

Análise de Dados Ambientais com R

Jônatan Tatsch

2018-04-20

Contents

Apresentação	5
1 Introdução	7
1.1 Análise de dados meteorológicos	7
1.2 Ciência de dados	7
1.3 Etapas para abordagem de um problema	7
1.4 Programação computacional	8
1.5 R	8
1.6 Por que o R?	8
1.7 Pacotes da comunidade do R	8
1.8 Por que um meteorologista usaria o R?	9
1.9 R não é perfeito!	9
1.10 Para saber mais sobre o R	9
2 Instalação do R e RStudio	11
2.1 Instalando o R	11
2.2 Diretório para instalação de pacotes	14
2.3 Rstudio no Ubuntu	15
3 Interface do Usuário	21
3.1 R no modo interativo	21
3.2 R no modo de processamento em lote	24
4 Operações básicas	25
4.1 Convenção	25
4.2 CalculadoRa	25
4.3 Variáveis	29
5 Tipos de dados	33
5.1 Classes de dados	33
5.2 Testes sobre tipos de dados	37
5.3 Conversão entre tipos de dados	38

Apresentação

Este material é uma composição das notas de aula da disciplina **Análise de Dados Ambientais com R** do curso de Graduação em METEOROLOGIA oferecido no Departamento de Física da Universidade Federal de Santa Maria (UFSM).

O livro é designado para quem não tem experiência em programação, ou qualquer um com interesse em aprender o R para manipular dados ambientais. O objetivo é prover uma material para ensinar os conceitos básicos de programação necessários para o processamento, a visualização e a análise de dados ambientais com o sistema computacional R. Estes procedimentos são potencializados com o uso do software RStudio, uma interface de desenvolvimento integrado (IDE) para o R.

Neste livro o leitor aprenderá a sintaxe básica da linguagem R (R Core Team, 2018), a importação e exportação de dados, a criação de gráficos, funções, a padronização e organização de conjunto de dados; e finalmente, a confecção de relatórios dinâmicos e reproduzíveis.

O material do livro inclui o uso de dados ambientais de diferentes áreas (meteorologia, climatologia, hidrologia, sensoriamento remoto) em exemplos práticos e em exercícios, para estimular a prática da programação.

O texto é intercalado com trechos de códigos que podem ser reproduzidos e os resultados visualizados no computador do leitor.

Após a introdução ao R apresenta-se as capacidades específicas do R para manipulação de dados. Baseado na experiência do autor são empregados os pacotes mais adequados para cada finalidade, como **dplyr** e **tidyr** para o processamento de dados e o **ggplot2** para visualização de dados.

A intenção do livro é que após a leitura, o leitor tenha o conhecimento suficiente para desenvolver códigos que automatizem tarefas repetitivas, assim reduzindo o tempo na etapa de preparação de dados. Esta programação mais efetiva permitirá focar mais na análise de dados e na comunicação dos resultados, seja ela na forma de documentos acadêmicos, ou relatórios técnicos em empresas públicas e privadas.

O texto está em formato html para tirar o melhor proveito de recursos de multimídia, da capacidade de busca de texto e links para websites.

O texto é organizado em 5 capítulos:

- 1 Introdução
- 2 Instalação do R e Rstudio
- 3 Interface do Usuário
- 4 Operações Básicas
- 5 Tipos de dados

Chapter 1

Introdução

Breve intro.

1.1 Análise de dados meteorológicos

Processo pelo qual adquire-se conhecimento, compreensão e percepção dos fenômenos meteorológicos a partir de observações (dados) qualitativas e quantitativas.

1.2 Ciência de dados

1.3 Etapas para abordagem de um problema

1. **Questão científica/problema**
2. **Obtenção de dados:** coleta/medida do(as) estado/condições da atmosfera
 - Instrumentos e sensores
3. **Processamento de dados:** *download* —> limpeza —> formatação —> transformação —> controle de qualidade
 - ferramenta/software
 - conhecimento em programação
4. **Análise de dados**
 - ferramenta/software
 - conhecimento em programação
5. **Solução para o problema**
 - Proposta de um modelo
 - estatístico, empírico, ou fisicamente baseado
 - conhecimento em programação
6. **Apresentação/divulgação/publicação**

1.4 Programação computacional

1.5 R

- R é o termo usado para se referir a linguagem de programação e ao software que interpreta os scripts escritos usando esta linguagem.
- Comunidade fantástica
- Contribuidores (R-core Team)
- milhares de pessoas usam o R diariamente e ajudam outras pessoas
- **Software Livre** (GPL), Código aberto e multiplataforma
- Ambiente para Análise de dados interativa

1.6 Por que o R?

- R não é uma GUI (Interface gráfica do usuário) e isso é bom
 - há uma natural resistência e dificuldade ao uso de códigos e scripts
 - scripts favorecem a **automatização** e **reprodutibilidade**
 - força você a ter um conhecimento mais aprofundado do que está fazendo
- Reprodutibilidade
 - qualquer pessoa (inclusive você mesmo no futuro) pode obter os mesmos resultados do mesmo conjunto de dados
 - R é integrado com outras ferramentas de que permitem atualizar seus resultados, figuras e análises automaticamente
- Relatório dinâmicos e interativos
- Acesso ao estado da arte da ciência de dados (*Big Data*, *Data Mining*, *Machine Learning*)
- é um software livre, de código fonte aberto e funciona em diversos sistemas operacionais (Linux, Windows e MacOS).
- Interface com Fortran, C, C++, Python
- Visualização de dados
- R produz gráficos de alta qualidade
- R trabalha com dados de todas formas e tamanhos
- Extensões para Manipulação de dados

1.7 Pacotes da comunidade do R

Evolução do nº de pacotes disponíveis no CRAN

1.8 Por que um meteorologista usaria o R?

A meteorologia é 4D:

```
meteorologia <- function(x, y, z, t){  
  ...muita coisa para caber em um slide...  
}
```

Logo, requer ferramentas específicas para:

- manipulação de dados espaciais
- análise de séries temporais
- importação e ferramentas de SIG
- leitura de dados em formatos específicos (netcdf, binários, grib2, ...)

1.9 R não é perfeito!

- Muitos códigos em R são escritos para resolver um problema;
 - foco nos resultados e não no processo
 - usuários não são programadores
 - códigos deselegantes, lentos e difíceis de entender
- Como o nosso idioma, há muitas exceções para serem lembradas
- R não é muito rápido e códigos mal escritos serão lentos
- São apenas ~20 anos de evolução
- Há muito o que melhorar

1.10 Para saber mais sobre o R

Documentação oficial - Manuais do R traduzidos

Lista de Livros relacionados ao R

- Livros gratuitos (em inglês)

Fóruns:

- lista Brasileira de discussão do programa R: **R-br**
- stackoverflow

Chapter 2

Instalação do R e RStudio

A interação do usuário com o R é por meio da linha de comando. Essa interação pode ser facilitada com o uso do RStudio.

A seguir descreve-se como instalar o R no Windows e no Linux Ubuntu. A forma de instalação do R no Linux tenta ser mais didática do que prática. Alguns comandos linux básicos serão utilizados, mas mesmo quem não é usuário linux será capaz de entendê-los.

2.1 Instalando o R

O R pode ser instalado a partir dos binários pré-compilados ou do código fonte. Aqui, descreve-se a instalação do R a partir dos binários

2.1.1 Windows

A forma de instalar o R no Windows é baixar o binário executável da **Rede Abrangente de Arquivos do R** (CRAN). Depois clicar em *Download R for Windows* e *install R for the first time*. Quando este tutorial foi escrito a última versão foi a R 3.4.4.

