Análise de dados para o Planejamento Territorial

Práticas em R

Flávia da Fonseca Feitosa Beatriz Milz

2025-05-29

Índice

Introdução

Boas vindas!

Este site apresenta o material de apoio para **aulas práticas** das disciplinas **"Análise de dados para o Planejamento Territorial"** e **"Métodos Quantitativos para Pesquisa em PGT"**, oferecidas no segundo quadrimestre de 2025 na Universidade Federal do ABC (UFABC).

O conteúdo das aulas teóricas está disponível no Moodle.

Calendário

Semana	Período	Práticas
1	02/06/2025 - 06/06/2025	Introdução ao R e RStudio
1	02/06/2025 - 06/06/2025	Linguagem R

Sobre este material

Este material contém partes adaptadas de:

- Material criado por Luis Felipe Bortolatto Cunha, que atuou como professor Assistente (estágio docência) em oferecimentos anteriores da disciplina.
- Material do curso Introdução à análise de dados no R, ministrado por Beatriz Milz, Pedro Cavalcanti e Rafael Pereira.

Licença

1 Introdução ao R e RStudio

1.1 Introdução

Ao longo deste curso, os softwares R e RStudio serão usados como uma **ferramenta** para auxiliar na análise de dados para o planejamento territorial.

É importante ressaltar o uso do R e do RStudio não pode ser dissociado do **processo de pesquisa**, que envolve a observação, formulação de hipóteses, coleta de dados e **análise de dados**, sendo este o foco deste curso.

1.2 O que é o R?

R é uma **linguagem de programação** com o foco em estatística, análise e visualização de dados.

Ela é uma linguagem de código aberto, o que significa que qualquer pessoa pode utilizá-la gratuitamente. Além disso, as pessoas com mais experiência na linguagem podem contribuir com o desenvolvimento de novas funcionalidades e pacotes.

Caso queira saber mais sobre a linguagem R, acesse o site oficial (R-Project).

Ao instalar o R, você terá acesso a um programa chamado "R Console" que permite escrever e executar códigos em R:

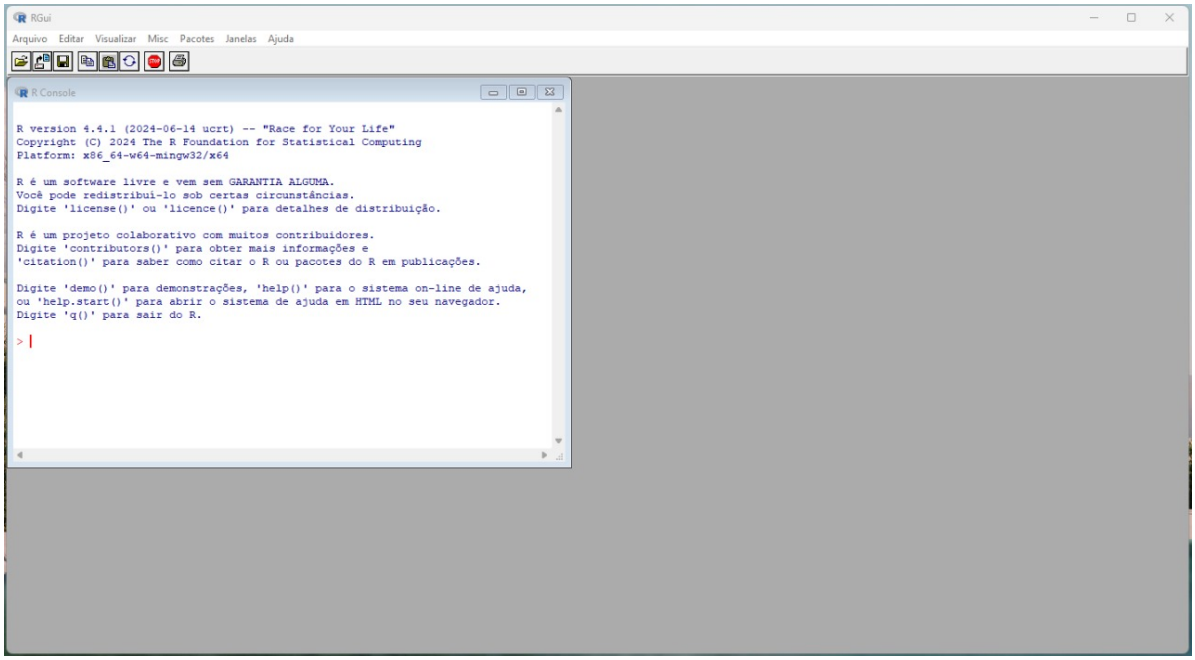


Figura 1.1: Captura de tela do R Console no Windows

Porém o `R Console` não é muito amigável para escrever códigos complexos ou realizar análises de dados. Por isso, é recomendado utilizar um ambiente de desenvolvimento integrado (IDE). A IDE mais utilizada por pessoas que programam em R é o RStudio.

1.3 O que é o RStudio?

O RStudio é um IDE focada em programação em R, e é desenvolvido pela Posit. Ele facilita a escrita de códigos, execução de scripts, e visualização dos resultados.

Existem algumas versões do RStudio. Neste curso, utilizaremos o RStudio Desktop, pois é a versão de código aberto (portanto é gratuita). Daqui em diante, sempre que mencionarmos "RStudio", estaremos nos referindo ao RStudio Desktop.

1.4 Instalando o R e o RStudio

Durante as aulas, utilizaremos os computadores do laboratório da universidade. Porém, caso você tenha acesso a um computador pessoal, recomendamos que instale o R e o RStudio nele, para praticar fora do período das aulas.

1.4.1 Instalação do R

Para instalar o R, acesse o site CRAN e escolha o link de download de acordo com o seu sistema operacional:

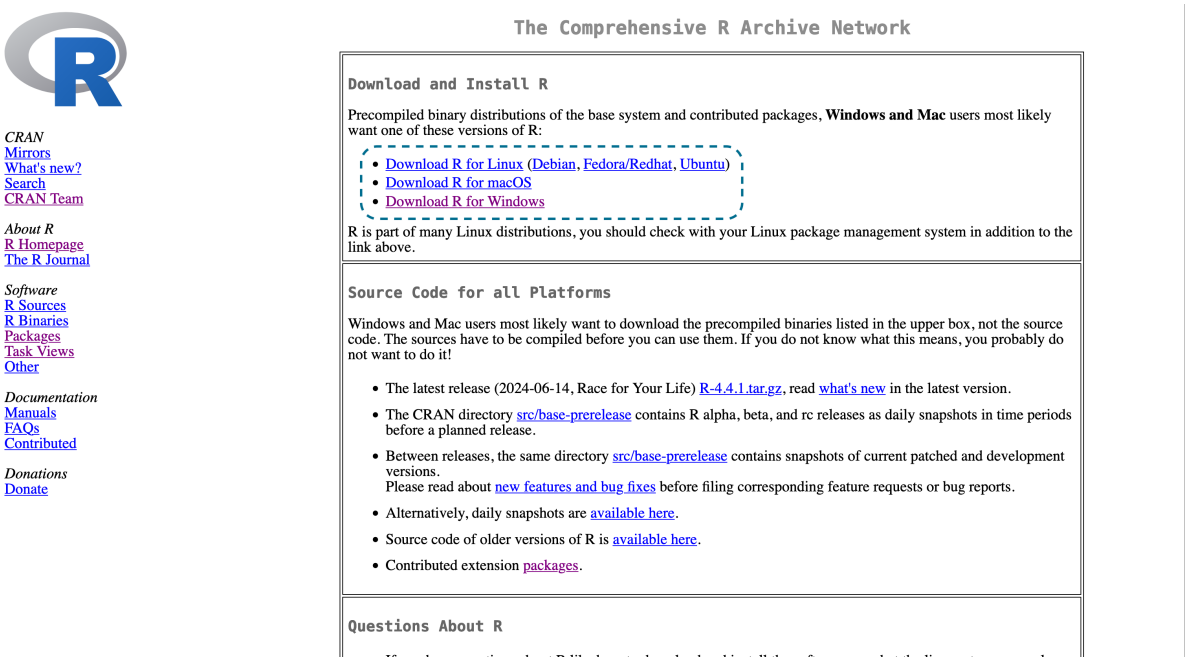


Figura 1.2: Captura de tela do site CRAN

Instale o R utilizando o instalador baixado.

1.4.2 Instalação do RStudio

Após instalar o R, acesse o site RStudio Desktop e escolha o link de download de acordo com o seu sistema operacional:

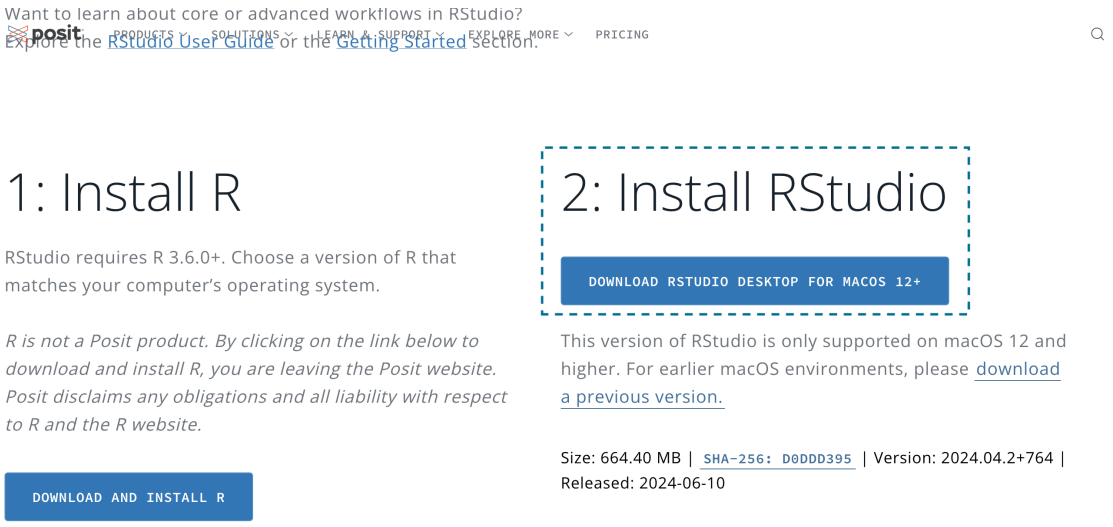


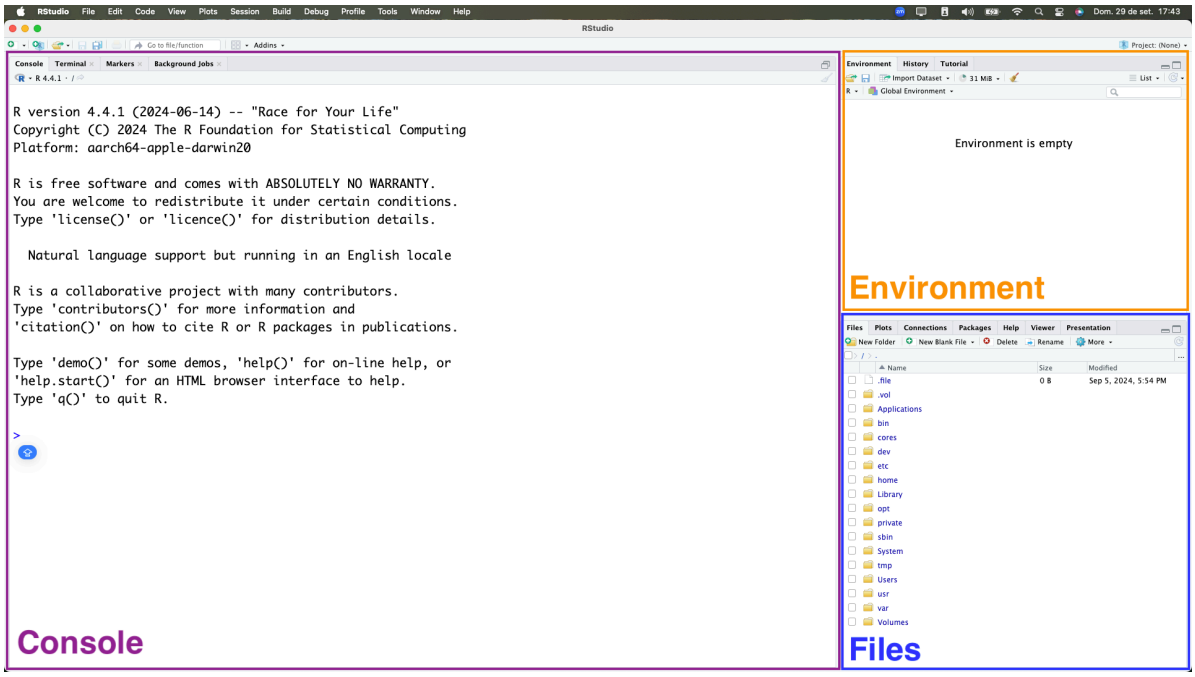
Figura 1.3: Captura de tela do site RStudio Desktop

Instale o RStudio utilizando o instalador baixado.

> **Dica**
>
> Caso o seu computador tenha limitações para instalação de programas, você pode utilizar o Posit Cloud, uma versão online do RStudio. Entretanto, a versão gratuita do Posit Cloud tem algumas limitações, como limite de tempo de uso (25 horas por mês) e de memória RAM (1 GB).
> O vídeo abaixo apresenta um tutorial sobre como utilizar o Posit Cloud:

1.5 Conhecendo o RStudio

Ao abrir o RStudio, veremos a seguinte tela:



Aos poucos, conheceremos os painéis e funcionalidades do RStudio. Neste momento, podemos destacar os três painéis que são apresentados:

- **Console**: painel onde os códigos são executados. É similar ao "R Console", citado anteriormente.
- **Environment**: painel onde as variáveis e dados carregados ficam listados.
- **Files**: painel onde podemos navegar por arquivos no computador. A página inicial é o diretório de trabalho: esse conceito será explicado mais adiante.

