# eRando em Ciênc1a de D4do\$

Luís Gustavo Schuck

16/08/2023

# Índice

В	em V Vers		5
Sc	bre c	o Autor	6
ı		6-76-	7
			8
	Con	nunidade	9
1	Intr	odução 1	0
	1.1	Iniciando o RStudio	10
	1.2	Console do R	10
		1.2.1 Executando Comandos	13
		1.2.2 Erros	13
	1.3	Trabalhando com Scripts	13
		1.3.1 Comentários	١7
	1.4	Objetos	۱7
		1.4.1 Vetores	۱7
	1.5	Criação de Objetos	18
	1.6	Operações com Objetos	18
		1.6.1 Coerção	19
	1.7	Exibindo Objetos	20
	1.8	Remoção de Objetos	20
	1.9	Usando Funções	20
			21
		1.9.2 Armazenando Retorno	22
2	Nor	neando Objetos 2	23
_	2.1	•	23
			24
			25
	2.2		26
3 Operador Pipe			
,	3.1	•	28 28
	0.1	<u> </u>	

	3.2	Placeholder						
4	Data Frames 32							
	4.1	O que são data frames ?						
		4.1.1 Criando Data Frames						
		4.1.2 Aplicar convenções de nomes						
	4.2	Atributos						
	4.3	Dimensões						
	4.4	Acessando Dados						
		4.4.1 Índices						
		4.4.2 Usando Nomes das Colunas						
		4.4.3 Filtrando Dados						
5	Оре	rações Lógicas 4:						
	5.1	Operadores Relacionais						
	5.2	Operadores Lógicos						
		5.2.1 Ou Exclusivo (Xor)						
	5.3	Precedência						
6	Ambientes 46							
•	6.1	Global Env						
	6.2	Ambiente de Pacotes						
	6.3	Ambientes "Pai"						
	0.0	6.3.1 Buscar Ambiente Pai (Recursivamente)						
	6.4	Criando Ambientes						
7	Funções 51							
•	7.1	Criando Funções						
	7.2	Função x Ambiente						
	1.2	7.2.1 Objetos no Ambiente da Função						
	7.3	Funções Genéricas						
8	Controles de Fluxo 56							
U	8.1	Introdução						
	8.2	If						
	8.3	Ifelse						
	8.4	If Else						
	8.5	Laço For						
	8.6	While						
	8.7	Repeat						
9	Gráficos 62							
•		Introducão						

П	Pacotes	65
	O que são pacotes?	66
10	Introdução a Pacotes	67
	10.1 Pacotes Instalados	67
	10.2 Pasta de Instalação	69
	10.3 Pacotes Disponíveis	70
	10.4 Dependências de Pacotes	70
	10.5 Instalação de Pacotes	71
111	RStudio	72
	O que é o RStudio?	73
11	Introdução ao RStudio	75
	11.1 Layout	75
	11.2 Console	75
	11.3 Output	76
	11.4 Environment	76
	11.5 Source	76
12	Menu Tools	78
	12.1 Opções	78
	12.1.1 Install Packages	79
	12.1.2 Check for Package Updates	79
	12.1.3 Version Control	82
	12.1.4 Terminal	82
	12.1.5 Background Jobs	82
	12.1.6 Global Options	82
IV	Ciência de Dados	84
	Definições	85
13	Trade-Off Viés x Variância	86
Co	onvenções	87
CU	Marcações no Texto	87
	Nomes de Objetos	87
	Status do Material	88
Re	eferências	89

# **Bem Vindo**

Este é um livro sobre a utilização da linguagem R em Ciência de Dados.

Este material é um projeto pessoal usado como fonte de consulta e aprendizado, sem compromisso com uma estrutura específica.

Muitas vezes o exposto aqui é a prática (para fixação e exploração) de conceitos apresentados em outros materiais. Assim todas as fontes utilizadas, mesmo que de forma subjetiva, são citadas.

### Versões

A versão do  ${\bf R}$  utilizada é a 4.3.0, Already Tomorrow.

A versão do **RStudio** é 2023.06.1+524, Mountain Hydrangea.

A versão do **Quarto** é 1.2.475.

Última atualização: 11/08/2023 - 23:17:11

# Sobre o Autor

### Status

Luís Gustavo Schuck é formado em Gestão Financeira (2013) pelo Centro Universitário Internacional - Uninter. Possui Especialização em Business Analytics (2021) pela Universidade Federal do Rio Grande do Sul - UFRGS, MBA em Administração e Finanças (2017) pelo Centro Universitário Internacional - Uninter e MBA em Gestão Bancária (2015) pelo Centro Universitário Leonardo da Vinci - Uniasselvi. Possui certificação ANBIMA CPA-10 (Certificação Profissional ANBIMA Série-10).

Atualmente é aluno do curso de **Análise e Desenvolvimento de Sistemas** pela Universidade Feevale e atua como **Analista na Unidade de Risco de Crédito** do Banco do Estado do Rio Grande do Sul (Banrisul). Utiliza R desde 2017.

Última atualização: 12/08/2023 - 15:56:19

# Parte I A Linguagem R

### Status

Conforme o **R Core Team** (R Core Team 2023c) 'R é uma linguagem e ambiente para computação estatística e gráficos'. Criada na década de 1990, R é uma **linguagem livre** e é distribuída sob a licença GPLv2.

Mantida pela R Foundation, atualmente (agosto/2023) é uma das linguagens mais usadas para Ciência de Dados e está entre as linguagens mais buscadas no Google como pode ser visto pelo PYPL - PopularitY of Programming Language.

Rank	Change	Language	Share	Trend
Num	Onlange	Language	onare	ITCIIG
1		Python	27.27 %	-0.5 %
2		Java	16.35 %	-1.6 %
3		JavaScript	9.52 %	+0.2 %
4		C#	6.92 %	-0.3 %
5		C/C++	6.55 %	-0.4 %
6		PHP	5.1 %	-0.5 %
7		R	4.34 %	-0.2 %
8		TypeScript	2.88 %	+0.3 %
9	<b>^</b>	Swift	2.3 %	+0.1 %
10	<b>V</b>	Objective-C	2.13 %	-0.1 %

Figura 1: 05/2023 - PYPL PopularitY of Programming Language

No IEEE Spectrum Top Programming Languages 2022 a linguagem R também aparece com bastante relevância.

### **R** Foundation

A R Foundation é uma organização sem fins lucrativos que tem como objetivo promover o desenvolvimento da linguagem R e ser ponto de referência para entidades que desejem interagir com a comunidade de desenvolvimento do R.

A R Foundation possui uma grande quantidade de apoiadores e doadores. Dentre os principais Patronos do R está a empresa **Posit**, anteriormente **RStudio**, que desenvolve o principal IDE para R, também chamado de **RStudio**.

Você pode ser um apoiador!

### Comunidade

Uma grande vantagem da liguagem R é a existência de uma grande comunidade de desenvolvimento, assim como uma gama enorme de conteúdos distribuídos através da Internet, muitos de forma livre e de fácil acesso. Abaixo alguns sites com conteúdos muito ricos sobre R:

- Análise de Dados Financeiros e Econômicos com o R Versão Online
- Introdução à Linguagem R: seus fundamentos e sua prática
- R Manuals
- Big Book of R
- R for Data Science
- Datacamp
- Statistics Globe
- R Charts
- Statistical tools for high-throughput data analysis

Última atualização: 14/08/2023 - 13:09:53

# 1 Introdução

### Status

Este capítulo tem como objetivo fornecer uma visão inicial mínima para que o usuário possa dar os primeiros passos na linguagem.

### 1.1 Iniciando o R...Studio

R é uma linguagem de programação e não está focada em oferecer uma interface sofisticada de interação com o usuário. Este papel fica por conta de outras ferramentas, como o **RStudio**, o **IDE** mais usado para a linguagem. Na prática "ninguém" usa o R puro para desenvolver seus projetos.

Desta forma usaremos o RStudio como ferramenta de desenvolvimento, pois ela irá nos fornecer muitas funcionalidades como preenchimento de código (code completion), janelas para instalar pacotes, janelas com arquivos de scripts, navegação por pastas, visualização e exportação de gráfico e claro comunicação direta com o R.

Ao longo deste livro serão usadas diversas funcionalidades do RStudio. Porém o foco será sempre no conteúdo, pois o detalhamento das principais funcionalidades do RStudio é tratada em seção específica.

Neste capítulo focaremos no painel **Console**, que "abriga" o R e no **Source**, que permite utilização de scripts.

### i Outros IDE's

Além do RStudio existem outras ferramentas para utilização em conjunto com R, entre elas: Jupyter, VS Code e RKWard.

### 1.2 Console do R

A tela inicial do R em si é um console, onde são passados comandos e seu interpratador os executa e, se for o caso, exibe saídas. O cursor fica posicionado ao lado do símbolo do *prompt* do R, >. Este símbolo indica que o sistema está pronto para receber novo comando.

