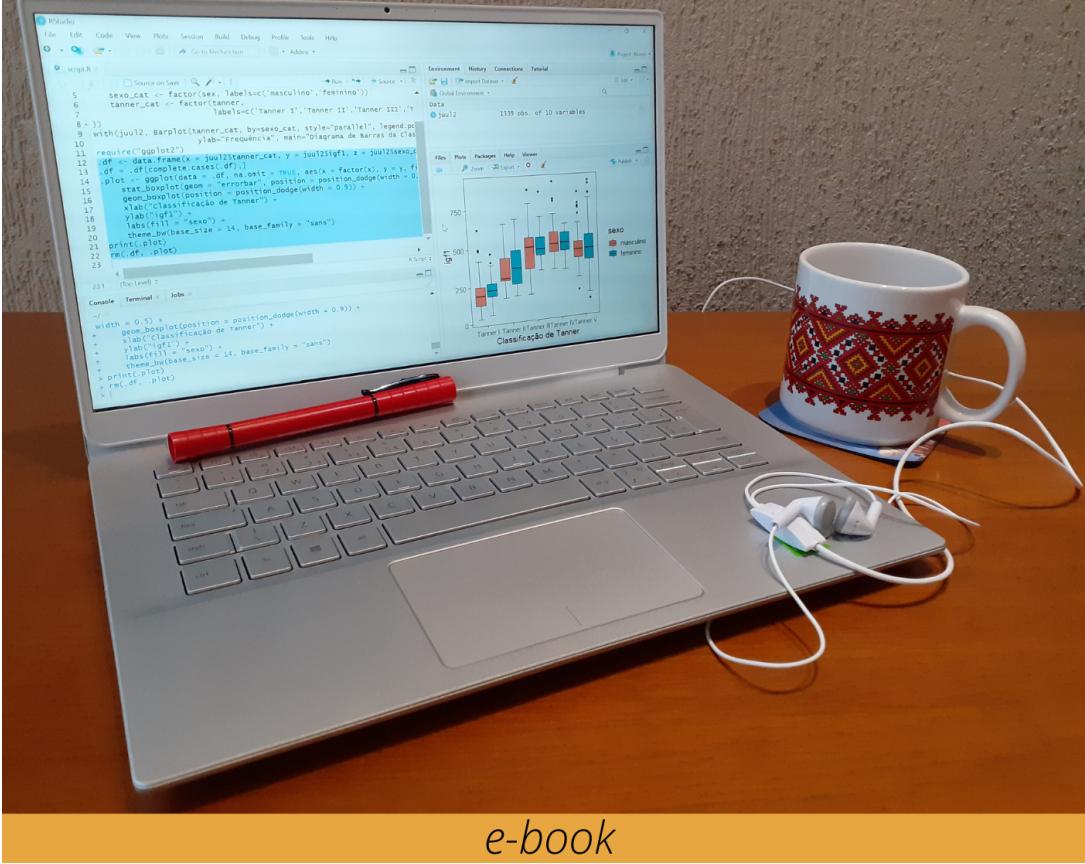


INTRODUÇÃO AO R

SERGIO MIRANDA FREIRE



e-book

Introdução ao R

Sergio Miranda Freire

Introdução ao R

Última atualização
“2021-12-09”

Endereço de acesso
<http://www.lampada.uerj.br/introducaoaoor>

Capa
Rosimary T. Almeida

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Freire, Sergio Miranda
Introdução ao R [livro eletrônico] / Sergio
Miranda Freire. -- Rio de Janeiro : Ed. do Autor,
2021.
PDF

Bibliografia.
ISBN 978-65-00-33953-6

1. Estatística - Programas de computador
2. Estatística matemática 3. R (Linguagem de
programação para computadores) 4. Visualização da
informação I. Título.

21-88249

CDD-005.4

Índices para catálogo sistemático:

1. R : Linguagem de programação : Computadores :
Processamento de dados 005.4

Cibele Maria Dias - Bibliotecária - CRB-8/9427



Exceto onde indicado, esta obra está licenciada com uma Licença Creative Commons Atribuição-Não Comercial 4.0 Internacional.

Prefácio

Este texto tem como objetivo familiarizar o usuário com o R a partir de um ambiente integrado de desenvolvimento, conhecido como *RStudio*, e de um dos pacotes do R, *R Commander*, que oferece uma interface gráfica para acesso a um número de funções do R. Com isso, espera-se tornar menos íngreme a curva de aprendizado do R, especialmente para usuários das áreas não exatas.

O capítulo 1 faz uma breve introdução ao R e mostra os procedimentos para a instalação do R, do *RStudio* e do pacote *R Commander*.

O capítulo 2 faz um breve *tour* de alguns dos recursos de manipulação de dados e gráficos do R, utilizando o *R Commander*. Os detalhes dos procedimentos realizados nesse *tour* serão descritos nos capítulos seguintes.

O capítulo 3 apresenta as classes atômicas e estruturas básicas do R que podem eventualmente ser úteis quando for necessário manipular os dados para realizar análises ou produzir efeitos que não podem ser obtidos diretamente por uma interface gráfica como o *R Commander*. Também esse capítulo é necessário para uma compreensão da sintaxe dos comandos gerados a partir do *R Commander*.

O capítulo 4 mostra como ler conjuntos de dados a partir de arquivos disponíveis em pacotes do R ou importá-los de outras fontes, como planilhas eletrônicas. Ele também mostra como gravar um conjunto de dados num formato que pode ser lido diretamente no R, ou exportá-lo para ser lido em um outro programa.

O capítulo 5 mostra como utilizar o *R Commander* para salvar e carregar scripts e a utilização do *R Markdown* para gerar relatórios em html a partir das análises realizadas.

O capítulo 6 apresenta uma série de funções para a manipulação de variáveis em um conjunto de dados, como criação de novas variáveis, conversão de variáveis numéricas para fator, recodificação de variáveis etc., e a manipulação do próprio conjunto de dados, como criação de outros conjuntos de dados a partir de um subconjunto do arquivo corrente, mudança da disposição do conjunto de dados, etc. Esse é um capítulo fundamental, porque raramente o usuário vai realizar uma análise de dados em um arquivo sem criar ou transformar variáveis originalmente presentes no conjunto de dados.

O capítulo 7 mostra como gerar tabelas de frequências para uma variável e tabelas de frequências conjuntas para duas ou mais variáveis via *R Commander* e via linha de comando.

O capítulo 8 introduz diversos gráficos que podem ser gerados pelo *R Commander*, assim como a eventual utilização da linha de comando para obter efeitos não disponíveis via interface gráfica.

O capítulo 9 mostra como manipular datas e tempos no R e o capítulo 10 introduz o conceito de sessão e apresenta alguns recursos para gerenciar uma sessão do R, como a noção de espaço de busca e diretório de trabalho, remoção de objetos etc.

Finalmente o capítulo 11 apresenta noções básicas sobre as estruturas de controle e funções no R para aqueles que eventualmente venham a desenvolver *scripts* no R.

Sergio Miranda Freire, Rio de Janeiro
Novembro de 2021

Agradecimentos

À Universidade do Estado do Rio de Janeiro (UERJ), que me propiciou as condições para me dedicar ao ensino e pesquisa ao longo de minha carreira.

A Mário João Jr, analista de sistemas do Departamento de Tecnologia da Informação e Educação em Saúde, da Faculdade de Ciências Médicas da UERJ, que zela para que este e-book esteja disponível via Web.

A todos aqueles que, com atos e palavras, trabalham para que a humanidade seja regida pela seguinte máxima: a cada um de acordo com as suas necessidades, de cada um de acordo com as suas possibilidades.

Sumário

Introdução ao R	ii
Prefácio	iii
Agradecimentos	v
1 Introdução	1
1.1 O que é o R?	1
1.2 Vantagens do R	2
1.3 Instalação do R e do pacote <i>R Commander</i>	2
1.4 Instalação do <i>RStudio</i>	4
1.5 Instalação do pacote do <i>R Commander</i> a partir do <i>RStudio</i>	6
2 Um breve <i>tour</i> do R, <i>RStudio</i> e <i>R Commander</i>	12
2.1 Carregando o <i>R Commander</i>	12
2.2 Carregando um conjunto de dados	13
2.3 Visualizando o conteúdo do conjunto de dados	17
2.4 Resumos numéricos	18
2.5 Recodificação de variáveis	19
2.6 Convertendo uma variável numérica para categórica (fator)	20
2.7 Diagrama de Barras	21
2.8 Boxplot	25
2.9 Histograma	27
2.10 Cálculo de nova variável	29
2.11 Diagrama de dispersão	30
2.12 Geração de relatório	32
2.13 Salvando o conjunto de dados	34
3 Estruturas básicas no R	36
3.1 Console do <i>RStudio</i>	37
3.2 Objetos no R	38
3.3 Valores especiais	41
3.4 Funções	42
3.5 Vetores (<i>vector</i>)	43
3.5.1 Acessando elementos de um vetor	45

3.5.2	Gerando sequências - Função <i>seq</i>	47
3.5.3	Gerando repetições: função <i>rep</i>	48
3.6	Listas (<i>list</i>)	49
3.6.1	Extraindo múltiplos elementos de uma lista	50
3.6.2	Extraindo elementos aninhados de uma lista	50
3.7	Nomes	51
3.8	Fatores	52
3.9	Coerção	54
3.10	Matrizes	57
3.10.1	Extraindo células/linhas/colunas/submatrizes de uma matriz	59
3.11	Data frame	61
3.11.1	Acessando os elementos de um <i>data frame</i>	62
3.12	Valores ausentes	64
3.13	Ajuda no R	64
3.14	Instalação de pacotes via linha de comando	65
3.15	Exercícios	66
4	Importando e exportando dados	71
4.1	Formatos de arquivos de dados	71
4.2	Importação de dados do <i>Excel</i> (.xls/.xlsx) usando o <i>RStudio</i>	73
4.3	Importando dados em outros formatos no <i>RStudio</i>	79
4.4	Importando arquivos pelo <i>R Commander</i>	79
4.5	Exportando arquivos pelo <i>R Commander</i>	81
4.6	Salvando e carregando um arquivo de dados no formato do R	83
4.7	Carregando conjuntos de dados disponíveis em pacotes do R	86
4.8	Exercícios	90
5	Utilização do <i>R Commander</i>	91
5.1	Obtendo resumos numéricos	91
5.2	<i>R Markdown</i>	94
5.3	Salvando <i>scripts</i> e arquivos do <i>R Markdown</i>	96
5.4	Executando <i>scripts</i> no <i>R Commander</i>	98
5.5	Exercício	98
6	Manipulação dos dados	99
6.1	Manipulação de variáveis	100
6.1.1	Recodificar variáveis	101
6.1.2	Converter variável numérica para fator	104
6.1.3	Reordenar os níveis dos fatores	105
6.1.4	Computar nova variável	110
6.1.5	Agrupar em classes uma variável numérica	112
6.1.6	Padronizar variáveis	115
6.1.7	Remover variáveis de um conjunto de dados	116
6.1.8	Renomear variáveis	117
6.1.9	Adição do número de observações aos dados	118

6.1.10	Converter variáveis do tipo <i>character</i> para fatores	119
6.2	Manipulação do conjunto de dados	120
6.2.1	Seleção do conjunto de dados ativo	120
6.2.2	Visualização dos dados	121
6.2.3	Renovar conjunto de dados ativo	122
6.2.4	Ajuda no conjunto de dados ativo (se disponível)	123
6.2.5	Variáveis no conjunto de dados ativo	123
6.2.6	Definir nomes dos casos	123
6.2.7	Definir subconjunto de dados ativo	125
6.2.8	Ordenar o conjunto de dados ativo	130
6.2.9	Remoção de linhas do conjunto de dados ativo	132
6.2.10	Remoção de observações com valores ausentes	133
6.2.11	Variáveis agregadas do conjunto de dados ativo	135
6.2.12	Empilhar variáveis no conjunto de dados ativo	136
6.2.13	Remodelar um conjunto de dados do formato longo para o formato largo	137
6.2.14	Remodelar um conjunto de dados do formato largo para o formato longo	140
6.2.15	Converter todas as variáveis do tipo <i>character</i> para o tipo <i>factor</i>	144
6.3	Exercício	145
7	Tabelas de frequências	146
7.1	Introdução	146
7.2	Tabelas de frequências no conjunto de dados <i>stroke</i>	154
7.2.1	Uma única variável categórica	154
7.2.2	Tabelas de frequências para duas variáveis categóricas	156
7.2.3	Obtendo os percentuais de cada célula em relação ao total da linha correspondente	156
7.2.4	Obtendo os percentuais de cada célula em relação ao total da coluna correspondente	158
7.2.5	Obtendo os percentuais de cada célula em relação ao total da tabela	159
7.2.6	Tabelas de frequência para mais de duas variáveis categóricas	164
7.2.7	Entrando diretamente com as frequências das células	171
7.3	Exercício	174
8	Visualização de dados	175
8.1	Diagrama de barras	181
8.2	Usando a linha de comando	188
8.2.1	Especificação dos rótulos dos eixos x e y e do título	188
8.2.2	Alteração dos tamanhos dos eixos X e Y	188
8.2.3	Alteração do título da legenda do diagrama	189
8.2.4	Alteração do espaçamento entre as barras	190
8.2.5	Tamanhos dos rótulos dos eixos X e Y, dos números no eixo Y e das categorias das barras	191
8.2.6	Alteração das categorias da variável do eixo X	192
8.2.7	Alteração das cores	193
8.2.8	Gráfico de barras horizontais	198

8.3	Diagrama de setores, torta ou pizza	198
8.4	Diagrama de caixa (<i>boxplot</i> ou <i>box and whisker plot</i>)	201
8.5	Histograma	205
8.5.1	Histograma de frequência x frequência relativa x densidade de frequência relativa	208
8.5.2	Histograma por grupos	211
8.6	Diagrama de pontos e <i>strip chart</i>	212
8.7	Diagrama de dispersão ou espalhamento	215
8.7.1	Alterando a espessura e cor da linha de regressão e o tipo dos pontos	217
8.8	Salvando gráficos em um arquivo	219
8.9	Recursos gráficos de outros plugins	222
8.10	Exercícios	225
9	Datas e tempo	229
9.1	Datas no R	229
9.2	Tempos no R	231
9.3	Exercício	234
10	Gerenciando uma sessão no R	235
10.1	Listando objetos do <i>workspace</i> - função <i>ls</i>	235
10.2	Espaço de busca de objetos - a função <i>search</i>	237
10.3	Especificando o diretório (pasta) corrente	240
10.4	Manipulando o workspace	242
10.5	Removendo objetos do workspace - a função <i>rm</i>	243
10.6	Histórico de comandos	243
10.7	Atualizando pacotes	246
10.8	Informações sobre a sessão	247
11	Estruturas de controle do fluxo de execução e funções	249
11.1	<i>if-else</i>	249
11.2	Laços com <i>for</i> (repetições)	252
11.3	Laços com <i>while</i>	253
11.4	Laços com <i>repeat</i> e <i>break</i>	254
11.5	<i>next</i>	255
11.6	Laços aninhados	256
11.7	Estrutura básica de uma função	258
11.8	Pareamento dos argumentos	265
11.9	O uso do argumento ...	267
Referências Bibliográficas	270	

Capítulo 1

Introdução

1.1 O que é o R?

O R é uma linguagem e um ambiente para a realização de análises estatísticas e construção de gráficos e é altamente extensível. R é disponível como software livre sob os termos da Licença Pública Geral GNU da Free Software Foundation.

O R é um dialeto da linguagem S, desenvolvida por John Chambers e outros na empresa Bell Telephone *Laboratories*, originalmente parte da *AT&T Corp.* De acordo com Roger Peng (Peng, 2016b), a filosofia da linguagem S foi assim descrita por John Chambers:

“[W]e wanted users to be able to begin in an interactive environment, where they did not consciously think of themselves as programming. Then as their needs became clearer and their sophistication increased, they should be able to slide gradually into programming, when the language and system aspects would become more important.”

Uma importante limitação da linguagem S é que ela estava somente disponível em um pacote comercial, S-PLUS. O R começou a ser desenvolvido por Robert Gentleman e Ross Ihaka (“R & R”), ambos do Departamento de Estatística da Universidade de Auckland, na Nova Zelândia, em 1991.

O primeiro relato da distribuição do R foi em 1993, quando algumas cópias foram disponibilizadas no StatLib, um sistema de distribuição de softwares estatísticos.

Com o incentivo de um dos primeiros usuários deste programa, Martin Mächler (do Instituto Federal de Tecnologia de Zurique, na Suíça), “R & R”, em 1995, lançaram o código fonte do R, disponível por ftp. Em 1997, foi formado um grupo de profissionais que têm acesso ao código fonte do R, possibilitando, assim, a atualização mais rápida do software. Desde então, o R vem ganhando cada vez mais adeptos em todo o mundo (Melo, 2017).

As seguintes referências foram utilizadas para desenvolver este material: (Peng, 2016b), (Peng, 2016a), (Peng, 2016c), (Dalgaard, 2008), (Melo, 2017), (Melo, 2019a) e (Melo, 2019b). As últimas três referências estão disponíveis neste [endereço](#).

Os vídeos referenciados ao longo do texto que não são de minha autoria foram desenvolvidos

pela equipe da professora Luciane Alcoforado, da Universidade Federal Fluminense, e disponibilizados neste [endereço](#).

1.2 Vantagens do R

O R possui as seguintes características:

- além de gratuito, é um programa poderoso, estável e pode ser copiado e distribuído sem nenhum problema;
- é um programa que tem uma longa história, com mais de 25 anos de desenvolvimento;
- é apoiado por uma grande equipe de desenvolvedores em todo o mundo;
- pode ser usado nos sistemas operacionais Windows, Linux e Mac OS;
- amplamente utilizado no meio acadêmico.

As seções seguintes descrevem o passo a passo para a instalação do R, de um programa que oferece um ambiente integrado de desenvolvimento baseado no R (*RStudio*), e de um pacote que fornece uma interface gráfica para a utilização do R (*R Commander - Rcmdr*).

1.3 Instalação do R e do pacote *R Commander*

Uma instalação do R contém uma ou mais bibliotecas ou pacotes. Alguns desses pacotes fazem parte da instalação básica do R. Outros podem ser baixados e instalados, à medida que for necessário.

Ao instalar um pacote, é criada uma pasta no disco do computador com o conteúdo do pacote. Você pode criar o seu próprio pacote.

Um pacote pode conter funções escritas na linguagem R, conjuntos de dados e/ou bibliotecas de códigos compilados em outras linguagens. Eles contêm funções que os usuários não irão utilizar todo o tempo.

Para um usuário iniciante no R, vamos utilizar um pacote que oferece uma interface gráfica para realizar análises estatísticas, criar gráficos, carregar, manipular, importar ou exportar conjuntos de dados. Esse pacote é chamado de *R Commander (Rcmdr)*.

Este [vídeo](#) mostra como instalar o R e o pacote *R Commander* no sistema operacional *Windows*. Este outro [vídeo](#) fornece um breve tour dos recursos do *R Commander*.

De maneira alternativa, são apresentados a seguir os passos para a instalação do R no *Windows*. Neste exemplo, será utilizada a versão 3.5.0 do R. Utilize a última versão que encontrar.

Para instalar o R, siga os seguintes passos:

- Baixe o programa do sitio <http://cran.r-project.org/bin/windows/base/>.
- Execute o programa de instalação R-3.5.0-win.exe. Para isso, basta dar um duplo clique no arquivo (Figura 1.1).

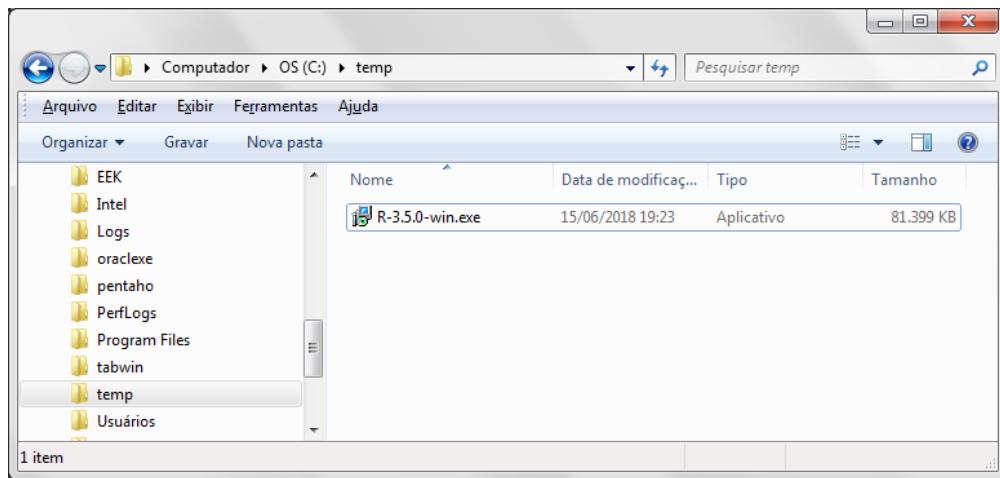


Figura 1.1: Programa para a instalação do R.

- Selecione o idioma e clique em avançar nas próximas telas, aceitando as opções padrões. Ao final, será exibida a tela da figura 1.2. Clique em concluir para encerrar a instalação.

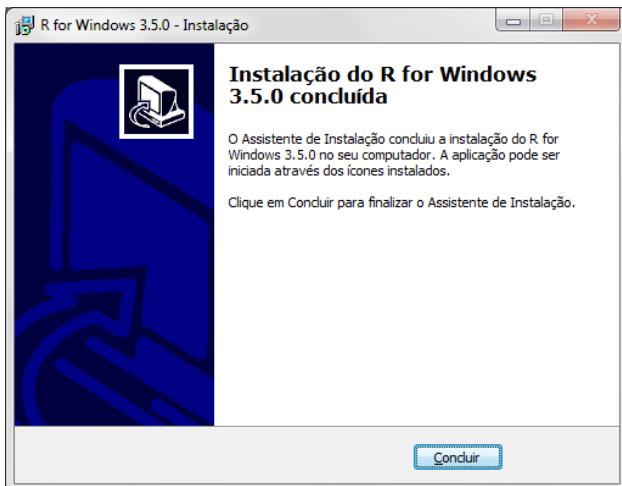


Figura 1.2: Tela de encerramento da instalação do R no Windows.

- O ícone do R aparece na área de trabalho em seu computador (figura 1.3).



Figura 1.3: Ícone do programa R.

- Para executar o R, basta dar um duplo clique neste ícone. Surge então a tela mostrada na figura 1.4.

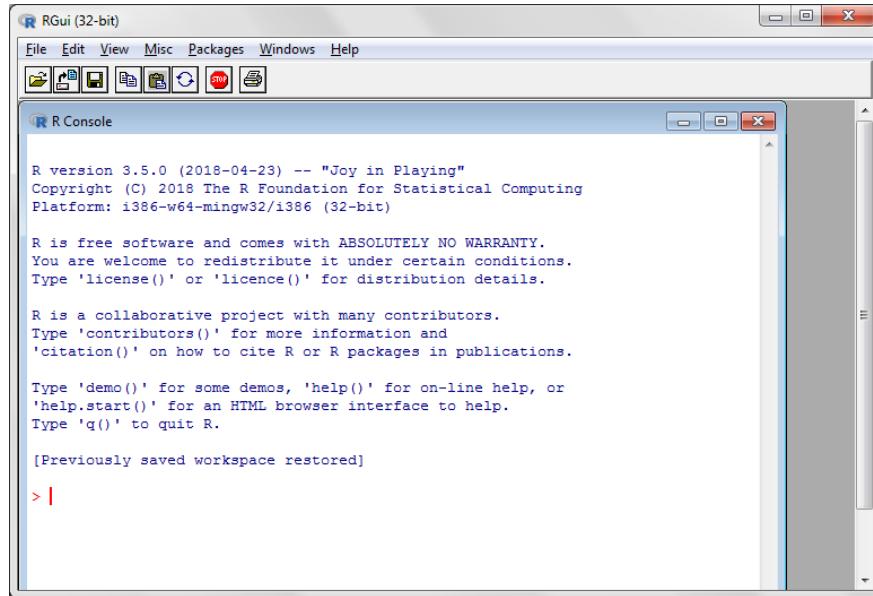


Figura 1.4: Tela inicial do R.

Pronto! O R já pode ser utilizado.

Observação: Para instalar o *R Commander* no *macOS*, é necessário instalar o [XQuartz](#) e também o [Tcl/Tk](#).

XQuartz é uma versão do X11 compatível com o *macOS*. X11 é um sistema gráfico para máquinas Unix.

Tcl/Tk é um kit de ferramentas para o desenvolvimento de aplicações *desktop*.

1.4 Instalação do *RStudio*

O *RStudio* é um ambiente integrado de código aberto para escrever *scripts* no R e utilizar outros recursos baseados no R.

Este [vídeo](#) mostra como instalar e fornece um breve tour dos recursos do *RStudio*.

As figuras a seguir mostram o passo a passo para a instalação do *RStudio*. Existe uma versão gratuita que pode ser instalada a partir do [sítio](#). Nessa página (figura 1.5), selecione o botão *Download RStudio*.

Há diversas versões do *RStudio*. Baixe a versão gratuita (figura 1.6) e, em seguida, o instalador para o seu sistema operacional (figura 1.7).

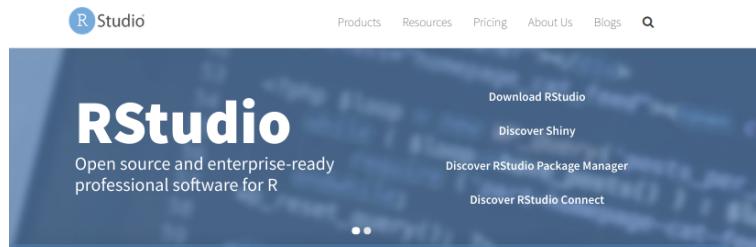


Figura 1.5: Sítio do *RStudio*.

This screenshot shows the "Choose Your Version" section of the RStudio website. It starts with a heading "Choose Your Version of RStudio" and a brief description of what RStudio is. To the right is an image of a computer monitor displaying the RStudio interface. Below this, there's a table comparing five versions of RStudio based on their features and pricing.

	RStudio Desktop Open Source License	RStudio Desktop Commercial License	RStudio Server Open Source License	RStudio Server Pro Commercial License	RStudio Server Pro + RStudio Connect Commercial License
FREE		\$995 per year	FREE	\$9,995 per year	\$29,995 per year
DOWNLOAD Learn More	BUY Learn More	DOWNLOAD Learn More	DOWNLOAD Learn More	TALK Learn More	
Integrated Tools for R	●	●	●	●	
Priority Support	●	●	●	●	
Access via Web Browser		●	●	●	
Enterprise Security			●	●	
Project Sharing			●	●	
Manage Multiple R Sessions & Versions			●	●	
Admin Dashboard			●	●	
Load Balancing			●	●	
One-Click Publishing				●	
Self-Managed Content				●	
Scheduled Reports				●	
License	AGPL	Commercial	AGPL	Commercial	Commercial
Pricing	FREE	\$995/yr	FREE	\$9,995/yr	\$29,995/yr
RStudio Desktop Open Source	DOWNLOAD NOW	RStudio Desktop Commercial	RStudio Server Open Source	RStudio Server Pro	RStudio Server Pro + RStudio Connect
	BUY NOW	DOWNLOAD NOW	DOWNLOAD NOW	CONTACT SALES	

Figura 1.6: Página do *RStudio* com as versões disponíveis para instalação.

[RStudio Desktop 1.2.1335 — Release Notes](#)

RStudio requires R 3.0.1+. If you don't already have R, download it [here](#).

Linux users may need to import RStudio's public code-signing key prior to installation, depending on the operating system's security policy.

Installers for Supported Platforms

Installers	Size	Date	MD5
RStudio 1.2.1335 - Windows 7+	126.9 MB	2019-04-08	d0e2470f1f8ef4cd35a669aa323a2136
RStudio 1.2.1335 - Mac OS X 10.12+ (64-bit)	121.1 MB	2019-04-08	6c570b0e2144583f7c48c284ce299eeef
RStudio 1.2.1335 - Ubuntu 14/Debian 8 (64-bit)	92.2 MB	2019-04-08	c1b07d0511469abfe582919b183eee83
RStudio 1.2.1335 - Ubuntu 16 (64-bit)	99.3 MB	2019-04-08	c142d69c210257fb10d18c045ffff13c7
RStudio 1.2.1335 - Ubuntu 18 (64-bit)	100.4 MB	2019-04-08	71a8d1990c0d97939804b46cfb0aea75
RStudio 1.2.1335 - Fedora 19+/RedHat 7+ (64-bit)	114.1 MB	2019-04-08	296b6ef88969a91297fab6545f256a7a
RStudio 1.2.1335 - Debian 9+ (64-bit)	100.6 MB	2019-04-08	1e32d4d6f6e216f086a81ca82ef65a91
RStudio 1.2.1335 - OpenSUSE 15+ (64-bit)	101.6 MB	2019-04-08	2795a63c7efd8e2aa2dae86ba09a81e5
RStudio 1.2.1335 - SLES/OpenSUSE 12+ (64-bit)	94.4 MB	2019-04-08	c65424b06ef6737279d982db9eefcae1

Zip/Tarballs

Zip/tar archives	Size	Date	MD5
RStudio 1.2.1335 - Windows 7+	186.6 MB	2019-04-08	f1e013ade0c241969400507cf258e0ad
RStudio 1.2.1335 - Ubuntu 14/Debian 8 (64-bit)	137.6 MB	2019-04-08	e3e1ea2dd113fd9cf40bc5035effdde
RStudio 1.2.1335 - Ubuntu 18 (64-bit)	147.8 MB	2019-04-08	5ee7dd7b501675f0a631c62d403ea1b6
RStudio 1.2.1335 - Debian 9+ (64-bit)	148.1 MB	2019-04-08	8090451cb7d520633eba80fd355ad4c1
RStudio 1.2.1335 - Fedora 19+/RedHat 7+ (64-bit)	147.2 MB	2019-04-08	34630cd7c66c3429879bd79982349380

Source Code

A tarball containing source code for RStudio v1.2.1335 can be downloaded from [here](#)

Figura 1.7: Instaladores disponíveis para o *RStudio*.

Ao baixar o instalador, basta executá-lo que o programa será instalado. Após a instalação, para executar o *RStudio*, basta selecioná-lo na lista de aplicações ou clicar em seu ícone na área de trabalho.

1.5 Instalação do pacote do *R Commander* a partir do *RStudio*

É possível instalar o pacote *R Commander*, e qualquer outro pacote do R, a partir do *RStudio*. Caso já tenha instalado o *R Commander* na seção 1.3, não é necessário executar os passos mostrados abaixo, mas aconselhamos a leitura para entender como instalar um pacote do R a partir do *RStudio*.

Para instalar o *R Commander*, ou qualquer outro pacote, a partir do *RStudio*, seguimos os passos abaixo:

- Executamos o *RStudio*. A tela de entrada do *RStudio* é mostrada na figura 1.8.

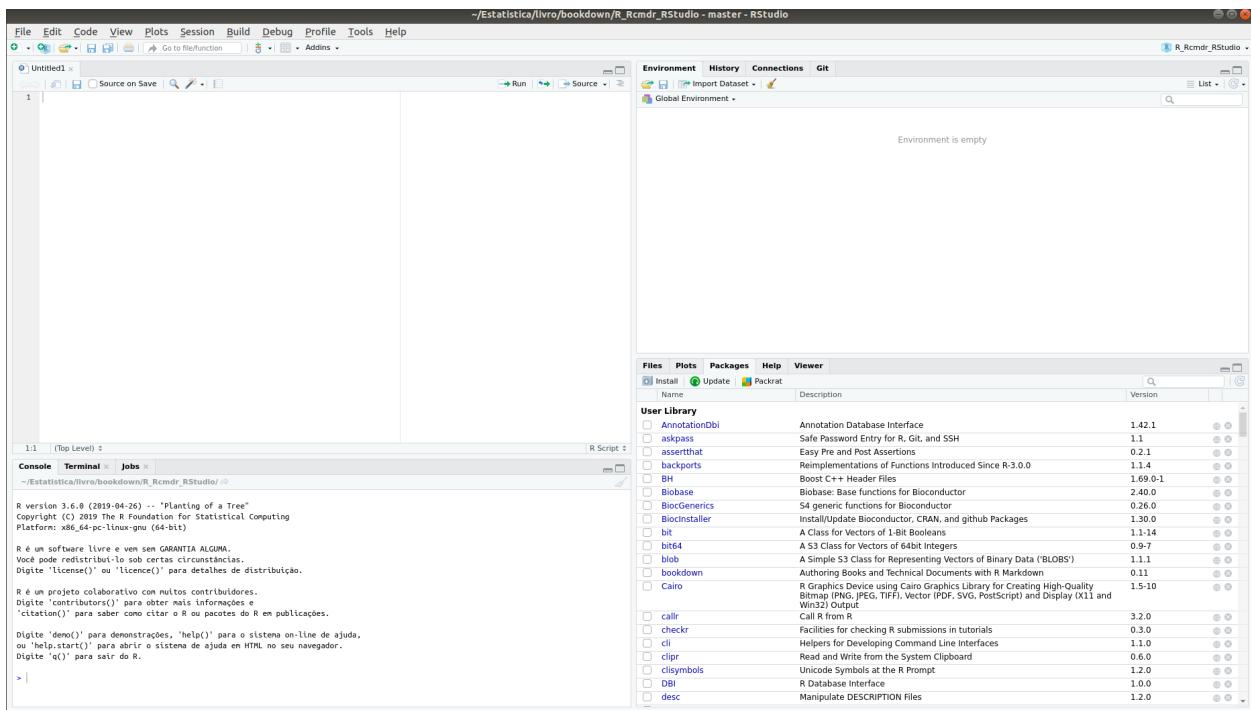


Figura 1.8: Tela de entrada do *RStudio*.

- Clicamos na aba *packages* e, em seguida, no botão *Install* (figura 1.9).

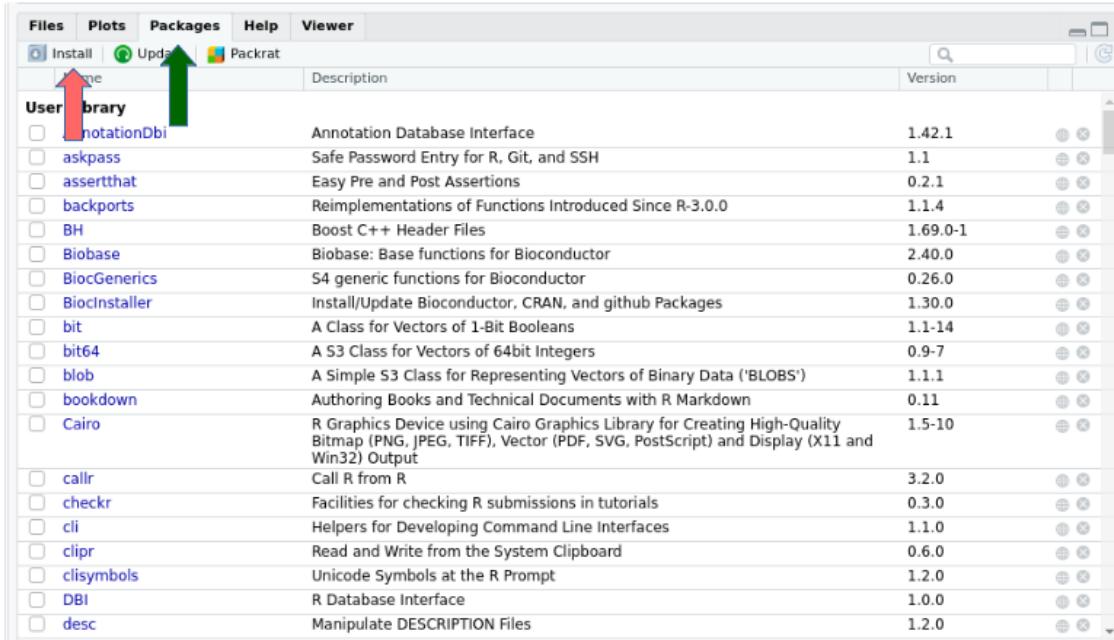


Figura 1.9: Para instalar um pacote, clicamos na aba *Packages* (seta verde) e, em seguida, no botão *Install* (seta vermelha).

- Na caixa de diálogo *Install packages*, começamos a digitar *Rcmdr* na caixa de texto *Packages*. Ao iniciarmos a digitação, uma lista suspensa mostra opções de pacotes. Selecionamos *Rcmdr* e clicamos no botão *Install* (figura 1.10). A instalação será inicializada e pode demorar um tempo. O progresso da instalação irá sendo exibido na janela da *Console* (canto inferior esquerdo do *RStudio*). Aguardamos até o sinal de *prompt* (>) aparecer na parte inferior da *console*.

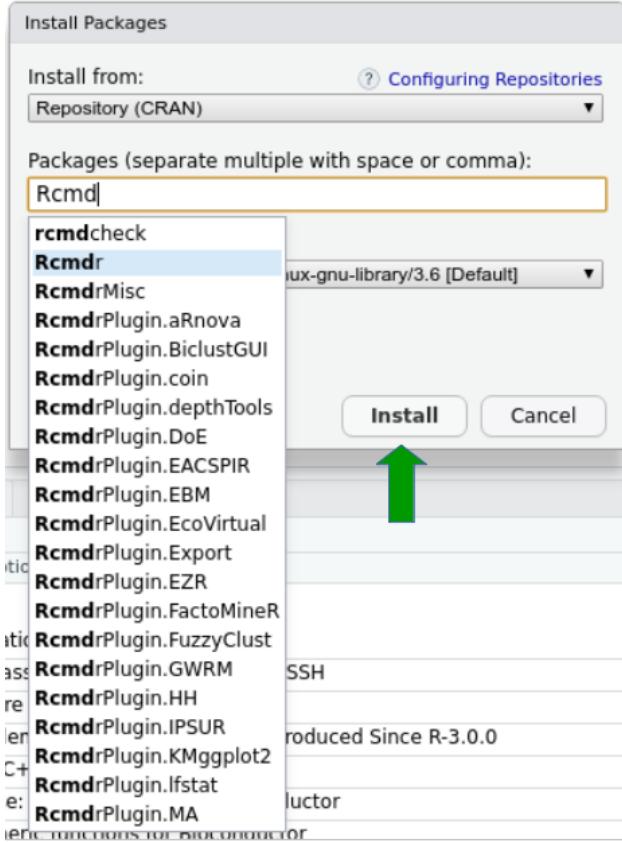


Figura 1.10: Para instalar o *R Commander*, digitamos *Rcmdr* na caixa de texto *Packages* e, em seguida, clicamos no botão *Install* (seta verde).

- Após a instalação, para carregarmos o *R Commander*, digitamos o comando `library(Rcmdr)` após o sinal de prompt na console do *RStudio* (figura 1.11) e pressionamos a tecla *Enter*.

```

Console Terminal Jobs
~/Estatistica/livro/bookdown/R_Rcmdr_RStudio/ ↵

R version 3.6.0 (2019-04-26) -- "Planting of a Tree"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R é um software livre e vem sem GARANTIA ALGUMA.
Você pode redistribuí-lo sob certas circunstâncias.
Digite 'license()' ou 'licence()' para detalhes de distribuição.

R é um projeto colaborativo com muitos contribuidores.
Digite 'contributors()' para obter mais informações e
'citation()' para saber como citar o R ou pacotes do R em publicações.

Digite 'demo()' para demonstrações, 'help()' para o sistema on-line de ajuda,
ou 'help.start()' para abrir o sistema de ajuda em HTML no seu navegador.
Digite 'q()' para sair do R.

> library(Rcmdr)

```

Figura 1.11: Comando para o carregamento do *R Commander* a partir do *RStudio*.

- Ao iniciarmos o carregamento do *R Commander*, pode acontecer de aparecer a tela

mostrada na figura 1.12, indicando que alguns pacotes estão faltando para carregar o *Rcmdr*. Nesse caso, selecionamos *Sim* e, na tela seguinte (figura 1.13), pressionamos OK. Após alguns instantes, os pacotes faltantes estarão instalados e o *R Commander* será inicializado.

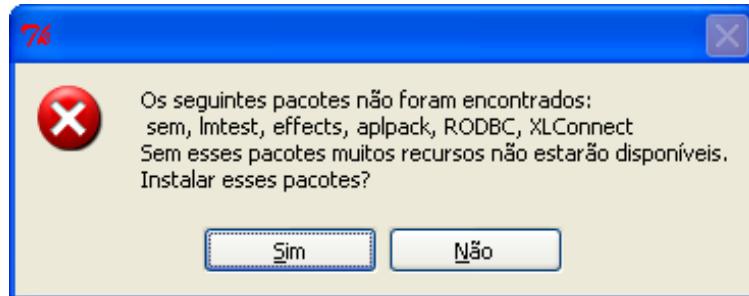


Figura 1.12: Mensagem que solicita a instalação de alguns pacotes por ocasião da primeira vez que o *R Commander* é executado.

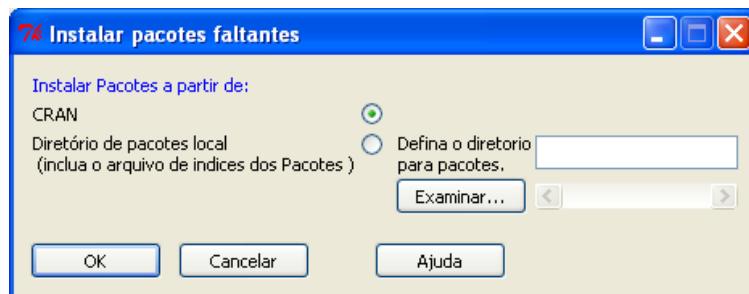


Figura 1.13: Tela de definição do local onde os pacotes dos quais o *R Commander* depende precisam ser obtidos. Utilizaremos a opção padrão.

- A figura 1.14 mostra a tela principal do *R Commander* quando o mesmo é carregado pelo *RStudio*.



Figura 1.14: Tela principal do *R Commander*.

Apesar de o R poder ser utilizado exclusivamente a partir de sua instalação, neste livro, sempre será utilizado o *R Commander*, ou o *RStudio*, eventualmente acompanhado do *R Commander*.

Este [vídeo](#) mostra como utilizar o *RStudio* em conjunto com o *R Commander*.

O próximo capítulo mostrará uma sessão do R para o usuário ter uma ideia dos tópicos que serão abordados neste livro.

Capítulo 2

Um breve *tour* do R, *RStudio* e *R Commander*

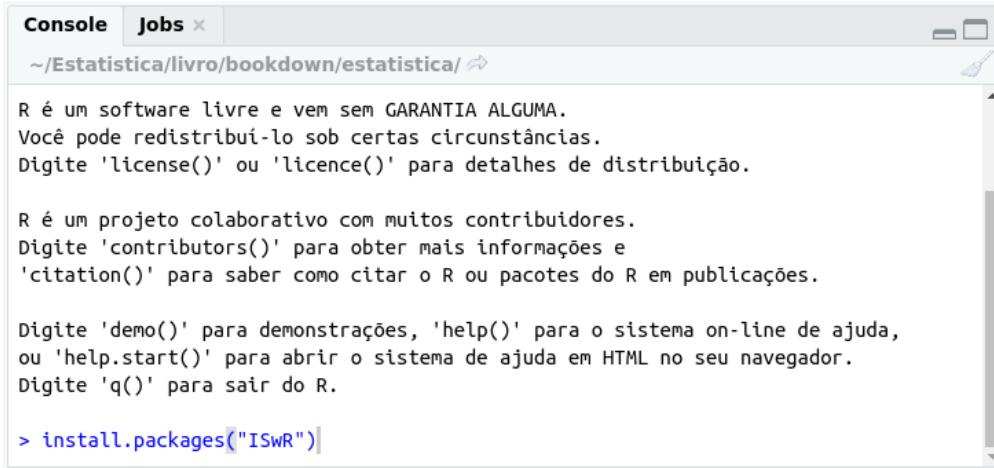
Vamos neste capítulo fazer um breve tour dos recursos do R. Não se preocupem em entender todo o processo realizado neste capítulo. Nos capítulos seguintes, serão vistos com mais detalhes como utilizar o R para realizar os procedimentos mostrados neste capítulo e outros recursos.

2.1 Carregando o *R Commander*

Execute o *RStudio*.

Vamos carregar um conjunto de dados já disponível em um pacote do R. Trata-se do conjunto de dados *juul2* do pacote *ISwR* (GPL-2 | GPL-3). O pacote *ISwR* precisa ser instalado. Para instalá-lo, pode-se digitar diretamente o comando abaixo na console do *RStudio* e pressionar a tecla *Enter* (figura 2.1).

```
install.packages("ISwR")
```



```

Console Jobs x
~/Estatistica/livro/bookdown/estatistica/ ↵

R é um software livre e vem sem GARANTIA ALGUMA.
Você pode redistribuí-lo sob certas circunstâncias.
Digite 'license()' ou 'licence()' para detalhes de distribuição.

R é um projeto colaborativo com muitos contribuidores.
Digite 'contributors()' para obter mais informações e
'citation()' para saber como citar o R ou pacotes do R em publicações.

Digite 'demo()' para demonstrações, 'help()' para o sistema on-line de ajuda,
ou 'help.start()' para abrir o sistema de ajuda em HTML no seu navegador.
Digite 'q()' para sair do R.

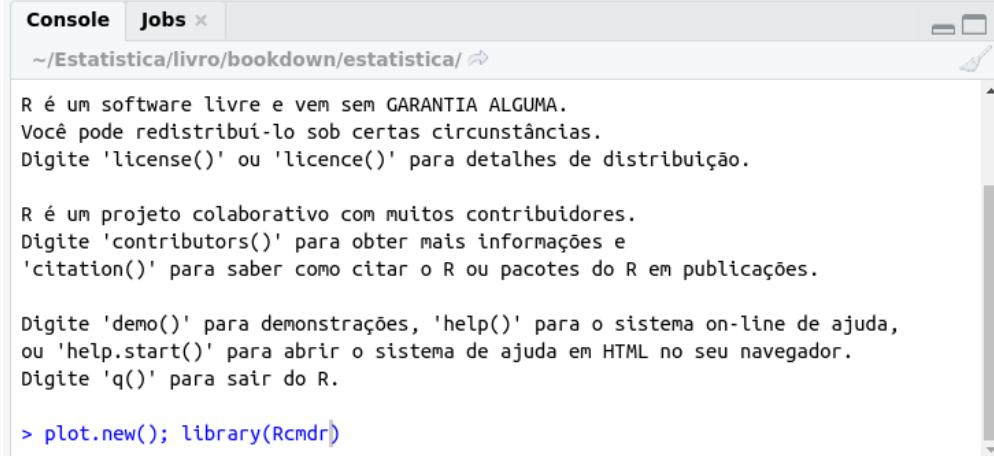
> install.packages("ISwR")

```

Figura 2.1: Console do *RStudio*, após a digitação da função `install.packages("ISwR")`.

Em seguida, digitamos os comandos abaixo na console do *RStudio* e pressionamos a tecla *Enter* (figura 2.2).

```
plot.new(); library(Rcmdr)
```



```

Console Jobs x
~/Estatistica/livro/bookdown/estatistica/ ↵

R é um software livre e vem sem GARANTIA ALGUMA.
Você pode redistribuí-lo sob certas circunstâncias.
Digite 'license()' ou 'licence()' para detalhes de distribuição.

R é um projeto colaborativo com muitos contribuidores.
Digite 'contributors()' para obter mais informações e
'citation()' para saber como citar o R ou pacotes do R em publicações.

Digite 'demo()' para demonstrações, 'help()' para o sistema on-line de ajuda,
ou 'help.start()' para abrir o sistema de ajuda em HTML no seu navegador.
Digite 'q()' para sair do R.

> plot.new(); library(Rcmdr)

```

Figura 2.2: Console do *RStudio*, após a digitação dos comandos para carregar o pacote *Rcmdr*.

2.2 Carregando um conjunto de dados

Vamos abrir o conjunto de dados *juul2* do pacote *ISwR*. No *R Commander*, digitamos o comando abaixo na área de *Script* e clicamos no botão *Submeter* (figura 2.3):

```
library(ISwR)
```

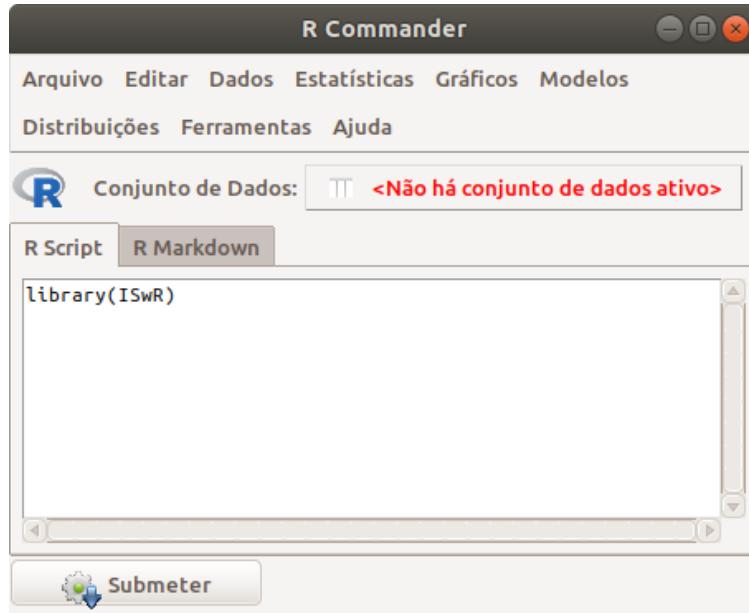


Figura 2.3: Área de *Script* do *R Commander*, com o comando para carregar o pacote *ISwR*.

Em seguida, selecionamos a opção abaixo no *R Commander* (figura 2.4):

Dados ⇒ Conjunto de dados em pacotes ⇒ Ler dados de pacotes 'attachados'

A partir de agora, toda opção a ser selecionada no menu será apresentada como uma sequência de itens a serem selecionados como acima.



Figura 2.4: Menu do *R Commander* com a opção para carregar arquivos de pacotes do R.

Na tela *Leia dados do pacote*, observem que alguns pacotes de dados aparecem na área à esquerda da figura 2.5: *carData*, *datasets*, *ISwR* e *sandwich*. Para vermos a lista dos conjuntos de dados em *ISwR*, damos um duplo clique nesse pacote e uma lista de conjuntos de dados será mostrada à direita. Rolamos essa lista e selecionamos o conjunto *juul2*. Para visualizar a estrutura desse conjunto de dados, clicamos no botão *Ajuda para o conjunto de dados*

selecionado (seta verde na figura). Uma descrição desse conjunto de dados será exibida na aba *Help* do *RStudio* (figura 2.6). Ao clicarmos no botão OK na figura 2.5, após termos selecionado *juul2*, esse conjunto de dados será carregado no *R Commander* (figura 2.7).

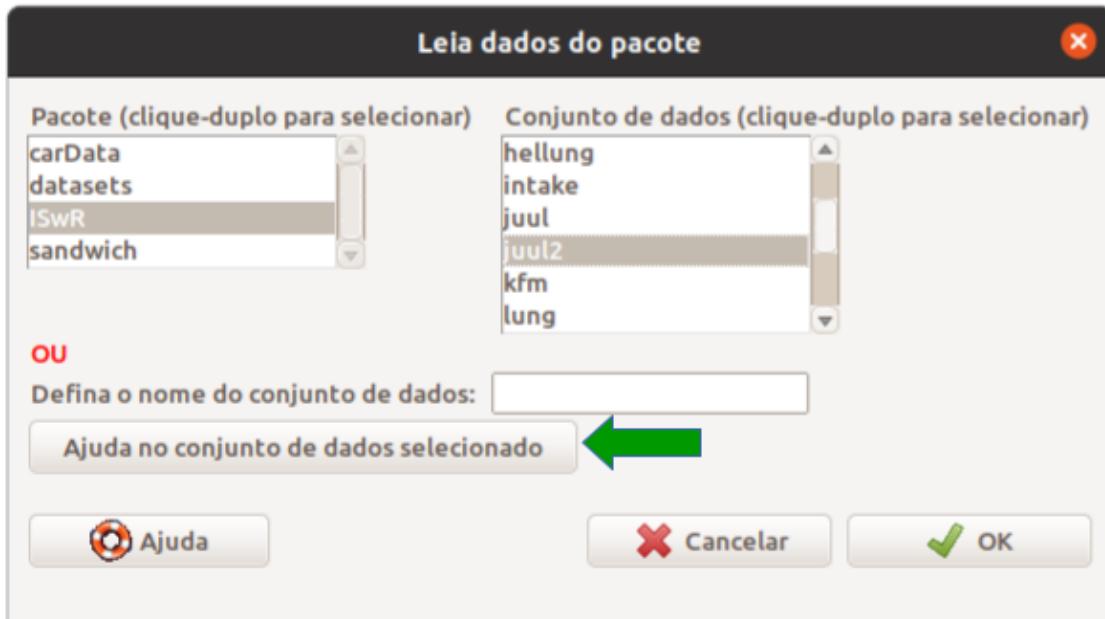


Figura 2.5: Visualizando a lista de conjuntos de dados do pacote *ISwR* e solicitando a ajuda para o conjunto *juul2* (seta verde).

The screenshot shows an R help browser window with the following interface elements:

- Top menu bar: Files, Plots, Packages, Help, Viewer.
- Toolbar icons: back, forward, home, search.
- Search bar: R: Juul's IGF data, extended version • Find in Topic.
- Text area:

```
juul2 {ISwR}
```

Juul's IGF data, extended version

Description

The `juul2` data frame has 1339 rows and 8 columns; extended version of `[juul]`.

Usage

```
juul2
```

Format

This data frame contains the following columns:

 - `age`
a numeric vector (years).
 - `height`
a numeric vector (cm).
 - `menarche`
a numeric vector. Has menarche occurred (code 1: no, 2: yes)?
 - `sex`
a numeric vector (1: boy, 2: girl).
 - `igf1`
a numeric vector, insulin-like growth factor (*microgram per liter*).
 - `tanner`
a numeric vector, codes 1-5: Stages of puberty ad modum Tanner.
 - `testvol`
a numeric vector, testicular volume (ml).
 - `weight`
a numeric vector, weight (kg).

Source

Original data.

Figura 2.6: Texto com a descrição do conjunto de dados `juul2` exibido no navegador de seu computador.

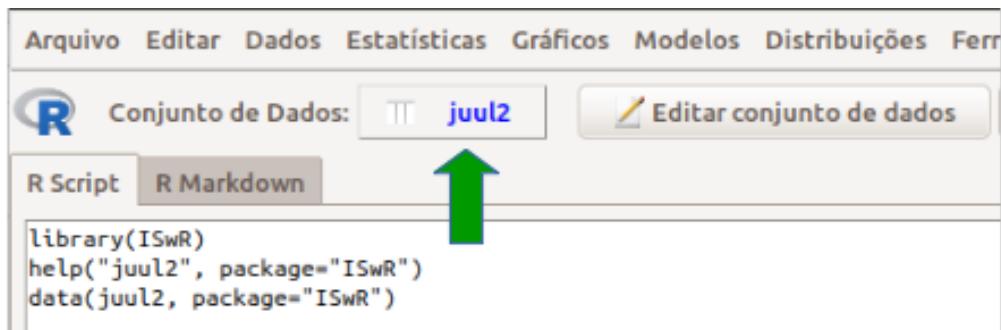


Figura 2.7: Tela do R commander após o carregamento do conjunto de dados *juul2*. Observem a função que foi executada – *data(juul2, package = "ISwR")* – e o nome do conjunto selecionado (seta verde).

Na console do *RStudio*, aparece a seguinte mensagem abaixo do comando, indicando o número de registros e de variáveis no conjunto de dados *juul2*:

```
RcmdrMsg: [2] NOTA: Os dados juul2 tem 1339 linhas e 8 colunas.
```

2.3 Visualizando o conteúdo do conjunto de dados

O conjunto de dados *juul2* possui 1339 registros, cada registro com valores de 8 variáveis . Ele contém uma amostra da distribuição da variável *insulin-like growth factor (igf1)*, com os dados coletados em exames físicos, sendo a maior parte dos dados de pessoas em idade escolar, mas também inclui outras faixas etárias. Vamos obter algumas medidas de tendência central e dispersão para as variáveis *idade* e *igf1*.

Para visualizarmos o conjunto de dados no *R Commander*, clicamos no botão *Ver conjunto de dados* na tela do *R Commander* (seta verde na figura 2.8).



Figura 2.8: Visualizando o conteúdo do conjunto de dados *juul2*.

2.4 Resumos numéricos

No item de menu *Estatística* do *R Commander*, vamos clicar em *Resumos* e, a seguir, em *Resumos numéricos*:

Estatística ⇒ Resumo... ⇒ Resumos numéricos...

Na tela *Resumos Numéricos*, selecionamos as variáveis na aba *Dados*. Para selecionarmos mais de uma variável, mantemos a tecla Ctrl pressionada enquanto clicar nas variáveis desejadas. Nesse exemplo, vamos selecionar as variáveis *age* e *igf1* (Figura 2.9). Em seguida, selecionamos a aba *Estatísticas* (seta verde).

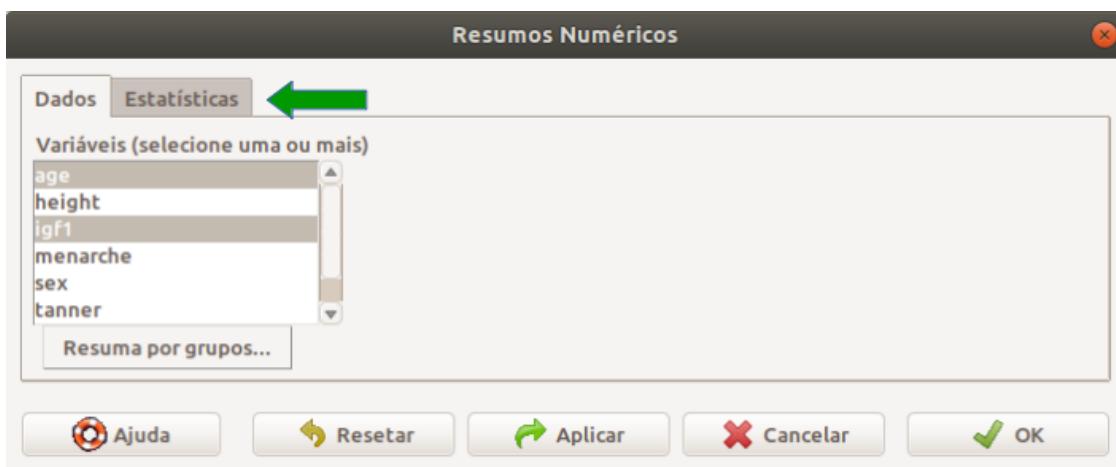


Figura 2.9: Seleção das variáveis para as quais resumos numéricos serão mostrados. A seta verde indica a aba onde podem ser selecionadas as medidas que serão apresentadas.

Na aba *Estatísticas* (figura 2.10), observem que as medidas média, desvio padrão, distância interquartil e quantis já estão marcadas. Se desejarmos outros percentis, basta digitá-los na caixa de texto com o rótulo *Quantis*, separados por vírgula. Ao clicarmos em *OK*, os resultados serão apresentados na console do *RStudio* (figura 2.11).



Figura 2.10: Tela para a seleção das medidas que serão apresentadas nos resumos numéricos.

```

Console Terminal × Jobs ×
~/Estatistica/livro/bookdown/estatistica/ ↵

Rcmdr> data(juul2, package="ISwR")
RcmdrMsg: [3] NOTA: Os dados juul2 tem 1339 linhas e 8 colunas.

Rcmdr> library(abind, pos=18)

Rcmdr> library(e1071, pos=19)

Rcmdr> numSummary(juul2[,c("age", "igf1"), drop=FALSE], statistics=c("mean", "sd",
Rcmdr+   "IQR", "quantiles"), quantiles=c(0,.25,.5,.75,1))
      mean      sd      IQR    0%    25%    50%    75% 100%     n   NA
age   15.09535 11.25288  7.8025  0.17  9.0525 12.56 16.855   83 1334     5
igf1 340.16798 171.03560 260.5000 25.00 202.2500 313.50 462.750   915 1018 321
> |

```

Figura 2.11: Resumos numéricos para as variáveis *age* e *igf1*.

2.5 Recodificação de variáveis

Vamos recodificar a variável *sex* de *juul2*, substituindo o valor 1 por “masculino” e o valor 2 por “feminino”.

A operação de recodificação de variáveis é acessada no *R Commander* da seguinte forma:

Dados ⇒ Modificação de variáveis no conjunto de dados ⇒ Recodificar variáveis

A figura 2.12 mostra a caixa de diálogo do *R Commander* para recodificar uma ou mais variáveis. Para especificarmos a recodificação da variável *sex*, selecionamos a variável *sex* e

escrevemos o nome da variável que será criada após a recodificação. Nesse exemplo, colocamos o nome *sexo_cat*. Caso usemos o mesmo nome da variável que será recodificada, os valores da variável *sex* seriam substituídos pelos valores recodificados.



Figura 2.12: Caixa de diálogo para especificar a recodificação de uma variável.

Na caixa de texto *Definições p/recodificação*, escrevemos em cada linha as recodificações. Por exemplo, a primeira linha na figura especifica que o valor 1 será substituído por masculino, a segunda linha especifica que o valor 2 será substituído por feminino. Se marcarmos a opção *Faça de cada nova variável um fator*, a nova variável será convertida para fator. clicamos em OK e a variável *sexo_cat* é criada a partir da recodificação da variável *sex* e é incorporada ao conjunto de dados *juul2* como fator. Observem os registros do conjunto de dados após a recodificação.

2.6 Convertendo uma variável numérica para categórica (fator)

Vamos converter a variável *tanner* para fator (categórica). Selecione a opção:

Dados ⇒ Modificação var. conj. dados ⇒ Converter var. numérica para fator

Na caixa de diálogo *Converter Variáveis Numéricas p/ Fator* (figura 2.13), selecione a variável que será convertida e escolha uma das opções: manter as categorias expressas como número, ou fornecer nomes às categorias. Vamos dar nomes às categorias neste exemplo. No campo

Novo nome da variável . . ., digite o nome da variável que será criada. Se nenhum nome for especificado nesse campo, os nomes das categorias substituirão os valores numéricos na própria variável que será convertida e não será criada uma nova variável.

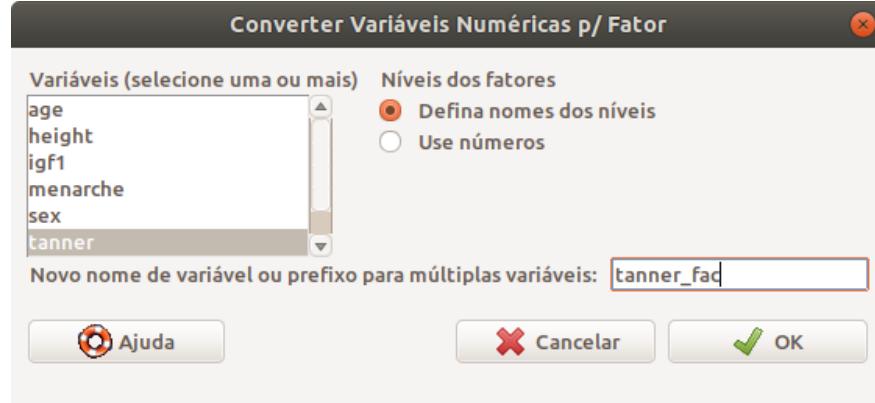


Figura 2.13: Passos para criar as categorias de uma variável: selecionamos a variável na lista da esquerda, escolhemos se as categorias serão dadas como texto e damos o nome da nova variável. Clicamos em OK.

Como selecionamos a opção de fornecer os nomes para as categorias, ao clicarmos em OK na figura 2.13, uma nova caixa de diálogo aparece para darmos os nomes das categorias para cada valor numérico (figura 2.14). Finalmente, ao clicarmos em OK, a nova variável, *tanner_cat*, será criada com as categorias apropriadas.

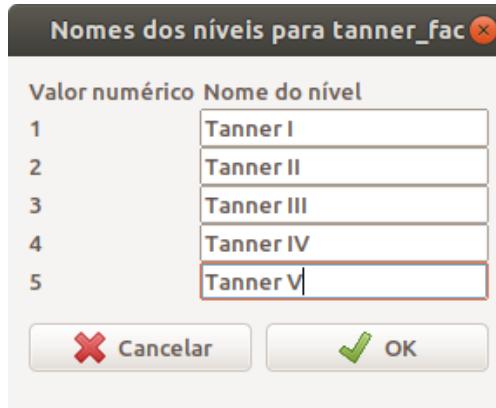


Figura 2.14: Especificação das categorias para a variável Tanner.

2.7 Diagrama de Barras

Vamos criar um diagrama de barras para as categorias de Tanner no conjunto *juul2*. Para criar um diagrama de barras no *R Commander*, selecione a opção:

Gráficos ⇒ Gráfico de barras

Na caixa de diálogo *Gráfico de Barra*, na aba *Dados*, é possível selecionar a variável categórica desejada. No nosso exemplo, vamos criar um diagrama de barras para a variável categórica *tanner_cat* (figura 2.15). Na aba *Opções* desta caixa de diálogo (figura 2.16), é possível especificar o tipo de diagrama de barras, a posição das legendas, as legendas do eixo x e y e o título do gráfico. Ao clicarmos em OK, o gráfico será exibido na aba *Plots* do RStudio (figura 2.17).

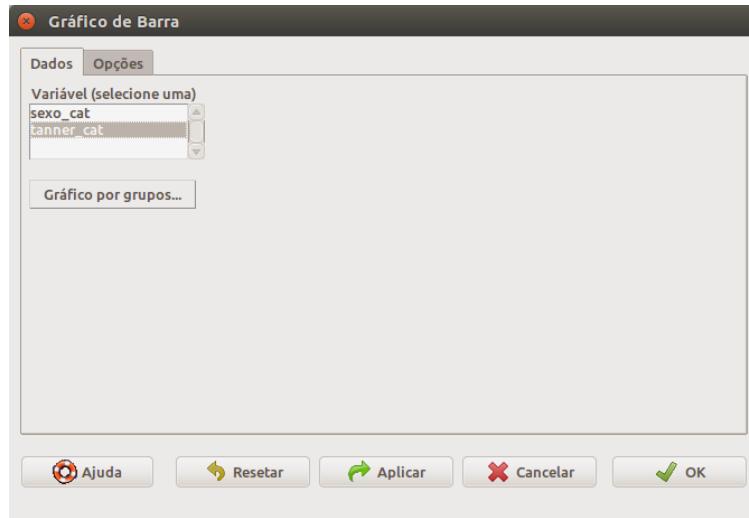


Figura 2.15: Caixa de diálogo para geração de um diagrama de barras: selecionando a variável.



Figura 2.16: Caixa de diálogo para geração de um diagrama de barras: especificando o título do gráfico e as legendas dos eixos x e y.

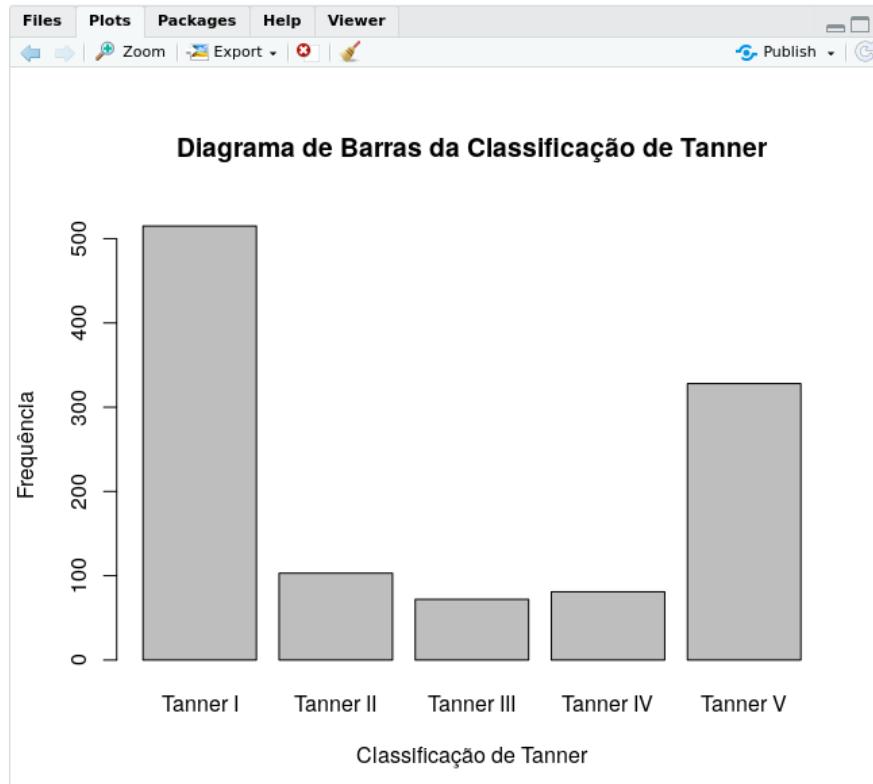


Figura 2.17: Diagrama de barras para a variável *tanner_cat*. São mostradas as frequências de cada categoria de tanner.

O gráfico da figura 2.17 mostra as frequências de cada uma das cinco categorias da classificação de Tanner no conjunto de dados *juul2*. A categoria I é a mais frequente, seguida da categoria V. As categorias II, III e IV apresentam frequências próximas umas das outras, mas com frequências bem menores do que as categorias I e V.

Caso desejemos visualizar o diagrama de barras da variável *sexo_cat* separadamente para cada categoria de Tanner, precisamos selecionar *sexo_cat* como uma variável de agrupamento. Para isso, clicamos na opção *Gráfico por grupos* na figura 2.15. Seremos então apresentados à caixa de diálogo da figura 2.18, onde selecionamos a variável de agrupamento (*sexo_cat*). Ao clicarmos em OK, voltamos à tela da figura 2.15. Clicando na aba *Opções*, mostrada novamente na figura 2.19, podemos escolher entre duas opções de como o diagrama de barras será construído: barras de cada categoria da classificação de Tanner lado a lado para cada valor da variável *sexo_cat*, ou empilhadas. Selecionando a primeira opção e clicando em OK, será plotado o gráfico da figura 2.20.

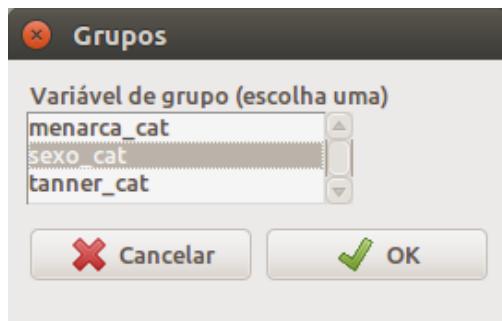


Figura 2.18: Selecionando uma variável de agrupamento para o diagrama de barras da variável *sexo_cat* para cada categoria da classificação de Tanner.

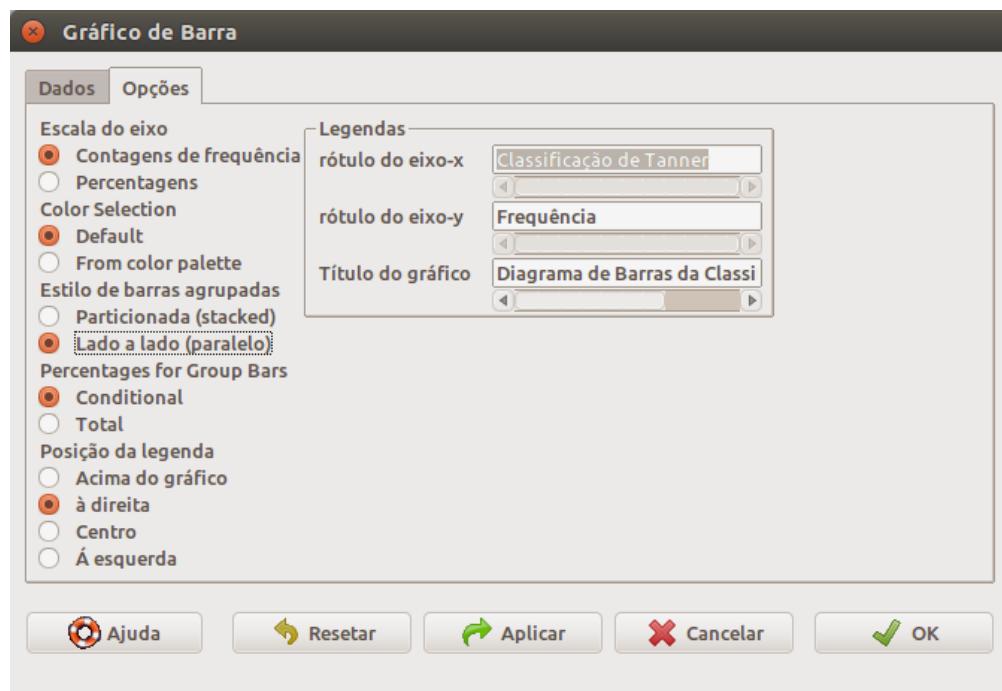


Figura 2.19: Selecionando a forma como as barras serão apresentadas: lado a lado ou empilhadas. Nesse exemplo, foi selecionada a opção lado a lado.

Diagrama de Barras da Classificação de Tanner

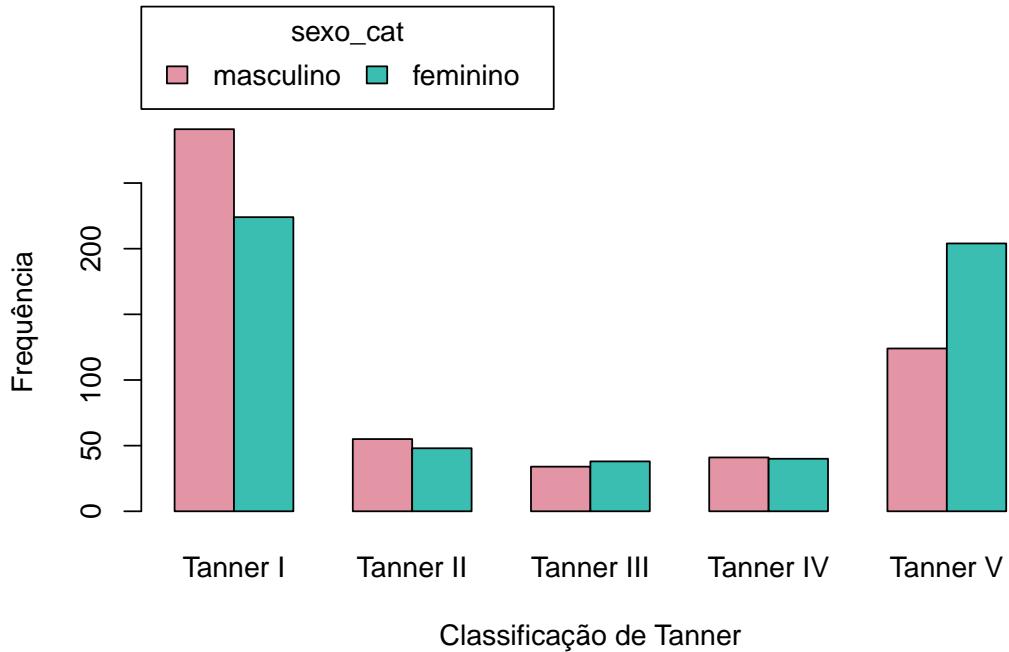


Figura 2.20: Diagrama de barras lado a lado das frequências das categorias da variável *sexo_cat* para cada categoria da variável *tanner_cat*.

2.8 Boxplot

Vamos criar um diagrama de *boxplot* da variável *igf1* (fator de crescimento semelhante à insulina tipo 1) para cada categoria da classificação de Tanner, selecionando a opção:

Gráficos ⇒ Boxplot

A figura 2.21 mostra a tela de configuração do *boxplot*. Na aba *Dados*, selecionamos a variável *igf1*.



Figura 2.21: Caixa de diálogo para a geração do *boxplot*. Nesse exemplo, estamos selecionando a variável *igf1*.

Para mostrar o *boxplot* de *igf1* para cada categoria da classificação de Tanner, clicamos no botão *Gráfico por grupos...* na caixa de diálogo do *boxplot* (figura 2.21) e selecionamos a variável *tanner_cat* para compor os grupos.

Na aba *Opções*, digitamos um título para o gráfico e marcamos a opção de não identificar os outliers (2.22). Ao clicarmos em OK, o resultado é mostrado na figura 2.23.



Figura 2.22: Aba *Opções* da caixa de diálogo para a geração do *boxplot*.

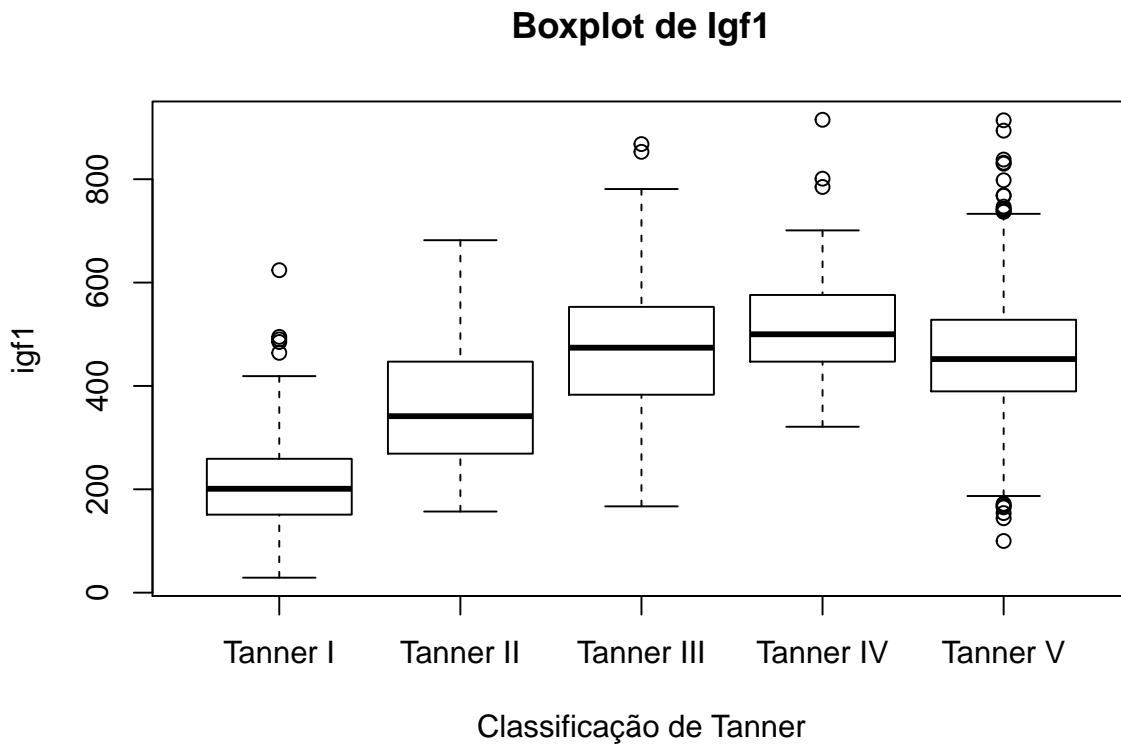


Figura 2.23: *Boxplots* para a variável *igf1* para cada categoria de Tanner.

2.9 Histograma

Para construir um histograma no *R Commander*, selecionamos a opção:

Gráficos ⇒ Histograma

Em seguida, selecionamos a variável desejada, *igf1* neste exemplo (figura 2.24). Na aba *Opções* (figura 2.25), vamos selecionar percentagens em *Escala do eixo* e digitar a legenda do eixo y. Ao clicarmos em OK, o gráfico resultante é mostrado na figura 2.26.