A instalação do R para Windows a partir do executável acima incluirá na instalação uma GUI chamada `RGui.exe`, mostrada abaixo.

2.1.2 Linux

2.1.2.1 Ubuntu

Há várias formas de instalar o R no Ubuntu, mas geralmente a versão compilada no repositório *default* do Ubuntu não é a última. Se isso não for problema para você então basta executar:

```
sudo apt-get install r-base
```

Entretanto, os pacotes do R recém lançados são compilados para última versão do R. Então você pode ter restrições de uso de pacotes novos, os quais geralmente incluem o estado da arte de análise de dados.

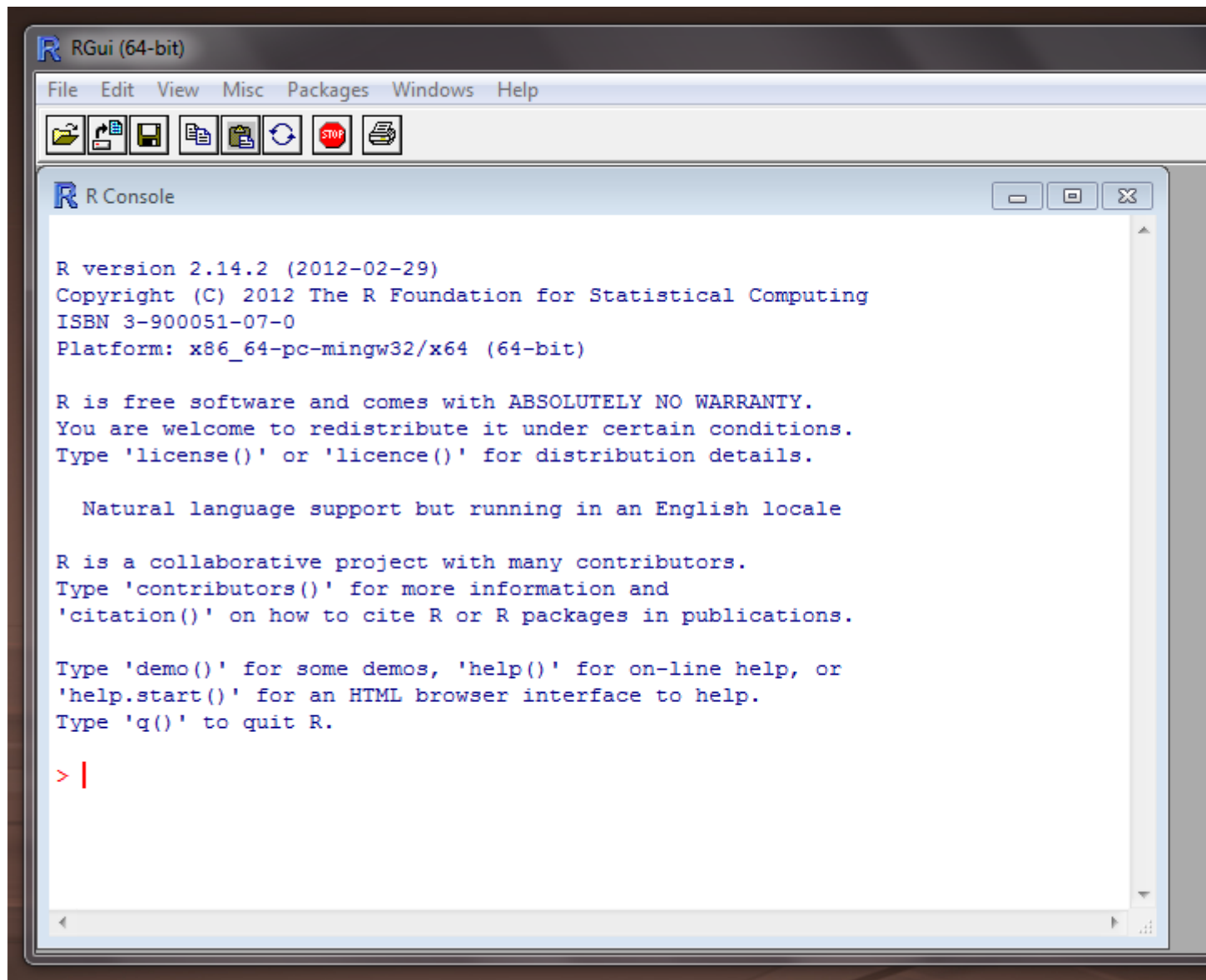


Figure 2.1: Interface gráfica do usuário no R para Windows.

2.1.2.2 R sempre atualizado

Se você quer trabalhar sempre com a última versão estável do R, é possível configurar o Linux Ubuntu para atualizar automaticamente o R. O procedimento de instalação requer senha de superusuário do sistema ou de privilégios `sudo`. Caso não tenha, consulte o administrador do sistema.

Ao utilizar distribuições Linux Ubuntu é importante optar por versões estáveis¹. As versões de Suporte de longo prazo (LTS) mais recentes são:

- 14.04 (abril de 2014, *codename* `trusty`)
- 16.04 (abril de 2016, *codename* `xenial`)

O R é distribuído na CRAN. Geralmente há duas atualizações ao ano. A versão mais atual é a R version 3.4.4 (2018-03-15). Para que ele seja atualizado automaticamente no Ubuntu precisamos adicionar o repositório do R mais próximo da nossa região à lista de repositórios do Linux. No nosso caso, o repositório mais próximo é o da UFPR (<http://cran-r.c3sl.ufpr.br/>).

2.1.2.2.1 Incluindo repositório do R na Lista de repositórios do Ubuntu

A lista de repositórios do sistema é armazenada no arquivo `/etc/apt/sources.list`. Mas primeiro, você precisa descobrir ou verificar o nome da versão do sistema operacional. Para isso, você pode utilizar o seguinte comando²:

```
$ lsb_release --codename | cut -f2
trusty
```

Precisamos incluir no arquivo `sources.list` o espelho do repositório do R mais próximo. Veja a lista de espelhos de repositórios do R aqui. Assim o gerenciador de pacotes `apt`³ fará a atualização do R quando uma nova versão estiver disponível. Ou seja, você estará utilizando sempre versão mais atual do R.

O endereço do repositório da UFPR será inserido na última linha do arquivo `sources.list` usando alguns comandos linux. Essa tarefa requer privilégios de superusuário. Vamos trocar do seu usuário para o superusuário.

```
$ sudo su
```

Vamos definir no terminal uma variável com o endereço do repositório (da UFPR nesse caso) e o nome de versão do Ubuntu.

```
# repos="deb http://cran-r.c3sl.ufpr.br/bin/linux/ubuntu `lsb_release --codename | cut -f2`/"
```

Note que a variável `repos` é uma sequência de caracteres com as seguintes informações:

```
deb `linkRepositorioSelecionado`/bin/linux/ubuntu `versaoUbuntu`/
```

O valor da variável `repos` é mostrado pelo comando: `echo $repos`. Certifique-se de que a última palavra corresponde ao nome da sua versão Ubuntu.

Para acrescentar essa informação no final do arquivo `sources.list` digite no terminal linux:

```
# echo $repos >> /etc/apt/sources.list
```

Feito isso, você pode retornar a sessão de usuário comum, usando o comando abaixo:

```
# exit
```

¹Essa lista de variáveis também é mostrada no painel *Environment* do RStudio (canto direito superior, aba *Environment*).

²Se o comando `lsb_release` não funcionar você precisa instalar o pacote `lsb-release` no sistema. Para isso, digite no terminal Linux `sudo apt-get install lsb-release`.

