

Análise de dados para o Planejamento Territorial

Práticas em R

Flávia da Fonseca Feitosa

Beatriz Milz

2025-06-03

Índice

Introdução	4
Calendário	4
Sobre este material	4
Licença	5
1 Introdução ao R e RStudio	6
1.1 Introdução	6
1.2 O que é o R?	6
1.3 O que é o RStudio?	7
1.4 Instalando o R e o RStudio	7
1.4.1 Instalação do R	8
1.4.2 Instalação do RStudio	8
1.5 Conhecendo o RStudio	9
1.6 Scripts	10
1.6.1 Como executar os códigos?	12
1.6.2 Comentários	13
1.7 Funções	13
1.8 Pacotes	14
1.8.1 Repositório de pacotes	16
1.9 Documentação	17
1.9.1 Documentação no RStudio	17
1.9.2 Documentação online	19
1.9.3 Cheatsheets	21
1.10 Materiais complementares	21
2 Conceitos básicos do R	22
2.1 Operações matemáticas	22
2.2 Objetos	23
2.2.1 Objetos existentes no R	23
2.2.2 Criando um objeto	24
2.3 Tipos de objetos	25
2.3.1 Vetores	25
2.3.2 Data.frames	26
2.4 Materiais complementares	26

3	Análise exploratória de dados - Parte 1	27
3.1	Criando um projeto	27
3.2	Salvando os dados no projeto	32
3.3	Importando os dados	33
3.4	Conhecendo a base de dados	34
3.5	Calculando estatísticas descritivas	36
3.6	Visualizando os dados	37
3.7	Materiais complementares	39
4	Erros e <i>warnings</i> frequentes	40
4.1	Instalação	40
4.1.1	RTools	40
4.2	Conceitos básicos	41
4.2.1	Instalando pacotes	41
4.2.2	Pacote não encontrado	42
4.2.3	Objeto não encontrado	42
4.2.4	Função não encontrada	43

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.

! Importante

Este material foi feito para guiar as aulas práticas, mas você verá que ele está bem detalhado. Dessa forma, você pode usá-lo para revisar os conceitos e praticar as atividades (dentro e fora do horário das aulas).

Calendário

Semana	Período	Práticas
1	02/06/2025 - 06/06/2025	Introdução ao R e RStudio
2	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

Esse material está disponível sob a [licença CC BY-SA 4.0](#).

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:

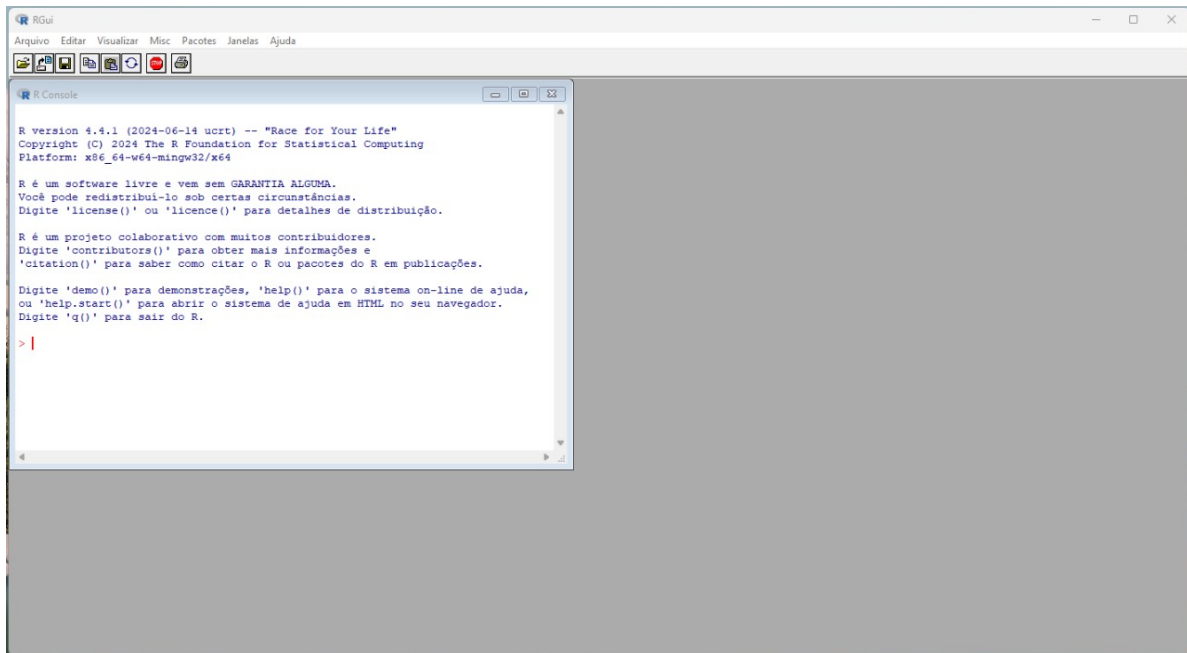


Figura 1.1: Captura de tela do R Console no Windows

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](https://cran.r-project.org/) e escolha o link de download de acordo com o seu sistema operacional:

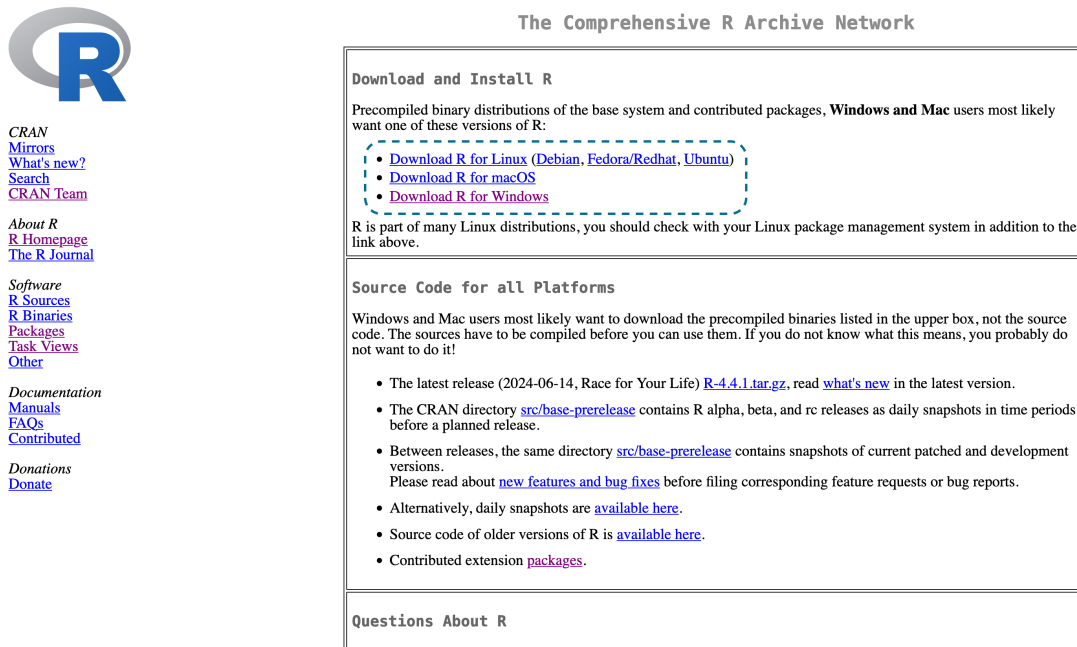


Figura 1.2: Captura de tela do site CRAN

Instale o R utilizando o instalador baixado.

1.4.2 Instalação do RStudio

Após instalar o R, acesse o site [RStudio Desktop](https://rstudio.com/) e escolha o link de download de acordo com o seu sistema operacional:

1: Install R

RStudio requires R 3.6.0+. Choose a version of R that matches your computer's operating system.

R is not a Posit product. By clicking on the link below to download and install R, you are leaving the Posit website. Posit disclaims any obligations and all liability with respect to R and the R website.

DOWNLOAD AND INSTALL R

2: Install RStudio

DOWNLOAD RSTUDIO DESKTOP FOR MACOS 12+

This version of RStudio is only supported on macOS 12 and higher. For earlier macOS environments, please [download a previous version](#).

Size: 664.40 MB | [SHA-256: D0DDD395](#) | Version: 2024.04.2+764 | Released: 2024-06-10

Figura 1.3: Captura de tela do site RStudio Desktop

Instale o RStudio utilizando o instalador baixado.

