

Análise de dados para o Planejamento Territorial

Práticas em R

Flávia da Fonseca Feitosa

Beatriz Milz

2025-06-30

Índice

Introdução	5
Calendário	5
Sobre este material	6
Licença	6
1 Introdução ao R e RStudio	7
1.1 Introdução	7
1.2 O que é o R?	7
1.3 O que é o RStudio?	8
1.4 Instalando o R e o RStudio	8
1.4.1 Instalação do R	9
1.4.2 Instalação do RStudio	9
1.5 Conhecendo o RStudio	10
1.6 Scripts	11
1.6.1 Como executar os códigos?	13
1.6.2 Comentários	14
1.7 Funções	14
1.8 Pacotes	15
1.8.1 Repositório de pacotes	17
1.9 Documentação	18
1.9.1 Documentação no RStudio	18
1.9.2 Documentação online	20
1.9.3 Cheatsheets	22
1.10 Materiais complementares	22
2 Conceitos básicos do R	23
2.1 Operações matemáticas	23
2.2 Objetos	24
2.2.1 Objetos existentes no R	24
2.2.2 Criando um objeto	25
2.3 Tipos de objetos	26
2.3.1 Vetores	26
2.3.2 Data.frames	27
2.4 Materiais complementares	27

3 Análise exploratória de dados - Parte 1	28
3.1 Criando um projeto	28
3.2 Salvando os dados no projeto	33
3.3 Importando os dados	34
3.4 Conhecendo a base de dados	35
3.5 Calculando estatísticas descritivas	38
3.6 Visualizando os dados	42
3.7 Materiais complementares	44
4 Análise exploratória de dados - Parte 2	45
4.1 Carregando pacotes	45
4.2 Carregando os dados	45
4.3 Conhecendo o operador pipe (>)	46
4.4 Principais funções do dplyr	47
4.5 Filtrando dados com filter()	47
4.6 Selecionando colunas com select()	50
4.7 Adicionando ou modificando colunas com mutate()	51
4.8 Ordenando dados com arrange()	53
4.9 Agrupando dados com group_by()	54
4.10 Resumindo dados com summarise()	55
4.11 Unindo tabelas com left_join()	56
4.12 Materiais complementares	59
5 Prática - Intervalo de confiança	60
5.1 Carregar pacotes	60
5.2 Importar os dados	61
5.3 Conhecendo os dados	62
5.4 Calcular a média, desvio padrão e número de respostas	64
5.5 Calcular o erro padrão	65
5.6 Definindo o valor crítico	66
5.6.1 Como calcular o valor crítico para um intervalo de confiança de 95% com a distribuição t de Student?	66
5.7 Calcular o intervalo de confiança	67
6 Prática - Correlação	69
6.1 Carregando os pacotes	69
6.2 Importando os dados	70
6.3 Pergunta norteadora	72
6.4 Gráfico de dispersão	72
6.5 Correlação	74
6.6 Matriz de correlação	76
6.6.1 Visualizando a matriz de correlação	77

6.7	EXTRA: Teste de significância	79
6.7.1	<code>statistic</code>	81
6.7.2	<code>parameter</code>	81
6.7.3	<code>p.value</code> ou valor-p	82
6.7.4	<code>estimate</code>	82
6.7.5	<code>null.value</code>	83
6.7.6	<code>alternative</code>	83
6.7.7	<code>method</code>	83
6.7.8	<code>data.name</code>	84
6.7.9	<code>conf.int</code>	84
6.8	Materiais complementares	84
	Apêndices	85
A	Erros e <i>warnings</i> frequentes	85
A.1	Instalação	85
A.1.1	<code>RTools</code>	85
A.2	Conceitos básicos	86
A.2.1	Instalando pacotes	86
A.2.2	Pacote não encontrado	87
A.2.3	Objeto não encontrado	87
A.2.4	Função não encontrada	88
B	Sugestão de vídeos, posts, e outros materiais	90
B.1	Estatística básica	90
B.2	Organização dos dados	90
B.3	Gráficos	91
B.4	Sobre os canais citados	91
B.4.1	Dr. Atila Iamarino	91
B.4.2	Dra. Fernanda Peres	91

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 - 08/06/2025	Introdução ao R e RStudio
2	09/06/2025 - 15/06/2025	Linguagem R
2	09/06/2025 - 15/06/2025	Análise exploratória de dados - Parte 1
3	16/06/2025 - 22/06/2025	Focar no trabalho da disciplina (semana com feriado)
4	23/06/2025 - 29/06/2025	Análise exploratória de dados - Parte 2 e Intervalo de Confiança
5	30/06/2025 - 06/07/2025	Correlação

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:

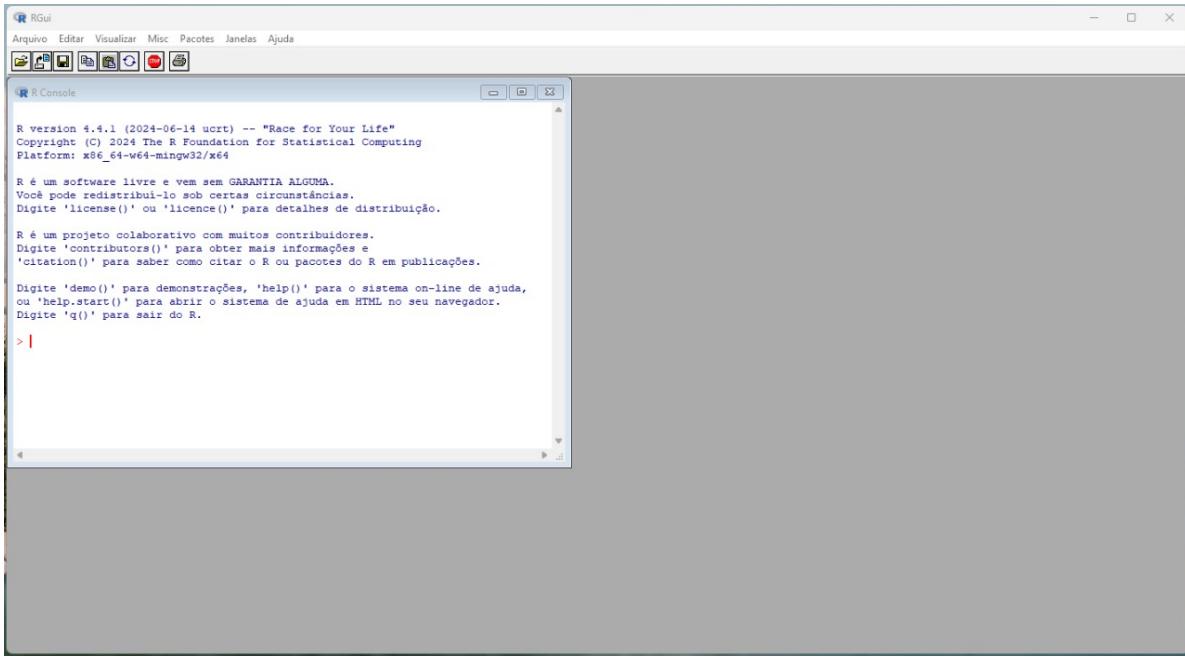


Figura 1.1: Captura de tela do R Console no Windows

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](#) e escolha o link de download de acordo com o seu sistema operacional:

The screenshot shows the CRAN homepage with a large 'R' logo at the top left. The main content area is titled 'The Comprehensive R Archive Network'. A section titled 'Download and Install R' contains a list of precompiled binary distributions for Linux, macOS, and Windows. Below this, a note says 'R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.' Another section, 'Source Code for all Platforms', provides instructions for Windows and Mac users on how to download precompiled binaries or source code. A final section, 'Questions About R', contains links for documentation, manuals, FAQs, and contributed packages.

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:

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: D0DDDD395](#) | 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.

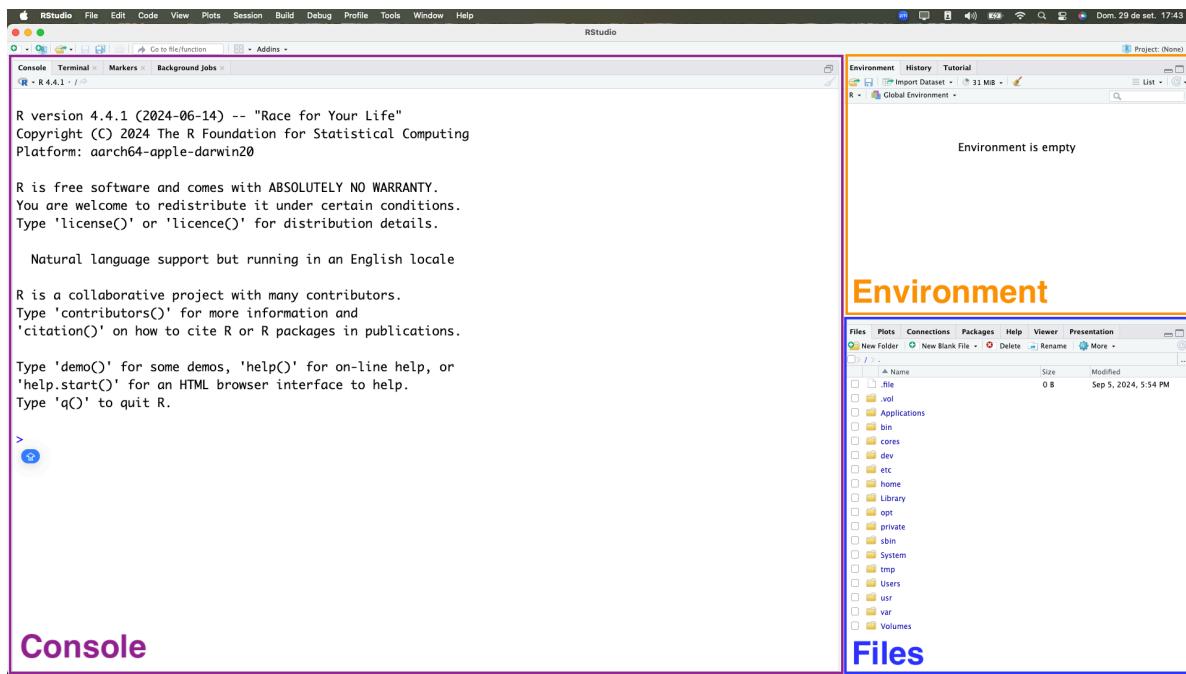
Dica

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, conhiceremos 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 deseja 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:

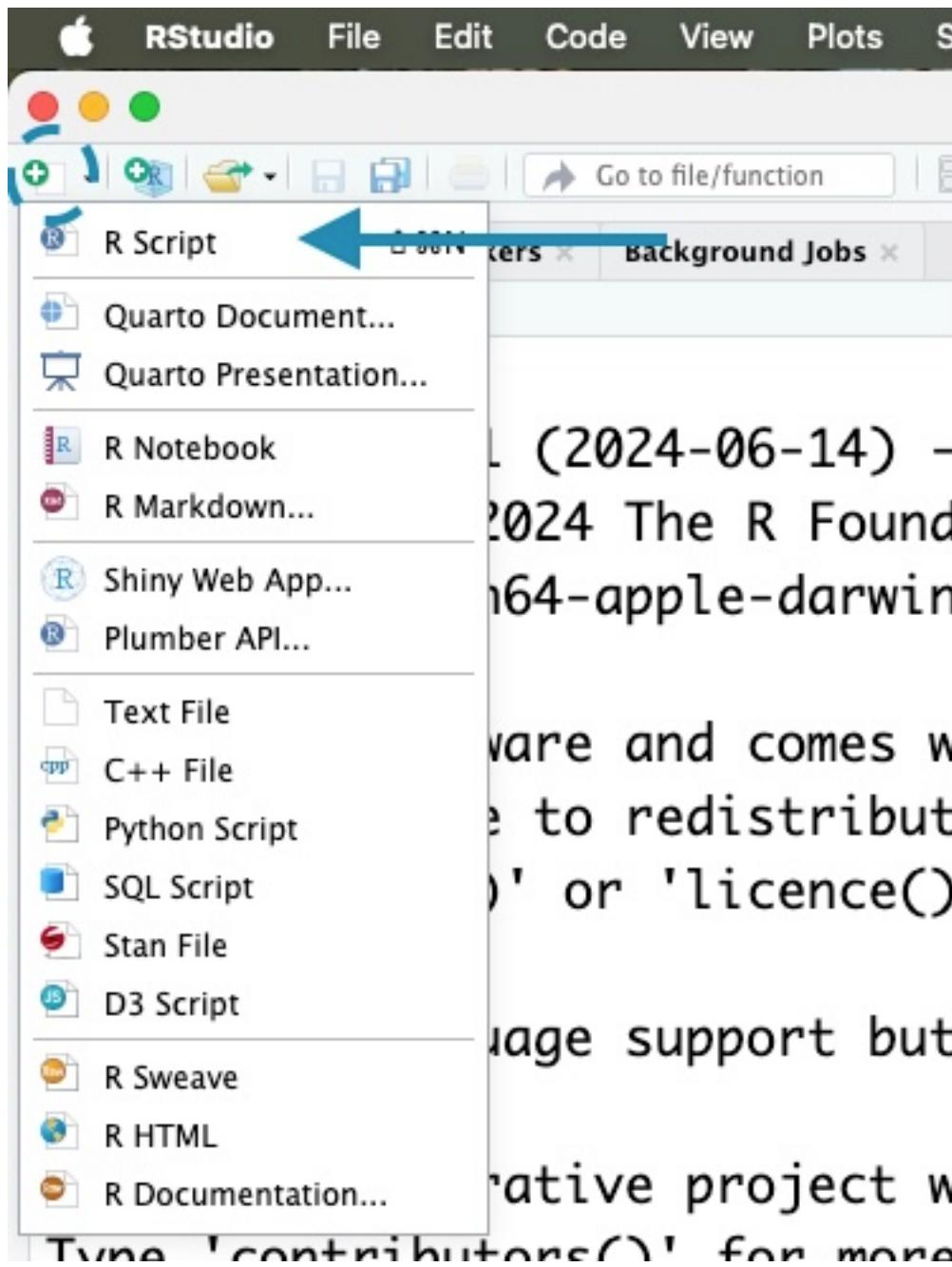


Figura 1.4: Captura de tela do RStudio: Opção para criar novo Script

Após abrir um script, o RStudio exibirá 4 painéis:

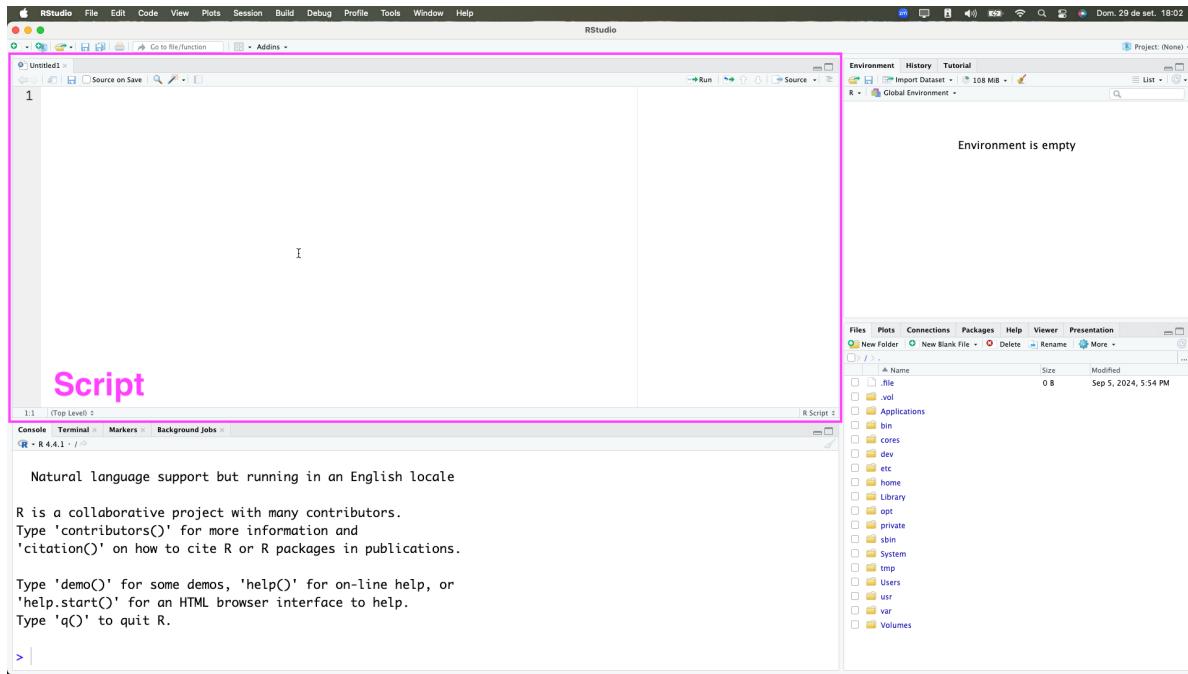


Figura 1.5: Captura de tela do RStudio

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] "2025-06-30"
```

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  
vforcats    1.0.0      v stringr    1.5.1  
v ggplot2   3.5.2      v tibble     3.3.0  
v lubridate 1.9.4      v tidyverse  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 becom
```

Precisamos carregar o pacote sempre que abrirmos um novo script, ou quando reiniciamos o RStudio. Uma prática 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 Reppositório 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 reproduzibilidade.

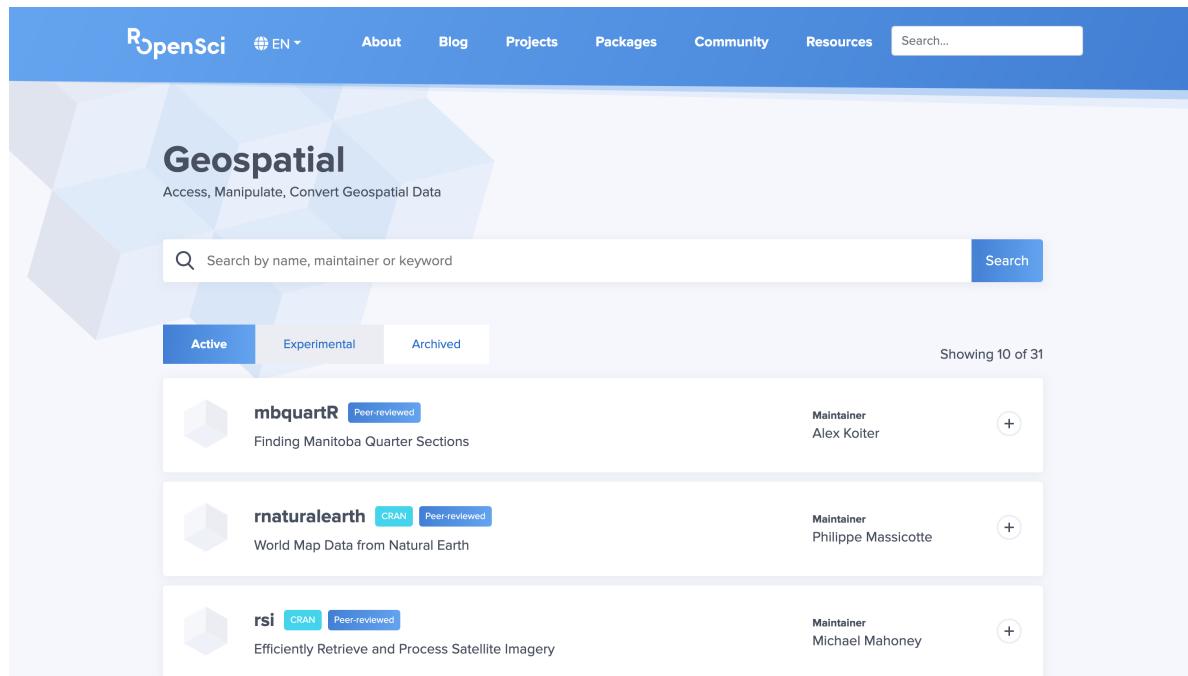


Figura 1.6: Captura de tela da página da rOpenSci: página de pacotes no tema Geoespacial

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.

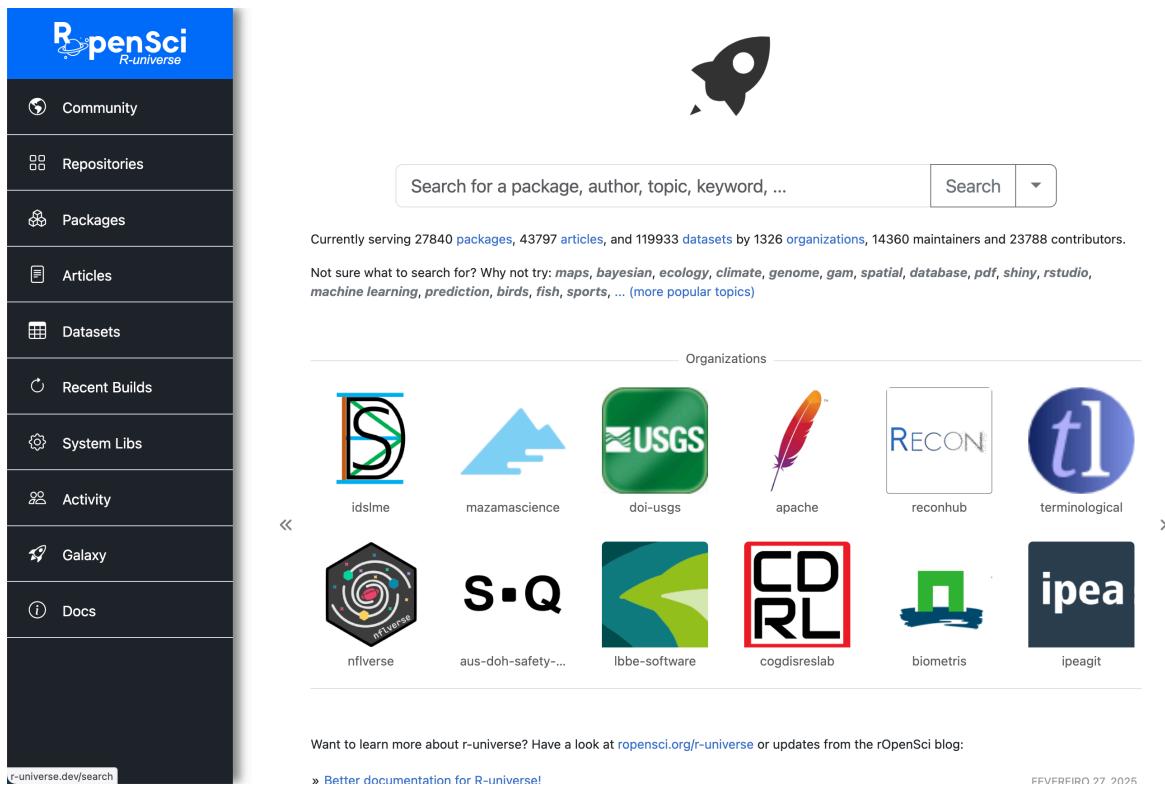


Figura 1.7: Captura de tela da página do R-Universe

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:

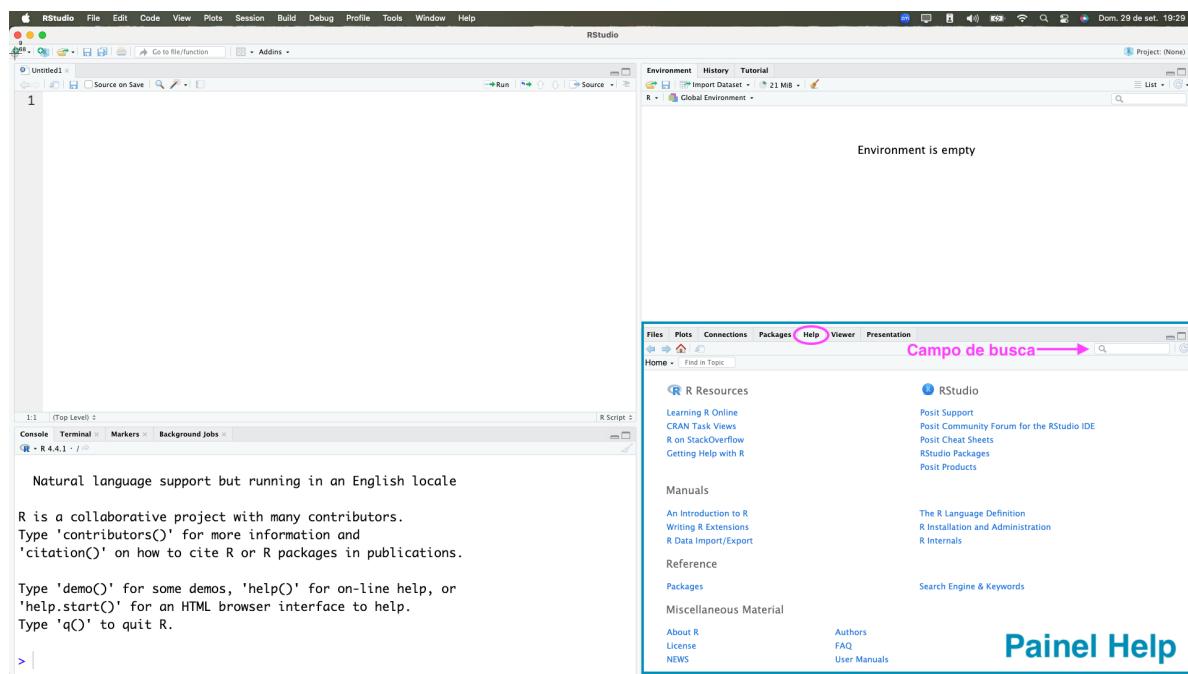


Figura 1.8: Captura de tela do RStudio: Painel Help

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:

dplyr 1.1.4 Get started Reference Articles ▾ News ▾

Search for



Function reference



Data frame verbs

Rows

Verbs that principally operate on rows.

`arrange()`

Order rows using column values

`distinct()`

Keep distinct/unique rows

`filter()`

Keep rows that match a condition

`slice()` `slice_head()` `slice_tail()` `slice_min()` `slice_max()`

`slice_sample()`

Subset rows using their positions

Columns

Verbs that principally operate on columns.

ON THIS PAGE

Data frame verbs

Rows

Columns

Groups

Data frames

Multiple columns

Vector functions

Built in datasets

Grouping helpers

Superseded

Remote tables

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

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

dplyr 1.1.4 Get started Reference Articles ▾ News ▾

Search for

Articles



Get started

Introduction to dplyr

Start here if this is your first time using dplyr. You'll learn the basic philosophy, the most important data manipulation verbs, and the pipe, `%>%`, which allows you to combine multiple verbs together to solve real problems.

Grouped data

To unlock the full potential of dplyr, you need to understand how each verb interacts with grouping. This vignette shows you how to manipulate grouping, how each verb changes its behaviour when working with grouped data, and how you can access data about the "current" group from within a verb.

Two-table verbs

Most dplyr verbs work with a single data set, but most data analyses involve multiple datasets. This vignette introduces you to the dplyr verbs that work with more than one data set, and introduces to the mutating joins, filtering joins, and the set operations.

dplyr <-> base R

How does dplyr compare to base R? This vignette describes the main differences in philosophy, and shows the base R code most closely equivalent to each dplyr verb.

ON THIS PAGE

Get started

Automate

Other

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:



¹O arquivo original não apresenta um dicionário de dados, portanto as descrições foram elaboradas com base no conteúdo do arquivo e na documentação do ICTEM (Indicador de Coleta e Tratabilidade de Esgoto da População Urbana de Município) disponível no site da CETESB.

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

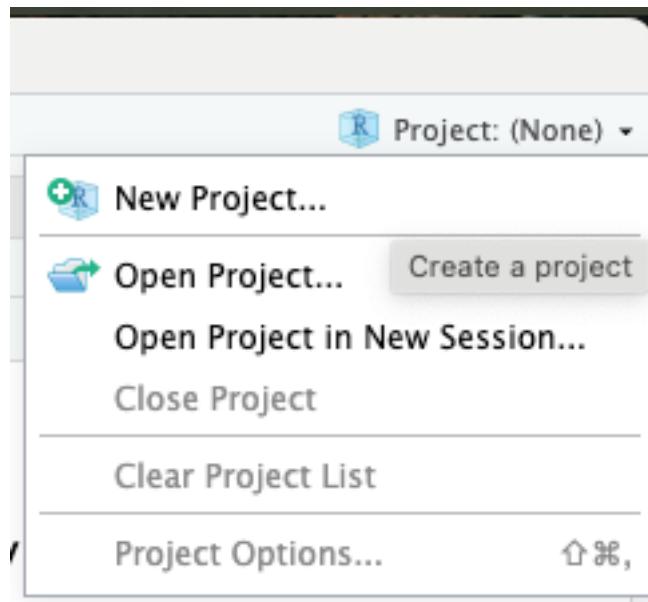


Figura 3.1: Captura de tela do RStudio: Menu de projetos

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

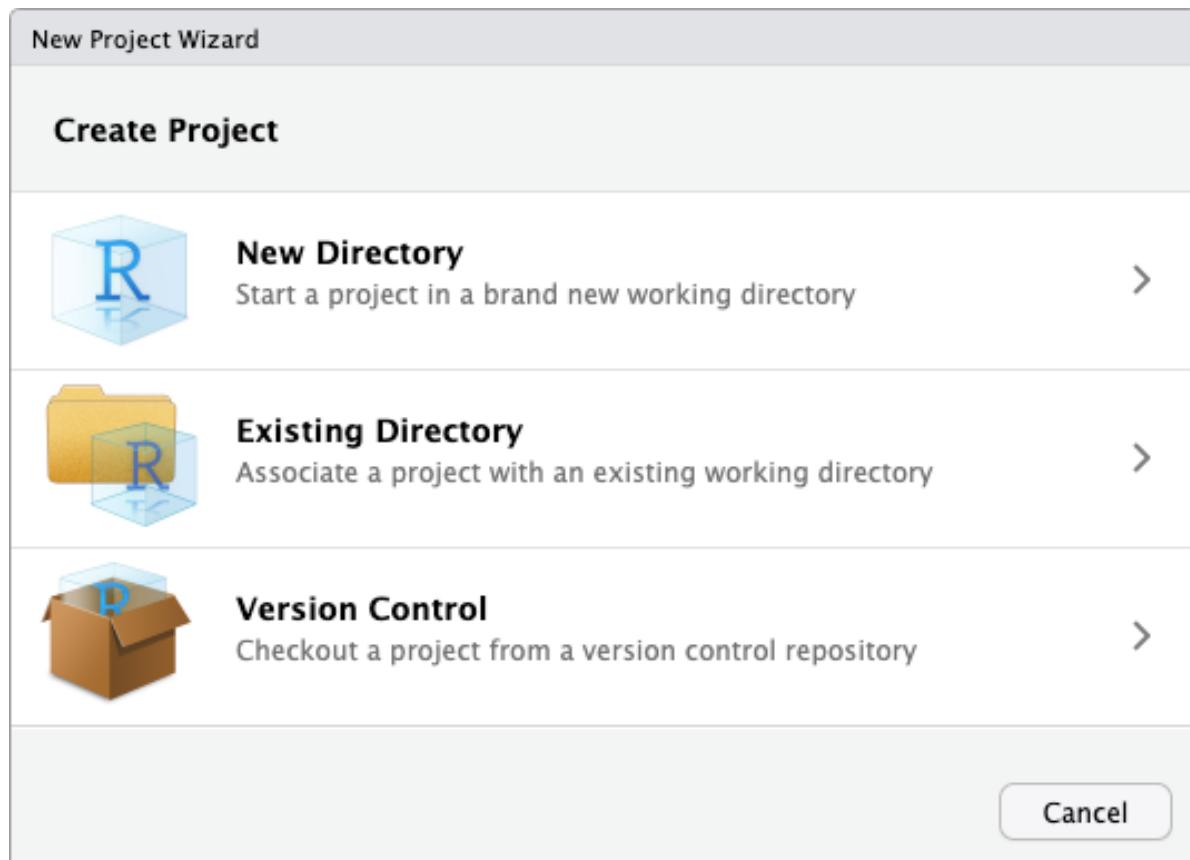


Figura 3.2: Captura de tela do RStudio: Criando um projeto

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:

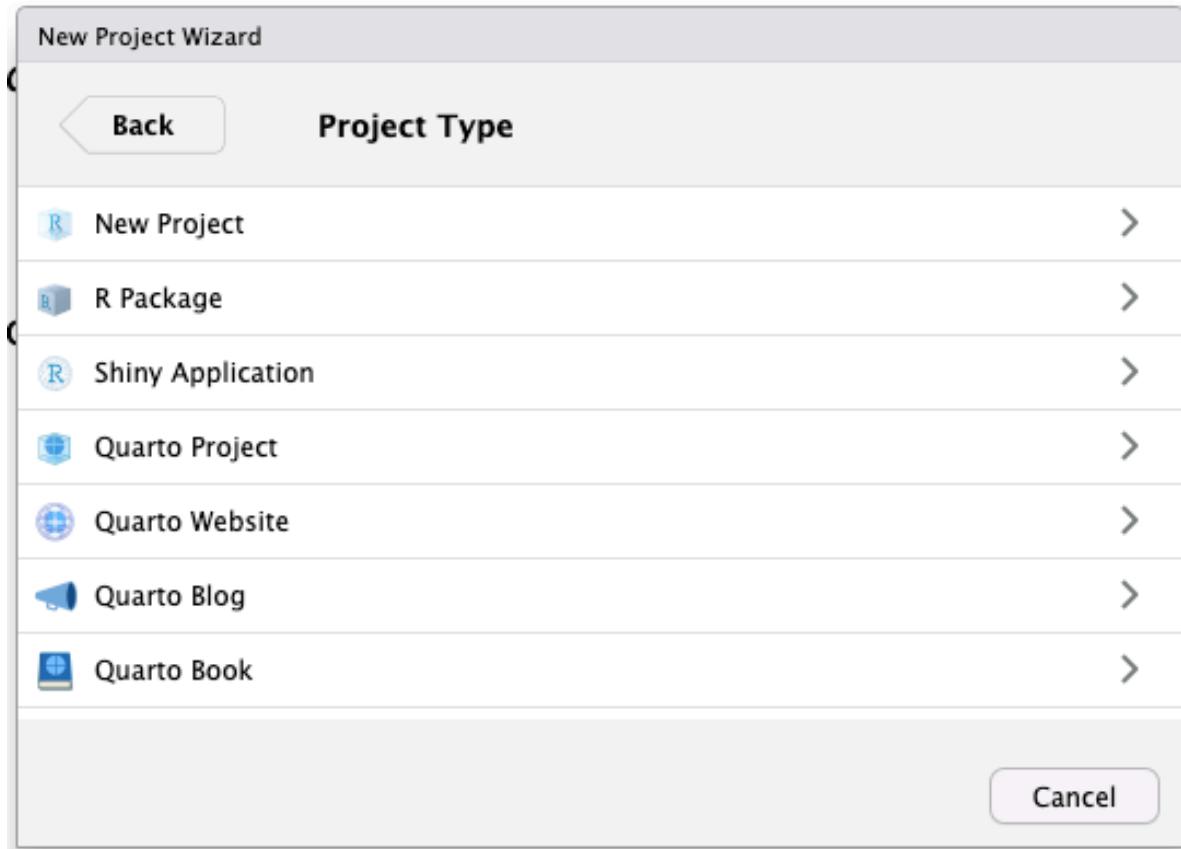


Figura 3.3: Captura de tela do RStudio: Escolhendo o tipo de projeto

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*):

