto como fatores, por meio dos argumentos `as.is = FALSE` OU `stringsAsFactors = FALSE`, poderíamos proceder da seguinte maneira:

```
# digamos no entanto, que eu tivesse importado a coluna como fator
avesc$fisionomia <- as.factor(fisionomia.copia)
class(avesc$fisionomia)
```

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

```
levels(avesc$fisionomia) # os níveis ou categorias do fator
```

```
## [1] "CC" "ce" "Ce" "CL"
```

```
# isso é verdadeiro, certo?:
sort(unique(avesc$fisionomia)) == levels(as.factor(avesc$fisionomia))
```

```
## [1] TRUE TRUE TRUE TRUE
```

```
# sendo um fator, para corrigir, eu precisaria apenas:
levels(avesc$fisionomia)[2] <- "Ce"
levels(avesc$fisionomia) # pronto, corrigido
```

```
## [1] "CC" "Ce" "CL"
```

```
table(avesc$fisionomia)
```

CC	Ce	CL
20	20	20

```
## Verificando novamente
str(avesc)
```

```
## 'data.frame':   60 obs. of  4 variables:
## $ fisionomia: Factor w/ 3 levels "CC","Ce","CL": 2 2 2 2 2 2 2 2 2 ...
## $ urubu      : num  5 7 5 3 4 0 6 4 2 5 ...
## $ carcara    : num  18 7 14 12 16 20 19 21 11 9 ...
## $ seriema    : num  6 6 4 5 4 0 4 10 6 7 ...
```

```
summary(avesc)
```

fisionomia	urubu	carcara	seriema
CC:20	Min. : 0.00	Min. : 0.00	Min. : 0.000
Ce:20	1st Qu.: 6.75	1st Qu.: 5.00	1st Qu.: 4.000
CL:20	Median :12.00	Median : 9.00	Median : 5.000
NA	Mean :11.73	Mean :10.08	Mean : 5.167
NA	3rd Qu.:16.25	3rd Qu.:14.00	3rd Qu.: 6.000
NA	Max. :22.00	Max. :24.00	Max. :12.000



12

AED de univariadas

12.1 Qual a distribuição dos valores numéricos?



Onde os dados estão centrados? Como eles estão espalhados? Eles são simétricos, i.e., a distribuição é normal? São enviesados, bimodais? Existem valores extremos¹?

Já vimos algumas operações matemáticas com vetores e também como usar as funções `hist()` e `boxplot()` para gerar figuras de distribuição de variáveis numéricas individualmente. Vimos também como fazer iterações usando funções da família `apply()`. Também já aprendemos sobre a função `summary()`, que faz um resumo de todas as variáveis nos seus dados e aqui vamos entender isso melhor, apresentando a função `quantile()`, que permite extrair os quantis², que são valores que dividem uma distribuição probabilística em intervalos iguais de probabilidade. Com essas ferramentas, podemos descrever a distribuição de nossas variáveis numéricas.

12.1.1 Dados do tutorial

Vamos importar novamente os conjuntos de dados de avistamento de aves do cerrado³ (utilizado no capítulo 11) e de parcelas em caixetas⁴ (utilizado no capítulo 7):

²<https://en.wikipedia.org/wiki/Quantile>

³https://github.com/LABOTAM/IntroR/blob/main/dados/aves_cerrado.csv

⁴<https://github.com/LABOTAM/IntroR/blob/main/dados/caixeta.csv>

```
## Lendo a planilha com read.table  
avesc <- read.table("aves_cerrado.csv", row.names = 1, header = T, sep = ";", dec = ",", as.is = T)
```

```
caixeta <- read.csv("caixeta.csv") ## arquivo caixeta.csv deve estar no diretório de trabalho  
# note que mantemos todos os argumentos padrão (veja o formato do arquivo caixeta)
```

```
## Resumo estatístico: medias, media truncada e mediana, quantis  
# pegando apenas as variáveis numéricas  
avesc[, 2:4]
```

	urubu	carcara	seriema
Ce1	5	18	6
Ce2	7	7	6
Ce3	5	14	4
Ce4	3	12	5
Ce5	4	16	4
Ce6	NA	20	NA
Ce7	6	19	4
Ce8	4	21	10
Ce9	2	11	6
Ce10	5	9	7
Ce11	6	12	8
Ce12	6	24	5
Ce13	7	17	6
Ce14	6	16	4
Ce15	5	15	6
Ce16	5	12	8
Ce17	3	14	3
Ce18	13	14	4
Ce19	8	21	2
Ce20	7	13	4
CC1	22	8	5
CC2	10	16	3
CC3	17	14	3
CC4	16	16	6
CC5	20	10	NA
CC6	18	7	3
CC7	16	8	5
CC8	14	8	4
CC9	12	7	4
CC10	15	14	4
CC11	9	11	8
CC12	11	12	4
CC13	20	11	4
CC14	18	11	3
CC15	8	7	6
CC16	15	14	5
CC17	17	8	5
CC18	17	14	6
CC19	12	7	6
CC20	12	9	12
CL1	11	3	9
CL2	10	5	6
CL3	19	5	5

```
# podemos fazer um resumo estatístico da distribuição de cada uma dessas colunas
summary(avesc[, 2:4])
```

	urubu	carcara	seriema
	Min. : 2.00	Min. : 3.00	Min. : 2.000
	1st Qu.: 7.00	1st Qu.: 5.50	1st Qu.: 4.000
	Median :12.00	Median : 9.00	Median : 5.000
	Mean :11.93	Mean :10.25	Mean : 5.345
	3rd Qu.:16.50	3rd Qu.:14.00	3rd Qu.: 6.000
	Max. :22.00	Max. :24.00	Max. :12.000
	NA's :1	NA's :1	NA's :2

```
# essa função me retorna várias estatísticas da distribuição de cada variável
# os valores mínimos e máximos
# a tendência central pela média e pela mediana
# e o 1 e o 3 quartil, que juntamente com o mínimo, o máximo e mediana,
# indicam as divisões dos dados em quatro partes identicas (vamos ver isso melhor abaixo)

# função summary, mas não retorna por exemplo, o desvio padrão ou a variância das colunas.
```

```
# e não podemos fazer isso apenas com a função sd para todas as colunas
sd(avesc[, 2:4]) # ops depreciado (eu estou trabalhando com uma matriz)
```

```
# mas posso usar a função apply (para cada coluna, argumento MARGIN)
apply(avesc[, 2:4], 2, sd, na.rm = TRUE)
```

```
##      urubu    carcara   seriema
## 5.429372 5.383862 2.013566
```

```
# summary já retorna isso, mas eu poderia usar para qualquer função
apply(avesc[, 2:4], 2, median, na.rm = TRUE)
```

```
##    urubu carcara seriema
##      12         9         5
```

```
apply(avesc[, 2:4], 2, mean, na.rm = TRUE)
```

```
##    urubu    carcara    seriema
## 11.932203 10.254237  5.344828
```

```
apply(avesc[, 2:4], 2, min, na.rm = TRUE)
```

```
##    urubu carcara seriema
##      2         3         2
```

```
apply(avesc[, 2:4], 2, max, na.rm = TRUE)
```

```
##    urubu carcara seriema
##      22        24        12
```

```
apply(avesc[, 2:4], 2, quantile, na.rm = TRUE)
```

	urubu	carcara	seriema
0%	2.0	3.0	2
25%	7.0	5.5	4
50%	12.0	9.0	5
75%	16.5	14.0	6
100%	22.0	24.0	12

```
# note que os valores dos quartis:
quantile(avesc$urubu, na.rm = TRUE)
```

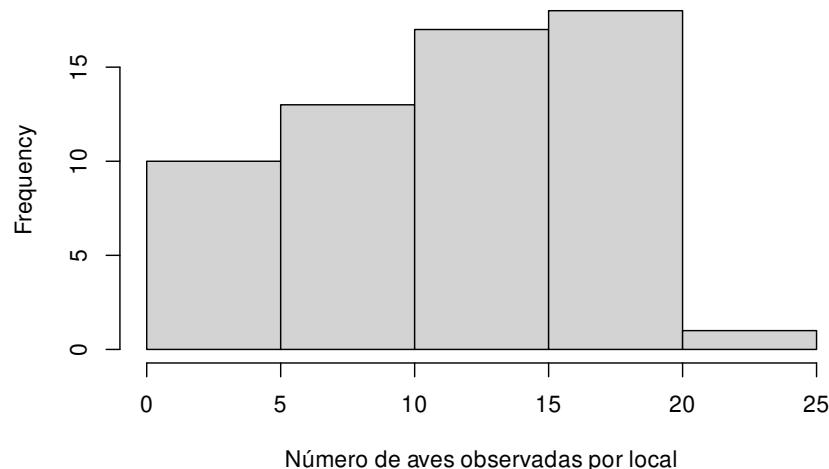
```
##    0%   25%   50%   75% 100%
##  2.0  7.0 12.0 16.5 22.0
```

```
# aparecem também quando usamos summary, que no entanto, retorna a média aritmética,
# que é o único valor que não é um quartil.
summary(avesc$urubu)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
2	7	12	11.9322	16.5	22	1

```
# para entender melhor isso vamos graficar:
# primeiro num histograma
hist(avesc$urubu, main = "Avistamentos de Urubu", xlab = "Número de aves observadas por local")
```

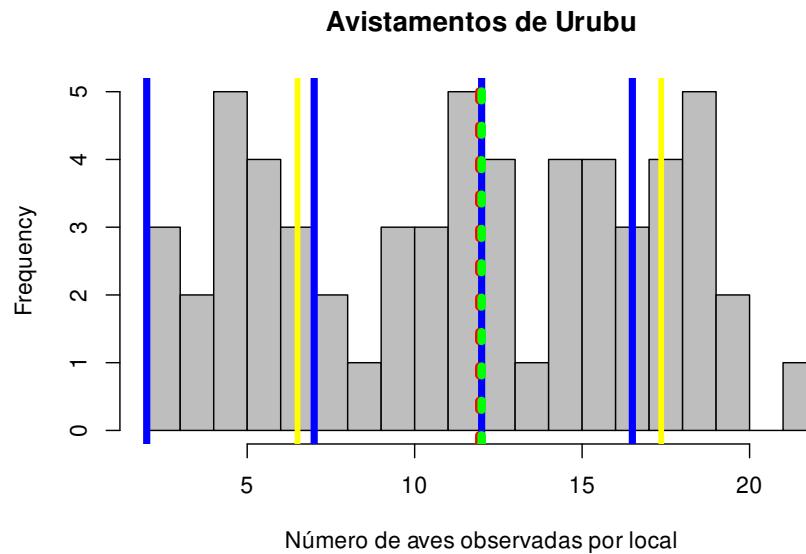
Avistamentos de Urubu



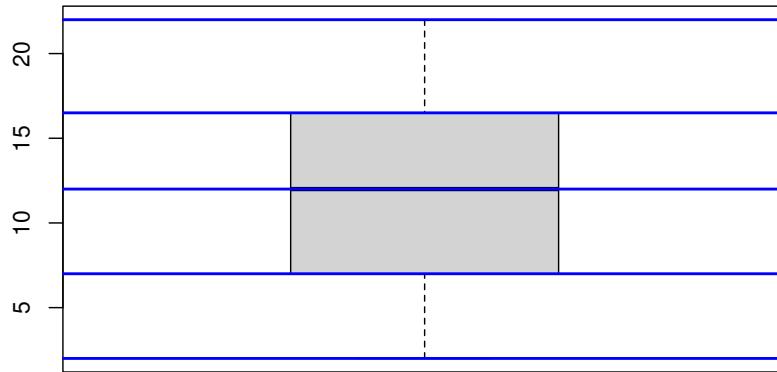
```

# melhorando um pouco
hist(avesc$urubu, main = "Avistamentos de Urubu", xlab = "Número de aves observadas por local")
# agora adicionamos em azul os quartis
abline(v = quantile(avesc$urubu, na.rm = TRUE), col = "blue", lwd = 5)
# note que as barras azuis estão igualmente espaçadas no eixo X, pois elas dividem a distribuição
# vamos plotar a média
abline(v = mean(avesc$urubu, na.rm = TRUE), col = "red", lty = "dotted", lwd = 6)
# e a mediana
abline(v = median(avesc$urubu, na.rm = TRUE), col = "green", lty = "dotted", lwd = 6)
# note como a mediana é equivalente ao quartil que indica 50% na divisão simétrica dos dados
# vamos adicionar o desvio padrão:
v1 <- sd(avesc$urubu, na.rm = TRUE) + mean(avesc$urubu, na.rm = TRUE)
v2 <- mean(avesc$urubu, na.rm = TRUE) - sd(avesc$urubu, na.rm = TRUE)
abline(v = c(v1, v2), col = "yellow", lty = "solid", lwd = 4)

```



```
# agora com um box plot:  
boxplot(avesc$urubu)  
abline(h = quantile(avesc$urubu, na.rm = TRUE), col = "blue", lwd = 2)
```



```
# Média truncada = e.g. TIRANDO 10% DOS VALORES NOS EXTREMOS (posso ver como muda, caso tenha  
?mean # veja o argumento trim
```

```
apply(avesc[, 2:4], 2, mean, trim = 0.1, na.rm = TRUE) # truncando
```

```
##      urubu    carcara    seriema  
## 12.000000  9.897959  5.145833
```

```
apply(avesc[, 2:4], 2, mean, trim = 0, na.rm = TRUE) # sem truncar
```

```
##      urubu    carcara    seriema  
## 11.932203 10.254237  5.344828
```

```
# valores de quantis em outras probabilidades
quantile(avesc$urubu, probs = seq(from = 0, to = 1, by = 0.1), na.rm = TRUE) # a cada 10%
```

```
##    0%   10%   20%   30%   40%   50%   60%   70%   80%   90% 100%
##  2.0   5.0   6.0   8.0  11.0  12.0  13.8  16.0  17.4  19.0 22.0
```

```
dim(caixeta) # dimensões
```

```
## [1] 1027      7
```

```
names(caixeta) # colunas
```

```
## [1] "local"   "parcela" "arvore"  "fuste"   "cap"     "h"       "especie"
```

```
head(caixeta) # cabeça
```

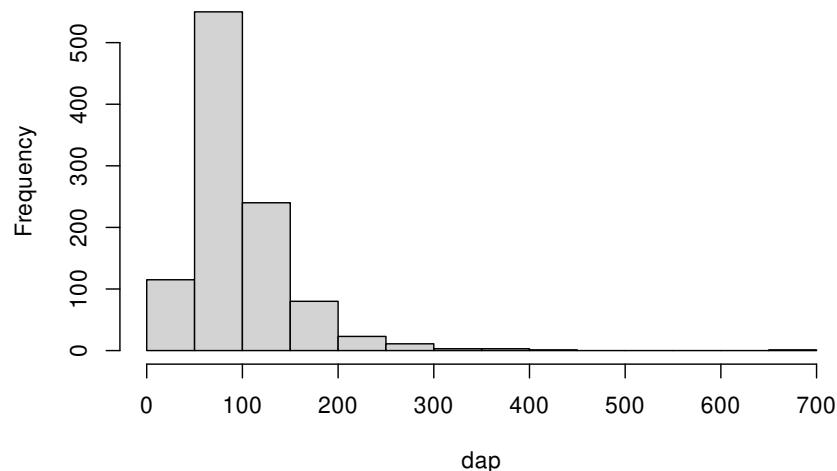
local	parcela	arvore	fuste	cap	h	especie
chauas	1	1	1	210	80	Myrcia sulfiflora
chauas	1	3	1	170	80	Myrcia sulfiflora
chauas	1	4	1	720	70	Syagrus romanzoffianus
chauas	1	5	1	200	80	Tabebuia cassinoides
chauas	1	6	1	750	170	indet.1
chauas	1	7	1	320	80	Myrcia sulfiflora

```
str(caixeta)
```

```
## 'data.frame': 1027 obs. of 7 variables:
## $ local : chr "chauas" "chauas" "chauas" "chauas" ...
## $ parcela: int 1 1 1 1 1 1 1 1 1 ...
## $ arvore : int 1 3 4 5 6 7 8 9 10 10 ...
## $ fuste : int 1 1 1 1 1 1 1 1 1 2 ...
## $ cap : int 210 170 720 200 750 320 480 240 290 310 ...
## $ h : int 80 80 70 80 170 80 160 140 120 120 ...
## $ especie: chr "Myrcia sulfiflora" "Myrcia sulfiflora" "Syagrus romanzoffianus" "Tabebuia cas...
```

```
# vamos calcular o DAP a partir do CAP (circunferência a altura do peito)

# se  $cap = 2\pi \cdot dap/2$  portanto  $dap = cap/\pi$ 
dap <- caixeta$cap / pi
hist(dap) # veja distribuição diamétrica desses dados (é uma típica log-normal)
```

Histogram of dap

```
# adicionando a nova coluna aos dados
caixeta$dap <- dap
head(caixeta, 2)
```

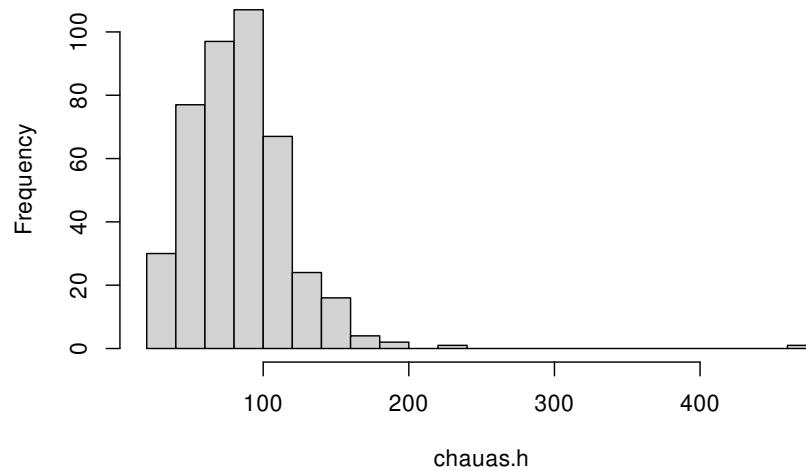
local	parcela	arvore	fuste	cap	h	especie	dap
chauas	1	1	1	210	80	Myrcia sulfiflora	66.84508
chauas	1	3	1	170	80	Myrcia sulfiflora	54.11268

```
# resume localidade
levels(caixeta$local)
```

```
## NULL
```

```
# altura das arvore em cada localidade
chauas.h <- caixeta[caixeta$local == "chauas", "h"]
jureia.h <- caixeta[caixeta$local == "jureia", "h"]
retiro.h <- caixeta[caixeta$local == "retiro", "h"]
# para a localidade chauas
hist(chauas.h, breaks = 20)
```

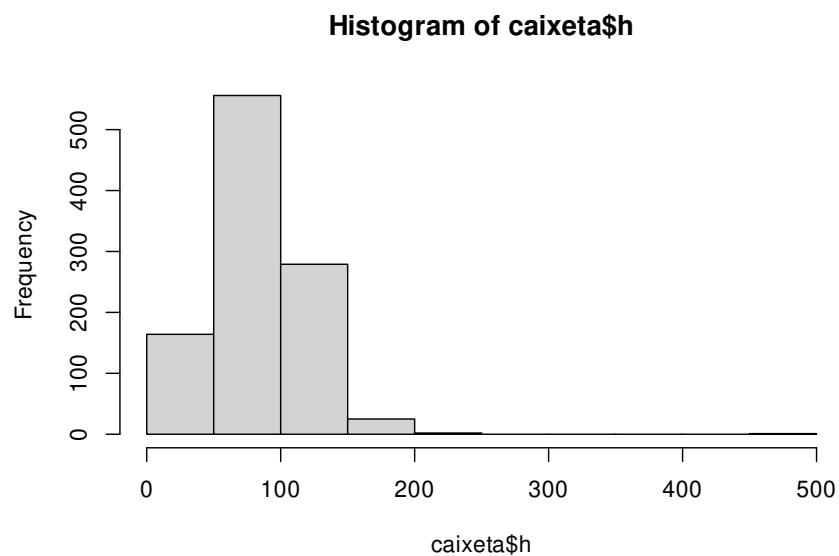
Histogram of chauas.h



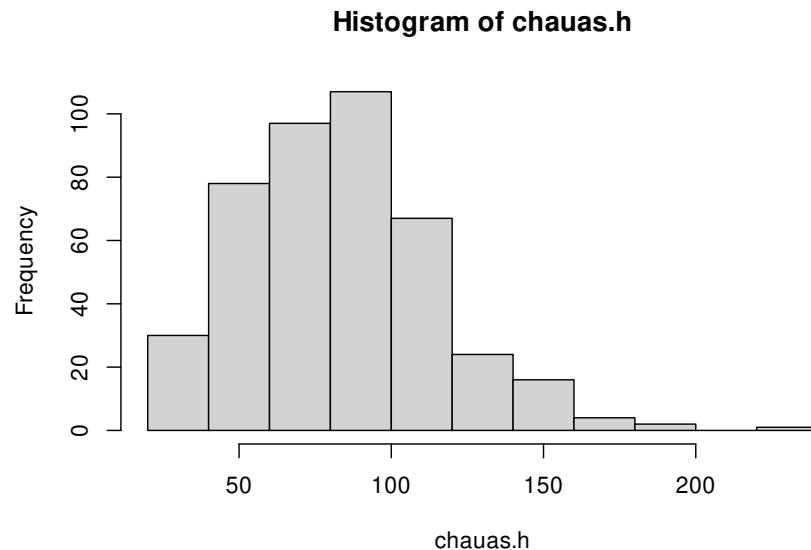
```
# ops tem um valor extremo, que era um erro  
chauas.h[chauas.h > 300]
```

```
## [1] 480
```

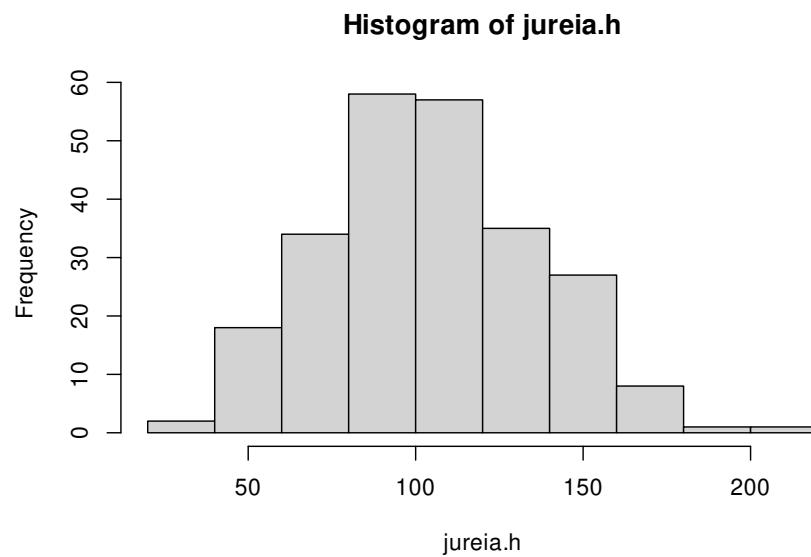
```
# deveria ser 48  
hist(caixeta$h) # um valor extremo de todo o conjunto de dados
```



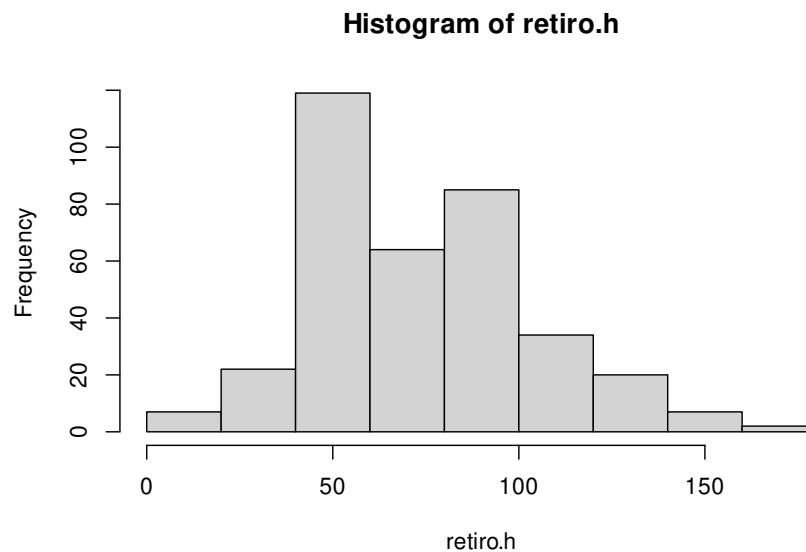
```
chauas.h[chauas.h > 300] <- 48 # corrigimos esse valor  
hist(chauas.h)
```



```
hist(jureia.h)
```



```
hist(retiro.h)
```



```
range(chauas.h) # amplitude de variacao
```

```
## [1] 20 230
```

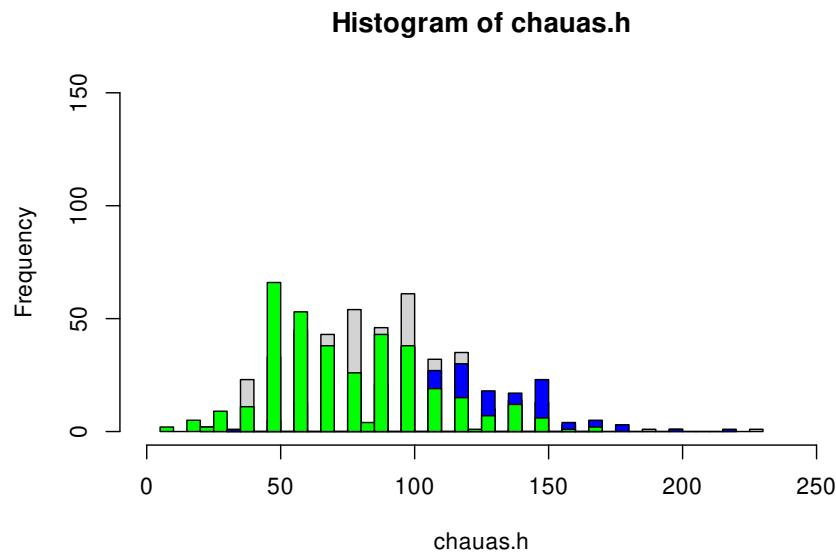
```
range(jureia.h)
```

```
## [1] 30 220
```

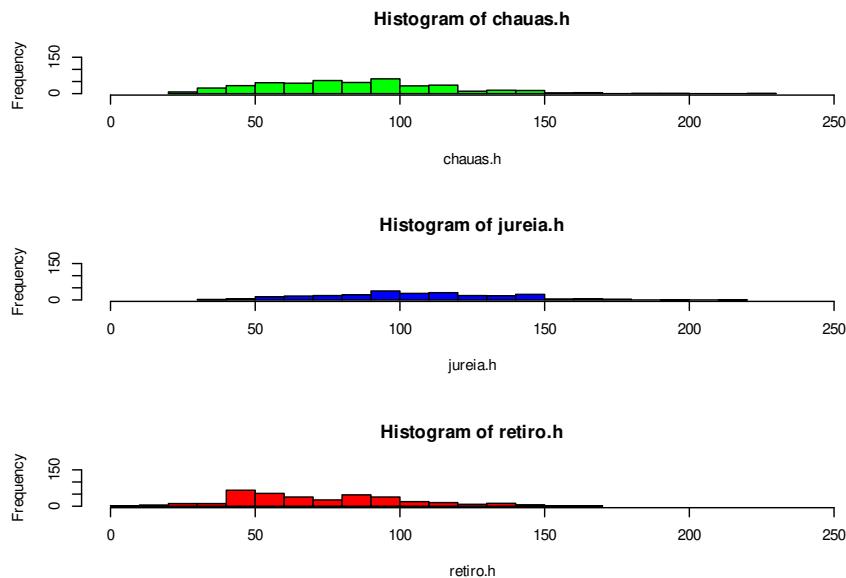
```
range(retiro.h)
```

```
## [1] 5 170
```

```
xl <- c(0, 250) # limitando o grafico aos extremos de todo o conjunto de dados
yl <- c(0, 150)
hist(chauas.h, breaks = 30, xlim = xl, ylim = yl)
hist(jureia.h, add = TRUE, breaks = 30, col = "blue")
hist(retiro.h, add = TRUE, breaks = 30, col = "green")
```



```
# mas seria melhor ver cada distribuicao individualmente por localidade
# dividimos o dispositivo em 3 linhas e uma coluna
par(mfrow = c(3, 1))
hist(chauas.h, breaks = 20, xlim = xl, ylim = yl, col = "green")
hist(jureia.h, breaks = 20, xlim = xl, ylim = yl, col = "blue")
hist(retiro.h, breaks = 20, xlim = xl, ylim = yl, col = "red")
```



```
par(mfrow = c(1, 1)) # retorna o dispositivo
```

12.2 Tabelas de variáveis categóricas

A função `table()` permite contar valores em fatores e vetores e você pode **relembra como usar a função `barplot()`** para gerar gráficos de barra simples:

```
# continuando com os dados de caixeta.csv  
head(caixeta)
```

local	parcela	arvore	fuste	cap	h	especie	dap
chauas	1	1	1	210	80	Myrcia sulfiflora	66.84508
chauas	1	3	1	170	80	Myrcia sulfiflora	54.11268
chauas	1	4	1	720	70	Syagrus romanzoffianus	229.18312
chauas	1	5	1	200	80	Tabebuia cassinoides	63.66198
chauas	1	6	1	750	170	indet.1	238.73241
chauas	1	7	1	320	80	Myrcia sulfiflora	101.85916

```
# tem a coluna especie
```

```
# podemos resumir quantos individuos tem de cada espécie (considerando que cada linha é um in
table(caixeta$especie)
```

Alchornea triplinervia	Andira fraxinifolia	bombacaceae	Cabralea canjerana	Callophyllum
15	4	1		4