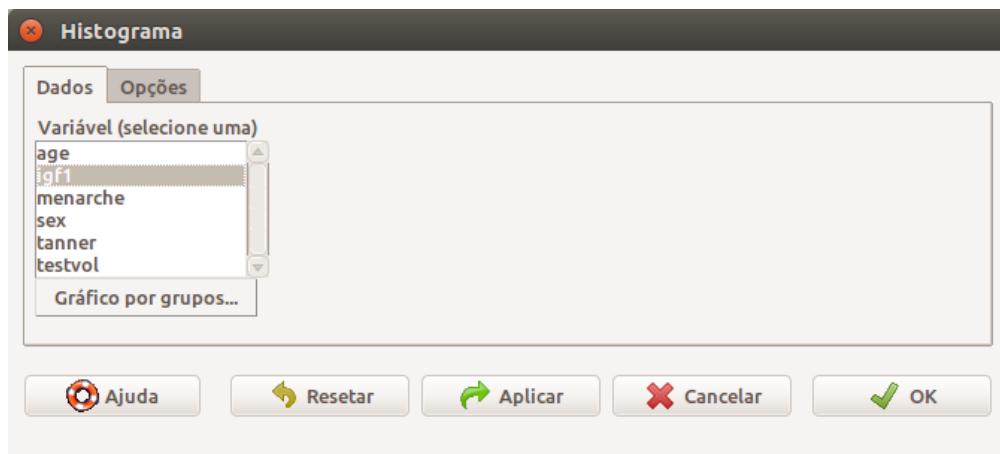


Figura 2.24: Caixa de diálogo para a criação de um histograma. Na aba *Dados*, selecionamos a variável numérica desejada.



Figura 2.25: Caixa de diálogo para a criação de um histograma. Na aba *Opções*, podemos especificar o número de faixas de valores (classes), a escala do eixo e as legendas.

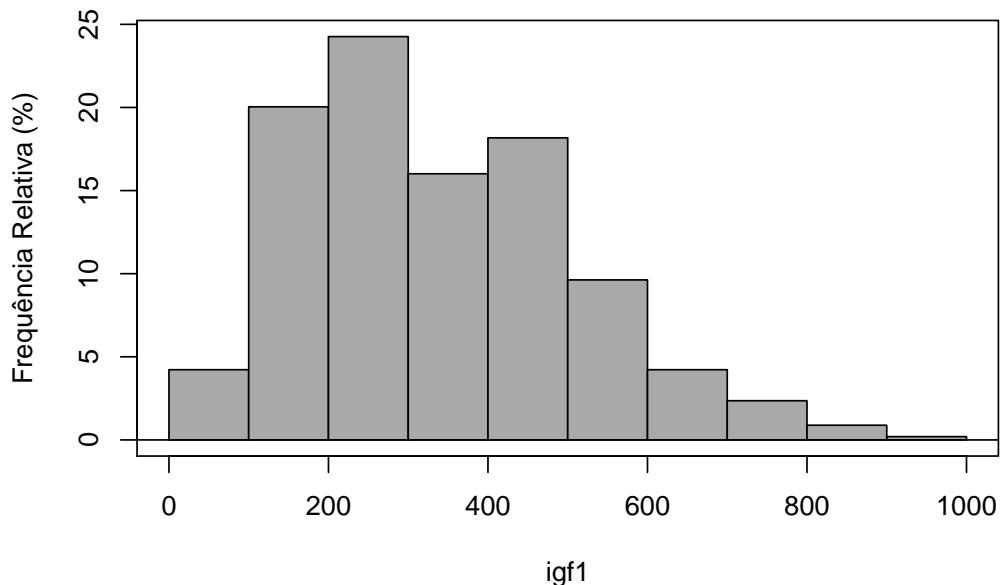


Figura 2.26: Histograma de frequência relativa da variável igf1.

2.10 Cálculo de nova variável

Vamos supor que desejamos calcular o índice de massa corporal (IMC) para as observações do conjunto de dados *juul2*. Para isto, utilizamos a seguinte opção no *R Commander*:

Dados ⇒ Modificação variáveis no conj. de dados... ⇒ Computar nova variável...

A figura 2.27 mostra a caixa de diálogo para computar o IMC a partir das variáveis *weight* e *height*. A variável *height* foi dividida por 100, porque ela está em cm.



Figura 2.27: Caixa de diálogo para especificar o cálculo de uma nova variável.

2.11 Diagrama de dispersão

Após o cálculo do IMC, vamos gerar um diagrama de dispersão do IMC x Idade separadamente para os homens e mulheres. Vamos selecionar a opção:

Gráficos ⇒ Diagrama de dispersão...

Na caixa de diálogo do diagrama de dispersão, selecionamos as variáveis *age* para o eixo X e a variável *imc* para o eixo Y (figura 2.28). Em seguida, clicamos no botão *Gráfico por grupos...* selecionamos a variável *sexo_cat* (figura 2.29). Clicamos em OK e, em seguida, na aba *Opções* (figura 2.30). Digitamos as legendas do eixo X e Y, selecionamos a posição das legendas da variável *sex* e marcamos a opção *Linha de quadrados mínimos*. Ao clicarmos em OK, o gráfico resultante é mostrado no *RStudio* (figura 2.31).

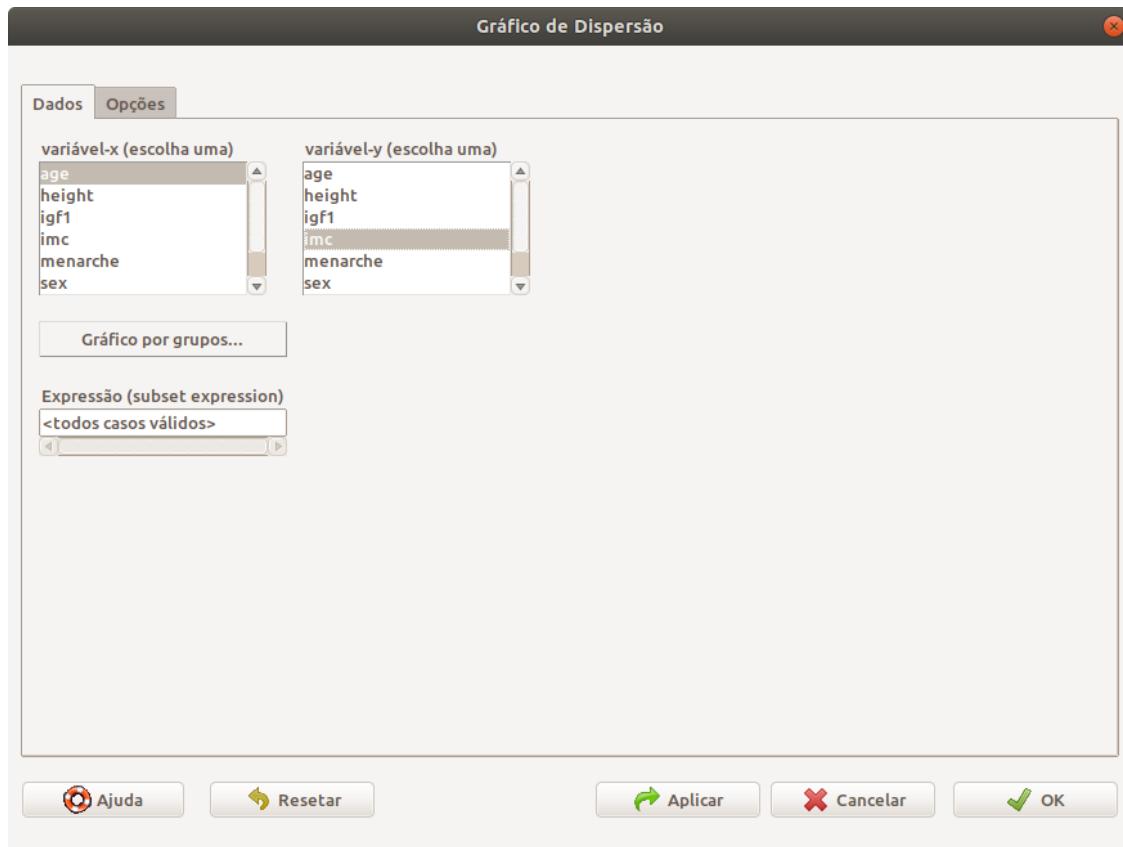


Figura 2.28: Caixa de diálogo para a geração de um diagrama de dispersão. Seleção das variáveis dos eixos X e Y.



Figura 2.29: Seleção da variável de agrupamento para gerar o diagrama de dispersão.



Figura 2.30: Opções para gerar o gráfico de dispersão.

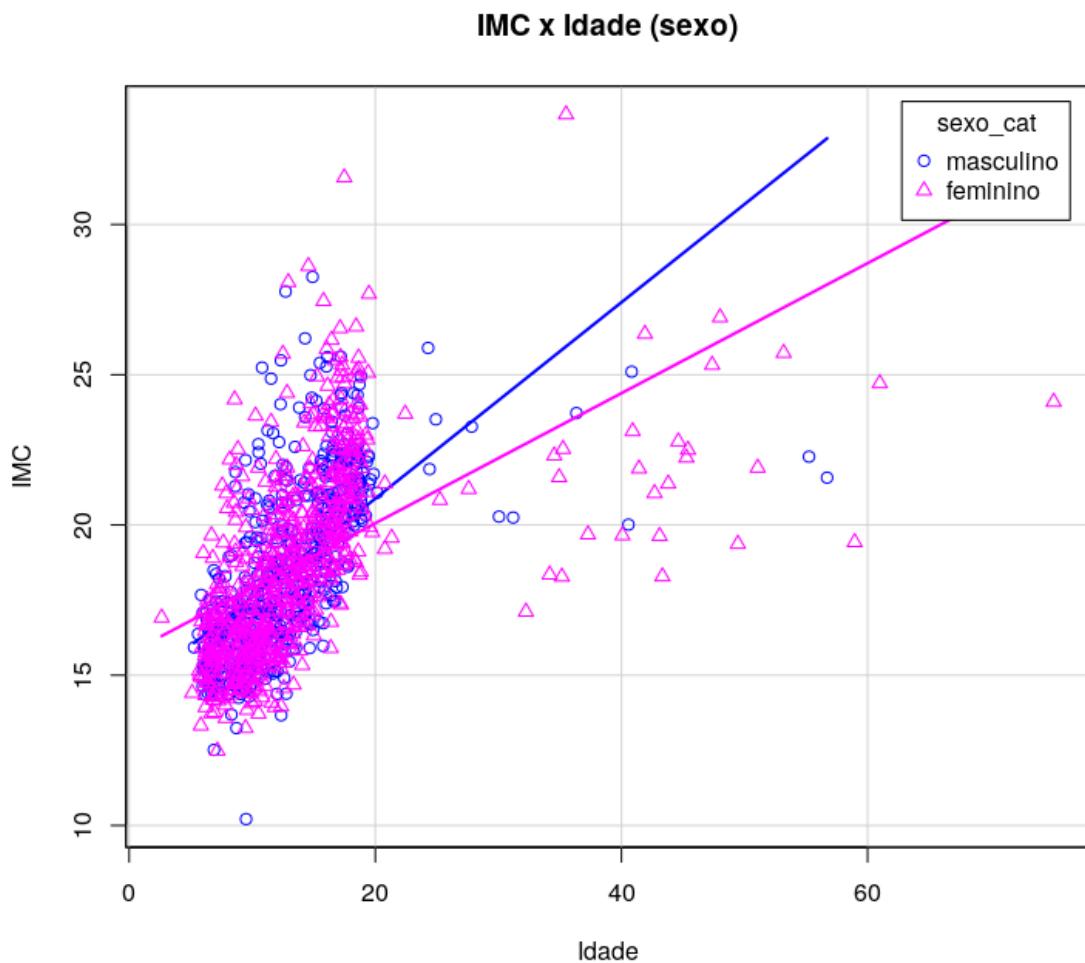


Figura 2.31: Gráfico de dispersão IMC x Idade para cada categoria de sexo.

2.12 Geração de relatório

O R Markdown é uma linguagem que permite que um relatório possa ser gerado a partir dos comandos que vão sendo executados no R. No *R Commander*, ele pode ser visualizado na aba R Markdown (seta verde na figura 2.32).



Figura 2.32: Acessando o R Markdown no R Commander.

Esse relatório pode ser personalizado pelo usuário. Por exemplo, no texto da figura 2.33, alteramos o título e o autor (seta verde na figura), depois selecionamos os comandos `help...` (figura 2.34) e os apagamos (figura 2.35). Ao clicarmos no botão *Gerar relatório*, o relatório será apresentado no navegador padrão de seu computador (figura 2.36)



```

R Commander
Arquivo Editar Dados Estatísticas Gráficos Modelos Distribuições Ferramentas Ajuda
Conjunto de Dados: juul2 Editar conjunto de dados Ver conjunto de dados Modelo: <sem modelo ativo>
R Script R Markdown
<!-- R Commander Markdown Template -->
Exemplo de Relatório
=====
### Autor do Relatório
### `r as.character(Sys.Date())`
```{r echo=FALSE}
include this code chunk as-is to set options
knitr::opts_chunk$set(comment=NA, prompt=TRUE, out.width=750, fig.height=8, fig.width=8)
library(Rcmdr)
library(car)
```
Gerar relatório

```

Figura 2.33: Personalizando o título e o autor do relatório no *R Markdown*.



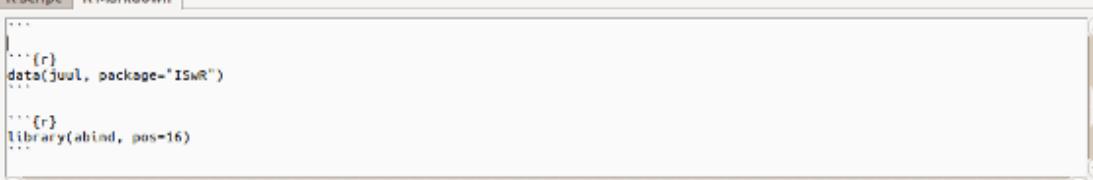
```

R Commander
Arquivo Editar Dados Estatísticas Gráficos Modelos Distribuições Ferramentas Ajuda
Conjunto de Dados: juul Editar conjunto de dados Ver conjunto de dados Modelo: <sem modelo ativo>
R Script R Markdown
...
```{r}
help(juul, package="ISwR")
```

```{r}
data(juul, package="ISwR")
```

```

Figura 2.34: Selecionando partes do relatório para edição.



```

R Commander
Arquivo Editar Dados Estatísticas Gráficos Modelos Distribuições Ferramentas Ajuda
Conjunto de Dados: juul Editar conjunto de dados Ver conjunto de dados Modelo: <sem modelo ativo>
R Script R Markdown
...
```{r}
data(juul, package="ISwR")
```

```{r}
library(abind, pos=16)
```

```

Figura 2.35: Remoção da área selecionada na figura 2.34.

Exemplo de Relatório

Autor do Relatório

2019-07-15

```
> library(ISwR)

> data(juul2, package="ISwR")

> numSummary(juul2[,c("age", "igf1"), drop=FALSE], statistics=c("mean", "sd",
+ "IQR", "quantiles"), quantiles=c(0,.25,.5,.75,1))
```

| | mean | sd | IQR | 0% | 25% | 50% | 75% | 100% | n |
|------|-----------|-----------|----------|-------|----------|--------|---------|------|------|
| age | 15.09535 | 11.25288 | 7.8025 | 0.17 | 9.0525 | 12.56 | 16.855 | 83 | 1334 |
| igf1 | 340.16798 | 171.03560 | 260.5000 | 25.00 | 202.2500 | 313.50 | 462.750 | 915 | 1018 |
| | NA | | | | | | | | |
| age | 5 | | | | | | | | |
| igf1 | 321 | | | | | | | | |

```
> mean(juul2$age)
```

```
[1] NA
```

```
> mean(juul2$age, na.rm = TRUE)
```

```
[1] 15.09535
```

Figura 2.36: Relatório gerado pelo *R Markdown* em html para os comandos utilizados nesta seção.

2.13 Salvando o conjunto de dados

Para salvar o conjunto de dados ativo em um arquivo que pode ser lido diretamente pelo R, selecionamos a opção

Dados ⇒ Conjunto de dados ativo ⇒ Salvar conjunto dados ativo

Na caixa de diálogo *Salvar Como* (figura 2.37), navegamos para a pasta onde desejamos salvar o arquivo e especificamos um nome para o arquivo, de preferência com a extensão *RData*. Clicamos em *Salvar*. O arquivo será gravado na pasta selecionada.

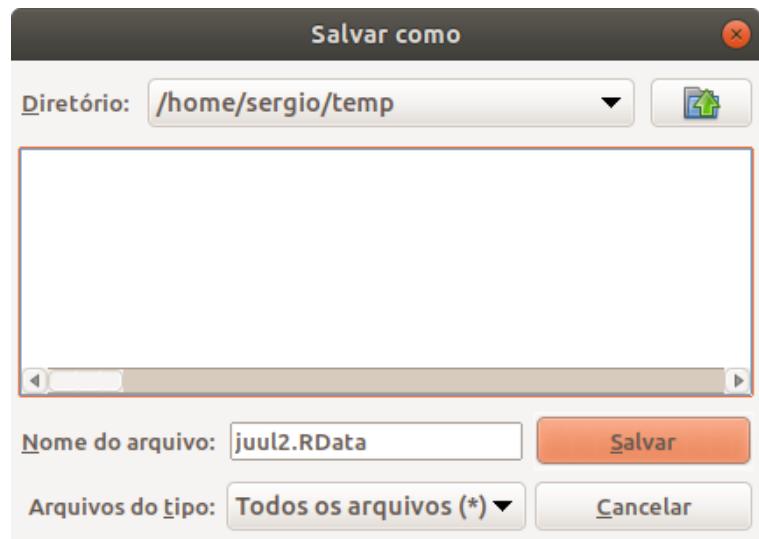


Figura 2.37: Especificação do nome do arquivo a ser gravado.

Capítulo 3

Estruturas básicas no R

Na maior parte das vezes que estivermos usando o R, estaremos trabalhando com arquivos de dados como o mostrado na figura 3.1.

| | age | height | menarche | sex | igf1 | tanner | testvol | weight |
|----|------|--------|----------|-----|------|--------|---------|--------|
| 1 | NA | NA | NA | NA | 90 | NA | NA | NA |
| 2 | NA | NA | NA | NA | 88 | NA | NA | NA |
| 3 | NA | NA | NA | NA | 164 | NA | NA | NA |
| 4 | NA | NA | NA | NA | 166 | NA | NA | NA |
| 5 | NA | NA | NA | NA | 131 | NA | NA | NA |
| 6 | 0.17 | NA | NA | 1 | 101 | 1 | NA | NA |
| 7 | 0.17 | NA | NA | 1 | 97 | 1 | NA | NA |
| 8 | 0.17 | NA | NA | 1 | 106 | 1 | NA | NA |
| 9 | 0.17 | NA | NA | 1 | 111 | 1 | NA | NA |
| 10 | 0.17 | NA | NA | 1 | 79 | 1 | NA | NA |
| 11 | 0.17 | NA | NA | 1 | 43 | 1 | NA | NA |
| 12 | 0.17 | NA | NA | 1 | 64 | 1 | NA | NA |
| 13 | 0.25 | NA | NA | 1 | 90 | 1 | NA | NA |
| 14 | 0.25 | NA | NA | 1 | 141 | 1 | NA | NA |
| 15 | 0.42 | NA | NA | 1 | 42 | 1 | NA | NA |
| 16 | 0.50 | NA | NA | 1 | 43 | 1 | NA | NA |
| 17 | 0.67 | NA | NA | 1 | 132 | 1 | NA | NA |
| 18 | 0.75 | NA | NA | 1 | 43 | 1 | NA | NA |
| 19 | 0.75 | NA | NA | 1 | 36 | 1 | NA | NA |
| 20 | 1.00 | NA | NA | 1 | 86 | 1 | NA | NA |
| 21 | 1.16 | NA | NA | 1 | 44 | 1 | NA | NA |
| 22 | 1.50 | NA | NA | 1 | 68 | 1 | NA | NA |
| 23 | 1.50 | NA | NA | 1 | 89 | 1 | NA | NA |
| 24 | 1.58 | NA | NA | 1 | 101 | 1 | NA | NA |
| 25 | 1.67 | NA | NA | 1 | 115 | 1 | NA | NA |
| 26 | 1.67 | NA | NA | 1 | 53 | 1 | NA | NA |
| 27 | 1.75 | NA | NA | 1 | 94 | 1 | NA | NA |
| 28 | 1.83 | NA | NA | 1 | 95 | 1 | NA | NA |
| 29 | 1.92 | NA | NA | 1 | 76 | 1 | NA | NA |
| 30 | 2.00 | NA | NA | 1 | 79 | 1 | NA | NA |

Figura 3.1: Parte do conteúdo do conjunto de dados *juul2* do pacote [ISwR](#) (GPL-2 | GPL-3).

Também frequentemente podemos realizar nossas análises e manipulações em arquivos de

dados, usando os menus do *R Commander*.

Porém eventualmente teremos que manipular as estruturas de dados que suportam esses arquivos ou utilizar recursos de comandos que não encontram suporte em uma interface gráfica.

Neste capítulo, serão mostrados as classes básicas do R, as operações e as comparações lógicas que podem ser realizadas com elas. O *RStudio* será utilizado neste capítulo.

Os conteúdos das seções 3.1, 3.2, 3.3 e 3.4 deste capítulo podem ser visualizados neste [vídeo](#).

3.1 Console do *RStudio*

Ao executarmos o *RStudio*, a console do *RStudio*, primeira aba da janela inferior à esquerda (figura 3.2), permite ao usuário digitar, executar os comandos e visualizar os resultados. Na console, é exibida a versão do R utilizada e algumas informações sobre o R e, na parte inferior, o sinal “>”. Esse símbolo é chamado *prompt* de comando e significa que o R está apto a receber um comando nesta linha. Por essa razão, essa linha é chamada **linha de comando**.

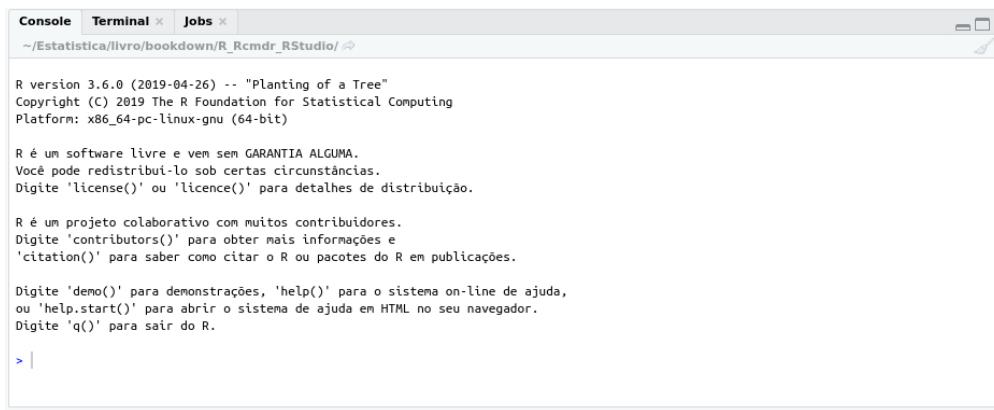


Figura 3.2: Console do *RStudio* com o prompt (>) e o cursor aguardando a digitação de um comando.

Para executarmos um comando, basta digitá-lo e, em seguida, apertarmos a tecla *<Enter>*. O resultado do comando é o que chamamos de *output* ou saída. Neste texto, os comandos serão mostrados sem o prompt e numa área sombreada e o resultado da execução do comando é mostrado a seguir, precedido de `##` quando houver resultados a serem exibidos.

Exemplo 1: para realizarmos a operação $4+3-5$, basta digitá-la na linha de comando como abaixo e pressionarmos *<Enter>*. Essa expressão apresenta os operadores de soma (+) e subtração (-).

```
4+3-5
```

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

O número 1 entre colchetes ([1]) precedendo o resultado apenas indica que o elemento a seguir é o primeiro elemento do resultado. Nesse exemplo, só há um elemento no resultado (2).

Mais de um comando podem ser digitados na mesma linha, separados por “;”.

Exemplo 2: Execute no R os comandos abaixo.

```
2*5+1; 3^2+28/7
```

```
## [1] 11
```

```
## [1] 13
```

Na primeira expressão, temos a multiplicação (*) e a soma, com a prioridade dada para a multiplicação. Na expressão seguinte, são realizadas a potenciação (^), que tem prioridade em relação às demais, a divisão (/) e a soma, nesta ordem. A ordem de prioridade das operações são portanto:

potenciação, multiplicação/divisão, adição/subtração

Observação: Para expressões nas quais desejarmos mudar as prioridades, devemos usar parênteses, e nunca colchetes ou chaves, como delimitadores (a rigor, as chaves podem ser utilizadas, mas não é recomendável). Por exemplo, se desejarmos somar 5 com 1 e depois multiplicar o resultado por dois, devemos fazer $2*(5+1)$, e não $2 * [5 + 1]$.

Exemplo 3: A expressão abaixo gera a sequência de números inteiros de 10 a 30. Agora o resultado tem mais de 1 elemento. O número entre colchetes indica a posição do elemento seguinte na sequência de números.

```
10:30
```

```
## [1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

Notas:

- Na atual linha de comando, é possível acessar comandos já executados, utilizando as setas ↑ e ↓ do teclado;
- Para incluir um comentário no R (ou seja, uma sequência de caracteres que deve ser ignorada pelo R, mas que serve como informação para o usuário), basta digitar o símbolo # e, em seguida, o comentário. Tudo que aparece depois do sinal # é ignorado pelo R.

```
2*5+1 # exemplo de uma expressão numérica no R
```

```
## [1] 11
```

3.2 Objetos no R

Tudo no R é um objeto. Para atribuirmos resultados de operações a algum objeto que pode posteriormente ser manipulado, usamos a expressão de atribuição “<-”:

```
objeto <- expressão
```

Os nomes dos objetos devem começar com uma letra (pode inclusive ser apenas uma letra), e podem conter letras, números, pontos, sublinhado (não são permitidos vírgula, ponto e vírgula, dois pontos e espaço). Veja o exemplo a seguir:

```
y <- 4 # lê-se "y recebe 4"
```

Ao executarmos o comando acima, nenhum resultado é mostrado na tela, mas o objeto y foi criado na memória do computador.

Para exibirmos o conteúdo de y, há duas formas. A primeira delas é digitarmos o nome do objeto na linha de comando e pressionarmos *<Enter>*. Isso é chamado de autoimpressão.

```
y
```

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

Outra forma é usar a função “print”, como mostrado abaixo:

```
print(y)
```

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

Observações:

- 1) no modo interativo, usualmente não usamos a função *print*. É muito mais fácil usar a autoimpressão. Porém, ao escrever scripts, funções, ou programas mais longos, há a necessidade de explicitamente imprimir objetos, porque a autoimpressão não funciona nesses cenários;
- 2) atualmente, o R também aceita o sinal de igual no lugar do operador de atribuição “<-”, produzindo o mesmo resultado. Neste texto, usaremos indistintamente tanto o sinal de “=” quanto “<-”;
- 3) é recomendável atribuir nomes que tenham um significado lógico. Isso facilita lidar com um grande número de objetos;
- 4) letras maiúsculas e minúsculas são consideradas diferentes para o R.

O R possui cinco classes básicas ou “atômicas” de objetos:

- *character*;
- *numeric* (números reais);
- *integer* (números inteiros);
- *complex* (números complexos);
- *logical* (Falso/Verdadeiro)

A função *class()*, aplicada a um objeto, retorna a sua classe. Vejamos alguns exemplos:

```
x <- "ola"  
class(x)
```

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

```
x <- 'ola'  
class(x)
```

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

```
y = 2  
class(y)
```

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

```
y = 2L  
class(y)
```

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

```
z.1 = TRUE  
class(z.1)
```

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

```
z.2 = FALSE  
class(z.2)
```

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

```
c1 = 1+0i  
class(c1)
```

```
## [1] "complex"
```

Observações:

- 1) expressões do tipo *character* devem aparecer entre aspas duplas ou simples;
- 2) números no R são geralmente tratados como objetos numéricos (números reais de dupla precisão). Mesmo números inteiros são tratados como numéricos. Para fazer um número inteiro ser tratado como objeto inteiro, deve-se utilizar a letra *L* após o número;
- 3) os valores lógicos (ou booleanos) são TRUE ou FALSE. T ou F também são aceitos;
- 4) Acima foram criados diversos objetos no R (x, y, z.1, z.2 e c1). A aba Environment do *RStudio* (seta verde na figura 3.3) mostra os objetos criados até então na sessão corrente.

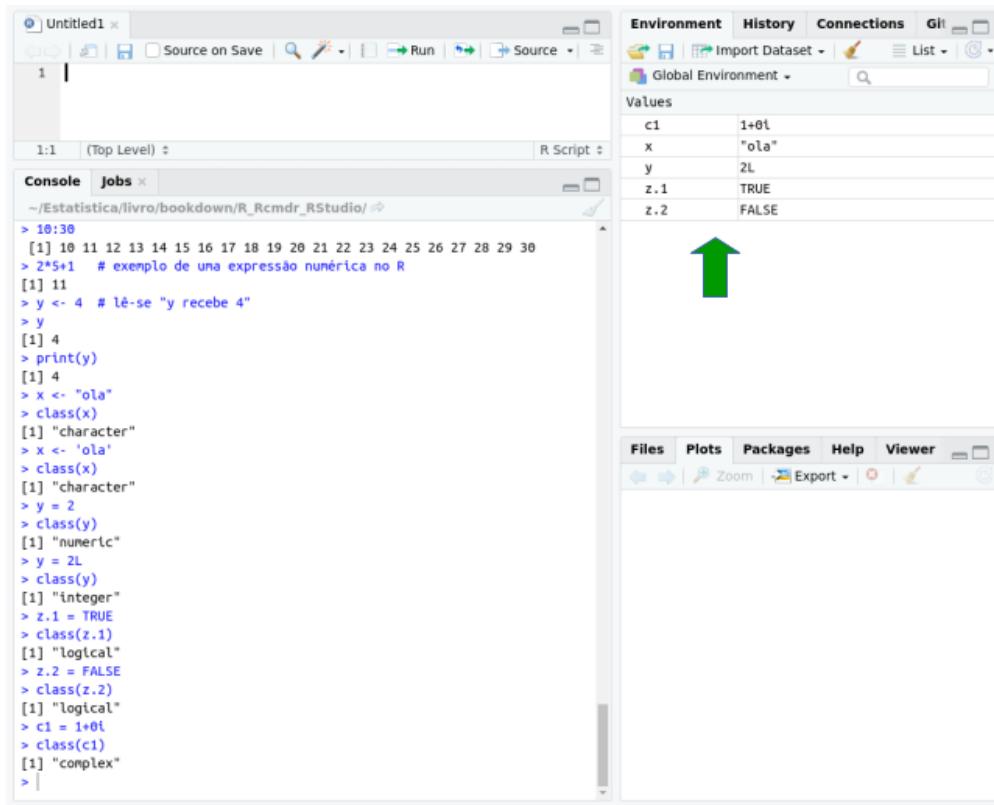


Figura 3.3: Ambiente de trabalho do R, mostrando os objetos criados na sessão corrente.

3.3 Valores especiais

O R possui alguns valores que são especiais: *NA*, *Inf*, *-Inf*, e *Nan*.

Inf significa infinito e nos permite representar resultados de operações como $1/0$ e pode ser usado em cálculos ordinários. Por exemplo, $1/\text{Inf}$ é zero. Analogamente $-\text{Inf}$ representa menos infinito. Vejam as expressões abaixo:

$1/0$

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

$-1/0$

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

$2/\text{Inf}$

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

```
-2/-Inf
```

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

O valor *NaN* representa um valor indefinido (*not a number*), por exemplo $0/0$. *NA* é usado para representar valores ausentes (*missing* em inglês). *NaN* também pode ser usado com essa finalidade. Voltaremos a essa questão na seção 3.12.

3.4 Funções

Como dito na introdução, o R é uma linguagem altamente extensível. Novas funcionalidades são adicionadas ao ambiente por meio da implementação de *funções* agrupadas em módulos ou *pacotes*. Em geral, ao realizar uma análise estatística, o usuário não irá implementar novas funções, e sim irá utilizar as funções já disponíveis no R.

As funções no R possuem um nome seguido de zero ou mais argumentos entre parênteses. Vejamos alguns exemplos de funções. Os argumentos podem ser especificados pela posição na lista de argumentos ou pelo nome.

```
x <- c(1, 4, 6, 1, 10) # cria um vetor contendo os números 1, 4, 6, 1 e 10
sort(x, decreasing=TRUE) # ordena x em ordem decrescente
```

```
## [1] 10 6 4 1 1
```

```
min(x) # menor valor de x
```

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

```
max(x) # maior valor de x
```

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

```
sum(x) # somatório dos elementos de x
```

```
## [1] 22
```

```
prod(x) # produto dos elementos de x
```

```
## [1] 240
```

```
length(x) # tamanho do vetor x (número de elementos)
```

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

```
unique(x) # elementos distintos de x
```

```
## [1] 1 4 6 10
```

Mais detalhes de funções serão apresentados no capítulo 11.

3.5 Vetores (*vector*)

Os conteúdos desta seção e da seção 3.5.1 podem ser visualizados neste [vídeo](#).

A figura 3.4 mostra um conjunto de dados contendo 1339 observações e 8 variáveis.

Cada uma dessas variáveis é tratada no R como uma coleção de valores da mesma classe. Nesse conjunto de dados, cada variável é chamada de vetor no R.

A coleção formada pelas 8 variáveis forma um objeto da classe *list* (lista). Então cada uma das variáveis é um vetor e os oito vetores formam uma lista. Essa é uma lista especial onde cada um dos vetores possuem o mesmo número de elementos.

Um objeto que encapsula este conjunto de dados é da classe *data.frame*. Assim um *data frame* contém uma lista na qual todos os seus componentes possuem o mesmo número de elementos.

Essas classes serão apresentadas nesta e nas próximas seções.

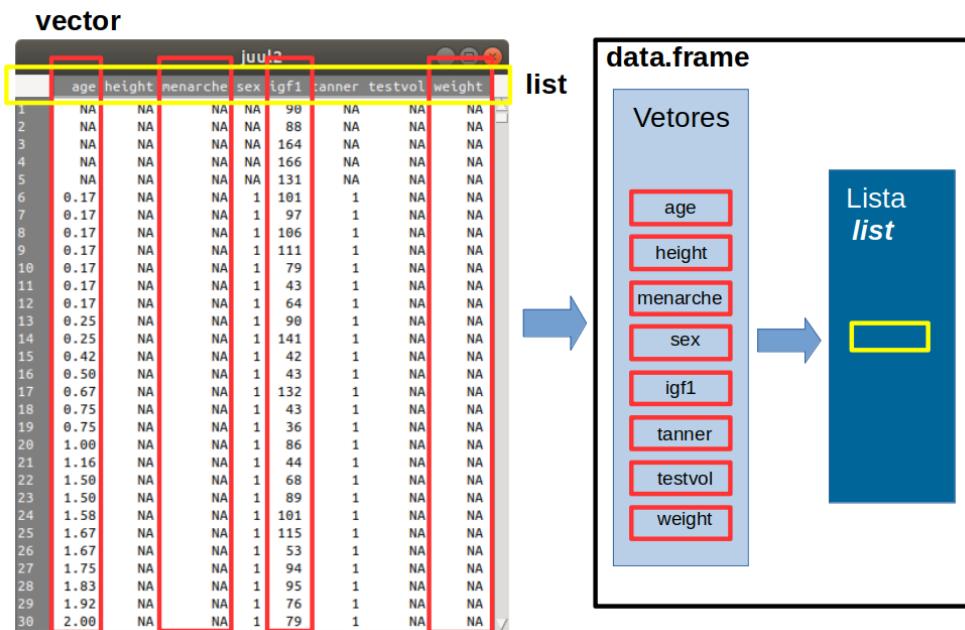


Figura 3.4: Representação de um conjunto de dados no R. Neste conjunto de dados, cada variável é um vetor no R. O conjunto das variáveis forma uma lista especial, da classe *data.frame*. Conjunto de dados: *juul2*, pacote *ISwR* (GPL-2 | GPL-3).

Uma maneira de criarmos vetores é usarmos a função `c(entradas do vetor separadas por vírgulas)`. O operador “`:`” entre dois números inteiros retorna uma sequência de números inteiros entre os dois números. Vejamos alguns exemplos.

```
c(3,7,2,12)
```

```
## [1] 3 7 2 12
```

```
v = c("Luiz", "Maria", "Rafael") # vetor de nomes  
v
```

```
## [1] "Luiz"   "Maria"  "Rafael"
```

```
x = c(TRUE, FALSE, FALSE) # vetor de elementos lógicos  
x
```

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

```
z = 10:30 # também é um vetor  
z
```

```
## [1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

```
z = 30:10 # ordem decrescente  
z
```

```
## [1] 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10
```

```
v = c(7,9)  
class(v) # a classe de um vetor é a classe de cada um de seus elementos  
## [1] "numeric"
```

Podemos gerar um vetor por meio da concatenação de um outro vetor com outros elementos:

```
c(v, 11, 3)
```

```
## [1] 7 9 11 3
```

A função `vector()` também pode ser usada para inicializar vetores.

```
x <- vector("numeric", length = 15)  
x
```

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

3.5.1 Acessando elementos de um vetor

Um elemento de um vetor pode ser acessado por meio de um índice entre colchetes que indica a posição do elemento no vetor. Vejam o exemplo abaixo.

```
x <- c(-3, 4, 7, 9, 10)
x[3]    # indica o terceiro elemento do vetor x (7)

## [1] 7
```

Múltiplos elementos podem ser extraídos por meio de um índice que consiste de um vetor de inteiros:

```
y <- 10:20
y[3:5]    # retorna o 3º, 4º e o 5º elementos de y

## [1] 12 13 14

y[c(4, 2)] # retorna o 4º e o 2º elementos de y, nesta ordem

## [1] 13 11
```

Índices negativos significam que os elementos indicados pelos índices não serão selecionados:

```
y[-3]    # retorna todos os elementos de y, menos o 3º

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

y[c(-3, -8)]    # retorna todos os elementos de y, menos o 3º e o 8º

## [1] 10 11 13 14 15 16 18 19 20
```

O comando abaixo retorna os elementos de y, com exceção dos situados entre o 3º e o 8º, inclusive:

```
y[-3:-8]

## [1] 10 11 18 19 20
```

Para selecionarmos elementos de um vetor, podemos utilizar também uma expressão lógica, com os operadores == (igual), > (maior), < (menor), >= (maior ou igual), <= (menor ou igual), != (diferente), ou combinações de expressões lógicas por meio dos operadores booleanos “OU” (|), “E” (&), e “Não” (!).

Este [vídeo](#) mostra alguns exemplos do uso de expressões lógicas.

Vejamos alguns exemplos de uso de expressões lógicas para selecionar elementos de um vetor:

```

y[y == 15]    # seleciona o elemento de y igual a 15
## [1] 15

y[y < 14]    # seleciona os elementos de y menores que 14
## [1] 10 11 12 13

y[!(y < 17) ]  # seleciona os elementos de y não menores do que 17
## [1] 17 18 19 20

```

O comando abaixo seleciona os elementos de y menores que 13 ou maiores que 18. Repare o uso dos parênteses. Nesse caso, eles não são necessários, mas facilita a leitura da expressão lógica.

```

y[(y < 13) | (y > 18)]
## [1] 10 11 12 19 20

```

O comando abaixo seleciona os elementos de y maiores do que 16 e menores do que 19.

```

y[(y > 16) & (y < 19)]
## [1] 17 18

```

Os operadores lógicos também funcionam com valores textuais, já que o R considera a ordem alfabética.

```

x = c("Antônio", "Joana", "Carla")
x[x > "H"]
## [1] "Joana"

```

As operações de seleção por meio de expressões lógicas são equivalentes a gerar um vetor de valores lógicos que indica em cada posição se o elemento correspondente deve ser ou não selecionado e utilizar este vetor para realizar a seleção.

O comando abaixo seleciona os elementos de y menores ou iguais a 13.

```

y[y <= 13]
## [1] 10 11 12 13

```

O mesmo resultado pode então ser obtido assim:

- a) o comando abaixo cria um vetor com 11 elementos, sendo os 4 primeiros *TRUE* e os restantes *FALSE*, porque os 4 primeiros elementos de *y* são menores do que 13:

```
u <- y <= 13
u
```

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

- b) o comando abaixo seleciona todos os elementos de *y* cujos elementos na posição correspondente em *u* possuem o valor *TRUE*:

```
y[u]
```

```
## [1] 10 11 12 13
```

Elementos de um vetor também podem ser alterados por meio dos operadores de atribuição. Vejam o exemplo abaixo, onde o terceiro elemento do vetor foi alterado para o valor 50:

```
y[3] <- 50 # altera o terceiro elemento do vetor y
y           # elementos do vetor y após a alteração do terceiro elemento
```

```
## [1] 10 11 50 13 14 15 16 17 18 19 20
```

3.5.2 Gerando sequências - Função *seq*

Os conteúdos desta seção e da seção seguinte podem ser visualizados neste [vídeo](#).

A função *seq* gera uma sequência do valor do argumento *from* até o valor do argumento *to*, com salto entre valores consecutivos de acordo com o argumento *by*. Caso os nomes dos argumentos não sejam especificados na chamada da função, o primeiro valor corresponde ao argumento *from*, o segundo valor corresponde ao argumento *to*, e o terceiro valor ao argumento *by*.

Para sequências decrescentes, o valor do argumento *by* obrigatoriamente deverá vir acompanhado do sinal negativo.

O término da sequência será o segundo argumento ou algum número anterior a este na sequência, a depender do valor do terceiro argumento.

A função *seq* pode ser escrita apenas com os dois primeiros argumentos. Nesse caso, o terceiro argumento será, por padrão, igual a 1 (para sequências crescentes) ou -1 (para sequências decrescentes).

```
# Alguns exemplos de sequências
seq(10, 22, 3)
```

```
## [1] 10 13 16 19 22
```

```

seq(10,22,5)

## [1] 10 15 20

seq(22,10,-3)

## [1] 22 19 16 13 10

seq(22,10,-5)

## [1] 22 17 12

seq(3,8) # 3<8, logo 3º argumento implícito = 1

## [1] 3 4 5 6 7 8

3:8 # gera o mesmo resultado acima

## [1] 3 4 5 6 7 8

seq(8,3) # 8>3, logo 3º argumento implícito = -1

## [1] 8 7 6 5 4 3

8:3 # gera o mesmo resultado acima

## [1] 8 7 6 5 4 3

seq(0, 1, .1) # sequência de 0 a 1 com incremento de 0,1

## [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

```

3.5.3 Gerando repetições: função *rep*

A função *rep*, como o nome indica, gera uma sequência onde o primeiro argumento representa a expressão a ser repetida. O argumento *times* representa o número de repetições.

```

rep(0,6) # repetição de um escalar

## [1] 0 0 0 0 0 0

rep(c(1,7),4) # repetição de um vetor

## [1] 1 7 1 7 1 7 1 7

```

```

rep(seq(-1,1,0.5),2) # repetição de uma sequência

## [1] -1.0 -0.5  0.0  0.5  1.0 -1.0 -0.5  0.0  0.5  1.0

rep(c(1,3,5),c(3,2,1)) # gera o vetor (1,1,1,3,3,5)

## [1] 1 1 1 3 3 5

```

Reparam a diferença entre o segundo e o quarto exemplo acima. O segundo exemplo, `rep(c(1,7), 4)`, significa que o vetor (1,7) será repetido quatro vezes. O quarto exemplo, `rep(c(1,3,5),c(3,2,1))`, indica que o primeiro elemento do primeiro argumento será repetido 3 vezes, o segundo elemento 2 vezes e o terceiro elemento 1 vez.

Um pouco mais sobre a criação de vetores numéricos pode ser visto neste [vídeo](#).

3.6 Listas (*list*)

Os conteúdos desta seção e suas subseções, bem como da seção 3.7 podem ser visualizados neste [vídeo](#).

Listas são coleções de elementos que não precisam ser da mesma classe. *list* é uma importante classe de objetos no R.

Listas podem ser criadas explicitamente usando a função `list()`. Vejamos alguns exemplos:

```

x <- list(5, "kiwi", c(FALSE, TRUE, TRUE), 3 + 1i, 2L)
x

## [[1]]
## [1] 5
##
## [[2]]
## [1] "kiwi"
##
## [[3]]
## [1] FALSE  TRUE  TRUE
##
## [[4]]
## [1] 3+1i
##
## [[5]]
## [1] 2

class(x)

## [1] "list"

```

```
x[[2]]  
## [1] "kiwi"
```

Observações:

- 1) uma lista pode conter vetores e outras listas;
- 2) cada componente da lista pode ser acessado por meio de um índice entre dois colchetes `[]`. Por exemplo, no exemplo acima, `x[[2]] = "kiwi"`;

Uma lista vazia com um comprimento pré-especificado pode ser criada por meio da função `vector()`. Todos os elementos de uma lista vazia são inicializados como NULL.

```
y <- vector("list", length = 3)  
y
```

```
## [[1]]  
## NULL  
##  
## [[2]]  
## NULL  
##  
## [[3]]  
## NULL
```

3.6.1 Extraindo múltiplos elementos de uma lista

Podemos extrair vários elementos da lista de maneira semelhante ao utilizado para vetores:

```
x <- list(5, "kiwi", c(FALSE, TRUE, TRUE), 3 + 1i, 2L)  
x[c(3,1)] # seleciona o 3º e o 1º componentes da lista x, nesta ordem
```

```
## [[1]]  
## [1] FALSE TRUE TRUE  
##  
## [[2]]  
## [1] 5
```

3.6.2 Extraindo elementos aninhados de uma lista

Uma lista pode ser composta de elementos que são eles próprios uma lista ou vetores. Elementos de uma lista que pertence a outra lista são chamados elementos aninhados.

Há duas maneiras de acessar um elemento aninhado de uma lista. Vejamos o exemplo a seguir:

```

x <- list(5, "kiwi", c(FALSE, TRUE, TRUE), 3 + 1i, 2L)
x[[c(3,1)]] # seleciona o 1º do 3º componente da lista x

## [1] FALSE
x[[3]][[1]] # mesmo efeito que o comando anterior

## [1] FALSE

```

No exemplo acima, temos uma lista x cujo terceiro elemento é um vetor com três elementos. Para acessarmos o primeiro elemento desse vetor, usamos a função $c()$ entre $[]$, onde o primeiro argumento de c indica a posição do vetor na lista e o segundo argumento a posição do elemento desejado dentro do vetor. Também podemos usar uma sequência de índices entre $[]$ até chegarmos ao elemento desejado na lista.

3.7 Nomes

Vetores e listas podem ser criados com cada elemento recebendo um nome, e esses nomes podem ser usados para referenciar o respectivo elemento. Vejamos alguns exemplos:

```

x <- c(a = 1, b = 2)
x

## a b
## 1 2

x["a"]

## a
## 1

x[1]

## a
## 1

y <- list(a = 1, b = FALSE, c = "kiwi")
y

## $a
## [1] 1
##
## $b
## [1] FALSE
##
## $c
## [1] "kiwi"

```

```
y$c  
## [1] "kiwi"
```

```
y[[3]]  
## [1] "kiwi"
```

Observações:

- 1) cada elemento do vetor pode ser acessado pela sua posição ou pelo seu nome entre aspas;
- 2) cada elemento de uma lista pode ser acessado pela sua posição entre [[]] ou pelo seu nome precedido por “\$”.

Para sabermos os nomes de uma lista ou vetor, usamos a função *names()*:

```
names(x)  
## [1] "a" "b"  
  
names(y)  
## [1] "a" "b" "c"
```

3.8 Fatores

O conteúdo desta seção pode ser visualizado neste [vídeo](#).

A figura 3.5 mostra parte do conteúdo do conjunto de dados *PimaIndiansDiabetes2*, disponível no pacote *mlbench* (GPL-2). Esse conjunto de dados contém observações de 768 mulheres com herança dos indígenas Pima, diabéticas e não diabéticas.

A variável *diabetes* é da classe *factor*. É uma variável dicotômica com dois níveis ou categorias, positivo ou negativo, indicando se a mulher é ou não diabética.

Fatores (*factor*) são usados para representar variáveis categóricas. As variáveis categóricas podem ser nominais (não ordenadas) ou ordinais (ordenadas).

Assim a lista que forma um conjunto de dados no R também pode conter, além de vetores, elementos da classe *factor*.

vector

factor

| | pregnant | glucose | pressure | triceps | insulin | mass | pedigree | age | diabetes |
|----|----------|---------|----------|---------|---------|------|----------|-----|----------|
| 1 | 6 | 148 | 72 | 35 | NA | 33.6 | 0.627 | 50 | pos |
| 2 | 1 | 85 | 66 | 29 | NA | 26.6 | 0.351 | 31 | neg |
| 3 | 8 | 183 | 64 | NA | NA | 23.3 | 0.672 | 32 | pos |
| 4 | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | neg |
| 5 | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | pos |
| 6 | 5 | 116 | 74 | NA | NA | 25.6 | 0.201 | 30 | neg |
| 7 | 3 | 78 | 50 | 32 | 88 | 31.0 | 0.248 | 26 | pos |
| 8 | 10 | 115 | NA | NA | NA | 35.3 | 0.134 | 29 | neg |
| 9 | 2 | 197 | 70 | 45 | 543 | 30.5 | 0.158 | 53 | pos |
| 10 | 8 | 125 | 96 | NA | NA | NA | 0.232 | 54 | pos |
| 11 | 4 | 110 | 92 | NA | NA | 37.6 | 0.191 | 30 | neg |
| 12 | 10 | 168 | 74 | NA | NA | 38.0 | 0.537 | 34 | pos |
| 13 | 10 | 139 | 80 | NA | NA | 27.1 | 1.441 | 57 | neg |
| 14 | 1 | 189 | 60 | 23 | 846 | 30.1 | 0.398 | 59 | pos |
| 15 | 5 | 166 | 72 | 19 | 175 | 25.8 | 0.587 | 51 | pos |
| 16 | 7 | 100 | NA | NA | NA | 30.0 | 0.484 | 32 | pos |
| 17 | 0 | 118 | 84 | 47 | 230 | 45.8 | 0.551 | 31 | pos |
| 18 | 7 | 107 | 74 | NA | NA | 29.6 | 0.254 | 31 | pos |
| 19 | 1 | 103 | 30 | 38 | 83 | 43.3 | 0.183 | 33 | neg |
| 20 | 1 | 115 | 70 | 30 | 96 | 34.6 | 0.529 | 32 | pos |
| 21 | 3 | 126 | 88 | 41 | 235 | 39.3 | 0.704 | 27 | neg |
| 22 | 8 | 99 | 84 | NA | NA | 35.4 | 0.388 | 50 | neg |
| 23 | 7 | 196 | 90 | NA | NA | 39.8 | 0.451 | 41 | pos |
| 24 | 9 | 119 | 80 | 35 | NA | 29.0 | 0.263 | 29 | pos |
| 25 | 11 | 143 | 94 | 33 | 146 | 36.6 | 0.254 | 51 | pos |
| 26 | 10 | 125 | 70 | 26 | 115 | 31.1 | 0.205 | 41 | pos |
| 27 | 7 | 147 | 76 | NA | NA | 39.4 | 0.257 | 43 | pos |
| 28 | 1 | 97 | 66 | 15 | 140 | 23.2 | 0.487 | 22 | neg |
| 29 | 13 | 145 | 82 | 19 | 110 | 22.2 | 0.245 | 57 | neg |
| 30 | 5 | 117 | 92 | NA | NA | 34.1 | 0.337 | 38 | neg |
| 31 | 5 | 109 | 75 | 26 | NA | 36.0 | 0.546 | 60 | neg |
| 32 | 3 | 158 | 76 | 36 | 245 | 31.6 | 0.851 | 28 | pos |
| 33 | 3 | 88 | 58 | 11 | 54 | 24.8 | 0.267 | 22 | neg |
| 34 | 6 | 92 | 92 | NA | NA | 19.9 | 0.188 | 28 | neg |
| 35 | 10 | 122 | 78 | 31 | NA | 27.6 | 0.512 | 45 | neg |

Figura 3.5: Exemplo de um conjunto de dados composto por vetores e fatores. Fonte: *PimaIndiansDiabetes2*, pacote [mlbench](#) (GPL-2)...

Objetos do tipo *factor* são criados por meio da função *factor*. Os valores possíveis de um objeto do tipo fator são chamados de níveis (*levels*). Os comandos a seguir criam o objeto *dor* da classe *factor* com os níveis *nenhuma*, *leve*, *razoável* e *muita dor*:

```
dor = factor(c("nenhuma", "nenhuma", "leve", "razoável",
               "muita dor", "leve"))
dor
```

```
## [1] nenhuma  nenhuma  leve      razoável  muita dor leve
## Levels: leve muita dor nenhuma razoável
```

Usar fatores com rótulos é melhor do que usar inteiros, porque os rótulos são autoexplicativos. Por exemplo, trabalhar com uma variável chamada *sexo* que possui os valores “masculino” e “feminino” é melhor do que uma variável que possui os valores 1 e 2.

Podemos ordenar os níveis de um fator por meio da função *levels*. Se nada for especificado, o R vai assumir a ordem alfabética, como indicado no exemplo anterior. Em muitas situações, é necessário especificar uma ordem mais natural para os níveis do fator, como mostrado a seguir, que ordena os níveis de acordo com a intensidade da dor:

```
levels(dor) = c("nenhuma", "leve", "razoável", "muita dor")
dor

## [1] razoável razoável nenhuma muita dor leve      nenhuma
## Levels: nenhuma leve razoável muita dor
```

Também podemos especificar a ordem dos níveis diretamente na criação do fator, por meio do argumento *levels*:

```
dor <- factor(c("nenhuma", "nenhuma", "leve", "razoável",
                 "muita dor", "leve"),
                  levels = c("nenhuma", "leve", "razoável", "muita dor"))
dor

## [1] nenhuma nenhuma leve      razoável muita dor leve
## Levels: nenhuma leve razoável muita dor
```

3.9 Coerção

O conteúdo desta seção pode ser visualizado neste [vídeo](#).

Ao criar um vetor com valores de diferentes classes, o R tenta uma maneira de transformar todos os objetos como sendo de uma mesma classe. Essa transformação é chamada de *coerção*. Para as classes atômicas, a conversão é feita de acordo com a seguinte sequência:

$$\textit{logical} \rightarrow \textit{integer} \rightarrow \textit{numeric} \rightarrow \textit{complex} \rightarrow \textit{character}$$

Exemplos:

```
y <- c(TRUE, 2L, FALSE)
y

## [1] 1 2 0

class(y)

## [1] "integer"
```

No exemplo acima, a classe do objeto *y* é *integer*, porque o número 2 (da classe *integer*) sucede aos elementos da classe *logical* na sequência de conversão. Ao convertermos um valor lógico para numérico, *TRUE* é equivalente a 1 e *FALSE* a 0.

```
y <- c(1.7, 4L)
```

```
y
```

```
## [1] 1.7 4.0
```

```
class(y)
```

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

No exemplo acima, a classe do objeto *y* é *numeric* e o número 4 (*integer*) é convertido para *numeric*.

```
y <- c(1.7, 3 + 5i)
```

```
y
```

```
## [1] 1.7+0i 3.0+5i
```

```
class(y)
```

```
## [1] "complex"
```

```
y <- c(1.7 + 2.3i, "a")
```

```
y
```

```
## [1] "1.7+2.3i" "a"
```

```
class(y)
```

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

Objetos podem ser convertidos de uma classe para outra por meio das funções *as.**: *as.numeric*, *as.logical*, *as.integer*, *as.character*, *as.complex*. Outras funções de coerção serão apresentadas à medida que for necessário.

Exemplos:

```
x <- c(0, 1, 4, 5.8)
as.logical(x)      # 0 é convertido em FALSE, os outros números em TRUE
```

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

```
as.character(x)
```

```
## [1] "0"    "1"    "4"    "5.8"
```

Quando o R não consegue realizar a conversão, *NA* é gerado como resultado e uma mensagem de alerta é emitida.

```
x <- c("a", "b", "c", 8.8)
as.numeric(x)

## Warning: NAs introduzidos por coerção
## [1] NA NA NA 8.8
```

A classe do objeto *x* é *character*. Portanto o elemento 8.8 é tratado como *character* e não como número, mas ele pode ser convertido para número. Os elementos “a”, “b” e “c” não podem ser convertidos nem para números nem para valores lógicos. Portanto os três primeiros elementos da avaliação de *as.numeric(x)* são *NA* e o último é o número 8.8.

```
as.logical(x)
```

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

Já a avaliação de *as.logical(x)* retorna todos os elementos *NA*, já que mesmo o elemento do tipo *character* de *x* (“8.8”) não pode ser convertido para um valor lógico.

Vamos supor que temos uma variável *sexo* representada como um vetor numérico, onde 1 representa feminino e 2 masculino.

```
sexo = c(1, 2, 2, 1, 1, 1, 2)
```

Na análise estatística, precisamos converter essa variável de numérica para categórica. Podemos usar a função *as.factor* como mostrado abaixo, onde nomeamos o objeto transformado como *sexo.cat*.

```
sexo.cat = as.factor(sexo)
sexo.cat
```

```
## [1] 1 2 2 1 1 1 2
## Levels: 1 2
```

Agora vamos colocar nomes descritivos aos níveis da variável *sexo.cat*, usando o atributo *levels*, como abaixo:

```
levels(sexo.cat) = c("feminino", "masculino")
sexo.cat[1:5]

## [1] feminino masculino masculino feminino feminino
## Levels: feminino masculino
```

3.10 Matrizes

Os conteúdos desta seção e de sua subseção podem ser visualizados neste [vídeo](#).

Matrizes são vetores com um atributo de dimensão (*dimension*), que é ele próprio um vetor inteiro de tamanho 2: número de linhas e número de colunas.

Uma matriz é criada por meio da função *matrix()*. Essa função possui os seguintes argumentos:

- *data*: vetor que contém os valores da matriz;
- *nrow*: número de linhas da matriz;
- *ncol*: número de colunas da matriz;
- *byrow*: indica como a matriz é montada. Se for TRUE, ela é montada linha a linha. Se for FALSE, ela é montada coluna a coluna;
- *dimnames*: nomes das linhas e colunas, respectivamente.

```
m = matrix(data = 1:6, nrow = 2, ncol = 3, byrow = T,
            dimnames = list(c("trat1", "trat2"), c("out1", "out2", "out3")))
m

##      out1 out2 out3
## trat1    1    2    3
## trat2    4    5    6
```

No exemplo acima, a matriz vai sendo preenchida com os números de 1 a 6, linha a linha.

O comando abaixo vai gerar uma matriz com os mesmos elementos da matriz acima, mas omitindo o nome do argumento *data* e o argumento *nrow*. Quando o nome de um argumento não é especificado, o R tenta associar um objeto à respectiva posição do argumento.

```
m = matrix(1:6, ncol = 3, byrow = T)
m

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

Assim, na chamada da função *matrix* acima, o argumento *data* recebe o vetor de 1 a 6, por ser o primeiro argumento da função. Para facilidade de leitura, é recomendável colocar os nomes dos argumentos.

Se *dimnames* não for especificado, os nomes das linhas e colunas serão os números das respectivas linhas e colunas.

Como há 6 elementos no argumento *data* e 3 colunas, então o número de linhas é calculado dividindo-se o número de elementos (6) pelo número de colunas (3).

Também poderíamos ter especificado o argumento *nrow* ao invés do argumento *ncol*, como mostrado a seguir. O número de colunas será obtido dividindo-se o número de elementos (6) pelo número de linhas (2).

Se o argumento *byrow* não for especificado, então a matriz será construída a partir das colunas. A matriz vai sendo preenchida com os números de 1 a 6 coluna a coluna.

```
m = matrix(1:6, nrow = 2)
m
```

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

Se os valores da matriz não forem especificados, as células da matriz são preenchidas com *NA*.

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

```
##      [,1] [,2] [,3]
## [1,]    NA   NA   NA
## [2,]    NA   NA   NA
```

Para obtermos a dimensão de uma matriz, usamos a função *dim*.

```
dim(m)
```

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

Se somente um valor for especificado para o argumento *data* da função *matrix*, a matriz será criada com todos os valores iguais.

```
matrix(data = 2, nrow = 3, ncol = 2)
```

```
##      [,1] [,2]
## [1,]    2    2
## [2,]    2    2
## [3,]    2    2
```

Matrizes também podem ser criadas a partir de vetores, adicionando um atributo de dimensão por meio da função *dim()*.

```
m <- 1:10
dim(m) <- c(2, 5)
m
```

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

Outra maneira de se criar matrizes é pela adição de linhas ou colunas por meio das funções `rbind()` e `cbind()`, respectivamente. `rbind()` monta uma matriz, empilhando os argumentos linha a linha na ordem em que aparecem na lista. Já `cbind()` monta a matriz colocando os argumentos lado a lado na ordem em que aparecem na lista.

```
x <- 1:4
y <- 10:13
rbind(x, y)

##   [,1] [,2] [,3] [,4]
## x     1     2     3     4
## y    10    11    12    13

cbind(x, y)

##      x  y
## [1,] 1 10
## [2,] 2 11
## [3,] 3 12
## [4,] 4 13
```

3.10.1 Extraindo células/linhas/colunas/submatrizes de uma matriz

Os elementos (células) de uma matriz podem ser acessados por um par de índices que indicam, respectivamente, a linha e a coluna do elemento desejado.

```
x = 1:16
A = matrix(data = x, nrow = 4, ncol = 4, byrow=TRUE)
A[2,4] # valor na segunda linha e quarta coluna de A

## [1] 8
```

Para extrairmos todos os elementos de uma linha, basta não especificarmos o índice correspondente à coluna.

```
A[3,] # terceira linha de A

## [1] 9 10 11 12
```

Para extrairmos todos os elementos de uma coluna, basta não especificarmos o índice correspondente à linha

```
A[,2] # segunda coluna de A

## [1] 2 6 10 14
```

Índices negativos excluem os elementos indicados.

```
A[-3,] # matriz A sem a terceira linha
```

```
##      [,1] [,2] [,3] [,4]
## [1,]     1     2     3     4
## [2,]     5     6     7     8
## [3,]    13    14    15    16
```

```
A[,-3] # matriz A sem a terceira coluna
```

```
##      [,1] [,2] [,3]
## [1,]     1     2     4
## [2,]     5     6     8
## [3,]     9    10    12
## [4,]    13    14    16
```

Podemos utilizar vetores de índices para selecionarmos uma submatriz de uma matriz.

```
A[1:3,] # submatriz de A: elementos da primeira à terceira linha
```

```
##      [,1] [,2] [,3] [,4]
## [1,]     1     2     3     4
## [2,]     5     6     7     8
## [3,]     9    10    11    12
```

```
A[,2:4] # submatriz de A: elementos da segunda à quarta coluna
```

```
##      [,1] [,2] [,3]
## [1,]     2     3     4
## [2,]     6     7     8
## [3,]    10    11    12
## [4,]    14    15    16
```

O comando a seguir seleciona a submatriz de A formada pelos elementos da primeira à terceira linha e da segunda à quarta coluna.

```
A[1:3,2:4]
```

```
##      [,1] [,2] [,3]
## [1,]     2     3     4
## [2,]     6     7     8
## [3,]    10    11    12
```

O comando a seguir seleciona a submatriz de A formada pelos elementos da primeira e terceira linhas, segunda e quarta colunas.

```
A[c(1,3),c(2,4)]
```

```
##      [,1] [,2]
## [1,]     2     4
## [2,]    10    12
```

3.11 Data frame

Os conteúdos desta seção e de sua subseção podem ser visualizados neste [vídeo](#).

Data frames são utilizados para armazenar dados tabulados no R. Eles são usualmente criados quando lemos um arquivo de dados para ser analisado. Internamente no R, os *data frames* são representados como uma lista especial onde cada elemento da lista possui o mesmo tamanho. Cada elemento da lista pode ser interpretado como uma coluna e o tamanho de cada elemento da lista é o número de linhas ou observações. Como se trata de uma lista, os elementos de um *data frame* não precisam ser da mesma classe.

Um *data frame* pode ser criado explicitamente por meio da função *data.frame()*.

```
pacientes = data.frame(id = c("P1", "P2", "P3", "P4"),
                        sexo = c("feminino", "feminino", "masculino", "masculino"),
                        pad=c(80, 85, 100, 95), pas = c(130, 140, 150, 145))
pacientes
```

```
##   id     sexo pad pas
## 1 P1  feminino  80 130
## 2 P2  feminino  85 140
## 3 P3 masculino 100 150
## 4 P4 masculino  95 145
```

Temos aqui um *data frame* com 4 variáveis, nomeadas: *id*, *sexo*, *pad* e *pas*.

O atributo *names* de um data frame retorna os nomes das colunas ou variáveis do *data frame*.

```
names(pacientes)
```

```
## [1] "id"   "sexo" "pad"  "pas"
```

Um outro atributo de um data frame é chamado *row.names*, que fornece uma informação sobre cada linha do *data frame*.

```
row.names(pacientes)
```

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

Podemos especificar os nomes das linhas, atribuindo valores a *row.names*:

```
row.names(pacientes) = c("Bia", "Claudia", "Leo", "Carlos")
pacientes

##      id      sexo pad pas
## Bia    P1  feminino  80 130
## Claudia P2  feminino  85 140
## Leo     P3 masculino 100 150
## Carlos  P4 masculino  95 145
```

3.11.1 Acessando os elementos de um *data frame*

Os elementos de um *data frame* podem ser acessados como os elementos de uma lista ou de uma matriz. As variáveis podem ser identificadas pela sua posição no *data frame*. Vejamos os seguintes exemplos, utilizando o *data frame* *pacientes*, criado na seção anterior:

```
pacientes$pad      # mostra os valores da pad dos pacientes

## [1] 80 85 100 95

pacientes[, c(3,4)]  # exibe os valores de pad e pas dos pacientes

##      pad pas
## Bia    80 130
## Claudia 85 140
## Leo     100 150
## Carlos  95 145

pacientes[c(1,2), ]  # exibe os dois primeiros registros de pacientes

##      id      sexo pad pas
## Bia    P1  feminino  80 130
## Claudia P2  feminino  85 140

pacientes[, c("id","pad")]  # colunas especificadas pelo nome

##      id pad
## Bia    P1 80
## Claudia P2 85
## Leo     P3 100
## Carlos  P4 95
```

```
pacientes [ , -c(1,3)] # não exibe as colunas 1 e 3
```

```
##           sexo pas
## Bia      feminino 130
## Claudia  feminino 140
## Leo      masculino 150
## Carlos   masculino 145
```

```
pacientes [ , c(-1,-3)] # idem anterior
```

```
##           sexo pas
## Bia      feminino 130
## Claudia  feminino 140
## Leo      masculino 150
## Carlos   masculino 145
```

```
pacientes [c(1,4) , c(3,4)] # exibe pad e pas dos pacientes 1 e 4
```

```
##           pad pas
## Bia      80 130
## Carlos   95 145
```

```
pacientes[pacientes$sexo == "feminino",] # pacientes do sexo feminino
```

```
##           id     sexo pad pas
## Bia      P1 feminino 80 130
## Claudia P2 feminino 85 140
```

```
pacientes[pacientes$pas > 140,] # seleciona os pacientes com pas > 140
```

```
##           id     sexo pad pas
## Leo      P3 masculino 100 150
## Carlos   P4 masculino 95 145
```

```
subconjunto <- pacientes [ , -1] # cria um novo data frame sem os ids  
subconjunto
```

```
##           sexo pad pas
## Bia      feminino 80 130
## Claudia  feminino 85 140
## Leo      masculino 100 150
## Carlos   masculino 95 145
```

3.12 Valores ausentes

Como mostrado anteriormente, valores ausentes (*missing values*) são denotados por *NA* ou *NAN*.

Um valor *NAN* também é *NA*, mas a recíproca não é verdadeira. As funções *is.na()* e *is.nan()* são usadas para testar se um objeto é *NA* ou *NAN*, respectivamente.

```
x <- c(1, 2, NA, 10, 3) # vetor com um elemento NA
```

```
is.na(x) # retorna um vetor lógico indicando que elementos são NA
```

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

```
is.nan(x) # retorna um vetor lógico indicando que elementos são NAN
```

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

```
x <- c(1, 2, NA, 10, NaN) # vetor com elementos NA e NaN
```

```
is.na(x) # observem que NaN também é considerado como NA
```

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

```
is.nan(x) # observem que NA não é considerado um NaN
```

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

```
x[ !is.na(x)] # remove todos os elementos NA ou NaN de x
```

```
## [1] 1 2 10
```

Observem no último exemplo que podemos utilizar a função *is.na()* para selecionarmos os elementos NA de um vetor.

3.13 Ajuda no R

Para obtermos ajuda sobre alguma função do R, basta digitarmos o nome da função precedido de uma interrogação na janela do R e apertar <Enter>. De maneira equivalente, podemos usar a função *help("nome_da_função")*. Para solicitarmos a ajuda sobre a função *data.frame()*, por exemplo, podemos usar um dos dois comandos a seguir:

```
help("data.frame")
```

```
?data.frame
```

A figura 3.6 exibe a descrição e argumentos da função *data.frame* na aba *Help*, ao executarmos a função `help("data.frame")` na console do *RStudio*.

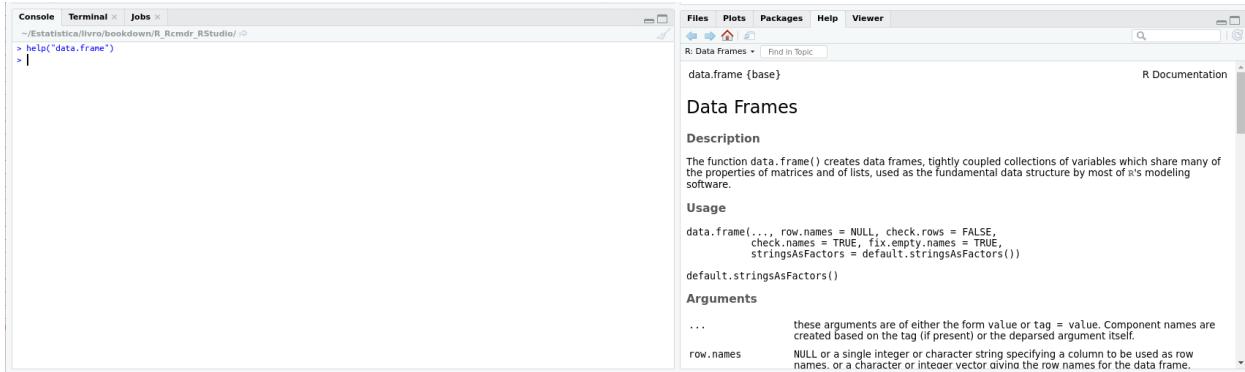


Figura 3.6: Aba de Ajuda do RStudio, mostrando a descrição e os argumentos da função `data.frame`.

Para localizar funções relacionadas a alguma expressão, usam-se duas interrogações em vez de uma antes da expressão, ou ainda a função `help.search("expressão_para_a_busca")`.

Exemplo: Qual função retorna o desvio padrão?