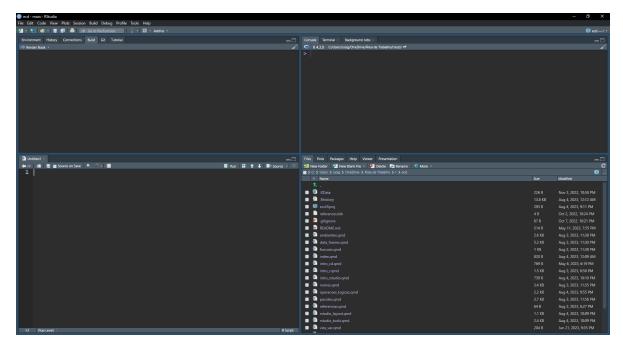


Figura 1.1: Tela Inicial do RStudio

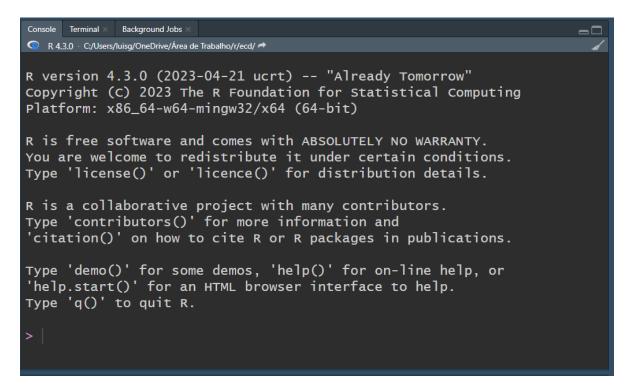


Figura 1.2: Aba Console

Figura 1.3: Aba Source

```
R version 4.3.0 (2023-04-21 ucrt) -- "Already Tomorrow"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

Figura 1.4: Tela Inicial

### 1.2.1 Executando Comandos

A tela inicial fornece algumas sugestões para consulta a dados sobre R, como licença da linguagem, citação, ajudas, etc. Usaremos como exemplo inicial o comando license(). Após a digitação do comando devemos confirmar com ENTER para que o R execute o comando informado e exiba na tela o resultado, no caso a licença da própria linguagem. Após a execução um novo sinal do prompt é exibido em aguardo de um possível próximo comando.

Podemos digitar q(), por exemplo, que é a função que efetua o encerramento do R.

Agora considere um cenário diferente, onde executamos o comando license() seguido do comando citation() (que mostra como deve ser feita a citação da Linguagem R). Conforme os comandos forem sendo passados, o console vai sendo preenchido com estes comandos e suas respectivas saídas. A medida que a tela vai ficando "cheia" os dados exibidos no topo vão "sumindo" para dar lugar aos mais recentes, na parte inferior.



### Buscando Comandos Anteriores

Para buscar comandos executados anteriormente, pode-se usar a seta para cima do teclado. Os comandos vão sendo apresentados do mais recente ao mais antigo.

### 1.2.2 Erros

Sempre que ocorrer algum erro na execução de um comando será exibida no console uma mensagem com o termo Error. Muitas vezes a mensagem de erro auxilia na identificação da causa do erro reportado. Abaixo um exemplo com erro retornado pelo R após a tentativa de execução de uma função inexistente (erro na digitação do comando).

```
citatin()
```

Error in citatin(): não foi possível encontrar a função "citatin"

# 1.3 Trabalhando com Scripts

Scripts são arquivos de texto que recebem códigos e conforme desejo do usuário são enviados ao console para execução. Na prática usar o console diretamente é útil para pequenas operações. No Rstudio você pode criar um script em File > New File > R Script. O arquivo de script será aberto no painel Source.

Para executar comandos de um arquivo de script você pode usar atalhos de teclado (Ctrl + Enter) ou através do botão Run no topo superior direito da aba Source. Ambas opções executam ou a linha corrente ou a parte do texto selecionada.

```
R version 4.3.0 (2023-04-21 ucrt) -- "Already Tomorrow"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors. Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
> license()
This software is distributed under the terms of the GNU General
Public License, either Version 2, June 1991 or Version 3, June 2007.
The terms of version 2 of the license are in a file called COPYING
which you should have received with
this software and which can be displayed by RShowDoc("COPYING").
Version 3 of the license can be displayed by RShowDoc("GPL-3").
Copies of both versions 2 and 3 of the license can be found
at https://www.R-project.org/Licenses/.
A small number of files (the API header files listed in
R_DOC_DIR/COPYRIGHTS) are distributed under the
LESSER GNU GENERAL PUBLIC LICENSE, version 2.1 or later.
This can be displayed by RShowDoc("LGPL-2.1"),
or obtained at the URI given.
Version 3 of the license can be displayed by RShowDoc("LGPL-3").
'Share and Enjoy.'
```

Figura 1.5: Tela Inicial - Licença

```
R version 4.3.0 (2023-04-21 ucrt) -- "Already Tomorrow"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
> license()
This software is distributed under the terms of the GNU General
Public License, either Version 2, June 1991 or Version 3, June 2007.
The terms of version 2 of the license are in a file called COPYING
which you should have received with
this software and which can be displayed by RShowDoc("COPYING").
Version 3 of the license can be displayed by RShowDoc("GPL-3").
Copies of both versions 2 and 3 of the license can be found
at https://www.R-project.org/Licenses/.
A small number of files (the API header files listed in
R_DOC_DIR/COPYRIGHTS) are distributed under the
LESSER GNU GENERAL PUBLIC LICENSE, version 2.1 or later.
This can be displayed by RShowDoc("LGPL-2.1"),
or obtained at the URI given.
Version 3 of the license can be displayed by RShowDoc("LGPL-3").
'Share and Enjoy.'
> q()|
```

Figura 1.6: Tela Inicial - Quit (sair)

```
This software is distributed under the terms of the GNU General Public License, either Version 2, June 1991 or Version 3, June 2007.
The terms of version 2 of the license are in a file called COPYING
which you should have received with
this software and which can be displayed by RShowDoc("COPYING").
Version 3 of the license can be displayed by RShowDoc("GPL-3").
Copies of both versions 2 and 3 of the license can be found at https://www.R-project.org/Licenses/.
A small number of files (the API header files listed in
R_DOC_DIR/COPYRIGHTS) are distributed under the
LESSER GNU GENERAL PUBLIC LICENSE, version 2.1 or later. This can be displayed by RShowDoc("LGPL-2.1"),
or obtained at the URI given.
Version 3 of the license can be displayed by RShowDoc("LGPL-3").
'Share and Enjoy.'
> citation()
To cite R in publications use:
  R Core Team (2023). _R: A Language and Environment for Statistical Computing_. R Foundation for
  Statistical Computing, Vienna, Austria. <a href="https://www.R-project.org/">https://www.R-project.org/</a>.
A BibTeX entry for LaTeX users is
  @Manual{,
    title = {R: A Language and Environment for Statistical Computing},
    author = {{R Core Team}},
organization = {R Foundation for Statistical Computing},
    address = {Vienna, Austria},
    year = {2023},
url = {https://www.R-project.org/},
We have invested a lot of time and effort in creating R, please cite it when using it for data analysis. See also 'citation("pkgname")' for citing R packages.
```

Figura 1.7: Tela Inicial - Atualização do Console



Figura 1.8: Source - Botão Run

#### 1.3.1 Comentários

R aceita comentários em seu código através do caractere sustenido (hashtaq), '#'. Qualquer texto após será ignorado pelo interpretador. Comentários são muito importantes para facilitar a leitura do código posteriormente.



### Porquê...

Em operações mais complexas procure colocar comentários que expliquem os motivos de se executar alguma operação e não o que o código está fazendo. Foque no 'porquê' de cada operação e não no 'o que'.

## 1.4 Objetos

Conforme o R Core Team(R Core Team 2023a, cap 3) 'as entidades nas quais R opera são tecnicamente conhecidas como objetos'. Existem diversos tipos de estrutura de dados em R, mas neste capítulo inicial serão usados objetos do tipo vetor, pois este é o objeto mais básico.



### i Variáveis

Muitas vezes objetos em R são chamados de variáveis, no sentido de que variáveis armazenam dados. Isto ocorre principalmente para objetos que armazenam um único valor, como um único número ou texto.

### 1.4.1 Vetores

Vetores são entidades que armazenam dados em posições (R Core Team 2023b, cap 2). Os vetores são ditos atômicos, pois seus dados são todos do mesmo tipo. Você pode pensar em um vetor como uma "local" onde serão armazenados dados. Os vetores podem ser de um dos 6 tipos abaixo:

Tabela 1.1: Tipos de Vetores

Tipo	Descrição	Exemplo
logical	lógico	TRUE
integer	número inteiro	1
double	número com ponto flutuante (real)	1.5
complex	número complexo	1i
character	texto $(strings)$	'R é software livre.'

Tipo	Descrição	Exemplo
$\overline{raw}$	bytes	

### 1.5 Criação de Objetos

Para criação de objetos no R são usados os operadores de atribuição, <- e =. O operador mais usado é o <-. Assim para criação de um objeto pode ser usado o código abaixo:

```
objeto1 <- 10 # atribui valor 10
```

Para criação de variáveis do tipo texto, devem ser usadas aspas, simples ou duplas. Aqui o **objeto2** foi criado com uso de aspas para que o R trate o valor como *character*.

```
objeto2 = 'texto' # atribui texto
```

## 1.6 Operações com Objetos

Objetos podem ser atualizados novamente com o operador <-. No exemplo abaixo vamos criar um vetor de nome **objeto3** com a funçãoc, que concatena valores, em conjunto com o operador :, que cria sequências de valores. Na sequência o **objeto3** será atualizado recebendo seu próprio conteúdo acrescido do valor 10.

```
objeto3 <- c(1:5)
objeto3

[1] 1 2 3 4 5

typeof(objeto3)

[1] "integer"

objeto3 <- objeto3 + 10</pre>
```

O vetor objeto3 foi criado com 5 posições, armazenando os valores de 1 a 5. Podemos acessar, por exemplo a terceira posição do vetor, através do operador de extração [ em combinação com o índice do vetor.

```
objeto3[3] # exibe terceiro elemento
[1] 13
  objeto3[3] + 10 # somando valor
[1] 23
```

Note que sem o operador de atribuição o valor da posição 3 do objeto3 não é atualizada, apenas exibda no console. Para atualizar seu valor usamos:

```
objeto3

[1] 11 12 13 14 15

objeto3[3] <- objeto3[3] + 30
objeto3

[1] 11 12 43 14 15
```

### 1.6.1 Coerção

Quando vetores recebem dados de um tipo diferente o R tenta fazer uma operação de **coerção**, transformando os valores a fim de "atender" a todos. Nem sempre esta operação é possível e ela muitas vezes altera o vetor original. No exemplo abaixo o valor da posição 1 do vetor será atualizado para receber a letra **A**. Como o vetor originalmente era do tipo **integer**, o R fará a conversão dos valores para tipo **character**. Desta forma operações matemáticas não serão mais possíveis sobre este vetor.

```
objeto3[1] <- 'A'
objeto3

[1] "A" "12" "43" "14" "15"

typeof(objeto3)

[1] "character"</pre>
```

```
objeto3 + 10
```

Error in objeto3 + 10: argumento não-numérico para operador binário

# 1.7 Exibindo Objetos

O R possui a função 1s que exibe os objetos existentes no ambiente.

```
ls()
[1] "objeto1" "objeto2" "objeto3"
```

# 1.8 Remoção de Objetos

Objetos podem ser removidos (excluídos) com a função rm.

```
rm(objeto2)
ls()

[1] "objeto1" "objeto3"
```

# 1.9 Usando Funções

O coração da linguagém R são suas funções. Através delas são feitas as mais diversas operações sobre os objetos. Basicamente funções devem ser usadas através de seus nomes e com os argumentos dentro de parênteses: funcao(argumento1, argumento2, ...).

Por exemplo, a função typeof exige a informação de um argumento (um objeto do R).