³O gerenciador de pacotes `apt` é usado para instalação, atualização e remoção de pacotes em distribuições Debian GNU/Linux.

2.1.2.2.2 APT protegido

Os arquivos binários do R para Ubuntu na CRAN são assinados com uma chave pública⁴ Para adicionar essa chave ao seu sistema digite os seguintes comandos:

```
$ gpg --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys E084DAB9
```

e então use essa informação como entrada no `apt-key` com

```
$ gpg -a --export E084DAB9 | sudo apt-key add -
```

Se aparecer a mensagem de que a chave pública foi importada, então não há necessidade de executar os comandos abaixo. Mas caso seja impresso alguma mensagem de erro, outra alternativa pode ser usada para obter a chave, via os comandos:

```
$ gpg --keyserver keyserver.ubuntu.com --recv-key E084DAB9
```

```
$ gpg -a --export E084DAB9 | sudo apt-key add -
```

2.1.2.2.3 Atualização da lista de repositórios do Ubuntu e instalação do R

Após fazer as configurações da lista de repositórios e adicionar a chave é necessário fazer a atualização dessa lista (requer poderes de super usuário):

```
$ sudo apt-get update
```

Agora, pode instalar o binário do R:

```
$ sudo apt-get install r-base
```

2.1.2.2.4 Testando o R

Para iniciar o R no Ubuntu, digite `R` no cursor do terminal:

```
$ R
```

A partir desse momento já começamos uma sessão no R. Vamos gerar uma sequência numérica de 1 a 10 e plotá-la.

```
> 1:10
[1] 1 2 3 4 5 6 7 8 9 10
> plot(1:10)
```

Você pode sair do R, sem salvar os dados da sessão, com o código a seguir:

```
> q(save = "no")
```

2.2 Diretório para instalação de pacotes

Uma boa prática é definir um diretório para armazenamento dos pacotes utilizados. Isso lhe dá mais controle sobre os pacotes do R instalados no sistema. Um local sugerido é o `/home/usuario/.R/libs`. O seu `home` ou `pasta pessoal` pode ser obtido com o comando `echo $HOME`. Para criar o diretório você pode digitar o comando abaixo:

```
$ mkdir -p `echo $HOME`/.R/libs/
```

⁴Chave pública de autenticação é um meio alternativo de se logar em um servidor ao invés de digitar uma senha. É uma forma mais segura e flexível, mas mais difícil de ser configurada. Esse meio alternativo de fazer login é importante se o computador está visível na internet. Para saber mais veja aqui.

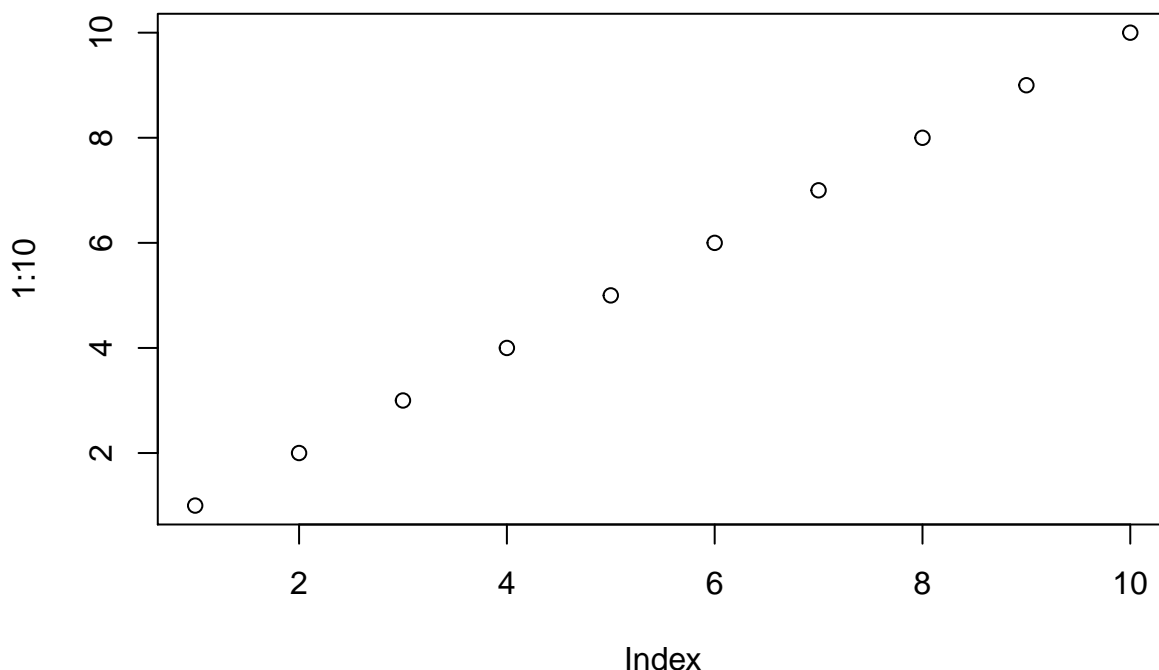


Figure 2.2: Gráfico da sequência de 10 números.

Para informar ao R onde procurar os pacotes instalados, você precisa criar um arquivo chamado `.Renviron`, no diretório `$HOME`, contendo a expressão `R_LIBS=/home/usuario/.R/libs/`. Você pode fazer isso em um terminal com os comandos:

```
$ R_LIBS=`echo $HOME/.R/libs/`
$ echo $R_LIBS >> `echo $HOME/.Renviron`
```

Esse caminho fica então visível ao R, o que pode ser verificado executando a função `.libPaths()` no console do R. Abra o R:

```
$ R
```

e ao digitar:

```
> .libPaths()
[1] "/home/hidrometeorologista/.R/libs" "/usr/local/lib/R/site-library"
[3] "/usr/lib/R/site-library"          "/usr/lib/R/library"
```

o seu diretório `/home/usuario/.R/libs`⁵ deve aparecer em primeiro lugar. Indicando que este local tem prioridade para instalação dos pacotes. Caso o diretório deixe de existir os seguintes diretórios serão usados.

2.3 Rstudio no Ubuntu

O RStudio é um ambiente integrado de desenvolvimento (IDE) construído especificamente para o R. O RStudio para Desktop pode ser baixado gratuitamente e é multiplataforma.

Para instalação da versão do RStudio para *Desktop*, você precisa saber se seu SO é 64 ou 32-bit e a versão do Linux Ubuntu. Essas informações podem ser obtidas, respectivamente, pelos comandos:

⁵Diretórios precedidos por “.” no Linux são diretórios ocultos. O diretório `/home/usuario/.R` é um diretório oculto, para visualizá-lo no Ubuntu, na interface gráfica do sistema, acesse *View > Show Hidden Files* (ou *Visualizar > Mostrar arquivos ocultos*). No terminal utilize `ls -a` para listar os arquivos ocultos.

```
$ arch
```

```
x86_64
```

```
$ lsb_release --release | cut -f2
```

```
14.04
```

Se retornar **x86_64** sua máquina é 64-bit.