1.6 Scripts

No RStudio, podemos escrever e executar códigos no Console, porém os códigos são perdidos quando fechamos o programa. Para salvar os códigos e reutilizá-los posteriormente, utilizamos scripts.

Os scripts são arquivos de texto onde podemos escrever códigos R e salvá-los para utilizar posteriormente. É recomendado que qualquer código que você deseje reutilizar ou que seja importante para a análise que você fizer seja salvo em um script.

Existem algumas formas de criar um novo script:

- No menu superior, clicando em `File > New File > R Script`.

- Utilizando o atalho `Ctrl + Shift + N` (Windows) ou `Cmd + Shift + N` (Mac).
- Clicando no ícone de um arquivo com um sinal de `+` no canto superior esquerdo do RStudio e selecionando `R Script`:

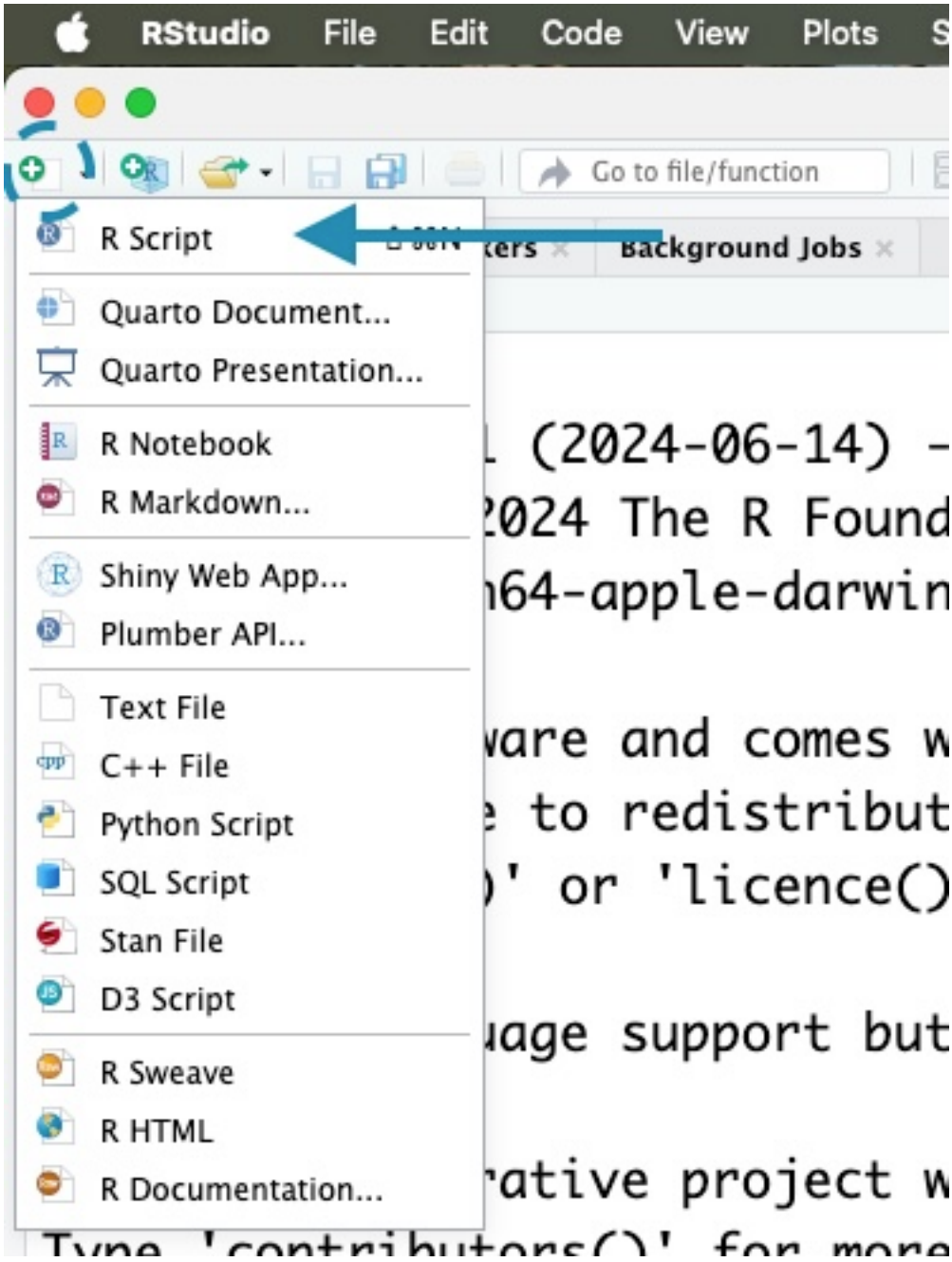


Figura 1.4: Captura de tela do RStudio: Opção para criar novo Script

Após abrir um script, o RStudio exibirá 4 paineis:

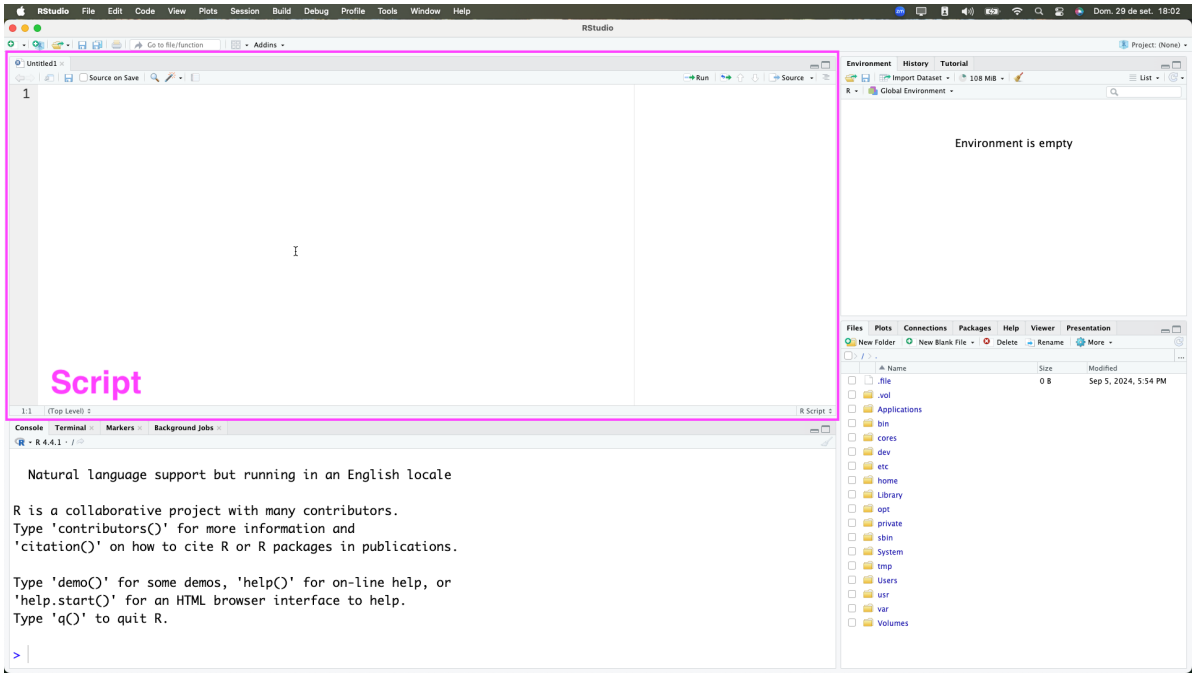


Figura 1.5: Captura de tela do RStudio

> **Dica**
>
> O script é um arquivo salvo no nosso computador. Lembre-se de salvar os scripts com frequência para evitar perder o nosso trabalho.
> Podemos salvar um script de algumas formas, como:
>
> - Clicando em `File > Save` no menu superior.
> - Clicando no ícone do disquete ( ).
> - Utilizando o atalho `Ctrl + S` (Windows) ou `Cmd + S` (Mac).

1.6.1 Como executar os códigos?

Podemos escrever e executar códigos no Console ou em um script.

No Console, escrevemos o código diretamente e pressionamos `Enter` para executá-lo.

Em um Script, escrevemos o código e podemos executá-lo de algumas formas:

- Selecionando o trecho de código que queremos executar e clicando no botão `Run` do RStudio, ou utilizando o atalho `Ctrl + Enter` (Windows) ou `Cmd + Enter` (Mac).
- Clicando no trecho que queremos executar e clicando no botão `Run` do RStudio, ou utilizando o atalho `Ctrl + Enter` (Windows) ou `Cmd + Enter` (Mac).

1.6.2 Comentários

Comentários são textos que não são executados pelo R. Podemos usar comentários para explicar o que um bloco de código faz, para anotar ideias e explicar escolhas feitas, ou para desativar temporariamente um trecho de código.

No R, todo texto em uma linha após um hashtag (`#`) é um comentário. Por exemplo:

```
# Este é um comentário
```

1.7 Funções

Agora que já sabemos onde escrever nossos códigos em R (no Console ou em um script), é importante entender o conceito de funções.

Uma função é tipo de objeto no R, que quando executado, executa um bloco de código específico. As funções são úteis para evitar repetição de códigos e organizar o nosso trabalho.

No R, existem muitas funções prontas que podemos utilizar. Por exemplo, a função `Sys.Date()` retorna a data atual do sistema:

```
# Consutar a data atual do sistema (computador)
Sys.Date()
```

```
[1] "2025-05-29"
```

Para utilizar uma função, escrevemos o nome dela seguido de parênteses. Dentro dos parênteses, podemos colocar dados e informações úteis para a função executar a tarefa desejada, e são chamados de **argumentos**.

Por exemplo, a função `sqrt()` calcula a raiz quadrada de um número. Para utilizá-la, podemos escrever `sqrt()` e informar esse número entre parênteses:

```
# Calcular a raiz quadrada de 25
sqrt(25)
```

```
[1] 5
```

> **Dica**
>
> Ao adquirir experiência com o R, podemos criar nossas próprias funções. Isso é útil para automatizar tarefas repetitivas e para organizar o código.

1.8 Pacotes

Pacotes do R são coleções de funções, dados e documentação que estendem a funcionalidade básica da linguagem.

O CRAN (*Comprehensive R Archive Network*) é o repositório oficial de pacotes do R. Ele contém milhares de pacotes que podem ser instalados e utilizados gratuitamente. Em maio de 2025, o CRAN continha mais de 22.000 pacotes disponíveis.

Para instalar um pacote, utilizamos a função `install.packages()` e informando o nome do pacote como texto entre aspas. Por exemplo, para instalar o pacote `{tidyverse}`, utilizamos o seguinte comando:

```
# Instalar o pacote tidyverse
install.packages("tidyverse")
```

Apenas precisamos instalar um pacote uma vez.

Depois de instalado, podemos carregá-lo com a função `library()`, para que as funções do pacote fiquem disponíveis para uso:

```
# Carregar o pacote tidyverse
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.2     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

Precisamos carregar o pacote sempre que abrirmos um novo script, ou quando reiniciamos o RStudio. Uma pratica frequente é carregar os principais pacotes necessários no início do script.

> **Cuidado**
>
> Uma outra forma de acessar uma função é utilizando o operador `::`. Por exemplo, para acessar a função `read_csv()` do pacote `{readr}`, podemos escrever `readr::read_csv()`. Essa sintaxe é menos frequente, porém útil para evitar problemas de conflito de funções com o mesmo nome em pacotes diferentes. Esse problema acontece mais frequentemente quando carregamos muitos pacotes em um mesmo script.
> Por exemplo: o pacote `{dplyr}` apresenta uma função `filter()`, e o pacote `{stats}` também apresenta uma função `filter()`. Se não usarmos o operador `::`, a função utilizada será a do pacote que foi carregado por último. Usando o operador `::`, podemos escolher qual função queremos utilizar.

1.9 Documentação

As funções e pacotes do R apresentam textos com explicações e exemplos de uso, chamados de **documentação**.

As documentações podem ser acessadas online, ou diretamente no RStudio.

1.9.1 Documentação no RStudio

No RStudio, podemos acessar a documentação de uma função ou pacote das seguintes formas:

- Para buscar informações sobre funções de pacotes já carregados (com `library`), podemos utilizar a função `help()`, informando o nome da função que queremos buscar como argumento (ex: `help(mean)`), ou utilizar o operador `?`, seguido do nome da função (ex: `?mean`).

```
# Abrir a documentação da função mean()
help(mean)
?mean
```

- Para fazer uma por funções presentes em todos os pacotes instalados no computador, podemos utilizar o operador `??`, seguido pelo termo que queremos buscar (ex: `??mean`). Essa é uma busca mais ampla, que procura pelo termo no nome e na descrição das funções.