```
typeof(objeto1)
[1] "double"
```

O R processa esta função e devolve seu retorno, no caso qual o tipo do objeto1.

Pode-se visualizar o valor armazenado em um objeto com a função **print**, bem como exibir os objetos criados no R com a função **ls**. Note que a função **ls** aparentemente não exige argumentos.

```
print(objeto1)

[1] 10

ls()

[1] "objeto1" "objeto3"
```

### 1.9.1 Argumentos de Funções

As funções em R podem ter diversos argumentos e muitas vezes estes argumentos possuem valores definidos por padrão. Assim caso o usuário não informe nenhum valor para os argumentos da função esta usará os valores previamente definidos em seu código. Por este motivo a função 1s usada "sem" argumentos é processada normalmente. Mais detlahes em Funções.

Importante notar que os argumento possuem nomes e estes podem/devem ser usados. Voltemos a função typeof, ela possui apenas um argumento de nome x. Podemos usar a função typeof informando o nome do argumento e obtendo mesmo resultado anterior.

```
typeof(x = objeto1)
[1] "double"
```

Os argumentos podem ser omitidos e serão interpretados pelo R na ordem em que forem informados. Por exemplo a função rep.int retorna valores os valores indicados no argumento x n (times) vezes.

```
rep.int(5, 4)
[1] 5 5 5 5

rep.int(x = 5, times = 4) # de forma explícita
```

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

Perceba que os argumentos podem ser informados em ordem diversa, entretanto devem ser atribuidos de forma explícita. rep.int(times = 4, x = 5) é diferente de rep.int(4, 5).

```
rep.int(times = 4, x = 5)

[1] 5 5 5 5

rep.int(4, 5)
```

### [1] 4 4 4 4 4

### i Nota

Algumas funções não possuem argumentos e "apenas" executam seu código, não exigindo interação de entrada por parte do usuário, como por exemplo as funções Sys.Date() e Sys.time(), que retornam a data e data e hora respectivamente.

### 1.9.2 Armazenando Retorno

Para que o valor retornado por uma função seja armazenado, basta usar o operador de traibuição:

```
tipo <- typeof(objeto1)
print(tipo)

[1] "double"

typeof(tipo)</pre>
```

[1] "character"

Agora o objeto **tipo** armazena o valor retornado pelo função **typeof**, no caso o texto (*character*) "double".

Última atualização: 16/08/2023 - 18:17:58

# 2 Nomeando Objetos

Status

### 2.1 Nomes

A linguagem R aceita muitas possibilidades para nomeação de objetos. Inclusive podem ser criados objetos com espaços em seus nomes e até mesmo com caracteres especiais (desde que entre aspas ou crases).

```
x <- 10
.x <- 10
`nome com espaco` <- 55
'teste 1' <- 2</pre>
```

# Nomes Significativos

Escolha nomes intuitivos e que facilitem a identificação do conteúdo armazenado nos objetos.

Um objeto criado através do uso de aspas ou crases tem seu conteúdo acessado quando "chamado" com crases (backticks). Aspas são entendidas como sinalização para strings e assim não retornam o conteúdo do objeto e sim a própria string informada.

```
'teste 1' #retorna como string

[1] "teste 1"

# Exibe conteúdo do objeto
`teste 1`

[1] 2
```

### 2.1.1 Primeiro Caractere

Existem algumas regras para iniciar o nome dos objetos. Alguns caracteres "especiais" não podem ser usados, bem como os números.

Perceba que números podem ser usados nos nomes, desde que o primeiro caractere seja 'válido'. Mas o mesmo não ocorre com caracteres "especiais".

```
x55x <- 888

x55x

[1] 888

x$ <- 10

Error: <text>:1:4: unexpected assignment

1: x$ <-
```

Uma alternativa se dá mais uma vez com o uso de aspas ou crases. Com elas é possível 'burlar' estas limitações.

```
`teste @!&` <- 123456
`teste @!&`
```

[1] 123456

```
'55 teste @!&' <- 10
```

[1] 10

Apesar de possível, objetos com nomes mais complicados como os exemplificados acabam tornando a vida do programador um pouco mais difícil. Em geral, evite caracteres caracteres especiais e espaços nos nomes. Caso algum dado (bases de dados) seja carregado de arquivo externo com este tipo de caracteres, faça a uniformização dos nomes o quanto antes.

### 2.1.1.1 Objetos "Ocultos"

Objetos podem ser criados com "." no início de seus nomes desde que o segundo carctere seja uma letra. Estes são objetos "ocultos" e portanto não aparecem em um comando ls "puro", por exemplo. Tampouco são exibidos na aba Environment do RStudio. Para visualizá-los através da função ls deve ser usado o parâmetro all.names = T.

#### 2.1.2 Case Sensitive

R é uma linguagem *case sensitive*, ou seja, ela diferencia maiúsculas de minúsculas. Assim um objeto com nome de Teste é diferente teste, tesTe, TESTE...

```
teste <- 10
Teste <- 15
tesTe <- 20
TESTE <- 25
```

### ls()

```
[1] "55 teste @!&"
                       "nome com espaco" "teste"
                                                               "tesTe"
[5] "Teste"
                        "TESTE"
                                           "teste @!&"
                                                               "teste 1"
[9] "x"
                        "x55x"
```



### Campos de Tabelas

Campos (variáveis) de dados tabulados, como planilhas de Excel, seguem as mesmas regras. Este tipo de dado será tratado no capítulo sobre data frames.

# 2.2 Convenções

Conforme o seu código em R (e de outra linguagem qualquer) for crescendo você perceberá rapidamente a necessidade de identificar de forma intuitiva os objetos criados. Assim, é muito interessante a utilização de alguma convenção para facilitar sua vida. Existem diversas delas, como camelCase, snake\_case, SCREAMING\_SNAKE\_CASE, PascalCase, etc.

```
# camelCase
objetoTeste <- 'Teste camelCase'
# snake_case
objeto_teste <- 'Teste snake_case'
```

Um bom guia é o The tidyverse style guide. Tenha sempre em mente que sue código deve ser lido com facilidade no futuro e muitas vezes por outros usuários.

Neste material os nomes de objetos e derivados seguirão a tabela abaixo. Estas definições foram escolhidas a fim de uniformizar o conteúdo apresentado e se baseiam em experiência de uso e no Tidyverse Style Guide. Mais detalhes em Convenções.

Tipo Objeto	Convenção	Exemplo
Data.frame, tibble ou data.table	snake_case iniciado por df (data frame)	df_clientes
Variáveis de datasets Funções	SCREAMING_SNAKE_CASE camelCase iniciado por <b>fn</b> , sendo a primeira palavra após fn um verbo	df_clientes\$NOME_CLIENTE fnBuscarClientes
Demais (vetores, listas, etc.)	snake_case	nomes_cidades

# **?** Dica

Evite usar "." em nome de objetos, pois através do **ponto** o R acessa funções (métodos) de acordo com a classe do objeto. Usar o ponto pode causar certa confusão. Mais detalhes Funções.

Grolemund (2014)

R Core Team (2023a)

Wikipedia (2023)

Última atualização: 13/08/2023 - 22:38:42

# 3 Operador Pipe

Status

## 3.1 Introdução

Muitas vezes seu código demanda muitas transformações e acaba ficando muito verboso e de dificíl entendimento. Uma forma de facilitar a compreensão em torno das operações em sequência é criar um fluxo em que as operações vão sendo efetuadas em sequência, onde as entradas são as saídas do passo anterior.

O operador |> (pipe) existe com este intuito, organizar as operações em um fluxo contínuo. O pipe foi implementado a partir da versão 4.1.0 do R e passa um valor para uma função. Os dados são passados do lado esquerdo (lfs - left hand side) para o lado direito (rhs - right hand side). O valor do lado esquerdo (lhs) é passado como o primeiro argumento da função do lado direito (rhs).

Vejamos um exemplo simplificado onde o vetor que possue números de 1 até 20 é passado para a função head. Com o uso do |> o vetor é passado como **primeiro argumento** da função head e esta por sua vez exibe os seis primeiros elementos.

```
c(1:20) |> head()
[1] 1 2 3 4 5 6
```

O código acima é equivalente a:

```
head(c(1:20))
```

[1] 1 2 3 4 5 6

Caso se deseje alterar o número de elementos, basta usar o argumento n.

```
c(1:20) \mid > head(n = 10)
```

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

Equivalente a:

```
head(c(1:20), n = 10)
[1] 1 2 3 4 5 6 7 8 9 10
```

Vejamos um outro exemplo, um pouco mais realista e complexo: usar a base **mtcars** e a partir desta selecionar casos em que o campo **mpg** seja maior do que 10 e após criar uma variável chamada **media\_hp**, que será a média a partir do campo **hp**. Poderia ser feito algo do tipo:

```
df_mtcars <- subset(mtcars, mpg > 10)
media_hp <- mean(df_mtcars$hp)
media_hp</pre>
```

#### [1] 146.6875

Mesmo sendo um processo pequeno com apenas 2 operações bastante corriqueiras, ler o código já se torna enfadonho, para dizer o mínimo. Também não fica claro, em uma passada de olhos, se as operações possuem relação entre si.

Imagine agora criar as mesmas operações de forma "concatenada" em que uma transformação é passada para a seguinte até que se chegue ao final do fluxo. Em linguagem "humana" algo do tipo:

data frame filtrar casos selecionar variável calcular média

Em R:

```
mtcars |>
  subset(mpg > 10) |>
  subset(select = hp, drop = T) |>
  mean()
```

[1] 146.6875

```
# ou de forma mais sucinta
mtcars |>
subset(mpg > 10, select = hp, drop = T) |>
```

```
mean()
```

```
[1] 146.6875
```

Este código é equivalente ao anterior, porém aqui fica mais claro que todas as transformações foram feitas a fim de obter o valor da média de hp dos casos desejados (mpg > 10). Para fazer a atribuição do resultado em uma variável basta, como de costume, ao início ou ao final usar o operador de atribuição <-.

```
media_hp <- mtcars |>
    subset(mpg > 10) |>
    subset(select = hp, drop = T) |>
    mean()

media_hp

[1] 146.6875

# ou de forma menos usual
mtcars |>
    subset(mpg > 10) |>
    subset(select = hp, drop = T) |>
    mean() -> media_hp

media_hp

[1] 146.6875
```

### 3.2 Placeholder

A partir da versão **4.2.0** o *pipe* passou a ter um *placeholder* (símbolo \_) que serve para que o valor **lfs** seja passado para outro argumento que não o primeiro da função **rhs**.