Com essa informação e versão do sistema operacional, siga os seguintes passos:

1. acesse RStudio
2. clique em *Download RStudio*
3. Procure a opção *RStudio Desktop* (FREE) e clique *download*

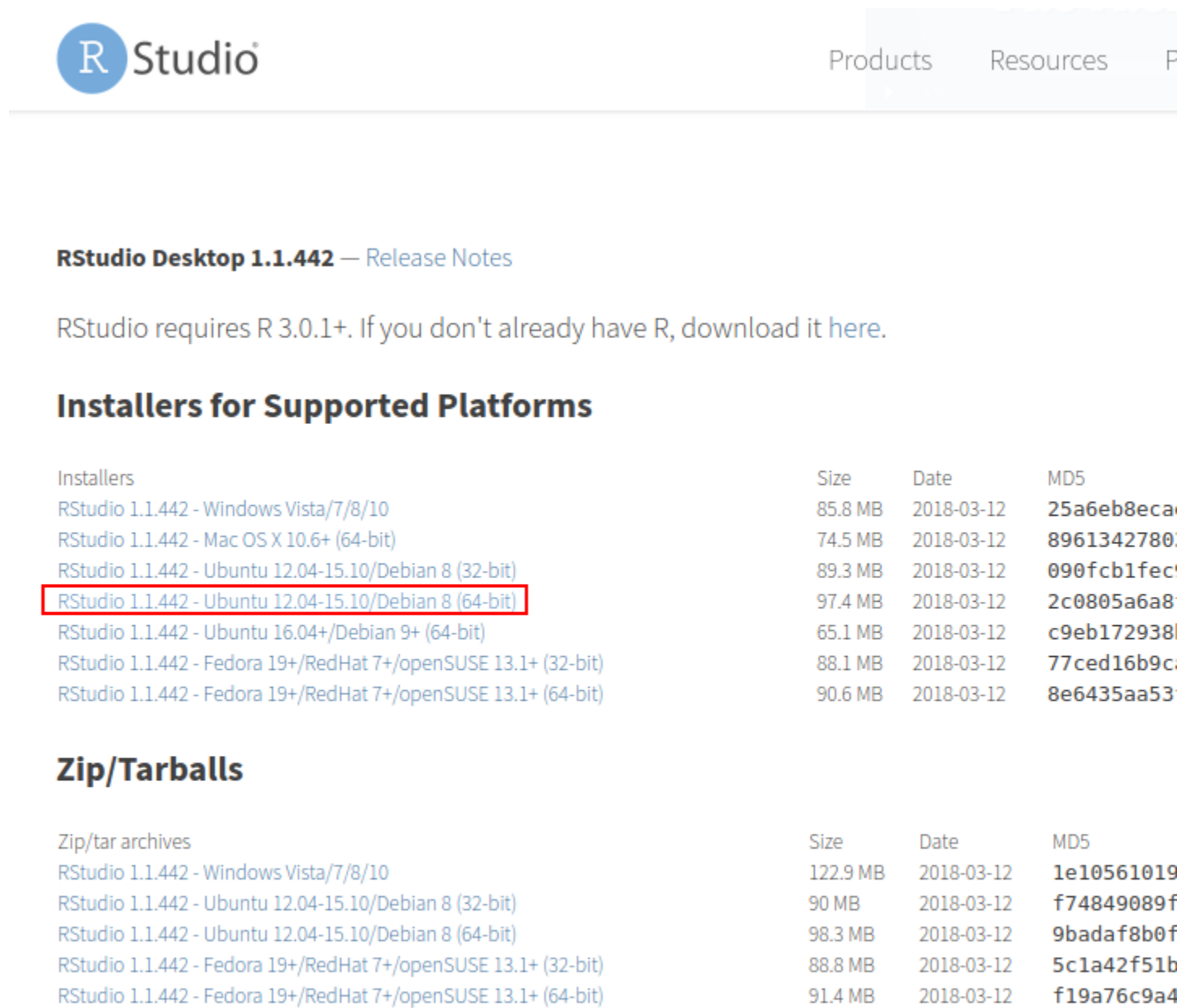
[Products](#)[Resources](#)[Pricing](#)

Choose Your Version of RStudio

RStudio is a set of integrated tools designed to help you be more productive with R. It includes a code editor with syntax-highlighting, a console for direct code execution, and a variety of robust tools for plotting, viewing history, debugging and managing your workspace. [Learn More about RStudio features.](#)

	RStudio Desktop Open Source License	RStudio Desktop Commercial License	RStudio Server Open Source License	RStudio Server Commercial License
	<u>FREE</u>	\$995 per year	FREE	\$9,995 per year
	 DOWNLOAD	BUY	DOWNLOAD	DOWNLOAD
	Learn More	Learn More	Learn More	Learn More
Integrated Tools for R	●	●	●	●
Priority Support		●		●
Access via Web Browser			●	
Enterprise Security				●
Project Sharing				●

5. Selecione sua plataforma



RStudio Desktop 1.1.442 — [Release Notes](#)

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

Installers for Supported Platforms

Installers	Size	Date	MD5
RStudio 1.1.442 - Windows Vista/7/8/10	85.8 MB	2018-03-12	25a6eb8eca
RStudio 1.1.442 - Mac OS X 10.6+ (64-bit)	74.5 MB	2018-03-12	8961342780
RStudio 1.1.442 - Ubuntu 12.04-15.10/Debian 8 (32-bit)	89.3 MB	2018-03-12	090fcb1fec
RStudio 1.1.442 - Ubuntu 12.04-15.10/Debian 8 (64-bit)	97.4 MB	2018-03-12	2c0805a6a8
RStudio 1.1.442 - Ubuntu 16.04+/Debian 9+ (64-bit)	65.1 MB	2018-03-12	c9eb172938
RStudio 1.1.442 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	88.1 MB	2018-03-12	77ced16b9c
RStudio 1.1.442 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	90.6 MB	2018-03-12	8e6435aa53

Zip/Tarballs

Zip/tar archives	Size	Date	MD5
RStudio 1.1.442 - Windows Vista/7/8/10	122.9 MB	2018-03-12	1e10561019
RStudio 1.1.442 - Ubuntu 12.04-15.10/Debian 8 (32-bit)	90 MB	2018-03-12	f74849089f
RStudio 1.1.442 - Ubuntu 12.04-15.10/Debian 8 (64-bit)	98.3 MB	2018-03-12	9badaf8b0f
RStudio 1.1.442 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	88.8 MB	2018-03-12	5c1a42f51b
RStudio 1.1.442 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	91.4 MB	2018-03-12	f19a76c9a4

Source Code

A tarball containing source code for RStudio v1.1.442 can be downloaded from [here](#)

clique sobre o link da sua plataforma, p.ex.: *RStudio x.xx.xxx - Ubuntu 12.04-15.10/Debian 8 (64-bit)*

6. Dependendo da sua versão Ubuntu, ao clicar sobre o sobre o arquivo baixado com o botão direito, há a opção de abrir com *Ubuntu Software Center* e então clicar em **instalar**. Se na versão de seu Desktop

não há esta opção ao clicar com botão direito sobre o arquivo, instale via **terminal**⁶ com os seguintes comandos:

```
$ cd /local/do/arquivo/baixado
$ sudo dpkg -i arquivoBaixado.deb
$ sudo apt-get install -f
```

Abra o RStudio digitando no terminal:

```
$ rstudio &
```

Agora você está pronto para começar a programar em R aproveitando as facilidades que o RStudio oferece.

⁶Para fazer isso, você pode usar um editor de texto qualquer (p.ex.: gedit no SO Linux, ou Notepad no SO Windows).

Chapter 3

Interface do Usuário

Na maior parte do tempo você provavelmente usará o R no **modo interativo**: rodando comandos e vendo os resultados.

Eventualmente esse processo pode ser inconveniente. Por exemplo, no caso de uma análise com um código bem extenso e que precisa ser repetida com dados atualizados semanalmente. Nessa situação, recomenda-se a criação de um script, ou seja, um arquivo texto, com a extensão `.R`, contendo o código de sua análise.