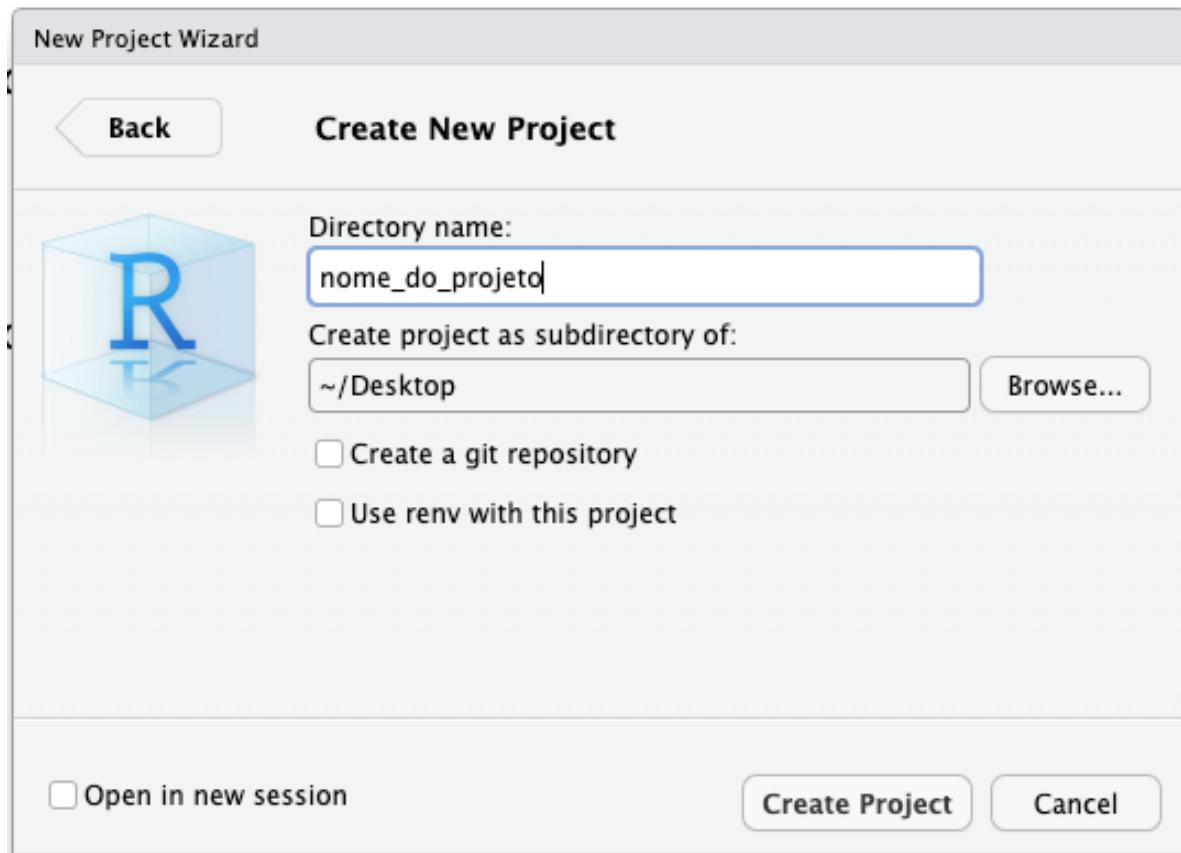


Figura 3.4: Captura de tela do RStudio: Nomeando o projeto

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

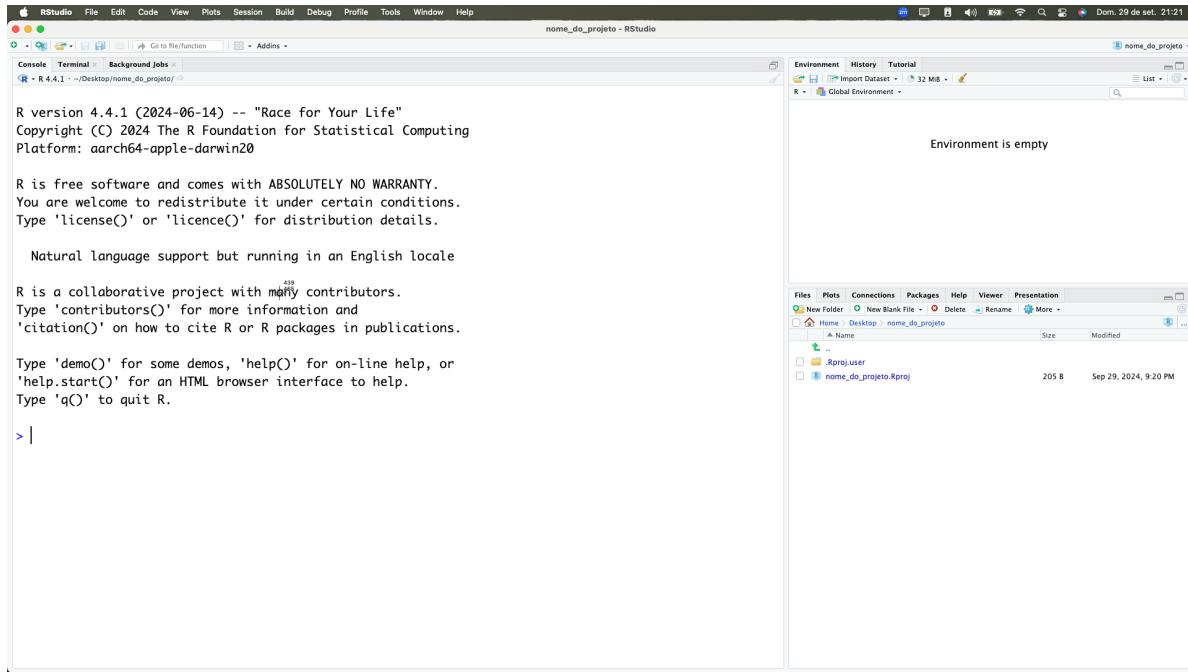


Figura 3.5: Captura de tela do RStudio: projeto criado

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    3.3.0
v lubridate  1.9.4     v tidyr    1.3.1
v purrr    1.0.4
-- 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
```

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~
$ uf                                <chr> "SP", "SP", "SP", "SP", "SP", "SP"~
$ ugrhi                             <dbl> 1, 1, 1, 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, ~
$ ictem                               <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 `nrow()` nos mostra o número de linhas da base de dados, e a função `ncol()` nos mostra o número de colunas:

```
nrow(dados_cetesb)
```

```
[1] 645
```

```
ncol(dados_cetesb)
```

```
[1] 12
```

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_remanescente" "ictem"
```

Segue abaixo uma tabela com os nomes das colunas e suas descrições²:

Nome da coluna	Descrição
ano	Ano de referência dos dados
uf	Unidade da Federação (estado)
ugrhi	Unidade de Gerenciamento de Recursos Hídricos
municipio	Nome do município
codigo_ibge	Código do município no IBGE
populacao_urbana	População urbana do município
atendimento_coleta_porc	Percentual de atendimento da coleta de esgoto
atendimento_tratamento_porc	Percentual de atendimento do tratamento de esgoto, para o esgoto coletado
eficiencia	Eficiência do sistema de esgoto
carga_poluidora_potencial	Carga poluidora potencial do esgoto coletado
carga_poluidora_remanescente	Carga poluidora remanescente do esgoto coletado
ictem	Indicador de Coleta e Tratabilidade de Esgoto da População Urbana de Município (ICTEM)

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

```
head(dados_cetesb)
```

²O arquivo original não apresenta um dicionário de dados, portanto as descrições foram elaboradas com base no conteúdo do arquivo e na documentação do ICTEM (Indicador de Coleta e Tratabilidade de Esgoto da População Urbana de Município) disponível no site da 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_porc <dbl>,
#   atendimento_tratamento_porc <dbl>, eficiencia <dbl>,
#   carga_poluidora_potencial <dbl>, carga_poluidora_remancescente <dbl>,
#   ictem <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_porc <dbl>,
#   atendimento_tratamento_porc <dbl>, eficiencia <dbl>,
#   carga_poluidora_potencial <dbl>, carga_poluidora_remancescente <dbl>,
#   ictem <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.

Também podemos calcular estatísticas descritivas específicas para colunas numéricas, como média, mediana, desvio padrão, entre outras.

função	descrição
<code>sum()</code>	calcula a soma dos valores
<code>mean()</code>	calcula a média
<code>median()</code>	calcula a mediana
<code>var()</code>	calcula a variância

função	descrição
<code>sd()</code>	calcula o desvio padrão
<code>min()</code>	calcula o valor mínimo
<code>max()</code>	calcula o valor máximo
<code>quantile()</code>	calcula os quantis (percentis)

! O operador \$

Quando estamos trabalhando com *data.frames* (ou *tibbles*) podemos utilizar o operador `$` para acessar uma coluna específica. O resultado será um vetor com os valores dessa coluna.

