

Uma Breve Introdução a R

Princípios Básicos

Anaih P. Pereira e Arthur Rocha



Copyright © 2017 Rocha, Pastana

PUBLICADO POR —

[HTTPS://SITES.GOOGLE.COM/SITE/CURSODERBSE/](https://sites.google.com/site/cursoderbse/)

Licenciado sobre liberdade criativa e atribuição comercial. Este arquivo só poderá ser utilizado e distribuído mediante sua licença.

Primeira impressão, Março 2018



I	Capítulo 1	
1	Introdução	8
II	Capítulo 2	
2	Download e Instalação	10
2.1	R Base	10
2.2	R Studio	13
III	Capítulo 3	
3	Princípios básicos	17
3.1	Primeiros passos	17
3.1.1	Comandos básicos	17
3.1.2	Atribuição de valores	18
3.1.3	Tipos de variáveis	18
3.1.4	Utilizando ajuda (HELP)	19
3.2	Operações básicas	19
3.3	Estruturas básicas	20
3.3.1	Vetor	20
3.3.2	Matriz	24
3.3.3	Array	26

3.3.4	Lista	27
3.3.5	Data frame	28
3.4	Tabelas	32
3.4.1	Tabelas simples	32
3.4.2	Tabelas de contingência	32
3.4.3	Tabelas de proporção	32
3.5	Funções	33
3.6	Funções Apply	34
3.6.1	Apply	34
3.6.2	Tapply	35
3.6.3	Sapply	35
3.6.4	Lapply	36
3.7	Entrada de dados	36
3.7.1	Csv	37
3.7.2	Txt	37
3.7.3	Xls ou Xlsx	37
3.7.4	Base de dados do R	37
3.7.5	Opções importantes	38
3.7.6	Conferência dos Dados	38
3.7.7	Funções adicionais	39
3.8	Pacotes	40

IV

Capítulo 4

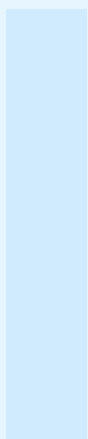
4	Estatística básica	42
4.1	Medidas de Posição	42
4.1.1	Média	42
4.1.2	Mediana	42
4.1.3	Mínimo e máximo	43
4.1.4	Quantis	43
4.1.5	Moda	44
4.1.6	Função Summary	44
4.2	Medidas de Dispersão	44
4.2.1	Variância	44
4.2.2	Desvio padrão	45
4.2.3	Coeficiente de variação	45
4.2.4	Amplitude	45
4.3	Correlação	46
4.3.1	Coeficiente de Pearson	46
4.3.2	Coeficiente de Kendall	46
4.3.3	Coeficiente de Spearman	46

5	Gráficos	48
5.1	Gráficos para variáveis qualitativas	48
5.1.1	Gráfico de Barras	48
5.1.2	Gráfico de Setores	49
5.2	Gráficos para variáveis quantitativas	51
5.2.1	Histograma	51
5.2.2	Polígono de frequência	51
5.2.3	Gráfico de bastões (hastes)	52
5.2.4	Gráfico de Dispersão	53
5.2.5	Gráfico de Caixas (Box-plot)	54
5.3	Ajustes Gráficos	55
5.3.1	Principais ajustes	55
5.3.2	Funções de sobreposição	56
5.3.3	Argumento type	58
5.3.4	Tipos de símbolos e linhas	59
5.3.5	Texto e legendas	61
5.3.6	Utilizando par() e alguns comandos adicionais	65
5.3.7	Cores	71
5.3.8	Janelas gráficas externas:	76
5.3.9	Representando tabelas de contingência	76
5.3.10	Representando funções	80
5.3.11	Exportando gráficos	81

6	Noções de probabilidade	84
6.1	Amostragem	84
6.2	Análise Combinatória	85
6.2.1	Permutação	85
6.2.2	Arranjo	85
6.2.3	Combinação	85
6.3	Distribuições de probabilidade	86
6.3.1	Geração de valores aleatórios	86
6.4	Função de densidade / probabilidade	87
6.5	Função acumulada	87
6.6	Quantis	88

7	Exercícios	90
7.1	Operações Básicas	90
7.2	Estruturas Básicas	90

7.3	Entrada de dados	91
7.4	Funções e gráficos	91



Capítulo 1



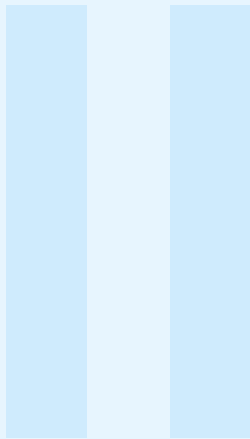
1. Introdução

R é um ambiente de desenvolvimento e linguagem interpretada, gratuito e de código aberto. É amplamente utilizado na análise e manipulação de dados e operações estatísticas. Seus criadores foram Ross Ihaka e Robert Gentleman, que criaram essa ferramenta em 1993 na Nova Zelândia, mais precisamente na Universidade de Auckland.

A estatística é a área do conhecimento que extrai de dados, informações pertinentes a tomada de decisão e conhecimento de uma forma geral. Desta forma, torna-se uma área muito ampla e aplicável, pois a melhoria de processos, entendimento de fenômenos e assertividade na tomada de decisões é algo de interesse para todas as áreas do conhecimento.

Segundo uma pesquisa realizada em 2016, conduzida por Gregory Piatetsky do site KDnuggets (disponível em <https://www.kdnuggets.com/2016/06/r-python-top-analytics-data-mining-data-science-software.html/2>), o R foi a ferramenta mais utilizada para análise pelos cientistas de dados que contribuíram para o estudo. Desta forma, torna-se imprescindível o conhecimento dessa linguagem no caso de um estatístico ou cientista de dados que queira estar atualizado nesse mundo, pois existe uma grande comunidade que trabalha com essa ferramenta e ajuda no seu desenvolvimento, já que seu código é aberto. Justamente pelo fato de ser *opensource* tem-se outra vantagem nessa ferramenta, pois ela se torna altamente customizável, praticamente não tendo barreiras para o seu desenvolvimento.

Esta apostila foi criada a partir de uma necessidade de um material didático fácil e com aplicações palpáveis para estudantes iniciantes no curso de estatística. Foi idealizada enquanto ministrávamos um curso de R básico para esse público na Universidade Estadual de Maringá. Trata-se de um material introdutório que aborda os conceitos básicos da linguagem e pode ser usado como um apoio à disciplina de Estatística Computacional.



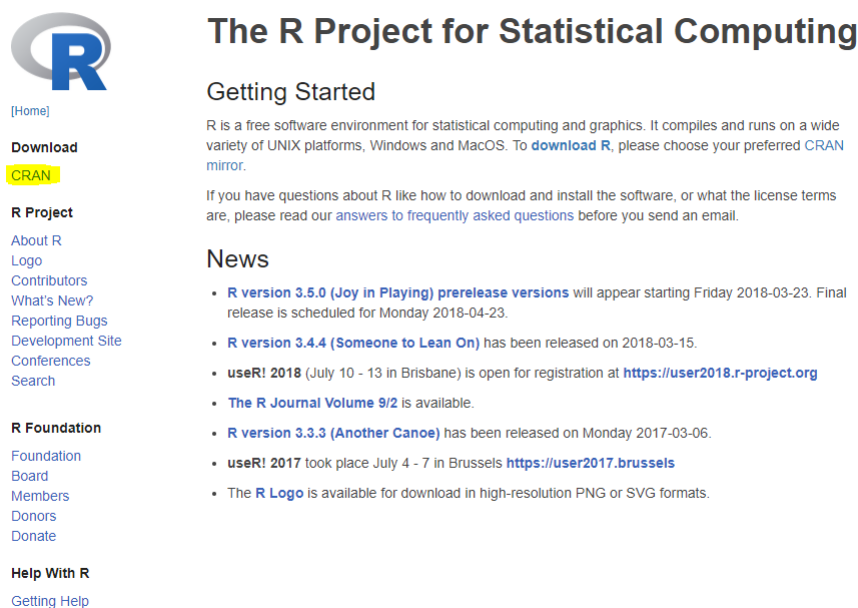
Capítulo 2

2	Download e Instalação	10
2.1	R Base	
2.2	R Studio	

2. Download e Instalação

2.1 R Base

Para conseguirmos utilizar o R é preciso instalar primeiro o interpretador da linguagem R, denominado R base. O download do interpretador pode ser feito pelo endereço <https://www.r-project.org/>. Ao se acessar a página, na aba **CRAN** deve-se escolher o local mais próximo a sua localização atual. Feito isso, deve-se escolher qual sistema operacional do computador e em seguida clicar no link **Base**. Os passos para o download serão ilustrados pelas figuras abaixo:



The R Project for Statistical Computing

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To [download R](#), please choose your preferred [CRAN mirror](#).

If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

News

- **R version 3.5.0 (Joy in Playing) prerelease versions** will appear starting Friday 2018-03-23. Final release is scheduled for Monday 2018-04-23.
- **R version 3.4.4 (Someone to Lean On)** has been released on 2018-03-15.
- **useR! 2018** (July 10 - 13 in Brisbane) is open for registration at <https://user2018.r-project.org>
- **The R Journal Volume 9/2** is available.
- **R version 3.3.3 (Another Canoe)** has been released on Monday 2017-03-06.
- **useR! 2017** took place July 4 - 7 in Brussels <https://user2017.brussels>
- The **R Logo** is available for download in high-resolution PNG or SVG formats.

R Project

- [About R](#)
- [Logo](#)
- [Contributors](#)
- [What's New?](#)
- [Reporting Bugs](#)
- [Development Site](#)
- [Conferences](#)
- [Search](#)

R Foundation

- [Foundation](#)
- [Board](#)
- [Members](#)
- [Donors](#)
- [Donate](#)


Help With R

- [Getting Help](#)

Figura 2.1: Página inicial R-project

http://cran.usthb.dz/	University of Science and Technology Houari Boumediene
Argentina	
http://mirror.fcaglp.unlp.edu.ar/CRAN/	Universidad Nacional de La Plata
Australia	
https://cran.csiro.au/	CSIRO
http://cran.csiro.au/	CSIRO
https://mirror.aarnet.edu.au/pub/CRAN/	AARNET
https://cran.ms.unimelb.edu.au/	School of Mathematics and Statistics, University of Melbourne
https://cran.curtin.edu.au/	Curtin University of Technology
Austria	
https://cran.vu.ac.at/	Wirtschaftsuniversität Wien
http://cran.vu.ac.at/	Wirtschaftsuniversität Wien
Belgium	
http://www.freeststatistics.org/cran/	K.U.Leuven Association
https://lib.ugent.be/CRAN/	Ghent University Library
http://lib.ugent.be/CRAN/	Ghent University Library
Brazil	
http://nbogib.uesc.br/mirrors/cran/	Center for Comp. Biol. at Universidade Estadual de Santa Cruz
https://cran-r.c3sl.ufpr.br/	Universidade Federal do Parana
http://cran-r.c3sl.ufpr.br/	Universidade Federal do Parana
https://cran.fiocruz.br/	Oswaldo Cruz Foundation, Rio de Janeiro
http://cran.fiocruz.br/	Oswaldo Cruz Foundation, Rio de Janeiro
https://vps.fmvz.usp.br/CRAN/	University of Sao Paulo, Sao Paulo
http://vps.fmvz.usp.br/CRAN/	University of Sao Paulo, Sao Paulo
https://brieger.esalq.usp.br/CRAN/	University of Sao Paulo, Piracicaba
http://brieger.esalq.usp.br/CRAN/	University of Sao Paulo, Piracicaba
Bulgaria	
https://ftp.uni-sofia.bg/CRAN/	Sofia University
http://ftp.uni-sofia.bg/CRAN/	Sofia University
Canada	

Figura 2.2: Página de CRAN’s disponíveis



[CRAN](#)
[Mirrors](#)
[What's new?](#)
[Task Views](#)
[Search](#)

[About R](#)
[R Homepage](#)
[The R Journal](#)

[Software](#)
[R Sources](#)
[R Binaries](#)
[Packages](#)
[Other](#)

[Documentation](#)
[Manuals](#)
[FAQs](#)
[Contributed](#)

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2018-03-15, Someone to Lean On) [R-3.4.4.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).
- Contributed extension [packages](#)

Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

Figura 2.3: Página de download conforme sistema operacional



CRAN
[Mirrors](#)
[What's new?](#)
[Task Views](#)
[Search](#)

About R
[R Homepage](#)
[The R Journal](#)

Software
[R Sources](#)
[R Binaries](#)
[Packages](#)
[Other](#)

Documentation
[Manuals](#)
[FAQs](#)
[Contributed](#)

R for Windows

Subdirectories:

base

[contrib](#)

[old contrib](#)

[Rtools](#)

Binaries for base distribution. This is what you want to [install R for the first time](#).

Binaries of contributed CRAN packages (for R \geq 2.13.x; managed by Uwe Ligges). There is also information on [third party software](#) available for CRAN Windows services and corresponding environment and make variables.

Binaries of contributed CRAN packages for outdated versions of R (for R $<$ 2.13.x; managed by Uwe Ligges).

Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself.

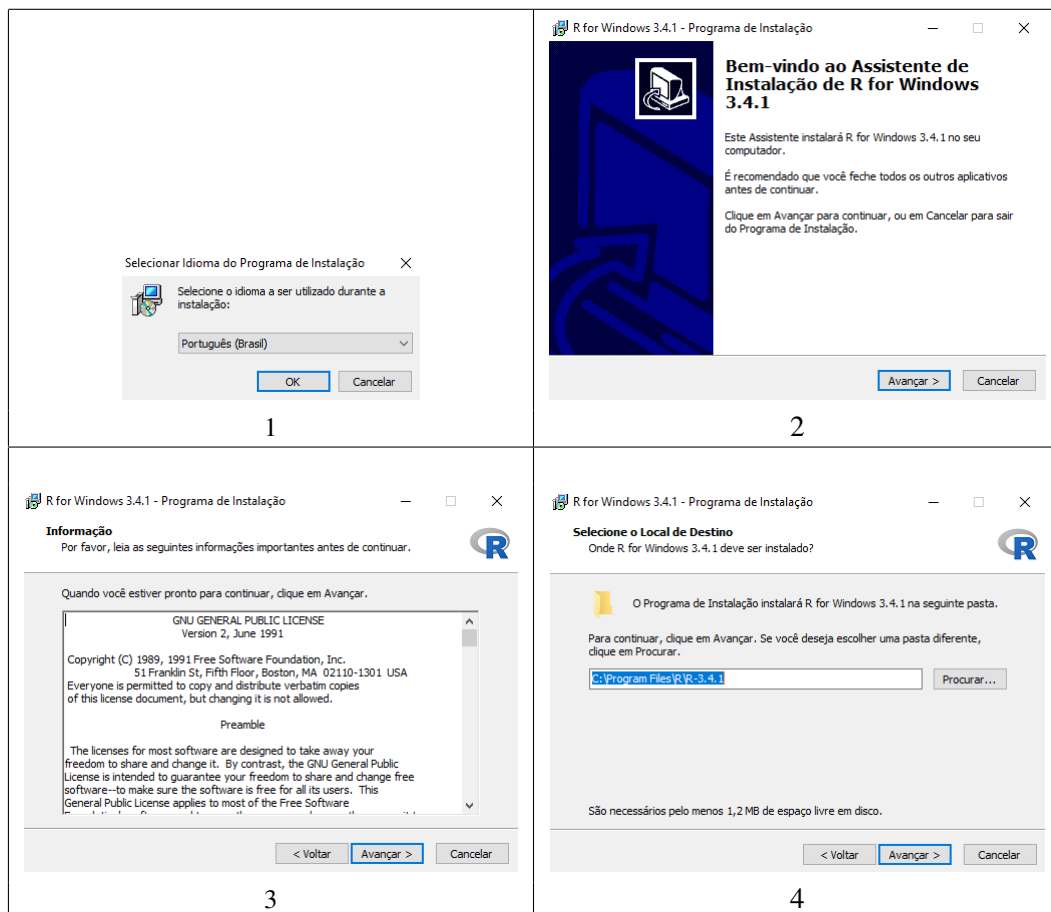
Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

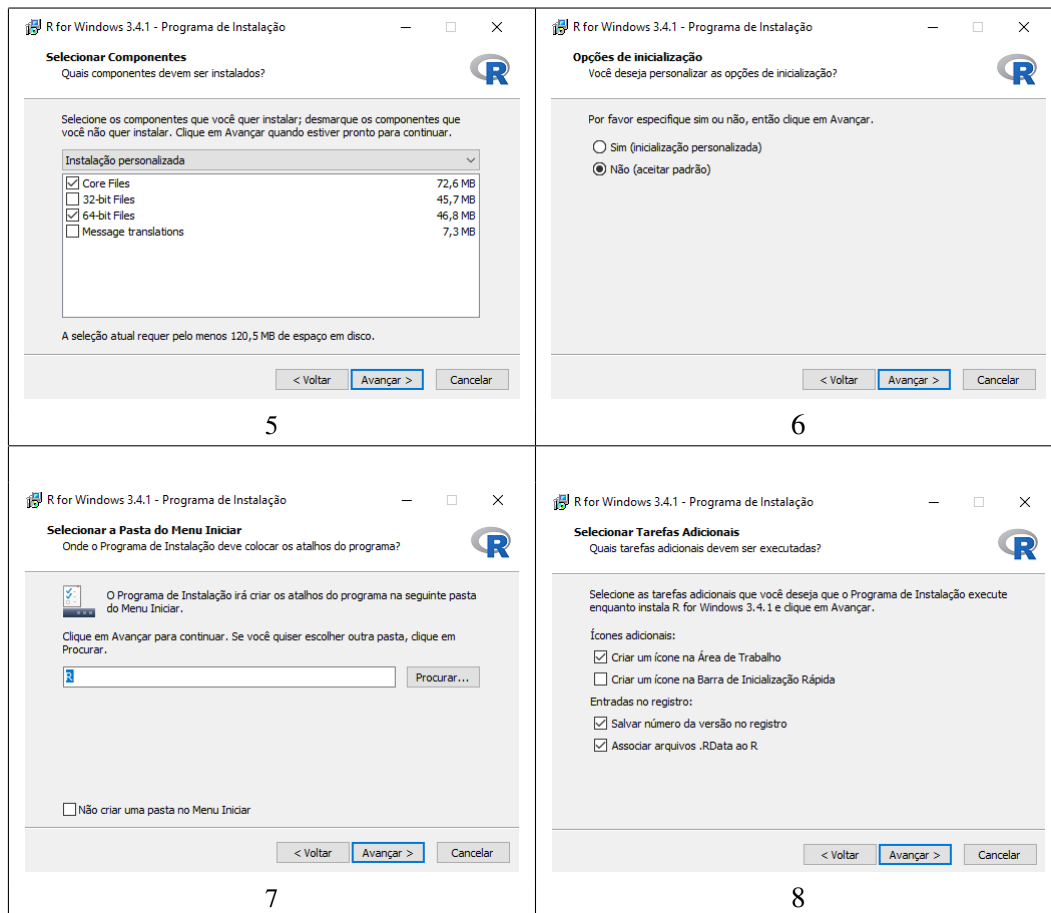
You may also want to read the [R FAQ](#) and [R for Windows FAQ](#).