```
sort(table(caixeta$especie), decreasing = T)[1:3] # quais são as tres especies mais abundantes
```

Tabebuia cassinoides	Myrcia sulfiflora	Mela 1
698	96	63

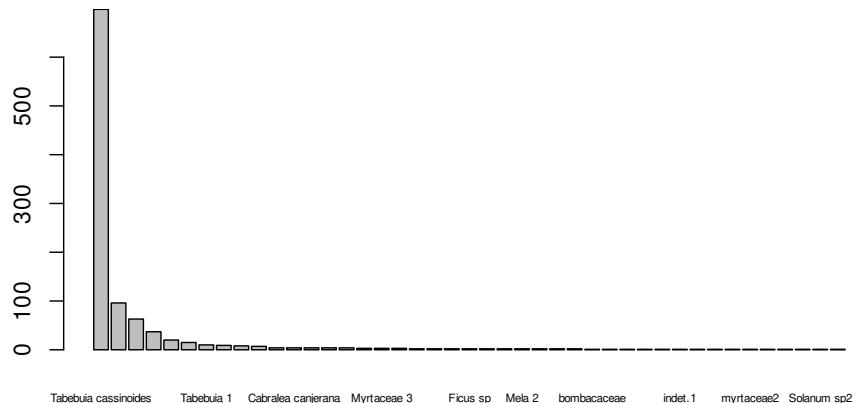
```
table(caixeta$local) # quantas localidades?
```

chauas	jureia	retiro
426	241	360

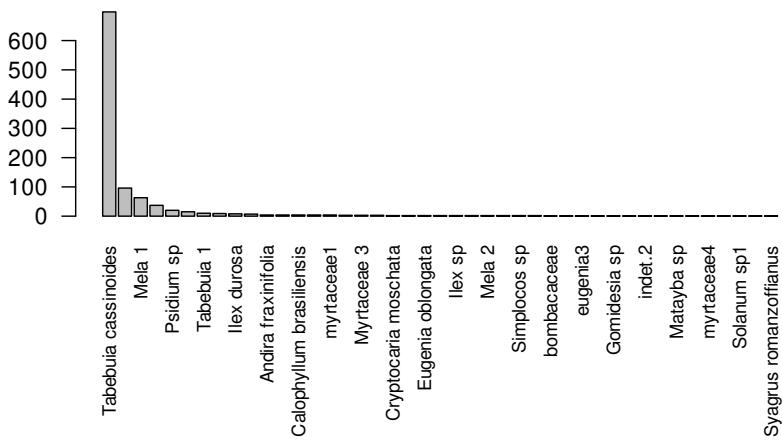
```
## Graficos de barra para representar uma tabela
op <- par(no.readonly = TRUE) # pega parametros gráficos atual
par(mar = c(10, 3, 0, 0)) # mudando as margens
vv <- sort(table(caixeta$especie), decreasing = T)
vv[1:5] # cinco especies mais abundantes
```

Tabebuia cassinoides	Myrcia sulfiflora	Mela 1	jussara	Psidium sp
698	96	63	37	20

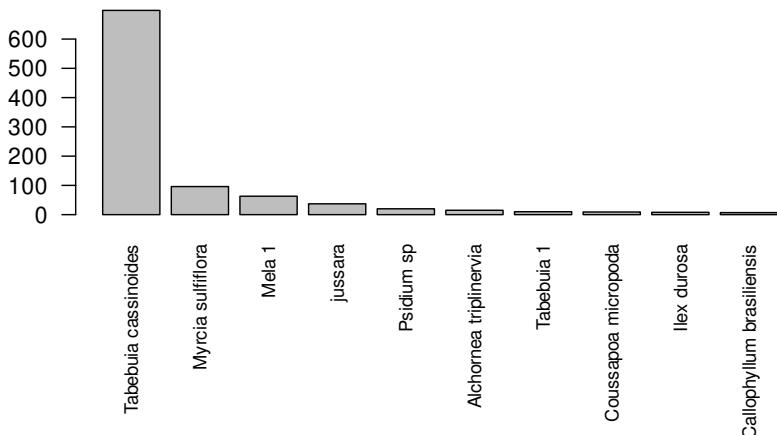
```
# gráfico de barras disso
barplot(vv, cex.names = 0.5)
```



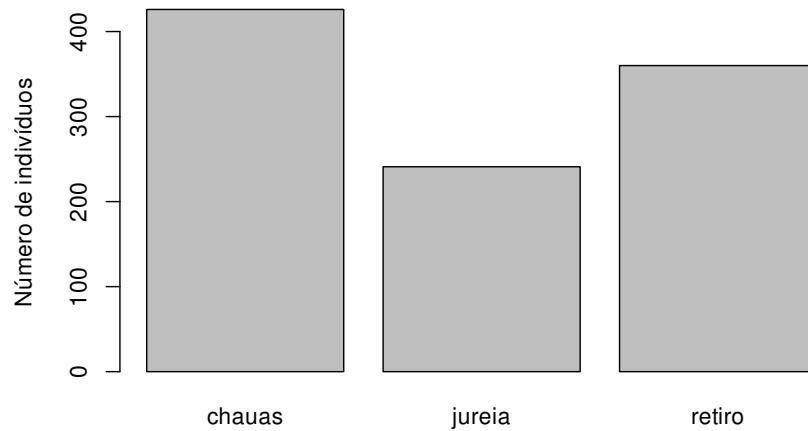
```
par(las = 2, mar = c(10, 5, 5, 1)) # mudando margens e orientacao dos eixos
barplot(sort(table(caixeta$especie), decreasing = T), cex.names = 0.8)
```



```
# muita coisa, pegando apenas as especies mais abundantes
barplot(sort(table(caixeta$especie), decreasing = T)[1:10], cex.names = 0.8)
```



```
# note como Tabebuia cassonoides é muito mais abundante que qualquer outra espécie nessas comunidades  
par(op) # volta aos parametros  
# numero de individuos por localidade  
barplot(table(caixeta$local), ylab = "Número de indivíduos")
```



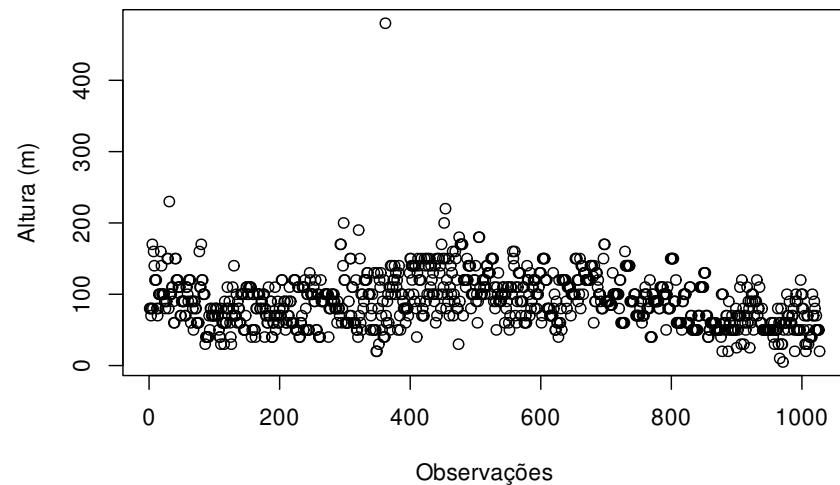
12.2.1 Resumo de gráficos univariados

Além das funções gráficas apresentadas acima, vamos ver aqui as funções `dotchart()` e `stripchart()`, úteis para visualizar dados brutos.

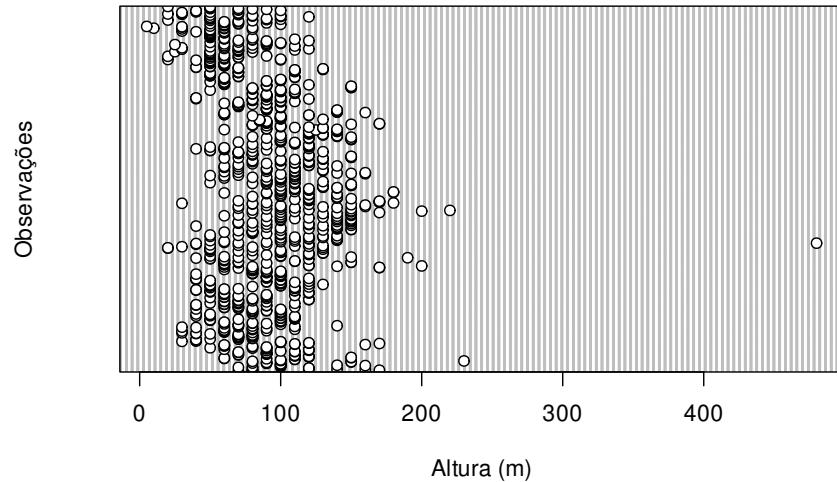
```
# para visualizar dados brutos  
head(caixeta)
```

local	parcela	arvore	fuste	cap	h	especie	dap
chauas	1	1	1	210	80	Myrcia sulfiflora	66.84508
chauas	1	3	1	170	80	Myrcia sulfiflora	54.11268
chauas	1	4	1	720	70	Syagrus romanzoffianus	229.18312
chauas	1	5	1	200	80	Tabebuia cassinoides	63.66198
chauas	1	6	1	750	170	indet.1	238.73241
chauas	1	7	1	320	80	Myrcia sulfiflora	101.85916

```
# plotar a coluna altura (h)
plot(caixeta$h, xlab = "Observações", ylab = "Altura (m)")
```



```
# note que o único valor extremos fica super evidente
# poderíamos usar a função dotchart para isso
dotchart(caixeta$h, ylab = "Observações", xlab = "Altura (m)")
```

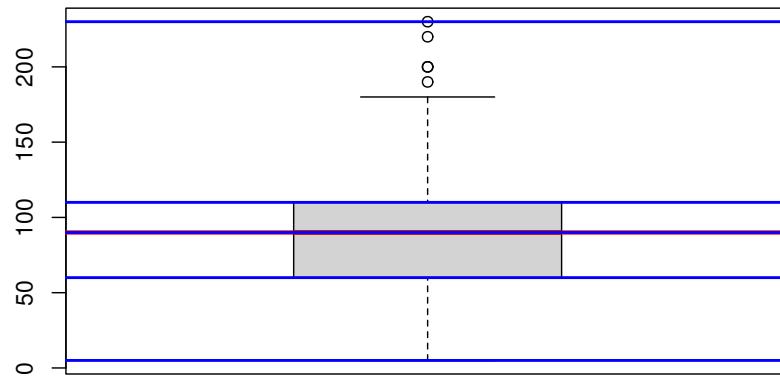


```
# inversão dos eixos..
# vamos corrigir o valor extremo
caixeta[which(caixeta$h > 300), "h"] <- 48

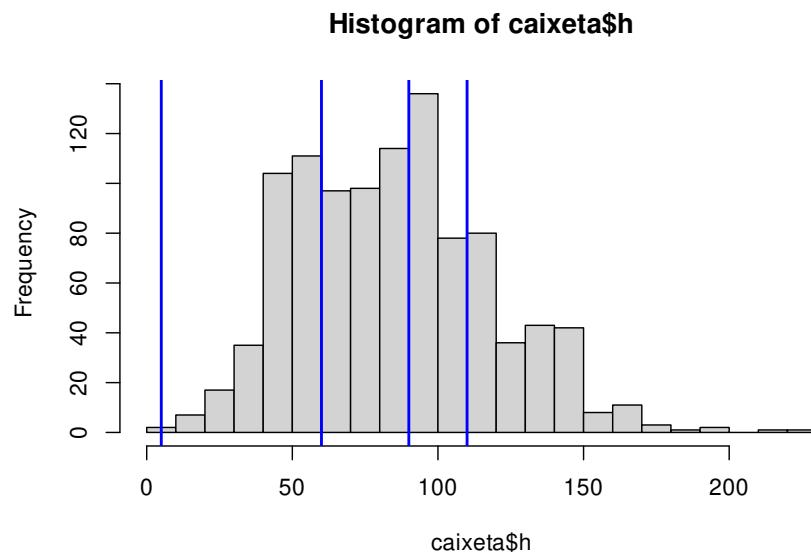
# faz um boxplot disso
boxplot(caixeta$h) # já vimos isso, mas note os pontos isolados dos boxes (caixas), esses são
summary(caixeta$h) # ve os quartis e média
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
5	60	90	89.86173	110	230

```
# plota a mediana
abline(h = median(caixeta$h), col = "red", lwd = 3)
# plota todos os quartis
abline(h = quantile(caixeta$h), col = "blue", lwd = 2)
```



```
# ve em forma de histograma  
hist(caixeta$h, breaks = 20)  
# plota os quartis  
abline(v = quantile(caixeta$h), col = "blue", lwd = 2)
```

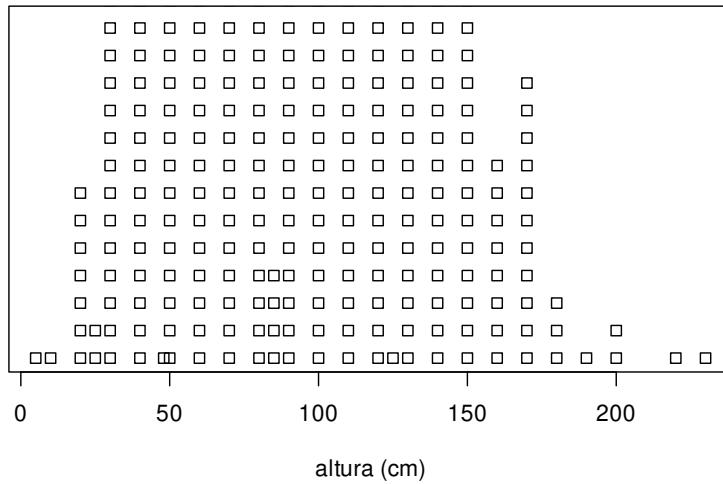


```
dim(caixeta)
```

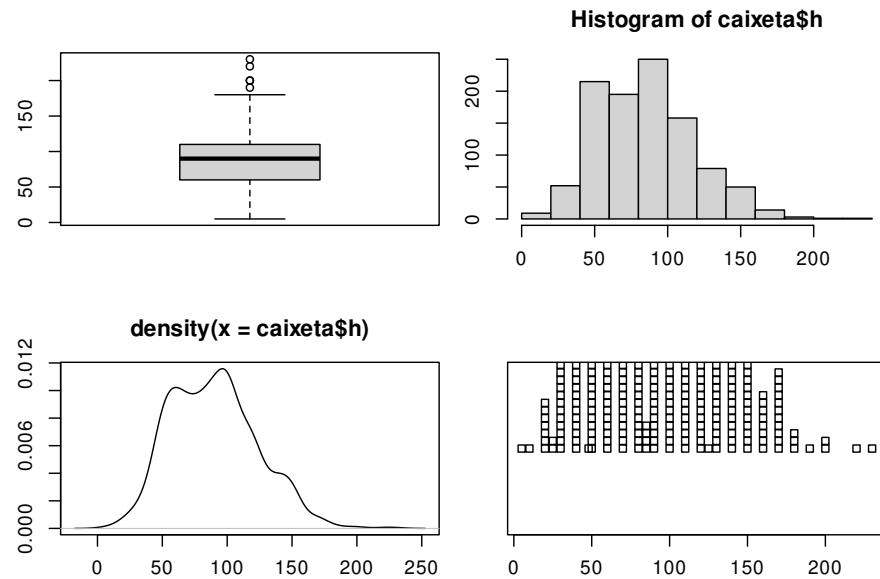
```
## [1] 1027     8
```

```
# ve o histograma na forma de pontos:  
?stripchart
```

```
stripchart(caixeta$h, method = "stack", jitter = 0, offset = 1, ylim = c(0, nrow(caixeta)), xla
```

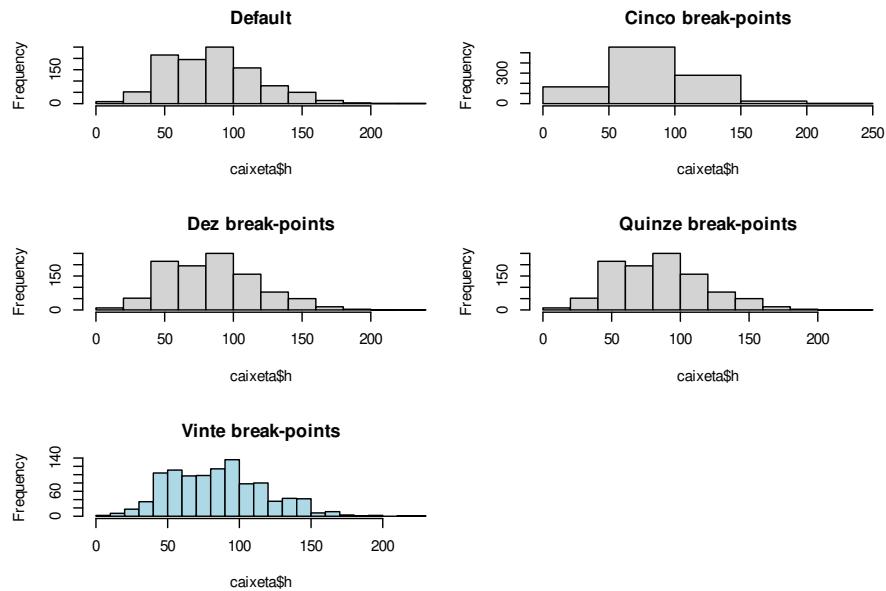


```
## Numa tela só boxplot, histograma, densidade e stripchart
olp <- par(no.readonly = TRUE)
par(mfrow = c(2, 2), mar = c(3, 3, 3, 0))
boxplot(caixeta$h)
hist(caixeta$h)
plot(stats::density(caixeta$h))
stripchart(caixeta$h, method = "stack")
```



```
par(olp) # resgata parametros graficos originais

## Histograma com diferentes larguras de barras
par(mar = c(5, 4, 3, 1), mfrow = c(3, 2))
hist(caixeta$h, main = "Default")
hist(caixeta$h, breaks = 5, main = "Cinco break-points")
hist(caixeta$h, breaks = 10, main = "Dez break-points")
hist(caixeta$h, breaks = 15, main = "Quinze break-points")
hist(caixeta$h, breaks = 20, main = "Vinte break-points", col = "lightblue")
par(olp)
```



12.3 As variáveis têm distribuição normal?

A função `density()` que juntamente com `hist()`, que você já conhece, permite visualizar a densidade probabilística de uma variável numérica, ou seja descreve a distribuição de probabilidade⁵, isto é, a chance de uma variável assumir um valor ao longo de um espaço (densidade) de valores.

A função `dnorm()` permite obter a densidade probabilística de uma distribuição normal teórica, para a mesma média e mesmo desvio padrão dos teus dados. Com isso você pode visualizar a distribuição dos seus dados e sobrepor a isso como seria a distribuição se os seus dados fossem normais.

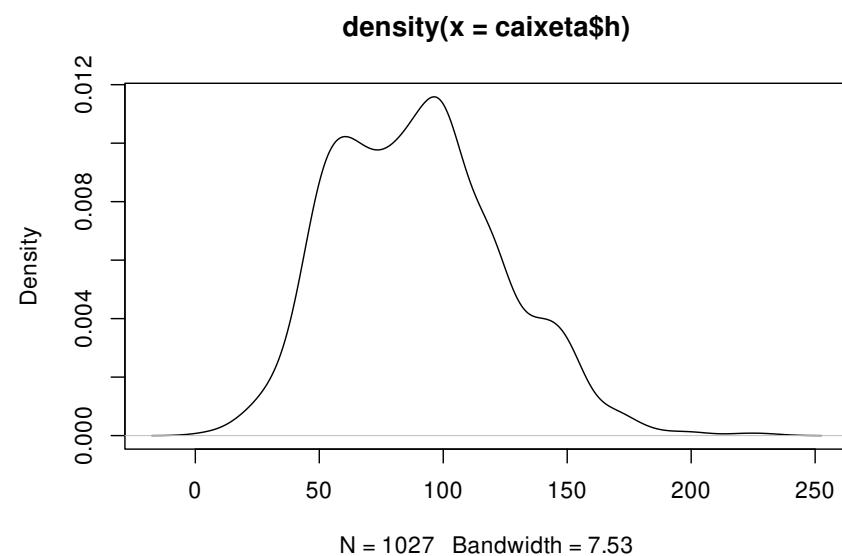
⁵https://pt.wikipedia.org/wiki/Distribui%C3%A7%C3%A3o_de_probabilidade

```
par(olp)
```

```
?density # veja o help disso
```

```
# plota a densidade probabilistica = a curva da probabilidade da variável assumir certos valores
```

```
plot(stats::density(caixeta$h))
```



```
## Histograma com área = 1 e density probabilistica sobreposta (argumento prob=TRUE, muda o eixo)
```

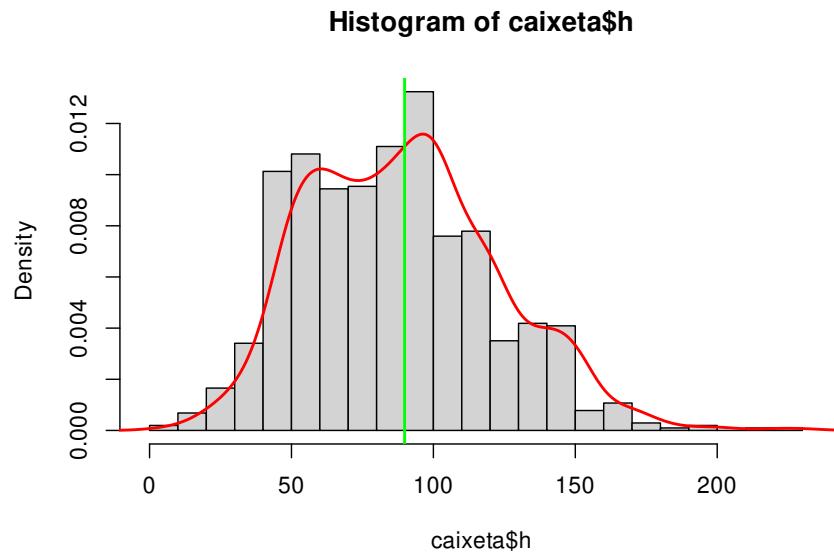
```
hist(caixeta$h, prob = T, breaks = 30, xlim = c(-1, max(caixeta$h) + 5))
```

```
# adiciona a linha da densidade
```

```
lines(stats::density(caixeta$h), col = "red", lwd = 2)
```

```
# adiciona a média
```

```
abline(v = mean(caixeta$h), col = "green", lwd = 2, lty = "solid")
```



```
# note que na média a densidade probabilística é maior que nas caudas da distribuição
# vamos adicionar sobre nossa distribuição REAL a densidade probabilística para uma distribuição
```

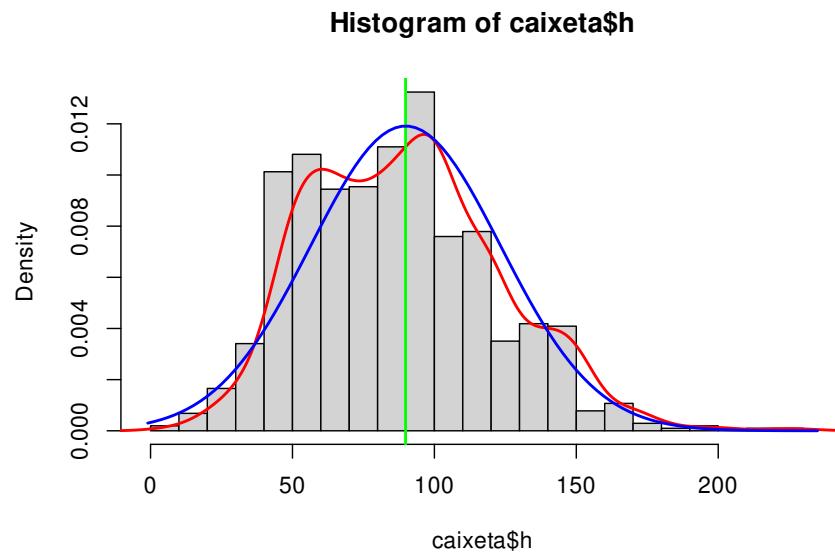
```
## Adicionando uma curva da normal aos graficos
?dnorm # veja o help dessa função e suas variantes. veremos isso melhor abaixo
```

```
# pega a densidade probabilística de uma distribuição normal teórica, para quantis de seu intervalo
dnorm(seq(0, 1, by = 0.25), mean = mean(caixeta$h), sd = sd(caixeta$h)) # esses são os valores
```

```
## [1] 0.0003251756 0.0003317476 0.0003384336 0.0003452351 0.0003521537
```

```
?curve # veja que curve depende de uma função, ela traça a curva de uma f(x), num intervalo es-
```

```
hist(caixeta$h, prob = T, breaks = 30, xlim = c(-1, max(caixeta$h) + 5))
# adiciona a linha da densidade
lines(stats::density(caixeta$h), col = "red", lwd = 2)
# adiciona a média
abline(v = mean(caixeta$h), col = "green", lwd = 2, lty = "solid")
# combinamos as coisas e adicionamos a distribuição
# teórica sobre os nossos dados
## Usamos a função curve,
curve(expr = dnorm(x, mean = mean(caixeta$h), sd = sd(caixeta$h)), add = T, col = "blue", lwd = 2)
```



```
# note que os dados neste caso seguem bem uma curva normal.
```

```
# Portanto, mesmo sem fazer um teste, essa figura sugere que os dados de altura do caixetal é
```

As funções `qqnorm()` e `qqline()` permitem visualizar rapidamente se

uma variável qualquer segue uma distribuição normal, ao compara os valores dos quantis empíricos (observados), com valores dos quantis teóricos (i.e. esperados por uma distribuição normal). A função `rnorm()` gera um conjunto de dados aleatórios que tem distribuição normal.

```
# Teste de normalidade
#####
## Exemplo para o qqplot
#####

# vamos simular valores
?rnorm # função que gera valores aleatórios que seguem uma distribuição normal
```

```
## Sorteio de 100 valores de uma normal com media=30 e desvio-padrão=3
zz <- rnorm(100, 30, 3)
mean(zz)
```

```
## [1] 30.03516
```

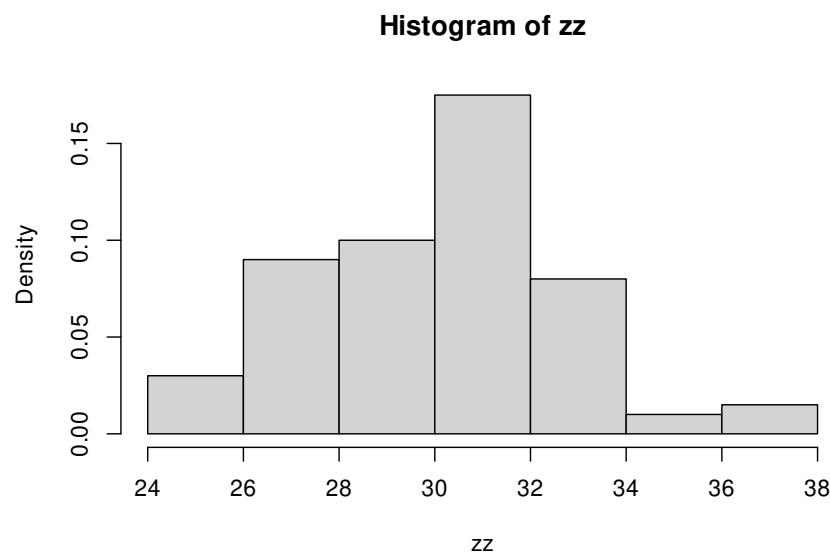
```
sd(zz)
```

```
## [1] 2.666064
```

```
length(zz)
```

```
## [1] 100
```

```
hist(zz, prob = T)
```



```
## Valores arredondados para 2 casa, e ordenados
x <- sort(round(rnorm(100, 30, 3), 2))
## Inspecionando os 5 primeiros e ultimos valores
x[1:5]
```

```
## [1] 24.18 24.33 24.33 24.72 25.15
```

```
x[95:100]
```

```
## [1] 35.50 35.69 35.73 35.78 36.48 38.45
```

```
## Calculo do percentil de cada valor
# relembrar a função order (ela retorna os índices dos valores ordenados)
order(x) # veja que os índices estão sequenciais, porque geramos um vetor já pre-ordenado
```

```
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
## [19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
## [37] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
## [55] 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
## [73] 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
## [91] 91 92 93 94 95 96 97 98 99 100
```

```
# calculamos o percentil de cada valor, que é uma medida que indica o valor abaixo do qual uma
px <- order(x) / 100
px
```

```
## [1] 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.10 0.11 0.12 0.13 0.14 0.15
## [16] 0.16 0.17 0.18 0.19 0.20 0.21 0.22 0.23 0.24 0.25 0.26 0.27 0.28 0.29 0.30
## [31] 0.31 0.32 0.33 0.34 0.35 0.36 0.37 0.38 0.39 0.40 0.41 0.42 0.43 0.44 0.45
## [46] 0.46 0.47 0.48 0.49 0.50 0.51 0.52 0.53 0.54 0.55 0.56 0.57 0.58 0.59 0.60
## [61] 0.61 0.62 0.63 0.64 0.65 0.66 0.67 0.68 0.69 0.70 0.71 0.72 0.73 0.74 0.75
## [76] 0.76 0.77 0.78 0.79 0.80 0.81 0.82 0.83 0.84 0.85 0.86 0.87 0.88 0.89 0.90
## [91] 0.91 0.92 0.93 0.94 0.95 0.96 0.97 0.98 0.99 1.00
```

```
# vejamos a correspondência
# quantos valores são menores que o percentil 0.2 (ou 20%)?
sum(px < 0.2) # obviamente 19 se temos apenas 100 valores no nosso vetor
```

```
## [1] 19
```

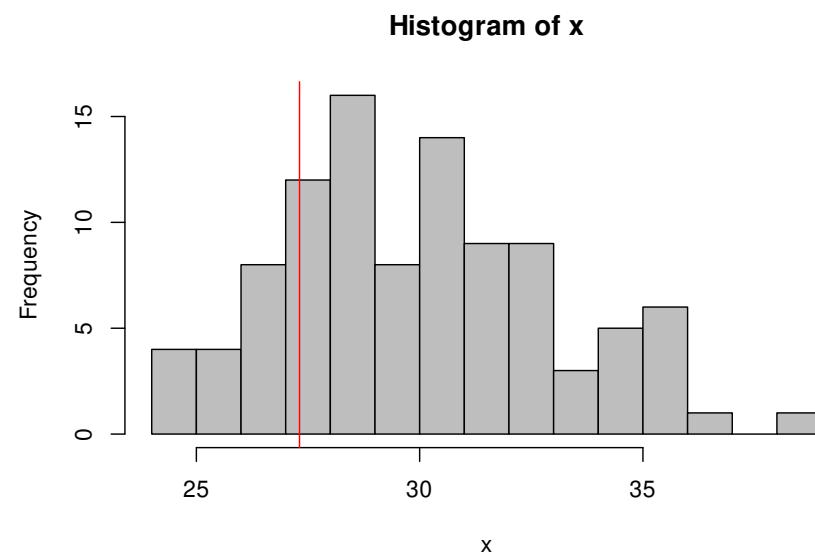
```
# quais valores são esses  
x[order(x)][px <= 0.2]
```

```
## [1] 24.18 24.33 24.33 24.72 25.15 25.20 25.65 26.00 26.17 26.26 26.51 26.60  
## [13] 26.62 26.79 26.93 26.96 27.07 27.28 27.31 27.31
```

```
# qual o valor do percentil  
x[order(x)][px == 0.2]
```

```
## [1] 27.31
```

```
hist(x, breaks = 20, col = "gray")  
abline(v = x[order(x)][px == 0.2], col = "red")
```



```
# as barras com valores menores ou iguais que o do percentil (linha vermelha), totalizam 20% da amostra

# com os percentis dos valores do dado original, podemos pegar a densidade probabilistica esperada
q.norm.x <- qnorm(px, mean = mean(x), sd = sd(x))

## Juntando valores originais, os percentis e os valores esperados em um dataframe, para facilitar a visualizacao
qq.plot.x <- data.frame(x = x, percentil = px, q.norm = q.norm.x)
qq.plot.x[1:5, ]
```

x	percentil	q.norm
24.18	0.01	22.79771
24.33	0.02	23.63630
24.33	0.03	24.16836
24.72	0.04	24.56860
25.15	0.05	24.89417