```
# Buscar por funções que contenham o termo "mean"
??mean
```

- Podemos utilizar o painel Help para buscar informações sobre funções e pacotes:

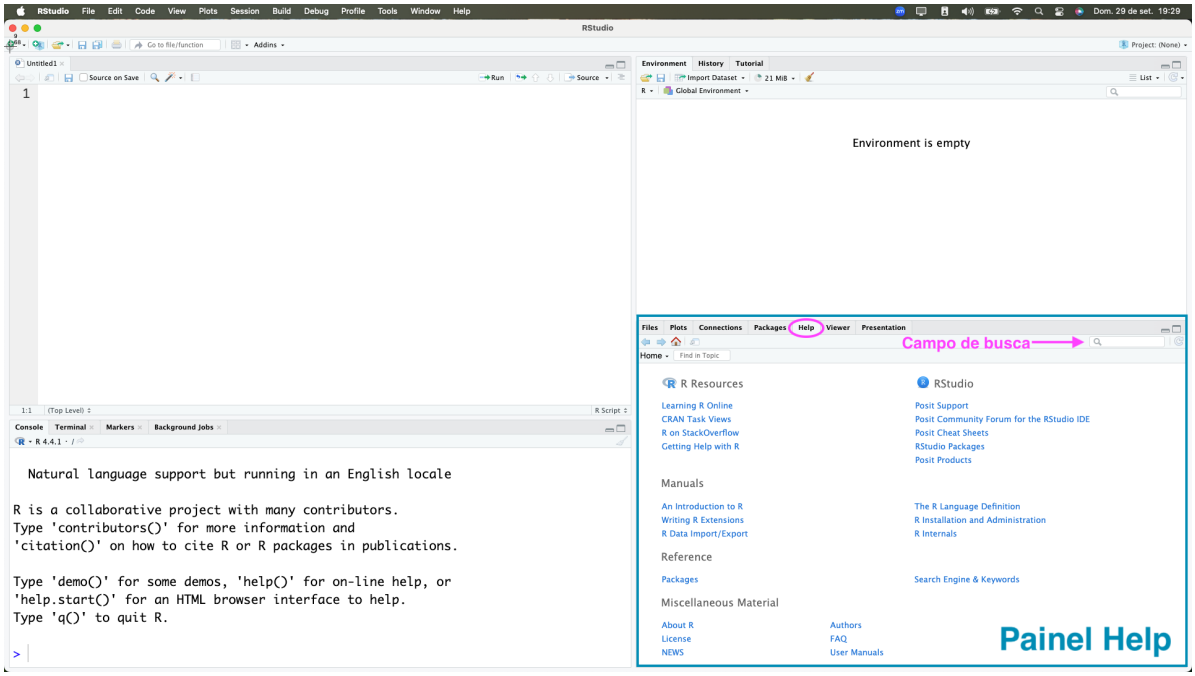


Figura 1.6: Captura de tela do RStudio: Painel Help

Além disso, a maioria dos pacotes vem com textos explicativos sobre como usá-los, chamadas de *vignettes*. Elas estão disponíveis online, mas também podem ser acessadas diretamente no RStudio.

Para acessar no RStudio, podemos usar a função `browseVignettes()` para listar as *vignettes* disponíveis para um pacote específico. A lista será apresentada em uma janela do navegador (ex: Google Chrome, Firefox, Safari, etc):

```
# Listar as vignettes do pacote dplyr
browseVignettes("dplyr")
```

Figura 1.7: Captura de tela: Lista de Vignettes do pacote dplyr

1.9.2 Documentação online

Como citado anteriormente, é possível acessar a documentação dos pacotes diretamente no RStudio e também online. No geral, o conteúdo disponível online é igual ao disponível no RStudio, mas pode ser mais fácil de buscar e navegar.

Uma forma de acessar a documentação online é fazendo uma busca no Google com os termos “`R documentation {nome da função}`”. Por exemplo: “`R documentation mean()`”.

Alguns pacotes apresentam também sites próprios com documentações e *vignettes*.

Por exemplo, o pacote `{dplyr}` (que usaremos no curso) tem um site próprio onde conseguimos acessar a documentação. Os pacotes do tidyverse apresentam sites similares, com páginas com os seguintes conteúdos:

- Em *Get started* encontramos uma introdução ao pacote, e exemplos de uso para quem quer aprender a usá-lo.
- Em *Reference*, encontramos a lista de funções disponíveis no pacote, e podemos acessar a documentação de cada uma delas:

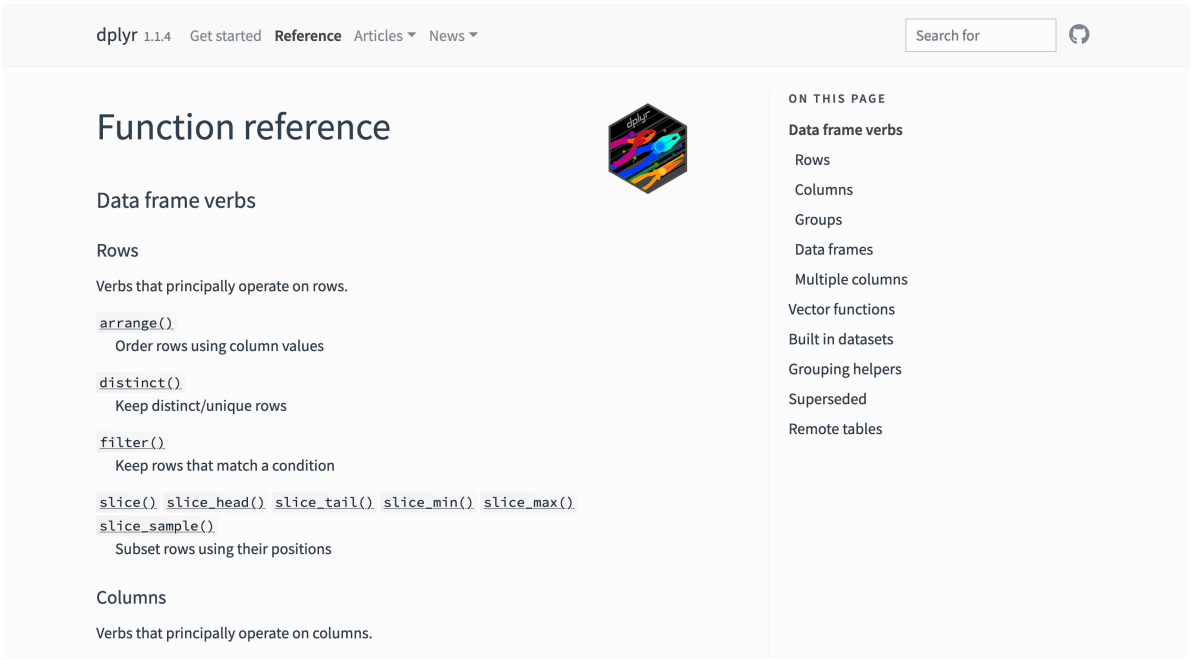


Figura 1.8: Captura de tela: Site do pacote dplyr - Reference

- Em *Articles* podemos acessar as *vignettes*:

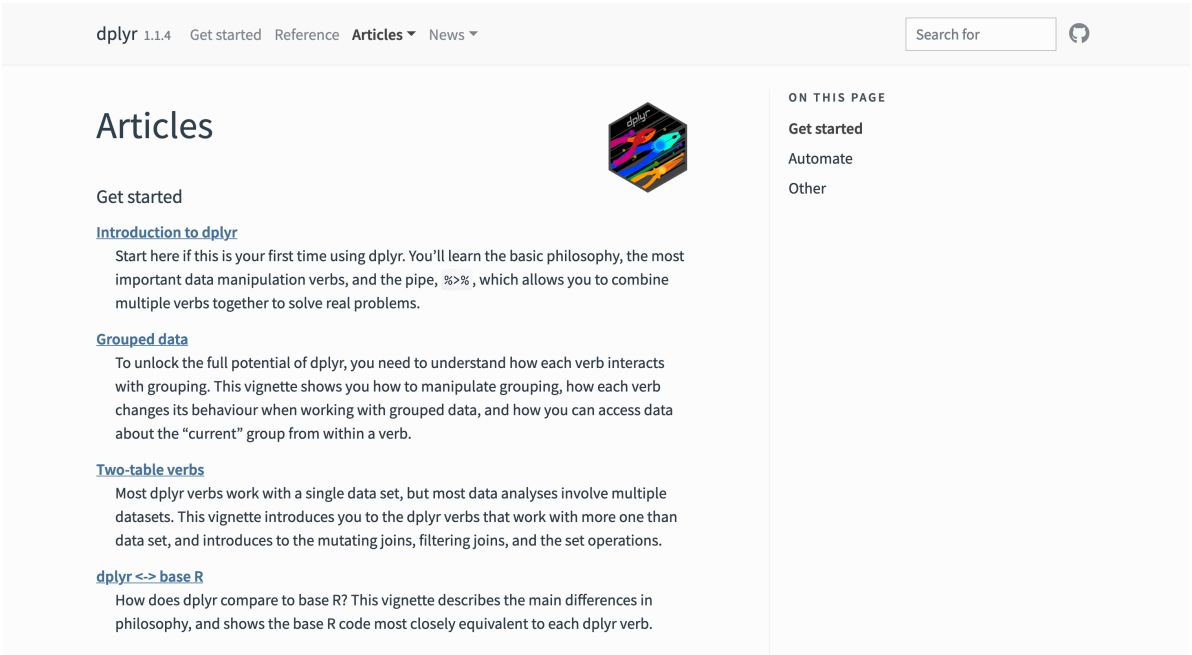


Figura 1.9: Captura de tela: Site do pacote dplyr - Vignettes

1.9.3 Cheatsheets

As *cheatsheets* (ou folhas de cola) são documentos resumidos com informações sobre funções e pacotes. Elas são úteis para consulta rápida.

A Posit (empresa que desenvolve o RStudio) disponibiliza *cheatsheets* para diversos pacotes e tópicos. Elas podem ser acessadas no site Posit Cheatsheets.

A lista a seguir apresenta algumas *cheatsheets* sobre temas que serão abordados ao longo do curso:

- RStudio IDE
- Importação de dados com o tidyverse
- Transformação de dados com dplyr
- Visualização de dados com ggplot2
- Arrumando dados com tidyr

1.10 Materiais complementares

- Materiais do curso Introdução à análise de dados no R:
  - Instalação
  - Conhecendo o R e o RStudio
- Livro R para Ciência de Dados 2ed:
  - Introdução > Pré-requisitos em diante
  - Fluxo de Trabalho: obtendo ajuda

2 Linguagem R

Existem muitos conceitos básicos que são fundamentais para quem está começando a programar em R.

Nesta aula, vamos abordar alguns conceitos considerados mais importantes para as próximas aulas.

2.1 Operações matemáticas

O R permite realizar operações matemáticas básicas, como soma, subtração, multiplicação, divisão, potenciação, entre outras.