```
?deviation # nada encontrado
??deviation # exibe uma lista de funções que contêm a palavra deviation
            # na sua descrição
help.search("deviation") # idem ??deviation
?sd # ajuda sobre a função sd do pacote stats
```

3.14 Instalação de pacotes via linha de comando

Para instalar um pacote no R, sem a necessidade de usarmos uma interface gráfica, como foi mostrado na instalação do *R Commander*, usamos a função `install.packages("nome_do_pacote")`. Vamos instalar o pacote *ISwR*, que será utilizado em outros capítulos. Executamos a função abaixo, onde o nome do pacote deve aparecer entre “”:

```
install.packages("ISwR")
```

3.15 Exercícios

- 1) Quais são os resultados das seguintes expressões, respectivamente: $3 + 5*8/2^2$ e $(3 + 5)*8/(2^2)$?
 - a) 13 e 16
 - b) 16 e 13
 - c) 403 e 13
 - d) 403 e 16
 - e) 16 e 16
- 2) Na expressão $30 - 6^2 / 2^3$, qual a sequência dos cálculos até chegar ao resultado final?
 - a) $30-6$, 24^2 , $576/3$, $192*3$
 - b) $30-6$, $2/2$, 1^3 , $30-27$
 - c) 6^2 , $30-36$, 2^3 , $-6/6$
 - d) 6^2 , $36/2$, $18*3$, $30-54$
- 3) Após a sequência de comandos $x = 3$; $y = 4$; $x = x + 2*y$; $x=20$, qual será o valor de x?
 - a) 3
 - b) 4
 - c) 11
 - d) 20
 - e) (3, 20)
- 4) Se executarmos a expressão $x = 6$, qual será a classe do objeto x, se usarmos a função class?
 - a) list
 - b) numeric
 - c) integer
 - d) complex
 - e) character
 - f) real
- 5) Associe os vetores abaixo aos respectivos tipos de dados:
 - a) (1, 2.5, 4, 0)
 - b) (TRUE, FALSE, FALSE, TRUE)
 - c) 1:40
 - d) ("a", "Jeannie", 5)
 - e) $(2 + 4i, 3i, 3, 6)$

- 6) Se você tiver dois vetores $x = c(2, 4, 5, 6)$ e $y = c(-3, -2, 0, 7)$, qual será o resultado da expressão `cbind(x, y)`?
- Um vetor de comprimento 8
 - Uma matriz 4x2
 - Uma matriz 2x4
 - Uma lista com dois elementos
 - Um vetor de comprimento 4
- 7) Que afirmações abaixo são verdadeiras em relação aos vetores no R?
- Os elementos de um vetor podem ser de diferentes classes.
 - Os elementos de um vetor precisam ser da mesma classe.
 - Os elementos de um vetor somente podem ser *numeric* ou *character*.
 - Se um vetor contiver elementos da classe *character* e da classe *numeric*, os elementos do tipo *character* são convertidos para *numeric*.
 - Se um vetor contiver elementos da classe *character* e da classe *numeric*, os elementos do tipo *numeric* são convertidos para *character*.
- 8) Suponha que você tenha uma lista definida como $x = list(4, "cara", "coroa", FALSE)$. O que a expressão $x[[1]]$ fornece? Marque todas as opções corretas.
- Um vetor do tipo *character* contendo o elemento “4”
 - Uma lista com o número 4
 - Um vetor numérico contendo o número 4
 - Um vetor numérico de comprimento 1
 - Uma lista contendo o elemento “4”
 - Um vetor contendo o elemento “cara”
- 9) Suponha que você tenha um vetor $x = c(4, 6, 8, 9, 12, 3, 2)$ e que você deseja fazer com que todos os elementos de x menor que 9 sejam iguais a zero. Selecione o código abaixo que gera o resultado esperado.
- $x[x == 0] < 9$
 - $x[x < 9] == 0$
 - $x[x >= 9] <- 0$
 - $x[x != 9] = 0$
 - $x[x < 9] <- 0$

10) Dado o vetor $x = 1:12$, que expressões abaixo irão gerar uma matriz 3x4?

- a) `matrix(x, ncol=4)`
- b) `matrix(x, nrow = 3)`
- c) `matrix(x, ncol=4, byrow = TRUE)`
- d) `matrix(x)`
- e) `matrix(x, nrow = 4)`
- f) `matrix(x, ncol = 3)`

11) Dada a matrix gerada pela expressão $x = matrix(1:12, ncol = 4)$, que afirmações abaixo são verdadeiras?

- a) $x[2,3] = 8$
- b) $x[2,3] = 6$
- c) $x[1,] = (1, 2, 3, 4)$
- d) $x[1,] = (1, 4, 7, 10)$
- e) $x[c(1,3), 3] = (7, 9)$
- f) $x[1, c(2,3)] = (1, 2, 3)$

12) Que afirmações abaixo são verdadeiras em relação a *data frames*?

- a) *data frames* são iguais a matrizes
- b) Os componentes de um *data frame* são listas onde cada lista pode possuir qualquer tamanho
- c) Todos os componentes de um *data frame* têm que ser do mesmo tipo
- d) Todos os componentes de um *data frame* devem possuir o mesmo tamanho
- e) Um estudo clínico, em geral, pode gerar um conjunto de dados, onde cada variável é uma coluna de um *data frame* e cada linha são os valores de cada variável para a unidade de observação correspondente

- 13) A figura 3.7 mostra as 6 primeiras linhas de um *data frame* chamado *Melanoma*. Que afirmações abaixo estão corretas?

| | time | status | sex | age | year | thickness | ulcer |
|---|-------------|---------------|------------|------------|-------------|------------------|--------------|
| 1 | 10 | 3 | 1 | 76 | 1972 | 6.76 | 1 |
| 2 | 30 | 3 | 1 | 56 | 1968 | 0.65 | 0 |
| 3 | 35 | 2 | 1 | 41 | 1977 | 1.34 | 0 |
| 4 | 99 | 3 | 0 | 71 | 1968 | 2.90 | 0 |
| 5 | 185 | 1 | 1 | 52 | 1965 | 12.08 | 1 |
| 6 | 204 | 1 | 1 | 28 | 1971 | 4.84 | 1 |

Figura 3.7: 6 primeiras observações do conjunto de dados Melanoma.

- a) `Melanoma[3,] = c(NA, NA, NA, NA, NA, NA)` faz com que todos os valores da terceira linha se tornem NA
- b) `Melanoma$age[1:6]` retorna o vetor (76, 56, 41, 71, 52, 28)
- c) `Melanoma[[4]][1:6]`, `Melanoma$age[1:6]`, `Melanoma[1:6,4]` e `Melanoma[["age"]][1:6]` retornam o mesmo resultado
- d) `names(Melanoma)` retorna os nomes das variáveis
- e) `names(Melanoma)` retorna os números 1, 2, 3, 4, 5, 6
- 14) Suponha que no conjunto de dados da questão anterior, os valores da variável *sex* fossem *masculino* ou *feminino*. Que expressão você usaria para obter os registros cuja idade fosse menor ou igual a 50 anos e cujo sexo fosse masculino?
- a) `Melanoma[Melanoma$age <= 50 & Melanoma$sex == "masculino",]`
- b) `Melanoma[Melanoma$age <= 50 | Melanoma$sex == masculino,]`
- c) `Melanoma[Melanoma$age <= 50 | Melanoma$sex == "masculino",]`
- d) `Melanoma[Melanoma$age <= 50 & Melanoma$sex == masculino,]`

Resposta dos exercícios:

- 1) a
- 2) d
- 3) d
- 4) b
- 5) a) Numeric, b) logical, c) Integer, d) character, e) complex
- 6) c
- 7) b, e
- 8) c, d
- 9) e
- 10) a, b, c
- 11) a, d, e
- 12) d, e
- 13) a, b, c, d
- 14) a

Capítulo 4

Importando e exportando dados

4.1 Formatos de arquivos de dados

Dados para análise estatística podem estar em diversos formatos (tipos de arquivos). O usuário poderia, por exemplo, ter criado um arquivo num software de planilha eletrônica, ou no próprio pacote estatístico que será utilizado. Alguns dos formatos mais usados são mostrados a seguir:

- Planilha eletrônica (.xls, .xlsx, .ods)
- Valores separados por vírgulas (.csv)
- Dbase (.dbf)
- Diversos Pacotes estatísticos: (.Rdata - R), (.sav - SPSS), etc

A figura 4.1 mostra um arquivo no formato do *Excel* com o nome Melanoma.xlsx, com a mesma estrutura do conjunto de dados *Melanoma* do pacote *MASS* ([GPL-2](#) | [GPL-3](#)) do R. Na primeira linha, aparecem os nomes das variáveis. As demais linhas contêm os valores de cada variável para cada paciente.

Numa planilha com essa estrutura, cada linha do arquivo é chamada de **Registro** ou **Observação**. Ao conjunto de registros, denominamos **conjunto de dados** ou **arquivo de dados**. Observem que cada coluna do conjunto de dados contêm os valores da variável identificada na primeira linha e que deve ser de uma classe do R previamente especificada. Os arquivos a serem analisados em um pacote estatístico devem possuir uma estrutura semelhante.

O conjunto de dados *Melanoma* contém dados de 205 pacientes na Dinamarca com melanoma maligno. As seguintes variáveis foram coletadas nesse estudo (Andersen et al., 1993):

- *time*: tempo de sobrevida em dias, possivelmente censurado;
- *status*: 1 - morreu de melanoma, 2 - vivo, 3 - morreu por outras causas;
- *sex*: 1 - masculino, 0 - feminino;
- *age*: idade em anos;
- *year*: ano da operação;
- *thickness*: espessura do tumor em mm;

- *ulcer*: 1 = presença, 0 = ausência;

The screenshot shows a LibreOffice Calc spreadsheet titled "Melanoma.xlsx - LibreOffice Calc". The menu bar includes Arquivo, Editar, Exibir, Inserir, Formatar, Estilos, Planilha, Dados, Ferramentas, Janela, and Ajuda. The toolbar includes various icons for file operations, text styling, and data manipulation. The spreadsheet has a header row with columns labeled "time", "status", "sex", "age", "year", "thickness", and "ulcer". Rows 2 through 10 contain data points. A green rectangular selection highlights rows 2 through 5, with the text "Conjunto de Dados" overlaid in green. Row 5 is specifically highlighted with a red border and the text "Registro" overlaid in red. The bottom status bar shows "Planilha 1 de 1 | PageStyle_Melanoma | Inglês (EUA) | Média: ; Soma: 0 |".

| | time | status | sex | age | year | thickness | ulcer |
|----|------|--------|-----|-----|------|-----------|-------|
| 2 | 10 | 3 | 1 | 76 | 1972 | 6,76 | 1 |
| 3 | 30 | 3 | 1 | 56 | 1968 | 0,65 | 0 |
| 4 | 35 | 2 | 1 | 41 | 1977 | 1,34 | 0 |
| 5 | 99 | 3 | 0 | 71 | 1968 | 2,9 | 0 |
| 6 | 185 | 1 | 1 | 52 | 1965 | 12,08 | 1 |
| 7 | 204 | 1 | 1 | 28 | 1971 | 4,84 | 1 |
| 8 | 210 | 1 | 1 | 77 | 1972 | 5,16 | 1 |
| 9 | 232 | 3 | 0 | 60 | 1974 | 3,22 | 1 |
| 10 | 232 | 1 | 1 | 49 | 1968 | 12,88 | 1 |

Figura 4.1: Exemplo de um arquivo de dados no formato do *Excel*, com o nome *Melanoma.xlsx*.
Fonte: conjunto de dados *Melanoma* do pacote *MASS* ([GPL-2](#) | [GPL-3](#)).

A figura 4.2 mostra o conjunto de dados *Melanoma* no formato *csv* (Melanoma.csv) com as variáveis e valores exportados a partir do *Excel*. Esse arquivo foi aberto num editor de texto. Apesar da extensão se chamar *valores separados por vírgulas* (comma separated values, em inglês), podemos usar outro caractere para separar os valores. No exemplo, foi utilizado o ponto e vírgula. O formato .csv é muito útil, porque todo pacote estatístico consegue importar um arquivo deste tipo. Assim, quando desejamos analisar dados gerados pelo *Epi Info* no pacote *SPSS*, por exemplo, podemos exportar os dados do *Epi Info* para uma arquivo com a extensão *csv* e, a seguir, importar esse arquivo para o *SPSS*.

Este capítulo mostra como importar arquivos para o R, salvar e carregar arquivos no formato do R, exportar conjuntos de dados do R para outros formatos e ler conjuntos de dados disponíveis em pacotes do R.

| time | status | sex | age | year | thickness | ulcer |
|-------------------------|--------|-----|-----|------|-----------|-------|
| 10;3;1;76;1972;6,76;1 | | | | | | |
| 30;3;1;56;1968;0,65;0 | | | | | | |
| 35;2;1;41;1977;1,34;0 | | | | | | |
| 99;3;0;71;1968;2,9;0 | | | | | | |
| 185;1;1;52;1965;12,08;1 | | | | | | |
| 204;1;1;28;1971;4,84;1 | | | | | | |
| 210;1;1;77;1972;5,16;1 | | | | | | |
| 232;3;0;60;1974;3,22;1 | | | | | | |
| 232;1;1;49;1968;12,88;1 | | | | | | |
| 279;1;0;68;1971;7,41;1 | | | | | | |
| 295;1;0;53;1969;4,19;1 | | | | | | |
| 355;3;0;64;1972;0,16;1 | | | | | | |

Figura 4.2: Arquivo *Melanoma.csv* gerado a partir do arquivo do *Excel* mostrado na figura 4.1.

4.2 Importação de dados do *Excel* (.xls/.xlsx) usando o *RStudio*

O conteúdo dsta seção pode ser visualizado neste [vídeo](#).

Nesta seção, vamos mostrar como importar para o R, a partir do *RStudio*, um arquivo de dados do *Excel*. Como exemplo, será utilizado o arquivo *Melanoma.xlsx*, citado na seção anterior e disponível neste endereço.

Vamos supor que, ao baixarmos o arquivo *Melanoma.xlsx*, ele esteja localizado na pasta temp do disco C de uma partição Windows.

Ao abrirmos o *RStudio*, selecionamos o item *Import dataset* do menu *File* e, em seguida, selecionamos a opção *From Excel...* (figura 4.3). Essa sequência de passos pode ser apresentada como:

File ⇒ Import dataset ⇒ From Excel...

Sempre que uma opção de menu for apresentada, os passos para selecioná-la serão mostrados como acima, tanto no *RStudio* quanto no *R Commander*.

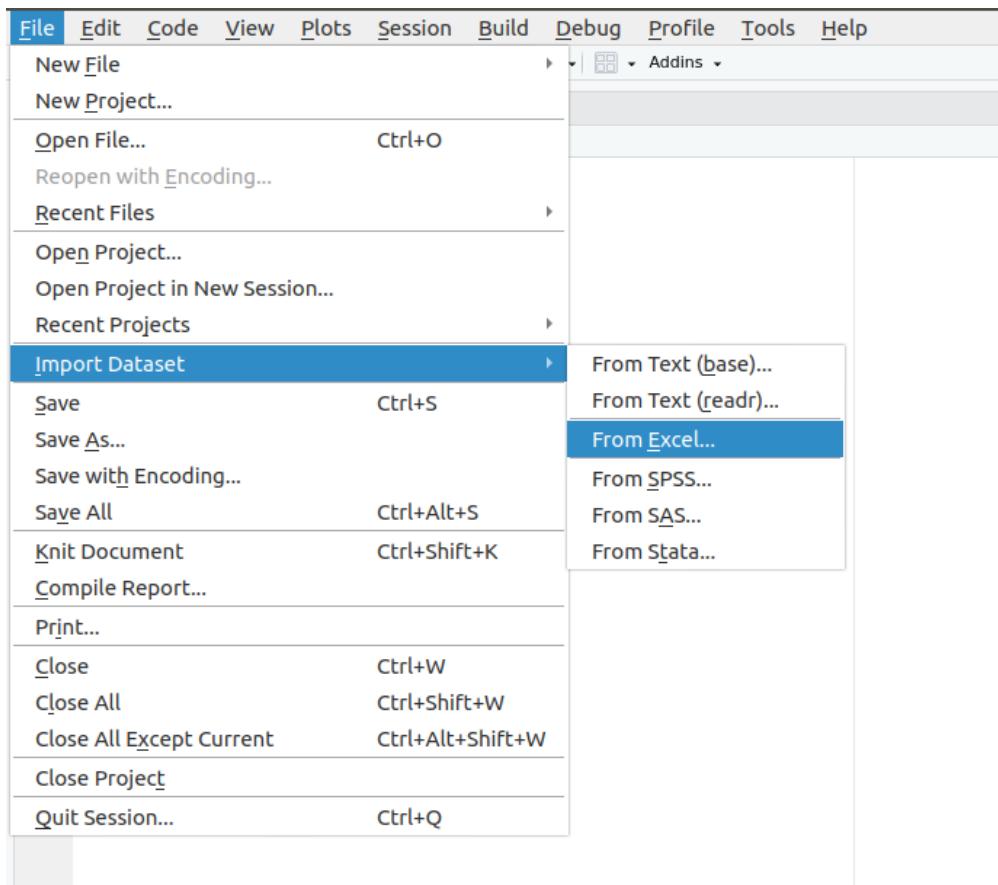


Figura 4.3: Opção do menu do *RStudio* para realizar a importação de dados do *Excel*.

Na caixa de diálogo para importação do arquivo, clicamos no ícone *Browser* (seta verde na figura 4.4), localizamos e selecionamos o arquivo de dados desejado (figura 4.5).

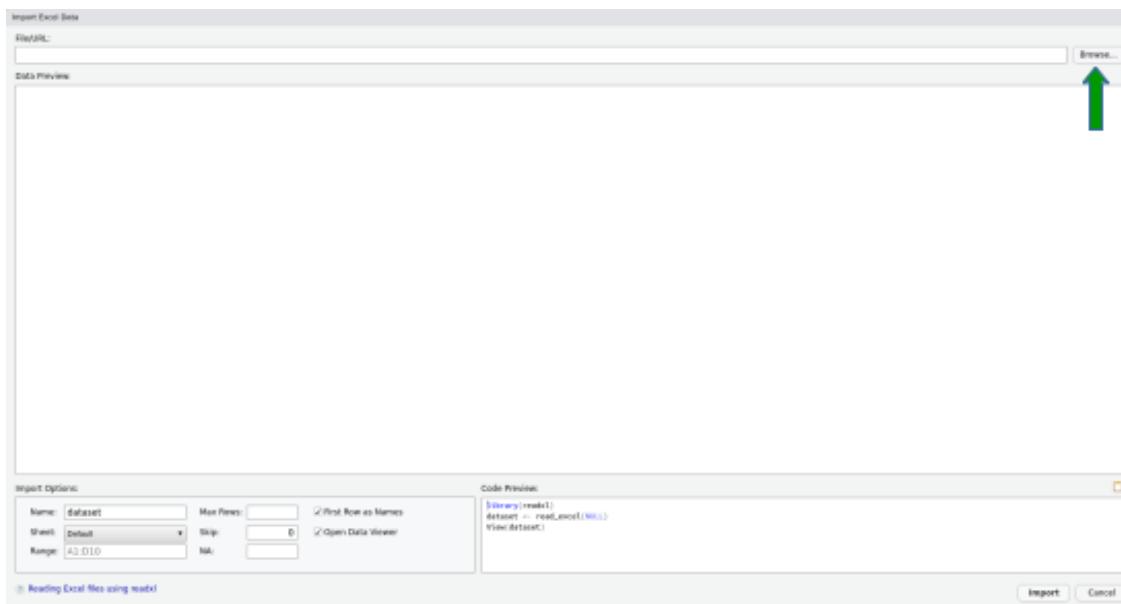


Figura 4.4: Tela do *RStudio* para importar um arquivo de dados do *Excel*. A seta verde indica o botão para navegar no sistema de arquivo e localizar o arquivo a ser importado.

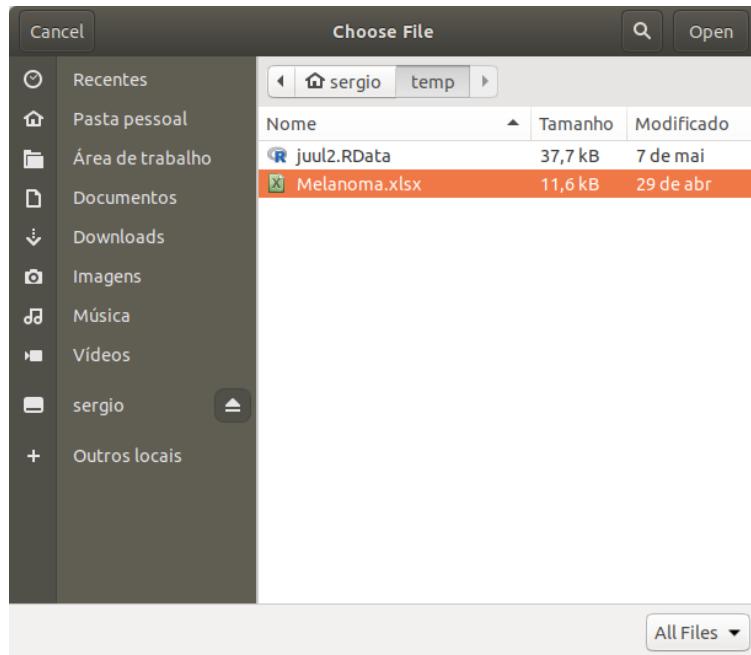


Figura 4.5: Caixa de diálogo para abrir o arquivo a ser importado do *Excel* para o R.

Ao clicarmos em *Open* na figura 4.5, é preciso configurar as opções para a importação do arquivo (figura 4.6). Nessa tela, podemos selecionar que planilha (*sheet*) será importada, o número máximo de registros a serem importados (*Max Rows*), a partir de que registro os dados serão lidos (*Skip*), como valores faltantes (*missing*) são representados na planilha (*NA*), se a primeira linha da planilha contém os nomes das variáveis (*First Row as Names*)

e se os dados serão exibidos no *RStudio* (*Open Data Viewer*). Uma pré-visualização dos dados também é mostrada nessa tela. Se o número máximo de linhas não for especificado, todos os registros serão importados. Se o valor de *Skip* não for especificado, os dados serão importados a partir do primeiro registro. A área de *Code Preview* mostra os comandos que serão executados no *RStudio*.

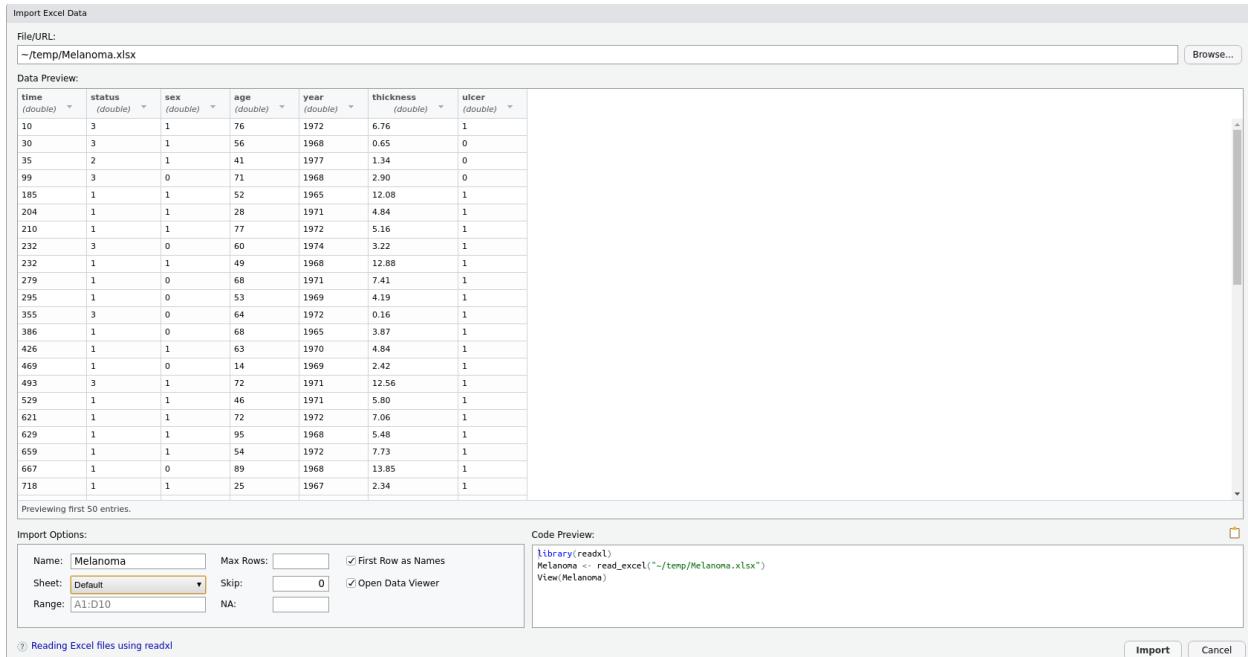


Figura 4.6: Tela do *RStudio* com a configuração para importar o arquivo de dados *Melanoma.xlsx*.

Ao clicarmos no botão *Import*, os dados serão importados para o R (figura 4.7). A tela superior à esquerda abre o visualizador dos dados do arquivo *Melanoma.xlsx*. Na aba *Environment*, a seta verde indica o objeto *Melanoma*, que contém os dados importados. Esse objeto é da classe *data.frame*. Ao clicarmos em *Melanoma*, a estrutura do conjunto de dados é mostrada abaixo.

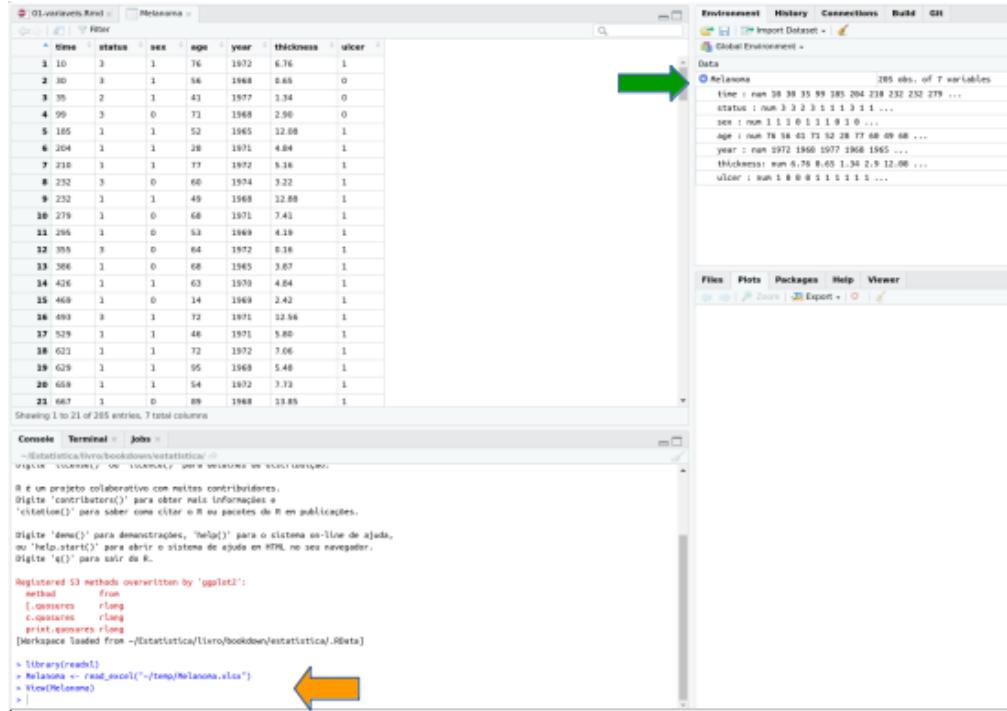


Figura 4.7: Tela do *RStudio* após a importação dos dados do *Excel*.

Na área da console do *RStudio*, são mostrados os comandos executados na importação do arquivo do *Excel* (seta alaranjada).

```
library(readxl)
Melanoma <- read_excel("~/temp/Melanoma.xlsx")
View(Melanoma)
```

A primeira função, *library*, carrega o pacote *readxl*, que contém funções para importar um arquivo do *Excel* para o R. As funções *library* ou *require*, são usadas para carregar pacotes no R. Todo pacote, com exceção daqueles que já são carregados ao executar o R, precisa ser carregado quando desejarmos executar funções ou ler conjuntos de dados contidos nesses pacotes.

O comando seguinte, *Melanoma <- read_excel("~/temp/Melanoma.xlsx")*, lê o arquivo *Melanoma.xlsx* e o carrega no objeto da classe *data.frame*, chamado *Melanoma*. Finalmente a última função, *View*, exibe o conteúdo do objeto *Melanoma* no *RStudio*.

Observem que a função *read_excel()* foi chamada com somente um argumento: o caminho e o nome do arquivo a ser importado. Se solicitarmos a ajuda para essa função (*help("read_excel")*), iremos verificar que diversos outros argumentos poderiam ser especificados para definir como os dados serão importados. Alguns argumentos, como *skip*, *sheet*, *na*, *n_max*, e *range*, poderiam ser configurados na tela de importação de um arquivo do *Excel* no *RStudio*. Quando esses argumentos não forem alterados na tela de importação, eles não precisam ser especificados na linha de comando, porque a função irá considerar os valores padrões (*default*).

Ao especificarmos o nome do arquivo, é necessário especificar todo o caminho até chegar ao arquivo, por exemplo: `file = "c:/temp/Melanoma.xlsx"` (Windows), se o arquivo estiver na pasta temp do disco C.

Observem o uso da barra comum (não invertida). Podemos usar as barras invertidas, desde que elas sejam duplicadas:

```
file = "c:\\temp\\Melanoma.xlsx"
```

Nas distribuições do linux, usariamnos, por exemplo, `file="/home/sergio/temp/Melanoma.xlsx"` ou `file="~/temp/Melanoma.xlsx"`, como foi utilizado no comando acima..

Para visualizarmos os primeiros registros de *Melanoma* via linha de comando, podemos usar a função *head* com dois argumentos, o primeiro é o nome do *data frame*, o segundo indica o número de registros a serem lidos. Se esse número não for especificado, serão mostrados 6 registros:

```
## # A tibble: 10 x 7
##   time status sex   age year thickness ulcer
##   <dbl>  <dbl> <dbl> <dbl> <dbl>      <dbl> <dbl>
## 1     10     3     1    76  1972       6.76     1
## 2     30     3     1    56  1968       0.65     0
## 3     35     2     1    41  1977       1.34     0
## 4     99     3     0    71  1968       2.9      0
## 5    185     1     1    52  1965      12.1     1
## 6    204     1     1    28  1971       4.84     1
## 7    210     1     1    77  1972       5.16     1
## 8    232     3     0    60  1974       3.22     1
## 9    232     1     1    49  1968      12.9     1
## 10   279     1     0    68  1971       7.41     1
```

A função *tail* possui os mesmos argumentos de *head*, mas mostra as últimas n linhas especificadas pelo segundo argumento:

```
## # A tibble: 10 x 7
##   time status sex   age year thickness ulcer
##   <dbl>  <dbl> <dbl> <dbl> <dbl>      <dbl> <dbl>
## 1  4124     2     0    30  1966       1.94     1
## 2  4207     2     1    22  1966       0.65     0
## 3  4310     2     1    55  1966       2.1      0
## 4  4390     2     0    26  1965       1.94     1
## 5  4479     2     0    19  1965       1.13     1
## 6  4492     2     1    29  1965       7.06     1
## 7  4668     2     0    40  1965       6.12     0
## 8  4688     2     0    42  1965       0.48     0
## 9  4926     2     0    50  1964       2.26     0
## 10 5565     2     0    41  1962       2.9      0
```

A função *names(data frame)* retorna os nomes das variáveis no *data frame*. Como os *data frames* são listas (vide capítulo anterior), cada coluna, ou variável, pode ser acessada por meio do sinal “\$”, ou também por meio do número da coluna no *data frame*. Vejam os exemplos abaixo:

```
names(Melanoma)[1:5] # mostra as primeiras 5 variáveis de Melanoma
```

```
## [1] "time"   "status"  "sex"     "age"     "year"
```

```
mean(Melanoma$age) # calcula e exibe a média de idade
```

```
## [1] 52.46341
```

```
max(Melanoma[1]) # calcula o maior valor do tempo de sobrevida
```

```
## [1] 5565
```

4.3 Importando dados em outros formatos no *RStudio*

A figura 4.3 mostra outras opções de importação de dados para o R a partir de outros formatos. A partir do *RStudio*, podemos importar dados diretamente de diversos programas estatísticos.

Arquivos textuais, por exemplo com a extensão *csv*, *txt* e *dat* podem ser importados por meio das opções “*From Text (readr)...*” ou “*From Text (base)...*”. Como virtualmente todos os programas estatísticos exportam dados para o formato *csv*, podemos importar para o R dados de qualquer programa estatístico.

4.4 Importando arquivos pelo *R Commander*

Os conteúdos desta seção e da seguinte podem ser visualizados neste [vídeo](#).

É possível também importar dados para o R a partir do *R Commander*, por meio da opção de menu (figura 4.8):

Dados ⇒ Importar arquivo de dados



Figura 4.8: Opções de importação de dados a partir do *R Commander*.

As opções de importação do *R Commander* incluem aquelas do *RStudio*.

Vamos realizar a importação do arquivo *Melanoma.xlsx* pelo *R Commander*. Usamos a opção do menu mostrada na figura 4.8:

Dados ⇒ Importar arquivo de dados ⇒ do arquivo Excel

As opções de importação de planilhas do *Excel* a partir do *R Commander* são mostradas na figura 4.9. São mostradas menos opções do que no *RStudio*.

É necessário dar um nome para o conjunto de dados. Como a primeira linha do arquivo contém os nomes das variáveis, é preciso marcar a opção *Nome da variável na primeira linha da planilha*.

Como a primeira coluna da planilha não contém os nomes das linhas, essa opção deve ser deixada desmarcada. Como a planilha não possui variáveis do tipo *character*, a opção *Converter caracter para fator* pode ser ou não desmarcada.

Finalmente é preciso especificar o símbolo para os dados ausentes. Nesse exemplo, não há dados ausentes na planilha.

Ao clicarmos em OK, é preciso selecionar o arquivo a ser importado (figura 4.10) e clicar em *Abrir*.

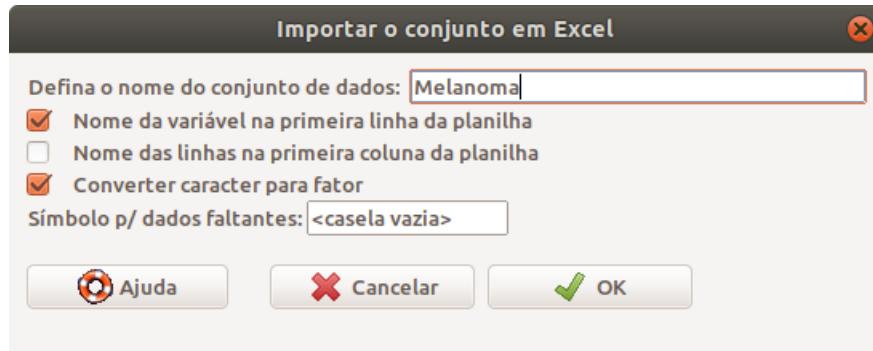


Figura 4.9: Opções de importação de dados do *Excel* a partir do *R Commander*.

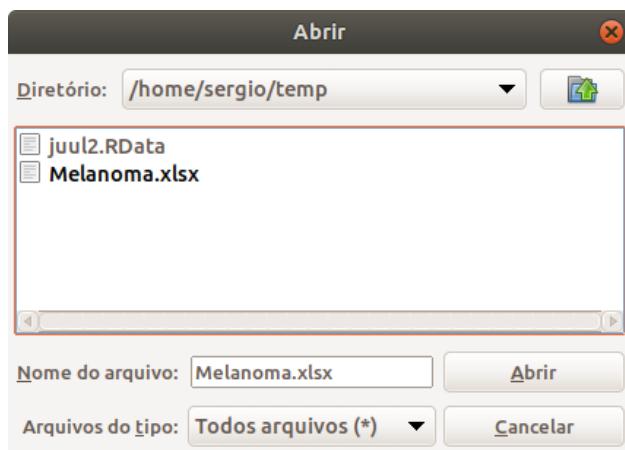


Figura 4.10: Seleção do arquivo do *Excel* a ser importado pelo *R Commander*.

O arquivo é importado e o comando gerado na área de *script* é mostrado abaixo:

```
Melanoma <- readXL("/home/sergio/temp/Melanoma.xlsx", rownames=FALSE,
                     header=TRUE, na="", sheet="Melanoma", stringsAsFactors=TRUE)
```

Observem que o *R Commander* utiliza uma função diferente (*readXL*) da utilizada pelo *RStudio* para importar uma planilha do *Excel*. Pacotes diferentes do R podem possuir funções com objetivos semelhantes.

4.5 Exportando arquivos pelo *R Commander*

Eventualmente pode ser necessário exportar os dados no formato do R para um formato textual para ser aberto em outro programa. Para exportarmos um conjunto de dados aberto no R para o formato de texto (.txt, .csv, .dat, etc), selecionamos a opção do *R Commander*:

Dados ⇒ Conjunto de dados ativo ⇒ Exportar conjunto de dados ativo...

Na figura 4.11, selecionamos as opções de exportação. Nesse exemplo, vamos escrever os nomes das variáveis na primeira, não incluir os nomes das linhas, usar *NA* como valores ausentes e “;” como separador dos campos.



Figura 4.11: Caixa de diálogo configurar a exportação de um conjunto de dados para um arquivo texto.

Ao pressionarmos o botão OK, definimos então o nome e a extensão do arquivo que será gravado (4.12). Ao pressionarmos o botão Salvar, o arquivo será gravado na pasta escolhida.

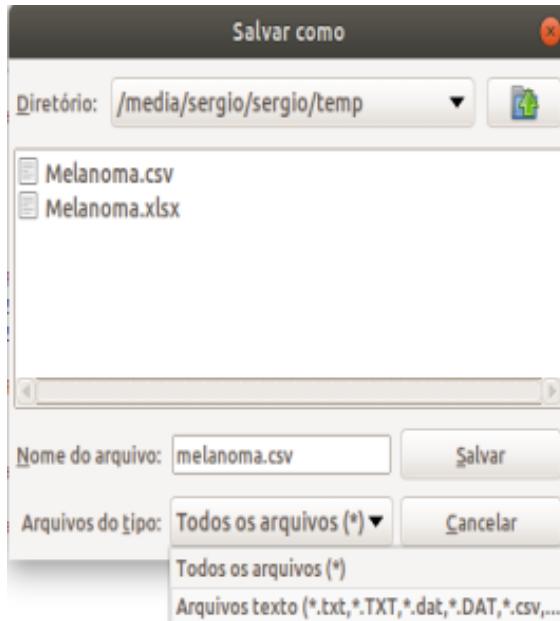


Figura 4.12: Caixa de diálogo para nomear o arquivo e definir a extensão do arquivo texto no qual o conjunto de dados será exportado.

O comando abaixo realizou a exportação do conjunto de dados *Melanoma* para um arquivo csv.

```
write.table(Melanoma, "/media/sergio/sergio/temp/Melanoma.csv",
sep=";", col.names=TRUE, row.names=FALSE, quote=FALSE, na="NA")
```

4.6 Salvando e carregando um arquivo de dados no formato do R

O conteúdo desta seção pode ser visualizado neste [vídeo](#).

Para salvarmos um arquivo de dados no formato utilizado pelo R, vamos utilizar o *R Commander* e salvar o conjunto de dados *Melanoma*, importado do *Excel* anteriormente.

Caso o *R Commander* não tenha sido carregado ainda, digite o comando abaixo na console do *RStudio*:

```
library(Rcmdr) # carregando o R Commander
```

Inicialmente, temos que ativar o conjunto de dados *Melanoma* no *R Commander*. Isso pode ser feito, clicando no botão *Não há conjunto de dados ativo* (seta verde na figura 4.13).



Figura 4.13: Botão para selecionar um conjunto de dados ativo no R Commander.

Na caixa de diálogo *Selecione o conjunto de dados* (figura 4.14), selecionamos *Melanoma* e clique em OK. Agora o conjunto de dados *Melanoma* aparece como rótulo no botão onde antes o rótulo era *Não há conjunto de dados ativo* (seta vermelha na figura 4.15).

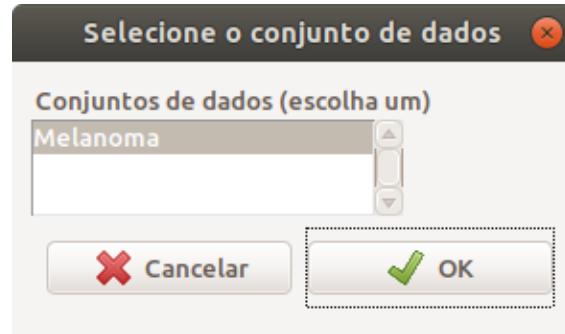


Figura 4.14: Seleção do conjunto de dados a ser ativado no *R Commander*.

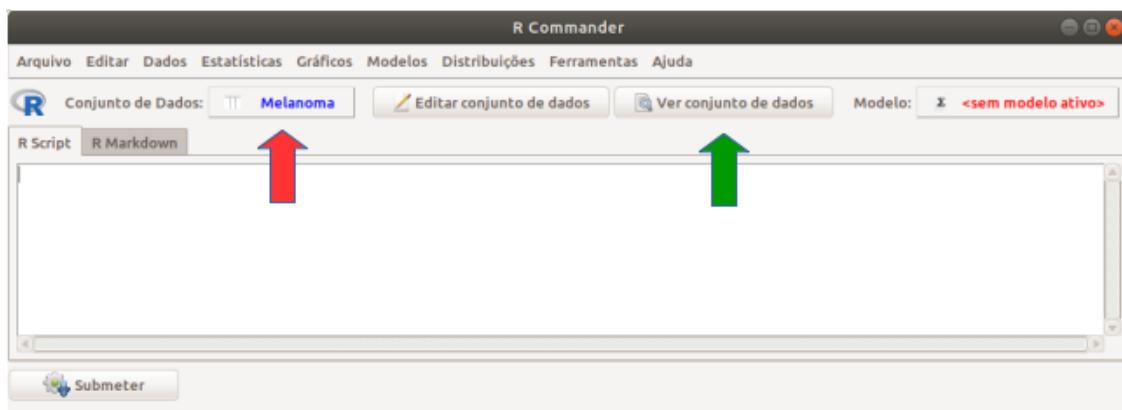


Figura 4.15: Tela do *R Commander* após a seleção do conjunto de dados *Melanoma*. A seta vermelha indica o nome do conjunto de dados. A seta verde mostra o botão para visualizar os dados do *data frame* *Melanoma*.

Para visualizarmos o conjunto de dados no *R Commander*, clicamos no botão *Ver conjunto de dados* na tela do *R Commander* (seta verde na figura 4.15). A figura 4.16 mostra os dados do arquivo *Melanoma*.

| Melanoma | | | | | | | |
|----------|-----------------|--------|-------|-------|-------|-----------|-------|
| | tibble: 205 x 7 | | | | | | |
| 1 | time | status | sex | age | year | thickness | ulcer |
| 2 | dbl | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> |
| 3 | 10 | 3 | 1 | 76 | 1972 | 6.76 | 1 |
| 4 | 30 | 3 | 1 | 56 | 1968 | 0.65 | 0 |
| 5 | 35 | 2 | 1 | 41 | 1977 | 1.34 | 0 |
| 6 | 99 | 3 | 0 | 71 | 1968 | 2.9 | 0 |
| 7 | 185 | 1 | 1 | 52 | 1965 | 12.1 | 1 |
| 8 | 204 | 1 | 1 | 28 | 1971 | 4.84 | 1 |
| 9 | 210 | 1 | 1 | 77 | 1972 | 5.16 | 1 |
| 10 | 232 | 3 | 0 | 60 | 1974 | 3.22 | 1 |

Figura 4.16: Dados do arquivo *Melanoma* importados para o R.

Para salvarmos o conjunto de dados ativo em um arquivo que possa ser lido diretamente pelo R, selecionamos a opção:

Dados ⇒ Conjunto de dados ativo ⇒ Salvar conjunto dados ativo...

Na caixa de diálogo *Salvar como* (figura 4.17), navegamos para a pasta onde desejamos salvar o arquivo e especificamos um nome para ele, de preferência com a extensão *.RData*. Clicamos em *Salvar*.

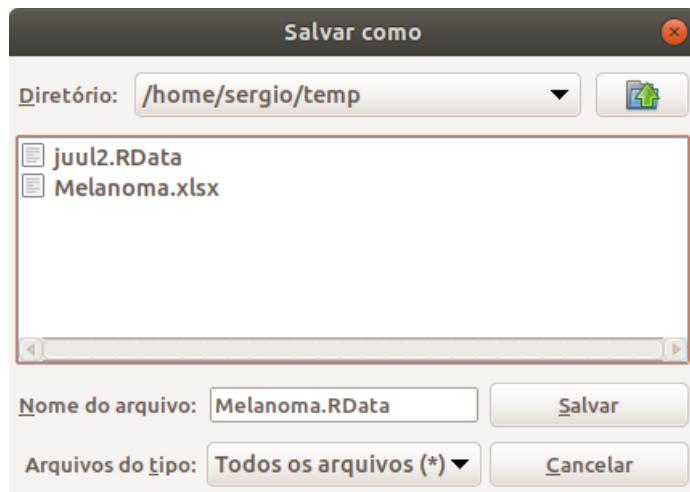


Figura 4.17: Especificação do nome do arquivo a ser gravado.

O comando executado na área de Script do *R Commander* foi:

```
save("Melanoma", file="/home/sergio/temp/Melanoma.RData")
```

Para salvarmos um *data frame* em um arquivo que possa ser lido pelo R posteriormente, sem necessidade de importação, usamos a função *save*.

Se você solicitar a ajuda para essa função, verá os seus argumentos. Na forma mais simples, temos que especificar o nome do *data frame* a ser salvo e o nome do arquivo que será gravado no disco.

O nome do arquivo poderia ser qualquer nome válido. Se o arquivo estiver em outra pasta que não a pasta de trabalho corrente, então devemos especificar o caminho para o arquivo no argumento *file*. A extensão *RData* é frequentemente usada para representar arquivos que contêm objetos que podem ser carregados no R.

Vamos supor que o arquivo tenha sido gravado com o nome *Melanoma.RData*. Em uma outra sessão do R, para carregarmos o arquivo no *R Commander* para ser analisado, usamos a opção:

Dados ⇒ Carregar conjunto de dados

Na caixa de diálogo *Abrir*, navegamos para a pasta onde o arquivo esteja localizado, o selecionamos e clicamos no botão *Abrir* (figura 4.18).

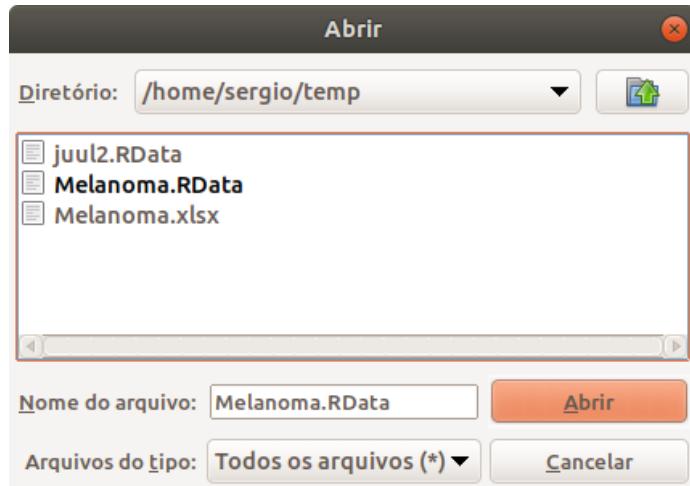


Figura 4.18: Carregando um arquivo de dados no formato do R.

Para carregarmos o conjunto de dados *Melanoma.RData*, é necessário usar a função *load(nome_do_arquivo)*, sem esquecermos de especificar o caminho do arquivo na árvore de diretórios, se o arquivo não estiver no diretório de trabalho corrente:

```
load("/home/sergio/temp/Melanoma.RData")
```

O *data frame* é carregado no objeto com o mesmo nome com que havia sido salvo.

4.7 Carregando conjuntos de dados disponíveis em pacotes do R

O conteúdo desta seção pode ser visualizado neste [vídeo](#).

Muitos pacotes do R contêm conjuntos de dados que podem ser utilizados para ilustrar os recursos disponíveis no pacote. O conjunto de dados *Melanoma*, que utilizamos nas seções anteriores, também está disponível no pacote *MASS* do R e poderia ser carregado a partir desse pacote.

Vamos carregar um outro conjunto de dados já disponível em um pacote do R. Trata-se do conjunto de dados *juul2* do pacote *ISwR*.

Se o leitor acompanhou o tour do R (capítulo 2) e instalou o pacote *ISwR* no início daquele capítulo, não será necessário realizar as instruções de instalação do pacote a seguir. Os passos para a instalação do pacote *ISwR* são os mesmos utilizados para a instalação do

R Commander, seção 1.3 ou seção 1.5. Alternativamente podemos digitar diretamente o comando a seguir na console do *RStudio* e pressionar a tecla Enter. Como o nome indica, a função *install.packages* instala o pacote especificado entre aspas.

```
install.packages("ISwR")
```

Antes de abrirmos o conjunto de dados *juul2*, é preciso carregar o pacote *ISwR* (GPL-2 | GPL-3). Para isso, podemos digitar *library(ISwR)* na console do *RStudio* e pressionar a tecla Enter, mas vamos fazer isso no *R Commander*. Na área de *script* do *R Commander*, digitamos *library(ISwR)* e, **com o cursor na linha do comando**, clicamos no botão *Submeter* (figura 4.19).



Figura 4.19: Tela do *R commander*, com a digitação da função *library(ISwR)* na área de Script.

Ao submetermos a função, ela aparece na console do *RStudio* (figura 4.20) e, se houver alguma coisa errada, uma mensagem de erro aparecerá abaixo da função na console do *RStudio*.

```
Console Terminal × Jobs ×
~/Estatistica/livro/bookdown/R_Rcmdr_RStudio/ ↵
Versão do Rcmdr 2.5-1

Attaching package: 'Rcmdr'

The following object is masked from 'package:car':
Confint

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

> library(ISwR)

Rcmdr> library(ISwR)
> |
```

Figura 4.20: Console do *RStudio* após a execução da função *library(ISwR)* conforme mostrado na figura 7.3.

Alternativamente, o pacote *ISwR* poderia ser carregado por meio da opção de menu do *R Commander*:

Ferramentas ⇒ Carregar pacote(s)...

A função *library(ISwR)* carregou a biblioteca *ISwR*, a qual contém uma série de conjuntos de dados que podemos utilizar. Para visualizarmos e, eventualmente, selecionarmos um desses conjuntos de dados, selecionamos a opção abaixo no *R Commander*:

Dados ⇒ Conjunto de dados em pacotes ⇒ Ler conjunto de dados de pacotes 'atachados'

Na tela *Leia dados do pacote*, observem que alguns pacotes de dados aparecem na área à esquerda da figura 4.21: *carData*, *datasets*, *ISwR* e *sandwich*. Para vermos a lista dos conjuntos de dados em *ISwR*, damos um duplo clique nesse pacote e uma lista de conjuntos de dados será mostrada à direita. Rolamos essa lista e selecionamos o conjunto *juul2*. Para sabermos a que esse conjunto de dados se refere, clicamos no botão *Ajuda para o conjunto de dados selecionado* (seta verde na figura). Uma descrição desse conjunto de dados será exibida na aba *Help* do *RStudio* (figura 4.22). Ao clicarmos no botão *OK* na figura 4.21, após termos selecionado *juul2*, esse conjunto de dados será carregado no *R Commander* (figura 4.23).



Figura 4.21: Visualizando a lista de conjuntos de dados do pacote *ISwR* e solicitando a ajuda para o conjunto *juul2* (seta verde).

The screenshot shows a web-based R help browser interface. At the top, there is a navigation bar with tabs: Files, Plots, Packages, Help, and Viewer. Below the navigation bar, there are icons for back, forward, and search, along with a home icon. The main content area displays the following information:

R: Juul's IGF data, extended version • Find in Topic

juul2 {ISwR}

Juul's IGF data, extended version

Description

The `juul2` data frame has 1339 rows and 8 columns; extended version of `[juul]`.

Usage

```
juul2
```

Format

This data frame contains the following columns:

- age**
a numeric vector (years).
- height**
a numeric vector (cm).
- menarche**
a numeric vector. Has menarche occurred (code 1: no, 2: yes)?
- sex**
a numeric vector (1: boy, 2: girl).
- igf1**
a numeric vector, insulin-like growth factor (*microgram per liter*).
- tanner**
a numeric vector, codes 1-5: Stages of puberty ad modum Tanner.
- testvol**
a numeric vector, testicular volume (ml).
- weight**
a numeric vector, weight (kg).

Source

Original data.

Figura 4.22: Texto com a descrição do conjunto de dados `juul2` exibido no navegador de seu computador.

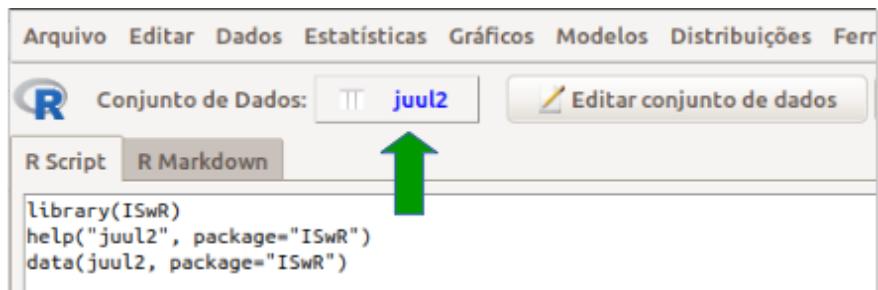


Figura 4.23: Tela do *R commander* após o carregamento do conjunto de dados *juul2*. Observem a função que foi executada – *data(juul2, package = "ISwR")* – e o nome do conjunto selecionado (seta verde).

Observem as funções que foram executadas no *R Commander* e também são exibidas na console do *RStudio*:

```
help("juul2", package="ISwR")
data(juul2, package="ISwR")
```

A função *help*, como visto no capítulo anterior, mostra uma ajuda sobre o conjunto de dados *juul2* do pacote *ISwR*.

A função *data(juul2, package = "ISwR")* carrega o conjunto de dados *juul2* que passa a ser o conjunto de dados ativo no *R Commander*. Observem o nome dele ao lado do rótulo conjunto de dados (seta verde na figura 4.23). Esse objeto pode ser acessado pelo próprio nome (*juul2* neste caso).

Na console do *RStudio*, aparece a seguinte mensagem abaixo do comando, indicando o número de registros e de variáveis no conjunto de dados *juul2*:

```
RcmdrMsg: [2] NOTA: Os dados juul2 tem 1339 linhas e 8 colunas.
```

4.8 Exercícios

- 1) Vamos trabalhar com o conjunto de dados denominado *stroke* do pacote *ISwR* (GPL-2 | GPL-3). No *R Commander*, faça as atividades seguintes.
 - a) Verifique a ajuda para o conjunto de dados.
 - b) Carregue o conjunto de dados.
 - c) Visualize o seu conteúdo.
 - d) Grave os dados em um arquivo do R.
 - e) Carregue no *R Commander* o arquivo que foi gravado no item d.
- 2) Repita o exercício anterior com o conjunto de dados *VA* do pacote *MASS* (GPL-2 | GPL-3).
- 3) Importe o arquivo *Melanoma* para o R. Visualize os dados e salve o conjunto de dados em um arquivo no formato do R.

Capítulo 5

Utilização do *R Commander*

Neste capítulo, vamos explorar um pouco mais a interface do *R Commander*, a saber:

- obtenção de resumos numéricos das variáveis do conjunto de dados carregado;
- execução de comandos ou funções na área de *script*;
- utilização do *R Markdown*;
- gravação dos *scripts*, markdown e resultados.

5.1 Obtendo resumos numéricos

Neste capítulo, vamos utilizar o conjunto de dados *juul2* do pacote [ISwR \(GPL-2 | GPL-3\)](#).

Para carregarmos esse conjunto de dados, veja a seção 4.7.

O conjunto de dados *juul2* possui 1339 registros, cada registro com dados de 8 variáveis. Ele contém uma amostra da distribuição da variável *insulin-like growth factor (igf1)*, com os dados coletados em exames físicos, sendo a maior parte dos dados de pessoas em idade escolar, mas também inclui outras faixas etárias. Vamos obter algumas medidas de tendência central e dispersão para as variáveis *age* e *igf1*.

Na barra de menus, selecionamos a opção:

Estatísticas ⇒ Resumos ⇒ Resumos numéricos...

Na tela *Resumos Numéricos*, selecionamos as variáveis na aba *Dados*. Para selecionarmos mais de uma variável, mantemos a tecl Ctrl pressionada enquanto clicamos nas variáveis desejadas. Nesse exemplo, vamos selecionar as variáveis *age* e *igf1* (Figura 5.1). Em seguida, clicamos na aba *Estatística* (seta verde na figura).



Figura 5.1: Seleção das variáveis para as quais um resumo numérico será mostrado. A seta verde indica a aba onde podem ser selecionadas as medidas que serão apresentadas.

Na aba *Estatísticas* (figura 5.2), observem que as medidas média, desvio padrão, distância interquartil e quantis já estão marcadas. Se desejarmos outros percentis, basta digitá-los na caixa de texto com o rótulo *Quantis*, separados por vírgula. Ao clicarmos em OK, os resultados serão apresentados na tela principal do *R Commander* (figura 5.3).



Figura 5.2: Tela para a seleção das medidas que serão apresentadas no resumo numérico.

| | mean | sd | IQR | 0% | 25% | 50% | 75% | 100% | n | NA |
|------|-----------|-----------|----------|-------|----------|--------|---------|------|------|-----|
| age | 15.09535 | 11.25288 | 7.8025 | 0.17 | 9.0525 | 12.56 | 16.855 | 83 | 1334 | 5 |
| igf1 | 340.16798 | 171.03560 | 260.5000 | 25.00 | 202.2500 | 313.50 | 462.750 | 915 | 1018 | 321 |

Figura 5.3: Estatísticas descritivas para as variáveis *age* e *igf1* no conjunto de dados *juul2*.

Observem a função que foi executada, *numSummary()*:

```
numSummary(juul2[, c("age", "igf1"), drop=FALSE],
           statistics=c("mean", "sd", "IQR", "quantiles"),
           quantiles=c(0, .25, .5, .75, 1))
```

As estatísticas descritivas são calculadas corretamente, mesmo considerando que, em alguns registros, alguns dados não foram preenchidos (*NA*). Os resultados indicam que 5 registros não possuem valores para a idade e 321 registros não possuem valores de *igf1*.

Existem funções para obter cada uma das estatísticas descritivas individualmente, mas alguns cuidados precisam ser observados. Por exemplo, digitando a função *mean(juul2\$age)* na área de *Script* do *R Commander* (ou na console do *RStudio*) e clicando no botão *Submeter*, o resultado será *NA*. Isso ocorreu devido aos 5 registros sem valores de idade.

```
mean(juul2$age) # o cálculo irá resultar em NA, pois há NA nos dados
## [1] NA
```

Para calcularmos a média, não incluindo os registros com valores ausentes, usamos o argumento *na.rm=TRUE*, o qual indica a remoção do cálculo os registros com valores ausentes. Assim, para obtermos a média de idade, usamos a função *mean(juul2\$age, na.rm=TRUE)*. Agora a média será obtida corretamente:

```
mean(juul2$age, na.rm = TRUE) # cálculo da média de idade, removendo NA
## [1] 15.09535
```

Observação:

1) como o *juul2* é carregado como um *data.frame*, as variáveis podem ser acessadas por meio do operador \$.

As funções *median*, *sd*, *IQR*, *min* e *max* calculam respectivamente, a mediana, o desvio padrão, a distância interquartil, o mínimo e o máximo da variável especificada como argumento.

```
median(juul2$age, na.rm = TRUE)
```

```
## [1] 12.56
```

```
sd(juul2$age, na.rm = TRUE)
```

```
## [1] 11.25288
```

```
IQR(juul2$age, na.rm = TRUE)
```

```
## [1] 7.8025
```

```
min(juul2$age, na.rm = TRUE)
```

```
## [1] 0.17
```

```
max(juul2$age, na.rm = TRUE)
```

```
## [1] 83
```

A função *quantile()* gera os quantis de uma distribuição (decis, quartis, tercis, percentis). Por exemplo, para obtermos os percentis 25 e 75 (1º e 3º quartis) da variável idade, utilizamos a função *quantile* conforme a seguir:

```
quantile(juul2$age, probs = c(0.25, 0.75), na.rm = TRUE)
```

```
##      25%      75%
## 9.0525 16.8550
```

5.2 R Markdown

O *R Markdown* é uma linguagem que permite que um relatório possa ser gerado a partir dos comandos que vão sendo executados no R. No *R Commander*, ele pode ser visualizado na aba *R Markdown* (seta verde na figura 5.4).

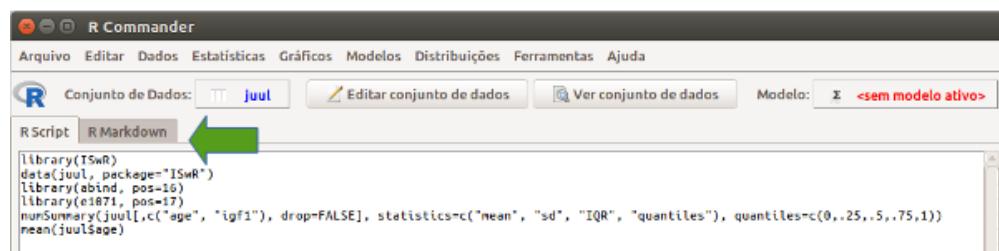


Figura 5.4: Acessando o *R Markdown* no *R Commander*.

Esse relatório pode ser personalizado pelo usuário. Por exemplo, no texto da figura 5.5, alteramos o título e o autor (seta verde na figura, depois selecionamos o comando *help...* (figura 5.6) e o apagamos (figura 5.7) para não exibir a ajuda do conjunto de dados em uma outra página web quando for o relatório for gerado. Ao clicarmos no botão *Gerar relatório*, o relatório será apresentado no navegador padrão de seu computador (figura 5.8)



```

<!-- R Commander Markdown Template -->
Exemplo de Relatório
=====
### Autor do Relatório
## `r as.character(Sys.Date())`

```{r echo=FALSE}
include this code chunk as-is to set options
knitr::opts_chunk$set(comment=NA, prompt=TRUE, out.width=750, fig.height=8, fig.width=8)
library(Rcmdr)
library(car)
```

```

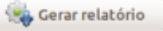


Figura 5.5: Personalizando o título e o autor do relatório no *R Markdown*.



```

...
```{r}
help("juul", package="ISwR")
```

```{r}
data(juul, package="ISwR")
```

```

Figura 5.6: Selecionando partes do relatório para edição.



```

...
```{r}
data(juul, package="ISwR")
```

```{r}
library(abind, pos=16)
```

```

Figura 5.7: Remoção da área selecionada na figura 5.6.

Exemplo de Relatório

Autor do Relatório

2019-07-15

```
> library(ISwR)

> data(juul2, package="ISwR")

> numSummary(juul2[,c("age", "igf1"), drop=FALSE], statistics=c("mean", "sd",
+   "IQR", "quantiles"), quantiles=c(0,.25,.5,.75,1))
```

| | mean | sd | IQR | 0% | 25% | 50% | 75% | 100% | n |
|------|-----------|-----------|----------|-------|----------|--------|---------|------|------|
| age | 15.09535 | 11.25288 | 7.8025 | 0.17 | 9.0525 | 12.56 | 16.855 | 83 | 1334 |
| igf1 | 340.16798 | 171.03560 | 260.5000 | 25.00 | 202.2500 | 313.50 | 462.750 | 915 | 1018 |
| | NA | | | | | | | | |
| age | 5 | | | | | | | | |
| igf1 | 321 | | | | | | | | |

```
> mean(juul2$age)
```

```
[1] NA
```

```
> mean(juul2$age, na.rm = TRUE)
```

```
[1] 15.09535
```

Figura 5.8: Relatório gerado pelo *R Markdown* em html para os comandos utilizados nesta seção.

5.3 Salvando *scripts* e arquivos do *R Markdown*

Todos os comandos utilizados numa seção do *R Commander* podem ser salvos em um arquivo. É possível também editar os comandos, inclusive removê-los, na janela de *script* antes de salvá-lo. Em outra seção do *R Commander* ou *RStudio*, o arquivo salvo pode ser reaberto e os comandos executados, sem necessidade de executar cada um deles individualmente.

Para salvarmos o *script* no *R Commander*, selecionamos a seguinte opção:

Arquivo ⇒ Salvar script como...

Na janela *Salvar como* (Figura 5.9), selecionamos a pasta onde o arquivo será gravado e digitamos o nome do mesmo na caixa de texto *Nome do Arquivo*. Vamos manter a extensão

do arquivo (.R). Em seguida, clicamos no botão *Salvar* e o arquivo será gravado na pasta desejada.

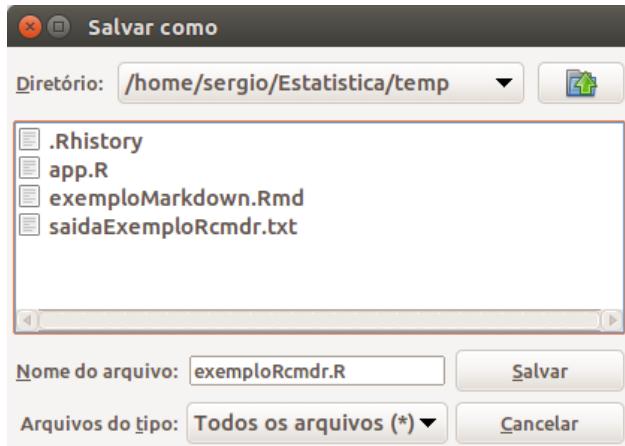


Figura 5.9: Selecionando a pasta e digitando o nome do *script* a ser gravado.

Para salvarmos o arquivo gerado na aba *R Markdown*, selecionamos a seguinte opção.

Arquivo ⇒ Salvar arquivo R Markdown como...

Na janela *Salvar como* (Figura 5.10), selecionamos a pasta onde o arquivo será gravado e digitamos o nome do mesmo na caixa de texto *Nome do Arquivo*. Vamos manter a extensão do arquivo (.Rmd). Em seguida, clicamos no botão *Salvar* e o arquivo será gravado na pasta desejada.

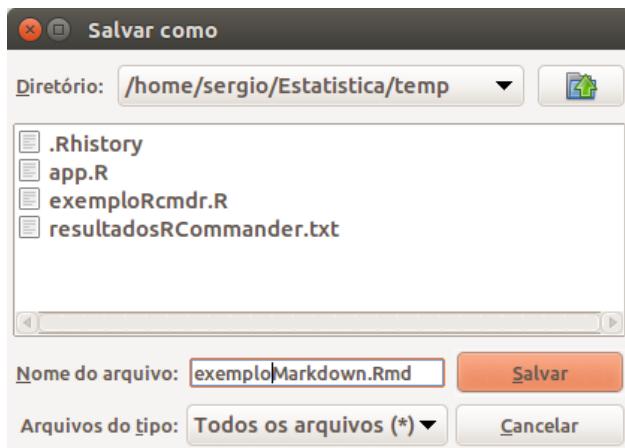


Figura 5.10: Selecionando a pasta e digitando o nome do arquivo do relatório a ser gravado.

5.4 Executando *scripts* no *R Commander*

Se, na janela do *R Script*, selecionarmos um conjunto de comandos ao mesmo tempo e clicarmos no botão *Submeter*, os comandos serão executados em sequência de maneira automática (figura 5.11).

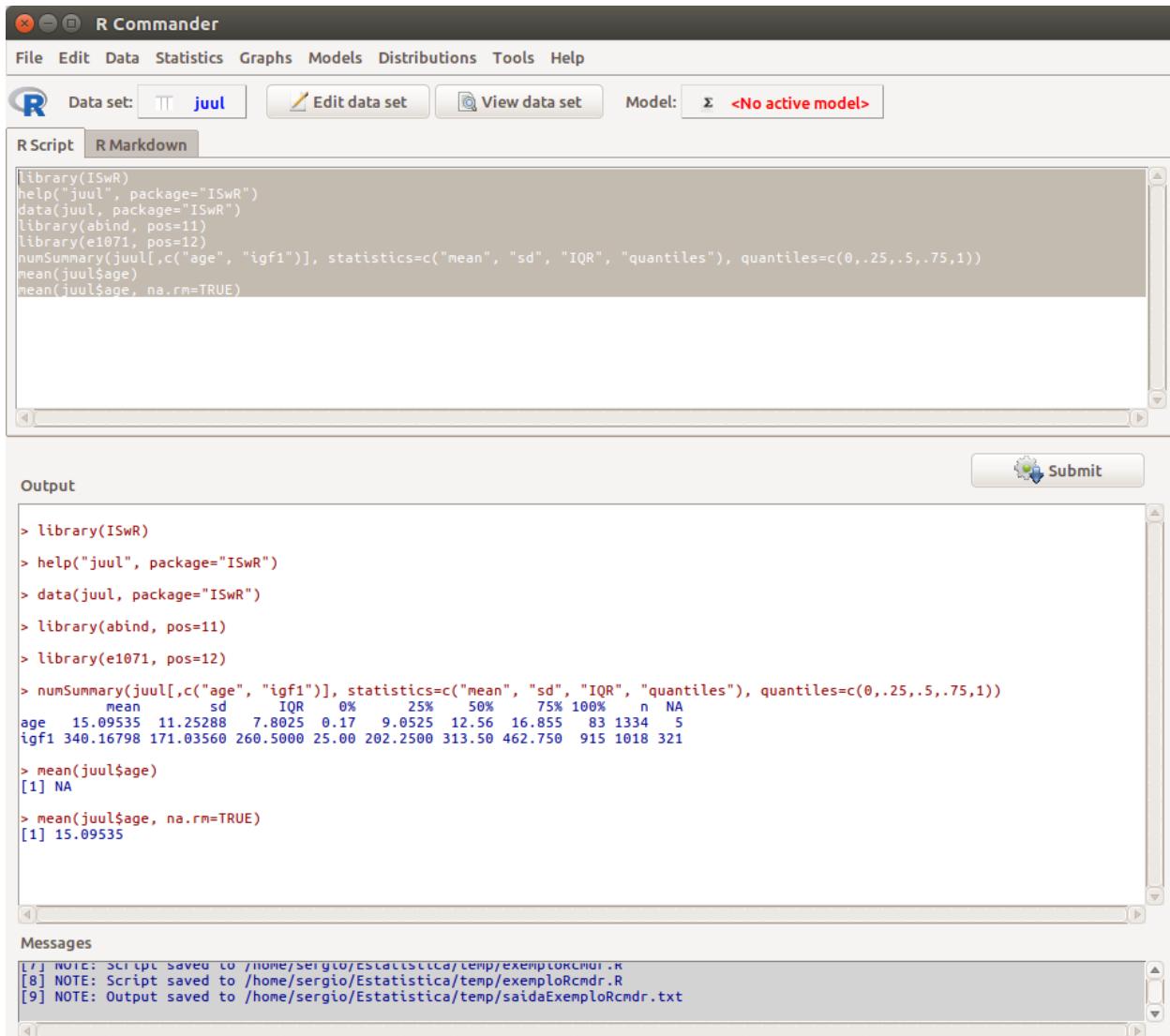


Figura 5.11: Execução automática e em sequência de um conjunto de comandos selecionados na aba *R Script*.

5.5 Exercício

- 1) Refaça os exercícios 1, 2 e 3 do capítulo anterior (Importando e Exportando Dados). Salve o *script* gerado no *R Commander*. No *R Markdown*, coloque um título e o seu nome e gere o relatório.

Capítulo 6

Manipulação dos dados

Neste capítulo, vamos utilizar o *R Commander* e quatro conjuntos de dados: *juul2*, *stroke* e *heart.rate* do pacote *ISwR* (GPL-2 | GPL-3) e *Melanoma* do pacote *MASS* (GPL-2 | GPL-3). Diversas funções estão disponíveis para manipular as variáveis de um conjunto de dados, bem como o próprio conjunto de dados.

Inicialmente, vamos carregar os pacotes *ISwR* e *MASS* e os conjuntos de dados *Melanoma* e *juul2*, nessa ordem.

```
library(ISwR)
library(MASS)
data(Melanoma, package="MASS")
data(juul2, package="ISwR")
```

A função abaixo exibe os 10 primeiros registros do conjunto de dados *juul2*.

```
head(juul2, 10)
```

```
##      age height menarche sex igf1 tanner testvol weight
## 1     NA      NA       NA  NA   90     NA      NA      NA
## 2     NA      NA       NA  NA   88     NA      NA      NA
## 3     NA      NA       NA  NA  164     NA      NA      NA
## 4     NA      NA       NA  NA  166     NA      NA      NA
## 5     NA      NA       NA  NA  131     NA      NA      NA
## 6  0.17      NA       NA   1  101      1     NA      NA
## 7  0.17      NA       NA   1   97      1     NA      NA
## 8  0.17      NA       NA   1  106      1     NA      NA
## 9  0.17      NA       NA   1  111      1     NA      NA
## 10 0.17      NA       NA   1    79      1     NA      NA
```

É possível observar que algumas variáveis são categóricas, mas as categorias são expressas como números e seriam tratadas pelo R como numéricas. Consultando a descrição das variáveis do arquivo *juul2* (figura 4.22), as categorias das variáveis *sex*, *menarche* e *tanner*

têm o significado dado abaixo:

sex:

1 – masculino, 2 – feminino

menarche:

1 – sim, 2 – não

tanner:

estágios de puberdade, codificados de 1 a 5, seguindo a escala de Tanner.

A escala de Tanner (também conhecida como estágios de Tanner) é uma escala do desenvolvimento físico de crianças, adolescente e adultos. A escala define medidas físicas de desenvolvimento baseadas em características externas primárias e secundárias, tais como tamanho da mama, genitais, volume testicular e desenvolvimento de pelos púbicos. A escala foi identificada primeiramente por James Tanner, um pediatra britânico, daí o seu nome ([wikipedia](#)).

6.1 Manipulação de variáveis

Com o conjunto de dados *juul2* ativo (ele deve aparecer ao lado do rótulo *Conjunto de dados ativo* abaixo da barra de menus do *R Commander*), o submenu com as opções de manipulação de variáveis pode ser acessado da seguinte forma:

Dados ⇒ Modificação de variáveis no conjunto de dados...

A figura 6.1 mostra a lista de opções para a manipulação de variáveis no *R Commander*. A opção *Definir contrastes para um fator* não será discutida neste texto. As demais serão apresentadas a seguir.

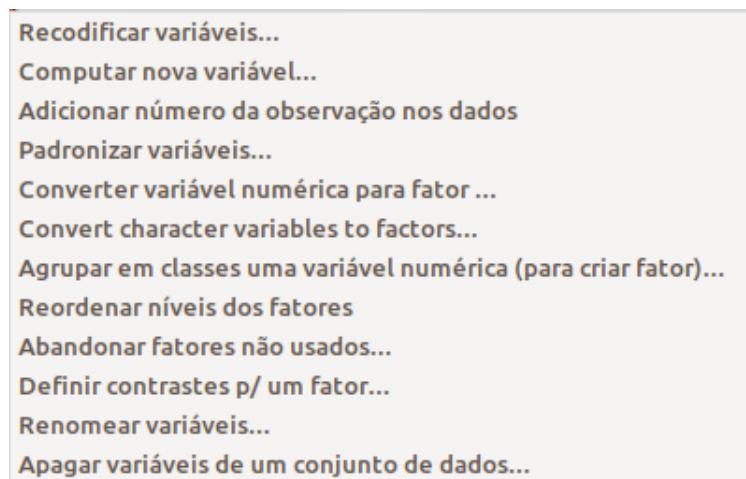


Figura 6.1: Recursos disponíveis para a manipulação de variáveis no conjunto de dados ativo.

6.1.1 Recodificar variáveis

O conteúdo desta seção pode ser visualizado neste [vídeo](#).

A operação de recodificação é utilizada para alterar ou agrupar os códigos de uma variável categórica ou numérica inteira. Na mesma operação, podemos definir a variável que recebe a recodificação como da classe *factor*.

Vamos ilustrar essa operação com dois exemplos.

No primeiro exemplo, vamos recodificar a variável *sex* de *juul2*, substituindo o valor 1 por “masculino” e o valor 2 por “feminino”.

A operação de recodificação de variáveis é acessada no *R Commander* da seguinte forma:

Dados ⇒ Modificação de variáveis no conjunto de dados... ⇒ Recodificar variáveis...

A figura 6.2 mostra a caixa de diálogo do *R Commander* para recodificarmos uma ou mais variáveis. Para especificarmos a recodificação da variável *sex*, selecionamos a variável *sex* e escrevemos o nome da variável que será criada após a recodificação no campo *Novo nome de variável para recodificação múltipla*. Nesse exemplo, colocamos o nome *sexo_cat*. Caso usássemos o mesmo nome da variável que será recodificada, os valores da variável *sex* seriam substituídos pelos valores recodificados.



Figura 6.2: Caixa de diálogo para especificarmos a recodificação de uma variável.

Na caixa de texto *Definições p/recodificação*, escrevemos em cada linha as recodificações. Por exemplo, a primeira linha na figura especifica que o valor 1 será substituído por masculino, a segunda linha especifica que o valor 2 será substituído por feminino. Se marcarmos a opção *Faça de cada nova variável um fator*, a nova variável será da classe *factor*. Em muitas situações, isso é o desejado. Ao pressionarmos o botão OK, o comando abaixo é executado:

```
juul2 <- within(juul2, {
  sexo_cat <- Recode(sex, '1 = "masculino"; 2 = "feminino"', as.factor=TRUE)
})
```

Na expressão de recodificação na figura 6.2, a palavra *masculino* está entre aspas, porque valores de uma variável que são caracteres devem ser expressos entre aspas. Já o valor que será recodificado (1) não aparece entre aspas, porque é um número, não um caracter.

A variável *sexo_cat* é criada a partir da recodificação da variável *sex* e é incorporada ao conjunto de dados *juul2* como fator. Observem os registros do conjunto de dados após a recodificação e a classe da variável *sexo_cat*.

```
head(juul2)
```

```
##   age height menarche sex igf1 tanner testvol weight  sexo_cat
## 1   NA      NA       NA  NA    90     NA      NA      NA    <NA>
## 2   NA      NA       NA  NA    88     NA      NA      NA    <NA>
## 3   NA      NA       NA  NA   164     NA      NA      NA    <NA>
## 4   NA      NA       NA  NA   166     NA      NA      NA    <NA>
## 5   NA      NA       NA  NA   131     NA      NA      NA    <NA>
## 6 0.17      NA       NA   1   101      1      NA      NA masculino
```

```
class(juul2$sexo_cat)
```

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

Vamos entender como essa recodificação foi efetuada. A função utilizada para recodificação é chamada *Recode*, disponível do pacote *car*. O primeiro argumento é a variável que será recodificada. O segundo argumento é uma string que especifica as recodificações que serão realizadas, separadas por “;”. O terceiro argumento informa se a variável a ser criada será convertida para fator ou não.

A função *Recode* vai retornar um vetor com todos os valores da variável *sex* recodificados. Esse vetor será armazenado na variável *sexo_cat*. A função *within* nesse exemplo possui dois argumentos: o conjunto de dados *juul2* e a expressão que será executada no primeiro argumento. Essa expressão aparece entre {}. Assim a função *Recode* vai operar sobre variáveis do conjunto de dados *juul2* e gerar um outro *data frame* com a nova variável recodificada.

Finalmente o *data frame* *juul2* será substituído pelo *data frame* criado pela função *within*:

```
juul2 <- within(...)
```

A operação de recodificação acima poderia também ser realizada da seguinte forma:

```
juul2$sexo_cat <- Recode(juul2$sex, '1 = "masculino"; 2 = "feminino"' ,  
                           as.factor=TRUE)
```

Lembremos que, para acessar os valores de uma variável de um *data frame*, precisamos de usar o `$` separando o *data frame* do nome da variável. A vantagem da função `within` é que podemos referenciar as variáveis diretamente pelo nome, porque o *data frame* que contém as variáveis já foi especificado no primeiro argumento.

Para ilustrar um outro exemplo de recodificação, vamos supor que desejemos criar uma outra variável que irá agrupar as categorias I, II e III de Tanner em uma única categoria e as categorias IV e V numa segunda categoria. A figura 6.3 mostra como configurar essa recodificação por meio do *R Commander*. Observem as diversas maneiras de especificarmos os valores a serem recodificados.



Figura 6.3: Caixa de diálogo para especificarmos uma possível recodificação da variável *tanner*.

O comando a seguir realiza essa recodificação:

```
juul2 <- within(juul2, {  
  tanner_bin <- Recode(tanner, '1:3 = "Tanner I-III";c(4,5) = "Tanner IV-V"' ,  
                        as.factor=TRUE)  
})
```

A função *Recode* permite a recodificação de variáveis das classes *integer*, *numeric*, *character*, *factor* e *logical*. Na recodificação, a nova variável pode ser criada como *factor* ou não.

A seção seguinte mostra uma opção no menu do *R Commander* que pode ser utilizada para converter uma variável da classe *numeric* para *factor*.

6.1.2 Converter variável numérica para fator

O conteúdo desta seção pode ser visualizado neste [vídeo](#).

Para analisar e visualizar as variáveis categóricas corretamente no R, temos que transformá-las em fatores, outro nome utilizado em análises estatísticas para variáveis categóricas. Para realizarmos a conversão de uma variável em fator no *R Commander*, selecionamos a opção:

Dados ⇒ Modificação var. no conjunto de dados... ⇒ Converter var. numérica para fator...

Na caixa de diálogo que *Converter Variáveis Numéricas p/ Fator* (figura 6.4), selecionamos a variável que será convertida e escolhemos uma das opções: *manter as categorias expressas como números ou fornecer nomes às categorias*. Vamos dar nomes às categorias nesse exemplo. No campo *Novo nome de variável ou prefixo para múltiplas variáveis*, digitamos o nome da variável que será criada. Se não for especificado nenhum nome nesse campo, os nomes das categorias serão sobreescritos aos valores numéricos na própria variável que será convertida e não será criada uma nova variável.

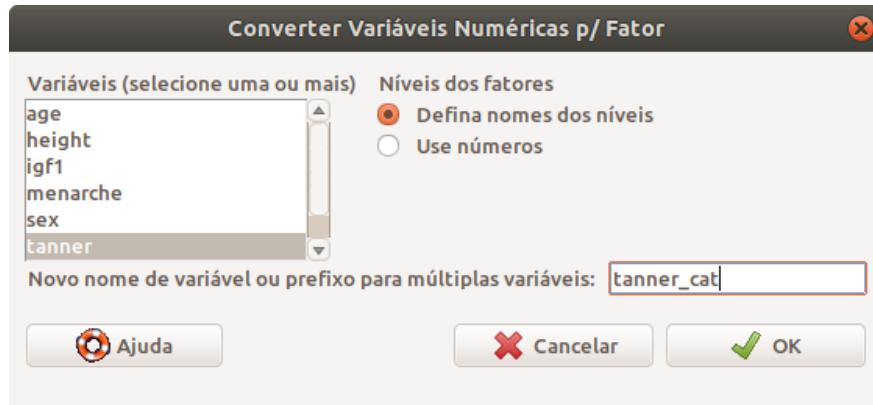


Figura 6.4: Passos para criar as categorias de uma variável: selecionamos a variável na lista da esquerda, escolhemos se as categorias serão dadas como texto e fornecemos o nome da nova variável. Clicamos em OK.

Como selecionamos a opção de fornecer os nomes para as categorias, ao clicarmos em OK na figura 6.4, uma nova caixa de diálogo aparece para darmos os nomes das categorias para cada valor numérico (figura 6.5). Finalmente, ao clicarmos em OK, a nova variável, *tanner_cat*, será criada com as categorias apropriadas.

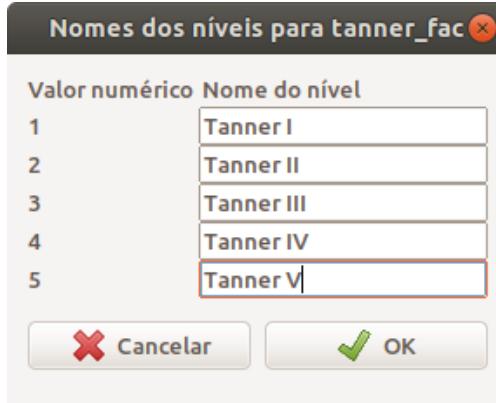


Figura 6.5: Especificação das categorias para a variável *tanner*.

O comando executado é mostrado abaixo:

```
juul2 <- within(juul2, {
  tanner_cat <- factor(tanner,
    labels=c('Tanner I', 'Tanner II', 'Tanner III', 'Tanner IV', 'Tanner V'))
})
```

Observamos que foi utilizada a função *factor*, mostrada na seção 3.8. A única diferença é que aqui foi usado o argumento *labels* e não *levels*. Nesse caso, o efeito é o mesmo.

Repitam o procedimento para a variável *menarche*, criando um nova variável *menarca_cat*.

```
juul2 <- within(juul2, {
  menarca_cat <- factor(menarche, labels=c('sim', 'não'))
})
```

Lembremos que também criamos uma variável, *sexo_cat*, da classe *factor* na seção 6.1.1, a partir da recodificação da variável *sex*, fazendo o argumento *as.factor* da função igual a *TRUE*.

6.1.3 Reordenar os níveis dos fatores

O conteúdo desta seção pode ser visualizado neste [vídeo](#).

Em diversas situações, por exemplo, ao montar uma tabela de contingência, poderíamos desejar alterar a ordem com que os níveis de um fator são apresentados nessa tabela.

Nesta seção, vamos utilizar o conjunto de dados *stroke* do pacote [ISwR \(GPL-2 | GPL-3\)](#).

Esse conjunto de dados contém todos os casos de AVC (acidente vascular cerebral) in Tartu na Estônia, durante o período de 1991 a 1993, com acompanhamento até 1 jan 1996.

Há várias variáveis nesse conjunto de dados, mas, nesse caso, estamos interessados nas variáveis *dead* e *minf*.

A variável *dead* é uma variável lógica que indica se a pessoa morreu ou não durante o estudo. A variável *minf* é um fator que indica se a pessoa possui ou não histórico de infarto do miocárdio.

Vamos supor que queiramos calcular o risco relativo de morte para pessoas com AVC e histórico ou não de infarto do miocárdio.

Esse risco é obtido dividindo-se o risco de morte para aqueles com histórico de infarto do miocárdio pelo risco de morte para aqueles sem histórico de infarto do miocárdio.

Então vamos verificar a associação entre as variáveis *minf* e *dead*.

Recordando, para abrirmos o conjunto de dados *ISwR* via *R Commander*, executamos a função

```
library(ISwR)
```

na área de script do *R commander* e, em seguida, selecionamos o conjunto *stroke* via:

Dados ⇒ Conjunto de dados em pacotes ⇒ Ler dados de pacotes 'atachados'...

Em seguida, selecionamos o conjunto de dados *stroke* no pacote *ISwR* (figura 6.6).

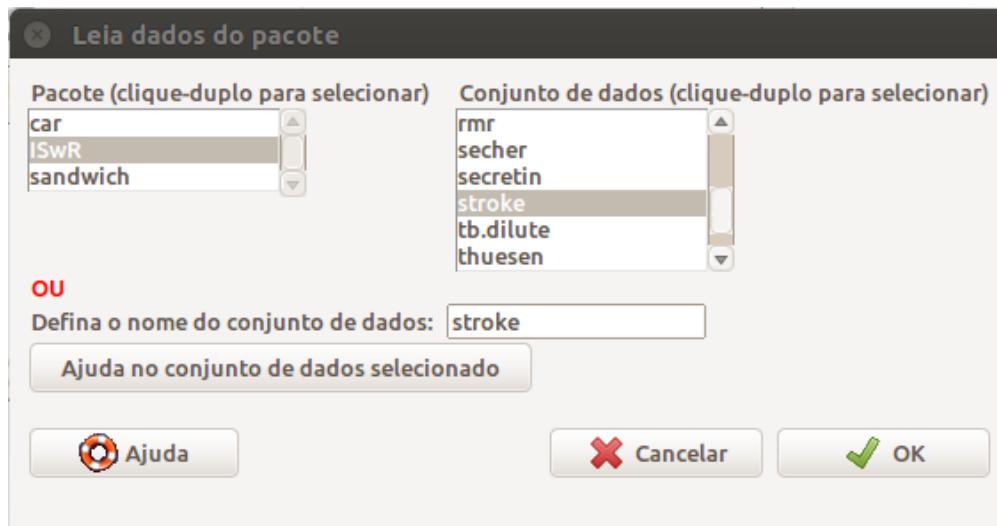


Figura 6.6: Tela para a leitura do conjunto de dados *stroke* do pacote *ISwR* via *R Commander*.

As tabelas que mostram a contagem para cada combinação das categorias das variáveis categóricas são também chamadas tabelas de contingência. Quando há duas variáveis categóricas binárias, a tabela de contingência também é chamada tabela de dupla entrada ou tabela 2x2.

Vamos montar uma tabela 2x2 no *R Commander*. Selecionamos a opção:

Estatísticas \Rightarrow Tabelas de contingência \Rightarrow Tabela de dupla entrada...

Na tela de configuração do comando para analisar uma tabela 2x2, é preciso selecionar a variável cujas categorias aparecerão nas linhas da tabela (*minf*) e a outra variável cujas categorias comporão as colunas da tabela (*dead*) (figura 6.7).

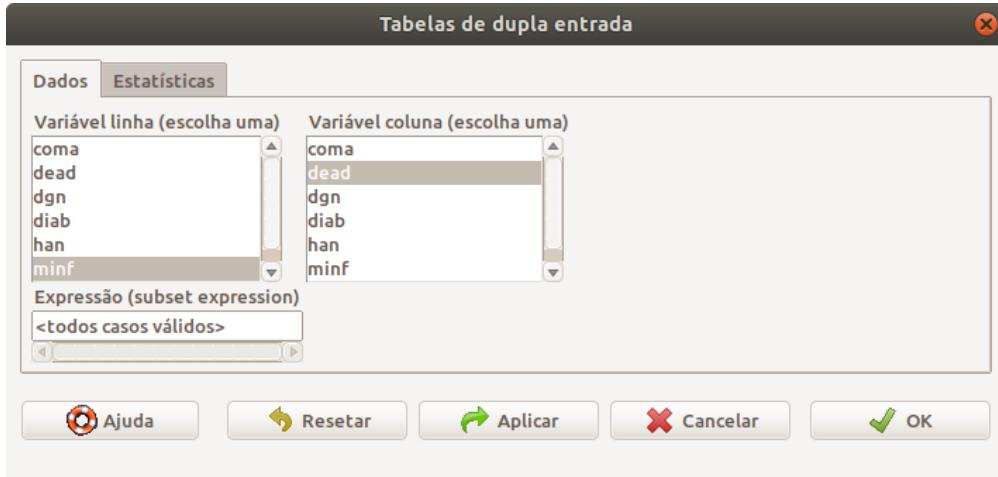


Figura 6.7: Selecionando as variáveis da tabela 2x2.

Os resultados são mostrados na figura 6.8. As frequências da tabela 2x2 são mostradas na primeira tabela. Observem que os valores das variáveis nas colunas e linhas são ordenados, por padrão, em ordem alfabética. Vemos que a tabela foi construída com o nível *No* de história de infarto do miocárdio na primeira linha e *Yes* na segunda linha. O nível *FALSE* da variável *dead* aparece na primeira coluna e *TRUE* na segunda.