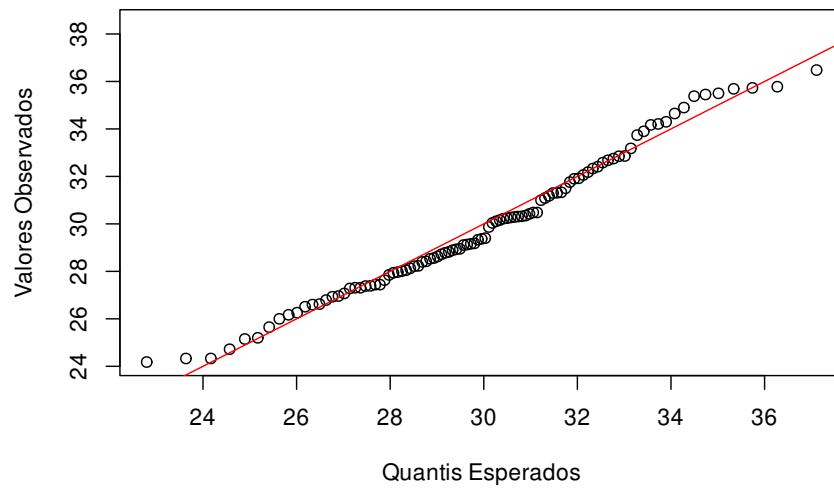
```
qq.plot.x[95:100, ]
```

	x	percentil	q.norm
95	35.50	0.95	35.01423
96	35.69	0.96	35.33980
97	35.73	0.97	35.74004
98	35.78	0.98	36.27210
99	36.48	0.99	37.11069
100	38.45	1.00	Inf

```
head(qq.plot.x)
```

x	percentil	q.norm
24.18	0.01	22.79771
24.33	0.02	23.63630
24.33	0.03	24.16836
24.72	0.04	24.56860
25.15	0.05	24.89417
25.20	0.06	25.17129

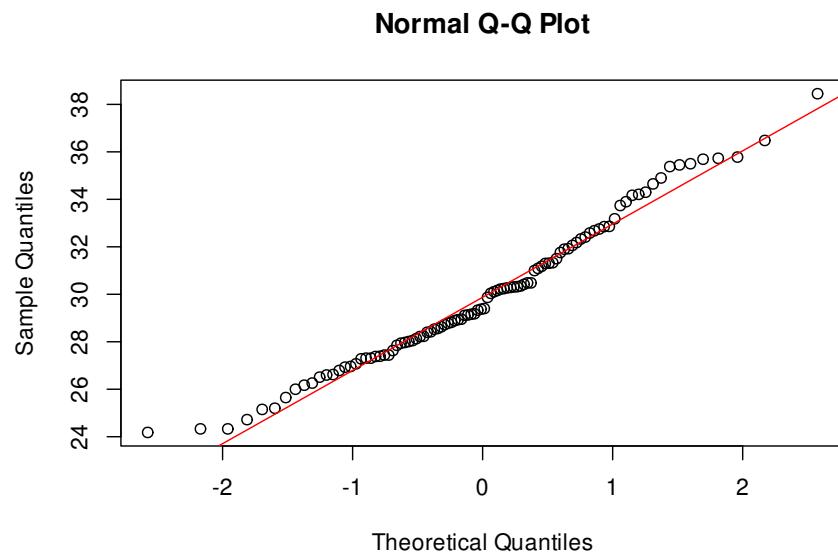
```
## com isso eu posso comparar meus valores observados com os valores esperados se a distribuiçao for normal
plot(x ~ q.norm, data = qq.plot.x, xlab = "Quantis Esperados", ylab = "Valores Observados")
abline(0, 1, col = "red") # relacão esperada, caso os dados venham de uma populacão normal
```



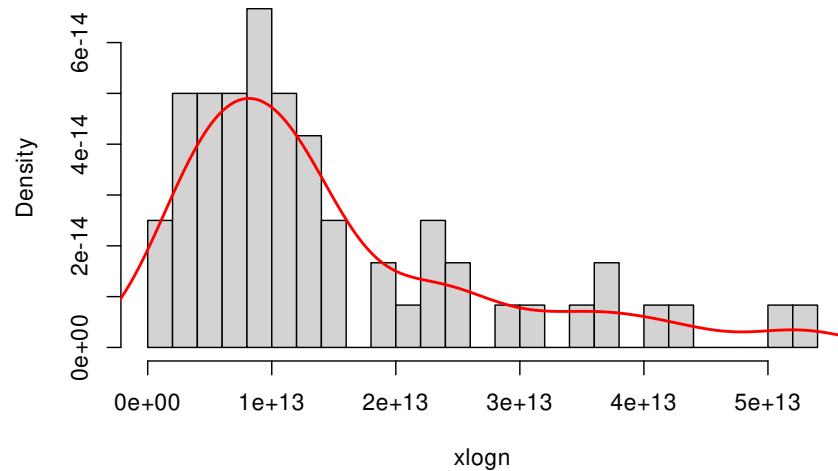
```
# note que a correlaçao é fortíssima, porque os usados no exemplo eram de fato normais, portanto
```

```
## A função qqnorm já faz isto de uma vez para você:
?qqnorm # veja o help
```

```
qqnorm(x)
qqline(x, col = "red")
```

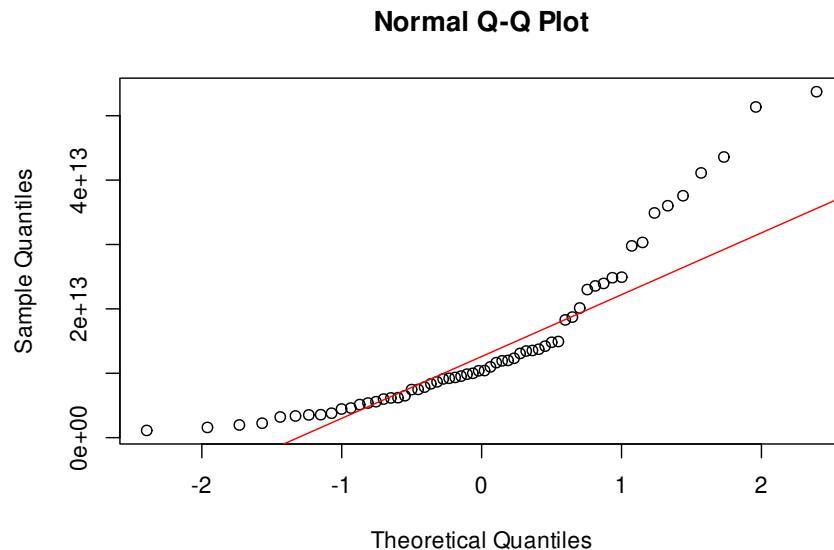


```
# suponha um dado não normal
# cria um exemplo lognormal (não é uma distribuição normal)
xlogn <- rlnorm(nrow(avesc), meanlog = 30, sdlog = 1)
hist(xlogn, prob = T, breaks = 20)
lines(stats::density(xlogn), col = "red", lwd = 2)
```

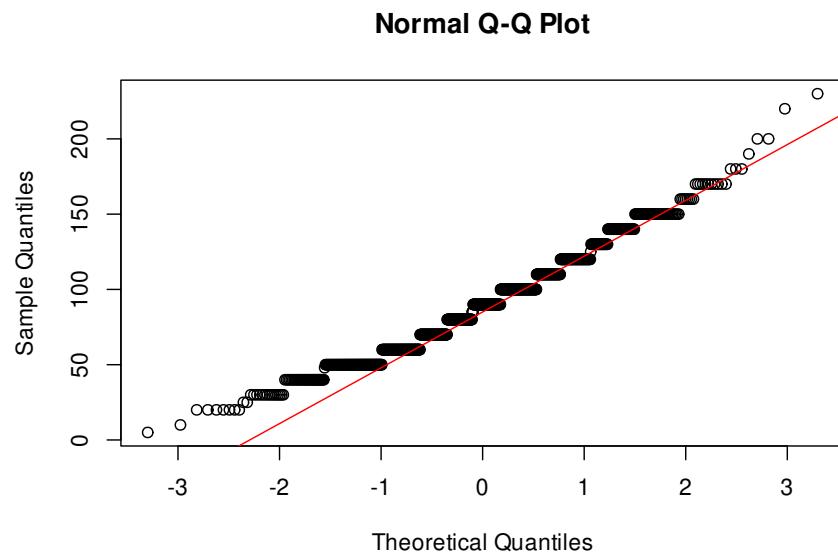
Histogram of xlogn

```
# não é uma curva normal, certo?
```

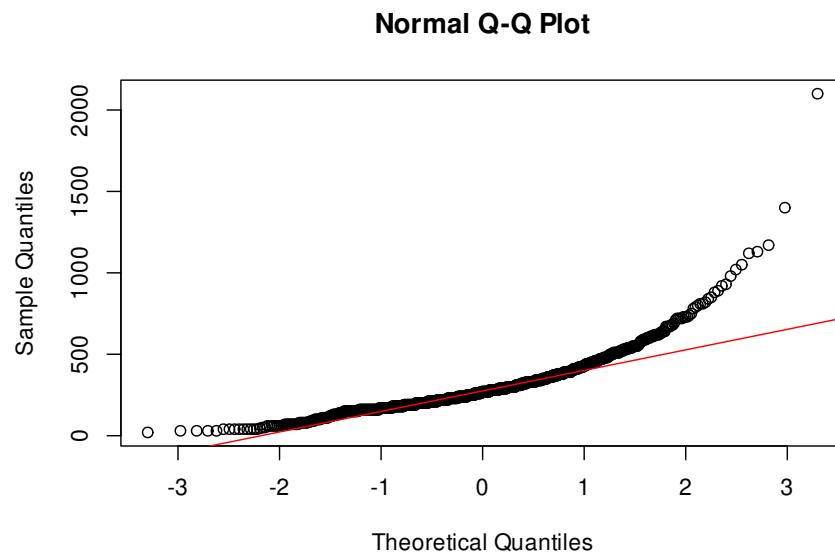
```
# mostra o QQ Plot nessa situação (veja como os pontos saem da linha)
qqnorm(xlogn)
qqline(xlogn, col = "red")
```



```
# entao vamos ver em dados reais
# altura, já vimos antes, tem distribuição bem normal
qqnorm(caixeta$h)
qqline(caixeta$h, col = "red")
```



```
# cap por outro lado, é mais log normal
qqnorm(caixeta$cap)
qqline(caixeta$cap, col = "red")
```



12.4 Para saber mais:

- Exercicio AED Univariadas⁶.

⁶http://www.botanicaamazonica.wiki.br/labotam/doku.php?id=disciplinas:bot89_2013:exercicio07

13

AED de bivariadas



Existe alguma relação entre as variáveis? A relação é linear? Há colinearidade? Ou seja, diferentes variáveis tem o mesmo padrão?

13.1 Dados do tutorial

Vamos importar novamente os conjuntos de dados de avistamento de aves do cerrado¹ (utilizado no capítulo 11) e de parcelas em caixetas² (utilizado no capítulo 7):

```
## Lendo a planilha com read.table
avesc <- read.table("aves_cerrado.csv", row.names = 1, header = T, sep = ";", dec = ",", as.is = TRUE)

caixeta <- read.csv("caixeta.csv") ## arquivo caixeta.csv deve estar no diretorio de trabalho
# note que mantemos todos os argumentos padrão (veja o formato do arquivo caixeta)
```

¹https://github.com/LABOTAM/IntroR/blob/main/dados/aves_cerrado.csv

²<https://github.com/LABOTAM/IntroR/blob/main/dados/caixeta.csv>

13.2 Fatores e contagens

Já vimos a função `table()` para contar valores em fatores e vetores em casos de **univariados**. Podemos usar a mesma função para gerar tabelas de contingência entre dois ou mais fatores.

```
## Numero de fustes de cada especie por local
tb <- table(caixeta$especie, caixeta$local)
class(tb)
```

```
## [1] "table"
```

```
tb
```

	chauas	jureia	retiro
Alchornea triplinervia	0	3	12
Andira fraxinifolia	0	4	0
bombacaceae	0	1	0
Cabralea canjerana	0	4	0
Callophyllum brasiliensis	7	0	0
Callophyllum brasiliensis	0	4	0
Cecropia sp	0	0	1
Coussapoa macrocarpa	0	3	0
Coussapoa micropoda	2	0	7
Cryptocaria moschata	0	2	0
Cyathea sp	0	0	2
Eugenia oblongata	0	0	2
eugenia3	0	1	0
fabaceae1	0	1	0
Ficus sp	0	2	0
Gomidesia sp	0	1	0
Ilex duxosa	0	8	0
Ilex sp	0	0	2
indet.1	1	0	0
indet.2	1	0	0
indet.3	1	0	0
Inga sp	0	4	0
Jacaranda puberula	0	2	0
jussara	0	37	0
Matayba sp	0	1	0
Mela 1	0	0	63
Mela 2	0	0	2
Myrcia sulfiflora	96	0	0
Myrtaceae 3	0	0	3
myrtaceae1	4	0	0
myrtaceae2	1	0	0
myrtaceae4	1	0	0
Pera glabrata	0	1	0
Persea sp	0	3	0
Pisonia sp	0	2	0
Psidium sp	3	17	0
Simplocos sp	0	2	0
Solanum sp1	0	0	1
Solanum sp2	0	0	1
Syagrus romanzoffianus	1	0	0
Tabebuia 1	0	0	10
Tabebuia cassinoides	306	138	254
Tibouchina nutticeps	2	0	0

```
# convertemos num data.frame
tb <- as.data.frame.matrix(tb)
class(tb)
```

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

```
# calculo o total de individuos por especie
total <- apply(tb, 1, sum)
total
```

## Alchornea triplinervia	Andira fraxinifolia	bombacaceae
## 15	4	1
## Cabralea canjerana	Calophyllum brasiliensis	Calophyllum brasiliensis
## 4	7	4
## Cecropia sp	Coussapoa macrocarpa	Coussapoa micropoda
## 1	3	9
## Cryptocaria moschata	Cyathea sp	Eugenia oblongata
## 2	2	2
## eugenia3	fabaceae1	Ficus sp
## 1	1	2
## Gomidesia sp	Ilex duxosa	Ilex sp
## 1	8	2
## indet.1	indet.2	indet.3
## 1	1	1
## Inga sp	Jacaranda puberula	jussara
## 4	2	37
## Matayba sp	Mela 1	Mela 2
## 1	63	2
## Myrcia sulfiflora	Myrtaceae 3	myrtaceae1
## 96	3	4
## myrtaceae2	myrtaceae4	Pera glabrata
## 1	1	1
## Persea sp	Pisonia sp	Psidium sp

```
##          3          2          20
##      Simplocos sp      Solanum sp1      Solanum sp2
##          2          1          1
##      Syagrus romanzoffianus      Tabebuia 1      Tabebuia cassinoides
##          1          10         698
##      Tibouchina nutticeps
##          2
```

```
# ordeno minha tabela orginal pelo total em ordem decrescente de abundância
tb <- tb[order(total, decreasing = T), ]
```

```
head(tb)
```

	chauas	jureia	retiro
Tabebuia cassinoides	306	138	254
Myrcia sulfiflora	96	0	0
Mela 1	0	0	63
jussara	0	37	0
Psidium sp	3	17	0
Alchornea triplinervia	0	3	12

```
# se eu quiser uma tabela de presença e ausência
# bastaria substituir os valores>0 por 1
tb[tb > 0] <- 1
```

```
head(tb)
```

	chauas	jureia	retiro
Tabebuia cassinoides	1	1	1
Myrcia sulfiflora	1	0	0
Mela 1	0	0	1
jussara	0	1	0
Psidium sp	1	1	0
Alchornea triplinervia	0	1	1

```
# assim, agora eu posso saber quantas especie por localidade
apply(tb, 2, sum)
```

```
## chauas jureia retiro
##      13      22      13
```

A função `xtabs()` tabula dados de frequência.

```
## xtabs: tabulacao de dados de frequencia
## Vamos usar Dataframe dos sobrevidentes dos sobrevidentes e mortos do Titanic
?Titanic # veja o que são esses dados
```

```
data("Titanic") # puxamos esse dado
class(Titanic)
```

```
## [1] "table"
```

```
tit <- as.data.frame(Titanic) # converte em data.frame
head(tit)
```

Class	Sex	Age	Survived	Freq
1st	Male	Child	No	0
2nd	Male	Child	No	0
3rd	Male	Child	No	35
Crew	Male	Child	No	0
1st	Female	Child	No	0
2nd	Female	Child	No	0

```
# classe de passageiros  
names(tit)
```

```
## [1] "Class"      "Sex"       "Age"        "Survived"   "Freq"
```

```
str(tit)
```

```
## 'data.frame':    32 obs. of  5 variables:  
## $ Class  : Factor w/ 4 levels "1st","2nd","3rd",...: 1 2 3 4 1 2 3 4 1 2 ...  
## $ Sex    : Factor w/ 2 levels "Male","Female": 1 1 1 1 2 2 2 2 1 1 ...  
## $ Age    : Factor w/ 2 levels "Child","Adult": 1 1 1 1 1 1 1 1 2 2 ...  
## $ Survived: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...  
## $ Freq   : num  0 0 35 0 0 17 0 118 154 ...
```

```
## Quanto sobreviventes por sexo?
```

```
## Precisamos da funcao xtabs  
?xtabs # veja o help dessa função:
```

```
xtabs(Freq ~ Sex + Survived, data = tit)
```

	No	Yes
Male	1364	367
Female	126	344

```
# em porcentagem
tb <- xtabs(Freq ~ Sex + Survived, data = tit)
```

```
prop.table(tb, margin = 1)
```

	No	Yes
Male	0.7879838	0.2120162
Female	0.2680851	0.7319149

```
# ou, se preferir
round(prop.table(tb, margin = 1) * 100)
```

	No	Yes
Male	79	21
Female	27	73

```
# Quanto sobrevidentes por classe de viagem?
xtabs(Freq ~ Class + Survived, data = tit)
```

	No	Yes
1st	122	203
2nd	167	118
3rd	528	178
Crew	673	212

```
# note que na primeira classe 203 sobreviveram  
# eu poderia ter perguntado isso assim:  
sum(tit[tit$Class == "1st" & tit$Survived == "Yes", "Freq"])
```

```
## [1] 203
```

```
# ou seja, a funcao xtabs calculou a soma da frequencia  
# porcentagem  
prop.table(xtabs(Freq ~ Class + Survived, data = tit), margin = 1)
```

	No	Yes
1st	0.3753846	0.6246154
2nd	0.5859649	0.4140351
3rd	0.7478754	0.2521246
Crew	0.7604520	0.2395480

```
## E para combinacoes de mais de duas variaveis  
tb2 <- xtabs(Freq ~ Class + Survived + Sex, data = tit)
```

```
tb2 # veja o resultado e observe duas virgulas
```

Class	Survived	Sex	Freq
1st	No	Male	118
2nd	No	Male	154
3rd	No	Male	422
Crew	No	Male	670
1st	Yes	Male	62
2nd	Yes	Male	25
3rd	Yes	Male	88
Crew	Yes	Male	192
1st	No	Female	4
2nd	No	Female	13
3rd	No	Female	106
Crew	No	Female	3
1st	Yes	Female	141
2nd	Yes	Female	93
3rd	Yes	Female	90
Crew	Yes	Female	20

```
tb2[, , 1] # para Female
```

	No	Yes
1st	118	62
2nd	154	25
3rd	422	88
Crew	670	192

```
tb2[, , 2] # note que não vimos isso antes, tb2, neste caso é um array, que um objeto que pode
```

	No	Yes
1st	4	141
2nd	13	93
3rd	106	90
Crew	3	20

13.3 Variável numérica vs. fator

A função `tapply()` faz uso de uma função sobre sobre um vetor numérico para cada categoria de um fator. A função `aggregate()` faz o mesmo, mas permite múltiplos fatores e retorna um `data.frame`.

```
## tapply: resumo de uma variavel numerica, separada por niveis de um ou mais fatores  
?tapply # veja o help dessa função
```

```
head(avesc) # se nao tem isso, importe novamente o arquivo aves_cerrado
```

	fisionomia	urubu	carcara	seriema
Ce1	Ce	5	18	6
Ce2	Ce	7	7	6
Ce3	Ce	5	14	4
Ce4	Ce	3	12	5
Ce5	Ce	4	16	4
Ce6	Ce	NA	20	NA

```
# numero de individuos de carcara por fisionomia  
tapply(avesc$carcara, avesc$fisionomia, sum)
```

```
## CC ce Ce CL  
## 212 14 291 NA
```

```
# numero de individuos de urubo por fisionomia  
tapply(avesc$urubu, avesc$fisionomia, sum)
```

```
## CC ce Ce CL  
## 299 13 NA 298
```

```
# numero médio de seriemas por localidade+fisionomia  
tapply(avesc$seriema, avesc$fisionomia, mean)
```

```
## CC ce Ce CL  
## NA 4.0 NA 5.6
```

```
## "Tabelas dinamicas": função aggregate  
## Criar data.frame com altura media dos fustes por especie e por local
```

```
?aggregate # veja o help dessa função
```

```
names(caixeta)
```

```
## [1] "local"  "parcela" "arvore"  "fuste"   "cap"     "h"       "especie"
```

```
# circunferencia máxima por especie  
ob1 <- aggregate(caixeta$cap, by = list(especie = caixeta$especie), FUN = max)  
class(ob1) # obtenho um data frame
```

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

```
head(ob1)
```

especie	x
Alchornea triplinervia	840
Andira fraxinifolia	340
bombacaceae	380
Cabralea canjerana	720
Calophyllum brasiliensis	1130
Calophyllum brasiliensis	2100

```
# neste caso também poderia fazer assim
ob2 <- tapply(caixeta$h, caixeta$especie, max)
class(ob2) # mas neste caso nos temos um array (um vetor unidimensional)
```

```
## [1] "array"
```

```
ob2[1:10]
```

```
##   Alchornea triplinervia    Andira fraxinifolia      bombacaceae
##                 140                  90                  150
##   Cabralea canjerana Calophyllum brasiliensis Calophyllum brasiliensis
##                 150                  200                  160
##   Cecropia sp     Coussapoa macrocarpa   Coussapoa micropoda
##                 70                  100                  110
##   Cryptocaria moschata
##                 140
```

```
# mas se eu quiser por localidade e por especie, preciso usar aggregate
caixeta.alt <- aggregate(caixeta$h, by = list(local = caixeta$local, especie = caixeta$especie),
head(caixeta.alt)
```

local	especie	x
jureia	Alchornea triplinervia	140
retiro	Alchornea triplinervia	100
jureia	Andira fraxinifolia	90
jureia	bombacaceae	150
jureia	Cabralea canjerana	150
chauas	Callophyllum brasiliensis	200

```
## Vamos calcular a área basal (soma da área de todo os fustes)
## calculando a área basal de cada fuste, considerando o fuste um círculo perfeito, poderíamos
caixeta$ab <- caixeta$cap^2 / 4 * pi
## e agora criamos a planilha, com aggregate, somando as áreas basais dos fustes
caixeta.2 <- aggregate(caixeta$ab, by = list(local = caixeta$local, parcela = caixeta$parcela,
class(caixeta.2)
```

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

```
head(caixeta.2)
```

local	parcela	especie	x
retiro	1	Alchornea triplinervia	53092.92
jureia	2	Alchornea triplinervia	554176.94
retiro	2	Alchornea triplinervia	90949.11
retiro	3	Alchornea triplinervia	230121.66
jureia	4	Alchornea triplinervia	292246.66
jureia	5	Alchornea triplinervia	273397.10

13.4 Variável numérica vs. numérica

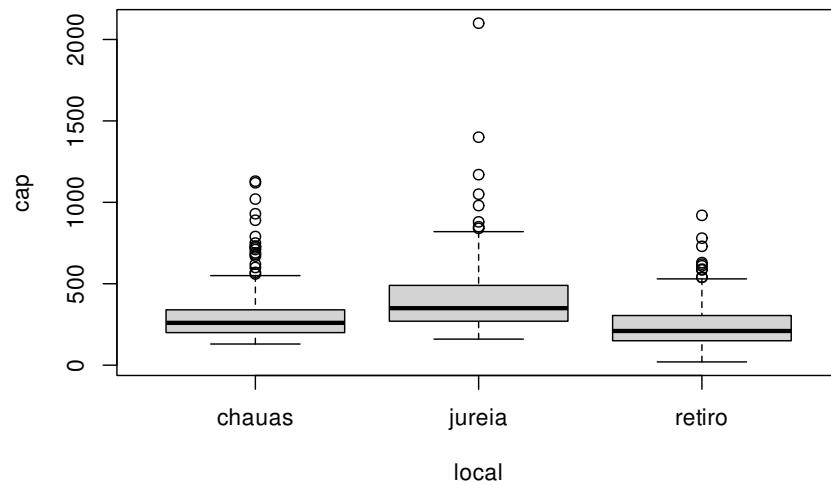


Qual a relação entre as variáveis? É linear? Que hipóteses ou interpretação biológica eu faço das relações entre as variáveis? Qual a colinearidade dos meus dados?

Para entender a razão e a importância dessas perguntas, veja a definição na WikiPedia³ sobre o efeito de colinearidade em regressões múltiplas.

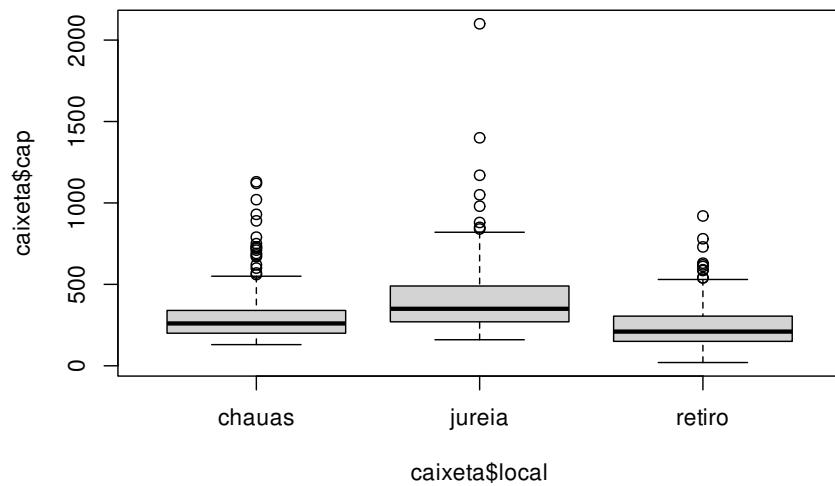
A função `pairs()` mostra as correlações das variáveis par a par de maneira gráfica, que podem ser estimadas por meio da função `cor()`.

```
## Exemplos de Graficos bivariados
## boxplot (já vimos o que isso significa)
# mostra a variação do avistamento de urubus nas diferentes fisionomias
boxplot(cap ~ local, data = caixeta)
```



³<https://en.wikipedia.org/wiki/Multicollinearity>

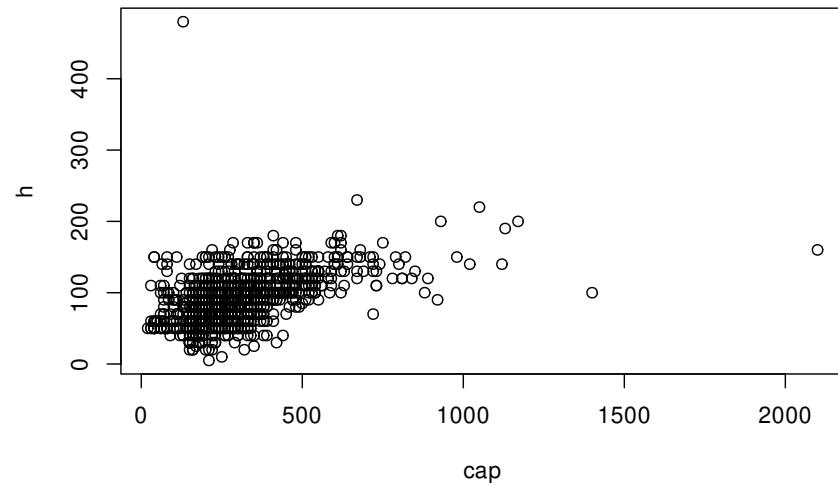
```
# ou poderia escrever assim
boxplot(caixeta$cap ~ caixeta$local)
```



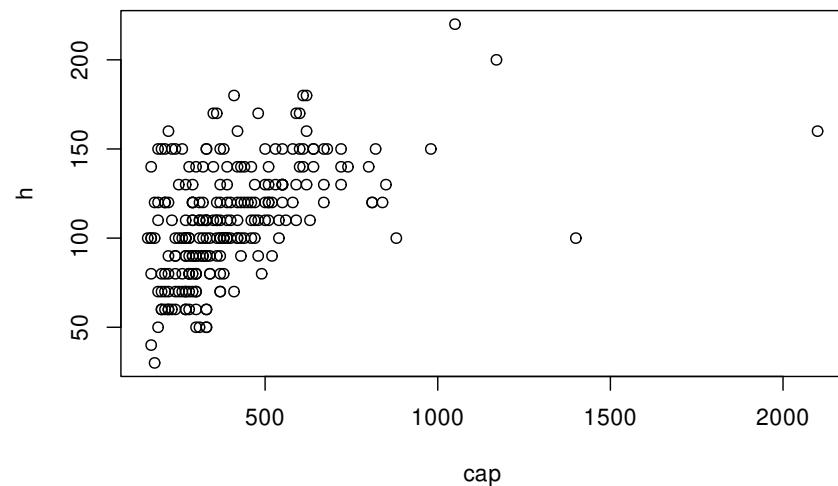
```
# note o valor extremo em jureia
vl <- caixeta$local == "jureia" & caixeta$cap > 1500
caixeta[vl, ]
```

	local	parcela	arvore	fuste	cap	h	especie	ab
557	jureia	4	106	1	2100	160	Calophyllum brasiliensis	3463606

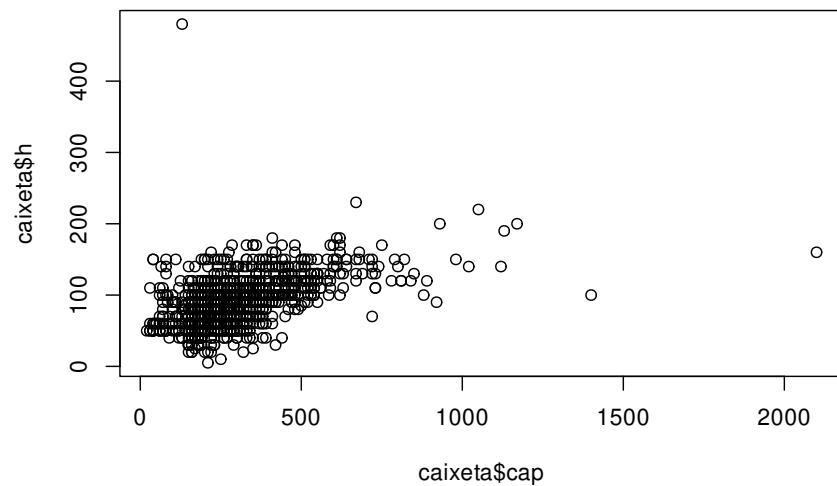
```
## espalhagrama
plot(h ~ cap, data = caixeta) # usando formula e especificacao dos dados
```



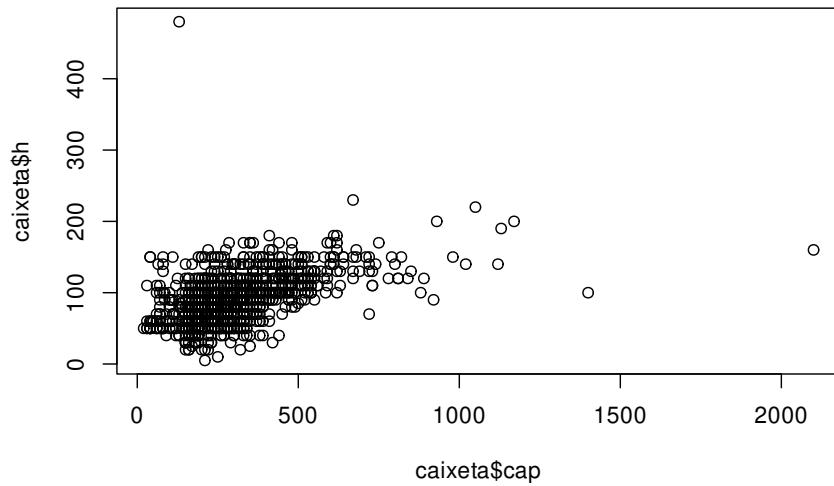
```
# apenas para jureia
plot(h ~ cap, data = caixeta, subset = local == "jureia")
```



```
plot(caixeta$h ~ caixeta$cap) # usando formula sem especificacao dos dados
```