```
8 |> head(c(1:20), n = _)
[1] 1 2 3 4 5 6 7 8
```

Equivalente a:

```
head(c(1:20), n = 8)
```

### [1] 1 2 3 4 5 6 7 8

A partir da versão **4.3.0** o *placeholder* também pode ser utilizado para operações de extrações com [. Replicando o exemplo do cálculo de **media\_hp**, porém agora fazendo a extração da variável **hp** que é retornada como um vetor e passada para a função **mean**.

```
media_hp <- mtcars |>
    subset(mpg > 10) |>
    _$hp |>
    mean()

media_hp

[1] 146.6875
```

R Core Team (2023d)

Wickham (2023/04/21)

Última atualização: 14/08/2023 - 20:55:44

# 4 Data Frames

Status

# 4.1 O que são data frames ?

Conforme o próprio manual do R, data frame é a estrutura que imita de forma mais próxima um dataset do SAS ou SPSS. De forma resumida um data frame é uma estrutura tabular com colunas (variáveis, atributos, etc) e linhas (registros, casos, observações, instâncias, etc). Diferente de uma matriz um data frame pode ter diferentes tipos de dados em suas colunas.

Um data frame possui todas as colunas com o mesmo tamanho (quantidade de regitros). A classe de um objeto data frame possui o nome data.frame. Abaixo pode ser visualizada a classe do data frame iris (muito usado em exemplos em Ciência de Dados) e também as primeiras linhas com o comando head.

```
class(iris)
```

[1] "data.frame"

head(iris)

Sepal.Length Sepal.Width Petal.Length Petal.Width Species 5.1 3.5 1.4 0.2 setosa 2 4.9 3.0 1.4 0.2 setosa 3 4.7 3.2 1.3 0.2 setosa 4 4.6 3.1 1.5 0.2 setosa 5 5.0 3.6 1.4 0.2 setosa 6 5.4 3.9 1.7 0.4 setosa

### 4.1.1 Criando Data Frames

Objetos da classe data.frame podem ser criados com a função data.frame.

Aqui serão usadas as convenções de nomes conforme capítulos Nomeando Objetos e Convenções.

```
df_exemplo <- data.frame(</pre>
    VAR_A = c(1:5),
    VAR_B = c(101:105)
  df_exemplo
 VAR_A VAR_B
1
      1
          101
      2
2
        102
3
      3
        103
4
      4
        104
5
      5
          105
```

### 4.1.2 Aplicar convenções de nomes

Para continuar os próximos tópicos vamos trabalhar com um data frame (df\_iris) criado a partir do data frame iris. Faremos ajustes nos nomes deste data frame.

```
# criar data frame df_iris
df_iris <- iris
# mudar nomes para maiusculas
names(df_iris) <- toupper(names(df_iris))
# substituir '.' por '_'
names(df_iris) <- gsub(names(df_iris), pattern = "\\.", replacement = "_")
class(df_iris)

[1] "data.frame"
head(df_iris)</pre>
```

	SEPAL_LENGTH	${\tt SEPAL\_WIDTH}$	${\tt PETAL\_LENGTH}$	PETAL_WIDTH	SPECIES
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

### 4.2 Atributos

Os atributos "padrão" de um **data frame** são: names, class e row.names. É possível acessálos com a função attributes. O atributo names também pode ser obtido com a função names.

```
attributes(df_iris)
```

#### \$names

[1] "SEPAL\_LENGTH" "SEPAL\_WIDTH" "PETAL\_LENGTH" "PETAL\_WIDTH" "SPECIES"

#### \$class

[1] "data.frame"

### \$row.names

```
[1]
        1
            2
                 3
                     4
                         5
                             6
                                  7
                                      8
                                          9
                                              10
                                                  11
                                                      12
                                                          13
                                                               14
                                                                            17
                                                                                18
                                                                   15
                                                                       16
 [19]
       19
           20 21
                    22
                        23
                            24
                                 25
                                     26
                                              28
                                                  29
                                                      30
                                                           31
                                                                   33
                                                                       34
                                                                            35
                                                                                36
                                         27
                                                               32
 [37]
                            42
       37
           38
               39
                    40
                        41
                                 43
                                     44
                                                  47
                                                      48
                                                           49
                                                                       52
                                         45
                                             46
                                                               50
                                                                   51
                                                                            53
                                                                                54
 [55]
       55
           56
               57
                    58
                        59
                            60
                                 61
                                     62
                                         63
                                              64
                                                  65
                                                      66
                                                           67
                                                               68
                                                                   69
                                                                       70
                                                                            71
                                                                                72
                                 79
 [73]
       73
           74
               75
                    76
                        77
                            78
                                             82
                                                  83
                                                      84
                                                           85
                                                                   87
                                                                       88
                                                                            89
                                                                                90
                                     80
                                         81
                                                               86
           92
               93
                    94
                        95
                            96
                                 97
                                     98
                                         99 100 101 102 103 104 105 106 107 108
[109] 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
[127] 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
[145] 145 146 147 148 149 150
```

```
names(df_iris)
```

[1] "SEPAL\_LENGTH" "SEPAL\_WIDTH" "PETAL\_LENGTH" "PETAL\_WIDTH" "SPECIES"

### 4.3 Dimensões

A função dim retorna as dimensões de um data frame (linhas e colunas). Estes dados também podem ser obtidos com as funções nrow e ncol.

### 4.4 Acessando Dados

### 4.4.1 Índices

Como a estrutura de um **data frame** é organizada em linhas e colunas, podemos acessar os dados utilizando colchetes ([]): base[linha, coluna]. Podem ser usados intervalos de índices com o operador :.

```
# Acessar primeira linha e segunda coluna (Sepal.Width)
df_iris[1, 2]

[1] 3.5

# Acessar linhas 1 até 3 e a segunda coluna
df_iris[1:3, 2]

[1] 3.5 3.0 3.2
```

Apesar de ser possível, utilizar o índice faz com que a referência seja relativa, ou seja, a variável '1' pode mudar caso o **data frame** seja editado. Por exemplo, caso em algum momento anterior a variável **PETAL\_LENGTH** tenha sido excluída, uma nova variável assumirá o índice 1. Além disto, no momento da leitura do código por um usuário não fica claro qual variável está sendo acessada.

### 4.4.2 Usando Nomes das Colunas

Existem diversas outras formas para acessar dados de um data frame, inclusive utilizando o nome da coluna de forma explícita.

```
# Acessar primeira linha e segunda coluna (pelo nome)
df_iris[1:3, 'SEPAL_WIDTH']
[1] 3.5 3.0 3.2
```

Uma forma bastante comum é através da utilização do operador \$ para acessar a coluna pelo seu nome.

```
# Acessar primeira linha e segunda coluna
df_iris[1, ]$SEPAL_WIDTH

[1] 3.5

# Acessar linhas 1 até 3 e a segunda coluna
df_iris[1:3, ]$SEPAL_WIDTH

[1] 3.5 3.0 3.2
```

### 4.4.3 Filtrando Dados

Digamos que se deseje acessar apenas dados que cumpram determinada condição. Para isto, na seleção das linhas do **data frame**, deve ser informada condição lógica na forma abaixo:

```
# Retorna valores de Speal.Width onde Petal.Length for maior do que 6
x <- df_iris[df_iris$PETAL_LENGTH > 6, 'SEPAL_WIDTH']
y <- df_iris[df_iris$PETAL_LENGTH > 6.5, ]$SEPAL_WIDTH
```

```
X
```

[1] 3.0 2.9 3.6 3.8 2.6 2.8 2.8 3.8 3.0

У

[1] 3.0 3.8 2.6 2.8

```
# Função que compara os objetos
identical(x, y)
```

[1] FALSE

O retorno é dado pelas linhas em que a variável **PETAL\_LENGTH** atende as condições declaradas. Este teste retorna um vetor de valores lógicos, e os valores TRUE são os que "permanecem". Abaixo outro exemplo:

```
head(df_iris$PETAL_LENGTH) > 1.4
```

[1] FALSE FALSE FALSE TRUE FALSE TRUE

Aplicando este vetor de valores lógicos, o R entende que as posições correspondentes a TRUE devem ser mantidas. No exemplo abaixo, as posições (linhas) 4 e 6 atendem a condição especificada, portanto apenas estas serão selecionadas.

```
df_iris2 <- head(df_iris)
filtro <- head(df_iris2$PETAL_LENGTH) > 1.4
filtro
```

[1] FALSE FALSE FALSE TRUE FALSE TRUE

```
df_iris2[filtro, 'SEPAL_WIDTH']
```

[1] 3.1 3.9

Equivalente ao comando abaixo:

```
df_iris2[c(4, 6), 'SEPAL_WIDTH']
[1] 3.1 3.9
```

#### 4.4.3.1 Classes de retorno

Os filtros em data frames usados com \$ ou [ ] (com apenas 1 variável) retornam vetores e não data frames. Desta forma se perde a classe e a estrutura tabular característica do data frame original.

```
class(df_iris[1:3, 1])

[1] "numeric"

class(df_iris[1:3, 'SEPAL_WIDTH'])

[1] "numeric"
```

Entretanto, sendo selecionadas mais de uma coluna, a classe retornada segue sendo data.frame.

```
class(df_iris[1:3, c("SEPAL_LENGTH", "SEPAL_WIDTH")])
[1] "data.frame"

class(df_iris[1:3, 1:2])
[1] "data.frame"
```

#### 4.4.3.2 Função Subset

A função subset permite efetuar filtro em um data frame e muitas vezes oferece uma forma mais organizada visualmente, principalmente quando em filtros com muitas condiões. Uma outra vantagem é que a função subset retorna faz a seleção em um data.frame e retorna um data frame, mesmo com a seleção de apenas 1 variável.

Esta função também permite seleção de colunas a serem mantidas. Note que a função subset não demanda que o **data frame** seja referenciado antes das variáveis e também aceita os nomes das variáveis sem aspas. Isto torna o código mais legível.