```

Frequency table:
  dead
minf  FALSE  TRUE
  No    320  405
  Yes   23   74

Pearson's Chi-squared test

data: .Table
X-squared = 14.681, df = 1, p-value = 0.0001274

```

Figura 6.8: Resultado da análise da tabela de contingência 2x2 por meio do *R Commander*.

Para calcularmos o risco relativo corretamente, será necessário invertermos a ordem tanto das linhas quanto das colunas, já que o nível de referência do fator de risco é sem histórico de infarto de miocárdio e o risco que queremos estimar é o de morrer e não o de sobreviver.

Então precisamos inverter a ordem com que os níveis das variáveis *minf* e *dead* são apresentados. Para alterarmos a ordem dos níveis da variável *minf*, vamos em:

Dados ⇒ Modificação de var. no conjunto de dados... ⇒ Reordenar níveis dos fatores...

Na caixa de diálogo *Reordene Níveis do Fator* (figura 6.9), selecionamos a variável *minf*. Se selecionarmos a opção *Faça fator ordenado*, iremos fazer com que a variável passe a ser uma variável categórica ordinal. Como a variável *minf* é dicotômica, não vamos marcar essa opção.

Podemos ou não criar uma nova variável. Se utilizarmos a mesma variável, uma tela irá solicitar a confirmação se desejamos sobrescrever a variável. Nesse exemplo, vamos sobrescrever a variável *minf*, por isso não vamos alterar o conteúdo do campo *Nome do Fator*.

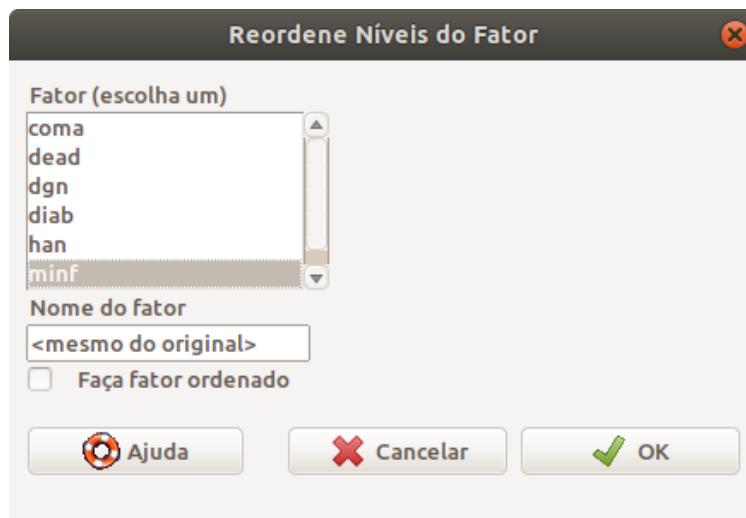


Figura 6.9: Caixa de diálogo para selecionar a variável *minf* cujos níveis serão reordenados.

Ao clicarmos em OK, uma nova caixa de diálogo nos permite especificar a nova ordem dos níveis (figura 6.10).



Figura 6.10: Caixa de diálogo para especificarmos como os níveis da variável *minf* serão ordenados.

O comando executado é mostrado abaixo. Observem que os níveis da variável *minf* foram colocados na ordem desejada.

```
stroke$minf <- with(stroke, factor(minf, levels=c('Yes', 'No')))
```

A função *factor* vai reordenar os níveis de *minf* de acordo com a ordem especificada no argumento *levels* e o resultado vai ser gravado na própria variável *minf*. Observem o \$ usado para separar a variável *minf* do conjunto de dados *stroke*, já que *stroke* é um *data.frame*, portanto as suas variáveis são referenciadas como elementos de uma lista.

Se tentássemos fazer a mesma coisa com a variável *dead*, não conseguiríamos, porque essa variável não é da classe *factor*. Nesse caso, podemos recorrer à função *Recode* (seção 6.1.1), ou usar a linha de comando, como mostrado a seguir.

Podemos copiar e colar o comando anterior e substituir a variável *minf* pela variável *dead* e os níveis de *minf* pelos níveis de *dead* na ordem desejada: *TRUE* e *FALSE*, como mostrado no comando abaixo:

```
stroke$dead <- with(stroke, factor(dead, levels=c('TRUE', 'FALSE')))
```

Agora a classe da variável *dead* passa a ser da classe *factor*.

Ao executarmos o comando anterior e novamente montarmos a tabela 2x2, relacionando as variáveis *minf* e *dead*, obtemos o resultado mostrado na figura 6.11. Agora o nível de referência da variável história de infarto do miocárdio está na linha de baixo e a primeira coluna mostra o número de mortes para os que possuem ou não histórico de infarto do miocárdio.

```
Frequency table:
  dead
minf  TRUE FALSE
  Yes    74    23
  No    405   320

Pearson's Chi-squared test

data: .Table
X-squared = 14.681, df = 1, p-value = 0.0001274
```

Figura 6.11: Análise da tabela 2 x 2 (*minf* x *dead*) após a reordenação dos níveis dos fatores.

Os dois comandos a seguir mostram que os níveis de *minf* e *dead* estão agora na ordem desejada.

```
levels(stroke$minf)
```

```
## [1] "Yes" "No"
```

```

levels(stroke$dead)

## [1] "TRUE"  "FALSE"

```

6.1.4 Computar nova variável

O conteúdo desta seção pode ser visualizado neste [vídeo](#).

Vamos supor que desejamos calcular o índice de massa corporal (IMC) para as observações do conjunto de dados *juul2*. Para isso, utilizamos a seguinte opção no *R Commander*:

Dados ⇒ Modificação de variáveis no conjunto de dados... ⇒ Computar nova variável...

A figura 6.12 mostra a caixa de diálogo para computar o *IMC* a partir das variáveis *weight* e *height*. A variável *height* foi dividida por 100, porque ela está em cm.



Figura 6.12: Caixa de diálogo para especificarmos o cálculo do IMC.

O comando executado é mostrado abaixo:

```

juul2$imc <- with(juul2, weight / (height/100)^2)

```

A função *with* aqui funciona de maneira semelhante à função *within*, porém ela não gera um novo *data frame*. Nesse exemplo, o resultado é um vetor com os valores de *imc* para cada registro de *juul2*.

Vejamos os últimos 6 valores das variáveis *weight*, *height* e *imc* no conjunto de dados *juul2*:

```
tail(juul2[,c("weight", "height", "imc")], 6)
```

```
##      weight height      imc
## 1334    52.5 164.4 19.42476
## 1335    70.5 168.9 24.71325
## 1336     NA     NA     NA
## 1337     NA     NA     NA
## 1338     NA     NA     NA
## 1339    68.0 168.0 24.09297
```

Essa operação também poderia ser realizada da seguinte forma:

```
juul2$imc <- juul2$weight / (juul2$height/100)^2
```

Vamos ver um outro exemplo. Vamos supor que queremos criar uma variável binária que indique se cada pessoa é menor ou maior de 18 anos. Novamente, vamos em:

Dados \Rightarrow Modificação de variáveis no conjunto de dados... \Rightarrow Computar nova variável...

A figura 6.13 mostra a caixa de diálogo para computar uma variável binária que indica se a pessoa é maior de idade ou não.



Figura 6.13: Caixa de diálogo para computar uma variável binária que indica se a pessoa é maior de idade ou não.

Ao clicarmos em OK, o comando executado é mostrado a seguir:

```
juul2$age_bin <- with(juul2, age >= 18)
```

Vejamos os últimos 6 valores das variáveis *age* e *age_bin* no conjunto de dados *juul2*:

```
tail(juul2[,c("age", "age_bin")], 6)
```

```
##      age age_bin
## 1334 58.95   TRUE
## 1335 60.99   TRUE
## 1336 62.73   TRUE
## 1337 65.00   TRUE
## 1338 67.88   TRUE
## 1339 75.12   TRUE
```

Observem que a nova variável é uma variável lógica, assumindo os valores *TRUE* para aqueles maiores de 18 anos e *FALSE* para os menores de 18 anos.

6.1.5 Agrupar em classes uma variável numérica

O conteúdo desta seção pode ser visualizado neste [vídeo](#).

Frequentemente é necessário agrupar os valores de uma variável numérica em classes (ou faixas de valores). Um caso clássico é o de agrupar os valores de idade em faixas etárias. Para realizarmos um agrupamento em classes no *R Commander*, usamos a opção:

Dados \Rightarrow Modificação var. no conjunto de dados... \Rightarrow Agrupar em classes var. numérica...

A figura 6.14 mostra a caixa de diálogo para especificarmos como será realizado o agrupamento. É preciso escolher a variável a ser agrupada, o nome para a variável que será criada, o número de classes a serem utilizadas, como serão nomeadas as classes e como será realizado o agrupamento. Nesse exemplo, foi escolhida a variável *age*, a variável a ser criada será *faixa_etaria*, o número de classes é 10, cada classe será identificada pelo intervalo de valores e cada classe terá a mesma largura.

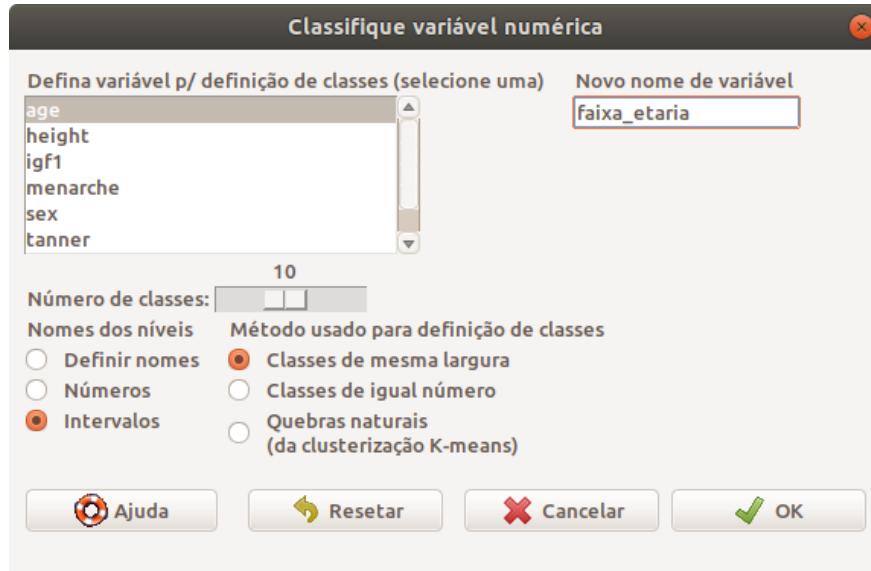


Figura 6.14: Caixa de diálogo para especificarmos como uma variável numérica será agrupada em faixas.

O comando executado é mostrado abaixo. A função *binVariable* do pacote *RcmdrMisc* foi utilizada para realizar o agrupamento.

```
juul2$faixa_etaria <- with(juul2, binVariable(age, bins=10,
                                              method='intervals', labels=NULL))
```

Vejamos os últimos 6 valores das variáveis *age* e *faixa_etaria* no conjunto de dados *juul2*:

```
tail(juul2[,c("age", "faixa_etaria")], 6)
```

```
##           age faixa_etaria
## 1334 58.95 (58.2,66.4]
## 1335 60.99 (58.2,66.4]
## 1336 62.73 (58.2,66.4]
## 1337 65.00 (58.2,66.4]
## 1338 67.88 (66.4,74.7]
## 1339 75.12 (74.7,83.1]
```

Como a variável criada, *faixa_etaria*, é um fator, podemos ver as classes que foram criadas por meio da função *levels*.

```
levels(juul2$faixa_etaria)
```

Como nesse agrupamento selecionamos a opção *intervalos* para os nomes dos níveis (classes), o argumento *labels* na função *binVariable* deve ser *NULL*. Se tivéssemos escolhido a opção *Definir nomes* para os níveis, uma nova tela iria aparecer para especificarmos os nomes, e o argumento *labels* conteria um vetor com os nomes especificados. Caso a opção para os níveis

fosse *Números*, o argumento *labels* seria igual a *FALSE*, e cada classe receberia um número inteiro correspondente à ordem da classe.

Nesse exemplo, criamos classes de mesma largura. Outra opção seria criar classes com tamanhos diferentes, mas que contivessem aproximadamente o mesmo número de observações. Finalmente, seria possível realizar uma análise de *cluster* para definir o agrupamento.

Observamos que as faixas criadas são um tanto artificiais. Se quisermos definir as faixas etárias de acordo com os padrões utilizados para faixas etárias, teremos que recorrer a outras funções. Uma função mais flexível para transformar uma variável numérica em categórica é a função *cut*. Vamos supor que desejamos criar as seguintes faixas etárias:

- crianças: 0 até 10 anos;
- adolescentes: acima de 10 até 20 anos;
- adultos: acima de 20 até 60 anos;
- idosos: acima de 60 anos.

O comando abaixo mostra como usar a função *cut* para realizar essa estratificação:

```
juul2$faixa_etaria2 <- with(juul2, cut(x = age,
                                         breaks = c(0, 10, 20, 60, Inf),
                                         labels=c('(0,10]', '(10-20]', '(20-60]', '>60'),
                                         ordered_result = TRUE, right = TRUE))
```

Vejamos os últimos 6 valores das variáveis *age* e *faixa_etaria2* no conjunto de dados *juul2*:

```
tail(juul2[, c("age", "faixa_etaria2")], 6)
```

```
##      age faixa_etaria2
## 1334 58.95      (20-60]
## 1335 60.99          >60
## 1336 62.73          >60
## 1337 65.00          >60
## 1338 67.88          >60
## 1339 75.12          >60
```

Vamos entender os argumentos da função *cut*:

- 1) o argumento *x* especifica que variável será transformada;
- 2) o argumento *breaks* especifica os limites de cada categoria (intervalo) que será criada;
- 3) o argumento *labels* especifica os rótulos que serão atribuídos a cada categoria. Se o argumento *labels* não for especificado, os rótulos serão criados automaticamente, de acordo com os limites especificados pelo argumento *breaks* e como os limites dos intervalos serão tratados (argumento *right*);
- 4) o argumento *ordered_result* especifica se as categorias criadas serão ordenadas ou não. Nesse caso, a variável criada será ordinal;
- 5) se o argumento *right* for *TRUE*, os intervalos criados serão fechados à direita e abertos à esquerda; se for *FALSE*, os intervalos criados serão abertos à direita e fechados à esquerda.

6.1.6 Padronizar variáveis

O conteúdo desta seção pode ser visualizado neste [vídeo](#).

Em diversas situações, pode ser necessário realizar a padronização de variáveis numéricas. Em geral a padronização de uma variável X cria uma outra variável Y, por meio da seguinte operação:

$$Y = \frac{X - \mu}{\sigma}$$

onde μ é a média e σ é o desvio padrão da variável X.

Para realizarmos a padronização de variáveis no *R Commander*, selecionamos a opção:

Dados \Rightarrow Modificação de variáveis no conjunto de dados... \Rightarrow Padronizar variáveis...

Na caixa de diálogo *Padronize Variáveis* (figura 6.15), selecionamos as variáveis a serem padronizadas e pressionamos o botão OK.



Figura 6.15: Caixa de diálogo para selecionarmos as variáveis que serão padronizadas.

O comando executado pelo *R Commander* é mostrado abaixo:

```
juul2 <- local({
  .Z <- scale(juul2[,c("igf1","imc")])
  within(juul2, {
    Z.imc <- .Z[,2]
    Z.igf1 <- .Z[,1]
  })
})
```

Vamos entender esse comando. A função *scale* faz a padronização das variáveis especificadas em seu argumento. Nesse exemplo, foram especificadas duas variáveis de *juul2*: *igf1* e *imc*. O resultado da padronização é armazenado no objeto *.Z*, que é uma matriz com duas colunas, onde a primeira coluna contém os valores padronizados de *igf1* e a segunda coluna os valores padronizados de *imc*. Novamente a função *within* é utilizada para criar um *data frame* a partir de *juul2*, acrescentando duas variáveis, *Z.imc* e *Z.igf1*, que contêm respectivamente a segunda e primeira coluna de *.Z*. Finalmente *juul2* é substituído por esse novo *data frame*.

Se observarmos atentamente o comando acima, verificamos que o objeto *.Z* foi criado apenas para realizar a padronização das variáveis. Após as duas padronizações serem armazenadas em variáveis do *juul2*, o objeto *.Z* não é mais necessário. A função *local* evita que o objeto *.Z* seja armazenado no sistema; ele é descartado após a execução do comando. A função *local* não é necessária para realizar a operação. Caso ela não fosse utilizada, o objeto *.Z* continuaria a existir na memória.

As variáveis *Z.imc* e *Z.igf1* são adicionadas ao *data frame*:

```
tail(juul2[,c("imc", "Z.imc", "igf1", "Z.igf1")], 6)

##           imc      Z.imc    igf1      Z.igf1
## 1334 19.42476 0.2618718 218 -0.7142839
## 1335 24.71325 2.0078782 226 -0.6675100
## 1336      NA        NA     NA        NA
## 1337      NA        NA    106 -1.3691183
## 1338      NA        NA    217 -0.7201306
## 1339 24.09297 1.8030923 135 -1.1995630
```

6.1.7 Remover variáveis de um conjunto de dados

Os conteúdos desta seção e das seções 6.1.8 e 6.1.9 podem ser visualizados neste [vídeo](#).

Ao trabalharmos com um conjunto de dados, frequentemente criamos diversas variáveis que, mais tarde, podem não ser mais necessárias. Por exemplo, em seções anteriores, convertemos as variáveis *sex*, *tanner* e *menarche* do conjunto de dados *juul2* para fator, gerando as variáveis *sexo_cat*, *tanner_cat* e *menarca_cat*, respectivamente. Após essas conversões, as variáveis *sex*, *tanner* e *menarche* não são mais necessárias e poderiam ser excluídas.

Para remover variáveis de um conjunto de dados que não são mais necessárias no *R Commander*, usamos a opção:

Dados ⇒ Modificação var. no conjunto de dados... ⇒ Apagar var. de conjunto de dados...

Na caixa de diálogo *Eliminar Variáveis* (figura 6.16), selecionamos as variáveis que desejamos remover. Vamos excluir as variáveis *sex*, *tanner* e *menarche*.

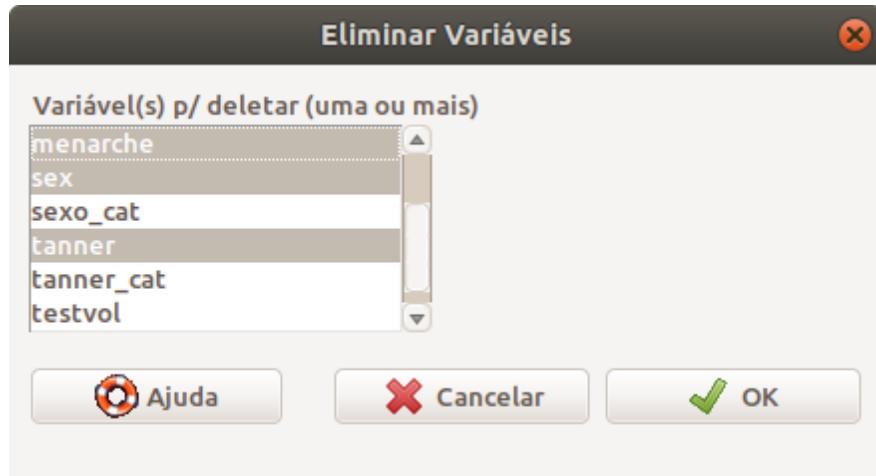


Figura 6.16: Caixa de diálogo para especificarmos que variáveis serão eliminadas do conjunto de dados.

Ao pressionarmos o botão OK e, após a tela de confirmação, o comando abaixo será executado. Para removermos uma variável de um conjunto de dados, basta atribuirmos *NULL* a ela.

```
juul2 <- within(juul2, {  
  menarche <- NULL  
  sex <- NULL  
  tanner <- NULL  
})
```

6.1.8 Renomear variáveis

Após removermos as antigas variáveis *sex*, *tanner* e *menarche* do conjunto de dados *juul2*, vamos alterar os nomes das variáveis *sexo_cat*, *tanner_cat* e *menarca_cat* para *sexo*, *tanner* e *menarca* respectivamente. Para renomear variáveis no *R Commander*, utilizamos a opção:

Dados ⇒ Modificação de variáveis no conjunto de dados... ⇒ Renomear variáveis...

A caixa de diálogo da figura 6.17 permite a seleção das variáveis que serão renomeadas.

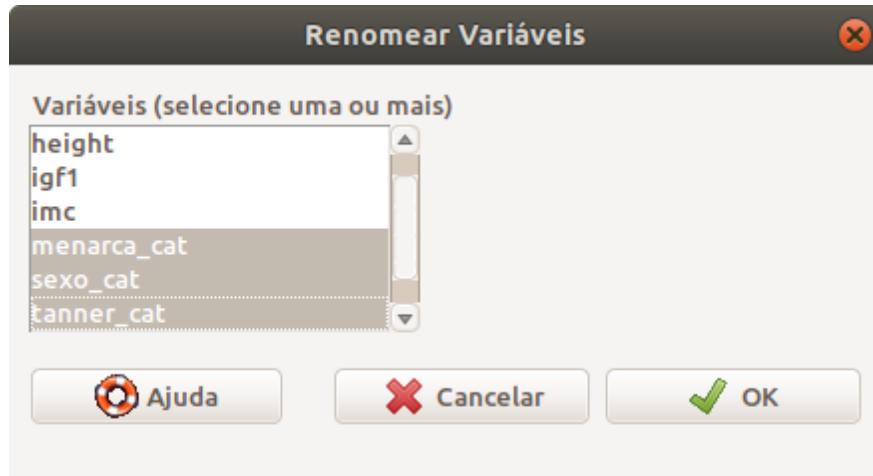


Figura 6.17: Caixa de diálogo para selecionar as variáveis que serão renomeadas.

Ao pressionarmos o botão OK, especificamos os novos nomes das variáveis na tela *Nomes das Variáveis* (figura 6.18).



Figura 6.18: Caixa de diálogo para especificarmos os novos nomes das variáveis selecionadas.

Após pressionarmos o botão OK, o comando abaixo será executado. Para alterarmos o nome de uma variável, basta alterar o seu nome na posição correspondente da variável no vetor *names*(conjunto de dados). Nesse exemplo, as três variáveis ocupam as 9^a, 6^a e 8^a posições, respectivamente, no conjunto de dados *juul2*.

```
names(juul2)[c(9,6,8)] <- c("menarca", "sexo", "tanner")
```

6.1.9 Adição do número de observações aos dados

A opção abaixo simplesmente adiciona mais uma variável ao conjunto de dados, onde cada valor indica o número da linha no conjunto de dados.

Dados ⇒ Modif. de var. no conj. de dados... ⇒ Adicionar número da obs. nos dados...

Observem no comando gerado que foi criada uma variável (*ObsNumber*) que contém a sequência de números inteiros de 1 a 1339 (o número de registros em *juul2*)

```
juul2$ObsNumber <- 1:1339  
head(juul2[, c("igf1", "sexo")])
```

```
##   igf1     sexo  
## 1   90    <NA>  
## 2   88    <NA>  
## 3  164    <NA>  
## 4  166    <NA>  
## 5  131    <NA>  
## 6  101 masculino
```

Salvem o conjunto de dados *juul2* após a conversão das variáveis *sex*, *menarche* e *tanner* para fator em um arquivo de dados no seu computador para que não tenham que repetir essas transformações a cada vez que forem utilizar esse conjunto de dados.

6.1.10 Converter variáveis do tipo *character* para fatores

Para realizarmos a conversão de uma variável do tipo *character* para fator, utilizamos a opção:

Dados ⇒ Modif. de var. no conj. de dados... ⇒ Convert character variables to factors...

Na caixa de diálogo dessa opção, basta selecionar a variável. Se desejarmos criar uma nova variável do tipo fator, mantendo a antiga variável como *character*, digitamos o nome da nova variável no campo *Novo nome de variável ou prefixo para múltiplas variáveis* e clicamos em OK. Caso desejemos simplesmente transformar a variável em fator, mantendo o mesmo nome, não mexemos no campo *Novo nome de variável ou prefixo para múltiplas variáveis* e clicamos em OK. Nesse caso, o programa irá perguntar se desejamos sobrescrever a variável. Se respondermos *Não* à pergunta, o comando será cancelado e a variável não será convertida.

Caso mais de uma variável seja selecionada, se desejarmos criar novas variáveis do tipo fator a partir das variáveis selecionadas, mantendo as antigas variáveis como *character*, precisamos digitar um prefixo no campo *Novo nome de variável ou prefixo para múltiplas variáveis*. O nome de cada nova variável que será criada será o nome da variável que será convertida precedido desse prefixo. Em seguida, cliquemos em OK. Caso desejarmos simplesmente transformar as variáveis em fator, mantendo o mesmo nome, não mexemos no campo *Novo nome de variável ou prefixo para múltiplas variáveis* e clicamos em OK. Nesse caso, o programa irá perguntar se desejamos sobrescrever cada variável selecionada. Se respondermos *Não* a alguma das perguntas, o comando será cancelado e nenhuma variável será convertida.

6.2 Manipulação do conjunto de dados

Além de manipular variáveis em um conjunto de dados, há uma série de recursos que permitem a manipulação do próprio conjunto de dados. A figura 6.19 mostra as opções disponíveis no *R Commander*. Esse submenu pode ser acessado a partir da barra de menus da seguinte forma:

Dados ⇒ *Conjunto de dados ativo*

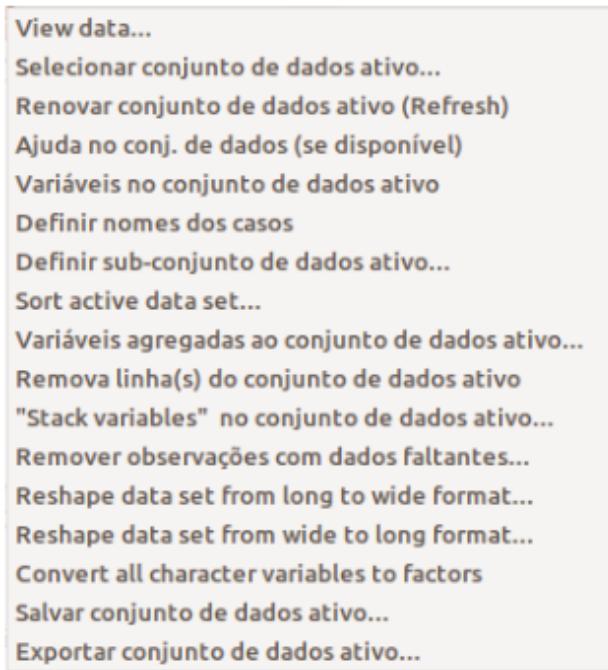


Figura 6.19: Recursos disponíveis para a manipulação do conjunto de dados ativo.

As duas últimas opções desse menu (*Salvar conjunto de dados ativo...* e *Exportar conjunto de dados ativo...*) foram mostradas no capítulo 4.

6.2.1 Seleção do conjunto de dados ativo

Os conteúdos desta seção e das seções 6.2.2 e 6.2.3 podem ser visualizados neste [vídeo](#).

As opções disponíveis no *R Commander* atuam sobre o conjunto de dados que está ativo no *R Commander*. Embora vários conjuntos de dados possam estar carregados na área de trabalho do R, somente um conjunto de dados está ativo em cada momento no *R Commander*.

Para escolhermos o conjunto de dados que estará ativo no *R Commander*, selecionamos a opção

Dados ⇒ *Conjunto de dados ativo* ⇒ *Selecionar conjunto de dados ativo...*

Na tela *Selecione o conjunto de dados* (figura 6.20), são mostrados os conjuntos de dados que estão carregados no R na sessão corrente. Vamos selecionar o conjunto *Melanoma*.

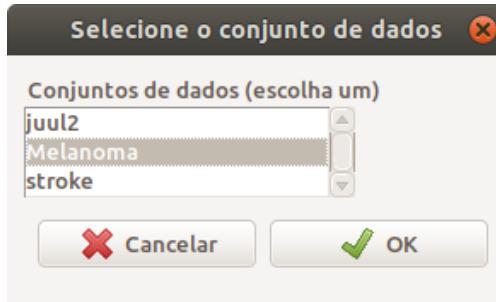


Figura 6.20: Caixa de diálogo para selecionar o conjunto de dados ativo no *R Commander*.

Um atalho para selecionarmos o conjunto de dados ativo é clicarmos sobre o nome do conjunto de dados ativo no momento, ao lado do rótulo *Conjunto de dados*, abaixo da barra de menus.

6.2.2 Visualização dos dados

O botão *Ver conjunto de dados*, abaixo da barra de menus, exibe o conteúdo do conjunto de dados. Para escolhermos que variáveis ou que subconjunto de observações serão visualizados, selecionamos a opção:

Dados \Rightarrow *Conjunto de dados ativo* \Rightarrow *View data...*

Na caixa de diálogo dessa opção (figura 6.21), podemos selecionar as variáveis que serão visualizadas ou marcar para visualizar todas as variáveis. Nesse exemplo, selecionamos as variáveis *status* e *thickness* e vamos exibir somente as observações relativas aos óbitos por melanoma (*status == 1*).

Ao pressionarmos o botão OK, somente as duas variáveis serão exibidas para os óbitos devido ao melanoma.

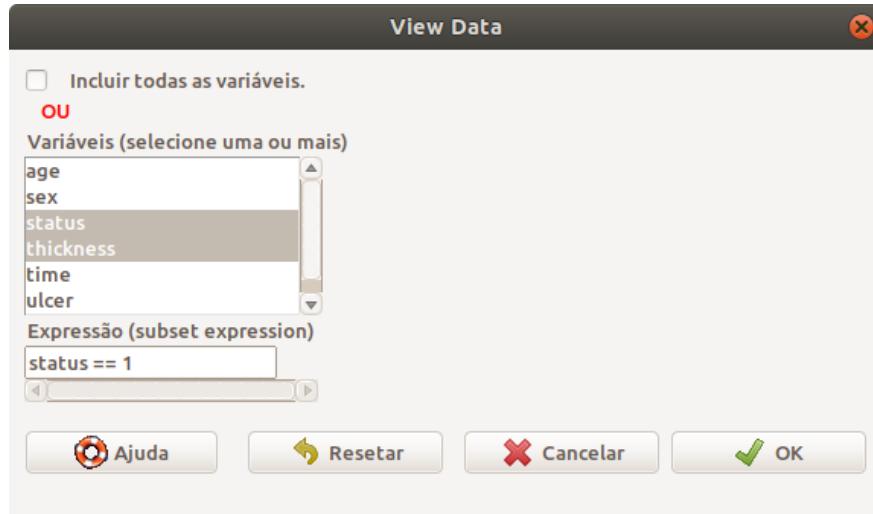


Figura 6.21: Caixa de diálogo para selecionar as variáveis a serem visualizadas.

6.2.3 Renovar conjunto de dados ativo...

Às vezes é necessário renovar o conjunto de dados no *R Commander*, especialmente quando fazemos alguma alteração no conjunto de dados diretamente a partir da linha de comando, para que essas alterações fiquem visíveis nas opções de menu e telas de configuração do *R Commander*.

Vamos supor que tenhamos convertido as variáveis *status* e *sex* do conjunto de dados *Melanoma*, executando o comando abaixo na área de script do *R Commander*, sem utilizarmos a opção para converter variável numérica para fator do menu.

```
Melanoma <- within(Melanoma, {
  status_fac <- factor(status, labels=c('óbito por melanoma','vivo',
                                         'óbito por outras causas'))
  sexo_fac <- factor(sex, labels=c('feminino','masculino'))
})
```

As variáveis *status_fac* e *sexo_fac* foram criadas e aparecem quando clicamos no botão *Ver conjunto de dados*, mas se tentarmos criar um gráfico de barras a partir do menu:

Gráficos

vemos que a opção *Gráfico de barras* não está habilitada. É necessário renovar o conjunto de dados ativo.

Para renovar o conjunto de dados ativo, utilizamos a opção:

Dados ⇒ Conjunto de dados ativo ⇒ Renovar conjunto de dados ativo (Refresh)

Após essa operação, a opção *Gráfico de barras* passa a estar habilitada e podemos selecionar uma das variáveis *status_fac* ou *sexo_fac* para criarmos um diagrama de barras.

6.2.4 Ajuda no conjunto de dados ativo (se disponível)

Os conteúdos desta seção e das seções 6.2.5 e 6.2.6 podem ser visualizados neste [vídeo](#).

A opção do menu abaixo irá mostrar a ajuda sobre o conjunto de dados ativo, se ela estiver disponível. Em geral os conjuntos de dados disponíveis em pacotes do R oferecem uma ajuda sobre o seu conteúdo, especificando que variáveis estão presentes no conjunto de dados e as respectivas unidades e codificações.

Por meio da opção abaixo, a ajuda para o conjunto de dados ativo será disponibilizada no seu navegador Web padrão.

Dados ⇒ Conjunto de dados ativo ⇒ Ajuda no conjunto de dados (se disponível)

Com o *Melanoma* como conjunto de dados ativo, o comando a seguir será executado:

```
help("Melanoma")
```

6.2.5 Variáveis no conjunto de dados ativo

A opção abaixo lista os nomes das variáveis do conjunto de dados ativo.

Dados ⇒ Conjunto de dados ativo ⇒ Variáveis no conjunto de dados ativo

Essa opção executa a função *names(conjunto de dados ativo)*.

```
names(Melanoma)
```

6.2.6 Definir nomes dos casos

Vamos supor que o conjunto de dados ativo contenha uma coluna com os nomes dos pacientes. O script abaixo cria um “nome” para cada linha, onde o primeiro nome é “n1” e o último é “n205”. Não se preocupem em entender esse *script* agora.

```
nomes = NULL
for (i in 1:nrow(Melanoma)) {
  nomes = c(nomes, paste("n", i, sep=''))
}
Melanoma$nomes = nomes
```

Se verificarmos os registros de *Melanoma*, veremos a adição da variável *nomes*. Além disso, os nomes das linhas são os números 1 até 205, precedidos da letra n. A função *row.names(data frame)* exibe os nomes das linhas do *data frame*.

```

head(Melanoma[, c("time", "status", "nomes")])

##   time status nomes
## 1    10      3    n1
## 2    30      3    n2
## 3    35      2    n3
## 4    99      3    n4
## 5   185      1    n5
## 6   204      1    n6

head(row.names(Melanoma))

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

```

Para fazermos com que os nomes das linhas sejam iguais aos de uma variável de um conjunto de dados, usamos a opção:

Dados ⇒ Conjunto de dados ativo ⇒ Definir nomes dos casos

Na caixa de diálogo *Defina nome do caso* (figura 6.22), selecionamos a variável que contém os nomes e pressionamos o botão OK. Pode ser necessário renovar o conjunto de dados ativo para a variável *nomes* aparecer na lista de variáveis (seção 6.2.3).

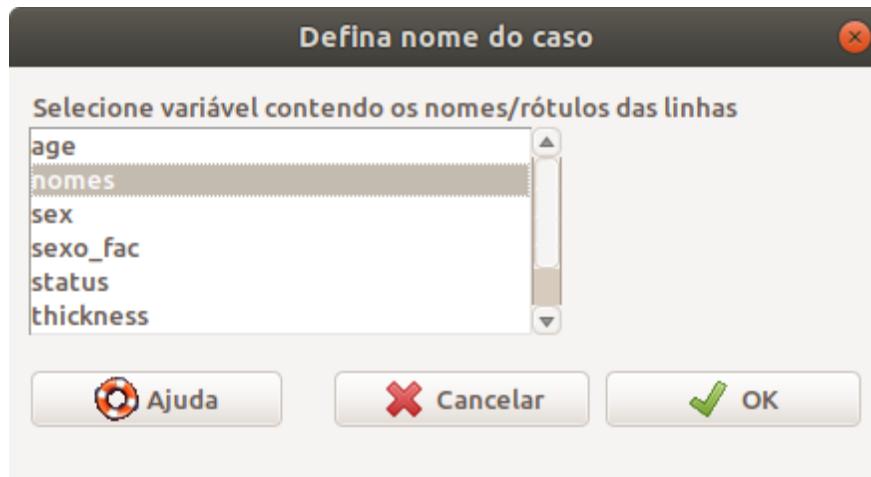


Figura 6.22: Caixa de diálogo para selecionar a variável que contém os nomes que serão usados para as observações do conjunto de dados ativo.

Os comandos executados são mostrados a seguir. Os valores das linhas do *Melanoma* são substituídos pelos valores da variável *nomes* e essa variável é removida do conjunto de dados.

```
row.names(Melanoma) <- as.character(Melanoma$nomes)
Melanoma$nomes <- NULL
```

Vejam as alterações ocorridas no conjunto de dados:

```
head(Melanoma[, c("time", "status")])
```

```
##      time status
## n1     10      3
## n2     30      3
## n3     35      2
## n4    99      3
## n5   185      1
## n6   204      1
```

```
head(row.names(Melanoma))
```

```
## [1] "n1" "n2" "n3" "n4" "n5" "n6"
```

6.2.7 Definir subconjunto de dados ativo

O conteúdo desta seção e da seção 6.2.7.1 podem ser visualizados neste [vídeo](#).

Vamos supor que desejamos fazer uma análise somente com menores de idade no conjunto de dados Melanoma. Então temos que gerar um subconjunto de dados. No *R Commander*, selecionamos a opção:

Dados ⇒ Conjunto de dados ativo ⇒ Definir subconjunto de dados ativo

Na caixa de diálogo *Definir um subconjunto dos dados* (figura 6.23), damos um nome para o novo conjunto de dados, indicamos que variáveis serão incluídas nesse conjunto de dados (o padrão é todas), e a expressão que especifica o filtro que será aplicado ao conjunto de dados ativo. Nesse caso, o filtro é: *age < 18*.

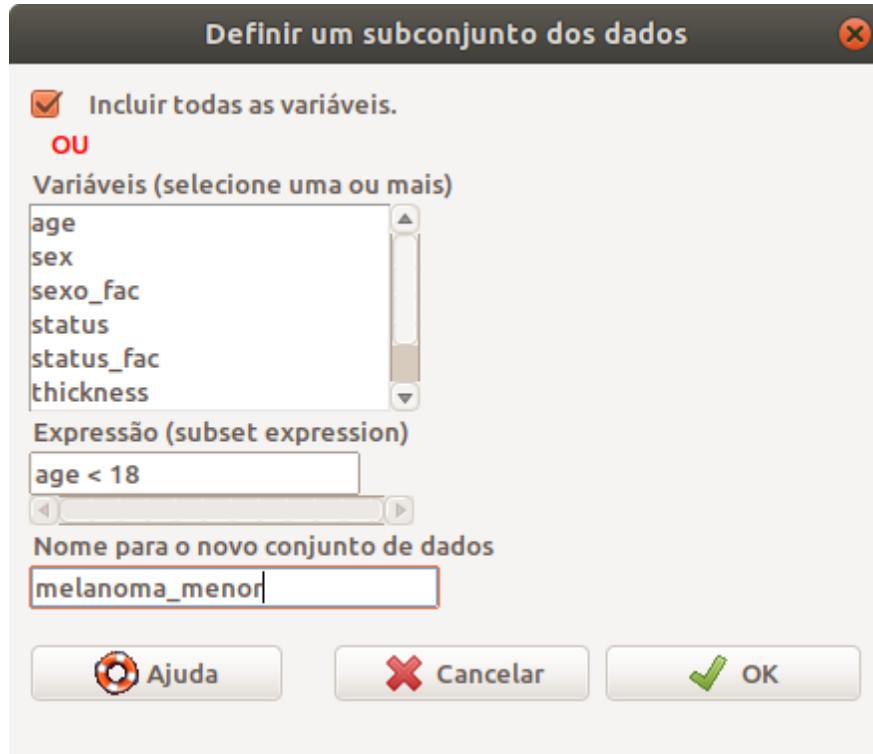


Figura 6.23: Caixa de diálogo para especificarmos um subconjunto de *Melanoma* composto pelos menores de 18 anos.

Ao pressionarmos OK, o comando abaixo é executado. A função *subset()* retorna um subconjunto do objeto especificado no primeiro argumento, de acordo com a expressão definida no argumento *subset*.

```
melanoma_menor <- subset(Melanoma, subset=(age < 18))
```

O conjunto de dados *melanoma_menor* possui somente 4 registros:

```
melanoma_menor[, c("time", "status", "age")]
```

```
##      time status age
## n15    469     1 14
## n29    858     1 16
## n74   1710     2 15
## n174  3385     2  4
## n186  3776     2 12
```

Vamos ver um outro exemplo. Vamos voltar a fazer com que *Melanoma* seja o conjunto de dados ativo e vamos criar um outro subconjunto de *Melanoma*, excluindo os óbitos por outras causas. Novamente, utilizamos a seguinte opção:

Dados ⇒ Conjunto de dados ativo ⇒ Definir subconjunto de dados ativo

Na caixa de diálogo *Definir um subconjunto dos dados* (figura 6.24), damos um nome para o novo conjunto de dados (*melanomaSemOutrosObitos*), selecionamos todas as variáveis e a expressão que especifica o filtro que será aplicado ao conjunto de dados ativo (*status != 3*). Poderíamos também ter utilizado a expressão *status_fac != "óbito por outras causas"* para especificarmos o subconjunto de *Melanoma*.

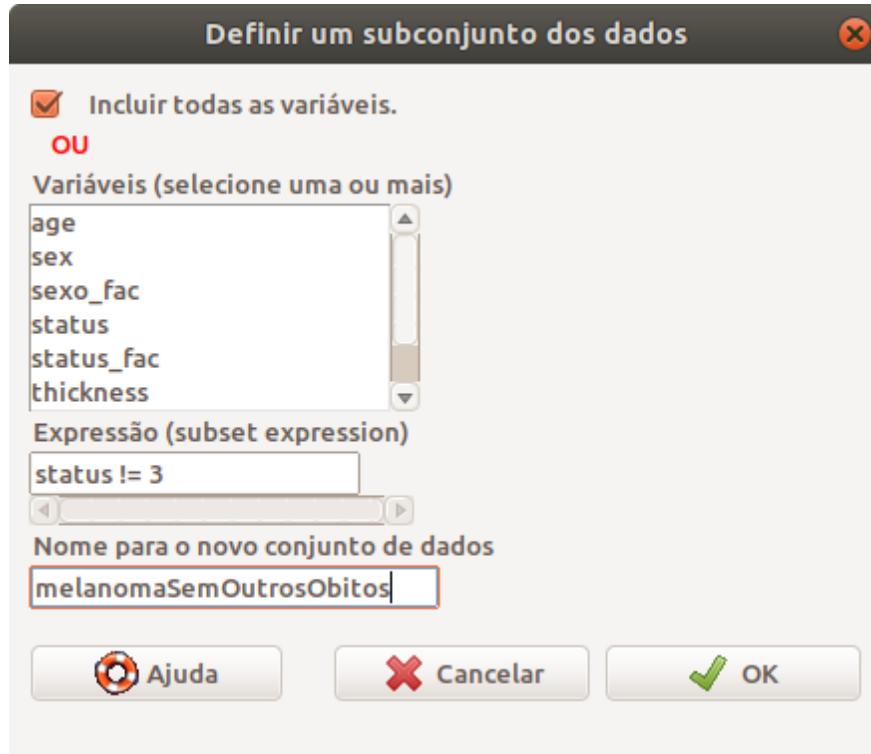


Figura 6.24: Caixa de diálogo para especificarmos um subconjunto do conjunto de dados *Melanoma*, excluindo os óbitos por outras causas.

Ao pressionarmos OK, o comando a seguir é executado.

```
melanomaSemOutrosObitos <- subset(Melanoma, subset=status != 3)
```

6.2.7.1 Abandonar fatores não usados

Continuando com o conjunto de dados *melanomaSemOutrosObitos* da seção anterior, onde excluímos de *Melanoma* todos os registros cujo status fosse “óbito por outras causas”.

Se observarmos os níveis da variável *status_fac*, veremos que ela continua com três níveis, apesar de o nível *óbito por outras causas* não ser utilizado.

```
levels(melanomaSemOutrosObitos$status_fac)
```

```
## [1] "óbito por melanoma"      "vivo"                  "óbito por outras causas"
```

Vamos fazer um diagrama de barras da variável *status_fac* por meio da opção:

Gráficos ⇒ Diagrama de barras

O gráfico resultante é mostrado na figura 6.25. Observamos que o nível correspondente a *óbito por outras causas* aparece, mesmo sabendo que não há nenhuma observação com esse nível no conjunto de dados.

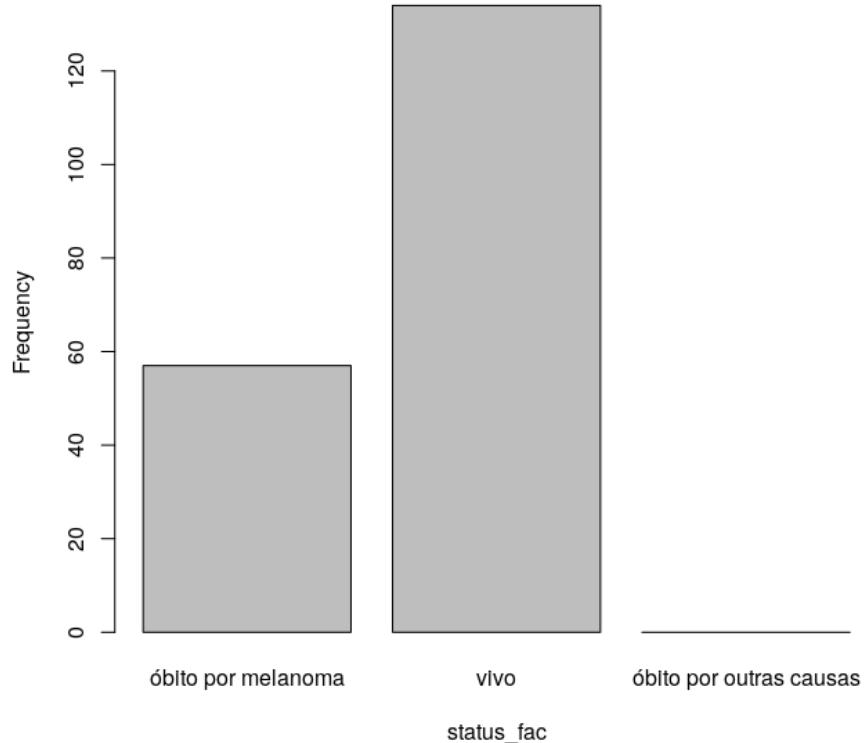


Figura 6.25: Diagrama de barras da variável *status_fac* do conjunto de dados *melanomaSemOutrosObitos*.

Para removermos os níveis não utilizados no *R Commander*, usamos a opção:

Dados ⇒ Modificação variáveis no conj. de dados ⇒ Abandonar fatores não usados

A caixa de diálogo *Abandonar níveis dos fatores não utilizados* (figura 6.26) nos permite selecionar as variáveis cujos níveis não utilizados serão abandonados. Podemos selecionar também todas as variáveis da classe *factor*.

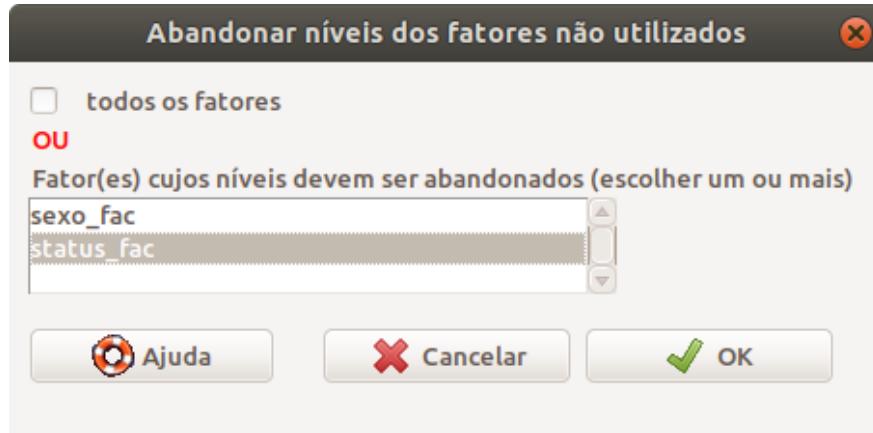


Figura 6.26: Caixa de diálogo para selecionar as variáveis cujos níveis não usados serão removidos.

Ao pressionarmos OK e, após uma confirmação, o comando abaixo será executado. A função `droplevels()` remove os níveis não usados.

```
melanomaSemOutrosÓbitos <- within(melanomaSemOutrosÓbitos, {
  status_fac <- droplevels(status_fac)
})
```

Ao verificarmos os níveis da variável `status_fac`, veremos que são listados somente os níveis utilizados.

```
levels(melanomaSemOutrosÓbitos$status_fac)

## [1] "óbito por melanoma" "vivo"
```

Se fizermos o gráfico de barras da variável `status_fac` novamente, vamos obter o gráfico mostrado na figura 6.27. Observamos que agora o nível correspondente a *óbito por outras causas* não aparece no gráfico.

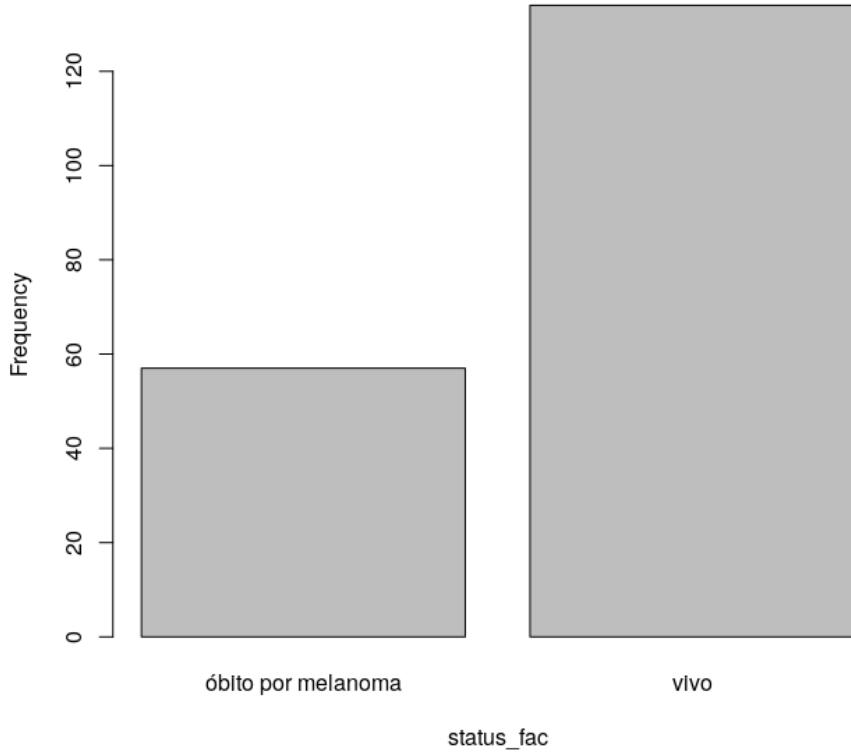


Figura 6.27: Diagrama de barras da variável `status_fac` do conjunto de dados *melanomaSemOutrosObitos*, após a remoção dos níveis não usados da variável `status_fac`.

6.2.8 Ordenar o conjunto de dados ativo

O conteúdo desta seção pode ser visualizado neste [vídeo](#).

Vamos tornar o conjunto de dados *Melanoma* novamente o conjunto de dados ativo no *R Commander*.

Para ordenarmos o conjunto de dados ativo de acordo com os valores de uma ou mais variáveis, podemos usar a função *order*. No *R Commander*, utilizamos a seguinte opção:

Dados ⇒ Conjunto de dados ativo ⇒ Sort active data set...

Na caixa de diálogo *Sort Active Data Set* (figura 6.28), selecionamos as variáveis que definirão a ordenação, a direção da ordenação (crescente ou decrescente) e o nome do conjunto de dados que será criado. Nesse exemplo, vamos sobrescrever o conjunto de dados corrente.



Figura 6.28: Caixa de diálogo para especificarmos como o conjunto de dados será ordenado.

Ao pressionarmos o botão OK, é necessário especificar a ordem de prioridade para a ordenação (figura 6.29). Nesse exemplo, o conjunto de dados será ordenado primeiramente de acordo com os valores da variável *status*. Depois todos os registros com o mesmo valor da variável *status* serão ordenados em ordem crescente de acordo com a idade.

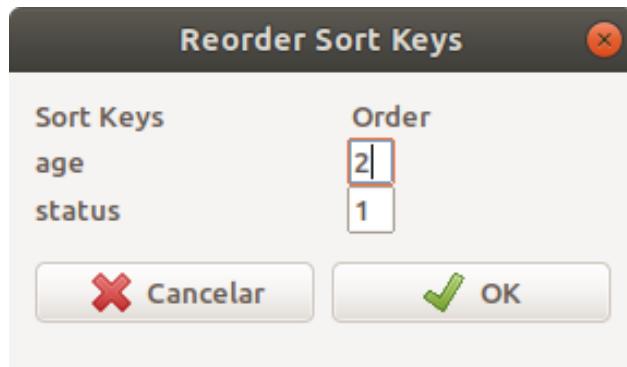


Figura 6.29: Caixa de diálogo para especificarmos a ordem de prioridade das variáveis que serão usadas para a ordenação.

Ao pressionarmos o botão OK, o comando a seguir é executado. Observem que as posições das linhas de *Melanoma* são alteradas de acordo com a ordem especificada.

```

Melanoma <- with(Melanoma, Melanoma[order(status, age, decreasing=FALSE), ])
head(Melanoma[, c("time", "status", "status_fac", "age")])

##      time status      status_fac age
## n15    469      1 óbito por melanoma 14
## n29    858      1 óbito por melanoma 16
## n37   1062      1 óbito por melanoma 19
## n22    718      1 óbito por melanoma 25
## n45   1435      1 óbito por melanoma 27
## n6     204      1 óbito por melanoma 28

```

Por meio do *R Commander*, temos que especificar a mesma direção de ordenação (crescente ou decrescente) para todas as variáveis. Por meio da linha de comando, podemos alterar o comando acima para ordenar o conjunto de dados em ordem crescente de *status* e decrescente de idade, fazendo o argumento *decreasing* ser igual a um vetor lógico onde cada elemento do vetor define se a ordenação da correspondente variável será decrescente ou não, conforme mostrado abaixo. Agora os registros estão ordenados em ordem crescente de *status* e os registros com mesmo valor de *status* estão ordenados em ordem decrescente de idade.

```

Melanoma <- with(Melanoma, Melanoma[order(status, age,
                                             decreasing=c(FALSE, TRUE),
                                             method="radix"), ])

head(Melanoma[, c("time", "status", "status_fac", "age")])

##      time status      status_fac age
## n19    629      1 óbito por melanoma 95
## n21    667      1 óbito por melanoma 89
## n72   1690      1 óbito por melanoma 83
## n51   1516      1 óbito por melanoma 80
## n149  2782      1 óbito por melanoma 78
## n7     210      1 óbito por melanoma 77

```

6.2.9 Remoção de linhas do conjunto de dados ativo

Os conteúdos desta seção e da seção 6.2.10 podem ser visualizados neste [vídeo](#).

Vamos supor que queiramos remover os registros 1 e 3 do conjunto de dados *Melanoma*. Para isso, selecionamos novamente o conjunto de dados *Melanoma* como o conjunto de dados ativo e selecionamos a opção abaixo no *R Commander*:

Dados ⇒ Conjunto de dados ativo ⇒ Remova linha(s) do conjunto de dados ativo

Na caixa de diálogo *Remova linhas do subconjunto de dados ativo* (figura 6.30), especificamos

as linhas que serão removidas e damos um nome para o novo conjunto de dados que será gerado, ou podemos sobreescriver o conjunto de dados ativo.

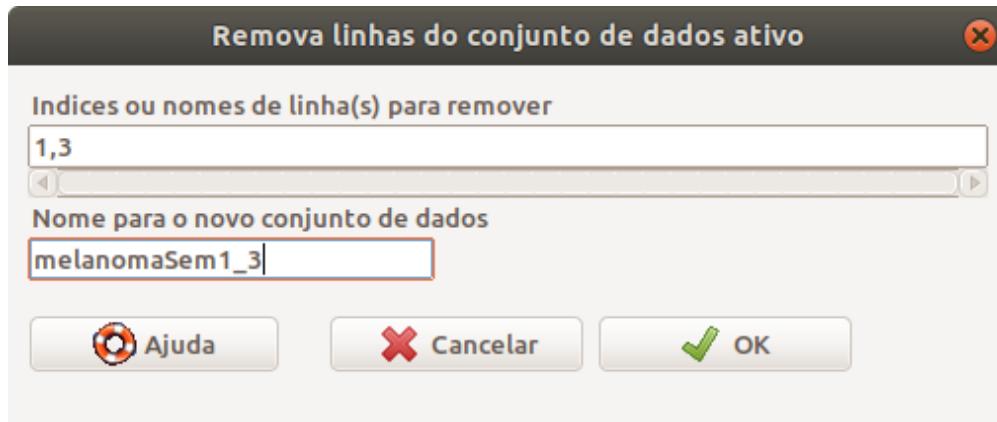


Figura 6.30: Caixa de diálogo para especificarmos as linhas que serão eliminadas do conjunto de dados.

Ao pressionarmos o botão OK, o comando abaixo é executado.

```
melanomaSem1_3 <- Melanoma[-c(1,3),]
```

Observamos abaixo que as linhas 1 e 3 do conjunto de dados *Melanoma* (correspondentes aos nomes *n19* e *n72*) não aparecem no conjunto de dados *melanomaSem1_3*.

```
head(melanomaSem1_3[, c("time", "status", "status_fac", "age")])
```

```
##      time status      status_fac age
## n21    667     1 óbito por melanoma  89
## n51   1516     1 óbito por melanoma  80
## n149  2782     1 óbito por melanoma  78
## n7    210      1 óbito por melanoma  77
## n96   1933     1 óbito por melanoma  77
## n36   1055     1 óbito por melanoma  75
```

6.2.10 Remoção de observações com valores ausentes

Voltando ao conjunto de dados *Melanoma*, suponhamos que alteramos os valores das 4 primeiras variáveis do primeiro registro para NA e o valor da 6^a variável no segundo registro para NA, conforme abaixo:

```
Melanoma[1, 1:4] = NA
Melanoma[2, 6] = NA
```

```
head(Melanoma[, c(1:7)])
```

| | time | status | sex | age | year | thickness | ulcer | |
|---------|------|--------|-----|-----|------|-----------|-------|---|
| ## n19 | NA | NA | NA | NA | 1968 | 5.48 | 1 | |
| ## n21 | 667 | | 1 | 0 | 89 | 1968 | NA | 1 |
| ## n72 | 1690 | | 1 | 1 | 83 | 1971 | 1.62 | 0 |
| ## n51 | 1516 | | 1 | 1 | 80 | 1968 | 2.58 | 1 |
| ## n149 | 2782 | | 1 | 1 | 78 | 1969 | 1.94 | 0 |
| ## n7 | 210 | | 1 | 1 | 77 | 1972 | 5.16 | 1 |

Para removermos essas linhas que contêm pelo menos um valor de uma variável igual a *NA*, selecionamos a opção abaixo no *R Commander*:

Dados ⇒ Conjunto de dados ativo ⇒ Remover observações com dados faltantes...

A caixa de diálogo *Remover Valores Faltantes* (figura 6.31) permite a seleção das variáveis que serão armazenadas no novo *data frame* (*melanoma_sem_missing*).



Figura 6.31: Caixa de diálogo para especificarmos o que será gravado após a remoção das observações com valores ausentes.

O comando executado é mostrado abaixo, onde a função *na.omit()* remove os registros com valores ausentes:

```
melanoma_sem_missing <- na.omit(Melanoma)
```

Vemos abaixo que os dois primeiros registros de *Melanoma* com valores faltantes foram removidos no conjunto de dados *melanoma_sem_missing*:

```
head(melanoma_sem_missing[, c(1:7)])
```

```
##      time status sex age year thickness ulcer
## n72    1690     1   1  83 1971       1.62     0
## n51    1516     1   1  80 1968       2.58     1
## n149   2782     1   1  78 1969       1.94     0
## n7     210      1   1  77 1972       5.16     1
## n96   1933     1   0  77 1972       1.94     0
## n36   1055     1   0  75 1967       2.58     1
```

6.2.11 Variáveis agregadas do conjunto de dados ativo

Os conteúdos desta seção e da seção 6.2.12 podem ser visualizados neste [vídeo](#).

A opção do *R Commander* abaixo permite a aplicação de funções de agregação de variáveis numéricas por categorias de uma ou mais variáveis categóricas.

Dados \Rightarrow Conjunto de dados ativo \Rightarrow Variáveis agregadas ao conjunto de dados ativo...

Para calcularmos a média das variáveis tempo de sobrevida (*time*) e espessura do tumor (*thickness*) para cada combinação dos níveis de *status_fac* e *sexo_fac*, por exemplo, completaríamos a caixa de diálogo *Observações agregadas* como mostra a figura 6.32.

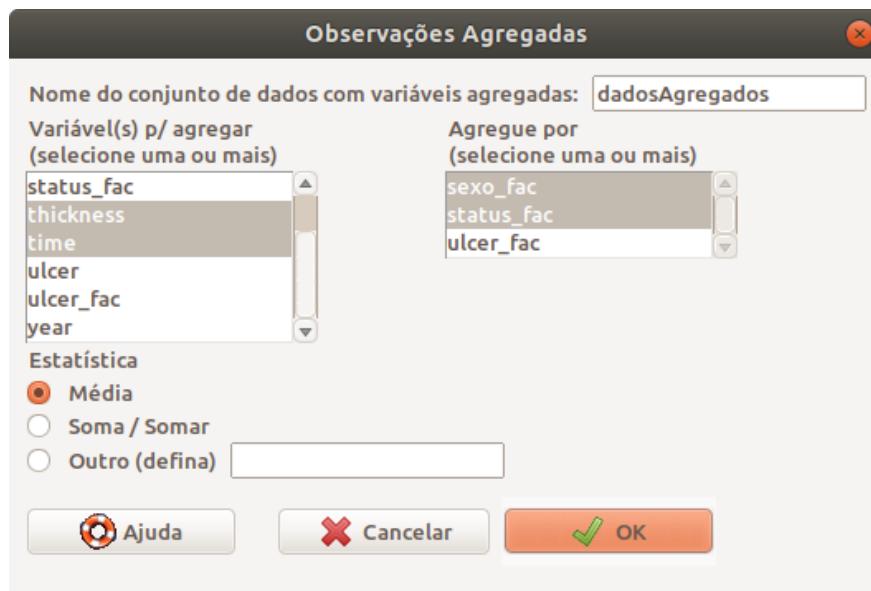


Figura 6.32: Caixa de diálogo para especificarmos como uma ou mais variáveis serão agregadas no conjunto de dados.

O comando executado é mostrado abaixo:

```
dadosAgregados<- aggregate(cbind(thickness, time) ~ sexo_fac + status_fac,  
                           data=Melanoma, FUN=mean)
```

A função *aggregate* funciona da seguinte forma: a função definida pelo argumento *FUN* é aplicada às variáveis definidas pelo primeiro argumento da função antes do símbolo *~* para cada combinação de categorias das variáveis após o símbolo *~*. As variáveis que definem as categorias são separadas pelo sinal *+*. O argumento *data* define o conjunto de dados que será utilizado. O resultado é um *data frame*, contendo as agregações e, nesse exemplo, é armazenado no objeto *dadosAgregados*.

Vejamos o *data frame* com as médias das variáveis *time* e *thickness* para cada combinação dos níveis de *status_fac* e *sexo_fac*:

```
dadosAgregados
```

```
##      sexo_fac          status_fac thickness      time  
## 1  feminino óbito por melanoma  3.620000 1408.667  
## 2 masculino óbito por melanoma  4.595000 1146.000  
## 3  feminino           vivo  2.016374 2641.011  
## 4 masculino           vivo  2.727907 2577.628  
## 5  feminino óbito por outras causas 2.601429 1225.714  
## 6 masculino óbito por outras causas 4.834286 1450.857
```

6.2.12 Empilhar variáveis no conjunto de dados ativo

A opção do *R Commander* abaixo permite a criação de um *data frame* com variáveis empilhadas uma abaixo da outra.

Dados ⇒ Conjunto de dados ativo ⇒ "Stack variables" no conjunto de dados ativo

A figura 6.33 mostra a caixa de diálogo para selecionar as variáveis a serem empilhadas. Nesse exemplo, foram selecionadas as variáveis *age* e *sex*. O *data frame* a ser criado será armazenado no objeto *dadosEmpilhados* com duas variáveis: *valor* e *variavel*.

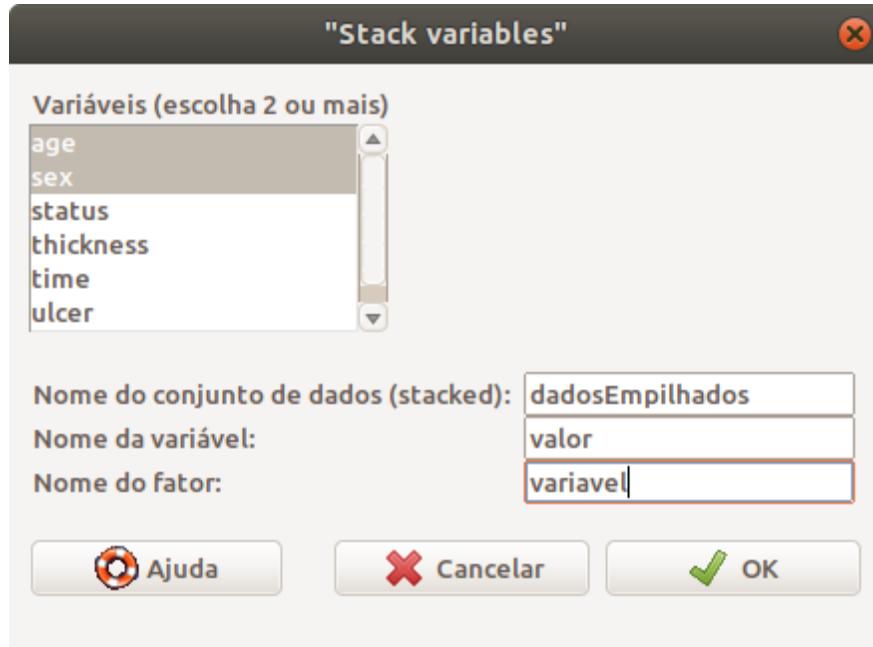


Figura 6.33: Tela para especificarmos as variáveis que serão empilhadas.

Ao pressionarmos o botão OK, os comandos abaixo serão executados:

```
dadosEmpilhados <- stack(Melanoma[, c("age", "sex")])
names(dadosEmpilhados) <- c("valor", "variavel")
```

A função *stack* realiza o empilhamento das variáveis definidas no primeiro argumento e cria um *data frame*. Se observarmos o conjunto de dados, verificaremos que o *data frame* contém o dobro de linhas de *Melanoma*, onde a primeira metade das linhas contém os valores de *age* na coluna *valor* e a palavra *age* na coluna *variavel*. As linhas restantes contêm os valores da variável *sex* na coluna *valor* e a palavra *sex* na coluna *variável*.

6.2.13 Remodelar um conjunto de dados do formato longo para o formato largo

Os conteúdos desta seção e da seção 6.2.14 podem ser visualizados neste [vídeo](#).

Vamos usar o conjunto de dados *heart.rate*, do pacote *ISwR*, que contém dados de frequência cardíaca (*hr*) de 9 pacientes com insuficiência cardíaca antes (*time* = 0) e em três instantes após a administração de enalaprilato (*time* = 30, 60 e 120 min). Nesse conjunto de dados (figura 6.34), uma variável, *subj*, identifica o indivíduo, e as variáveis *hr* e *time* indicam o valor da frequência cardíaca e o instante em que foi medida, respectivamente. Há, portanto, quatro valores de frequência cardíaca para cada indivíduo.

| | hr | subj | time |
|----|-----|------|------|
| 1 | 96 | 1 | 0 |
| 2 | 110 | 2 | 0 |
| 3 | 89 | 3 | 0 |
| 4 | 95 | 4 | 0 |
| 5 | 128 | 5 | 0 |
| 6 | 100 | 6 | 0 |
| 7 | 72 | 7 | 0 |
| 8 | 79 | 8 | 0 |
| 9 | 100 | 9 | 0 |
| 10 | 92 | 1 | 30 |
| 11 | 106 | 2 | 30 |
| 12 | 86 | 3 | 30 |
| 13 | 78 | 4 | 30 |
| 14 | 124 | 5 | 30 |
| 15 | 98 | 6 | 30 |
| 16 | 68 | 7 | 30 |
| 17 | 75 | 8 | 30 |
| 18 | 106 | 9 | 30 |
| 19 | 86 | 1 | 60 |
| 20 | 108 | 2 | 60 |
| 21 | 85 | 3 | 60 |
| 22 | 78 | 4 | 60 |
| 23 | 118 | 5 | 60 |
| 24 | 100 | 6 | 60 |
| 25 | 67 | 7 | 60 |
| 26 | 74 | 8 | 60 |
| 27 | 104 | 9 | 60 |
| 28 | 92 | 1 | 120 |
| 29 | 114 | 2 | 120 |
| 30 | 83 | 3 | 120 |
| 31 | 83 | 4 | 120 |
| 32 | 118 | 5 | 120 |
| 33 | 94 | 6 | 120 |
| 34 | 71 | 7 | 120 |
| 35 | 74 | 8 | 120 |
| 36 | 102 | 9 | 120 |

Figura 6.34: Conteúdo do conjunto de dados *heart.rate*.

Esse formato é chamado de **longo**, pois há um registro para cada valor de frequência cardíaca.

Uma outra forma de organizar esse conjunto de dados é mostrada na figura 6.35. Aqui há 5 variáveis, onde a variável *subj* identifica cada indivíduo e as variáveis *hr0*, *hr30*, *hr60* e *hr120* correspondem às frequências cardíacas de cada indivíduo nos instantes antes da administração e 30, 60 e 120 min após a administração do enalaprilato, respectivamente.

| | subj | hr0 | hr30 | hr60 | hr120 |
|---|------|-----|------|------|-------|
| 1 | 1 | 96 | 92 | 86 | 92 |
| 2 | 2 | 110 | 106 | 108 | 114 |
| 3 | 3 | 89 | 86 | 85 | 83 |
| 4 | 4 | 95 | 78 | 78 | 83 |
| 5 | 5 | 128 | 124 | 118 | 118 |
| 6 | 6 | 100 | 98 | 100 | 94 |
| 7 | 7 | 72 | 68 | 67 | 71 |
| 8 | 8 | 79 | 75 | 74 | 74 |
| 9 | 9 | 100 | 106 | 104 | 102 |

Figura 6.35: Conteúdo do conjunto de dados *heart.rate* no formato largo (*wide*).

Vamos chamar esse formato de formato largo – *wide format*. Nesse formato, há um registro para cada unidade de análise, ou paciente. Assim precisamos de 4 colunas, contendo as medidas de frequência cardíaca para cada indivíduo nos instantes 0, 20, 60 e 120 minutos.

Às vezes precisamos converter de um formato para o outro. Vamos ver como fazer isso no *R Commander*.

Vamos supor que o conjunto de dados *heart.rate* foi carregado e está como o conjunto de dados ativo no *R Commander*. Para convertermos o conjunto de dados *heart.rate* para o formato largo, usamos a seguinte opção:

Dados ⇒ Conjunto dados ativo ⇒ Reshape dataset from long to wide format...

Na tela de configuração da transformação do conjunto de dados (figura 6.36), damos um nome para o conjunto de dados que será gerado (*heart.rateWide*) e selecionamos a variável que identifica cada indivíduo (*subj*), a variável resposta (*hr* - *variables that vary by occasion*) e o fator (*time* - *within subject factors*).

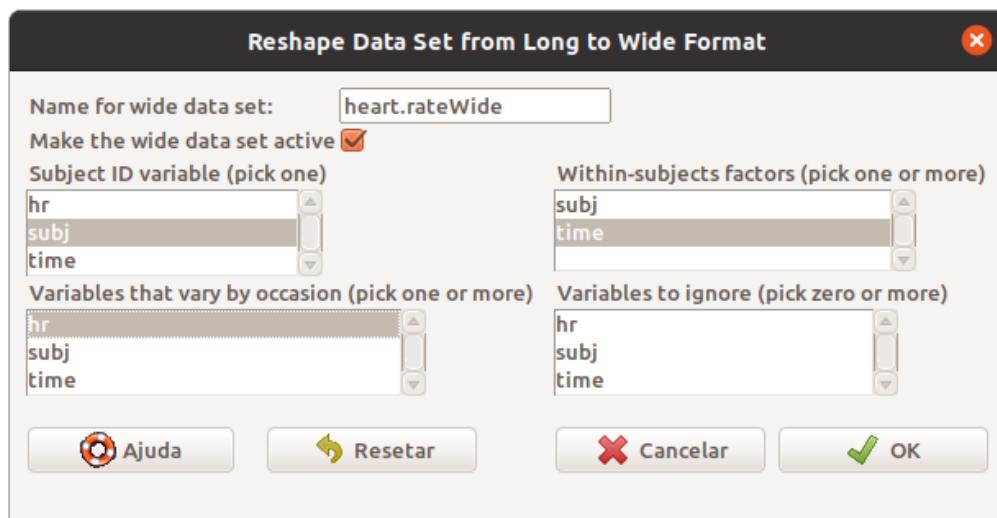


Figura 6.36: Tela para a seleção das variáveis que serão utilizadas para converter o formato longo de *heart.rate* para o formato largo.

Ao clicarmos em OK, o comando a seguir é executado e o conjunto de dados *heart.rateWide* será criado a partir de *heart.rate*, com quatro variáveis ($X0$, $X30$, $X60$, $X120$), representando as medidas de frequência cardíaca para cada indivíduo nos instantes 0, 20, 60 e 120 minutos, respectivamente.

```
heart.rateWide <- reshapeL2W(heart.rate, within="time", id="subj",
                             varying="hr")
```

Podemos, se for desejado, alterar os nomes das variáveis $X0$, $X30$, $X60$ e $X120$ por meio da seguinte opção no menu, como mostrado na seção 6.1.8:

Dados ⇒ Modificação de variáveis no conjunto de dados ⇒ Renomear variáveis...

Na tela da figura 6.37, selecionamos as variáveis que desejamos renomear. Ao clicarmos em OK, na tela seguinte (figura 6.38), damos os novos nomes das variáveis de *heart.rateWide*.



Figura 6.37: Seleção das variáveis do conjunto de dados *heart.rateWide* que serão renomeadas.

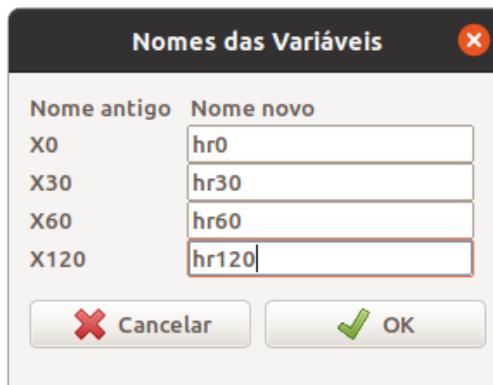


Figura 6.38: Especificação dos novos nomes das variáveis de *heart.rateWide*.

Ao clicarmos em OK, o comando a seguir é executado, atribuindo os novos nomes das variáveis de *heart.rateWide*.

```
names(heart.rateWide)[c(1,2,3,4)] <- c("hr0", "hr30", "hr60", "hr120")
```

6.2.14 Remodelar um conjunto de dados do formato largo para o formato longo

Agora, vamos fazer a conversão do formato largo para o formato longo, partindo do conjunto de dados *heart.rateWide*, gerado na seção anterior.

Para convertermos o conjunto de dados *heart.rateWide* para o formato longo, usamos a seguinte opção:

Dados \Rightarrow Conjunto dados ativo \Rightarrow Reshape dataset from wide to long format...

No conjunto de dados *heart.rateWide*, temos somente um fator que indica os diferentes instantes onde a frequência cardíaca foi medida. Assim vamos preencher os campos da aba *One Repeated-Measures Factor* na tela de configuração da transformação de *heart.rateWide* para o formato longo (figura 6.39). Vamos colocar o nome *time* para *Name for the within-subjects factor*. Teremos quatro níveis para essa variável, cada um indicando um instante de tempo em que a frequência cardíaca foi medida. A seguir, vamos selecionar *hr0*, *hr30*, *hr60* e *hr120* para os níveis 1 a 4 respectivamente.

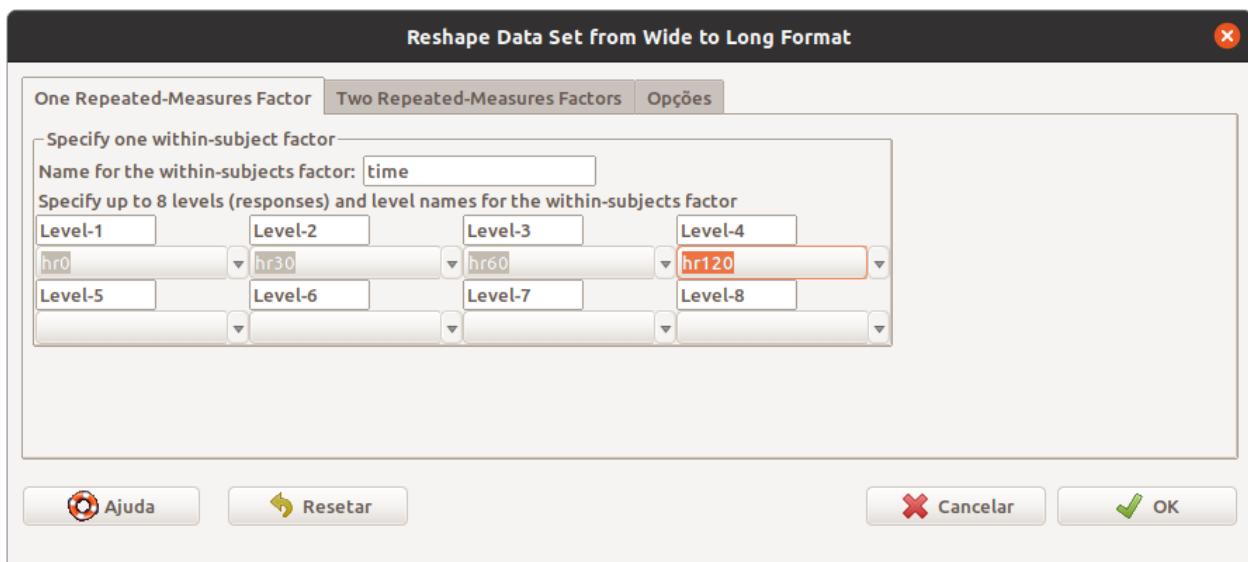


Figura 6.39: Tela para a seleção das variáveis que serão utilizadas para converter o formato largo de *heart.rate* (*heart.rateWide*) para o formato longo.

Na aba *Opções* da tela de configuração da transformação de *heart.rateWide* para o formato longo (figura 6.40), vamos dar o nome para o conjunto de dados que será criado (*heart.rateLong*), o nome da variável resposta, *hr* (*heart rate*) e o nome da variável que identifica cada indivíduo (*subj*). Também vamos tornar o conjunto de dados que será criado como conjunto de dados ativo.

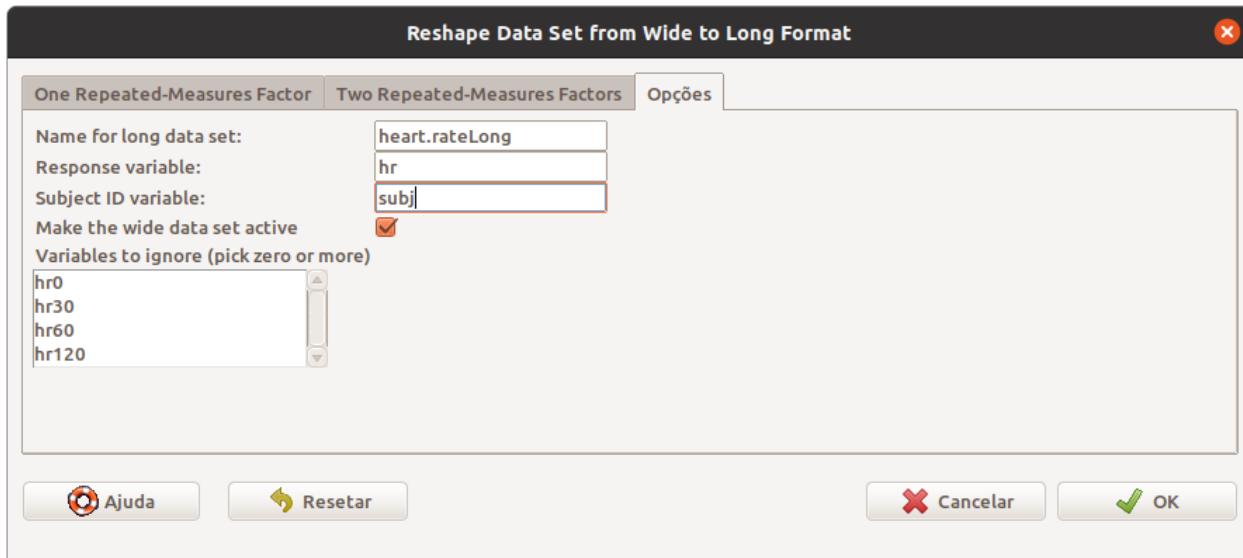


Figura 6.40: Aba *Opções* da configuração da transformação de *heart.rateWide* para o formato longo.

Ao clicarmos em OK, o comando a seguir será executado e o conjunto de dados *heart.rateLong* será criado.