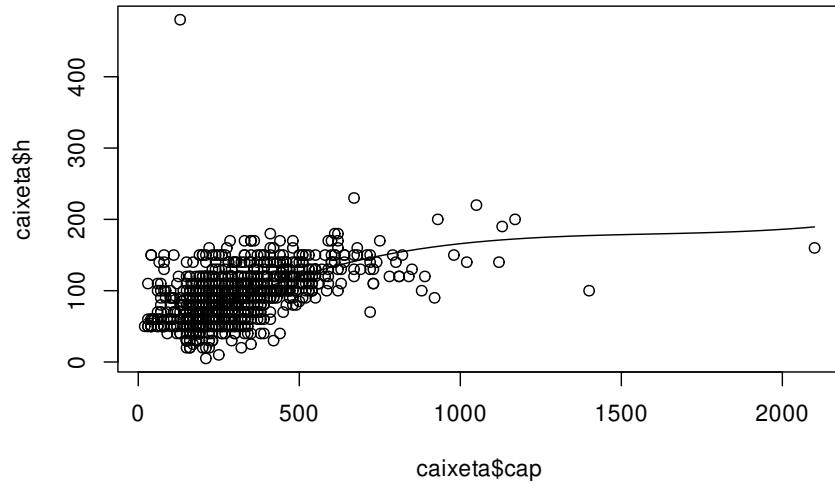
```
plot(caixeta$cap, caixeta$h) # especificando eixos separadamente (veja inversao)
```



```
names(caixeta)
```

```
## [1] "local"   "parcela" "arvore"  "fuste"   "cap"     "h"       "especie"  
## [8] "ab"
```

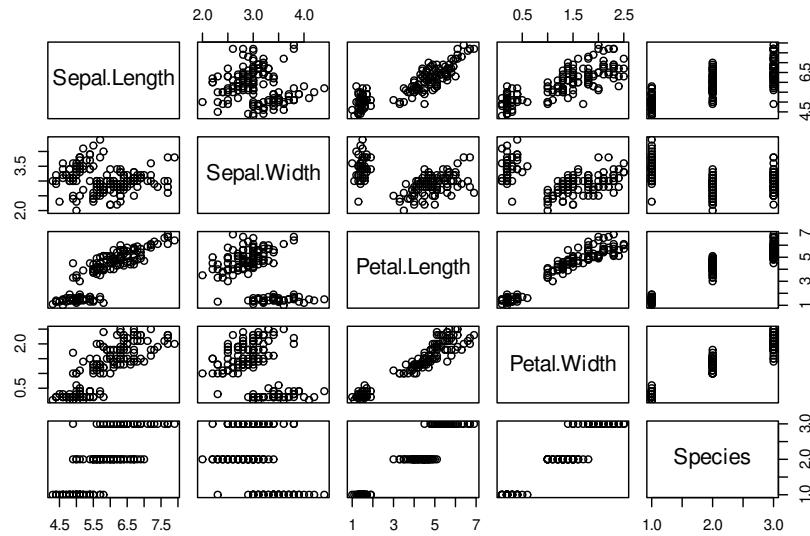
```
# mostra linha de tendencia da relacao  
scatter.smooth(caixeta$cap, caixeta$h)
```



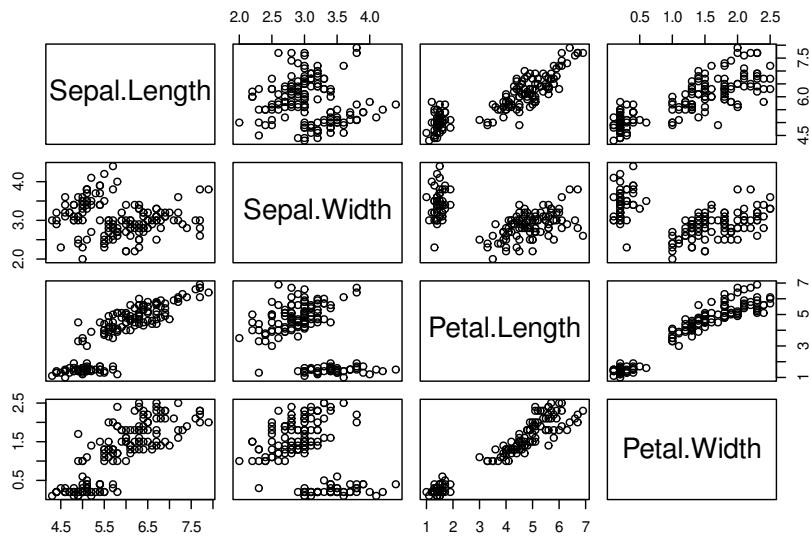
```
## pairs
## Matriz de espalhogramas das medidas das arvores no dataframe iris
data(iris)
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
```

```
pairs(iris) # todas as variáveis
```

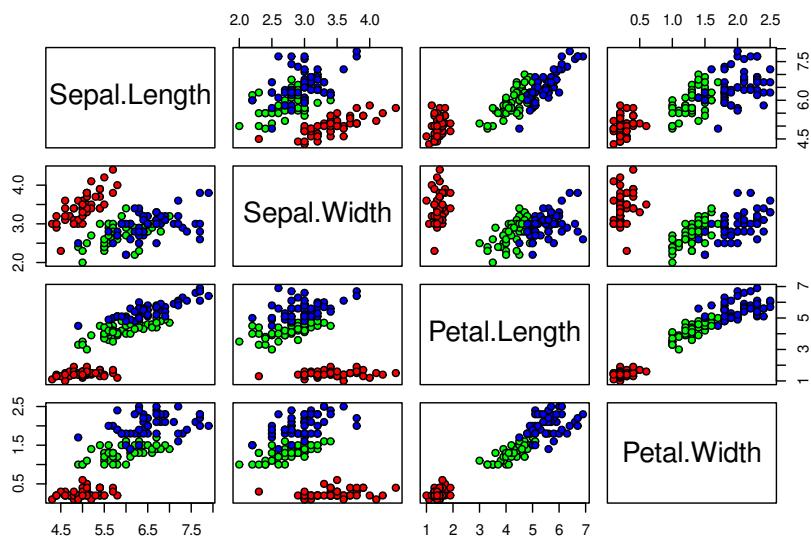


```
pairs(iris[, -ncol(iris)], ) # menos a ultima coluna = especie
```



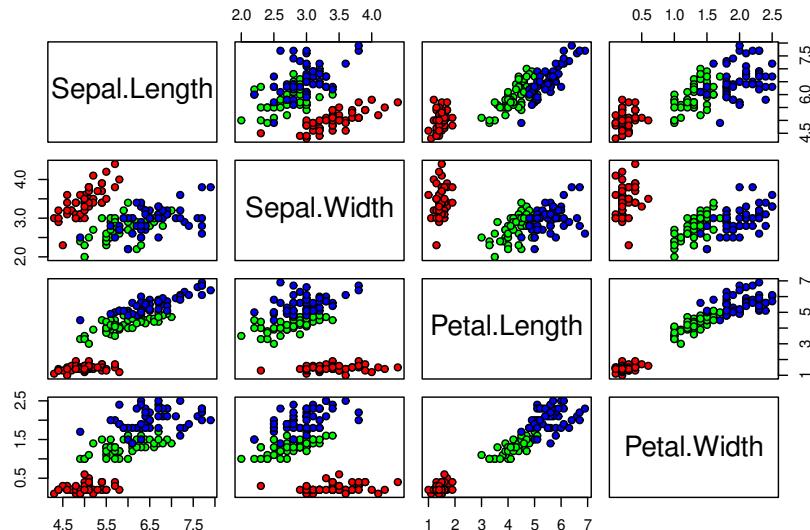
```
# colorindo por especie
```

```
pairs(iris[, -ncol(iris)], pch = 21, bg = c("red", "green", "blue")[unclass(iris$Species)])
```



```
?unclass # remove o atributo classe do objeto, então especies viram números
```

```
# poderia fazer assim, tendo em vista que iris$Species é um fator:
pairs(iris[, -ncol(iris)], pch = 21, bg = c("red", "green", "blue")[as.numeric(iris$Species)])
```



```
## Essa figura é basicamente a expressão gráfica da matriz de correlações entre todas as variáveis
cor(iris[, -ncol(iris)])
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Sepal.Length	1.0000000	-0.1175698	0.8717538	0.8179411
Sepal.Width	-0.1175698	1.0000000	-0.4284401	-0.3661259
Petal.Length	0.8717538	-0.4284401	1.0000000	0.9628654
Petal.Width	0.8179411	-0.3661259	0.9628654	1.0000000

```
# veja que essa matriz é simétrica e a diagonal tem valores =1, pois a correlação entre a variável com ela mesma é sempre 1
tbcor <- cor(iris[, -ncol(iris)])
# na diagonal
diag(tbcor)

## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##                 1             1             1             1

# acima da diagonal
vacima <- tbcor[upper.tri(tbcor)]
# abaixo da diagonal
vabaixo <- tbcor[lower.tri(tbcor)]
# entao, se é simétrica, os vetores contém os mesmos valores (a ordem não é a mesma por isso comparo)
sort(vacima) == sort(vabaixo)

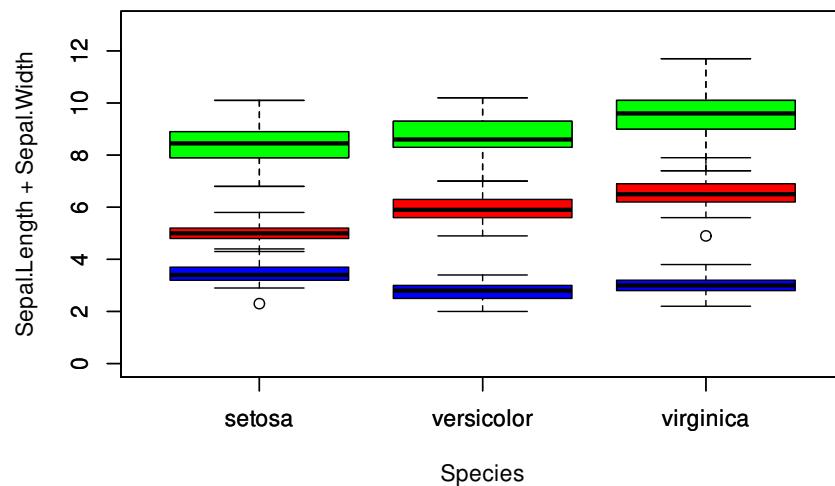
## [1] TRUE TRUE TRUE TRUE TRUE
```

13.5 Outros gráficos bivariados

As funções `xyplot()` e `bwplot()` são oriundas do pacote `lattice` ([Sarkar, 2020](#)) e permitem visualizar rapidamente relações entre variáveis por subgrupos de forma simples e rápida.

```
# muitas funções do R interpretam formulas, que é uma forma simbólica curta para designar coisas
?formula # leia com atenção a sessão de detalhes de como você pode especificar formulas, se achar necessário
```

```
# no objeto iris
plot(Sepal.Length + Sepal.Width ~ Species, data = iris, ylim = c(0, 13))
plot(Sepal.Length ~ Species, data = iris, add = T, col = "red", xlab = "", ylab = "", xaxt = "n")
plot(Sepal.Width ~ Species, data = iris, add = T, col = "blue", xlab = "", ylab = "", xaxt = "n")
# ou seja a primeira figura é o mesmo que fazer:
tt <- iris$Sepal.Length + iris$Sepal.Width
plot(tt ~ iris$Species, add = T, col = "green")
```

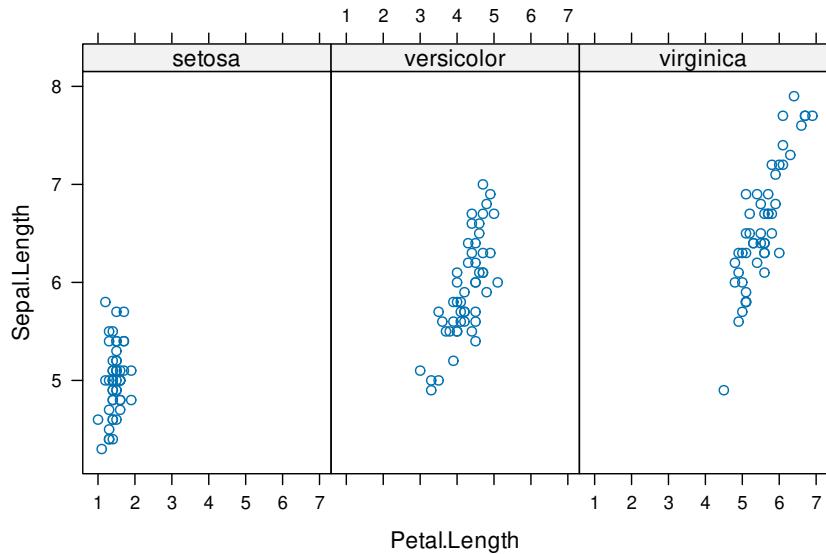


```
# pois neste caso estamos plotando boxplots e a distribuicao dos valores da interacao entre co
```

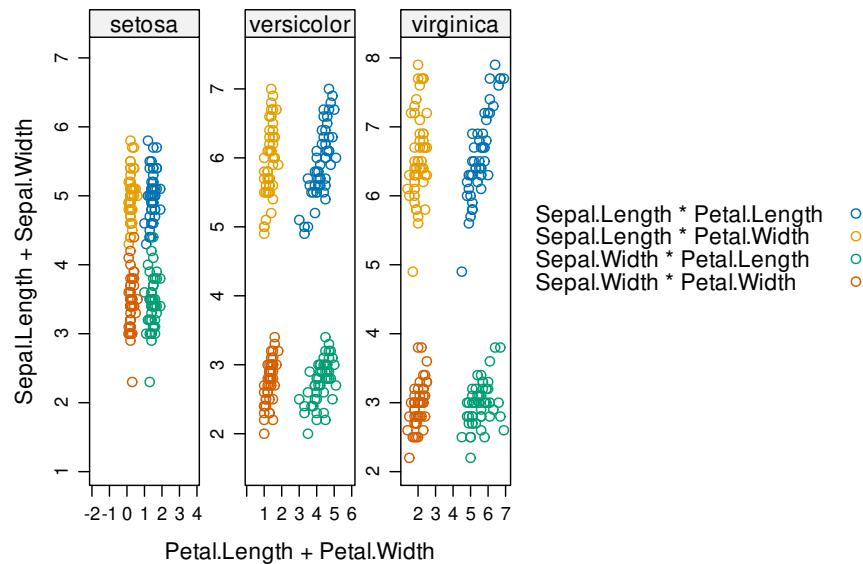
```
## Graficos condicionados com o pacote lattice
library("lattice") # carregue o pacote
```

```
# qual a relacao entre comprimento de sepals e comprimento de petals por especie?  
?xyplot # veja o help dessa funcao
```

```
xyplot(Sepal.Length ~ Petal.Length | Species, data = iris)
```



```
# ou mais complexo. Qual a relacao entre as quatro variaveis em iris, por especie?  
xyplot(Sepal.Length + Sepal.Width ~ Petal.Length + Petal.Width | Species, data = iris, scales
```



```
# note que neste caso as correlacoes estao individualizadas por especie e que as cores representam
```

```
?bwplot # para multiplos boxplots
```

```
## um data.frame com as duas especies mais abundantes do caixeta
head(caixeta)
```

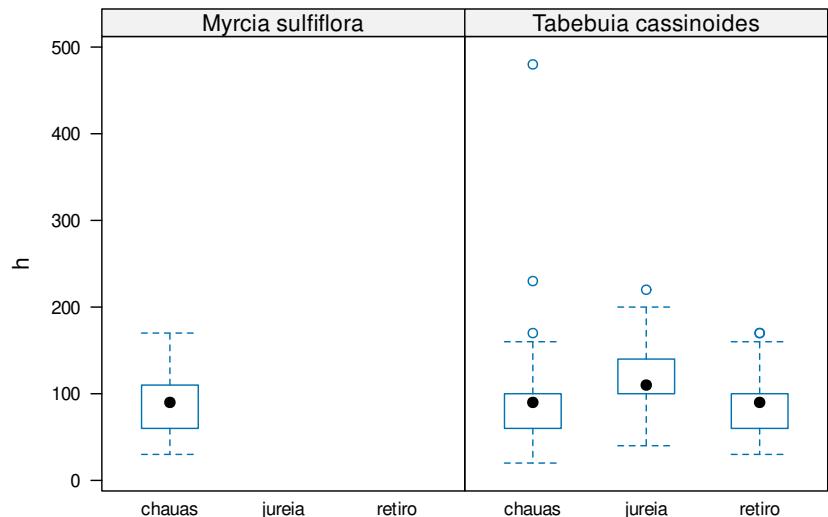
local	parcela	arvore	fuste	cap	h	especie	ab
chauas	1	1	1	210	80	Myrcia sulfiflora	34636.06
chauas	1	3	1	170	80	Myrcia sulfiflora	22698.01
chauas	1	4	1	720	70	Syagrus romanzoffianus	407150.41
chauas	1	5	1	200	80	Tabebuia cassinoides	31415.93
chauas	1	6	1	750	170	indet.1	441786.47
chauas	1	7	1	320	80	Myrcia sulfiflora	80424.77

```
tb <- table(caixeta$especie)
maisabund <- names(tb[order(tb, decreasing = T)][1:2])
maisabund
```

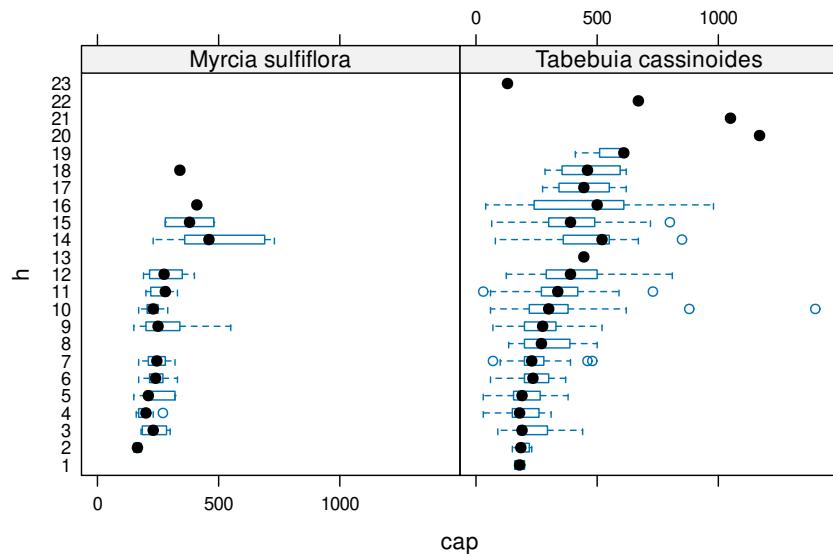
```
## [1] "Tabebuia cassinoides" "Myrcia sulfiflora"
```

```
# filtra os dados originais para essas espécies
caixeta2 <- caixeta[caixeta$especie %in% maisabund, ]

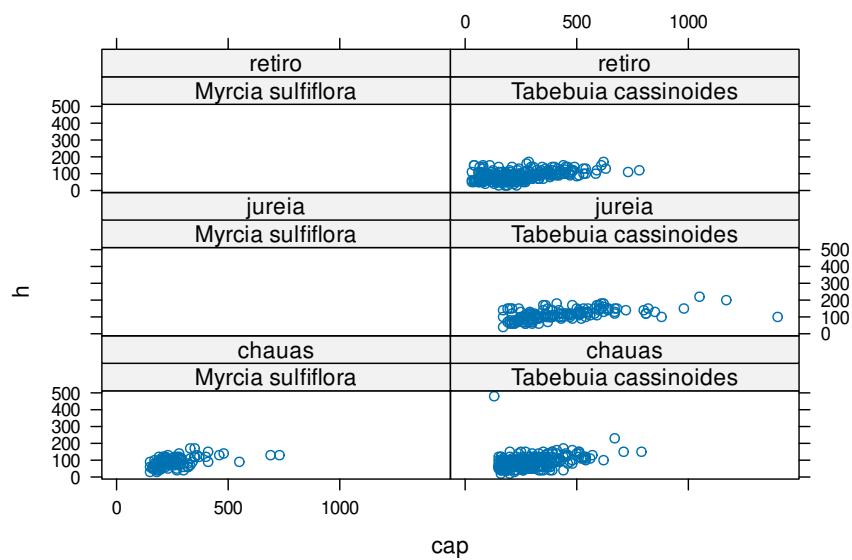
# distribuição dos valores de altura por local para cada espécie
bwplot(h ~ local | especie, data = caixeta2)
```



```
# distribuição dos valores de cap por classes de altura, por espécie
bwplot(h ~ cap | especie, data = caixeta2)
```



```
# relacao altura vs cap por especie e por local
xyplot(h ~ cap | especie + local, data = caixeta2, auto.key = T)
```





14

AED de multivariadas

Neste tutorial, vamos utilizar os pacotes abaixo. Caso você não possua algum dos pacotes listados, lembre-se de instalar cada um utilizando o comando abaixo:

```
# Para instalar pacotes no R, use a função `install.packages()`
install.packages("ape")
install.packages("labdsv")
```

Carregue-os todos e siga em frente.

```
library("labdsv")
library("ape")
library("vegan")
```

14.1 Matrizes de distância

Matrizes de distância ou dissimilaridade são muito usadas em AEDs multivariadas. Por exemplo, para estimar a similaridade entre diferentes comunidades vegetais segundo a presença ou ausência de espécies (ou um índice de similaridade que leva em conta as abundâncias relativas); para estimar a similaridade entre espécies ou a relação entre similaridade genética ou morfológica e distância geográfica etc.

A função `dist()` é a mais básica do R para calcular dissimilaridades

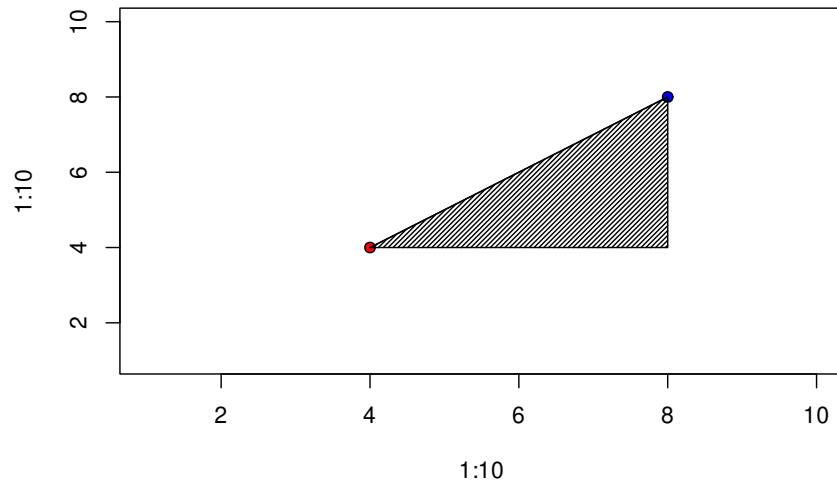
entre objetos. Ela calcula dissimilaridades segundo diferentes métodos (argumento `method`). Há também a função `vegdist()` do pacote `vegan` (Oksanen et al., 2020) que faz a mesma coisa, mas tem índices de dissimilaridade que `dist()` não implementa, muito dos quais muito usados em ecologia de comunidades.

Busque ler o `?` dessas duas funções (execute `?dist` e `?vegdist` no console), atentando para os diferentes índices de dissimilaridade. Na função `vegdist()`, você encontra os principais **índices de dissimilaridade** usados em ecologia.

Para entender o que essas funções fazem, vamos ver um exemplo simples de cálculo de distância euclidiana, que é o método padrão de `dist()`.

```
# usando o método euclidiano
# plota um gráfico vazio com coordenadas x e y de 1 a 10
plot(1:10, 1:10, type = "n")

# adiciona dois pontos:
# um na coordenada 4,4
points(4, 4, pch = 21, bg = "red")
# outro na coordenada x=8, y=8
points(8, 8, pch = 21, bg = "blue")
# a distância euclidiana entre eles é dada por essa linha
segments(4, 4, 8, 8)
# essa linha é a hipotenusa do triângulo
polygon(x = c(4, 8, 4), y = c(4, 8, 4), density = 40)
```



```
# portanto a distância entre os pontos, por Pitágoras, é
# sqrt(hipotenusa) = sum(catetoA^2+catetoB^2)
d <- sqrt((8 - 4)^2 + (8 - 4)^2)

# agora usando a função dist
# coloco as coordenadas dos dois pontos acima num data.frame
Pontos <- data.frame(X = c(4, 8), Y = c(4, 8))
Pontos
```

X	Y
4	4
8	8

```
# calcula a distância euclidiana para essas variáveis (X e Y)
dist(Pontos, method = "eucl")
```

```
##           1
## 2 5.656854
```

```
# entao isso é verdadeiro
d == dist(pontos, method = "eucl")
## [1] TRUE
```

14.2 Ordenação com matrizes de distância

Reducir espaços multivariados em poucas dimensões a partir de matrizes de distância é útil quando nossas variáveis não têm distribuição normal, que é uma das premissas da *Análise de Componentes Principais* (PCA) e outros métodos de ordenação paramétricos.

Se você parte de uma matriz de distância, pode fazer ordenações multivariadas com dados não normais ou mesmo dados categóricos e semi-quantitativos, desde que com eles você possa calcular uma matriz de distância.

14.3 Escalonamento Não-Métrico Multidimensional (NMDS)

A NMDS¹ é uma técnica de ordenação multivariada que permite visualizar graficamente distâncias entre objetos. No R há várias funções que executam isso: `isoMDS()`; `cmdscale()`; e `nmds()` e `bestnmds()` do pacote `labdsv` (Roberts, 2019). Vamos usar a função `bestnmds()` nos exemplos abaixo. Para entender, veja um exemplo para distâncias geográficas entre cidades na região norte do Brasil. Se queremos representar graficamente distâncias geográficas, estaremos de certa forma reproduzindo um mapa:

¹https://en.wikipedia.org/wiki/Multidimensional_scaling

Vamos utilizar o conjunto de dados contendo coordenadas geográficas de municípios do Brasil² para esta prática, utilizado na seção 3.4.2.

```
# visualizando distâncias usando NMDS  
# Vamos usar o arquivo com coordenadas dos municípios brasileiros  
muni <- read.table(file = "municípiosbrasil.csv", header = T, as.is = T, sep = "\t", na.string = "")
```

Para simplificar, vamos filtrar apenas algumas cidades da região Norte:

```
cids <- c("Rio Branco", "Cruzeiro do Sul", "Tabatinga", "S o Gabriel da Cachoeira", "Manaus",  
  
# filtrando os dados  
vl <- muni$Municipio %in% cids  
muni <- muni[vl, ]  
  
# calcula a distancia geografica entre essas cidades (em graus de latitude).  
# Idealmente dever amos converter latitude e longitude em d cimos de graus para UTM para obter  
mdist <- dist(muni[, c("Longitude", "Latitude")], method = "euclidean")
```

```
# calculando um nmds em dois eixos (reduzindo a variação na matriz em dois eixos)
?bestnmds
```

Vamos utilizar a função `bestnmds` do pacote `labdsrv`. Vamos agora instalar o pacote e carregar o pacote `labdsrv`:

```
onmds <- bestnmds(mdists, k = 2)
```

²<https://github.com/LABOTAM/IntroR/blob/main/dados/municipiosbrasil.csv>

```
## [1] 3.528972e-13 8.167216e-03 2.228642e+01 3.791436e+01 6.603351e-
03
##      [6] 6.029306e-03 2.109747e+01 9.041554e-03 4.415954e-
03 4.861851e-01
##      [11] 4.001459e-03 8.141675e-03 9.111469e-
03 3.737914e+01 6.265940e-03
##      [16] 4.124426e+01 9.818992e+00 2.702450e+01 3.316186e-
03 3.712228e+01
##
## best result = 1
## with stress = 3.528972e-13
```

```
# veja a estrutura do resultado
str(onmds)
```

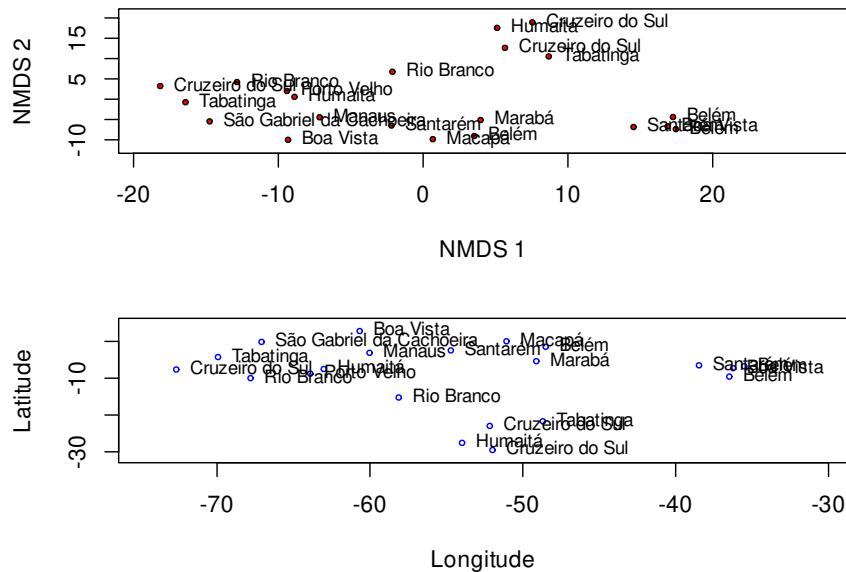
```
## List of 3
## $ points: num [1:21, 1:2] -18.2 -12.9 17.3 -14.7 -16.4 ...
##   ..- attr(*, "dimnames")=List of 2
##     ...$ : chr [1:21] "3" "14" "74" "135" ...
##     ...$ : NULL
## $ stress: num 3.53e-13
## $ type  : chr "NMDS"
## - attr(*, "class")= chr [1:2] "dsvord" "nmlds"
## - attr(*, "call")= language bestnmlds(dis = mdist, k = 2)
## - attr(*, "timestamp")= chr "Wed Oct 4 14:26:25 2023"
```

```
# o valor do stress indica o ajuste. Se o stress for 0, o ajuste é perfeito:
# a posição dos pontos é proporcional a distância entre eles
onmds$stress
```

```
## [1] 3.528972e-13
```

```
# quanto da variação foi explicada?  
# essa pergunta a gente faz com PCA, não faz sentido fazer com NMDS.  
# Mas, se você quer ter uma ideia de quanto um eixo capturou da variação,  
# pode correlacionar a matriz de distância original com uma matriz de distância  
# gerada pelos valores dos eixos nmds  
  
# pega os valores dos eixos NMDS  
ptsnmds <- onmds$points  
# calcula a distância  
adist <- dist(ptsnmds)  
# qual a correlação entre essas matrizes de distância?  
cor(mdist, adist)  
  
## [1] 1
```

```
# por isso o stress é baixo  
  
# vamos comparar graficamente:  
# divide o dispositivo em duas partes  
par(mfrow = c(2, 1), mar = c(5, 5, 1, 1))  
# adiciona limite no eixo X e y  
xl <- range(ptsnmds[, 1]) + c(-1, 10)  
yl <- range(ptsnmds[, 2]) + c(-2, 2)  
# plota pontos  
plot(ptsnmds, type = "p", pch = 21, bg = "red", xlab = "NMDS 1", ylab = "NMDS 2", xlim = xl, ylim = yl)  
# adiciona o nome das cidades  
text(ptsnmds, labels = muni$Municipio, cex = 0.8, pos = 4)  
  
xl <- range(muni$Longitude) + c(-2, 5)  
yl <- range(muni$Latitude) + c(-2, 2)  
plot(muni$Longitude, muni$Latitude, type = "p", xlab = "Longitude", ylab = "Latitude", xlim = xl, ylim = yl)  
text(muni$Longitude, muni$Latitude, labels = muni$Municipio, cex = 0.8, pos = 4)
```



14.3.1 Exemplo com dados morfológicos

Um exemplo de NMDS para mostra a similaridade entre indivíduos de *Iris* a partir de uma matriz de distância morfológica usando os dados de `iris` do R.