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

Figura 2.4: Página de download do R Base

Prosseguindo, para efetuar a instalação basta abrir o executável instalador e escolher as opções que forem de interesse, indicamos deixar o *default*. Esse processo é ilustrado pelas figuras a seguir.





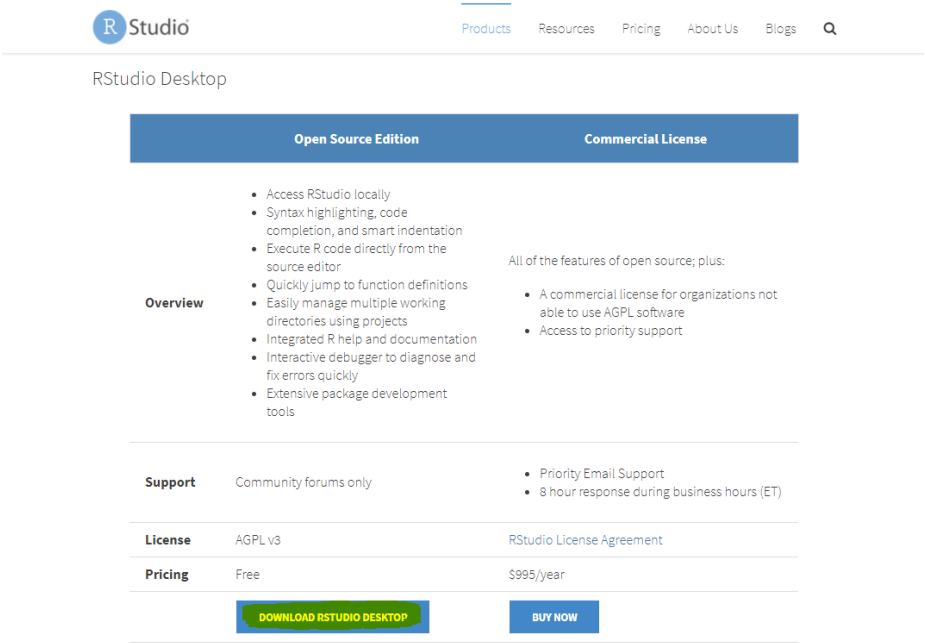


Figura 2.6: Página de download do R Studio

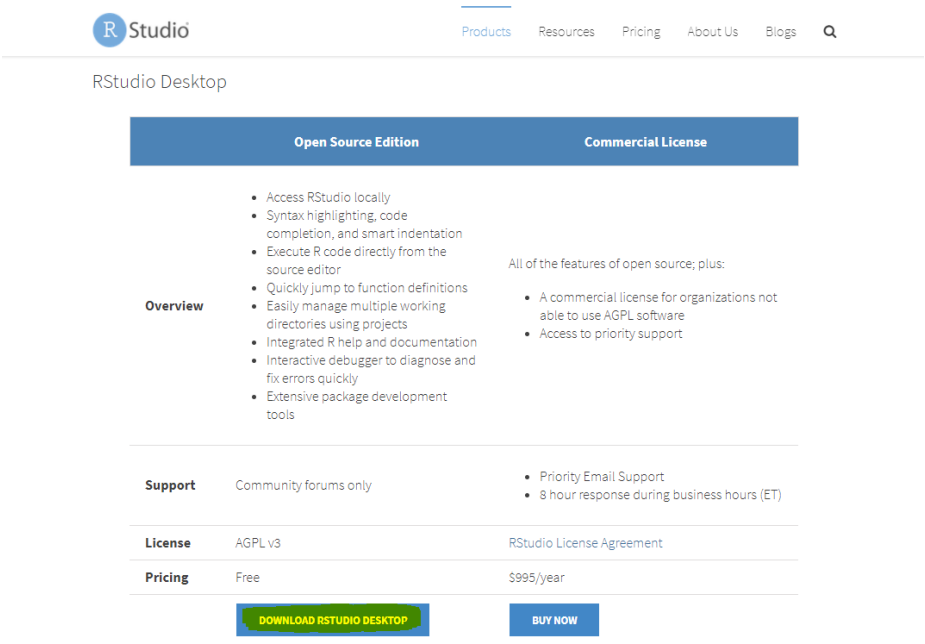


Figura 2.7: Página de escolha do instalador do R Studio conforme sistema operacional

Dando continuidade, para se realizar a instalação, novamente basta abrir o executável instalador e avançar nas páginas. Mais uma vez recomendamos deixar as opções padrão. O processo é ilustrado abaixo:

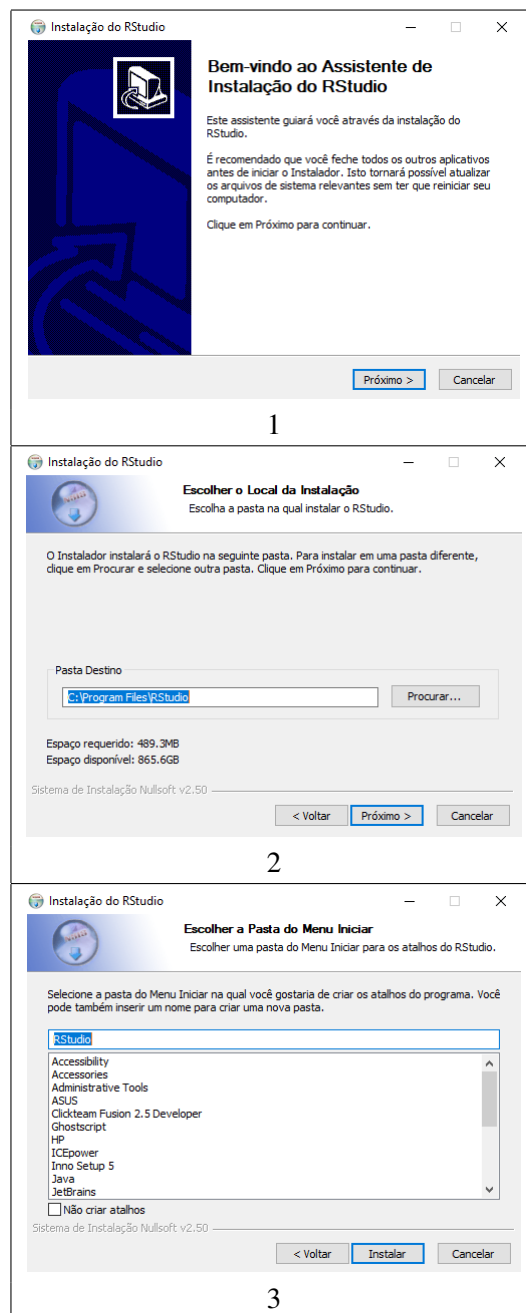



Figura 2.8: Passos para a instalação do R Studio



Capítulo 3

3	Princípios básicos	17
3.1	Primeiros passos	
3.2	Operações básicas	
3.3	Estruturas básicas	
3.4	Tabelas	
3.5	Funções	
3.6	Funções Apply	
3.7	Entrada de dados	
3.8	Pacotes	



3. Princípios básicos

3.1 Primeiros passos

R é uma linguagem orientada à objetos que são armazenados na memória ativa do computador. Uma variável é um objeto que irá representar um valor ou expressão atribuído a ela. Só é possível armazenar um dado ou expressão pra cada variável, quando for atribuído mais de uma informação, o dado que estava antes armazenado será substituído.

3.1.1 Comandos básicos

O R possui alguns comandos muito usuais, que serão utilizados em praticamente todas as sessões iniciadas nele. Esses comandos são:

control + I: Utilizado pra limpar o console

control + R ou control + enter: Utilizado para compilar o código escrito

rm(list=ls()): Utilizado para apagar todos os "objetos" da memória

: Utilizado para fazer comentários no código

O R também pode ser usado como uma calculadora para operações básicas, como:

```
1 2+2
2 3+2
3 10/2
```

```
4 4*2
```

Que irá nos retornar:

```
> 2+2
[1] 4
> 3+2
[1] 5
> 10/2
[1] 5
> 4*2
[1] 8
```

3.1.2 Atribuição de valores

O comando de atribuição de valores é o que define o que será armazenado a uma variável. Para atribuir valores à variáveis utilizamos:

<- ou =

Obs: O nome dado ao objeto não pode começar com números ou ponto seguido de número.

Exemplo 1: atribuindo os valores 10 e 5 nas variáveis X e Y respectivamente:

```
1 X <- 10
2 Y = 5
```

Neste exemplo o R retornará os valores 10 e 5:

```
> X
[1] 10
> Y
[1] 5
```

3.1.3 Tipos de variáveis

No R temos alguns tipo de variáveis como, numérica, caracteres, lógicas, números complexos.

Exemplo 2: Variável de caracteres.

Obs: Sempre colocar o character/string entre aspas simples ou duplas.

```
1 X <- "a"
```

No exemplo 2, o R retornará:

```
> X
[1] "a"
```

Exemplo 3: Variável lógica.

```
1 X <- 6>5
```

Neste exemplo, o R nos retornará se a informação que atribuímos a variável é verdadeira ou falsa, neste caso retornará TRUE.

```
> X
[1] TRUE
```

3.1.4 Utilizando ajuda (HELP)

Para buscar ajuda no R, temos uma função, um operador e um atalho:

help()

?

F1

Exemplo 4: Buscando ajuda para a função lm(modelo linear):

```
1 help(lm)
2 ?lm
```

que irá nos retornar:

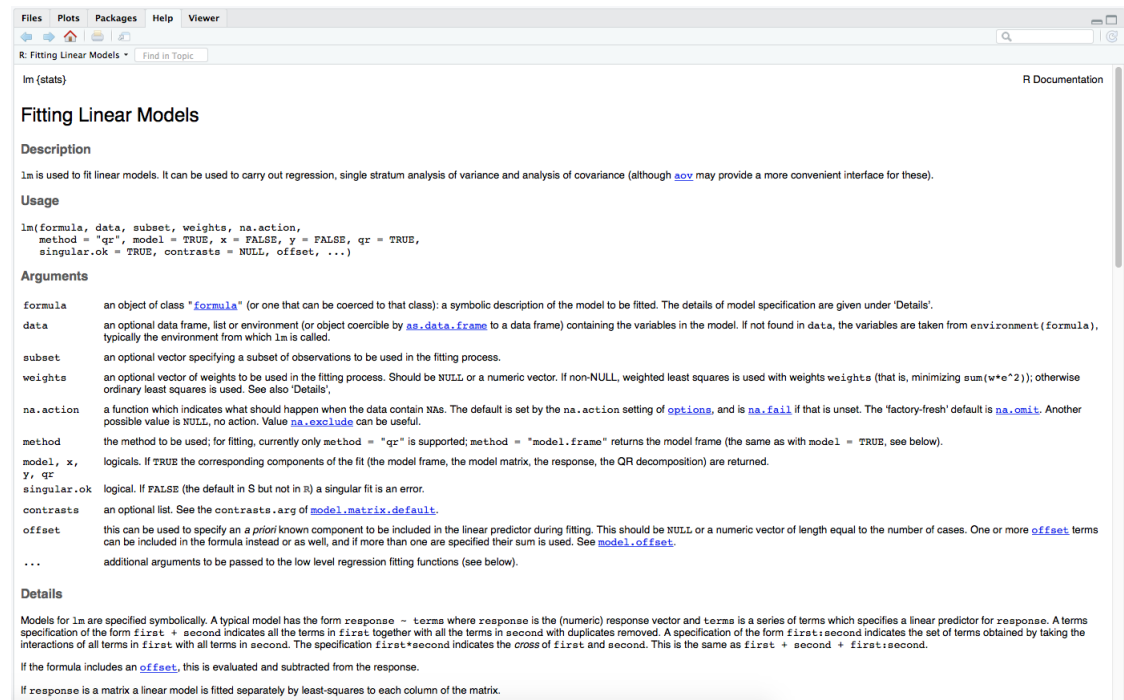


Figura 3.1: Help

3.2 Operações básicas

No ambiente R, existem uma série de operações básicas que são muito usuais e de grande importância. Essas operações são descritas pelas tabelas abaixo:

Tabela 3.1: Operações matemáticas simples

^	Potenciação
/	Divisão à direita
*	Multiplicação
+	Adição
-	Subtração

Tabela 3.2: operadores relacionais e operadores lógicos:

Símbolo	Descrição
<	Menor
<=	Menor ou igual
>	Maior
>=	Maior ou igual
==	Igual(comparação)
!=	Diferente
&	AND
!	NOT
	OR
FALSE ou 0	Valor booleano falso (0)
TRUE ou 1	Valor booleano verdadeiro (1)

Tabela 3.3: Outras funções matemáticas

Função	Descrição
abs(x)	valor absoluto de x
log(x,b)	logaritmo de x com base b
log(x)	logaritmo natural de x
log10(x)	logaritmo de x com base 10
exp(x)	exponencial elevado a x
sin(x)	seno de x
cos(x)	cosseno de x
tan(x)	tangente de x
round(x, digits=n)	arredonda x com n decimais
ceiling(x)	arredonda x para o maior valor
floor(x)	arredonda x para o menor valor
sqrt(x)	raiz quadrada de x

3.3 Estruturas básicas

3.3.1 Vetor

Podemos ter vetores que contém um só elemento, como as atribuições vistas anteriormente ou vetores com mais de um elemento, para isso utilizaremos o comando:

c()

Exemplo 5: Criando alguns tipo de vetores:

```

1 vetor1 <- c(1, 4, 10.5, 54.48, 9, 10)
2 vetor2 <- (1:10)
3 vetor3 <- c((1:3), (3:1))
4 vetor4 <- c(0, vetor3, 0)
```

O R retornará:

```
> vetor1
[1] 1.00 4.00 10.50 54.48 9.00 10.00
> vetor2
[1] 1 2 3 4 5 6 7 8 9 10
> vetor3
[1] 1 2 3 3 2 1
> vetor4
[1] 0 1 2 3 3 2 1 0
```

Sequências

Criar uma sequência é uma outra forma de fazer um vetor, para isso vamos utilizar as estruturas:

- Para uma vetor de A a Z:

```
seq(from= A, to= Z)
```

- Para um vetor de A a Z com N passos:

```
seq(from= A, to= Z, by= N)
```

- Para um vetor de A a Z com N elementos:

```
seq(from= A, to= Z, length.out=N)
```

Exemplo 6: Criar um vetor de 1 a 5:

```
1 seq(from=1, to=5)
```

Que irá nos retornar:

```
> seq(from=1, to=5)
[1] 1 2 3 4 5
```

Exemplo 7: Criar um vetor de 1 a 5 com 0.5 passos:

```
1 seq(from=1, to=5, by=0.5)
```

Que irá nos retornar:

```
> seq(from=1, to=5, by=0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Exemplo 8: Criar um vetor de 0 a 10 com 4 elementos:

```
1 seq(from=0, to=10, length.out= 4)
```

Que irá nos retornar:

```
> seq(from=0, to=10, length.out= 4)
[1] 0.000000 3.333333 6.666667 10.000000
```

Operações em vetores

É possível aplicar uma série de operações nos vetores, a tabela abaixo descreve as operações:

Tabela 3.4: Operações para vetores

Função	Descrição
length(x)	numero de elementos no vetor x
sum(x)	soma dos elementos do vetor x
prod(x)	produto dos elementos do vetor x
max(x)	seleciona o maior elemento do vetor x
min(x)	seleciona o menor elemento do vetor x
range(x)	retorna o menor e o maior elemento do vetor x

Exemplo

```
1 x=1:10
2 sum(x)
3 length(x)
```

```
> sum(x)
[1] 55
> length(x)
[1] 10
```

Criando vetor usando o comando paste

É possível criar vetores de character "colando" partes. Utilizamos a sintaxe:

```
paste("prefixo",vetor numérico, sep="separador")
```

```
paste(vetor numérico,"sufixo", sep="separador")
```

Exemplo

```
1 f<-c(paste("a",1:5, sep="")) # x é um vetor com prefixo "a"
2 y<-c(paste0("b",1:5)) # y é um vetor com prefixo "b"
3 f
4 y
```

```
> f
[1] "a1" "a2" "a3" "a4" "a5"
> y
[1] "b1" "b2" "b3" "b4" "b5"
```

Exemplo

```
1 z<-c(paste(1:5,"c", sep="")) # z é um vetor com sufixo "c"
2 z
```

```
> z
[1] "1c" "2c" "3c" "4c" "5c"
```

Repetições

Para se criar um vetor de repetições usamos a sintaxe:

```
x <- rep("dados",...)
```

É possível repetir tanto números quanto vetores. Quando estamos repetindo um número, no campo "...", informamos com a opção `times` o número de vezes que o número será repetido. Em contrapartida, quando estamos repetindo um vetor, podemos informar a mesma opção `times` que irá definir a quantidade de vezes que o vetor inteiro será repetido, mas também é possível informar as opções `each`, que define a quantidade de vezes que cada elemento do vetor será repetido, e `length.out` que informará o tamanho de saída do vetor de repetições.

Exemplo

Repetindo um número 15 vezes:

```
1 rep(x=1, times=15)
```

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

Exemplo

Repetindo um vetor inteiro 3 vezes.

```
1 x=c("a","b","c")
2 rep(x, times=3)
```

```
[1] "a" "b" "c" "a" "b" "c" "a" "b" "c"
```

Exemplo

Repetindo cada elemento do vetor 3 vezes.

```
1 x=c("a","b","c")
2 rep(x, each=3)
```

```
[1] "a" "a" "a" "b" "b" "b" "c" "c" "c"
```

Exemplo

Criando um vetor de repetição de tamanho 8.

```
1 x=c("a","b","c")
2 rep(x, length.out=8)
```

```
[1] "a" "b" "c" "a" "b" "c" "a" "b"
```

Selecionar elemento do vetor

Para acessar os elementos de vetores utilizamos a sintaxe abaixo, em que *i* representa a posição do elemento desejado no vetor:

```
elemento <- vetor[i]
```

Exemplo: Acessando o 5º elemento do vetor *v*.