Esse *script* pode ser executado pelo R no **modo de processamento em lote** (do termo em inglês *Batch Processing*) através de um terminal do SO Linux, ou via o Prompt de comando (`cmd.exe`) do SO Windows.

Nesta seção apresenta-se ao leitor estes dois modos de execução do R.

3.1 R no modo interativo

No Linux o R pode ser aberto simplesmente digitando em um terminal a letra `R`.

```
$ R

R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (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.

Natural language support but running in an English locale

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.

>
```

A janela com a linha de comando do R apresenta o *prompt* do R (`>`). Após este símbolo digitamos os comandos, pressionamos a tecla `<enter>`, o R interpreta o comando e retorna o resultado.

Os comandos digitados na linha de comando são chamados de expressões. Esse é o modo iterativo do R. Portanto, a linha de comando é a mais importante ferramenta do R, pois todas expressões são avaliadas através dela.

```
> 62 + 38
[1] 100
```

A expressão é avaliada pelo R, o resultado é mostrado, mas o seu valor é perdido.

O número entre colchetes que aparece como resultado da operação (“[1]” no caso acima) indica o conteúdo resultante da operação iniciando na posição 1 desse objeto. O significado dessa informação torna-se mais óbvio quando trabalhamos com objetos maiores, como por exemplo com vetores. Observe os valores nos colchetes para uma sequência de 100 até 1.

```
> 100:1
 [1] 100  99  98  97  96  95  94  93  92  91  90  89  88  87  86  85  84
[18]  83  82  81  80  79  78  77  76  75  74  73  72  71  70  69  68  67
[35]  66  65  64  63  62  61  60  59  58  57  56  55  54  53  52  51  50
[52]  49  48  47  46  45  44  43  42  41  40  39  38  37  36  35  34  33
[69]  32  31  30  29  28  27  26  25  24  23  22  21  20  19  18  17  16
[86]  15  14  13  12  11  10   9   8   7   6   5   4   3   2   1
```

O elemento [18] da sequência de 100 até 1 é o número 83.

Pode ocorrer da expressão digitada na linha ser muito extensa e ir além de uma linha. Se a expressão estiver incompleta o R mostra um sinal de +.

```
> 1 * 2 * 3 * 4 * 5 *
+ 6 * 7 * 8 * 9 * 10
[1] 3628800
```

Execute a expressão abaixo até o sinal de menos e tecle <enter>. Enquanto a expressão não estiver completa o sinal de + se repetirá. Até que você digite o número que deseja subtrair de 4.

```
> 4 -
+
+ 3
[1] 1
```

3.1.1 Expressões em sequência

Podemos executar todas expressões anteriores em apenas uma linha, usando o ponto e vírgula ; para separar as expressões:

```
> 62 + 38; 100:1; 1 * 2 * 3 * 4 * 5 * 6 * 7 * 8 * 9 * 10; 4 - 3
[1] 100
 [1] 100  99  98  97  96  95  94  93  92  91  90  89  88  87  86  85  84
[18]  83  82  81  80  79  78  77  76  75  74  73  72  71  70  69  68  67
[35]  66  65  64  63  62  61  60  59  58  57  56  55  54  53  52  51  50
[52]  49  48  47  46  45  44  43  42  41  40  39  38  37  36  35  34  33
[69]  32  31  30  29  28  27  26  25  24  23  22  21  20  19  18  17  16
[86]  15  14  13  12  11  10   9   8   7   6   5   4   3   2   1
[1] 3628800
[1] 1
```

3.1.2 Navegação entre as expressões já avaliadas

Você pode usar as teclas `e` e `l` para navegar entre as expressões já avaliadas pelo R. O que é útil quando precisamos repetir um comando anterior com alguma mudança ou para corrigir um erro de digitação ou a omissão de um parentêses.

Quando a linha de comando é usada por muito tempo a sua tela pode ficar poluída com a saída das expressões anteriores. Para limpar a tela, tecla `Ctrl+l`. Assim o console aparece na parte superior do terminal.

```
> 15 + 4
[1] 19
> 100:1
[1] 100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84
[18] 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67
[35] 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50
[52] 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33
[69] 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
[86] 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
> #tecle <Ctrl + l>
```

Para parar ou cancelar a execução de uma expressão utilize as teclas `Ctrl + C`. As teclas `Ctrl + l` tem o efeito de limpar a tela.

3.1.3 Comentários

No R, a cerquilha `#` (hashtag) é um caracter especial. Qualquer coisa após esse caracter será ignorada pelo R. Somente as expressões antes da `#` são avaliadas. Por meio desse símbolo de comentário podemos fazer anotações e comentários no código sem atrapalhar a interpretação das expressões pelo R.

```
> # comentário antes do código

> 17 + 3 # comentário ao lado do código: adicionando 17 e 3
[1] 20
```

3.1.4 Auto preenchimento de funções

O R inclui o preenchimento automático de nomes de funções e arquivos por meio da tecla `<tab>`. Uma lista de possíveis funções que começam com as letras inicialmente digitadas aparecerão.

```
> read#<tab> pressione <tab> para ver as opções de comandos que iniciam com o termo read
```

3.1.5 Primeiro *script*

O trecho de código abaixo apresenta nas primeiras linhas algumas expressões do R executadas anteriormente. Mas há também, na segunda parte, códigos para salvar um gráfico de pontos num arquivo *pdf*. Na última parte do trecho, define-se uma variável `x` que contém aquela mesma sequência numérica usada no gráfico.

```
# Primeiro script no R
#-----
# cálculos básicos
15 + 4
1:100
1 * 2 * 3 * 4 * 5 * 6 * 7 * 8 * 9 * 10
4-3
#-----
```

```
# salvando um gráfico em um arquivo pdf
arquivo_pdf <- "plot-script1.pdf"
pdf(arquivo_pdf)      # cria e abre um arquivo pdf
plot(1:100)           # faz o gráfico
dev.off()              # fecha o arquivo pdf
#-----
# definindo uma variável x
x <- 1:100
x
```

Este conjunto de linhas de código, quando inseridos em um arquivo texto¹ formam um primeiro *script* R. Este *script* pode ser executado pelo R através da função `source()`, usando como argumento o caminho para o local do *script*.

```
> source("/home/usuario/adar/script1.R")
```

Este *script* produzirá como saída o arquivo `/home/usuario/adar/plot-script1.pdf`. Você pode visualizar o arquivo para conferir o gráficos de pontos gerado.

3.2 R no modo de processamento em lote

Para rodar um *script* no modo de processamento em lote do R através do seguinte comando no terminal Linux:

```
$ R CMD BATCH opcoes arqentrada arqsaida
```

Onde: `arqentrada` é o nome do script (arquivo com a extensão `.R`) a ser executado; `arqsaida` é o arquivo (com a extensão `.Rout`) com as saídas dos comandos executados no R; `opcoes` é a lista de opções que controlam a execução.

Vamos rodar como exemplo, o `script1.R` da seção 3.1.5.

```
$ R CMD BATCH /home/usuario/adar/script1.R
```

O comando acima, produzirá dois arquivos de saída:

1. `script1.Rout`² criado por *default* quando o `arqsaida` não é especificado, e;
2. arquivo `"plot-script1.pdf"`.

Você pode especificar o nome do `arqsaida` como desejar. No exemplo abaixo, mostra-se como salvar o arquivo de saída incluindo a data em que ele foi gerado, `script1-saida-adatadehoje.log`.

```
$ R CMD BATCH script1.R script1-saida-`date +%Y%m%d`\.log
```

Após a execução do último comando, os mesmos arquivos resultantes do comando anterior serão gerados, exceto pelo primeiro (`.Rout`), que será nomeado `script1-saida-20180420.Rout`.