```
data(iris) # #carrega o conjunto de dados de iris para a area de trabalho
head(iris)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

```
# calcula a distancia morfológica  
?dist
```

```
dmorfo <- dist(iris[, 1:4], method = "eucl")
```

```
# calculando um nmds em dois eixos  
onmds <- bestnmds(dmorfo, k = 2)
```

```
# ops objetos 102 e 143 devem ser identificos  
iris[c(102, 143), ]
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
102	5.8	2.7	5.1	1.9	virginica
143	5.8	2.7	5.1	1.9	virginica

```
# entao eliminamos 1, porque senao nao funciona  
iris2 <- iris[-102, ]
```

```
# calculamos novamente a distancia  
dmorfo <- dist(iris2[, 1:4], method = "eucl")  
# calculando um nmds em dois eixos  
onmds <- bestnmds(dmorfo, k = 2)
```

```
# veja a estrutura do resultado  
str(onmds)
```

```
## List of 3  
## $ points: num [1:149, 1:2] -2.67 -2.71 -2.87 -2.74 -2.72 ...  
##   ..- attr(*, "dimnames")=List of 2  
##     ... .:$ : chr [1:149] "1" "2" "3" "4" ...  
##     ... .:$ : NULL  
## $ stress: num 2.58  
## $ type  : chr "NMDS"  
## - attr(*, "class")= chr [1:2] "dsvord" "nmlds"  
## - attr(*, "call")= language bestnmlds(dis = dmorfo, k = 2)  
## - attr(*, "timestamp")= chr "Fri Feb  5 10:50:06 2021"
```

```
# o valor do stress indica o ajuste. Se stress for 0, o ajuste é perfeito  
onmds$stress
```

```
## [1] 2.579978
```

```
# quanto da variação foi explicada?  
# essa pergunta a gente faz com PCA, não faz sentido fazer com NMDS.  
# Mas, se você quer ter uma ideia de quanto um eixo capturou da variação,  
# pode correlacionar a matriz de distância original  
# com uma matriz de distância gerada pelos valores dos eixos nmlds  
  
# pega os valores dos eixos NMDS  
ptsnmlds <- onmds$points  
# cada linha nessa tabela corresponde  
# à mesma linha na tabela iris2  
head(ptsnmlds)
```

-2.673344	0.2751282
-2.705573	-0.1645018
-2.866471	-0.0970515
-2.737119	-0.2591412
-2.717433	0.2980718
-2.296804	0.6687755

```
# essas colunas são os dois eixos NMDS  
  
# isso deve ser verdadeiro  
nrow(ptsnmds) == nrow(iris2)
```

```
## [1] TRUE
```

```
# calcula a distancia entre os pontos pelos eixos NMDS  
adist <- dist(ptsnmds)  
# qual a correlacao entre essa matriz e a original?  
cor(dmorfo, adist)
```

```
## [1] 0.9988093
```

```
# por isso o stress é baixo, a correlação é alta
```

Agora vamos fazer com outro índice de distância chamado *gower*, por exemplo, que é um bom índice quando se tem dados semiquantitativos na matriz (não é o caso aqui). Vamos utilizar a função `vegdist()` do pacote `vegan` (Oksanen et al., 2020).

```
data(iris)
iris2 <- iris[-102, ]
```

```
dmorfo2 <- vegdist(iris2[, 1:4], method = "gower")
onmds2 <- bestnmds(dmorfo2, k = 2)
```

```
# pega os valores dos eixos NMDS
ptsnmds2 <- onmds2$points
# cada linha nessa tabela corresponde
# à mesma linha na tabela iris2
head(ptsnmds2)
```

-0.2907098	0.0456498
-0.2712303	-0.0244397
-0.3013570	-0.0090738
-0.2933819	-0.0245441
-0.3066071	0.0494793
-0.2595641	0.1165398

```
# essas colunas são os dois eixos NMDS
# isso deve ser verdadeiro
nrow(ptsnmds2) == nrow(iris2)
```

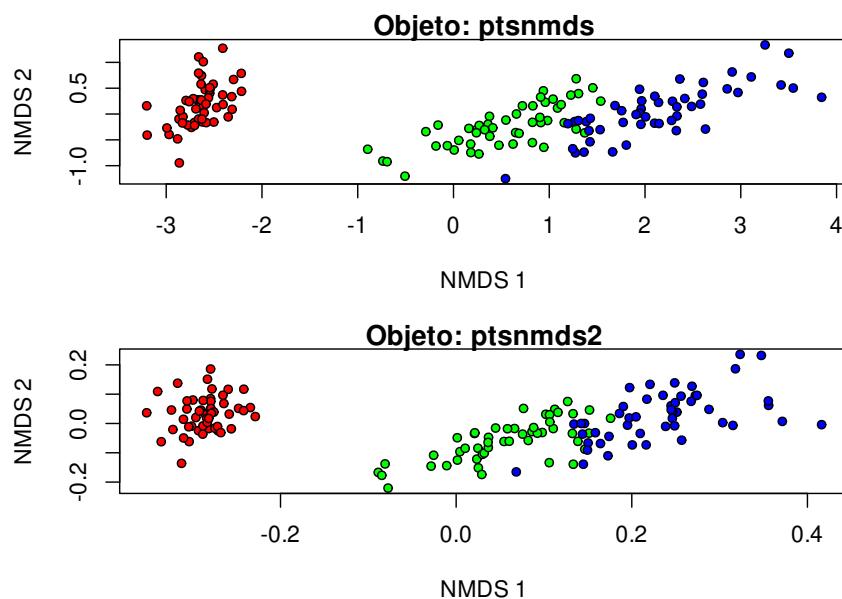
```
## [1] TRUE
```

```
# calcula a distancia entre os pontos pelos eixos NMDS
adist2 <- dist(ptsnmds2)
# qual a correlacao entre essa matriz e a original?
cor(dmorfo2, adist2)
```

```
## [1] 0.9950709
```

```
# por isso o stress é baixo, a correlação é alta
```

```
# vamos visualizar os dois resultados, onmds e onmds2, graficamente
# divide o dispositivo em dois
par(mfrow = c(2, 1), mar = c(5, 5, 1, 1))
cores <- c("red", "green", "blue")[as.numeric(iris2$Species)]
plot(ptsnmds, pch = 21, bg = cores, cex = 0.8, xlab = "NMDS 1", ylab = "NMDS 2", main = "Objeto: ptsnmds")
plot(ptsnmds2, pch = 21, bg = cores, cex = 0.8, xlab = "NMDS 1", ylab = "NMDS 2", main = "Objeto: ptsnmds2")
```



14.4 Análises de agrupamento

A função `hclust()` faz uma análise de agrupamento a partir de uma matriz de distância e segundo um método. Gera um objeto de

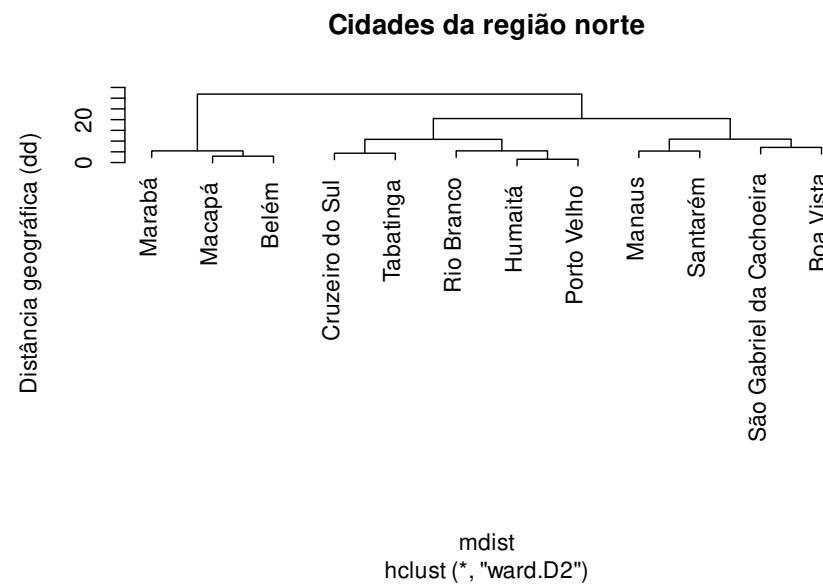
classe `hclust` que contém a estrutura hierárquica da similaridade entre os seus dados (a hierarquia dada pela distância mais o método de agrupamento).

Em análises de agrupamento, é normal lidar com objetos de classe `dendrogram`. Podemos converter alguns objetos para esta classe usando a função `as.dendrogram()`, que facilita a geração de gráficos.

Outra classe importante é `phylo`, utilizada em objetos que contenham árvores filogenéticas (pacote `ape` de [Paradis et al. \(2020\)](#)). Há a função `as.phylo()` que pode converter um objeto `hclust` para classe `phylo` e manipular o dendrograma como se fosse uma filogenia. Também facilita a geração de gráficos.

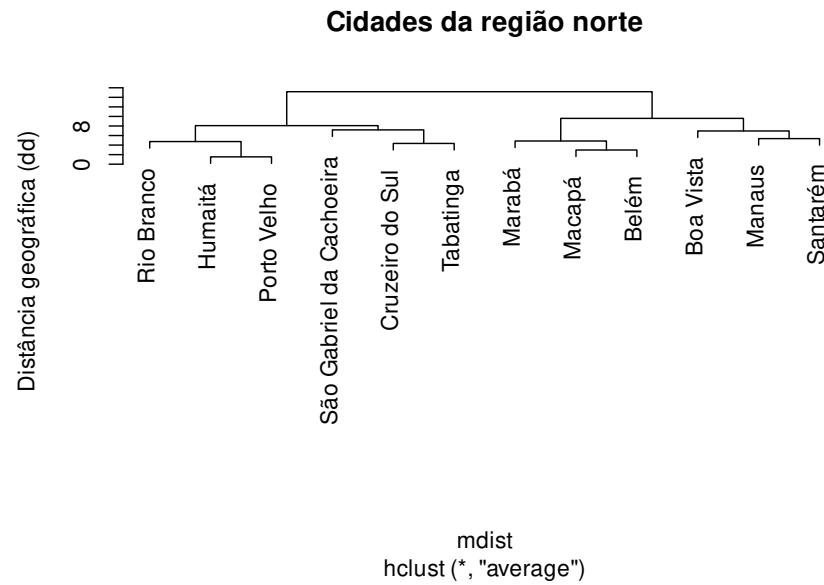
14.4.1 Agrupamento pelo método da mínima variância

```
gp <- hclust(mdist, method = "ward.D2")
# visualizando
plot(gp, main = "Cidades da região norte", ylab = "Distância geográfica (dd)")
```



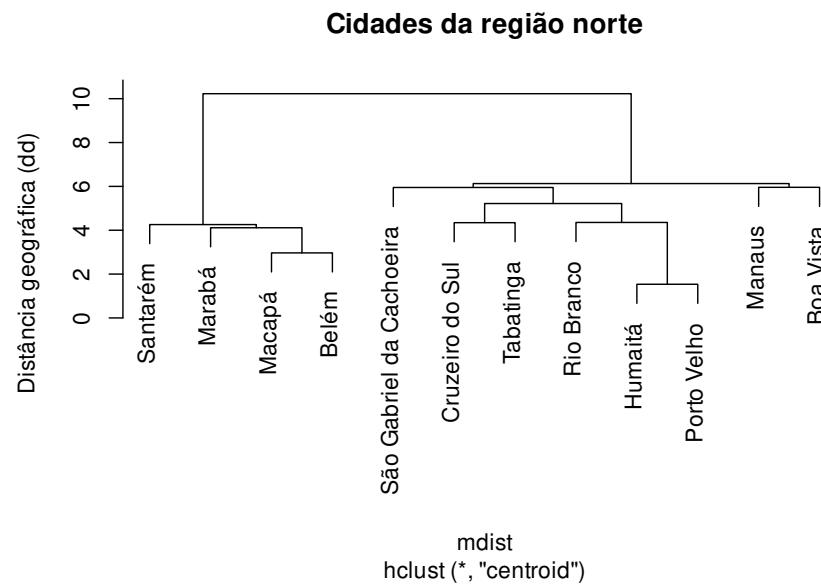
14.4.2 Agrupamento por UPGMA

```
gp2 <- hclust(mdist, method = "average")
plot(gp2, hang = 0.1, main = "Cidades da região norte", ylab = "Distância geográfica (dd)")
```



14.4.3 Agrupamento por centróides

```
gp3 <- hclust(mdist, method = "centroid")
plot(gp3, hang = 0.1, main = "Cidades da região norte", ylab = "Distância geográfica (dd)")
```

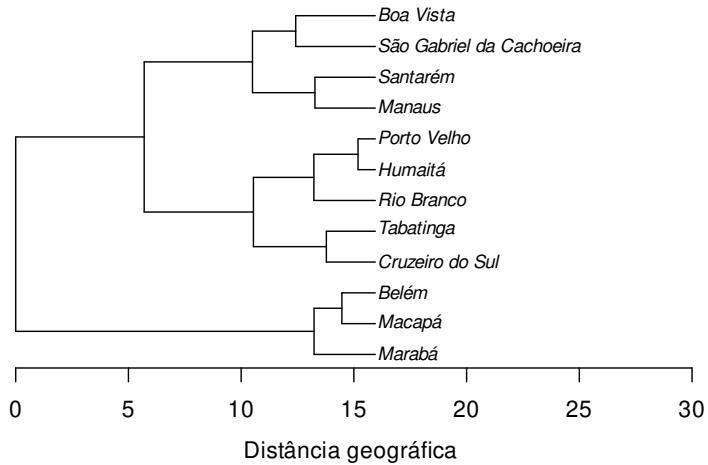


```
# teste outros métodos (entenda-os)
```

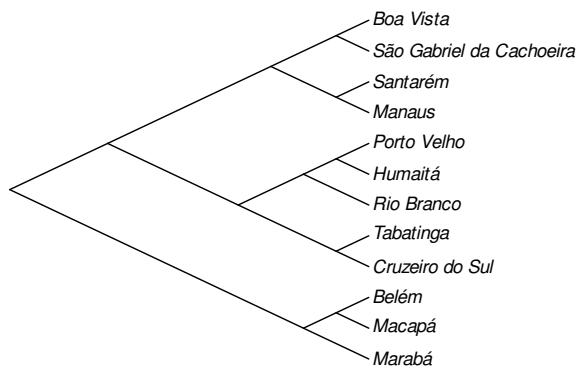
Vamos converter nossos objetos `gp` e `gp3` para objetos de classe `phylo`, e gerar um gráfico para cada um.

```
?plot.phylo
```

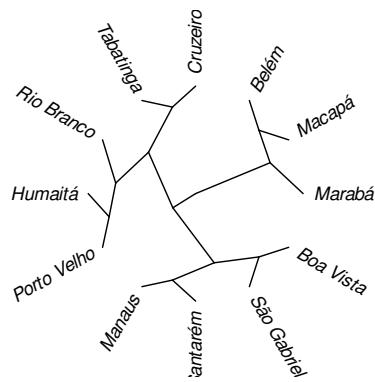
```
plot(as.phylo(gp), type = "phylogram", label.offset = 0.1, cex = 0.8)
axis(side = 1)
mtext(side = 1, line = 2.5, text = "Distância geográfica")
```



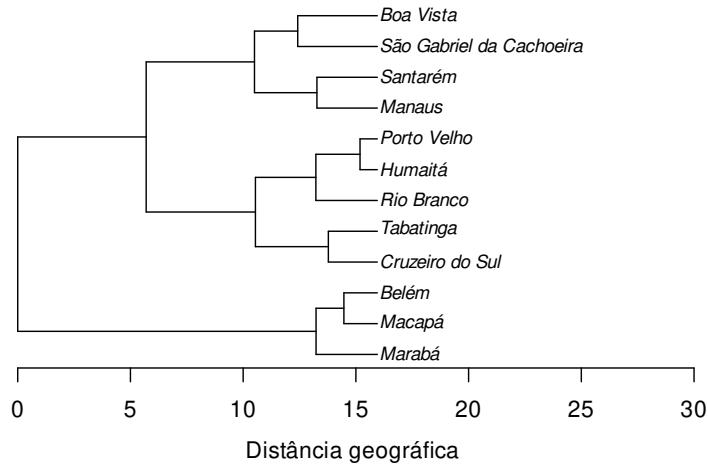
```
# ou entao, como cladograma, e nao usando o comprimento dos ramos (i.e. as distancias)
plot(as.phylo(gp), type = "cladogram", label.offset = 0.1, cex = 0.8, use.edge.length = F)
```



```
# ou entao, radial, com distancias  
plot(as.phylo(gp), type = "radial", label.offset = 0.1, cex = 0.8, use.edge.length = T)
```

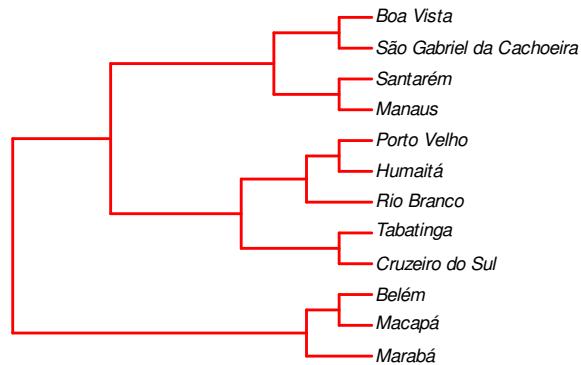


```
# usando comprimento dos ramos (distancias+relacoes)  
plot(as.phylo(gp), type = "phylogram", label.offset = 0.1, cex = 0.8, use.edge.length = T)  
axis(side = 1)  
mtext(side = 1, line = 2.5, text = "Distância geográfica")
```



```
# não usando o comprimento dos ramos (relações apenas)
```

```
plot(as.phylo(gp), type = "phylogram", label.offset = 0.1, cex = 0.8, use.edge.length = F, edg
```



14.4.4 Exemplo florístico

Vamos explorar a similaridade florística entre parcelas com os dados de caixetais novamente.

```
caixeta <- read.table("caixeta.csv", sep = ",", header = T, na.strings = c("NA", "", "NULL"))
```

```
head(caixeta)
```

local	parcela	arvore	fuste	cap	h	especie
chauas	1	1	1	210	80	Myrcia sulfiflora
chauas	1	3	1	170	80	Myrcia sulfiflora
chauas	1	4	1	720	70	Syagrus romanzoffianus
chauas	1	5	1	200	80	Tabebuia cassinoides
chauas	1	6	1	750	170	indet.1
chauas	1	7	1	320	80	Myrcia sulfiflora

```
names(caixeta)
```

```
## [1] "local"   "parcela" "arvore"  "fuste"   "cap"     "h"       "especie"
```

```
# vamos visualizar a similaridade florística entre parcelas
# entao geramos uma tabela de parcela vs. especies
# primeiro um vetor com valores únicos para local+parcela
# porque o número da parcela repete entre locais
parcelas <- paste(caixeta$local, caixeta$parcela, sep = "-")
tb <- table(parcelas, especies = caixeta$especie)
```

```
tb[, 1:5]
```

	Alchornea triplinervia	Andira fraxinifolia	bombacaceae	Cabralea canjerana	Ca
chauas-1	0	0	0	0	0
chauas-2	0	0	0	0	0
chauas-3	0	0	0	0	0
chauas-4	0	0	0	0	0
chauas-5	0	0	0	0	0
jureia-1	0	0	0	0	3
jureia-2	1	1	1	1	1
jureia-3	0	2	0	0	0
jureia-4	1	0	0	0	0
jureia-5	1	1	0	0	0
retiro-1	1	0	0	0	0
retiro-2	9	0	0	0	0
retiro-3	2	0	0	0	0
retiro-4	0	0	0	0	0
retiro-5	0	0	0	0	0

```
dim(tb)
```

```
## [1] 15 43
```

```
# essa tabela contém o número de indivíduos de cada espécie
# em cada parcela

# calculando um índice de distância de Jaccard para dados de presença e ausência

# transformando em uma tabela de presença e ausencia
tb2 <- tb
tb2[tb2 > 0] <- 1
```

Vamos utilizar a função `vegdist()` do pacote `vegan`.

```
?vegdist
```

```
djac <- vegdist(tb2, method = "jaccard")
class(djac)
```

```
## [1] "dist"
```

```
# é uma matriz de distância entre parcela
as.matrix(djac)[1:4, 1:4]
```

	chauas-1	chauas-2	chauas-3	chauas-4
chauas-1	0.0000000	0.7500000	0.80	0.8181818
chauas-2	0.7500000	0.0000000	0.50	0.5714286
chauas-3	0.8000000	0.5000000	0.00	0.2500000
chauas-4	0.8181818	0.5714286	0.25	0.0000000

Podemos fazer um NMDS com esse resultado:

```
onmds <- bestnmds(djac, k = 2)
# parcelas 3 e 5 tem exatamente as mesmas espécies
# vamos com colocar um valor super pequeno para essa distância (quase zero)
```

```
djac[djac == 0] <- 0.00000000000000000000000000000001
```

```
# agora funciona
onmds <- bestnmrds(djac, k = 2)
```

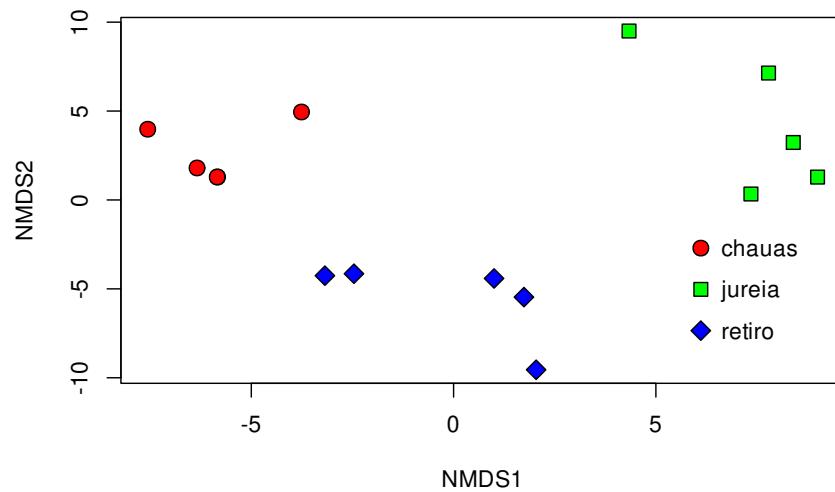
```
# plotando o resultado
# pega cores e simbolos segundo o local
ll <- data.frame(local = caixeta$local, parcelas)
ll <- unique(ll)
rownames(ll) <- ll$parcelas
ll
```

	local	parcelas
chauas-1	chauas	chauas-1
chauas-2	chauas	chauas-2
chauas-3	chauas	chauas-3
chauas-4	chauas	chauas-4
chauas-5	chauas	chauas-5
jureia-1	jureia	jureia-1
jureia-2	jureia	jureia-2
jureia-3	jureia	jureia-3
jureia-4	jureia	jureia-4
jureia-5	jureia	jureia-5
retiro-1	retiro	retiro-1
retiro-2	retiro	retiro-2
retiro-3	retiro	retiro-3
retiro-4	retiro	retiro-4
retiro-5	retiro	retiro-5

```
rn <- rownames(as.matrix(tb2))
locais <- as.factor(ll[rn, "local"])

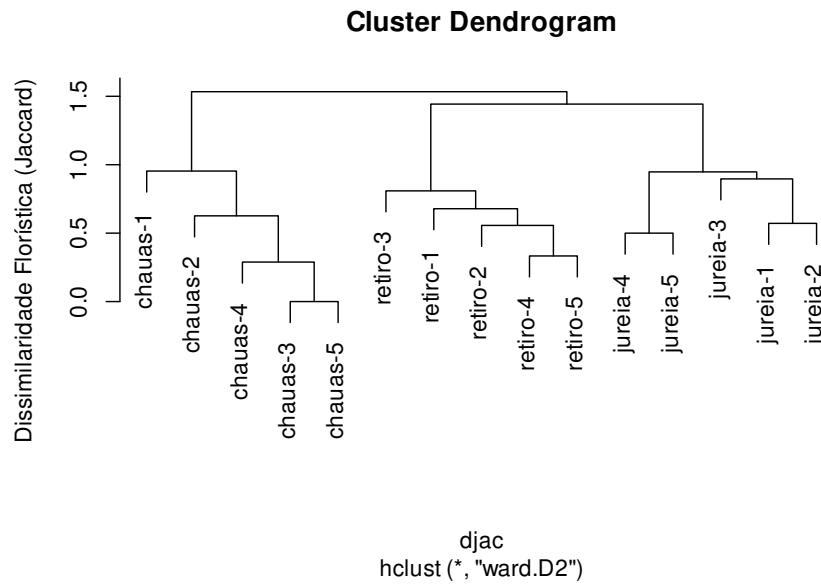
cores <- c("red", "green", "blue")[as.numeric(locais)]
pchs <- (21:23)[as.numeric(locais)]
```

```
plot(onmds$points, pch = pchs, bg = cores, xlab = "NMDS1", ylab = "NMDS2", cex = 1.5)
legend("bottomright", legend = levels(locais), pch = 21:23, pt.bg = c("red", "green", "blue"))
```



Agora vamos visualizar o resultado na forma de um agrupamento:

```
cluster <- hclust(djac, method = "ward.D2")
plot(cluster, ylab = "Dissimilaridade Florística (Jaccard)")
```



Para aprimorar a figura, vamos utilizar a função `as.phylo()` do pacote [ape](#) ([Paradis et al., 2020](#)) para converter o objeto `cluster` de classe `dendrogram` para um de classe `phylo`.

```
pcl <- as.phylo(cluster)
par(mar = c(5, 4, 3, 3))
plot(pcl, tip.color = cores, label.offset = 0.02, cex = 0.8)
# pontos
tiplabels(pch = 21, frame = NULL, bg = cores)
# eixo
axisPhylo()
# nome do eixo
mtext(text = "Jaccard índice (0 ou 1)", side = 1, line = 2.5)
legend("topleft", legend = levels(locais), pch = 21:23, pt.bg = c("red", "green", "blue"), ins
```

Vamos repetir essa operação considerando a abundância de espécies por parcelas contidos no objeto `tb` (usamos o índice de Sorensen).