```
1 v=11:20
2 v[5]
```

```
> v[5]
[1] 15
```

3.3.2 Matriz

Uma matriz é uma generalização de um vetor, tendo duas dimensões. Podemos pensar em um vetor como uma matriz com uma de suas dimensões igual a 1.

A sintaxe é dada abaixo, em que "L" é o número de linhas, "C" é o número de colunas e se "Q" = 1 ativa disposição por linhas, se "Q" = 0 mantém disposição por colunas (ou T ou F).

$$x <- \text{matrix}(\text{data} = \text{dados}, \text{nrow} = L, \text{ncol} = C, \text{byrow} = Q)$$

Exemplo 9: Criando uma matriz de 2 linhas, 5 colunas e disposição por linhas:

```
1 ml <- matrix( data=c(1:10), nrow=2, ncol=5, byrow=1)
```

O R retornará:

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

Exemplo 10: Criando uma matriz de 2 linhas, 5 colunas e disposição por colunas:

```
1 mc <- matrix( data=c(1:10), nrow=2, ncol=5, byrow=0)
```

O R retornará:

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

Selecionar um elemento da matriz

Para selecionar um elemento de uma matriz utilizamos a indexação por colchetes na variável que representa a matriz com os índices separados por vírgula.

Sintaxe sendo matriz o nome da matriz:

$$\text{matriz}[\text{linha}, \text{coluna}]$$

Utilizando as matrizes do exercício anterior:

$$ml = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \end{bmatrix}$$

$$mc = \begin{bmatrix} 1 & 3 & 5 & 7 & 9 \\ 2 & 4 & 6 & 8 & 10 \end{bmatrix}$$

Exemplo 11: Selecionando o elemento da linha 2 e coluna 4 da matriz ml:

```
1 ml[2,4]
```

O R retornará:


```
> ml[2,4]
[1] 9
```

Exemplo 12: Seleccionando o elemento da linha 2 e coluna 4 da matriz ml e subtraindo o elemento da linha 1 e coluna 5 da matriz mc:

```
1 ml[2,4] - mc[1,5]
```

O R retornará:

```
> ml[2,4] - mc[1,5]
[1] 0
```

Exemplo 13: Seleccionando a linha 2 da matriz ml:

```
1 ml[2,]
```

O R retornará:

```
> ml[2,]
[1] 6 7 8 9 10
```

Exemplo 14: Seleccionando as colunas 2,3 4 da matriz ml:

```
1 ml[,2:4]
```

O R retornará:

```
> ml[,2:4]
      [,1] [,2] [,3]
[1,]     2     3     4
[2,]     7     8     9
```

Exemplo 15: Outra forma de ler a matriz ml:

```
1 ml[,]
```

O R retornará:

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

Operações com matrizes

Utilizando a matriz dos exemplos anteriores:

$$ml = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \end{bmatrix}$$

Exemplo 16: Fazendo a transposta da matriz ml:

Tabela 3.5: Operações com matrizes

Função	Descrição
A*B	produto elemento a elemento de A e B
A%*%B	produto matricial de A por B
apern(A)	matriz transposta de A
t(A)	matriz transposta de A
solve(A)	matriz inversa de A
solve(A,B)	resolve o sistema linear $Ax=B$
det(A)	retorna o determinante de A
diag(v)	retorna uma matriz diagonal onde o vetor v é a diagonal
diag(A)	retorna um vetor que é a diagonal de A
diag(n)	sendo n um inteiro, retorna uma matriz identidade de ordem n
eigen(A)	retorna os autovalores e autovetores de A

```
1 t (m1)
```

O R retornara:

```
> t(m1)
      [,1] [,2]
[1,]     1     6
[2,]     2     7
[3,]     3     8
[4,]     4     9
[5,]     5    10
```

3.3.3 Array

Um array é uma generalização de uma matriz, em que os dados podem ser distribuídos em n dimensões de tamanhos $t_i, i \in \{1, 2, \dots, n\}$. A sintaxe utilizada é dada abaixo, em que "dim" é um vetor de dimensão do array.

$$x <- \text{array}(\text{dados}, \text{dim} = c())$$

Exemplo: Criando um array com dimensão de 2 linhas e 5 colunas:

```
1 a <- array(dados=c(1:10), dim = c(2,5))
```

O R retornará:

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

Exemplo: Criando um array de 3 dimensões:

```
1 b <- array(1:18, dim = c(2,3,3))
```

```
> b
, , 1
```

```

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

```

```
, , 2
```

```

      [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

```

```
, , 3
```

```

      [,1] [,2] [,3]
[1,]   13   15   17
[2,]   14   16   18

```

Selecionar um elemento do array

O acesso dos elementos de um array é análogo ao de matriz e vetor, diferenciando no fato de que são informados n campos, considerando que são n dimensões.

Exemplo: Acessando um elemento do array **b** do exemplo anterior:

```
1 b[1,2,3]
```

```
> b[1,2,3]
[1] 15
```

3.3.4 Lista

Listas são estruturas genéricas e flexíveis que permitem armazenar diversos formatos em um único objeto.

Sintaxe:

```
list(elemento1, elemento2, ..., elementon)
```

Exemplo 17: Criando vetores **s** e **b**, e formando uma lista **x** com esses vetores:

```
1 s <- c("aa", "bb", "cc", "dd", "ee")
2 b <- c(TRUE, FALSE, TRUE, FALSE, FALSE)
3 x <- list(s, b, 3)
4 x
```

O R retornará:

```
> x
[[1]]
[1] "aa" "bb" "cc" "dd" "ee"

[[2]]
[1] TRUE FALSE TRUE FALSE FALSE

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

Operações com membros da lista

Sintaxe:

lista[]

Utilizando a lista **x** criada anteriormente:**Exemplo 18:** Imprimindo o segundo membro da lista **x**:

```
x[2]
```

O R retornará:

```
> x[2]
[[1]]
[1] TRUE FALSE TRUE FALSE FALSE
```

Exemplo 19: Imprimindo o segundo e o terceiro membro da lista **x**:

```
x[c(2,3)]
```

O R retornará:

```
> x[c(2,3)]
[[1]]
[1] TRUE FALSE TRUE FALSE FALSE

[[2]]
[1] 3

> x[[2]][3]
[1] TRUE
```

Operações com elementos dos membros da lista

Sintaxe:

lista[[*membro*]][*elemento*]

Exemplo 19: Imprimindo o terceiro elemento do segundo membro de **x**:

```
x[[2]][3]
```

O R retornará:

```
> x[[2]][3]
[1] TRUE
```

3.3.5 Data frame

O data-frame é uma estrutura semelhante à uma matriz porém com cada coluna sendo tratada separadamente. Desta forma podemos ter colunas de valores numéricos e colunas de caracteres no mesmo objeto. Dentro da mesma coluna todos elementos tem que ser do mesmo tipo.

Cada vetor (coluna) tem que ter o mesmo número de observações.

Sintaxe sendo todos os elementos de mesmo tipo e tamanho:

```
data.frame(elemento1,...,elementoN)
```

Exemplo 20: Criando um data-frame df com elementos(vetores) n, s, b e t:

```
1 n <- c(2, 3, 5)
2 s <- c("aa", "bb", "cc")
3 b <- c(TRUE, FALSE, TRUE)
4 t <- c(paste0("H", 1:3))
5 df <- data.frame(n, s, b, t)
6 df
```

O R retornará:

```
> df
  n s    b t
1 2 aa TRUE H1
2 3 bb FALSE H2
3 5 cc TRUE H3
```

Nomes para as linhas do data-frame

Exemplo 21: Dando nome as linhas do data-frame df:

```
1 row.names(df) <- c("linha1", "linha2", "linha3")
2 df
```

O R retornará:

```
> row.names(df)
[1] "linha1" "linha2" "linha3"
> df
      n s    b t
linha1 2 aa TRUE H1
linha2 3 bb FALSE H2
linha3 5 cc TRUE H3
```

Operações com data-frame

Sintaxe:

```
dataframe[linha,coluna]
```

Exemplo 22: Imprimir a observação da primeira linha e segunda coluna do data-frame:

```
1 df[1,2]
```

O R retornará:

```
> df[1,2]
[1] aa
```

Exemplo 23: Outra forma de imprimir a observação da primeira linha e segunda coluna do data-frame:

```
1 df["linha1", "s"]
```

O R retornará:

```
> df["linha1", "s"]
[1] aa
```

Colunas do data-frame

Neste caso o R retornará um vetor com elementos da coluna.

Sintaxe:

```
dataframe[[coluna]]
```

Exemplo 23: Duas formas de imprimir um vetor com os elementos da terceira coluna:

```
1 df[[3]]
2 df[["b"]]
```

O R retornará:

```
> df[[3]]
[1] TRUE FALSE TRUE

> df[["b"]]
[1] TRUE FALSE TRUE
```

Exemplo 23: Outras formas de imprimir um vetor com os elementos da terceira coluna:

```
1 df$b
2 df[, "b"]
3 df[, 3]
```

O R retornará:

```
> df$b
[1] TRUE FALSE TRUE
> df[, "b"]
[1] TRUE FALSE TRUE
> df[, 3]
[1] TRUE FALSE TRUE
```

Colunas específicas do data-frame

Neste caso o R retornará na mesma estrutura do data-frame apenas a coluna específica.

Sintaxe:

```
dataframe[coluna]
```

Exemplo 24: Imprimir apenas a terceira coluna:

```
1 df[3]
2 df["b"]
```

O R retornará:

```
> df[3]
      b
linha1 TRUE
linha2 FALSE
linha3 TRUE

> df["b"]
      b
```

```
linha1 TRUE
linha2 FALSE
linha3 TRUE
```

Exemplo 25: Imprimir apenas a segunda e a terceira coluna:

```
1 df[c("b", "s")]
```

O R retornará:

```
> df[c("b", "s")]
      b  s
linha1 TRUE aa
linha2 FALSE bb
linha3 TRUE cc
```

Linhas específicas do data-frame

Neste caso o R retornará na mesma estrutura do data-frame apenas a linha específica.

Sintaxe:

```
dataframe[linha,]
```

Exemplo 24: Imprimir apenas a segunda linha:

```
1 df[2,]
2 df['linha2',]
```

O R retornará:

```
> df[2,]
      n  s      b  t
linha2 3 bb FALSE H2

> df['linha2',]
      n  s      b  t
linha2 3 bb FALSE H2
```

Exemplo 25: Imprimir apenas a segunda e a terceira linha:

```
1 df[c(2,3),]
2 df[c("linha2", "linha3"),]
```

O R retornará:

```
> df[c(2,3),]
      n  s      b  t
linha2 3 bb FALSE H2
linha3 5 cc  TRUE H3

> df[c("linha2", "linha3"),]
      n  s      b  t
linha2 3 bb FALSE H2
linha3 5 cc  TRUE H3
```

3.4 Tabelas

3.4.1 Tabelas simples

Uma das formas de se facilitar a visualização de conjuntos de dados podemos utilizar tabelas. Para a construção de tabelas simples (de frequência), utilizamos:

```
table(dados)
```

Exemplo

```
1 Var1=c("A","A","B","B","B")
2 table(Var1)
```

Retorno do R:

```
Var1
A B
2 3
```

3.4.2 Tabelas de contingência

Quando tratamos de mais de uma variável de interesse e queremos cruzar informações, usualmente fazemos uso de tabelas de contingência, que representam a frequência das observações conforme as variáveis estudadas. O comando para construção desse tipo de tabela é similar ao caso anterior, bastando informar as outras variáveis que serão utilizadas na tabela, ficando desta forma:

```
table(dados$ Variável1,dados$ Variável2,...)
```

Exemplo

```
1 Var1=c("Pessoa1","Pessoa1","Pessoa2","Pessoa2","Pessoa2")
2 Var2=c("A","H","A","H","H")
3 table(Var1,Var2)
```

```
      Var2
Var1    A H
Pessoa1 1 1
Pessoa2 1 2
```

3.4.3 Tabelas de proporção

Quando é de interesse termos noção da frequência relativa a uma das variáveis da nossa tabela de contingência, criamos as tabelas de proporção. Para isso, usamos:

```
prop.table(X=tabela,margin=...)
```

OBS: Note que já é preciso ter criado um objeto do tipo tabela previamente para utilizar esse comando. Além disso, a opção `margin=` indica qual é a marginal (linha, coluna, ...) da tabela a qual a proporção será relativa. Para esclarecer, vejamos o exemplo:

Exemplo

Caso tenhamos uma tabela das variáveis Sexo x Hábito de fumar da forma:


```

      Fumante
Sexo Não Sim
  F     1    1
  M     0    2

```

E queremos saber a proporção de fumantes e não fumantes dentro de cada nível da variável Sexo (linha). Usamos:

```
1 prop.table(X=tabela, margin=1)
```

```

      Fumante
Sexo Não Sim
  F 0.5 0.5
  M 0.0 1.0

```

3.5 Funções

É possível escrever funções no R utilizando a seguinte sintaxe:

```
função <- function(variável(argumentos)){ função em si }
```

Exemplo:

```
1 fx<-function(x){ x^2 }
```

Para “chamar” a função basta digitar seu nome e informar o(s) parâmetro(s):

```
1 fx(2)
```

```
[1] 4
```

Funções são muito úteis e aplicáveis em vários modos dentro do R. Pode-se aplicar funções em vetores, matrizes, arrays, etc. Como exemplo vemos:

Exemplo: Aplicando função em um vetor

```
1 x<- 1:10
2 fx(x)
```

```
[1] 1 4 9 16 25 36 49 64 81 100
```

Exemplo: Aplicando função em uma matriz

```
1 x<-matrix(1:15, nrow = 5, ncol = 3)
2 print(list(x, fx(x)))
```

```

[[1]]
      [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14

```

```
[5,]    5    10    15

[[2]]
      [,1] [,2] [,3]
[1,]     1    36   121
[2,]     4    49   144
[3,]     9    64   169
[4,]    16    81   196
[5,]    25   100   225
```

3.6 Funções Apply

No R existem funções específicas para se aplicar outras funções nas estruturas de dados (matriz, data frame, lista, ...). Essas funções são denominadas `apply` e facilitam a manipulação de dados por substituírem estruturas de repetição (for, while, until e outras) presente na maioria das linguagens de programação.

3.6.1 Apply

Trata-se da função mais simples desse gênero, é utilizada em dados estruturados como matrizes, data frames ou arrays para aplicar alguma função conforme alguma marginal (linha, coluna, etc...). Como exemplo é possível se obter a soma das linhas de uma matriz:

```
1 matriz=matrix(1:16,4,4)
2 matriz
3 apply(matriz,1,sum)
```

```
> matriz
      [,1] [,2] [,3] [,4]
[1,]     1     5     9    13
[2,]     2     6    10    14
[3,]     3     7    11    15
[4,]     4     8    12    16
> apply(matriz,1,sum)
[1] 28 32 36 40
```

É possível aplicar qualquer função que seja criada dentro do R.

Exemplo

```
1 matriz=matrix(1:16,4,4)
2 matriz
3 apply(matriz,1,function(x){x^2+0.5})
```

```
> matriz
      [,1] [,2] [,3] [,4]
[1,]     1     5     9    13
[2,]     2     6    10    14
[3,]     3     7    11    15
[4,]     4     8    12    16
> apply(matriz,1,function(x){x^2+0.5})
      [,1] [,2] [,3] [,4]
```

```
[1,] 1.5 4.5 9.5 16.5
[2,] 25.5 36.5 49.5 64.5
[3,] 81.5 100.5 121.5 144.5
[4,] 169.5 196.5 225.5 256.5
```

3.6.2 Tapply

Trata-se de uma extensão do apply comum, na qual é possível utilizar uma outra variável como marginal. Note que como será utilizada outra variável como marginal nos restringimos a dados estruturados em data frame. Como exemplo é possível tirar a média de idade conforme o sexo em uma base de dados que possua essas variáveis.

```
1 #criando a base de dados
2 dados=data.frame(sexo=rep(c("M","F"),c(9,11)),          idade=c
3   (79,2,95,22,25,73,82,23,6,19,43,39,9,88,
4   89,41,4,13,92,33))
5 head(dados)
```

```
      sexo idade
1      M     79
2      M      2
3      M     95
4      M     22
5      M     25
6      M     73
```

```
1 tapply(dados$idade,dados$sexo,mean)
```

```
      F      M
42.72727 45.22222
```

3.6.3 Sapply

Trata-se de uma função mais genérica, que aplica alguma função em estruturas de dados não necessariamente estruturadas (lista) não considerando marginais. Como exemplo, é possível calcular a média de cada parte de uma lista.

Exemplo

```
1 x=1:10
2 y=2:14
3 z=60:90
4 lista=list(x,y,z)
5 lista
6 sapply(lista, mean)
```

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

[[2]]
[1] 2 3 4 5 6 7 8 9 10 11 12 13 14
```

```
[[3]]
[1] 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
    84 85 86 87 88 89 90
```

```
> sapply(lista, mean)
[1] 5.5 8.0 75.0
```

3.6.4 Lapply

A ideia dessa função é similar a da Sapply, com a diferença de retornar uma lista como resultado ao invés de um vetor.

Exemplo

```
1 x=1:10
2 y=2:14
3 z=60:90
4 lista=list(x,y,z)
5 lista
6 lapply(lista, mean)
```

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

```
[[2]]
[1] 2 3 4 5 6 7 8 9 10 11 12 13 14
```

```
[[3]]
[1] 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
[24] 84 85 86 87 88 89 90
```

```
> lapply(lista, mean)
[[1]]
[1] 5.5
```

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

```
[[3]]
[1] 75
```

3.7 Entrada de dados

Discutiremos nessa seção a entrada de dados no R, mais precisamente nos formatos `txt`, `csv` e `xls \ xlsx`, pois são os mais comuns de serem encontrados.

Antes de começar a importação de dados é importante definir o local (pasta) em que trabalharemos, para isso usa-se:

```
setwd("C:\\ Usuário\\ Local")
```

OBS: Note que é preciso utilizar aspas (") para indicar o local e também barras duplas (\\) ou barras invertidas (/) para separação.

Caso queiramos saber qual o local que está definido como pasta de trabalho, usamos:

```
getwd()
```

3.7.1 Csv

Para se ler dados com extensão `csv` dentro do R, utiliza-se o comando:

```
read.csv("dados.csv",...)
```

É possível ainda ler arquivos da Web, basta indicar o endereço (URL) dentro da função.

Exemplo

Neste exemplo iremos ler um banco de dados da Web sobre gatos domésticos.

```
1 gatos = read.csv("https://vincentarelbundock.github.io/Rdatasets/csv/boot/catsM.csv")
```

3.7.2 Txt

Para se ler dados com extensão `txt` dentro do R, utiliza-se o comando:

```
read.table("dados.txt",...)
```

Essa função é mais genérica, pois também lê arquivos `csv`, mas seu uso é análogo à função vista anteriormente.

3.7.3 Xls ou Xlsx

Existem duas maneiras para leitura de arquivos desse tipo (planilha de Excel). A primeira maneira consiste em exportar, a partir do Excel, esse arquivo em formato `csv` ou `txt` e a partir daí repetir os métodos vistos anteriormente nessa mesma seção. Para fazê-lo deste modo basta selecionar (dentro do Excel) *Salvar Como* e na segunda aba (tipo de arquivo) escolher `csv` ou `txt`.

O segundo método é a partir de pacotes próprios para leitura e manipulação de arquivos dessa natureza, uma opção é o pacote `xlsx` que possui a função:

```
read.xlsx(...)
```

3.7.4 Base de dados do R

O R base já vem com alguns bancos de dados que são usados para aprendizado ou exemplificação. Para listá-los usamos:

```
data()
```

Para utilizá-los basta atribuímos o nome do banco escolhido a algum objeto, ou usando pelo próprio nome no R.

3.7.5 Opções importantes

Quando importamos um conjunto de dados no R é importante informarmos a estrutura desse conjunto, isto é, a separação de elementos, o identificador de decimais, entre outros. As principais configurações são mostradas no quadro abaixo:

- **header= T** ou **F** (indica se a primeira linha representa o cabeçalho ou não)
- **dec=** (indica qual o símbolo utilizado para separação decimal)
- **sep=** (indica qual o símbolo separador das informações do banco de dados)

Exemplo

Neste exemplo carregaremos o mesmo banco de dados da Web sobre gatos que vimos antes, porém pelo comando `read.table`, em que precisamos informar o separador e se existe cabeçalho.

