

Introdução à Programação em



R Studio®

Apresentações

O QUE É LUGAR DE FALA?



A **FALA** PRECISA
DA **ESCUTA**
PARA VIRAR
DIÁLOGO



Djamila Ribeiro



Minha Jornada em
Analytics foi iniciada em

2006

UFPE



USP



KANTAR



CONRE



R-Ladies



Insper



IME-USP



Learning



Trevisan



nutes

IBOPE

vivo

cabify

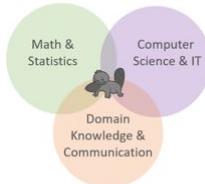
itau

**Nathália
Demetrio**



<https://bit.ly/NathaliaDemetrio>

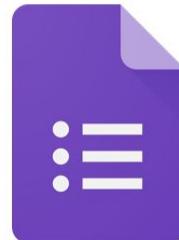
Quase **20 anos** depois
atuo como professora e consultora
em Data & Advanced Analytics



E vocês?



Forms de Check In no curso



Acesse o link: <https://forms.gle/7huwE9Tp52f8nLLp9>

Check in - Curso de Introdução à Programação em R (2025)

Este formulário tem como objetivo:

- 1) levantar o perfil dos alunos do curso "Introdução à Programação em R", do "Programa De Verão do IME USP", por meio de perguntas sobre filiação, e conhecimentos prévios, visando a construção de um curso mais alinhado aos interesses de todas e todos; e
- 2) registrar um e-mail por pessoa, visando o posterior compartilhamento de quizzes, projetos, materiais, e avaliações. Sendo, portanto, imprecindível que todos e todas utilize O MESMO e-mail em todos os forms.

Qualquer dúvida estou à disposição,
Nathália Demetrio

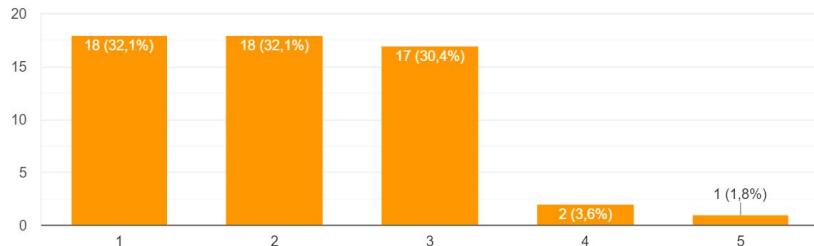
* Indica uma pergunta obrigatória

Conhecimento em Programação

Qual o seu nível de conhecimento em linguagem R?

56 respostas

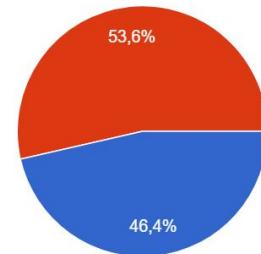
 Copiar



Você sabe programar (escrever linhas de código) em alguma linguagem de programação que não seja o R?

56 respostas

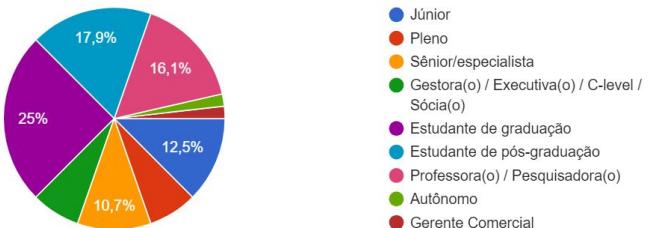
- Sim
- Não



Qual das opções abaixo melhor descreve o seu perfil profissional atual?

56 respostas

 Copiar



- Júnior
- Pleno
- Sênior/especialista
- Gestora(o) / Executiva(o) / C-level / Sócia(o)
- Estudante de graduação
- Estudante de pós-graduação
- Professora(o) / Pesquisadora(o)
- Autônomo
- Gerente Comercial

Sobre os Conteúdos

Serão constantemente atualizados aqui:

The screenshot shows a GitHub repository page for 'natydem/i/intro_R_language'. The repository is public and has one branch ('master') and one tag ('1'). The commit history on the master branch includes updates to HTML files, relocate scripts, update slides, relocate slides, and rename files. A dark purple rectangular overlay covers the right side of the page, containing the following text:

Introdução à Programação em R
Programa De Verão

Nathália Demetrio
jan/2024

1/10

Link: https://github.com/natydem/i/intro_R_language

Alinhamentos sobre o curso

Sobre as aulas EAD

- As aulas acontecerão na plataforma Moodle, e serão gravadas. Os arquivos são disponibilizados algumas horas após o fim da aula na mesma seção em que acessamos a sala (***)
- Importante: até 2023 as gravações expiravam após 3 meses! (***)

(***) a plataforma Moodle passou por alterações recentes, talvez hajam modificações nos períodos citados

The screenshot shows the USP OPUS Moodle LMS interface. At the top, there are three cards: 'Acesse aqui: Sala de Aula, Gravações e Slides' (Access here: Classroom, Recordings and Slides), 'Boas-vindas e 1ª atividade!' (Welcome and first activity!), and 'Ótimo para estatísticas' (Great for statistics) with a link to 'Aula 1 - Da instalação à primeira análise no R'. Below these is a large card for 'Acesse aqui: Sala de Aula, Gravações e Slides'. It contains text about accessing the classroom during official hours and links to various resources: 'Sala de aula remota – acesse aqui a aula!', 'Link para o nosso board de interação (easyretro)', 'Link para apresentação (google slides) utilizado em aula', and 'Link para acesso aos scripts utilizados em aula (github)'. A red arrow points to the 'Acesse aqui' button. Another red arrow points to the 'Sala de aula remota – acesse aqui a aula!' button on the 'Sala de aula remota' page. A third red arrow points to the 'Gravadas' (Recorded) tab in the 'Histórico' (History) section of the same page.

Registros de participação

- O próprio estudante deve registrar a sua presença durante o horário da aula, na seção “Registro de Presença”, na página inicial do curso – no caso de não ser registrado nenhuma entrada na sala de aula do Moodle, a presença será desconsiderada
- Avaliação – este curso não tem nota, temos apenas o acompanhamento das presença e a entrega dos quizzes propostos – pelo menos 1 quizz por aula
- Os forms de Check-in/out não são obrigatórios, mas são um indicativo de participação! Além de serem super importantes para acompanhar a evolução dos alunos e direcionar o curso! Por favor, não deixem de preenchê-los! ♥

Boas Vindas ao Programa de Verão 2024

Para mais informações sobre este curso acesse:
<https://www.ime.usp.br/verao/index.php/turmas/descricao/486>

Professora:

- Nathália Demetrio Vasconcelos Moura

Monitores:

- Agatha Priscila Alves Nogueira
- Alan da Silva
- Beatriz Proença Carvalho
- Natan Sant Anna Borges

Aulas:

de 09 a 30/01/2023, terças e quintas, das 19:00 às 21:30



Presença – para registro da presença dos estudantes



Avisos – para a professora fazer os informes do curso (alunos podem adicionar comentários)



Fórum do curso – área para interação entre todos! Utilizem à vontade



Monitorias e canais de comunicação

- Monitorias:
 - 40 minutos antes de cada aula – no caso, entre 18h10 e 18h50
 - 30 minutos após cada aula – entre 21h30 e 22h
 - Sob demanda às 2^a, 4^a e 6^a, entre 19 e 20h
- Meios de Comunicação oficiais:
 - Seção “Avisos” no Moodle: a professora fará os Informes – alunos podem add comentários
 - Seção “Fórum do Curso” no Moodle: é de vocês! Utilizem à vontade
 - Mural de Interação (easy retro): para dar mais dinâmica às aulas e para envio de perguntas assíncronas: [link de acesso](#)
 - grupo whats: para network e troca entre os alunos das turmas de 2023 e 2024 [link convite](#)
- Caso tenha algum problema relacionado ao curso, recomendo
 - envio de e-mail para: **verao@ime.usp.br**, com copia para: **cursoverao@gmail.com**

Emissão de Certificado

É necessário ter um percentual de presença nas aulas e de entrega dos quizzes. No caso:

- presença em pelo menos 5 das 6 aulas (83% de presença) e entrega de pelo menos 4 dos quizzes propostos

OU

- presença em 4 das 6 aulas (66% de presença) somado com a entrega de pelo menos 6 quizzes propostos (teremos 1 quizz “oficial” por aula, casualmente teremos algum quizz extra)

Em relação ao quizzes

- devem ser entregues ANTES das correções (início da aula seguinte).

Nome ↑

23.Q1 - objetos e atribuições

23.Q2 - tipos e estruturas de dados

23.Q2.1 - descritivas RBase

23.Q3 - manipulação de dados RBase

23.Q4 - tidyverse ~ 1st steps e dplyr

23.Q4.1 - explorando funções tidyverse

23.Q5 - explorando bibliotecas tidyverse

23.Q5.1 - ggplot2

23.Q6 (sala) - investigando uma base de dados

23.Q6.1 - recursos de programação R

Check in - Intro Programação R (2023) ↗

Check out - Intro Programação R (2023)

Antes de começar... Uma dica sobre o processo!

and be able to see how the various layers, shapes, and data are pieced together to make a finished plot.

The Right Frame of Mind

It can be a little disorienting to learn a programming language like R, mostly because at the beginning there seem to be so many pieces to fit together in order for things to work properly. It can seem like you have to learn everything before you can do anything. The language has some possibly unfamiliar concepts that define how it works, like “object,” “function,” or “class.” The syntactic rules for writing code are annoyingly picky. Error messages seem obscure; help pages are terse; other people seem to have had *not quite* the same issue as you. Beyond that, you sense that doing one thing often involves learning a bit about some other part of the language. To make a plot you need a table of data, but maybe you need to filter out some rows, recalculate some columns, or just get the computer to see it is there in the first place. And there is also a wide environment of supporting applications and tools that are good to know about but involve new concepts of their own—editors that highlight what you write; applications that help you organize your

code and its output; ways of writing your code that let you keep track of what you have done. It can all seem a bit confusing. Don’t panic. You have to start somewhere. Starting with graphics is more rewarding than some of the other places you might begin, because you will be able to see the results of your efforts very quickly. As you build your confidence and ability in this area, you will gradually see the other tools as things that help you sort out some issue or solve a problem that’s stopping you from making the picture you want. That makes them easier to learn. As you acquire them piecemeal—perhaps initially using them without completely understanding what is happening—you will begin to see how they fit together and be more confident of your own ability to do what you need to do.

[Data Visualization](#), by Kieran Healy

O ambiente R

R – O que é?

O R é uma linguagem de programação interpretada, que oferece larga, coerente e integrada coleção de recursos para todo o workflow de análise de dados, desde a importação e manipulação de dados até a modelagem estatística/machine learning e comunicações técnicas.



R – Origem

Criada em 1993, no departamento de Estatística da Universidade de Auckland - Nova Zelândia, por Ross Ihaka e Robert Gentleman, a linguagem R é uma implementação da linguagem de programação S – tendo seu nome inspirado tanto na inicial de seus autores, quanto como uma provação à linguagem S. Atualmente é mantida pelo R Foundation for Statistical Computing e R Core Group.



R – Por que usar?

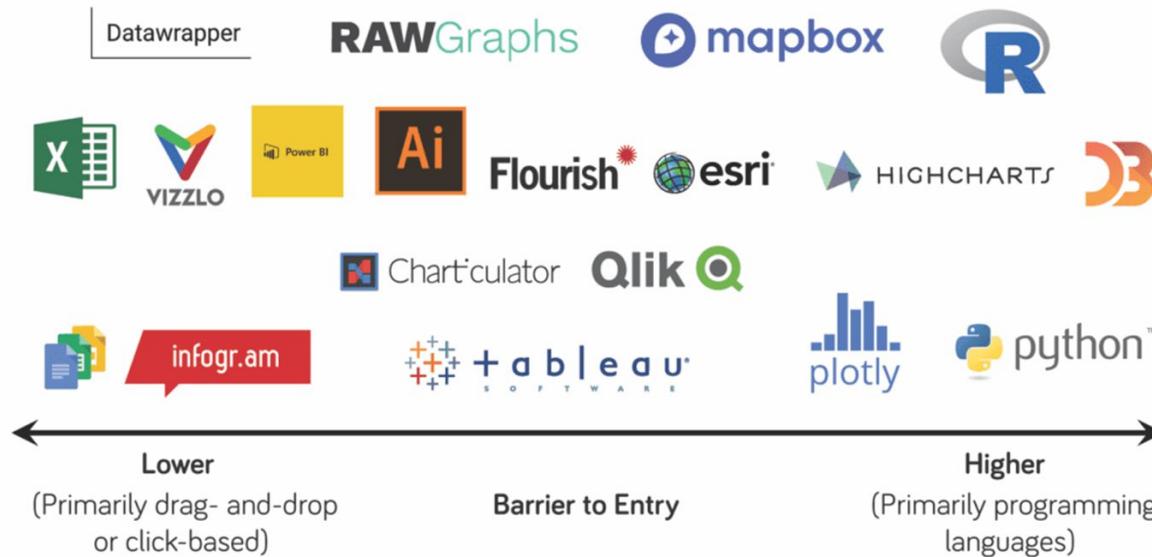
- Muitas facilidades voltadas à análise de dados (+18K bibliotecas no CRAN, fora outros repositórios como Bioconductor, R-Forge, R-Hub e Github)
- Diminui a barreira de uso de recursos avançados para iniciantes
- Porta de entrada para novas metodologias no contexto de analytics
- Comunidade com uma cultura de promoção à diversidade considerando níveis de conhecimento, áreas e públicos em geral
- Stickers!



Olha o meu note! ❤



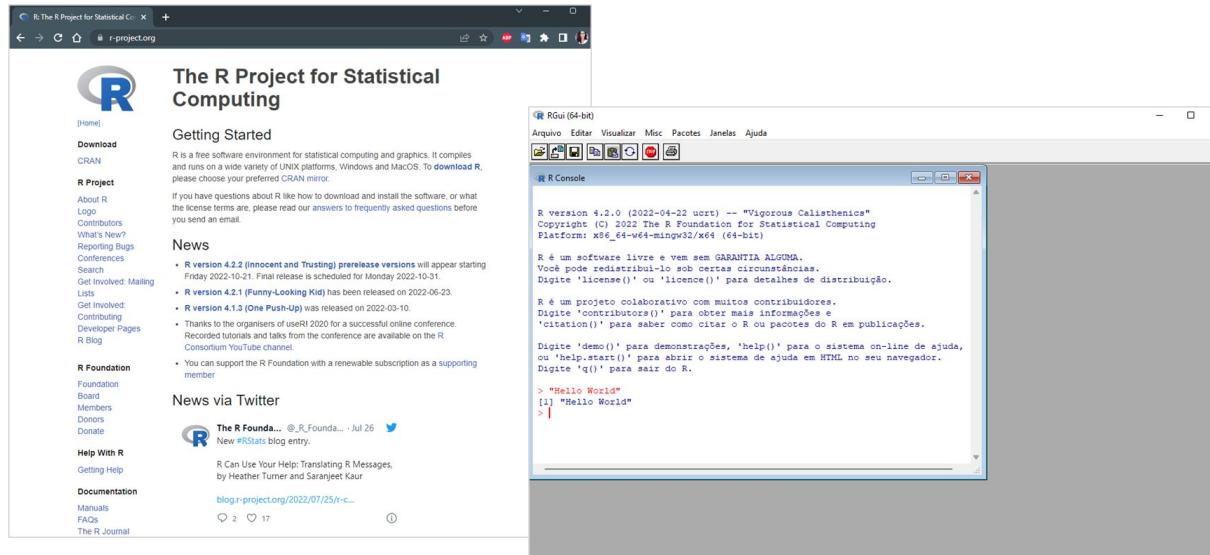
Bônus: comunidades em diferentes áreas (#dataviz)



Referência: <https://policyviz.com/2022/02/01/the-data-visualization-tools-wars/>

R – Interface Básica

O R pode ser instalado acessando: <http://www.r-project.org/>. Tal página inclui desde tutoriais e listas de mailing, até a lista de eventos de interesse e o mapeamento de upgrades:



A IDE RStudio

R – IDE

Como alternativa à interface básica, podemos trabalhar com um dos muitos Ambientes de Desenvolvimento Integrados (IDEs) disponíveis:



Em 2022 a empresa RStudio passou a chamar Posit – mas a IDE segue RStudio

R – a IDE RStudio

RStudio Desktop 2022.07.2+576 - Release Notes [View](#)

1. Install R. RStudio requires R 3.3.0+ [Get](#).
2. Download RStudio Desktop. Recommended for your system:

[DOWNLOAD RSTUDIO FOR WINDOWS](#)
2022.07.2+576 | 190.49MB

Requires Windows 10/11 (64-bit)



All Installers

Linux users may need to import RStudio's public code-signing key [Get](#) prior to installation, depending on the operating system's security policy.
RStudio requires a 64-bit operating system. If you are on a 32 bit system, you can use an older version of RStudio.

OS	Download	Size	SHA-256
Windows 10/11	RStudio-2022.07.2-576.exe	190.49 MB	b30bf925
macOS 10.15+	RStudio-2022.07.2-576.dmg	224.49 MB	35028d02
Ubuntu 18+/Debian 10+	rstudio-2022.07.2-576-amd64.deb	133.19 MB	b7d0c386
Ubuntu 22	rstudio-2022.07.2-576-amd64.deb	134.06 MB	e1c51083
Fedor 19/Red Hat 7	rstudio-2022.07.2-576-x86_64.rpm	103.29 MB	6594c7bf
Fedor 34/Red Hat 8	rstudio-2022.07.2-576-x86_64.rpm	150.13 MB	bccfce754
OpenSUSE 15	rstudio-2022.07.2-576-x86_64.rpm	134.10 MB	a266d996

Options

RStudio theme: Modern

Zoom: 90% ▾

Editor font: Lucida Console

Editor font size: 10

Editor theme: Ambiance

Ambiance
Chaos
Chtz
Clouds
Clouds Midnight
Cobalt
Crimson Editor
David
Dusk
Dreamweaver
Eclipse
Gob
Idle Fingers
Ivy
Ketama

```
# plotting of R objects
plot <- function(x, y, ...)
{
  if (is.function(x) &
    is.null(attr(x, "class")))
  {
    if (missing(y))
      y <- NULL

    # check for 'lab' argument
    hasylab <- function(...)
      all(sapply(names(list(...)), pmatch, "lab")))

    if (hasylab(...))
      plot.function(x, y, ...)

    else
      plot.function(
        x, y,
        ylab = paste(
          deparse(substitute(x)),
          "-(x)",
          ...
        )
      )
    else
      useMethod("plot")
  }
}
```

OK Cancel Apply

R – a IDE RStudio na Posit Cloud!

Google posit cloud

Imagens Vídeos Notícias Livros Voos F

Cerca de 3 630 000 resultados (0,31 segundos)

Posit Cloud
<https://posit.cloud> · Traduzir esta página

Posit Cloud

Friction free data science. **Posit Cloud** (formerly RStudio Cloud) lets you access Posit's powerful set of data science tools right in your browser – no ...

posit™ Cloud

posit Cloud Your Workspace

Spaces

Your Workspace

New Space

New Project

New Project from Template

New RStudio Project

New Jupyter Project

New Project from Git Repository

R – voltando a IDE RStudio

The screenshot displays the RStudio IDE interface with several panes:

- Source**: A code editor pane containing a single line of code: "1 |".
- Environment**: A pane showing the global environment, which is currently empty.
- Console / Terminal / Jobs**: A pane displaying the R startup message and help text. The text includes:

```
R version 4.2.0 (2022-04-22 ucrt) -- "Vigorous Calisthenics"
Copyright (C) 2022 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.
```
- Help**: A pane listing the main help categories: Histórico (History), Arquivos (Files), Gráficos (Plots), Bibliotecas (Packages), and Ajuda (Help).

Introdução ao R

R – Como calculadora

- No Console do R os textos precedidos por `#` tratam-se de comentários, enquanto que os antecedidos por `>` caracterizam resultados - neste material trabalharemos com ` #>` para identificar os resultados
- Antes de cada resultado o R indica a quantidade de elementos da referida linha entre colchetes, `[1]` em todos os exemplos dados

```
#subtração
  7 - 2
  #> [1] 5
#divisão
  9 / 3
  #> [1] 3
#multiplicação
  100 * 10
  #> [1] 1000
#potência
  2^4
  #> [1] 16
#resto da divisão de 10 por 3
  10 %% 3
  #> [1] 1
# parte inteira da divisão de 10 por 3
  10 %/% 3
  #> [1] 3
```

R – Como calculadora

- O R permite ampliar as equações mantendo a notação básica de operações algébricas, como a aplicação hierárquica de parênteses e operações
- somente os parênteses podem ser utilizados nas expressões matemáticas
- os operadores lógicos no R possuem as saídas `TRUE` ou `FALSE`.
- podemos escrever dois comandos em uma única linha utilizando o símbolo `;`

```
#equação ok
(2 * ( 2 * ( 2 * (4-3)))))

#equação não ok
(2 * { 2 * [ 2 * (4-3)]})
#> Error: <text>:4:14: '[' inesperado
#> 3: #equação não ok
#> 4:   (2 * { 2 * [
#>                                ^
```

```
#maior/menor
1 < 0 ; 1 > 0
#> [1] FALSE
#> [1] TRUE
#menor/maior ou igual
1 <= 1 ; 1 >= 1
#> [1] TRUE
#> [1] TRUE
#igual/diferente
1 == 1 ; 1 != 1
#> [1] TRUE
#> [1] FALSE
```

R – Como calculadora

- No caso de comandos incompletos, o R mostrará o símbolo `+`, indicando que o R está aguardando os próximos comandos
- Para iniciar uma nova linha comando, desconsiderando comandos parciais anteriores, basta pressionar a tecla 'Esc'
- no caso de um comando que o R não reconheça, ele retornará uma mensagem de erro, mas basta digitar o comando desejado na sequência

```
#erro
3 % 9
#> Error: <text>:2:5: unexpected input
#> 1: #erro
#> 2:   3 % 9
#>           ^
```

Funções

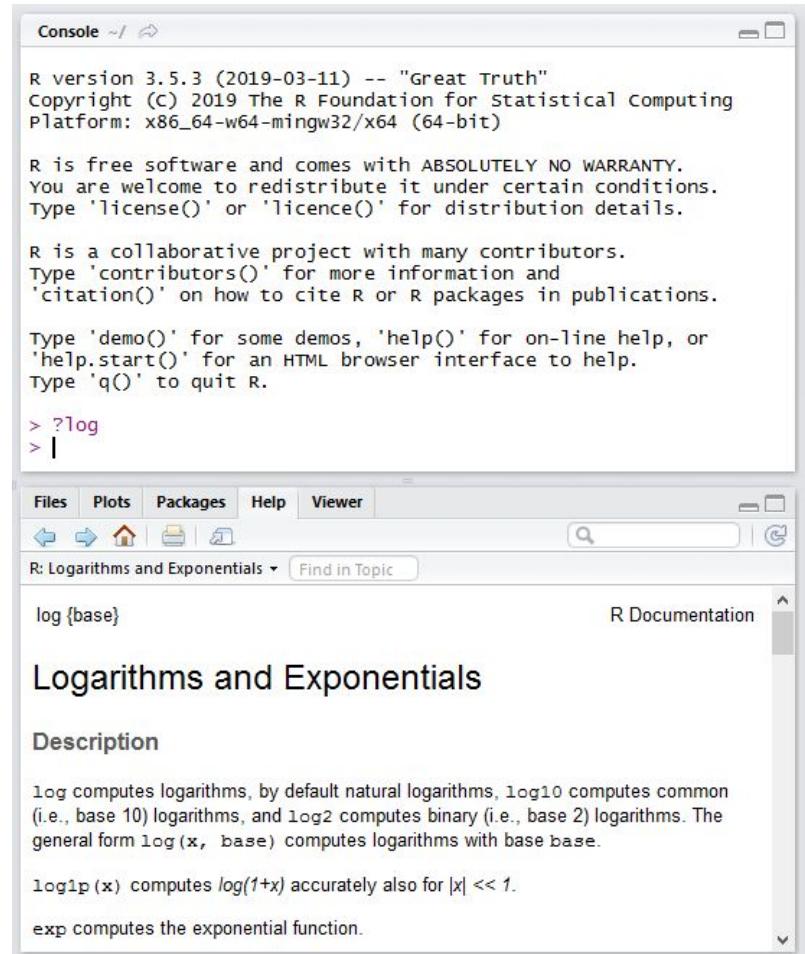
R – Aplicação de funções

- Assim como em muitas linguagens, no R uma função é um conjunto de instruções visando executar uma tarefa
- O R disponibiliza várias funções na sua instalação básica, permitindo desde leitura e organização de dados, até tabelas, gráficos e modelos estatísticos
- O nome de tais funções são em geral intuitivos, como a função `exp()`, por exemplo, que calcula o exponencial de um número

```
#aplicando a função exponencial ao número 1
exp(1)
#> [1] 2.718282
#podemos ver o código da função ao digitar a função sem
os parênteses
exp
#> function (x) .Primitive("exp")
```

R – A função help

- Uma das funções mais importantes no R é a função `help()`, ou ainda `?`, o seu operador equivalente
- Por meio desta função temos acesso à informações referente ao objeto de consulta, como: descrição, possibilidades de alterações, funções relacionadas, exemplos, entre outros
- Consultando o `?log` (`help(log)`), por exemplo, temos que é possível mudar a base segundo a qual o logaritmo é computado, e como fazer isso



The screenshot shows an R console window with the following text:

```
R version 3.5.3 (2019-03-11) -- "Great Truth"
Copyright (C) 2019 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.
```

Below the console, the R Help interface is visible, showing the help page for the `log` function:

Files Plots Packages Help Viewer

R: Logarithms and Exponentials ▾ Find in Topic

log {base}

R Documentation

Logarithms and Exponentials

Description

`log` computes logarithms, by default natural logarithms, `log10` computes common (i.e., base 10) logarithms, and `log2` computes binary (i.e., base 2) logarithms. The general form `log(x, base)` computes logarithms with base `base`.

`log1p(x)` computes $\log(1+x)$ accurately also for $|x| \ll 1$.

`exp` computes the exponential function.

R – Funções e seus argumentos/parâmetros

- Ao alterar a base de cálculo da função `log()` por exemplo, não é necessário especificar o nome do primeiro argumento da função uma vez que o R assume a lógica de ordem.
- Tal prática não é recomendada nos casos em que utilizamos parâmetros não obrigatórios (como foi o caso do argumento base), ou para funções que possuem muitos argumentos.

```
#por default o log é calculado na base exponencial
log(10)
#> [1] 2.302585

#de modo que o resultado não é alterado mesmo se explicitarmos o argumento 'base'
log(10, base = exp(1))
#> [1] 2.302585

#mas podemos alterá-lo para qualquer número positivo,
#como a base 10 por exemplo
log(10, base = 10)
#> [1] 1
```

Objetos

R – Criando objetos

- Objetos no R são elementos que podem ser armazenados em variáveis, isto vale para dados, funções e até expressões
- Para criação de objetos é recomendado trabalhar com `<-`, considerando atribuições da direita para a esquerda
- Quanto a nomeação o R permite: letras (diferenciando maiúsculas e minúsculas), números, e os símbolos `` e `_, desde que o nome não seja iniciado com números ou com um símbolo imediatamente seguido por número

```
#sintaxe: letras minúsculas
objeto <- 1 #da direita para a esquerda (RECOMENDADO)

#sintaxe: letras minúsculas com a primeira letra maiúscula
Objeto = 2 #da direita para a esquerda

#sintaxe: letras minúsculas, simbolo e número
3 -> objeto_1 #da esquerda para a direita
```

R – Objetos especiais

- No R existem alguns nomes reservados para representar os seguintes casos especiais:
 - NA (Not Available): dado faltante ou indisponível (similar ao null do SQL/SAS)
 - NaN (Not a Number): representa indefinições matemáticas, como o log de números negativos
 - Inf (Infinito): conceito matemático (positivo ou negativo)
 - NULL: representa a ausência de informação, utilizada como retorno de funções cujos valores são indefinidos

```
is.na(NA)
#> [1] TRUE
is.nan(NaN)
#> [1] TRUE
is.infinite(Inf); is.infinite(-Inf)
#> [1] TRUE
#> [1] TRUE
is.null(NA)
#> [1] FALSE
```

R – Objetos especiais

- Ao criar objetos passamos a ter a informação que estes carregam salvos na memória, e podemos utilizá-los para operações com outros elementos e/ou objetos
- Para excluirmos um objeto:
`rm(nome_do objeto)`

```
#atribuindo um elemento diretamente
a <- 3; a
#> [1] 3
#resultado de operações entre elementos
b <- 5 + a; b
#> [1] 8
#excluindo o objeto a
rm(a); a
#> Error in eval(expr, envir, enclos): objeto 'a' não encontrado

#operações entre outros objetos
c <- b * (-1); c
#> [1] -8
#resultado da aplicação de funções
d <- abs(c); d
#> [1] 8
```

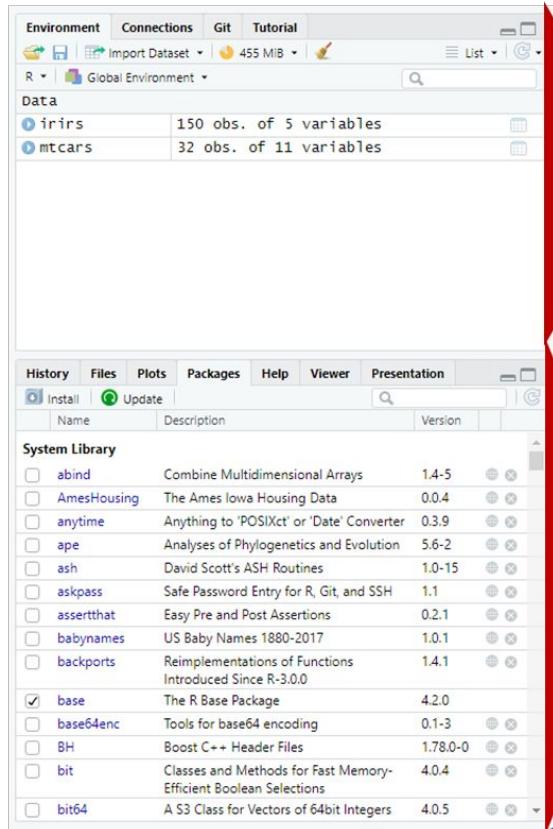
Fim da 1^a
aula!