Para mais opções do comando `R CMD BATCH` digite no terminal do Linux `R --help`.

¹Para fazer isso, você pode usar um editor de texto qualquer (p.ex.: `gedit` no SO Linux, ou `Notepad` no SO Windows).

²Você pode notar que este arquivo tem o mesmo nome do `arqentrada`, exceto que a sua extensão foi alterada para `.Rout`.

Chapter 4

Operações básicas

Nesta seção veremos:

- operações aritméticas básicas com R
- a atribuição de valores a uma variável
- o uso de funções matemáticas internas do R
- valores numéricos especiais do R
- os cuidados ao nomear variáveis

4.1 Convenção

A partir deste capítulo, os códigos a serem avaliadas no R terão o prompt do R (>) omitidos. Essa convenção é para tornar mais fácil a ação de copiar e colar os códigos na linha de comando do R. O resultado da avaliação das expressões será mostrado precedido do símbolo (#>). Esses valores são os resultados que esperam-se sejam reproduzidos pelo leitor na sessão do R em seu computador. Por exemplo:

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

No trecho de código acima, a primeira linha contém o código a ser copiado pelo leitor para execução em seu computador. A segunda linha é a saída do código avaliado pelo R.

4.2 CalculadoRa

O R é uma calculadora turbinada com diversas funções matemáticas disponíveis. Para quem não conhece o R, essa é uma forma de familiarizar-se com a linha de comandos do R.

4.2.1 Aritmética básica

Todas operações feitas em uma calculadora podem ser realizadas na linha de comandos do R.

```
10 + 2 + 4  
#> [1] 16  
# Exemplo de divisao  
(5 + 14)/2  
#> [1] 9.5  
# exponenciação
```

```

2^3
#> [1] 8
4^0.5
#> [1] 2
# operador aritmético para se determinar o resto de uma divisao
10 %% 2
#> [1] 0
2001 %% 2
#> [1] 1
# operador de divisão inteira
11 %/% 2
#> [1] 5

```

Note no R, o separador decimal é o ponto (".").

Conheça mais operadores aritméticos, digitando na linha de comando:

```
? "Arithmetic"
```

A janela que se abrirá mostrará outros operadores aritméticos disponíveis com o R. O texto mostrado faz parte do manual de ajuda do R.

4.2.2 Constantes

O R possui algumas constantes pré-definidas, como o a constante pi ().

```

pi
#> [1] 3.141593

```

O R também trabalha com caracteres, alguns vetores de caracteres pré-definidos são:

```

LETTERS
#> [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q"
#> [18] "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
letters
#> [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
#> [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
month.abb
#> [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov"
#> [12] "Dec"
month.name
#> [1] "January" "February" "March" "April" "May"
#> [6] "June" "July" "August" "September" "October"
#> [11] "November" "December"

```

Note que caracteres estão sempre entre aspas: "".

“caracteres são entre aspas”

```

aeiou
#> Error in eval(expr, envir, enclos): object 'aeiou' not found

"aeiou"
#> [1] "aeiou"

```

4.2.3 Funções matemáticas internas

Existem diversas funções internas do R que permitem, por exemplo, sortear números aleatoriamente, arredondar números, calcular o fatorial, calcular o seno, cosseno de um ângulo e etc. A sintaxe para chamar uma função no R é:

`função(argumento)`

Por exemplo:

```
# funções trigonométricas
sin(pi/6)
#> [1] 0.5
cos(pi)
#> [1] -1
# raiz quadrada
sqrt(100)
#> [1] 10
# exponencial
exp(1)
#> [1] 2.718282
# fatorial
factorial(4)
#> [1] 24
```

No R você verá que parênteses são frequentemente utilizados. Eles são sempre associados à funções. Qualquer palavra antecedendo um parênteses é uma função.

Para ver a lista completa de funções trigonométricas:

```
? "Trig"
```

4.2.4 Valores numéricos especiais

Um caso particular sobre operação aritméticas no R, são os valores numéricos `Inf` e `NaN` que resultam de operações como:

```
2/0
#> [1] Inf
-12/0
#> [1] -Inf
exp(-Inf)
#> [1] 0
log(0)
#> [1] -Inf
0/Inf
#> [1] 0
(0:3)^Inf
#> [1] 0 1 Inf Inf
log(-0.5)
#> Warning in log(-0.5): NaNs produced
#> [1] NaN
sqrt(-1)
#> Warning in sqrt(-1): NaNs produced
#> [1] NaN
0/0
#> [1] NaN
```

Table 4.1: Tabela 1. Operações com NA.

operação	resultado
NA + 5	NA
sqrt(NA)	NA
NA^2	NA
NA/NaN	NA

```
Inf-Inf
#> [1] NaN
Inf/Inf
#> [1] NaN
mean(c(NA, NA), na.rm = TRUE)
#> [1] NaN
```

NaN é a abreviação para *Not a Number*. Geralmente surge quando um cálculo não tem sentido matemático ou não pode ser propriamente realizado.

A demonstração das diferentes formas de se obter essas constantes especiais é importante para entender a origem delas durante a execução de um script mais extenso.

Outra constante especial do R é o NA (*Not Available*) que representa valor faltante, um problema comum em análise de dados. Qualquer operação envolvendo NA resultará em NA (Tabela 1).

4.2.5 Notação científica e número de dígitos

Na maioria das vezes precisamos trabalhar com números grandes e consequentemente acabamos usando uma notação científica ou exponencial. No R há diferentes formas de representar números com expoentes:

```
1.2e-6
#> [1] 1.2e-06
# expressões equivalentes
1.2E6; 1.2*10^6
#> [1] 1200000
#> [1] 1200000
```

Os resultados dos cálculos no R são mostrados com 7 dígitos significativos, o que pode ser verificado pela `getOption()`. É possível mudar para `n` dígitos usando a função `options()`, conforme exemplo abaixo.

```
# opção de dígitos padrão
getOption("digits")
#> [1] 7
exp(1)
#> [1] 2.718282
# alterando para 14
options(digits = 14)
exp(1)
#> [1] 2.718281828459
getOption("digits")
#> [1] 14
# redefinindo para o número de casas decimais padrão
options(digits = 7)
getOption("digits")
#> [1] 7
```

4.3 Variáveis

4.3.1 Formas de atribuição

4.3.1.1 Variável recebe valor

Até agora nós usamos expressões para fazer uma operação e obter um resultado. O termo "expressão" significa uma sentença de código que pode ser executada. Se a avaliação de uma expressão é salva usando o operador `<-`, esta combinação é chamada "atribuição". O resultado da "atribuição" é armazenado em uma variável e pode ser utilizado posteriormente. Então uma variável é um nome usado para guardar os dados.

```
variavel <- valor
```

```
p <- 1013
# para mostrar a variável digite o nome da variável
p
#> [1] 1013
# ou use a função print()
print(p)
#> [1] 1013
```

O R diferencia letras maiúsculas de minúsculas. Portanto p e P são variáveis diferentes.

```
p
#> [1] 1013
P
#> Error in eval(expr, envir, enclos): object 'P' not found
```

Como criamos apenas a variável p, P não foi encontrada.