Por exemplo, para acessar a coluna `atendimento_coleta_porc`, podemos fazer:

```
dados_cetesb$atendimento_coleta_porc
```

```
[1] 52.6 46.7 98.0 70.0 95.6 100.0 88.1 100.0 100.0 84.6 74.3 90.0
[13] 60.4 98.2 30.7 98.2 100.0 83.5 73.2 99.6 84.4 96.0 68.0 100.0
[25] 84.3 85.0 77.2 44.3 93.6 64.0 57.3 84.7 94.4 88.3 95.9 100.0
[37] 98.6 78.6 40.5 44.6 34.7 100.0 99.0 100.0 96.4 100.0 91.4 99.0
[49] 85.2 88.5 100.0 99.0 99.5 99.2 100.0 96.2 100.0 96.9 100.0 92.8
[61] 100.0 100.0 91.7 100.0 96.5 95.7 95.0 96.0 98.0 67.8 20.0 89.6
[73] 96.3 60.2 95.0 86.9 100.0 100.0 98.0 94.6 100.0 95.2 97.7 90.5
[85] 100.0 89.7 74.2 98.0 29.7 65.2 99.5 100.0 86.6 100.0 95.0 90.7
[97] 89.9 12.8 96.2 94.9 70.6 99.0 96.0 52.9 100.0 100.0 99.8 99.0
[109] 100.0 98.0 99.0 100.0 100.0 100.0 90.0 95.0 50.0 94.3 46.4 84.2
[121] 91.5 71.5 83.5 53.1 74.0 74.4 69.8 54.0 97.5 72.3 40.5 82.7
[133] 58.2 74.0 86.4 52.0 64.5 63.0 74.1 25.8 93.0 60.7 77.8 51.0
[145] 96.8 71.2 50.2 77.3 46.9 99.5 92.3 100.0 89.5 94.5 89.5 66.0
[157] 51.3 69.4 49.3 82.9 81.2 80.0 96.6 79.4 100.0 99.0 99.7 97.0
[169] 100.0 100.0 100.0 96.9 100.0 98.8 100.0 100.0 87.5 100.0 100.0 100.0
[181] 100.0 87.1 97.8 98.3 100.0 95.9 98.1 98.8 99.0 100.0 100.0 100.0
[193] 97.0 99.1 100.0 100.0 93.7 100.0 99.8 100.0 100.0 100.0 100.0 70.0
[205] 100.0 99.2 96.0 100.0 100.0 100.0 98.9 96.4 100.0 100.0 97.0 100.0
[217] 100.0 96.3 92.4 100.0 78.3 100.0 77.5 100.0 56.2 70.9 86.9 42.1
[229] 40.0 89.0 88.3 100.0 66.2 75.6 98.0 100.0 82.1 49.6 90.2 100.0
[241] 90.0 96.4 77.0 100.0 73.8 79.7 99.9 69.2 98.9 48.5 58.0 99.0
[253] 97.2 86.5 89.2 35.4 99.0 61.9 87.6 73.9 86.7 83.3 97.9 60.3
[265] 44.5 67.9 61.5 82.1 38.3 100.0 83.0 16.6 86.6 93.9 58.3 99.6
[277] 62.7 31.6 75.8 79.5 89.3 100.0 99.1 100.0 100.0 100.0 99.5 90.4
[289] 100.0 100.0 93.1 100.0 100.0 99.7 77.0 99.2 98.8 99.0 98.3 100.0
[301] 100.0 100.0 100.0 100.0 98.5 100.0 100.0 93.0 100.0 95.0 100.0 100.0
```

[313]	95.0	87.5	100.0	100.0	100.0	97.0	100.0	98.8	96.0	100.0	100.0	99.1
[325]	98.5	100.0	86.6	83.3	99.1	100.0	80.6	97.2	83.9	94.9	97.9	100.0
[337]	78.9	85.8	99.0	88.1	83.5	95.9	87.8	89.5	91.2	90.0	64.1	70.3
[349]	73.2	100.0	81.0	100.0	97.5	70.1	96.0	100.0	100.0	75.8	92.0	94.5
[361]	100.0	99.0	100.0	100.0	99.9	100.0	100.0	100.0	99.1	100.0	100.0	99.5
[373]	100.0	100.0	100.0	100.0	100.0	80.0	100.0	93.4	100.0	98.7	100.0	99.1
[385]	98.6	100.0	80.0	100.0	99.8	97.7	100.0	100.0	100.0	92.1	100.0	98.5
[397]	80.7	100.0	100.0	80.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	99.0
[409]	100.0	100.0	71.0	100.0	100.0	100.0	100.0	100.0	98.2	100.0	100.0	100.0
[421]	100.0	100.0	100.0	99.0	100.0	91.4	100.0	100.0	99.0	100.0	100.0	100.0
[433]	100.0	100.0	98.9	100.0	83.4	100.0	99.0	100.0	87.3	100.0	100.0	100.0
[445]	100.0	98.0	100.0	100.0	90.0	100.0	100.0	98.6	67.0	100.0	100.0	83.3
[457]	100.0	73.2	99.0	100.0	99.8	99.0	100.0	99.6	92.0	90.0	99.0	100.0
[469]	100.0	100.0	94.1	100.0	93.2	100.0	100.0	100.0	83.2	100.0	100.0	92.3
[481]	98.1	93.1	100.0	99.3	100.0	100.0	70.5	95.3	100.0	87.5	100.0	95.3
[493]	99.0	93.9	70.0	100.0	100.0	100.0	100.0	100.0	96.6	100.0	100.0	100.0
[505]	98.8	91.5	100.0	100.0	100.0	95.0	100.0	100.0	90.1	79.7	100.0	100.0
[517]	100.0	100.0	100.0	100.0	100.0	97.0	98.0	100.0	96.5	98.6	98.5	100.0
[529]	99.0	92.4	100.0	98.0	100.0	89.2	100.0	100.0	94.9	82.1	85.4	100.0
[541]	100.0	80.0	99.0	100.0	100.0	99.0	100.0	90.0	100.0	100.0	92.5	100.0
[553]	76.9	100.0	99.0	94.0	100.0	99.0	100.0	100.0	100.0	100.0	100.0	100.0
[565]	99.8	95.9	99.2	98.0	100.0	93.1	100.0	99.5	80.0	99.0	96.3	97.6
[577]	100.0	97.9	86.4	99.0	100.0	100.0	72.9	92.8	100.0	60.0	97.1	95.0
[589]	100.0	100.0	99.5	88.0	96.2	88.7	100.0	100.0	100.0	98.0	100.0	99.0
[601]	100.0	99.0	100.0	90.6	98.7	89.4	100.0	98.5	97.0	100.0	100.0	100.0
[613]	97.0	80.0	100.0	100.0	95.1	100.0	95.0	97.0	100.0	96.8	91.7	97.5
[625]	100.0	100.0	100.0	86.6	95.0	97.0	84.1	100.0	100.0	94.5	81.6	93.1
[637]	100.0	98.0	100.0	79.5	87.7	97.8	100.0	88.1	100.0			

Vamos ver alguns exemplos de como calcular estatísticas descritivas utilizando as funções apresentadas acima:

```
# Soma dos valores da coluna `populacao_urbana`  
# Ou seja, a população urbana total do estado de São Paulo  
sum(dados_cetesb$populacao_urbana)
```

[1] 44767107

```
# Média dos valores da coluna `ictem`  
mean(dados_cetesb$ictem)
```

[1] 7.759767

```
# Mediana dos valores da coluna `ictem`  
median(dados_cetesb$ictem)
```

```
[1] 8.21
```

```
# Variância dos valores da coluna `ictem`  
var(dados_cetesb$ictem)
```

```
[1] 6.09937
```

```
# Desvio padrão dos valores da coluna `ictem`  
sd(dados_cetesb$ictem)
```

```
[1] 2.46969
```

```
# Valor mínimo da coluna `ictem`  
min(dados_cetesb$ictem)
```

```
[1] 0.75
```

```
# Valor máximo da coluna `ictem`  
max(dados_cetesb$ictem)
```

```
[1] 10
```

```
# Quantis dos valores da coluna `ictem`  
quantile(dados_cetesb$ictem)
```

```
0%    25%    50%    75%   100%  
0.75  6.78  8.21  9.94 10.00
```

Atenção aos valores faltantes (NA): Por padrão, as funções acima consideram os valores faltantes (NA) no cálculo, e portanto, o resultado também será um valor faltante (NA). Veja esse exemplo:

```
mean(dados_cetesb$eficiencia)
```

```
[1] NA
```

Se você quiser remover os valores faltantes, pode utilizar o argumento `na.rm = TRUE` nas funções. Por exemplo:

```
mean(dados_cetesb$eficiencia, na.rm = TRUE)
```

```
[1] 77.03656
```

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:

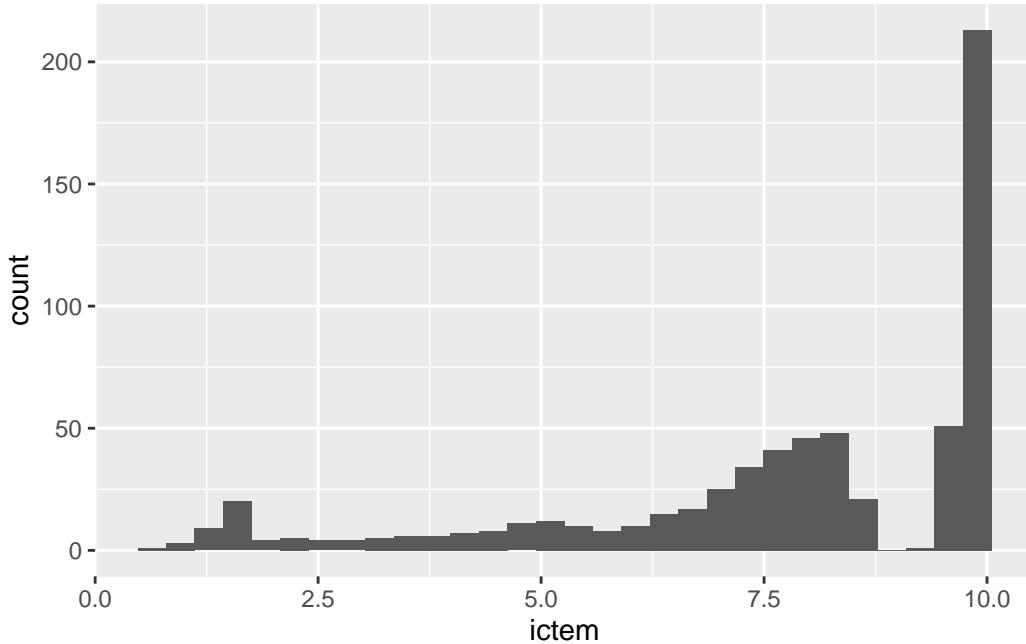
```
library(esquisse)
esquisser(dados_cetesb)
```

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:

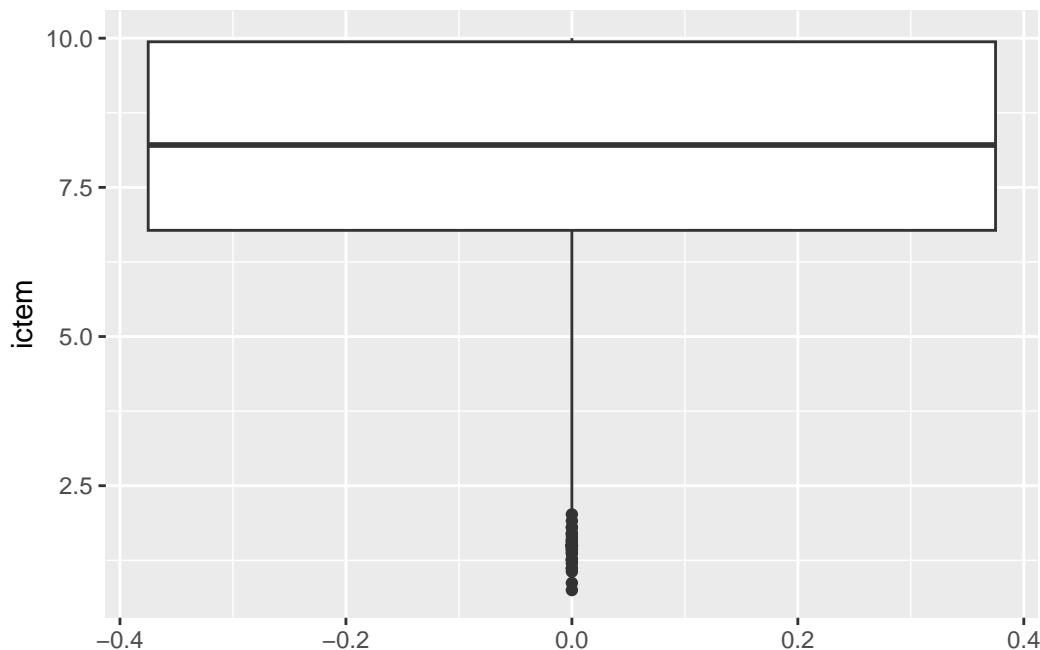
```
# inicia o gráfico com os dados `dados_cetesb`
ggplot(dados_cetesb) +
  # define a variável que será plotada no eixo x
  aes(x = ictem) +
  # adiciona a geometria de histograma
  geom_histogram()
```

```
`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 = ictem) +  
  # 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 Análise exploratória de dados - Parte 2

Nesta segunda parte da análise exploratória de dados, vamos conhecer as funções mais importantes do pacote `dplyr` para manipulação de dados.

4.1 Carregando pacotes

```
library(tidyverse)
```

4.2 Carregando os dados

Vamos continuar trabalhando nos dados da CETESB, que já foram carregados na [primeira parte da análise exploratória \(aula passada\)](#). Caso você não tenha feito isso, recomendo que você siga as instruções da aula anterior antes de continuar a aula de hoje.