```
heart.rateLong <- reshapeW2L(heart.rateWide, within="time",
                           levels=list(time=c("Level-1", "Level-2", "Level-3", "Level-4")),
                           varying=list(hr=c("hr0", "hr30", "hr60", "hr120")), id="subj")
```

Se compararmos o conjunto de dados *heart.rateLong* com o conjunto de dados *heart.rate* original (figura 6.41), vemos que o instante 0 de *heart.rate* é indicado pelo valor *Level-1* em *heart.rateLong*, o instante 30 é indicado pelo valor *Level-2*, o instante 60 por *Level-3* e o instante 120 por *Level-4*. A ordem das linhas é diferente, mas há uma correspondência entre as medidas de frequência cardíaca de cada indivíduo em cada instante de tempo.

heart.rate

| hr | subj | time |
|-----|------|------|
| 96 | 1 | 0 |
| 118 | 2 | 0 |
| 89 | 3 | 0 |
| 95 | 4 | 0 |
| 128 | 5 | 0 |
| 100 | 6 | 0 |
| 72 | 7 | 0 |
| 79 | 8 | 0 |
| 100 | 9 | 0 |
| 92 | 1 | 30 |
| 106 | 2 | 30 |
| 86 | 3 | 30 |
| 78 | 4 | 30 |
| 124 | 5 | 30 |
| 98 | 6 | 30 |
| 68 | 7 | 30 |
| 75 | 8 | 30 |
| 106 | 9 | 30 |
| 86 | 1 | 60 |
| 108 | 2 | 60 |
| 85 | 3 | 60 |
| 78 | 4 | 60 |
| 118 | 5 | 60 |
| 100 | 6 | 60 |
| 67 | 7 | 60 |
| 74 | 8 | 60 |
| 104 | 9 | 60 |
| 92 | 1 | 120 |
| 114 | 2 | 120 |
| 83 | 3 | 120 |
| 83 | 4 | 120 |
| 118 | 5 | 120 |
| 94 | 6 | 120 |
| 71 | 7 | 120 |
| 74 | 8 | 120 |
| 102 | 9 | 120 |

heart.rateLong

| subj | time | hr |
|------|------|-------------|
| 1.1 | 1 | Level-1 96 |
| 1.2 | 1 | Level-2 92 |
| 1.3 | 1 | Level-3 86 |
| 1.4 | 1 | Level-4 92 |
| 2.1 | 2 | Level-1 110 |
| 2.2 | 2 | Level-2 106 |
| 2.3 | 2 | Level-3 108 |
| 2.4 | 2 | Level-4 114 |
| 3.1 | 3 | Level-1 89 |
| 3.2 | 3 | Level-2 86 |
| 3.3 | 3 | Level-3 85 |
| 3.4 | 3 | Level-4 83 |
| 4.1 | 4 | Level-1 95 |
| 4.2 | 4 | Level-2 78 |
| 4.3 | 4 | Level-3 78 |
| 4.4 | 4 | Level-4 83 |
| 5.1 | 5 | Level-1 128 |
| 5.2 | 5 | Level-2 124 |
| 5.3 | 5 | Level-3 118 |
| 5.4 | 5 | Level-4 118 |
| 6.1 | 6 | Level-1 100 |
| 6.2 | 6 | Level-2 98 |
| 6.3 | 6 | Level-3 100 |
| 6.4 | 6 | Level-4 94 |
| 7.1 | 7 | Level-1 72 |
| 7.2 | 7 | Level-2 68 |
| 7.3 | 7 | Level-3 67 |
| 7.4 | 7 | Level-4 71 |
| 8.1 | 8 | Level-1 79 |
| 8.2 | 8 | Level-2 75 |
| 8.3 | 8 | Level-3 74 |
| 8.4 | 8 | Level-4 74 |
| 9.1 | 9 | Level-1 100 |
| 9.2 | 9 | Level-2 106 |
| 9.3 | 9 | Level-3 104 |
| 9.4 | 9 | Level-4 102 |

Figura 6.41: Conjunto de dados *heart.rate* e *heart.rateLong* lado a lado.

Podemos alterar os valores da variável *time* no conjunto *heart.rateLong* para ficarem iguais aos correspondentes valores da variável *time* em *heart.rate*, recodificando a variável *time* no conjunto de dados *heart.rateLong*. Para isso, utilizamos o seguinte item de menu (seção 6.1.1):

Dados ⇒ Modificação de variáveis no conjunto de dados ⇒ Recodificar variáveis...

Na tela para recodificação da variável (figura 6.42), vamos selecionar a variável *time*, vamos sobrescrever a variável, colocando o valor *time* para novo nome da variável, vamos manter a variável como fator e vamos especificar a recodificação como mostrada na figura.



Figura 6.42: Renomeação das categorias da variável *time* por meio da função *Recode*.

Ao clicarmos em **OK**, o programa pergunta se desejamos sobrescrever a variável *time*. Vamos responder *Sim*. Agora os dois conjuntos de dados, *heart.rate* e *heart.rateLong*, são essencialmente idênticos.

6.2.15 Converter todas as variáveis do tipo *character* para o tipo *factor*

A opção do menu:

Dados ⇒ Conjunto de dados ativo ⇒ Convert all character variables to factors

utiliza a função *strings2factors*, do pacote *car*, para converter variáveis do conjunto de dados ativo do tipo *character* para fator. Na forma como ela é executada no *R Commander*, todas as variáveis do tipo *character*, com exceção daquelas cujos valores são únicos (ou seja, todos os valores da variável são diferentes uns dos outros), são convertidas para fatores. As variáveis do tipo *character* cujos valores são todos diferentes uns dos outros são tipicamente identificadores de casos e não são variáveis categóricas.

6.3 Exercício

- 1) Com o conjunto de dados *VA* do pacote *MASS* (GPL-2 | GPL-3), faça as seguintes atividades.
 - a) Verifique a ajuda para o conjunto de dados. Observe os códigos para as seguintes variáveis:
status: 0=*censored*; 1=*dead*
treat: 0=*standard*; 1=*test*
prior: 0=*no*; 10=*yes*
cell: 1=*squamous*; 2=*smallcell*; 3=*adeno*; 4=*large*
 - b) Carregue o conjunto de dados.
 - c) Liste os 10 primeiros registros do conjunto de dados.
 - d) Verifique a classe das variáveis *status*, *treat*, *cell* e *prior*.
 - e) Converta a variável *status* para fator, atribuindo nomes descritivos aos seus valores.
 - f) Recodifique as variáveis *cell*, *treat* e *prior*, atribuindo nomes descritivos aos seus valores;
 - g) Crie uma nova variável cujos valores sejam o logaritmo da variável *stime*. Use a função *log*.
 - h) Crie uma variável que agrupe os valores de idade nas seguintes faixas: <31, [32-40], [41-60], >60.
 - i) Padronize as variáveis *age*, *diag.time* e *stime*.
 - j) Salve os dados em um arquivo do R.
 - k) Verifique a classe da variável *cell*.
 - l) Crie um subconjunto de *VA* com um nome apropriado que contenha somente os registros cujo valor de *cell* sejam dos tipos *adeno* ou *large*.
 - m) Remova os níveis não usados da variável *cell* no subconjunto de dados criado.
 - n) Gere um relatório no *R Markdown* com as operações acima.

Capítulo 7

Tabelas de frequências

7.1 Introdução

Os conteúdos desta seção e da seção 7.2.1 podem ser visualizados neste [vídeo](#).

Neste capítulo, serão apresentadas diversas formas para apresentar em tabelas a frequência ou porcentagem das categorias de uma variável categórica ou de combinação de categorias de variáveis categóricas em um conjunto de dados. As tabelas assim obtidas são também chamadas de **tabelas de contingência**.

A figura 7.1 mostra a tabela 2 do estudo de Barata e Valete (Barata and Valete, 2018), intitulado “Perfil clínico-epidemiológico de 106 pacientes pediátricos portadores de urolitíase no Rio de Janeiro”. Essa tabela mostra a frequência de ocorrência das categorias das variáveis sexo, cor da pele, idade no início dos sintomas e idade no diagnóstico.

A **frequência de uma categoria** de uma variável categórica em um conjunto de dados é o número de observações daquela categoria da variável no conjunto de dados. Na figura 7.1, vemos que a frequência de mulheres no estudo é 52 e de homens, 54. Em relação à cor da pele, 67 pessoas eram brancas, 30 pardas, 8 negras e 1 amarela.

Se dividirmos a frequência de uma categoria de uma variável pelo número total de observações, obtemos a **proporção da respectiva categoria** da variável no conjunto de dados. Multiplicando essa proporção por 100, obtemos então a **porcentagem da categoria da variável** no conjunto de dados. Na figura 7.1, vemos que a porcentagem de mulheres no estudo é 49,1% e de homens, 50,9%.

Tabela 2 Características demográficas dos pacientes em seguimento no Hospital Federal dos Servidores do Estado, entre janeiro de 2012 e dezembro de 2014.

| | n | % |
|--|----|------|
| Gênero | | |
| Feminino | 52 | 49,1 |
| Masculino | 54 | 50,9 |
| Cor da pele | | |
| Branca | 67 | 63,2 |
| Parda | 30 | 28,3 |
| Amarela | 1 | 0,9 |
| Negra | 8 | 7,6 |
| Idade no início dos sintomas^a (anos) | | |
| <5 | 17 | 16,0 |
| ≥5 a≤10 | 54 | 50,9 |
| >10 a≤18 | 35 | 33,0 |
| Idade no diagnóstico^b (anos) | | |
| <5 | 8 | 7,5 |
| ≥5 a≤10 | 52 | 49,1 |
| >10 a≤18 | 46 | 43,4 |

Total da amostra: n=106; ^amédia 8,9±3,8; ^bmédia 9,9±3,6.

Frequência

Porcentagem

Figura 7.1: Características demográficas dos pacientes pediátricos portadores de urolitíase no Rio de Janeiro em seguimento no Hospital Federal dos Servidores do Estado. Fonte: (Barata and Valete, 2018) (CC BY).

O conjunto dos pares formados pelas categorias de uma variável em um conjunto de dados, juntamente com as suas respectivas frequências, é chamado de **distribuição de frequências da variável**.

A figura 7.2 mostra a tabela 2 do estudo de Vanin et al. (Vanin et al., 2019), intitulado “Fatores de risco materno-fetais associados à prematuridade tardia”. Essa tabela mostra, entre outras, a distribuição de frequência conjunta das variáveis *sexo* e *maturidade*. A variável *maturidade* possui dois níveis ou categorias: *RNPT* - recém-nascido prematuro tardio e *RNT* - recém-nascido a termo. Para cada categoria da variável *maturidade*, a tabela mostra as frequências de cada categoria da variável *sexo* e entre parênteses a porcentagem da respectiva categoria de *sexo* em relação à frequência total do nível de maturidade correspondente. Por exemplo, entre os *recém-nascidos a termo*, a frequência do sexo feminino é 124, correspondendo a 44,1% do total de crianças *recém-nascidas a termo* (124 de 281).

Tabela 2 Análise comparativa entre as características fetais e neonatais com o nascimento prematuro tardio e a termo.

| | RNPT | | RNT | | p-valor |
|------------------------|-------------------|--------|-----|--------|---------|
| | n | (%) | n | (%) | |
| Pequeno para IG | | | | | |
| Sim | 34 | (24,1) | 22 | (7,8) | <0,001 |
| Não | 107 | (75,9) | 260 | (92,2) | |
| Grande para IG | | | | | |
| Sim | 5 | (3,5) | 36 | (12,8) | <0,001 |
| Não | 136 | (96,5) | 246 | (87,2) | |
| Sexo | Frequência | | | | |
| Masculino | 64 | (45,4) | 157 | (55,9) | 0,042 |
| Feminino | 77 | (54,6) | 124 | (44,1) | |

RNPT: recém-nascido prematuro tardio; RNT: recém-nascido a termo; IG: idade gestacional.

Porcentagem em relação ao total da coluna

Figura 7.2: Tabela 2 do estudo de Vanin et al., mostrando a frequência conjunta de diversas variáveis e o nível de maturidade das crianças recém-nascidas do estudo. Fonte: (Vanin et al., 2019) (CC BY).

Para mostrar como obtemos distribuições de frequências de variáveis ou combinação de variáveis em um conjunto de dados no ambiente R, vamos utilizar o conjunto de dados *stroke* do pacote *ISwR* (GPL-2 | GPL-3). Esse conjunto de dados contém todos os casos de AVC (acidente vascular cerebral) em Tartu, Estonia, durante 1991-1993, com acompanhamento até 1º de janeiro de 1996.

Caso o pacote *ISwR* não esteja instalado, é preciso instalá-lo (veja a seção 4.7).

Antes de abrirmos o conjunto de dados *stroke*, é preciso carregar o pacote *ISwR*. Na sequência deste capítulo, utilizaremos o *R Commander* carregado a partir do R, e não a partir do *RStudio*.

Para carregarmos o pacote *ISwR* a partir do *R Commander*, digitamos *library(ISwR)* na área de *script* do *R Commander* e, **com o cursor na linha do comando**, clicamos no botão *Submeter* (figura 7.3).



Figura 7.3: Tela do *R commander*, com a digitação da função *library(ISwR)* na área de *Script*.

Ao submetermos a função, ela aparece na área de output do *R Commander* (figura 7.4) e, se houver alguma coisa errada, uma mensagem de erro apareceria na área de mensagens do *R Commander*.

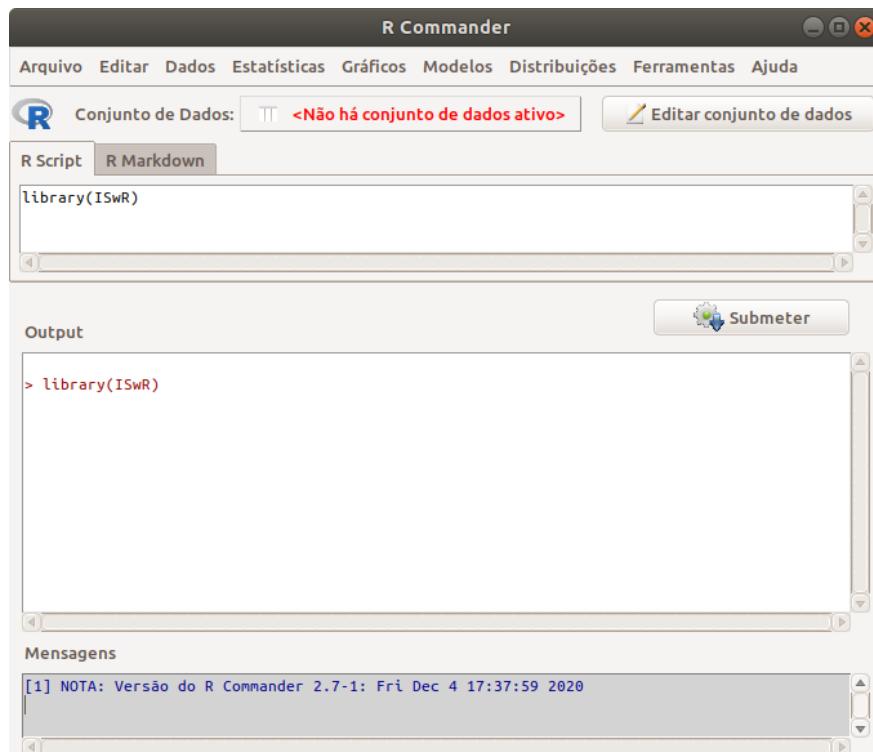


Figura 7.4: Tela do *R Commander* após a execução da função *library* conforme mostrado na figura 7.3.

Alternativamente o pacote *ISwR* poderia ser carregado por meio da opção de menu do *R Commander*:

Ferramentas ⇒ Carregar pacote(s)...

Para carregarmos o conjunto de dados *stroke*, selecionamos a opção abaixo no *R Commander* (figura 7.5):

Dados ⇒ Conjunto de dados em pacotes ⇒ Ler dados de pacote 'attachado'

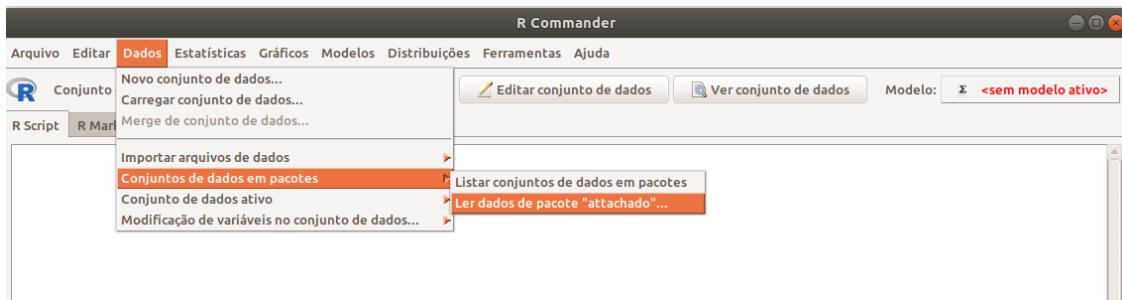


Figura 7.5: Menu do *R Commander* com a opção para carregar arquivos de pacotes do R.

Para vermos a lista dos conjuntos de dados em *ISwR*, damos um duplo clique nesse pacote e uma lista de conjuntos de dados será mostrada à direita (figura 7.6). Rolamos essa lista e clicamos no conjunto *stroke* para selecioná-lo. Para conhecermos a estrutura desse conjunto de dados, clicamos no botão *Ajuda para o conjunto de dados selecionado* (seta verde na figura). Uma descrição desse conjunto de dados será exibida no seu navegador padrão (figura 7.7). Ao clicarmos no botão OK na figura 7.6, após termos selecionado *stroke*, esse conjunto de dados será carregado no *R commander* (figura 7.8).

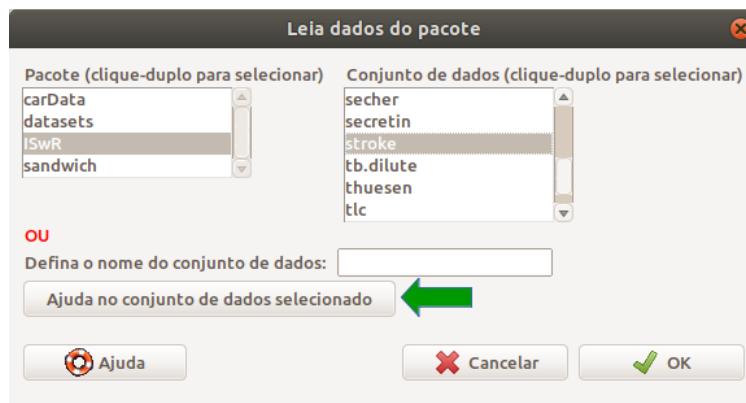


Figura 7.6: Visualizando a lista de conjuntos de dados do pacote *ISwR* e selecionando o conjunto *stroke*.

```

stroke {ISwR}

Description
All cases of stroke in Tartu, Estonia, during the period 1991-1993, with follow-up until January 1, 1996.

Usage
stroke

Format
A data frame with 829 observations on the following 10 variables.

sex
  a factor with levels Female and Male.

died
  a Date, date of death.

dstr
  a Date, date of stroke.

age
  a numeric vector, age at stroke.

dgn
  a factor, diagnosis, with levels ICH (intracranial haemorrhage), UD (unidentified), INF (infarction, ischaemic), SAH (subarachnoid haemorrhage).

coma
  a factor with levels No and Yes, indicating whether patient was in coma after the stroke.

diab
  a factor with levels No and Yes, history of diabetes.

mif
  a factor with levels No and Yes, history of myocardial infarction.

han
  a factor with levels No and Yes, history of hypertension.

obsmonths
  a numeric vector, observation times in months (set to 0.1 for patients dying on the same day as the stroke).

dead
  a logical vector, whether patient died during the study.

Source
Original data.

References
J. Korp, M. Roose, and A.E. Kaasik (1997). Stroke Registry of Tartu, Estonia, from 1991 through 1993. Cerebrovascular Disorders 7:154-162.

[Package ISwR version 2.0-8 Index]

```

Figura 7.7: Texto com a descrição do conjunto de dados *stroke*.

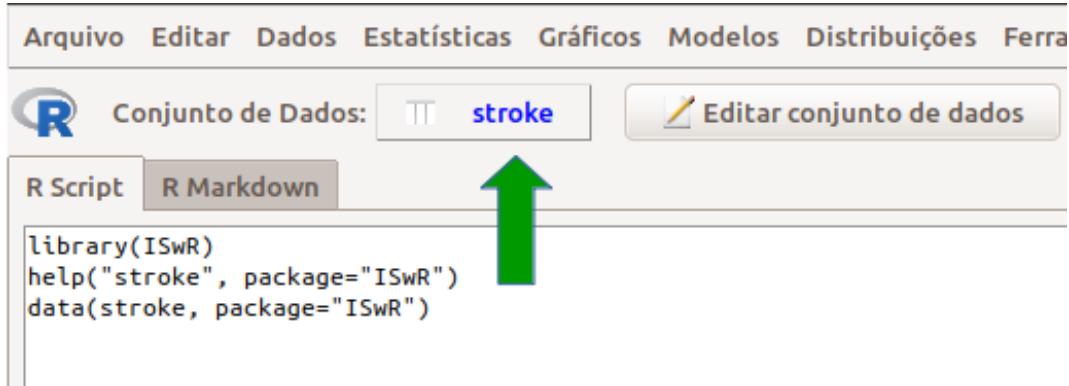


Figura 7.8: Tela do *R commander* após o carregamento do conjunto de dados *stroke*. Observem a função que foi executada – *data(stroke, package="ISwR")* – e o nome do conjunto selecionado (seta verde).

Observem as funções que foram executadas no *R Commander*:

```
library(ISwR)
help("stroke", package="ISwR")
data(stroke, package= "ISwR")
```

A função *help* mostra uma ajuda sobre o conjunto de dados *stroke* do pacote *ISwR*.

A função *data(stroke, package="ISwR")* carrega o conjunto de dados *stroke* que passa a ser o conjunto de dados ativo no *R Commander*. Observem o nome dele ao lado do rótulo conjunto de dados (seta verde na figura 7.8). Esse objeto pode ser acessado pelo próprio nome (*stroke* nesse caso).

Na área de mensagens do *R Commander*, aparece a seguinte mensagem abaixo do comando, indicando o número de registros e de variáveis no conjunto de dados *stroke*:

NOTA: Os dados *stroke* tem 829 linhas e 11 colunas.

Para visualizarmos o conteúdo do conjunto de dados *stroke*, clicamos no botão *Ver conjunto de dados* (seta verde na figura 7.9).



Figura 7.9: Botão do *R Commander* (seta verde) para exibir o conteúdo do conjunto de dados ativo.

A figura 7.10 mostra as observações do conjunto de dados *stroke*.

| | stroke | | | | | | | | | | | |
|----|--------|------------|------------|-----|-----|------|------|------|-----|-------|-------------|--|
| | sex | died | dstr | age | dgn | coma | diab | minf | han | dead | obsmonths | |
| 1 | Male | 1991-01-07 | 1991-01-02 | 76 | INF | No | No | Yes | No | TRUE | 0.16339869 | |
| 2 | Male | <NA> | 1991-01-03 | 58 | INF | No | No | No | No | FALSE | 59.60784314 | |
| 3 | Male | 1991-06-02 | 1991-01-08 | 74 | INF | No | No | Yes | Yes | TRUE | 4.73856209 | |
| 4 | Female | 1991-01-13 | 1991-01-11 | 77 | ICH | No | Yes | No | Yes | TRUE | 0.06535948 | |
| 5 | Female | <NA> | 1991-01-13 | 76 | INF | No | Yes | No | Yes | FALSE | 59.28104575 | |
| 6 | Male | 1991-01-13 | 1991-01-13 | 48 | ICH | Yes | No | No | Yes | TRUE | 0.10000000 | |
| 7 | Female | 1993-12-01 | 1991-01-14 | 81 | INF | No | No | No | Yes | TRUE | 34.37908497 | |
| 8 | Male | 1991-12-12 | 1991-01-14 | 53 | INF | No | No | Yes | Yes | TRUE | 10.84967320 | |
| 9 | Female | <NA> | 1991-01-15 | 73 | ID | No | No | No | Yes | FALSE | 59.21568627 | |
| 10 | Female | 1993-11-10 | 1991-01-15 | 69 | INF | No | No | No | Yes | TRUE | 33.66013072 | |
| 11 | Female | 1991-01-20 | 1991-01-16 | 86 | ID | No | No | No | No | TRUE | 0.13071895 | |
| 12 | Female | <NA> | 1991-01-16 | 79 | INF | No | No | No | Yes | FALSE | 59.18300654 | |
| 13 | Male | 1994-01-26 | 1991-01-21 | 69 | INF | No | No | No | No | TRUE | 35.98039216 | |
| 14 | Female | <NA> | 1991-01-22 | 58 | INF | No | No | No | Yes | FALSE | 58.98692810 | |
| 15 | Female | <NA> | 1991-01-23 | 71 | INF | No | No | No | Yes | FALSE | 58.95424837 | |
| 16 | Female | 1991-02-04 | 1991-01-26 | 84 | INF | No | No | No | Yes | TRUE | 0.29411765 | |
| 17 | Male | 1995-07-27 | 1991-01-27 | 63 | INF | No | No | No | No | TRUE | 53.66013072 | |
| 18 | Female | 1991-02-11 | 1991-01-28 | 85 | ID | No | No | No | No | TRUE | 0.45751634 | |
| 19 | Female | 1991-01-29 | 1991-01-28 | 81 | ICH | No | Yes | Yes | Yes | TRUE | 0.03267974 | |
| 20 | Female | 1991-02-08 | 1991-01-29 | 77 | INF | Yes | No | No | Yes | TRUE | 0.32679739 | |
| 21 | Male | <NA> | 1991-01-30 | 62 | INF | No | No | No | Yes | FALSE | 58.72549020 | |
| 22 | Female | 1991-04-20 | 1991-01-31 | 84 | INF | No | No | No | Yes | TRUE | 2.58169935 | |
| 23 | Female | 1991-07-04 | 1991-02-03 | 77 | INF | No | No | No | Yes | TRUE | 4.93464052 | |
| 24 | Male | 1995-05-19 | 1991-02-03 | 61 | ICH | No | No | No | Yes | TRUE | 51.17647059 | |
| 25 | Female | <NA> | 1991-02-04 | 79 | INF | No | No | No | Yes | FALSE | 58.56209150 | |
| 26 | Male | 1994-06-08 | 1991-02-04 | 71 | INF | No | No | Yes | Yes | TRUE | 39.86928105 | |
| 27 | Female | 1991-10-25 | 1991-02-06 | 84 | ID | No | No | No | Yes | TRUE | 8.52941176 | |
| 28 | Male | 1991-02-14 | 1991-02-07 | 86 | ID | No | No | No | No | TRUE | 0.22875817 | |
| 29 | Male | <NA> | 1991-02-10 | 62 | INF | No | No | Yes | Yes | FALSE | 58.36601307 | |
| 30 | Male | 1991-02-23 | 1991-02-11 | 68 | INF | No | No | Yes | No | TRUE | 0.39215686 | |

Figura 7.10: Conteúdo do conjunto de dados *stroke*.

Para mostrar como obter distribuições de frequências de variáveis categóricas e gerar tabulações de dados com combinações de variáveis categóricas, vamos trabalhar com as seguintes variáveis do conjunto de dados *stroke* (figura 7.7):

- *dead*: variável categórica binária, com os valores *TRUE*, se o paciente faleceu, e *FALSE*, se o paciente continuava vivo ao final do estudo;
- *dgn*: diagnóstico do paciente, variável categórica nominal, com as categorias ICH (hemorragia intracranial), ID (não identificado), INF (infarto), SAH (hemorragia subaracnóide);
- *minf*: história de infarto do miocárdio, variável categórica binária, com os valores *No* e *Yes*;
- *diab*: história de diabetes, variável categórica binária, com os valores *No* e *Yes*.

7.2 Tabelas de frequências no conjunto de dados *stroke*

7.2.1 Uma única variável categórica

Para construirmos uma tabela que mostra a frequência ou porcentagem de cada categoria de uma variável categórica no *R Commander*, usamos a opção:

Estatísticas ⇒ Resumos ⇒ Distribuições de frequência...

Na tela dessa opção (figura 7.11), selecionamos uma ou mais variáveis para as quais desejamos a distribuição de frequências. Serão montadas tabelas para cada variável separadamente. Nesse exemplo, selecionamos a variável *dead*.



Figura 7.11: Caixa de diálogo para a seleção das variáveis categóricas cujas distribuições marginais serão exibidas.

Ao clicarmos em OK, os comandos abaixo serão executados, com os resultados mostrados logo a seguir.

```
local({  
  .Table <- with(stroke, table(dead))  
  cat("\ncounts:\n")  
  print(.Table)  
  cat("\npercentages:\n")  
  print(round(100*.Table/sum(.Table), 2))  
}  
  
##  
## counts:  
## dead  
## FALSE TRUE  
## 344 485  
##  
## percentages:
```

```
## dead
## FALSE TRUE
## 41.5 58.5
```

São mostradas duas tabelas, uma com a frequência de cada categoria da variável *dead* (TRUE, FALSE) no conjunto de dados *stroke*, e outra com as porcentagens de cada categoria.

No primeiro comando da sequência acima, mostrado novamente abaixo, a função *with* possui dois argumentos: o primeiro indica o conjunto de dados que será utilizado (*stroke*), e o segundo argumento indica a função que será executada com variáveis do conjunto de dados especificado no primeiro argumento. Nesse caso, é executada a função *table* para gerar uma tabela de frequência da variável entre parênteses.

```
.Table <- with(stroke, table(dead))
```

O resultado da execução do comando é armazenado no objeto *.Table*.

O comando seguinte *cat* imprime uma linha na tela para informar que a tabela mostrada a seguir é relativa à contagem (*counts:*) ou frequência das categorias da variável. A barra invertida (“\”), seguida da letra “n” antes e depois da expressão “counts:”, indica que uma linha deve ser pulada antes e depois de escrever a expressão “counts:”.

```
cat("\ncounts:\n")
```

O comando seguinte, *print*, mostra então a tabela de frequências da variável *dead*.

```
print(.Table)
```

Vemos que 485 pacientes morreram e 344 continuavam vivos até o final do estudo. Esses números são obtidos, contando-se o número de observações no conjunto de dados *stroke* cujos valores da variável *dead* são *TRUE* ou *FALSE*, respectivamente.

Os dois comandos seguintes imprimem a expressão “percentages:” na tela e, em seguida, a tabela com as porcentagens de cada categoria da variável *dead*.

```
cat("\npercentages:\n")
print(round(100*.Table/sum(.Table), 2))
```

Vamos entender como as porcentagens foram calculadas. A função *sum* aplicada ao objeto *.Table* irá somar as frequências das categorias da variável *dead* ($344 + 485 = 829$). A expressão *.Table/sum(.Table)* então divide a frequência de cada categoria da variável *dead* pela soma das frequências, resultando nas proporções de cada categoria ($344/829=0,41496$; $485/829=0,58504$). Esses dois valores são então multiplicados por 100 para fornecer as porcentagens de cada categoria (41,496; 58,504). A função *round* irá arredondar esses valores com duas casas decimais.

7.2.2 Tabelas de frequências para duas variáveis categóricas

Os conteúdos desta seção e suas subseções, com exceção da subseção 7.2.5.1 podem ser visualizados neste [vídeo](#).

Para gerarmos tabelas de frequências ou porcentagens para combinação das categorias de duas variáveis categóricas no *R Commander*, usamos a opção:

Estatísticas ⇒ Tabelas de contingência ⇒ Tabela de dupla entrada...

Esse tipo de tabela é chamada também de **tabela de dupla entrada** (por envolver duas variáveis).

Na aba *Dados* da tela dessa opção (figura 7.12), selecionamos as duas variáveis (uma cujas categorias irão aparecer nas linhas e outra cujas categorias irão aparecer nas colunas da tabela), para as quais desejamos a distribuição conjunta de frequências. Nesse exemplo, selecionamos a variável *dgn* (diagnóstico) para as linhas e *dead* para as colunas.

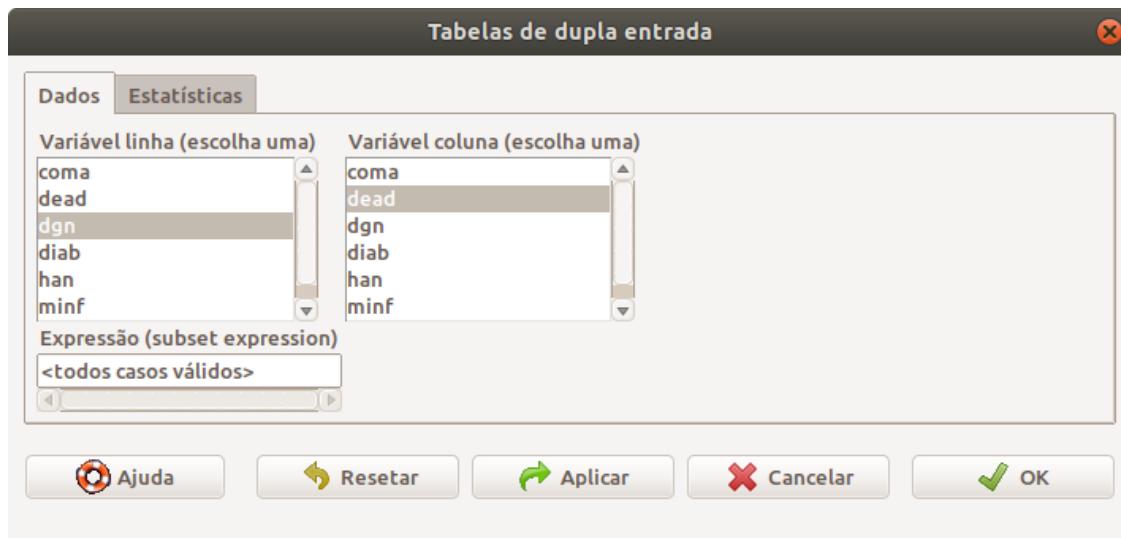


Figura 7.12: Tela para a seleção das variáveis categóricas que comporão a tabela de dupla entrada.

7.2.3 Obtendo os percentuais de cada célula em relação ao total da linha correspondente

Na aba *Estatísticas* (figura 7.13), vamos marcar a opção *percentual nas linhas* e desmarcar a opção *Teste de independência de Qui-Quadrado*. Nesse caso, a tabela de dupla entrada irá mostrar os percentuais de cada célula da tabela em relação ao total da linha correspondente.

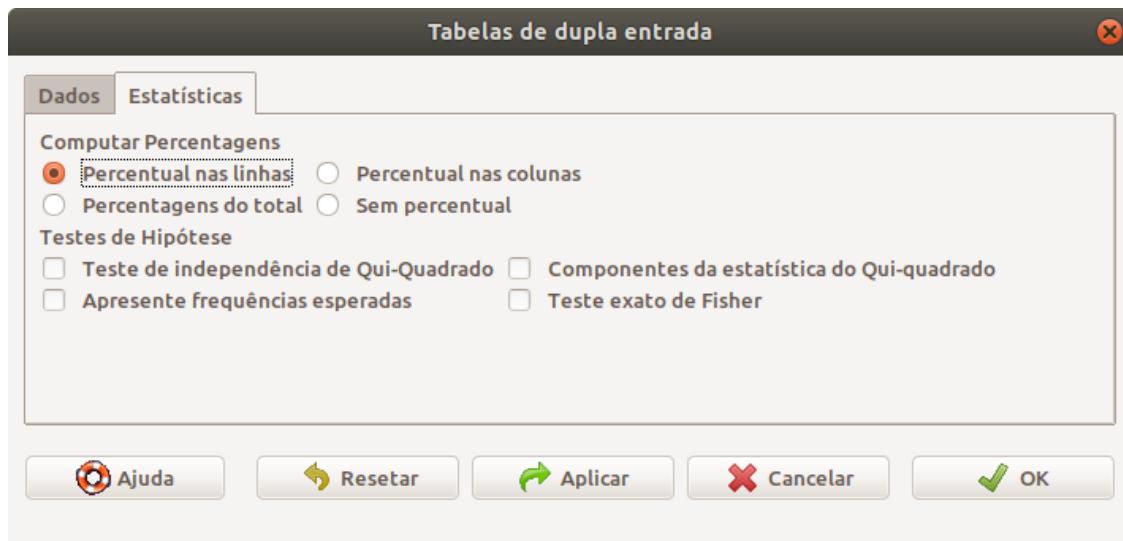


Figura 7.13: Tela para especificar que a tabela de dupla entrada irá mostrar os percentuais de cada célula em relação ao total da linha correspondente.

Ao clicarmos em OK, os comandos abaixo serão executados, com os resultados mostrados logo a seguir.

```
local({
  .Table <- xtabs(~dgn+dead, data=stroke)
  cat("\nFrequency table:\n")
  print(.Table)
  cat("\nRow percentages:\n")
  print(rowPercents(.Table))
})

## 
## Frequency table:
##       dead
## dgn  FALSE TRUE
##   ICH    25   54
##   ID     54  148
##   INF   239  262
##   SAH    26   21
##
## Row percentages:
##       dead
## dgn  FALSE TRUE Total Count
##   ICH  31.6 68.4    100    79
##   ID   26.7 73.3    100   202
##   INF  47.7 52.3    100   501
##   SAH  55.3 44.7    100    47
```

Dessa vez, a tabela foi gerada por meio da função *xtabs*. Essa função utiliza uma fórmula para especificar as variáveis que comporão a tabela. Nessa fórmula, as variáveis são especificadas após o sinal \sim , sendo a primeira variável aquela cujas categorias aparecerão nas linhas e a segunda variável aquela cujas categorias aparecerão nas colunas. As variáveis são separadas pelo sinal “+”. O argumento *data* especifica o conjunto de dados que contém as variáveis descritas na fórmula.

A função *rowPercents*, do pacote *RcmdrMisc*, gera os percentuais de cada célula em relação ao total da linha correspondente.

Assim a tabela de frequência mostra que, dos 79 pacientes cujo diagnóstico era “ICH”, 54 morreram e 25 não morreram no período de tempo considerado. Esses valores são obtidos, contando-se no conjunto de dados quantos pacientes que possuem o diagnóstico de hemorragia intracranial vieram ou não a óbito, respectivamente. A tabela com as porcentagens nas linhas mostra que, dos 79 pacientes com diagnóstico “ICH”, 31,6% (25) não morreram e 68,4% (54) morreram. Interpretação análoga se aplica às demais linhas da tabela.

7.2.4 Obtendo os percentuais de cada célula em relação ao total da coluna correspondente

Se, na aba *Estatísticas* para gerar uma tabela de dupla entrada, marcarmos a opção *percentual nas colunas* (figura 7.14), a tabela de dupla entrada irá mostrar os percentuais de cada célula em relação ao total da coluna correspondente.

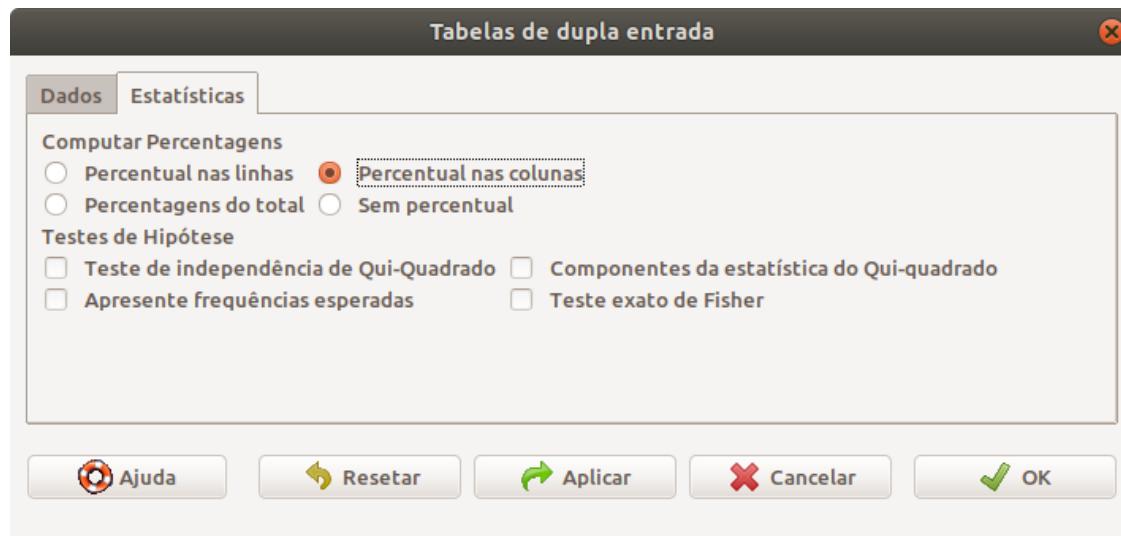


Figura 7.14: Tela para especificar que a tabela de dupla entrada irá mostrar os percentuais de cada célula em relação ao total da coluna correspondente à célula.

Ao clicarmos em OK, os comandos a seguir serão executados, com os resultados mostrados logo após.

```

local({
  .Table <- xtabs(~dgn+dead, data=stroke)
  cat("\nFrequency table:\n")
  print(.Table)
  cat("\nColumn percentages:\n")
  print(colPercents(.Table))
})

## Frequency table:
##      dead
## dgn  FALSE TRUE
##   ICH    25   54
##   ID     54  148
##   INF   239  262
##   SAH    26   21
##
## Column percentages:
##      dead
## dgn  FALSE  TRUE
##   ICH    7.3 11.1
##   ID    15.7 30.5
##   INF   69.5 54.0
##   SAH    7.6  4.3
##   Total 100.1 99.9
##   Count 344.0 485.0

```

A função *colPercents*, do pacote *RcmdrMisc*, gera os percentuais de cada célula em relação ao total da coluna correspondente.

Assim a tabela de frequência mostra novamente que, dos 79 pacientes cujo diagnóstico era “ICH”, 54 morreram e 25 não morreram no período de tempo considerado. A tabela com as porcentagens nas colunas mostra que, dos 344 pacientes que não morreram, 7,3% (25) tiveram o diagnóstico “ICH”, e dos 485 que morreram, 11,1% (54) tiveram o diagnóstico “ICH”. Interpretação análoga se aplica às demais linhas da tabela.

7.2.5 Obtendo os percentuais de cada célula em relação ao total da tabela

Se, na aba *Estatísticas* para gerar uma tabela de dupla entrada, marcarmos a opção *percentagens do total* (figura 7.15), a tabela de dupla entrada irá mostrar os percentuais de cada célula em relação ao total da tabela.

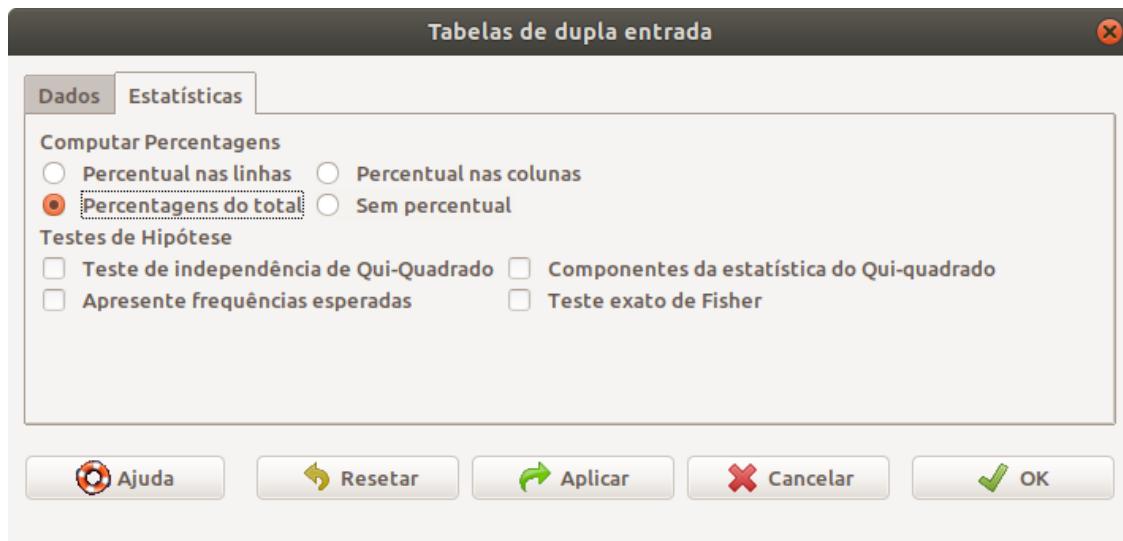


Figura 7.15: Tela para especificar que a tabela de dupla entrada irá mostrar os percentuais de cada célula em relação ao total da tabela.

Ao clicarmos em OK, os comandos abaixo serão executados, com os resultados mostrados logo a seguir.

```
local({
  .Table <- xtabs(~dgn+dead, data=stroke)
  cat("\nFrequency table:\n")
  print(.Table)
  cat("\nTotal percentages:\n")
  print(totPercents(.Table))
})

## 
## Frequency table:
##      dead
## dgn FALSE TRUE
##   ICH    25   54
##   ID     54  148
##   INF   239   262
##   SAH    26   21
##
## Total percentages:
##      FALSE TRUE Total
##   ICH     3.0  6.5  9.5
##   ID      6.5 17.9 24.4
##   INF    28.8 31.6 60.4
##   SAH     3.1  2.5  5.7
## Total  41.5 58.5 100.0
```

A função *totPercents*, do pacote *RcmdrMisc*, gera os percentuais de cada célula em relação ao total da tabela.

Assim a tabela de frequência mostra que 54 pacientes cujo diagnóstico era “ICH” morreram. A tabela com as porcentagens de cada célula mostra que, do total de 829 pacientes, 6,5% (54) tiveram o diagnóstico “ICH” e morreram. Interpretação análoga se aplica às demais células da tabela.

7.2.5.1 Outras funções para obter percentuais das células de uma tabela

Nos exemplos das seções anteriores, usamos o *R Commander* para gerar as tabelas de contingência. As funções *rowPercents*, *colPercents* e *totPercents*, todas do pacote *RcmdrMisc*, foram utilizadas para gerar os percentuais nas linhas, colunas e totais, respectivamente, para uma tabela de dupla entrada (duas variáveis categóricas).

Há outras funções disponíveis na base do R que também fornecem esses percentuais. Nesse caso, não há necessidade de carregar nenhum pacote, mas é preciso utilizar a linha de comando.

Vamos ilustrar o uso dessas funções com o mesmo exemplo da seção anterior, que gerou uma tabela de frequência para as variáveis *dgn* e *dead* do conjunto de dados *stroke*. Vamos gerar novamente um objeto que conterá a tabela de frequência das duas variáveis.

```
.Table <- xtabs(~dgn+dead, data=stroke)
# .Table <- table(stroke$dgn, stroke$dead) também funcionaria
.Table

##          dead
## dgn    FALSE TRUE
##   ICH     25   54
##   ID      54  148
##   INF    239  262
##   SAH     26   21
```

- 1) Obtendo as frequências marginais de *dgn* e *dead*.

A partir do objeto *.Table*, podemos obter as frequências de cada categoria de uma das variáveis individualmente. Essas frequências são chamadas de marginais porque correspondem aos totais nas margens da tabela, obtidos somando as frequências das células ao longo das linhas e ao longo das colunas. A função *margin.table*, mostrada a seguir, gera essas frequências para a variável *dgn*. O valor 1 no segundo argumento indica que os valores marginais serão obtidos para a variável na primeira dimensão, que corresponde às linhas da tabela. Podemos observar que as frequências das categorias de *dgn* são iguais à soma das frequências em cada linha da tabela *.Table*.

```
margin.table(.Table, 1)
```

```
## dgn  
## ICH ID INF SAH  
## 79 202 501 47
```

Para obtermos as frequências marginais da variável *dead*, substituímos o valor do segundo argumento do comando anterior pelo valor 2, indicando que os valores marginais serão obtidos para a variável na segunda dimensão, que corresponde às colunas da tabela.

```
margin.table(.Table, 2)
```

```
## dead  
## FALSE TRUE  
## 344 485
```

2) Obtendo as proporções e percentuais das linhas da tabela *.Table*

Para obtermos as proporções de cada célula em relação ao total da linha correspondente à célula, podemos utilizar a função *prop.table*, especificando como primeiro argumento o objeto correspondente à tabela que estamos trabalhando e, como segundo argumento, o valor 1, que indica a primeira dimensão da tabela. Para expressarmos as proporções como porcentagens, basta multiplicarmos os resultados por 100 como mostrado nos comandos a seguir.

```
prop.table(.Table, 1)
```

```
##      dead  
## dgn      FALSE      TRUE  
##   ICH 0.3164557 0.6835443  
##   ID  0.2673267 0.7326733  
##   INF 0.4770459 0.5229541  
##   SAH 0.5531915 0.4468085
```

```
prop.table(.Table, 1)*100
```

```
##      dead  
## dgn      FALSE      TRUE  
##   ICH 31.64557 68.35443  
##   ID  26.73267 73.26733  
##   INF 47.70459 52.29541  
##   SAH 55.31915 44.68085
```

3) Obtendo as proporções e percentuais das colunas da tabela *.Table*

Para obtermos as proporções de cada célula em relação ao total da coluna correspondente à célula, podemos utilizar a função *prop.table*, especificando como primeiro argumento o objeto

correspondente à tabela que estamos trabalhando e, como segundo argumento, o valor 2, que indica a segunda dimensão da tabela. Para expressarmos as proporções como porcentagens, basta multiplicarmos os resultados por 100 como mostrado nos comandos a seguir.

```
prop.table(.Table, 2)
```

```
##      dead
## dgn      FALSE      TRUE
##   ICH 0.07267442 0.11134021
##   ID  0.15697674 0.30515464
##   INF 0.69476744 0.54020619
##   SAH 0.07558140 0.04329897
```

```
prop.table(.Table, 2)*100
```

```
##      dead
## dgn      FALSE      TRUE
##   ICH 7.267442 11.134021
##   ID  15.697674 30.515464
##   INF 69.476744 54.020619
##   SAH 7.558140  4.329897
```

3) Obtendo as proporções e percentuais de cada célula da tabela *.Table*

Para obtermos as proporções de cada célula em relação ao total da tabela, basta dividirmos o objeto correspondente à tabela que estamos trabalhando pelo soma das frequências de todas as células da tabela, obtida por meio da função *sum* aplicada à tabela. Para expressarmos as proporções como porcentagens, basta multiplicarmos os resultados por 100 como mostrado nos comandos a seguir.

```
.Table/sum(.Table)
```

```
##      dead
## dgn      FALSE      TRUE
##   ICH 0.03015682 0.06513872
##   ID  0.06513872 0.17852835
##   INF 0.28829916 0.31604343
##   SAH 0.03136309 0.02533172
```

```
.Table/sum(.Table)*100
```

```
##      dead
## dgn      FALSE      TRUE
##   ICH 3.015682 6.513872
##   ID  6.513872 17.852835
##   INF 28.829916 31.604343
##   SAH 3.136309 2.533172
```

7.2.6 Tabelas de frequência para mais de duas variáveis categóricas

Para gerarmos tabelas de frequências ou porcentagens para a combinação das categorias de três ou mais variáveis categóricas no *R Commander*, usamos a opção:

Estatísticas ⇒ Tabelas de contingência ⇒ Tabela multientrada...

Esse tipo de tabela é chamada também de **tabela de múltiplas entradas** (por envolver mais de duas variáveis).

Na tela dessa opção (figura 7.16), selecionamos uma variável cujas categorias irão aparecer nas linhas da tabela, outra variável cujas categorias irão aparecer nas colunas da tabela e outra(s) variável(is) (chamadas de variáveis de controle no *R Commander*) para as quais desejamos a distribuição conjunta de frequências. Nesse exemplo, selecionamos a variável *dgn* (diagnóstico) para as linhas, *dead* para as colunas e *minf* (história de infarto do miocárdio) para a variável de controle. Também selecionamos a opção de mostrar os percentuais de cada célula na linha correspondente da tabela.

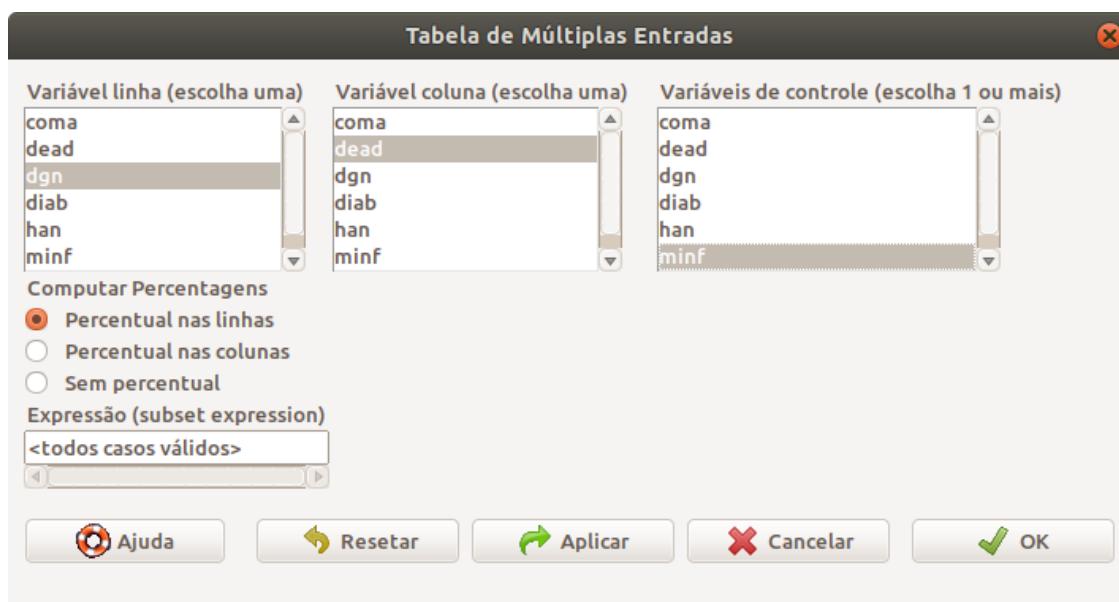


Figura 7.16: Tela para a seleção das variáveis categóricas que comporão a tabela multientrada com três variáveis.

Ao clicarmos em OK, a sequência de comandos abaixo será executada e os resultados mostrados a seguir.

```
local({  
  .Table <- xtabs(~dgn+dead+minf, data=stroke)  
  cat("\nFrequency table:\n")
```

```

print(.Table)
cat("\nRow percentages:\n")
print(rowPercents(.Table))
})

## Frequency table:
## , , minf = No
##
##      dead
## dgn FALSE TRUE
##   ICH    25    46
##   ID     49   131
##   INF    222   207
##   SAH    24    21
##
## , , minf = Yes
##
##      dead
## dgn FALSE TRUE
##   ICH     0     8
##   ID      4    12
##   INF    17    54
##   SAH     2     0
##
## Row percentages:
## , , minf = No
##
##      dead
## dgn FALSE TRUE Total Count
##   ICH  35.2 64.8    100    71
##   ID   27.2 72.8    100   180
##   INF  51.7 48.3    100  429
##   SAH  53.3 46.7    100    45
##
## , , minf = Yes
##
##      dead
## dgn FALSE TRUE Total Count
##   ICH   0.0 100.0    100     8
##   ID    25.0 75.0    100    16
##   INF   23.9 76.1    100    71
##   SAH 100.0   0.0    100     2

```

Uma tabela de frequências de dupla entrada (com as variáveis *dgn* e *dead*) é construída para cada categoria da variável *minf* e as porcentagens nas linhas são calculadas para cada uma das tabelas separadamente. Na função *xtabs*, a primeira variável após o “~” é aquela cujas categorias aparecerão nas linhas de cada tabela, a segunda variável após o “~” é aquela cujas categorias aparecerão nas colunas de cada tabela. Uma tabela de frequências será construída com as duas primeiras variáveis para cada categoria da terceira variável.

Se desejássemos os percentuais nas colunas, bastaria selecionar a opção correspondente na figura 7.16.

Podemos selecionar mais de três variáveis para montar uma tabela de múltiplas entradas. Na tela mostrada na figura 7.17, selecionamos quatro variáveis: *dgn* para as linhas, *dead* para as colunas, *diab* (história de diabetes) e *minf* (história de infarto do miocárdio) para as variáveis de controle. Dessa vez, foi selecionada a opção de mostrar os percentuais de cada célula na coluna correspondente da tabela. Uma tabela será construída para cada combinação das categoriais das variáveis *diab* e *minf*. Nesse caso, serão mostradas quatro tabelas.



Figura 7.17: Tela para a seleção das variáveis categóricas que comporão a tabela multientrada com quatro variáveis.

Ao clicarmos em OK, a sequência de comandos abaixo será executada e os resultados mostrados em seguida.

```
local({
  .Table <- xtabs(~dgn+dead+diab+minf, data=stroke)
  cat("\nFrequency table:\n")
  print(.Table)
  cat("\nColumn percentages:\n")
  print(colPercents(.Table))
})
```

```

##  

## Frequency table:  

## , , diab = No, minf = No  

##  

##      dead  

## dgn FALSE TRUE  

##   ICH    23   43  

##   ID     39  114  

##   INF   203  171  

##   SAH    24   21  

##  

## , , diab = Yes, minf = No  

##  

##      dead  

## dgn FALSE TRUE  

##   ICH     2     3  

##   ID      10   14  

##   INF     19   35  

##   SAH     0     0  

##  

## , , diab = No, minf = Yes  

##  

##      dead  

## dgn FALSE TRUE  

##   ICH     0     6  

##   ID      4    12  

##   INF    13   46  

##   SAH     2     0  

##  

## , , diab = Yes, minf = Yes  

##  

##      dead  

## dgn FALSE TRUE  

##   ICH     0     2  

##   ID      0     0  

##   INF     4     8  

##   SAH     0     0  

##  

##  

##      dead  

## dgn FALSE TRUE  

##   ICH    8.0  12.3

```

```

##   ID      13.5 32.7
##   INF     70.2 49.0
##   SAH      8.3  6.0
##   Total   100.0 100.0
##   Count   289.0 349.0
##
## , , diab = Yes, minf = No
##
##       dead
## dgn    FALSE  TRUE
##   ICH      6.5  5.8
##   ID      32.3 26.9
##   INF     61.3 67.3
##   SAH      0.0  0.0
##   Total   100.1 100.0
##   Count   31.0  52.0
##
## , , diab = No, minf = Yes
##
##       dead
## dgn    FALSE  TRUE
##   ICH      0.0  9.4
##   ID      21.1 18.8
##   INF     68.4 71.9
##   SAH     10.5  0.0
##   Total   100.0 100.1
##   Count   19.0  64.0
##
## , , diab = Yes, minf = Yes
##
##       dead
## dgn    FALSE  TRUE
##   ICH      0   20
##   ID      0   0
##   INF     100  80
##   SAH      0   0
##   Total   100 100
##   Count    4   10

```

7.2.6.1 Forma alternativa de apresentar tabelas de frequência para mais de duas variáveis categóricas

Vimos na seção anterior que a função *xtabs* (assim como a função *table*) irá apresentar, para mais de duas variáveis, tantas tabelas quantas forem as combinações possíveis das categorias das variáveis de controle (variáveis após a segunda na fórmula ou na lista de variáveis).

Existe uma outra forma de apresentar os resultados da distribuição de frequências conjuntas, por meio da função *ftable*.

Se quisermos obter a distribuição de frequências conjuntas das variáveis *dgn*, *dead* e *minf*, com *dgn* nas linhas, usamos o comando a seguir.

```
ftable(minf + dead ~ dgn, data=stroke)
```

```
##      minf     No      Yes
##      dead FALSE TRUE FALSE TRUE
## dgn
## ICH        25    46      0     8
## ID         49   131      4    12
## INF       222   207     17    54
## SAH        24    21      2     0
```

As variáveis após o “~” serão mostradas nas linhas e aquelas antes do “~” serão mostradas nas colunas. Nesse exemplo, uma tabela relacionando *dgn* com *dead* será mostrada para a categoria *No* de *minf* ao lado de outra tabela relacionando *dgn* com *dead* para a categoria *Yes* de *minf*.

Se colocarmos *dead* antes de *minf* na fórmula, como no comando a seguir, uma tabela relacionando *dgn* com *minf* será mostrada para a categoria *FALSE* de *dead* ao lado de outra tabela relacionando *dgn* com *minf* para a categoria *TRUE* de *dead*.

```
ftable(dead + minf ~ dgn, data=stroke)
```

```
##      dead FALSE      TRUE
##      minf     No Yes     No Yes
## dgn
## ICH        25    0    46    8
## ID         49    4   131   12
## INF       222   17   207   54
## SAH        24    2    21    0
```

Para obtermos as proporções de cada célula em relação ao total da linha correspondente, podemos utilizar a função *prop.table*, especificando como primeiro argumento o objeto correspondente à tabela que estamos trabalhando (gerado pela função *ftable*) e, como segundo argumento, o valor 1, que indica a primeira dimensão da tabela.

```
prop.table(ftable(dead + minf ~ dgn, data=stroke), 1)
```

```
##      dead      FALSE          TRUE
##      minf       No        Yes       No        Yes
## dgn
## ICH      0.31645570 0.00000000 0.58227848 0.10126582
## ID       0.25000000 0.02040816 0.66836735 0.06122449
## INF      0.44400000 0.03400000 0.41400000 0.10800000
## SAH      0.51063830 0.04255319 0.44680851 0.00000000
```

Para expressar as proporções como porcentagens, basta multiplicar os resultados por 100 como mostrado a seguir.

```
prop.table(ftable(dead + minf ~ dgn, data=stroke), 1) * 100
```

```
##      dead      FALSE          TRUE
##      minf       No        Yes       No        Yes
## dgn
## ICH      31.645570 0.000000 58.227848 10.126582
## ID       25.000000 2.040816 66.836735 6.122449
## INF      44.400000 3.400000 41.400000 10.800000
## SAH      51.063830 4.255319 44.680851 0.000000
```

Para obtermos as proporções de cada célula em relação ao total da coluna correspondente, utilizamos a função *prop.table*, especificando como segundo argumento o valor 2, que indica a segunda dimensão da tabela.

```
prop.table(ftable(dead + minf ~ dgn, data=stroke), 2)
```

```
##      dead      FALSE          TRUE
##      minf       No        Yes       No        Yes
## dgn
## ICH      0.07812500 0.00000000 0.11358025 0.10810811
## ID       0.15312500 0.17391304 0.32345679 0.16216216
## INF      0.69375000 0.73913043 0.51111111 0.72972973
## SAH      0.07500000 0.08695652 0.05185185 0.00000000
```

O comando abaixo mostra a função *ftable* com quatro variáveis, sendo as categorias de *dgn* mostradas nas linhas.

```
ftable(diab + minf + dead ~ dgn, data=stroke)
```

```
##      diab    No          Yes
##      minf    No      Yes      No      Yes
##      dead FALSE TRUE FALSE TRUE FALSE TRUE
## dgn
## ICH      23   43      0     6      2     3      0     2
## ID       39  114      4    12     10    14      0     0
## INF     203  171     13    46     19    35      4     8
## SAH      24   21      2     0      0     0      0     0
```

O comando abaixo mostra a função *ftable* com quatro variáveis, sendo as categorias de *dgn* e *diab* mostradas nas linhas. Observem a flexibilidade nos arranjos das tabelas, de acordo com a posição das variáveis (antes e depois do “~”) e da ordem em que elas são colocadas.

```
ftable(minf + dead ~ dgn + diab, data=stroke)
```

```
##           minf    No          Yes
##           dead FALSE TRUE FALSE TRUE
## dgn diab
## ICH No      23   43      0     6
##     Yes      2     3      0     2
## ID  No      39  114      4    12
##     Yes      10   14      0     0
## INF No     203  171     13    46
##     Yes      19   35      4     8
## SAH No      24   21      2     0
##     Yes      0     0      0     0
```

7.2.7 Entrando diretamente com as frequências das células

Às vezes, temos disponível uma tabela de frequências já construída, por exemplo, publicada em artigos científicos ou relatórios técnicos e desejamos obter as proporções ou percentuais nas linhas ou colunas e realizar alguma análise estatística.

A opção a seguir nos permite digitar as frequências de cada célula de uma tabela de dupla entrada no *R Commander* e obter os percentuais nas linhas, colunas ou células, além de realizar um teste estatístico de associação entre as duas variáveis:

Estatísticas ⇒ Tabelas de contingência ⇒ Digite e analise tabela dupla entrada...

Na aba *Tabela* dessa opção (figura 7.18), podemos escrever os nomes para as variáveis cujas categorias irão aparecer nas linhas e colunas da tabela, selecionar o número de categorias nas linhas e colunas e digitar os valores das células da tabela de dupla entrada. Nesse exemplo, reproduzimos a tabela de frequências conjuntas das variáveis *dgn* (diagnóstico) para as linhas e *dead* para as colunas (seção 7.2.2). Na aba *Estatísticas* (não mostrada aqui), selecionamos a opção *Percentuais nas linhas* e desmarcamos a opção *Teste de independência de Qui-Quadrado*, porque esse teste não será abordado neste texto.

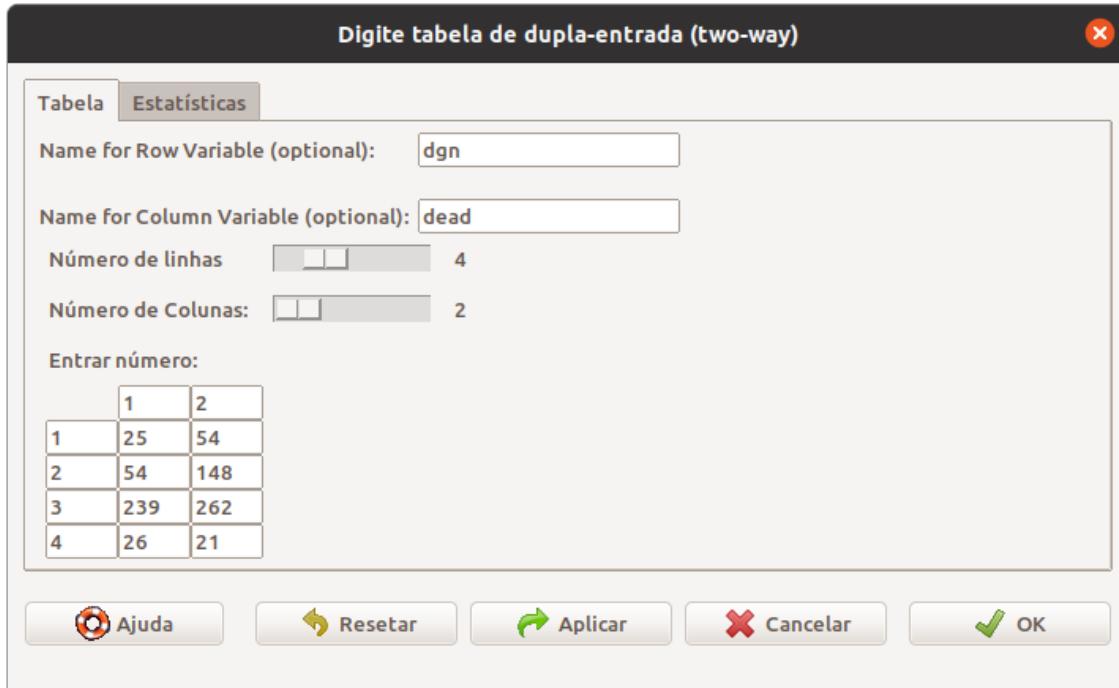


Figura 7.18: Tela para a digitação das frequências das células que comporão uma tabela de dupla entrada.

Ao clicarmos em OK, a sequência de comandos a seguir será executada e os resultados mostrados logo após.

```
.Table <- matrix(c(25,54,54,148,239,262,26,21), 4, 2, byrow=TRUE)
dimnames(.Table) <- list("dgn"=c("1", "2", "3", "4"), "dead"=c("1", "2"))
.Table # Counts

##      dead
## dgn   1   2
##   1 25 54
##   2 54 148
##   3 239 262
##   4 26 21
```

```
rowPercents(.Table) # Row Percentages
```

```
##      dead
## dgn    1   2 Total Count
##   1 31.6 68.4   100    79
##   2 26.7 73.3   100   202
##   3 47.7 52.3   100   501
##   4 55.3 44.7   100    47
```

Observamos que os resultados são idênticos aos mostrados na seção 7.2.2, porém os nomes das categorias nas linhas e colunas aparecem como números.

A tabela foi construída por meio da função *matrix*. Podemos tornar os nomes das categorias mais descriptivos por meio das funções *colnames* e *rownames* aplicadas ao objeto que representa a tabela gerada (*.Table*), como mostrado a seguir.

```
colnames(.Table) <- c("Não", "Sim")
rownames(.Table) <- c("ICH", "ID", "INF", "SAH")
.Table
```

```
##      dead
## dgn  Não Sim
##   ICH 25 54
##   ID  54 148
##   INF 239 262
##   SAH 26 21
```

Podemos alterar os nomes das dimensões da tabela para valores mais descriptivos, usando a função *names*, como mostrado a seguir.

```
names(dimnames(.Table)) <- c("diagnóstico", "morte")
.Table
```

```
##          morte
## diagnóstico Não Sim
##       ICH 25 54
##       ID  54 148
##       INF 239 262
##       SAH 26 21
```

Poderíamos combinar os comandos anteriores (*colnames*, *rownames* e *names*) em um único comando, como mostrado a seguir.

```
dimnames(.Table) <- list("diagnóstico"=c("ICH", "ID", "INF", "SAH"),
                         "morte"=c("Não", "Sim"))
.Table
```

```

##          morte
## diagnóstico Não Sim
##          ICH  25  54
##          ID   54 148
##          INF 239 262
##          SAH  26   21

```

Se quisermos trocar as linhas pelas colunas na tabela de frequência, podemos usar a função `t` (de transposta), como mostra o comando a seguir.