```
class(subset(df_iris, select = SEPAL_WIDTH))
[1] "data.frame"
  df_mtcars <- mtcars</pre>
  # mudar nomes para maiusculas
  names(df_mtcars) <- toupper(names(df_mtcars))</pre>
  subset(x = df_mtcars, # dados
         subset = MPG > 25, # filtro
         select = c(MPG, CYL, HP)) # colunas
                MPG CYL
                          HP
Fiat 128
               32.4
                      4
                          66
                       4 52
Honda Civic
               30.4
Toyota Corolla 33.9
                      4 65
Fiat X1-9
               27.3
                      4 66
Porsche 914-2 26.0
                       4 91
Lotus Europa
               30.4
                       4 113
```

Usando um filtro um pouco mais complexo e sem inserir o nome dos argumentos da função (x, subset e select):

MPG CYL HP
Porsche 914-2 26.0 4 91
Lotus Europa 30.4 4 113

R Core Team (2023b)

Última atualização: 13/08/2023 -  $22{:}39{:}20$ 

# 5 Operações Lógicas

#### Status

A linguagem R oferece uma série de operadores para utilização em testes lógicos.

## 5.1 Operadores Relacionais

Operador	Função		
>	Maior que		
<	Menor que		
>=	Maior ou igual a		
<=	Menor ou igual a		
==	Igual a		
!=	Diferente de		

5 > 6

[1] FALSE

5 <= 6

[1] TRUE

5 == 6

[1] FALSE

5!= 6

[1] TRUE

## 5.2 Operadores Lógicos

Operador	Função
!	Negação
&	E
	Ou
xor	Ou exclusivo
isTRUE	Testa se verdadeiro
is FALSE	Testa se falso

```
!FALSE
[1] TRUE
!TRUE
[1] FALSE
[5 > 6
[1] FALSE
!5 > 6
[1] TRUE

    isTRUE(5 > 6)
[1] FALSE

    isFALSE(5 > 6)
```

### 5.2.1 Ou Exclusivo (Xor)

O operador xor fornece saída verdadeira quando apenas um dos valores for verdadeiro.

```
# Falso XOR Falso = Falso
xor(5 > 6, 6 > 9)
```

#### [1] FALSE

```
# Verdadeiro XOR Verdadeiro = Falso
xor(5 > 4, 6 > 5)
```

#### [1] FALSE

```
# Verdadeiro XOR Falso = Verdadeiro
xor(5 > 4, 6 > 9)
```

#### [1] TRUE

```
# Falso XOR Verdadeiro = Falso
xor(5 > 6, 6 > 5)
```

[1] TRUE

#### 5.3 Precedência

Na utilização de testes lógicos é importante observar a ordem (precedência) de aplicação dos operadores. O uso de parênteses altera a o escopo de aplicação dos operadores.

Tabela 5.3: Precedência de Operadores

Ordem	Operador
1	<, >, <=, >=, ==, !=
2	!
3	&
4	

Abaixo alguns testes.

```
# Falso E Falso = Falso
  5 > 6 & 4 > 5
[1] FALSE
  # Verdadeiro E Verdadeiro = Verdadeiro}
  !5 > 6 & !4 > 5
[1] TRUE
  # Verdadeiro E Falso = Falso
  !5 > 6 & 4 > 5
[1] FALSE
  # Negação de( Falso E Falso = Falso) = Verdadeiro
 !(5 > 6 & 4 > 5)
[1] TRUE
 # Falso OU Falso = Falso
 5 > 6 | 4 > 5
[1] FALSE
  # Verdadeiro OU Verdadeiro = Verdadeiro
  !5 > 6 | !4 > 5
[1] TRUE
  # Verdadeiro OU Falso = Verdadeiro
  !5 > 6 | 4 > 5
```

[1] TRUE

```
# Negação de( Falso OU Falso = Falso) = Verdadeiro
!(5 > 6 | 4 > 5)
```

[1] TRUE

R Core Team (2023c)

Última atualização: 12/08/2023 -  $20{:}27{:}56$ 

## 6 Ambientes

Status

#### 6.1 Global Env

O Global Env é o ambiente "atual" do usuário. É nele que ficam armazenadas, por padrão, as funções criadas pelos usuários por exemplo. Ele pode ser "visualizado" com os comandos abaixo:

```
globalenv()

<environment: R_GlobalEnv>

.GlobalEnv

<environment: R_GlobalEnv>
Os objetos presentes no almbiente desejado podem ser visualizados com a função ls.

variavel <- 5
ls(globalenv())

[1] "variavel"

ls()</pre>
```

#### 6.2 Ambiente de Pacotes

Os pacotes também possuem ambientes e podemos listar seu "conteúdo" com a função ls. Abaixo usando ls para mostrar os 10 primeiros elementos presentes no ambiente do pacote data.table.

```
library(data.table)
  as.environment("package:data.table")
<environment: package:data.table>
attr(,"name")
[1] "package:data.table"
attr(,"path")
[1] "C:/Users/luisg/AppData/Local/R/win-library/4.3/data.table"
  ls(as.environment('package:data.table'))[1:10]
                     "%chin%"
 [1] "%between%"
                                      "%flike%"
                                                      "%ilike%"
 [5] "%inrange%"
                     "%like%"
                                                      "address"
 [9] "alloc.col"
                     "as.data.table"
```

#### 6.3 Ambientes "Pai"

Cada ambiente possui um ambiente de nível superior associado, com exceção do R\_EmptyEnv.

```
# Ambiente superior ao GlobalEnv
parent.env(.GlobalEnv)

<environment: package:data.table>
attr(,"name")
[1] "package:data.table"
attr(,"path")
[1] "C:/Users/luisg/AppData/Local/R/win-library/4.3/data.table"

# Ambiente superior ao do apcote stats e base
parent.env(as.environment("package:stats"))
```

```
<environment: package:graphics>
attr(,"name")
[1] "package:graphics"
attr(,"path")
[1] "C:/Program Files/R/R-4.3.0/library/graphics"

parent.env(as.environment("package:base"))

<environment: R_EmptyEnv>
```

#### 6.3.1 Buscar Ambiente Pai (Recursivamente)

Abaixo função que busca recursivamente os ambientes e seus 'pais' até que se chege no 'último' ambiente, o R\_EmptyEnv.

```
fnBuscarEnvsPai <- function(ambiente, nivel = 1){</pre>
     if(environmentName(ambiente) == "R_EmptyEnv"){
     return ('Ambiente informado é R_EmptyEnv. Fim da busca.')
     }
     marcacao <- ''
     for (i in 1:nivel){
       marcacao <- paste0(' ', marcacao)</pre>
     }
     writeLines(paste0(marcacao, '|-- ', environmentName(parent.env(ambiente))))
     nivel <- nivel + 1
     fnBuscarEnvsPai(parent.env(ambiente), nivel = nivel)
 fnBuscarEnvsPai(globalenv())
|-- package:data.table
|-- tools:quarto
 |-- tools:quarto
  |-- package:stats
   |-- package:graphics
```

```
|-- package:grDevices
|-- package:utils
|-- package:datasets
|-- package:methods
|-- Autoloads
|-- base
|-- R_EmptyEnv
```

[1] "Ambiente informado é R\_EmptyEnv. Fim da busca."

#### 6.4 Criando Ambientes

Em R é possível que se faça a criação de novos ambientes.

```
amb1 <- new.env()
amb1
<environment: 0x0000019d4bd1d3b8>
parent.env(amb1)
```

<environment: R\_GlobalEnv>

Objetos criados dentro de um ambiente podem ser acessados através do operador \$ após o nome do ambiente. Também é possível utilizar a função 1s com o nome do ambiente desejado para que sejam listados seus objetos.

```
# Objeto x do amb1
amb1$x <- 10
amb1$y <- 99

# Objeto x do GlobalEnv
x <- 15</pre>
```

[1] 15

```
amb1$x

[1] 10

ls(amb1)

[1] "x" "y"

amb1$x * amb1$y

[1] 990
```

Grolemund (2014)

Dowle e Srinivasan (2023)

Mastropietro (2019)

Última atualização: 12/08/2023 -  $20{:}27{:}12$ 

# 7 Funções

Status

### 7.1 Criando Funções

Funções podem ser criadas através do comando function.

```
fnSomar <- function(param1, param2) {
   param1 + param2
}
fnSomar(5, 8)</pre>
```

Para visualizar o código de uma função podemos usar seu nome sem os parênteses.

```
fnSomar

function(param1, param2) {
  param1 + param2
}
```

### 7.2 Função x Ambiente

As funções possuem seus próprios ambientes. Abaixo uma função criada para exibir seu ambiente e seu ambiente 'pai'.

```
fnExibirEnvs <- function() {
  print('Ambiente atual:')
  print(environment())</pre>
```

#### 7.2.1 Objetos no Ambiente da Função

Objetos que são criados dentro de uma função existem apenas dentro do ambiente desta função. Abaixo um exemplo de variável criada dentro do ambiente da função e que não é acessível no GlobalEnv.

```
fnTeste <- function(){
    y <- 15
    x <- 80
    ls()
}
fnTeste()

[1] "x" "y"</pre>
```

Error in eval(expr, envir, enclos): objeto 'y' não encontrado

Objetos que existam no ambiente corrente não são alterados caso por estarem dentro do ambiente de fuma função. A variável x é inicializada com valor 10 no ambiente corrente. Ela pode ser acessada pela função memso não sendo informada em algum argumento.

```
x <- 10
```

```
fnTeste2 <- function(){
   y <- 15
   x + y
}

fnTeste2()

[1] 25

y

Error in eval(expr, envir, enclos): objeto 'y' não encontrado
   x

[1] 10</pre>
```

Entretando, caso a variável x seja alterada no ambiente da função ela não é alterada no ambiente corrente.

```
x <- 10

fnTeste3 <- function(){
   y <- 15
   x <- 80
   x + y
}

fnTeste3()

[1] 95

x</pre>
```

#### 7.2.1.1 Operador <<-

Usando o operador de atribuição <<- é possível alterar objetos que estejam fora do ambiente de uma função. Neste caso a variável x é atualizada no Environment que está acima do Environment da função. A variável y continua não existindo fora da função, porém agora a variável x é atualizada tanto no ambiente da função como no ambiente acima deste.

```
ls(envir = globalenv())
[1] "fnExibirEnvs" "fnSomar"
                                     "fnTeste"
                                                      "fnTeste2"
                                                                       "fnTeste3"
[6] "x"
  X
[1] 10
  fnTeste4 <- function(){</pre>
    y <- 15
    x <<- 80
     x + y
  fnTeste4()
[1] 95
  у
Error in eval(expr, envir, enclos): objeto 'y' não encontrado
  \mathbf{x}
[1] 80
```

Apesar de, neste caso, produzirem o mesmo retorno, as funções fnTeste3 e fnTeste4 impactam de formas distintas o ambiente do R.

## 7.3 Funções Genéricas

Grolemund (2014)

Última atualização: 12/08/2023 -  $21{:}54{:}00$ 

## 8 Controles de Fluxo

Status

## 8.1 Introdução

Assim como outras linguagens de programação R oferece uma série de operadores para controle de fluxo de código.

#### i Nota

Controles de fluxo são declarações usadas na linguagem, mas **não são funções**.

#### 8.2 If

O controle **if** é a estrutura de controle mais básica que tomada de decisão e "direcionamento" de código. Em caso negativo do teste lógico nenhuma operação é executada.