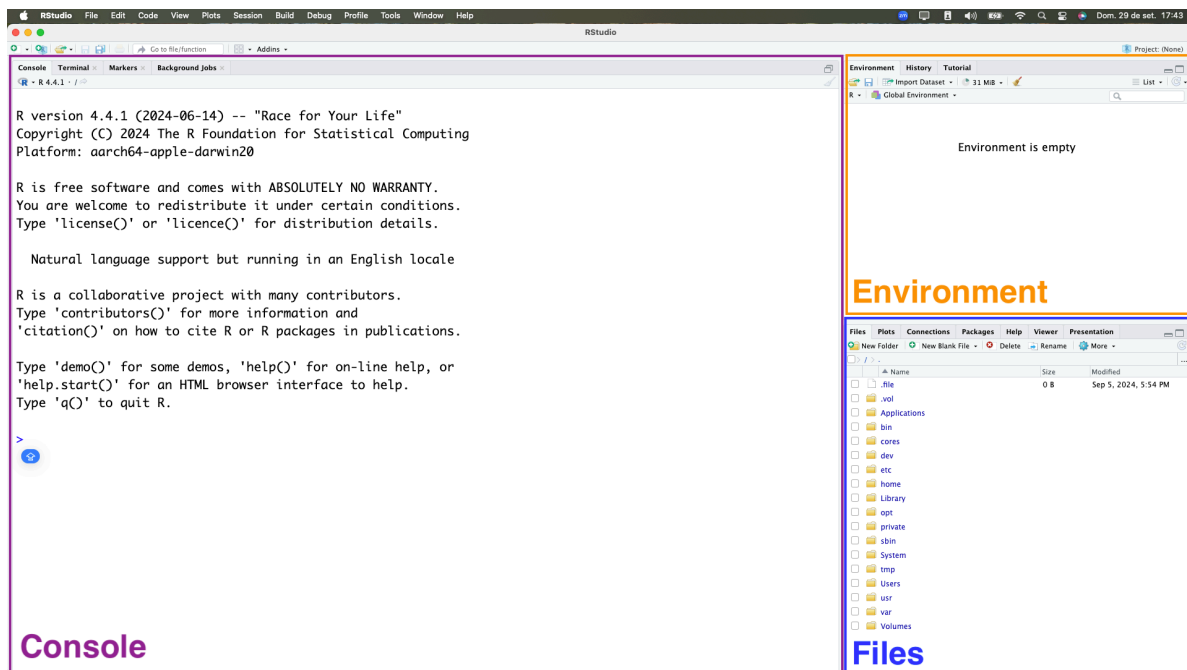
Dica

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`:

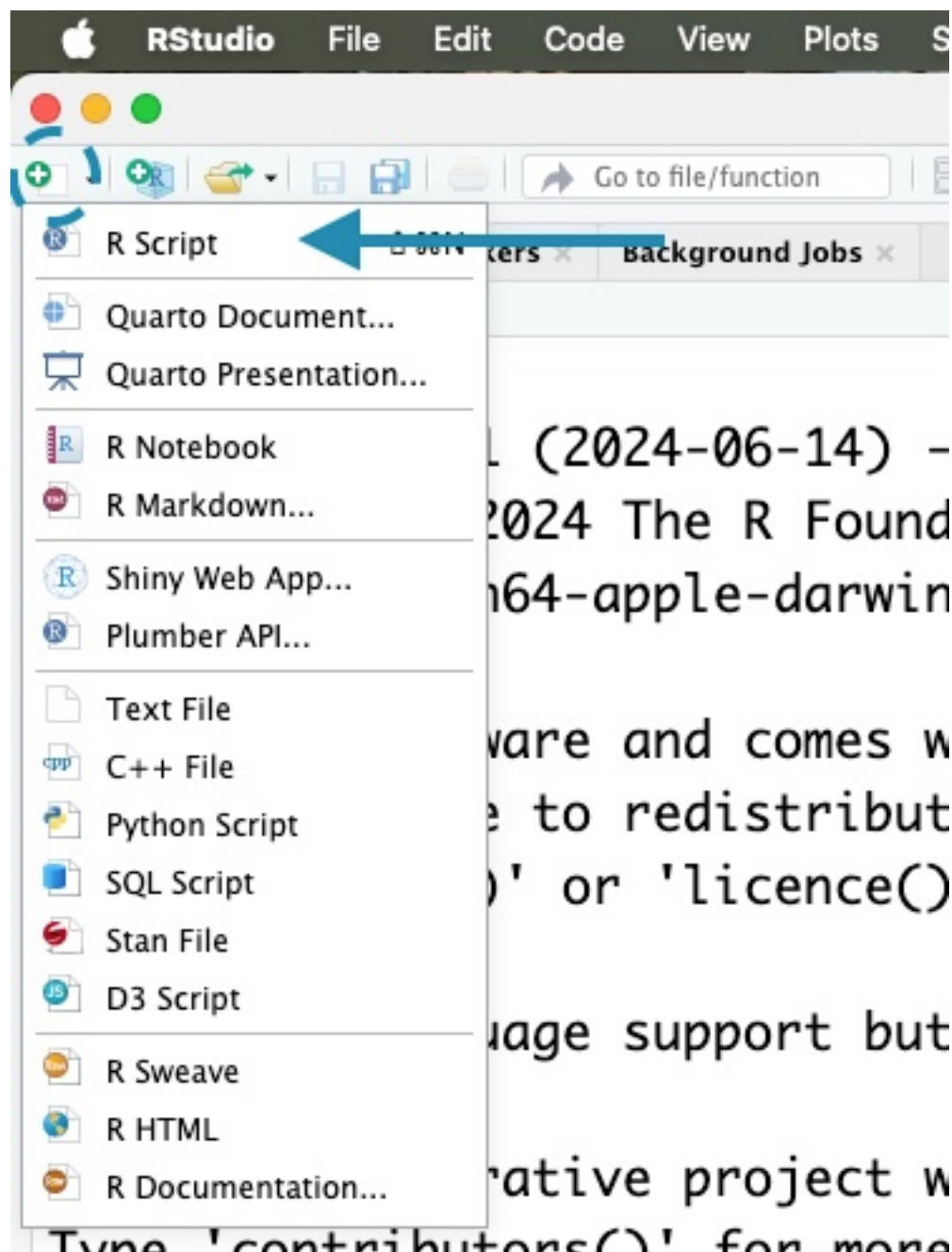


Figura 1.4: Captura de tela do RStudio: Opção para criar novo Script

Após abrir um script, o RStudio exibirá 4 painéis:

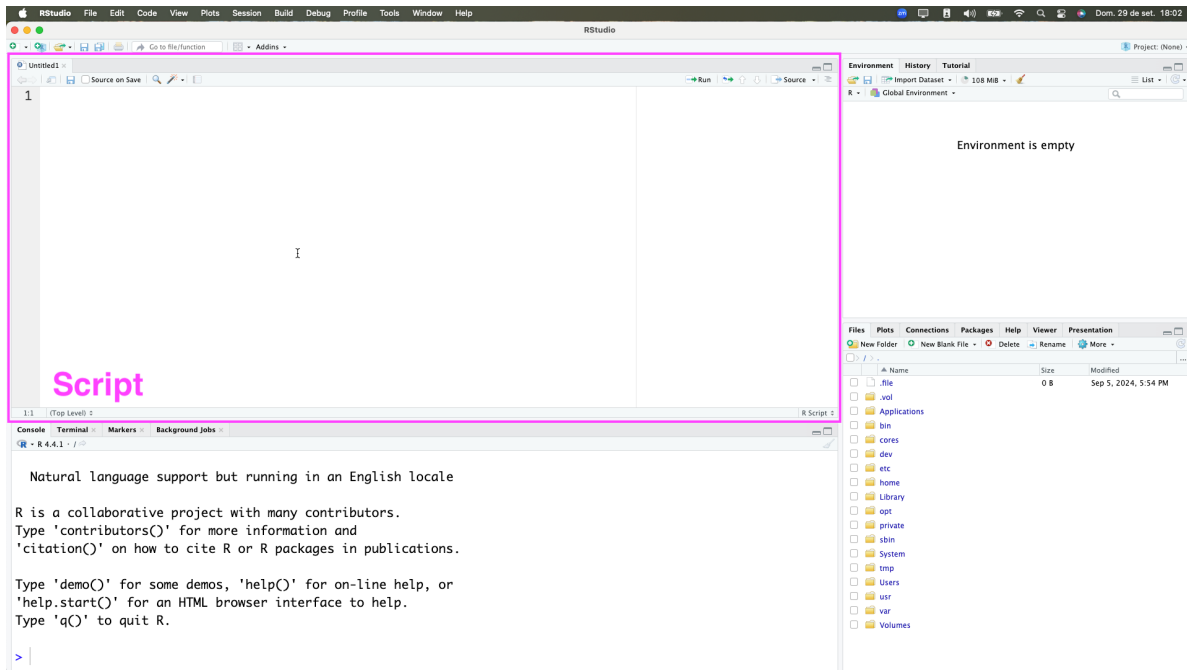



Figura 1.5: Captura de tela do RStudio

💡 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:

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

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

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
```

Algumas funções podem receber mais de um argumento. Por exemplo, a função `round()` arredonda um número para um determinado número de casas decimais. Para utilizá-la, podemos escrever `round()` e informar o número e o número de casas decimais entre parênteses:

```
pi
```

```
[1] 3.141593
```

```
# Sem argumentos: arredondar o número pi para um número inteiro (0 casas decimais)
round(pi)
```

```
[1] 3
```

```
# Com argumentos: arredondar o número pi para 2 casas decimais
round(pi, digits = 2)
```

```
[1] 3.14
```

Podemos consultar a documentação de uma função para entender como ela funciona, quais argumentos ela aceita e como utilizá-la. Falaremos mais sobre isso na seção de documentação.

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.

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 become
```

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.