```
1 + 1 # Soma
```

```
[1] 2
```

```
1 - 1 # Subtração
```

```
[1] 0
```

```
2 * 3 # Multiplicação
```

```
[1] 6
```

```
10 / 2 # Divisão
```

```
[1] 5
```

```
2 ^ 3 # Potenciação
```

```
[1] 8
```

A ordem matemática das operações também vale no R. Por exemplo, a expressão `2 + 3 * 4` será calculada como `2 + (3 * 4)`:

```
2 + 3 * 4
```

```
[1] 14
```

2.2 Objetos

No R, um objeto é uma estrutura de dados que armazena valores: podemos armazenar um valor único, um conjunto de valores, uma base de dados, entre outros.

É muito útil armazenar valores em objetos, pois podemos reutilizá-los em diferentes partes do código, sem precisar digitar o valor novamente.

2.2.1 Objetos existentes no R

Existem alguns objetos já criados no R, como por exemplo o objeto `letters`, que armazena as letras do alfabeto:

```
pi
```

```
[1] 3.141593
```

```
letters
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

> **Aviso**
>
> O R é *case-sensitive*, ou seja, ele diferencia letras maiúsculas de minúsculas. Portanto, `nome` é diferente de `Nome`.
> Por exemplo, o objeto `pi` armazena o valor de (com um número limitado de casas decimais). O nome do objeto é escrito em minúsculas:
>
> ```
> pi
> ```
>
> ```
> [1] 3.141593
> ```

Se tentarmos acessar o objeto com o nome em maiúsculas, o R irá retornar um erro, pois esse objeto não existe:

```
Pi
```

```
Error: object 'Pi' not found
```

2.2.2 Criando um objeto

Para criar um objeto, precisamos definir um nome, e atribuir um valor à este nome. Para isso, usamos o operador de atribuição: `<-`. Um atalho para esse operador é o `Ctrl` + `-` no Windows, ou `Option` + `-` no Mac .

No exemplo a seguir, criamos um objeto chamado `nome_do_curso` e atribuímos a ele o texto `"Universidade Federal do ABC"`:

```
nome_do_curso <- "Universidade Federal do ABC"
```

Podemos acessar o valor armazenado em um objeto digitando o nome do objeto:

```
nome_do_curso
```

```
[1] "Universidade Federal do ABC"
```

O objeto apenas será alterado se utilizarmos o operador de atribuição novamente. Por exemplo, a função `tolower()` transforma todas as letras de um texto em minúsculas:

```
tolower(nome_do_curso)
```

```
[1] "universidade federal do abc"
```

Mas como não utilizamos a atribuição, o objeto `nome_do_curso` não foi alterado:

```
nome_do_curso
```

```
[1] "Universidade Federal do ABC"
```

Para alterar o objeto, precisamos atribuir o resultado da função `tolower()` ao objeto `nome_do_curso`:

```
nome_do_curso <- tolower(nome_do_curso)
```

Agora, o objeto `nome_do_curso` foi alterado:

```
nome_do_curso
```

```
[1] "universidade federal do abc"
```

Portanto, cuidado: ao criar um objeto com nome igual à outro objeto existente, o objeto anterior será substituído pelo novo objeto.