```
x <- 5

# Códigos equivalentes
if(x > 4) print('x é maior do que quatro')

[1] "x é maior do que quatro"

if(x > 4) { print('x é maior do que quatro')}

[1] "x é maior do que quatro"

if(x > 4) {
    print('x é maior do que quatro') }
```

```
[1] "x é maior do que quatro"

if(x > 4) { print('x é maior do que quatro')}

[1] "x é maior do que quatro"

if(x > 4) {
  print('x é maior do que quatro')}
} # o mais organizado
```

[1] "x é maior do que quatro"

Note que se o teste não retornar TRUE ou FALSE o R reportará erro.

```
x \leftarrow NA
if (x > 4) print('x é maior do que quatro')
```

Error in if (x > 4) print("x é maior do que quatro"): valor ausente onde TRUE/FALSE necessár

#### 8.3 Ifelse

R possui a **função ifelse**, que apesar de não ser para controle de fluxo, possui lógica de uso muito semelhante ao **if** e por este motivo será tratada neste capítulo. Esta função efetua teste em valor de entrada e define um valor a ser retornado caso verdadeiro e outro caso falso.

O retorno de ifelse possui o mesmo formato da estrutura informada no argumento test. Esta função pode ser usada para atribuição em data frames de forma mais sucinta.

Vejamos um exemplo:

```
df_mtcars6 <-
  mtcars |>
  subset(select = c('hp', 'mpg', 'cyl')) |>
  head()

df_mtcars6
```

```
hp mpg cyl
Mazda RX4
                  110 21.0
                             6
Mazda RX4 Wag
                  110 21.0
                             6
Datsun 710
                  93 22.8
                             4
Hornet 4 Drive
                  110 21.4
                             6
Hornet Sportabout 175 18.7
                             8
Valiant
                  105 18.1
  df_mtcars6[df_mtcars6$hp > 100, 'RESULTADO'] <-</pre>
    df_mtcars6[df_mtcars6$hp > 100, ]$mpg
  df_mtcars6[df_mtcars6$hp <= 100, 'RESULTADO'] <-</pre>
    df_mtcars6[df_mtcars6$hp <= 100,]$cyl</pre>
  df_mtcars6
                   hp mpg cyl RESULTADO
Mazda RX4
                  110 21.0
                             6
                                     21.0
                  110 21.0
                                     21.0
Mazda RX4 Wag
                             6
Datsun 710
                  93 22.8
                             4
                                    4.0
Hornet 4 Drive
                  110 21.4
                            6
                                     21.4
Hornet Sportabout 175 18.7
                             8
                                    18.7
```

105 18.1

6

18.1

Valiant

	hp	mpg	cyl	${\tt RESULTADO}$	RESULTADO2
Mazda RX4	110	21.0	6	21.0	21.0
Mazda RX4 Wag	110	21.0	6	21.0	21.0
Datsun 710	93	22.8	4	4.0	4.0
Hornet 4 Drive	110	21.4	6	21.4	21.4
Hornet Sportabout	175	18.7	8	18.7	18.7
Valiant	105	18.1	6	18.1	18.1

#### 8.4 If Else

O if else pode ser usado para inserir uma ação após o retorno negativo do teste feito pelo if.

```
x <- 3
if(x > 4) {
  print('x é maior do que quatro')
} else {
  print('x não é maior do que quatro')
}
```

[1] "x não é maior do que quatro"

Veja que podem ser usadas muitas declaração else em sequência.

```
x <- 3
if(x > 3) {
  print('x é maior do que três')
} else if (x < 3){
  print('x é menor do que três')
} else if (x == 3){
  print('x é igual a três')
}</pre>
```

[1] "x é igual a três"

### 8.5 Laço For

Um laço **for** é uma estrutura que efetua uma determinada quantidade de passos de acordo com a sequência informada. a declaração deve ser feita no formato: for(x in seq), sendo x a variável que será atualizada a cada iteração iniciando no 'rimeiro valor informado em seq e ecerrando no último. Um exemplo:

```
for(x in 1:5){
  print(paste('Iteração:', x))
}
```

```
[1] "Iteração: 1"
[1] "Iteração: 2"
[1] "Iteração: 3"
[1] "Iteração: 4"
[1] "Iteração: 5"
```

Caso se deseje mudar o incremento a cada passo pode ser usada a função seq. Também é possível usar um passo decrescente.

```
for(x in seq(2, 1, -0.25)) {
   print(paste('Valor de x:', x))
}

[1] "Valor de x: 2"
[1] "Valor de x: 1.75"
[1] "Valor de x: 1.5"
[1] "Valor de x: 1.25"
[1] "Valor de x: 1"
```

No eemplo acima, x é inicializado com valor 1 e vvai sendo incrementado em 1 unidade ao início do próximo passo. Um laço **for** também pode fazer iterações sobre vetores com texto, por exemplo.

```
for(i in c('São Paulo', 'Rio de Janeiro', 'Porto Alegre')){
   print(paste('Cidade atual:', i))
}

[1] "Cidade atual: São Paulo"
[1] "Cidade atual: Rio de Janeiro"
[1] "Cidade atual: Porto Alegre"
```

No laço **for** a sequência no qual será feita a iteração é considerada antes de se iniciar o laço, assim mesmo se houver alguma alteração nesta sequência em um dos passos esta alteração não impactará na execução.

```
x <- 3
for(i in 1:x) {
   x <- x + 2
   print(x)
}</pre>
```

- [1] 5
- [1] 7
- [1] 9

## 8.6 While

### i For x While

Um laço for é utilizado quando se tem uma sequência definida de passos. Caso se deseje executar alguma operação até o atendimeto de uma condição, use **while**.

## 8.7 Repeat

Última atualização: 16/08/2023 - 18:18:50

## 9 Gráficos

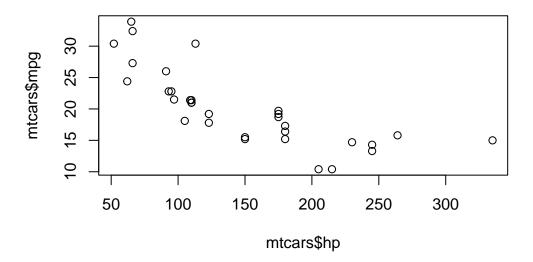
Status

## 9.1 Introdução

R oferece uma série de funções nativas para criação de gráficos. Estas funções possuem muitos parâmetros que permitem melhorar visualamente as apreesentações dos gráficos.

Abaixo um exemplo de um gráfico de pontos:

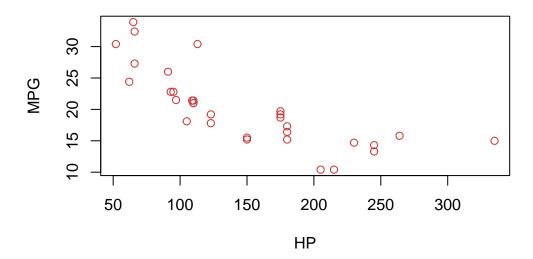
plot(mtcars\$hp, mtcars\$mpg)



Customizando:

```
plot(mtcars$hp, mtcars$mpg,
     col ='brown3', main = 'Scatter Plot',
     xlab = 'HP', ylab = 'MPG')
```

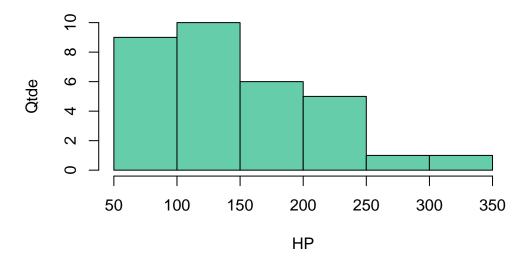
## **Scatter Plot**



Abaixo um exemplo de um histograma

```
hist(mtcars$hp, col ='aquamarine3',
    main = 'Histograma', xlab = 'HP',
    ylab = 'Qtde')
```





\_\_\_\_

Última atualização: 14/08/2023 -  $20{:}54{:}01$ 

Parte II

**Pacotes** 

#### Status

#### O que são pacotes?

Um pacote em R é basicamente um conjunto de funções e/ou funcionalidades criadas por terceiros que "expandem" o poder da linguagem. A principal opção para instalação de pacotes é através do CRAN. O CRAN é um repositório que contém milhares de pacotes (19904 em 16/08/2023). Nele também podem ser encontrados pacotes em suas versões "antigas". Caso algum pacote não esteja hospedado no CRAN, ele também pode ser instalado, diretamente do arquivo fornecido pelo desenvolvedor do pacote por exemplo (muitos distribuem através do Github).

Existem alguns pacotes "especiais" em R que compõem a própria linguagem. Estes pacotes possuem suas versões idênticas à da linguagem e são "classificados" com prioridade "base". Assim quando se faz a instalação da linguagem R, muitos pacotes também são instalados.

R Core Team (2023c)

Última atualização: 12/08/2023 - 20:39:38

## 10 Introdução a Pacotes

Status

#### 10.1 Pacotes Instalados

Podemos ver os pacotes instalados com o comando installed.packages:

```
# Exibindo 5 primeiros
as.data.frame(installed.packages())$Package[1:5]

[1] "askpass" "backports" "base64enc" "bit" "bit64"
```

A função installed.packages retorna uma série de informações a respeito dos pacotes. Abaixo alguns exemplos de pacotes bastante utilizados. Para simplificar a visualização foi usada função t, que transpõe o data.frame de colunas para linhas.