1.8.1 Repositório de pacotes

Existem diferentes repositórios de pacotes do R, que são locais onde os pacotes são armazenados e disponibilizados para instalação.

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 que um pacote seja adicionado ao CRAN, ele deve atender a critérios de qualidade de software.

A [rOpenSci](#) é uma organização que mantém uma [coleção de pacotes](#) que foram revisados por pares e que atendem a critérios de qualidade. Esses pacotes são voltados para pesquisa, ciência aberta e reprodutibilidade.

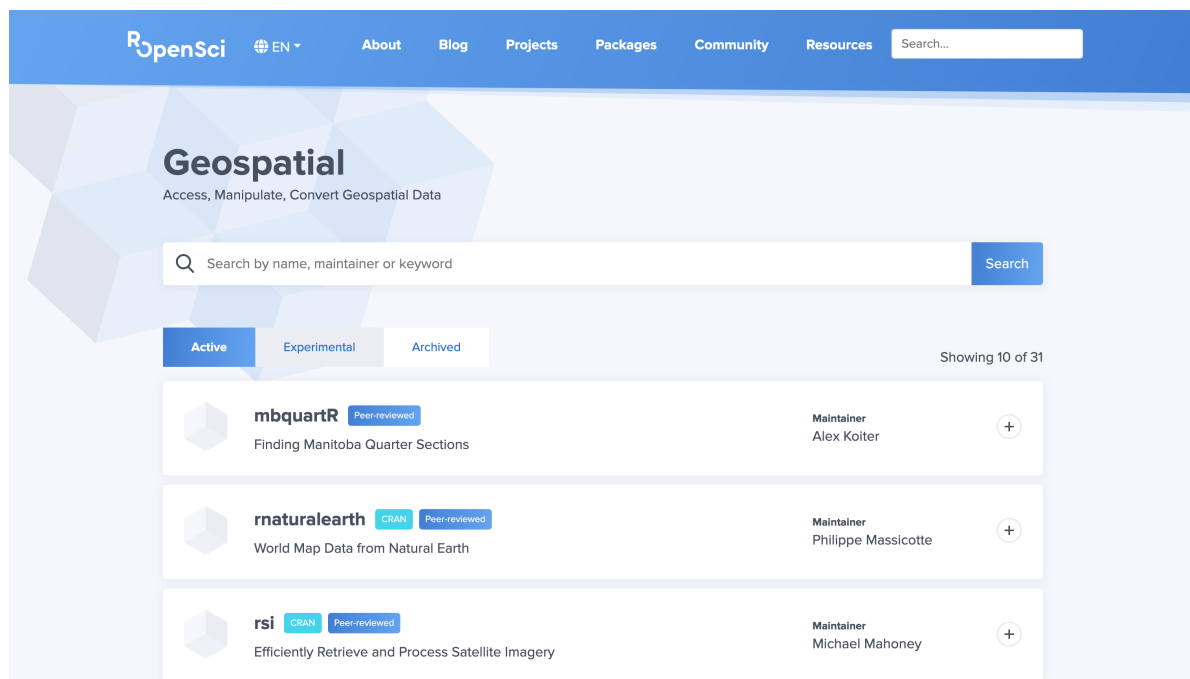


Figura 1.6: Captura de tela da página da rOpenSci: página de pacotes no tema Geoespacial

A rOpenSci também mantém o [R-universe](#), uma plataforma que permite que pacotes sejam publicados e compartilhados de forma mais fácil. O R Universe é uma alternativa ao CRAN, e permite que pacotes sejam publicados sem a necessidade de passar pelo processo de revisão do CRAN.

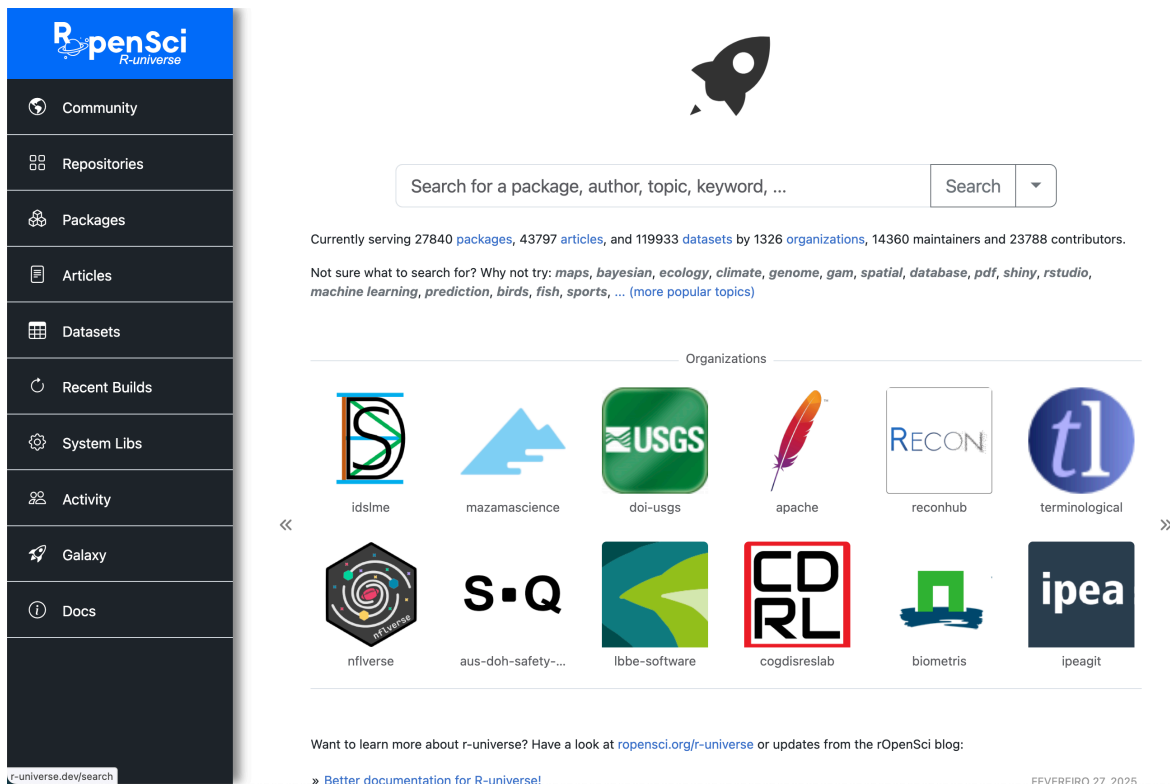


Figura 1.7: Captura de tela da página do R-Universe

Outros repositórios de pacotes também existem, como o [Bioconductor](#), que é voltado para análise de dados biológicos e genômicos.

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:

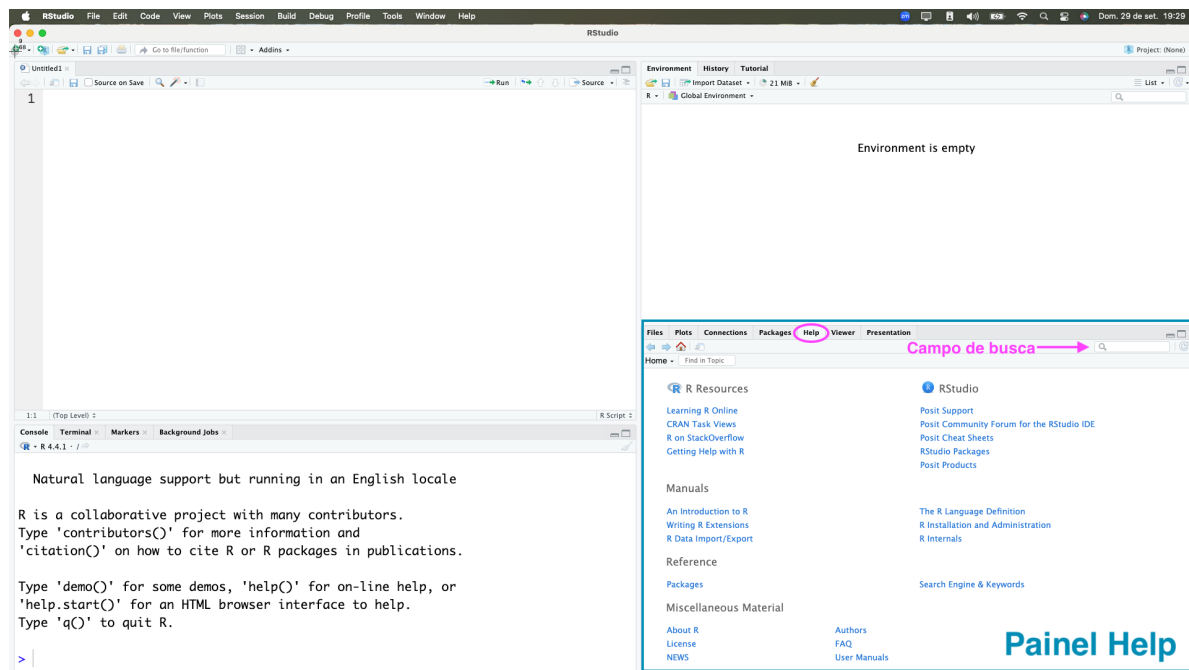
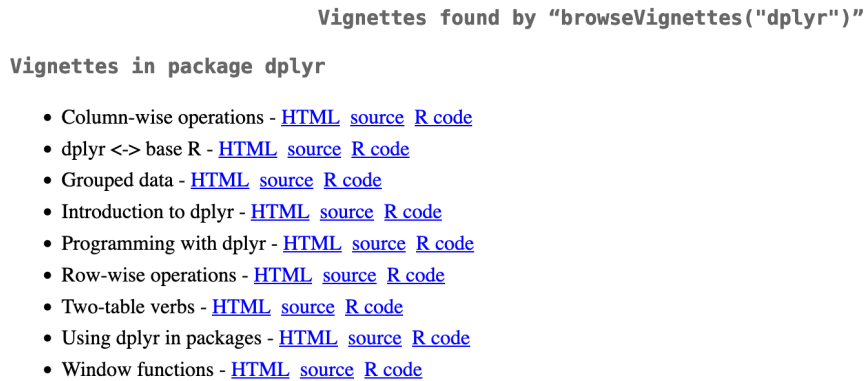