2.2.3 Consultando os objetos criados

Para consultar os objetos criados, podemos usar a função `ls()` (*list objects*), que lista todos os objetos criados no *Painel Environment*:

```
ls()
```

```
[1] "nome_do_curso"   "pandoc_dir"      "quarto_bin_path"
```

Outra forma é consultar o *Painel Environment* no RStudio, que exibe todos os objetos criados, e permite acessar o valor de cada objeto:

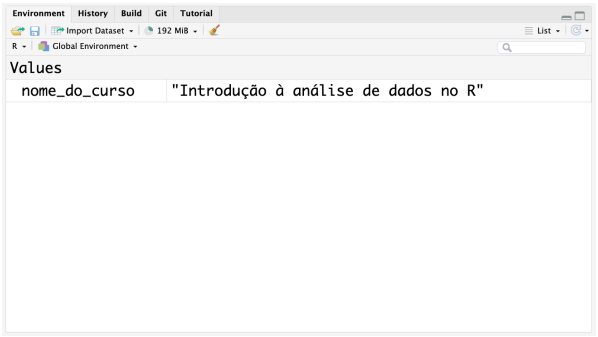


Figura 2.1: Captura de tela do RStudio: Painel Environment

2.2.4 Removendo objetos

Caso queira remover um objeto, podemos usar a função `rm()` (*remove objects*). Por exemplo, para remover o objeto `nome_do_curso`:

```
rm(nome_do_curso)
```

Podemos consultar novamente os objetos existentes e verificar se o objeto foi removido:

```
ls()
```

```
[1] "nome_do_curso"   "pandoc_dir"      "quarto_bin_path"
```

Caso queira remover todos os objetos carregados, podemos usar a função `rm(list = ls())`.

```
# Remover todos os objetos do Global Environment
rm(list = ls())                                                    ①
```

2.2.5 Nomeando objetos

Existem regras e boas práticas para nomear objetos no R. As regras são obrigatórias: se não seguirmos, o código irá gerar um erro. As boas práticas são recomendações, com o objetivo de evitar erros futuros, e facilitar a leitura do código.

> **Dica**
>
> Recomendamos seguir essas recomendações não apenas para o nome dos objetos, mas também para nome de pastas e arquivos, nome de colunas, entre outros.

2.2.5.1 Regras para nomear objetos no R

- O nome não deve começar com um número. Ex: `1objeto` não é um nome válido.

```
1objeto <- "exemplo"
```

```
Error in parse(text = input): <text>:1:2: unexpected symbol
1: 1objeto
     ^
```

- O nome não deve começar com underline (_). Ex: `_objeto` não é um nome válido.

```
_objeto <- "exemplo"
```

```
Error in parse(text = input): <text>:1:2: unexpected symbol
1: _objeto
     ^
```

- O nome do objeto não deve conter traços (-), pois o R interpreta o traço como um operador de subtração. Ex: `meu-objeto` não é um nome válido.

```
objeto-1 <- "exemplo"
```

```
Error: object 'objeto' not found
```

2.2.5.2 Boas práticas para nomear objetos no R

- O nome não deve começar com um ponto. Isso não gerará um erro, porém é contra-indicado pois essa sintaxe é utilizada para nomear objetos ocultos no R (portanto, não aparecerá no *Painel Environment*), e isso pode dificultar o acesso posteriormente. Ex: `.objeto` não deve ser usado.
- Não utilizar acentos, cedilhas, ou outros caracteres especiais. Isso pode gerar problemas de compatibilidade com outros sistemas (chamamos de *encoding*). Ex: `aviões` não é um nome recomendado.
- Não utilizar espaços, pois dificulta o acesso ao objeto posteriormente: precisaremos utilizar a crase em volta do nome do objeto para acessá-lo.
- Existem diferentes estilos para nomear objetos[1], como *snake_case*, onde todas as letras devem ser minúsculas, e as palavras separadas por underline. O importante é escolher um estilo e manter a consistência.

[1] A documentação da função `snakecase::to_any_case()` fornece uma lista de estilos de nomenclatura disponíveis no pacote `snakecase`.

2.3 Tipos de dados

Existem diferentes tipos de dados que podemos armazenar em objetos no R, como números, textos, lógicos, fatores, datas, entre outros.

É muito importante identificar o tipo de dado que estamos trabalhando, pois cada tipo de dado permite usar funções específicas.

A função `class()` permite verificar a classe de um objeto. Por exemplo, podemos verificar a classe do objeto `pi`:

```
class(pi)
```

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

2.3.1 Numérico

Os objetos numéricos no R podem ser de dois tipos:

- inteiros (*integer*): armazena apenas números inteiros, ou seja, sem casas decimais. Ex: 1, 2, 3, 4, 5.
- numéricos (*numeric* ou *double*): armazena números que podem conter casas decimais. Ex: 1, 2, **3.14**, 4, 5.

Na maioria dos casos, utilizamos o tipo *numeric* para armazenar números, pois ele é mais flexível.

Por exemplo, para criar um objeto que armazena o número 42:

```
exemplo_numero <- 42
class(exemplo_numero)
```

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

Para criar um objeto que armazena um valor usado para converter dólar em reais (R$), podemos usar o tipo `numeric`:

```
conversao_dolar <- 5.45
class(conversao_dolar)
```

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

2.3.2 Lógico (booleano)

O tipo lógico (*logical*) armazena valores booleanos: `TRUE` (verdadeiro) ou `FALSE` (falso). Os valores binários podem ser convertidos para números, onde `TRUE` será 1, e `FALSE` será 0.

Por exemplo, para criar objetos com os valores lógicos `TRUE` e `FALSE`:

```
exemplo_logico_verdadeiro <- TRUE
class(exemplo_logico_verdadeiro)
```

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

```
exemplo_logico_falso <- FALSE
class(exemplo_logico_falso)
```

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

2.3.3 Character (texto)

O tipo texto (*character*, ou também conhecido como *string*) armazena textos. Para criar um objeto com um texto, precisamos colocar o texto entre aspas:

```
nome_do_curso <- "Universidade Federal do ABC"
class(nome_do_curso)
```

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

2.3.4 Fator (categórico)

O tipo fator (*factor*) armazena variáveis categóricas, ou seja, variáveis que possuem um número limitado de categorias. Os valores possíveis de categoria são chamados de *levels*. Os fatores podem ser ordenados ou não ordenados.

Os fatores são muito úteis para a visualização de dados e para alguns modelos usados em análise de dados.

Neste momento, não vamos aprofundar no conceito de fatores: falaremos mais sobre eles nas próximas aulas.

2.3.5 Datas

O tipo data (*Date*) armazena datas no formato `aaaa-mm-dd` (ano-mês-dia).