```
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.
```

Outra forma de importar os dados é utilizando o link direto do repositório do GitHub onde os dados estão armazenados. Você pode fazer isso utilizando a função `read_csv()` do pacote `readr`:

```
dados_cetesb <- read_csv("https://raw.githubusercontent.com/beatrizmilz/ESHT011-21-analise-d
```

A função `glimpse()` do pacote `dplyr`, que conhecemos na aula passada, é uma forma rápida de visualizar a estrutura dos dados. Ela nos mostra o nome das colunas, o tipo de dado de cada coluna e os primeiros valores de cada coluna:

```
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~
$ 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, ~
$ ictem                      <dbl> 5.97, 4.63, 9.97, 1.55, 9.86, 1.50, 7.61~
```

4.3 Conhecendo o operador pipe (|>)

O operador pipe (`|>` ou `%>%`) permite encadear operações de forma mais legível e intuitiva.

Por exemplo, podemos utilizar o operador pipe para aplicar a função `glimpse()` diretamente nos dados:

```
dados_cetesb |>
  glimpse()
```

```
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~
$ 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, ~
$ ictem                     <dbl> 5.97, 4.63, 9.97, 1.55, 9.86, 1.50, 7.61-

```

Com apenas uma função, não é tão óbvio o benefício do operador pipe. No entanto, quando começamos a encadear várias funções, ele se torna muito útil. Veremos exemplos disso ao longo desta aula!

4.4 Principais funções do dplyr

O pacote `dplyr` é uma das ferramentas mais poderosas para manipulação de dados no R. Ele oferece uma série de funções que facilitam a transformação e análise de dados. Vamos conhecer algumas das principais funções do `dplyr`:

- `filter()`: filtra linhas com base em condições específicas.
- `select()`: seleciona colunas específicas de um data frame.
- `mutate()`: adiciona ou modifica colunas.
- `arrange()`: ordena as linhas de um data frame com base em uma ou mais colunas.
- `summarise()`: resume os dados, calculando estatísticas agregadas.
- `group_by()`: agrupa os dados com base em uma ou mais colunas, permitindo aplicar funções de resumo a cada grupo.

Ao apresentar essas funções, não vamos abordar todos os casos de uso, mas sim o básico de cada uma delas. Você pode consultar os materiais listados no final da aula para aprender mais sobre cada função e suas possibilidades.

4.5 Filtrando dados com `filter()`

A A função `filter()` é utilizada para filtrar linhas de um data frame com base em condições específicas.

Por exemplo, podemos filtrar os dados da CETESB para mostrar apenas os municípios que possuem uma população urbana maior que 1 milhão de habitantes:

```

dados_cetesb |>
  filter(populacao_urbana > 1000000)

```

```

# A tibble: 3 x 12
  ano uf    ugrhi municipio codigo_ibge populacao_urbana
  <dbl> <chr> <dbl> <chr>           <dbl>           <dbl>
1 2022 SP      5 Campinas     3509502       1202203
2 2022 SP      6 Guarulhos   3518800       1404694
3 2022 SP      6 São Paulo   3550308       12284940
# i 6 more variables: atendimento_coleta_porc <dbl>,
#   atendimento_tratamento_porc <dbl>, eficiencia <dbl>,
#   carga_poluidora_potencial <dbl>, carga_poluidora_remancescente <dbl>,
#   ictem <dbl>

```

Podemos também filtrar as linhas dos municípios do Grande ABC. Neste caso, é mais fácil criar um vetor com os nomes dos municípios, e utilizar o operador `%in%` para filtrar os dados:

```

municipios_grande_abc <- c("São Caetano do Sul", "São Bernardo do Campo", "Diadema", "Santo
dados_cetesb |>
  filter(municipio %in% municipios_grande_abc)

```

```

# A tibble: 7 x 12
  ano uf    ugrhi municipio           codigo_ibge populacao_urbana
  <dbl> <chr> <dbl> <chr>           <dbl>           <dbl>
1 2022 SP      6 Diadema        3513801       429550
2 2022 SP      6 Mauá          3529401       481725
3 2022 SP      6 Ribeirão Pires 3543303       125238
4 2022 SP      6 Rio Grande da Serra 3544103       52009
5 2022 SP      6 Santo André     3547809       723889
6 2022 SP      6 São Bernardo do Campo 3548708       835657
7 2022 SP      6 São Caetano do Sul    3548807       162763
# i 6 more variables: atendimento_coleta_porc <dbl>,
#   atendimento_tratamento_porc <dbl>, eficiencia <dbl>,
#   carga_poluidora_potencial <dbl>, carga_poluidora_remancescente <dbl>,
#   ictem <dbl>

```

Podemos também consultar quais são os municípios que coletam 100% do esgoto produzido:

```

dados_cetesb |>
  filter(atendimento_coleta_porc == 100)

# A tibble: 253 x 12
  ano uf  ugrhi municipio      codigo_ibge populacao_urbana
  <dbl> <chr> <dbl> <chr>          <dbl>                <dbl>
1 2022 SP    2 Areias        3503505            2619
2 2022 SP    2 Caçapava     3508504            81929
3 2022 SP    2 Cachoeira Paulista 3508603            27623
4 2022 SP    2 Jambeiro     3524907            3269
5 2022 SP    2 Pindamonhangaba 3538006            165703
6 2022 SP    2 Taubaté       3554102            313898
7 2022 SP    4 Altinópolis   3501004            14172
8 2022 SP    4 Caconde       3508702            12975
9 2022 SP    4 Casa Branca   3510807            25075
10 2022 SP   4 Jardinópolis  3525102            43706
# i 243 more rows
# i 6 more variables: atendimento_coleta_porc <dbl>,
#   atendimento_tratamento_porc <dbl>, eficiencia <dbl>,
#   carga_poluidora_potencial <dbl>, carga_poluidora_remancescente <dbl>,
#   ictem <dbl>

```

Destes, podemos filtrar os que tratem 100% do esgoto coletado:

```

dados_cetesb |>
  filter(atendimento_coleta_porc == 100, atendimento_tratamento_porc == 100)

# A tibble: 218 x 12
  ano uf  ugrhi municipio      codigo_ibge populacao_urbana
  <dbl> <chr> <dbl> <chr>          <dbl>                <dbl>
1 2022 SP    2 Cachoeira Paulista 3508603            27623
2 2022 SP    2 Jambeiro       3524907            3269
3 2022 SP    2 Pindamonhangaba 3538006            165703
4 2022 SP    2 Taubaté         3554102            313898
5 2022 SP    4 Altinópolis    3501004            14172
6 2022 SP    4 Santa Cruz da Esperança 3546256            1467
7 2022 SP    4 Serrana        3551504            45677
8 2022 SP    4 Tambaú         3553302            20663
9 2022 SP    5 Corumbataí    3512704            2200
10 2022 SP   5 Holambra       3519055            11303
# i 208 more rows

```

```
# i 6 more variables: atendimento_coleta_porc <dbl>,
#   atendimento_tratamento_porc <dbl>, eficiencia <dbl>,
#   carga_poluidora_potencial <dbl>, carga_poluidora_remanescente <dbl>,
#   ictem <dbl>
```

As possibilidades são muitas! Podemos combinar várias condições utilizando os operadores lógicos (como o | (OU) e o ! (NEGAÇÃO)).

4.6 Selecionando colunas com select()

A função `select()` é utilizada para selecionar colunas específicas de um data frame.

Por exemplo, podemos selecionar apenas as colunas `municipio`, `populacao_urbana` e `atendimento_coleta_porc` dos dados da CETESB:

```
dados_cetesb |>
  select(municipio, populacao_urbana, atendimento_coleta_porc)
```

```
# A tibble: 645 x 3
  municipio      populacao_urbana atendimento_coleta_porc
  <chr>                <dbl>                  <dbl>
1 Campos do Jordão        52384                 52.6
2 Santo Antônio do Pinhal    4067                  46.7
3 São Bento do Sapucaí       5251                  98 
4 Aparecida                  35684                 70 
5 Araçatuba                   1844                 95.6
6 Areias                      2619                  100
7 Bananal                     8808                 88.1
8 Caçapava                    81929                 100
9 Cachoeira Paulista          27623                 100
10 Canas                      4890                  84.6
# i 635 more rows
```

Podemos também indicar quais colunas queremos excluir, utilizando o operador - antes do nome da coluna. Por exemplo, para excluir as colunas `ano` e `uf`, podemos fazer o seguinte:

```
dados_cetesb |>
  select(-ano, -uf)
```

```

# A tibble: 645 x 10
  ugrhi municipio      codigo_ibge populacao_urbana atendimento_coleta_p~1
  <dbl> <chr>          <dbl>           <dbl>           <dbl>
1     1 Campos do Jordão    3509700        52384        52.6
2     1 Santo Antônio do P~  3548203        4067         46.7
3     1 São Bento do Sapuc~  3548609        5251          98
4     2 Aparecida          3502507        35684         70
5     2 Arapeí             3503158        1844         95.6
6     2 Areias              3503505        2619         100
7     2 Bananal            3504909        8808         88.1
8     2 Caçapava           3508504        81929         100
9     2 Cachoeira Paulista 3508603        27623         100
10    2 Canas              3509957        4890         84.6
# i 635 more rows
# i abbreviated name: 1: atendimento_coleta_porc
# i 5 more variables: atendimento_tratamento_porc <dbl>, eficiencia <dbl>,
#   carga_poluidora_potencial <dbl>, carga_poluidora_remancescente <dbl>,
#   ictem <dbl>

```

4.7 Adicionando ou modificando colunas com `mutate()`

A função `mutate()` é utilizada para adicionar novas colunas ou modificar colunas existentes em um data frame.

A sintaxe básica da função `mutate()` é:

```

base_de_dados |>
  mutate(nome_da_nova_coluna = o_que_queremos_que_seja_salvo_nela)

```

Por exemplo, podemos criar uma coluna nova chamada `populacao_urbana_mil` que representa a população urbana em mil habitantes:

```

dados_cetesb |>
  # selecionando as colunas municipio e populacao_urbana
  # para facilitar a visualização
  select(municipio, populacao_urbana) |>
  mutate(populacao_urbana_mil = populacao_urbana / 1000)

```