Figura 1.8: Captura de tela do RStudio: Painel Help

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")
```



Vignettes found by "browseVignettes("dplyr")"

Vignettes in package dplyr

- Column-wise operations - [HTML](#) [source](#) [R code](#)
- dplyr <-> base R - [HTML](#) [source](#) [R code](#)
- Grouped data - [HTML](#) [source](#) [R code](#)
- Introduction to dplyr - [HTML](#) [source](#) [R code](#)
- Programming with dplyr - [HTML](#) [source](#) [R code](#)
- Row-wise operations - [HTML](#) [source](#) [R code](#)
- Two-table verbs - [HTML](#) [source](#) [R code](#)
- Using dplyr in packages - [HTML](#) [source](#) [R code](#)
- Window functions - [HTML](#) [source](#) [R code](#)

Figura 1.9: 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:

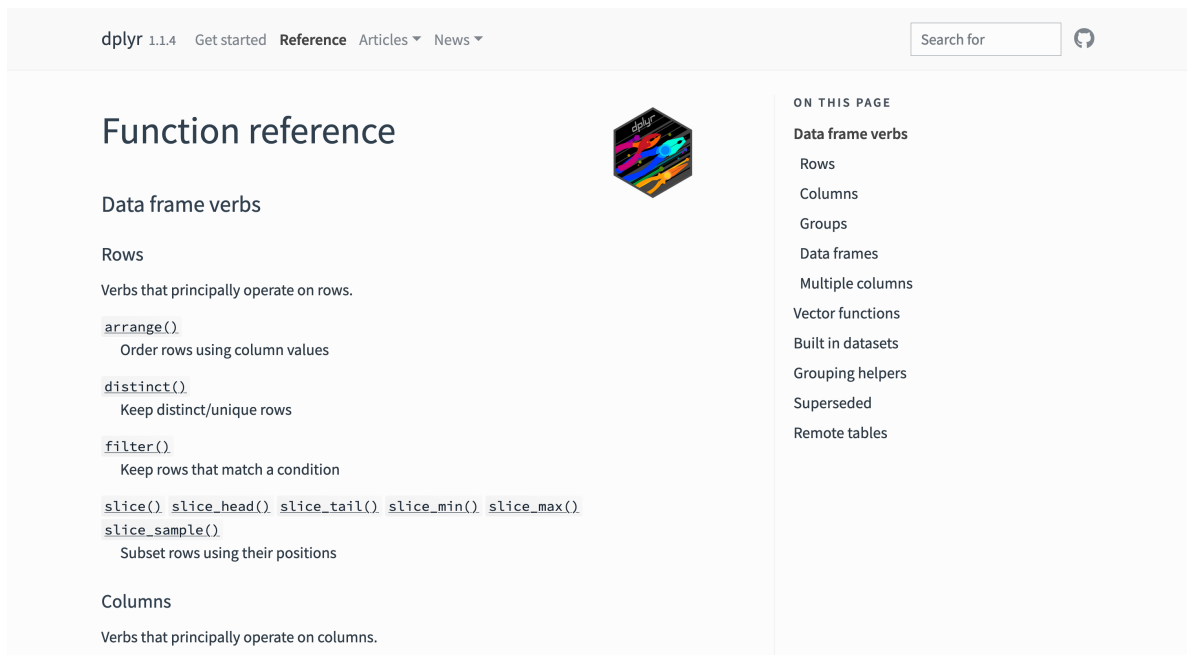


Figura 1.10: Captura de tela: Site do pacote dplyr - Reference

- Em [Articles](#) podemos acessar as *vignettes*:

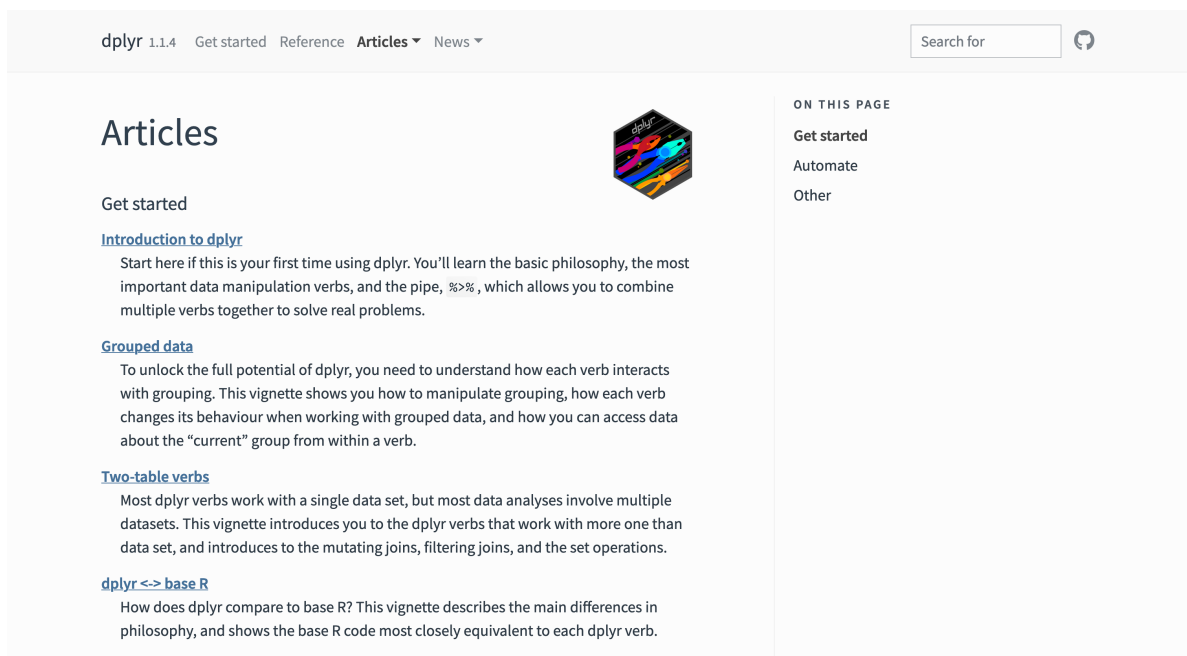


Figura 1.11: Captura de tela: Site do pacote dplyr - Vignettes

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 Conceitos básicos do 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 π (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.

2.3 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 e *data.frames*.

2.3.1 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.

2.3.2 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.

2.4 Materiais complementares

- Materiais do curso [Introdução à análise de dados no R](#):
 - [Diretório de trabalho e projetos](#)
 - [Linguagem R](#)

3 Análise exploratória de dados - Parte 1

Nesta aula, vamos conhecer algumas funções do R e do pacote tidyverse que nos ajudam a fazer uma análise exploratória dos dados.

Dados

Utilizaremos dados de saneamento por município do estado de São Paulo, disponibilizados pela CETESB¹, referente à 2022.

O arquivo csv pode ser baixado através [deste link](#).

Esses dados foram [originalmente disponibilizados em PDF](#), e foram [extraídos e organizados](#) por Beatriz Milz.

3.1 Criando um projeto

O RStudio possui uma funcionalidade chamada **projetos**. Quando criamos um projeto no RStudio, uma nova pasta é criada no computador, e o RStudio define essa pasta como o diretório de trabalho. Além disso, o RStudio também cria um arquivo com a extensão **.Rproj** dentro dessa pasta, que contém informações sobre o projeto.