```
t(.Table)
```

```

##          diagnóstico
## morte ICH  ID INF SAH
## Não  25   54 239 26
## Sim   54 148 262 21

```

7.3 Exercício

- 1) Com o conjunto de dados *respiratory* do pacote *HSAUR2* (GPL-2). Para abrir esse conjunto de dados, é necessário que o pacote *HSAUR2* esteja instalado, faça as atividades seguintes.
 - a) Verifique a ajuda do conjunto de dados e o significado das variáveis *status*, *treatment* e *centre*.
 - b) Crie um subconjunto de *respiratory*, chamado *respMonth4*, com os dados dos pacientes somente do mês 4.
 - c) Monte tabelas de frequências e de porcentagens para o conjunto de dados *respMonth4* para cada uma das variáveis da letra a.
 - d) Monte uma tabela de contingência relacionando as variáveis *treatment* e *status* no conjunto de dados *respMonth4*. Quais são as porcentagens de sucesso de cada um dos níveis da variável *treatment*?
 - e) Monte uma tabela de contingência para o conjunto de dados *respMonth4*, relacionando as variáveis *treatment* e *status* e *centre* com porcentagens obtidas a partir das linhas. Comente os resultados.

Capítulo 8

Visualização de dados

Uma introdução sobre diversos diagramas que são frequentemente utilizados para explorar os dados de um conjunto de dados pode ser visualizada neste [vídeo](#).

O capítulo anterior deu início à exploração de dados com as medidas de tendência central e de dispersão. Neste capítulo, serão abordados alguns recursos gráficos para visualizar a distribuição dos valores das variáveis de um determinado conjunto de dados. Os diagramas que serão abordados são:

diagrama de barras

diagrama de setores (pizza ou torta)

diagrama de caixa (*boxplot*)

histograma

histograma de densidade de frequência

diagrama de pontos

diagrama de *strip chart*

diagrama de dispersão

Os dois primeiros diagramas (barras e tortas) são utilizados para indicar a contagem (frequência) ou proporção de cada categoria de uma variável categórica. Os demais são utilizados para variáveis numéricas.

A figura 8.1 é um dos resultados do estudo de Furuta et al. (Furuta et al., 2003), que consistiu de um estudo clínico randomizado duplo-cego em crianças com adenoide obstrutiva submetidas a tratamento homeopático. Ela mostra um diagrama de barras das frequências dos tratamentos de acordo com a evolução, considerando a avaliação nasofibroscópica.

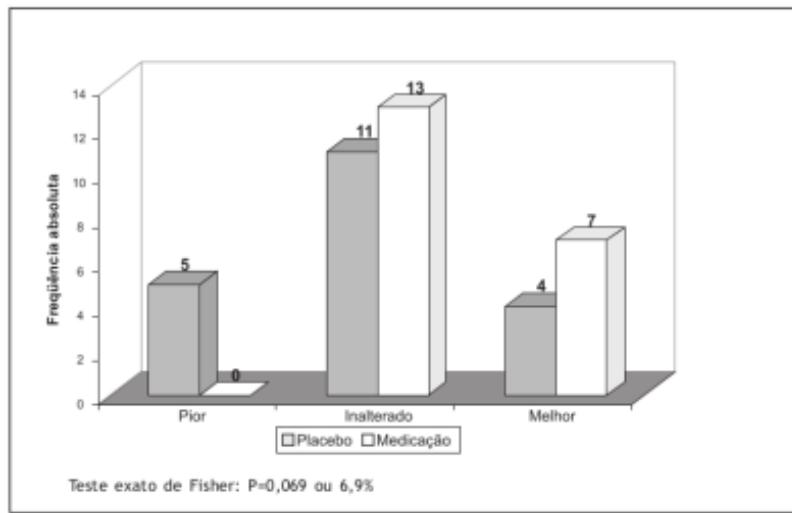


Figura 8.1: Diagrama de barras das frequências dos tratamentos segundo a evolução dos pacientes de acordo com a avaliação nasofibroscópica inicial e final. Fonte: (Furuta et al., 2003) ([CC BY-NC](#)).

As figuras 8.2 e 8.3 apresentam os diagramas de pizza mostrando o percentual de cada uma das categorias da evolução de acordo com a avaliação nasofibroscópica inicial e final para os tratamentos homeopático (figura 8.2) e para o placebo (figura 8.3). O percentual de inalterado no tratamento homeopático foi de 65%, enquanto que, no placebo, ele foi de 55%. Nenhum paciente teve uma piora na homeopatia e 25% teve um piora no grupo placebo. Finalmente 35% por cento tiveram uma melhora com o tratamento homeopático versus 20% com o placebo.

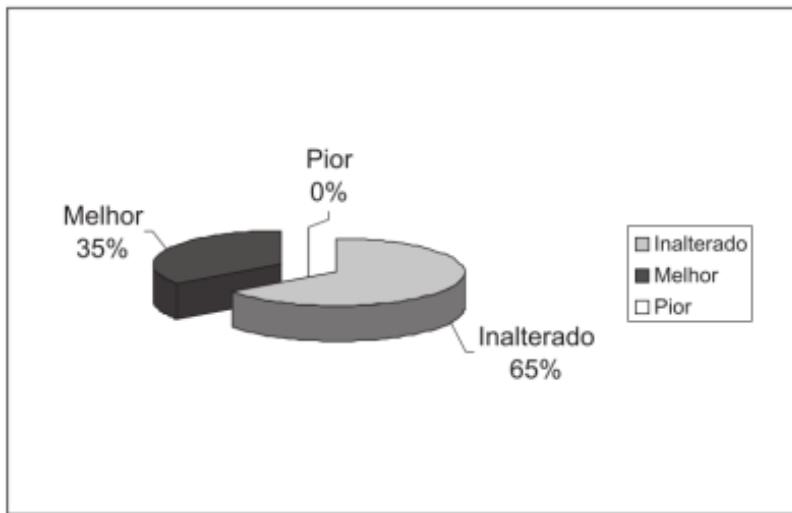


Figura 8.2: Diagrama de pizza da evolução dos pacientes do grupo I (medicamento homeopático), comparando as nasofibroscopias inicial e final do tratamento. Fonte: (Furuta et al., 2003) ([CC BY-NC](#)).

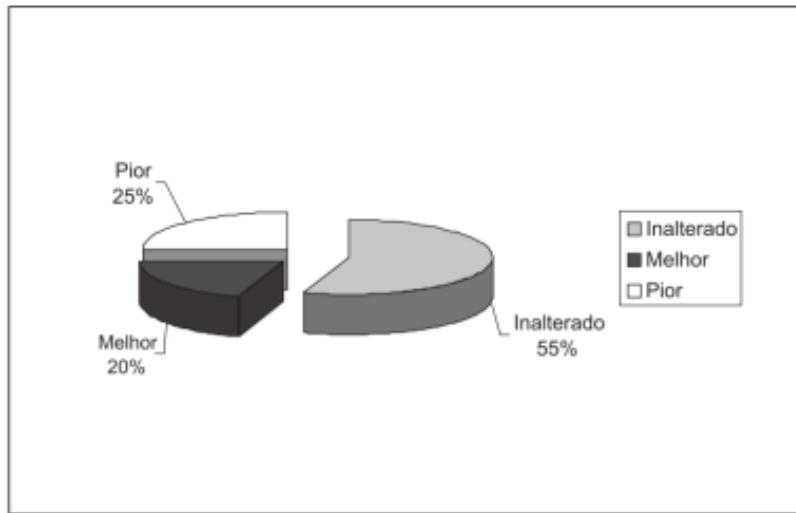


Figura 8.3: Diagrama de pizza da evolução dos pacientes do grupo II (placebo), comparando as nasofibroscopias inicial e final do tratamento. Fonte: (Furuta et al., 2003) ([CC BY-NC](#)).

Os dois diagramas de pizza (figuras 8.2 e 8.3) fornecem em conjunto a mesma informação que o diagrama de barras (figura 8.1) de que o tratamento homeopático tem o melhor desempenho do que o placebo em relação à avaliação nasofibroscópica, porém a análise estatística mostrou que essas diferenças não foram estatisticamente significativas ao nível de 5%.

O *boxplot* é utilizado para verificar a distribuição dos valores de uma variável numérica. A figura 8.4 apresenta diagramas de *boxplot* da contagem de diversos marcadores imunológicos em pacientes com a pele exposta à radiação ultravioleta (em vermelho) ou coberta (em azul). O diagrama sugere que as contagens dos marcadores CD45 e CD68 assim como o número de mastócitos ATM são maiores nos indivíduos expostos do que nos indivíduos não expostos à radiação ultra-violeta.

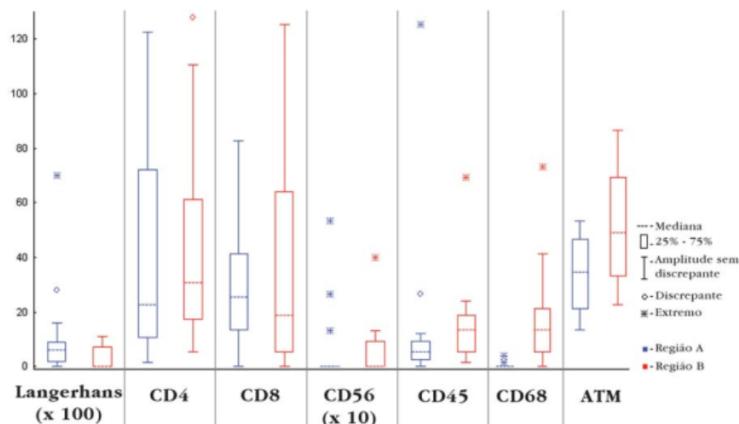


Figura 8.4: *Boxplots* dos marcadores imunológicos em pele coberta e exposta. Fonte: (Bezerra et al., 2011) ([CC BY-NC](#)).

A figura 8.5 mostra quatro gráficos de pontos dos níveis séricos das citocinas TNF-fator de necrose tumoral, interleucina-6, interleucina 8 assim como da RANTES (CCL5) para os seguintes grupos de pacientes: controles saudáveis, pacientes com DPOC (Doença Pulmonar Obstrutiva Crônica) e limitação do fluxo aéreo não reversível e pacientes com DPOC e limitação ao fluxo aéreo parcialmente reversível. O diagrama de pontos também é utilizado para verificar a distribuição dos valores de uma variável numérica.

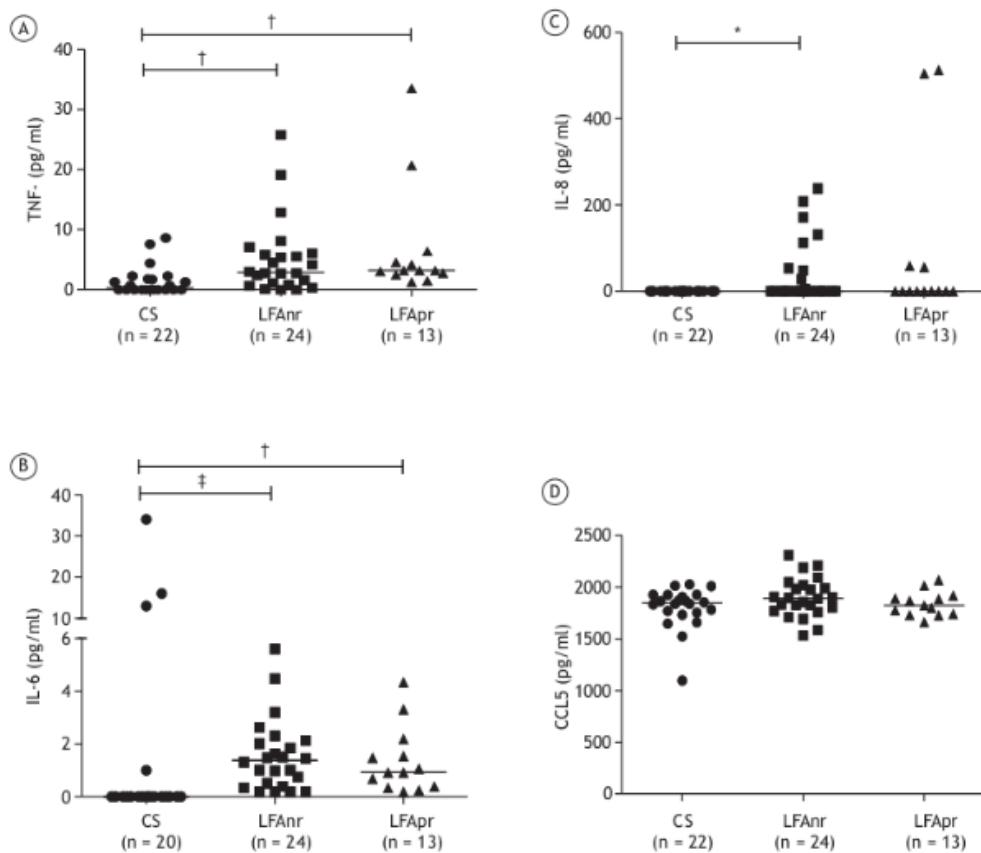


Figura 8.5: Gráfico de pontos dos níveis séricos das citocinas TNF (A), IL-6 (B) e IL-8 (C), assim como da RANTES (CCL5; D), em controles saudáveis (CS), pacientes com DPOC e limitação ao fluxo aéreo não reversível (LFAnr) e pacientes com DPOC e limitação ao fluxo aéreo parcialmente reversível (LFApr). Fonte: (Queiroz et al., 106) (CC BY-NC).

A figura 8.6 mostra um histograma do escore SYNTAX em pacientes com doença arterial coronariana multiarterial com indicação de intervenção coronária percutânea eletiva ou na vigência de síndrome coronariana aguda sem elevação do segmento ST com stents farmacológicos de primeira ou segunda gerações. O escore SYNTAX permite a estratificação do paciente quanto à complexidade angiográfica das lesões coronarianas.

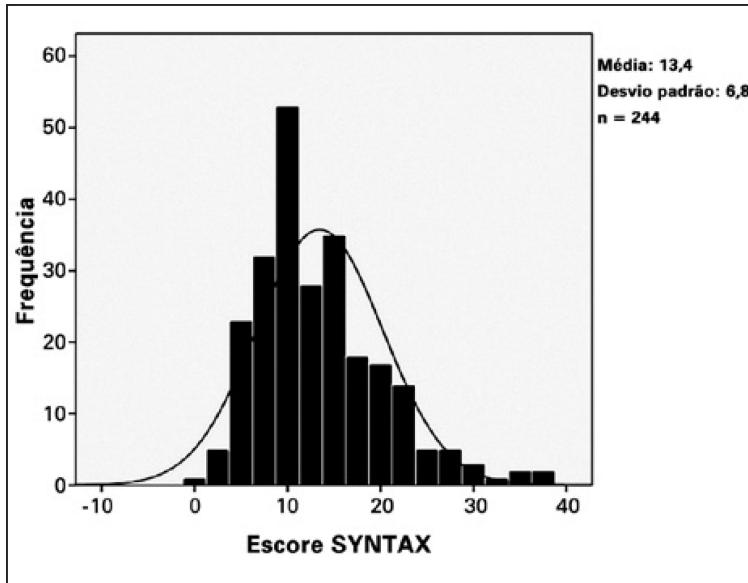


Figura 8.6: Histograma do escore SYNTAX em pacientes com doença arterial coronariana multiarterial. Fonte: (Silva et al., 2014) ([CC BY-NC](#)).

A figura 8.7 mostra um diagrama de dispersão ou espalhamento da relação fração de ejeção do ventrículo esquerdo versus relação neutrófilo-linfócito. Cada ponto corresponde aos valores da relação neutrófilo-linfócito na abscissa e a fração de ejeção do ventrículo esquerdo na ordenada para cada paciente do estudo. Esse gráfico sugere que a fração de ejeção do ventrículo esquerdo tende a diminuir à medida que a relação neutrófilo-linfócito aumenta e os autores apresentaram no gráfico a reta que melhor se ajusta a esses pontos.

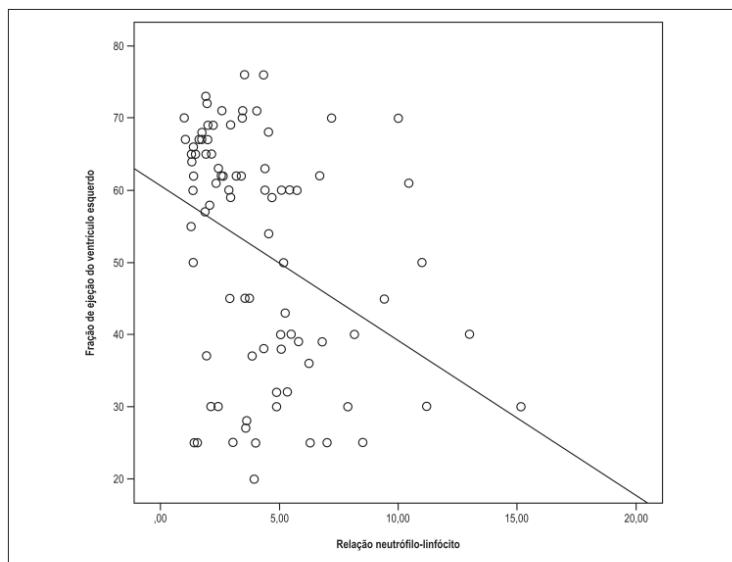


Figura 2 – Análise de correlação da relação neutrófilo-linfócito com a fração de ejeção do ventrículo esquerdo.

Figura 8.7: Diagrama de dispersão da relação neutrófilo-linfócito versus fração de ejeção do ventrículo esquerdo. Fonte: (Durmus et al., 2015) ([CC BY](#)).

Vamos utilizar neste capítulo o *R Commander*, carregado a partir do *RStudio*, mas também poderíamos utilizar o *R Commander*, carregado a partir da tela de entrada do R.

Inicialmente, vamos executar o *RStudio*.

Em seguida, digitamos os comandos abaixo na console do *RStudio* e pressionamos a tecla *Enter* (figura 8.8).

```
plot.new(); library(Rcmdr)
```

```
Console Terminal × Jobs ×
~/Estatistica/livro/bookdown/R_Rcmdr_RStudio/ ↗

R version 3.6.1 (2019-07-05) -- "Action of the Toes"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R é um software livre e vem sem GARANTIA ALGUMA.
Você pode redistribuí-lo sob certas circunstâncias.
Digite 'license()' ou 'licence()' para detalhes de distribuição.

R é um projeto colaborativo com muitos contribuidores.
Digite 'contributors()' para obter mais informações e
'citation()' para saber como citar o R ou pacotes do R em publicações.

Digite 'demo()' para demonstrações, 'help()' para o sistema on-line de ajuda,
ou 'help.start()' para abrir o sistema de ajuda em HTML no seu navegador.
Digite 'q()' para sair do R.

> plot.new(); library(Rcmdr)
```

Figura 8.8: Console do *RStudio*, após a digitação dos comandos para carregar o pacote *Rcmdr*.

Observação: A função *plot.new()*, é executada antes de carregar o *R Commander* para garantir que os gráficos gerados pelo *R Commander* sejam mostrados na interface do *RStudio*. Caso ela (ou alguma função que gere um gráfico no *RStudio*) não seja executada antes de carregar o *R Commander*, os gráficos gerados pelo *R Commander* serão exibidos em uma janela separada.

Vamos utilizar o conjunto de dados *juul2*, após a conversão das variáveis *sex*, *menarche* e *tanner* para fator. A seção 6.1.2 mostra como fazer essas conversões no *R Commander*. Os comandos a seguir carregam o pacote *ISwR* e o conjunto de dados *juul2*, e convertem as variáveis numéricas *sex*, *tanner* e *menarche* para fator, gerando respectivamente as variáveis *sexo_cat*, *tanner_cat* e *menarca_cat*.

```

library(ISwR)
data(juul2, package="ISwR")
juul2 <- within(juul2, {
  sexo_cat <- factor(sex, labels=c('masculino','feminino'))
  tanner_cat <- factor(tanner,
    labels=c('Tanner I','Tanner II','Tanner III','Tanner IV','Tanner V'))
  menarca_cat <- factor(menarche, labels=c('sim','não'))
})

```

8.1 Diagrama de barras

O conteúdo desta seção pode ser visualizado neste [vídeo](#).

As variáveis categóricas nominais ou ordinais podem ser visualizadas graficamente por diagramas que fornecem as contagem (frequência) ou a proporção (ou porcentagem) de cada categoria da variável no conjunto de dados. Um dos diagramas mais utilizados com essa finalidade é o diagrama de barras, o qual mostra para cada categoria uma barra com altura proporcional à frequência ou proporção da respectiva categoria. Para criar um diagrama de barras no *R Commander*, selecionamos a opção:

Gráficos ⇒ Gráfico de barras

Na aba *Dados* da caixa de diálogo *Gráfico de Barras*, é possível selecionar a variável categórica desejada. Nesse exemplo, vamos criar um diagrama de barras para a variável categórica *tanner_cat* (figura 8.9). Na aba *Opções* dessa caixa de diálogo (figura 8.10), é possível especificar a escala do eixo Y (porcentagem ou frequência), selecionar a cor e a posição das legendas, especificar as legendas dos eixos X e Y e o título do gráfico e selecionar se as frequências ou porcentagens serão exibidas nas barras. As demais opções serão discutidas logo adiante. Ao clicarmos em OK, o gráfico será exibido na aba *Plots* do *RStudio* (figura 8.11), mostrando a frequência de cada barra. Se utilizarmos o *R Commander*, carregado a partir do R, o gráfico será exibido em uma outra janela ou na janela do R.

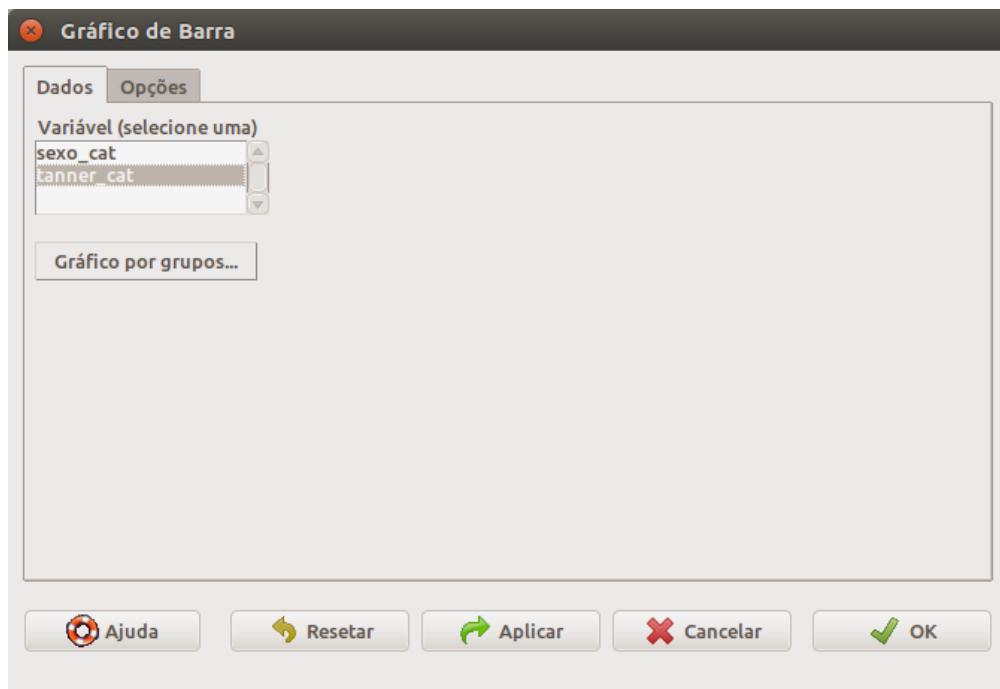


Figura 8.9: Caixa de diálogo para a geração de um diagrama de barras: selecionando a variável.



Figura 8.10: Caixa de diálogo para a geração de um diagrama de barras: especificando o título do gráfico e as legendas dos eixos X e Y.

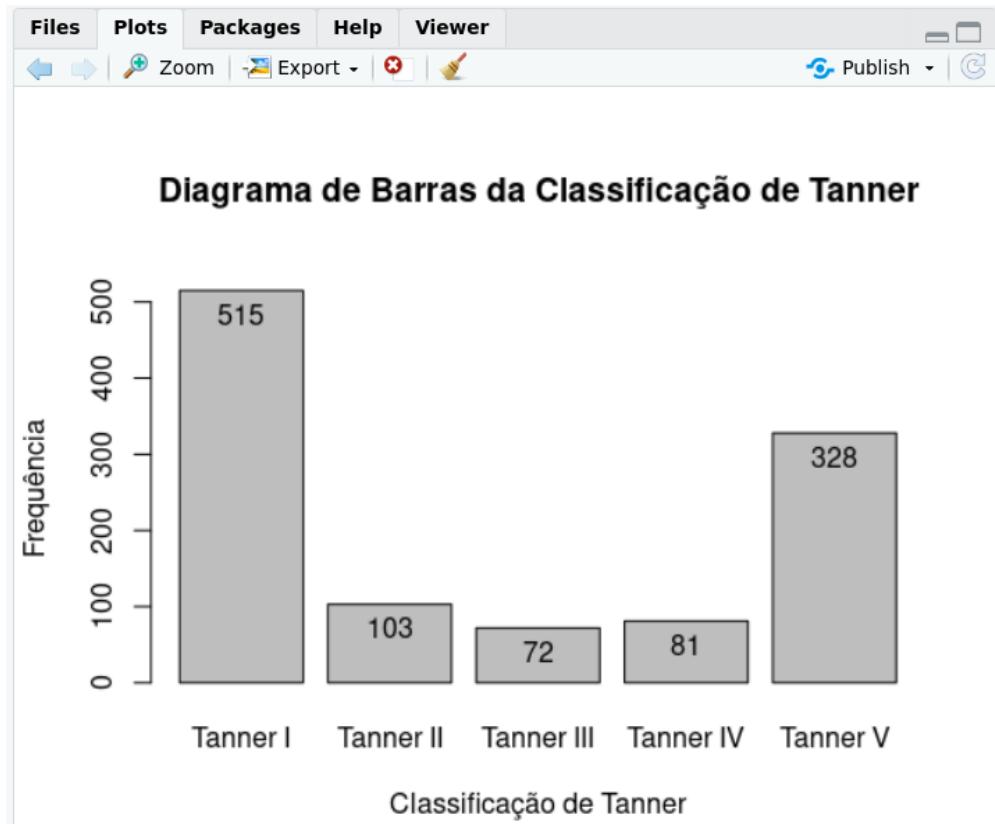


Figura 8.11: Diagrama de barras para a variável *tanner_cat*. São mostradas as frequências de cada categoria de Tanner.

O gráfico da figura 8.11 mostra as frequências de cada uma das cinco categorias da classificação de Tanner no conjunto de dados *juul2*. A categoria I é a mais frequente, seguida da categoria V. As categorias II, III e IV apresentam frequências próximas umas das outras, mas com frequências bem menores do que as categorias I e V.

Caso desejemos visualizar o diagrama de barras da variável *sexo_cat* separadamente para cada categoria de Tanner, precisamos selecionar *sexo_cat* como uma variável de agrupamento. Para isso, clicamos na opção *Gráfico por grupos* na figura 8.9. Seremos, então, apresentados à caixa de diálogo da figura 8.12, onde selecionaremos a variável de agrupamento (*sexo_cat*). Ao clicarmos em OK, voltamos à tela da figura 8.9.

Clicando na aba *Opções*, mostrada novamente na figura 8.13, podemos, em *Estilo de barras agrupadas*, escolher entre duas opções de como o diagrama de barras será construído: barras de cada categoria da variável *sexo_cat* empilhadas (particionada) para cada categoria da variável *tanner_cat*, ou lado a lado. Selecionando a segunda opção e clicando em OK, será plotado o gráfico da figura 8.14. Ao selecionarmos a primeira opção, obteremos o gráfico da figura 8.15, onde desmarcamos a opção *Show counts or percentages in bars* na figura 8.13.

Ambos os gráficos mostram que há uma predominância de homens nas categorias I, II e IV (especialmente na I), e mais mulheres nas categorias III e V (principalmente na V). Para cada sexo, as frequências de cada categoria de Tanner seguem o padrão mostrado na figura 8.11.

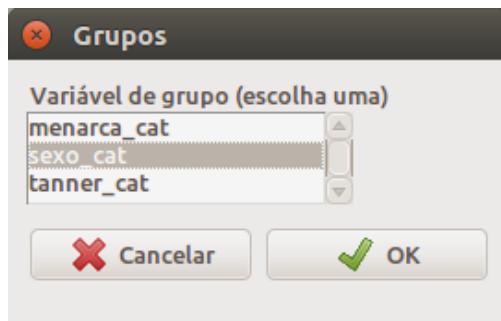


Figura 8.12: Selecionando uma variável de agrupamento para o diagrama de barras das frequências das categorias da variável *sexo_cat* para cada categoria da classificação de Tanner.



Figura 8.13: Selecionando a forma como as barras serão apresentadas (lado a lado ou empilhadas). Neste exemplo, foi selecionada a opção lado a lado.

Diagrama de Barras de Sexo por Categoria de Tanner

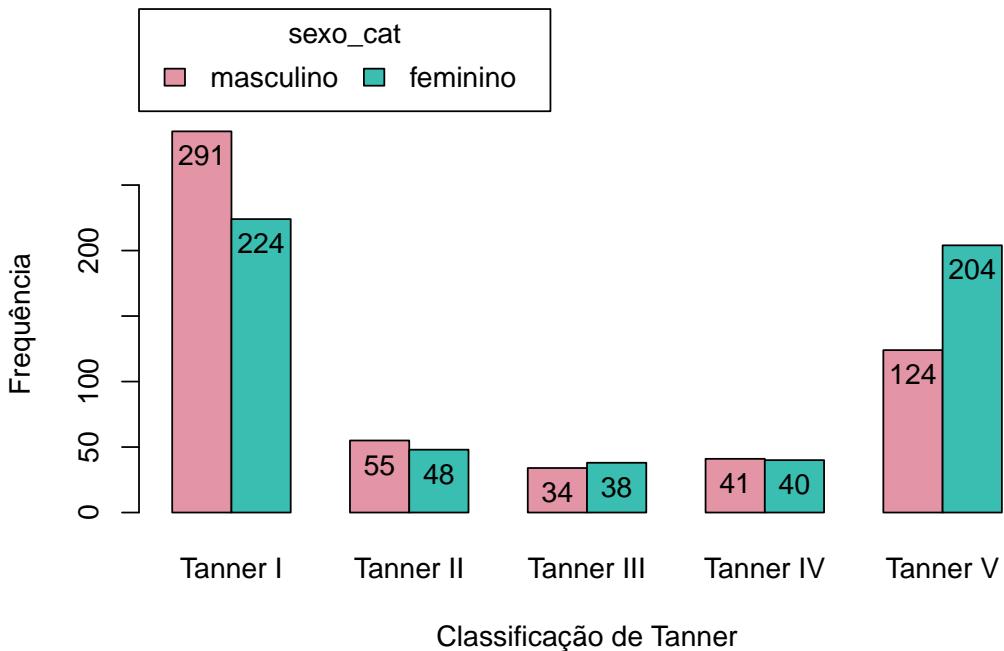


Figura 8.14: Diagrama de barras lado a lado das frequências das categorias da variável *sexo_cat* para cada categoria da variável *tanner_cat*.

Diagrama de Barras de Sexo por Categoria de Tanner

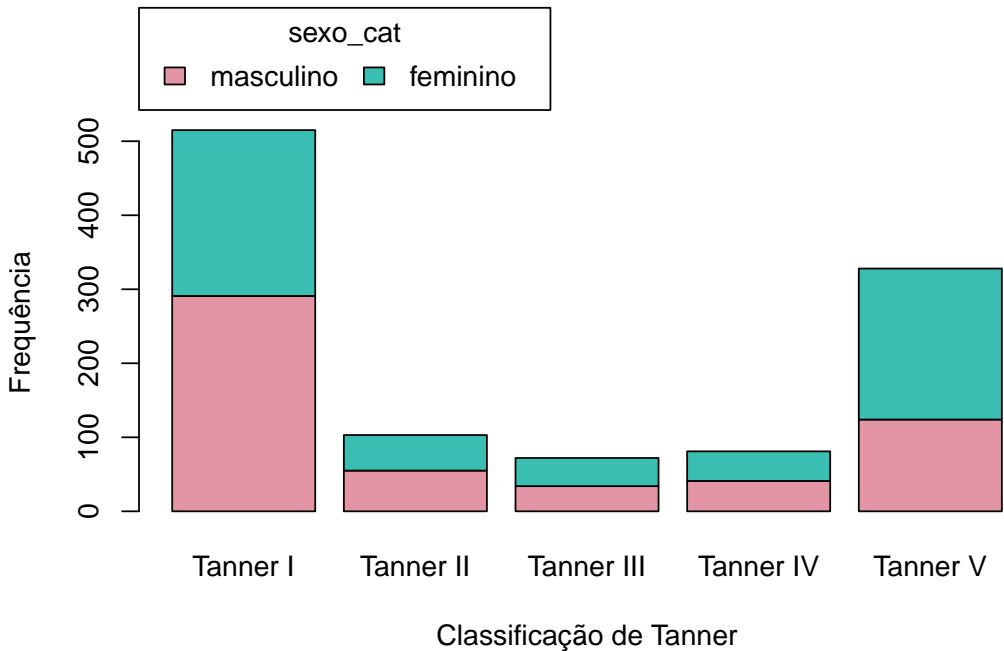


Figura 8.15: Diagrama de barras empilhadas das frequências das categorias da variável *sexo_cat* para cada categoria da variável *tanner_cat*.

Vamos supor que queiramos criar um diagrama de barras com percentuais de cada categoria, em vez de frequências. Na caixa de diálogo de opções do comando para a geração de um diagrama de barras, selecionamos as opções como mostrado na figura 8.16. Observem que foram selecionadas as opções *Percentagens* em *Escala do eixo*, e *Total* em *Percentages for Group Bars*.



Figura 8.16: Configuração para gerar um diagrama de barras com percentagens do total para as categorias da variável *sexo_cat* para cada categoria da variável *tanner_cat*.

O comando gerado pelo *R Commander* é mostrado abaixo e o gráfico é apresentado na figura 8.17. Para cada categoria de Tanner, as barras mostram o percentual do total de observações para cada sexo naquela categoria. A soma de todos os percentuais é igual a 100%.

```
with(juul2, Barplot(tanner_cat, by=sexo_cat, style="parallel",
  legend.pos="above", xlab="Classificação de Tanner",
  ylab="Porcentagens", conditional=FALSE, label.bars=TRUE,
  main="Diagrama de Barras de Sexo por Categoria de Tanner",
  scale="percent"))
```

Diagrama de Barras de Sexo por Categoria de Tanner

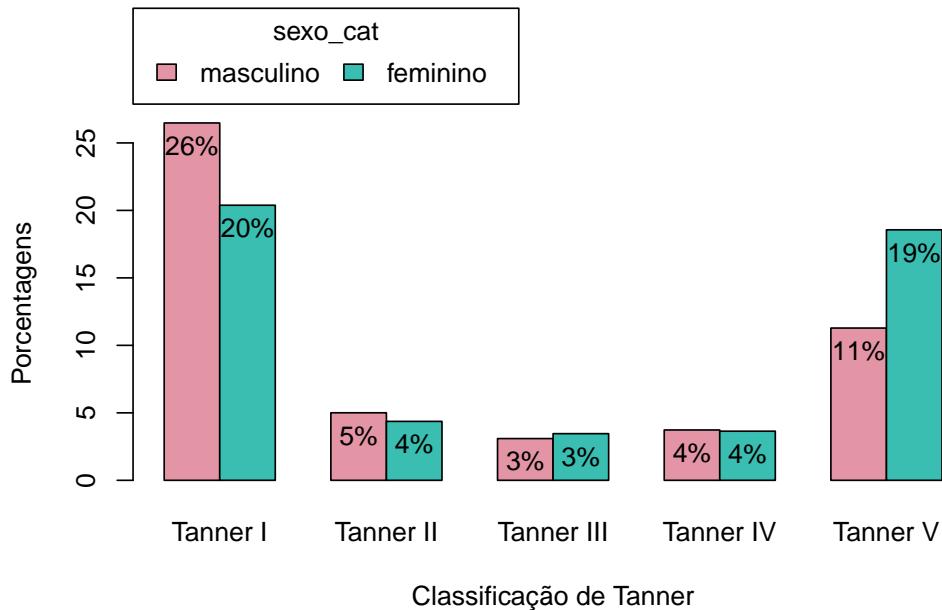


Figura 8.17: Diagrama de barras lado a lado para as porcentagens do total de observações de cada categoria de *sexo_cat* por categoria da variável *tanner_cat*.

Caso tivéssemos selecionado a opção *Conditional* em *Percentages for Group Bars* na figura 8.16, obteríamos o diagrama da figura 8.18. Em cada categoria de Tanner, os percentuais de cada sexo somam 100%.

Diagrama de Barras Condisional de Sexo por Categoria de Tanner

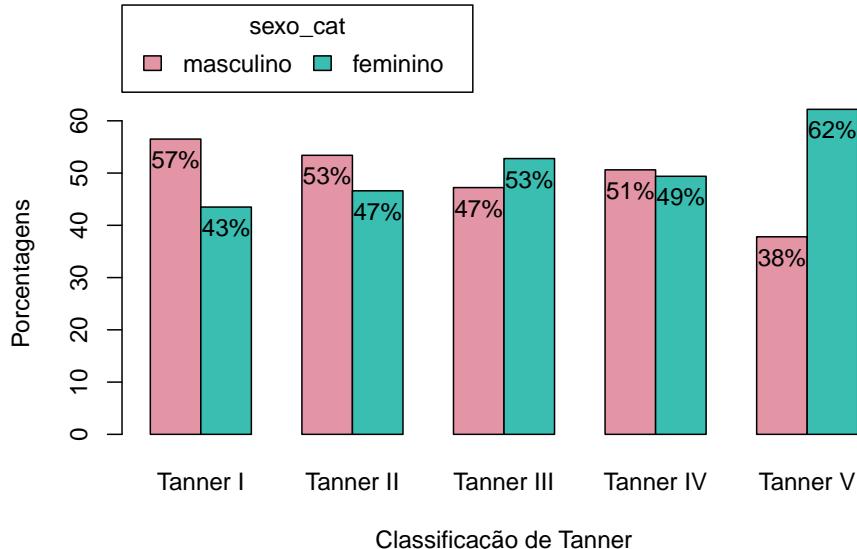


Figura 8.18: Diagrama de barras lado a lado para as porcentagens das categorias da variável *sexo_cat* em cada categoria da variável *tanner_cat*.

8.2 Usando a linha de comando

O conteúdo das seções 8.2.2, 8.2.3, 8.2.4, 8.2.5 e 8.2.6 podem ser visualizados neste [vídeo](#).

Vamos aproveitar o comando anterior e verificar como podemos alterar diversos outros aspectos do gráfico, alguns dos quais não podem ser configurados via menu do *R Commander*. Alguns argumentos apresentados nas funções a seguir são comuns a diversos tipos de gráficos, outros são específicos do diagrama de barras.

8.2.1 Especificação dos rótulos dos eixos x e y e do título

Os argumentos *xlab*, *ylab* e *main* descrevem respectivamente os rótulos dos eixos x, y e título dos gráficos. Eles são comuns a todos os gráficos exibidos pelo *R Commander*.

8.2.2 Alteração dos tamanhos dos eixos X e Y

O gráfico da figura 8.17 mostra a altura de uma barra maior do que o tamanho do eixo Y. O argumento *ylim* é usado para alterar os limites do eixo Y. Ele é especificado como um vetor de dois elementos, onde o primeiro elemento representa o limite inferior e o segundo elemento representa o limite superior do eixo.

No comando a seguir, é acrescentado o argumento *ylim = c(0, 30)* à função *Barplot*, e as alturas das barras ficarão menor do que o valor máximo do eixo Y (figura 8.19). Um argumento análogo, *xlim*, seria usado para o eixo X.

```
with(juul2, Barplot(tanner_cat, by=sexo_cat, style="parallel",
                     legend.pos="above", xlab="Classificação de Tanner",
                     ylab="Porcentagens", conditional=FALSE, label.bars=TRUE,
                     main="Diagrama de Barras de Sexo por Categoria de Tanner",
                     scale="percent", ylim = c(0, 30)))
```

Diagrama de Barras de Sexo por Categoria de Tanner

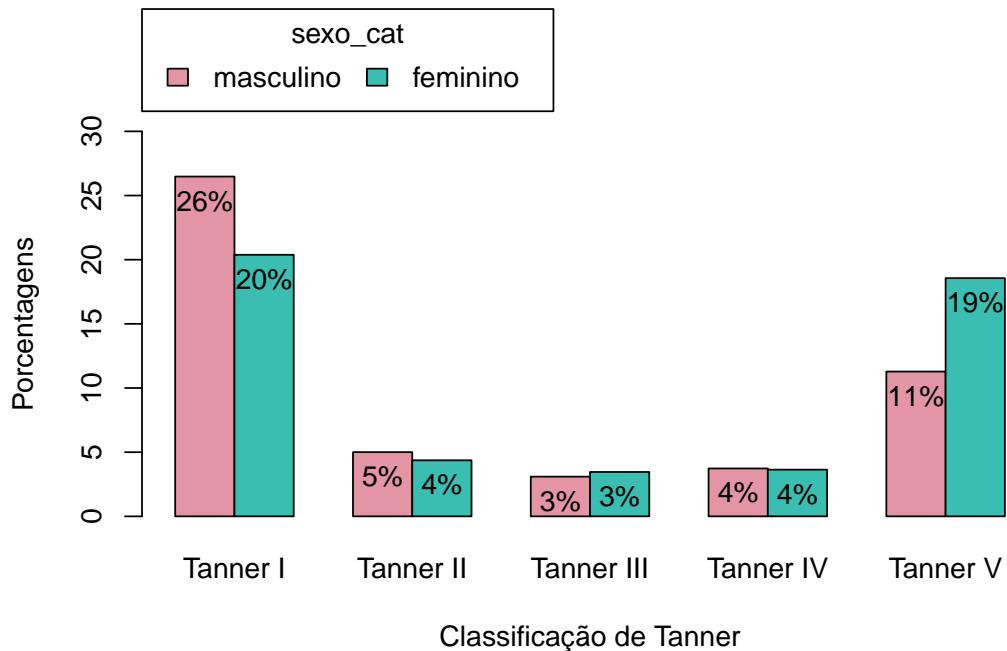


Figura 8.19: Diagrama de barras lado a lado para as porcentagens do total de observações de cada categoria de *sexo_cat* por categoria da variável *tanner_cat*, com a altura do eixo Y alterada para uma altura maior do que a altura da maior barra.

8.2.3 Alteração do título da legenda do diagrama

O argumento *legend.title* permite a alteração do título da legenda de um gráfico. O comando a seguir substitui o título original da legenda (“*sexo_cat*”) do gráfico 8.19 por “Sexo”. O resultado é mostrado na figura 8.20.

```
with(juul2, Barplot(tanner_cat, by=sexo_cat, style="parallel",
                     legend.pos="above", xlab="Classificação de Tanner",
                     ylab="Porcentagens", conditional=FALSE, label.bars=TRUE,
                     main="Diagrama de Barras de Sexo por Categoria de Tanner",
                     scale="percent", ylim = c(0, 30), legend.title = "Sexo"))
```

Diagrama de Barras de Sexo por Categoria de Tanner

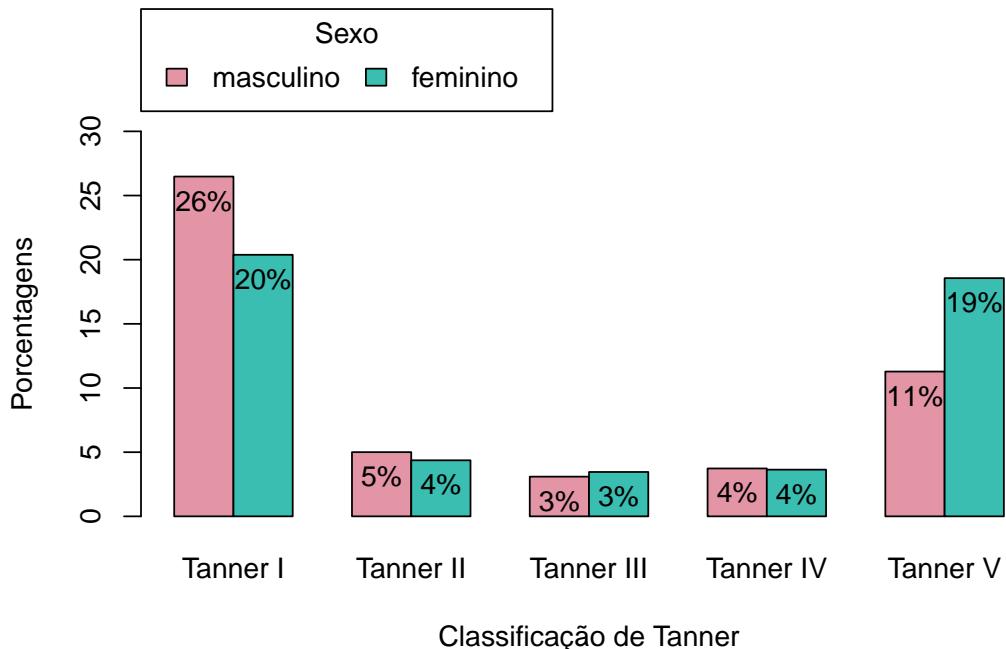


Figura 8.20: Diagrama de barras lado a lado para as porcentagens do total de observações de cada categoria de *sexo_cat* por categoria da variável *tanner_cat*, com a alteração do título da legenda.

8.2.4 Alteração do espaçamento entre as barras

O argumento *space* é utilizado para especificar o espaçamento entre as barras. Indiretamente, ele define a largura das barras. O espaçamento é expresso como uma fração da largura das barras. *space* pode ser expresso por dois números, sendo o primeiro o espaçamento entre as barras de um mesmo grupo e o segundo o espaçamento entre os grupos. Se *space* não for especificado, os seguintes valores são assumidos por padrão:

- 1) se o diagrama de barras incluir mais de um fator e as barras forem desenhadas lado a lado, então $space = c(0,1)$, ou seja, o espaçamento entre os grupos de barras é igual à largura das mesmas;
- 2) para os demais casos $space = 0.2$, significando que o espaçamento entre as barras é igual a 20% da largura das barras.

O comando a seguir altera o espaçamento das barras para 20% da largura da barra dentro de cada grupo e para 1,5 vezes a largura da barra para a separação entre os grupos (figura 8.21).

```

with(juul2, Barplot(tanner_cat, by=sexo_cat, style="parallel",
  legend.pos="above", xlab="Classificação de Tanner",
  ylab="Porcentagens", conditional=FALSE, label.bars=TRUE,
  main="Diagrama de Barras de Sexo por Categoria de Tanner",
  scale="percent", ylim = c(0, 30), legend.title = "Sexo",
  space = c(.2, 1.5)))

```

Diagrama de Barras de Sexo por Categoria de Tanner

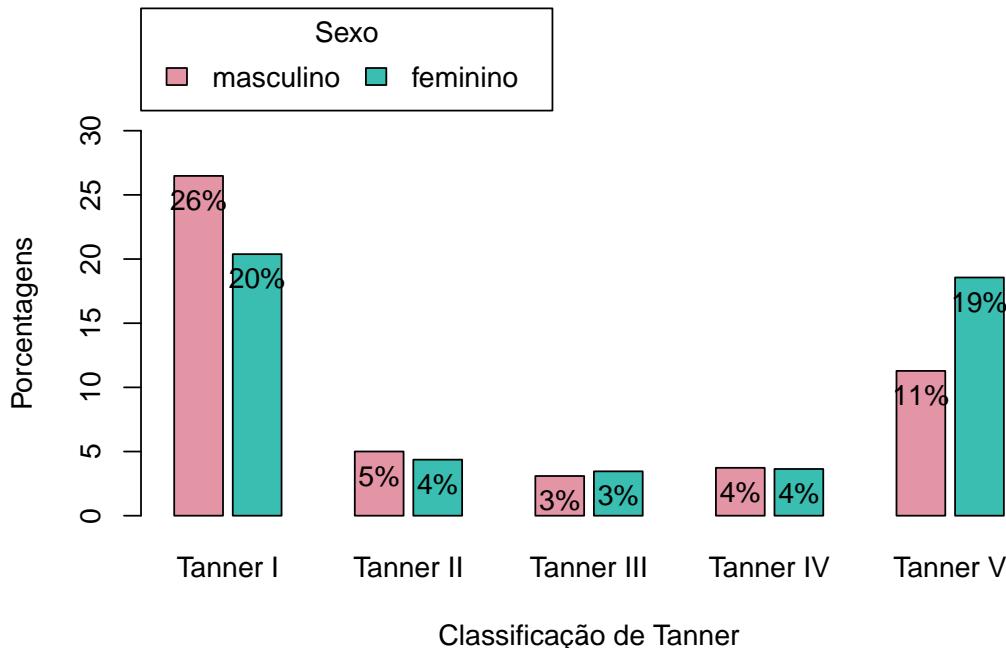


Figura 8.21: Diagrama de barras lado a lado para as porcentagens do total de observações de cada categoria de *sexo_cat* por categoria da variável *tanner_cat*, com a alteração do espaçamento entre as barras.

8.2.5 Tamanhos dos rótulos dos eixos X e Y, dos números no eixo Y e das categorias das barras

Para alterar o tamanho dos rótulos que aparecem nos eixos X e Y, o tamanho das categorias da variável do eixo X e o tamanho dos números das escalas dos eixos X e Y, são usados os argumentos *cex.lab*, *cex.names*, e *cex.axis*, respectivamente, sendo o número 1 o padrão.

Ao fazermos os argumentos *cex.lab* = 1.8, *cex.names* = 1.2 e *cex.axis* = 1.5 na função *Barplot* (comando a seguir), o gráfico resultante mostra os rótulos dos eixos X e Y com tamanho 80% maior, as categorias de Tanner 20% maiores, e os números das escalas do eixo Y 50% maiores (figura 8.22).

```
with(juul2, Barplot(tanner_cat, by=sexo_cat, style="parallel",
  legend.pos="above", xlab="Classificação de Tanner",
  ylab="Porcentagens", conditional=FALSE, label.bars=TRUE,
  main="Diagrama de Barras de Sexo por Categoria de Tanner",
  scale="percent", ylim = c(0, 30), legend.title = "Sexo",
  space = c(.2, 1.5),
  cex.lab = 1.8, cex.names = 1.2, cex.axis = 1.5))
```

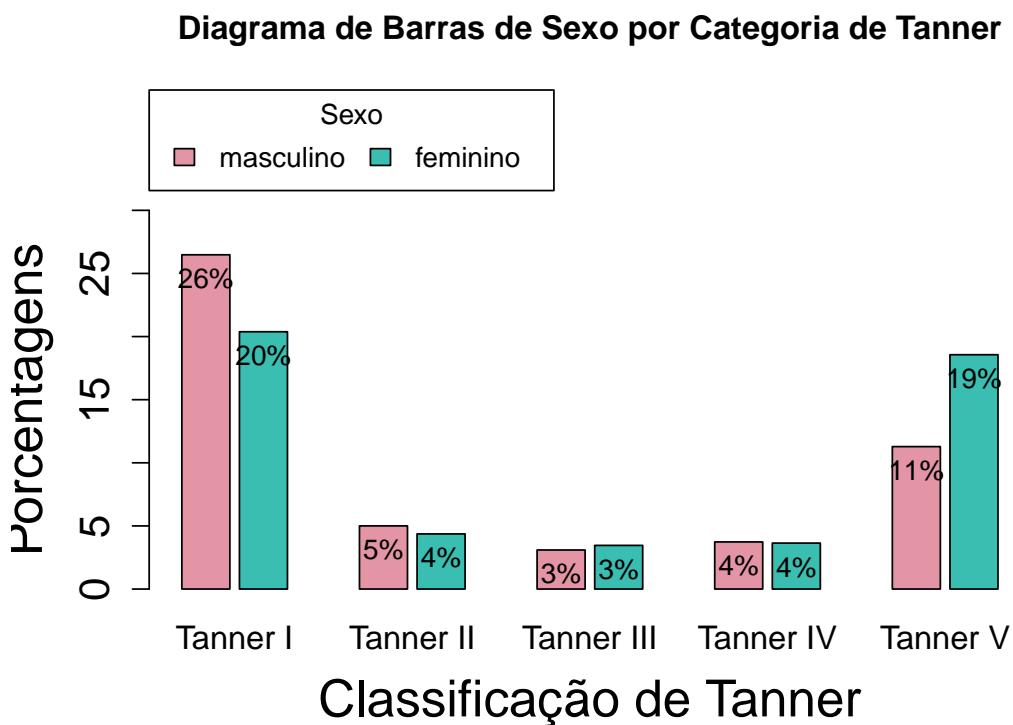


Figura 8.22: Diagrama de barras lado a lado para as porcentagens do total de observações de cada categoria de *sexo_cat* por categoria da variável *tanner_cat*, com alterações dos tamanhos dos rótulos dos eixos, das categorias de Tanner e da escala do eixo Y.

8.2.6 Alteração das categorias da variável do eixo X

Os rótulos das categorias da variável do eixo X podem ser alterados por meio do argumento *names.arg*, o qual deve ser especificado como um vetor com um valor do tipo *character* para cada categoria, ou seja, cada categoria deve ser especificada entre aspas.

O comando a seguir altera os nomes das categorias de Tanner de I a V para ‘T1’, ‘T2’, ‘T3’, ‘T4’, e ‘T5’, respectivamente. O gráfico resultante é mostrado na figura 8.23.

```

with(juul2, Barplot(tanner_cat, by=sexo_cat, style="parallel",
  legend.pos="above", xlab="Classificação de Tanner",
  ylab="Porcentagens", conditional=FALSE, label.bars=TRUE,
  main="Diagrama de Barras de Sexo por Categoria de Tanner",
  scale="percent", ylim = c(0, 30), legend.title = "Sexo",
  space = c(.2, 1.5),
  cex.lab = 1.8, cex.names = 1.2, cex.axis = 1.5,
  names.arg = c('T1', 'T2', 'T3', 'T4', 'T5')))

```

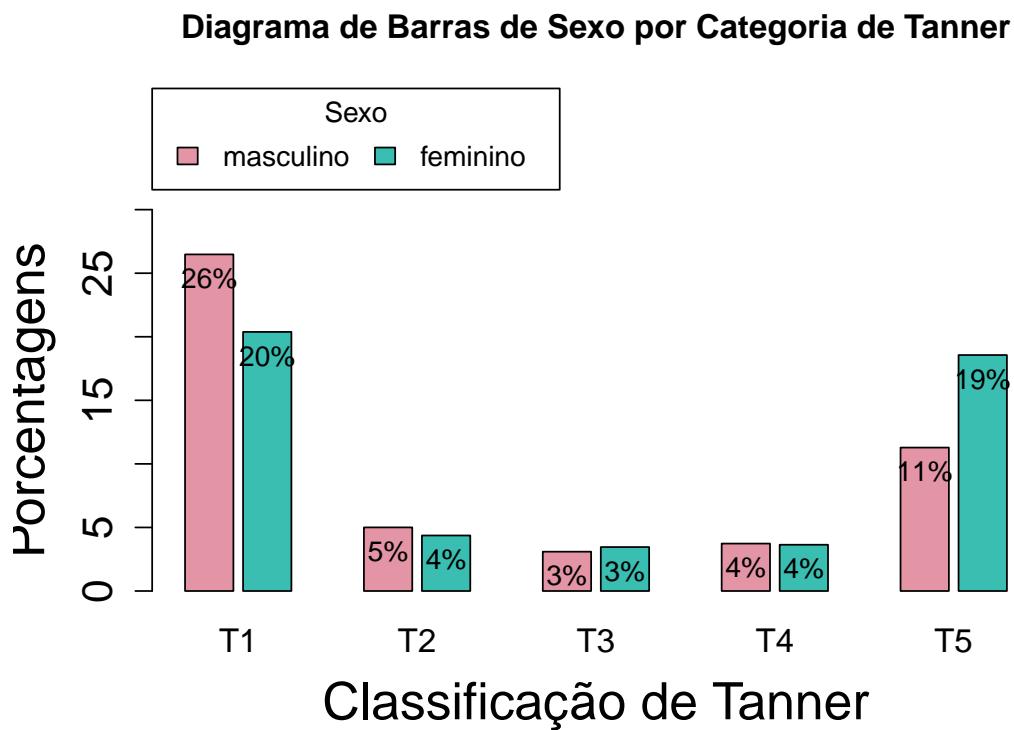


Figura 8.23: Diagrama de barras lado a lado para as porcentagens do total de observações de cada categoria de *sexo_cat* por categoria da variável *tanner_cat*, com a alteração dos nomes das categorias de Tanner.

8.2.7 Alteração das cores

O conteúdo desta seção pode ser visualizado neste [vídeo](#).

O gráfico exibido na figura 8.11 mostra as barras na cor cinza. Já o gráfico da figura 8.17 mostra as barras nas cores rosa para o sexo masculino e verde para o feminino.

A função *Barplot*, por padrão, mostra as barras em cinza se um fator somente for especificado. Se dois fatores forem especificados, o segundo por meio do argumento *by*, as cores das barras são obtidas por meio da função *rainbow_hcl* do pacote *colorspace*.

Como fazer para alterar as cores das barras? O argumento *col* é usado para alterar as cores. Vamos ver algumas possibilidades.

No comando a seguir, foi acrescentado o argumento `col` com o valor `c("blue", "orange")`, indicando as duas cores que serão utilizadas agora. Também foi acrescentado o argumento `ylim = c(0,30)` para alterar os limites do eixo Y. Ao executarmos o comando, selecionando todas as linhas do comando e clicando no botão *Submeter*, obteremos o gráfico da figura 8.24.

```
with(juul2, Barplot(tanner_cat, by=sexo_cat, style="parallel",
                     legend.pos="above", xlab="Classificação de Tanner",
                     ylab="Porcentagens", conditional=FALSE, label.bars=TRUE,
                     main="Diagrama de Barras de Sexo por Categoria de Tanner",
                     scale="percent", ylim = c(0, 30), col = c("blue", "orange")))
```

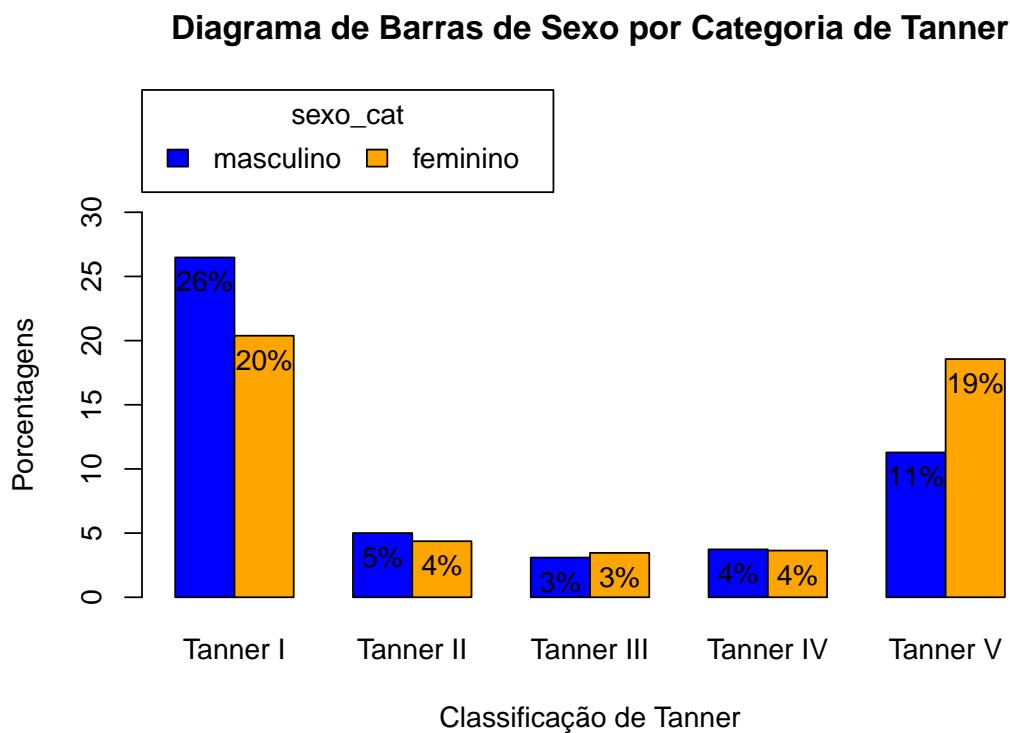


Figura 8.24: Diagrama de barras da figura 8.19, após a alteração das cores das barras.

Caso desejemos alterar as cores das barras em um diagrama de barras simples, sem variável de agrupamento, há diversas opções. Na aba de opções da figura 8.10, podemos marcar a opção *From color palette* no item *Color Selection*. O comando resultante é mostrado abaixo e o gráfico com barras vermelhas para cada categoria de Tanner é mostrado na figura 8.25.

```
with(juul2, Barplot(tanner_cat, xlab="Classificação de Tanner",
                     ylab="Frequência", label.bars=TRUE,
                     main="Diagrama de Barras da Classificação de Tanner",
                     col=palette()[2])
      )
```

Diagrama de Barras da Classificação de Tanner

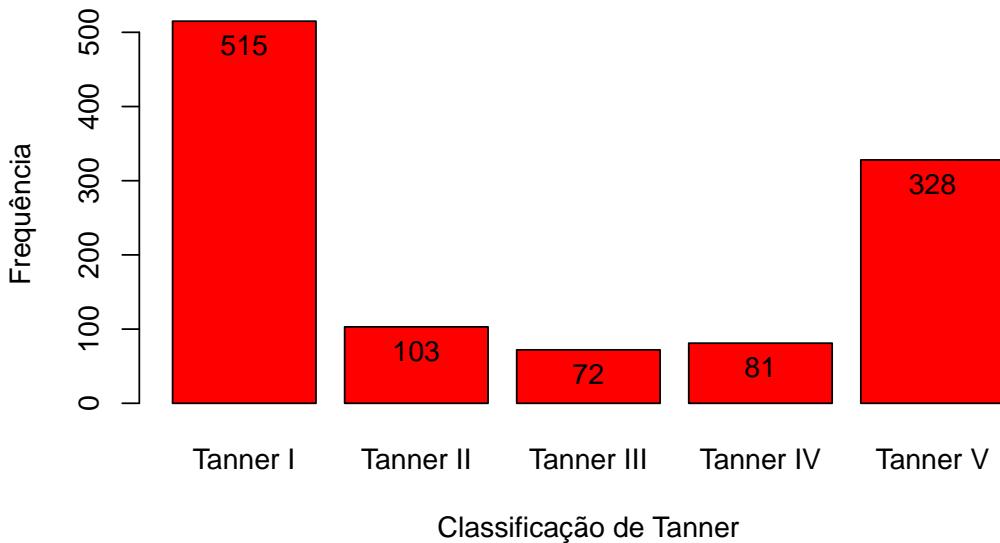


Figura 8.25: Diagrama de barras da figura 8.11, com as barras vermelhas.

A cor vermelha foi selecionada, porque o argumento *col* no comando anterior especificou a segunda cor da paleta de cores do *R Commander* (*palette()[-2]*). A função *palette* retorna a paleta de cores do *R Commander* e o índice 2 indica a segunda cor desta paleta.

A figura 8.26 mostra a paleta de cores padrão do *R Commander*. Ela pode ser acessada pela opção do menu:

Gráficos ⇒ Gradiente de cores (color palette)



Figura 8.26: Paleta de cores do *R Commander*. Ao clicarmos na cor indicada pela seta verde, podemos substituí-la por outra.

Se quiséssemos a quarta cor da paleta nas barras, faríamos *col = palette()[-4]* na chamada da função *Barplot*, ou simplesmente *col = 4*.

Podemos alterar cada cor dessa paleta, bastando clicar na cor que desejamos alterar. Se clicarmos na cor indicada pela seta verde na figura 8.26, poderemos alterá-la por meio da

caixa de diálogo *Select a color* (figura 8.27). Após selecionarmos a cor e clicarmos no botão OK, a paleta será alterada (figura 8.28).

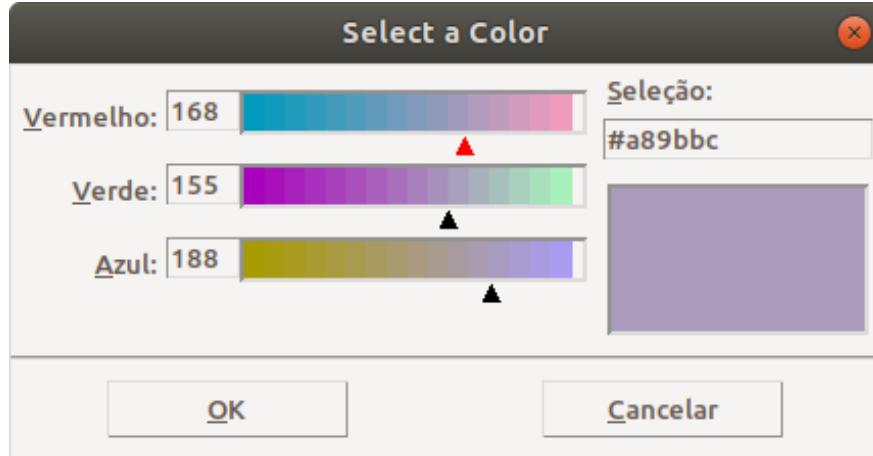


Figura 8.27: Caixa de diálogo para alterar uma cor da paleta.



Figura 8.28: Paleta com a última cor alterada.

Se fizermos `col=c(2:6)` para gerar o diagrama de barras das categorias de Tanner, obteremos o comando a seguir, que gera barras com cores diferentes (figura 8.29).

```
with(juul2, Barplot(tanner_cat, xlab="Classificação de Tanner",
                     ylab="Frequência", label.bars=TRUE,
                     main="Diagrama de Barras da Classificação de Tanner",
                     col=c(2:6)))
```

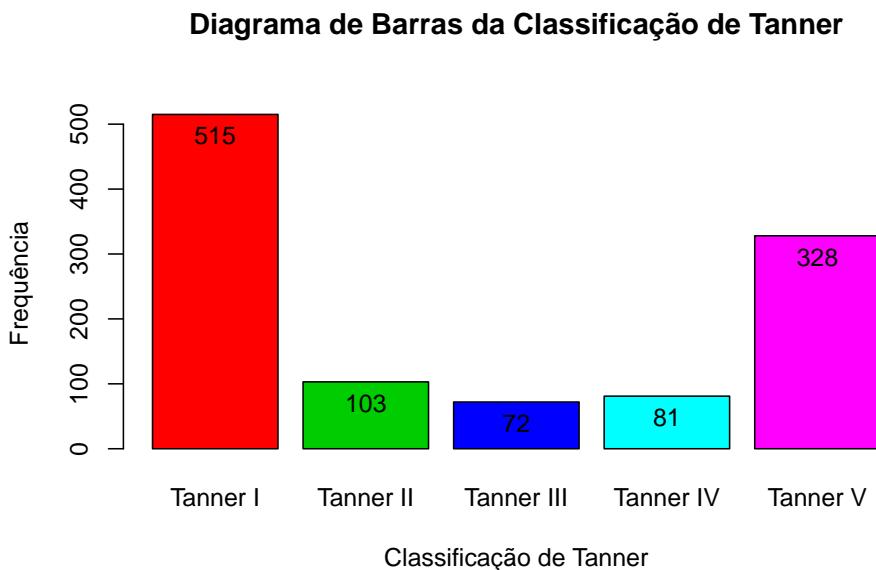


Figura 8.29: Diagrama de barras da figura 8.11, com barras de cores diferentes.

Observem que as cores foram especificadas como `c(2:6)`, significando que foram utilizadas as cores de números 2 a 6 da paleta corrente. Cada barra será de uma cor. Experimentem executar o comando. Outra possibilidade seria especificarmos as cores pelos nomes como mostrado a seguir (figura 8.30).

```
with(juul2, Barplot(tanner_cat, xlab="Classificação de Tanner",
                     ylab="Frequência", label.bars=TRUE,
                     main="Diagrama de Barras da Classificação de Tanner",
                     col=c("red", "green", "blue", "cyan", "yellow")))
```

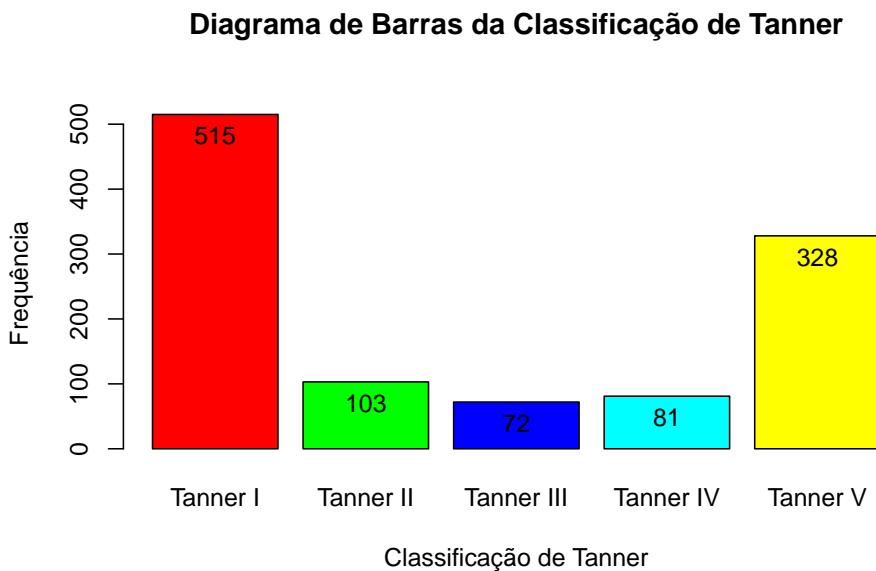


Figura 8.30: Diagrama de barras da figura 8.11, com barras de cores diferentes, especificadas pelo nome.

Observação: As interfaces gráficas não oferecem todos os recursos de cada função gráfica. Para conhecermos os argumentos disponíveis para cada função, usamos `help(nome_da_função)`. A internet é uma excelente fonte de ajuda para entender como conseguir os efeitos desejados.

8.2.8 Gráfico de barras horizontais

Para plotar as barras na direção horizontal, usa-se o argumento `horiz = TRUE`.

8.3 Diagrama de setores, torta ou pizza

O conteúdo desta seção pode ser visualizado neste [vídeo](#).

O diagrama de setores também é utilizado para a visualização de variáveis categóricas. Nesse diagrama, um círculo é dividido em fatias, onde a área de cada fatia é proporcional à frequência de cada categoria da variável no conjunto de dados. Essa informação também é transmitida pelo diagrama de barras.

Para construirmos um diagrama de setores no *R Commander*, selecionamos a opção do menu:

Gráficos ⇒ Gráfico de Pizza

Em seguida, selecionamos a variável categórica para a qual o diagrama será construído, digitamos um título para o gráfico e clicamos em OK (figura 8.31). A figura 8.32 mostra o gráfico resultante.

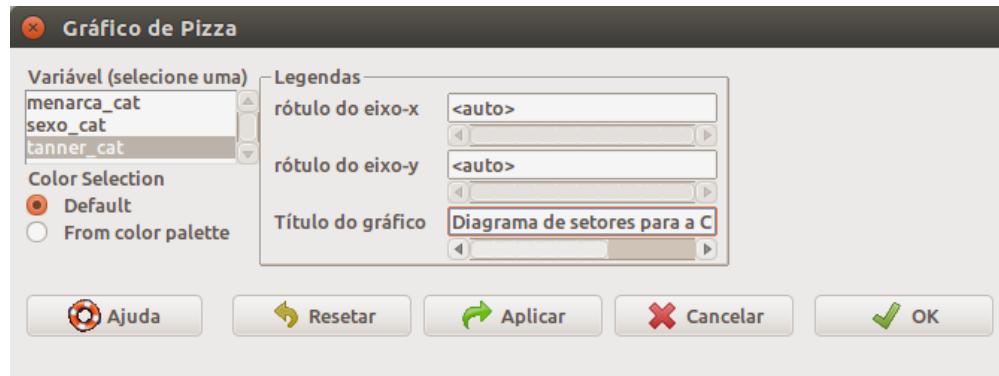


Figura 8.31: Caixa de diálogo para a geração de um diagrama de setores. Selecione a variável e digitamos o título do gráfico (opcional).

Observem que o gráfico da figura 8.32 não mostra os percentuais ou as frequências de cada fatia, apesar de dar uma ideia da frequência relativa (ou porcentagens) de cada categoria no conjunto de dados.

Diagrama de Setores para a Classificação de Tanner

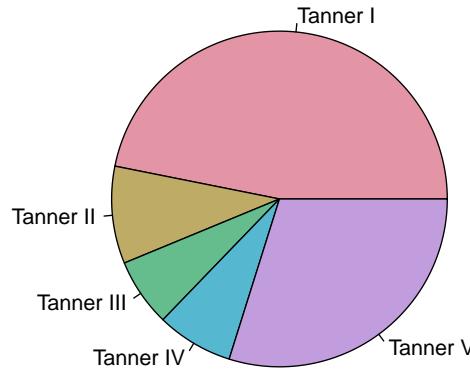


Figura 8.32: Diagrama de setores das categorias de Tanner.

Para obter um gráfico com as frequências (ou percentuais) de cada categoria, podemos utilizar a sequência de comandos a seguir, e o resultado é o gráfico da figura 8.33.

```
par(mar=c(1,1,1,1))
frequencias <- as.vector(with(juul2, table(tanner_cat)))
piepercent<- paste(as.character(
    round(100*frequencias/sum(frequencias), 1)), "%")
with(juul2, pie(table(tanner_cat), labels=piepercent,
    main="Classificação de Tanner",
    col=c(1:length(levels(tanner_cat)))))
legend(.9, .1, legend=c(levels(juul2$tanner_cat)), cex = 0.7,
    fill = c(1:length(levels(juul2$tanner_cat)) ))
```

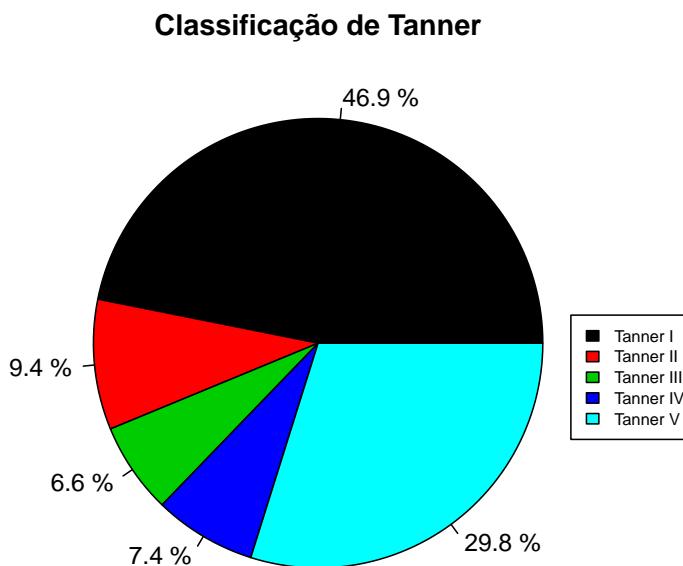


Figura 8.33: Diagrama de setores das categorias de Tanner, com os percentuais de cada categoria.

Vamos entender a sequência de comandos para gerar o gráfico de setores com porcentagens:

- 1) criamos um vetor com as frequências de cada categoria com o comando a seguir:

```
frequencias <- as.vector(with(juul2, table(tanner_cat)))
```

A função *table* cria uma tabela com a frequência de cada categoria de Tanner no conjunto de dados.

- 2) a partir dessas frequências, criamos um outro vetor (chamado *piepercent*) que fornece os percentuais de cada categoria, com o comando a seguir:

```
piepercent <- paste(as.character(round(100*frequencias/sum(frequencias), 1)), "%")
```

A função *round* arredonda o número especificado no primeiro argumento de acordo com o número de decimais especificado pelo segundo argumento. A função *as.character* transforma o número arredondado para *character* e *paste* concatena essa string com o sinal de porcentagem.

- 3) Então usamos o comando gerado pelo *R Commander* para a criação do diagrama e o modificamos conforme a seguir. Observem que alteramos o argumento *labels*, substituindo o seu valor por *piepercent*:

```
with(juul2, pie(table(tanner_cat), labels=piepercent, main="Classificação de Tanner", col=c(1:length(levels(tanner_cat)))))
```

- 4) Finalmente adicionamos uma legenda, indicando as cores de cada categoria com o comando a seguir:

```
legend(.9, .1, legend=c(levels(juul2$tanner_cat)), cex = 0.7, fill = c(1:length(levels(juul2$tanner_cat))))
```

Se for desejado criar um gráfico de pizza com frequências em vez de percentuais, basta alterar o comando do passo 3 acima, substituindo o valor do argumento *labels*, como a seguir. Lembrem-se de que o vetor *frequencias* foi obtido no passo 1. O gráfico é exibido na figura 8.34.