R – Dica importante!

Citando John Chambers
(criador da linguagem S e membro da
organização que mantêm o R):

“Para entender a programação em R
dois slogans são interessantes:

- ✓ **Tudo o que existe no R é um objeto**
- ✓ **Tudo o que acontece, é uma
chamada de função**



Objetos
carregados na
nossa sessão
atual

As bibliotecas
nos dão acesso
a novas funções



Minha Primeira Análise

R – Análises Descritivas

- Iremos trabalhar com a base mtcars, uma das bases disponíveis por default no R
 - **head()** - que retorna as primeiras linhas da base de dados, ou similarmente,
 - **tail()** - que contempla as últimas linhas

```
head(mtcars)
#>          mpg cyl disp  hp drat    wt  qsec vs am gear carb
#> Mazda RX4     21.0   6 160 110 3.90 2.620 16.46  0  1    4    4
#> Mazda RX4 Wag 21.0   6 160 110 3.90 2.875 17.02  0  1    4    4
#> Datsun 710    22.8   4 108  93 3.85 2.320 18.61  1  1    4    1
#> Hornet 4 Drive 21.4   6 258 110 3.08 3.215 19.44  1  0    3    1
#> Hornet Sportabout 18.7   8 360 175 3.15 3.440 17.02  0  0    3    2
#> Valiant       18.1   6 225 105 2.76 3.460 20.22  1  0    3    1
```

R – Análises Descritivas

- **str()** - exibe a estrutura interna de um objeto, no caso da base de dados que estamos trabalhando p.e., temos: a estrutura dos dados (data.frame), o número de linhas (32 observações) e colunas (11 variáveis), além da classe de cada uma das colunas, e uma amostra das primeiras observações de cada uma das colunas:

```
str(mtcars)
#> 'data.frame': 32 obs. of 11 variables:
#> $ mpg : num 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
#> $ cyl : num 6 6 4 6 8 6 8 4 4 6 ...
#> $ disp: num 160 160 108 258 360 ...
#> $ hp : num 110 110 93 110 175 105 245 62 95 123 ...
#> $ drat: num 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
#> $ wt : num 2.62 2.88 2.32 3.21 3.44 ...
#> $ qsec: num 16.5 17 18.6 19.4 17 ...
#> $ vs : num 0 0 1 1 0 1 0 1 1 1 ...
#> $ am : num 1 1 1 0 0 0 0 0 0 ...
#> $ gear: num 4 4 4 3 3 3 3 4 4 4 ...
#> $ carb: num 4 4 1 1 2 1 4 2 2 4 ...
```

R – Análises Descritivas

- **summary()** - é uma função genérica usada para produzir resumos de resultados segundo várias funções descritivas, no caso de variáveis numéricas, por exemplo:

```
#primeiras colunas
summary(mtcars[,1:4])
#>      mpg          cyl          disp         hp
#> Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   :52.0
#> 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.:96.5
#> Median :19.20   Median :6.000   Median :196.3   Median :123.0
#> Mean    :20.09   Mean    :6.188   Mean    :230.7   Mean    :146.7
#> 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0
#> Max.    :33.90   Max.    :8.000   Max.    :472.0   Max.    :335.0
```

As mesmas informações obtidas poderiam ser acessadas por meio das funções , temos estatísticas como média (`mean()`), mediana (`median()`) ou máximo (`max()`), porém precisariam ser aplicadas a cada uma das colunas - ou por meio de loops, que veremos na próxima seção.

R – Análises Descritivas

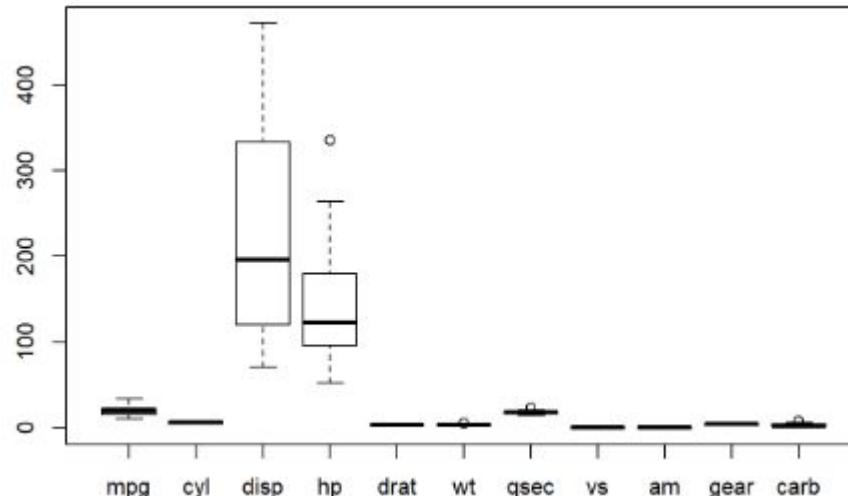
- No que se tivéssemos outras classes de dados no data.frame a função `summary()` retornaria outras estatísticas, como por exemplo:

```
ex <- data.frame(var_logical = c(TRUE,TRUE,FALSE,TRUE,FALSE),
                  var_factor = factor(c("A","A","B","B","C"), ordered = TRUE))
str(ex)
#> 'data.frame': 5 obs. of 2 variables:
#> $ var_logical: logi TRUE TRUE FALSE TRUE FALSE
#> $ var_factor : Ord.factor w/ 3 levels "A"<"B"<"C": 1 1 2 2 3
summary(ex)
#> var_logical    var_factor
#> Mode :logical   A:2
#> FALSE:2          B:2
#> TRUE :3          C:1
```

R – Análises Descritivas

- Para aumentar o grau de entendimento da base, podemos visualizar os dados, por meio do `boxplot()`, que apresenta parte das informações do `summary()` de forma gráfica:

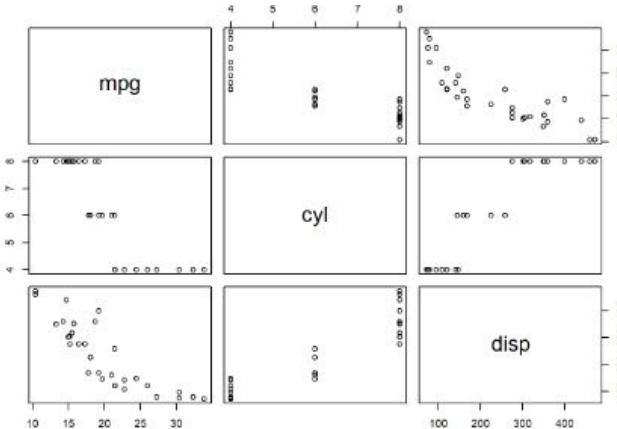
```
boxplot(mtcars)
```



R – Análises Descritivas

- Ou entender a relação entre os dados, considerando, por exemplo a matriz de correlações, que avalia as variáveis dois a dois, tanto do ponto de vista gráfico quanto numérico:

```
#considerando as 3 primeiras colunas:  
#gráfico  
plot(mtcars[,1:3])
```



```
#numérico  
cor(mtcars[,1:3])  
#>          mpg      cyl      disp  
#> mpg  1.0000000 -0.8521620 -0.8475514  
#> cyl  -0.8521620  1.0000000  0.9020329  
#> disp -0.8475514  0.9020329  1.0000000
```

R – Análises Descritivas

- Outro aspecto importante de uma análise diz respeito a transformações que podemos ter o interesse em fazer, como reordenar uma coluna, ou substituir algum elemento.

```
dados_mtcars <- mtcars

#arredondando os dados para uma casa decimal
dados_mtcars$qsec <- round(dados_mtcars$qsec,1)
#substituindo os valores pela descrição que consta no `?mtcars` da variável
dados_mtcars$vs <- factor(dados_mtcars$vs, labels = c("V", "S"))

#assim
str(dados_mtcars)
#> 'data.frame': 32 obs. of 11 variables:
#> $ mpg : num 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
#> $ cyl : num 6 6 4 6 8 6 8 4 4 6 ...
#> $ disp: num 160 160 108 258 360 ...
#> $ hp : num 110 110 93 110 175 105 245 62 95 123 ...
#> $ drat: num 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
#> $ wt : num 2.62 2.88 2.32 3.21 3.44 ...
#> $ qsec: num 16.5 17 18.6 19.4 17 20.2 15.8 20 22.9 18.3 ...
#> $ vs : Factor w/ 2 levels "V","S": 1 1 2 2 1 2 1 2 2 2 ...
#> $ am : num 1 1 1 0 0 0 0 0 0 0 ...
#> $ gear: num 4 4 4 3 3 3 4 4 4 ...
#> $ carb: num 4 4 1 1 2 1 4 2 2 4 ...
```

Trabalhando com Bibliotecas

R – Trabalhando com bibliotecas

- Bibliotecas tratam-se de uma coleção de funções, dados e códigos compilados.
- A instalação destas bibliotecas, ou pacotes, como são também chamados, é simples, podendo ser feita via point-and-click, ou pelo comando `install.packages("nome_do_pacote")`.
- Após a instalação é necessário apenas carregar os pacotes desejados, por meio do comando `library(nome_do_pacote)` - sem aspas duplas. Ou carregando a função de interesse precedida por `::` e o nome da função desejada: `nome_da_biblioteca::função_de_interesse()`.

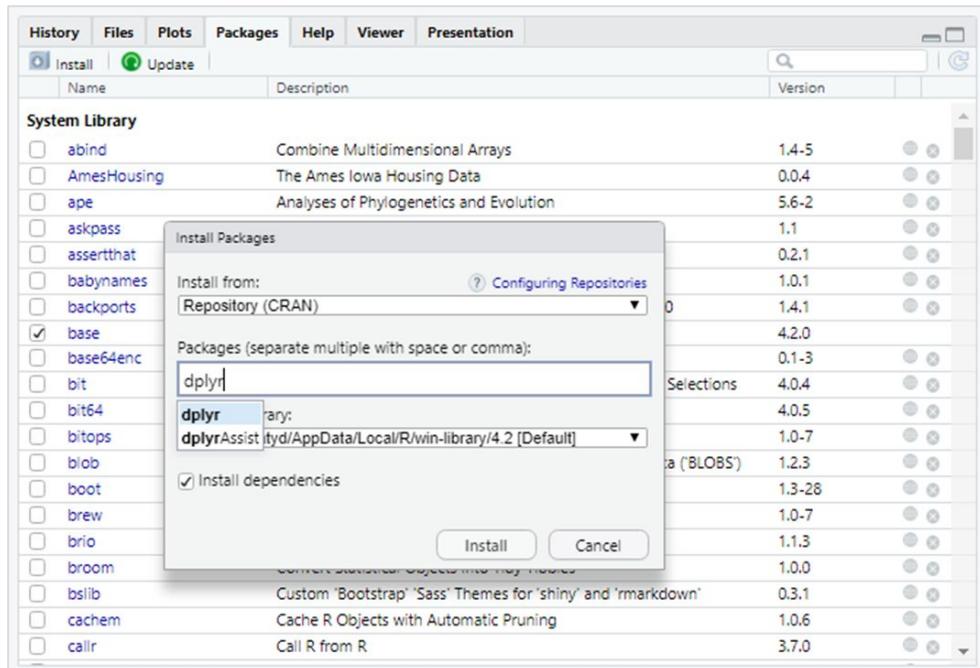
```
#instalação do pacote 'ggplot2'  
install.packages("ggplot2")  
  
#carregando o pacote  
library(ggplot2)
```

R – Trabalhando com bibliotecas

- Tais bibliotecas se encontram disponíveis pela rede de distribuição oficial do R, o CRAN. Mas existem também bibliotecas disponibilizadas via repositórios como o bioconductor, ou o GitHub
- Neste último caso é necessário fazer a instalação com `devtools::install_github()`, após a instalação do pacote `devtools`, ou, no caso de um arquivo zipado, adicionando o parâmetro repos = NULL, na função `install.packages()`

```
#instalação do pacote a partir do CRAN
install.packages("ggplot2")
#instalação a partir do git (versão mais recente da biblioteca)
install.packages("ggplot2")
devtools::install_github("tidyverse/ggplot2")
#instalação a partir de um arquivo .zip
install.packages("C:/Users/ggplot2.zip", repos = NULL)
```

R – Resumo



The screenshot shows the R console window. The session starts with the R version information: 'R version 4.2.0 (2022-04-22 ucrt) -- "Vigorous Calisthenics"'. It then displays a welcome message: 'Copyright (C) 2022 The R Foundation for Statistical Computing Platform: x86_64-w64-mingw32/x64 (64-bit)'. The message continues: '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.' It also notes: '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.' Finally, it provides help instructions: '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.'

```
> install.packages(" ... ")
library()
  abind
  AmesHousing
  ape
  askpass
  assertthat
  babynames
  backports
  base
  base64enc
  bit
```

R – exemplo de uma biblioteca: skimr



```
> skim(mtcars)
Numeric Variables
# A tibble: 11 x 13
   var     type missing complete    n     mean        sd      min `25% quantile` median `75% quantile` max hist
   <chr>   <chr>   <dbl>   <dbl> <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl> <chr>
 1 am     numeric     0       32   0.406250  0.4989909  0.000   0.00000  0.000   0.00000  0.000   1.00  1.000  █
 2 carb   numeric     0       32   2.812500  1.6152000  1.000   2.00000  2.000   4.00000  6.000   4.00  8.000  ████
 3 cyl    numeric     0       32   6.187500  1.7859216  4.000   4.00000  6.000   8.00000  8.000   8.00  8.000  ████
 4 disp   numeric     0       32 230.721875 123.9386938 71.100 120.82500 196.300 326.00 472.000  ████
 5 drat   numeric     0       32   3.596563  0.5346787  2.760   3.08000  3.695   3.92    4.930  3.92  4.930  ████
 6 gear   numeric     0       32   3.687500  0.7378041  3.000   3.00000  4.000   4.00    5.000  4.00  5.000  ████
 7 hp    numeric     0       32 146.687500 68.5628685 52.000  96.50000 123.000 180.00 335.000  ████
 8 mpg   numeric     0       32  20.090625  6.0269481 10.400  15.42500 19.200  22.80  33.900  ████
 9 qsec   numeric     0       32  17.848750  1.7869432 14.500  16.89250 17.710  18.90  22.900  ████
10 vs    numeric     0       32   0.437500  0.5040161  0.000   0.00000  0.000   1.00    1.000  1.00  1.000  █
11 wt    numeric     0       32   3.217250  0.9784574  1.513   2.58125  3.325   3.61    5.424  3.61  5.424  ████
```



Dialectos para manipulação de Dados no R (Rbase, tidyverse e data.table)

R – Dialetos para manipulação de dados no R



R-Base	tidyverse	data.table
Linguagem nativa do R (acompanha a instalação); Estável	Enfatiza legibilidade e flexibilidade; sistema de pacotes coerente para manipulação de dados, e interfaces como SQL e Spark.	Sintaxe concisa e eficiente – inclusive escalonando com a máquina; tem apenas o R-Base como dependência
Sintaxe com variações de formato; eficiência depende da otimização da implementação	Pacotes com muitas dependências; pacotes oficiais relacionados ainda em fase de experimentação	Longos encadeamentos de código que tendem a ser pouco legíveis

Vamos iniciar falando exclusivamente do Rbase!



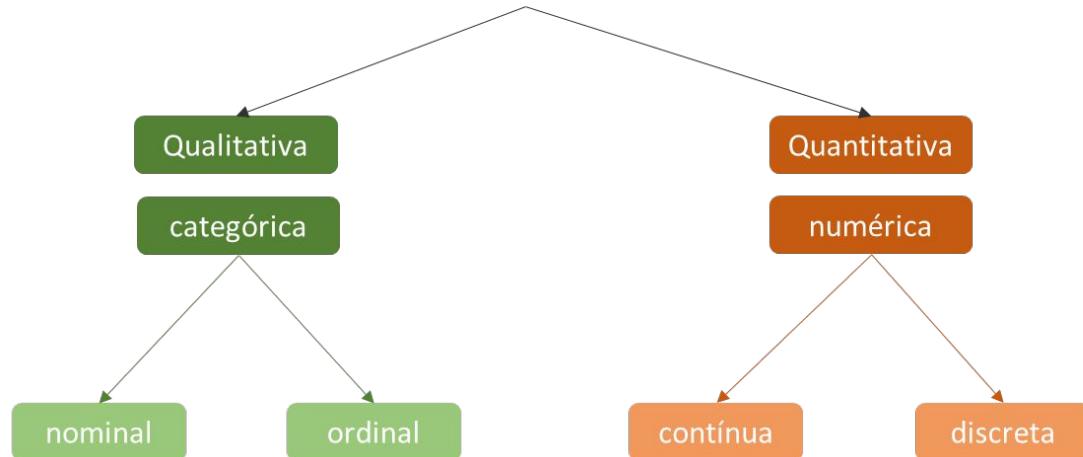
Entendendo os tipos, classes e estruturas de dados no R

pt. I

(conceitos chave)

Tipos de Dados: o conceito estatístico/matemático

Variável, Característica,
Atributos ou Features



Exemplos:

- | | | | |
|-------------|----------------|----------------------|---------------------|
| • Raça | • Escolaridade | • Salário | • Qtd de Filhos |
| • Gênero | • Medalhas | • Altura | • Qtd Clientes |
| • Profissão | • Ranking | • Consumo de energia | • Produtos vendidos |

Alguns
exemplos de
casos especiais:

lógico

TRUE/FALSE

data

Dia-mês-ano

hora

H - M - S

Classes de Dados no R: como o dado é processado

A tibble: 27 × 7

filme	classificacao_indicativa	nota_cinema_score	duracao	nota_rotten_tomatoes	data_lancamento	classificacao_livre
		<ord>	<dbl>	<int>	<date>	<lgl>
Toy Story - Um Mundo de Aventuras	Livre	A	81	100	1995-11-22	TRUE
Vida de Inseto	Livre	A	95	92	1998-11-25	TRUE
Toy Story 2	Livre	A+	92	100	1999-11-24	TRUE
Monstros S. A.	Livre	A+	92	96	2001-11-02	TRUE
Procurando Nemo	Livre	A+	100	99	2003-05-30	TRUE
Os Incríveis	Orientação parental sugerida	A+	115	97	2004-11-05	FALSE
Carros	Livre	A	117	74	2006-06-09	TRUE
Ratatouille	Livre	A	111	96	2007-06-29	TRUE
WALL-E	Livre	A	98	95	2008-06-27	TRUE
Up - Altas Aventuras	Orientação parental sugerida	A+	96	98	2009-05-29	FALSE
Toy Story 3	Livre	A	103	98	2010-06-18	TRUE
Carros 2	Livre	A-	106	40	2011-06-24	TRUE
1-12 of 27 rows	Previous	1	2	3	Next	

categórica

nominal

categórica

nominal

categórica

ordinal

numérica

contínua

numérica

discreta

categórica

data

categórica

lógico

character

<chr>

factor

<fct>

ordered

<ord>

double

<dbl>

integer

<int>

date

<date>

logical

<lgl>



Tipos de Dados no R: como o dado é armazenado

No R existem seis tipos de dados, indicando como os objetos são internamente armazenados pela ferramenta, são eles:

character: textos ou strings

logical: valores booleanos

double : valores decimais

integer: números inteiros

complex: números complexos

raw: bytes

É importante comentar que:

- Algumas funções só farão sentido para determinados tipos de dados
- O armazenamento de objetos no R segue uma hierarquia de coerção:
character > complex > double > integer > logical > raw
- Para consultar o tipo de um dado utilize a função: `typeof()`

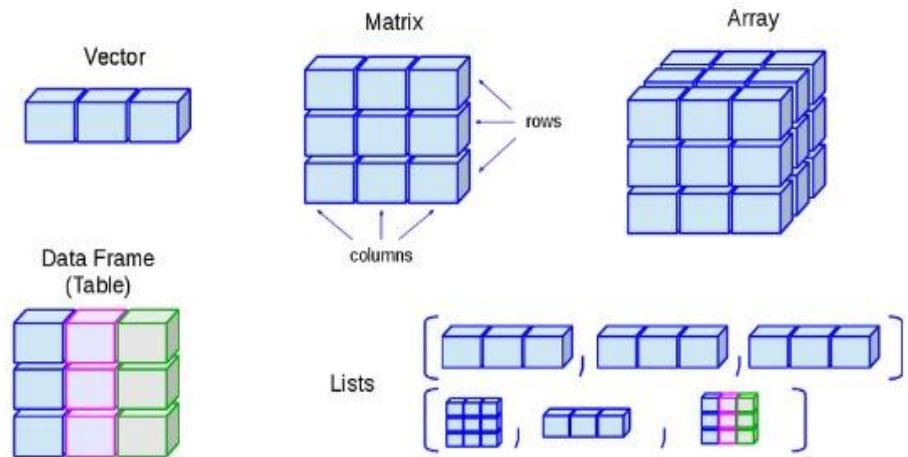


conceito da estatística:
Tipo de Dados
&
representação no R:
Classe de Dados

Estrutura de Dados no R: como organizar os dados

Estruturas de Dados é a maneira como o R permite que os dados sejam organizados, no caso temos:

- Vetores atômicos
- Matrizes
- Arrays
- Data frames (tibbles)
- Listas

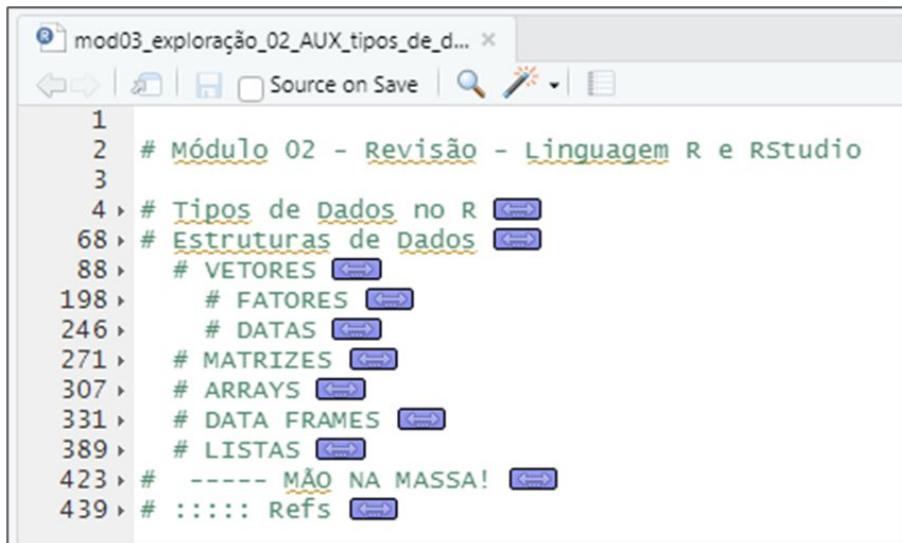


E para cada estrutura temos propriedades próprias considerando ações como: sua criação, consulta, e a maneira de aplicar operações.

Estrutura de Dados no R: como organizar os dados

Estruturas de Dados é a maneira como o R permite que os dados sejam organizados, no caso temos:

- Vetores atômicos
- Matrizes
- Arrays
- Data frames (tibbles)
- Listas



A screenshot of the RStudio interface showing a code editor window titled "mod03_exploração_02_AUX_tipos_de_d...". The editor displays a series of numbered code snippets, each preceded by a line number and a hash symbol (#). The snippets are organized into a hierarchical tree structure, indicated by small blue arrows pointing to the right next to each snippet. The snippets are as follows:

- 1 # Módulo 02 - Revisão - Linguagem R e RStudio
- 2
- 3
- 4 # Tipos de Dados no R
- 68 # Estruturas de Dados
- 88 # VETORES
- 198 # FATORES
- 246 # DATAS
- 271 # MATRIZES
- 307 # ARRAYS
- 331 # DATA FRAMES
- 389 # LISTAS
- 423 # ----- MÃO NA MASSA!
- 439 # :::::: Refs

E para cada estrutura temos propriedades próprias considerando ações como: sua criação, consulta, e a maneira de aplicar operações.

Como funciona o R?!

The image is a collage of three photographs of a woman's face, each featuring different mathematical content overlaid.

Top Left: A circle with radius r . Below it are the formulas $A = \pi r^2$ and $C = 2\pi r$.

Top Right: A cylinder with radius r and height h . Below it is the formula $V = \pi r^2 h$. To its left is a cone with radius r and height h , with a right-angle symbol at the base.

Bottom Left: Trigonometric values for angles 30° , 45° , and 60° :

	30°	45°	60°
\sin	$\frac{1}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{3}}{2}$
\cos	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{1}{2}$
\tan	$\frac{\sqrt{3}}{3}$	1	$\sqrt{3}$

Below these are two right-angled triangles. The first has angles 30° and 60° , with sides labeled $2x$, x , and $x\sqrt{3}$. The second has angles 45° and 45° , with sides labeled x and $\sqrt{2}x$.

Bottom Center: Mathematical integrals and their solutions:

- $\int \sin x dx = -\cos x + C$
- $\int \frac{dx}{\cos^2 x} = \operatorname{tg} x + C$
- $\int \operatorname{tg} x dx = -\ln|\cos x| + C$
- $\int \frac{dx}{\sin x} = \ln\left|\operatorname{tg}\frac{x}{2}\right| + C$
- $\int \frac{dx}{a^2 + x^2} = \frac{1}{a} \operatorname{arctg} \frac{x}{a}$
- $\int \frac{dx}{x^2 + 1} = \frac{1}{2} \ln\left|\frac{x-1}{x+1}\right| + C$

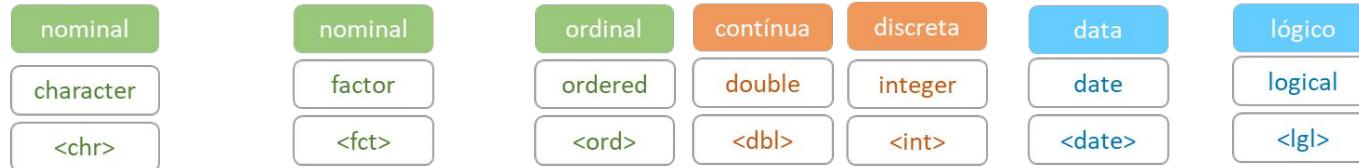
Bottom Right: A graph of the tangent function $\tan(\theta)$ versus θ/rad . The y-axis ranges from -5 to 10. The x-axis shows vertical asymptotes at $\theta/\text{rad} = \dots, -\pi/2, \pi/2, 3\pi/2, \dots$. The curve passes through $(0,0)$ and has vertical asymptotes at the points of intersection with the x-axis.