É recomendado que sempre trabalhemos em projetos no RStudio, pois isso facilita a organização dos arquivos e a reprodução do código.

É recomendado também salvar os arquivos referentes ao projeto (como scripts, bases de dados, resultados, etc) dentro do projeto. Isso não significa que precisamos colocar todos os arquivos dentro da pasta principal do projeto: podemos criar sub-pastas para organizar os arquivos.

Para criar um projeto no RStudio, primeiro precisamos acessar o menu de criação de projetos (*New project Wizard*). Podemos fazer isso de três formas:

- No menu superior, clicando em **File > New Project...**

- Clicando no ícone de novo projeto na barra de ferramentas do RStudio:



¹CETESB - Companhia Ambiental do Estado de São Paulo.

- No canto superior esquerdo, clicando no botão referente à projetos, e depois em **New Project...**:

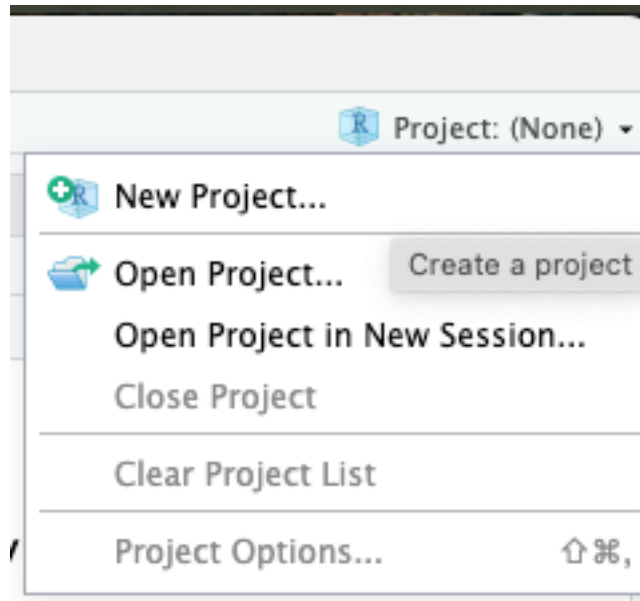


Figura 3.1: Captura de tela do RStudio: Menu de projetos

Depois, escolhemos o tipo de projeto que queremos criar. No geral, escolhemos a opção **New Directory**, para criar uma nova pasta no computador:

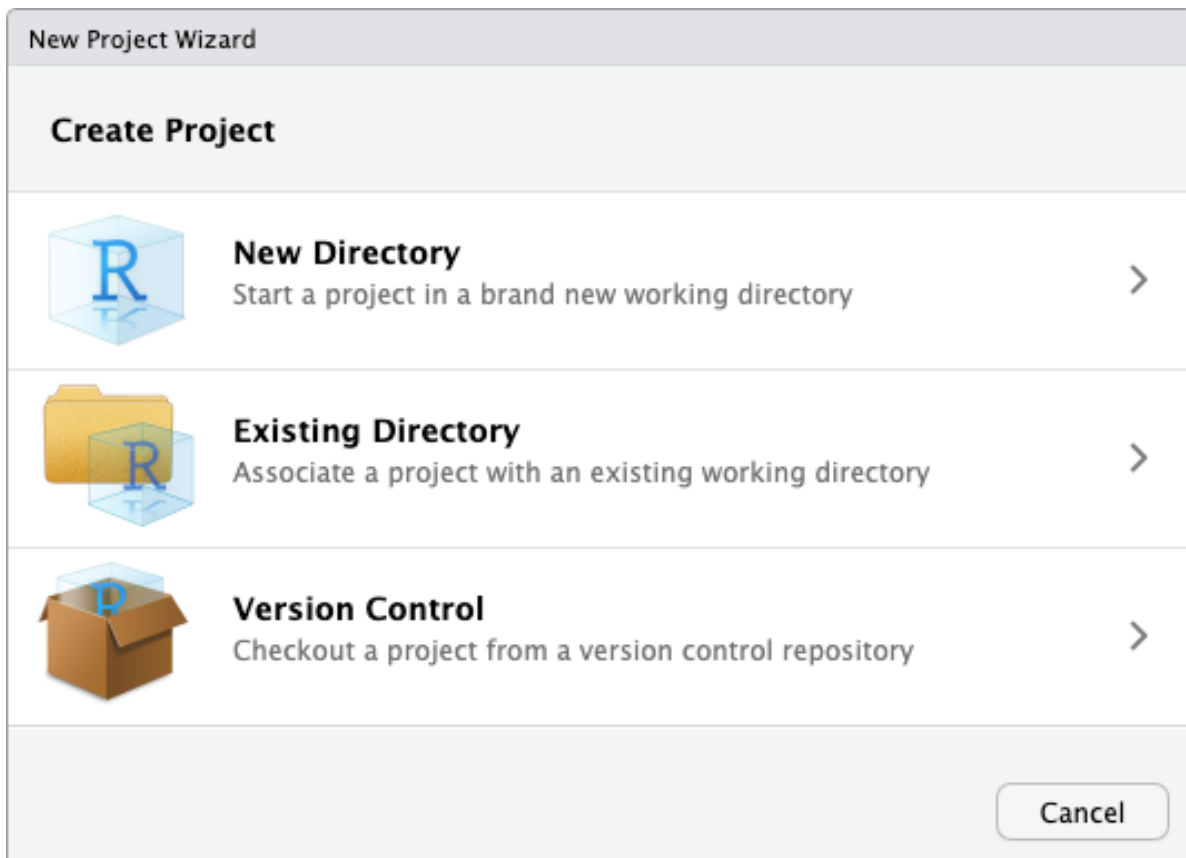


Figura 3.2: Captura de tela do RStudio: Criando um projeto

Depois, escolhemos o tipo de projeto que queremos criar. Cada tipo de projeto apresenta arquivos específicos de template. O RStudio apresenta algumas opções de projeto, porém é possível adicionar novos tipos de projeto instalando pacotes específicos.

No geral, escolhemos a opção **New Project**, para criar um projeto simples:

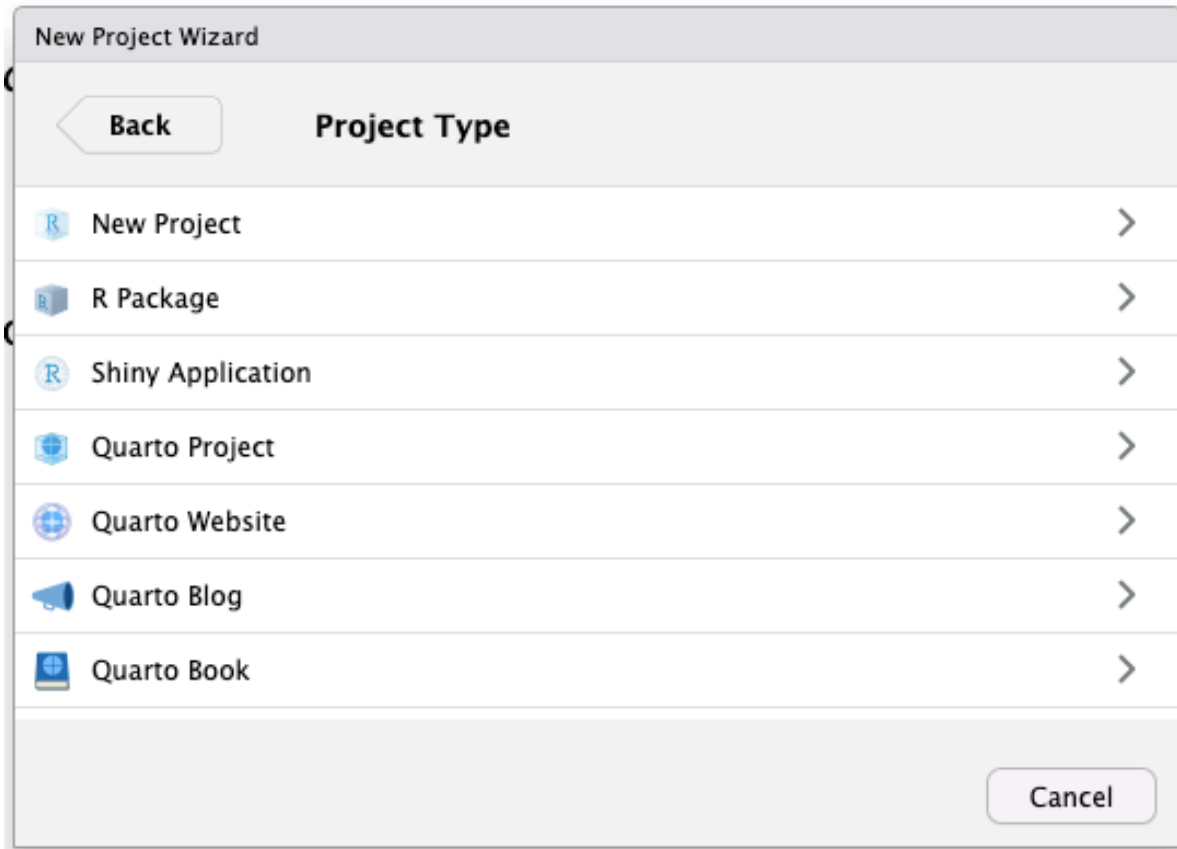


Figura 3.3: Captura de tela do RStudio: Escolhendo o tipo de projeto

Na tela seguinte, precisamos informar o nome do projeto (no campo *Directory name*) e o diretório onde ele será criado (no campo *Create project as subdirectory of*):

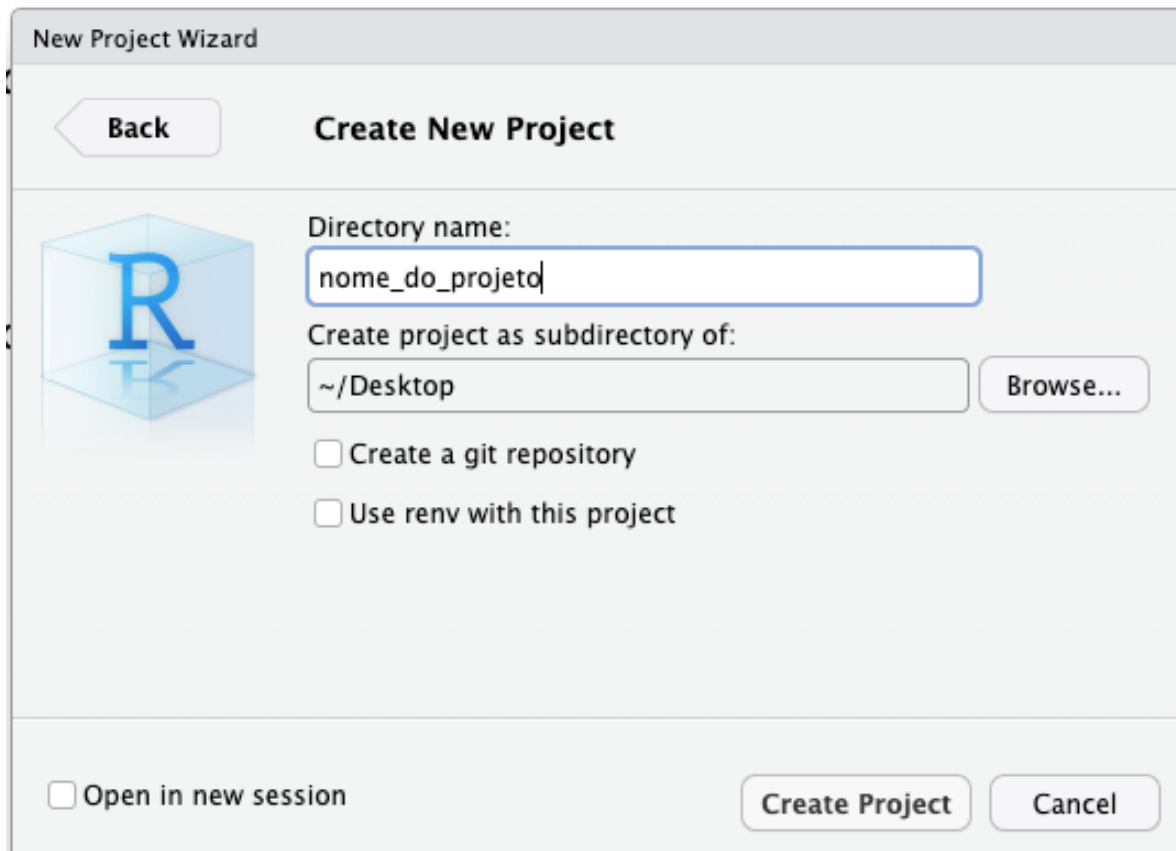


Figura 3.4: Captura de tela do RStudio: Nomeando o projeto

Após preencher as informações solicitadas, clicamos em **Create Project**. O RStudio criará o projeto e o abrirá:

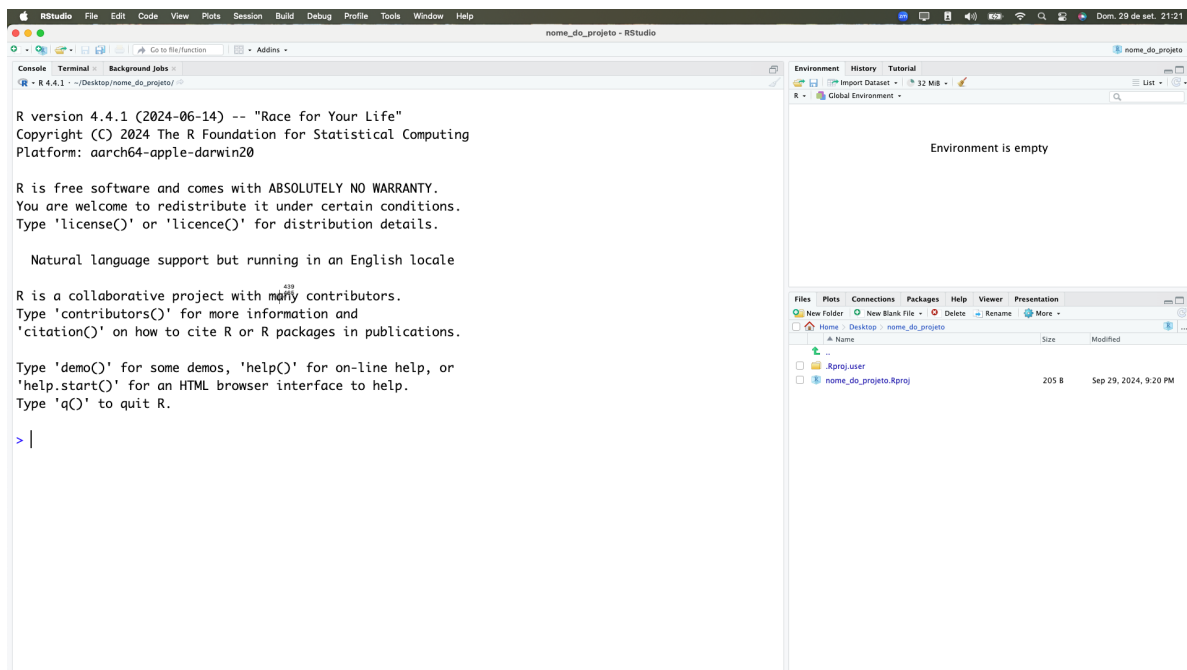


Figura 3.5: Captura de tela do RStudio: projeto criado

Dica

Note que o nome do projeto que criamos aparece no canto superior direito do RStudio.

3.2 Salvando os dados no projeto

Para facilitar o trabalho, vamos salvar os dados que utilizaremos nesta aula dentro do projeto que acabamos de criar.

Primeiro, vamos criar uma pasta chamada `dados` dentro do projeto:

```
dir.create("dados")
```

Depois, vamos baixar os dados da CETESB e salvar na pasta `dados` que acabamos de criar:

```
download.file(
  url = "https://raw.githubusercontent.com/beatrizmilz/cetesb_saneamento/refs/heads/main/data",
  destfile = "dados/dados-cetesb-2022.csv"
)
```


Confira se o arquivo foi baixado corretamente, e se está na pasta `dados` do projeto.

3.3 Importando os dados

Para importar os dados que acabamos de baixar, vamos utilizar a função `read_csv()` do pacote `readr`, que faz parte do `tidyverse`.

Lembre-se de carregar o pacote `tidyverse` antes de utilizar a função `read_csv()`:

```
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 become
```

Agora, podemos importar os dados:

```
dados_cetesb <- read_csv("dados/dados-cetesb-2022.csv")
```

```
Rows: 645 Columns: 12
-- Column specification -----
Delimiter: ","
chr  (2): uf, municipio
dbl (10): ano, ugrhi, codigo_ibge, populacao_urbana, atendimento_coleta_porc...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Vamos conferir se os dados foram importados corretamente, utilizando a função `glimpse()`:

```
glimpse(dados_cetesb)
```

```

Rows: 645
Columns: 12
$ ano                <dbl> 2022, 2022, 2022, 2022, 2022, 2022, 2022~
$ uf                 <chr> "SP", "SP", "SP", "SP", "SP", "SP", "SP"~
$ ugrhi              <dbl> 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2~
$ municipio          <chr> "Campos do Jordão", "Santo Antônio do Pi~
$ codigo_ibge         <dbl> 3509700, 3548203, 3548609, 3502507, 3503~
$ populacao_urbana    <dbl> 52384, 4067, 5251, 35684, 1844, 2619, 88~
$ atendimento_coleta_porc <dbl> 52.6, 46.7, 98.0, 70.0, 95.6, 100.0, 88.~
$ atendimento_tratamento_porc <dbl> 100.0, 100.0, 100.0, 0.0, 95.6, 0.0, 100~
$ eficiencia          <dbl> 93.0, 80.0, 91.3, 0.0, 99.0, 0.0, 75.0, ~
$ carga_poluidora_potencial <dbl> 2829, 220, 284, 1927, 100, 141, 476, 442~
$ carga_poluidora_remancescente <dbl> 1445, 138, 30, 1927, 9, 141, 161, 1858, ~
$ icitem              <dbl> 5.97, 4.63, 9.97, 1.55, 9.86, 1.50, 7.61~

```

A função `View()` também pode ser utilizada para visualizar os dados em uma tabela interativa:

```
View(dados_cetesb)
```

3.4 Conhecendo a base de dados

Para conhecer melhor a base de dados, podemos utilizar algumas funções para explorar as colunas e os tipos de dados.