```
1 gatos=read.table("https://vincentarelbundock.github.io/Rdatasets/csv/boot/catsM.csv", sep=";", header = T)
```

3.7.6 Conferência dos Dados

É recomendável conferir a importação dos dados para evitar erros futuros na análise. O comando para verificar as primeiras observações (linhas) do banco é dado por:

```
head(dados)
```

Em que é possível informar a quantidade de linhas que serão mostradas. Por *default* o R apresenta as 6 primeiras observações.

Exemplo

Nesse exemplo, utilizaremos o conjunto de dados *cars*, que já está presente no R.

```
1 head(cars)
```

	speed	dist
1	4	2
2	4	10
3	7	4
4	7	22
5	8	16
6	9	10

Exemplo

Observando o conjunto de dados dos exemplos anteriores:

```
1 head(gatos,3)
```

	X	Sex	Bwt	Hwt
1	1	M	2.0	6.5
2	2	M	2.0	6.5
3	3	M	2.1	10.1

Para se verificar a estrutura do banco de dados, isto é, o formato que está, o tipo de cada uma das suas variáveis e as dimensões, usamos:

```
str(dados)
```

Exemplo

```
1 str(cars)
```

```
'data.frame': 50 obs. of 2 variables:  
 $ speed: num 4 4 7 7 8 9 10 10 10 11 ...  
 $ dist : num 2 10 4 22 16 10 18 26 34 17 ...
```

3.7.7 Funções adicionais

É possível facilitar o acesso às colunas de uma base de dados, isto é, ao invés de utilizar o operador \$ para acessar alguma coluna podemos utilizar o seu próprio nome. Para fazer isso, utiliza-se o comando:

```
attach(dados)
```

Esse comando irá trazer para a memória do computador cada coluna como um objeto, logo não é recomendável fazer isso com tanta frequência e com uma quantidade grande de dados.

Para desfazer esse anexo de dados na memória, usamos:

```
detach(dados)
```

3.8 Pacotes

Como o R é *opensource*, a comunidade como um todo desenvolve e implementa a linguagem com uma série de funcionalidades novas. Para disponibilizar essas outras funcionalidades que não estão presentes no pacote básico do R existem os denominados **pacotes**, que podem ser baixados no CRAN do R a partir do comando:

```
install.packages("nome do pacote")
```

Feito o download do pacote, é preciso carregá-lo para começar a utilizar suas funcionalidades, para isso, utilizamos:

```
library(nome do pacote)
```

ou ainda:

```
require(nome do pacote)
```

Exemplo

Como exemplo iremos baixar e carregar o pacote *nortest* que tem algumas funções para realizar testes de normalidade.

```
1 install.packages("nortest")  
2 library(nortest)
```

Utilizando a função *ad.test()* desse pacote obtemos:

```
1 ad.test(1:100)
```

Anderson-Darling normality test

data: 1:100

A = 1.0837, p-value = 0.007308

IV

Capítulo 4

4	Estatística básica	42
4.1	Medidas de Posição	
4.2	Medidas de Dispersão	
4.3	Correlação	



4. Estatística básica

Para calcular algumas medidas de estatística básica, tais como média, mediana, variância e outras, existem funções prontas implementadas no R. Essas funções serão apresentadas e exemplificadas nessa seção.

4.1 Medidas de Posição

As medidas descritivas são úteis para a representação dos dados a serem trabalhados e normalmente precedem qualquer análise estatística. Nesta parte discutiremos sobre as medidas de posição, e seguiremos com exemplos.

4.1.1 Média

Para se calcular a média de um conjunto de dados utiliza-se a função:

```
mean()
```

Exemplo

```
1 x = 1:11
2 mean(x)
```

```
[1] 6
```

4.1.2 Mediana

Para se calcular a média de um conjunto de dados utiliza-se a função:

```
median()
```

Exemplo

```
1 x=1:11
2 median(x)
```

```
[1] 6
```

4.1.3 Mínimo e máximo

Assim como quase tudo no R, existe mais de uma maneira de se calcular o mínimo e o máximo em um conjunto de dados. A primeira maneira é utilizando as funções:

min()

max()

Exemplo

```
1 x=1:11
2 min(x)
3 max(x)
```

```
> min(x)
[1] 1
> max(x)
[1] 11
```

A outra forma é utilizando o comando:

range()

Mas neste caso o R retornará um vetor de 2 posições, sendo o mínimo representado pelo primeiro elemento e o máximo pelo segundo.

Exemplo

```
1 x=1:11
2 range(x)
```

```
[1] 1 11
```

4.1.4 Quantis

Para se obter os quantis de um conjunto de dados utiliza-se a função:

quantiles()

Em que é possível indicar quais são os quantis de interesse dentro da função. Por *default* a função retorna os quantis de 0%, 25%, 50%, 75% e 100%, mas é possível achar os qualquer quantil de interesse em uma sequência de valores.

Exemplo

```
1 x=1:11+c(rep(1.2,4),rep(2.3,5),rep(4.3,2))
2 quantile(x)
3 quantile(x,c(.05,.95))
```

```
> quantile(x)
 0%  25%  50%  75% 100%
2.2  4.7  8.3 10.8 15.3
> quantile(x,c(.05,.95))
 5%  95%
2.7 14.8
```

4.1.5 Moda

No R base não existe uma função específica para se calcular a moda, isto é, o valor mais frequente no conjunto. Então, uma sugestão para se verificar esse valor mais frequente é criar uma tabela de frequência através do comando `table()`, visto anteriormente, e investigar nessa tabela o valor que mais se repete.

Obs¹: Note que este método só é válido para dados de natureza quantitativa discreta.

Obs²: Utilizaremos a função `which.max()` que irá retornar a posição da tabela com o máximo da frequência.

Exemplo

```
1 x=c(2,1,2,2,1,4,4,5,2,6,5,3,2,4,1,6)
2 tb=table(x)
3 tb
4 which.max(tb)
```

```
> tb
x
1 2 3 4 5 6
3 5 1 3 2 2
> which.max(tb)
2
2
```

4.1.6 Função Summary

A função `summary()` é uma função genérica dentro do R que, quando aplicada a objetos da classe de vetor numérico, retorna algumas medidas de posição.

Exemplo

```
1 x=c(1.3,1.5,1.3,1.7,1.9,2,1.4)
2 summary(x)
```

```
> summary(x)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
1.300   1.350   1.500   1.586   1.800   2.000
```

4.2 Medidas de Dispersão

4.2.1 Variância

Uma medida muito importante para se entender o comportamento dos dados é a variância (σ^2), para calcularmos essa medida usamos:

var()

Exemplo

```
1 x=c(1.2,1,4,8.5,2,2.5)
2 var(x)
```

```
[1] 7.9
```

4.2.2 Desvio padrão

Sabendo que o desvio padrão é a raiz quadrada da variância (σ), é possível achá-lo aplicando a função de raiz quadrada, vista nas sessões anteriores, na função de variância:

```
sqrt(var())
```

Porém existe uma forma mais direta, sendo:

```
sd()
```

Exemplo

```
1 x=c(1.2,1,4,8.5,2,2.5)
2 sqrt(var(x))
3 sd(var(X))
```

```
> sqrt(var(x))
[1] 2.810694
> sd(x)
[1] 2.810694
```

4.2.3 Coeficiente de variação

Para se ter uma métrica da variabilidade relativa dos dados é interessante vermos o coeficiente de variação, sendo ele $\frac{\sigma}{\bar{x}} \times 100$. Analogamente, no R usamos:

```
sd()/mean()*100
```

Exemplo

```
1 x=c(1.2,1,4,8.5,2,2.5)
2 coef=sd(x)/mean(x)*100
3 round(coef,2)
```

```
[1] 87.83
```

4.2.4 Amplitude

Para se calcular a amplitude basta subtrair o valor mínimo do máximo presente no conjunto de dados. Desta forma, ficamos com:

```
Amp = max()-min()
```

Exemplo

```

1 x=c(1.2,1,4,8.5,2,2.5)
2 amp=max(x)-min(x)
3 amp

```

```
[1] 7.5
```

4.3 Correlação

Quando tratamos de variáveis quantitativas, principalmente contínuas, é interessante ter uma medida explicativa a respeito da quantidade de variabilidade compartilhada entre duas variáveis. Essa medida é usualmente o coeficiente de correlação, um valor entre -1 e 1 que explica a força (proximidade de 11) e direção (positiva ou negativa) da correlação. A seguir vemos como calcular essa medida em 3 abordagens diferentes (Pearson, Kendall e Spearman).

4.3.1 Coeficiente de Pearson

Para se calcular o coeficiente de correlação de Pearson usamos o comando:

```
cor(...,method="pearson")
```

Obs: Por *default* no comando `cor()` calcula o coeficiente de Pearson.

Exemplo

```

1 x=1:10
2 y=11:20
3 cor(x,y,method="pearson")

```

```
[1] 1
```

4.3.2 Coeficiente de Kendall

Para se calcular o coeficiente de correlação de Kendall usamos o comando:

```
cor(...,method="kendall")
```

Exemplo

```

1 x=1:10
2 y=10:1
3 cor(x,y,method="kendall")

```

```
[1] -1
```

4.3.3 Coeficiente de Spearman

Para se calcular o coeficiente de correlação de Spearman usamos o comando:

```
cor(...,method="spearman")
```

Exemplo

```

1 x=1:10
2 y=c(0.1,0.3,0.1,0.5,0.3,0.7,0.3,0.5,0.6,0.6)
3 cor(x,y,method="spearman")

```

```
[1] 0.7308218
```



Capítulo 5

5	Gráficos	48
5.1	Gráficos para variáveis qualitativas	
5.2	Gráficos para variáveis quantitativas	
5.3	Ajustes Gráficos	



5. Gráficos

Neste capítulo iremos apresentar um pouco da capacidade do **R** para a construção de gráficos, para isso utilizaremos apenas a biblioteca gráfica nativa do R. Os bancos de dados utilizados nos exemplos serão disponibilizados via links conforme sua utilização.

5.1 Gráficos para variáveis qualitativas

Para representarmos variáveis de natureza qualitativa é usual utilizarmos gráficos de barras, colunas e setores, desta forma apresentaremos nos tópicos abaixo uma forma de fazê-los no **R**.

O conjunto de dados que será usado aqui é o "Curso R.csv" disponível em: <https://sites.google.com/site/cursoderbse/dados>

5.1.1 Gráfico de Barras

Antes de se criar o gráfico é preciso criar um objeto na forma de **tabela** para então representá-lo graficamente.

```
1 tabela <- table( variavel )
```

O comando para se fazer gráficos de barras é dado por:

```
barplot(tabela)
```

Para criar-se um gráfico com barras horizontais basta inserir o argumento:

```
horiz=T
```

Exemplos:

Utilizaremos o banco de dados *Curso R.csv*, disponível em url do banco para fazer um gráfico de barras para a variável *Cor.Raca*.

```
1 tabela <- table( dados$Cor.Raca )
```



```
2 barplot(tabela)
```

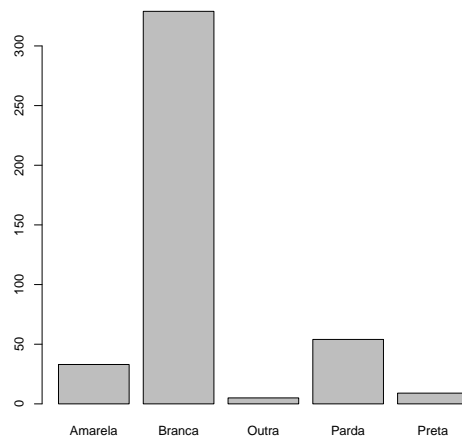


Figura 5.1: Gráfico de barras para a variável Cor.

```
1 tabela <- table(dados$Cor, Raca)
2 barplot(tabela, horiz=T)
```

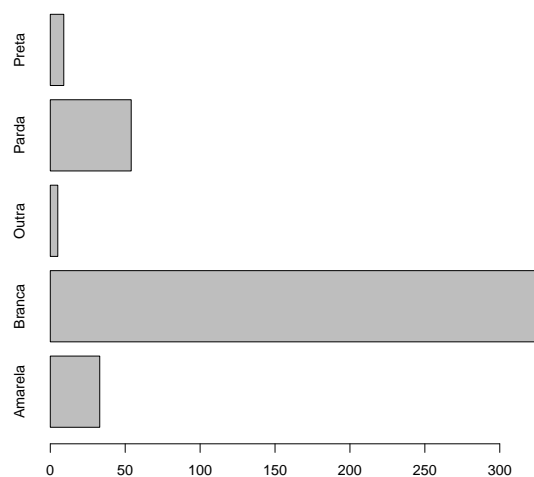


Figura 5.2: Gráfico de barras horizontais para a variável Cor.

5.1.2 Gráfico de Setores

Assim como o Gráfico de Barras, é preciso criar uma **tabela** para se fazer o gráfico de setores. O comando para se fazer gráficos de setores é dado por:

pie(tabela)

Exemplo:

Continuaremos utilizando o mesmo banco de dados do tópico anterior, porém agora consideraremos a variável *Sexo*.

```
1 tabela <- table(dados$Sexo)
2 pie(tabela)
```

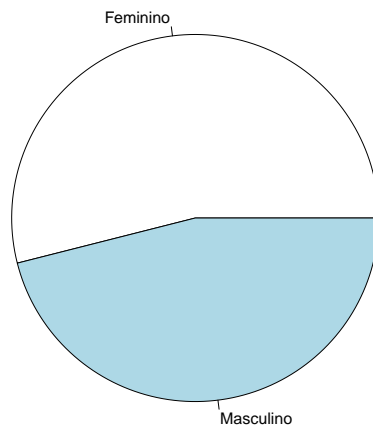


Figura 5.3: Gráfico de setores para a variável Sexo.

5.2 Gráficos para variáveis quantitativas

5.2.1 Histograma

A forma mais utilizada para se representar variáveis quantitativas contínuas é através do histograma. São dados os comandos abaixo para a construção de um histograma.

`hist(variável)`

Exemplo Criando um histograma para a variável *Peso* do mesmo conjunto de dados utilizado anteriormente.

```
1 hist(dados$Peso)
```

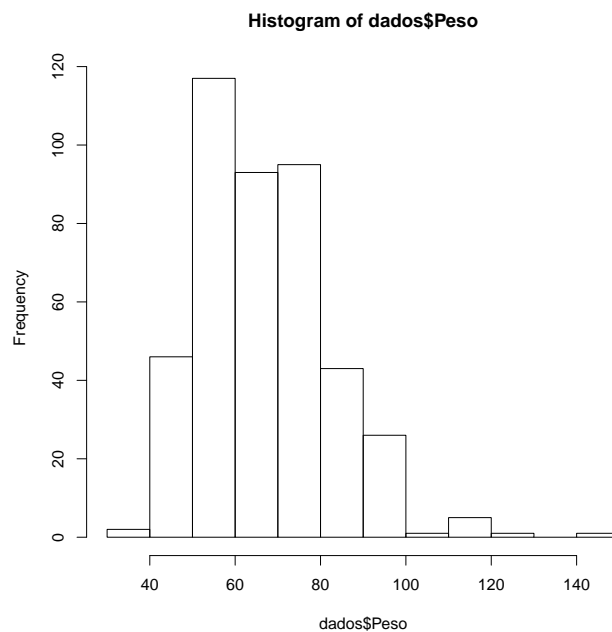


Figura 5.4: Histograma para a variável Peso.

5.2.2 Polígono de frequência

Para construirmos o polígono de frequência vamos utilizar outras funções gráficas: A função *lines()* sobrepõe o gráfico com alguma linha, para a qual daremos as coordenadas x e y. No caso x são os pontos médios do histograma e y é a frequência absoluta.

```
1 histograma<-hist(dados$Peso)
2 lines(x = c(histograma$breaks[1], histograma$mids,
3           histograma$breaks[length(histograma$breaks)]),
4       y = c(0, histograma$counts, 0), col=2, lwd=3)
```

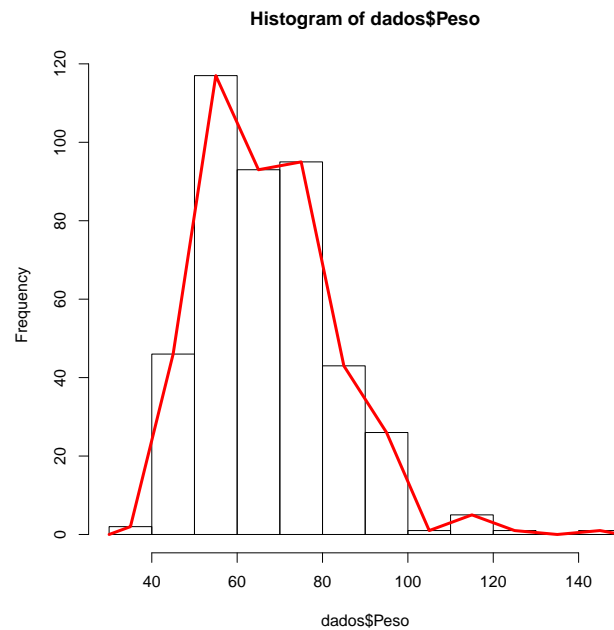


Figura 5.5: Polígono de frequência para a variável Peso.

Note que atribuímos o histograma a um objeto para conseguir acessar as informações `mids` (pontos médios), `counts` (frequência absoluta) e `breaks` (intervalos de classe).

OBS: Um polígono de frequência deve começar na primeira classe e terminar na última, portando o X está variando do primeiro elemento do vetor `histogram$breaks` até o último elemento desse mesmo vetor.

O argumento `col=` informa a cor e `lwd=` informa a espessura da linha.

5.2.3 Gráfico de bastões (hastes)

Para se representar variáveis de natureza quantitativa discreta é comum utilizarmos o gráfico de bastões ou hastes. Para fazê-lo basta utilizar a função genérica **plot()** nos dados em forma de tabela e ajustar alguns argumentos.

Exemplo

Criando gráfico para frequência da variável idade, nas condições de estarem entre 20 e 25 anos.

```
1 idades_20_25<-subset(dados$Idade, dados$Idade >=20 & dados$Idade <=25)
2 tabela<-table(idades_20_25)
3 plot(tabela, type='h', lwd=3)
```

A opção `type=` informa o tipo de gráfico que queremos. Nesse caso, `h` é o tipo bastão. Entraremos em mais detalhes sobre isso em breve.

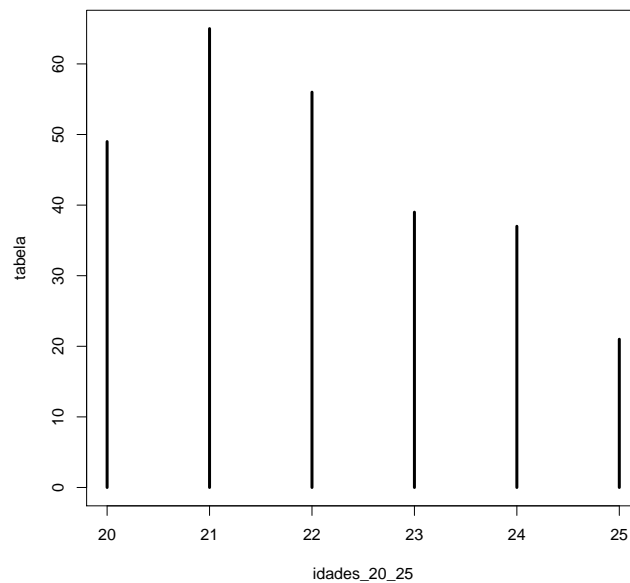


Figura 5.6: Gráfico de hastes da variável Idade entre 20 e 25 anos.

5.2.4 Gráfico de Dispersão

Quando tratamos de duas variáveis quantitativas é interessante observar o comportamento conjunto entre elas. A maneira mais usual para se verificar esse comportamento é o gráfico de dispersão. Para construirmos esse gráfico basta utilizarmos a função genérica **plot()** do R, dessa forma o comando fica:

```
plot(variável 1, variável 2)
```

Exemplo

Verificando comportamento conjunto das variáveis Peso e Altura.