A variável p pode ser utilizado para criar outras variáveis.

```
p_pa <- p * 100
# pressão em Pascal
p_pa
#> [1] 101300
```

A seta de atribuição pode ser usada em qualquer sentido. Parênteses, além de estarem sempre acompanhando uma função, também são usados para indicar a prioridade dos cálculos.

```
7/3 + 0.6 -> y1
y1
#> [1] 2.933333
7/(3 + 0.6) -> y2
y2
#> [1] 1.944444
```

Os espaços em torno do símbolo de atribuição (`<-`) não são obrigatórios mas eles ajudam na legibilidade do código.

```
x <- 1
x < -1
# atribuição ou menor que?
x<-1
```

Vamos criar uma variável chamada `ndias3` que recebe o nº de dias no mês de Março e `ndias4` que recebe o nº de dias no mês de Abril.

```
nd3 <- 31
nd4 <- 30
```

O total de dias nos meses de março e abril será armazenado na variável `totdias`:

```
totd <- nd3 + nd4
totd
#> [1] 61
```

A atribuição de um mesmo valor para diferentes variáveis pode ser feita da seguinte forma:

```
# número de dias em cada mês
jan <- mar <- mai <- jul <- ago <- out <- dez <- 31
abr <- jun <- set <- nov <- 30
fev <- 28
# verificação
jan; jul
#> [1] 31
#> [1] 31
jun; set
#> [1] 30
#> [1] 30
fev
#> [1] 28
```

Nós estamos definindo a variável, digitando o nome dela na linha de comando e teclando enter para ver o resultado. Há uma forma mais prática de fazer isso e mostrar o resultado cercado a atribuição por parênteses:

```
# ao invés de
# tar <- 20
# tar
# é mais prático
(tar <- 20)
#> [1] 20
```

Se desejamos calcular e já visualizar o valor da pressão de vapor de saturação obtida com a equação de Tetens, podemos fazer:

```
(es <- 0.611 * exp((17.269 * tar)/(tar + 237.3)))
#> [1] 2.338865
```

Quando usamos a mesma variável numa sequência de atribuições o seu valor é sobrescrito. Portanto não é bom usar nomes que já foram usados antes, exceto se a intenção for realmente essa. Para saber os nomes das variáveis já usados use a função `ls()`¹ para verificar as variáveis existentes:

```
ls()
#> [1] "abr"      "ago"      "dez"      "es"      "fev"      "jan"
#> [7] "jul"      "jun"      "mai"      "mar"      "nd3"      "nd4"
#> [13] "nov"      "oper_nas" "out"      "p"       "pcks"     "p_pa"
#> [19] "rblue"    "set"      "tar"      "totd"     "y1"       "y2"
```

```
totd <- jan*7; totd <- totd + fev; totd <- totd + 4*abr
totd
#> [1] 365
```

4.3.1.2 Atribuição com a função `assign()`

Outra forma de atribuição é através da função `assign()`:

¹Essa lista de variáveis também é mostrada no painel *Environment* do RStudio (canto direito superior, aba *Environment*).

```

es
#> [1] 2.338865
assign(x = "es_hpa", value = es/10)
es_hpa
#> [1] 0.2338865
# usando função assign sem nome dos parâmetros
assign("u", 2.5)
u
#> [1] 2.5

```

Um exemplo mais elaborado de uso da função `assign()` para criar várias variáveis pode ser visto aqui.

4.3.2 Removendo variáveis

Para remover variáveis usa-se a função `rm()`.

```

# lista de variáveis existentes
ls()
#> [1] "abr"      "ago"      "dez"      "es"      "es_hpa"   "fev"
#> [7] "jan"      "jul"      "jun"      "mai"     "mar"     "nd3"
#> [13] "nd4"      "nov"      "oper_nas" "out"     "p"       "pcks"
#> [19] "p_pa"     "rblue"    "set"      "tar"     "totd"    "u"
#> [25] "y1"      "y2"

```

Vamos remover a variável `u` criada previamente e ver a lista de objetos no espaço de trabalho.

```

rm(u)
# lista de variáveis existentes, sem u
ls()
#> [1] "abr"      "ago"      "dez"      "es"      "es_hpa"   "fev"
#> [7] "jan"      "jul"      "jun"      "mai"     "mar"     "nd3"
#> [13] "nd4"      "nov"      "oper_nas" "out"     "p"       "pcks"
#> [19] "p_pa"     "rblue"    "set"      "tar"     "totd"    "y1"
#> [25] "y2"

```

Podemos remover mais de uma variável ao mesmo tempo.

```

rm(es_hpa, es, tar, y1, y2)
# lista de variáveis existentes, sem es_hpa, es, tar, y1, y2
ls()
#> [1] "abr"      "ago"      "dez"      "fev"     "jan"     "jul"
#> [7] "jun"      "mai"      "mar"      "nd3"     "nd4"     "nov"
#> [13] "oper_nas" "out"      "p"        "pcks"    "p_pa"    "rblue"
#> [19] "set"      "totd"

```

Para remover todas variáveis do espaço de trabalho (use com cautela):

```

# apagando tudo
rm(list = ls())
ls()
#> character(0)

```

4.3.3 Nomeando variáveis

É preciso ter cuidado ao nomear variáveis no R porque existem algumas regras:

- não iniciar com um número e não conter espaços

```
1oAno <- 1990
raizDe10 <- sqrt(2)
variavel teste <- 67
```

```
# nomes alternativos para as variaveis
ano1 <- 1990
variavel_teste <- 67
variavel.teste <- 68
```

- não conter símbolos especiais:

^, !, \$, @, +, -, /, ou *

```
dia-1 <- 2
#> Error in dia - 1 <- 2: object 'dia' not found
# alternativa
dia_1 <- 2
```

- evitar o uso de nomes usados em objetos do sistema (funções internas do R ou constantes como o número):

```
c q s t C D F I T diff exp log mean pi range rank var

FALSE Inf NA NaN NULL TRUE
```

```
break else for function if in next repeat while
```

- variáveis com acento são permitidas mas não recomendadas.

```
verão <- "DJF"
verão
#> [1] "DJF"
```

Uma boa prática de programação é usar nomes informativos para as variáveis para legibilidade do código. Uma boa referência para isso é a seção **Sintaxe** do Guia de estilo tidyverse (ou universo arrumado).

Chapter 5

Tipos de dados

Nesta seção vamos:

- conhecer os tipos de dados mais usados no R
- descobrir qual é o tipo de dado de uma variável
- aprender a fazer testes com operadores lógicos
- saber como converter uma variável de um tipo para outro

faltando:

- fórmulas
- factor

5.1 Classes de dados

Existem várias classes de dados no R. As mais utilizadas são:

- `numeric` (números)
- `character` (sequência de caracteres)
- `logical` (TRUE/FALSE)
- `Date` (datas)
- `POSIXct` (datas e horários)

A classe dos dados de um objeto é verificada com a função `class()`.

```
> x <- 51
> class(x)
[1] "numeric"
```

5.1.1 *numeric*

É a classe de objeto mais usada. Essa classe é similar a *float* ou *double* em outras linguagens. Ela trata de inteiros e decimais, positivos e negativos e zero. Um valor numérico armazenado em um objeto é automaticamente assumido ser numérico. Para testar se um objeto é numérico usa-se a função `is.numeric()`.

```
> is.numeric(x)
[1] TRUE
> is.numeric(pi)
[1] TRUE
```

Outro tipo é o `integer` (inteiro), ou seja não há parte decimal. Para definir um objeto como inteiro é necessário acrescentar ao valor numérico um `L`. Analogamente, uma forma de verificação se o objeto é inteiro é através função `is.integer()`.

```
> i <- 3L
> is.integer(i)
[1] TRUE
> is.integer(pi)
[1] FALSE
```

Mesmo com o objeto `i` sendo inteiro, ele também passa na verificação `is.numeric()`.

```
> is.numeric(i)
[1] TRUE
```