Below the graph are several quadratic equations:

- $ax^2 + bx + c = 0$
- $a(x^2 + \frac{b}{a}x + \frac{c}{a}) = 0$
- $x^2 + 2\frac{b}{2a}x + (\frac{b}{2a})^2 - (\frac{b}{2a})^2 + \frac{c}{a} = 0$
- $(x + \frac{b}{2a})^2 - \frac{b^2 - 4ac}{4a} = 0$

Resumo: Tipos, Classes e Estruturas de Dados no R

filme <chr>	classificacao_indicativa <fctr>	nota_cinema_score <ord>	duracao <dbl>	nota_rotten_tomatoes <int>	data_lancamento <date>	classificacao_livre <lgl>
Toy Story - Um Mundo de Aventuras	Livre	A	81	100	1995-11-22	TRUE
Vida de Inseto	Livre	A	95	92	1998-11-25	TRUE
Toy Story 2	Livre	A+	92	100	1999-11-24	TRUE
Monstros S. A.	Livre	A+	92	96	2001-11-02	TRUE
Procurando Nemo	Livre	A+	100	99	2003-05-30	TRUE
Os Incríveis	Orientação parental sugerida	A+	115	97	2004-11-05	FALSE
Carros	Livre	A	117	74	2006-06-09	TRUE
Ratatouille	Livre	A	111	96	2007-06-29	TRUE
WALL-E	Livre	A	98	95	2008-06-27	TRUE



Estruturas de Dados no R

Existem diferentes maneiras de organizar dados no R, sendo as três principais:

1) Vetor atômico é a estrutura mais básica, pois possui uma dimensão, e guarda objetos da mesma classe:

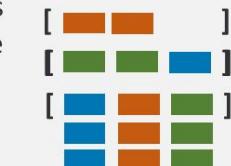


Seguindo a hierarquia de coerção:
character > complex > double > integer > logical > raw

2) Data Frames (ou **tibbles**), possuem duas dimensões, e para cada coluna podemos ter uma classe específica:



3) Listas: estrutura mais complexa, que permite elementos de diferentes dimensões e classes



No mais, existem: **matrix** e **array** (versões atômicas, assim como o vetor, porém com mais dimensões)

Entendendo os tipos, classes e estruturas de dados no R

pt. II

(detalhamento)

Lembrando: Tipos de dados na Estatística (na vida!)

NUMÉRICAS

dado quantitativo

Discreto

filhos, # eletrodomésticos

Contínuo

salário, altura

CATEGÓRICAS

dado qualitativo

Nominal

profissão, sexo

Ordinal

escolaridade, ranking

R – Tipos de dados no R

- No R existem seis tipos de dados, que basicamente indicam como os objetos são armazenados ():
 - **character**: textos ou strings
 - **double** : valores decimais
 - **integer**: números naturais
 - **logical**: valores booleanos
 - **complex**: números complexos
 - **raw**: bytes

```
#definindo um objeto
x <- 1
#o objeto é um inteiro?
is.integer(x)
#> [1] FALSE
#conferindo o tipo do objeto (double e numeric são sinônimos no R)
typeof(x) ; class(x)
#> [1] "double"
#> [1] "numeric"
#transformando 'x' em um texto, e redefinindo o objeto
x <- as.character(x)
#conferindo novamente o tipo do objeto
class(x)
#> [1] "character"
```

Para consultar o tipo de um dado: `typeof()`

R – Tipos de dados no R

- Os tipos de dados são importantes pois algumas funções só farão sentido para determinados tipos de dados
- Uma característica do armazenamento de objetos no R, é que eles respeitam a seguinte hierarquia de coerção:

character > complex > double > integer > logical > raw

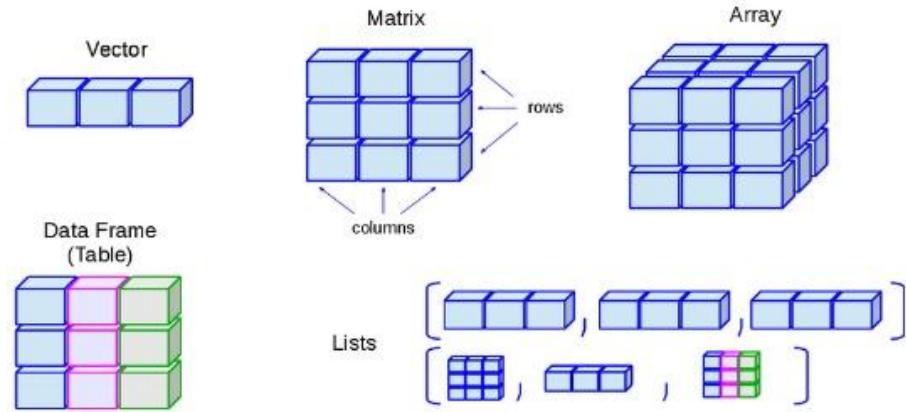
- Tendo que objetos do tipo `complex` e `raw` são pouco usuais, nos limitaremos a discutir os demais

```
#tentativa de cálculo com um objeto de texto
log(x)
#> Error in log(x): non-numeric argument to mathematical
function
```

R – Estruturas de dados

Adicionalmente ao armazenamento em memória, os dados do R podem ser organizados segundo as seguintes estruturas:

- **Vetores atômicos**
- **Matrizes**
- **Arrays**
- **Data frames**
- **Listas**

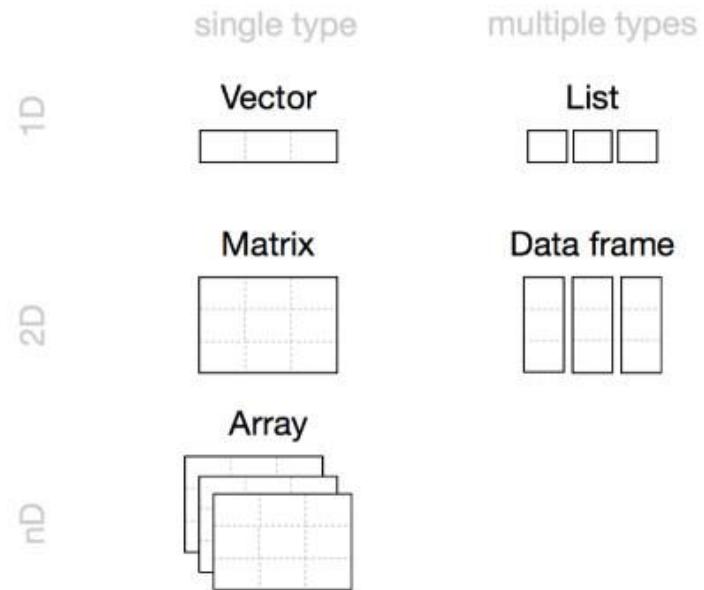


Vamos discutir estas estruturas considerando: sua criação, consulta, classes e operações

R – Estruturas de dados

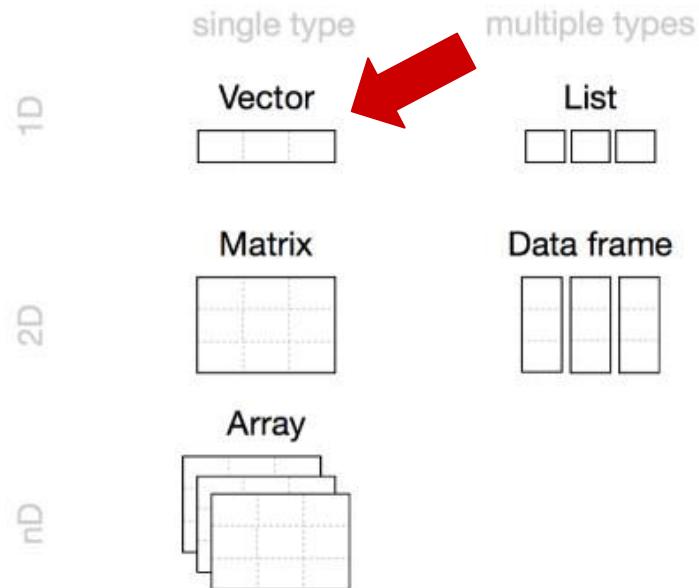
Adicionalmente ao armazenamento em memória, os dados do R podem ser organizados segundo as seguintes estruturas:

- **Vetores atômicos**
- **Matrizes**
- **Arrays**
- **Data frames**
- **Listas**



Vamos discutir estas estruturas considerando: sua criação, consulta, classes e operações

Manipulando as estruturas de dados no R



R – Vetores: como criar?

- Vetores atômicos são a estrutura mais básica do R, caracterizados por uma única dimensão e objetos atômicos, isto é, de um mesmo tipo
- A forma mais usual para **criar** um vetor no R é por meio da função `c()`, que combina os elementos nele listados. Porém existem muitas outras opções, como o operador `::`, que gera sequências, ou a função `rep()` que replica valores, além de combinações entre estas

```
#criando vetores com o comando `c()`
ex_vetor_character <- c("laranja", "roxo", "verde")
ex_vetor_character
#> [1] "laranja" "roxo"     "verde"

ex_vetor_numerical <- c(1,2,3,4,5)
ex_vetor_numerical
#> [1] 1 2 3 4 5

ex_vetor_integer <- c(1L,2L,3L,4L,5L)
ex_vetor_integer
#> [1] 1 2 3 4 5

ex_vetor_logical <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
ex_vetor_logical
#> [1] TRUE FALSE  TRUE  TRUE FALSE
```

```
# outras opções para a criação de vetores
ex_vetor_character2 <- rep("azul", times = 5)
ex_vetor_numerical2 <- 1:5
ex_vetor_integer2 <- c(1L:3L,4L,5L)
ex_vetor_logical2 <- c(TRUE, FALSE, c(FALSE,TRUE))
```

R – Vetores: como consultar valores pela posição?

- Para **consultas** basta indicar o índice do elemento de interesse entre colchetes – notando que o índice de contagem do R se inicia no 1, e não no 0, como costuma ser em outras linguagens
- Para consultar múltiplos elementos, podemos indicar os índices de interesse por meio de novos vetores, ou ainda explicitar apenas os os elementos que não temos interesse precedidos por `-

```
ex_acesso <- c("primeiro", "segundo", "terceiro")
ex_acesso
#> [1] "primeiro" "segundo" "terceiro"

#acessando o primeiro elemento
ex_acesso[1]
#> [1] "primeiro"

#acessando o último elemento
n <- 3; ex_acesso[n]
#> [1] "terceiro"
```

```
#excluindo o primeiro elemento
ex_acesso[-1]
#> [1] "segundo" "terceiro"
#excluindo os três últimos elementos
n <- 5; ex_acesso[-((n-2):n)]
#> [1] "primeiro" "segundo"
```

R – Vetores: como saber a classe?

- Um vetor possui como classe o tipo dos objetos que guarda. Sendo importante ter em mente a hierarquia de coerção comentada na seção anterior

```
ex_vetor_combinado <- c(ex_vetor_character, ex_vetor_integer)
class(ex_vetor_combinado)
#> [1] "character"
```

- Tendo em conta que algumas funções fazem a coerção de forma automática

```
#coerção da classe lógica para inteira
class(length(ex_vetor_logical)) #`length()` retorna o tamanho do objeto
#> [1] "integer"
#coerção da classe inteira para numérica
class(cos(ex_vetor_integer)) #`cos()` retorna o cosseno de um número
#> [1] "numeric"
#coerção da classe numérica para a de texto
class(paste(ex_vetor_numerical)) #`paste()` concatena objetos
#> [1] "character"
```

R – Vetores: como aplicar operações?

- Para operações entre vetores, o R alinha os objetos, e faz o cálculo elemento a elemento
- No caso dos vetores possuírem tamanhos diferentes, o R irá utilizar o esquema de reciclagem, ou seja, o vetor menor será repetido até completar o vetor maior

```
#operação entre vetores do mesmo tamanho  
1:5 + 1:5  
#> [1] 2 4 6 8 10  
paste(1:5,"_",1:5)  
#> [1] "1 _ 1" "2 _ 2" "3 _ 3" "4 _ 4" "5 _ 5"
```

```
#operações entre vetores de tamanhos diferentes  
1:3 + 1  
#> [1] 2 3 4  
  
#operações entre vetores de tamanhos diferentes, múltiplos  
1:3 != 1:6  
#> [1] FALSE FALSE FALSE TRUE TRUE TRUE  
  
#no caso de objetos não múltiplos o R retorna o resultado + um aviso  
1:3 == c(1:3,4)  
#> Warning in 1:3 == c(1:3, 4): comprimento do objeto maior não é múltiplo do  
#> comprimento do objeto menor  
#> [1] TRUE TRUE TRUE FALSE
```

R – Vetores: como aplicar operações?

- Para operar com vetores, os vetores devem ter o mesmo comprimento. Caso contrário, é necessário transformar os vetores em objetos, e depois aplicar as operações.
- No caso de vetores que possuírem comprimentos diferentes, é necessário utilizar o comando `rep` para reciclar o menor vetor, ou repetir a operação com o maior vetor.



Fim da 2^a
aula!

R – Vetores: como adicionar atributos?

O R permite que adicionemos atributos aos objetos, o que significa associar propriedades como: nomes, dimensões, classes, comentários, etc. Tais atributos não alteram o armazenamento do objeto, mas modificam o comportamento do objeto em alguns contextos.

```
#verificando classe e tamanho, atributos básicos
class(ex_vetor_integer); length(ex_vetor_integer)
#> [1] "integer"
#> [1] 5

#adicionando o atributo nome
names(ex_vetor_integer) <- c("um", "dois", "três", "quatro", "cinco")

#permitindo o acesso diretamente pelos nomes definidos
ex_vetor_integer["três"]
#> três
#>     3
```

No caso dos vetores, `length` e `class` são atributos default, enquanto `names` é opcional extremamente importante, visto que permite a construção de alguns casos especiais.

R – Vetores especiais: Fatores

Fatores tratam-se de vetores atômicos que possuem um número limitado de categorias, o que nos permite trabalhar com variáveis categóricas, que são tratadas de forma diferenciada para algumas análises (como em modelos de regressão p.e.).

```
ex_fator <- factor(LETTERS[1:5])
ex_fator
#> [1] A B C D E
#> Levels: A B C D E
#os fatores são armazenados de acordo com os tipos de objetos discutidos.
typeof(ex_fator)
#> [1] "integer"
#porém possuem uma classe própria
class(ex_fator)
#> [1] "factor"
#que permite o uso de funções específicas
levels(ex_fator)
#> [1] "A" "B" "C" "D" "E"
```

Tais objetos podem ser armazenados como strings ou inteiros, e possuem nomes associados, chamados categorias ou níveis (levels)

R – Vetores especiais: Fatores (variáveis nominais)

Os fatores permitem que seus níveis sejam ordenados, de modo que podemos diferenciar as variáveis categóricas nominais e ordinais, em que:

- **variáveis nominais:** variáveis que possuem categorias sem qualquer tipo de ordenação entre elas e, portanto, não podem ser submetidas a operações aritméticas. Exemplos: cores e gênero

```
var_nominal <- c(rep(0,2), rep(1, 3)); var_nominal
#> [1] 0 0 1 1 1
var_factor <- factor(var_nominal); var_factor
#> [1] 0 0 1 1 1
#> Levels: 0 1

#podemos checar os niveis de fator
levels(var_factor)
#> [1] "0" "1"

#e altera-los
levels(var_factor)[1] <- "masculino"; levels(var_factor)[2] <- "feminino"
```

R – Vetores especiais: Fatores (variáveis ordinais)

- **variáveis ordinais**: possuem uma ordenação entre as suas categorias, apesar de não haver uma escala bem definida entre as categorias. De modo que podemos fazer alguns cálculos, como a obtenção do nível máximo. Exemplos: classe social, escolaridade ou pesquisas de opinião do tipo concordo/neutro/discordo.

```
var_ordinal <- factor(c(rep("baixo",2), rep("alto", 3))); var_ordinal
#> [1] baixo baixo alto  alto  alto 
#> Levels: alto baixo

var_ordinal <- ordered(var_ordinal); var_ordinal
#> [1] baixo baixo alto  alto  alto 
#> Levels: alto < baixo

#podemos fazer a alteração redefinindo os níveis
levels(var_ordinal) <- c("baixo", "alto"); var_ordinal
#> [1] alto  alto  baixo baixo baixo
#> Levels: baixo < alto
```

Visto ser um caso particular dos vetores a forma de acesso é a mesma. No mais podemos utilizar funções como: `is.factor`/`is.ordered` e `as.factor`/`as.ordered`.

R – Vetores especiais: Datas

O R possui uma classe própria para objetos que armazenam datas. De modo que para transformar um elemento para tal classe podemos utilizar a função `as.Date()`:

```
ex_date <- as.Date(c("1988-03-25", "2019-03-25"))
typeof(ex_date)
#> [1] "double"
class(ex_date)
#> [1] "Date"
```

Desta forma podemos fazer cálculos cabíveis à dados desta natureza, como, por exemplo, a quantidade de dias entre duas datas:

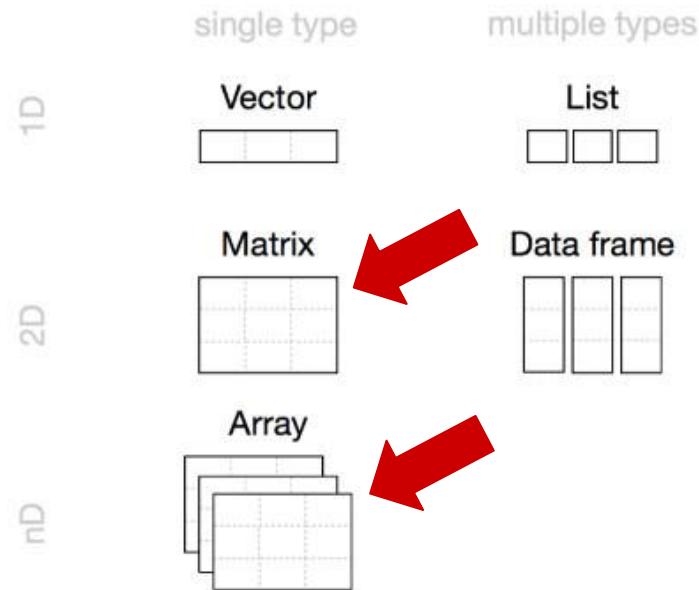
```
ex_date[1] - ex_date[2]
#> Time difference of -11322 days
```

R – Vetores especiais: Datas

Abaixo estão algumas alternativas de formato - para mais opções consulte `?strptime`:

- %d: dias numéricos (0-31)
- %a: dias da semana, abreviados (Mon)
- %A: dias da semana, não abreviados (Monday)
- %m: meses numéricos (00-12)
- %b: meses em texto, abreviados (Jan)
- %B: meses em texto, não abreviados (January)
- %y: anos com dois dígitos (19)
- %Y: anos com quatro dígitos (2019)

Manipulando as estruturas de dados no R



R – Matrizes (criação)

- Matrizes são estruturas que nos permitem trabalhar com dados bidimensionais que contenham o mesmo tamanho e o mesmo tipo de dados.

```
x <- 1:8

#criando uma matriz a partir da atribuição de dimensões
dim(x) <- c(2,4); x
#>      [,1] [,2] [,3] [,4]
#> [1,]    1    3    5    7
#> [2,]    2    4    6    8

#definindo uma matriz a partir do número de linhas
matrix(x, nrow = 2)
#>      [,1] [,2] [,3] [,4]
#> [1,]    1    3    5    7
#> [2,]    2    4    6    8
matrix(x, nrow = 2, byrow = TRUE)
#>      [,1] [,2] [,3] [,4]
#> [1,]    1    2    3    4
#> [2,]    5    6    7    8

#ou por linhas e colunas
ex_matriz <- matrix(x, 2, 4); ex_matriz
#>      [,1] [,2] [,3] [,4]
#> [1,]    1    3    5    7
#> [2,]    2    4    6    8
```

R – Arrays (criação)

- Arrays são similares às matrizes, porém permitindo mais que duas dimensões.
Para criar uma matriz, podemos ou atribuir o atributo dimensão a um vetor, por meio da função dim(), ou reorganizar um vetor por meio de funções específicas

```
ex_array <- array(x, dim = c(2, 2, 2)); ex_array
#> , , 1
#>
#>      [,1] [,2]
#> [1,]    1    3
#> [2,]    2    4
#>
#> , , 2
#>
#>      [,1] [,2]
#> [1,]    5    7
#> [2,]    6    8
```

R – Matrizes e Arrays (consultas)

- A consulta de valores se dá de modo similar aos vetores, com a diferença de precisarmos identificar qual a dimensão que estamos interessados, utilizando vírgulas para separar as dimensões, especificadas em ordem crescente
- Estas estruturas possuem classes próprias

```
#matrix
ex_matrix[1,1]
#> [1] 1
#array - consulta do valor contido no cruzamento do primeiro elemento das três dimensões
ex_array[1,1,1]
#> [1] 1
#array - consulta do valor contido no primeiro elemento da terceira dimensão
ex_array[ , ,1]
#>      [,1] [,2]
#> [1,]    1    3
#> [2,]    2    4

#podemos também consultar informações como a dimensão do objeto
dim(ex_array)
#> [1] 2 2 2
```

```
class(ex_matrix)
#> [1] "matrix"
class(ex_array)
#> [1] "array"
```

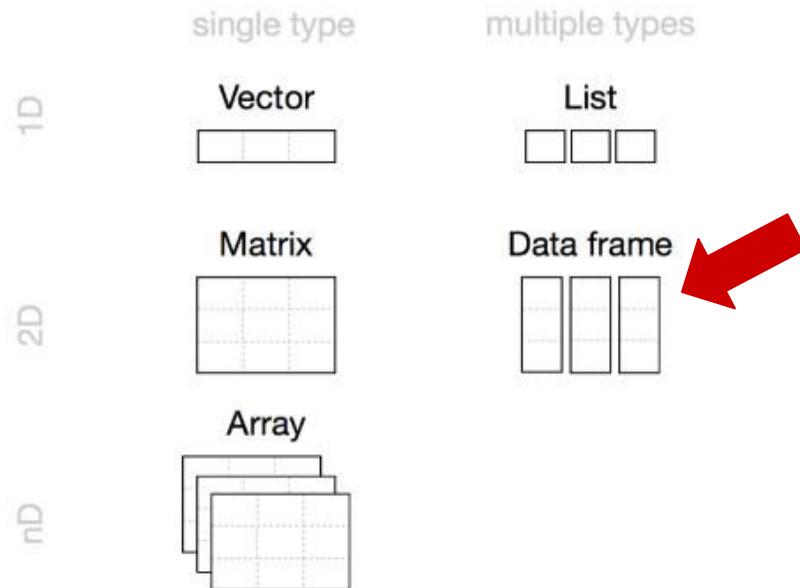
R – Matrizes e Arrays (operações)

- Em relação às operações, as matrizes e arrays funcionam similarmente aos vetores, além de permitirem cálculos algébricos

```
#operações entre objetos
rep(2,4) * ex_array
#> , , 1
#>
#>      [,1] [,2]
#> [1,]    2     6
#> [2,]    4     8
#>
#> , , 2
#>
#>      [,1] [,2]
#> [1,]   10    14
#> [2,]   12    16

#multiplicação matricial (considerando a matriz transposta)
ex_matriz %*% t(ex_matriz)
#>      [,1] [,2]
#> [1,]   84   100
#> [2,]  100   120
```

Manipulando as estruturas de dados no R



R – Data frames (criação)

- Data Frames tratam-se de estruturas bidimensionais mais genéricas que as matrizes, uma vez que as colunas podem conter tipos de objetos diferentes. Tal estrutura, similar a uma tabela SQL ou uma planilha do Excel, é o formato mais usual no R

```
#é possível criar data frames a partir de objetos existentes
idade <- c(31,30,25,40)
classe <- factor(c("AB", "C", "C", "B"), ordered = T)
flag <- c(TRUE,TRUE,NA,FALSE)
ex_df <- data.frame(idade,classe,flag); ex_df
#>   idade classe flag
#> 1    31     AB  TRUE
#> 2    30      C  TRUE
#> 3    25      C   NA
#> 4    40      B FALSE

#ou diretamente, especificando, ou não, o nome das colunas
data.frame(nome_da_coluna1 = 1:3, 4:6)
#>   nome_da_coluna1 X4.6
#> 1                   1     4
#> 2                   2     5
#> 3                   3     6
```

R – Data frames (consultas)

- Além das consultas similares às que vimos para as demais estruturas, um data.frame permite algumas outras opções:

```
#considerando o nome das colunas
ex_df[c("idade","classe")]
#>   idade classe
#> 1    31     AB
#> 2    30      C
#> 3    25      C
#> 4    40      B
ex_df$idade
#> [1] 31 30 25 40
#as linhas de interesse
ex_df[ex_df$idade>35,]
#>   idade classe flag
#> 4    40      B FALSE
ex_df[which(flag==TRUE),]
#>   idade classe flag
#> 1    31     AB TRUE
#> 2    30      C TRUE
# ou ambos
ex_df[1:2,"flag"]
#> [1] TRUE TRUE
```

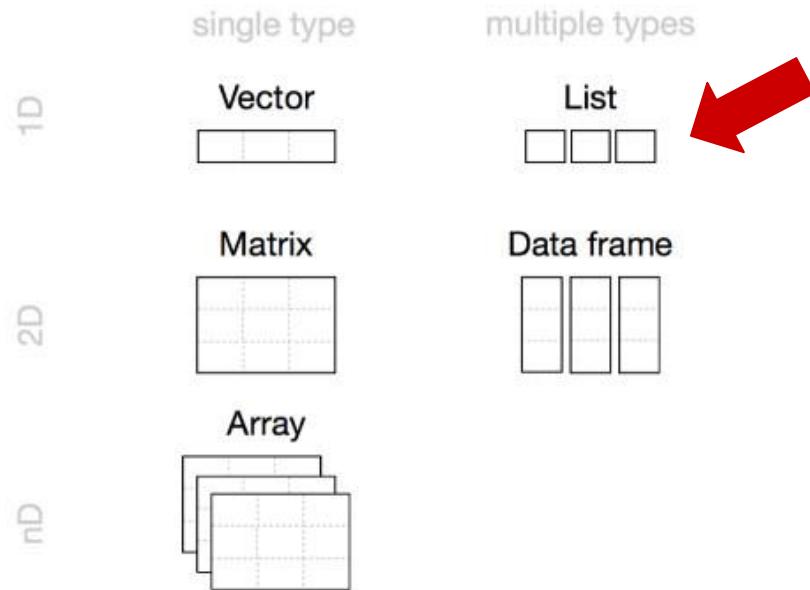
R – Data frames (classes e operações)

- Os Data Frames possuem uma classe própria, podendo ser consultada via `is.data.frame()` ou aderida com `as.data.frame()`. Porém, em ambos os casos, os elementos do data.frame permanecem com as suas próprias classes
- Em relação às operações, temos todas as opções já discutidas, tendo como restrição apenas se o tipo de objeto é cabível:

```
class(ex_df)
#> [1] "data.frame"
class(ex_df$idade)
#> [1] "numeric"
```

```
ex_df$idade + 1
#> [1] 32 31 26 41
ex_df$classe == "C"
#> [1] FALSE TRUE TRUE FALSE
```

Manipulando as estruturas de dados no R



R – Listas (criação)

- Listas são estruturas também de uma dimensão, similarmente aos vetores, porém permitindo que diferentes tipos de objetos sejam guardados em um único vetor
- Listas são também conhecidas como vetores recursivos, isto porque podem conter outras listas, de diferentes tamanhos. Tal característica permite que sua dimensão seja ampliada, e esta possa conter inclusive data frames
- Por tais características listas são as estruturas mais genéricas do R

```
list(4L, "a")
#> [[1]]
#> [1] 4
#>
#> [[2]]
#> [1] "a"
```

```
ex_list <- list(9:1, ex_df = ex_df, TRUE)
ex_list
#> [[1]]
#> [1] 9 8 7 6 5 4 3 2 1
#>
#> $ex_df
#>   idade classe flag
#> 1    31     AB  TRUE
#> 2    30      C  TRUE
#> 3    25      C    NA
#> 4    40      B FALSE
#>
#> [[3]]
#> [1] TRUE
```

R – Listas (consultas)

- Para consultas podemos seguir o padrão dos vetores usando `[,` tendo sempre listas como retorno. Ou por meio de `[[` ou ` `\$` caso os elementos sejam nomeados - tendo como retorno a classe do elemento consultado
- Podemos também acessar os elementos de cada um dos sub-objetos, seguindo o padrão de consultas das suas estruturas originais

```
ex_list[1] ; class(ex_list[1])
#> [[1]]
#> [1] 9 8 7 6 5 4 3 2 1
#> [1] "list"
ex_list[[1]] ; class(ex_list[[1]])
#> [1] 9 8 7 6 5 4 3 2 1
#> [1] "integer"
ex_list$ex_df ; class(ex_list$ex_df)
#>   idade classe  flag
#> 1     31     AB TRUE
#> 2     30      C TRUE
#> 3     25      C  NA
#> 4     40      B FALSE
#> [1] "data.frame"
```

```
ex_list[[1]][1]
#> [1] 9
ex_list$ex_df[1,1]
#> [1] 31
```

R – Listas (classes e operações)

- Listas possuem uma classe própria e, assim como o data frame, permite que seus elementos mantenham as suas próprias classes
- Listas seguem a mesma lógica dos data frames em relação às operações, porém, para termos maior flexibilidade, precisaremos trabalhar com estruturas de controle no R

```
class(ex_list)
#> [1] "list"
class(ex_list[[1]])
#> [1] "integer"
class(ex_list$ex_df)
#> [1] "data.frame"
class(ex_list[[3]])
#> [1] "logical"
```

```
#operação entre elementos da lista
paste(ex_list[[1]][1:4], ex_list[[3]])
#> [1] "9 TRUE" "8 TRUE" "7 TRUE" "6 TRUE"

#operação com uma coluna específica do data frame
ex_list$ex_df$idade + ex_list[[1]]
#> Warning in ex_list$ex_df$idade + ex_list[[1]]:
comprimento do objeto maior
#> não é múltiplo do comprimento do objeto menor
#> [1] 40 38 32 46 36 34 28 42 32
```

Importação / Exportação de arquivos

R – Importação/Exportação

- Visando facilitar a dinâmica, vamos primeiro exportar a base de dados criada no último capítulo `dados_mtcars`, e então vamos importar esta mesma base de dados novamente. Para exportar os dados precisamos especificar o nome e a extensão que o arquivo será salvo, por exemplo:

```
write.table(dados_mtcars, file = "dados_mtcars.txt")
```

- Uma pergunta natural seria: onde o arquivo foi salvo? É a resposta é: no diretório de trabalho. Para saber qual o diretório de trabalho atual, utilizamos o comando `getwd()` (get working directory), o que nos retornará algo como "C:/Users/Documents". Enquanto que para alterar o diretório de trabalho temos a função `setwd()` (set working directory), por meio de comandos como `setwd("C:/Users/Desktop")`.