```

# A tibble: 645 x 3
  municipio      populacao_urbana populacao_urbana_mil
  <chr>           <dbl>           <dbl>
1 Campos do Jordão    3509700        3509.7
2 Santo Antônio do P~  3548203        3548.2
3 São Bento do Sapuc~  3548609        3548.6
4 Aparecida          3502507        3502.5
5 Arapeí             3503158        3503.2
6 Areias              3503505        3503.5
7 Bananal            3504909        3504.9
8 Caçapava           3508504        3508.5
9 Cachoeira Paulista 3508603        3508.6
10 Canas              3509957        3509.9
# i 635 more rows
# i abbreviated name: 1: populacao_urbana
# i 5 more variables: populacao_urbana_mil <dbl>, atendimento_coleta_porc <dbl>,
#   atendimento_tratamento_porc <dbl>, eficiencia <dbl>,
#   ictem <dbl>

```

```

1 Campos do Jordão           52384      52.4
2 Santo Antônio do Pinhal    4067       4.07
3 São Bento do Sapucaí       5251       5.25
4 Aparecida                  35684      35.7
5 Arapeí                      1844      1.84
6 Areias                      2619      2.62
7 Bananal                     8808      8.81
8 Caçapava                    81929     81.9
9 Cachoeira Paulista         27623      27.6
10 Canas                      4890      4.89
# i 635 more rows

```

A função `mutate()` também pode ser utilizada para modificar colunas existentes. Por exemplo, podemos mudar a classe de algumas colunas:

```

dados_cetesb |>
  mutate(
    ugrhi = as.character(ugrhi), # convertendo ugrhi para character
    codigo_ibge = as.character(codigo_ibge) # convertendo codigo_ibge para character
  )

# A tibble: 645 x 12
  ano uf     ugrhi municipio               codigo_ibge populacao_urbana
  <dbl> <chr>  <chr>   <chr>                <chr>          <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
7 2022 SP     2 Bananal                3504909      8808
8 2022 SP     2 Caçapava               3508504      81929
9 2022 SP     2 Cachoeira Paulista   3508603      27623
10 2022 SP    2 Canas                  3509957      4890
# i 635 more rows
# i 6 more variables: atendimento_coleta_porc <dbl>,
#   atendimento_tratamento_porc <dbl>, eficiencia <dbl>,
#   carga_poluidora_potencial <dbl>, carga_poluidora_remanescente <dbl>,
#   ictem <dbl>

```

4.8 Ordenando dados com arrange()

A função `arrange()` é utilizada para ordenar as linhas de um data frame com base em uma ou mais colunas.

Podemos ordenar os dados da CETESB pela coluna `carga_poluidora_remancescente`, em ordem crescente:

```
dados_cetesb |>
  arrange(carga_poluidora_remancescente)

# A tibble: 645 x 12
  ano uf    ugrhi municipio      codigo_ibge populacao_urbana
  <dbl> <chr> <dbl> <chr>          <dbl>                <dbl>
1 2022 SP      14 Ribeirão Grande   3543253            2427
2 2022 SP      13 Trabiju           3554755            1609
3 2022 SP      21 Borá              3507209            653 
4 2022 SP      15 Turmalina        3555307            1186
5 2022 SP      2 Jambeiro          3524907            3269
6 2022 SP      18 Santa Salete     3547650            882 
7 2022 SP      18 Santana da Ponte Pensa 3547205            968 
8 2022 SP      15 Mesópolis         3529658            1481
9 2022 SP      18 Nova Canaã Paulista 3532843            759 
10 2022 SP     19 Turiúba           3555208            1657
# i 635 more rows
# i 6 more variables: atendimento_coleta_porc <dbl>,
#   atendimento_tratamento_porc <dbl>, eficiencia <dbl>,
#   carga_poluidora_potencial <dbl>, carga_poluidora_remancescente <dbl>,
#   ictem <dbl>
```

Também podemos ordenar os dados em ordem decrescente, utilizando a função `desc()`:

```
dados_cetesb |>
  arrange(desc(populacao_urbana))

# A tibble: 645 x 12
  ano uf    ugrhi municipio      codigo_ibge populacao_urbana
  <dbl> <chr> <dbl> <chr>          <dbl>                <dbl>
1 2022 SP      6 São Paulo        3550308            12284940
2 2022 SP      6 Guarulhos       3518800            1404694 
3 2022 SP      5 Campinas         3509502            1202203
```

```

4 2022 SP      6 São Bernardo do Campo    3548708    835657
5 2022 SP      6 Santo André             3547809    723889
6 2022 SP      2 São José dos Campos   3549904    722310
7 2022 SP      4 Ribeirão Preto        3543402    718072
8 2022 SP      6 Osasco                 3534401    701428
9 2022 SP      10 Sorocaba              3552205    688252
10 2022 SP     6 Mauá                  3529401    481725
# i 635 more rows
# i 6 more variables: atendimento_coleta_porc <dbl>,
#   atendimento_tratamento_porc <dbl>, eficiencia <dbl>,
#   carga_poluidora_potencial <dbl>, carga_poluidora_remancescente <dbl>,
#   ictem <dbl>

```

4.9 Agrupando dados com group_by()

A função `group_by()` é utilizada para agrupar os dados com base em uma ou mais colunas. Isso é especialmente útil quando queremos aplicar funções de resumo a cada grupo.

Por exemplo, podemos agrupar os dados da CETESB pela coluna `ugrhi` (Unidade de Gerenciamento de Recursos Hídricos) e calcular a soma da população urbana para cada UGRHI:

```

dados_cetesb |>
  group_by(ugrhi)

# A tibble: 645 x 12
# Groups:   ugrhi [22]
  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
7 2022 SP      2 Bananal             3504909    8808
8 2022 SP      2 Caçapava            3508504    81929
9 2022 SP      2 Cachoeira Paulista 3508603    27623
10 2022 SP     2 Canas               3509957    4890
# i 635 more rows
# i 6 more variables: atendimento_coleta_porc <dbl>,

```

```
# atendimento_tratamento_porc <dbl>, eficiencia <dbl>,
# carga_poluidora_potencial <dbl>, carga_poluidora_remanescente <dbl>,
# ictem <dbl>
```

A função `group_by()` não altera os dados, mas prepara o data frame para que possamos aplicar funções de resumo a cada grupo. Portanto, ela é frequentemente utilizada em conjunto com a função `summarise()`.

4.10 Resumindo dados com `summarise()`

A função `summarise()` é utilizada para resumir os dados, calculando estatísticas agregadas. Ela é frequentemente utilizada em conjunto com a função `group_by()`.

Como vimos na aula anterior, 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()`.

```
estatisticas_descritivas <- 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_ictem = mean(ictem) |> round(2),
    mediana_ictem = median(ictem) |> round(2),
    desvio_padrao_ictem = sd(ictem) |> round(2)
  )

knitr::kable(estatisticas_descritivas)
```

ugrhi	quantidade_municipios	soma_populacao_urbana	media_ictem	mediana_ictem	desvio_padrao_ictem
1	3	61702	6.86	5.97	2.78
2	34	2124413	5.87	6.98	3.30
3	4	337159	5.10	4.66	1.74
4	23	1215586	7.09	8.12	2.83

ugrhi	quantidade_municipios	media_populacao_urbana	media_icteemediana	media_icterdesvio_padrão_ictem	
5	57	5737151	7.10	7.85	2.99
6	34	21626154	4.93	5.03	2.39
7	9	1893370	5.01	5.09	1.22
8	22	707923	9.18	9.98	1.34
9	38	1542781	6.91	7.64	3.02
10	33	1937230	7.05	7.10	2.66
11	23	273446	7.10	7.09	1.87
12	12	346186	7.92	8.45	2.59
13	34	1608773	7.94	8.08	2.29
14	34	631467	8.01	8.20	1.84
15	64	1289129	8.77	9.89	1.66
16	33	520262	8.36	9.70	2.24
17	42	666615	8.88	9.75	1.23
18	25	212661	9.01	10.00	1.63
19	42	772089	8.31	8.15	1.54
20	32	348271	8.69	8.57	1.34
21	26	444578	8.96	9.70	1.80
22	21	470161	9.09	9.70	1.01

Podemos salvar o resultado em um arquivo CSV ou Excel, utilizando as funções `readr::write_csv()` ou `writexl::write_excel_csv()`:

```
write_csv(estatisticas_descritivas,
          "dados/estatisticas-descritivas-ugrhi.csv")

writexl::write_xlsx(estatisticas_descritivas,
                     "dados/estatisticas-descritivas-ugrhi.xlsx")
```

4.11 Unindo tabelas com `left_join()`

A função `left_join()` é utilizada para unir duas tabelas com base em uma ou mais colunas em comum (chamada de chave - `key`).

Para este exemplo, vamos importar [uma tabela](#) que contém informações sobre o IDH (Índice de Desenvolvimento Humano) dos municípios de São Paulo, referente ao ano de 2010. Esses dados são baseados no Censo, mas ainda não temos o IDH calculado para o ano de 2022.

```
dados_idhm <- read_csv("https://raw.githubusercontent.com/beatrizmilz/ESHT011-21-analise-dados/2022/dados_idhm.csv")
```

```

Rows: 645 Columns: 15
-- Column specification -----
Delimiter: ","
chr (3): muni_nm, uf_sigla, regiao_nm
dbl (12): ano, muni_id, idhm, idhm_e, idhm_l, idhm_r, espvida, rdpc, gini, p...
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.

```

Para fazer o join, precisamos garantir que as colunas que vamos utilizar como chave estejam no mesmo formato. Neste caso, vamos utilizar a coluna referente ao código do IBGE como chave para unir as duas tabelas:

```
names(dados_idhm)
```

```
[1] "ano"          "muni_id"      "muni_nm"      "uf_sigla"     "regiao_nm"    "idhm"
[7] "idhm_e"       "idhm_l"       "idhm_r"       "espvida"     "rdpc"        "gini"
[13] "pop"         "lat"          "lon"
```

```
names(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" "ictem"
```

É importante que a variável que usaremos como chave esteja no mesmo formato nas duas tabelas. Usamos a função `class()` para verificar o tipo de dado de cada coluna:

```
class(dados_cetesb$codigo_ibge)
```

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

```
class(dados_idhm$muni_id)
```

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

Obs: caso as colunas não estejam no mesmo formato, podemos utilizar a função `mutate()` para alterar o tipo de dado de uma das colunas, junto com funções como `as.character()` ou `as.numeric()`, dependendo da classe de dado que queremos.

Agora podemos unir as duas tabelas utilizando a função `left_join()`. Essa função irá manter todas as linhas da tabela da esquerda (`dados_cetesb`) e adicionar as colunas da tabela da direita (`dados_idhm`) onde houver correspondência na chave especificada.

```
dados_unidos <- left_join(  
  # tabela da esquerda  
  dados_cetesb,  
  # tabela da direita  
  dados_idhm,  
  # o argumento `by` especifica as colunas que serão utilizadas como chave para o join  
  by = c("codigo_ibge" = "muni_id")  
)
```

A nova tabela `dados_unidos` agora contém todas as colunas de ambas as tabelas, com os dados do IDH adicionados aos municípios correspondentes. Se um município não tiver um valor correspondente na tabela de IDH, as colunas relacionadas ao IDH terão valores NA.