```
par(mar=c(1,1,1,1))
with(juul2, pie(table(tanner_cat), labels=frequencias,
                 main="Classificação de Tanner",
                 col=c(1:length(levels(tanner_cat)) )))
legend(.9, .1, legend=c(levels(juul2$tanner_cat)), cex = 0.7,
       fill = c(1:length(levels(juul2$tanner_cat))))
```

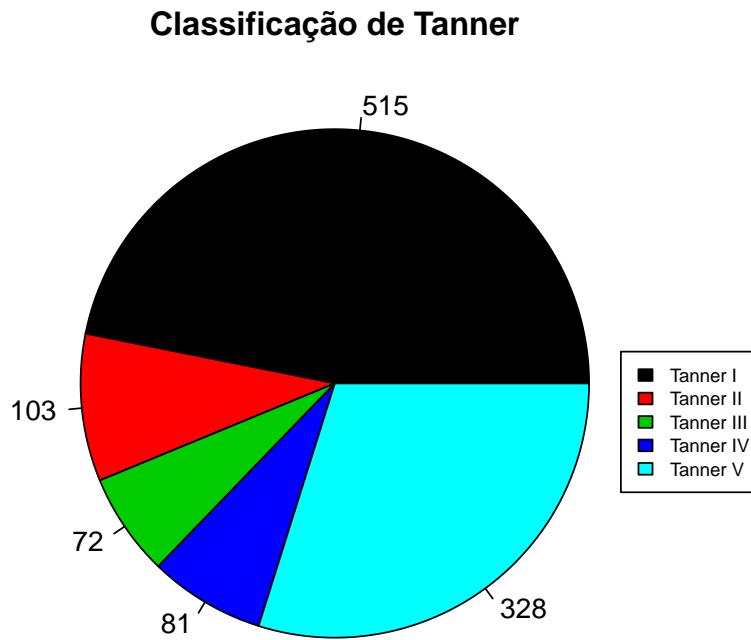


Figura 8.34: Diagrama de setores das categorias de Tanner, com as frequências de cada categoria.

A partir da próxima seção, serão mostrados recursos para a visualização da distribuição dos valores de variáveis numéricas.

8.4 Diagrama de caixa (*boxplot* ou *box and whisker plot*)

O conteúdo desta seção pode ser visualizado neste [vídeo](#), seguido deste [vídeo](#).

O diagrama de caixa (em inglês, *box and whisker plot*, ou simplesmente *boxplot*) é um dos mais úteis diagramas para visualizar a distribuição de dados numéricos. Para explicar como o mesmo é construído, vamos criar um diagrama de *boxplot*, selecionando a opção:

Gráficos ⇒ Boxplot

A figura 8.35 mostra a tela de configuração do *boxplot*. Na aba *Dados*, selecionamos a variável. Neste exemplo, selecionamos a variável *igf1* (fator de crescimento parecido com a insulina tipo 1). Na aba *Opções*, digitamos um título para o gráfico e marcamos a opção de não identificar os *outliers* (figura 8.36). O gráfico é mostrado na figura 8.37.



Figura 8.35: Caixa de diálogo para a geração do *boxplot*. Nesse exemplo, estamos selecionando a variável *igf1*.



Figura 8.36: Aba *Opções* da caixa de diálogo para a geração do *boxplot*.

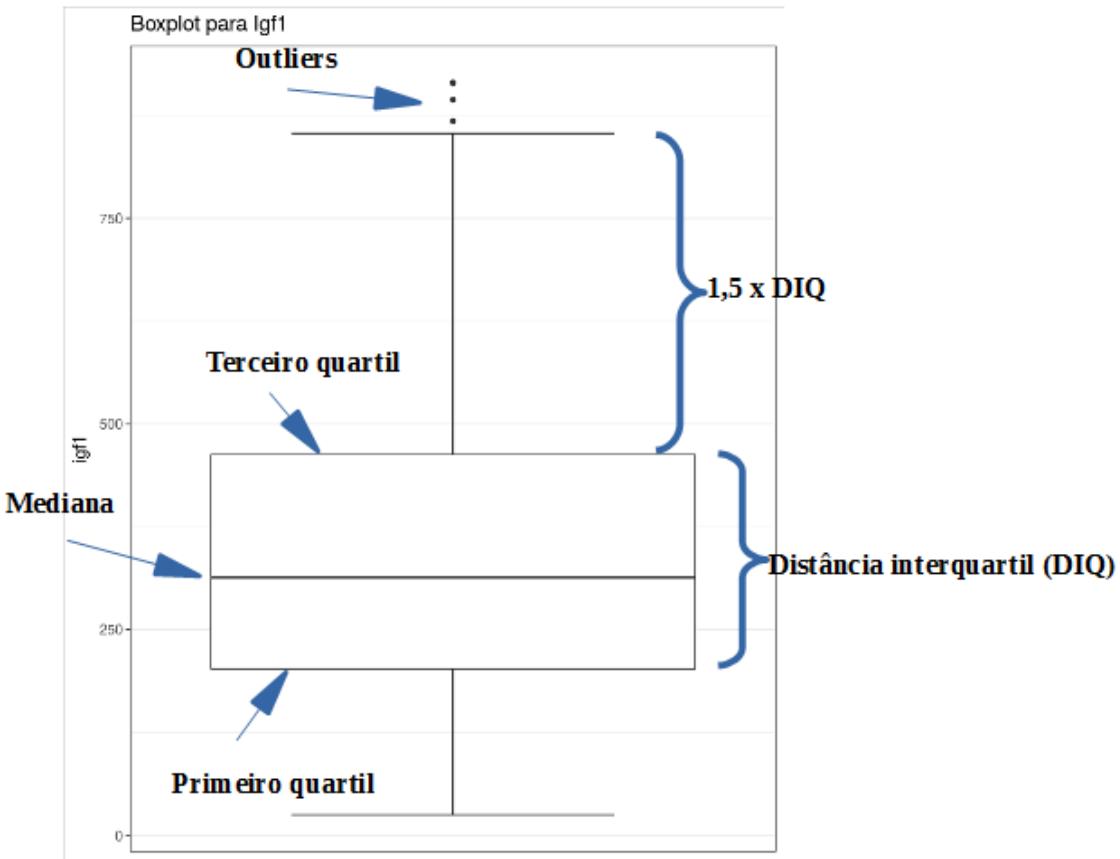


Figura 8.37: *Boxplot* da variável *Igf1*.

O *boxplot* consiste de uma caixa cuja linha inferior indica o valor do primeiro quartil da variável, e a linha superior indica o terceiro quartil. Logo a altura da caixa indica a distância interquartil (DIQ). Uma terceira linha horizontal, a mediana, divide a caixa em duas partes. Partindo do meio da linha superior da caixa, uma linha vertical (*whisker* = bigode) liga o terceiro quartil ao valor imediatamente inferior ou igual ao valor do terceiro quartil somado a $1,5 \times \text{DIQ}$. Valores acima do *whisker* são considerados *outliers* e indicados por pontos. De maneira semelhante, uma linha vertical parte do meio da linha inferior da caixa e liga o primeiro quartil ao valor imediatamente acima ou igual ao primeiro quartil subtraído de $1,5 \times \text{DIQ}$. Pontos inferiores a esse valor também seriam considerados *outliers* e representados por pontos. Nesse exemplo, o menor valor de *Igf1* está a uma distância do primeiro quartil menor que $1,5 \times \text{DIQ}$. Por isso o *whisker* inferior não possui o mesmo tamanho do superior e não aparece *outliers* na porção inferior do diagrama.

O *boxplot* fornece diversas informações: os valores do primeiro e terceiro quartis, a mediana, a simetria ou assimetria dos dados e a presença de outliers. No diagrama da figura 8.37, verificamos uma certa assimetria dos dados de *Igf1* e a presença de *outliers*.

É possível construir os *whiskers* com diferentes tamanhos do que aqui mostrado, assim como podem ser usados a média e desvio padrão para construir os limites da caixa. Porém o método apresentado acima é o mais utilizado e é o padrão na função *Boxplot*.

O *boxplot* da figura 8.37 mostra a distribuição de todos os valores de *igf1*. Podemos construir um *boxplot* de *igf1* para cada categoria da classificação de Tanner, ou por sexo, ou por cada combinação de sexo e classificação de Tanner.

Para mostrarmos o *boxplot* de *igf1* para cada categoria da classificação de Tanner, clicamos no botão *Gráfico por grupos...* na caixa de diálogo do *boxplot* (figura 8.35) e selecionamos a variável *tanner_cat* para compor os grupos. Na aba *Opções* (figura 8.36), podemos digitar uma legenda para o eixo X. O resultado é mostrado na figura 8.38.

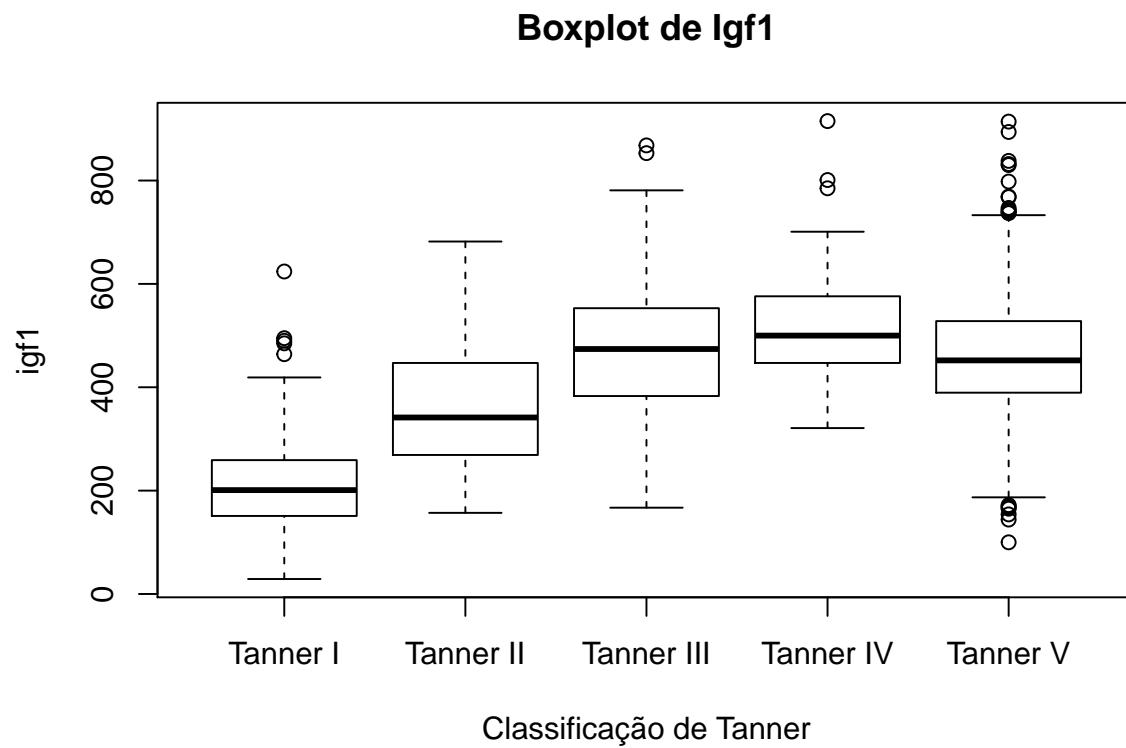


Figura 8.38: *Boxplots* para a variável *igf1* para cada categoria de Tanner.

Esse gráfico nos fornece uma visão melhor de como os valores de *igf1* estão distribuídos. À medida que as pessoas vão crescendo, os valores de *igf1* tendem a aumentar até o nível III da classificação de Tanner, tendendo a se estabilizarem a partir desta categoria. As distribuições dos valores de *igf1* tendem a ser simétricas nas categorias III e V de Tanner e ligeiramente assimétricas nas demais categorias.

Observação: Devemos ter cautela, porém, com as afirmações do parágrafo anterior, porque os dados de *igf1* foram coletados a partir de um estudo transversal. Um conjunto de pessoas foram selecionadas e os valores de idade e *igf1* foram coletados para cada uma delas. Um estudo mais apropriado para verificar a dependência de *igf1* com a idade seria um estudo longitudinal, onde um grupo de pessoas fosse acompanhado ao longo do tempo e os valores de *igf1* fossem medidos em diversos instantes para cada indivíduo à medida que ele ou ela fosse envelhecendo.

8.5 Histograma

O conteúdo desta seção pode ser visualizado neste [vídeo](#).

Os histogramas, ao lado dos *boxplots*, são os gráficos mais utilizados para visualizarmos dados numéricos. Os histogramas são construídos, agrupando os valores numéricos da variável em faixas de valores e desenhandando uma barra com largura igual ao tamanho da faixa e com altura, por exemplo, igual à frequência relativa de valores (percentual de valores) na faixa correspondente. As faixas são contíguas e o conjunto de barras compõem o histograma.

Para construirmos um histograma no *R Commander*, selecionamos a opção:

Gráficos ⇒ Histograma

Em seguida, selecionamos a variável desejada, *igf1* nesse exemplo (figura 8.39). Na aba *Opções* (figura 8.40), vamos selecionar *percentagens* em *Escala do eixo* e digitar a legenda do eixo Y. Ao clicarmos em OK, o gráfico resultante é mostrado na figura 8.41.



Figura 8.39: Caixa de diálogo para a criação de um histograma. Na aba *Dados*, selecionamos a variável numérica desejada.



Figura 8.40: Caixa de diálogo para a criação de um histograma. Na aba *Opções*, podemos especificar o número de faixas de valores (classes), a escala do eixo e as legendas.

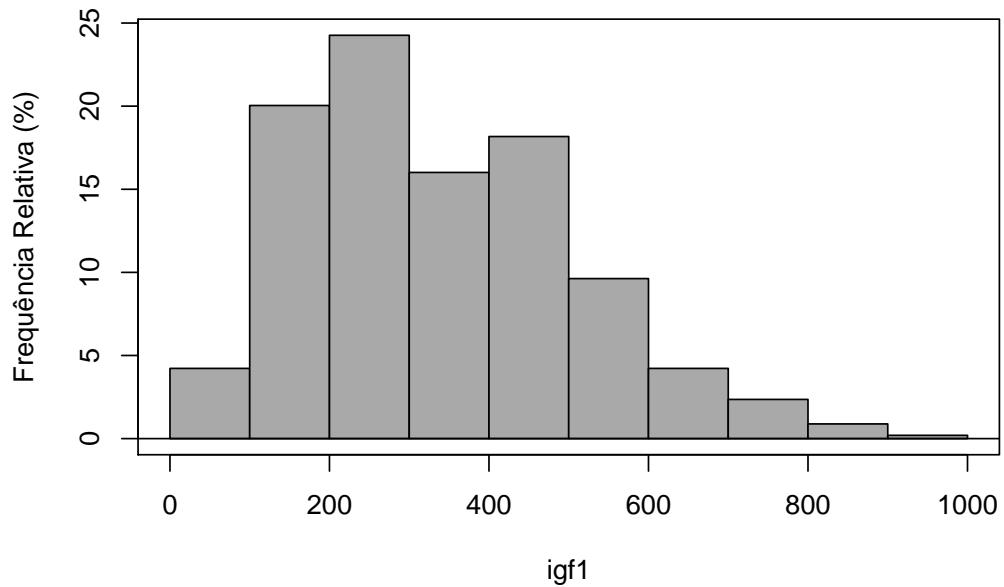


Figura 8.41: Histograma de frequência relativa da variável *igf1*.

Os passos para a construção de um histograma são:

- 1) definir um conjunto exaustivo de faixas de valores para a variável em questão. Cada faixa de valores é usualmente denominada classe. No exemplo acima, os valores de *igf1* foram distribuídos em 10 classes de amplitude 100 cada uma;
- 2) para cada classe, calcular a frequência dos valores nela contidas;
- 3) no caso de um histograma de frequência relativa, dividir a frequência de cada classe pelo número total de valores;
- 4) plotar uma barra para cada classe, com altura igual à frequência relativa (ou a mesma multiplicada por 100 para mostrar os percentuais de cada classe).

No exemplo dado, os limites das classes são:

0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000

Os valores da primeira classe são aqueles que pertencem ao intervalo $(0, 100]$. Desse modo, todos os valores da primeira classe devem satisfazer a seguinte desigualdade: $0 < x \leq 100$. As demais classes devem ser interpretadas de maneira análoga. Essa é a forma padrão que a função *hist* do R monta as classes do histograma. A tabela 8.1 mostra, para cada classe, os seus limites, contagem de valores (frequência) e a correspondente frequência relativa em porcentagem.

Tabela 8.1: Definição das classes e frequência relativa de cada uma delas. Cada classe é definida por um intervalo da forma $(a, b]$, onde a é o limite inferior e b o limite superior.

| Classe | Limite Inferior
($>$) | Limite Superior
(\leq) | Frequência | Frequência Relativa (%) |
|--------|----------------------------|-------------------------------|------------|-------------------------|
| 1 | 0 | 100 | 43 | 4,22 |
| 2 | 100 | 200 | 204 | 20,04 |
| 3 | 200 | 300 | 247 | 24,26 |
| 4 | 300 | 400 | 163 | 16,01 |
| 5 | 400 | 500 | 185 | 18,17 |
| 6 | 500 | 600 | 98 | 9,63 |
| 7 | 600 | 700 | 43 | 4,22 |
| 8 | 700 | 800 | 24 | 2,36 |
| 9 | 800 | 900 | 9 | 0,88 |
| 10 | 900 | 1000 | 2 | 0,2 |
| | | | 1018 | 100 |

No exemplo apresentado, o número de classes utilizado para agrupar os valores de *igf1* foi 10, determinado automaticamente pela função de geração do histograma. Entretanto esse número pode ser escolhido pelo usuário: basta digitar o número de classes desejado na opção *Numero de classes* da figura 8.40. O número de classes não deve ser um número muito baixo, de modo que tenhamos uma visão muito grosseira da distribuição de dados, nem muito alto, de modo que cada classe tenha poucos valores. O número de classes deve fornecer uma boa ideia de como os dados estão distribuídos, geralmente um número entre 10 e 20.

O comando que foi utilizado para a criação do histograma da figura 8.41 é mostrado a seguir:

```
with(juul2, Hist(igf1, scale="percent", breaks="Sturges", col="darkgray",
                 ylab="Frequência Relativa (%)))
```

O argumento *breaks* indica como as classes serão definidas. O valor nesse exemplo, *Sturges*, indica o nome de um algoritmo utilizado para calcular o número de classes do histograma. Outros nomes de algoritmos são *Scott* e *Freedman-Diaconis*. Não vamos entrar em detalhes desses algoritmos. Além deles, o usuário pode especificar o nome de uma função qualquer

que calcule o número de classes, ou fixar o número de classes, ou mesmo especificar os limites das classes. Nós veremos essa última opção mais adiante.

8.5.1 Histograma de frequência x frequência relativa x densidade de frequência relativa

O conteúdo desta seção pode ser visualizado neste [vídeo](#).

Na seção anterior, criamos um histograma de frequência relativa para a variável *igf1*. Porém um histograma também pode ser criado, partindo-se da frequência ou da densidade de frequência relativa. Quando a amplitude de cada classe é a mesma, a aparência dos histogramas é a mesma, variando apenas a escala do eixo vertical.

Vamos alterar as classes do histograma para a variável *igf1* de modo que as suas amplitudes sejam diferentes e vamos ver as diferenças entre os três tipos de histogramas. A tabela 8.2 mostra as classes, a **frequência**, a **frequência relativa** e a **densidade de frequência relativa** para cada classe. Esse último termo será explicado mais adiante. Observem que as classes 1 e 10 possuem amplitude igual a 100, a classe 11 possui amplitude igual a 400 e as demais classes possuem amplitude igual a 50.

Tabela 8.2: Definição das classes de um histograma e respectivas frequências, frequências relativas e densidade de frequência relativa para a variável *igf1* do conjunto de dados *juul2*.

| Classe | Limite Inferior
($>$) | Limite Superior
(\leq) | Frequência | Frequência Relativa (%) | Densidade de Frequência Relativa ($\times 10^{-3}$) |
|--------|----------------------------|-------------------------------|------------|-------------------------|---|
| 1 | 0 | 100 | 43 | 4,22 | 0,42 |
| 2 | 100 | 150 | 74 | 7,27 | 1,45 |
| 3 | 150 | 200 | 130 | 12,77 | 2,55 |
| 4 | 200 | 250 | 129 | 12,67 | 2,53 |
| 5 | 250 | 300 | 118 | 11,59 | 2,32 |
| 6 | 300 | 350 | 69 | 6,78 | 1,36 |
| 7 | 350 | 400 | 94 | 9,23 | 1,85 |
| 8 | 400 | 450 | 93 | 9,14 | 1,82 |
| 9 | 450 | 500 | 92 | 9,04 | 1,80 |
| 10 | 500 | 600 | 98 | 9,63 | 0,96 |
| 11 | 600 | 1000 | 78 | 7,66 | 0,19 |
| | | | 1018 | 100 | |

A figura 8.42 mostra diversos histogramas para a variável *igf1*. Essa figura foi construída com a sequência de comandos a seguir:

```

par(mfrow = c(2,2))
with(juul2, Hist(igf1, scale="percent", breaks="Sturges", col="darkgray",
                 ylab="Frequênci Relativa (%)"))
text(-150, -80, labels="a)", pos = 1, xpd = T, cex = 1.5)
with(juul2, Hist(igf1, scale="frequency", freq = TRUE,
                  breaks=c(0,100,150,200,250,300,350,400,450,500, 600, 1000),
                  col="darkgray", ylab="Frequênci"))
text(-150, -40, labels="b)", pos = 1, xpd = T, cex = 1.5)
with(juul2, Hist(igf1, scale="percent", freq = TRUE,
                  breaks=c(0,100,150,200,250,300,350,400,450,500, 600, 1000),
                  col="darkgray", ylab="Frequênci Relativa(%)))
text(-150, -40, labels="c)", pos = 1, xpd = T, cex = 1.5)
with(juul2, Hist(igf1, scale="density",
                  breaks=c(0,100,150,200,250,300,350,400,450,500, 600, 1000),
                  col="darkgray", ylab="Densidade de Frequênci Relativa"))
text(-150, -0.0010, labels="d)", pos = 1, xpd = T, cex = 1.5)

```

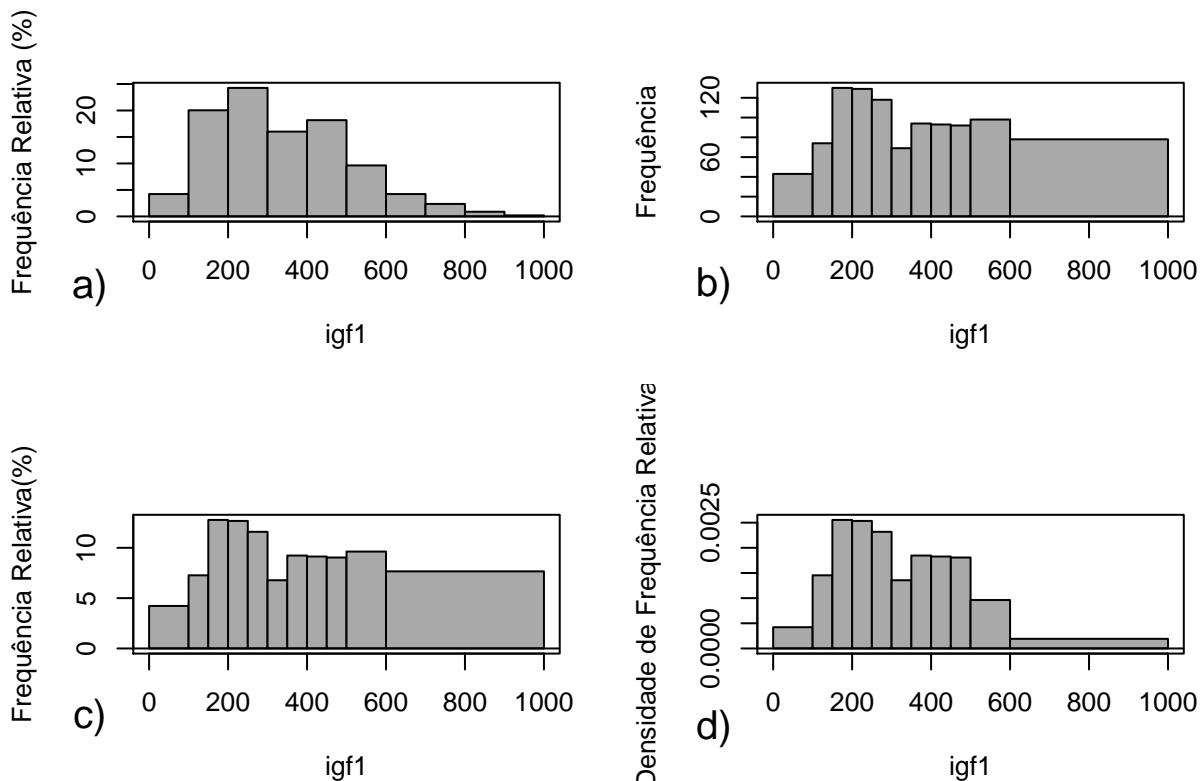


Figura 8.42: a) Histograma de frequênci relativa da variável *igf1* para 10 classes com igual amplitude (igual ao da figura 8.41); b) histograma de frequênci da variável *igf1* para as classes definidas conforme a tabela 8.2; c) histograma de frequênci relativa da variável *igf1* para as classes definidas conforme a tabela 8.2; d) histograma de densidade de frequênci relativa da variável *igf1* para as classes definidas conforme a tabela 8.2.

A função `par(mfrow = c(2,2))` indica que os gráficos serão exibidos em duas colunas sendo cada linha preenchida antes de avançar para a próxima.

O histograma da figura 8.42a é idêntico ao da figura 8.41 (10 classes de mesma amplitude). Repetimos a seguir o comando usado para gerá-lo:

```
with(juul2, Hist(igf1, scale="percent", breaks="Sturges", col="darkgray",
                  ylab="Frequência Relativa (%)))
```

O histograma da figura 8.42b é o histograma de frequência de `igf1` para as classes definidas na tabela 8.2 gerado pelo comando:

```
with(juul2, Hist(igf1, scale="frequency", freq = TRUE,
                  breaks=c(0,100,150,200,250,300,350,400,450,500, 600, 1000),
                  col="darkgray", ylab="Frequência"))
```

O histograma da figura 8.42c é o histograma de frequência relativa de `igf1` para as classes definidas na tabela 8.2 gerado pelo comando:

```
with(juul2, Hist(igf1, scale="percent", freq = TRUE,
                  breaks=c(0,100,150,200,250,300,350,400,450,500, 600, 1000),
                  col="darkgray", ylab="Frequência Relativa(%)))
```

No histograma de frequência, a altura de cada classe é igual ao número de valores nela contidos. No histograma de frequência relativa, a altura é a proporção de valores contidos na classe (contagem de valores / número total de valores), eventualmente multiplicada por 100 para ser expressa em porcentagem. Assim tanto o histograma de frequência quanto o de frequência relativa possuem a mesma forma, diferindo apenas na escala do eixo Y.

Quando a amplitude das classes são diferentes, porém, o histograma de frequência (e o de frequência relativa) dão uma visão distorcida da distribuição dos valores da variável. Observem que uma mensagem aparece na área de mensagens para esses dois histogramas com o seguinte teor:

AVISO: Warning in plot.histogram(r, freq = freq1, col = col, border = border, angle = angle, the AREAS in the plot are wrong – rather use 'freq = FALSE'

Essa mensagem indica que a distorção é causada porque cada classe de um histograma deve ter a altura tal que a área de cada classe (altura x amplitude) deve ser proporcional à frequência (ou frequência relativa) de cada uma delas. Não é o que acontece nos histogramas das figuras 8.42b e 8.42c. Por exemplo, as áreas das classes 10 e 11 na figura 8.42b deveriam ser proporcionais a 78 (frequência da classe 11) e 98 (frequência da classe 1) respectivamente, ou seja, as áreas deveriam ser iguais, respectivamente, a 78 e 98, multiplicadas por uma constante qualquer. No entanto a área da classe 11 é igual a 400 x 78, e a área da classe 10 é 100 x 98, indicando que o número que multiplica a altura da classe 11 é 4 vezes maior do que o número que multiplica a altura da classe 10, distorcendo a distribuição dos valores da variável. Fato semelhante ocorre no histograma de frequência relativa (figura 8.42c). Para contornar esse problema, quando as classes possuem amplitude diferentes, utiliza-se o **conceito de densidade de frequência relativa**. Nesse caso, para

cada classe, divide-se a sua frequência relativa pela sua amplitude, obtendo-se os valores na última coluna da tabela 8.2.

Com o comando a seguir, obtemos o histograma de densidade de frequência relativa para a variável *igf1*, com as classes definidas na tabela 8.2 (figura 8.42d):

```
with(juul2, Hist(igf1, scale="density",
                  breaks=c(0,100,150,200,250,300,350,400,450,500, 600, 1000),
                  col="darkgray", ylab="Densidade de Frequênci a Relativa(%)))
```

Observem agora que a altura da classe 11 é relativamente bem menor do que a das demais classes, refletindo o fato de que os 78 valores dessa classe estão distribuídos em uma faixa maior de valores do que as demais classes.

A execução da função *par(mfrow = c(1,1))* a seguir indica que os gráficos voltarão a ser exibidos na forma normal.

```
par(mfrow = c(1,1)) # volta a exibir os gráficos da forma normal
```

8.5.2 Histograma por grupos

Da mesma forma que *boxplots*, é possível gerar histogramas de uma variável numérica para cada categoria de uma variável categórica. Por exemplo, para criarmos um histograma de *igf1* para cada categoria de Tanner, clicamos no botão *Gráfico por grupos* na figura 8.39 e selecionamos a variável *tanner_cat* na caixa de diálogo *Grupos* (figura 8.43). Os histogramas de *igf1* para cada categoria de Tanner são mostrados na figura 8.44.



Figura 8.43: Selecionando uma variável de agrupamento para a construção de histogramas.

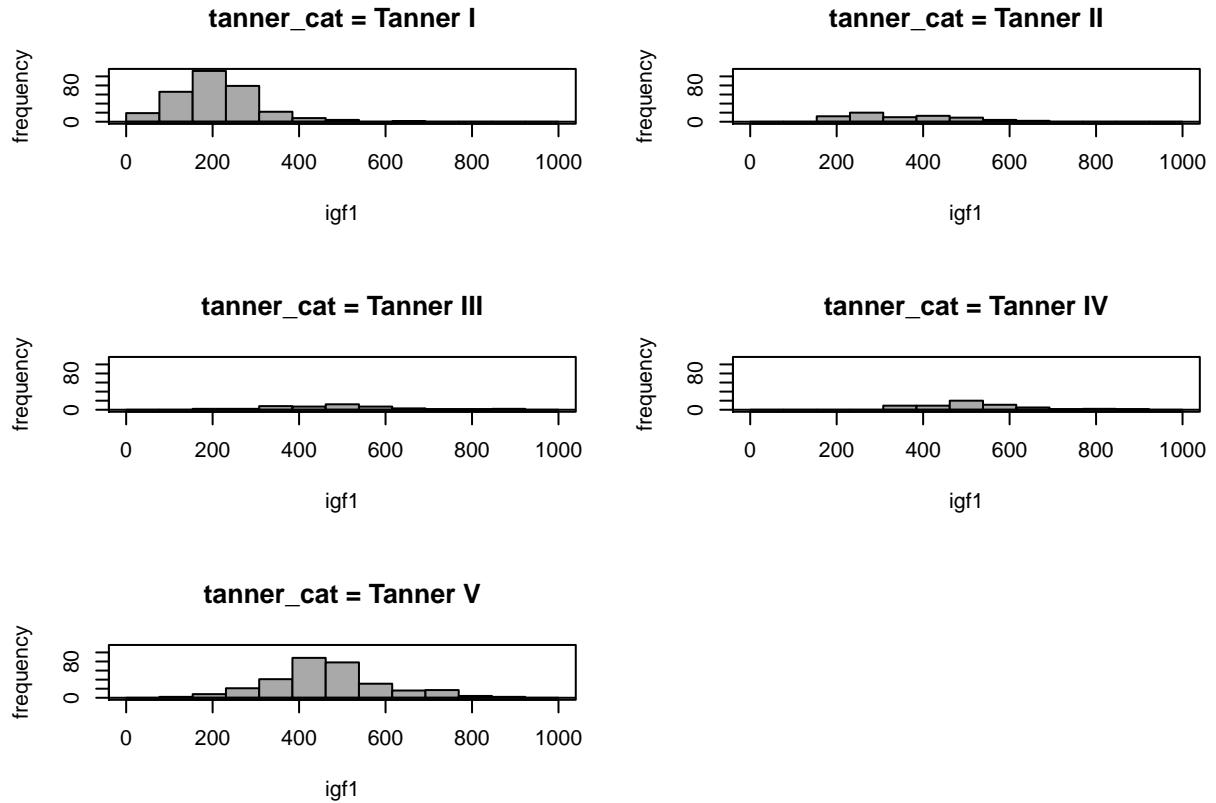


Figura 8.44: Histogramas de $igf1$ para cada categoria de Tanner.

8.6 Diagrama de pontos e *strip chart*

O conteúdo desta seção pode ser visualizado neste [vídeo](#).

O diagrama de pontos também fornece um visão da distribuição dos valores de uma variável numérica. Ele pode ser apresentado de formas diferentes. Para criar um diagrama de pontos, acessamos como sempre o menu *Gráficos* e selecionamos a opção:

Gráficos \Rightarrow Diagrama de Pontos

Na caixa de diálogo do gráfico de pontos (figura 8.45), podemos selecionar a variável desejada e uma variável para criar um gráfico de pontos para cada categoria de outra variável (Gráfico por grupos). Nesse exemplo, selecionamos $igf1$ como a variável desejada e $tanner_cat$ como a variável de agrupamento. O diagrama resultante é mostrado na figura 8.46.



Figura 8.45: Caixa de diálogo para a criação de um diagrama de pontos. Na aba *Dados*, selecionamos a variável numérica desejada. Ao clicarmos no botão *Gráfico por grupos...*, podemos selecionar uma variável de agrupamento.

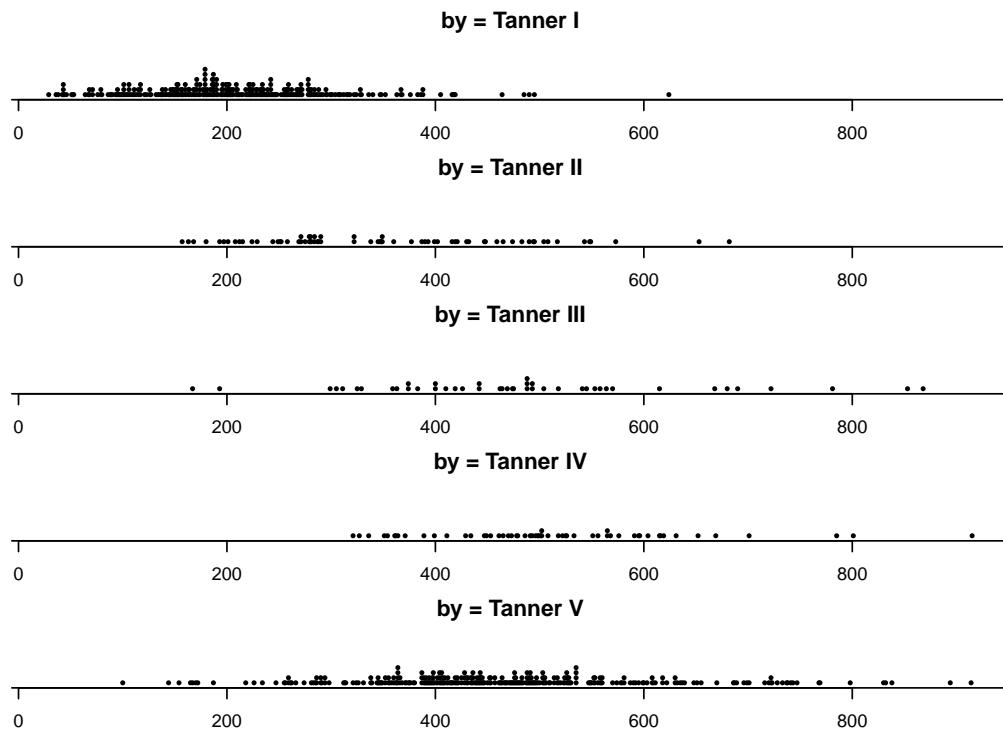


Figura 8.46: Diagrama de pontos de *igf1* para cada categoria de Tanner.

Uma forma alternativa de visualizar as distribuições de pontos é plotar os pontos ao longo de uma linha vertical para cada grupo. Isso pode ser feito por meio do diagrama de *strip chart*, que pode ser configurado com a opção:

Gráficos ⇒ Gráfico Strip Chart

Na caixa de diálogo da figura 8.47, selecionamos a variável numérica (variável resposta) e a(s) variável(is) de agrupamento (fatores), se desejado.



Figura 8.47: Caixa de diálogo para a criação de um diagrama de *strip chart*. Na aba *Dados*, selecionamos a variável numérica e a variável de agrupamento, se desejado.

O gráfico é mostrado na figura 8.48.

Diagrama de Strip Chart de igf1 por categorias de Tanner

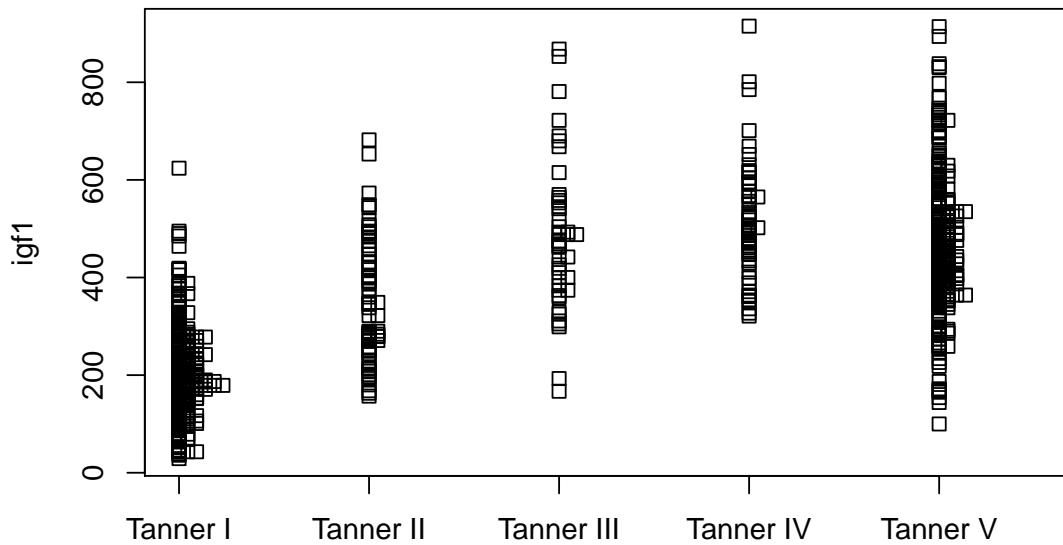


Figura 8.48: Diagrama de *strip chart* de *igf1* para cada categoria de Tanner.

Essencialmente os diagramas 8.46 e 8.48 apresentam a mesma mensagem do que os *boxplots* da figura 8.38.

8.7 Diagrama de dispersão ou espalhamento

O conteúdo desta seção pode ser visualizado neste [vídeo](#).

O diagrama de dispersão ou espalhamento é bastante utilizado para mostrar o relacionamento entre duas variáveis numéricas. Ele simplesmente plota os pontos no plano de coordenadas XY, sendo uma variável escolhida para o eixo X e outra para o eixo Y. Vamos criar o diagrama de dispersão das variáveis *igf1* x *idade*. No menu *Gráficos* do *R Commander*, selecionamos a opção:

Gráficos ⇒ Diagrama de Dispersão

Na caixa de diálogo *Gráfico de Dispersão*, selecionamos as variáveis dos eixo X e Y na aba *Dados* (figura 8.49). Também podemos selecionar uma variável de agrupamento, para criar um diagrama de dispersão para cada categoria de uma outra variável. Nesse exemplo, não vamos fazer gráficos por grupos. Na aba *Opções* (figura 8.50), há uma série de opções que podem ser selecionadas. Marquemos a opção *Linha de quadrados mínimos*. Ao clicarmos em OK, o gráfico resultante é mostrado na figura 8.51.

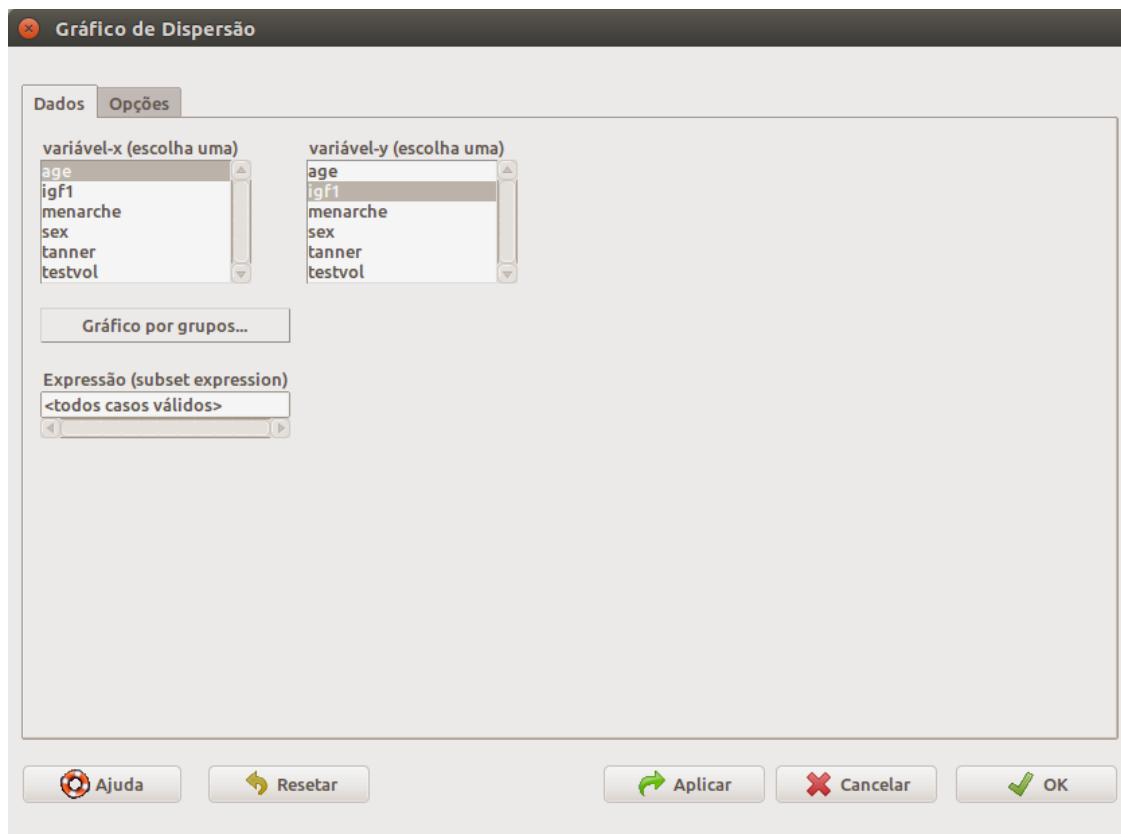


Figura 8.49: Caixa de diálogo para a criação de um diagrama de dispersão. Na aba *Dados*, selecionamos as variáveis numéricas dos eixos X e Y.

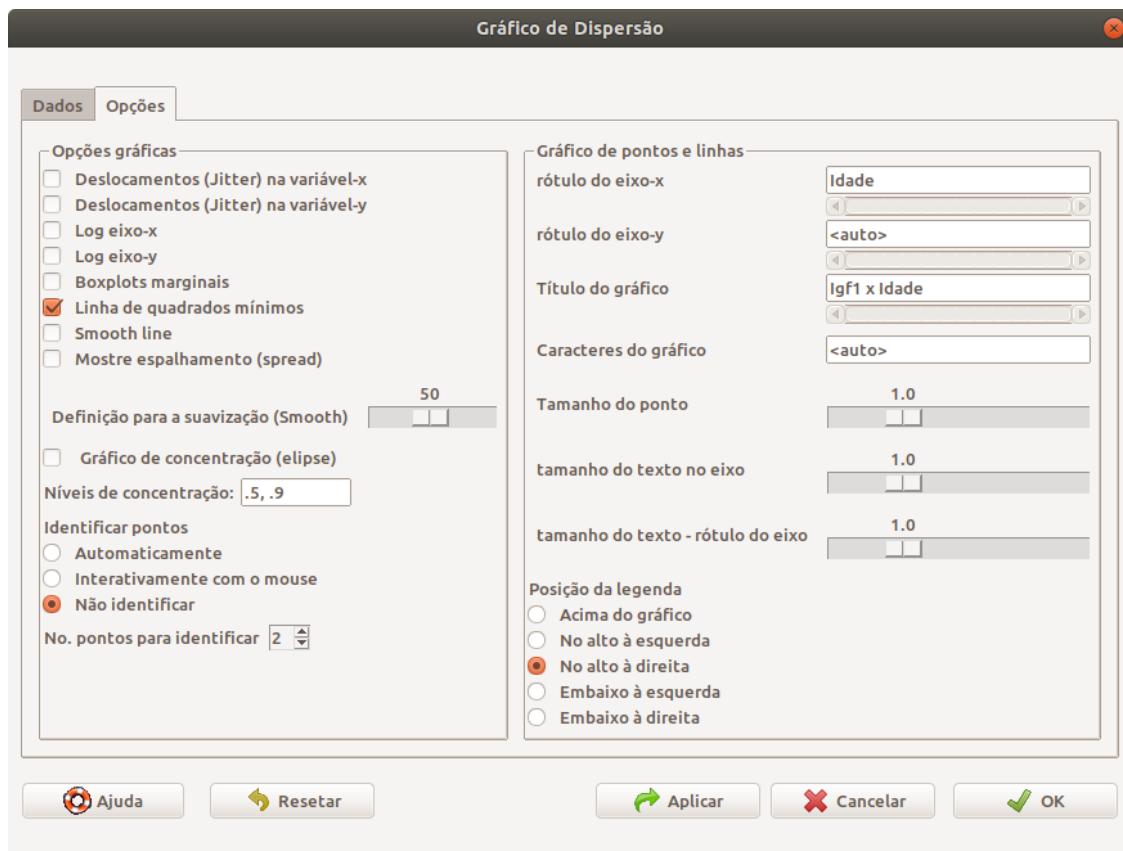


Figura 8.50: Aba *Opções* da caixa de diálogo do gráfico de dispersão.

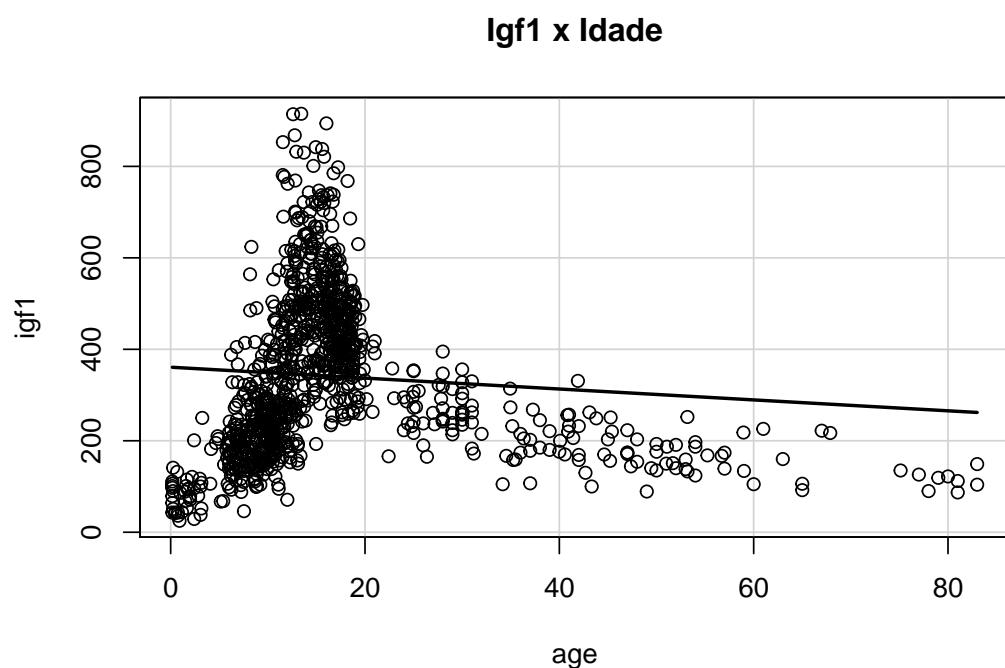


Figura 8.51: Diagrama de dispersão das variáveis *igf1* x *age*.

Esse diagrama de dispersão sugere que o hormônio *igf1* tende a aumentar com a idade até o final da adolescência e, após esse período, ele tende a decrescer com a idade. Entretanto a mesma observação ao final da seção do diagrama de *boxplot* se aplica aqui.

A reta de regressão é construída de modo a ajustar uma reta aos dados em um diagrama de espalhamento. Como visivelmente a idade não está linearmente associada a *igf1*, a reta de regressão nesse exemplo não reflete como a idade está associada à variável *igf1*.

8.7.1 Alterando a espessura e cor da linha de regressão e o tipo dos pontos

A espessura e a cor da linha de regressão podem ser alteradas por meio do argumento *regLine*. Se quisermos fazer a espessura ser o dobro da original e a cor azul, por exemplo, temos que fazer *regLine=list(lwd = 4, col = "blue")*. O argumento *lwd* (*line width*) especifica a espessura e o padrão é igual a 2.

O tipo dos pontos pode ser alterado por meio do argumento *pch*. Fazendo *pch = 19*, por exemplo, irá plotar pontos cheios. Esta [página](#) mostra uma tabela dos símbolos com os respectivos números. Vide comando a seguir e figura 8.52:

```
scatterplot(igf1~age, regLine=list(lwd = 2, col = "blue"), smooth=FALSE,
           boxplots=FALSE, main="Igf1 x Idade", data=juul2,
           col = "black", pch = 19)
```

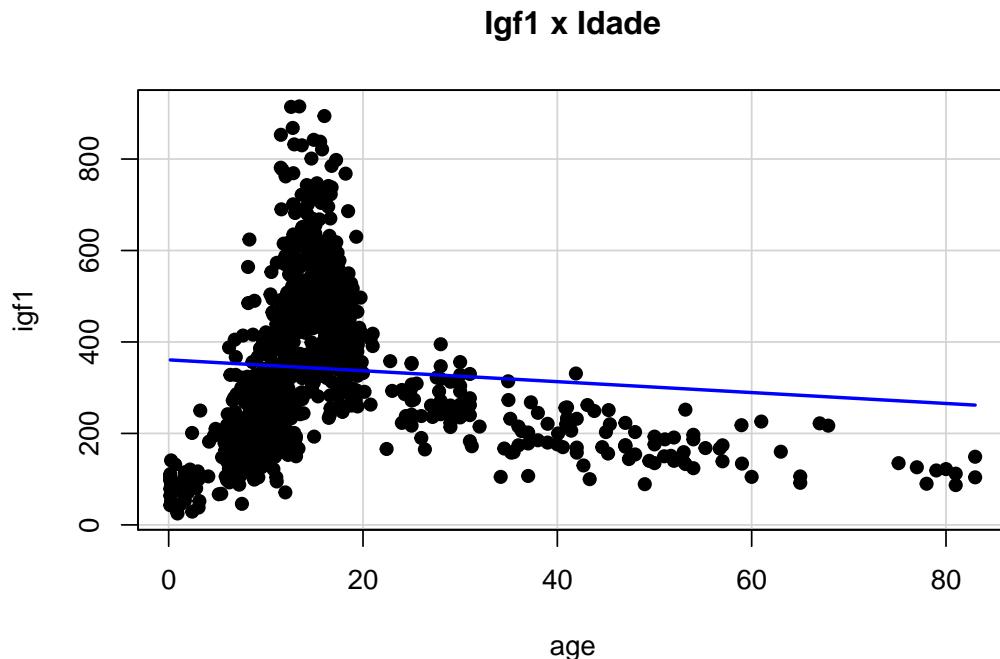


Figura 8.52: Diagrama de dispersão das variáveis *igf1* x *age*, com pontos cheios e reta azul.

Ao selecionarmos as variáveis numéricas que serão plotadas no diagrama de dispersão, podemos distinguir os pontos e as retas de regressão por categorias de uma variável categórica. Ao

clicarmos no botão *Gráfico por grupos...* (figura 8.49), podemos selecionar uma variável de agrupamento. Selecionando a variável *tanner_cat* e configurando as opções do gráfico como mostra a figura 8.53, o resultado é mostrado na figura 8.54.



Figura 8.53: Aba *Opções* da caixa de diálogo do gráfico de dispersão.

igf1 x idade por categorias de Tanner

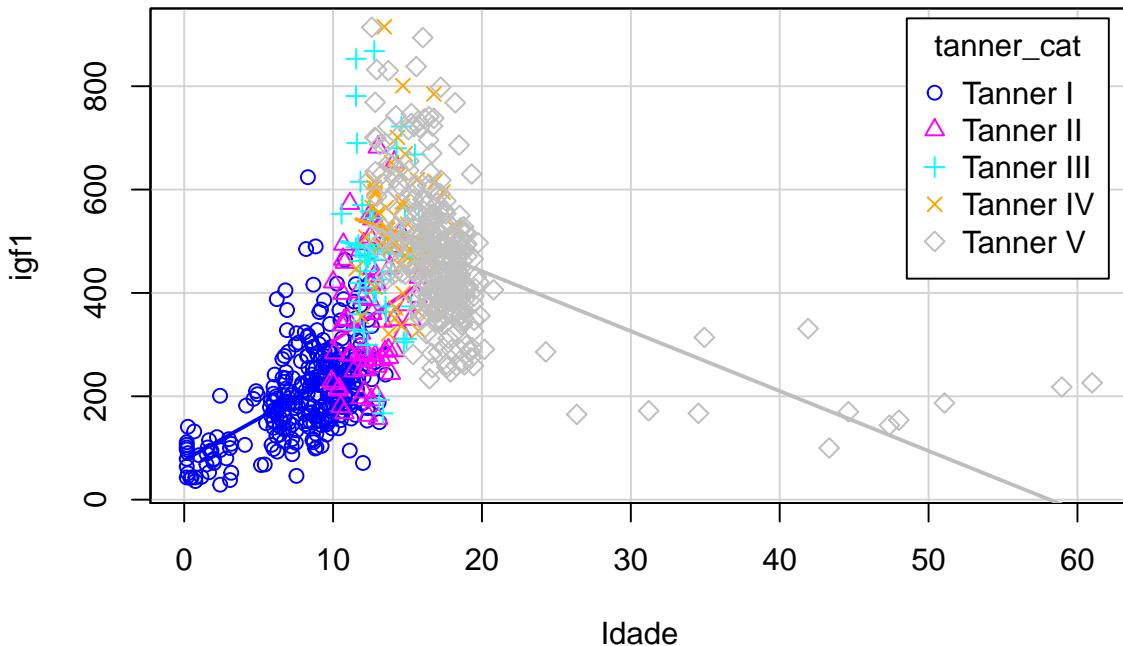


Figura 8.54: Diagrama de dispersão das variáveis *igf1* x *age* por categorias de Tanner.

É possível observar que, dentro de cada categoria de Tanner, com exceção da categoria I, não é tão evidente qual é a relação entre *igf1* e a idade. Na categoria I, aparentemente, *igf1* tende a aumentar com a idade.

8.8 Salvando gráficos em um arquivo

No *RStudio*, os gráficos são exibidos na aba *Plots* (figura 8.55). Nessa aba, o usuário pode navegar entre os diversos gráficos que foram gerados na sessão corrente (seta vermelha na figura), bem como salvar o gráfico como imagem, pdf ou copiar para a área de transferência (seta verde). Ao selecionarmos a opção *Salvar como imagem*, surge uma tela que permite ao usuário configurar uma série de opções sobre como o gráfico será gravado.

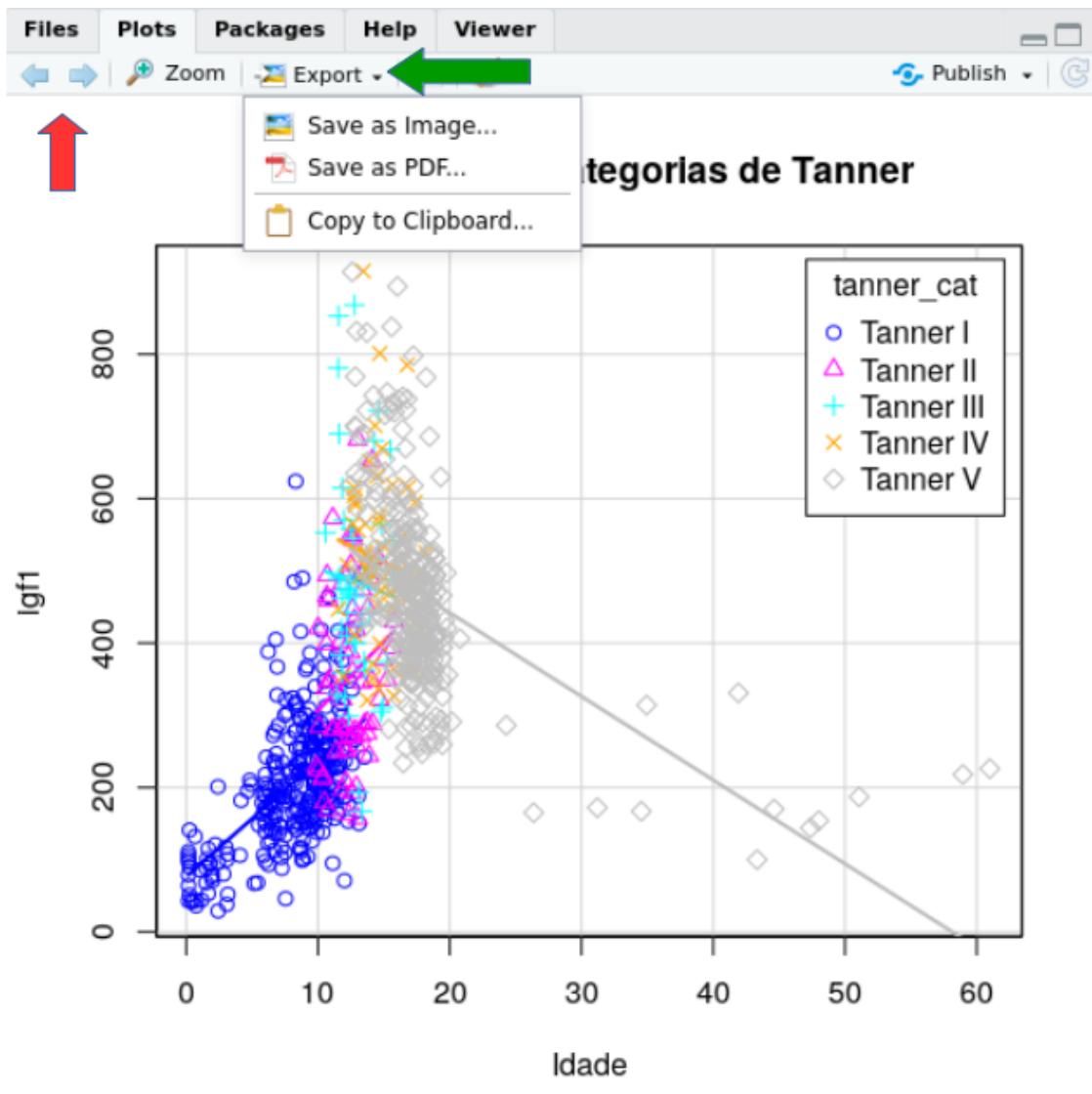


Figura 8.55: Aba *Plots* do *RStudio* com opções para navegar pelos gráficos, ampliar e exportar os gráficos para diferentes formatos.

A caixa de diálogo mostrada na figura 8.56 permite ao usuário selecionar o formato da imagem a ser gravada (png, tiff, jpeg, bmp, svg, eps), a largura e a altura da figura, o nome do arquivo e o diretório (pasta) onde o arquivo será gravado. Ao clicarmos em *Save*, o arquivo será gravado no local especificado.

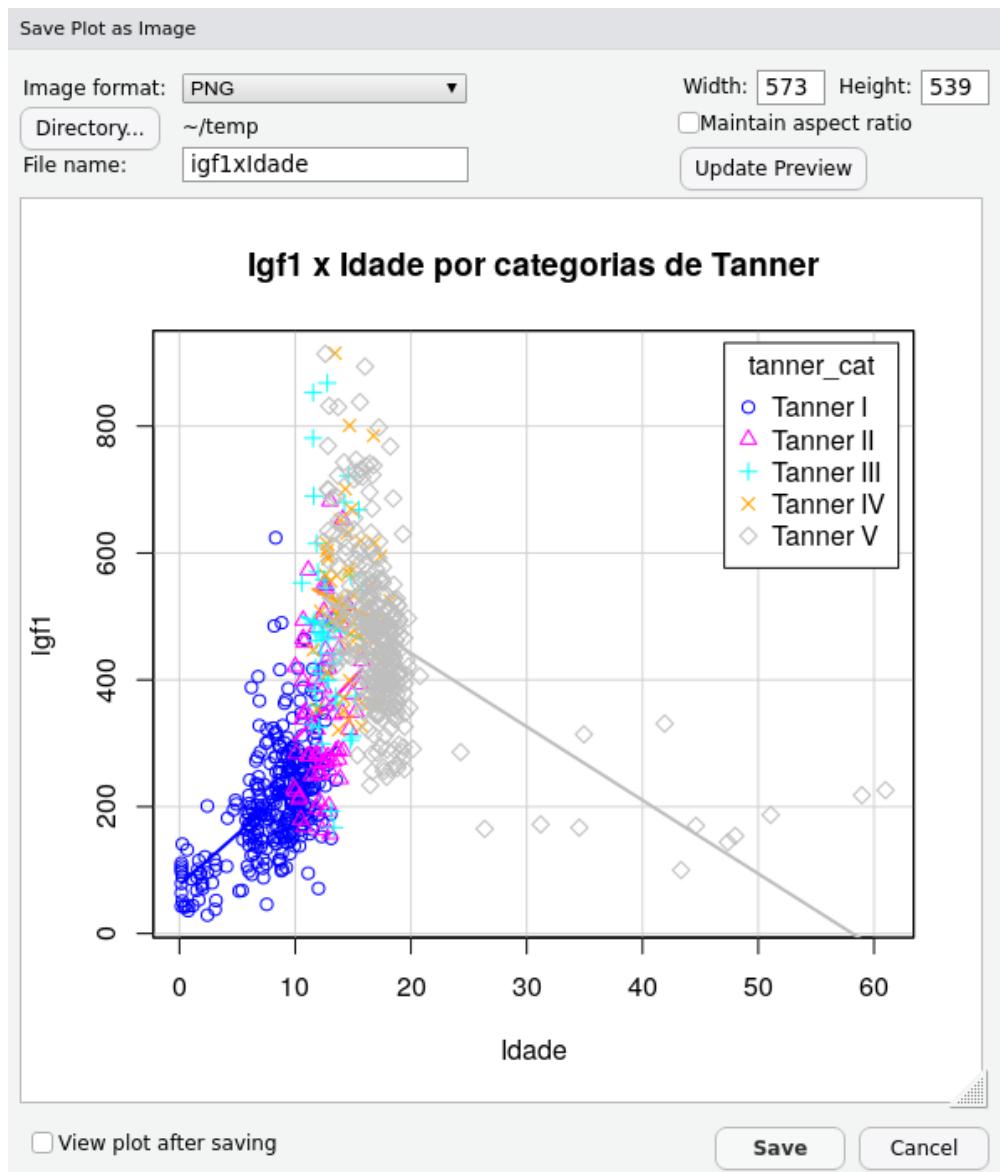


Figura 8.56: Tela para configurar as opções de exportação de um gráfico no *RStudio*.

No *R Commander*, para gravarmos o gráfico em um arquivo, utilizamos a opção:

Gráficos ⇒ Salvar Gráfico em arquivo ⇒ Como Bitmap

A caixa de diálogo mostrada na figura 8.57 permite ao usuário selecionar o formato da imagem a ser gravada (png ou jpeg), a largura e a altura da figura, o tamanho do texto e a resolução. Ao clicarmos em OK, será mostrada uma tela para o usuário especificar o nome do arquivo e o local no disco onde o mesmo será gravado.



Figura 8.57: Tela para configurar as opções de exportação de um gráfico no *R Commander*.

O comando gerado pelo *R Commander* é mostrado a seguir:

```
dev.print(png, filename="/home/sergio/temp/grafico.png", width=6, height=6,
         pointsize=12, units="in", res=300)
```

Para salvarmos o gráfico com uma resolução maior do que 300 dpi, basta alterarmos o valor do argumento *res*.

8.9 Recursos gráficos de outros plugins

O *R Commander* oferece uma maneira fácil de construir alguns tipos de gráficos. Porém cada tipo de gráfico possui diversos outros recursos que podem ser utilizados, mas que não são apresentados na interface gráfica. Nesse caso, há outras possibilidades: a) utilizar outros *plugins* que possuem mais recursos na sua interface gráfica; b) digitar o comando que cria o gráfico na janela de script, com as alterações para criar o efeito gráfico desejado; c) utilizar outros pacotes com mais recursos, como o [ggplot2 \(MIT License\)](#). Nesta seção, vamos mostrar como carregar outros *plugins* no *R Commander*.

Na sua própria interface gráfica, o *R Commander* pode acrescentar outros *plugins*, com recursos variáveis, inclusive gráficos. Vamos aqui mostrar um desses *plugins*, o *RcmdrPlugin.KMggplot2*. O processo de carregamento desse *plugin* é semelhante para os demais.

Para carregarmos um novo *plugin*, é preciso que ele esteja instalado. Para instalarmos um

plugin, o procedimento é o mesmo para instalar qualquer pacote do R. Uma vez instalado o *plugin*, acessamos a opção:

Ferramentas ⇒ Carregar plugins do Rcmdr

Na caixa de diálogo *Carregar Plug-ins*, selecionamos o *plugin* desejado, ou os *plugins*, e clicamos em OK (figura 8.58). Caso um *plugin* desejado não apareça nessa lista, ele precisará ser instalado.

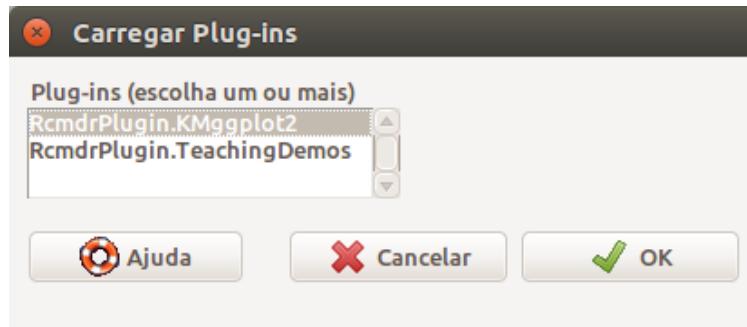


Figura 8.58: Diálogo para selecionar e carregar um novo *plugin* no *R Commander*.

Para carregarmos qualquer *plugin*, será necessário reiniciar o *R Commander* (figura 8.59).

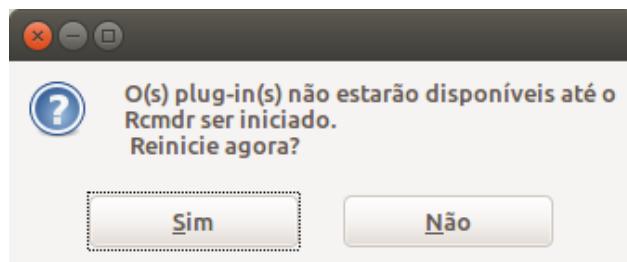


Figura 8.59: É necessário reiniciar o *R Commander* para carregar um novo *plugin*.

Ao reiniciarmos o *R Commander*, se algum conjunto de dados estava sendo usado, ele não está mais ativo. É preciso ativá-lo. Para escolhermos o conjunto de dados que estará ativo no *R Commander*, selecionamos a opção:

Dados ⇒ Conjunto de dados ativo ⇒ Selecionar conjunto de dados ativo...

Neste capítulo, estamos trabalhando com o conjunto de dados *juul2*. Ele aparece na lista de conjuntos de dados disponíveis (figura 8.60). Selecione o conjunto *juul2* e clique no botão OK. *juul2* será ativado e poderemos acessar o menu do *Kmggplot2* (figura 8.61).

Observem as opções de gráficos desse plugin, experimentem e vejam as diferenças em relação aos recursos padrões do *R Commander*.

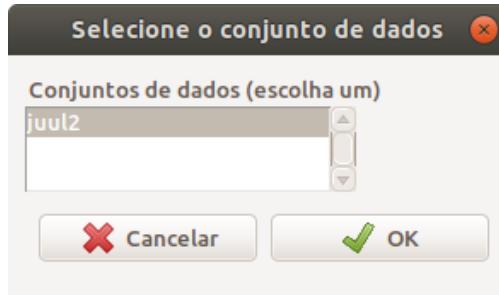


Figura 8.60: Seleção do conjunto de dados a ser ativado.

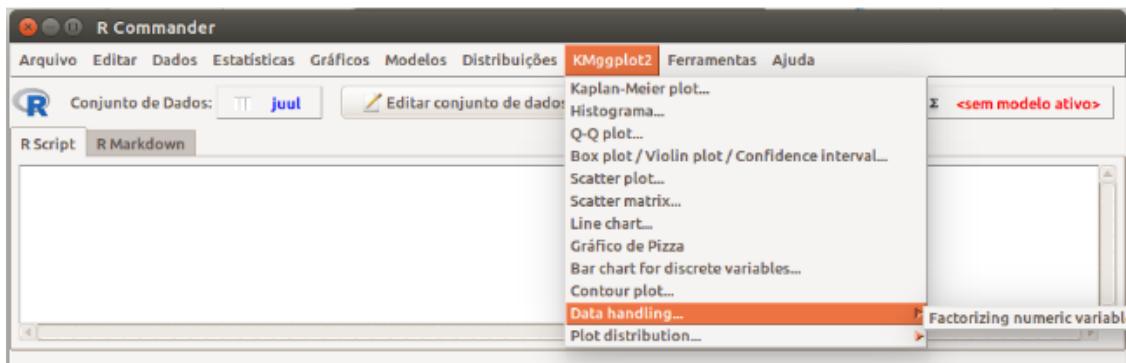


Figura 8.61: Plugin *KMggplot2* com diversos recursos para a criação de gráficos.

8.10 Exercícios

- 1) O diagrama de barras da figura 8.62 mostra a porcentagem de mulheres cujos bebês foram internados em uma UTI neonatal em diversas faixas de escolaridade e estratificada de acordo com o critério se tiveram ou não doença hipertensiva específica da gravidez (DHEG). Responda às questões abaixo.
- Qual a faixa de escolaridade mais frequente?
 - Qual a porcentagem aproximada da faixa de escolaridade de 4-7 anos?
 - Qual a porcentagem aproximada de mulheres na faixa de escolaridade >12 anos e que tiveram DHEG?
 - A partir do diagrama, que relação é sugerida entre a escolaridade e a DHEG? Comente sobre essa relação.

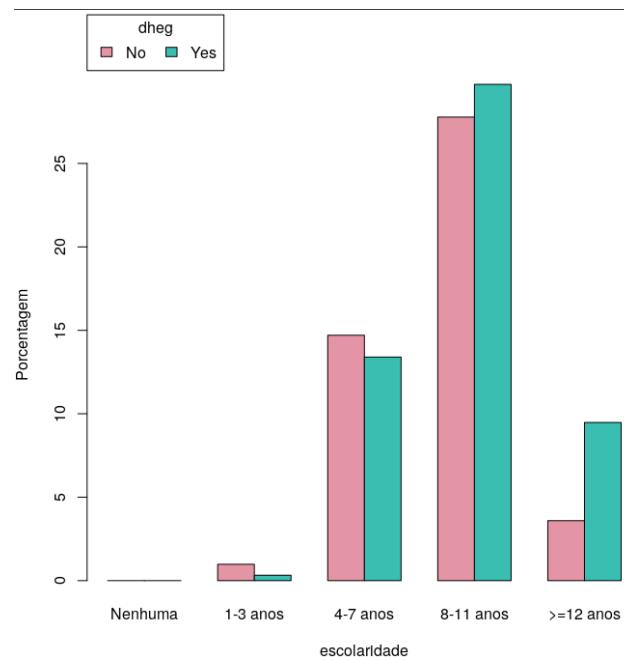


Figura 8.62: Diagrama de barras Escolaridade X DHEG.

- 2) Os dois gráficos da figura 8.63 representam um diagrama de barras para as variáveis peso ao nascimento de recém-nascidos internados em uma UTI neonatal e do número de visitas no pré-natal das respectivas mães.
- Faça uma crítica sobre a adequabilidade dos dois gráficos em mostrar a distribuição dos dados das respectivas variáveis.
 - Que conclusões você pode tirar sobre o uso do diagrama de barras para ilustrar a distribuição dos dados para variáveis numéricas?

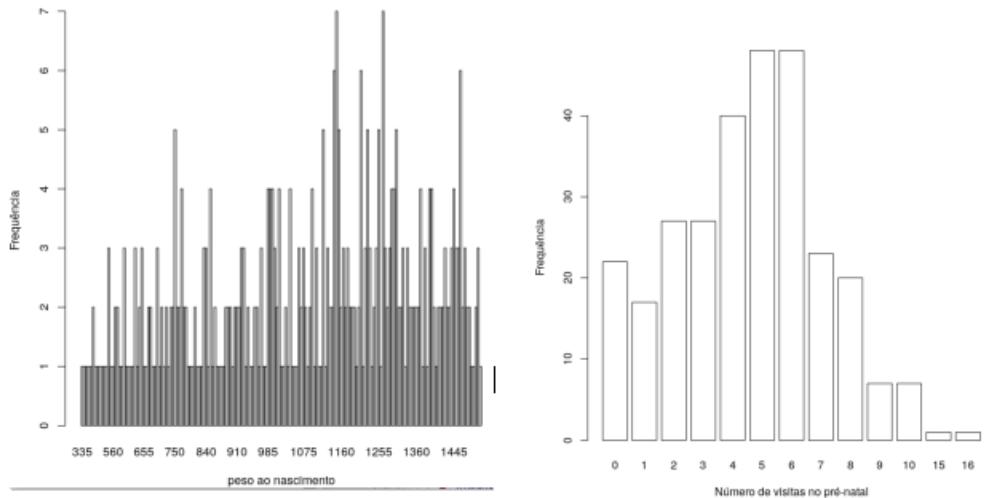


Figura 8.63: Diagrama de barras para variáveis numéricicas: a) peso ao nascimento , b) número de visitas no pré-natal.

- 3) Os três gráficos da figura 8.64 são histogramas da idade gestacional de recém-nascidos que foram internados em uma UTI neonatal. Todos os histogramas foram criados com classes de mesma amplitude.
- O que representa cada barra no histograma superior à esquerda?
 - Qual a diferença entre os histogramas?

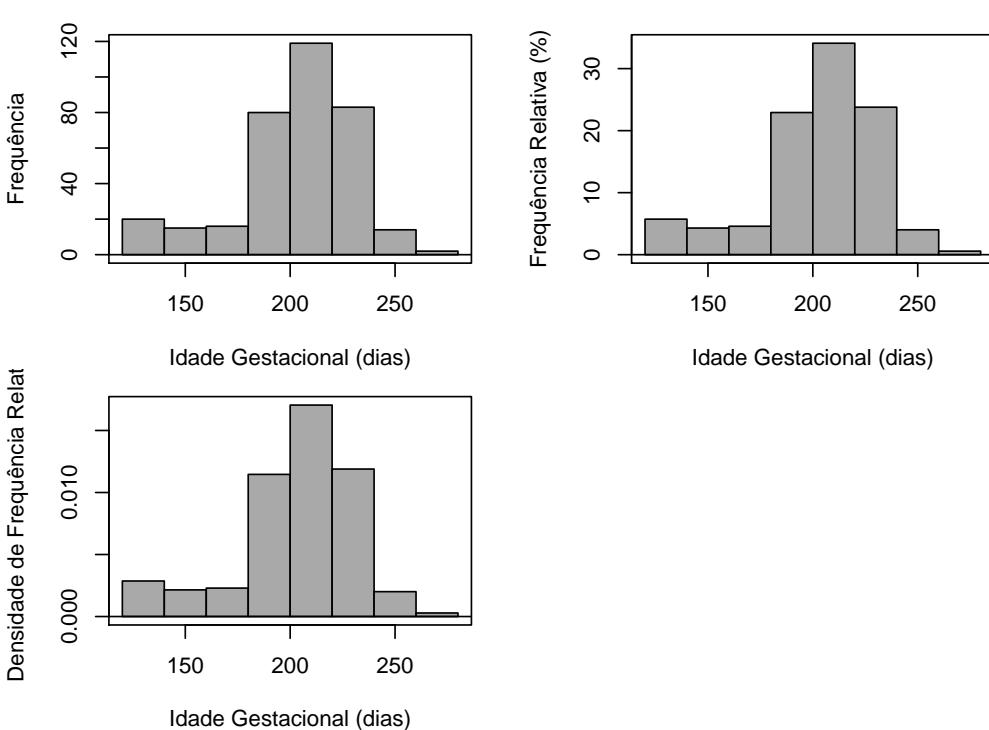


Figura 8.64: Histogramas da idade gestacional de recém-nascidos internados em uma UTI neonatal.

4) Os quatro gráficos da figura 8.65 são histogramas da idade gestacional de recém-nascidos que foram internados em uma UTI neonatal.

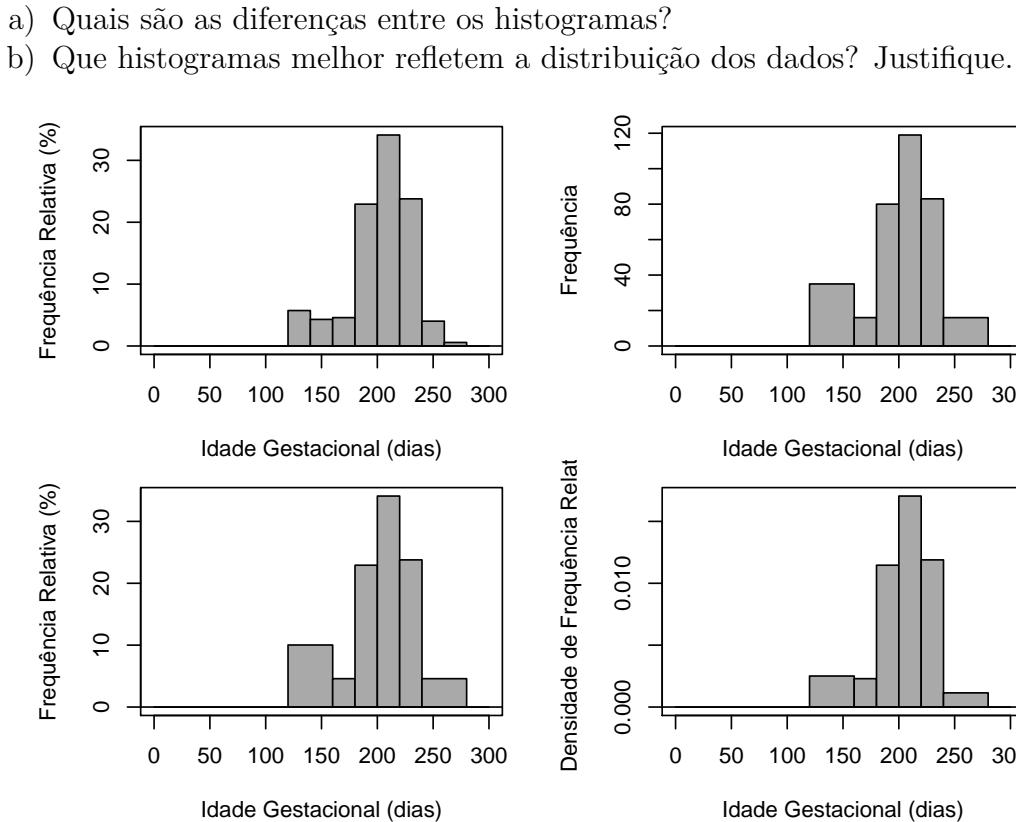


Figura 8.65: Histogramas da idade gestacional de recém-nascidos internados em uma UTI neonatal.

5) O gráfico da figura 8.66 mostra os resultados de 5 experimentos para medir a velocidade da luz. Responda às questões abaixo.

- O que significa a linha mais espessa no interior de cada caixa?
- O que significam as linhas superior e inferior que delimitam as caixas?
- O que significam as linhas superior e inferior fora das caixas?
- Por que os experimentos 1 e 3 contêm alguns pontos representados por círculos e os demais não?
- Por que as linhas tracejadas no experimento 2 não possuem o mesmo comprimento?
- Qual experimento mostrou maior variabilidade? E a menor? Que critério você utilizou para avaliar a variabilidade dos dados?
- Que experimento teve a menor mediana? E a maior?
- Indique os experimentos em que a média e o desvio padrão poderiam ser utilizadas como boas medidas de tendência central e dispersão.

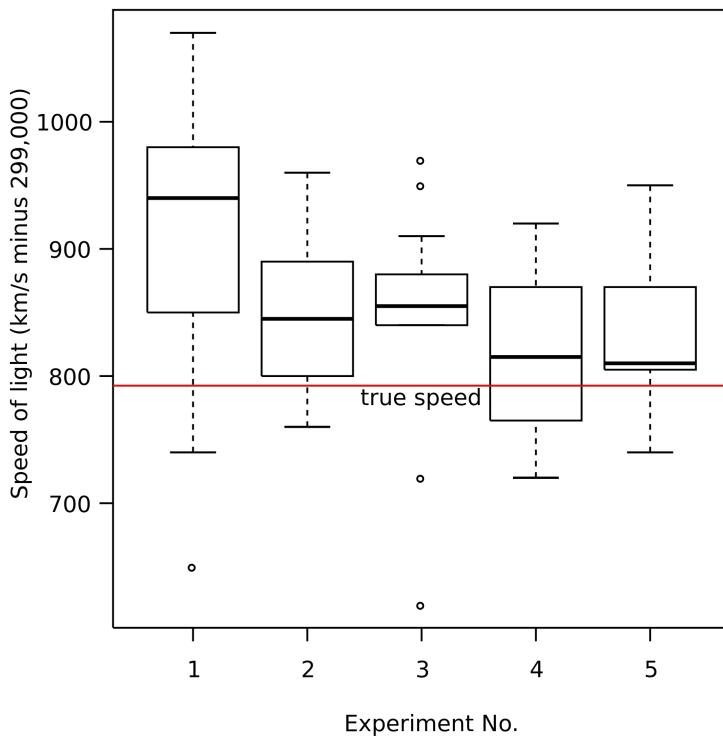


Figura 8.66: Boxplots de experimentos para medir a velocidade da luz. Fonte: [wikipedia \(Domínio Público\)](#).

- 6) Carregue o conjunto de dados *Pima.te* do pacote [*MASS* \(GPL-2 | GPL-3\)](#).
 - a) Veja a ajuda do conjunto de dados.
 - b) Use a função *cut* (seção 6.1.5) para gerar uma variável, *glu_cat*, com as seguintes faixas de glicose: (0-99], (99-120] e >120 mg/dl.
 - c) Faça um diagrama de barras condicional, lado a lado com as porcentagens da variável Status para cada categoria da variável *glu_cat*. Comente o gráfico.
 - d) Faça um *boxplot* da variável *glu* para cada categoria da variável *type*. Comente o diagrama.
 - e) Faça um histograma da variável *glu* para cada categoria da variável *type*. Comente o diagrama.
 - f) Faça um diagrama de *stripchart* da variável *bmi* por categoria da variável *type*. Comente o gráfico.
 - g) Faça um diagrama de dispersão de *glu* por *bmi*. Comente o gráfico.

Capítulo 9

Datas e tempo

Este capítulo pode ser visualizado neste [vídeo](#).

Datas no R são representadas pela classe *Date*. Tempos são representados pelas classes *POSIXct* ou *POSIXlt*.

Internamente, as datas são armazenadas como o número de dias desde 01/01/1970. Horas são armazenadas como o número de segundos desde 01/01/1970.

9.1 Datas no R

Datas podem ser criadas por coerção de uma string de caracteres usando a função *as.Date()*. Se solicitarmos a ajuda para essa função (*?as.Date*), teremos mais detalhes sobre como essa função converte objetos para datas, seus argumentos, etc. Vamos ver aqui os usos mais simples dessa função.

Se usarmos essa função com uma string no formato “ano-mês-dia” ou “ano/mês/dia”, o resultado será um objeto do tipo *Date* que armazena a data especificada pela string. A data será exibida no formato *ano-mês-dia*.

```
x = as.Date("2019-03-01")
x
## [1] "2019-03-01"
```

O argumento *format* permite ao usuário especificar como a data será construída. Por exemplo, no comando abaixo, o argumento *format* = “%m-%d-%Y” faz com que a data seja criada a partir do formato mês-dia-ano.

```
as.Date("10-26-2019", format = "%m-%d-%Y")
## [1] "2019-10-26"
```

A representação interna de um objeto da classe *Date* pode ser vista por meio da função *unclass*:

```
unclass(x)
```

```
## [1] 17956
```

Vamos carregar o conjunto de dados *stroke* do pacote *ISwR* (GPL-2 | GPL-3), utilizado no capítulo 7.

```
library(ISwR)
data(stroke, package="ISwR")
```

Recordando, o conjunto de dados *stroke* contém 829 casos de AVC em Tartu, Estônia, durante o período 1991-1993, com seguimento até 1 de janeiro de 1996. Há duas variáveis da classe *Date* em *stroke*, *died* e *dstr*.

```
class(stroke$died)
```

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

```
class(stroke$dstr)
```

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

A variável *dead* de *stroke* é um vetor lógico que indica se o paciente morreu ou não durante o estudo.

É possível realizar as operações de adição e subtração com datas e tempos, e utilizar expressões lógicas (==, <=, >=, !=, >, <).

Por exemplo, se quiséssemos saber o número de dias desde o AVC até o óbito (caso o paciente tenha falecido) ou até o último dia de seguimento (“01/01/1996”), poderíamos utilizar o código a seguir:

```
# cria uma cópia da variável dead de stroke, que
# indica se cada paciente morreu ou não
morto = stroke$dead
# dias do AVC até o óbito para os paciente que morreram
stroke$diasObs[morto] = with(stroke[morto,], died - dstr)
# dias do AVC até "01-01-1996" para os paciente que não morreram
stroke$diasObs[!morto] = with(stroke[!morto,], as.Date("1996-01-01") - dstr)
```

O primeiro comando copia a variável *dead* de *stroke* para o objeto *morto*, que indica se cada paciente morreu ou não. O segundo comando irá contar o número de dias do AVC até o óbito para os pacientes que morreram. O cálculo somente será realizado para os pacientes que morreram, por isso o objeto *morto* entre colchetes após a variável *diasObs* e na seleção das observações de *stroke* dentro do comando *with*.

O terceiro comando irá contar o número de dias do AVC até o óbito para os paciente que não morreram. O cálculo somente será realizado para os pacientes que não morreram, por isso o objeto *morto* precedido do sinal de exclamação entre colchetes após a variável *diasObs* e na seleção das observações de *stroke* dentro do comando *with*.

A criação do objeto *morto* foi feita apenas para facilitar a compreensão dos comandos seguintes. Caso isso não fosse feito, os demais comandos deveriam ser escritos da seguinte forma:

```
stroke$diasObs[stroke$dead] = with(stroke[stroke$dead,], died - dstr)
stroke$diasObs[!stroke$dead] = with(stroke[!stroke$dead,], died - dstr)
```

A listagem a seguir mostra os 5 primeiros registros do conjunto de dados *stroke* com a variável *diasObs* acrescentada ao conjunto de dados:

```
head(stroke[, c("sex", "age", "dstr", "died", "dead", "diasObs")], 5)
```

```
##      sex age      dstr     died   dead diasObs
## 1  Male  76 1991-01-02 1991-01-07  TRUE      5
## 2  Male  58 1991-01-03       <NA> FALSE    1824
## 3  Male  74 1991-01-08 1991-06-02  TRUE     145
## 4 Female 77 1991-01-11 1991-01-13  TRUE      2
## 5 Female 76 1991-01-13       <NA> FALSE    1814
```

As classes de data e tempo no R cuidam de todos os detalhes sobre datas e horas, como anos bissextos, horários de verão e fusos horários. Abaixo é mostrado um exemplo onde um ano bissexto está envolvido e podemos observar que o número de dias entre as duas datas foi contado corretamente.

```
x <- as.Date("2016-03-01")
y <- as.Date("2016-02-28")
x-y
```

```
## Time difference of 2 days
```

9.2 Tempos no R

Tempos no R são representados pelas classes *POSIXct* ou *POSIXlt*. *POSIXct* é útil quando se deseja armazenar tempos em *data frames*, por exemplo.

POSIXlt é uma lista que armazena informações como o dia da semana, dia do ano, mês, dia do mês. Isso é útil quando precisamos deste tipo de informação.

Há um conjunto de funções genéricas que trabalham com datas e tempos para extrair partes de datas e/ou tempos, por exemplo:

- *weekdays*: fornece o dia da semana
- *months*: fornece o nome do mês
- *quarters*: fornece o número do trimestre (“Q1”, “Q2”, “Q3”, ou “Q4”)

Tempos podem ser criados por coerção de uma string, usando as funções `as.POSIXlt` ou `as.POSIXct`. O comando a seguir armazena no objeto `x` a hora do sistema por meio da função `Sys.time`. Vemos que `x` é da classe `POSIXct`.

```
x <- Sys.time() # hora do sistema  
x
```

```
## [1] "2021-12-09 21:41:04 -03"
```

```
class(x)
```

```
## [1] "POSIXct" "POSIXt"
```

Um objeto da classe `POSIXlt` contém metadados úteis

```
y <- as.POSIXlt(x)  
names(unclass(y))[1:8]
```

```
## [1] "sec"   "min"   "hour"  "mday"  "mon"   "year"  "wday"  "yday"
```

```
names(unclass(y))[9:11]
```

```
## [1] "isdst"  "zone"   "gmtoff"
```

```
y$wday  # dia da semana do valor de y
```

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

Não se pode utilizar os metadados de `POSIXlt` com um objeto `POSIXct`. O código abaixo irá gerar uma mensagem de erro:

```
x$sec # não se pode fazer isto com *POSIXct*
```

```
Error in x$sec : $ operator is invalid for atomic vectors
```

Já o código abaixo irá fornecer o valor desejado:

```
y$sec # isto pode, já que y é do tipo POSIXlt
```

```
## [1] 4.049686
```

A função *strptime* recebe um vetor de caracteres que contêm datas e tempos e os converte em um objeto da classe *POSIXlt*.

```
data <- c("Junho 13, 2019 11:40", "Janeiro 20, 2018 6:35")
x <- strptime(data, "%B %d, %Y %H:%M")
x

## [1] "2019-06-13 11:40:00 -03" "2018-01-20 06:35:00 -02"

class(x)

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

Os símbolos estranhos que aparecem no segundo argumento da função *strptime* (começando com "%") fixam o formato de datas e tempos. Veja a ajuda da função *strptime* (*?strptime*) para saber como formatar uma *string* de data.