```
pacotes <- as.data.frame(installed.packages())
# pacote base
t(pacotes[pacotes$Package == 'base',])</pre>
```

```
base
                       "base"
Package
                       "C:/Program Files/R/R-4.3.0/library"
LibPath
                       "4.3.0"
Version
Priority
                       "base"
Depends
                       NA
Imports
                       NA
LinkingTo
                       NA
                       "methods"
Suggests
Enhances
                       "Part of R 4.3.0"
License
License_is_FOSS
```

```
License_restricts_use NA
OS_type
                       NA
                       NA
MD5sum
NeedsCompilation
                       NA
                       "4.3.0"
Built
  # pacote MASS
  t(pacotes[pacotes$Package == 'MASS',])
                       MASS
                       "MASS"
Package
LibPath
                       "C:/Program Files/R/R-4.3.0/library"
                       "7.3-58.4"
Version
                       "recommended"
Priority
Depends
                       "R (>= 4.3.0), grDevices, graphics, stats, utils"
Imports
                       "methods"
LinkingTo
                       NA
Suggests
                       "lattice, nlme, nnet, survival"
Enhances
License
                       "GPL-2 | GPL-3"
License_is_FOSS
License_restricts_use NA
OS_type
                       NA
\mathtt{MD5sum}
                       NΑ
NeedsCompilation
                       "yes"
                       "4.3.0"
Built
  # pacote data.table
  t(pacotes[pacotes$Package == 'data.table',])
                       data.table
                       "data.table"
Package
LibPath
                       "C:/Users/luisg/AppData/Local/R/win-library/4.3"
                       "1.14.8"
Version
Priority
                       NA
Depends
                       "R (>= 3.1.0)"
                       "methods"
Imports
LinkingTo
                       NA
                       "bit64 (>= 4.0.0), bit (>= 4.0.4), curl, R.utils, xts,\nnanotime, zoo
Suggests
Enhances
                       NA
```

License "MPL-2.0 | file LICENSE"
License\_is\_FOSS NA
License\_restricts\_use NA
OS\_type NA
MD5sum NA
NeedsCompilation "yes"
Built "4.3.0"

Pode ser visto no campo *Priority* que para o pacote base o conteúdo é "base", isto significa que este faz parte da instalação do R. Já o pacote MASS, por exemplo, é um pacote recomendado. O pacote data.table, que é um pacote "normal", não possui informação no campo *Priority*.

Também podemos visualizar dados do pacote (arquivo *DESCRIPTION* do próprio pacote) com o comando packageDescription:

```
Package: base
Version: 4.3.0
Priority: base
Title: The R Base Package
Author: R Core Team and contributors worldwide
Maintainer: R Core Team <do-use-Contact-address@r-project.org>
Contact: R-help mailing list <r-help@r-project.org>
Description: Base R functions.
License: Part of R 4.3.0
Suggests: methods
Built: R 4.3.0; ; 2023-04-21 09:22:06 UTC; windows
-- File: C:/PROGRA~1/R/R-43~1.0/library/base/Meta/package.rds
```

## 10.2 Pasta de Instalação

O R possui pastas de instalação dos pacotes. Para visualizá-las basta usar o comando .libPaths. A pasta padrão de instalação traz os diversos pacotes que foram instalados junto com o R (os "básicos" e os recomendados).

```
.libPaths()
[1] "C:/Users/luisg/AppData/Local/R/win-library/4.3"
[2] "C:/Program Files/R/R-4.3.0/library"
```

```
# Exibir 10 primeiros da primeira pasta
 list.files(.libPaths()[1])[1:10]
                "backports" "base64enc" "bit"
[1] "askpass"
                                                     "bit64"
                                                                  "blob"
[7] "broom"
                "bslib"
                             "cachem"
                                         "callr"
 # Exibir 10 primeiros da segunda pasta
 list.files(.libPaths()[2])[1:10]
[1] "base"
                "boot"
                             "class"
                                         "cluster"
                                                      "codetools" "compiler"
[7] "datasets"
                "foreign"
                             "graphics"
                                         "grDevices"
```

#### 10.3 Pacotes Disponíveis

A função available.packages procura pacotes disponíveis no valor informado no argumento repos. Por padrão é buscado de getOption("repos").

### 10.4 Dependências de Pacotes

Os pacotes podem e em sua maioria utilizam funções de outros pacotes. Estes "outros pacotes" são denominadas de dependências. As informações de dependências também constam no DESCRIPTION do pacote.

O pacote tools, que faz parte da base do R, oferece uma função para busca de dependências de pacotes. Inclusive existe a opção de recursividade, ou seja, busca também as dependências das dependências do pacote desejado.

```
tools::package_dependencies('dplyr')
```

```
$dplyr
 [1] "cli"
                  "generics"
                               "glue"
                                            "lifecycle" "magrittr"
 [6] "methods"
                  "pillar"
                               "R6"
                                            "rlang"
                                                         "tibble"
[11] "tidyselect" "utils"
                               "vctrs"
  tools::package_dependencies('dplyr', recursive = T)
$dplyr
 [1] "cli"
                               "glue"
                  "generics"
                                            "lifecycle"
                                                         "magrittr"
                  "pillar"
 [6] "methods"
                               "R6"
                                            "rlang"
                                                         "tibble"
[11] "tidyselect" "utils"
                               "vctrs"
                                            "fansi"
                                                         "utf8"
[16] "pkgconfig" "withr"
                               "grDevices"
                                            "graphics"
                                                         "stats"
```

### 10.5 Instalação de Pacotes

Para efetuar a instalação de pacotes usa-se a função install.packages. Os pacotes podem ser instalados diretamente de repositórios na Internet (como o CRAN) ou de arquivos locais.

Última atualização: 12/08/2023 - 20:42:38

# Parte III

# **RStudio**

## O que é o RStudio?

O **RStudio** é um IDE (Integrated Development Environment) criado pela **Posit** para as linguagens R e Python. Apesar de não ser necessário para uilização de R, o RStudio fornece muitas funcionalidades para programação. Nesta seção serão apresentados alguns de seus principais recursos.

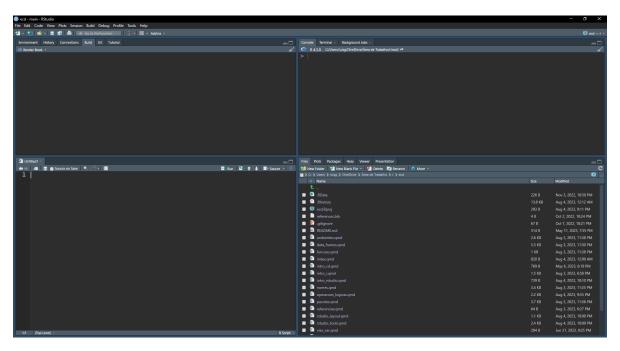


Figura 10.1: RStudio - Tela inicial

RStudio - User Guide

IDE

Última atualização: 12/08/2023 - 20:23:16

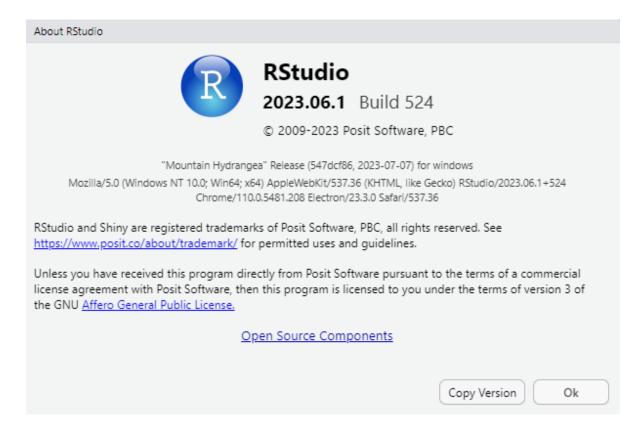


Figura 10.2: About RStudio

# 11 Introdução ao RStudio

### 11.1 Layout

#### Status

O RStudio possui basicamente 4 painéis dimensionáveis. Cada um destes painéis pode trazer uma série de abas.

#### 11.2 Console

Neste painel está embutido o R propriamente dito.

```
Terminal ≥ Background Jobs ≥ R 4.3.0 · Cr/Users/luisg/OneDrive/Area de Trabalho/r/ecd/ → R version 4.3.0 (2023-04-21 ucrt) -- "Already Tomorrow" Copyright (C) 2023 The R Foundation for Statistical Computing Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY. You are welcome to redistribute it under certain conditions. Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors. Type 'contributors()' for more information and 'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or 'help.start()' for an HTML browser interface to help.