```
tb[1:4, 1:5]
```

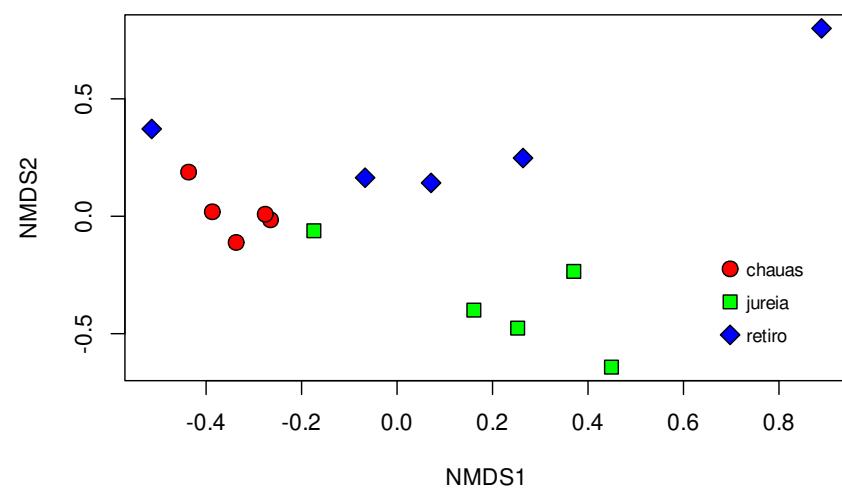
	Alchornea triplinervia	Andira fraxinifolia	bombacaceae	Cabralea canjerana	Ca
chauas-1	0	0	0	0	0
chauas-2	0	0	0	0	0
chauas-3	0	0	0	0	0
chauas-4	0	0	0	0	0

```
# sorensen (bray na convencao do R)
dsor <- vegdist(tb, method = "bray")
```

```
onmds2 <- bestnmds(dsor, k = 2)
```

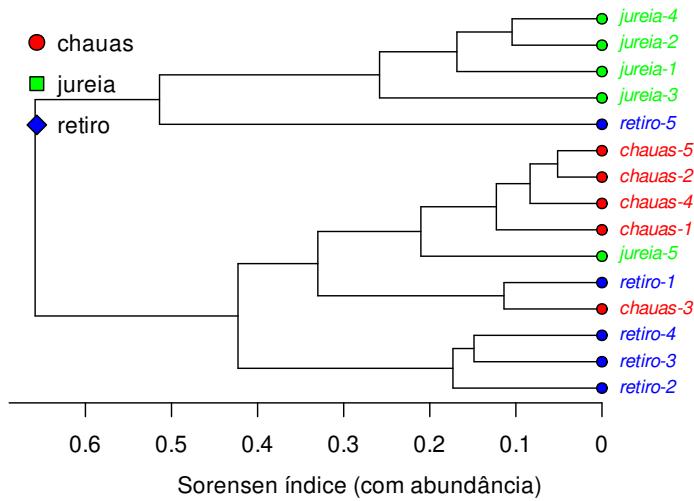
Plota o NMDS:

```
plot(onmds2$points, pch = pchs, bg = cores, xlab = "NMDS1", ylab = "NMDS2", cex = 1.5)
legend("bottomright", legend = levels(locais), pch = 21:23, pt.bg = c("red", "green", "blue"))
```



Agora, vamos plotar o agrupamento:

```
cluster2 <- hclust(dsor, method = "ward.D2")
pcl2 <- as.phylo(cluster2)
par(mar = c(5, 4, 3, 3))
plot(pcl2, tip.color = cores, label.offset = 0.02, cex = 0.8)
tiplabels(pch = 21, frame = NULL, bg = cores)
axisPhylo()
mtext(text = "Sorensen índice (com abundância)", side = 1, line = 2.5)
legend("topleft", legend = levels(locais), pch = 21:23, pt.bg = c("red", "green", "blue"), ins
```



14.4.5 Análise de Coordenadas Principais (PCoA)

A função `capscale()` faz uma análise de coordenadas principais (ou escalonamento multidimensional métrico ou clássico). É parecida com uma Análise de Componentes Principais (PCA), mas é baseada em matrizes de distância. Indica os efeitos das variáveis (parâmetro “species”) sobre os eixos. Já a função `ordiplot()` do pacote `vegan`

(Oksanen et al., 2020) permite graficar uma ordenação e os efeitos das variáveis (sites vs. species).

Leiam aqui uma comparação entre PCA e PCoA³

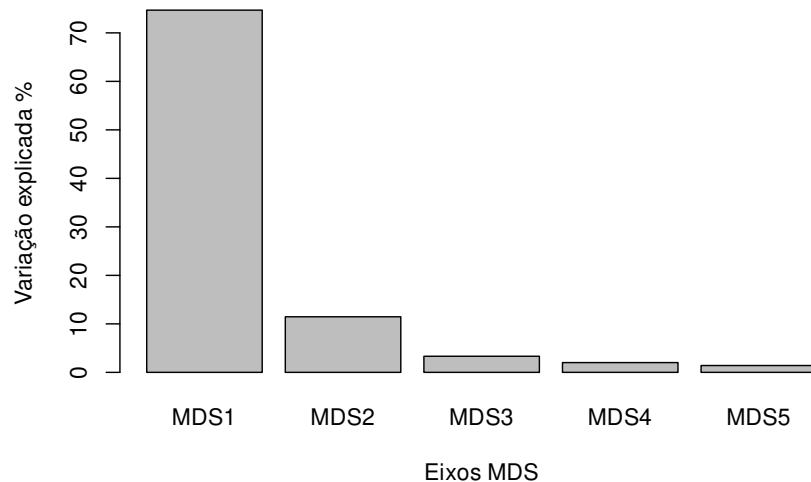
```
# análise de coordenadas principais
data(iris) # carrega o conjunto de dados iris
dt <- iris[, 1:4]
mypcoa <- capscale(dt ~ 1, distance = "gower", add = F)

# quando da variação está explicado pelos eixos
resumo <- summary(mypcoa)
var.expl <- resumo$cont$importance
# veja a proporção cumulativa dos primeiros cinco eixos
var.expl["Cumulative Proportion", ][1:5]
```

```
##          MDS1        MDS2        MDS3        MDS4        MDS5
## 0.7468617 0.8614730 0.8946967 0.9148675 0.9289785
```

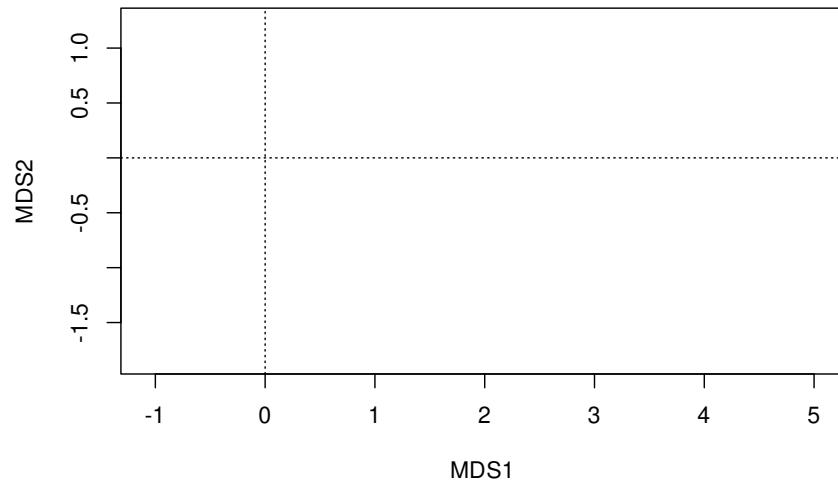
```
# pega a proporção explicada por cada eixo
tt <- var.expl["Proportion Explained", ][1:5]
tt <- tt * 100
# gera um gráfico de barras com isso
barplot(tt, xlab = "Eixos MDS", ylab = "Variação explicada %")
```

³http://occamstypewriter.org/boboh/2012/01/17/pca_and_pcoa_explained/



```
# fazendo uma figura com ordiplot  
# veja o help  
?ordiplot
```

```
# define cor e simbolo por especie  
tx <- as.factor(iris$Species)  
# simbolos para os niveis  
upchs <- 21:23  
# cores para os niveis  
cores <- rainbow(length(levels(tx)), alpha = 0.5)  
  
# gera a figura  
fig <- ordiplot(mypcoa, type = "n")
```



fig

```
## $species
##           MDS1      MDS2
## Sepal.Length  2.0020165 -1.1230766
## Sepal.Width   -0.4980034 -1.8451477
## Petal.Length  4.7366457  0.1241131
## Petal.Width   2.0004259 -0.1699399
##
## $sites
##           MDS1      MDS2
## 1     -0.639346910 -0.334337841
## 2     -0.596126508  0.275887062
## 3     -0.653319917 -0.016126551
## 4     -0.636358760  0.140021832
## 5     -0.674817738 -0.392213943
## 6     -0.579155725 -0.546656575
## 7     -0.671063265 -0.221682254
## 8     -0.626572082 -0.242899975
```

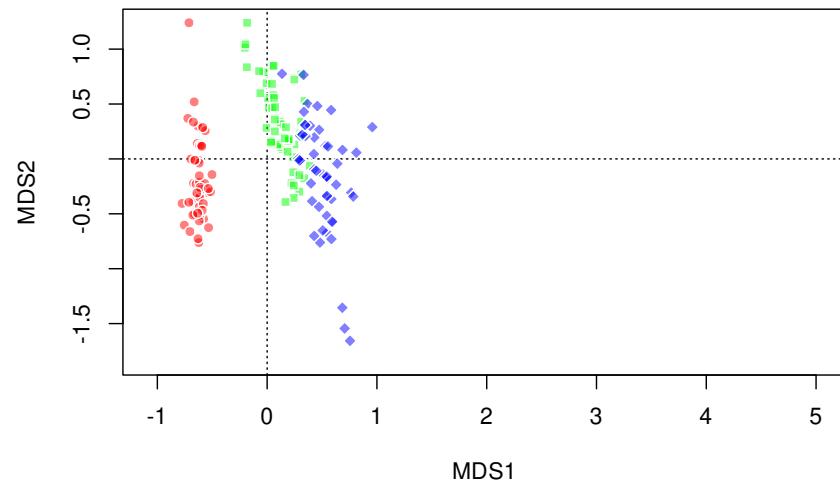
```
## 9 -0.664279148 0.519847477
## 10 -0.614966776 0.122714593
## 11 -0.615294321 -0.511938998
## 12 -0.646472500 -0.228283528
## 13 -0.629731020 0.299388944
## 14 -0.722261196 0.369205877
## 15 -0.622437432 -0.762492184
## 16 -0.629173895 -0.726881442
## 17 -0.618980764 -0.564370107
## 18 -0.615191970 -0.323676845
## 19 -0.533502504 -0.626059993
## 20 -0.661661401 -0.501714454
## 21 -0.542715066 -0.306701055
## 22 -0.616801217 -0.436985257
## 23 -0.774448860 -0.405722239
## 24 -0.502107026 -0.142145011
## 25 -0.614590494 -0.211062294
## 26 -0.563614627 0.256656581
## 27 -0.568570813 -0.223711949
## 28 -0.613242267 -0.349056953
## 29 -0.604086865 -0.278203211
## 30 -0.624882833 -0.013140058
## 31 -0.598972751 0.125800250
## 32 -0.515682484 -0.298552065
## 33 -0.754577597 -0.603205019
## 34 -0.703082687 -0.661783818
## 35 -0.595079133 0.114900783
## 36 -0.618584029 -0.038036413
## 37 -0.580529961 -0.407379782
## 38 -0.713019060 -0.394969549
## 39 -0.672009242 0.335685222
## 40 -0.610912068 -0.259288303
## 41 -0.640144469 -0.309883661
## 42 -0.711752273 1.240346847
## 43 -0.695263034 -0.001191832
## 44 -0.535343600 -0.269742736
## 45 -0.591424977 -0.468299556
```

```
## 46 -0.586455349 0.286593625
## 47 -0.676667980 -0.511683895
## 48 -0.658539186 -0.011843595
## 49 -0.632891474 -0.494385045
## 50 -0.617572023 -0.153342992
## 51 0.292460092 -0.297630508
## 52 0.221901481 -0.221174100
## 53 0.339843741 -0.177183638
## 54 0.051126045 0.843979557
## 55 0.300908538 0.232678671
## 56 0.111814267 0.339772311
## 57 0.241893772 -0.355988490
## 58 -0.202930046 1.012615941
## 59 0.248080500 0.131744670
## 60 0.007993323 0.554619740
## 61 -0.182673684 1.239393372
## 62 0.139701498 0.091146313
## 63 0.059288614 0.848318236
## 64 0.209733460 0.169175362
## 65 -0.005068163 0.282602772
## 66 0.238141035 -0.120973367
## 67 0.122837631 0.109976117
## 68 0.016222534 0.467788379
## 69 0.311051516 0.770411121
## 70 -0.005187421 0.689554745
## 71 0.242920369 -0.241807259
## 72 0.125256797 0.316113208
## 73 0.340574069 0.527424554
## 74 0.171841708 0.288331764
## 75 0.189798259 0.162593178
## 76 0.238519249 0.016465753
## 77 0.338082159 0.216400983
## 78 0.387884789 -0.062564151
## 79 0.198311313 0.179797967
## 80 -0.061965220 0.599067756
## 81 -0.032867870 0.789833613
## 82 -0.073171586 0.800055220
```

```
## 83  0.047214150  0.463286100
## 84  0.309497015  0.339528626
## 85  0.093549599  0.131907477
## 86  0.167239565 -0.392321622
## 87  0.294024647 -0.147628397
## 88  0.247163015  0.723407445
## 89  0.035767804  0.161489751
## 90  0.044401583  0.682945567
## 91  0.057076478  0.581691950
## 92  0.186717733  0.064915200
## 93  0.063819056  0.555072534
## 94  -0.197650713  1.043312580
## 95  0.072437323  0.471714137
## 96  0.036318404  0.149934488
## 97  0.070911803  0.250386339
## 98  0.160141241  0.186612762
## 99  -0.184452975  0.835312341
## 100 0.070658484  0.359870688
## 101 0.544341091 -0.672495149
## 102 0.346531367  0.308880496
## 103 0.628620224 -0.234682070
## 104 0.425700939  0.044029368
## 105 0.548076546 -0.174012341
## 106 0.765563971 -0.307081449
## 107 0.135746747  0.775251807
## 108 0.639181023 -0.042979700
## 109 0.582823197  0.444894512
## 110 0.684892213 -1.355474212
## 111 0.409601151 -0.385190295
## 112 0.473494068  0.264687908
## 113 0.545418856 -0.176398832
## 114 0.367484149  0.500353060
## 115 0.429508150  0.195416139
## 116 0.472591785 -0.436408978
## 117 0.429846283 -0.095370475
## 118 0.754725571 -1.656863357
## 119 0.957805291  0.289914582
```

```
## 120  0.331639633  0.765685406
## 121  0.589370455 -0.569532674
## 122  0.300489412  0.212345851
## 123  0.812656344  0.057296569
## 124  0.391276588  0.301638945
## 125  0.506555039 -0.651564472
## 126  0.544049996 -0.518977132
## 127  0.348827435  0.208563553
## 128  0.310203435 -0.013129804
## 129  0.532244657  0.118476560
## 130  0.505511316 -0.139381364
## 131  0.686686146  0.082078523
## 132  0.706147930 -1.544179310
## 133  0.554407566  0.113570942
## 134  0.321286620  0.222228206
## 135  0.334641562  0.429203845
## 136  0.784610274 -0.342923892
## 137  0.481116598 -0.761450083
## 138  0.399240507 -0.223139690
## 139  0.284133608  0.001057007
## 140  0.533100767 -0.337438424
## 141  0.584583765 -0.365420874
## 142  0.545530105 -0.337529051
## 143  0.346531367  0.308880496
## 144  0.592570293 -0.572375037
## 145  0.585340077 -0.729980814
## 146  0.543070756 -0.160569316
## 147  0.457165585  0.481666379
## 148  0.447862748 -0.107690190
## 149  0.428195734 -0.701597535
## 150  0.296121356 -0.013659497
##
## attr(),"const")
## [1] 5.974659
## attr(),"class")
## [1] "ordiplot"
```

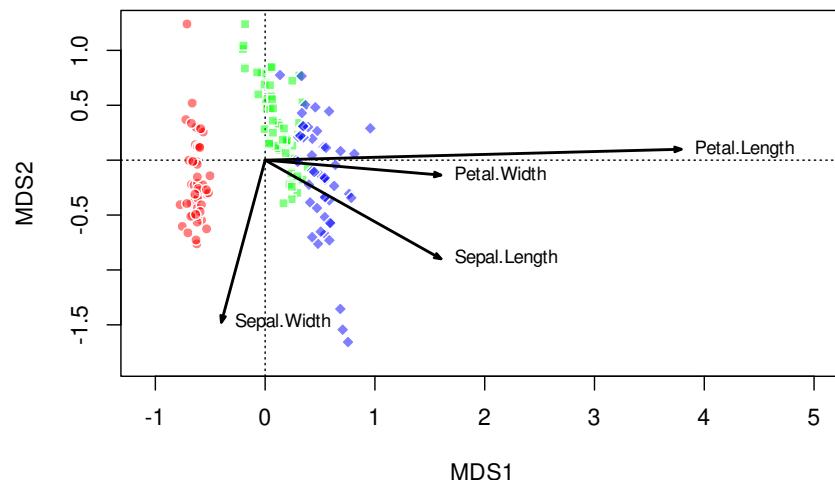
```
# adiciona os pontos de cada linha - ATENCAO - Rode os dois comandos abaixo juntos
ordiplot(mypcoa, type = "n")
points(fig, "sites", pch = upchs[as.numeric(tx)], bg = cores[as.numeric(tx)], col = "white")
```



```
# pega os scores das variáveis
# que mostram os efeitos das variáveis usadas
mls <- vegan::scores(mypcoa, display = "species")
mls
```

	MDS1	MDS2
Sepal.Length	2.0020165	-1.1230766
Sepal.Width	-0.4980034	-1.8451477
Petal.Length	4.7366457	0.1241131
Petal.Width	2.0004259	-0.1699399

```
ordiplot(mypcoa, type = "n")
points(fig, "sites", pch = upchs[as.numeric(tx)], bg = cores[as.numeric(tx)], col = "white")
# plota flechas para esses efeitos
arrows(0, 0, mls[, 1] * 0.8, mls[, 2] * 0.8, length = 0.05, angle = 20, col = "black", lwd = 2)
text(mls[, 1] * 0.8, mls[, 2] * 0.8, labels = rownames(mls), col = "black", cex = 0.8, pos = 4)
```



14.5 Componentes Principais (PCA)

Análise de Componentes Principais (PCA)⁴ é o método mais conhecido de ordenação, mas diferentemente dos métodos que se baseiam em matrizes de distância (ver abaixo) que são mais flexíveis e menos exigentes quanto à premissas estatísticas, a ordenação com PCA tem as seguintes limitações:

⁴https://pt.wikipedia.org/wiki/An%C3%A1lise_de_componentes_principais

- Os componentes principais são independentes apenas se os dados possuirem **distribuição normal conjuntamente**;
- A PCA é sensível à escala relativa das variáveis originais.

Um exemplo:

```
# análise de componentes principais  
?prcomp
```

```
data(iris) # carrega o conjunto de dados iris para a rea de trabalho  
dt <- iris[, 1:4]
```

```
meu.pca <- prcomp(dt, scale. = T, tol = 0, retx = T)  
# entenda os argumentos usados
```

```
# classe gerada  
class(meu.pca)
```

```
## [1] "prcomp"
```

```
# elementos do resultado  
names(meu.pca)
```

```
## [1] "sdev"      "rotation"   "center"    "scale"     "x"
```

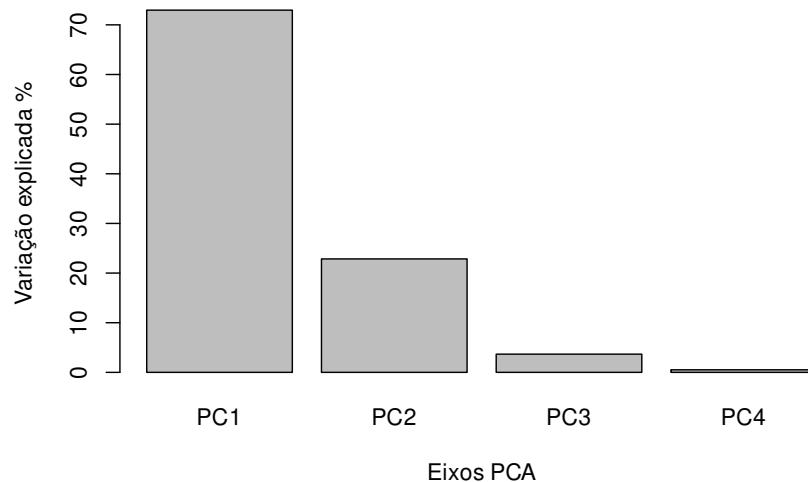
Vamos fazer uma figura utilizando a função `ordiplot()` do pacote `vegan`:

```
# fazendo uma figura com ordiplot  
# veja o help  
?ordiplot
```

```
# quando da variação está explicado pelos eixos  
resumo <- summary(meupca)  
var.expl <- resumo$importance  
# veja a proporção cumulativa dos eixos gerados  
var.expl["Cumulative Proportion", ]
```

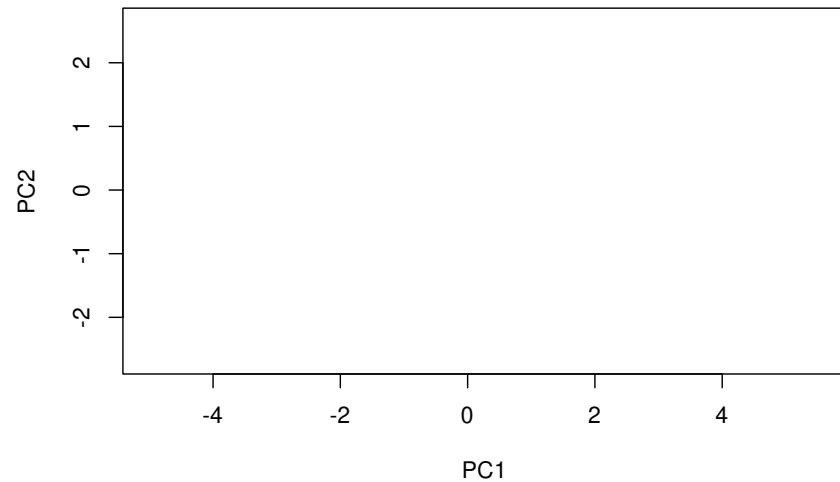
```
##      PC1      PC2      PC3      PC4  
## 0.72962 0.95813 0.99482 1.00000
```

```
# quatro eixos explicam 100% da variacao  
# pega a proporção explicada por cada eixo  
tt <- var.expl["Proportion of Variance", ]  
tt <- tt * 100  
# gera um gráfico de barras com isso  
barplot(tt, xlab = "Eixos PCA", ylab = "Variação explicada %")
```

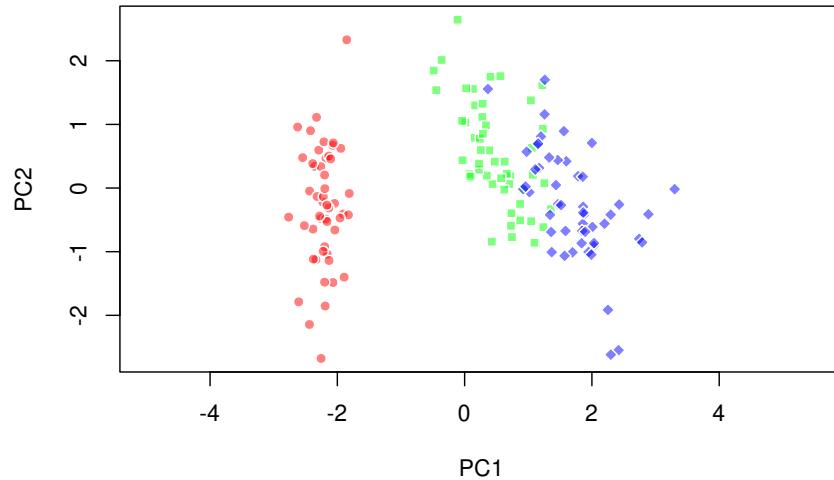


```
# define cor e simbolo por especie
tx <- as.factor(iris$Species)
# simbolos para os niveis
upchs <- 21:23
# cores para os niveis
cores <- rainbow(length(levels(tx)), alpha = 0.5)

# gera a figura
fig <- ordiplot(meu.pca, type = "n")
```



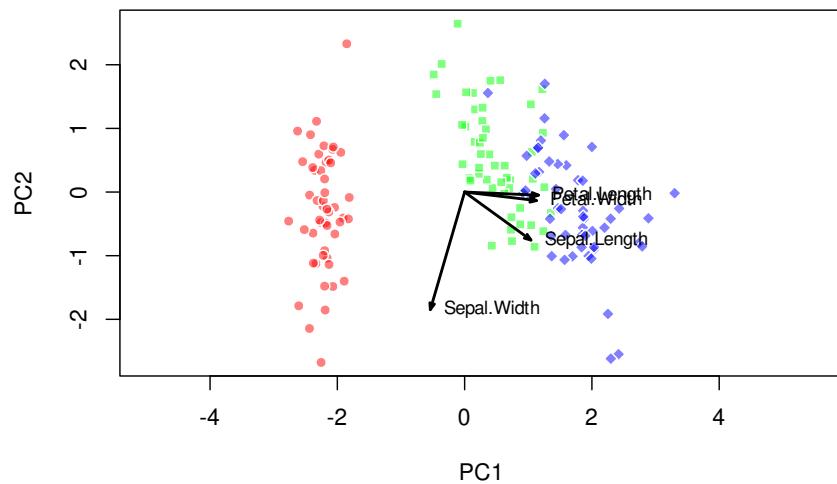
```
# adiciona os pontos de cada linha
ordiplot(meu.pca, type = "n")
points(fig, "sites", pch = upchs[as.numeric(tx)], bg = cores[as.numeric(tx)], col = "white")
```



```
# pega os scores das variaveis
# que mostram os efeitos das variáveis usadas
mls <- vegan::scores(meu.pca, display = "species")
mls
```

	PC1	PC2	PC3	PC4
Sepal.Length	0.5210659	-0.3774176	0.7195664	0.2612863
Sepal.Width	-0.2693474	-0.9232957	-0.2443818	-0.1235096
Petal.Length	0.5804131	-0.0244916	-0.1421264	-0.8014492
Petal.Width	0.5648565	-0.0669420	-0.6342727	0.5235971

```
# plota flexas para esses efeitos
ordiplot(meu.pca, type = "n")
points(fig, "sites", pch = upchs[as.numeric(tx)], bg = cores[as.numeric(tx)], col = "white")
ft <- 2 # para aumentar as flexas um pouco
arrows(0, 0, mls[, 1] * ft, mls[, 2] * ft, length = 0.05, angle = 20, col = "black", lwd = 2)
text(mls[, 1] * ft, mls[, 2] * ft, labels = rownames(mls), col = "black", cex = 0.8, pos = 4)
```





A

Base R vs. Tidyverse

A linguagem R completou 20 anos de idade neste mês de março de 2020, e a cada dia se torna mais popular. Pesquisa recente¹ publicada no mês de março feita pelo TIOBE Programming Community Index, um site que agrupa indicadores sobre a popularidade de linguagens de programação, posiciona a linguagem como a 11º entre as mais populares no mundo. A TIOBE agrupa dados do Google, Yahoo, Bing, Wikipedia, Youtube, Badu, e Amazon, para gerar estes resultados, que são apresentados mensalmente.

Parte dessa popularidade é relativamente recente, podendo ser verificada a partir do ano de 2014. Coincidência ou não, 2014 é o ano de surgimento do pacote `dplyr`, o primeiro de uma série de pacotes que coletivamente vieram a se tornar conhecidos como *Tidyverse*.

A.1 O que é o Tidyverse?²

O Tidyverse³ é um conjunto de pacotes de R desenvolvidos para a ciência de dados. Todos os pacotes compartilham uma mesma filosofia e gramática da linguagem. Por exemplo, a estrutura das funções é sempre a mesma:

- **o primeiro argumento sempre é `data`**, isto é, você deve sempre apresentar os dados neste local. Já que o universo destes pacotes

¹<https://fossbytes.com/most-popular-programming-languages/>

²Texto publicado originalmente no blog de R.O.Perdigz (<https://www.ricardoperdigz.com/blog/2020-04-tidyverse/>)

³<https://www.tidyverse.org/>

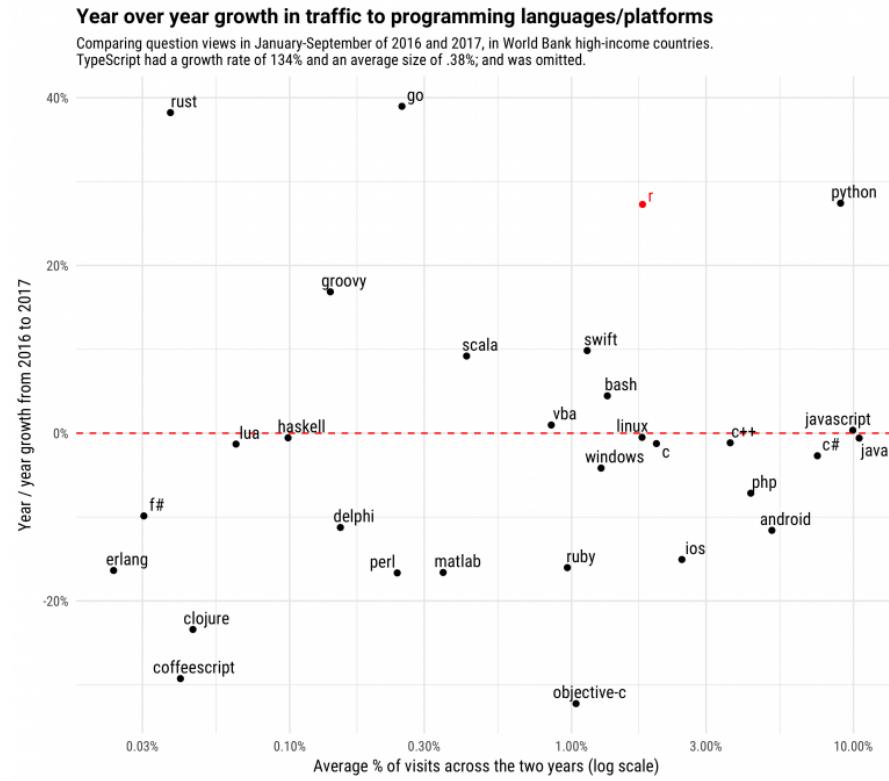


Figura A.1: Porcentagem de visitas às questões pertinentes a algumas linguagens de programação na plataforma *Stack Overflow*. Dados obtidos apenas de países desenvolvidos segundo o Banco Mundial. Nota-se o aumento quase linear referente à linguagem R a partir do ano 2014. Figura extraída desta postagem: <https://stackoverflow.blog/2017/10/10/impressive-growth-r/>

é focado em dados na forma de uma tabela, aqui sempre deve ser fornecido um `data.frame`;

- **Argumentos posteriores modificam o `data.frame`**

- por exemplo, na função `select()` do pacote `dplyr`, você deve fornecer os nomes das colunas que deseja *selecionar* no seu conjunto de dados;
- na função `separate_rows` do pacote `tidyr`, você deve fornecer

os nomes das colunas que se deseja separar em uma ou mais colunas além de indicar o separador (por exemplo, você pode ter uma coluna que possui os nomes `Sapotaceae;Burseraceae` e deseja separar isso em duas colunas; você deve indicar que o separador é ;).

- **A função sempre deve retornar um `data.frame`** (existem algumas exceções feitas às funções de alguns pacotes voltados exclusivamente para lidar com vetores, como por exemplo o pacote `purrr`; porém o uso dessas funções é geralmente utilizado dentro de colunas do seu `data.frame`)

Pretendemos aqui apresentar apenas funcionalidades básicas de dois dos pacotes deste universo, e mostrar como se tornam poderosas quando integrados ao mesmo fluxo de trabalho.

A.1.1 `dplyr` e `ggplot2`, símbolos do Tidyverse

Talvez os pacotes mais conhecidos deste universo sejam o `dplyr` e o `ggplot2`. Enquanto o primeiro é especializado na manipulação de dados, o segundo é voltado para a produção de plots. O `dplyr` surgiu com o objetivo de fornecer um conjunto de ferramentas (suas funções!) para uma manipulação eficiente de conjuntos de dados sob a forma de `data.frames`, e rapidamente, ganhou muitos adeptos devido à facilidade de uso de suas funções dentro de uma **nova gramática** para a manipulação de dados (palavras do criador do pacote, Hadley Wickham, em sua postagem de introdução do pacote⁴). Essa **nova gramática** inclui tanto o uso de funções com nomes de verbos (em inglês, vale ressaltar) desenhados para executar muito bem apenas uma ação (Tabela A.1), quanto o uso do que se convencionou chamar de **pipe**, criado para encadear ações da esquerda para a direita, resultando em menos objetos intermediários estocados na área de trabalho e facilitando a leitura do código. Com o uso de verbos como nome de funções e uma sintaxe diferente da tradicionalmente utilizada em R, o pacote ganhou

⁴<https://blog.rstudio.com/2014/01/17/introducing-dplyr/>

muitos adeptos deste sua disponibilização no CRAN em janeiro de 2012. Seguindo o mesmo caminho, o pacote `ggplot2` (Tabela A.2), também do mesmo autor do pacote `dplyr`, porém já de muito mais idade (foi lançado oficialmente em 10 de junho de 2007) se tornou uma referência na produção de gráficos utilizando a linguagem R, ao propor a construção de gráficos por camadas, similar ao utilizado em programas de SIG. Dentro desta nova sintaxe em R, o operador `+` ganhou uma nova função. Nas próximas seções, vamos ver alguns exemplos práticos utilizando esses dois pacotes.

A.2 Usando o `dplyr`

Vamos utilizar o famoso conjunto `iris`⁵ para aprender a manipular os dados com as ferramentas do `dplyr`.

Vamos aprender brevemente como funcionam as principais funções deste pacote (Tabela A.1). Primeiro vamos carregar o pacote para a sessão de trabalho:

```
library("dplyr")
```

A.2.1 Selecionando colunas com `select()`

Agora, vamos utilizar a função `select()` para selecionar colunas. Ela funciona da seguinte maneira:

- primeiro, utiliza-se como primeiro argumento o nome do `data.frame` que se deseja trabalhar; em nosso caso, o `data.frame` se chama `iris`:
`select(iris, ...);`

⁵O famoso conjunto de dados `iris` consiste de observações de comprimentos e larguras de sépalas e pétalas de 3 espécies de *Iris*, um gênero de plantas herbáceas da família Iridaceae. O conjunto de dados possui 150 linhas e 5 colunas. Para quem quiser saber mais sobre esses dados, leia aqui⁶.

Tabela A.1: Principais funções do pacote R ‘dplyr’.

Função	O que faz
select()	seleciona colunas dos dados
filter()	filtra linhas específicas dos dados
arrange()	ordena as linhas do ‘data.frame’
mutate()	cria novas colunas no ‘data.frame’
summarise()	sumariza os dados de acordo com grupos
group_by()	agrupa os dados segundo grupos

- depois, colocamos no lugar de ... o nome das colunas que desejamos selecionar, **sem aspas**. Por exemplo, se quisermos selecionar a coluna das espécie, fazemos assim:

```
head(  
  select(iris, Species),  
  10  
)
```

Species
setosa

Ou se quisermos a coluna de comprimento de pétala mais a coluna das espécies:

```
head(  
  select(iris, Petal.Length, Species),  
  10  
)
```

Petal.Length	Species
1.4	setosa
1.4	setosa
1.3	setosa
1.5	setosa
1.4	setosa
1.7	setosa
1.4	setosa
1.5	setosa
1.4	setosa
1.5	setosa

Também podemos utilizar funções auxiliares para executar buscas nos nomes das colunas segundo determinados padrões. Entre essas funções auxiliares, destacamos a função `contains()`. Por exemplo, se quisermos selecionar todas as variáveis que contêm “Petal” em seus nomes:

```
head(  
  select(iris, contains("Petal")),  
  10  
)
```

Petal.Length	Petal.Width
1.4	0.2
1.4	0.2
1.3	0.2
1.5	0.2
1.4	0.2
1.7	0.4
1.4	0.3
1.5	0.2
1.4	0.2
1.5	0.1

A.2.2 Filtrando dados com `filter()`

Se desejamos filtrar os dados segundo alguma informação, devemos utilizar a função `filter()`. Por exemplo, se quiser checar os dados de pétalas apenas para a espécie `setosa`, fazemos assim:

```
head(  
  filter(iris, Species == "setosa"),  
  10  
)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa

Ou então quais amostras da espécie `virginica` possuem comprimento de sépala maior que 7 cm:

```
head(  
  filter(iris, Species == "virginica", Sepal.Length > 7),  
  10  
)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
7.1	3.0	5.9	2.1	virginica
7.6	3.0	6.6	2.1	virginica
7.3	2.9	6.3	1.8	virginica
7.2	3.6	6.1	2.5	virginica
7.7	3.8	6.7	2.2	virginica
7.7	2.6	6.9	2.3	virginica
7.7	2.8	6.7	2.0	virginica
7.2	3.2	6.0	1.8	virginica
7.2	3.0	5.8	1.6	virginica
7.4	2.8	6.1	1.9	virginica

E se quisermos adicionar uma coluna em `iris` que consiste na razão entre o comprimento da pétala pelo comprimento da sépala? Chamaremos nossa nova coluna de `razaopetsep`:

```
head(  
  mutate(iris, razaopetsep = Petal.Length / Sepal.Length),  
  10  
)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	razaopetsep
5.1	3.5	1.4	0.2	setosa	0.2745098
4.9	3.0	1.4	0.2	setosa	0.2857143
4.7	3.2	1.3	0.2	setosa	0.2765957
4.6	3.1	1.5	0.2	setosa	0.3260870
5.0	3.6	1.4	0.2	setosa	0.2800000
5.4	3.9	1.7	0.4	setosa	0.3148148
4.6	3.4	1.4	0.3	setosa	0.3043478
5.0	3.4	1.5	0.2	setosa	0.3000000
4.4	2.9	1.4	0.2	setosa	0.3181818
4.9	3.1	1.5	0.1	setosa	0.3061224

A.2.3 Destrinchando as funções `group_by` e `summarise`

As funções `group_by` e `summarise` resumem o propósito do pacote `dplyr`, pois permitem em poucas linhas de comando sumariar os dados, e partem do princípio, muito presente no R através das funções da família `apply`, chamado **split-apply-combine** que, em tradução livre, pode ser entendido como uma sequência lógica de ação: quebre em grupos, aplique uma função, e combine os resultados. Vamos partir para o uso dessas funções agrupando os dados em função da coluna `Species` e calculando a média do comprimento das pétalas (variável `Petal.Length`):

```
iris_grouped <- group_by(iris, Species)
iris_sumario <- summarise(iris_grouped, petala_l_media = mean(Petal.Length, na.rm = TRUE))
iris_sumario
```

Species	petala_l_media
setosa	1.462
versicolor	4.260
virginica	5.552

Vamos destrinchar o que fizemos acima. A função `group_by` os dados em função de alguma ou algumas variáveis. Essa função geralmente

é utilizada em conjunto com a função `summarise` para gerar sumários estatísticos de uma ou mais variáveis.

```
head(  
  group_by(iris, Species),  
  10  
)
```

```
## # A tibble: 10 × 5  
## # Groups:   Species [1]  
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
##       <dbl>      <dbl>      <dbl>      <dbl> <fct>  
## 1     5.1        3.5        1.4       0.2  setosa  
## 2     4.9        3.0        1.4       0.2  setosa  
## 3     4.7        3.2        1.3       0.2  setosa  
## 4     4.6        3.1        1.5       0.2  setosa  
## 5     5.0        3.6        1.4       0.2  setosa  
## 6     5.4        3.9        1.7       0.4  setosa  
## 7     4.6        3.4        1.4       0.3  setosa  
## 8     5.0        3.4        1.5       0.2  setosa  
## 9     4.4        2.9        1.4       0.2  setosa  
## 10    4.9        3.1        1.5       0.1  setosa
```

Vejam que, acima das linhas do conjunto de dados, há a seguinte sentença:

```
## # Groups:   Species [1]
```

Ela informa que o objeto gerado a partir de `group_by` está agrupado ao redor da variável `Species`. Pensando no pacote base, é como pensar que a variável `Species` é o argumento `INDEX` da função `tapply()`: todos os cálculos a partir desse objeto ocorrerão em função dessa variável. Após agrupar os dados, nós colocamos esse `data.frame` agrupado via `group_by()`, `iris_grouped <- group_by(iris, Species)`, como primeiro argumento da função `summarise()` para então calcular a média do comprimento de pétala:

```
iris_grouped <- group_by(iris, Species)
summarise(iris_grouped, petala_l_media = mean(Petal.Length, na.rm = TRUE))
```

Species	petala_l_media
setosa	1.462
versicolor	4.260
virginica	5.552

A partir desse mesmo `data.frame` agrupado, `iris_grouped`, podemos responder várias outras perguntas:

Quantas amostras por espécies existem nesse conjunto de dados?

Utilizaremos a função `n()`, que pertence ao mesmo pacote `dplyr`, para contar o número de grupos. Vejamos:

```
summarise(iris_grouped, n())
```

Species	n()
setosa	50
versicolor	50
virginica	50

Médias de comprimento de sépalas e pétalas

```
summarise(iris_grouped, sepala_l_media = mean(Sepal.Length, na.rm = TRUE))
```

Species	sepala_l_media
setosa	5.006
versicolor	5.936
virginica	6.588

```
summarise(iris_grouped, petala_l_media = mean(Petal.Length, na.rm = TRUE))
```

Species	petala_l_media
setosa	1.462
versicolor	4.260
virginica	5.552

Todas as operações anteriores na mesma linha de comando:

```
iris_sumario <- summarise(iris_grouped, N = n(), sepala_l_media = mean(Sepal.Length, na.rm = TRUE), petala_l_media = mean(Petal.Length, na.rm = TRUE))
head(iris_sumario, 10)
```

Species	N	sepala_l_media	petala_l_media
setosa	50	5.006	1.462
versicolor	50	5.936	4.260
virginica	50	6.588	5.552

A.3 O operador %>% e o encadeamento de ações

Notem que nada do que vimos até aqui parece ser muito relevante se comparamos com o que pode ser feito com o pacote `base` do R. Vejamos:

```
# Queremos selecionar colunas? Operadores '$' e `[]` dao conta
head(
  iris[, which(names(iris) == "Species")],
  10
)
```

```
## [1] setosa setosa setosa setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

```
head(
  iris[, "Sepal.Length"],
  10
)
```

```
## [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
```

```
# Filtrar linhas? Vetores lógicos em conjunto com o operador `[]` em um data.frame resolvem o
head(
  iris[iris$Species == "virginica" & iris$Sepal.Length > 7, ],
  10
)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
103	7.1	3.0	5.9	2.1	virginica
106	7.6	3.0	6.6	2.1	virginica
108	7.3	2.9	6.3	1.8	virginica
110	7.2	3.6	6.1	2.5	virginica
118	7.7	3.8	6.7	2.2	virginica
119	7.7	2.6	6.9	2.3	virginica
123	7.7	2.8	6.7	2.0	virginica
126	7.2	3.2	6.0	1.8	virginica
130	7.2	3.0	5.8	1.6	virginica
131	7.4	2.8	6.1	1.9	virginica

```
# Ou podemos filtrar também usando a função `subset`:
head(
  subset(iris, Species == "virginica" & iris$Sepal.Length > 7),
  10
)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
103	7.1	3.0	5.9	2.1	virginica
106	7.6	3.0	6.6	2.1	virginica
108	7.3	2.9	6.3	1.8	virginica
110	7.2	3.6	6.1	2.5	virginica
118	7.7	3.8	6.7	2.2	virginica
119	7.7	2.6	6.9	2.3	virginica
123	7.7	2.8	6.7	2.0	virginica
126	7.2	3.2	6.0	1.8	virginica
130	7.2	3.0	5.8	1.6	virginica
131	7.4	2.8	6.1	1.9	virginica

```
# Criar novas colunas? Podemos atribuir novas colunas a qualquer data.frame existente usando o
iris_novo <- iris
iris_novo$razaopetsep <- iris_novo$Petal.Length / iris_novo$Sepal.Length
head(
  iris_novo,
  10
)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	razaopetsep
5.1	3.5	1.4	0.2	setosa	0.2745098
4.9	3.0	1.4	0.2	setosa	0.2857143
4.7	3.2	1.3	0.2	setosa	0.2765957
4.6	3.1	1.5	0.2	setosa	0.3260870
5.0	3.6	1.4	0.2	setosa	0.2800000
5.4	3.9	1.7	0.4	setosa	0.3148148
4.6	3.4	1.4	0.3	setosa	0.3043478
5.0	3.4	1.5	0.2	setosa	0.3000000
4.4	2.9	1.4	0.2	setosa	0.3181818
4.9	3.1	1.5	0.1	setosa	0.3061224

```
# Sumariar resultados - Aqui temos um pouco mais de trabalho, porem nada muito complexo
iris_count <- as.data.frame(table(iris$Species))
names(iris_count) <- c("Species", "N")
iris_sumario2 <- cbind(iris_count, sepala_c_media = tapply(iris$Sepal.Length, iris$Species, "mean"))
head(
  iris_sumario2,
  10
) # comparem o resultado do objeto `iris_sumario2` com os de `iris_sumario` criado com as funçõ
```

	Species	N	sepala_c_media	sepala_l_media	petala_c_media	petala_l_media
setosa	setosa	50	5.006	3.428	1.462	0.24
versicolor	versicolor	50	5.936	2.770	4.260	1.32
virginica	virginica	50	6.588	2.974	5.552	2.02

O operador `%>%` foi introduzido no R por meio do pacote `magrittr`, de autoria de Stefan Milton Bache, com o intuito de encadear ações na manipulação de data.frames e facilitar a leitura do código. Segundo palavras do próprio autor⁷, o operador `%>%` modifica semanticamente o código em R e o torna mais intuitivo tanto na escrita quanto na leitura. Será? Vamos tentar entender isso na prática. Vamos retomar os exemplos acima com a introdução do operador `%>%` e usá-lo para efetuar dois conjuntos de comandos, expostos abaixo:

⁷<https://magrittr.tidyverse.org/articles/magrittr.html>

A.3.1 Conjunto de comandos 1

```
# Chamar o data.frame `iris`, então...
# Selecionar as colunas `Species`, `Petal.Length`, e `Sepal.Length`, então ...
# Agrupar os dados em função de `Species`, então ...
# Sumariar os dados para obter o número de observações por grupo, nomeando esta variável como `N`;
# Atribui o resultado dessa operação a um objeto chamado `res1`
```

A.3.2 Conjunto de comandos 2

```
# Chamar o data.frame `iris`, então...
# Selecionar as colunas `Species`, `Petal.Length`, e `Sepal.Length`, então ...
# Filtrar os dados para conter apenas a espécie `virginica` e espécimes com comprimento de sépala m...
# Criar uma nova coluna chamada `razaopetsep` que contenha a razão entre os comprimentos de pétala e ...
# Sumariar os dados para obter o número total de observações, nomeando esta variável como `N`; obtém-se ...
a como `media_razaopetsep`, então ...
# Atribui o resultado dessa operação a um objeto chamado `res2`
```

Primeiramente, carreguemos o pacote `magrittr`:

```
library("magrittr")
```

Executando o conjunto de **comandos 1**, temos:

```
# Chamar o data.frame `iris`, então...
res1 <-
  iris %>%
  # Selecionar as colunas `Species`, `Petal.Length`, e `Sepal.Length`, então ...
  select(Species, Petal.Length, Sepal.Length) %>%
  # Agrupar os dados em função de `Species`, então ...
  group_by(Species) %>%
  # Sumariar os dados para obter o número de observações por grupo, nomeando esta variável como `N`;
  summarise(
    N = n(),
```

```

petala_l_media = mean(Petal.Length, na.rm = TRUE),
sepala_l_media = mean(Sepal.Length, na.rm = TRUE)
)
res1

```

Species	N	petala_l_media	sepala_l_media
setosa	50	1.462	5.006
versicolor	50	4.260	5.936
virginica	50	5.552	6.588

Fazendo o mesmo com o conjunto de **comandos 2**, temos:

```

# Chamar o data.frame `iris`, então...
res2 <-
  iris %>%
    # Selecionar as colunas `Species`, `Petal.Length`, e `Sepal.Length`, então ...
    select(Species, Petal.Length, Sepal.Length) %>%
    # Filtrar os dados para conter apenas a espécie `virginica` e espécimes com comprimento de s
    filter(Species == "virginica" & Sepal.Length > 7) %>%
    # Criar uma nova coluna chamada `razaopetsep` que contenha a razão entre os comprimentos de
    mutate(
      razaopetsep = Petal.Length / Sepal.Length
    ) %>%
    # Sumariar os dados para obter o número total de observações, nomeando esta variável como `N
    summarise(
      N = n(),
      petala_l_media = mean(Petal.Length, na.rm = TRUE),
      sepala_l_media = mean(Sepal.Length, na.rm = TRUE)
    )
res2

```

N	petala_l_media	sepala_l_media
12	6.3	7.475

Notem que o código fica formatado da maneira que funciona nosso pensamento sobre as ações a serem executadas: pegamos os dados,

efetuamos transformações, e agregamos os resultados, praticamente da mesma maneira que o código é executado. Como diz o autor na vinheta de introdução ao operador `%>%`⁸, é como uma receita, fácil de ler, fácil de seguir (*It's like a recipe - easy to read, easy to follow!*). Em conformidade com este entendimento, sugere-se que leiamos o operador `%>%` como **ENTÃO**, implicando em uma passagem do resultado da ação à esquerda para a função à direita. Por isso, eu fiz questão de incluir em ambos os conjuntos de comandos, **1** e **2**, a palavra `então...` ao fim de cada sentença.

Um ponto importante que deve ser levado em consideração é que o uso do operador `%>%` permite que escondamos o `data.frame` de entrada nas funções. Vejamos na prática para entender. Suponha que nós queiramos selecionar apenas as colunas `Species` e `Petal.Length` de `iris`. Podemos executar isso de duas maneiras, todas com o mesmo resultado:

```
# podemos representar iris de três maneiras utilizando o operador `%>%
iris %>% select(Species, Petal.Length) # como temos feito ate aqui
iris %>% select(., Species, Petal.Length) # explicitamos que `iris` esta dentro de select por
```

Isso pode ficar mais fácil de entender com outro exemplo. Suponha que tenhamos o vetor `meuvetor <- c(1:20)` e queiramos obter o somatório deste vetor. Podemos executar isso de três maneiras utilizando o operador `%>%`:

```
meuvetor <- c(1:20)
meuvetor %>% sum(.) # representando o vetor na forma de um `.`.
meuvetor %>% sum() # deixando a funcao vazia
meuvetor %>% sum() # sem parenteses e sem o `.`. O que?????
```

Todas as maneiras acima executam e geram o mesmo resultado, 210. Essa multiplicidade de maneiras de expor o `data.frame` (ou o vetor

⁸<https://magrittr.tidyverse.org/articles/magrittr.html>

no exemplo acima) é alvo de críticas por parte de alguns estudiosos, devido ao pacote `magrittr` não exigir que o argumento seja explícito quando usamos o operador `%>%` (vejam uma boa argumentação nesta postagem de John Mount⁹).

Vale ressaltar que poderíamos muito bem encadear todas as ações executadas acima sem o operador `%>%`, porém perderíamos a chance de ler o código da esquerda para a direita, oportunidade oferecida pelo uso do operador. Vejamos, usando o conjunto de comandos 2:

```
summarise(  
  mutate(  
    filter(  
      select(iris, Species, Petal.Length, Sepal.Length),  
      Species == "virginica" & Sepal.Length > 7  
    ),  
    razaopetsep = Petal.Length / Sepal.Length  
  ),  
  N = n(),  
  petala_l_media = mean(Petal.Length, na.rm = TRUE),  
  sepala_l_media = mean(Sepal.Length, na.rm = TRUE)  
)
```

Reparam que o código fica mais difícil de ser lido, pois temos de identificar primeiro quem é o `data.frame` que serve de entrada para a função `summarise`. Depois, há outros desafios, como entender o que cada função faz, e em qual ordem. Por fim, o código é lido de dentro para fora, um sentido nada intuitivo. Foi pensando em tornar a leitura do código mais fácil que o autor decidiu criar este operador na linguagem R, uma vez que essa lógica já é extensivamente utilizada em algumas outras linguagens de programação, como F# (representada como `|>` e o bash (e similares) (representada como `|`).

⁹<http://www.win-vector.com/blog/2018/03/r-tip-make-arguments-explicit-in-magrittr-dplyr-pipelines/>

Tabela A.2: Principais funções do pacote R ‘ggplot2’.

Função	O que faz
ggplot()	Recebe os dados a serem plotados
geom_point()	Plota um gráfico de barra
geom_boxplot()	Plota um diagrama de caixa
aes()	Estética do gráfico
xlab()	Modifica o texto do eixo X
ylab()	Modifica o texto do eixo Y
ggtitle()	Adiciona o título do gráfico
facet_wrap()	Divide os gráficos segundo categoria especificada

A.3.3 Resumo do operador %>%:

- transforma a leitura do código da **esquerda** para a **direita**;
- evita a criação de muitos objetos intermediários na sessão de trabalho;
- facilita a leitura do código, pois transforma a própria escrita em uma receita.

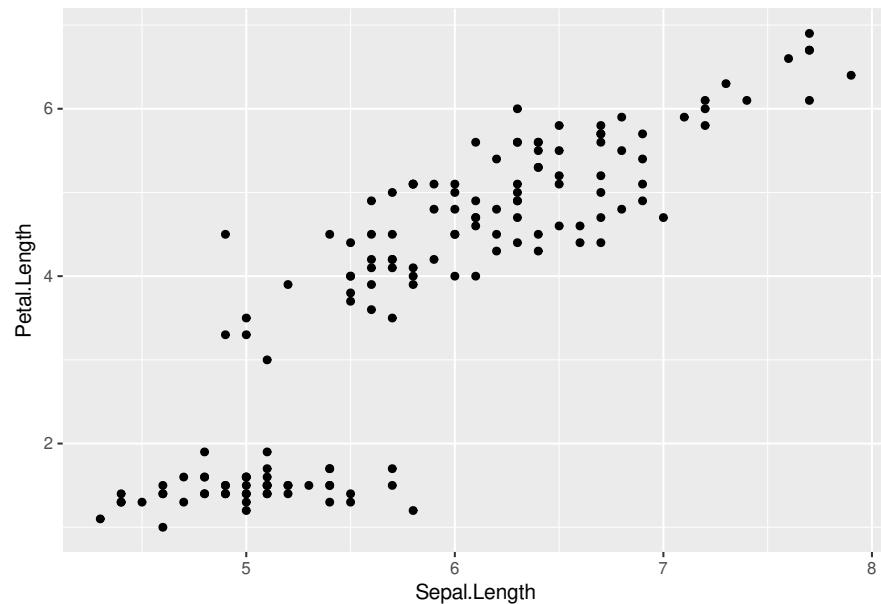
A.4 Usando o ggplot2

O pacote `ggplot2` funciona de maneira diferente da função `plot()` do pacote `base` do R, pois trabalha em camadas. Similarmente ao pacote `dplyr`, começamos com o `data.frame` que desejamos plotar, contudo, passos posteriores são bem diferentes, e se assemelham mais ao uso do operador `%>%` do pacote `magrittr`. No `ggplot2`, utilizamos o operador `+` para adicionar as camadas.

As principais funções do pacote estão exemplificadas na tabela A.2. A função básica do pacote é `ggplot()`: nela, informamos nosso conjunto de dados no primeiro argumento. Após o primeiro

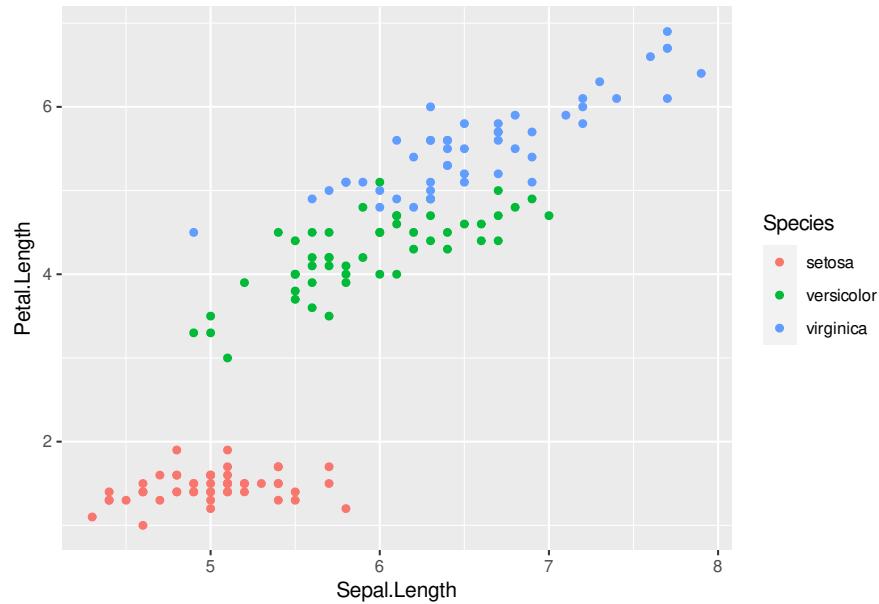
passo, fazemos uso de funções para plotar dados em forma de um espalhamento (scatterplots usando a função `geom_point()`), gráficos de barra (`geom_bar`), diagramas de caixa (`geom_boxplot()`), entre outras. Vejamos na prática como funciona:

```
# um grafico de espalhamento da variavel Sepal.Length no eixo X e Petal.Length no eixo Y utilizando ggplot
ggplot(iris) +
  geom_point(aes(x = Sepal.Length, y = Petal.Length))
```



Dentro das funções que plotam os dados efetivamente (e.g., `geom_point()`, `geom_boxplot()`), devemos sempre usar a função `aes()`: nela inserimos os eixos `x` e `y`, informando sempre o nome das colunas *sem aspas*. Se quisermos colorir os pontos em função das espécies, fazemos:

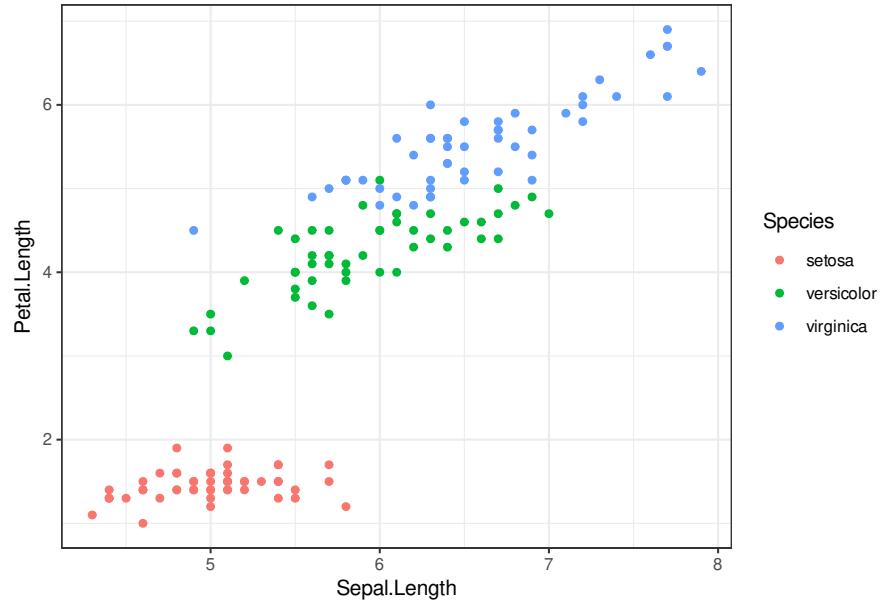
```
ggplot(iris) +
  geom_point(aes(x = Sepal.Length, y = Petal.Length, color = Species))
```



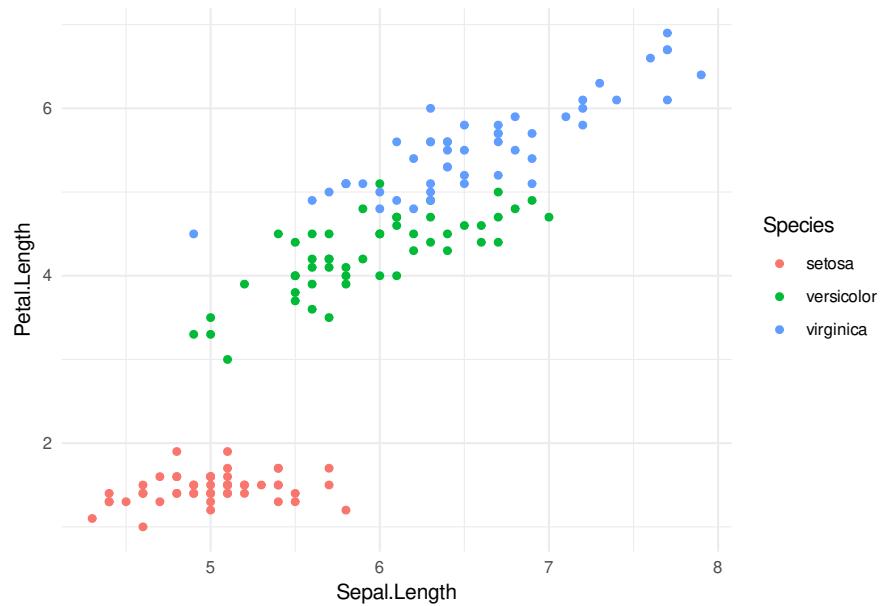
Por trabalhar em camadas, podemos atribuir os resultados dessas operações a objetos. Por exemplo, vamos passar o resultado da ação acima para um objeto `meugrafico` e mudar os temas do gráfico:

```
meugrafico <- ggplot(iris) +
  geom_point(aes(x = Sepal.Length, y = Petal.Length, color = Species))
```

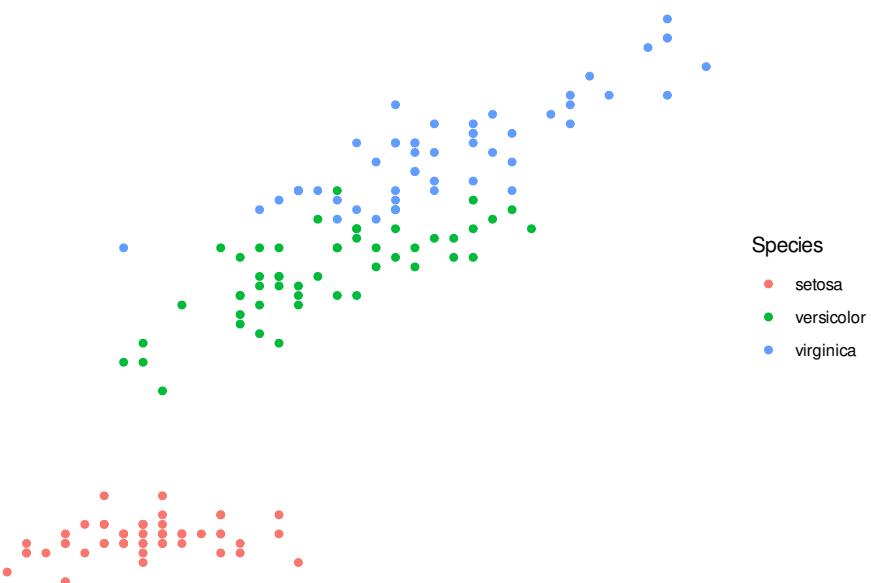
```
meugrafico + theme_bw() # Existem varios outros temas pre-definidos no ggplot2
```



```
meugrafico + theme_minimal() # Para utilizar os outros temas, é só verificar o help de funções
```

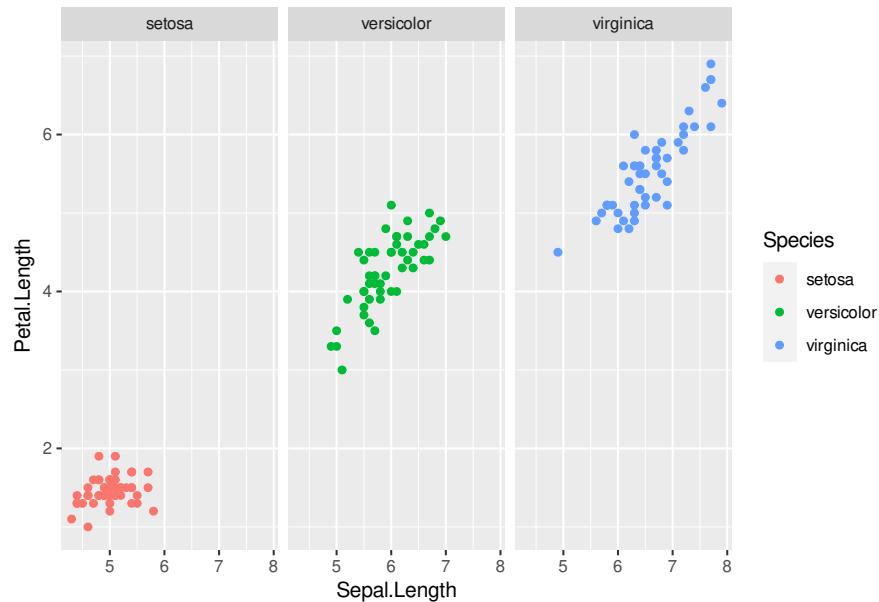


```
meugrafico + theme_void() # - existem, por exemplo, os temas theme_grey, theme_classic, theme_
```



Podemos facilmente também gerar um gráfico para cada espécie utilizando a função `facet_wrap()`:

```
meugrafico + facet_wrap(~Species)
```



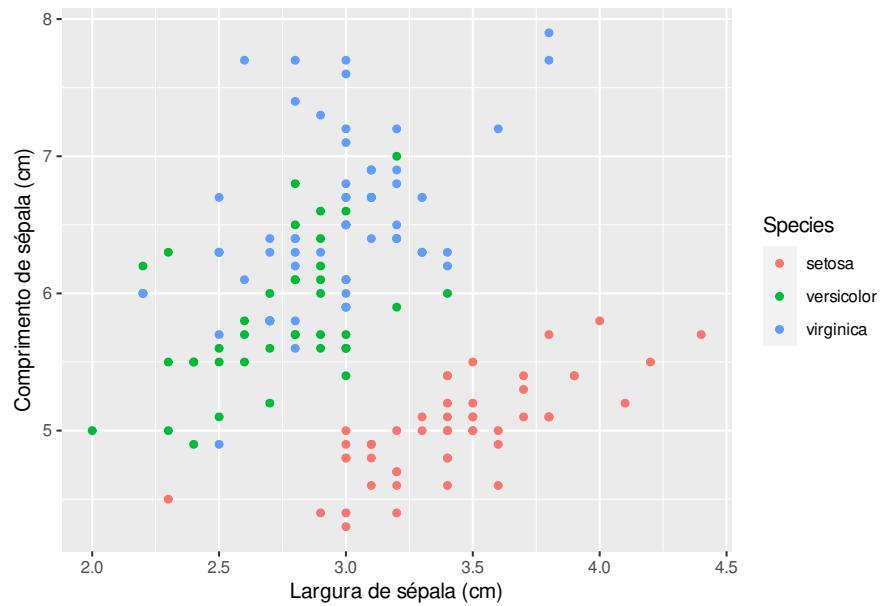
Não temos a intenção de cobrir todo o uso do pacote `ggplot2` nesta postagem. Existem muitas páginas com excelentes tutoriais na internet que podem ser visitadas para um maior aprofundamento nas ferramentas deste pacote (vejam abaixo na seção **Para saber mais**). Queremos aqui demonstrar o uso concomitante do pacote `ggplot2` dentro de uma linha de trabalho associada ao pacote `dplyr`. Passemos para a seção abaixo.