A função `colnames()` nos mostra os nomes das colunas:

```
colnames(dados_cetesb)
```

```

[1] "ano"                "uf"
[3] "ugrhi"              "municipio"
[5] "codigo_ibge"        "populacao_urbana"
[7] "atendimento_coleta_porc" "atendimento_tratamento_porc"
[9] "eficiencia"          "carga_poluidora_potencial"
[11] "carga_poluidora_remancescente" "icitem"

```

A função `head()` nos mostra as primeiras linhas da base de dados, e a função `tail()` nos mostra as últimas linhas:

```
head(dados_cetesb)
```

```
# A tibble: 6 x 12
  ano uf    ugrhi municipio      codigo_ibge populacao_urbana
<dbl> <chr> <dbl> <chr>      <dbl>      <dbl>
1  2022 SP      1 Campos do Jordão      3509700      52384
2  2022 SP      1 Santo Antônio do Pinhal 3548203      4067
3  2022 SP      1 São Bento do Sapucaí    3548609      5251
4  2022 SP      2 Aparecida      3502507     35684
5  2022 SP      2 Arapeí        3503158      1844
6  2022 SP      2 Areias        3503505      2619
# i 6 more variables: atendimento_coleta_porco <dbl>,
#   atendimento_tratamento_porco <dbl>, eficiencia <dbl>,
#   carga_poluidora_potencial <dbl>, carga_poluidora_remanescente <dbl>,
#   icitem <dbl>
```

```
tail(dados_cetesb)
```

```
# A tibble: 6 x 12
  ano uf    ugrhi municipio      codigo_ibge populacao_urbana
<dbl> <chr> <dbl> <chr>      <dbl>      <dbl>
1  2022 SP     22 Rosana      3544251     12828
2  2022 SP     22 Sandovalina 3545506      3074
3  2022 SP     22 Santo Anastácio 3547700     19434
4  2022 SP     22 Taciba      3552908      5410
5  2022 SP     22 Tarabai     3553906      7035
6  2022 SP     22 Teodoro Sampaio 3554300     18996
# i 6 more variables: atendimento_coleta_porco <dbl>,
#   atendimento_tratamento_porco <dbl>, eficiencia <dbl>,
#   carga_poluidora_potencial <dbl>, carga_poluidora_remanescente <dbl>,
#   icitem <dbl>
```

A função `summary()` nos mostra um resumo de estatísticas descritivas para todas as colunas. Mas cuidado: nem todos os resultados fazem sentido (exemplo: a coluna `codigo_ibge` é um identificador, e não devemos calcular estatísticas descritivas para ela).

```
summary(dados_cetesb)
```

```
      ano      uf      ugrhi      municipio
Min.   :2022   Length:645   Min.    : 1.00   Length:645
```

1st Qu.:2022	Class :character	1st Qu.: 7.00	Class :character
Median :2022	Mode :character	Median :13.00	Mode :character
Mean :2022		Mean :12.38	
3rd Qu.:2022		3rd Qu.:17.00	
Max. :2022		Max. :22.00	

codigo_ibge	populacao_urbana	atendimento_coleta_porc
Min. :3500105	Min. : 653	Min. : 12.80
1st Qu.:3514601	1st Qu.: 4498	1st Qu.: 88.30
Median :3528700	Median : 11524	Median : 98.70
Mean :3528698	Mean : 69406	Mean : 91.06
3rd Qu.:3543204	3rd Qu.: 39045	3rd Qu.:100.00
Max. :3557303	Max. :12284940	Max. :100.00

atendimento_tratamento_porc	eficiencia	carga_poluidora_potencial
Min. : 0.00	Min. : 0.00	Min. : 35
1st Qu.:100.00	1st Qu.:74.97	1st Qu.: 243
Median :100.00	Median :83.45	Median : 622
Mean : 89.73	Mean :77.04	Mean : 3748
3rd Qu.:100.00	3rd Qu.:88.80	3rd Qu.: 2108
Max. :100.00	Max. :99.10	Max. :663387
NA's :5	NA's :5	

carga_poluidora_remancescente	icitem
Min. : 1.0	Min. : 0.75
1st Qu.: 49.0	1st Qu.: 6.78
Median : 156.0	Median : 8.21
Mean : 1409.2	Mean : 7.76
3rd Qu.: 617.8	3rd Qu.: 9.94
Max. :231038.0	Max. :10.00
NA's :5	

3.5 Calculando estatísticas descritivas

Como vimos acima, a função `summary()` nos dá um resumo de estatísticas descritivas para todas as colunas. Porém, podemos querer calcular estatísticas descritivas apenas para algum subconjunto dos dados.

Imagine que queremos calcular algumas estatísticas descritivas considerando cada UGRHI (Unidade de Gerenciamento de Recursos Hídricos) separadamente. Podemos fazer isso utilizando as funções `group_by()` e `summarise()`.

```
dados_cetesb |>
  # agrupar os dados pela coluna ugrhi
  group_by(ugrhi) |>
  # calcular estatísticas descritivas para cada grupo
  summarise(
    quantidade_municipios = n(),
    soma_populacao_urbana = sum(populacao_urbana),
    media_atendimento_coleta_porc = mean(atendimento_coleta_porc, na.rm = TRUE),
    mediana_atendimento_coleta_porc = median(atendimento_coleta_porc, na.rm = TRUE),
    desvio_padrao_atendimento_coleta_porc = sd(atendimento_coleta_porc, na.rm = TRUE)
  )
```

```
# A tibble: 22 x 6
  ugrhi quantidade_municipios soma_populacao_urbana media_atendimento_coleta_~1
  <dbl>                <int>                <dbl>                <dbl>
1     1                 3             61702             65.8
2     2                34            2124413            84.2
3     3                 4            337159            49.6
4     4                23            1215586            97.2
5     5                57            5737151            87.0
6     6                34            21626154            72.4
7     7                 9            1893370            72.9
8     8                22             707923            98.1
9     9                38            1542781            97.0
10    10                33            1937230            79.8
# i 12 more rows
# i abbreviated name: 1: media_atendimento_coleta_porc
# i 2 more variables: mediana_atendimento_coleta_porc <dbl>,
#   desvio_padrao_atendimento_coleta_porc <dbl>
```

3.6 Visualizando os dados

Para visualizar os dados, podemos utilizar a função `ggplot()` do pacote `ggplot2`, que também faz parte do `tidyverse`.

Para quem está começando, recomendo utilizar o pacote `esquisse`, que facilita a criação de gráficos com o `ggplot2`.

```
install.packages("esquisse")
install.packages("plotly")
```

Depois de instalar o pacote, podemos carregá-lo e utilizar a função `esquisser()` para criar gráficos interativos:

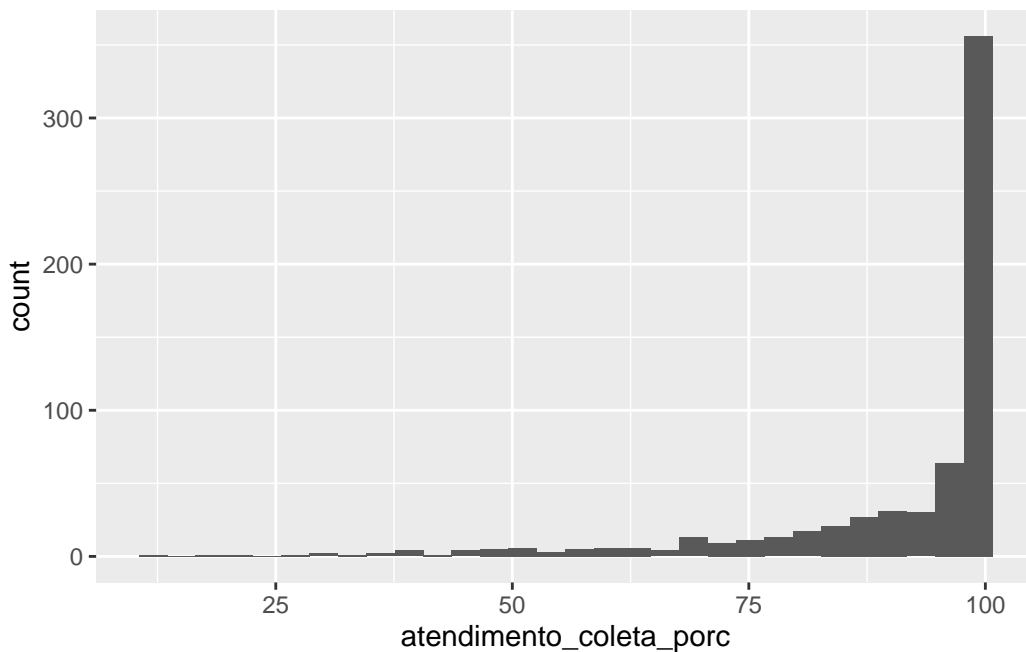
```
library(esquisse)
esquisser(dados_cetesb)
```

O `esquisse` oferece uma interface amigável para criar gráficos com o `ggplot2`, permitindo que você arraste e solte variáveis, escolha tipos de gráficos e customize os elementos do gráfico. Ao usar o `esquisse`, você pode gerar o código correspondente ao gráfico que está criando, e depois copiá-lo para o seu script R.