```
1 plot(dados$Peso, dados$Altura)
```

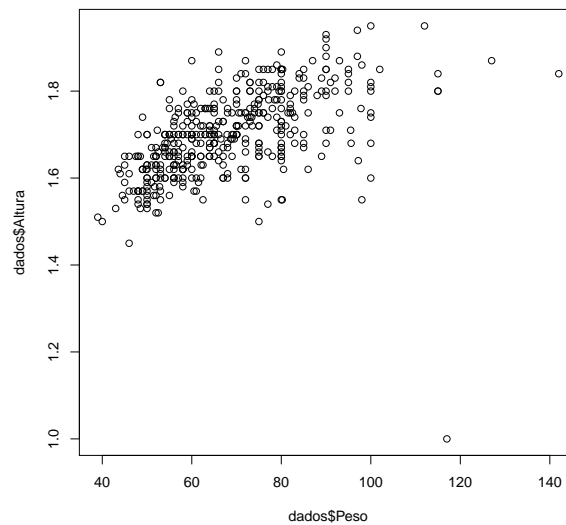


Figura 5.7: Gráfico de dispersão das variáveis Peso e Altura.

5.2.5 Gráfico de Caixas (Box-plot)

Quando tratamos de variáveis quantitativas é interessante observar, a distribuição dos dados de um modo geral. Uma forma de se verificar isso é pelo gráfico de Caixas, em que conseguimos visualizar os quartis, mediana, limites superior e inferior e outliers.

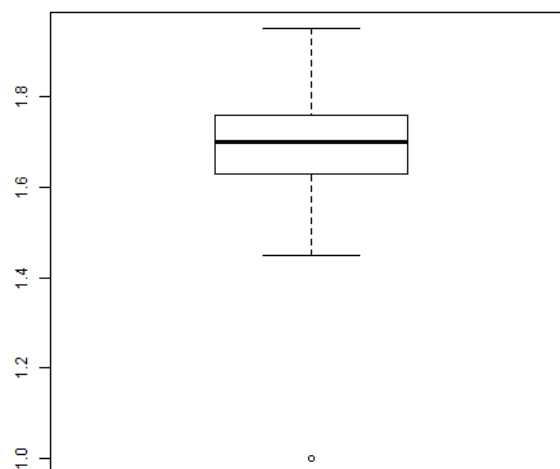
Para construirmos esse gráfico basta utilizarmos a função:

```
boxplot(variável)
```

Exemplo

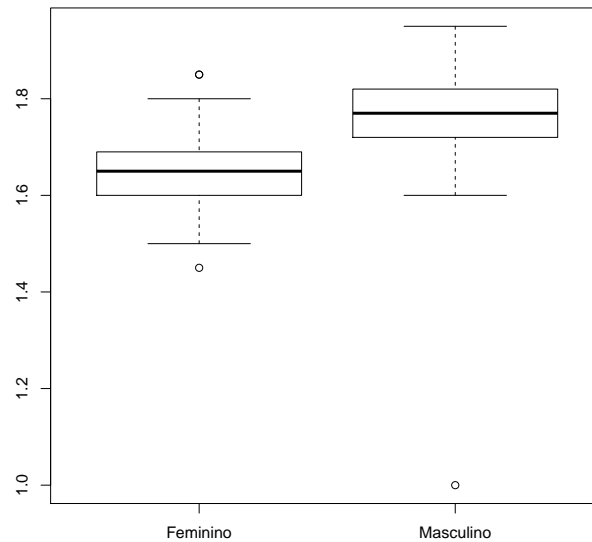
Criando um Box-plot para a variável altura.

```
1 boxplot(dados$Altura)
```



Criando um Box-plot para a variável altura conforme a variável Sexo.

```
1 boxplot(dados$Altura~dados$Sexo)
```



5.3 Ajustes Gráficos

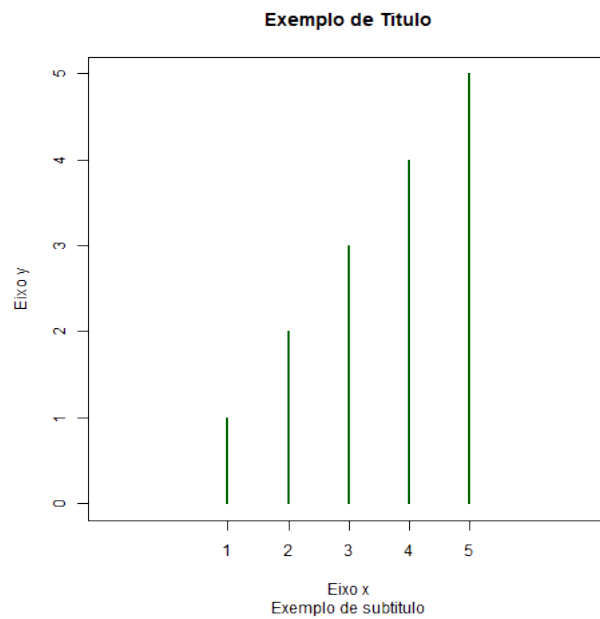
5.3.1 Principais ajustes

Em praticamente qualquer gráfico, teremos as seguintes opções:

- **main=** Informa o título do gráfico
- **sub=** Cria um subtítulo do gráfico
- **xlab=** e **ylab=** Trocam as legendas dos eixos
- **xlim=** e **ylim=** Trocam os limites dos eixos
- **col=** Informa a cor do gráfico

Exemplo

```
1 plot(table(rep(1:5,1:5)),main = "Exemplo de Titulo",sub= "Exemplo de subtítulo",  
xlab = "Eixo x",ylab = "Eixo y",xlim = c(-1,7),ylim = c(0,5),col="darkgreen")
```



5.3.2 Funções de sobreposição

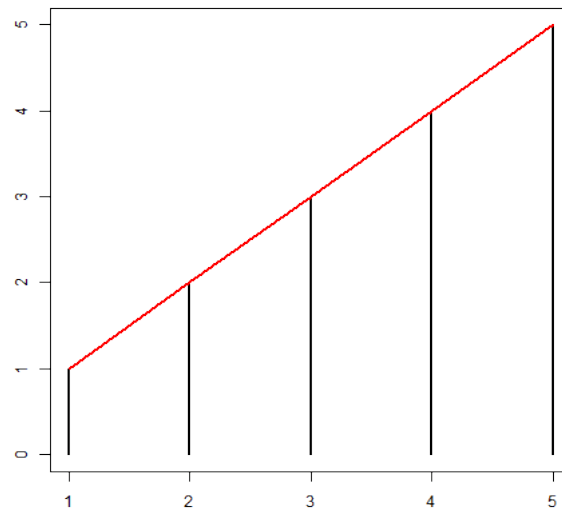
No R existem algumas funções de sobreposição de gráficos:

- **lines()** : cria uma linha contínua sobrepondo o gráfico
- **points()** : cria pontos sobrepondo o gráfico
- **polygon()** : cria polígonos sobrepondo o gráfico

A seguir são apresentados alguns exemplos de uso de cada uma dessas funções.

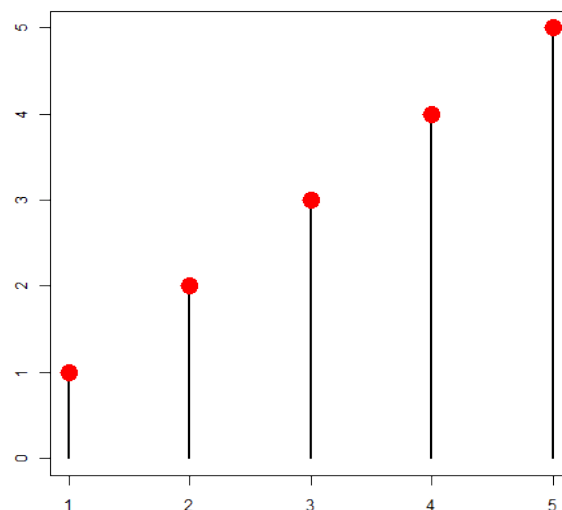
Função lines()

```
1 plot(table(rep(1:5, 1:5)), ann=F)
2 lines(x=1:5, y=1:5, lwd=2, col=2)
```

Função points()

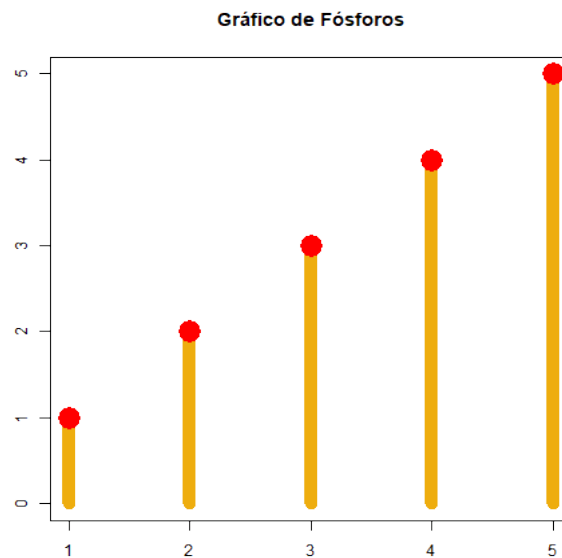
```
1 plot(table(rep(1:5, 1:5)), ann=F)
2 points(x=1:5, y=1:5, lwd=10, col=2)
```



Exemplo prático

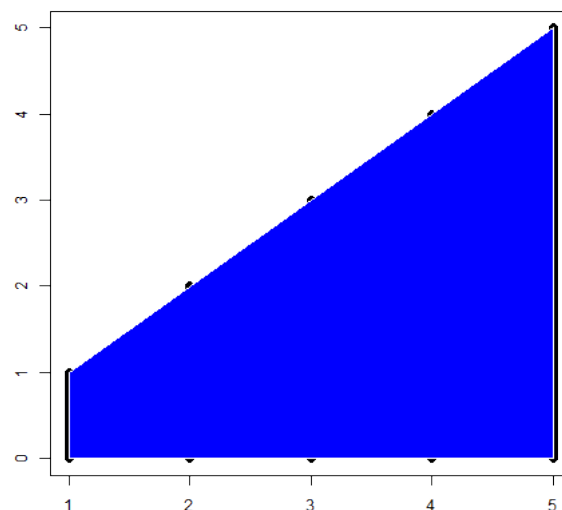
Note que é possível sobrepor indeterminadamente os gráficos, ou seja, é possível criar pontos, retas e formas em um só gráfico, como visto no exemplo a seguir:

```
1 plot(table(rep(1:5, 1:5)), xlab="", ylab = "", main="Gráfico de óFsforos", lwd=10,
   col="darkgoldenrod2")
2 points(x=1:5, y=1:5, lwd=13, col=2)
```



Função `polygon()`

```
1 plot(table(rep(1:5, 1:5)), ann=F, lwd=7)
2 polygon(x=c(1, 5, 5, 1), y=c(0, 0, 5, 1), lwd=0.01, col='blue', border = "white")
```



OBS: Também existem as funções `rect()`, que desenha retângulos no gráfico, e a função `symbols()` que desenha outros símbolos (círculos, quadrados, estrelas, boxplot, ...), mas por serem menos usuais, não trataremos delas aqui.

5.3.3 Argumento `type`

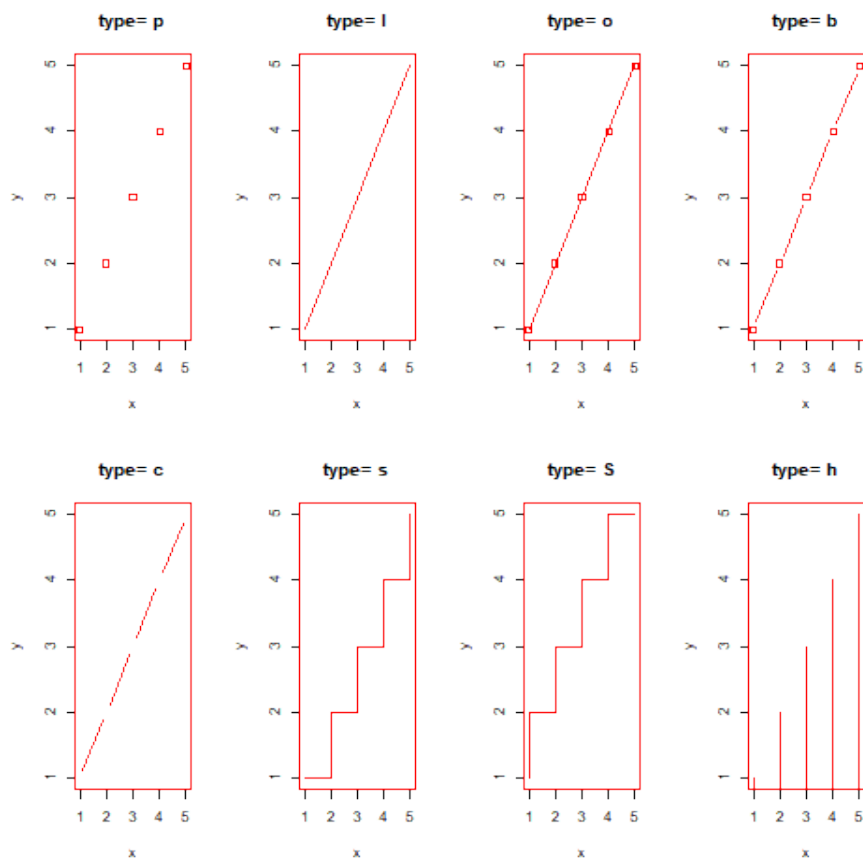
O argumento `type` dentro da função `plot()` indica o tipo de gráfico que será criado. Existem 9 tipos, são eles:

- "p"
- "l"
- "o"
- "b"
- "c"
- "s"
- "S"
- "h"
- "n"

Todos esses tipos são ilustrados pelo gráfico abaixo:

```

1 x <- c(1:5)
2 y <- x
3 par(pch=22, col="red")
4 par(mfrow=c(2,4))
5 opts = c("p","l","o","b","c","s","S","h")
6 for(i in 1:length(opts)){
7   heading = paste("type=",opts[i])
8   plot(x, y, type=opts[i], main=heading)
9 }
10 }
```

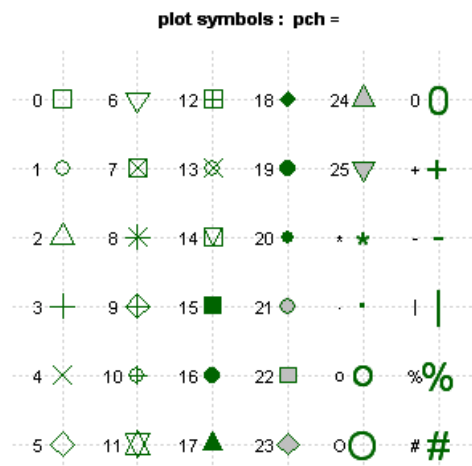


OBS: A opção `type="n"` cria um gráfico em branco.

5.3.4 Tipos de símbolos e linhas

Símbolos

Os símbolos utilizados em gráficos no R base são nomeados *pch*. Todos eles são expostos pela figura a seguir:



Linhas

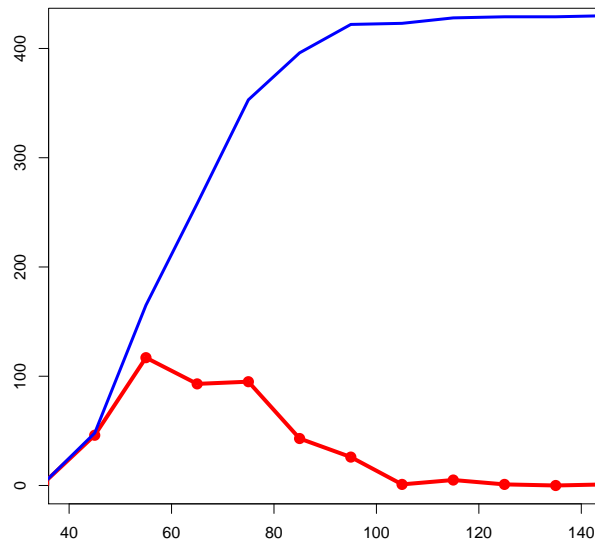
As linhas utilizadas em gráficos no R base são nomeadas *lty*. São apresentados os tipos de linha abaixo:



Exemplo:

Colocando um polígono de frequência absoluta e acumulada no mesmo gráfico.