Type 'q()' to quit R.
```

Figura 11.1: Console

## 11.3 Output

Painel com diversas saídas fornecidas. Gráficos (Plots), Estrutura de Pastas(Files), Ajuda (Help), Pacotes(Packages), etc aparecem neste painel. Este é um painel muito útil para navegação nos arquivos do projeto e visualização/exportação de gráficos.

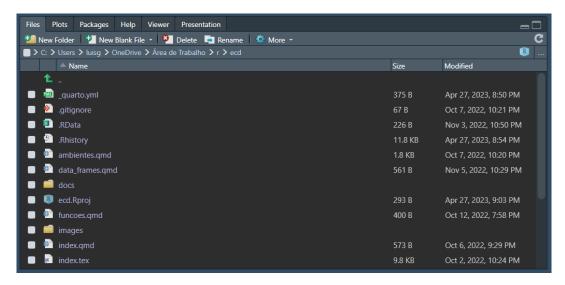


Figura 11.2: Files

#### 11.4 Environment

Apresenta os objetos criados no ambiente do R.

#### 11.5 Source

Aqui são abertos os arquivos de códigos (scripts, Rmarkdown, Quarto, SQL, etc).

RStudio - User Guide

IDE

Última atualização: 12/08/2023 - 20:23:44

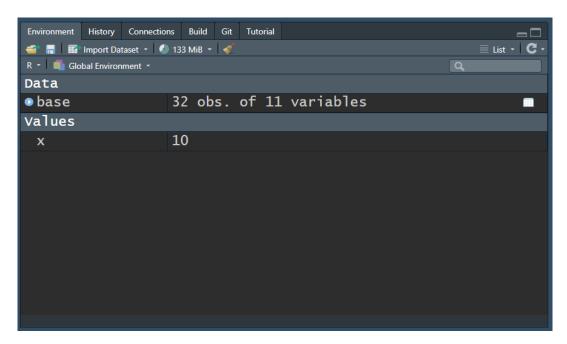


Figura 11.3: Environment

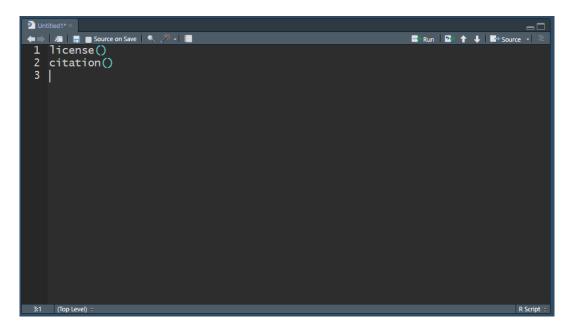


Figura 11.4: Source

# 12 Menu Tools

#### Status

## 12.1 Opções

O menu Tools oferece uma série de funcionalidades para configuração do ambiente de trabalho.

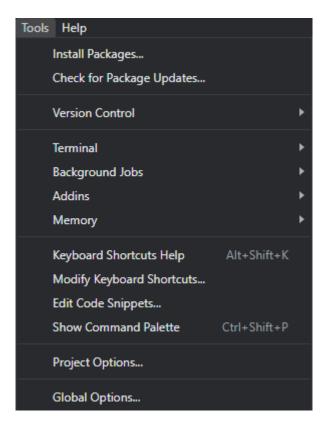


Figura 12.1: Menu Tools

#### 12.1.1 Install Packages

Nesta opção é aberta a janela para instalação de pacotes.

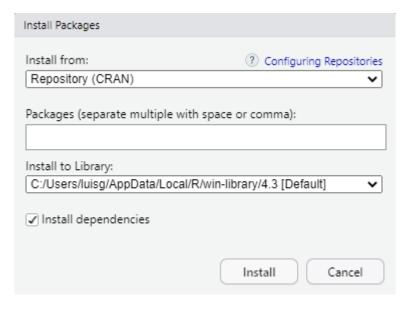


Figura 12.2: Install Packages

- Install From: local de busca dos pacotes a serem instalados
  - Repository: repositório configurado
    - \* Packages: nome dos pacotes a serem instalados. Podem ser escolhidos múltiplos pacotes, devendo ser separados por espaço ou vírgula
  - Package Archive File: opção para busca de arquivo a partir da máquina do usuário. Esta opção habilita botão para busca do pacote
    - \* Package archive: arquivo do pacote a ser instalado
- Install to Library: pasta de instalação dos pacotes
- Install Dependencies: marcação para que seja feita instalação de dependências dos pacotes selecionados.

#### 12.1.2 Check for Package Updates

Este opção abre a janela Update Packages, permitindo visualizar quais pacotes possuem versões mais recentes. A coluna NEWS possibilita visualizar o arquivo com dados de atuializações feitas no pacote.



Figura 12.3: Install From

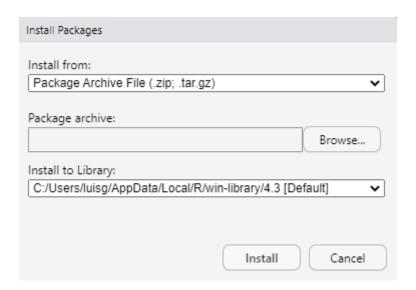


Figura 12.4: Package Archive File

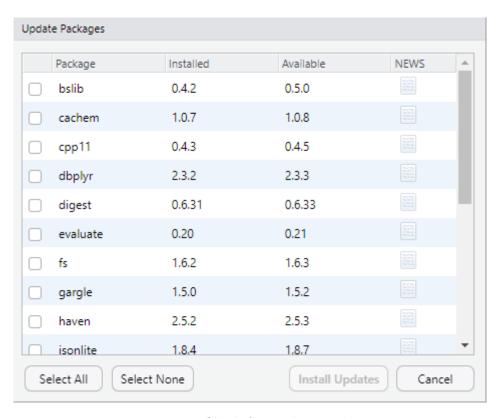


Figura 12.5: Check for Package Updates

12.1.3 Version Control

Oferece opção de controle de versões de código através do Git ou SVN.

12.1.4 Terminal

Permite acesso ao terminal do sistema operacional a partir do RStudio.

12.1.5 Background Jobs

Fornece opções para execução de 'Jobs', basicamente scripts em R, em outra instância do R. Desta forma a sessão aberta no RStudio não fica ocupada e permite que o usuário continue

seu trabalho. Esta opção é muito útil para processamentos mais demorados.

12.1.6 Global Options

Esta opção abre a janela Options do RStudio onde podem ser feitas as principais configurações de comportamento da ferramenta.

\_\_\_\_\_

RStudio - User Guide

IDE

Última atualização: 12/08/2023 - 20:44:16

82

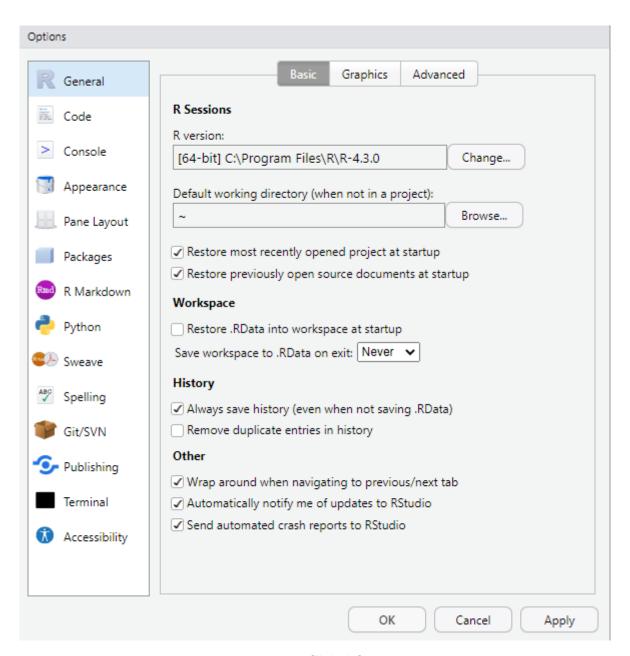


Figura 12.6: Global Options

# Parte IV Ciência de Dados

#### Status

## **Definições**

Segundo Provost e Fawcett (Provost e Fawcett 2016, p 3) 'data science é um conjunto de princípios fundamentais que norteiam a extração de conhecimentos a partir de dados'.

Uma definição da IBM (2023) para Ciência de Dados:

'A ciência de dados combina matemática e estatística, programação especializada, análise avançada, inteligência artificial (IA) e aprendizado de máquina com conhecimento em domínio específico para revelar insights acionáveis ocultos nos dados de uma organização'.

Uma definição mais sucinta da AWS (2023):

'A ciência de dados é o estudo dos dados para extrair insights significativos para os negócios'.

Última atualização: 12/08/2023 - 20:28:18

# 13 Trade-Off Viés x Variância

Status:			

Última atualização: 12/08/2023 -  $20{:}23{:}23$ 

# Convenções

#### Status

# Marcações no Texto

A fim de facilitar a leitura e evidenciar itens importantes no texto, serão adotadas as seguintes marcações:

Tipo do Texto	Marcação	Exemplo
Funções, operadores do R	Código	print
Objetos criados no R, termos relevantes	Negrito	<b>Negrito</b>
Palavras de língua estrangeira	Itálico	<i>Itálico</i>

# Nomes de Objetos

Abaixo convenções a serem usadas neste material.

Tabela 13.2: Convenções de código

Tipo Objeto	Convenção	Exemplo
Data.frame, tibble ou data.table	snake_case iniciado por df ( $\mathbf{d}$ ata $\mathbf{f}$ rame)	df_clientes
Variáveis de datasets	SCREAMING_SNAKE_CASE	$df\_clientes$NOME\_CLIENTE$
Funções	camelCase iniciado por <b>fn</b> , sendo a primeira palavra após fn um verbo	fnBuscarClientes
Demais (vetores, listas, etc.)	snake_case	nomes_cidades

#### Status do Material

Para indicação de status do material apresentado serão usados os símbolos baixo no topo de cada capítulo:

Indicador	Estrutura	Conteúdo	Status Geral
	não iniciado	não iniciado	não iniciado
	incipiente	não criado	incipiente
	incipiente	incipiente	incipiente
	incipiente	em revisão	incipiente
	em revisão	em revisão	em revisão
	em revisão	amadurecido	em revisão
	em revisão	incipiente	incipiente
	incipiente	amadurecido	incipiente
	incipiente	desatualizado	desatualizado
	amadurecido	incipiente	incipiente
	amadurecido	em revisão	em revisão
	amadurecido	desatualizado	desatualizado
	a madure cido	a madure cido	a madure cido

- **incipiente:** recém iniciado, é o status mais volátil. Após melhores definições passa a ser marcado como .
- **em revisão:** em alterações (grandes) para melhorias. Após este status será marcado como .
- **amadurecido:** já passado por revisão. Pode sofrer pequenas alterações e atualizações. Caso se identifique que necessite de grandes alterações será marcado como:
  - para revisão por decisão
  - para revisão por desatualização
- desatualizado: necessita ser reescrito por força maior, como desatualização de conceitos ou códigos.

**não iniciado:** serve como marcação de 'todo'. usado para seções que se entendem necessárias mas que ainda não foram iniciadas.

Última atualização: 14/08/2023 - 20:35:27

# Referências

- AWS. 2023. "O que é ciência de dados?" 2023. https://aws.amazon.com/pt/what-is/data-science/.
- Dowle, Matt, e Arun Srinivasan. 2023. data.table: Extension of 'data.frame'.
- Grolemund, Garrett. 2014. *Hands-On Programming with R.* O'Reilly. https://rstudio-education.github.io/hopr/.
- IBM. 2023. "What is data science?" 2023. https://www.ibm.com/topics/data-science.
- Mastropietro, Daniel. 2019. "Getting an environment's name in R: the envnames package". 2019. https://www.r-bloggers.com/2019/05/getting-an-environments-name-in-r-the-envnames-package.
- Provost, Foster, e Tom Fawcett. 2016. DataScience para Negócios. Alta Books.
- R Core Team. 2023a. An Introduction to R. R Foundation for Statistical Computing. https://cran.r-project.org/doc/manuals/r-release/R-intro.html.
- ——. 2023b. R Language Definition. R Foundation for Statistical Computing. https://cran.r-project.org/doc/manuals/r-release/R-lang.html.
- ——. 2023c. R: A Language and Environment for Statistical Computing. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.
- ———. 2023d. "R: A Language and Environment for Statistical Computing". Vienna, Austria: R Foundation for Statistical Computing. 2023. https://cran.r-project.org/doc/manuals/r-devel/NEWS.html.
- Wickham, Hadley. 2023/04/21. "Differences between the base R and magrittr pipes". 2023/04/21. https://www.tidyverse.org/blog/2023/04/base-vs-magrittr-pipe/.
- Wikipedia, the free encyclopedia. 2023. "Naming convention (programming)". 2023. https://en.wikipedia.org/wiki/Naming\_convention\_(programming).