A.5 `dplyr` e `ggplot2` em conjunto

Durante uma análise exploratória de dados, muitas perguntas surgem com a análise de gráficos simples, que podemos criar com poucas linhas de comando. Com os comandos ensinados nos passos anteriores, e novamente utilizando o conjunto de dados `iris`, vamos fazer uma exploração muito breve nesses dados.

A.5.1 Gráfico de espalhamento

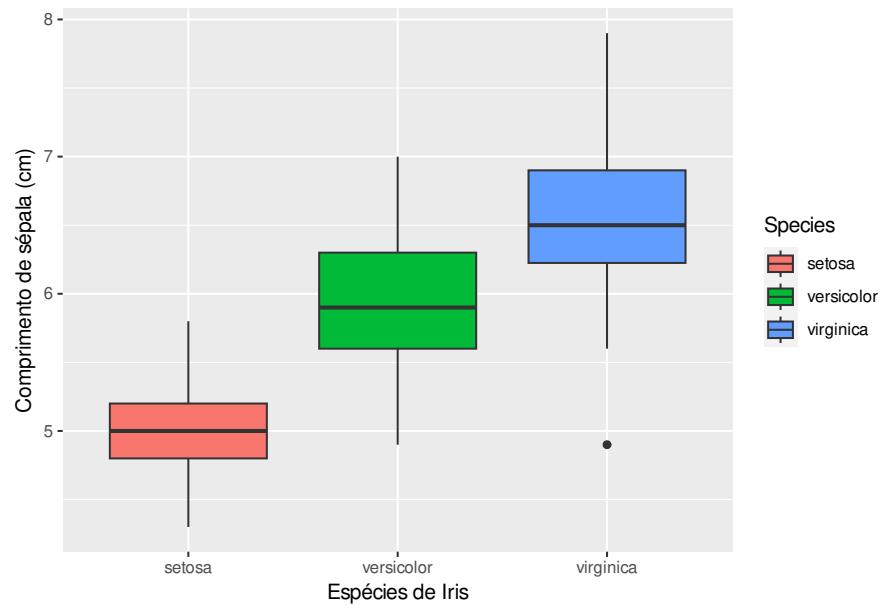
```
iris %>%
  select(Species, Sepal.Width, Sepal.Length) %>%
  ggplot(.) + # lembrem-se que o data.frame com colunas selecionadas acima aqui é representado por .
  geom_point(aes(x = Sepal.Width, y = Sepal.Length, color = Species)) +
  xlab("Largura de sépala (cm)") +
  ylab("Comprimento de sépala (cm)")
```



A.5.2 Diagrama de caixas

```
iris %>%
  select(Species, Sepal.Length) %>%
  ggplot(.) +
  geom_boxplot(aes(x = Species, y = Sepal.Length, fill = Species)) +
```

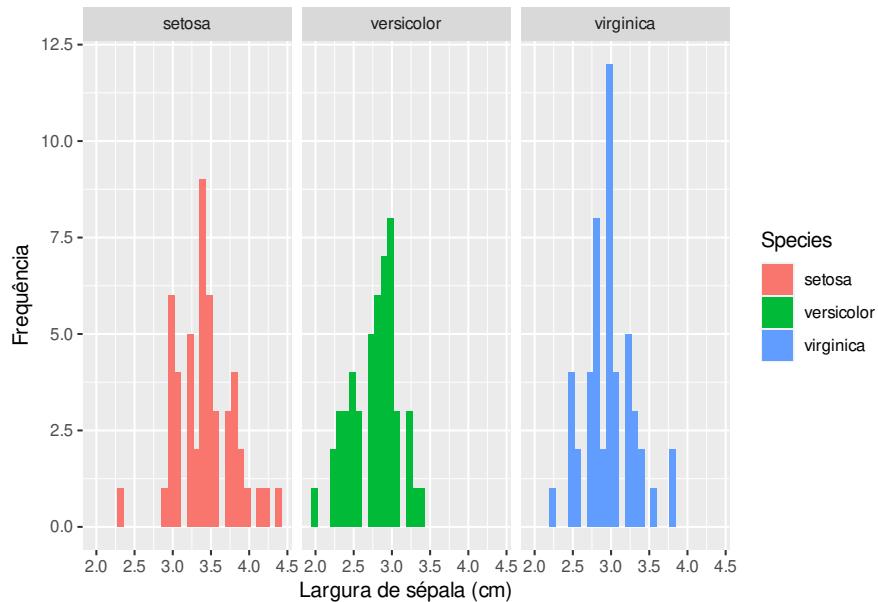
```
xlab("Espécies de Iris") +  
ylab("Comprimento de sépala (cm)")
```



A.5.3 Histograma

```
iris %>%  
  select(Species, Sepal.Width) %>%  
  ggplot(.) +  
  geom_histogram(aes(x = Sepal.Width, fill = Species)) +  
  xlab("Largura de sépala (cm)") +  
  ylab("Frequência") +  
  facet_wrap(~Species)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



A.6 Comparando o mesmo conjunto de ações entre base R e dplyr

Abaixo, para efeitos de comparação, executamos as mesmas ações usando os pacotes `base` e `dplyr`; ao utilizar o `dplyr`, também fizemos uso do operador `%>%`.

```
# para avaliar os objetos criados no ambiente de trabalho, vamos apagar tudo da area de trabalho
rm(list = ls())
# filtra os dados em iris
## quem tem sepala menor que 6 cm e tem petala maior que 5 cm?
iris2 <- subset(iris, Sepal.Length < 6 & Petal.Length < 5)
head(iris2, 10)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa

```
# checa as dimensões do novo objeto
dim(iris2)
```

```
## [1] 78 5
```

```
# ordena os dados segundo comprimento da petala
iris_ord <- iris2[order(iris2$Petal.Length), ]
head(iris_ord, 10)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
23	4.6	3.6	1.0	0.2	setosa
14	4.3	3.0	1.1	0.1	setosa
15	5.8	4.0	1.2	0.2	setosa
36	5.0	3.2	1.2	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
17	5.4	3.9	1.3	0.4	setosa
37	5.5	3.5	1.3	0.2	setosa
39	4.4	3.0	1.3	0.2	setosa
41	5.0	3.5	1.3	0.3	setosa
42	4.5	2.3	1.3	0.3	setosa

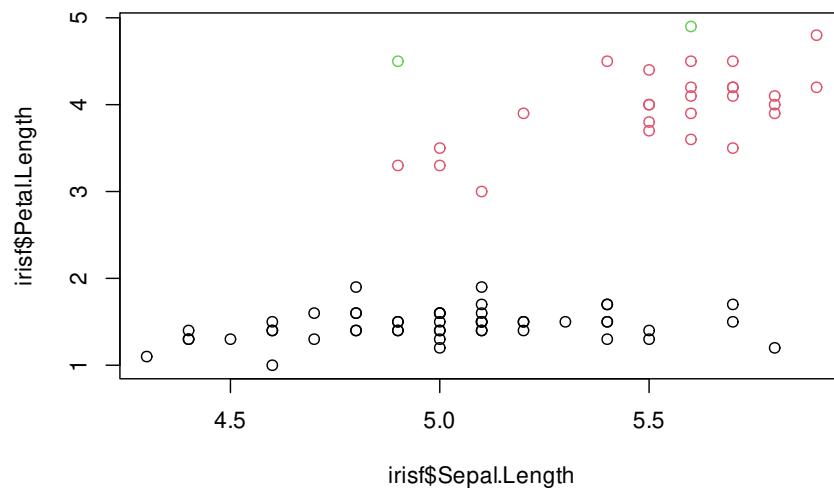
```
# muda a ordem das colunas, colocando a coluna Species primeiro
colsp <- which(colnames(iris_ord) == "Species") # qual a posicao da coluna Species?
irisf <- iris_ord[, c(colsp, (which(!(1:ncol(iris_ord)) %in% colsp)))]
# cria uma nova coluna de razao entre comprimento da sepala e comprimento da petala
irisf$razao_sepall_petall <- irisf$Sepal.Length / irisf$Petal.Length
head(irisf, 10)
```

	Species	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	razao_sepall_petall
23	setosa	4.6	3.6	1.0	0.2	4.600000
14	setosa	4.3	3.0	1.1	0.1	3.909091
15	setosa	5.8	4.0	1.2	0.2	4.833333
36	setosa	5.0	3.2	1.2	0.2	4.166667
3	setosa	4.7	3.2	1.3	0.2	3.615385
17	setosa	5.4	3.9	1.3	0.4	4.153846
37	setosa	5.5	3.5	1.3	0.2	4.230769
39	setosa	4.4	3.0	1.3	0.2	3.384615
41	setosa	5.0	3.5	1.3	0.3	3.846154
42	setosa	4.5	2.3	1.3	0.3	3.461539

```
# faz um sumario estatistico agrupando as especies e checando min, max, desvio, media e mediana
mean_pl <- aggregate(irisf$Petal.Length, list(Species = irisf$Species), FUN = "mean")
mean_raz <- aggregate(irisf$razao_sepall_petall, list(Species = irisf$Species), FUN = "mean")
sd_pl <- aggregate(irisf$Petal.Length, list(Species = irisf$Species), FUN = "sd")
sd_raz <- aggregate(irisf$razao_sepall_petall, list(Species = irisf$Species), FUN = "sd")
res <- as.data.frame(mean_pl)
res2 <- cbind(res, media_razao = mean_raz[, -1], desvio_pl = sd_pl[, -1], desvio_raz = sd_raz[, -1])
names(res2)[2] <- "media_pl"
res2
```

Species	media_pl	media_razao	desvio_pl	desvio_raz
setosa	1.462000	3.464906	0.1736640	0.4302168
versicolor	3.969231	1.404475	0.4249887	0.1195457
virginica	4.700000	1.115873	0.2828427	0.0381613

```
# plota grafico de pontos com eixo X correspondendo ao comprimento da sepala, e eixo Y ao comprimento da petala
plot(irisf$Sepal.Length, irisf$Petal.Length, col = irisf$Species)
```



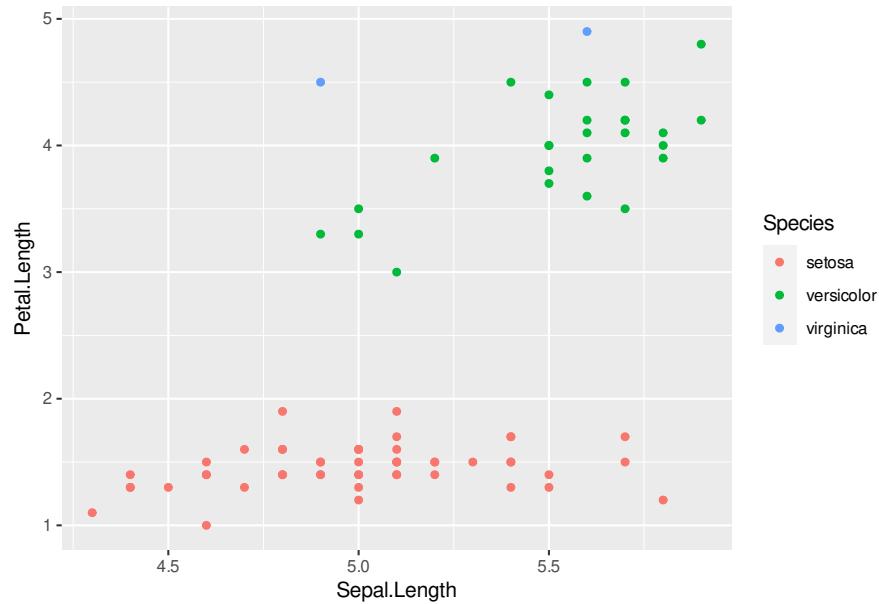
```
# para avaliar os objetos criados no ambiente de trabalho, vamos apagar tudo da area de trabalho
rm(list = ls())
# chama os pacotes tidyverse
library("dplyr") # o pacote para manipular dados
library("ggplot2") # o pacote para plotar graficos
# utilizando a filosofia tidyverse, utilizarei o pipe (" %>% ")
iris2t <- # salvamos o objeto como `iris2t` para diferenciar do salvo acima
  iris %>% # chama os dados e passa para a funcao abaixo
  filter(Sepal.Length < 6 & Petal.Length < 5) %>% # quem tem sepala menor que 6 cm E petala menor que 5 cm
  arrange(Petal.Length) %>%
  mutate(razao_sepall_petall = Sepal.Length / Petal.Length) %>%
  select(Species, everything()) # reordena as colunas
head(iris2t, 10)
```

Species	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	razao_sepall_petall
setosa	4.6	3.6	1.0	0.2	4.600000
setosa	4.3	3.0	1.1	0.1	3.909091
setosa	5.8	4.0	1.2	0.2	4.833333
setosa	5.0	3.2	1.2	0.2	4.166667
setosa	4.7	3.2	1.3	0.2	3.615385
setosa	5.4	3.9	1.3	0.4	4.153846
setosa	5.5	3.5	1.3	0.2	4.230769
setosa	4.4	3.0	1.3	0.2	3.384615
setosa	5.0	3.5	1.3	0.3	3.846154
setosa	4.5	2.3	1.3	0.3	3.461539

```
# faz um sumario estatistico agrupando as especies e checando min, max, desvio, media e mediana
res_t <- iris2t %>%
  group_by(Species) %>% # agrupa dados pela coluna Species
  summarise( # summariza dados de acordo com os grupo estabelecidos acima
    mean_pl = mean(Petal.Length), # media da coluna comp de petala
    mean_raz = mean(razao_sepall_petall), # media da coluna razao de comp de sepala/petala
    sd_pl = sd(Petal.Length), # desvio do comp da petala
    sd_raz = sd(razao_sepall_petall)
  ) # desvio do comp da sepala
head(res_t, 10)
```

Species	mean_pl	mean_raz	sd_pl	sd_raz
setosa	1.462000	3.464906	0.1736640	0.4302168
versicolor	3.969231	1.404475	0.4249887	0.1195457
virginica	4.700000	1.115873	0.2828427	0.0381613

```
# plota grafico de pontos com eixo X correspondendo ao comprimento da sepala, e eixo Y ao comprimento da petala
tgraf <- ggplot(data = iris2t) +
  geom_point(aes(x = Sepal.Length, y = Petal.Length, color = Species))
tgraf
```



Vejam que os resultados são iguais, com exceção da estética diferente proporcionada pelo pacote `ggplot2`. Porém, há uma mudança notável em como a manipulação dos dados é feita quando utilizamos o operador `%>%` dentro do fluxo de trabalho, como [resumido acima](#): a leitura é feita da esquerda para a direita, podemos reduzir a criação de objetos intermediários na área de trabalho, e o código pode ser lido com mais clareza. Postagens futuras abordarão com mais detalhes outros usos deste operador e de alguns outros inclusos no mesmo pacote, porém com funções levemente diferentes.

A.6.1 E o uso do operador `%>%` com o pacote `base`?

Apesar de termos explorado até aqui o uso do operador `%>%` apenas com o pacote `dplyr`, ele pode ser utilizado com qualquer função no R. Vamos retomar o exemplo da [seção acima](#) com comandos do pacote `base` do R, porém adicionando o operador `%>%` na manipulação dos dados:

```

iris_ord <-
  iris %>%
  subset(Sepal.Length < 6 & Petal.Length < 5) %>% # filtramos os dados com a função subset()
  .[order(.Petal.Length), ] # usamos o `.` para representar o data.frame filtrado no passo anterior
colsp <- which(colnames(iris_ord) == "Species")
irisf <- iris_ord[, c(colsp, (which(!(1:ncol(iris_ord)) %in% colsp)))]
irisf$razao_sepall_petall <- irisf$Sepal.Length / irisf$Petal.Length
head(irisf, 10)

```

	Species	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	razao_sepall_petall
23	setosa	4.6	3.6	1.0	0.2	4.600000
14	setosa	4.3	3.0	1.1	0.1	3.909091
15	setosa	5.8	4.0	1.2	0.2	4.833333
36	setosa	5.0	3.2	1.2	0.2	4.166667
3	setosa	4.7	3.2	1.3	0.2	3.615385
17	setosa	5.4	3.9	1.3	0.4	4.153846
37	setosa	5.5	3.5	1.3	0.2	4.230769
39	setosa	4.4	3.0	1.3	0.2	3.384615
41	setosa	5.0	3.5	1.3	0.3	3.846154
42	setosa	4.5	2.3	1.3	0.3	3.461539

```

mean_pl <- aggregate(irisf$Petal.Length, list(Species = irisf$Species), FUN = "mean")
mean_raz <- aggregate(irisf$razao_sepall_petall, list(Species = irisf$Species), FUN = "mean")
sd_pl <- aggregate(irisf$Petal.Length, list(Species = irisf$Species), FUN = "sd")
sd_raz <- aggregate(irisf$razao_sepall_petall, list(Species = irisf$Species), FUN = "sd")
res <- as.data.frame(mean_pl)
res2 <- cbind(res, media_razao = mean_raz[, -1], desvio_pl = sd_pl[, -1], desvio_raz = sd_raz[, -1])
names(res2)[2] <- "media_pl"
res2

```

Species	media_pl	media_razao	desvio_pl	desvio_raz
setosa	1.462000	3.464906	0.1736640	0.4302168
versicolor	3.969231	1.404475	0.4249887	0.1195457
virginica	4.700000	1.115873	0.2828427	0.0381613

Reparem que, mesmo utilizando o operador `%>%`, algumas

transformações realizadas com **poucas linhas de código** com o pacote `dplyr` em uma cadeia de comandos (o chamado `pipeline` em inglês) não são possíveis com o uso do pacote `base` do R, notadamente a criação de colunas, o rearranjo de colunas, e o sumário de dados.

A.7 Tidyverse, um resumo

David Robinson, autor do pacote `broom`¹⁰, um dos membros do tidyverse, explica em sua postagem **Teach the tidyverse to beginners**¹¹ que sua preferência por ensinar as ferramentas dos pacotes pertencentes ao tidyverse primeiramente a seus alunos se deve à compatibilidade da filosofia de ensino deste universo com o que ele acredita que seja ideal em uma filosofia de ensino: **deve haver uma, preferencialmente apenas uma, maneira óbvia de se fazer**, mote emprestado do guia de 19 princípios da linguagem de programação Python, conhecido como *Zen of Python*¹². Essa filosofia se contrapõe, por exemplo, ao mote da linguagem Perl, que é **há sempre mais de uma maneira de se fazer** (nota pessoal: Isso pode ser observado na linguagem R, são sempre múltiplos os caminhos para se chegar a um resultado).

O mesmo autor também afirma nesta postagem que talvez um fluxo de trabalho comum aos alunos de cursos de ciência de dados seja:

- trabalhar com dados interessantes e reais;
- Criar gráficos informativos e atrativos;
- Chegar a conclusões úteis.

Finalizando, ele conclui dizendo que o *tidyverse* oferece ferramentas a seus usuários que tornam esse caminho mais suave.

O uso do `%>%` oferece outros aspectos a serem considerados no uso dessas ferramentas:

¹⁰<https://github.com/tidymodels/broom>

¹¹<http://varianceexplained.org/r/teach-tidyverse/>

¹²https://en.wikipedia.org/wiki/Zen_of_Python

- Cada passo do operador resolve um problema;
- Cada passo ensina uma função;
- Acelera o uso da linguagem para análise exploratória de dados.

Nem tudo são flores em nenhum dos pontos abordados acima. Há muita controvérsia sobre o caminho a ser tomado em aulas da linguagem R nos dias atuais, que passam pela pergunta: ensinar primeiro o pacote `base` do R e suas funcionalidades básicas, ou começar pelas ferramentas do `tidyverse` (ver mais na seção **Para saber mais**). Não vamos entrar nesse embate nesta postagem. O que podemos afirmar é que qualquer pacote de R, com exceção do pacote `base`, foi construído em cima das funcionalidades deste último, logo, sem este pacote, não estaríamos nem aqui falando de funcionalidades do `dplyr` e afins.

Quisemos aqui apresentar funcionalidades básicas de alguns pacotes que podem ser adotadas ou não pelos alunos, e mostrar como podem ser incorporadas no fluxo diário das manipulações de dados. Caso queiram ver manipulações mais complexas em tabelas utilizando as ferramentas do `tidyverse`, convido-os a checarem estas duas postagens: importando e manipulando dados no R¹³, e um tutorial para gerar o mapa de distribuição de *Macrolobium aracaense* Farroñay¹⁴.

A.8 Para saber mais:

As postagens abaixo variam entre posições favoráveis ou desfavoráveis ao `Tidyverse`. Abrangem desde opiniões pessoais sobre o porquê de não ensinar essas ferramentas aos iniciantes na linguagem a questões de performance computacional desse universo quando comparado com o base R. Leiam e tirem suas próprias conclusões.

¹³<https://www.ricardoperdiz.com/blog/2020-03-obtendo-dados-e-plotando-mapas-no-r-versão-3/>

¹⁴<https://www.ricardoperdiz.com/blog/2019-09-mapa-macrolobium/>

- (4) A Thousand Gadgets: My Thoughts on the R Tidyverse¹⁵ - Opinião pessoal do autor da postagem sobre o porquê de ele não simpatizar nem utilizar nenhuma ferramenta do tidyverse.
- (5) Tidyverse¹⁶ - Página principal deste universo de pacotes;
- (6) R for Data Science¹⁷ - Livro disponível gratuitamente na internet, escrito por Garrett Grolemund e Hadley Wickham, este criador e mente criativa por muitos dos pacotes deste universo
- (7) What is the tidyverse¹⁸ - Postagem introdutória sobre o tidyverse e algumas de suas funcionalidades.
- (8) Why I don't use the tidyverse¹⁹ - opinião pessoal do autor sobre o porquê do mesmo não utilizar nenhum pacote do tidyverse.
- (9) Don't teach built-in plotting to beginners (teach ggplot2)²⁰ - postagem do autor do pacote **broom**, David Robinson, sobre o porquê de ele ensinar seus alunos a manipularem primeiro o **ggplot2** em detrimento das funções gráficas do pacote **base** do R.
- (10) Teach the tidyverse to beginners²¹ - postagem de David Robinson sobre sua opinião pessoal de se ensinar primeiro o tidyverse aos alunos de cursos de introdução ao R.
- (11) Introducing magrittr²² - Vinheta de introdução às funcionalidades do operador %>%

¹⁵<https://towardsdatascience.com/a-thousand-gadgets-my-thoughts-on-the-r-tidyverse-2441d8504433>

¹⁶<https://www.tidyverse.org/>

¹⁷<https://r4ds.had.co.nz/>

¹⁸<https://rviews.rstudio.com/2017/06/08/what-is-the-tidyverse/>

¹⁹<https://www.r-bloggers.com/why-i-dont-use-the-tidyverse/>

²⁰http://varianceexplained.org/r/teach_ggplot2_to_beginners/

²¹<http://varianceexplained.org/r/teach-tidyverse/>

²²<https://magrittr.tidyverse.org/articles/magrittr.html>



B

Baixar e descomprimir um arquivo

Caso você queira baixar e descomprimir um arquivo .zip no próprio R, ao invés de baixá-lo manualmente, siga os comandos abaixo. Utilizaremos um arquivo utilizado na seção 4.2.

```
# grava o endereco do arquivo em um objeto
arq_zip_url <- "https://github.com/LABOTAM/IntroR/blob/main/dados/municipiosshape.zip"
# cria um diretorio temporario
dirtemp <- tempdir()
# cria um arquivo temporario para arquivar esse zip
arq_temp <- tempfile(tmpdir = dirtemp, fileext = ".zip")
# baixa o arquivo para dentro do arquivo temporario
download.file(arq_zip_url, arq_temp)

# Apos executar o comando acima, o arquivo '.zip' é baixado para o objeto `arq_temp`
# agora utilizaremos a funcao `unzip()` para descomprimir o arquivo

# pega o nome do primeiro arquivo dentro do zip == corresponde ao nome da pasta
fname <- unzip(arq_temp, list = TRUE)$Name[1]
nomes_arqs <- unzip(arq_temp, list = TRUE)
class(nomes_arqs) # é um dataframe; a primeira coluna mostra o caminho de cada arquivo presente
str(nomes_arqs)
nomes_arqs

# Descomprime o arquivo zip para a pasta temporaria
unzip(arq_temp, files = nome_pasta, exdir = dirtemp, overwrite = TRUE)
# caminho completo do arquivo extraido
caminho_completo <- file.path(dirtemp, nome_pasta)
caminho_completo
```

```
# lista arquivos dentro do caminho contido em caminho_completo
list.files(paste0(caminho_completo, "./"), all.files = TRUE)
```

C

Aulas em vídeo

Gravamos vídeoaulas para a maior parte do material aqui ofertado. Elas estão agrupadas abaixo para facilitar a visualização e o entendimento dos tópicos abordados.

C.1 Importando e exportando dados

Acesso em: <https://www.youtube.com/embed/3OYkXsoAHS4>

C.2 Criação de vetores

Acesso em: <https://www.youtube.com/embed/qXSZkGoDkIY>

C.3 Classes de vetores e fatores

Acesso em: <https://www.youtube.com/embed/CAXnQpYgJ3Y>

C.4 Sequências numéricas e repetições, operações e funções com vetores

Acesso em: <https://www.youtube.com/embed/PJ02yj0gnWc>

C.5 Listas

Acesso em: <https://www.youtube.com/embed/gBGFslcbfU>

C.6 Criando matrizes

Acesso em: <https://www.youtube.com/embed/cA4ETA6qfB4>

C.7 Criando dataframes, e operações importantes em matrizes e dataframes

Acesso em: <https://www.youtube.com/embed/27HgQHV5zBs>

C.8 Indexação de matrizes e dataframes

Acesso em: <https://www.youtube.com/embed/CJILnDzVviQ>

C.9 Condicionais

Acesso em: https://www.youtube.com/embed/8Z_k02PwQZc

C.10 Funções da família apply()

Acesso em: <https://www.youtube.com/embed/FsOis2251Sw>

C.11 Funções de manipulação de pasta e arquivos

Acesso em: <https://www.youtube.com/embed/TbY5nEQTwLw>

C.12 Funções de manipulação de texto - parte 01

Acesso em: <https://www.youtube.com/embed/at5XpBAT1sI>

C.13 Funções de manipulação de texto - parte 02

Acesso em: <https://www.youtube.com/embed/j6O6e96qmEc>

C.14 Objetos complexos

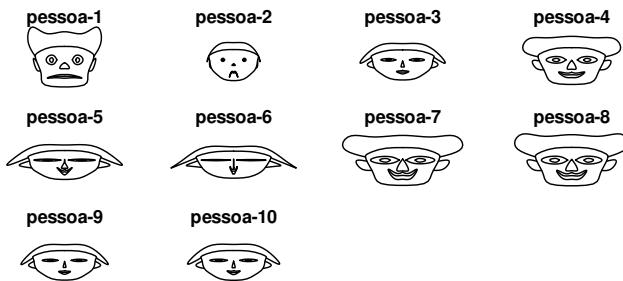
Acesso em: <https://www.youtube.com/embed/hvWJrOkhuwc>

D

Caras de Chernoff

```
#as carinhas de Chernoff são geradas pelo pacote TeachingDemos
library("TeachingDemos")

#uma matriz com variaveis
pessoas = paste("pessoa",1:10,sep="-")
altura = rnorm(10,mean=1.7,sd=0.2)
dedao = rnorm(10,mean=0.05,sd=0.002)
mm = data.frame(altura,dedao)
rownames(mm) = pessoas
faces(mm)
```





Referências Bibliográficas

- APG (2016). An update of the Angiosperm Phylogeny Group classification for the orders and families of flowering plants: APG IV. *Botanical Journal of the Linnean Society*, 181:1––20.
- Batista, J., P.I., P., and Oliveira, A. A. (2009). *Introdução ao R - Uma Apostila online*.
- Bivand, R., Keitt, T., and Rowlingson, B. (2020). *rgdal: Bindings for the Geospatial Data Abstraction Library*. R package version 1.5-18.
- Brownrigg, R. (2018). *maps: Draw Geographical Maps*. R package version 3.3.0.
- Burns, P. (2012). *The R Inferno*. Lulu.com.
- Chalom, A., Salles, M., Prado, P. I., and Adalardo, A. (2012). *notaR - Um sistema para notas automatizadas em cursos que utilizam a linguagem R (Version 2.0)*.
- Chamberlain, S., Szoecs, E., Foster, Z., and Arendsee, Z. (2020). *taxize: Taxonomic Information from Around the Web*. R package version 0.9.99.
- Chambers, J. M. (2008). *Programming with R*. Springer.
- Crawley, M. J. (2007). *The R book*. John Wiley & Sons, Ltd.
- Dowle, M. and Srinivasan, A. (2020). *data.table: Extension of 'data.frame'*. R package version 1.13.4.
- Gotelli, N. J. and Ellison, A. M. (2013). *A Primer of Ecological Statistics*. Sinauer Associates, 2 edition.

- Oksanen, J., Blanchet, F. G., Friendly, M., Kindt, R., Legendre, P., McGlinn, D., Minchin, P. R., O'Hara, R. B., Simpson, G. L., Solymos, P., Stevens, M. H. H., Szoecs, E., and Wagner, H. (2020). *vegan: Community Ecology Package*. R package version 2.5-7.
- Paradis, E., Blomberg, S., Bolker, B., Brown, J., Claramunt, S., Claude, J., Cuong, H. S., Desper, R., Didier, G., Durand, B., Dutheil, J., Ewing, R., Gascuel, O., Guillerme, T., Heibl, C., Ives, A., Jones, B., Krah, F., Lawson, D., Lefort, V., Legendre, P., Lemon, J., Louvel, G., Marcon, E., McCloskey, R., Nylander, J., Opgen-Rhein, R., Popescu, A.-A., Royer-Carenzi, M., Schliep, K., Strimmer, K., and de Vienne, D. (2020). *ape: Analyses of Phylogenetics and Evolution*. R package version 5.4-1.
- Roberts, D. W. (2019). *labdsv: Ordination and Multivariate Analysis for Ecology*. R package version 2.0-1.
- RStudio Team (2020). *RStudio: Integrated Development Environment for R*. RStudio, PBC., Boston, MA.
- Sarkar, D. (2020). *lattice: Trellis Graphics for R*. R package version 0.20-41.
- Tukey, J. W. (1980). We need both exploratory and confirmatory. *The American Statistician*, 34(1):23–25.
- Venables, W. N. and Ripley, B. D. (2002). *Modern Applied Statistics with S*. Springer, New York. ISBN 0-387-95457-0, fourth edition.
- Wickham, H. and Bryan, J. (2019). *readxl: Read Excel Files*. R package version 1.3.1.
- Wickham, H. and Hester, J. (2020). *readr: Read Rectangular Text Data*. R package version 1.4.0.
- Zuur, A. F., Ieno, E. N., and Elphick, C. S. (2010). A protocol for data exploration to avoid common statistical problems. *Methods in Ecology and Evolution*, 1(1):3–14.