O R converte inteiros para numéricos quando necessário. Vamos usar a função `typeof()` para determinar o tipo de dado e as conversões que o R faz. Por exemplo:

```
> ## integer * numeric
> typeof(5L)
[1] "integer"
> typeof(4.5)
[1] "double"
> (prod_i <- 5L * 4.5)
[1] 22.5
> typeof(prod_i)
[1] "double"
> ## integer/integer
> typeof(5L)
[1] "integer"
> typeof(2L)
[1] "integer"
> typeof(5L/2L)
[1] "double"
> # número complexo
> typeof(3 + 2i)
[1] "complex"
```

5.1.2 *character*

O tipo de dado *character* (*string*) é bastante utilizado e deve ser manipulado com cuidado. No R há duas principais formas de lidar com caracteres: a função `character()` e `factor()`. Embora pareçam similares eles são tratados de forma diferente.

```
> (char <- "Vai chover hoje?")
[1] "Vai chover hoje?"
> charf <- factor("Vai chover hoje?")
> charf
[1] Vai chover hoje?
Levels: Vai chover hoje?
> levels(charf)
[1] "Vai chover hoje?"
> ordered(charf)
[1] Vai chover hoje?
Levels: Vai chover hoje?
```

`char` contém as palavras "Vai chover hoje?", enquanto, `charf` tem as mesmas palavras porém sem as aspas e a segunda linha de informação sobre os níveis (*levels*) de `charf`. Nós veremos esse tipos de dado futuramente em vetores.

Lembre-se que caracteres em letras minúsculas e maiúsculas são coisas diferentes no R.

Para encontrar o tamanho de um `character` usamos a função `nchar()`.

```
> nchar(char)
[1] 16
> nchar("abc")
[1] 3
```

Esta função não funcionará para um objeto do tipo `factor`.

```
> nchar(charf)
Error in nchar(charf): 'nchar()' requires a character vector
```

5.1.3 *logical*

`logical` (lógico) é uma forma de representar dados que podem assumir valores booleanos, isto é, **TRUE** (verdadeiro) ou **FALSE** (falso).

```
> # variável lógica
> v1 <- FALSE
```

Então em operações aritméticas envolvendo dados lógicos eles serão convertidos numericamente para 1 (TRUE) e 0 (FALSE).

```
> v1 * 5
[1] 0
> TRUE * 5
[1] 5
> TRUE + TRUE
[1] 2
> FALSE - TRUE
[1] -1
```

Assim como as outras classes de dados existem funções para verificar a classe de dados lógicos.

```
> class(v1)
[1] "logical"
> is.logical(v1)
[1] TRUE
```

O R aceita as abreviaturas T e F para representar TRUE e FALSE, respectivamente, mas não é recomendado usá-las, conforme exemplo abaixo.

```
TRUE
[1] TRUE
T
[1] TRUE
class(T)
[1] "logical"
T <- 10
class(T)
[1] "numeric"
```

Valores lógicos resultam da comparação de números ou caracteres.

Table 5.1: Tabela 1. Operadores Lógicos

Operador	Descrição
<	menor que
<=	menor ou igual a
>	maior que
>=	maior ou igual
==	idêntico
!=	diferente
!x	não é x (negação)
x y	x ou y
x & y	x e y
isTRUE(x)	teste se x é verdadeiro
%in%	está contido em

```

> 4 == 3 # 4 é idêntico a 3?
[1] FALSE
> teste2i2 <- 2*2 == 2+2
> teste2i2
[1] TRUE
> teste2d2 <- 2*2 != 2+2 # operador: diferente de
> teste2d2
[1] FALSE
> 4 < 3
[1] FALSE
> 4 > 3
[1] TRUE
> 4 >= 3 & 4 <= 5
[1] TRUE
> 4 <= 3 | 4 <= 5
[1] TRUE
> "abc" == "defg"
[1] FALSE
> "abc" < "defg"
[1] TRUE
> nchar("abc") < nchar("defg")
[1] TRUE

```

5.1.4 Date

Lidar com datas e horários pode ser difícil em qualquer linguagem e pode complicar mais ainda quando há diversas opções de classes de datas disponíveis, como no R.

As mais úteis são:

- Date
- POSIXct

Date armazena apenas a data enquanto POSIXct armazena a data e o horário. Ambos dados são representados como o número de dias (Date) ou segundos (POSIXct) decorridos desde 1 de Janeiro de 1970.

```

> data1 <- as.Date("2012-06-28")
> data1

```

```
[1] "2012-06-28"
> class(data1)
[1] "Date"
> as.numeric(data1)
[1] 15519
> data2 <- as.POSIXct("2012-06-28 17:42")
> data2
[1] "2012-06-28 17:42:00 -03"
> class(data2)
[1] "POSIXct" "POSIXt"
> as.numeric(data2)
[1] 1340916120
```

A manipulação de dados da classe de datas e horários (**Date-time**) torna-se mais versátil através dos pacotes `lubridate` e `chron`, o que será visto posteriormente no curso.

Funções como `as.numeric()` e `as.Date()` não apenas mudam o formato de um objeto mas muda realmente a classe original do objeto.

```
> class(data1)
[1] "Date"
> class(as.numeric(data1))
[1] "numeric"
```

5.2 Testes sobre tipos de dados

Além função `typeof()`, a família `is.*()` também permite descobrir o tipo de dado, p.ex.: `is.numeric()`, `is.character()` e etc.

```
> x; typeof(x)
[1] 51
[1] "double"
> vl; typeof(vl)
[1] FALSE
[1] "logical"
> data1; typeof(data1)
[1] "2012-06-28"
[1] "double"
> x; is.numeric(x)
[1] 51
[1] TRUE
> # num.real?
> is.double(x/5)
[1] TRUE
> is.double(5L)
[1] FALSE
> is.character("12.34")
[1] TRUE
> charf; is.factor(charf)
[1] Vai chover hoje?
Levels: Vai chover hoje?
[1] TRUE
> i; is.integer(i)
[1] 3
```

```
[1] TRUE
> is.function(sqrt)
[1] TRUE
> is.finite(i)
[1] TRUE
> is.nan(x)
[1] FALSE
> is.na(x)
[1] FALSE
```

5.3 Conversão entre tipos de dados

Em algumas circunstâncias precisamos alterar o tipo de uma variável. A maioria das funções `is.*()` possui uma função `as.*()` correspondente de conversão para aquele tipo de dado.

```
> # de character para numeric
> as.numeric("12.34")
[1] 12.34
> # ou
> as("12.34", "numeric")
[1] 12.34
> # de factor para character
> as.character(charf)
[1] "Vai chover hoje?"
> # character para factor
> as.factor("a")
[1] a
Levels: a
> # de double para integer
> typeof(x)
[1] "double"
> typeof(as.integer(x))
[1] "integer"
> as.integer(x) == 51L
[1] TRUE
> as.integer("12.34")
[1] 12
> # arredondamento
> as.integer(12.34)
[1] 12
> # lógico para inteiro
> as.integer(TRUE)
[1] 1
> # numérico para lógico
> as.logical(0:2)
[1] FALSE TRUE TRUE
> # character para numérico?
> as.numeric("a")
Warning: NAs introduced by coercion
[1] NA
> # de character para date
> dt_char <- "2016-03-17"
```

```
> dt <- as.Date(dt_char)
> dt
[1] "2016-03-17"
> # de character para date-time
> data_hora <- as.POSIXct("2016-03-17 15:30:00")
> data_hora
[1] "2016-03-17 15:30:00 -03"
```


Bibliography

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.