Para criar um histograma simples, por exemplo, podemos usar o seguinte código:

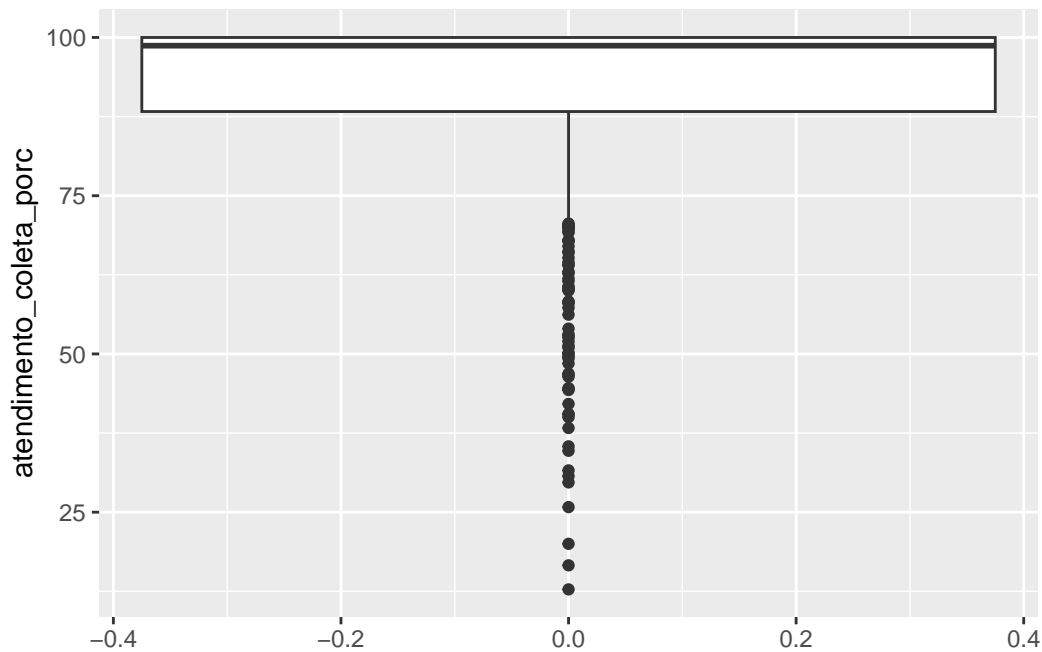
```
# inicia o gráfico com os dados `dados_cetesb`
ggplot(dados_cetesb) +
  # define a variável que será plotada no eixo x
  aes(x = atendimento_coleta_porc) +
  # adiciona a geometria de histograma
  geom_histogram()
```

``stat_bin()` using `bins = 30`. Pick better value with `binwidth`.`



Podemos criar também um boxplot com um código semelhante:

```
# inicia o gráfico com os dados `dados_cetesb`
ggplot(dados_cetesb) +
  # define a variável que será plotada no eixo y
  aes(y = atendimento_coleta_porc) +
  # adiciona a geometria de histograma
  geom_boxplot()
```



i Nota

Nesta aula, o objetivo foi apresentar algumas funções básicas que veremos outras vezes ao longo do curso. Não se preocupe se não entendeu tudo agora, pois iremos aprofundar esses conceitos em aulas futuras.

Se você quiser praticar mais, recomendo que explore os materiais complementares.

3.7 Materiais complementares

- Materiais do curso [Introdução à análise de dados no R](#):
 - [Diretório de trabalho e projetos](#)
 - [Importando dados](#)
 - [Conhecendo a base de dados](#)

4 Erros e *warnings* frequentes

A lista a seguir apresenta alguns erros e *warnings* mais comuns.

4.1 Instalação

4.1.1 RTools

Para pessoas que utilizam o sistema operacional Windows, a aviso (*warning*) abaixo pode aparecer em alguns contextos:

```
WARNING: Rtools is required to build R packages but is not currently installed.  
Please download and install the appropriate version of Rtools before proceeding:
```

```
https://cran.rstudio.com/bin/windows/Rtools/  
Instalando pacote em 'C:/Users/.../AppData/Local/R/win-library/4.4'  
(como 'lib' não foi especificado)
```

Para que esse aviso não apareça mais, você pode instalar o Rtools no seu computador. O RTools é um software (**não** é um pacote do R), portanto você precisa fazer o download da versão compatível com a versão do R que você está utilizando, e instalar no seu computador.

Para fazer o download, acesse o link <https://cran.rstudio.com/bin/windows/Rtools/>, e escolha a versão do RTools compatível com a versão do R que você está utilizando:

RTools: Toolchains for building R and R packages from source on Windows

Choose your version of Rtools:

RTools 4.4	for R versions from 4.4.0 (R-release and R-devel)
RTools 4.3	for R versions 4.3.x (R-oldrelease)
RTools 4.2	for R versions 4.2.x
RTools 4.0	for R from version 4.0.0 to 4.1.3
old versions of RTools	for R versions prior to 4.0.0

Figura 4.1: Captura de tela: página de download do RTools

Para consultar a versão do R que você está utilizando, você pode rodar o seguinte comando no console do R:

```
R.version.string
```

```
[1] "R version 4.5.0 (2025-04-11)"
```

4.2 Conceitos básicos

4.2.1 Instalando pacotes

O erro a seguir ocorre quando o usuário tenta instalar um pacote sem aspas. O correto é colocar o nome do pacote entre aspas.

```
# O código abaixo gerará um erro:  
install.packages(janitor)
```

```
Error in eval(call, envir = parent.frame()): object 'janitor' not found
```

A função deve receber o nome do pacote **entre aspas**, pois é **um texto**:

```
# O código abaixo funcionará:  
install.packages("janitor")
```

4.2.2 Pacote não encontrado

O erro a seguir ocorre quando tentamos carregar um pacote que não foi instalado anteriormente. Para resolver, precisamos instalar o pacote.

```
# O código abaixo gerará um erro:  
library(quarto)
```

Para que consiga acessar, é necessário instalar o pacote, e depois carregá-lo:

```
install.packages("quarto")  
library(quarto)
```

4.2.3 Objeto não encontrado

O erro a seguir ocorre quando tentamos acessar um objeto que não consta no painel *Environment*. Existe alguns motivos para isso acontecer:

- O objeto não foi criado (provavelmente precisa executar o código que cria o objeto);
- O objeto existe no painel *Environment*, mas estamos tentando acessá-lo com o nome incorreto.

No exemplo a seguir, o código gerará um erro pois o objeto que estamos tentando acessar ainda não foi criado:

```
# O código abaixo gerará um erro:  
length(estados_sudeste)
```

Error: object 'estados_sudeste' not found

Após criar o objeto, conseguimos utilizá-lo:

```
estados_sudeste <- c("SP", "RJ", "MG", "ES")  
length(estados_sudeste)
```

```
[1] 4
```

4.2.4 Função não encontrada

O erro `could not find function` ocorre quando tentamos acessar uma função que não está sendo encontrada. Isso pode acontecer por alguns motivos:

- A função faz parte de um pacote que não foi carregado (precisamos carregar o pacote antes);
- A função foi escrita de forma incorreta (por exemplo, com letras maiúsculas ou minúsculas incorretas).

4.2.4.1 Pacote não carregado

No exemplo a seguir, queremos limpar o nome das colunas do *data frame* `iris`:

```
head(iris)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

O código a seguir gerará um erro pois a função `clean_names()` faz parte do pacote `janitor`, mas o pacote não foi carregado:

```
clean_names(iris)
```

```
Error in clean_names(iris): could not find function "clean_names"
```

Para corrigir, precisamos carregar o pacote `janitor`:

```
library(janitor)
iris_nome_limpo <- clean_names(iris)
head(iris_nome_limpo)
```

	sepal_length	sepal_width	petal_length	petal_width	species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

4.2.4.2 Erro de digitação

No exemplo a seguir, o código gerará um erro pois a função `length()` está escrito de forma incorreta:

```
# O código abaixo gerará um erro:
lenght(letters)
```

```
Error in lenght(letters): could not find function "lenght"
```

Para corrigir, precisamos escrever a função corretamente:

```
length(letters)
```

```
[1] 26
```