R – Importação/Exportação



R – Importação/Exportação

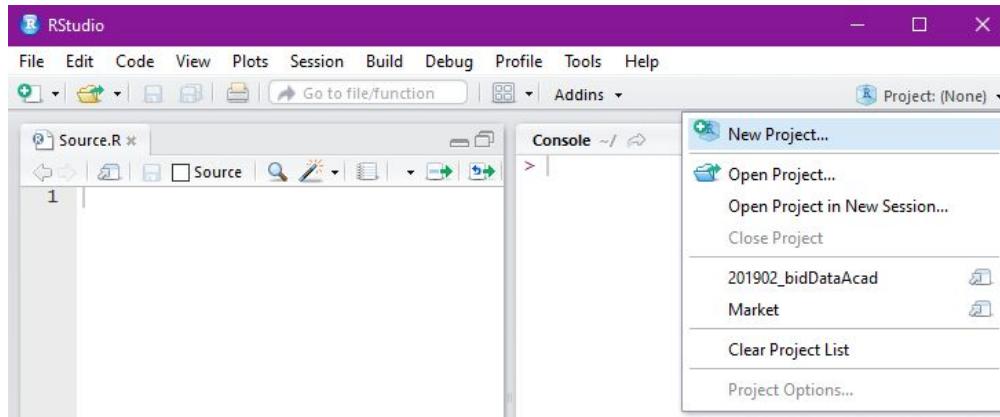
- Alternativamente podemos especificar o destino do arquivo de interesse, de modo independente do diretório de trabalho, precedendo o nome do arquivo com o diretório desejado, considerando o caso assim p.e.: write.table(dados_mtcars, file = "C:/Users/dados_mtcars.txt") - note que trabalhamos com barras simples (/) para especificação de diretórios. Para a leitura de dados seguimos o mesmo padrão, assim:

```
dados_lidos <- read.table("dados_mtcars.txt")
str(dados_lidos)
#> 'data.frame': 32 obs. of 11 variables:
#> $ mpg : num 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
#> $ cyl : int 6 6 4 6 8 6 8 4 4 6 ...
#> $ disp: num 160 160 108 258 360 ...
#> $ hp : int 110 110 93 110 175 105 245 62 95 123 ...
#> $ drat: num 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.
#> $ wt : num 2.62 2.88 2.32 3.21 3.44 ...
#> $ qsec: num 16.5 17 18.6 19.4 17 20.2 15.8 20 22.9
#> $ vs : Factor w/ 2 levels "S","V": 2 2 1 1 2 1 2 1
#> $ am : int 1 1 1 0 0 0 0 0 0 ...
#> $ gear: int 4 4 4 3 3 3 3 4 4 4 ...
#> $ carb: int 4 4 1 1 2 1 4 2 2 4 ...
```

Fim da 3^a
aula!

R – Importação/Exportação + Diretórios e Projetos

- Existe ainda uma terceira alternativa: trabalhar com a opção de projetos do RStudio (.RProj). Nesta, o diretório em que o arquivo .RProj estiver salvo, passa a ser a pasta raiz de todas as análises. Assim todas as especificações dos arquivos de input/output, scripts, e etc, podem ser setados sem a necessidade de detalhar a localização exata do diretório. Tal prática, bem como a adoção de estruturas para a organização dos arquivos, facilitam a evolução e o compartilhamento do trabalho.



Operadores Lógicos

Passou para
a aula 6

R – Operadores Lógico

Os operadores lógicos são usados para realizar operações lógicas. No R, os valores lógicos podem ser TRUE (T) ou FALSE (F):

```
#igualdade: se os elementos à esquerda, par a par,  
estão contidos à direita  
x == y  
  
#maior que  
x > y  
  
#maior ou igual que  
x >= y  
  
#menor que  
x < y  
  
#menor ou igual que  
x <= y  
  
#diferente: se os elementos à esquerda, par a par,  
não estão contidos à direita  
x != y
```

Alguns exemplos:

```
x <- 1  
y <- 2  
  
# x é igual a y?  
x == y  
#> [1] FALSE  
  
# x é diferente de y?  
x != y  
#> [1] TRUE  
  
# x é maior que y?  
x > y  
#> [1] FALSE  
  
# x é menor que y?  
x < y  
#> [1] TRUE  
  
# x é maior ou igual a y?  
x >= y  
#> [1] FALSE  
  
# x é menor ou igual a y?  
x <= y  
#> [1] TRUE
```

R – Operadores Lógico

Os operadores lógicos são usados para realizar operações lógicas. No R, os valores lógicos podem ser TRUE (T) ou FALSE (F):

- **comparações:** `%in%`

#contem: se os elementos à esquerda, um a um, estão contidos à direita
`x %in% y`

Algumas aplicações:

```
x <- c(1, 2, 3, 4)

# 1 está contido em x?
1 %in% x
#> [1] TRUE

# 5 está contido em x?
5 %in% x
#> [1] FALSE
```

Já se invertermos a comparação, passamos a fazer uso da característica de reciclagem do R

```
# x está contido em 1?
x %in% 1
#> [1] TRUE FALSE FALSE FALSE

# x está contido em 5?
x %in% 5
#> [1] FALSE FALSE FALSE FALSE
```

R – Operadores Lógico

Lembrando: o comando `%in%` é recomendado para trabalhar com vetores, pois a comparação é feita avaliando cada um dos elementos à esquerda. De modo que todos os elementos da esquerda são comparados, individualmente, com a lista de opções à direita. Enquanto que o simbolo `==` trabalha com a reciclagem, e faz a avaliação em pares, 1º elemento da esquerda com o 1º da direita, e assim por diante.

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

```
#1 e 3 está contido em x  
c(1,3) %in% x  
#> [1] TRUE TRUE
```

```
#os elementos de x estão contidos em 1 e 3  
x %in% c(1,3)  
#> [1] TRUE FALSE TRUE TRUE
```

```
# x está pareado com 1 e 3?  
x == c(1,3)  
#> [1] TRUE FALSE FALSE TRUE
```

```
# 1 e 3 está pareado com x?  
c(1,3) == x  
#> [1] TRUE FALSE FALSE TRUE
```

R – Operadores Lógico

Os operadores lógicos são usados para realizar operações lógicas. No R, os valores lógicos podem ser TRUE (T) ou FALSE (F):

- **negação:** !

```
#negação: inverte o valor Lógico de um objeto  
!x
```

Exemplos considerando comparações de elementos:

```
x <- TRUE
```

```
# negação de x  
!x  
#> [1] FALSE
```

Ou comparações de mais de um elemento:

```
x <- 1  
y <- 2
```

```
# x é menor que y?  
x < y  
#> [1] TRUE
```

```
# negação de x < y  
!(x < y)  
#> [1] FALSE
```

```
x <- c(1, 2, 3, 4)
```

```
# 1 está contido em x?  
1 %in% x  
#> [1] TRUE  
!(1 %in% x)  
#> [1] FALSE
```

R – Operadores Lógico

Os operadores lógicos são usados para realizar operações lógicas. No R, os valores lógicos podem ser TRUE (T) ou FALSE (F):

- **operadores combinados:** | e &

```
#casos mais comuns
#ou: retornando T/F para cada elemento das comparações
x | y
#e: retornando T/F para cada elemento das comparações
x & y

#casos mais específicos
#ou: retornando apenas um T/F (1a comparação à esquerda do vetor)
x || y
#e: retornando apenas um T/F (1a comparação à esquerda do vetor)
x && y
```

Alguns exemplos:

```
x <- c(1, 2, 3)
y <- c(4, 5, 6)

x >= 3
#> [1] FALSE FALSE  TRUE
x <= 5
#> [1] TRUE TRUE TRUE

x >= 3 | x <= 5
#> [1] TRUE TRUE TRUE
x >= 3 & x <= 5
#> [1] FALSE FALSE  TRUE
```

```
x <- 1
y <- 1:3

# x é igual a y ou x é maior que y?
x | y
#> [1] TRUE TRUE TRUE

# x é igual a y ou x é menor que y?
x || y
#> [1] TRUE
```

R – Operadores Lógico

Os operadores lógicos são usados para realizar operações lógicas. No R, os valores lógicos podem ser TRUE (T) ou FALSE (F):

- **pcionais:** `any()`, `all()` e `identical()`

No caso de vetores lógicos, podemos também trabalhar com as funções `any()` e `all()`, ou ainda a função `identical()`. No caso desta última, trata-se de uma função que precisa ser analisada com cuidado, visto trabalhar com diferenças sutis, retornando `FALSE` para comparações como por exemplo:

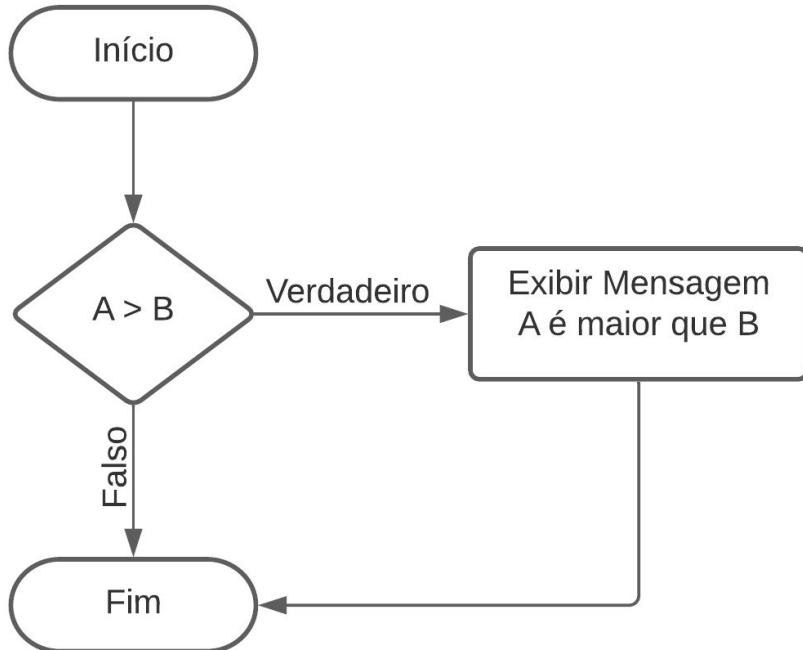
```
identical(1,1L) .
```

Estruturas de Controle

Passou para
a aula 6

R – Estruturas de controle

Estruturas de controle são blocos de programação que, baseado em parâmetros pré-definidos, definem a direção a ser seguida. No R temos todas as funções usualmente existentes em outras linguagens. Vamos repassar alguns dos fluxos condicionais e de repetição mais usuais, no caso:



R – Estruturas de controle

As principais estruturas considerando fluxos condicionais e de repetição são:

if - se a declaração testada for verdade, ou seja, retornar TRUE, então os comandos especificados dentro das chaves {} serão executados.

If-else - similar ao comando if() porém seguido de um segundo bloco, que será avaliado se, e somente se, o resultado do primeiro bloco for FALSE.

ifelse - Existe também o ifelse(), uma versão mais simples, em que especificamos ambas as ações como parametro de uma função.

```
x <- 1
if(x != 0) {
  print(x+1)
}
#> [1] 2
```

```
x <- 30
if (x<=10) {
  print("x é menor ou igual a 10")
} else if (x>10 & x<20) {
  print("x está entre 10 e 20")
} else{
  print("x é maior ou igual a 20")
}
#> [1] "x é maior ou igual a 20"
```

```
x <- "olar"
ifelse(x == "olar", "flor.do.campo", "xovens")
#> [1] "flor.do.campo"
```

R – Estruturas de controle

As principais estruturas considerando fluxos condicionais e de repetição são:

for() - uma sequência de instruções que são repetidas com cada um dos elementos especificados:

Podemos trabalhar sem as chaves caso as instruções sejam dadas em até uma linha após o comando for:

nested loops - De modo similar ao anterior temos o caso dos loops aninhados

```
#exemplo
x <- 1:3
for(i in x){
  print(i)
}
#> [1] 1
#> [1] 2
#> [1] 3
```

```
#exemplo ~ podemos utilizar a função `seq` para acessar os indices de interesse
x <- 1:2; y <- c(1,10)
for(i in seq(x)){
  for(j in seq(y)) print(x[i]*y[j])
}
#> [1] 1
#> [1] 10
#> [1] 2
#> [1] 20
```

R – Estruturas de controle

As principais estruturas considerando fluxos condicionais e de repetição são:

- **while()** - repetição de um bloco de comandos até que certa condição não seja mais satisfeita:

além dos comandos citados, funções como `break()`, que pode ser utilizado para interromper um loop e dar continuidade ao fluxo do programa, ou `next()`, que descontinua uma interação particular, e pula para a próxima, também existem no R. Além do `repeat()`, um loop que executa um bloco de comandos repetidamente, até que o mesmo seja quebrado.

```
#exemplo
i <- 1
while (i<=6){
  print(i*i)
  i = i+1
}
#> [1] 1
#> [1] 4
#> [1] 9
#> [1] 16
#> [1] 25
#> [1] 36
```

Criando Funções

Passou para
a aula 6

R – Criando funções

- No R podemos criar as nossas próprias funções, por meio do `function()`, prática que evita retrabalho, e permite um código mais simples e mais facilmente reproduzível.
- Para criar funções existem três características:
 - o nome: como a função será armazenada no ambiente R, em qual objeto podemos acessá-la
 - o argumentos: os parâmetros que podem ser utilizados internamente. É possível ter parâmetros opcionais, default ou mesmo não possuir argumentos.
 - o corpo da função; o código que será executado (a depender dos argumentos). Aqui definimos o que a função faz, e o que retorna.

```
# para criar uma função
nome_da_funcao <- function(argumentos){
  corpo da função
}
# para usar essa função
nome_da_funcao(argumentos = ...)
```

```
#modelo
nome_funcao <- function(arg_1, arg_2, ...){
  corpo da função
}
```

R – Criando funções

- Em relação ao retorno da função, o R, por default, retorna a última linha do corpo da função. Porém, alternativamente, podemos utilizar a função `return()`, independente da última linha

```
temp_c <- 25

celcius_fahrenheit <- function(){
  temp_f <- (temp_c * 9/5) + 32
  temp_f #ou, equivalentemente, return(temp_f)
}

temp_f
#> Error in eval(expr, envir, enclos): objeto 'temp_f' não encontrado
celcius_fahrenheit()
#> [1] 77
temp_f
#> Error in eval(expr, envir, enclos): objeto 'temp_f' não encontrado
```

- Funções possuem o seu próprio environment, de modo que os objetos/cálculos realizados dentro da função não alteram os demais ambientes. Porém o environment da função consegue consultar o ambiente em que a função foi definida, bem como o ambiente pai deste, e assim por diante, de forma hierárquica.

R – Criando funções

- O exemplo acima foi dado visando ilustrar como o R lida com os diferentes ambientes, porém, em termos práticos é extremamente desaconselhável deixar que a função dependa de parâmetros externos ao seu próprio environment. De modo que todas as dependências devem ser especificadas e passadas como argumentos da função, assim:

```
celcius_fahrenheit <- function(temp_c) {  
  temp_f <- (temp_c * 9/5) + 32  
  temp_f  
}  
  
temp_c <- 25  
celcius_fahrenheit()  
#> Error in celcius_fahrenheit(): argumento "temp_c" ausente, sem padrão  
celcius_fahrenheit(temp_c)  
#> [1] 77  
  
#ou o que faremos mais recorrentemente  
temp_f <- celcius_fahrenheit(temp_c)
```

R – Criando funções

- Em relação aos argumentos, assim como ocorre nas funções built-in, podemos especificar valores default para os parâmetros, bem como definir o corpo da função em uma mesma linha sem o uso de chaves, como descrito para o for():
- Adicionalmente podemos utilizar recursos como break() ou cat() (uma versão um pouco menos genérica da função print()), permitindo, por exemplo:

```
ex_function <- function(x, y=1) x+y  
ex_function(1,0)  
#> [1] 1  
ex_function(1)  
#> [1] 2
```

```
celcius_fahrenheit <- function(temp_c){  
  temp_f <- (temp_c * 9/5) + 32  
  
  if(temp_c>10){  
    cat("\n Por que 10? Porque sim! \n")  
    stop  
  }  
  
  temp_f  
}  
  
celcius_fahrenheit(25)  
#>  
#> Por que 10? Porque sim!  
#> [1] 77
```

R na
Ciência de Dados,
Estatística,
Análise de Dados,
Data Mining,
Business Inteligen

...

Início aula 4

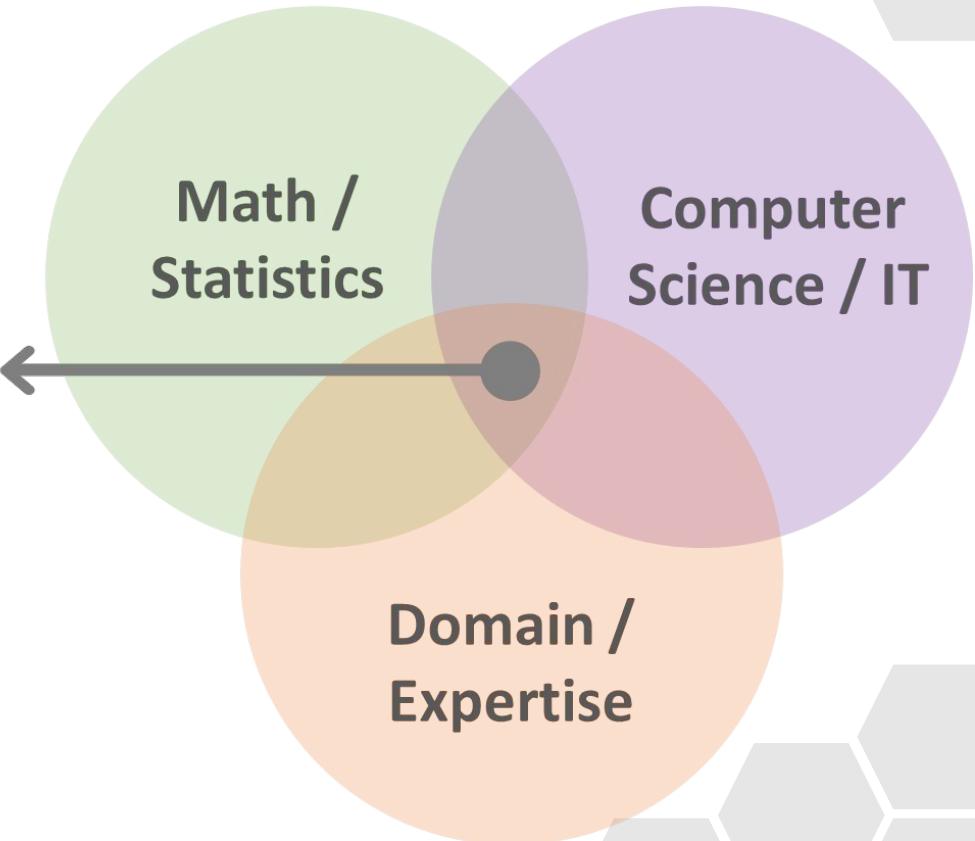
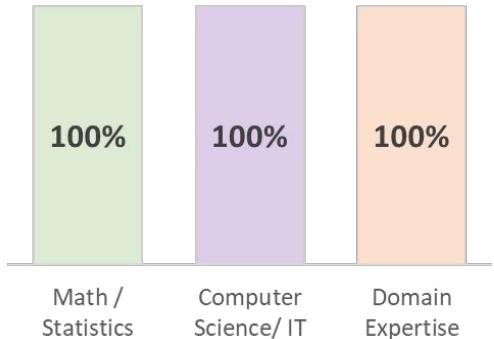
**Math /
Statistics**

**Computer
Science / IT**

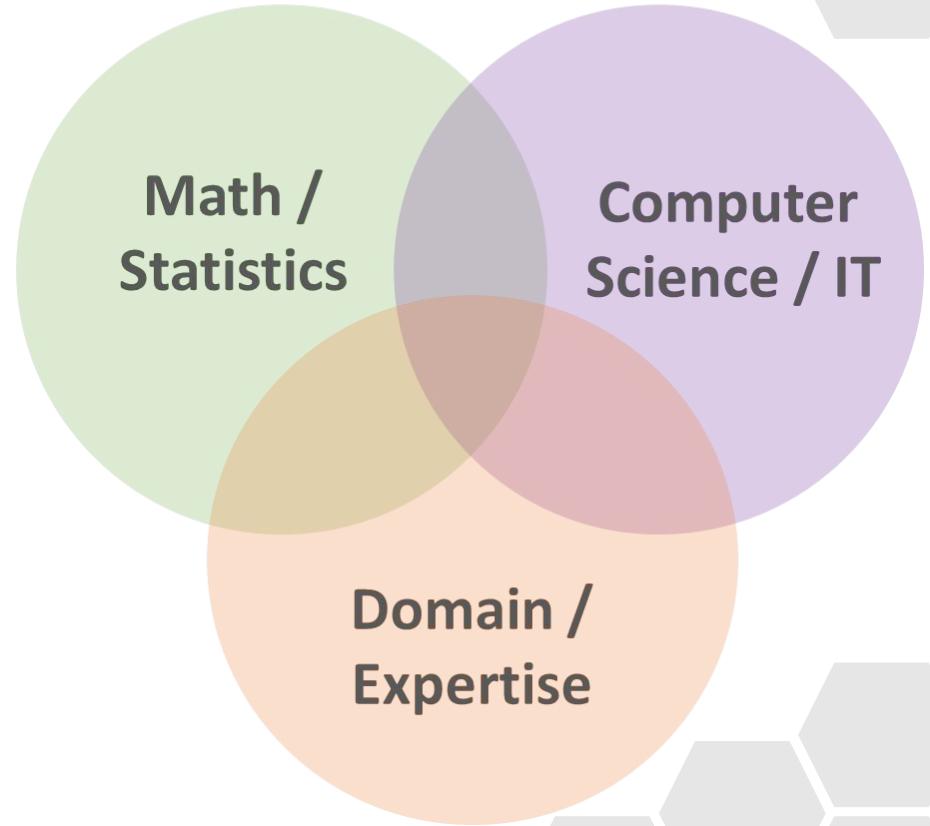
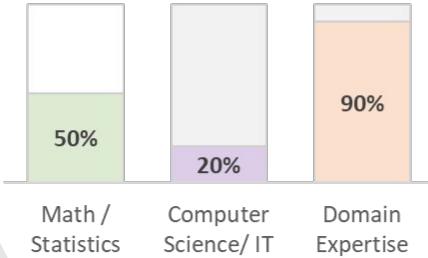
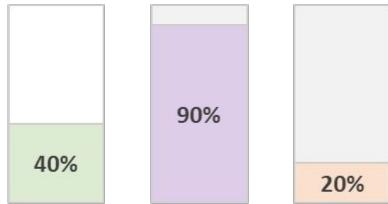


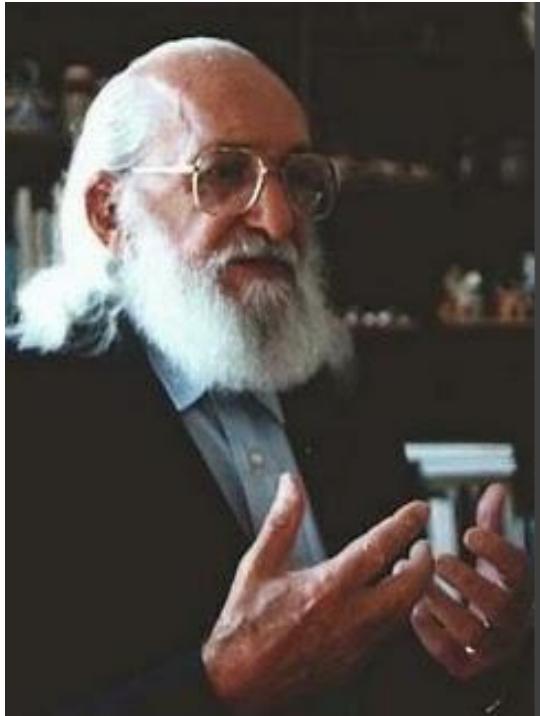
**Domain /
Expertise**

Expectation



Reality





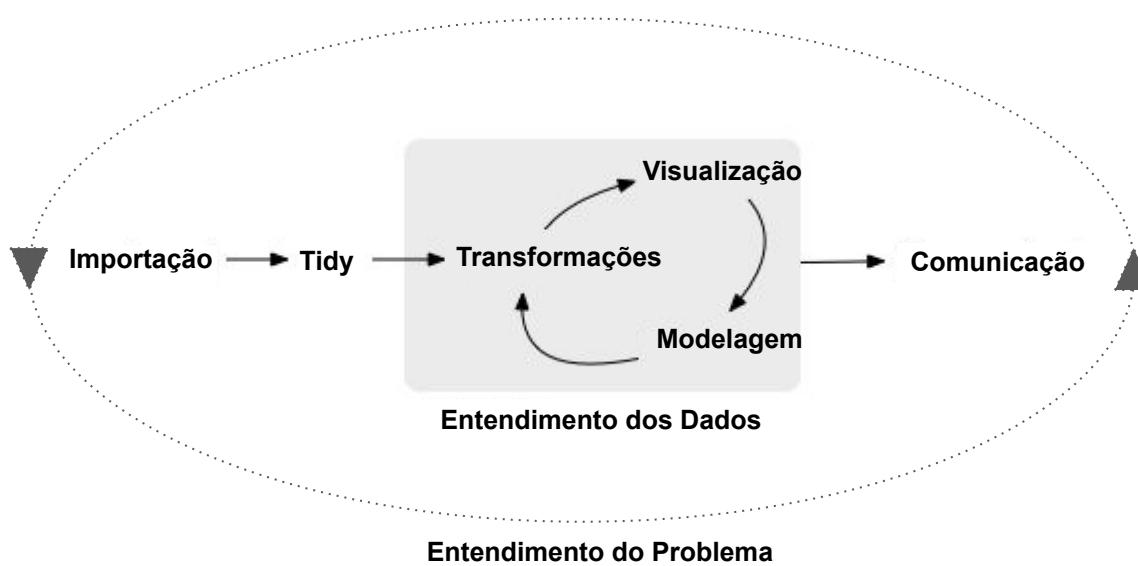
Ninguém ignora tudo.
Ninguém sabe tudo.
Todos nós sabemos
alguma coisa. Todos nós
ignoramos alguma coisa.
Por isso aprendemos
sempre.