```
1 histograma<-hist(dados$Peso)
2 plot(2,xlim=c(40,140),ylim=c(0,420),type="n",ann=F)
3 lines(x=c(histograma$breaks[1],histograma$mids,
4           tail(histograma$breaks,n=1)),y=c(0,histograma$counts,0),
5       col=2,lwd=4,type="o")
6 lines(x=c(histograma$breaks[1],histograma$mids,tail(histograma$breaks,n=1)),y=c(
7           0,cumsum(histograma$counts),tail(cumsum(histograma$counts),n=1)),col="blue",
8       lwd=3)
```



OBS: Note que inicialmente foi criado um gráfico em branco informando os limites de X e Y e em seguida sobrepôs-se o gráfico com as duas linhas.

5.3.5 Texto e legendas

Texto

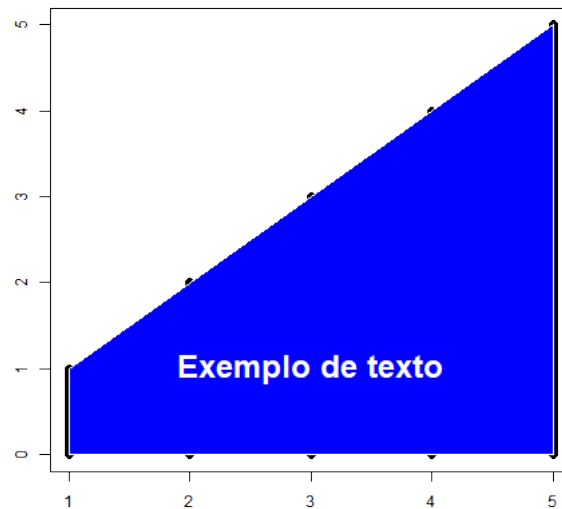
Para se adicionar texto no Gráfico, utilizaremos a função:

`text()`

Aonde informaremos:

Exemplo:

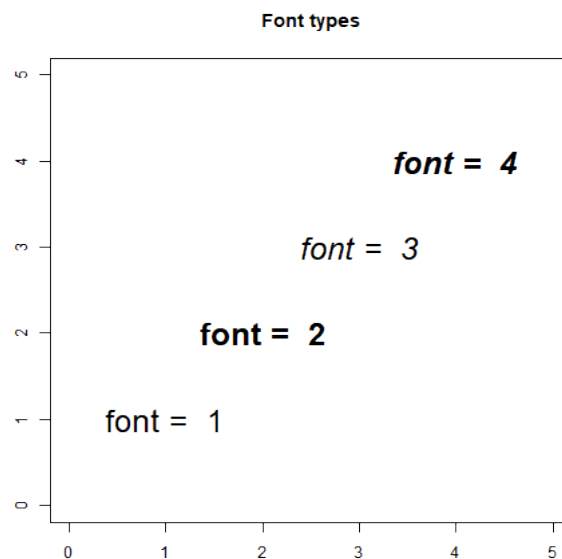
```
1 plot(table(rep(1:5, 1:5)), ann=F, lwd=7)
2 polygon(x=c(1, 5, 5, 1), y=c(0, 0, 5, 1), lwd=0.01, col='blue', border = "white")
3 text(x = 3, y=1, labels = "Exemplo de texto", col = "white", cex=2, font = 2)
```



OBS: o comando `cex=` informa o tamanho do texto e `font=` especifica o tipo de letra (negrito, itálico, etc..).

Exemplo do `font=`

```
1 plot(0:5, 0:5, type="n", main="Font types", xlab = "", ylab="")
2 text(1:4, 1:4, paste("font = ", 1:4), font=1:4, cex=2)
```



Legenda

Para se adicionar legenda ao Gráfico, utilizaremos a função:

`legend()`

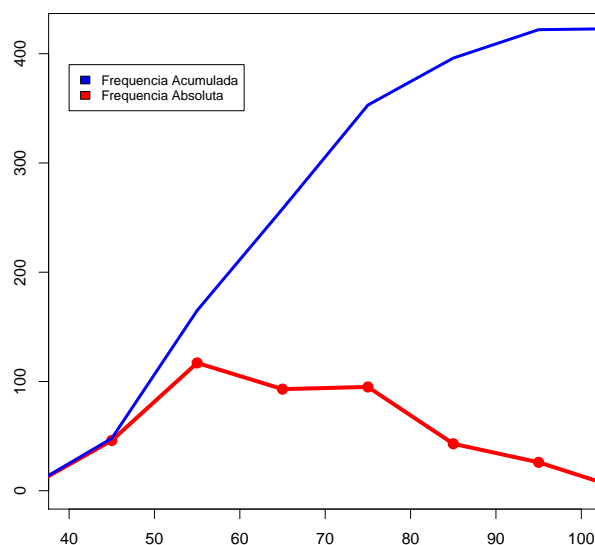
Aonde informaremos:

- A posição da legenda referente ao gráfico, informando as coordenadas X e Y.
- O texto das legendas.
- O preenchimento (cores) das legendas.
- Outras opções (cor,tamanho,etc..)

Exemplos práticos

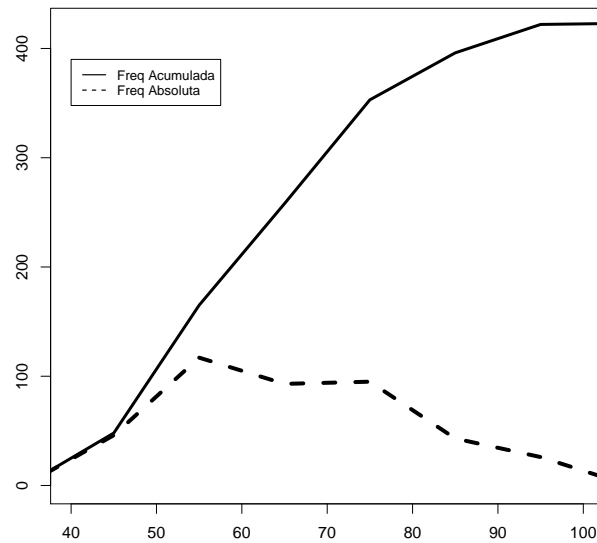
Colocando uma legenda no gráfico dos polígonos de frequência.

```
1 plot(2,xlim=c(40,100),ylim=c(0,420),type="n",ann=F)
2 lines(x=c(histograma$breaks[1],histograma$mids,tail(histograma$breaks,n=1)),y=c(
  0,histograma$counts,0),col=2,lwd=4,type="o")
3 lines(x=c(histograma$breaks[1],histograma$mids),y=c(0,cumsum(histograma$counts)
  ),col="blue",lwd=3)
4 legend(x=40,y=390,legend=c("Frequencia Acumulada","Frequencia Absoluta"),
  fill=c("blue","red"),cex=.8)
```



Diferenciando legenda por tipo de linha.

```
1 plot(2,xlim=c(40,100),ylim=c(0,420),type="n",ann=F)
2 lines(x=c(histograma$breaks[1],histograma$mids,tail(histograma$breaks,n=1)),y=c(
  0,histograma$counts,0),col=1,lwd=4,lty=2)
3 lines(x=c(histograma$breaks[1],histograma$mids),y=c(0,cumsum(histograma$counts)
  ),col=1,lwd=3)
4 legend(x=40,y=390,legend=c("Freq Acumulada","Freq Absoluta"),lty=c(1,2),
  cex=.8,lwd=1.2)
```

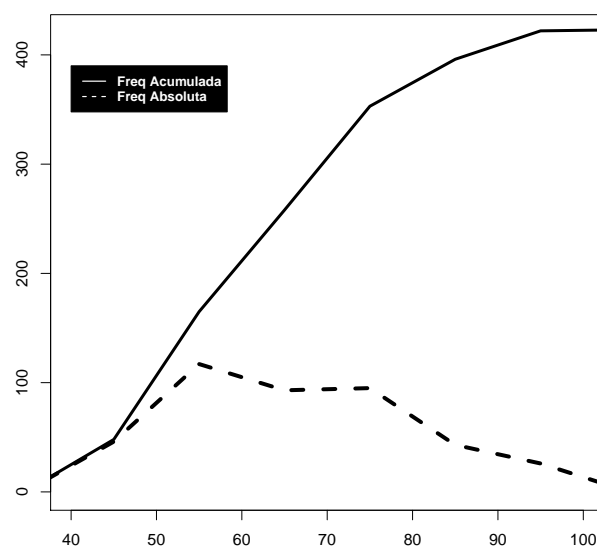


Legenda com outras opções.

```

1 plot(2,xlim=c(40,100),ylim=c(0,420),type = "n",ann = F)
2 lines(x=c(histograma$breaks[1],histograma$mids,tail(histograma$breaks,n=1)),y=c(
  (0,histograma$counts,0),col=1,lwd=4,lty=2)
3 lines(x=c(histograma$breaks[1],histograma$mids),y=c(0,cumsum(histograma$counts)
  ),col=1,lwd=3)
4 legend(x = 40,y = 390,legend = c("Freq Acumulada","Freq Absoluta"),lty = c(1,2)
  ,cex = .8,lwd = 1.2,bty = "o",bg = 1,col = "white",text.col = "white",text.
  font = 2)

```



5.3.6 Utilizando `par()` e alguns comandos adicionais

A função `par` altera os parâmetros gráficos do R, sendo assim, tudo o que será modificado de parâmetro gráfico (margem, tamanho de todas as letras, quantidade de gráficos na janela, ...) deverá ser colocado dentro da função `par()`.

OBS: antes de mudar os parâmetros recomenda-se salvar as configurações *default* do R em um objeto:

```
pardefault<-par()
```

Para retornar ao *default*, basta informar os parâmetros salvos anteriormente:

```
par(pardefault)
```

Opções `mfrow` e `mfc`

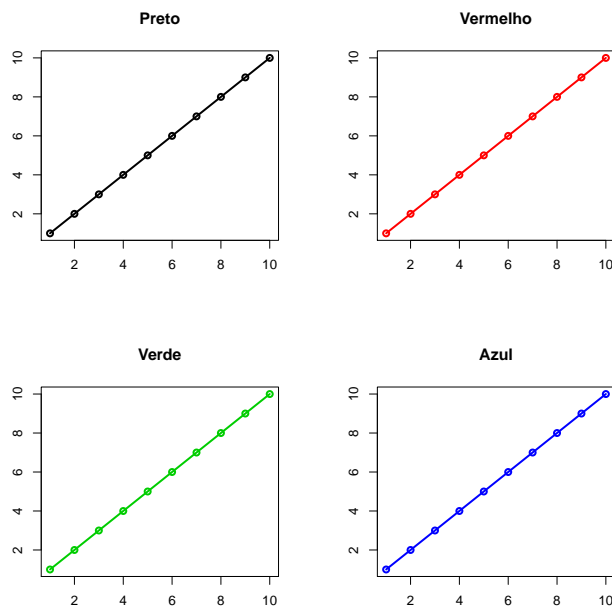
Essas opções servem para conseguirmos inserir mais de um gráfico na janela. Elas dividem a janela em uma matriz aonde deverão ser informados os números de linhas e colunas respectivamente. Ao definir o tamanho da matriz de saída gráfica, a janela será preenchida por linha se utilizarmos o `mfrow` e por coluna se utilizarmos o `mfc`. Seu uso se dá por:

```
par(mfrow=c(...,...))
```

```
par(mfcol=c(...,...))
```

Exemplo (`mfrow`):

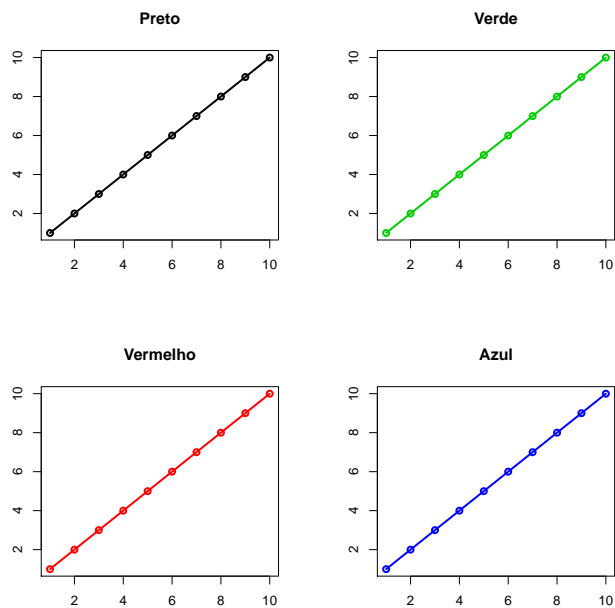
```
1 par ( mfrow=c ( 2 , 2 ) )
2 plot ( 1:10 , 1:10 , type = "o" , col=1 , main = " Preto " , lwd=2 ,
3       xlab = " " , ylab = " " )
4 plot ( 1:10 , 1:10 , type = "o" , col=2 , main = " Vermelho " , lwd=2 ,
5       xlab = " " , ylab = " " )
6 plot ( 1:10 , 1:10 , type = "o" , col=3 , main=" Verde " , lwd=2 ,
7       xlab = " " , ylab = " " )
8 plot ( 1:10 , 1:10 , type = "o" , col=4 , main = " Azul " , lwd=2 ,
9       xlab = " " , ylab = " " )
```

**Exemplo (mfcol):**

```

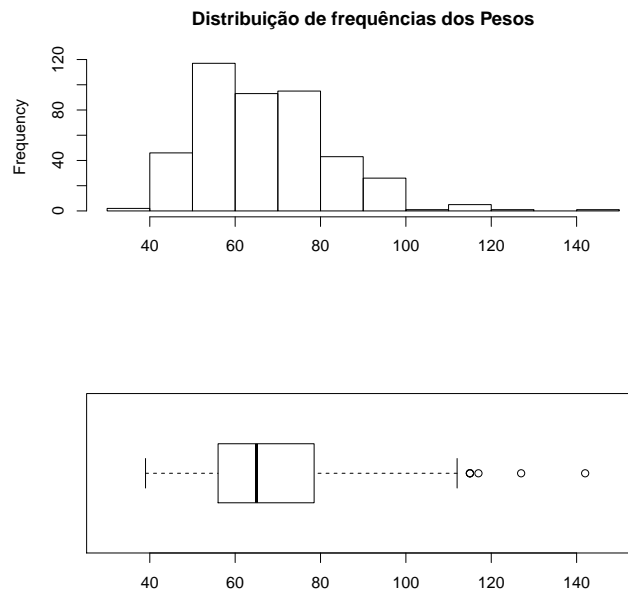
1 par(mfcol=c(2,2))
2 plot(1:10,1:10,type = "o",col=1,main = "Preto",lwd=2,
3     xlab = "",ylab = "")
4 plot(1:10,1:10,type = "o",col=2,main = "Vermelho",lwd=2,
5     xlab = "",ylab = "")
6 plot(1:10,1:10,type = "o",col=3,main="Verde",lwd=2,
7     xlab = "",ylab = "")
8 plot(1:10,1:10,type = "o",col=4,main = "Azul",lwd=2,
9     xlab = "",ylab = "")

```



Exemplo: Colocando histograma e boxplot juntos:

```
1 par(mfrow=c(2,1))
2 hist(dados$Peso, xlab = "", xlim = c(30,150), main = "Histograma dos Pesos")
3 boxplot(dados$Peso, horizontal = T, ylim = c(30,150))
```

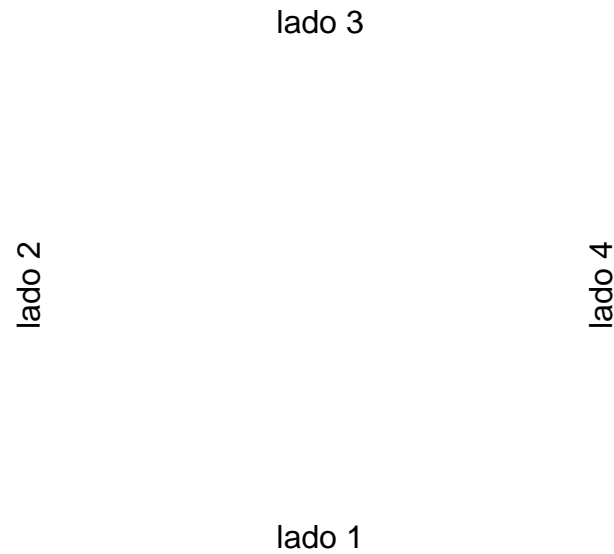


Opção mar

Essa opção tem a função de mudar as margens da janela gráfica.

Para utilizá-la deverá ser informado um vetor com as medidas das margens de cada lado. Cada elemento do vetor representa um lado do gráfico, isto é exemplificado pela imagem a seguir:

```
1 plot(1, type = 'n', ann = F, axes = F)
2 mtext("lado 1", 1, cex = 2)
3 mtext("lado 2", 2, cex = 2)
4 mtext("lado 3", 3, cex = 2)
5 mtext("lado 4", 4, cex = 2)
```

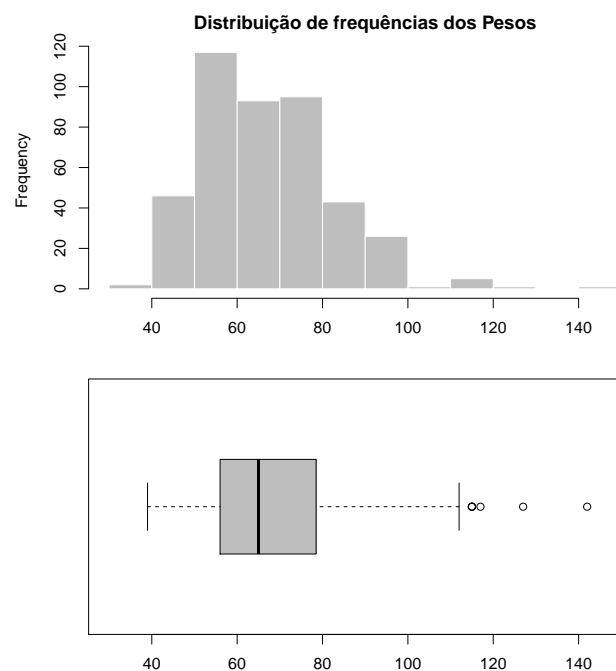


OBS: O comando `mtext` escreve textos nas margens do gráfico. O *default* é:

```
mar=c(5.1, 4.1, 4.1, 2.1)
```

Exemplo:

```
1 par(mfrow=c(2,1),mar=c(2.1,4.1,2.1,2.1))
2 hist(dados$Peso,xlab="",xlim=c(30,150),main="Histograma dos Pesos",col="
  grey",border="white")
3 boxplot(dados$Peso,horizontal=T,ylim=c(30,150),col="grey")
```



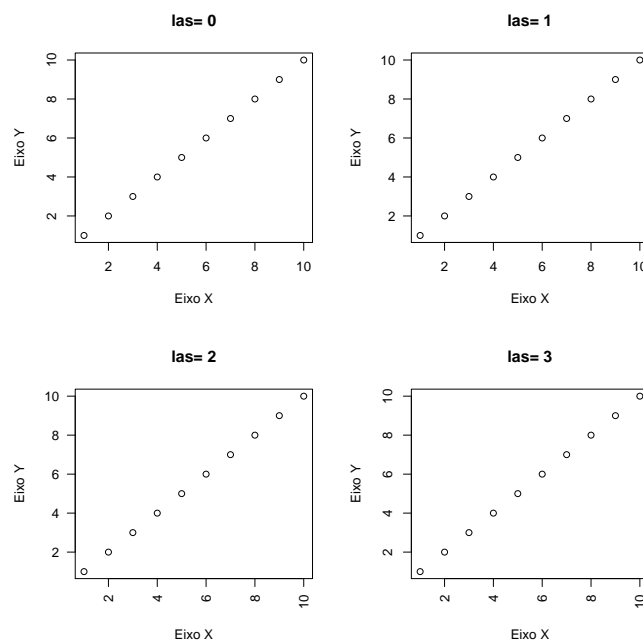
Opção *las*

Essa opção tem a função de mudar a orientação dos eixos do gráfico. Para utilizá-la, escreve-se:

```
par(las=...)
```

Todos os tipos de orientação são dados pela figura a seguir:

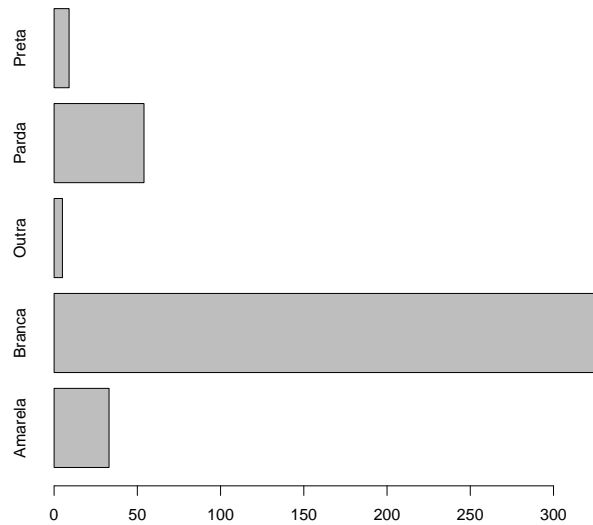
```
1 par(mfrow=c(2,2))
2 lss = 0:3
3 for(i in 1:4){
4   heading = paste("las=", lss[i])
5   par(las=lss[i])
6   plot(1:10, 1:10, main=heading, xlab = "Eixo X", ylab = "Eixo Y")
7 }
```

**Exemplo:**

Mudando orientação do gráfico de barras do exemplo 5.1.1:

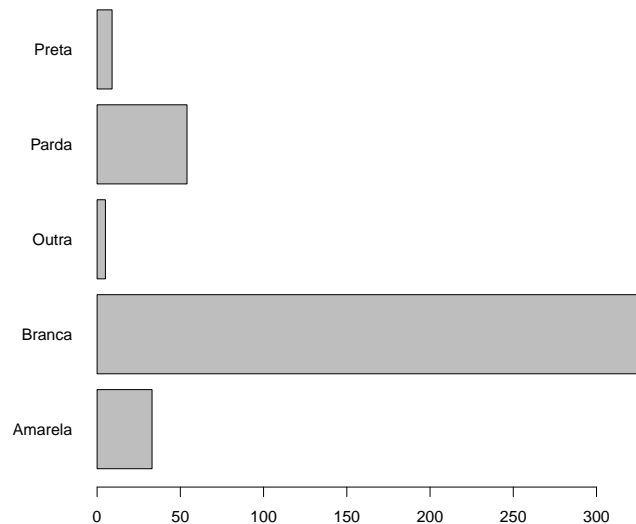
Relembrando que o gráfico da sessão anterior estava da seguinte forma:

```
1 plot(dados$Cor.çRaa, horiz=TRUE)
```



Ao utilizar o comando `las=1`, percebemos que o gráfico se acerta:

```
1 par(las=1)
2 plot(dados$Cor.Raca, horiz=TRUE, xlim=c(0,350),
3      border="white")
```



Comandos adicionais:

A seguir são apresentados alguns comandos adicionais para modificação dos gráficos. O funcionamento de todas essas funções não foram exemplificados neste material.

Derivações do comando `cex`:

- **cex.axis** : muda o tamanho (aparência) dos eixos;
- **cex.lab**: muda o tamanho das labels;
- **cex.main**: muda o tamanho do título;
- **cex.sub**: muda o tamanho do subtítulo;

Derivações do comando col:

- **col.axis**: muda a cor dos eixos;
- **col.lab**: muda a cor das labels;
- **col.main**: muda a cor do título;
- **col.sub**: muda a cor do subtítulo;

Derivações do comando font:

- **font.axis**: muda o tipo de escrita dos eixos;
- **font.lab**: muda o tipo de escrita das labels;
- **font.main**: muda o tipo de escrita do título;
- **font.sub**: muda o tipo de escrita do subtítulo;

5.3.7 Cores

Para listar todas as cores existentes no R, usamos o comando:

```
colors()
```

```
1 head(colors(), 10)
```

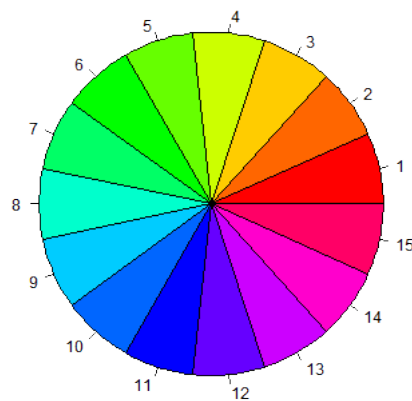
```
[1] "white"          "aliceblue"      "antiquewhite"   "antiquewhite1" "antiquewhite2"  
[6] "antiquewhite3" "antiquewhite4" "aquamarine"     "aquamarine1"   "aquamarine2"
```

Escalas de cores padrões do R

No R existem escalas de cores padrões para facilitar a escolha de cores nos gráficos. As escalas são ilustradas pelas figuras a seguir:

rainbow()

```
1 pie(rep(1, 15), col = rainbow(15))
```



heat.colors

```
1 pie(rep(1,15),col = heat.colors(15))
```

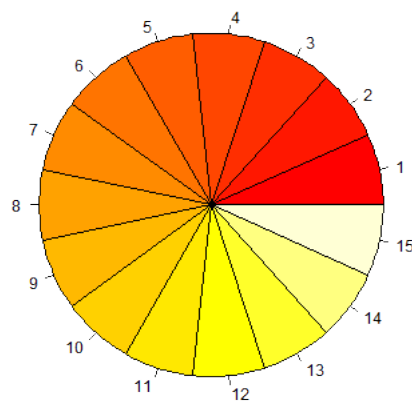
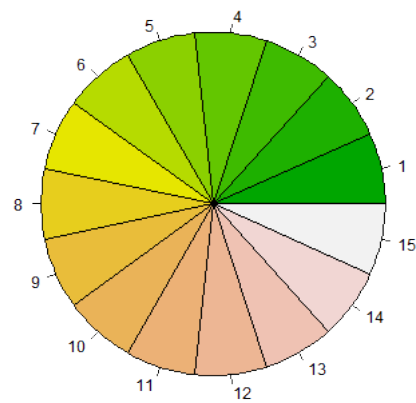


Figura 5.8: Janelas gráficas abertas

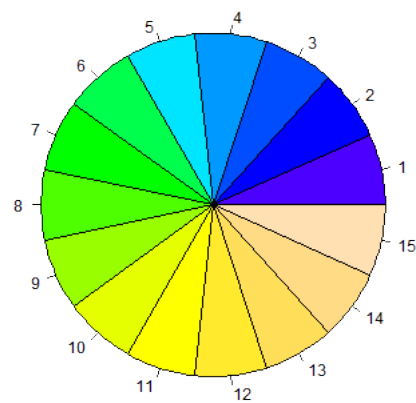
terrain.colors

```
1 pie(rep(1,15),col = terrain.colors(15))
```

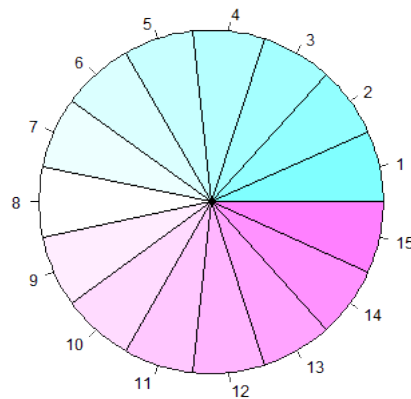
topo.colors

```
1 pie(rep(1,15),col = topo.colors(15))
```



cm.colors

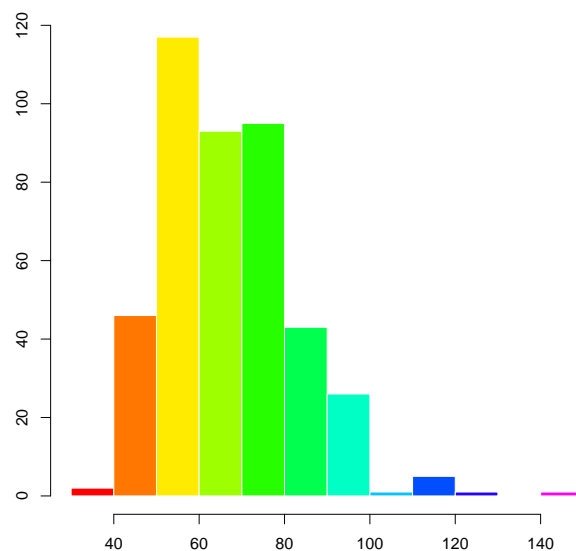
```
1 pie(rep(1,15),col = cm.colors(15))
```



Exemplo:

Criando um histograma com as cores do arco-íris.

```
1 histograma=hist(dados$Peso)
2 hist(dados$Peso,col = rainbow(length(histograma$breaks)),ann=F,border = "white"
   )
```



Função rgb

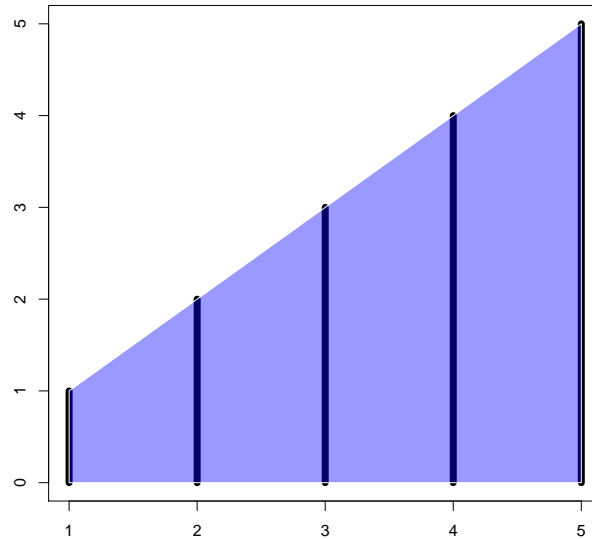
É a função que retorna um valor de cor RGB a partir de um conjunto de componentes vermelho, verde e azul, além da opacidade(alpha). Portanto é necessário informar a porcentagem de cada uma das 3 cores (0 a 1) e o nível de opacidade (alpha).

Exemplo

```

1 plot(table(rep(1:5,1:5)),ann=F,lwd=7)
2 polygon(x=c(1,5,5,1),y=c(0,0,5,1),lwd=0.01,col=rgb(red = 0,green = 0,blue = 1,
  alpha = .4)
3       ,border = "white")

```



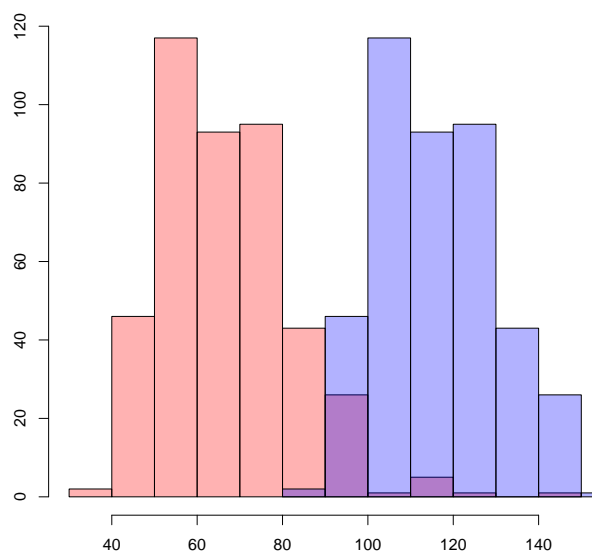
Exemplo

Criando histogramas sobrepostos:

```

1 hist(dados$Peso,col = rgb(1,0,0,.3),ann=F)
2 hist(dados$Peso+50,col = rgb(0,0,1,.3),add=T)

```



OBS: `add=T` permite sobreposição dos histogramas.

5.3.8 Janelas gráficas externas:

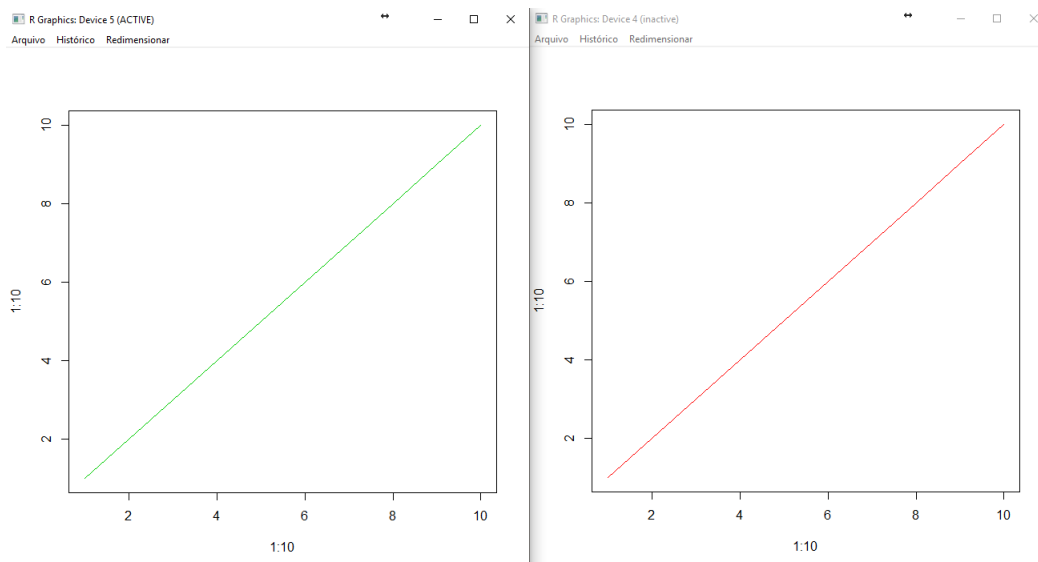
Algumas vezes por comodidade e para que haja mais de uma janela com gráficos para fins de comparação é interessante criar novas janelas gráficas. Para se abrir uma janela gráfica separada utiliza-se os comandos:

`X11()` ou `x11()`

É possível criar várias janelas, cada uma comportando um gráfico.

Exemplo:

```
1 x11()
2 plot(1:10,1:10,col=2)
3 x11()
4 plot(1:10,1:10,col=3)
```



5.3.9 Representando tabelas de contingência

Como discutido nos capítulos anteriores, as tabelas de contingência são usadas para registrar observações de duas ou mais variáveis, normalmente qualitativas.

A partir daqui utilizaremos o conjunto de dados "Dados_Graficos.csv", disponível em <https://sites.google.com/site/cursoderbse/dados>.

Aqui será utilizada a tabela criada abaixo:

```
1 dados$Est_civil<-factor(dados$Est_civil,levels = 1:3,labels = c("solteiro","
  casado","outro"))
2 (tabela.est.sex<-table(dados$Est_civil,dados$sexo))
```

	Feminino	Masculino
solteiro	43	67
casado	52	53
outro	53	65

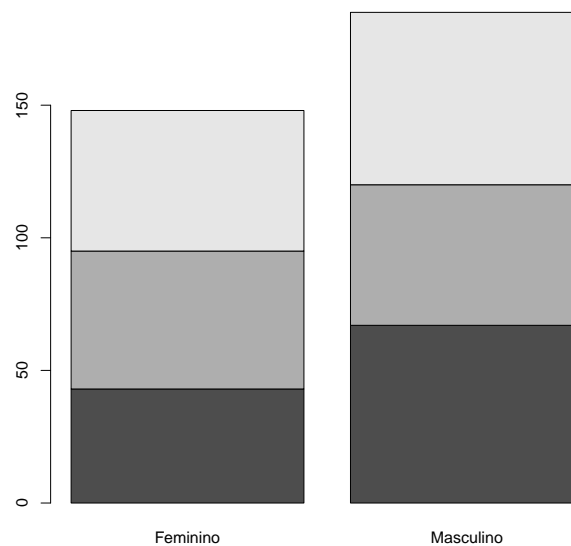
Uma tentativa inicial para representa-la é usar o `plot()`, porém, o gráfico não fica muito claro.

```
1 plot(tabela.est.sex)
```



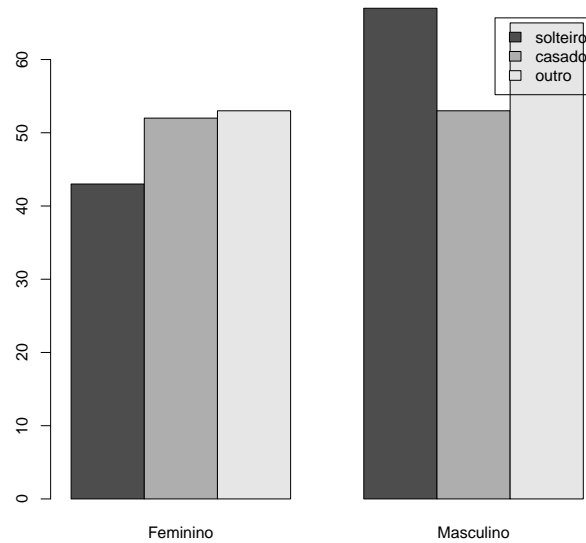
Outra alternativa é o gráfico de barras/columnas.

```
1 barplot(tabela.est.sex)
```



A opção `beside = T` define barras lado a lado e `legend.text=T` cria uma legenda automaticamente.

```
1 barplot(tabela.est.sex, beside = T, legend.text = T)
```

**Exemplo:**

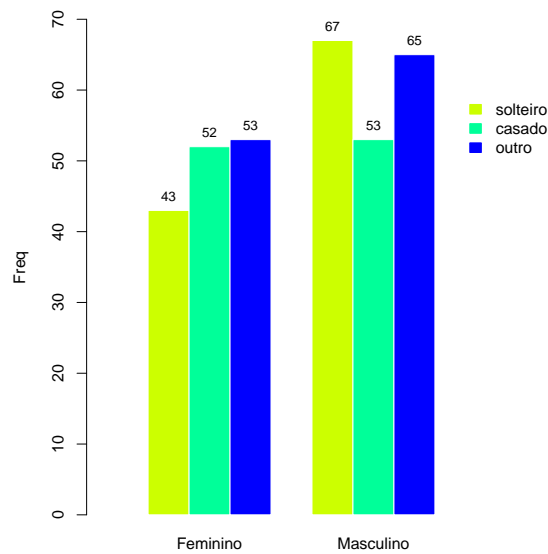
Ajustando o gráfico de barras laterais para representar a tabela criada anteriormente.