Não se deve realizar operações com uma mistura de objetos das classes *Date*, *POSIXlt* e *POSIXct*. O código abaixo irá gerar o seguinte erro:

```
x <- as.Date("2019-06-01") # objeto da classe Date
y <- strptime("9 Jun 2019 10:01:45", "%d %b %Y %H:%M:%S") # objeto do tipo
# POSIXlt
y-x
```

```
Warning: Incompatible methods ("-.Date", "-.POSIXt") for "-"
Error in x - y: non-numeric argument to binary operator
```

Se convertermos x acima para *POSIXlt*, a operação a seguir pode ser realizada:

```
x <- as.Date("2019-06-01") # objeto da classe Date
y <- strptime("9 Jun 2019 10:01:45", "%d %b %Y %H:%M:%S") # objeto do tipo
# POSIXlt
x <- as.POSIXlt(x)
y-x

## Time difference of 8.542882 days
```

Abaixo é mostrado um exemplo de uma operação com diferentes fusos horários:

```
x <- as.POSIXct("2019-06-13 02:30:00")
y <- as.POSIXct("2019-06-13 09:30:00", tz = "GMT")
y-x

## Time difference of 4 hours
```

O metadado *year* da classe *POSIXlt* mostra o ano, contando a partir de 1900. Assim se quiséssemos extrair o ano da variável *dstr* (data do AVC) do conjunto de dados *stroke*, devemos fazer:

```
anoAVC = as.POSIXlt(stroke$dstr)$year + 1900  
head(anoAVC)
```

```
## [1] 1991 1991 1991 1991 1991 1991
```

Há pacotes no R que facilitam o trabalho com datas e tempos, como o pacote [lubridate](#).

9.3 Exercício

- 1) Vamos utilizar o conjunto de dados *udca* do pacote [survival](#) ([LGPL-2 | LGPL-2.1](#) | [LGPL-3](#)).
 - a) Verifique a ajuda para o conjunto de dados.
 - b) Carregue o conjunto de dados.
 - c) Gere uma nova variável no conjunto de dados que contenha o tempo de observação de cada paciente em dias: tempo desde a entrada no estudo até a última visita. Verifique que a classe dessa variável é *difftime*.
 - d) observe que se formos gerar resumos numéricos para esse tempo de observação via *R Commander*, essa variável não aparece na lista de variáveis numéricas. Converta essa variável para numérica, usando a função *as.numeric*.
 - e) Renove o conjunto de dados e gere os resumos numéricos para o tempo de observação dos pacientes.

Capítulo 10

Gerenciando uma sessão no R

Os conteúdos desta introdução e das seções 10.1 e 10.2 podem ser visualizados neste [vídeo](#).

Ao executar o R, o usuário inicia uma **sessão**, a qual persiste até o momento que o usuário fecha o programa ou encerra a sessão.

O menu *Session* do *RStudio* mostra diversas opções para gerenciar uma sessão do R (figura 10.1). Por esse menu, o usuário pode, por exemplo, criar uma nova sessão, terminar ou reiniciar a execução do R, encerrar a sessão e outras funções que serão vistas mais adiante.

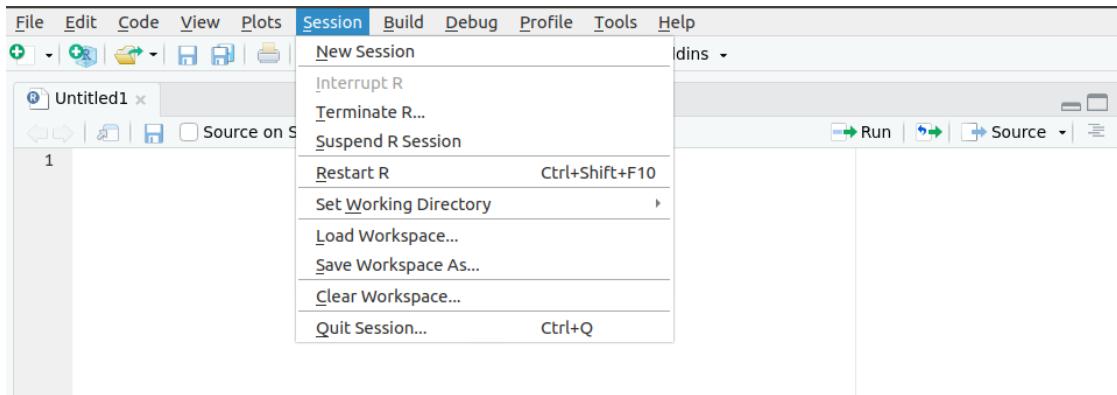


Figura 10.1: Menu Session do *RStudio*.

Todos os objetos criados ou carregados no R são armazenados em uma área comum de trabalho, denominada *workspace*.

Vamos iniciar uma sessão do R.

10.1 Listando objetos do *workspace* - função *ls*

Para verificarmos que objetos estão definidos no *workspace*, podemos usar a função *ls* (listar):

```
ls()  
## character(0)
```

No *RStudio*, uma das abas na janela superior direita, chamada *Environment*, exibe os objetos que estão no *workspace* da sessão corrente (figura 10.2).

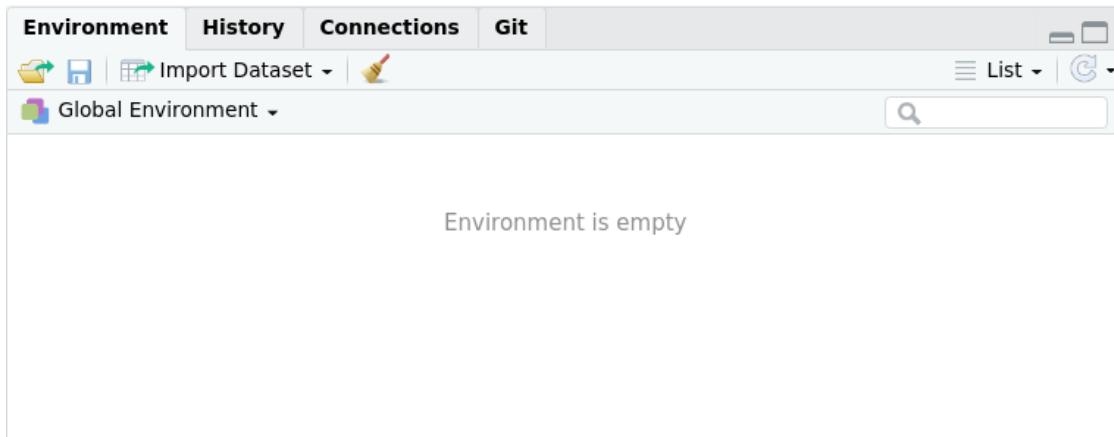


Figura 10.2: Aba ambiente (*environment*) no *RStudio*.

Aparentemente nada está carregado no *workspace*, mas a função *ls*, sem argumentos, não lista objetos cujos nomes se iniciam com “.”. Para isso, precisamos usar a função *ls(all=TRUE)*.

```
ls(all=TRUE)  
## [1] ".Random.seed" ".Table"
```

Há um objeto *.Random.seed* no *workspace*. A janela *Environment* não exibe objetos cujos nomes se iniciam com “.”.

Vamos novamente ler o conjunto de dados *Melanoma* do pacote *MASS* e listar novamente os objetos do *workspace*. Agora, o conjunto de dados *Melanoma* aparece na listagem e na janela *Environment* (figura 10.3).

```
library(MASS)  
data(Melanoma, package="MASS")  
ls()  
  
## [1] "Melanoma"
```



Figura 10.3: Ambiente do *RStudio* mostrando o objeto *Melanoma* recém carregado.

Ao clicarmos sobre o objeto *Melanoma* na janela *Environment*, detalhes do conjunto de dados são exibidos (figura 10.4).

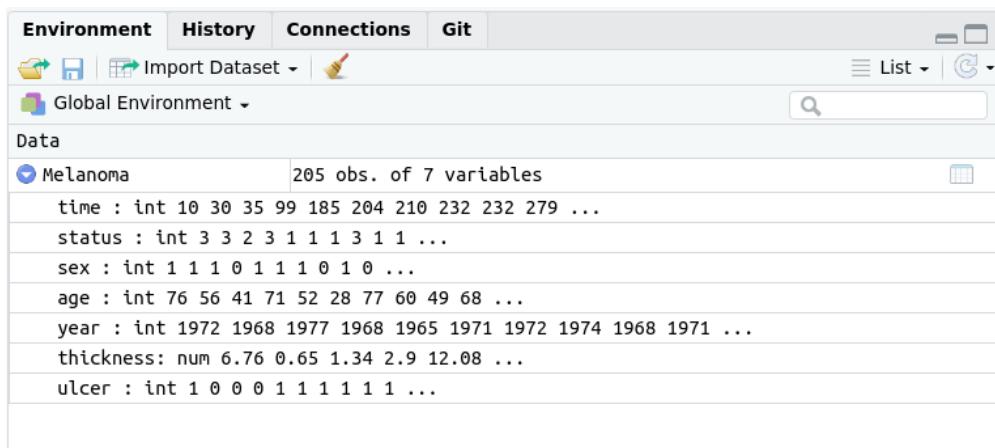


Figura 10.4: Detalhes do objeto *Melanoma* mostrados na aba *Environment* do *RStudio*.

10.2 Espaço de busca de objetos - a função *search*

Se quisermos listar os 10 primeiros valores da variável *age* (de *Melanoma*), usando o comando abaixo:

```
head(age, 10)
```

obteremos uma mensagem de erro parecida com esta: *Error in head(age, 10) : objeto ‘age’ não encontrado.*

Isso ocorre porque as variáveis de *Melanoma* não estão diretamente acessíveis ao R. Para acessá-las, precisamos precedê-las por *Melanoma\$*.

```
head(Melanoma$age, 10)
```

```
## [1] 76 56 41 71 52 28 77 60 49 68
```

Como o R localiza os objetos em seu ambiente?

De uma maneira simplificada, há diversos ambientes no R onde os objetos estão localizados. A função *search* lista o conjunto de ambientes disponíveis no R. Esse conjunto é chamado de espaço de busca, porque objetos nesses ambientes podem ser encontrados a partir do *workspace*. Vamos entender como isso funciona. Vejamos o resultado da execução da função *search* (figura 10.5).

```
search()
```

```
[1] ".GlobalEnv"      "package:MASS"      "tools:rstudio"      "package:stats"  
[5] "package:graphics" "package:grDevices" "package:utils"      "package:datasets"  
[9] "package:methods"   "Autoloads"       "package:base"
```

Figura 10.5: Espaço de busca do R.

A função *search* lista os ambientes em ordem, formando uma hierarquia onde o ambiente seguinte é o pai do anterior. O ambiente global (*.GlobalEnv*) é o *workspace*. O R busca os objetos de acordo com essa hierarquia, começando pelo *.GlobalEnv* e subindo na hierarquia até encontrar o objeto procurado. Caso o objeto não seja encontrado em nenhum ambiente, uma mensagem de erro é enviada.

Na lista apresentada acima, não se encontra o *Melanoma*. Como a variável *age* não existe nos demais ambientes do espaço de busca, uma mensagem de erro foi gerada anteriormente.

Vamos agora executar a função *attach(Melanoma)*:

```
attach(Melanoma)
```

Ao verificarmos o novo espaço de busca por meio da função *search*, o resultado é mostrado na figura 10.6.

```
search()
```

```
[1] ".GlobalEnv"      "Melanoma"       "package:MASS"      "tools:rstudio"  
[5] "package:stats"    "package:graphics" "package:grDevices" "package:utils"  
[9] "package:datasets" "package:methods"  "Autoloads"       "package:base"
```

Figura 10.6: Espaço de busca do R após “anexar” o conjunto de dados *Melanoma*.

Verificamos que o conjunto de dados *Melanoma* é listado logo após *.GlobalEnv* e passa a ser o segundo ambiente a ser verificado na busca por objetos.

Agora a variável *age* é localizada, sem a necessidade de especificarmos o conjunto de dados a que ela pertence:

```
head(age)
## [1] 76 56 41 71 52 28
```

Observações:

1) Ao alterarmos um *data frame* após o mesmo ser *attachado*, isso não irá alterar as variáveis disponíveis diretamente no espaço de busca, porque a função *attach* realiza uma cópia virtual do *data frame*. É como se outros objetos fossem criados com os mesmos nomes das variáveis do *data frame*. Vejamos os exemplos a seguir.

```
Melanoma$age[1] = 100      # alterando o primeiro valor da
                           # variável age dentro do Melanoma
head(Melanoma$age)

## [1] 100 56 41 71 52 28

Melanoma <- within(Melanoma, {  # convertendo a variável sex
                                # do Melanoma para factor
  sex <- factor(sex, labels=c('masculino','feminino'))
})
head(age)                  # a variável age no workspace não é alterada

## [1] 76 56 41 71 52 28

class(sex)                 # a variável sex no workspace não é alterada

## [1] "integer"
```

2) Da mesma forma, alterações nas variáveis “copiadas” do *data frame* não serão refletidas nas variáveis dentro do *data frame*. Vejam os comandos a seguir:

```
age[2] = 90      # alterando o segundo valor da variável age
head(Melanoma$age) # a variável age dentro do Melanoma não é alterada

## [1] 100 56 41 71 52 28
```

Ao executarmos a função *detach(Melanoma)*, o *Melanoma* é removido do espaço de busca (figura 10.7).

```
detach(Melanoma)

search()
```

```
[1] ".GlobalEnv"      "package:MASS"      "tools:rstudio"      "package:stats"  
[5] "package:graphics" "package:grDevices" "package:utils"      "package:datasets"  
[9] "package:methods"   "Autoloads"        "package:base"
```

Figura 10.7: Espaço de busca do R após a remoção do objeto *Melanoma*.

Podem haver diversos objetos com o mesmo nome no espaço de busca. Nesse caso, o R escolhe o primeiro que encontrar, seguindo a ordem do espaço de busca. Por isso é preciso tomar cuidado com objetos que são definidos fora de um *data frame*, porque eles serão usados antes de qualquer objeto com o mesmo nome em um *data frame* “atachado”. Pela mesma razão, não é uma boa ideia dar o mesmo nome a um *data frame* de uma variável do próprio *data frame*.

10.3 Especificando o diretório (pasta) corrente

O conteúdo desta e das seções seguintes deste capítulo podem ser visualizados neste [vídeo](#).

Ao importarmos um arquivo do excel para o R (seção 4.2), foi executado o seguinte comando:

```
Melanoma <- read_excel("~/temp/Melanoma.xlsx")
```

Nesse comando, foi necessário especificar o espaço do arquivo no sistema de arquivos. Caso o arquivo estivesse na pasta temp do disco C no Windows, o caminho *c:/temp/* teria que ser especificado juntamente com o nome do arquivo. Isso pode ser evitado se a pasta corrente de trabalho for *c:/temp*.

Para saber qual é o diretório corrente, usa-se a função *getwd*.

A função *setwd* especifica a pasta que será tratada como pasta corrente no R. Então poderíamos usar *setwd("c:/temp")* e só precisaríamos especificar o nome do arquivo a ser lido em *read_excel*.

No *RStudio*, podemos especificar o diretório corrente por meio da opção (figura 10.8):

Session ⇒ *Set Working Directory*

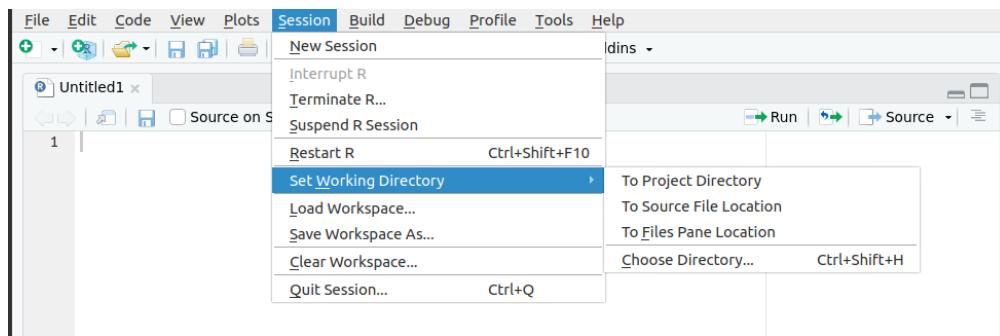


Figura 10.8: Menu do *RStudio* para selecionar o diretório de trabalho corrente.

No *RStudio*, é possível estabelecer o diretório corrente das seguintes formas, lidas de cima para baixo na figura 10.8:

- 1) o diretório do projeto que está aberto no *RStudio*, caso tenha algum;
- 2) o diretório do arquivo que está aberto na janela principal do *RStudio* (a ser visto no próximo capítulo);
- 3) o diretório aberto na aba *Files* do *RStudio* (área inferior à direita);
- 4) um diretório selecionado pelo usuário. Com essa opção, uma janela se abre para o usuário selecionar a pasta no sistema de arquivos que será considerada como o diretório corrente.

Para especificarmos o diretório corrente no *R Commander*, usamos a opção de menu:

Arquivo ⇒ Altere o diretório de trabalho...

A tela mostrada na figura 10.9 permite ao usuário navegar no sistema de arquivos e selecionar o diretório corrente

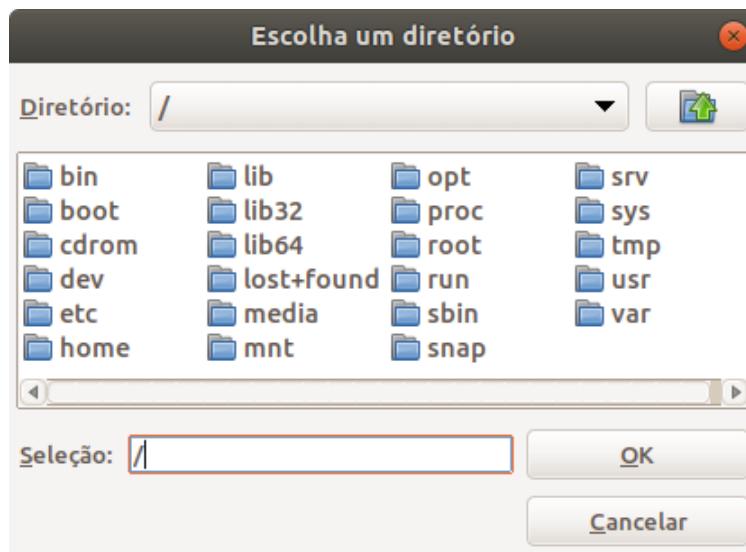


Figura 10.9: Caixa de diálogo para selecionar o diretório de trabalho corrente.

10.4 Manipulando o workspace

É possível salvar o *workspace* em um arquivo a qualquer momento, por meio da função `save.image("nome_do_arquivo")`. Por exemplo:

```
save.image("/home/sergio/teste.Rdata") # o workspace será gravado no  
# arquivo teste.Rdata no  
# diretório /home/sergio
```

Se usarmos a função `save.image` sem argumentos, o *workspace* será gravado em um arquivo chamado *.RData* no diretório corrente.

Para salvarmos o *workspace* no *RStudio*, basta acessar a opção de menu abaixo, escolher a pasta e fornecer o nome do arquivo a ser gravado:

Session ⇒ *Save Workspace As...*

Analogamente, para salvarmos o *workspace* no *R Commander*, basta acessar a opção de menu abaixo, escolher a pasta e fornecer o nome do arquivo a ser gravado:

Arquivo ⇒ *Salvar workspace do R...*

ou

Arquivo ⇒ *Salvar workspace do R como...*

Para carregarmos um *workspace* em uma sessão do R, usamos a função `load("nome_do_arquivo")`.

```
load("/home/sergio/teste.Rdata") # carregando o workspace  
# gravado anteriormente
```

O arquivo *.RData* é carregado por padrão sempre que o R for iniciado em um diretório que contenha um arquivo com esse nome.

Para carregarmos um *workspace* via *RStudio*, basta acessar a opção de menu abaixo, escolher a pasta e o arquivo a ser carregado:

Session ⇒ *Load Workspace...*

Ao sairmos do R, sempre será perguntado por padrão se desejamos salvar o *workspace*.

Os pacotes que foram carregados em uma sessão não são considerados como parte do *workspace*. Ao iniciarmos uma nova sessão no R com um *workspace* salvo em disco, será necessário carregar os pacotes novamente.

Os resultados gerados pela execução de funções no R também não são gravados no *workspace*.

Para limparmos o *workspace* via *RStudio*, basta acessar a opção de menu:

Session ⇒ *Clear Workspace...*

Uma tela de confirmação irá perguntar se é isso mesmo que o usuário deseja.

10.5 Removendo objetos do workspace - a função *rm*

A função *rm* remove os objetos, que devem ser especificados dentro dos parênteses (separados por vírgulas).

```
rm(Melanoma)
```

Os objetos do *workspace* podem ser removidos completamente por meio da função *rm(list=ls())*. Isso não remove as variáveis cujos nomes começam com um ponto.

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

```
## character(0)
```

Se executarmos a função *rm(list=ls(all=T))*, todos os objetos do *workspace* serão removidos incluindo os que começam com ponto. **Isso não é recomendável** porque esses nomes são usados pelo sistema.

10.6 Histórico de comandos

Cada função ou comando executado no R vai sendo armazenado no histórico da sessão. No *RStudio*, a aba na janela superior direita, chamada *History* exibe os comandos anteriores executados na sessão (figura 10.10).

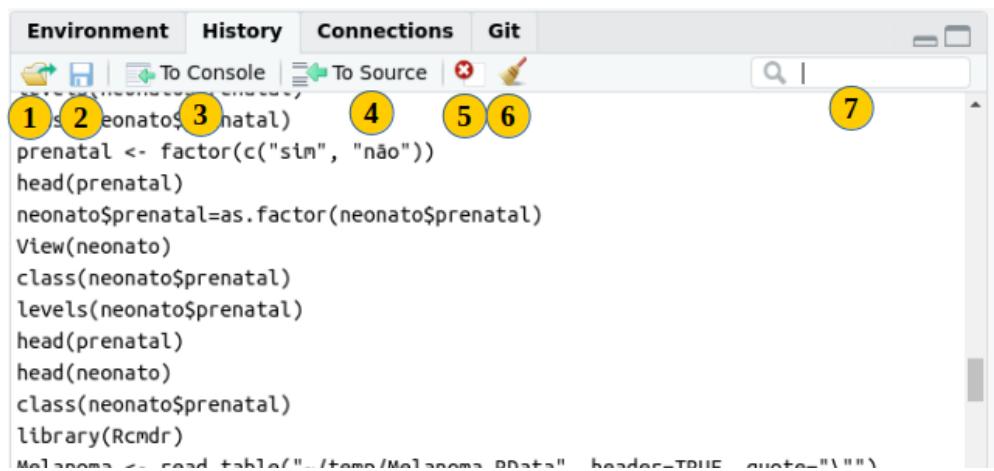


Figura 10.10: Aba *History* do *RStudio* mostrando o histórico dos comandos executados na console do *RStudio*.

As opções na barra de ferramentas da aba *History*, indicada pelos números na figura 10.10 permitem as seguintes operações com o histórico:

- 1) carregar um histórico de comandos de um arquivo (função `loadhistory("nome_do_arquivo")`);
- 2) salvar o histórico em um arquivo (função `savehistory("nome_do_arquivo")`). Usualmente, se usa a extensão *Rhistory* no arquivo;
- 3) copiar os comandos selecionados no histórico para a console do *RStudio*;
- 4) copiar os comandos selecionados no histórico para um arquivo de texto na janela principal do R. Essa janela será vista no próximo capítulo;
- 5) apagar os os comandos selecionados no histórico;
- 6) limpar todo o histórico;
- 7) realizar uma busca por um termo no histórico.

Um arquivo de histórico de comandos é um arquivo de texto e, como tal, pode ser aberto em qualquer editor de texto. A listagem a seguir mostra o histórico de comandos executados neste capítulo:

```

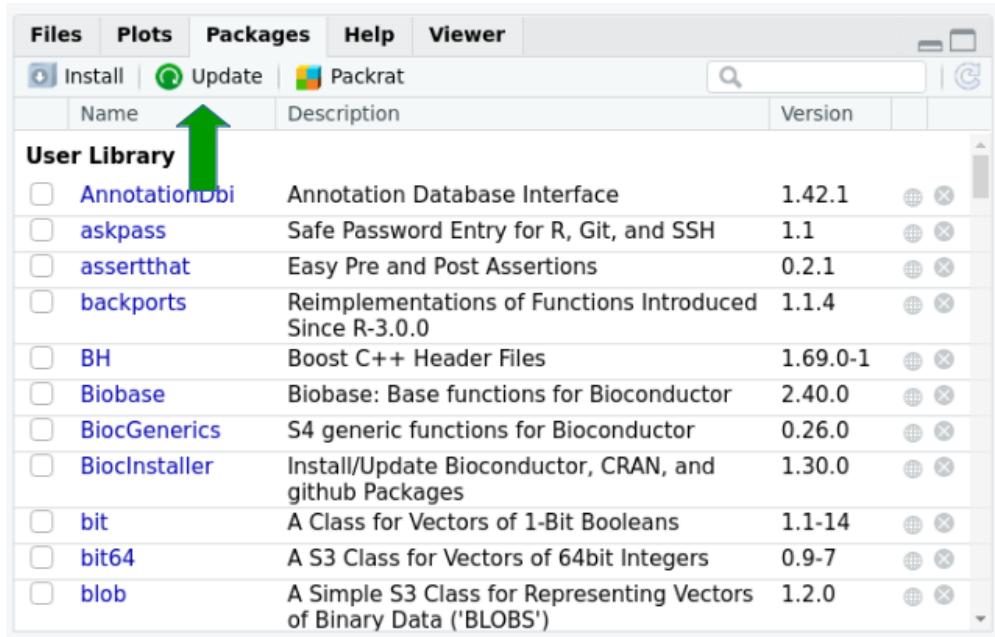
ls()
ls(all=TRUE)
library(MASS)
data(Melanoma, package="MASS")
ls()
head(age, 10)
head(Melanoma$age, 10)
search()
attach(Melanoma)
search()
head(age)
Melanoma$age[1] = 100
head(Melanoma$age)

```

```
Melanoma <- within(Melanoma, { # convertendo a variável sex para factor
  sex <- factor(sex, labels=c('masculino','feminino')))
})
head(age)
class(sex)
detach(Melanoma)
search()
save.image("~/home/sergio/teste.Rdata")
load("~/home/sergio/teste.Rdata")
rm(melanoma_idoso)
melanoma_idoso
rm(list=ls())
ls()
```

10.7 Atualizando pacotes

Tanto o núcleo do R quanto os pacotes são frequentemente atualizados. Para atualizar pacotes via *RStudio*, clicamos no botão *Update*, na aba *Packages* (figura 10.11) e, em seguida, selecionamos os pacotes a serem atualizados e clicamos no botão *Install Updates* (figura 10.12).



| Name | Description | Version | Actions |
|--|---|----------|---|
| User Library | | | |
| <input type="checkbox"/> AnnotationDbi | Annotation Database Interface | 1.42.1 |   |
| <input type="checkbox"/> askpass | Safe Password Entry for R, Git, and SSH | 1.1 |   |
| <input type="checkbox"/> assertthat | Easy Pre and Post Assertions | 0.2.1 |   |
| <input type="checkbox"/> backports | Reimplementations of Functions Introduced Since R-3.0.0 | 1.1.4 |   |
| <input type="checkbox"/> BH | Boost C++ Header Files | 1.69.0-1 |   |
| <input type="checkbox"/> Biobase | Biobase: Base functions for Bioconductor | 2.40.0 |   |
| <input type="checkbox"/> BiocGenerics | S4 generic functions for Bioconductor | 0.26.0 |   |
| <input type="checkbox"/> BiocInstaller | Install/Update Bioconductor, CRAN, and github Packages | 1.30.0 |   |
| <input type="checkbox"/> bit | A Class for Vectors of 1-Bit Booleans | 1.1-14 |   |
| <input type="checkbox"/> bit64 | A S3 Class for Vectors of 64bit Integers | 0.9-7 |   |
| <input type="checkbox"/> blob | A Simple S3 Class for Representing Vectors of Binary Data ('BLOBS') | 1.2.0 |   |

Figura 10.11: Aba *Packages* do *RStudio*. A seta verde mostra o botão para atualizar pacotes instalados.

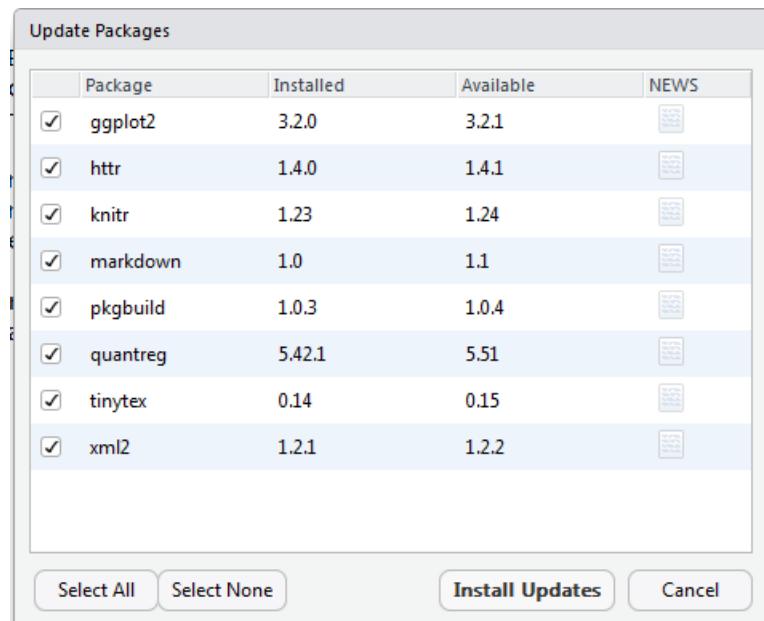


Figura 10.12: Tela para atualizar pacotes do R no *RStudio*.

10.8 Informações sobre a sessão

A função *sessionInfo* imprime informação de versão do R, do sistema operacional e pacotes “atachados” ou carregados (figura 10.13). Essas informações são úteis, por exemplo, para registrar o ambiente em que foi executado determinado *script*, o que pode facilitar a identificação de erros na execução de alguma função.

```
sessionInfo()
```

```

R version 3.6.3 (2020-02-29)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.3 LTS

Matrix products: default
BLAS:    /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK:  /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3

locale:
[1] LC_CTYPE=pt_BR.UTF-8          LC_NUMERIC=C           LC_TIME=pt_BR.UTF-8
[4] LC_COLLATE=pt_BR.UTF-8       LC_MONETARY=pt_BR.UTF-8   LC_MESSAGES=pt_BR.UTF-8
[7] LC_PAPER=pt_BR.UTF-8        LC_NAME=C              LC_ADDRESS=C
[10] LC_TELEPHONE=C            LC_MEASUREMENT=pt_BR.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats      graphics   grDevices utils      datasets   methods    base

other attached packages:
[1] MASS_7.3-53

loaded via a namespace (and not attached):
[1] bookdown_0.24   packrat_0.7.0   digest_0.6.28   magrittr_2.0.1  evaluate_0.14
[6] rlang_0.4.11    stringi_1.7.5   rstudioapi_0.13 rmarkdown_2.11 tools_3.6.3
[11] stringr_1.4.0   tinytex_0.34    xfun_0.26      yaml_2.2.0     fastmap_1.1.0
[16] compiler_3.6.3  htmltools_0.5.2  knitr_1.36

```

Figura 10.13: Informações sobre a sessão corrente do R.

Capítulo 11

Estruturas de controle do fluxo de execução e funções

Os conteúdos desta introdução e da seção 11.1 podem ser visualizados neste [vídeo](#).

Nos capítulos anteriores, utilizamos dezenas de funções para realizar os processamentos desejados com os dados. O uso de funções é um dos principais instrumentos para modularizar uma linguagem de programação.

As funções isolam um código que podem ser executado inúmeras vezes. Elas criam uma interface para o código, por meio de zero ou mais argumentos, que permite ao usuário utilizar a função sem conhecer os detalhes de como ela é executada.

Funções são elas próprias objetos. Assim podem ser tratadas como quaisquer outros objetos no R.

Ao escrevermos uma função, frequentemente diversas estruturas de controle do fluxo de execução dos comandos são utilizadas. Estruturas de controle frequentemente usadas são:

- *if-else*: testa uma condição e executa um código dependendo do resultado do teste;
- *for*: executa uma sequência de comandos (laço - *loop*) um certo número de vezes;
- *while*: executa um laço enquanto uma condição for verdadeira;
- *repeat*: executa um laço um número infinito de vezes (para parar é preciso usar *break*);
- *break*: interrompe a execução de um laço;
- *next*: pula uma iteração de um laço.

Essas estruturas podem ser utilizadas também fora de funções, em sessões interativas com o R. Vamos entendê-las antes de entrarmos em funções.

11.1 *if-else*

Como os termos indicam, uma estrutura do tipo *if-else* permite a execução de um ou outro código, dependendo de uma condição ser verdadeira ou falsa.

A sintaxe básica é mostrada abaixo. Observem o uso de uma expressão lógica entre parênteses e chaves envolvendo os códigos executados como cada resultado da expressão lógica.

```
if (expressão lógica) { # se expressão lógica for verdadeira
    # faça isso
} else {
    # senão faça outra coisa
}
```

É possível não termos a parte do *else*, como mostrado abaixo. Nesse caso, um código será executado se a expressão lógica for verdadeira. Se a expressão não for verdadeira, então nada será executado.

```
if (expressão lógica) { # se expressão lógica for verdadeira
    # faça isso
} # se não, não faça nada
```

É possível termos *ifs* aninhados dentro de outros *ifs* quantas vezes forem necessárias, como abaixo.

```
if (expressão lógica) { # se expressão lógica for verdadeira
    # faça isso
} else if (outra expressão lógica) {
    # se outra expressão for verdadeira, faça isso
} else {
    # se outra expressão não for verdadeira, faça isso
}
```

As expressões lógicas são construídas por meio dos operadores == (igual), > (maior), < (menor), >= (maior ou igual), <= (menor ou igual), != (diferente), ou combinações de expressões lógicas por meio dos operadores booleanos “OU” (|), “E” (&), e “Não” (!). É recomendável isolar cada expressão lógica por parênteses para facilitar a leitura do código.

Vamos ver um exemplo:

```
x = sample(6,1) # simula o lançamento de um dado não viciado
if ( (x %% 2) == 0) {
    print(paste(x, "é par"))
} else {
    print(paste(x, "é ímpar"))
}

## [1] "3 é ímpar"
```

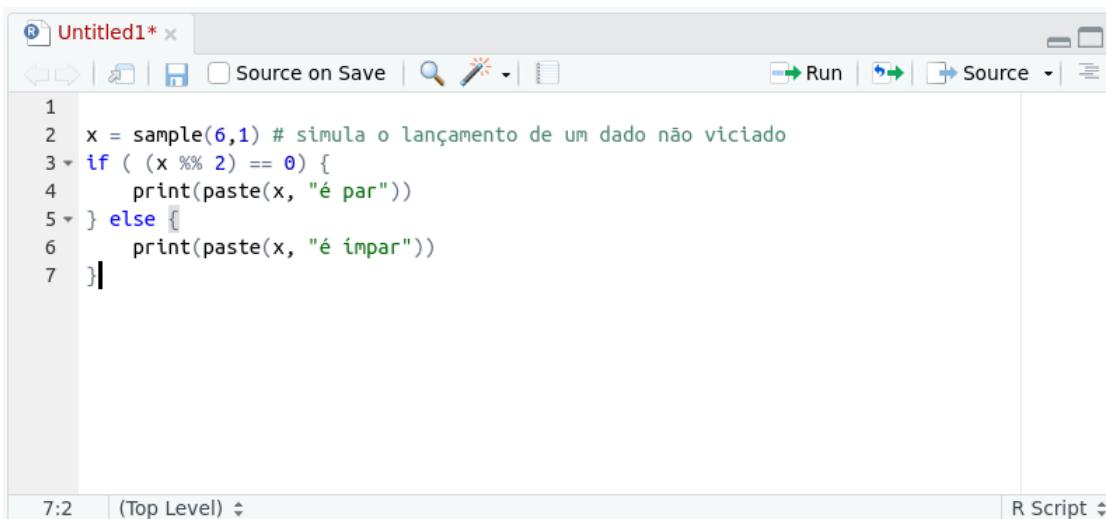
O primeiro comando simula o lançamento de um dado, gerando aleatoriamente um número inteiro de 1 a 6 e armazena o resultado no objeto x.

O comando *if* avalia a expressão lógica entre parênteses. Essa expressão testa se o resto da divisão de x por 2 é igual a zero. O símbolo de porcentagem duplicado realiza a operação para obter o resto da divisão do número à esquerda pelo número à direita.

Se o resto da divisão for igual a zero, então será executada a função *print*, que imprime na tela que o valor de x é par.

Se o resto da divisão não for igual a zero, então será executada a função *print*, que imprime na tela que o valor de x é ímpar.

Vamos utilizar a janela de edição de arquivos do *RStudio* (área superior à esquerda) para executarmos uma série de comandos simultaneamente. A figura 11.1 mostra os comandos acima digitados na área de edição de arquivos no *RStudio*.



```
1 x = sample(6,1) # simula o lançamento de um dado não viciado
2 if ( (x %% 2) == 0 ) {
3   print(paste(x, "é par"))
4 } else {
5   print(paste(x, "é ímpar"))
6 }
7 |
```

Figura 11.1: Área superior à esquerda do *RStudio* com o código mostrado anteriormente.

Se selecionarmos os comandos no arquivo (ou parte deles) e, em seguida, clicarmos no botão *Run* (figura 11.2), os comandos selecionados serão executados na console do *RStudio*.

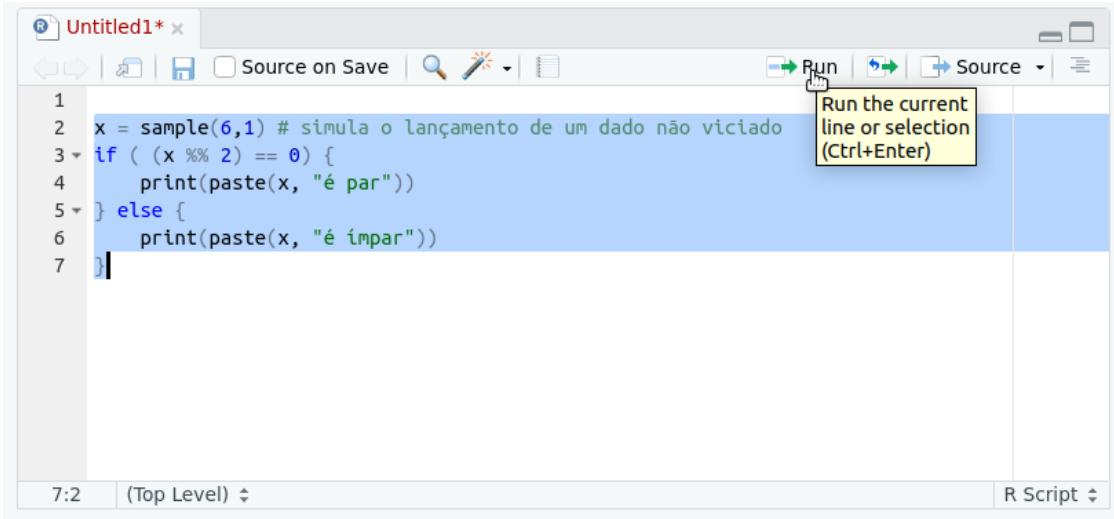


Figura 11.2: Área de script do *RStudio*. Ao selecionarmos os comandos e clicarmos no botão *Run*, os comandos serão executados na console do *RStudio*.

11.2 Laços com *for* (repetições)

Os conteúdos desta seção e das seções 11.3, 11.4, 11.5 e 11.6 podem ser visualizados neste [vídeo](#).

Frequentemente, ao desenvolvermos um script no R, precisamos executar uma sequência de comandos um certo número de vezes. Essa estrutura de repetições de uma sequência de comandos é chamada de laço (*loop* em inglês). Cada uma das repetições é chamada de **iteração**. O conjunto de comandos é executado uma vez e, em seguida, o fluxo de execução retorna ao primeiro comando do conjunto e o processo continua um certo número de vezes.

Suponhamos que queiramos somar os elementos de um vetor numérico e não tivéssemos a função *sum* à nossa disposição. Uma maneira de realizar essa soma no R é usar uma estrutura como no exemplo a seguir:

```
# soma de números usando o laço "for"
x = c(11:13)
soma = 0
n = length(x)
for (i in 1:n) {
  soma = soma + x[i]
  print(paste("i = ", i, " ", ", x[", i, "] = ", x[i], " ",
             ", soma = ", soma, sep=''))
}
## [1] "i = 1 , x[1] = 11 , soma = 11"
## [1] "i = 2 , x[2] = 12 , soma = 23"
## [1] "i = 3 , x[3] = 13 , soma = 36"
```

```
soma
## [1] 36
```

Os laços formados com a palavra chave *for* executam os comandos entre chaves quantas vezes a expressão entre parênteses especificar. No exemplo acima, criamos um vetor (*x*) com os números inteiros de 11 a 13. Depois iniciamos a variável *soma* com o valor 0 e *n* com o comprimento do vetor *x*, nesse caso, *n* = 3. O comando *soma* = *soma* + *x[i]* e print (...) serão executados para cada valor de *i*, que assume em sequência os valores de 1 até 3. Assim a variável *soma* vai acumulando sucessivamente os valores do vetor *x*.

Existe uma função chamada *seq_len* que substitui a expressão *1:length*. Assim o laço *for* acima poderia ser escrito como:

```
for (i in seq_len(x)) {
  soma = soma + x[i]
  print(paste("i = ", i, " ", ", x[[" , i, "]] = ", x[i], " ",
             ", soma = ", soma, sep=''))
}
```

11.3 Laços com *while*

Laços do tipo *while* possuem a seguinte estrutura:

```
while ( expressão lógica ) { # se expressão lógica for verdadeira
  # sequência de comandos }
```

Inicialmente, a expressão lógica é avaliada. Se ela for verdadeira, a sequência de comandos entre chaves é executada. Em seguida a expressão lógica é novamente avaliada e assim por diante, até que a expressão lógica seja falsa e então o fluxo de execução sai do laço.

Vamos reescrever o código para calcular a soma dos elementos de um vetor numérico da seção 11.2, utilizando o laço com *while*, mostrado abaixo.

```
# soma de números usando o laço "while"
x = c(11:13)
n = length(x) # n recebe o valor do comprimento do vetor x, no caso n = 3
i = 1
soma = 0
while (i <= n) { # executa os comandos entre chaves 3 vezes, com i
  # assumindo os valores de 1 a 3 sucessivamente
  soma = soma + x[i]
  print(paste("i = ", i, " ", ", x[[" , i, "]] = ", x[i], " ",
             ", soma = ", soma, sep=''))
  i = i + 1
}
```

```

## [1] "i = 1 , x[1] = 11 , soma = 11"
## [1] "i = 2 , x[2] = 12 , soma = 23"
## [1] "i = 3 , x[3] = 13 , soma = 36"

soma

## [1] 36

```

Observem a expressão lógica $i \leq n$ que é testada a cada iteração. Inicialmente i é igual a 1 e é aumentado de uma unidade a cada iteração. Quando i atinge um valor maior que o comprimento de x , o laço é interrompido.

Nesse exemplo, o laço com *for* é mais simples. Em laços com *while*, é preciso tomar cuidado para que a expressão lógica não seja sempre verdadeira. Caso a expressão lógica nunca se torne falsa, o laço nunca terminará.

11.4 Laços com *repeat* e *break*

Um laço com *repeat* no R é executado sucessivas vezes até ser interrompido pelo comando *break*.

Vamos novamente reescrever o código para calcular a soma dos elementos de um vetor numérico da seção 11.2, utilizando agora o laço com *repeat*:

```

x = c(11:13)
soma = 0 # inicia a variável soma com 0
n = length(x) # n recebe o valor do comprimento do vetor x, no caso n = 3
i = 1
repeat { # executa os comandos entre {} enquanto i for menor ou igual a n
  soma = soma + x[i]
  print(paste("i = ", i, ", x[", i, "] = ", x[i], ", soma = ", soma, sep=' '))
  i = i + 1
  if (i > n) { # quando i > n, o comando break é executado
    break      # e o laço é interrompido
  }
}

## [1] "i = 1, x[1] = 11, soma = 11"
## [1] "i = 2, x[2] = 12, soma = 23"
## [1] "i = 3, x[3] = 13, soma = 36"

soma

## [1] 36

```

O fluxo do laço *repeat* é parecido com o do laço *while*. A diferença é que a condição é testada em qualquer ponto do laço, não antes da execução do laço. Inicialmente i é igual

a 1 e é aumentado de uma unidade a cada iteração. Quando i atinge um valor maior que o comprimento de x , a condição $i > n$ é verdadeira e o comando *break* é executado, interrompendo o laço.

O comando *break*, como o nome indica, interrompe a execução de um laço sempre que for executado.

Os códigos dos laços *for*, *while* e *repeat* mostrados nesses exemplos são apenas para fins didáticos. Nós poderíamos obter a soma dos elementos do vetor x , usando a função *sum* dessa forma:

```
sum(x)
```

```
## [1] 36
```

11.5 *next*

O comando *next* é usado para interromper uma iteração de um laço.

Vamos supor que temos um vetor com os números inteiros de 10 a 30 e que desejamos somar somente os elementos de x a partir do décimo-primeiro elemento. O código a seguir realiza essa soma:

```
x = c(10:30)
n = length(x)
soma = 0
for (i in 1:n) {
  if (i < 11) {
    next
  }
  soma = soma + x[i]
  print(paste("i = ", i, ", x[", i, "] = ", x[i], ", soma = ",
             soma, sep=' '))
}
## [1] "i = 11, x[11] = 20, soma = 20"
## [1] "i = 12, x[12] = 21, soma = 41"
## [1] "i = 13, x[13] = 22, soma = 63"
## [1] "i = 14, x[14] = 23, soma = 86"
## [1] "i = 15, x[15] = 24, soma = 110"
## [1] "i = 16, x[16] = 25, soma = 135"
## [1] "i = 17, x[17] = 26, soma = 161"
## [1] "i = 18, x[18] = 27, soma = 188"
## [1] "i = 19, x[19] = 28, soma = 216"
## [1] "i = 20, x[20] = 29, soma = 245"
## [1] "i = 21, x[21] = 30, soma = 275"
```

```
soma  
## [1] 275
```

No laço acima, enquanto $i < 11$, o comando *next* é executado, interrompendo a iteração corrente e indo para o início da próxima iteração. Então, para $i = 1, 2, \dots, 11$, os dois comandos após o *if* não são executados.

Quando i for igual a 11 em diante, o comando *next* não é executado e os comandos após o *if* são executados. A variável *soma* ao final vai ser igual à adição dos números do vetor a partir do décimo-primeiro elemento.

Novamente esse exemplo visa somente a ilustrar o uso do comando *next*. O comando a seguir obtém o mesmo resultado do laço acima.

```
sum(x[11:length(x)])  
## [1] 275
```

Assim, antes de utilizarmos construções mais complexas, usando *if* ou laços, tentemos verificar se não é possível obter a funcionalidade que desejamos por meio de funções já disponíveis no R.

11.6 Laços aninhados

Laços podem ser aninhados quantas vezes forem necessárias, embora, normalmente, mais de três níveis de aninhamento seja raramente necessário.

O código abaixo calcula a soma dos elementos de uma matriz, onde no laço externo, i percorre as linhas da matriz e, no laço interno, j percorre as colunas da matriz.

Inicialmente criamos uma matriz com 2 linhas e 3 colunas, contendo os números 1 a 3 na primeira linha e os números 4 a 6 na segunda linha.

```
x <- matrix(1:6, 2, 3)  
x  
##      [,1] [,2] [,3]  
## [1,]     1     3     5  
## [2,]     2     4     6
```

Em seguida, codificamos o laço que realiza a soma dos elementos da matriz.

```

soma = 0
n_linhas = nrow(x) # n_linhas igual ao número de linhas da matriz x (2)
n_colunas = ncol(x) # n_colunas igual ao número de colunas da matriz x (3)
for(i in 1:n_linhas) {
  for(j in 1:n_colunas) {
    soma = soma + x[i,j]
  }
  print(paste("i = ", i, ", soma acumulada = ", soma))
}
## [1] "i = 1 , soma acumulada = 9"
## [1] "i = 2 , soma acumulada = 21"

soma
## [1] 21

```

Primeiramente inicializamos a variável *soma* com 0, *n_linhas* com o número de linhas da matriz *x* (2), e *n_colunas* com o número de colunas da matriz *x* (3).

Ao entrarmos no laço externo, $i = 1$. Agora, o laço interno vai ser então executado com os valores de $i = 1$ e j variando de 1 a 3. Então, como vimos nos laços anteriores, *soma* vai assumir a soma dos valores da primeira linha da matriz, $x[1,1] + x[1, 2] + x[1,3]$ e o fluxo de execução sai do laço interno e passa ser o próximo comando após o laço interno. A função *print* imprime os valores correntes de *i* e *soma*.

Em seguida, o fluxo retorna para o laço mais externo, *i* passa a ser 2 e o laço interno será executado mais uma vez, agora com $i = 2$ e j variando de 1 a 3 novamente. No laço interno, vamos somando a variável *soma* sucessivamente com os valores da segunda linha de *x*. Então *soma* passa a ser $9 + 2 + 4 + 6 = 21$. Ao somarmos o terceiro elemento da segunda linha, abandonamos o laço interno e são impressos os valores correntes de *i* e *soma*.

Como *i* percorreu os valores de 1 a 2, o laço externo não será mais executado. O fluxo de execução passa para a próxima a linha após o laço externo e é exibido o valor final de *soma* (21).

Outra vez, esse exemplo visa somente a ilustrar o uso de laços aninhados. A soma acima seria obtida de maneira mais fácil com a função a função *sum*.

```

sum(x)
## [1] 21

```

11.7 Estrutura básica de uma função

O conteúdo desta seção pode ser visualizado neste [vídeo](#).

Nós temos usado muitas funções ao longo desses vídeos. As funções conferem modularidade a uma linguagem de programação e permite a reutilização de código.

Em vez de o usuário ter que escrever códigos extensos para realizar uma análise de seu interesse, basta combinar funções escritas por terceiros que, em conjunto, produzem os resultados desejados. Dessa forma, os usuários não precisam conhecer os detalhes de como cada função foi implementada, basta saber as funcionalidades que ela proporciona e como ela deve ser chamada.

Uma função embute dentro dela uma sequência de comandos que realiza a funcionalidade sugerida pelo seu nome. Uma função bem simples é apresentada abaixo.

```
minha_funcao <- function() {      # definindo a função
  print("Alô mundo!")
}
minha_funcao()          # chamando a função

## [1] "Alô mundo!"
```

Para criarmos uma função, usamos a palavra chave *function* seguida de uma lista de argumentos formais entre parênteses e separados por vírgulas. O código que é executado cada vez que a função é chamada aparece entre chaves. No exemplo acima, a função simplesmente imprime na tela a expressão “Alô mundo!”. Para executarmos a função, usamos o nome dela com a lista de argumentos entre parênteses. Nesse exemplo, a função não possui argumentos.

Vamos criar uma função mais complexa, que realiza a soma dos elementos de um vetor numérico.

```
soma.vetor <- function(vetor, na.rm = TRUE) {
  if (!is.numeric(vetor) & !is.integer(vetor)) {
    print ("vetor não é numérico")
    return()
  }
  if (na.rm == TRUE) {
    vetor_sem_NA = vetor[!is.na(vetor)]
  } else if (any(is.na(vetor)) == TRUE) {
    return (NA)
  }
  soma = 0
  for (valor in vetor_sem_NA) {
    soma = soma + valor
  }
  soma
}
```

O nome dado para a função é *soma.vetor*. Essa função possui dois argumentos formais: o primeiro argumento é o vetor cujos elementos desejamos somar, o segundo argumento define se é para remover os elementos do vetor que são *NA*.

Cada argumento formal de uma função possui um nome, o qual pode ser usado para especificar que valor o argumento irá assumir na chamada de uma função. Ao segundo argumento *na.rm*, foi atribuído o valor *default TRUE*.

A primeira parte da função *soma.vetor* testa se o vetor é numérico ou inteiro. Se não for, uma mensagem é impressa na tela e a função retorna, sem executar o restante do código.

Se o vetor for numérico ou inteiro, a execução da função continua.

O comando seguinte verifica o valor do argumento *na.rm*.

Se o valor de *na.rm* for igual a *TRUE*, então todos os elementos do vetor que são *NA* são removidos por meio do comando *vetor[!is.na(vetor)]*, e o resultado é armazenado no objeto *vetor_sem_NA*.

Se o valor de *na.rm* for igual a *FALSE*, então, se houver algum valor *NA* no vetor, a função irá encerrar, retornando o valor *NA*.

Finalmente, se somente houver valores numéricos no vetor, ou *na.rm* é igual a *TRUE*, o trecho de código seguinte percorre todos os elementos de *vetor_sem_NA* e vai acumulando a soma dos elementos no objeto *soma*, que é retornado ao final da execução da função.

O valor da última expressão executada no código da função é o valor de retorno da mesma.

A figura 11.3 mostra o código da função *soma.vetor* na janela de edição de arquivos do *RStudio*.

```

1  #
2  # definindo uma função para realizar a soma de
3  # elementos de um vetor numérico
4  #
5  soma.vetor <- function(vetor, na.rm = TRUE) {
6    if (!is.numeric(vetor) & !is.integer(vetor)) {
7      print ("vetor não é numérico")
8      return()
9    }
10   if (na.rm == TRUE) {
11     vetor_sem_NA = vetor[!is.na(vetor)]
12   } else if (any(is.na(vetor)) == TRUE) {
13     return (NA)
14   }
15   soma = 0
16   for (valor in vetor_sem_NA) {
17     soma = soma + valor
18   }
19   soma
20 }

```

1:1 (Top Level) R Script

Figura 11.3: Código da função *soma.vetor* na janela de edição de arquivos do *RStudio*.

Poderíamos selecionar todas as linhas que definem a função e clicar no botão *Run* para compilar a função e armazená-la como um objeto no *workspace*. Porém vamos salvar o *script* da função em um arquivo e aprendermos como carregar uma função no R a partir de um arquivo que contém o seu código. A figura 11.4 mostra a aba *Files* do *RStudio* (área inferior à direita). Nessa figura, a pasta selecionada é *temp* e ela mostra os arquivos dessa pasta.

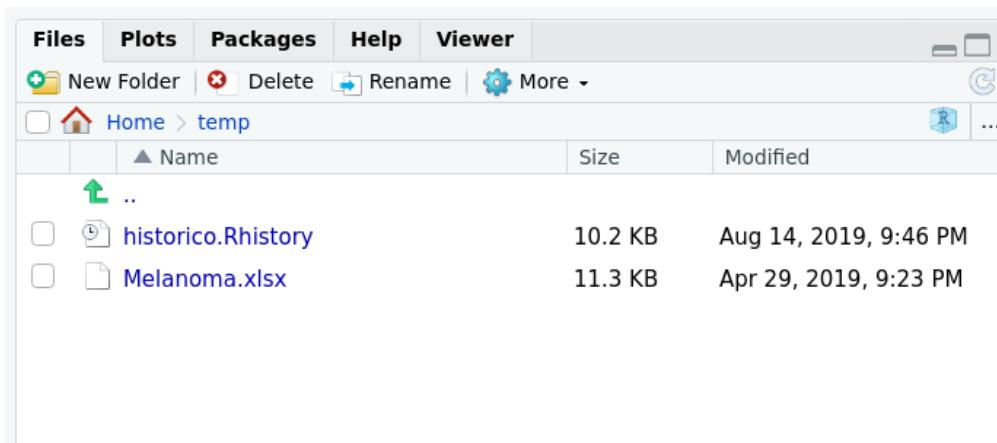


Figura 11.4: Aba *Files* do *RStudio*.

Na aba *Files*, é possível navegar no sistema de arquivos de seu computador, remover e renomear arquivos, criar pastas etc. Além disso, ao clicarmos no ícone *More* (figura 11.5), é

possível, entre outras coisas, fazermos com que a pasta exibida na aba *Files* passe a ser o diretório corrente, ou irmos para a pasta que foi estabelecida anteriormente como diretório corrente.

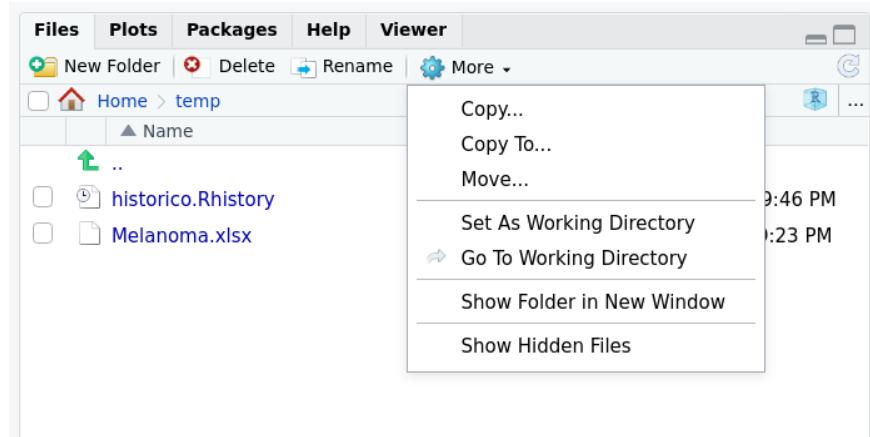


Figura 11.5: Aba *Files* do *RStudio* com a opção para especificarmos a pasta mostrada em *Files* como o diretório corrente.

Ao clicarmos no ícone indicado pela seta verde na figura 11.6 na área de edição de arquivos, poderemos salvar o *script* da função *soma.vetor* em um arquivo. Poderemos então selecionar a pasta e o nome do arquivo que será gravado (figura 11.7). Arquivos contendo um *script* do R, por convenção, devem possuir a extensão *R*.

```

1 # definindo uma função para realizar a soma de
2 # elementos de um vetor numérico
3 #
4 soma.vetor <- function(vetor, na.rm = TRUE) {
5   if (!is.numeric(vetor) & !is.integer(vetor)) {
6     print ("vetor não é numérico")
7     return()
8   }
9   if (na.rm == TRUE) {
10     vetor_sem_NA = vetor[!is.na(vetor)]
11   } else if (any(is.na(vetor)) == TRUE) {
12     return (NA)
13   }
14   soma = 0
15   for (valor in vetor_sem_NA) {
16     soma = soma + valor
17   }
18   soma
19 }
20 }

```

Figura 11.6: Ícone para salvar o *script* em um arquivo no *RStudio*.

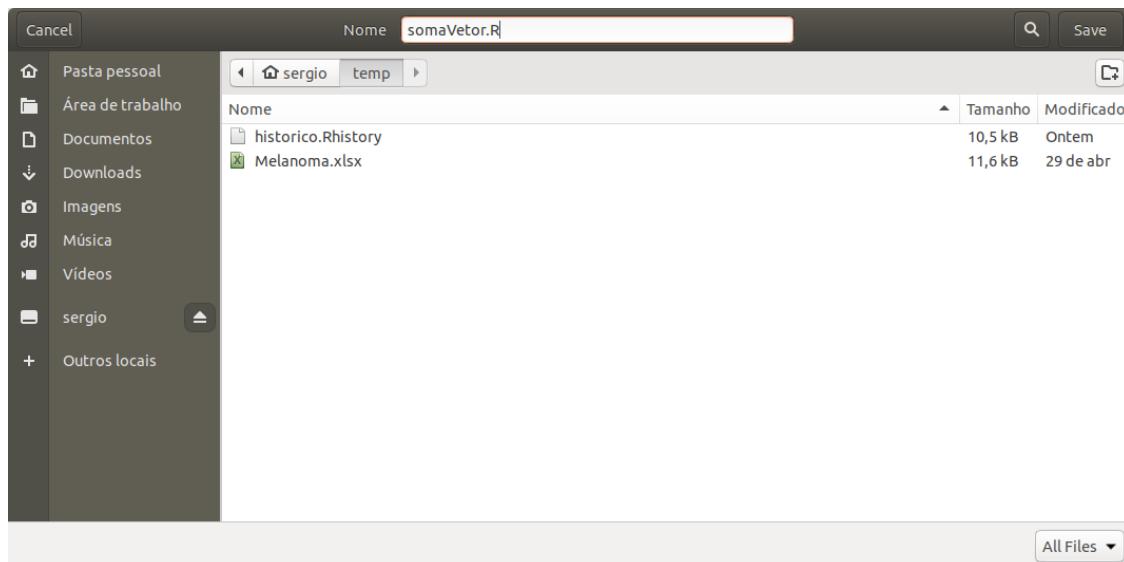


Figura 11.7: Escolhendo o nome e o local do arquivo do conteúdo da área de *script* do *RStudio*.

Ao clicarmos em *Save* na figura 11.7, o arquivo será gravado na pasta especificada. A figura 11.8 mostra a pasta *temp* com o arquivo *somaVetor.R* recém gravado.

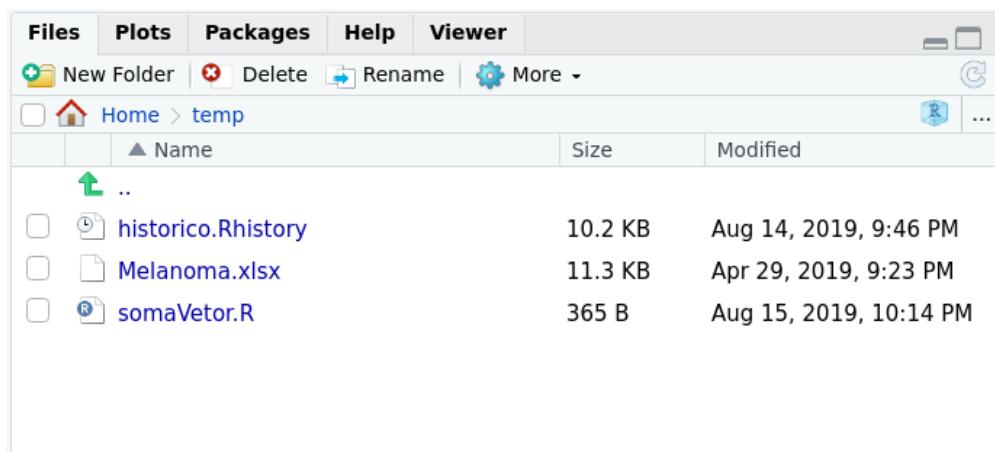


Figura 11.8: Aba *Files* com o arquivo do *script* gravado.

A figura 11.9 mostra novamente a área de edição de arquivos do *RStudio*, agora com o nome do arquivo fornecido anteriormente.

The screenshot shows the RStudio interface with the script file 'somaVetor.R' open. The code defines a function 'soma.vetor' that takes a vector and an optional argument 'na.rm'. It checks if the input is numeric and integer, prints an error message if not, and then sums the elements of the vector, ignoring NA values if 'na.rm' is TRUE. The code is color-coded for readability.

```
1 #  
2 # definindo uma função para realizar a soma de  
3 # elementos de um vetor numérico  
4 #  
5 soma.vetor <- function(vetor, na.rm = TRUE) {  
6   if (!is.numeric(vetor) & !is.integer(vetor)) {  
7     print ("vetor não é numérico")  
8     return()  
9   }  
10  if (na.rm == TRUE) {  
11    vetor_sem_NA = vetor[!is.na(vetor)]  
12  } else if (any(is.na(vetor)) == TRUE) {  
13    return (NA)  
14  }  
15  soma = 0  
16  for (valor in vetor_sem_NA) {  
17    soma = soma + valor  
18  }  
19  soma  
20}  
21
```

Figura 11.9: Área de *script*, mostrando o nome do arquivo do *script*.

Vemos que o arquivo está salvo, mas a função *soma.vetor* ainda não está disponível no ambiente de trabalho. Poderíamos compilá-la, selecionando todo o *script* que a define e clicando no botão *Run*, mas vamos fazer de outra forma. Vamos testar a função *soma* e aprender mais algumas particularidades das funções.

Para compilar a função, vamos marcar a opção *Source On Save* (figura 11.10) e, em seguida, salvamos o arquivo. O arquivo será executado por meio da função *source*. O comando e o resultado da execução do arquivo são mostrados na console do *RStudio* (figura 11.11).

```

1  #
2  # definindo uma função para realizar a soma de
3  # elementos de um vetor numérico
4  #
5  soma.vetor <- function(vetor, na.rm = TRUE) {
6    if (!is.numeric(vetor) & !is.integer(vetor)) {
7      print ("vetor não é numérico")
8      return()
9    }
10   if (na.rm == TRUE) {
11     vetor_sem_NA = vetor[!is.na(vetor)]
12   } else if (any(is.na(vetor)) == TRUE) {
13     return (NA)
14   }
15   soma = 0
16   for (valor in vetor_sem_NA) {
17     soma = soma + valor
18   }
19   soma
20 }
21

```

Figura 11.10: Área de *script* com a opção de executar o *script* ao salvar o arquivo.

```

Console Terminal × Jobs ×
~/temp/
> source('~/temp/somaVetor.R')
>

```

Figura 11.11: Salvando o *script* com a opção *Source on save* selecionada fará com que o *script* seja compilado e o resultado mostrado na console do *RStudio*.

Se houver erros, as mensagens serão exibidas. Nesse exemplo, nenhum resultado é mostrado na console, mas a função *soma.vetor* foi criada (vejam na aba *Environment*) e agora ela pode ser utilizada.

Nos comandos abaixo, executados na console do *RStudio*, criamos um vetor *x* e, em seguida, executamos a função *soma.vetor*, atribuindo *x* ao argumento *vetor* da função.

```

x = c(1:20, NA, NA)
soma.vetor(vetor = x)  # calcula a soma dos elementos de x, excluindo NA

## [1] 210

```

Não foi necessário especificar o valor do argumento *na.rm*, porque, se o mesmo não for especificado, o valor assumido será *TRUE*, conforme a definição da função.

No exemplo abaixo, como desejamos incluir elementos *NA* na soma, então devemos especificar o valor de *na.rm* como *FALSE*. O resultado da soma do vetor *x* obviamente será *NA*, já que a soma de qualquer valor com *NA* será *NA*.

```
soma.vetor(vetor = x, na.rm = FALSE) # calculando a soma não excluindo NA  
## [1] NA
```

11.8 Pareamento dos argumentos

Os conteúdos desta seção e da seção 11.9 podem ser visualizados neste [vídeo](#).

A função *formals(nome_da_funcao)* fornece os parâmetros formais da função. No comando a seguir, obtemos os dois argumentos da função *soma.vetor*, *vetor* e *na.rm*, sendo *TRUE* o valor *default* para *na.rm*.

```
formals(soma.vetor)
```

```
## $vetor  
##  
##  
## $na.rm  
## [1] TRUE
```

Na chamada da função *soma.vetor* abaixo, não especificamos o nome do primeiro argumento; simplesmente passamos o vetor *x*. Não ocorreu erro, porque o R associa uma posição a cada argumento formal de acordo com a definição da função. Assim o primeiro argumento na chamada será atribuído a *vetor*, o segundo argumento a *na.rm*.

```
soma.vetor(x) #
```

```
## [1] 210
```

```
soma.vetor(x, FALSE)
```

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

Podemos misturar na chamada de uma função argumentos de acordo com a posição ou pelo nome.

```
soma.vetor(x, na.rm = FALSE)  
## [1] NA
```

Se um argumento que não possui valor padrão não for especificado na chamada, ocorrerá um erro. Por exemplo, se não especificarmos o valor de vetor na chamada da função *soma.vetor*:

```
soma.vetor()
```

ocorrerá o seguinte erro:

```
Error in soma.vetor() : argumento "vetor" ausente, sem padrão
```

Caso o vetor não seja numérico, a função imprimirá uma mensagem e retornará o valor NULL.

```
y = c("a", 1, "b")
s = soma.vetor(y)
```

```
## [1] "vetor não é numérico"
s
## NULL
```

Voltando à questão da notação posicional para a especificação dos argumentos de uma função, consideremos a função $rnorm(n, mean = 0, sd = 1)$. Essa função gera n números aleatórios de acordo com uma distribuição normal com argumentos formais $mean$ (média) e sd (desvio padrão).

Ao especificarmos os argumentos pelo nome, a ordem dos argumentos é irrelevante. Assim $rnorm(n=10, mean = 20, sd = 4)$ é equivalente a $rnorm(mean = 20, sd = 4, n=10)$, por exemplo.

Quando um argumento é especificado pelo nome, ele é “removido” da lista de argumentos e os argumentos não nomeados restantes são pareados na ordem em que eles são listados na definição da função.

Assim, na chamada da função $rnorm(10, sd = 8, 70)$, sd é removido da lista de argumentos não nomeados, o primeiro valor, 10, corresponde ao valor de n , e o valor 70 corresponde ao argumento $mean$.

Não é recomendável misturar chamadas com argumentos nomeados e de acordo com a posição, pois isso pode gerar confusão e eventualmente a resultados não esperados.

O R também realiza um pareamento parcial dos argumentos. No comando a seguir, a função $rnorm$ foi chamada com os argumentos $me = 20$ e $s = 4$. O R reconheceu que me correspondia ao argumento $mean$ e s ao argumento sd .

```
rnorm(2, me = 20, s = 4)
```

```
## [1] 12.39806 17.50789
```

11.9 O uso do argumento ...

O argumento `...` é usado para indicar que existe um número variável de argumentos que usualmente são passados para outras funções. Ele é frequentemente usado quando uma função estende outra função agregando alguma funcionalidade extra e depois chamando a função estendida. Isso evita que, ao definir uma função, tenhamos que incluir toda a lista de argumentos da função que será estendida.

Vamos ver um exemplo. O código abaixo cria uma amostra aleatória (`y`) com tamanho 2000 de uma distribuição normal padrão e, em seguida, plota o histograma de `y` (figura 11.12).

```
y = rnorm(2000)  
hist(y)
```

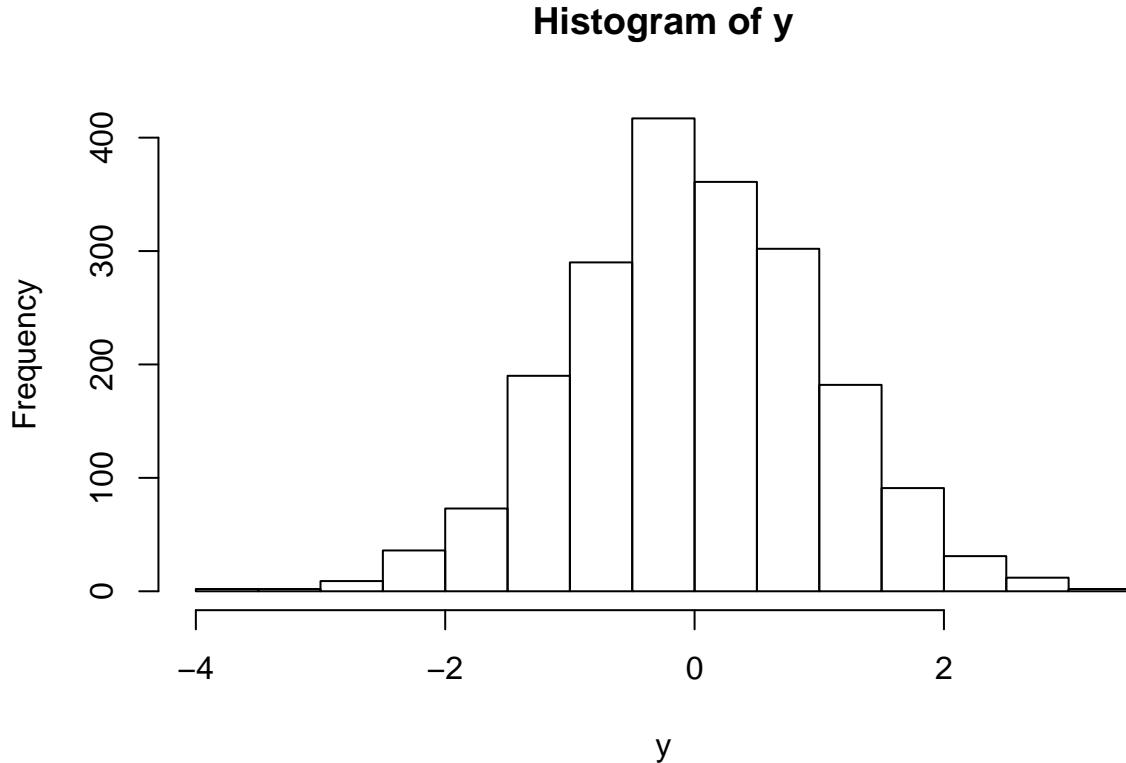


Figura 11.12: Histograma de frequência de uma amostra aleatória com 2000 elementos de uma distribuição normal padrão.

A função `hist` gera por padrão um histograma de frequência. Vide a ajuda dessa função. Para gerar um histograma de densidade de frequência relativa, é necessário especificar o argumento `freq = FALSE`. Vamos definir uma função que gera por padrão o histograma de densidade de frequência relativa. A função `meu_hist` abaixo faz exatamente isso. Ela recebe um vetor (`dados`) e chama a função `hist` com o argumento `freq = FALSE`. Observem o argumento “`...`” na definição da função e na chamada de `hist`. Isso ocorre para evitar de ter que colocar todos os argumentos que são utilizados pela função `hist` e pelas funções que `hist` eventualmente chama.

Ao executarmos a função `meu_hist`, especificamos, além do vetor de dados (`y`), os rótulos dos eixos `x` e `y`. Os argumentos `xlab` e `ylab` serão repassados para a função `hist` (figura 11.13).

```
meu_hist <- function(dados, ...) {  
  hist(dados, freq = FALSE, ...)  
}  
meu_hist(y, xlab = "y", ylab = "Dens. de Freq. Relativa")
```

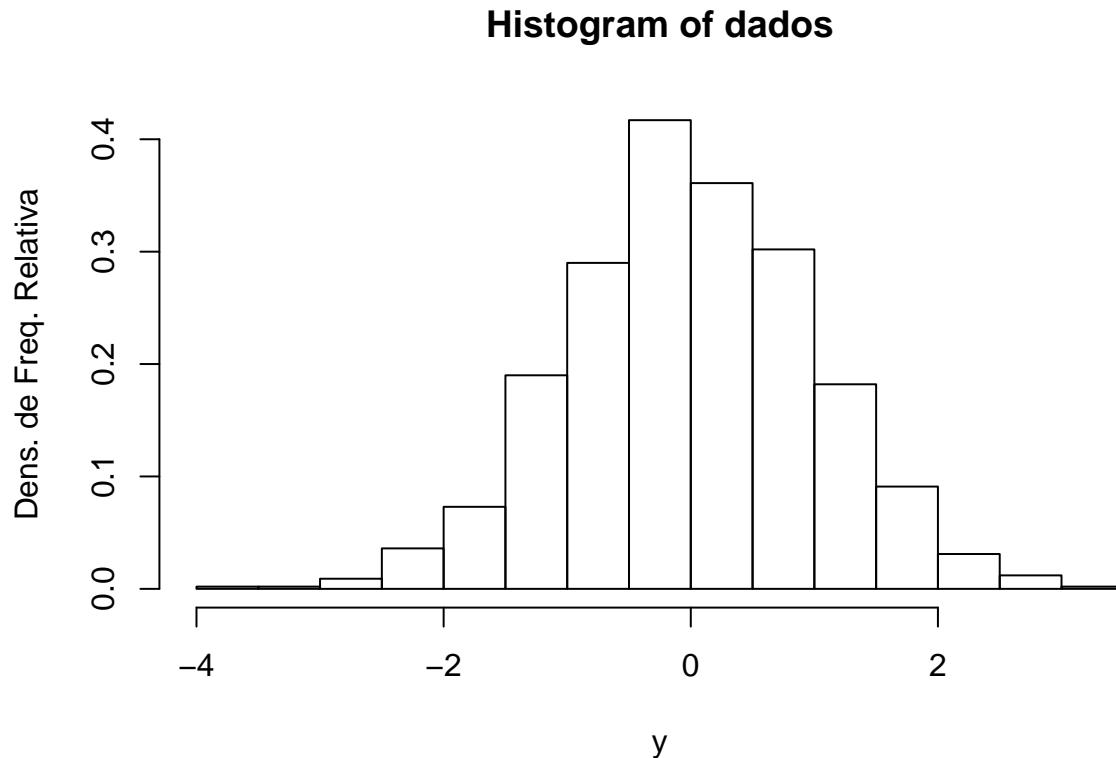


Figura 11.13: Histograma de densidade de frequência relativa de uma amostra aleatória com 2000 elementos de uma distribuição normal padrão.

Funções genéricas usam o argumento “...”, de modo que argumentos extras possam ser repassados para funções chamadas internamente.

A função `str` exibe a estrutura interna de um objeto do R. Se chamarmos `str` com uma função, ela irá exibir os argumentos da função. Abaixo, ela foi executada com o argumento `paste`.

```
str(paste)  
  
## function (..., sep = " ", collapse = NULL)
```

A função `paste` concatena vetores após convertê-los para `character`. Veja uma execução da função `paste` a seguir. O resultado foi uma string formada pela concatenação dos elementos 3, 4, 5 e 6, separados por `.`

```
paste(3, 4, 5, 6)
```

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

Na definição da função *paste*, é preciso inserir o argumento ... no início, porque não sabemos de antemão quantos vetores de caracteres serão passados para a função.

Observação:

Quaisquer argumentos que aparecem depois de ... na lista de argumentos têm que ser nomeados explicitamente e não podem ser parcialmente pareados ou pareados pela posição. Por exemplo, na chamada de *paste* abaixo, foi gerada uma *string* com os elementos separados por “;”.

```
paste(3, 4, 5, 6, sep = "; ") # string com os elementos separados por ";"
```

```
## [1] "3; 4; 5; 6"
```

Se tentarmos um pareamento parcial do argumento *sep*, não obteremos o resultado desejado:

```
paste(3, 4, 5, 6, se = "; ")
```

```
## [1] "3 4 5 6 ; "
```

Nesse caso, a função *paste* não reconhece “se” como o argumento *sep*. O “;” será tratado como mais um elemento a ser concatenado, e os elementos são separados por um espaço, o separador padrão.

Referências Bibliográficas

- Andersen, P. K., Borgan, O., Gill, R. D., and Keiding, N. (1993). *Statistical Models based on Counting Processes*. Springer.
- Barata, C. B. and Valete, C. O. S. (2018). Perfil clínico-epidemiológico de 106 pacientes pediátricos portadores de urolitíase no Rio de Janeiro. *Revista Paulista de Pediatria*, 36(3):261–267.
- Bezerra, S. M. F. M. C., Sotto, M. N., Orii, N. M., Alves, C., and Duarte, A. J. S. (2011). Efeitos da radiação solar crônica prolongada sobre o sistema imunológico de pescadores profissionais em Recife, Brasil. *Anais Brasileiros de Dermatologia*, 86(2):222–233.
- Dalgaard, P. (2008). *Introductory Statistics with R*. Springer.
- Durmus, E., Kivrak, T., Gerin, F., Sunbul, M., Sari, I., and Erdogan, O. (2015). Relações Neutrófilo-Linfócito e Plaqueta-Linfócito Como Preditores de Insuficiência Cardíaca. *Arquivos Brasileiros de Cardiologia*, 105(6):606–613.
- Furuta, S. E., Weckx, L. M., and Figueiredo, C. R. (2003). Estudo clínico, randomizado, duplo-cego, em crianças com adenóide obstrutiva, submetidas a tratamento homeopático. *Revista Brasileira de Otorrinolaringologia*, 69(3 parte 1):343–347.
- Melo, F. R. R. (2017). Introdução à programação com a linguagem R.
- Melo, F. R. R. (2019a). Introdução ao R Commander.
- Melo, F. R. R. (2019b). R Commander: um pouco além dos menus gráficos.
- Peng, R. D. (2016a). *Exploratory Data Analysis with R*. Leanpub.
- Peng, R. D. (2016b). *R Programming for Data Science*. Leanpub.
- Peng, R. D. (2016c). *Report Writing for Data Science*. Leanpub.
- Queiroz, C. F., Lemos, A. C. M., Bastos, M. d. L. S., Neves, M. C. L. C., Camelier, A. A., Carvalho, N. B., and Carvalho, E. M. (106). Perfil inflamatório e imunológico em pacientes com DPOC: relação com a reversibilidade do VEF 1. *Jornal Brasileiro de Pneumologia*, 42(4):241–247.
- Silva, P. V., Salman, A. A., Cristóvão, S. A. B., Carnieto, N. M., Erudilho, E., Mauro, M. F. Z., Ticly, M. C., Dutra, G., Giordano, B., and Mangione, J. A. (2014). Impacto do Escore

- SYNTAX no Prognóstico de Pacientes com Doença Multiarterial Tratados por Intervenção Coronária Percutânea. *Revista Brasileira de Cardiologia Invasiva*, 22(3):258–263.
- Vanin, L. K., Zatti, H., Soncini, T., Nunes, R. D., and Siqueira, L. B. S. (2019). Fatores de risco materno-fetais associados à prematuridade tardia. *Revista Paulista de Pediatria*, 38(e2018136).