Paulo Freire

R na Ciência de Dados

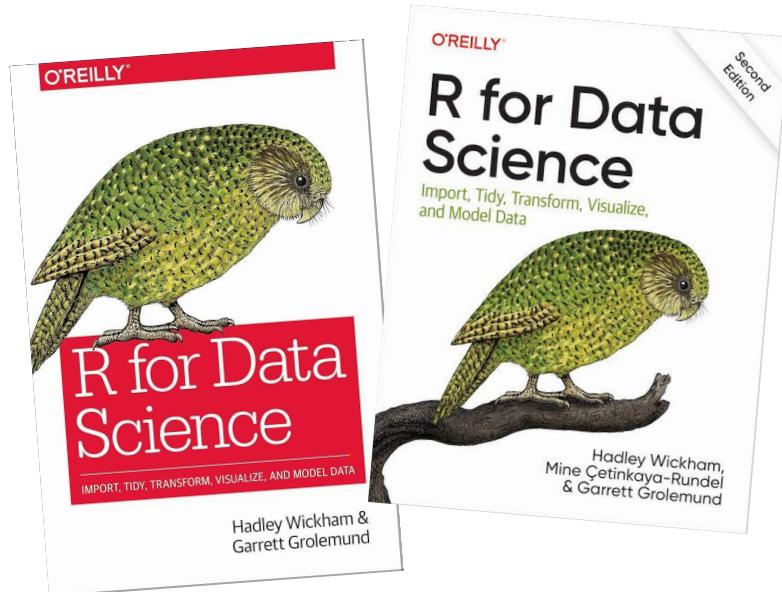
R na Ciência de Dados

Vamos explorar o R do ponto de vista da Ciência de Dados, considerando um ciclo usual de análise, contemplando: leitura, tidy (organização), transformações, visualizações, modelagens e comunicação, bem como as possíveis repetições cabíveis:



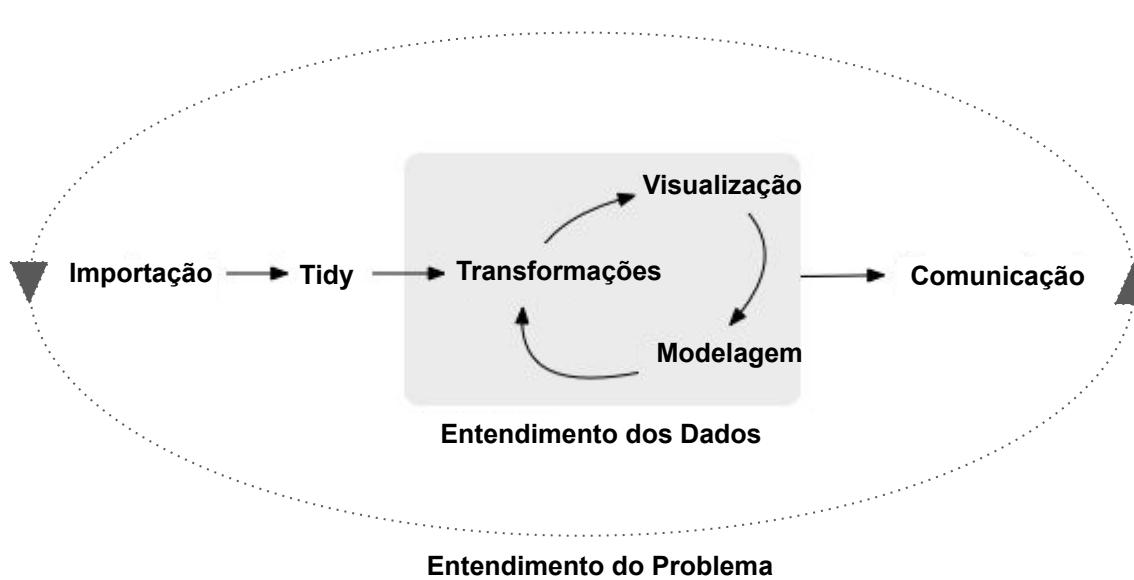
R na Ciência de Dados

Este ciclo é baseado no livro “R for Data Science” disponível no link: <https://r4ds.had.co.nz/>
Alternativamente a 2^a edição que está sendo traduzida para Português pelas R-Ladies, disponível no link: <https://cienciadedatos.github.io/pt-r4ds/>

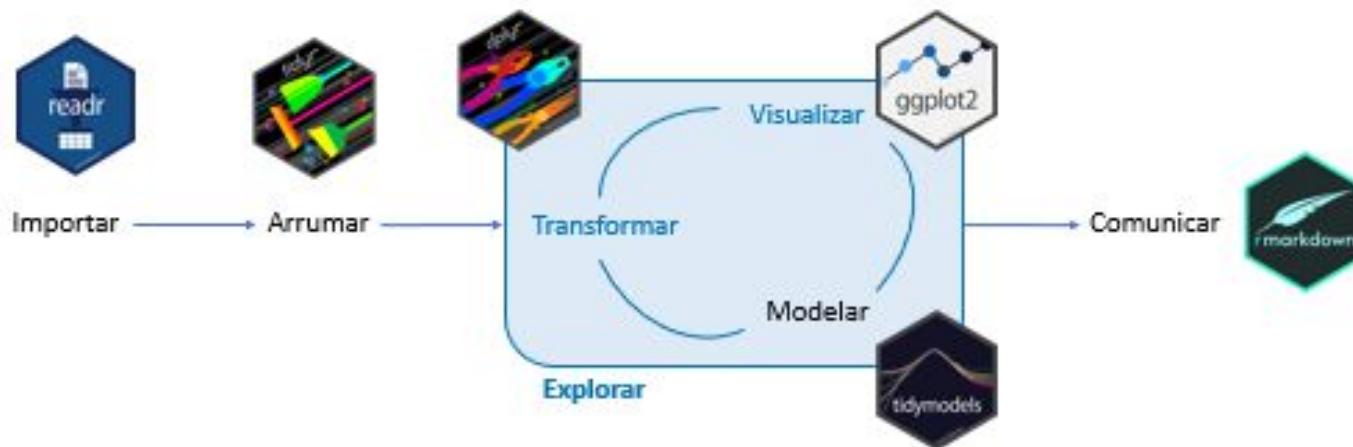


R na Ciência de Dados

Voltando ao nosso ciclo, iremos estudar a sua implementação considerando a coleção de pacotes do tidyverse



Ciclo da Ciência de Dados vs. Pacotes do tidyverse



Ciclo da Ciência de Dados por Hadley Wickham (PT)
<https://r4ds.had.co.nz/explore-intro.html>

R – O universo tidyverse



O tidyverse é um conjunto de bibliotecas que compartilham a mesma gramática, contemplando todo o ciclo de análise da ciência de dados, e tendo como base dados tidy - uma forma padronizada de vincular a estrutura de um conjunto de dados (seu layout físico) com sua semântica (seu significado):

```
install.packages("tidyverse")
```

```
library(tidyverse)
#> -- Attaching packages ----- tidyverse 1.2.1 --
#> v ggplot2 3.1.1      v purrr    0.3.2
#> v tidyr   0.8.3      v dplyr    0.8.0.1
#> v readr   1.3.1      v stringr  1.4.0
#> v ggplot2 3.1.1      vforcats  0.4.0
#> -- Conflicts ----- tidyverse_conflicts() --
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()   masks stats::lag()
```

R – Dados: Tidy

Dados tidy possuem uma estrutura padronizada visando relacionar o layout dos dados com seu significado. Para tal temos que, independente da origem, características, problemas e layout dos dados, estes precisam ser organizados garantindo que:

cada variável tenha a sua própria coluna;

cada observação tenha a sua própria linha; e

cada valor tenha a sua própria célula.

country	year	cases	population
Afghanistan	1990	45	147071
Afghanistan	2000	566	2095360
Brazil	1999	3737	17206362
Brazil	2000	8488	17404898
China	1999	21258	127215272
China	2000	21366	12803583

variables

country	year	cases	population
Afghanistan	1990	45	147071
Afghanistan	2000	566	2095360
Brazil	1999	3737	17206362
Brazil	2000	8488	17404898
China	1999	21258	127215272
China	2000	21366	12803583

observations

country	year	cases	population
Afghanistan	1990	45	147071
Afghanistan	2000	566	2095360
Brazil	1999	3737	17206362
Brazil	2000	8488	17404898
China	1999	21258	127215272
China	2000	21366	12803583

values

A

Untidy Data

species	habitat	weight	length	latitude/longitude	date
Alligator mississippiensis	swamp	431 lb	4 ft 2	29.531,-82.184	Sept 15, 2015
Puma concolor	forest	125 lb	2.2m	29.125,-81.682	08/10/2015
Ursus americanus	forest	88 kg	133 cm	N29°7'30"/W81°40'55.2"	07-13-2015

B

Tidy Data

meta-data

data

species_code	date	station_code	weight_kg	length_cm
TSN 551771	2015-09-15	1	196	127
TSN 55247	2015-08-10	2	57	220
TSN 180544	2015-07-13	2	88	133

station_code	habitat	latitude	longitude
1	swamp	29.531	-82.184
2	forest	29.125	-81.682

species_code	class	genus	species
TSN 551771	Reptilia	Alligator	mississippiensis
TSN 55247	Mammalia	Puma	concolor
TSN 180544	Mammalia	Ursus	americanus

"Dad why is my sisters name Rose"
"Because your mother loves Roses"
"Thanks Dad"
"No problem **library(tidyverse)**"





R – Dados: Tidy

No R tal estrutura pode ser alcançada por meio dos data.frames, porém as bibliotecas do tidyverse trabalham com uma estrutura mais moderna, chamada `tbl_df` ou `tibble`.

- Note que a visualização que temos ao chamar os dados no formato `'tibble'` é muito mais simples e descriptiva que a visualização original. Adicionalmente questões como performance e consistência também apresentam melhorias frente aos `'data.frames'`.

```
#como data.frame (estrutura original)
class(mtcars)
#> [1] "data.frame"
#como tibble
mtcars_2 <- as_tibble(mtcars)
class(mtcars_2)
#> [1] "tbl_df"     "tbl"        "data.frame"
#visualização como tibble:
mtcars_2
#> # A tibble: 32 x 11
#>   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
#>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 21      6   160   110  3.9   2.62  16.5    0     1     4     4
#> 2 21      6   160   110  3.9   2.88  17.0    0     1     4     4
#> 3 22.8    4   108   93   3.85  2.32  18.6    1     1     4     1
#> 4 21.4    6   258   110  3.08  3.22  19.4    1     0     3     1
#> 5 18.7    8   360   175  3.15  3.44  17.0    0     0     3     2
#> 6 18.1    6   225   105  2.76  3.46  20.2    1     0     3     1
#> 7 14.3    8   360   245  3.21  3.57  15.8    0     0     3     4
#> 8 24.4    4   147.   62   3.69  3.19  20      1     0     4     2
#> 9 22.8    4   141.   95   3.92  3.15  22.9    1     0     4     2
#> 10 19.2   6   168.   123  3.92  3.44  18.3    1     0     4     4
#> # ... with 22 more rows
```



R – Gramática: Pipe

Em relação a gramática o tidyverse permite que trabalhemos com a sintaxe utilizada até então, em que a leitura de uma sequência de operações aplicadas a um objeto é feita de dentro para fora, isto é:

```
#leitura de dentro para fora
funçãoN(...(função2(função1(dados))))
```

- Contudo os pacotes do tidyverse oferecem recursos que são melhor aproveitados quando aplicamos operações em sequência, por meio do operador pipe %>%:

```
#leitura em sequência
dados %>%
  função1() %>%
  função2() %>%
  ...
  funçãoN()
```

Assim, além do claro ganho em termos de leitura, passamos a ter uma análise modular, de modo que podemos alterar, remover ou inserir tais módulos de forma simples, sem comprometer os demais.



R – Gramática: Pipe

- O operador `%>%` foi originalmente introduzido no R por meio da biblioteca `magrittr`, porém ganhou tamanha importância que passou a ser contemplado por diversos pacotes, inclusive, claro, o `tidyverse`. A título de comparação, segue um exemplo contemplando os dois paradigmas descritos, em qual deles seria mais simples retirar o log?

```
#assim, aplicar o modulo, o log e arredondar para 2 casas decimais:  
floor(log(abs(-3:3)))  
#> [1] 1 0 0 -Inf 0 0 1
```

#pode ser reescrito como:

```
-3:3 %>%  
  abs() %>%  
  log() %>%  
  floor()  
#> [1] 1 0 0 -Inf 0 0 1
```

#ou ainda com |>
`-3:3 |>`
`abs() |>`
`log() |>`
`floor()`

#reflita: qual o ganho?



R – O Workflow de análise

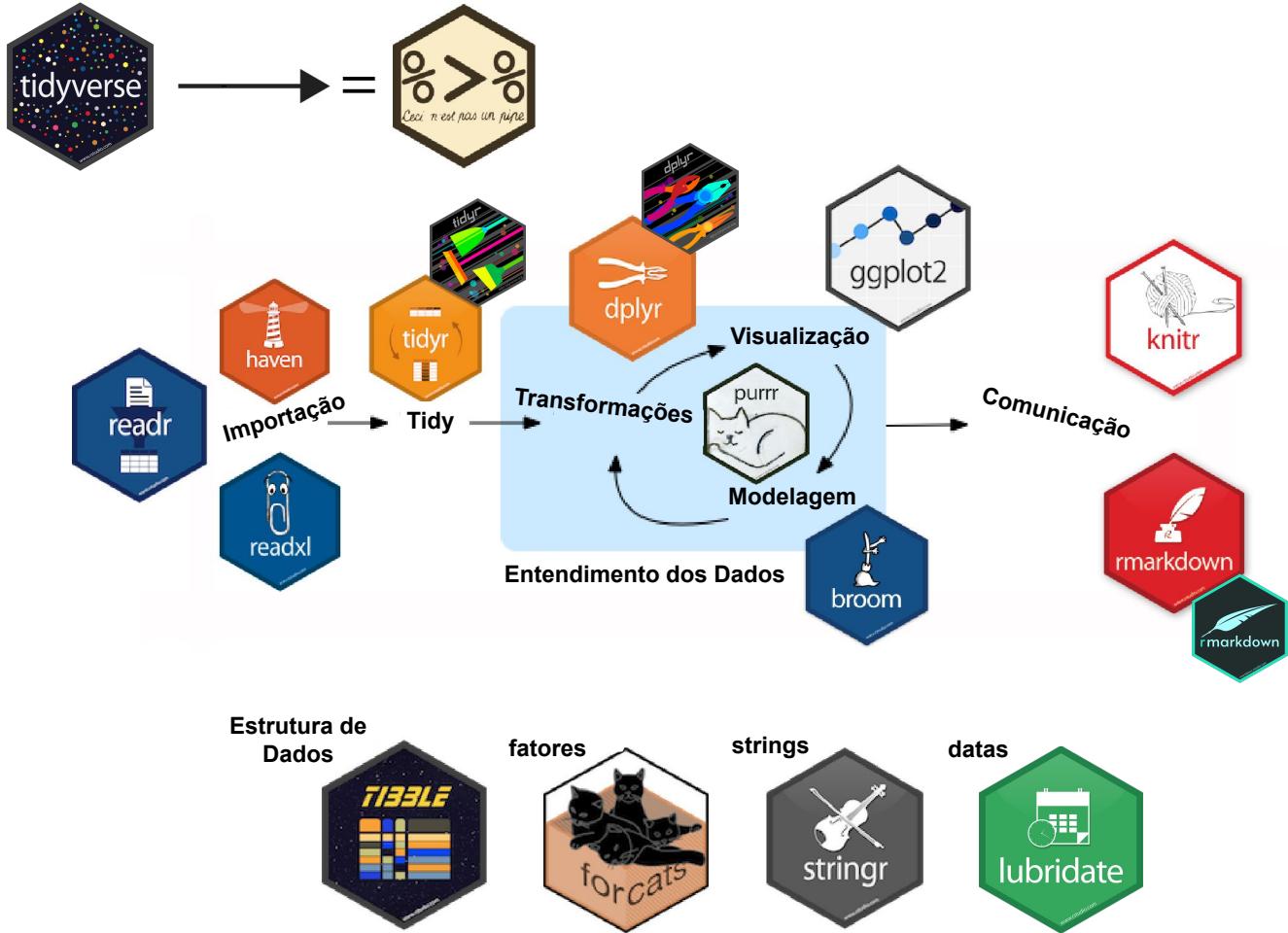
Por fim temos a questão dos pilares do workflow de análise, apresentado no início deste capítulo, e para exemplificar como o tidyverse conversa com este flow, segue a descrição das bibliotecas que são instaladas e carregadas ao instalarmos e carregarmos o tidyverse:

- **readr**: leitura dos dados
- **tibble**: opção ao data frame, otimizada
- **tidyr**: reformulação de layout dos dados
- **dplyr**: manipulação de dados
- **forcats**: operações com variáveis categóricas
- **stringr**: operações com strings
- **lubridate**: operações para trabalhar com datas
- **ggplot2**: criação de gráficos
- **purrr**: programação funcional

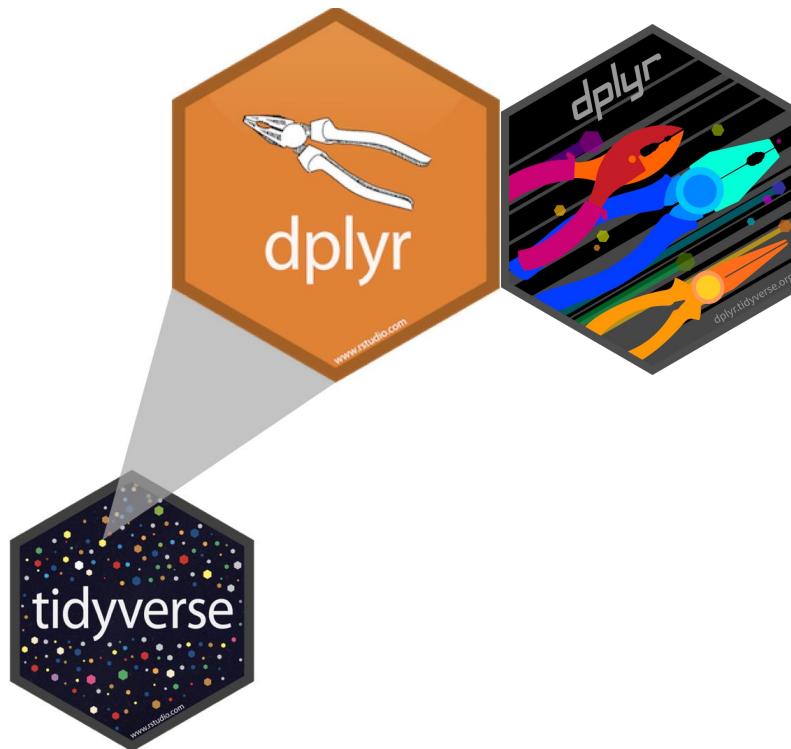
Adicionalmente, algumas bibliotecas apesar de serem instaladas juntamente com o tidyverse, não são carregadas com o library(tidyverse). Como por exemplo: **readxl** para leitura e escrita de arquivos .xls e .xlsx; **haven** para leitura e escrita de arquivos SPSS, Stata, e SAS; e **dbplyr** que permite que você use tabelas de banco de dados remotas convertendo o código dplyr em SQL. Para acessar a lista completa e atualizada de bibliotecas, acesse: <https://www.tidyverse.org/packages/>

Tidyverse:

Somando todas estas bibliotecas ao Rmarkdown, recurso que permite a execução de códigos e relatórios automaticamente. O workflow de análise passa a ser:



R – Manipulação + de variáveis





R – Manipulação + Variáveis

A próxima etapa no processo de análise é a manipulação da base:

- `select()` - seleciona variáveis, permitindo o uso de recursos como: `starts_with()`, `matches()`, `num_range()`, ou `everything()`
- `mutate()` - cria/modifica variáveis
- `rename()` - renomeia variáveis

```
#base exemplo
iris[1:5, ] %>% as_tibble()
#> # A tibble: 5 x 5
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#>   <dbl>       <dbl>       <dbl>       <dbl>   <fct>
#> 1     5.1        3.5        1.4        0.2  setosa
#> 2     4.9        3          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          3.6        1.4        0.2  setosa

#exemplo: seleção, mudança de nome, e definição de uma nova variável
iris[1:5, ] %>%
  as_tibble() %>%
  select(Sepal.Length) %>%
  rename(variavel_original = Sepal.Length) %>%
  mutate(variavel_original_normalizada = variavel_original %>% scale)
#> # A tibble: 5 x 2
#>   variavel_original variavel_original_normalizada[,1]
#>   <dbl>                  <dbl>
#> 1     5.1                 1.16
#> 2     4.9                 0.193
#> 3     4.7                -0.772
#> 4     4.6                -1.25
#> 5     5                  0.675
```

R – Manipulação + Casos



- `filter()` - filtra linhas da base de dados a partir de critérios lógicos
- `slice_()` - seleciona linhas por suas posições ordinais
- `top_n()` - ordena as primeiras `n` observações das colunas listadas
- `arrange()` - ordena as linhas de acordo com as colunas especificadas
- `distinct()` - remove as linhas duplicadas de uma dada entrada

```
# exemplo: seleção segundo padrão de nomenclatura + filtro e ordenação
iris %>%
  select(starts_with("Petal")) %>%
  filter(Petal.Length > 6, Petal.Width > 2) %>%
  arrange(Petal.Width)
#>   Petal.Length Petal.Width
#> 1       6.6      2.1
#> 2       6.7      2.2
#> 3       6.9      2.3
#> 4       6.1      2.3
#> 5       6.1      2.5

# exemplo: valor máximo de Sepal.Length para cada specie
iris %>%
  group_by(Species) %>%
  top_n(1, Sepal.Length)
#> # A tibble: 3 x 5
#> # Groups:   Species [3]
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#>         <dbl>     <dbl>      <dbl>      <dbl>   <fct>
#> 1       5.8        4       1.2       0.2 setosa
#> 2       7          3.2       4.7       1.4 versicolor
#> 3      7.9        3.8       6.4       2 virginica
```

R – Manipulação + Sumarização



- `group_by()` - manipula a base de dados segmentando por cada “grupo” da variável especificada e, posteriormente, combina os resultados, considerando os comandos seguintes dados.

```
#exemplo: calculo da quantidade de observações de cada categoria, e
# média para a variável Petal.Length considerando a visão por Species
iris %>%
  group_by(Species) %>%
  summarise(N = n(),
            PetalLength_media = mean(Petal.Length))
#> # A tibble: 3 x 3
#>   Species      N PetalLength_media
#>   <fct>     <int>          <dbl>
#> 1 setosa      50           1.46
#> 2 versicolor  50           4.26
#> 3 virginica  50           5.55
```

- `summarise()` - calcula resumos de uma tabela conforme especificação

R – Manipulação + Amostra



- `sample_n()` - seleciona uma amostra aleatória considerando o número de elementos especificado
- `sample_frac()` - seleciona uma amostra aleatória considerando a proporção especificada

```
#exemplo: amostra considerando o número de observações desejado
# seguido de uma amostra considerando proporção
iris %>%
  sample_n(10) %>%
  sample_frac(0.5)
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
#> 1       6.1        2.6       5.6        1.4 virginica
#> 2       5.0        3.4       1.5        0.2  setosa
#> 3       6.2        2.9       4.3        1.3 versicolor
#> 4       6.1        3.0       4.9        1.8 virginica
#> 5       5.9        3.0       5.1        1.8 virginica
```

mudou para slice_*

R – CheatSheet

Data transformation with dplyr :: CHEAT SHEET

dplyr functions work with pipes and expect `tidy data`. In tidy data:



Manipulate Cases

EXTRACT CASES

`Row` functions return a subset of rows as a new table.

`filter(data, ..., preserve = FALSE)` Extract rows that meet logical criteria.

`filter(mtcars, mpg > 20)`

`distinct(data, ..., keep_all = FALSE)` Remove rows with duplicate values.

`distinct(mtcars, mpg, wt)`

`slice(data, ..., preserve = FALSE)` Select rows by position.

`slice(mtcars, 10:15)`

`slice_sample(data, ..., n, prop, weight_by = NULL, replace = FALSE)` Randomly select rows.

Use `n` to select a number of rows and `prop` to select a fraction of rows.

`slice_sample(mtcars, n = 5, replace = TRUE)`

`slice_min(data, order_by = ..., n, prop, with_ties = TRUE) and slice_max(...)` Select rows with the lowest and highest value.

`slice_min(mtcars, prop = 0.25)`

`slice_head(data, n, prop) and slice_tail(...)` Select the first or last rows.

`slice_head(mtcars, n = 5)`

Group Cases

Use `group_by(data, ...)`, `add = FALSE`, `drop = TRUE` to create a "grouped" copy of a table grouped by columns in ... `dplyr` functions will manipulate each "group" separately and combine the results.



Use `rowwise(data, ...)` to group data into individual rows. `dplyr` functions will compute results for each row. Also apply functions to list columns. See tidy cheat sheet for list column workflow.



`ungroup(x, ...)` Returns ungrouped copy of table.

`x %>% ungroup()`



Logical and boolean operators to use with filter()

`== < <= > >= !is.na() | & xor()`

See `?base::Logic` and `?Comparison` for help.

ARRANGE CASES

`arrange(data, ..., by_group = FALSE)` Order rows by values of a column or columns (low to high), use `desc` to order from high to low.

`arrange(mtcars, desc(mpg))`

ADD CASES

`add_row(data, ..., before = NULL, after = NULL)` Add one or more rows to a table.

`add_rows(data, ..., n = 2, id = 2)`

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

`pull(data, var = "name", name = NULL)` Extract column values as a vector, by name or index.

`pull(mtcars, wt)`

`select(data, ..., keep_all = FALSE)` Extract columns as a table.

`select(mtcars, mpg, wt)`

`relocate(data, ..., before = NULL, after = NULL)` Move columns to new position.

`relocate(mtcars, mpg, cyl, after = last_col)`

Use these helpers with select() and across()

`contains(match)` all_of(x) | any_of(x)

`contains_all(match)` all_of(x) | any_of(x, ..., vars = TRUE)

`ends_with(match)` matches_with(match)

`starts_with(match)` matches_starting_with(match)

MANIPULATE MULTIPLE VARIABLES AT ONCE

`across(data, fns, ..., names = NULL)` Summarise or mutate multiple columns in the same way.

`summarise(mtcars, across(cyl, mean))`

`c_across(data, c)` Compute across columns in row-wise mode.

`transmute(across(mtcars, total = sum(c_across(1:2)))`

MAKE NEW VARIABLES

Apply `vectorized` functions to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).

vectorized function

`mutated(data, ..., keep = "all", before = NULL, after = NULL)` Compute new column(s). Also `add_column()`, `add_count()`, and `add_tally()`.

`mutate(mtcars, gmt = 1 / mpg)`

`transmute(data, ...)` Compute new column(s), drop others.

`transmute(mtcars, gmt = 1 / mpg)`

`renamed(data, ...)` Rename columns. Use `rename_with()` to rename with a function.