```
1 graf<-barplot(tabela.est.sex,beside = T)
2 graf
```

```
      [,1] [,2]
[1,]  1.5  5.5
[2,]  2.5  6.5
[3,]  3.5  7.5
```

OBS: Atribuiu-se o barplot a um objeto para conseguir informações sobre o eixo x.

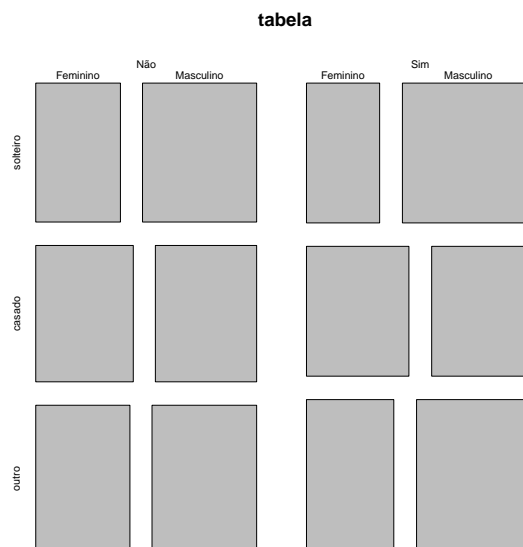
```
1 barplot(tabela.est.sex,beside = T,xlim = c(0,max(graf)+5),ylim = c(0,70),col =
  rainbow(3,start = .2),border = 'white',ylab = "Freq")
2 legend(x = max(graf)+1,y=60,fill = rainbow(3,start = .2),
3       legend = row.names(tabela.est.sex),border=rainbow(3,start = .2),
4       bty="n" )
5 text(c(graf[,1],graf[,2]),c(tabela.est.sex[,1]+2,tabela.est.sex[,2]+2),labels =
  c(tabela.est.sex[,1],tabela.est.sex[,2]),cex=.8)
```



Exemplo: Representando uma tabela de 3 variáveis

Para a tentativa inicial, podemos usar:

```
1 tabela<-table(dados$Bebe,dados$Est_civil,dados$sexo)
2
3 plot(tabela)
```



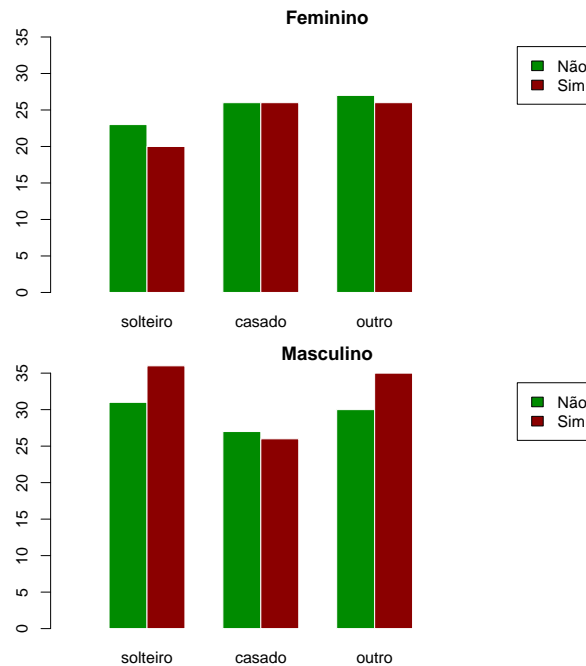
Percebe-se que não há muita clareza nas informações, logo vamos tentar representar a tabela de forma mais clara:

```
1 par(mfrow=c(2,1),mar=c(2.1,4.1,2.1,2.1))
2 bp1<-barplot(tabela[,1],beside = T,main = dimnames(tabela)[[3]][1])
```

```

3 bp2<-barplot(tabela[,2], beside = T, main = dimnames(tabela)[[3]][2])
4
5 barplot(tabela[,1], beside = T, main = dimnames(tabela)[[3]][1], xlim = c(0,max(
  bp1)+5), ylim = c(0,35), legend.text = T, col = c("green4", "red4"), border = "
  white")
6 barplot(tabela[,2], beside = T, main = dimnames(tabela)[[3]][2], xlim = c(0,max(
  bp2)+5), ylim = c(0,35), legend.text = T, col = c("green4", "red4"), border = "
  white")

```



Note que foram utilizadas muitas opções e o código se tornou um pouco complexo, uma das desvantagens de se usar o pacote básico do R para gráficos. Por outro lado, é possível ver o poder de customização em gráficos dessa ferramenta.

5.3.10 Representando funções

Como visto na sessão anterior, o R permite criar funções. É muito interessante poder representar graficamente essas funções.

Para fazer o gráfico de uma função, utiliza-se o comando:

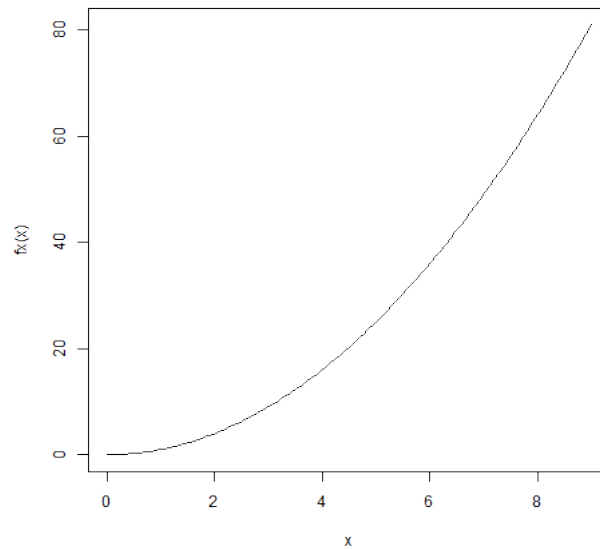
```
curve(função, from=..., to=...)
```

Exemplo:

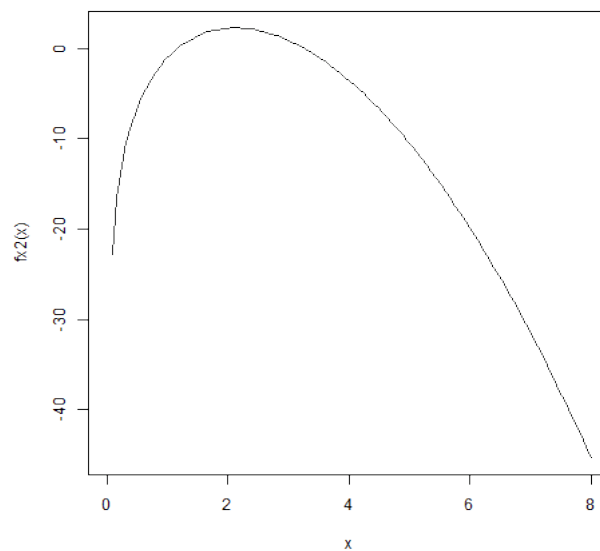
```

1 fx<-function(x){x^2}
2 curve(fx, from = 0, to=9)

```


**Exemplo:**

```
1 fx2<-function(x){-x^2+log(x^9)}  
2 curve(fx2, from = 0, to = 8)
```

**5.3.11 Exportando gráficos**

Para se exportar gráficos no R, pode-se considerar dois modos. O primeiro consiste em utilizar o botão *Export* presente no R Studio. A vantagem desse método é a facilidade, porém não se torna algo reproduzível, um exemplo disso é não conseguirmos exportar vários gráficos automaticamente. O outro modo é a partir de linhas de comando, desta forma é possível tornar a exportação automática, definindo as opções apenas uma vez.

Botão Export

Nesse modo basta escolher as opções prontas de formato de arquivo e tamanho, e então clicar conforme aparecem os botões. As figuras abaixo ilustram o processo:

Linhas de comando

É interessante utilizarmos linhas de comando, pois desta forma o gráfico fica reproduzível e é possível exportar mais de um gráfico de uma vez.

Existem os seguintes formatos básicos de exportação:

Formato	Driver
JPG	jpg
PNG	png
WMF	win.metafile
PDF	pdf
Postscript	postscript

Existem outros formatos, tais como **bmp** e **tiff**, além de pacotes que permitem salvar de outros modos.

O algoritmo básico para exportar é:

```
jpeg('nome_do_grafico.jpg',outras opções)
```

Ou pode-se usar qualquer uma dessas funções:

```
pdf(),png(),postscript(),win.metafile()
```

```
plot()
```

```
dev.off()
```

ATENÇÃO: O gráfico será salvo aonde o diretório está definido como local de trabalho. Caso queira salvar em outro diretório é possível informar o local onde será salvo:

```
jpeg(file = "C://R//SALVEAQUI//grafico.jpeg")
```

```
plot(x,y)
```

```
dev.off()
```

Por *default*, esse método salva o gráfico em tela cheia, mas é possível modificar nas opções.

Outra opção, é salvar o gráfico exatamente do jeito que está disposto na janela gráfica, fazendo:

```
plot()
```

```
savePlot(grafico.png)
```

É possível também salvar o gráfico do *device*, usando:

```
x11()
```

```
plot()
```

```
dev.copy(device=x11,'grafico.png')
```

```
dev.off()
```



Capítulo 6

6	Noções de probabilidade	84
6.1	Amostragem	
6.2	Análise Combinatória	
6.3	Distribuições de probabilidade	
6.4	Função de densidade / probabilidade	
6.5	Função acumulada	
6.6	Quantis	

6. Noções de probabilidade

Esse capítulo se propõe a exemplificar alguns conceitos básicos de probabilidade dentro do R. Como nas seções anteriores, serão apresentadas as funções pertinentes ao assunto estudado e um exemplo em seguida.

6.1 Amostragem

O comando `sample()` tem a função de obter amostras de alguma estrutura de dados que for de interesse.

É preciso informar:

- Os dados
- O tamanho da amostra
- Com reposição ou sem.
- As probabilidades de cada elemento (*Default=1/2*)

Exemplo (Numérico)

```
1 numeros<-1:10
2 amostra.sem.repos<-sample(x = numeros, size = 8, replace = F )
3 amostra.com.repos<-sample(x = numeros, size = 8, replace = T )
4 print(rbind( amostra.com.repos , amostra.sem.repos ))
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]
amostra.com.repos	5	10	1	1	8	7	10	7
amostra.sem.repos	9	10	8	1	4	3	6	5

Exemplo (Character)

```
1 letras<- letters
2 amostra.sem.repos<-sample(x = letras , size = 8, replace = F )
3 amostra.com.repos<-sample(x = letras , size = 8, replace = T )
```

```
4 print(rbind(amostra.com.repos, amostra.sem.repos))
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
amostra.com.repos "y"  "a"  "l"  "y"  "z"  "y"  "u"  "l"
amostra.sem.repos "i"  "s"  "j"  "a"  "x"  "l"  "d"  "v"
```

OBS: ao objeto letras foi atribuído `letters`, que é um vetor presente no R com todas as letras minúsculas. Também temos o `LETTERS` que contém as maiúsculas.

6.2 Análise Combinatória

6.2.1 Permutação

A permutação de n elementos distintos é um agrupamento ordenado desses elementos. Pode ser calculada por:

$$P_n = n!$$

No R basta utilizar a função de fatorial já implementada:

```
factorial(x)
```

Exemplo

```
1 factorial(10)
```

```
[1] 3628800
```

6.2.2 Arranjo

Um arranjo considera a posição dos elementos além das suas combinações. Não há uma função pronta no R base para fazê-lo, mas é possível implementar essa função facilmente com o seguinte código:

```
1 arranjo <- function(n, x) {
2   return(factorial(n) / factorial(n-x))
3 }
```

Exemplo

Calculando $Arr(5,3)$:

```
1 arranjo(5,3)
```

```
[1] 60
```

6.2.3 Combinação

Para fazer uma combinação de $\binom{n}{k}$ utiliza-se o comando:

```
choose(n,k)
```

Exemplo

Calculando $\binom{5}{3}$:

```
1 choose(5,3)
```

```
[1] 10
```

6.3 Distribuições de probabilidade

Quando trabalhamos com variáveis aleatórias, nos deparamos com distribuições de probabilidade associadas a essas variáveis. É necessário termos conhecimento dessas distribuições para que possamos tirar informações de dados a partir da estatística.

O pacote básico do R conta com as seguintes distribuições implementadas:

- Beta;
- Binomial
- Cauchy
- Chi-quadrado
- Exponencial
- F
- Gamma
- Geométrica
- Hipergeométrica
- log-normal
- Multinomial
- Binomial Negativa
- Normal
- Poisson
- T de Student
- Uniforme
- Weibull

6.3.1 Geração de valores aleatórios

Para se gerar valores aleatórios que venham de alguma distribuição de probabilidade utiliza-se o prefixo *r* conjugado com a abreviação da distribuição, ficando:

`r"distribuição"(n,...)`

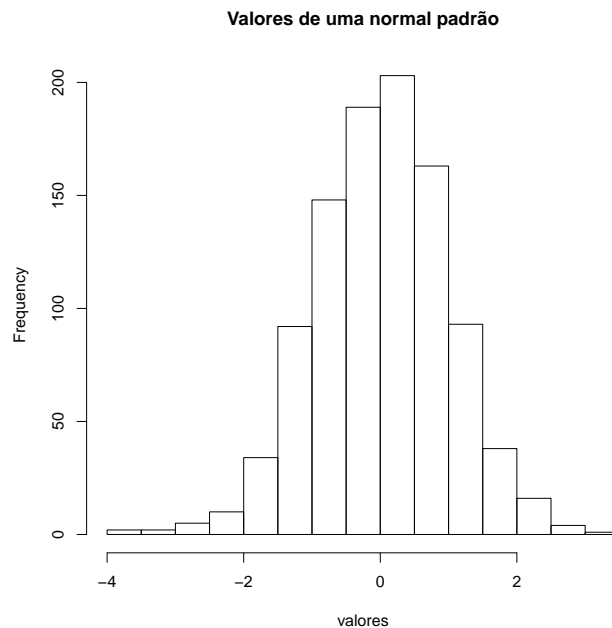
Para essa função é preciso informar:

- **n** : a quantidade de valores aleatórios gerados
- **...** : os parâmetros da distribuição a ser simulada

Exemplo

Gerando valores aleatórios de uma distribuição normal padrão e criando um histograma.

```
1 valores<-rnorm(n = 1000,mean = 0,sd = 1)
2 hist(valores ,main = "Valores de uma normal padrao")
```



6.4 Função de densidade / probabilidade

Para conhecermos as probabilidades da função densidade ou de probabilidade (caso discreto) de determinada distribuição, usamos:

d"distribuição"(p,...)

Para essa função é preciso informar:

- **x** : o quantil associado a probabilidade de interesse
- ... : os parâmetros da distribuição estudada

Exemplo

Achando $P(X = 2)$, sendo que $X \sim \text{Bin}(n = 4, p = 0.5)$

```
1 dbinom(x=2, size = 4, prob = 0.5)
```

```
[1] 0.375
```

6.5 Função acumulada

Para conhecermos as probabilidades da função acumulada de determinada distribuição, usamos:

p"distribuição"(p,...)

Para essa função é preciso informar:

- **q** : o quantil associado a probabilidade de interesse
- ... : os parâmetros da distribuição estudada

Exemplo

Achando $P(X \leq 2)$, sendo que $X \sim \text{Bin}(n = 4, p = 0.5)$

```
1 pbinom(q = 2, size = 4, prob = 0.5)
```

```
[1] 0.6875
```

6.6 Quantis

Para conhecermos os quantis de determinada distribuição, usamos:

$q_{\text{"distribuição"}}(p, \dots)$

Para essa função é preciso informar:

- **p** : a probabilidade associada ao quantil de interesse
- ... : os parâmetros da distribuição estudada

Exemplo

Achando y tal que $P(X \leq y) = 0.6875$.

OBS: $X \sim \text{Bin}(n = 4, p = 0.5)$

```
1 qbinom(p = 0.6875, size = 4, prob = 0.5)
```

```
[1] 2
```


VII

Capítulo 7

7	Exercícios	90
7.1	Operações Básicas	
7.2	Estruturas Básicas	
7.3	Entrada de dados	
7.4	Funções e gráficos	



7. Exercícios

7.1 Operações Básicas

1. Resolva os Exercícios utilizando as operações básicas do R, isto é, utilize-o como uma calculadora.
 - (a) $\sum_{n=1}^{10} n$
 - (b) $\sum_{n=1}^5 2n$
 - (c) $\sum_{n=1}^5 n^2$
 - (d) $\log(\sum_{n=1}^5 2n)$
 - (e) $\prod_{i=1}^3 i^2$
2. Um Estatístico fez, em certo dia, as 20 primeiras páginas de um livro sobre R. A partir desse dia, ele escreveu a cada dia tantas páginas quanto havia escrito no dia anterior e mais 5 páginas. Se o Estatístico trabalhou 4 dias, quantas páginas então ele escreveu?

7.2 Estruturas Básicas

1. Defina uma variável de cada tipo e 3 variáveis do tipo numérico e faça pelo menos 3 operações entre elas.
2. Crie um vetor de 1 até 15 com $n=5$ elementos.
3. Crie um vetor de 0 a 20 com $n=0.5$ passos.
4. Resolva as questões utilizando a ideia de vetor.
 - (a) $\sum_{n=1}^{10} n$
 - (b) $\sum_{n=1}^5 2n$
 - (c) $\sum_{n=1}^5 n^2$
 - (d) $\log(\sum_{n=1}^5 2n)$
 - (e) $\prod_{i=1}^3 i^2$
5. Refaça o Exercício 2 da seção 7.1, porém utilize a ideia de vetores.
6. Crie uma matriz[6,5] com 30 elementos.
7. Use a matriz do exercício anterior com disposição por linhas.

8. Utilizando a matriz do exercício 6 faça:
 - (a) Imprima apenas a coluna 2
 - (b) Imprima apenas a linha 4
 - (c) Imprima a matriz sem a linha 2 e 3
9. Faça a transposta da matriz do exercício 7.
10. Faça o que se pede:
 - (a) Crie uma lista (L) com os seguintes vetores:
 - `V1<- (N1, N2, N3, N4, N5)` (usando o comando paste)
 - `V2<- (T, F, F, T, F)`
 - `V3<- ("a", "b", "c", "d", "e")`
 - `V4<- (6, 7, 8, 9, 10)`
 - (b) Imprima o segundo e o terceiro vetor de L
 - (c) Modifique o quarto elemento do terceiro vetor para "DD"
11. Faça um data-frame (DF) com os mesmos vetores do exercício 8 e:
 - (a) Imprima um vetor com os membros da coluna 1.
 - (b) Imprima apenas a coluna 4.
 - (c) Imprima a linha 5.
 - (d) Nomeie as linha do data-frame.

7.3 Entrada de dados

1. Transforme o arquivo *DADOS_CURSO.xlsx*, disponível em URL, em .csv e leia-o no R
2. Com o banco de dados do exercício anterior faça o que se pede:
 - (a) Transforme a variável *Sexo* de "0" e "1" para "M" e "F"
 - (b) Calcule e apresente as estatísticas descritivas da variável *Peso* em forma de lista

7.4 Funções e gráficos

1. Carregue o banco de dados *frutas.csv* disponível em <https://sites.google.com/site/dadoscursor/m/frutas.csv?attredirects=0&d=1> e faça:
 - (a) Uma tabela para as Variáveis *Fruta* e *Cor*
 - (b) Um gráfico de barras e/ou setores para as mesmas variáveis
 - (c) Um gráfico de dispersão entre *Peso* e *Altura*
 - (d) Calcule a correlação entre as mesmas variáveis
 - (e) Faça um histograma para essas duas variáveis
 - (f) Ache a média da altura conforme a fruta
 - (g) Ache o máximo e o mínimo do peso conforme a fruta
2. Implemente as seguintes funções de probabilidade, construa o gráfico e exporte em pdf por linha de comando:
 - (a) Poisson com $\lambda = 3$;

Lembrando que se $X \sim \text{Pois}(\lambda)$, então:

$$P(X = x) = \frac{e^{-\lambda} \lambda^x}{x!}$$

(b) Binomial com $n = 10$ e $\pi = 0.5$;

Lembrando que se $X \sim \text{Bin}(n, \pi)$, então:

$$P(X = x) = \binom{n}{x} \pi^x (1 - \pi)^{n-x}$$

(c) Normal com $\mu = 0$ e $\sigma^2 = 1$;

Lembrando que se $X \sim N(\mu, \sigma^2)$, então:

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-1}{2\sigma^2}(x-\mu)^2}$$

3. Coloque no mesmo gráfico a densidade de duas Variáveis aleatórias X e Y , sabendo que $X \sim N(0, 1)$ e $Y \sim N(0, 2)$. Crie uma legenda identificando cada curva com uma cor.