Para criar um objeto com uma data, podemos usar a função `as.Date()`, que converte um texto para o tipo *Date*. Por exemplo, para criar um objeto com a data de início do curso:

```
data_inicio_curso <- as.Date("2025-06-02")
class(data_inicio_curso)
```

```
[1] "Date"
```

A função `Sys.Date()` retorna a data atual do sistema:

```
data_sistema <- Sys.Date()
class(data_sistema)
```

```
[1] "Date"
```

Podemos fazer operações com datas:

```
# Diferença entre duas datas
data_inicio_curso - data_sistema
```

```
Time difference of 4 days
```

```
# Somar um dia à data atual
data_sistema + 1
```

```
[1] "2025-05-30"
```

```
# Somar um dia à data de início do curso
as.Date("2025-06-02") + 1
```

```
[1] "2025-06-03"
```

2.4 Conversões entre tipos de dados

Existem várias funções que podemos usar para transformar variáveis de um tipo para outro. Essas funções tem começam com `as.` seguido pelo tipo de dado que queremos que seja convertido. Por exemplo:

- `as.character()`: converte valores para texto
- `as.numeric()`: converte valores para número
- `as.logical()`: converte valores para lógico
- `as.factor()`: converte valores para fator
- `as.Date()`: converte valores para data

```
#  Converter número para texto
as.character(2024)
```

```
[1] "2024"
```

```
# Converter lógico para número
as.numeric(TRUE)
```

```
[1] 1
```

```
# Converter texto para data
as.Date("2024-10-01")
```

```
[1] "2024-10-01"
```

```
# Converter texto para lógico
as.logical("TRUE")
```

```
[1] TRUE
```

Porém, nem toda conversão fará sentido. Por exemplo, podemos converter um número para texto, porém nem toda conversão de texto para número funcionará como esperado:

```
# Converter para número com a função as.numeric()
as.numeric("2025")
```

```
[1] 2025
```

```r
as.numeric("Aprendendo R")
```

```
Warning: NAs introduced by coercion
```

```
[1] NA
```

Quando a conversão não é possível, o R irá retornar um valor `NA` (*Not Available*), que indica um valor faltante. Falaremos mais sobre valores faltantes na próxima seção.

2.5 Valores faltantes (`NA`)

Valores faltantes, conhecidos também como ***missing values***, são valores que não estão disponíveis, ou que não foram informados. No R, esses valores são representados pelo valor `NA` (*Not Available*).

Podemos testar se um valor é `NA` usando a função `is.na()`: essa função retornará `TRUE` se o valor for `NA`, e `FALSE` caso contrário.

```r
is.na(NA)
```

```
[1] TRUE
```

Algo importante é que o R não consegue fazer operações matemáticas com valores `NA`. Por exemplo, se tentarmos realizar qualquer operação matemática com `NA`, o resultado será `NA`:

```r
NA + 1
```

```
[1] NA
```

```r
NA + NA
```

```
[1] NA
```

```r
NA * 2
```

```
[1] NA
```

É importante identificar os valores NA em nossos dados, pois eles podem afetar o resultado de nossas análises. Por exemplo, se tentarmos calcular a média de um conjunto com valores `NA`, o resultado será `NA`:

```
numeros_com_na <- c(1, 2, NA, 4, 5)
mean(numeros_com_na)
```

```
[1] NA
```

Em aulas futuras falaremos sobre como identificar os `NA` em nossos dados, e algumas estratégias para lidar com eles. Neste momento, podemos utilizar o argumento `na.rm = TRUE` para que os `NA` sejam removidos antes de executar a função que calcula a média:

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

```
[1] 3
```

2.6 Tipos de objetos

Existem diferentes tipos de objetos no R, e cada tipo de objeto possui diferentes propriedades. Os principais tipos de objetos que utilizaremos ao longo do curso são: vetores, *data.frames* e listas.

2.6.1 Vetores

Vetores armazenam um conjunto de valores de uma dimensão. Eles podem ser criados com a função `c()`, que significa *combine* (combinar). Por exemplo, para criar um vetor com os números de 1 a 5:

```
vetor_de_numeros <- c(1, 2, 3, 4, 5)
```

Os vetores podem armazenar diferentes tipos de dados, como números, textos, fatores, entre outros. Porém cada vetor pode armazenar apenas um tipo de dado. Por exemplo, se tentarmos criar um vetor que armazena números e textos, o R irá converter todos os valores para texto. Essa propriedade é chamada de **coerção**.

```
vetor_misto <- c(1, 2, "três", 4, 5)
class(vetor_misto)
```

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

```
vetor_misto
```

```
[1] "1"    "2"    "três" "4"    "5"
```

No geral, podemos converter dados sem perder informação seguindo essa ordem: Lógico > Inteiro > Numérico > Texto.

2.6.2 Matrizes

As matrizes são conjunto de valores com duas dimensões: linhas e colunas. Assim como os vetores, as matrizes podem armazenar apenas um tipo de dado.

Para criar uma matriz, usamos a função `matrix()`. Por exemplo, para criar uma matriz com 2 linhas e 3 colunas, armazenando os números de 1 a 6, podemos usar o seguinte código:

```
matriz <- matrix(data = 1:6, nrow = 2, ncol = 3)
matriz
```

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

2.6.3 Data.frames

Os *data.frames* são conjuntos de valores com duas dimensões: linhas e colunas. Porém, diferente do que vimos para as matrizes, os *data.frames* podem armazenar diferentes tipos de dados em cada coluna.

Esse é o principal tipo de objeto que utilizaremos nesse curso, pois ele é muito útil para armazenar dados tabulares.

Existem alguns *data.frames* já criados no R, como o `airquality`, que armazena dados sobre a qualidade do ar na cidade de Nova York, em 1973. Essas são as primeiras linhas do *data.frame* `airquality`:

```
head(airquality)
```

```
  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
6    28      NA 14.9   66     5   6
```

Para criar um data.frame, podemos usar a função `data.frame()`. Entretanto, o mais comum é importar dados de arquivos, como CSV, Excel, ou de bancos de dados. Falaremos sobre como importar dados na próxima aula.

2.6.4 Listas

As listas são os objetos mais flexíveis do R: podemos armazenar diferentes tipos de objetos dentro de uma mesma lista. Por exemplo, podemos armazenar um vetor, uma matriz, um *data.frame*, e até mesmo outra lista dentro de uma lista.

Podemos criar uma lista com a função `list()`. Os elementos da lista podem ser nomeados ou não. Vamos criar uma lista nomeada, com três elementos: o número , o *data.frame* `airquality`, e o vetor `letters`:

```
lista_exemplo <- list(numero_pi = pi,
                      df_qualidade_do_ar = airquality,
                      letras = letters)
class(lista_exemplo)
```

```
[1] "list"
```

A função `str()` (*structure*) exibe a estrutura da lista, mostrando os elementos e seus tipos:

```
str(lista_exemplo)
```

```
List of 3
 $ numero_pi         : num 3.14
 $ df_qualidade_do_ar:'data.frame': 153 obs. of  6 variables:
  ..$ Ozone  : int [1:153] 41 36 12 18 NA 28 23 19 8 NA ...
  ..$ Solar.R: int [1:153] 190 118 149 313 NA NA 299 99 19 194 ...
  ..$ Wind   : num [1:153] 7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
  ..$ Temp   : int [1:153] 67 72 74 62 56 66 65 59 61 69 ...
  ..$ Month  : int [1:153] 5 5 5 5 5 5 5 5 5 5 ...
  ..$ Day    : int [1:153] 1 2 3 4 5 6 7 8 9 10 ...
 $ letras            : chr [1:26] "a" "b" "c" "d" ...
```

2.7 Materiais complementares

- Materiais do curso Introdução à análise de dados no R:
  - Diretório de trabalho e projetos
  - Linguagem R
- Livro R para Ciência de Dados 2ed:
  - Fluxo de Trabalho: básico
- Livro Zen do R:
  - Capítulo .RData e .Rhistory