`renamed(mtcars, distance_dif)`



Vectorized Functions

TO USE WITH MUTATE()

`mutate() and transmute()` apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSET

`data[lag]` - offset elements by 1

`data[lead]` - offset elements by -1

CUMULATIVE AGGREGATE

`dplyr::cumall()` - cumulative all

`dplyr::cumany()` - cumulative any

`dplyr::cummean()` - cumulative mean

`dplyr::cumprod()` - cumulative prod

`dplyr::cumsum()` - cumulative sum

RANKING

`cume_dist()` proportion of all values <=

`dense_rank()` rank with ties = min, no gaps

`min_rank()` - rank with ties = min

`percent_rank()` - min_rank scaled to [0,1]

`row_number()` - rank with ties = "first"

MATH

`*`, `^`, `*, %%, %%`, `%/%` - arithmetic ops

`log10()`, `log2()` - logs

`<`, `<=`, `>`, `>=` - logical comparison

`between()` - x <= left & x <= right

`near()` - safe == for floating point numbers

MISCELLANEOUS

`case_when()` - multi-case if_else()

`match_case()` - case_when(...)

`height = 200` (max > 200 - "large",

`width = 200` (max > 200 - "wide")

`TRUE`, `FALSE`, `NA` - other

`coalesce(...)` - first non-NA values by element, across a set of vectors

`if_else(f, t, else_f, else_t)`

`na_if(f)` - replace specific values with NA

`pmax(...)` - element-wise max

`pmin(...)` - element-wise min

ROW NAMES

Tidy data does not use rownames.

Move rownames outside of the column

rownames(data) = "rowname"

rownames(data) = 1:nrow(data)

rownames(data) = c("rowname",

"rowname", ..., "rowname")

Also `rowid_to_column()`

`remove_rownames()`

`has_rownames()`

`remove_rownames()`

Summary Functions

TO USE WITH SUMMARISE()

`summarise()` applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNT

`data[n]` - number of values/rows

`data[, n]` - # of unique

`sum(is.na(x))` - # of non-NAs

POSITION

`mean(..., na.rm = TRUE)` - mean

`median()` - median

LOGICAL

`mean(..., na.rm = TRUE)` - proportion of TRUE's

`sum(..., na.rm = TRUE)` - sum of TRUE's

ORDER

`data[1]` - first value

`data[-1]` - last value

`data[c]` - value in nth location of vector

RANK

`quantile(..., n)` - nth quantile

`min(...)` - minimum value

`max(...)` - maximum value

SPREAD

`IQR()` - Inter-Quartile Range

`mad(...)` - median absolute deviation

`sdi(...)` - standard deviation

`var(...)` - variance

ROW NAMES

Tidy data does not use rownames.

Move rownames outside of the column

rownames(data) = "rowname"

rownames(data) = 1:nrow(data)

rownames(data) = c("rowname",

"rowname", ..., "rowname")

Also `rowid_to_column()`

`remove_rownames()`

`has_rownames()`

`remove_rownames()`

Combine Tables

COMBINE VARIABLES

`mnm` + `mnp` = `mnp`

`bind_rows(..., id = NULL)`

Bind rows of the other as a single table. Set `id` to a column name to add a column of the original table names (as pictured).

COMBINE CASES

`mnm` + `mnp` = `mnp`

`bind_rows(..., id = NULL)`

Bind rows of another.



Use a "Filtering Join" to filter one table against the rows of another.

`x` + `y` = `y`

`semi_join(x, y, by = NULL, copy = FALSE,`

`..., nz_matches = "na")` Return rows of `x` that have a match in `y`. Use to see what will be included in a join.

`anti_join(x, y, by = NULL, copy = FALSE,`

`..., nz_matches = "na")` Return rows of `x` that do not have a match in `y`. Use to see what will not be included in a join.

Use a "Nested Join" to inner join one table to another into a nested data frame.

`nest_join(x, y, by = NULL, copy = FALSE,`

`..., suffix = c("x", "y"))` - keep = FALSE, `na_matches = "na")` Return data. Retain only rows with matches.

`full_join(..., by = NULL, copy = FALSE,`

`..., suffix = c("x", "y"))` - keep = FALSE, `na_matches = "na")` Return all values, all rows.

Fim da 4^a aula!

<https://github.com/tidyverse/dplyr>

R – Manipulação + Combinando Bases

- `inner_join()` - retorna todas as linhas da base à esquerda que possuem valores correspondentes na base à direita.
- `left_join()` - retorna todas as linhas de x e todas as colunas de x e y. Linhas em x sem correspondência em y terão valores de NA nas novas colunas.
- `full_join()` - retorna todas as linhas e todas as colunas de x e y. Onde não há valores correspondentes, retorna NA para o ausente.

```
#bases para exemplo
band_members %>% glimpse()
#> Observations: 3
#> Variables: 2
#> $ name <chr> "Mick", "John", "Paul"
#> $ band <chr> "Stones", "Beatles", "Beatles"
band_instruments %>% glimpse()
#> Observations: 3
#> Variables: 2
#> $ name <chr> "John", "Paul", "Keith"
#> $ plays <chr> "guitar", "bass", "guitar"
band_instruments2 %>% glimpse()
#> Observations: 3
#> Variables: 2
#> $ artist <chr> "John", "Paul", "Keith"
#> $ plays <chr> "guitar", "bass", "guitar"

#exemplo: em que as variáveis para join possuem nomes iguais
left_join(band_members, band_instruments) %>% glimpse()
#> Joining, by = "name"
#> Observations: 3
#> Variables: 3
#> $ name <chr> "Mick", "John", "Paul"
#> $ band <chr> "Stones", "Beatles", "Beatles"
#> $ plays <chr> NA, "guitar", "bass"

#exemplo: em que as variáveis para join possuem nomes diferentes
left_join(band_members, band_instruments2, by = c("name"="artist")) %>% glimpse()
#> Observations: 3
#> Variables: 3
#> $ name <chr> "Mick", "John", "Paul"
#> $ band <chr> "Stones", "Beatles", "Beatles"
#> $ plays <chr> NA, "guitar", "bass"
```



R – Tidy



- Após a importação dos dados, passamos para a etapa de tidy dos dados, também chamada de tidying data. Para isto iremos primeiramente visualizar os dados, no R existem várias funções built-in para isto, como a função names(), que retorna o nome das colunas, ou str(), que permite a visualização da estrutura de um objeto arbitrário.

```
#exemplo: função `glimpse` para visualização
mtcars %>% glimpse()
#> Observations: 32
#> Variables: 11
#> $ mpg <dbl> 21.0, 21.0, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8, 19....
#> $ cyl <dbl> 6, 6, 4, 6, 8, 6, 8, 4, 4, 6, 6, 8, 8, 8, 8, 8, 4, 4, ...
#> $ disp <dbl> 160.0, 160.0, 108.0, 258.0, 360.0, 225.0, 360.0, 146.7, 1...
#> $ hp <dbl> 110, 110, 93, 110, 175, 105, 245, 62, 95, 123, 123, 180, ...
#> $ drat <dbl> 3.90, 3.90, 3.85, 3.08, 3.15, 2.76, 3.21, 3.69, 3.92, 3.9...
#> $ wt <dbl> 2.620, 2.875, 2.320, 3.215, 3.440, 3.460, 3.570, 3.190, 3...
#> $ qsec <dbl> 16.46, 17.02, 18.61, 19.44, 17.02, 20.22, 15.84, 20.00, 2...
#> $ vs <dbl> 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, ...
#> $ am <dbl> 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, ...
#> $ gear <dbl> 4, 4, 4, 3, 3, 3, 4, 4, 4, 3, 3, 3, 3, 3, 3, 4, 4, ...
#> $ carb <dbl> 4, 4, 1, 1, 2, 1, 4, 2, 2, 4, 4, 3, 3, 4, 4, 4, 1, 2, ...
```

R – Tidy + Dividir/Combinar células



- **separate()** - divide uma única coluna em várias colunas, similarmente a opção `separate_rows()`, que faz a mesma coisa, para linhas.

```
#base para exemplo
table3
#> # A tibble: 6 x 3
#>   country     year    rate
#>   <chr>      <int> <chr>
#> 1 Afghanistan 1999 745/19987071
#> 2 Afghanistan 2000 2666/20595360
#> 3 Brazil      1999 37737/172006362
#> 4 Brazil      2000 80488/174504898
#> 5 China       1999 212258/1272915272
#> 6 China       2000 213766/1280428583
```

```
#exemplo: dividindo colunas
table3 %>% separate(rate, into = c("cases", "pop"))
#> # A tibble: 6 x 4
#>   country     year  cases  pop
#>   <chr>      <int> <chr> <chr>
#> 1 Afghanistan 1999    745 19987071
#> 2 Afghanistan 2000   2666 20595360
#> 3 Brazil      1999  37737 172006362
#> 4 Brazil      2000  80488 174504898
#> 5 China       1999  212258 1272915272
#> 6 China       2000  213766 1280428583
```

R – Tidy + Dividir/Combinar células



- `unite()` - e de maneira oposta as separações, temos a função que combina várias colunas.

```
#base para exemplo
table3_separate <- table3 %>% separate(rate, into = c("cases", "pop"))

#exemplo: separando a coluna rate em duas
table3_separate %>%
  unite(cases, pop, col = "rate", sep = "/")
#> # A tibble: 6 x 3
#>   country     year    rate
#>   <chr>       <int> <chr>
#> 1 Afghanistan 1999 745/19987071
#> 2 Afghanistan 2000 2666/20595360
#> 3 Brazil      1999 37737/172006362
#> 4 Brazil      2000 80488/174504898
#> 5 China       1999 212258/1272915272
#> 6 China       2000 213766/1280428583
```

R – Tidy + Dados Faltantes



- **drop_na()** - exclui todas as linhas que apresentam elementos faltantes.
- **fill()** - substituí os dados faltantes pelo valor mais recente da referida coluna.
- **replace_na()** - substituí os dados faltantes por um valor pré especificado.

```
#base para exemplo
airquality[1:5,]
#>   Ozone Solar.R Wind Temp Month Day
#> 1    41     190  7.4   67    5   1
#> 2    36     118  8.0   72    5   2
#> 3    12     149 12.6   74    5   3
#> 4    18     313 11.5   62    5   4
#> 5    NA      NA 14.3   56    5   5

#exemplo: exclusão das linhas identificadas com NA
airquality[1:5,] %>% drop_na()
#>   Ozone Solar.R Wind Temp Month Day
#> 1    41     190  7.4   67    5   1
#> 2    36     118  8.0   72    5   2
#> 3    12     149 12.6   74    5   3
#> 4    18     313 11.5   62    5   4
airquality[1:5,] %>% fill(c(Ozone, Solar.R))
#>   Ozone Solar.R Wind Temp Month Day
#> 1    41     190  7.4   67    5   1
#> 2    36     118  8.0   72    5   2
#> 3    12     149 12.6   74    5   3
#> 4    18     313 11.5   62    5   4
#> 5    18     313 14.3   56    5   5
airquality[1:5,] %>% replace_na(list(Ozone=0, Solar.R=0))
#>   Ozone Solar.R Wind Temp Month Day
#> 1    41     190  7.4   67    5   1
#> 2    36     118  8.0   72    5   2
#> 3    12     149 12.6   74    5   3
#> 4    18     313 11.5   62    5   4
#> 5     0      0 14.3   56    5   5
```

R – Tidy + Redimensionamento de dados



- **pivot_wider, antigo gather()** -
reorganiza os dados, combinando as colunas especificadas na coluna parâmetro key, e seus valores na coluna parâmetro value. Com tal tipo de ajuste podemos organizar os dados de modo a consultar várias colunas de interesse de uma única vez, algo útil para a aplicação de loops, funções e gráficos, conforme veremos na seção Visualização.

```
#base para exemplo
table4a
#> # A tibble: 3 x 3
#>   country    `1999` `2000`
#>   * <chr>     <int> <int>
#> 1 Afghanistan    745   2666
#> 2 Brazil        37737  80488
#> 3 China         212258 213766

#exemplo: passando as colunas para linhas
table4a_new <- table4a %>%
  gather(`1999`, `2000`, key = "ano", value = "valores")

table4a_new
#> # A tibble: 6 x 3
#>   country     ano   valores
#>   <chr>      <chr>   <int>
#> 1 Afghanistan 1999      745
#> 2 Brazil       1999    37737
#> 3 China        1999   212258
#> 4 Afghanistan 2000     2666
#> 5 Brazil       2000    80488
#> 6 China        2000   213766
```

R – Tidy + Redimensionamento de dados



- **pivot_longer, antigo spread()** - de maneira oposta a função anterior, aqui podemos expandir linhas em colunas:

```
#exemplo: passando linhas para colunas
table4a_new %>%
  spread(ano, valores)
#> # A tibble: 3 x 3
#>   country     `1999` `2000`
#>   <chr>       <int>  <int>
#> 1 Afghanistan    745   2666
#> 2 Brazil        37737  80488
#> 3 China         212258 213766
```

R – Manipulação + Fatores



Conforme já comentado, existem bibliotecas para trabalharmos com diferentes classes de dados. Vamos citar algumas funcionalidades dos três pacotes com tal finalidade contidos no tidyverse:

- `fct_count()` - conta o número de valores de cada nível
- `fct_relevel()` - reordenar os níveis dos fatores
- `fct_explicit_na()` - adicionar o NA como um dos níveis

```
#base para exemplo
gss_cat %>% glimpse
#> Observations: 21,483
#> Variables: 9
#> $ year    <int> 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, ...
#> $ marital <fct> Never married, Divorced, Widowed, Never married, Divor...
#> $ age     <int> 26, 48, 67, 39, 25, 25, 36, 44, 44, 47, 53, 52, 52, 51...
#> $ race    <fct> White, White, White, White, White, White, White, White...
#> $ rincome <fct> $8000 to 9999, $8000 to 9999, Not applicable, Not appl...
#> $ partyid <fct> "Ind,near rep", "Not str republican", "Independent", ...
#> $ relig    <fct> Protestant, Protestant, Protestant, Orthodox-christian...
#> $ denom   <fct> Southern baptist, Baptist-dk which, No denomination, N...
#> $ tvhours <int> 12, NA, 2, 4, 1, NA, 3, NA, 0, 3, 2, NA, 1, NA, 1, 7, ...

#exemplo: contabilizando a quantidade de religiões declaradas
gss_cat$race %>%
  fct_count()
#> # A tibble: 4 x 2
#>   f                  n
#>   <fct>              <int>
#> 1 Other               1959
#> 2 Black                3129
#> 3 White              16395
#> 4 Not applicable       0
```

R – Manipulação + Strings



- `str_detect()` - identificar a presença de padrões em uma string
- `str_count()` - contabiliza o número de vezes que um padrão é encontrado
- `str_replace()` - substitui um dado padrão em uma string
- `str_to_lower()` - converter strings maiúsculas e minúsculas

```
#base para exemplo
fruit %>%
  enframe %>% #versão tibble para vetores
  glimpse
#> Observations: 80
#> Variables: 2
#> $ name <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1...
#> $ value <chr> "apple", "apricot", "avocado", "banana", "bell pepper", ...

#exemplo: obtenção da quantidade de palavras contendo a string "berry"
fruit %>%
  str_count("berry") %>%
  sum()
#> [1] 14
```

R – Manipulação + Datas



- `as_date()` - converter um objeto para uma data ou hora
- `wday()` - retorna o dia da semana como um número decimal
- `today()` - retorna a data corrente
- `floor_date()` - arredonda para o limite inferior mais próximo da unidade de tempo

```
#exemplo: retorna erro por não ser uma data válida
birthday <- lubridate::dmy("29/02/1971")
#> Warning: 1 failed to parse.

#data ok
birthday <- lubridate::dmy("29/02/1972"); birthday
#> [1] "1972-02-29"

#carregando `lubridate`, tornando o prefixo `lubridate::` desnecessário
library(lubridate)

#obtenção do dia da semana da data especificada
wday(birthday, label = TRUE)
#> [1] ter
#> Levels: dom < seg < ter < qua < qui < sex < sáb
```



R – Importação

- A etapa de importação de dados pode significar ler dados armazenados em: arquivos de diferentes extensões, dados provenientes de outras ferramentas, APIs da Web, e etc.
- Para o primeiro caso, como alternativa ao `read.table()` do R base, temos a versão `readr` do `tidyverse`. Nesta a leitura é mais rápida, com argumentos padronizados, e retornam tibbles ao invés de data frames. Seguem alguns exemplos para a leitura de um arquivo "file":

- `read_csv("file.csv")` - lê arquivos delimitados por vírgula
- `read_csv2("file.csv")` - lê arquivos delimitados por ponto e vírgula
- `read_delim("file.txt", delim="|")` - lê arquivos delimitados por tabulação (exemplo dado para o delimitador |)
- `read_fwf("file.fwf", col_positions = c(1,3,5))` - lê arquivos com largura fixa (exemplo dado para o padrão 1,3,5)

- Adicionalmente temos opções como os pacotes `readxl` e `haven`, anteriormente citadas, o R dispõe de recursos como: `DBI`, `jsonlite`, `xml2`, `httr` ou o `sparklyr`, esta última será melhor detalhada na seção referente à Big Data.

R – Importação

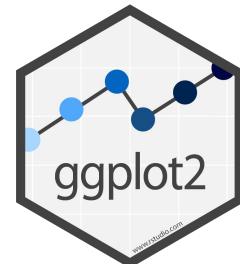
- Em todos os casos é possível especificar características como: nome das colunas, cabeçalho, linhas para serem puladas, ou ainda a pré-especificação da classe de cada uma das colunas do arquivo.
- No caso deste último parâmetro não ser listado, as funções do `readr` irão “adivinar” a classe das colunas, retornando uma mensagem com tais informações, conforme exemplo:

```
read_csv(readr_example("mtcars.csv"))
#> Parsed with column specification:
#> cols(
#>   mpg = col_double(),
#>   cyl = col_double(),
#>   disp = col_double(),
#>   hp = col_double(),
#>   drat = col_double(),
#>   wt = col_double(),
#>   qsec = col_double(),
#>   vs = col_double(),
#>   am = col_double(),
#>   gear = col_double(),
#>   carb = col_double()
#> )
#> # A tibble: 32 x 11
#>   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
#>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 21     6   160   110  3.9   2.62  16.5   0     1     4     4
#> 2 21     6   160   110  3.9   2.88  17.0   0     1     4     4
#> 3 22.8   4   108   93   3.85  2.32  18.0   1     1     4     4
#> 4 21.4   6   258   110  3.08  3.21  19.8   1     1     4     4
#> 5 18.7   8   360   175  3.15  3.44  17.8   0     0     3     4
#> 6 18.1   6   225   105  2.76  3.46  20.0   1     0     3     4
#> 7 14.3   8   360   245  3.21  3.46  16.8   0     0     3     4
#> 8 24.4   4   147.   62   3.69  3.19  22.8   1     1     4     4
#> 9 22.8   4   141.   95   3.92  3.15  23.4   1     1     4     4
#> 10 19.2   6   168.   123  3.92  3.15  22.0   1     0     3     4
#> # ... with 22 more rows
```



Fim da 5^a
aula!

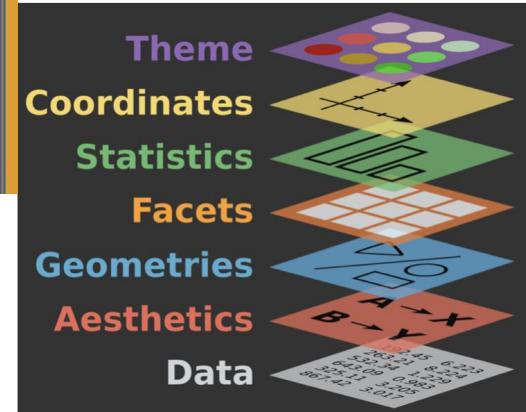
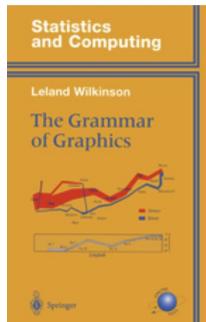
R – Visualização e a gramática dos gráficos



ggplot2 é a implementação da Grammar of Graphics (daí o gg do ggplot2), feita por Hadley Wickham com base no livro The Grammar of Graphics, do Leland Wilkinson

Vantagens:

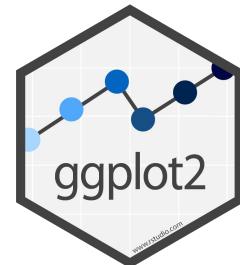
- Sintaxe única e estruturada
- Maior controle e versatilidade
- Portfólio amplo e em expansão
- Oferece recursos de design
- Fomenta a análise semântica dos dados
- Cada variável é representada de forma única
- Por ter como base uma linguagem científica, otimiza a reproduzibilidade
- Ao decodificar gráficos, tornamos a sua comunicação mais clara e intencional



Desvantagens:

Curva de aprendizado inicialmente íngreme

R – Visualização e o ggplot2



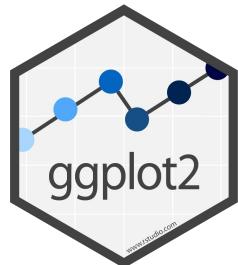
Um gráfico é o mapeamento de: dados (data), atributos estéticos (aesthetics), objetos geométricos (geometries, no caso, pontos, barras, linhas, etc) e camadas opcionais de: facetas, sistemas de coordenadas, temas e transformações estatísticas



- Utilizamos o símbolo `+` para somar as camadas.

```
ggplot(data, aesthetics) +  
  geometries(statistics) +  
  facets +  
  coordinates +  
  theme
```

R – Visualização e o ggplot2

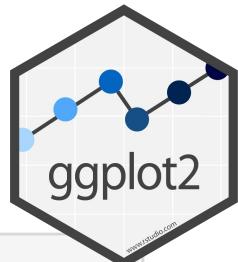
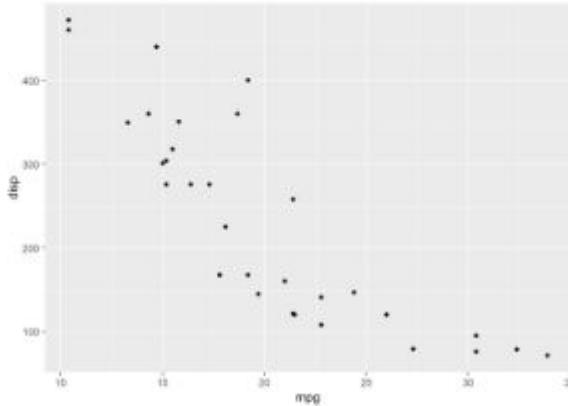


R – Visualização

Gráficos são ferramentas poderosas para a visualização de gráficos, e, quando devidamente utilizados, permitem ao usuário o entendimento das informações de uma forma rápida, permitindo novas questões, hipóteses, necessidades ou pontos de atenção.

E por meio da biblioteca ggplot2, passamos a trabalhar com gráficos de uma perspectiva mais flexível, onde o gráfico é trabalhado como um mapeamento de atributos, e assim podemos dividi-lo em camadas.

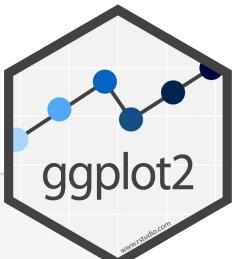
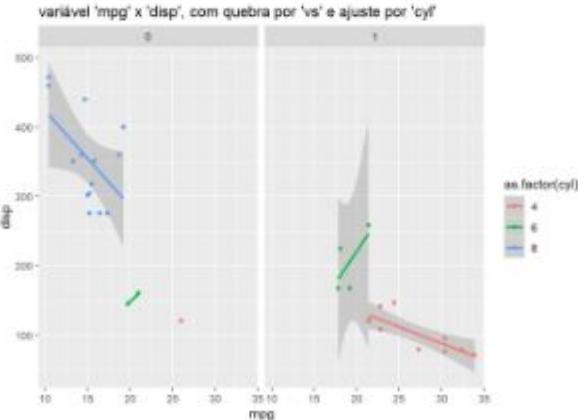
```
#exemplo: gráfico de dispersão  
mtcars %>%  
  ggplot(aes(x=mpg, y=disp)) +  
  geom_point()
```



R – Visualização

Assim temos opções de gráficos como:
geom_boxplot, geom_area, ou
geom_col, além de uma série de opções
para estilização, como: sistema de
coordenadas, escalas, cores, posição,
cores, legendas, quebras, etc.

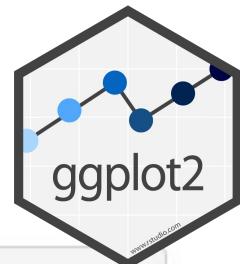
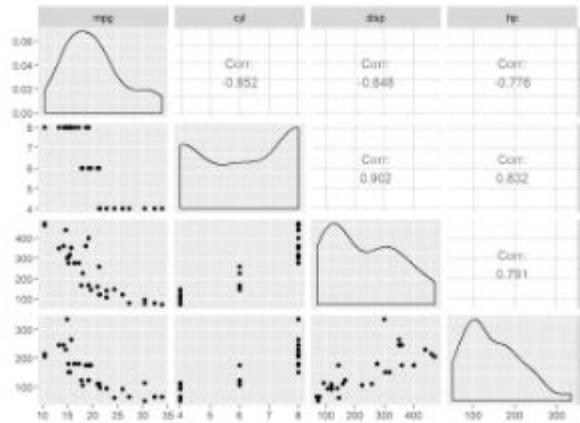
```
#exemplo: gráfico ggplot
mtcars %>%
  ggplot(aes(x = mpg, y = disp, color = as.factor(cyl))) +
  geom_point() +
  facet_grid(~as.factor(vs)) +
  stat_smooth(method = 'lm') +
  labs(title = "variável 'mpg' x 'disp', com quebra por 'vs'  
e ajuste por 'cyl'")
```



R – Visualização

Para visualização dos gráficos de dispersão dois a dois, uma opção é trabalhar com a biblioteca GGally, uma das extensões do `ggplot2`, que permite uma visão mais limpa e completa, uma vez que também retorna a correlação entre os pares e a distribuição individual das variáveis:

```
mtcars[,1:4] %>%
  GGally::ggpairs()
```



R – Comunicação



- Por fim temos a apresentação dos resultados, uma parte crítica do processo, visto que, usualmente, é neste momento que toda a análise passa a ter um significado prático. Para tal podemos exportar as informações de interesse para outras ferramentas, como excel, power point, ou etc, o que não costuma ser muito prático. Como alternativa podemos fazer a análise e montar o report conjuntamente pelo R, garantindo reproduzibilidade e histórico, por meio do Rmarkdown.



Exemplo de um Rmarkdown no Posit cloud

Note que temos scripts .R e o .Rmd, e é o Rmd que permite combinar texto, código e outputs, e gerar arquivos compartilháveis

Botão knitr para gerar um novo arquivo de saída

The screenshot shows the RStudio interface within the Posit Cloud workspace. On the left, the code editor displays an Rmd file named 'step06_visualizações.Rmd' containing R code and a chunk of text. A callout box points to the 'Knit' button in the toolbar above the editor. Another callout box points to the 'Console' tab at the bottom, which shows the output of the 'knit' command. To the right, the 'Environment' pane shows an 'Empty Environment'. Below it, the 'Files' pane lists various project files, including several HTML outputs generated from the Rmd file.

Name	Size	Modified
Rhistory	0 B	Jul 29, 2024, 8:23 PM
aula02.R	2.7 KB	Aug 5, 2024, 6:34 PM
project.Rproj	205 B	Aug 8, 2024, 11:20 AM
step01_guiaDeBolso_04_rmarkdown.Rmd	646.7 KB	Aug 5, 2024, 9:09 PM
step01_guiaDeBolso_04_rmarkdown.html	6.2 KB	Aug 5, 2024, 9:09 PM
step02_importação_01_leitura_de_da.Rmd	635.5 KB	Aug 5, 2024, 9:11 PM
step02_importação_01_leitura_de_da.html	18.6 KB	Aug 5, 2024, 9:10 PM
step03_descrição.html	631.2 KB	Aug 8, 2024, 11:21 AM
step04_exploração.Rmd	13.2 KB	Aug 5, 2024, 9:10 PM
step04_exploração.html	1.4 MB	Aug 8, 2024, 11:24 AM
step05_transformações.Rmd	21.2 KB	Aug 5, 2024, 9:12 PM
step05_transformações.html	669.1 KB	Aug 8, 2024, 11:25 AM
step05_transformações.Rmd	26.2 KB	Aug 5, 2024, 9:13 PM
step06_01_vis_exGapminder.html	14.8 MB	Aug 8, 2024, 11:29 AM
step06_01_vis_exGapminder.Rmd	96.1 KB	Aug 8, 2024, 11:29 AM
step06_visualizações.html	10.8 MB	Aug 8, 2024, 11:35 AM
step06_visualizações.Rmd	23.1 KB	Aug 5, 2024, 9:13 PM

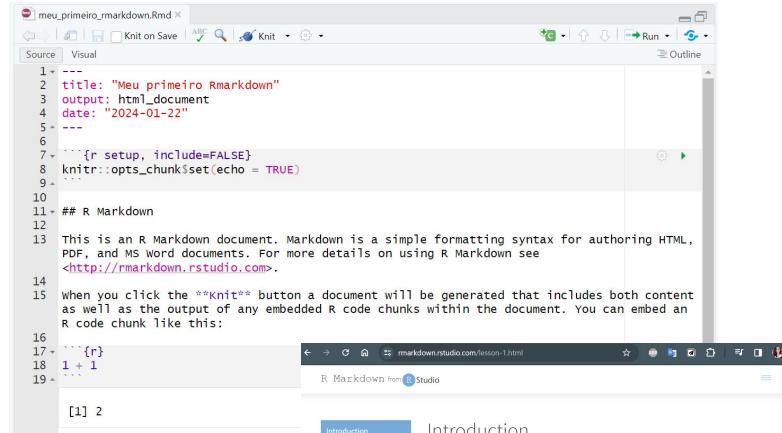
Para rodar apenas um chunk, o output aparecerá em 2 lugares: logo abaixo do chunk e no Console

Para inserir novos chunks (trechos de código)

Os arquivos html são os “outputs” dos respectivos códigos, e podem ser abertos diretamente no browser

Link para o acesso: <https://posit.cloud/content/8481815>

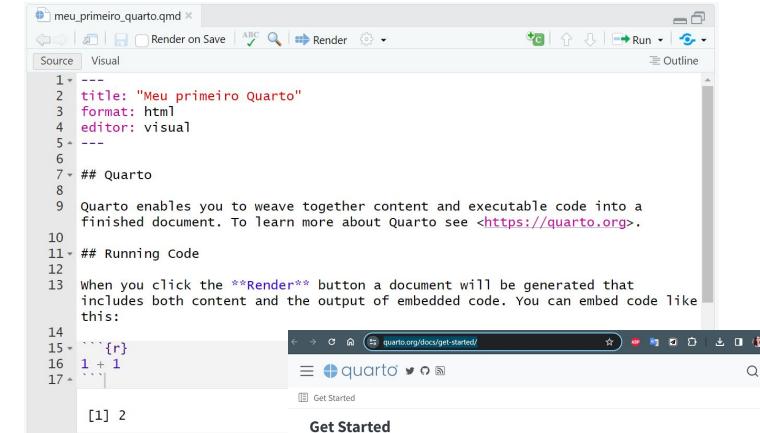
O Quarto trata-se da nova geração do Rmarkdown



A screenshot of the RStudio interface showing an R Markdown document titled "meu_primeiro_markdown.Rmd". The code includes setup code, a title, and a section about R Markdown. The preview pane shows the rendered content, which is identical to the screenshot below.



<https://rmarkdown.rstudio.com/lesson-1.html>



A screenshot of the Quarto interface showing a Quarto document titled "meu_primeiro_quarto.qmd". The code includes a title, format, editor, and sections about Quarto and running code. The preview pane shows the rendered content, which includes a link to the Quarto documentation.



<https://quarto.org/docs/get-started/>

R – Programação

- Como ferramenta transversal a todas as etapas do workflow de análise, temos a programação funcional, que permite a criação de códigos mais rápidos, simples e reproduzíveis. No contexto do tidyverse a biblioteca purrr disponibiliza de uma série de funcionalidades visando aprimorar a programação funcional no R. Como principal função temos o map(), transformando a sua entrada por meio da aplicação de uma função a cada elemento, e retornando uma lista com o mesmo comprimento que a entrada:



```
#exemplo: , contabilização de tal identificação de dados faltantes por coluna
starwars %>%
  map(is.na) %>% #identificação de quais elementos, de cada coluna, são NA's
  map(sum) %>% #contabilização dos dados faltantes por coluna
  glimpse() #sumarização do resultado
#> List of 13
#> $ name      : int 0
#> $ height    : int 6
#> $ mass      : int 28
#> $ hair_color: int 5
#> $ skin_color: int 0
#> $ eye_color : int 0
#> $ birth_year: int 44
#> $ gender     : int 3
#> $ homeworld : int 10
#> $ species    : int 5
#> $ films      : int 0
#> $ vehicles   : int 0
#> $ starships  : int 0

#outras opções de sintaxe para obter o mesmo resultado
#starwars %>% map(~sum(is.na(.)))
#starwars %>% map( function(x) sum(is.na(x)) )
```

R – Programação



O map() retorna listas por default, para que o resultado tenha outra estrutura/classe, podemos utilizar funções como: map_dbl que retorna um vetor número, map_dfc para um data.frame que combina os resultados por coluna, entre outras opções. Adicionalmente :

- walk() - similar ao map, porém sem ter o retorno da lista no console
- pmap() - aplica uma função a um grupo de elementos de um grupo de listas
- append() - adiciona valores ao fim de uma lista

```
#exemplo: para visualizar todos os arquivos de um dado diretório
list.files("\diretório", pattern="*.xlsx") %>%
  map(read_excel) %>%
  walk(glimpse)
```

R – Modelagem



- Após um entendimento mais profundo sobre o comportamento, natureza e especificidades dos dados, podemos passar para a fase de modelagem. No tidyverse a biblioteca broom é direcionada para esta etapa da análise, de um ponto de vista de integração. Em relação à modelagem em si, visto ser algo que demanda conhecimentos prévios, não iremos nos aprofundar. Contudo segue uma lista com opções de bibliotecas para algumas metodologias/áreas:

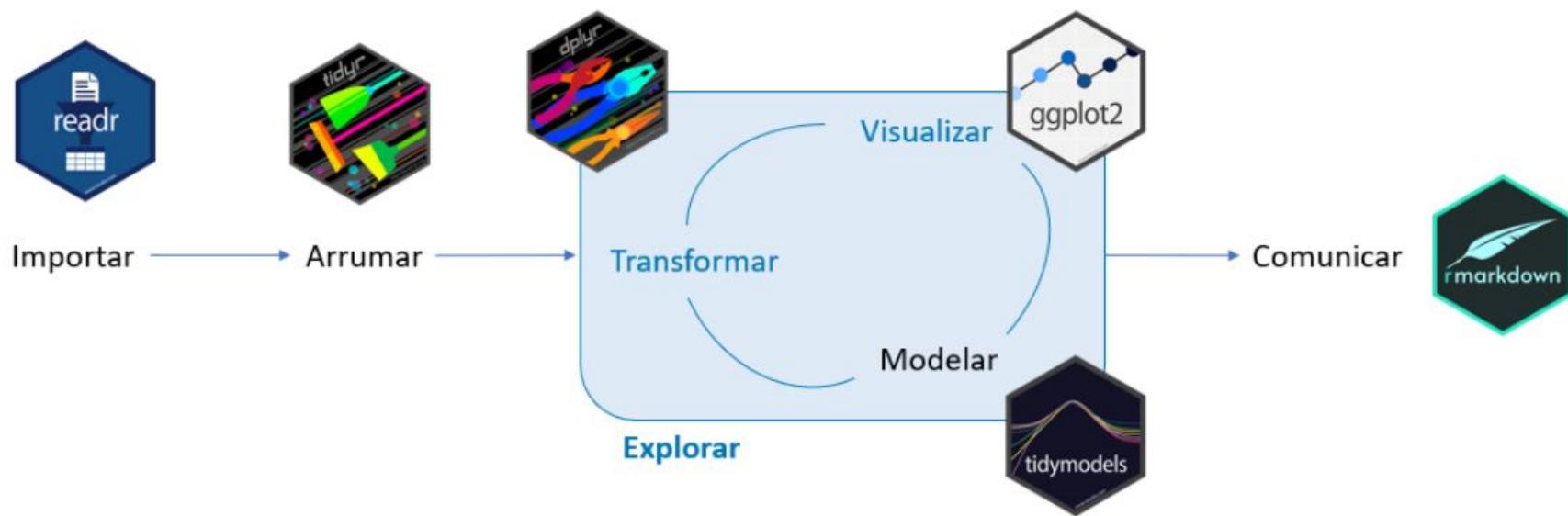
- Árvore de Decisão - `part` e `rpart`
- Clusterização - `stats`, `cluster` e `fpc`
- Deep Learning - `keras`
- Random Forest - `randomForest`
- Redes Neurais - `nnet`, `neuralnet` e `RSNNS`
- Regressões - `stats`, `nlme` e `gbm`
- Séries Temporais - `forecast` e `dtw`
- Text Mining - `tm` e `wordcloud`
- Validação Cruzada - `caret`

Extras

Extra:

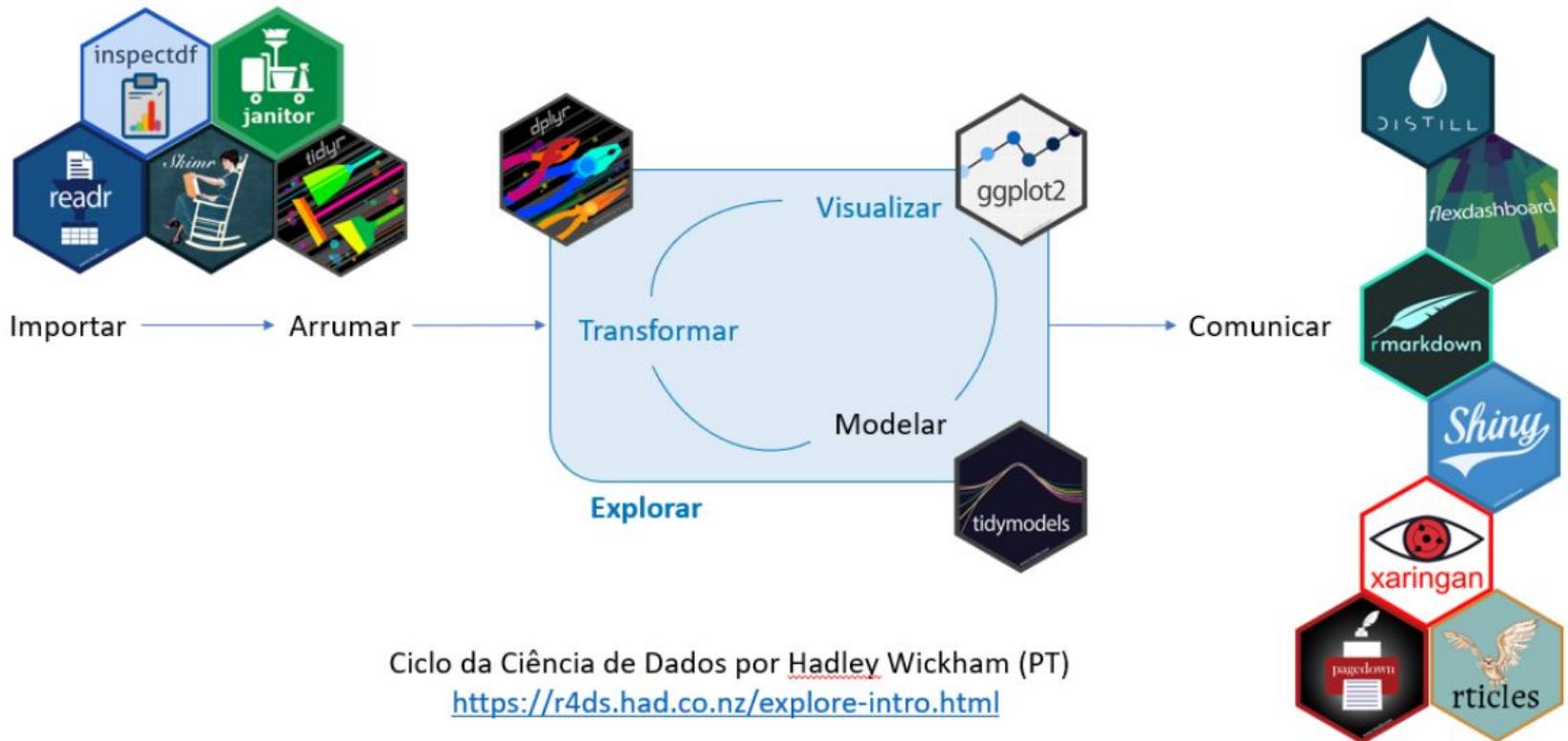
Comunicação com dados no R

Extra: Comunicação



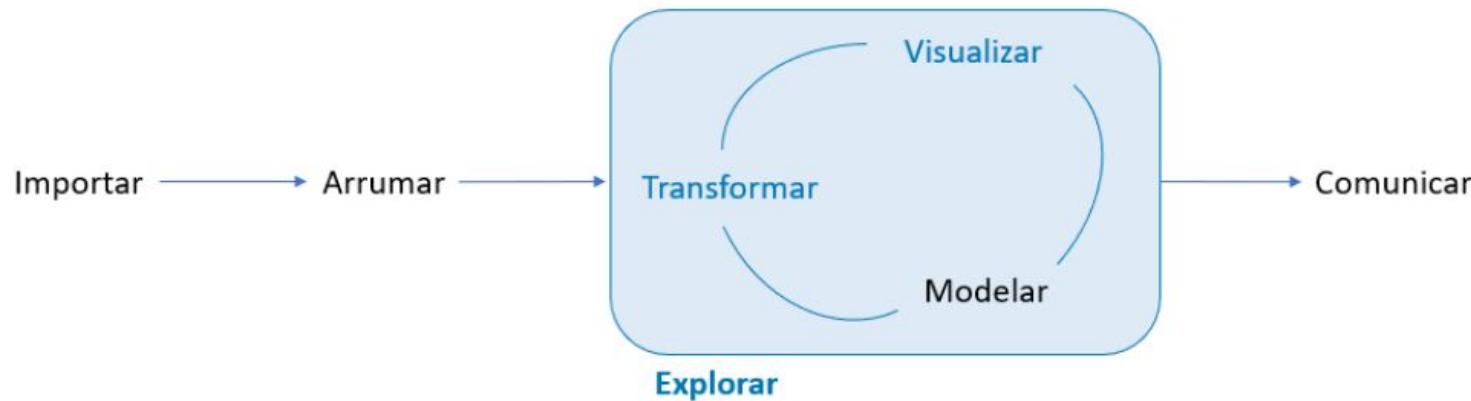
Ciclo da Ciéncia de Dados por Hadley Wickham (PT)
<https://r4ds.had.co.nz/explore-intro.html>

Extra: Comunicação



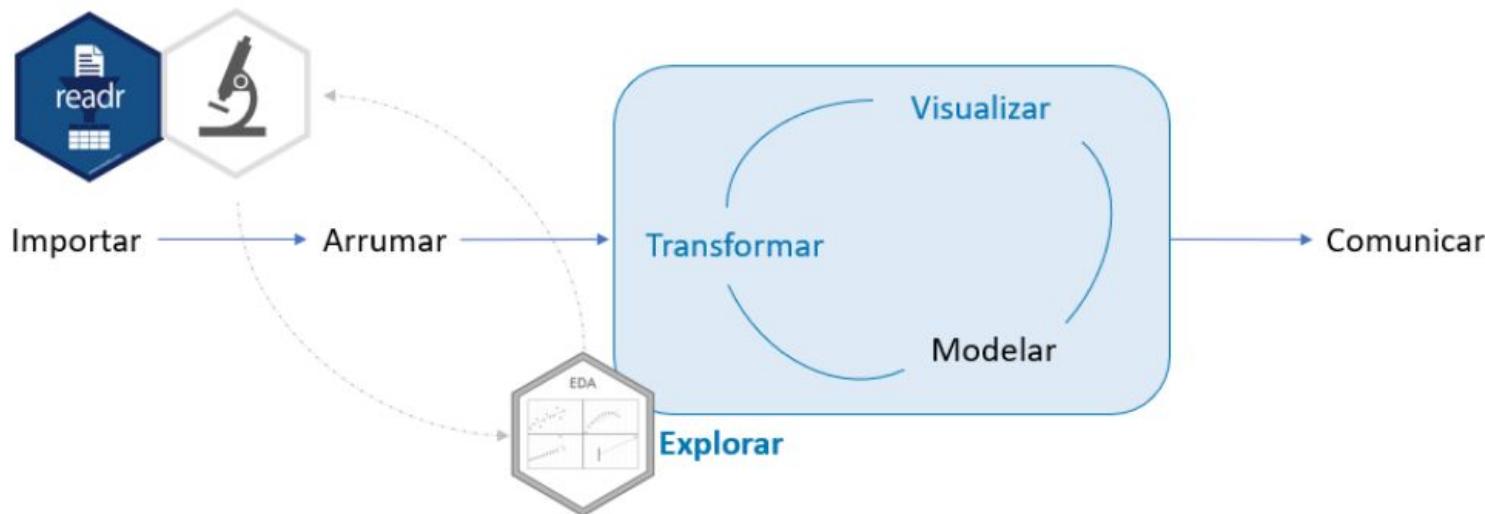
Extra:
Um pouco mais sobre EDA
(Exploratory Data Analysis)
ou
Análise Exploratória de Dados

Extra: EDA (Exploratory Data Analysis)



Ciclo da Ciência de Dados por Hadley Wickham (PT)
<https://r4ds.had.co.nz/explore-intro.html>

Extra: EDA (Exploratory Data Analysis)

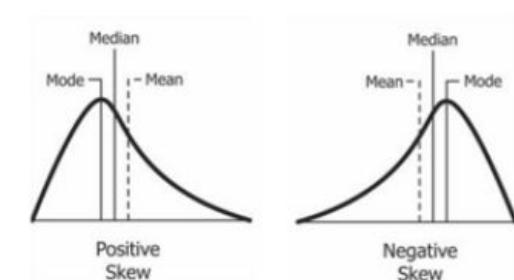
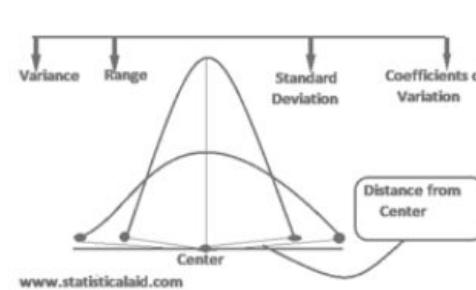
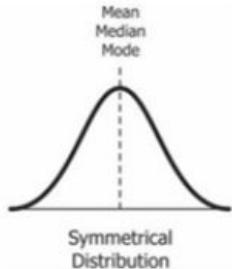


Ciclo da Ciéncia de Dados por Hadley Wickham (PT)
<https://r4ds.had.co.nz/explore-intro.html>

Extra: EDA

Medidas de Resumo:

- Medidas de Posição, Concentração ou Tendência Central: média (μ ou \bar{X}), moda (M_o) e mediana (M_d)
- Medidas de Ordenação (posição relativa): máximo, mínimo, quartis (1º quartil = Q_1 e 3º = Q_3), decis e percentis
- Medidas de Dispersão (absoluta): variância (σ^2 ou s^2), desvio padrão (σ ou s), amplitude total (A_t) e interquartil (IQR)
- Medidas de Dispersão (relativa): coeficiente de variação (CV)
- Medidas de Forma: coeficiente de curtose e coeficiente de assimetria



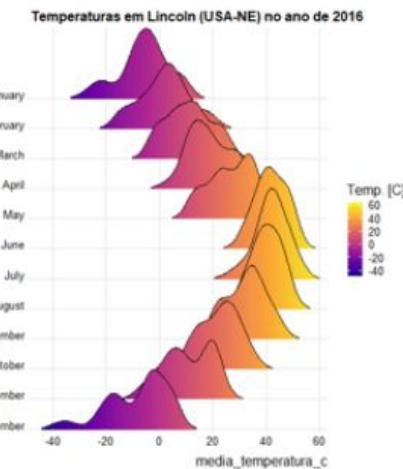
Extra: EDA

Medidas de Resumo:

- Medidas de Posição, Concentração ou Tendência Central: média (μ ou \bar{X}), moda (M_o) e mediana (M_d)
- Medidas de Ordenação (posição relativa): máximo, mínimo, quartis (1º quartil = Q_1 e 3º = Q_3), decís e percentis
- Medidas de Dispersão (absoluta): variância (σ^2 ou s^2), desvio padrão (σ ou s), amplitude total (A_t) e interquartil (IQR)
- Medidas de Dispersão (relativa): coeficiente de variação (CV)
- Medidas de Forma: coeficiente de curtose e coeficiente de assimetria

Caracterização de uma Distribuição de Frequência :

- Concentração: ponto (s) ou intervalo (s) de concentração das observações
- Dispersão: o quanto próximo ou distante as observações estão entre si
- Outliers: observações que se diferenciam de maneira significativa das demais
- Forma: aspectos de forma, como simetria ou assimetria à esquerda ou à direita



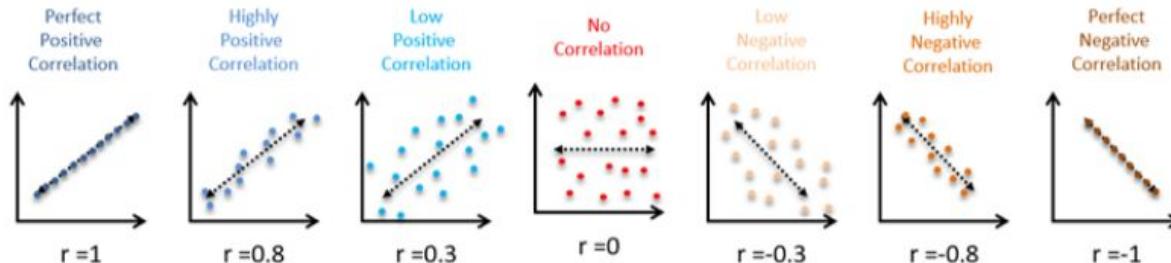
Extra: EDA

Medidas de Resumo (univariadas ou unidimensionais)

- Medidas de Posição, Concentração ou Tendência Central: média (μ ou \bar{X}), moda (M_o) e mediana (M_d)
- Medidas de Ordenação (posição relativa): máximo, mínimo, quartis (1º quartil = Q_1 e 3º = Q_3), decis e percentis
- Medidas de Dispersão (absoluta): variância (σ^2 ou s^2), desvio padrão (σ ou s), amplitude total (A_t) e interquartil (IQR)
- Medidas de Dispersão (relativa): coeficiente de variação (CV)
- Medidas de Forma: coeficiente de curtose e coeficiente de assimetria

Medidas de Resumo (bivariadas ou bidimensionais)

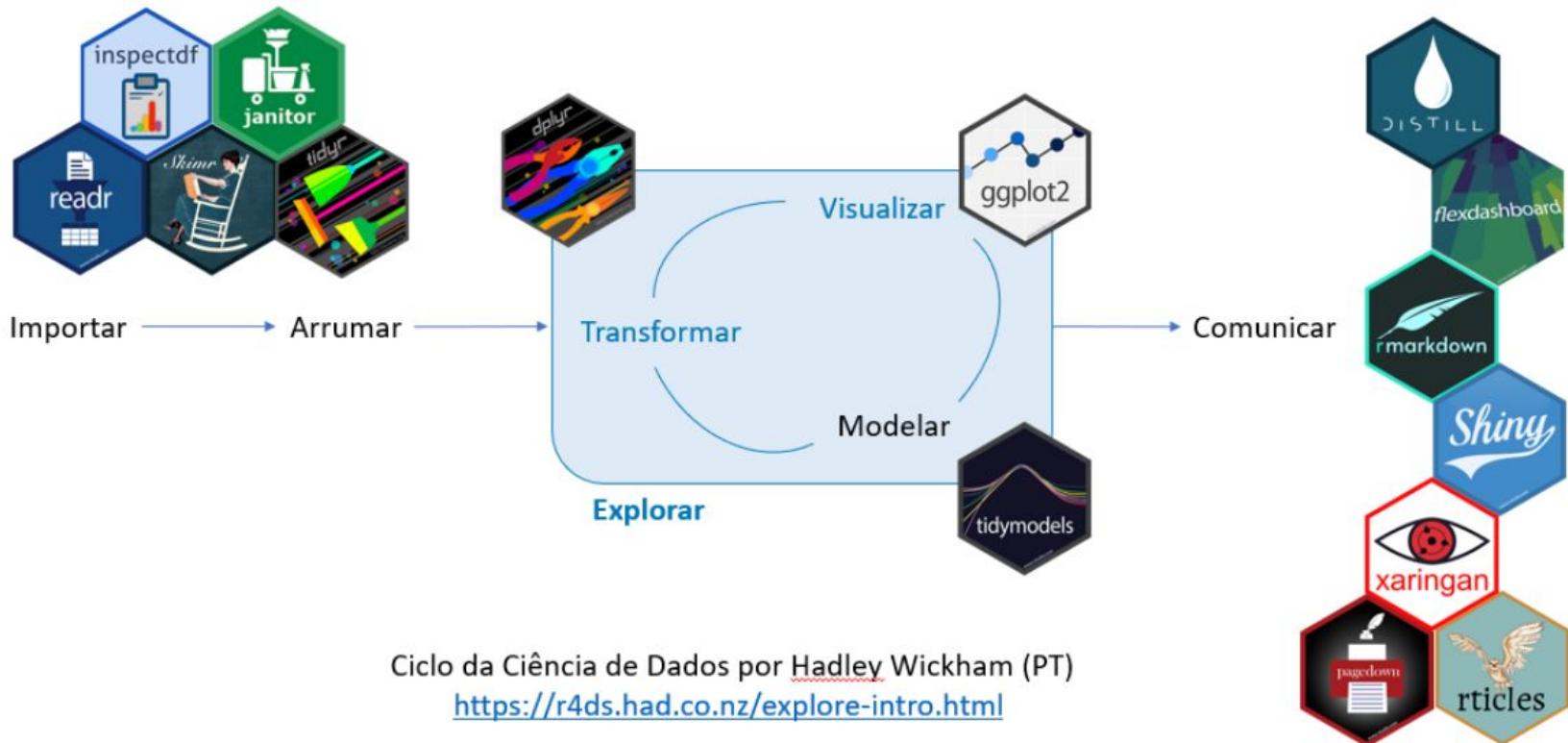
- Medidas de Dependência (associação): covariância (Cov, medida absoluta) e correlação (ρ ou r , medida relativa)



Correlação:

- Pearson
- Spearman
- Kendall
- etc

Extra: EDA (Exploratory Data Analysis)



Extra: EDA (Exploratory Data Analysis)

EDA, do inglês Exploratory Data Analysis, e em português Análise Exploratória de Dados, trata o processo que, tipicamente, inclui desde examinar a estrutura de um dataset, até o entendimento mais profundo sobre o comportamento das variáveis.

Algumas das possibilidades da EDA incluem:

- Encontrar padrões nos dados, não necessariamente conhecidos a priori;
- Identificar se há alguma inconsistência no conjunto de dados analisado;
- Avaliar se as informações disponíveis nos permitem responder as perguntas que temos interesse; e
- Desenvolver um esboço de resposta para as hipóteses levantadas, por vezes até mesmo respondê-las.

Assim, seja considerando em uma linha de exploração descritiva, por meio de volumetrias, estatísticas e visualizações, quanto utilizando uma abordagem mais diagnóstica, por meio da exploração de comportamentos mais complexos, como padrões de associação por exemplo. Em linhas gerais, é na etapa de exploração que **aprendemos** e damos **sentido** aos dados!

Extra: EDA (Exploratory Data Analysis)



Vamos começar falando sobre a biblioteca `skimr()`, em português `skim` significa algo como “tirar a nata”, no caso, tirar a nata dos dados. E é isto que o pacote `skimr` faz, visto nos permitir uma visão macro de toda a base, com uma série de estatísticas organizadas de acordo com a classe das colunas – inclusive com um gráfico de frequência! Muito legal né? :p

```
> skim(mtcars)
Numeric Variables
# A tibble: 11 × 13
   var     type missing complete    n      mean        sd    min `25% quantile` median `75% quantile`      max hist
   <chr>   <chr>   <dbl>   <dbl> <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl> <chr>
1 am numeric     0       32      32  0.406250  0.4989909  0.000  0.00000  0.000  1.00  1.000  1.000  bar
2 carb numeric    0       32      32  2.812500  1.6152000  1.000  2.00000  2.000  4.00  8.000  8.000  bar
3 cyl numeric     0       32      32  6.187500  1.7859216  4.000  4.00000  6.000  8.00  8.000  8.000  bar
4 disp numeric    0       32     3230.721875 123.9386938 71.100 120.82500 196.300 326.00 472.000 472.000  bar
5 drat numeric    0       32      32  3.596563  0.5346787  2.760  3.08000  3.695  3.92  4.930  4.930  bar
6 gear numeric     0       32      32  3.687500  0.7378041  3.000  3.00000  4.000  4.00  5.000  5.000  bar
7 hp numeric       0       32      32  146.687500 68.5628685 52.000  96.50000 123.000 180.00 335.000 335.000  bar
8 mpg numeric      0       32      32  20.090625  6.0269481 10.400  15.42500 19.200  22.80  33.900 33.900  bar
9 qsec numeric     0       32      32  17.848750  1.7869432 14.500  16.89250 17.710  18.90  22.900 22.900  bar
10 vs numeric      0       32      32  0.437500  0.5040161  0.000  0.00000  0.000  1.00  1.000  1.000  bar
11 wt numeric       0       32      32  3.217250  0.9784574 1.513  2.58125  3.325  3.61  5.424  5.424  bar
```

E podemos inclusive combinar esta função com funções como `group_by()`, para mais informações, visite o github oficial do pacote: <https://github.com/ropensci/skimr>

Extra: EDA (Exploratory Data Analysis)



O pacote janitor, algo como faxineiro em inglês, disponibiliza algumas funções para limpar e avaliar bases de dados, com funções como:

- `janitor::remove_empty_cols()` para a remoção de colunas vazias;
- `janitor::get_dupes()` para a identificação de linhas duplicadas; e
- `janitor::round_half_up()` para arredondar resultados numéricos.

As minhas duas funções preferidas deste pacote são:

- `janitor::clean_names()` - que limpa o nome das colunas
- `janitor::tabyl()` - que cria tabelas de frequência rapidamente, além de ter uma série de funções relacionadas, como `janitor::adorn_totals()` e `janitor::adorn_percentages()`.

Aqui o github oficial do pacote: <https://github.com/sfirke/janitor>

Extra: EDA (Exploratory Data Analysis)



O pacote inspectdf faz exatamente o que o nome sugere, nos auxilia a inspecionar dados por meio de uma série de recursos para apresentação de resumos dos dados, como: contabilização de volumetrias e dados faltantes:

- `inspectdf::inspect_types()` - inspeção da classe das variáveis
- `inspectdf::inspect_num()` - medidas de resumo relacionadas às variáveis numéricas
- `inspectdf::inspect_imb()` - visibilidade dos níveis mais frequente de cada uma das variáveis categóricas
- `inspectdf::inspect_na()` - inspeção dos dados faltantes por coluna

Adicionalmente temos a função `inspectdf::show_plot()` que permite a visualização gráfica de algumas das funções acima. Para explorar mais sobre este pacote acesse:

<https://alastairrushworth.github.io/inspectdf/>

Extra: EDA (Exploratory Data Analysis)

No R você irá encontrar dezenas de pacotes com o objetivo de facilitar o seu estudo, particularmente para a exploração descritiva de dados. Bibliotecas como:

visdat



naniar



DataExplorer

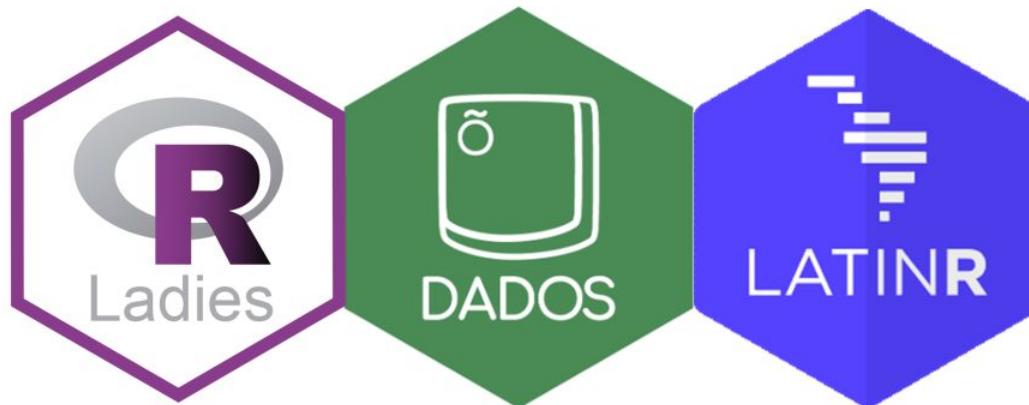


Entre tantas outras. Que tal explorar um pouco estes pacotes?

Extra:

Fontes de informação

Pacote dados



O [pacote dados](#) disponibiliza a tradução de várias bases de dados que são originalmente disponíveis em outros pacotes de R.

O pacote começou a ser desenvolvido em Junho de 2020, e é um irmão do pacote [dados](#). O dados foi desenvolvido para ser usado na tradução do livro [R para Ciéncia de Dados](#) em espanhol, feita voluntariamente pela comunidade Latino-Americana de R.

O pacote está **disponível no CRAN** e pode ser instalado utilizando o seguinte código:

```
install.packages("dados")
```



Tipos de Dados – como funciona no R

A tibble: 27 × 7

filme	classificacao_indicativa	nota_cinema_score	duracao	nota_rotten_tomatoes	data_lancamento	classificacao_livre
<chr>	<ctr>	<ord>	<dbl>	<int>	<date>	<lgl>
Toy Story - Um Mundo de Aventuras	Livre	A	81	100	1995-11-22	TRUE
Vida de Inseto	Livre	A	95	92	1998-11-25	TRUE
Toy Story 2	Livre	A+	92	100	1999-11-24	TRUE
Monstros S. A.	Livre	A+	92	96	2001-11-02	TRUE
Procurando Nemo	Livre	A+	100	99	2003-05-30	TRUE
Os Incríveis	Orientação parental sugerida	A+	115	97	2004-11-05	FALSE
Carros	Livre	A	117	74	2006-06-09	TRUE
Ratatouille	Livre	A	111	96	2007-06-29	TRUE
WALL-E	Livre	A	98	95	2008-06-27	TRUE
Up - Altas Aventuras	Orientação parental sugerida	A+	96	98	2009-05-29	FALSE
Toy Story 3	Livre	A	103	98	2010-06-18	TRUE
Carros 2	Livre	A-	106	40	2011-06-24	TRUE

1-12 of 27 rows

Previous 1 2 3 Next

categórica

nominal

categórica

nominal

categórica

ordinal

numérica

contínua

numérica

discreta

categórica

data

categórica

lógico

character

<chr>

factor

<fct>

ordered

<ord>

double

<dbl>

integer

<int>

date

<date>

logical

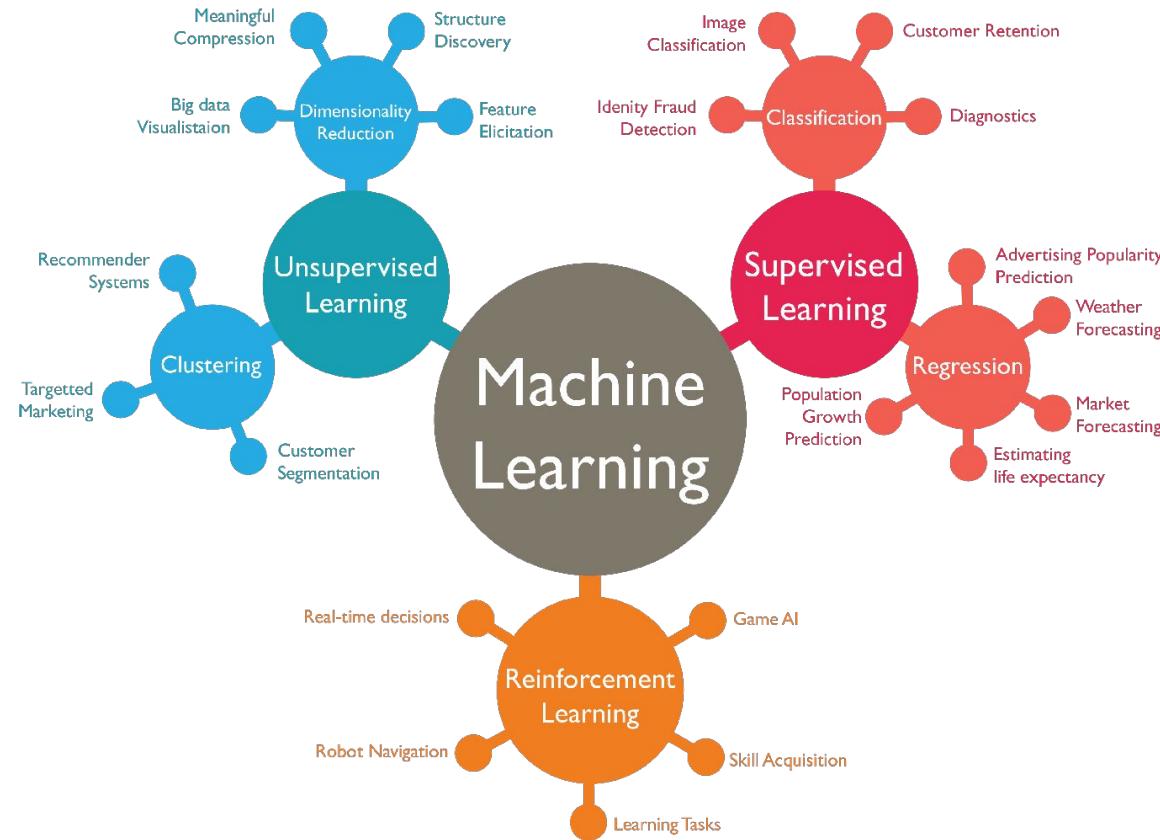
<lgl>



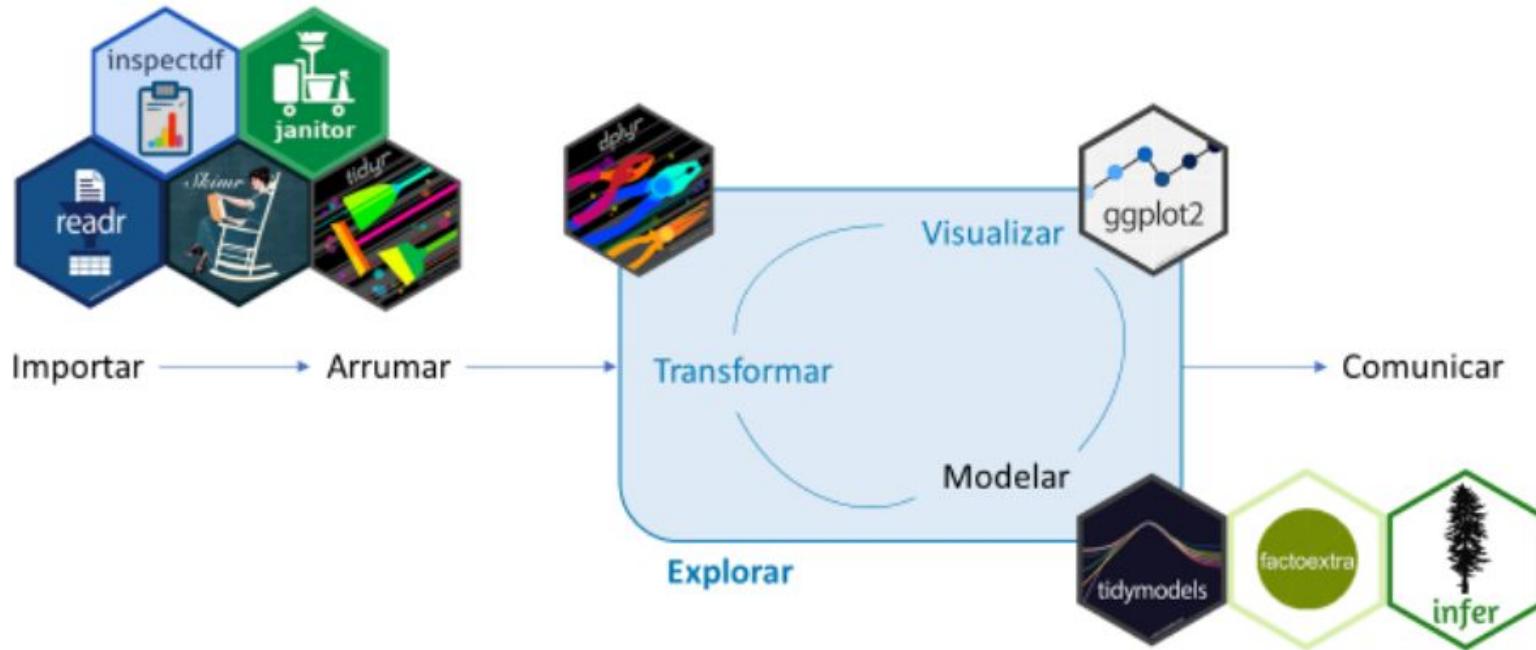
Extra:

Um pouco mais sobre Modelagem de dados ML

Extra: Modelagem de dados



Extra: Modelagem de dados



Ciclo da Ciência de Dados por Hadley Wickham (PT)
<https://r4ds.had.co.nz/explore-intro.html>

Extra: Modelagem de dados

Para explorar sobre como fazer as instalações necessárias e as possibilidades deste pacote:



Site oficial do tidymodels

The screenshot shows the homepage of the tidymodels website. At the top, there's a navigation bar with links for PACKAGES, GET STARTED, LEARN, HELP, CONTRIBUTE, ABOUT, FIND, and a search icon. The main visual is a hexagonal grid composed of smaller hexagons, each containing a different package icon: tidymodels, rsample, parsnip, recipes, TUNE, and yardstick. To the right of the grid, the text "TIDYMODELS" is displayed, followed by a paragraph explaining the framework: "The tidymodels framework is a collection of packages for modeling and machine learning using tidyverse principles." Below this, there's a section titled "Install tidymodels with:" containing the R code: `install.packages("tidymodels")`.

<https://www.tidymodels.org/find/>

Meetup das R-Ladies

The image is a promotional graphic for a Meetup online organized by R-Ladies São Paulo. It features a red background with white text. At the top left, it says "Meetup online". In the center, the title "Tidymodels: Estruturando o processo de Machine Learning" is displayed. To the right, the R-Ladies São Paulo logo is shown. Below the title, there's a photo of a woman, Viviane Sanchez, with the text "apresentação Viviane Sanchez". Further down, there are icons for a clock (representing the date and time) and a location pin (representing the virtual nature of the event), with the text "sábado, 8 de maio 16h00" and "ao vivo no YouTube https://bit.ly/rladessp-youtube". In the bottom right corner, there's a small illustration of a person at a desk with a laptop and charts, and a digital clock showing "1:28:19".

<https://www.youtube.com/live/szi7zfNEMXo?si=7oDeMU-GIS56V7wP>

Extra:

Interoperabilidade

R & Python

Interoperabilidade R & Python: reticulate



No RStudio, podemos trabalhar diretamente com a linguagem de programação Python através do pacote reticulate, permitindo combinar as funcionalidades de ambas as linguagens em um único ambiente de desenvolvimento. Existem duas maneiras de fazer isso:

- Scripts Python (.py): Crie um novo arquivo .py no RStudio e escreva seu código Python como faria em qualquer outro editor
- Utilizar Chunks Python em RMarkdown: Incorpore código Python em documentos RMarkdown usando chunks.

Note que com esta integração você pode inclusive combinar as duas linguagens, trazendo objetos e funções de um ambiente para outro.

The screenshot shows the RStudio interface with the following details:

- Script Editor:** A file named "script_python.py" is open, containing Python code:

```
1 import seaborn as sns
2 dados_py = sns.load_dataset("tips")
3
4 dados_py.head()
5
```
- Console:** The Python console output is shown:

```
> reticulate::repl_python()
>>> import seaborn as sns
>>> dados_py = sns.load_dataset("tips")
>>> dados_py.head()
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

Interoperabilidade R & Python: reticulate



No RStudio, podemos trabalhar diretamente com a linguagem de programação Python através do pacote reticulate, permitindo combinar as funcionalidades de ambas as linguagens em um único ambiente de desenvolvimento. Existem duas maneiras de fazer isso:

- Scripts Python (.py): Crie um novo arquivo .py no RStudio e escreva seu código Python como faria em qualquer outro editor
- Utilizar Chunks Python em RMarkdown: Incorpore código Python em documentos RMarkdown usando chunks.

Note que com esta integração você pode inclusive combinar as duas linguagens, trazendo objetos e funções de um ambiente para outro.

The screenshot shows the RStudio interface with a Python script named "script_python.py" open in the code editor. The code imports seaborn and loads the "tips" dataset. The "Console" tab shows the Python interpreter executing the script and displaying the first few rows of the dataset. The data is as follows:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

Interoperabilidade R & Python: reticulate



No RStudio, podemos trabalhar diretamente com a linguagem de programação Python através do pacote reticulate, permitindo combinar as funcionalidades de ambas as linguagens em um único ambiente de desenvolvimento. Existem duas maneiras de fazer isso:

- Scripts Python (.py): Crie um novo arquivo .py no RStudio e escreva seu código Python como faria em qualquer outro editor
- Utilizar Chunks Python em RMarkdown:
Incorpore código Python em documentos RMarkdown usando chunks.

Note que com esta integração você pode inclusive combinar as duas linguagem, trazendo objetos e funções de um ambiente para outro.

```
{python}
#chunk python
import seaborn as sns
dados_py = sns.load_dataset("tips")

dados_py.head()
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

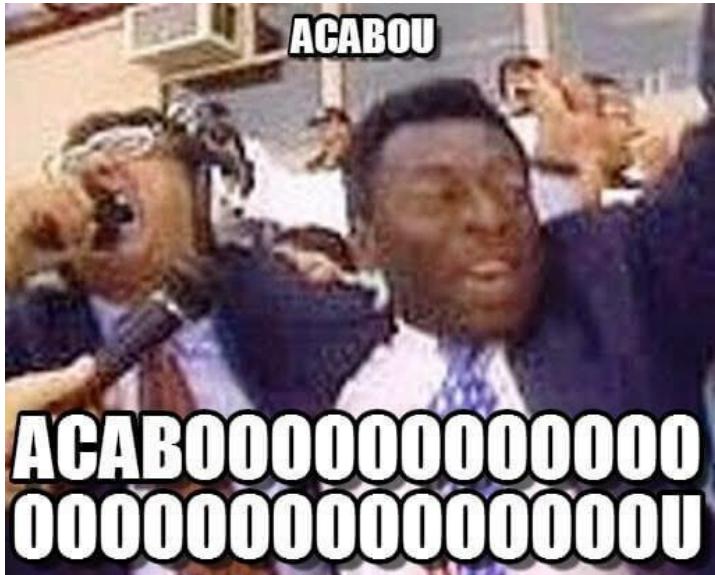
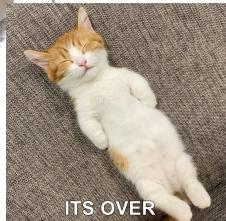

```
{r}
#chunk r
library(reticulate)

dados_r <- py$dados_py %>%
  mutate(across(where(is.factor), ~str_to_lower(.x))) %>%
  mutate(tip_p = round(tip/total_bill, 2)) %>%
  mutate(tip_p_tidy = scales::percent(tip_p, accuracy = 1))

dados_r %>% head()
```

Description: df [6 x 9]

	total_bill	tip	sex	sm...	d...	time	s...	tip...	tip_p_tidy
	<dbl>	<dbl>	<chr>	<chr>	<ch...>	<chr>	<dbl>	<dbl>	<chr>
1	16.99	1...	fem...	no	s...	dinner	2	0.06	6%
2	10.34	1...	male	no	s...	dinner	3	0.16	16%
3	21.01	3...	male	no	s...	dinner	3	0.17	17%
4	23.68	3...	male	no	s...	dinner	2	0.14	14%
5	24.59	3...	fem...	no	s...	dinner	4	0.15	15%
6	25.29	4...	male	no	s...	dinner	4	0.19	19%



R – Considerações Finais

Além das soluções e bibliotecas aqui comentados, é importante lembrar que o R oferece muitas formas de completar uma mesma atividade. Sendo assim, é sempre recomendado que procure opções para otimizar sua rotina.

Segue a principal referência open-source para revisão dos pontos vistos ao longo do curso:

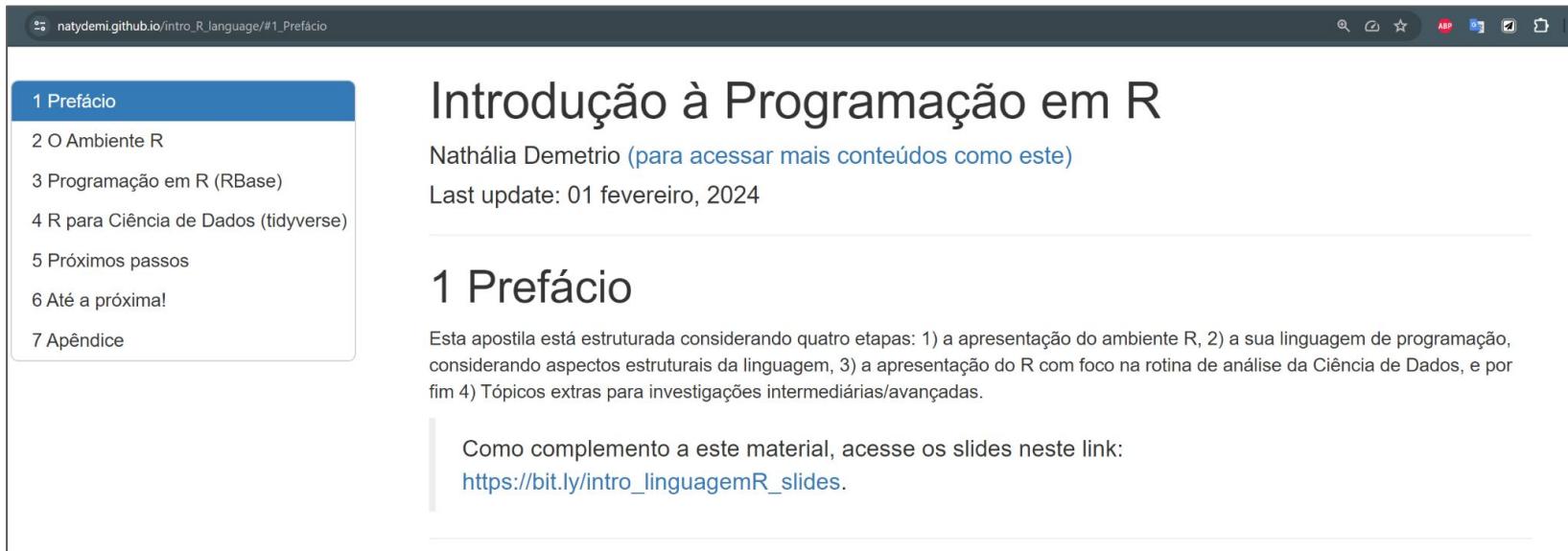
- R for data science: <https://r4ds.had.co.nz/introduction.html>
 - 2^a edição em processo de tradução para o português: <https://cienciadedatos.github.io/pt-r4ds/>

E aqui algumas referências para revisão e aprofundamento de conteúdos

- Curso-R: <https://www.curso-r.com/material/>
- Materiais das R-Ladies São Paulo: <https://rladies-sp.org/>
- Advanced R: <http://adv-r.had.co.nz/>
- Hands-On Programming with R: <https://rstudio-education.github.io/hopr/>

Até a próxima!

Para acessar a versão handout deste curso, acesse:
https://bit.ly/intro_linguagemR_handout



The screenshot shows a web browser window with the URL `natydemi.github.io/intro_R_language/#1_Prefácio` in the address bar. The page title is "Introdução à Programação em R". Below the title, it says "Nathália Demetrio ([para acessar mais conteúdos como este](#))" and "Last update: 01 fevereiro, 2024". On the left, there is a sidebar with a blue header containing the number "1" and the word "Prefácio", followed by a list of numbered sections: 2 O Ambiente R, 3 Programação em R (RBase), 4 R para Ciência de Dados (tidyverse), 5 Próximos passos, 6 Até a próxima!, and 7 Apêndice. The main content area starts with a large section titled "1 Prefácio" which contains a paragraph about the structure of the guide and a link to additional slides.

natydemi.github.io/intro_R_language/#1_Prefácio

1 Prefácio

2 O Ambiente R

3 Programação em R (RBase)

4 R para Ciência de Dados (tidyverse)

5 Próximos passos

6 Até a próxima!

7 Apêndice

Introdução à Programação em R

Nathália Demetrio ([para acessar mais conteúdos como este](#))

Last update: 01 fevereiro, 2024

1 Prefácio

Esta apostila está estruturada considerando quatro etapas: 1) a apresentação do ambiente R, 2) a sua linguagem de programação, considerando aspectos estruturais da linguagem, 3) a apresentação do R com foco na rotina de análise da Ciência de Dados, e por fim 4) Tópicos extras para investigações intermediárias/avançadas.

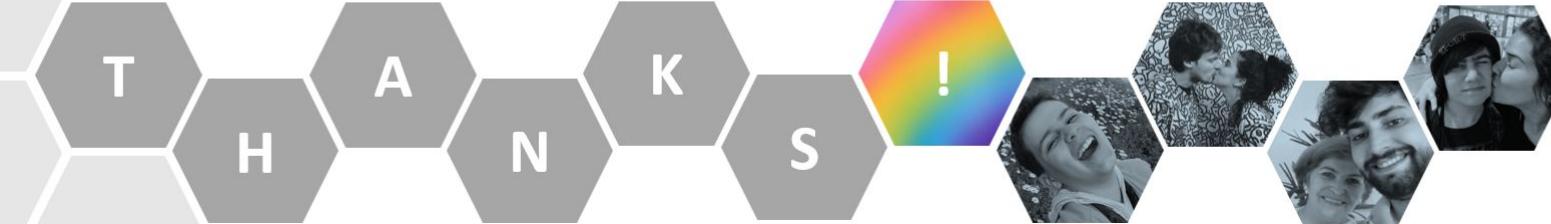
Como complemento a este material, acesse os slides neste link:
https://bit.ly/intro_linguagemR_slides.



Nathália Demetrio (She/Her)

Data & Advanced Analytics Specialist • Translator • Consultant • Advocate • [in]structor

Talks about #rstats, #dataviz, #datascience, #dataliteracy, and #machinelearning



[https://bit.ly/
NataliaDemetrio](https://bit.ly/NataliaDemetrio)

Extra: Big Data

R – R para Big Data

O R trabalha com dados carregados em memória, por meio de processamento sequencial, o que limita tanto o tamanho da base de dados que podemos trabalhar, quanto a velocidade de análise. Alternativas para quando estamos sob alguma destas limitações incluem: trabalhar com amostragens, usar um computador com mais recursos, acessar os dados considerando outras formas de leitura/processamento, ou segmentação de dados. Detalhando estas duas últimas opções temos:

- outras formas de acesso: substituir os data.frames/tibbles pelo data.table, da biblioteca data.table(), uma vez que este trabalha com otimização de processamento e memória;
- processar os dados em disco, ao invés da memória, por meio de bibliotecas como bigmemory, ff ou RevoScaleR;
- dado que o R é por padrão uma linguagem interpretada, utilizar funções como compile() ou enableJIT() para acelerar o processamento de uma análise, principalmente se ela for executada repetidamente; e
- utilizar pacotes que possibilitam paralelismo, como Foreach ou Multicore.

R – R para Big Data

Segmentação:

- aqui podemos trabalhar “manualmente”, carregando sub-conjuntos das base de dados, analisando cada uma delas, e depois combinando os resultados;
- carregar bibliotecas que permitam integração do R com o ecossistema de Big Data, permitindo a distribuição e análise de forma automática. O RHadoop, por exemplo, é um conjunto de pacotes que permite conectar o ambiente R em clusters HDFS e HBASE para a execução de operações. Outra opção é o sparklyr, que possibilita a execução de comandos do R em clusters do Apache Spark. Tanto o RHadoop quanto o sparklyr possibilitem o uso de comandos do pacote dplyr, e a sintaxe do tidyverse(). Assim, as tecnologias se complementam: é possível escrever código de forma mais simples e rápida no R e executá-lo distribuidamente em grandes bases de dados no Hadoop ou Spark, e podemos alavancar os tradicionais sistemas de Big Data, integrando a eles as funcionalidades de análise e machine learning disponíveis no ecossistema do R.