```
glimpse(dados_unidos)
```

```
Rows: 645  
Columns: 26  
$ ano.x          <dbl> 2022, 2022, 2022, 2022, 2022, 2022, 2022~  
$ uf             <chr> "SP", "SP", "SP", "SP", "SP", "SP"~  
$ ugrhi          <dbl> 1, 1, 1, 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, ~  
$ ictem           <dbl> 5.97, 4.63, 9.97, 1.55, 9.86, 1.50, 7.61~  
$ ano.y           <dbl> 2010, 2010, 2010, 2010, 2010, 2010, 2010~  
$ muni_nm         <chr> "CAMPOS DO JORDÃO", "SANTO ANTÔNIO DO PI~  
$ uf_sigla        <chr> "SP", "SP", "SP", "SP", "SP", "SP"~  
$ regiao_nm       <chr> "Sudeste", "Sudeste", "Sudeste", "Sudest~  
$ idhm            <dbl> 0.749, 0.706, 0.720, 0.755, 0.680, 0.697~
```

```
$ idhm_e <dbl> 0.648, 0.632, 0.638, 0.706, 0.612, 0.672~  
$ idhm_l <dbl> 0.852, 0.812, 0.812, 0.828, 0.812, 0.803~  
$ idhm_r <dbl> 0.761, 0.685, 0.719, 0.735, 0.634, 0.627~  
$ espvida <dbl> 76.10, 73.69, 73.69, 74.67, 73.72, 73.18~  
$ rdpc <dbl> 911.40, 569.08, 701.89, 775.17, 414.19, ~  
$ gini <dbl> 0.59, 0.47, 0.55, 0.46, 0.39, 0.44, 0.51~  
$ pop <dbl> 47287, 6450, 10370, 34630, 2481, 3632, 1~  
$ lat <dbl> -22.739, -22.827, -22.689, -22.847, -22.~  
$ lon <dbl> -45.591, -45.663, -45.731, -45.230, -44.~
```

Podemos salvar os dados unidos em um arquivo CSV ou Excel, utilizando as funções `readr::write_csv()` ou `writexl::write_excel_csv()`:

```
write_csv(dados_unidos,  
          "dados/dados-join-cetesb-idhm.csv")
```

Este caso é um exemplo de um join bem simples, onde temos uma chave única em cada tabela. No entanto, existem outros tipos de joins, como `inner_join()`, `right_join()`, `full_join()`, que podem ser utilizados dependendo da situação.

4.12 Materiais complementares

- Materiais do curso [Introdução à análise de dados no R](#):
 - [Transformando dados](#)
 - [Análise exploratória de dados](#)

5 Prática - Intervalo de confiança

Nota

Os conceitos foram apresentados na aula teórica. A seguir, temos uma prática para aplicar esses conceitos. É importante que você tenha assistido à aula antes de realizar esta prática, pois ela se baseia nos conceitos discutidos.

Nesta prática, vamos imaginar a seguinte situação: queremos saber qual é a média das alturas das pessoas que estão matriculadas nessa disciplina, considerando o sexo biológico. Para isso, criamos um formulário do Google Forms e solicitamos que as pessoas respondessem. Entretanto, nem todas as pessoas responderam, e só temos uma amostra.

Revisando alguns conceitos apresentados em aula: neste exemplo...

- a **população** é o conjunto de todas as pessoas matriculadas na disciplina, enquanto a **amostra** é composta pelas pessoas que responderam ao formulário.
- A partir dessa amostra, podemos fazer **inferências** sobre a população, como calcular a **média** das alturas por sexo biológico.
- No entanto, como estamos lidando com uma amostra, não podemos afirmar com certeza qual é a média da população, apenas **estimá-la**.
- Para isso, usamos o **intervalo de confiança**, que nos dá uma faixa de valores dentro da qual acreditamos que a média populacional esteja.

5.1 Carregar pacotes

```
library(tidyverse)
library(janitor)
```

5.2 Importar os dados

Os dados estão disponíveis em um formulário do Google. Podemos importá-los de duas formas:
- usando o pacote `googlesheets4` (requer autenticação) - baixar o arquivo CSV exportado do Google Forms e ler com `read_csv()`. Salve o arquivo CSV na pasta `dados/` do seu projeto.

Neste exemplo, vamos importar o arquivo CSV exportado do Google Forms. **O CSV que será utilizado é um recorte do arquivo completo, pois removemos as colunas de nome e email.**

Se você fez o download do arquivo CSV, coloque-o na pasta `dados/` do seu projeto. Caso contrário, você pode usar o link direto para o arquivo CSV no GitHub.

```
# leitura do CSV exportado do Google Forms
dados_brutos <- read_csv("dados/respostas_forms_altura.csv")  
  
# leitura de uma cópia dos dados que está salva no GitHub
dados_brutos <- read_csv("https://raw.githubusercontent.com/beatrizmilz/ESHT011-21-analise-d  
  
Rows: 33 Columns: 5
-- Column specification -----
Delimiter: ","
chr (2): Sexo biológico, Turma
dbl (2): Idade, Altura (em metros)
dttm (1): Carimbo de data/hora  
  
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 dar uma olhada nos dados que importamos:

```
glimpse(dados_brutos)  
  
Rows: 33
Columns: 5
$ `Carimbo de data/hora` <dttm> 2025-06-23 10:04:10, 2025-06-23 10:08:13, 2025-
$ Idade <dbl> 36, 43, 20, 56, 42, 35, 36, 26, 21, 28, 29, 31, ~
$ `Sexo biológico` <chr> "Feminino", "Feminino", "Masculino", "Masculino~
$ `Altura (em metros)` <dbl> 1.58, 1.68, 1.73, 1.73, 1.69, 1.68, 1.94, 1.76, ~
$ Turma <chr> "Pós-graduação - Matutino", "Pós-graduação - Ma~
```

Algo que podemos fazer é limpar os nomes das colunas, pois eles podem conter espaços, caracteres especiais ou estarem em letras maiúsculas. Vamos usar a função `clean_names()` do pacote `janitor` para isso.

```
dados <- clean_names(dados_brutos)
```

Agora podemos verificar os nomes das colunas novamente para garantir que estão limpos e prontos para uso:

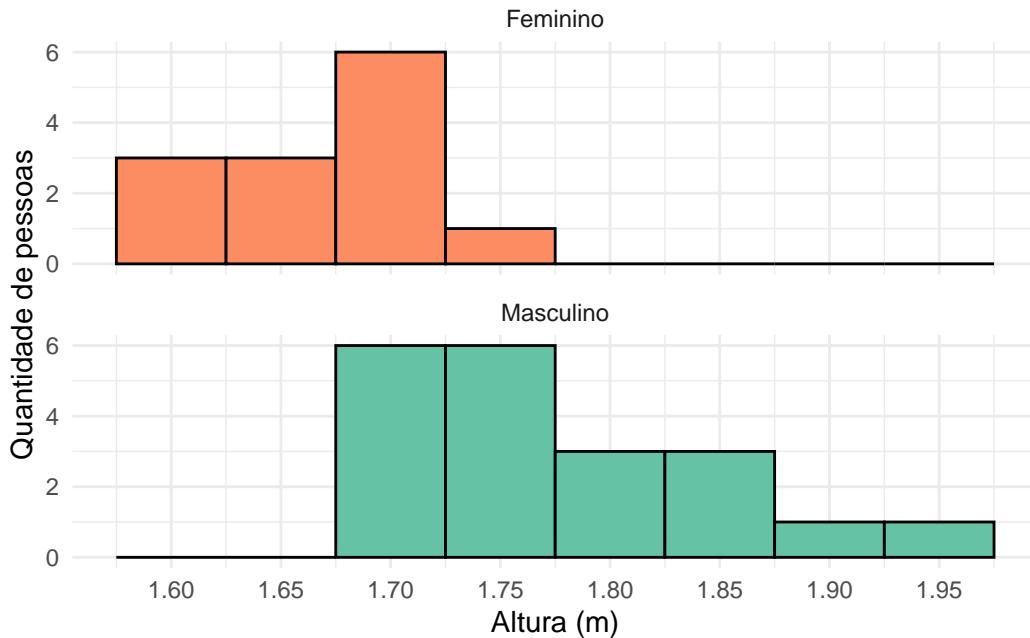
```
glimpse(dados)
```

```
Rows: 33
Columns: 5
$ carimbo_de_data_hora <dttm> 2025-06-23 10:04:10, 2025-06-23 10:08:13, 2025-0-
$ idade <dbl> 36, 43, 20, 56, 42, 35, 36, 26, 21, 28, 29, 31, 4-
$ sexo_biológico <chr> "Feminino", "Feminino", "Masculino", "Masculino", ~
$ altura_em_metros <dbl> 1.58, 1.68, 1.73, 1.73, 1.69, 1.68, 1.94, 1.76, 1-
$ turma <chr> "Pós-graduação - Matutino", "Pós-graduação - Matu-
```

5.3 Conhecendo os dados

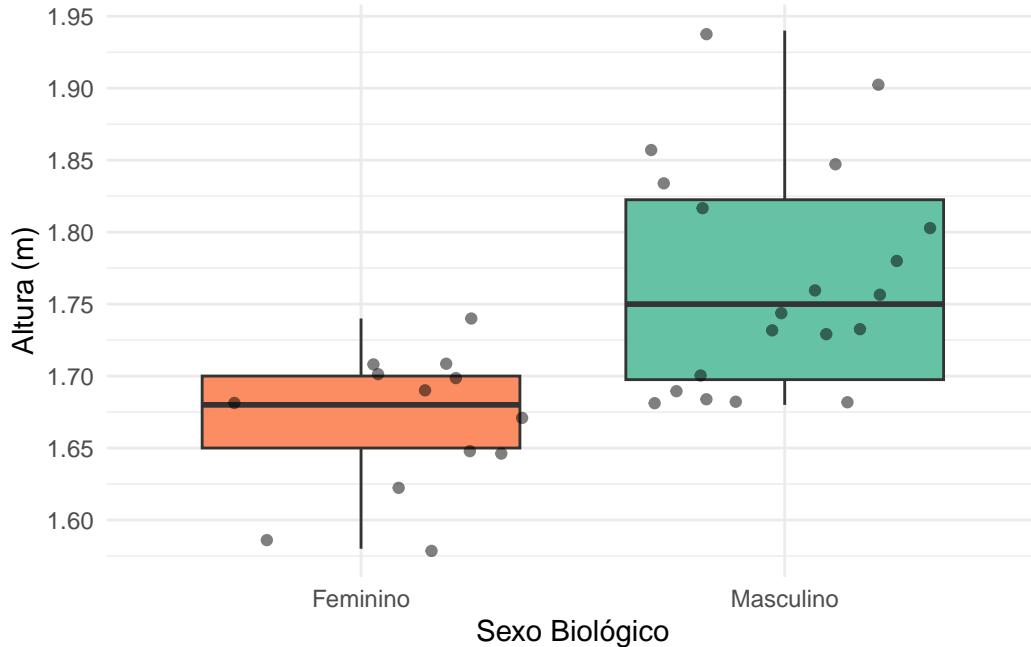
Vamos começar visualizando a distribuição das alturas por sexo biológico. Para isso, podemos criar um histograma para cada grupo de sexo biológico.

```
dados |>
  ggplot(aes(x = altura_em_metros, fill = sexo_biológico)) +
    geom_histogram(binwidth = 0.05, color = "black", show.legend = FALSE) +
    labs(y = "Quantidade de pessoas",
         x = "Altura (m)") +
    scale_x_continuous(breaks = seq(1.50, 2.1, by = 0.05)) +
    theme_minimal() +
    facet_wrap(~ sexo_biológico, nrow = 2
               ) +
    scale_fill_brewer(palette = "Set2", direction = -1)
```



Vamos criar um box-plot para visualizar a distribuição das alturas por sexo biológico. Isso nos ajudará a entender como as alturas estão distribuídas em cada grupo.

```
dados |>
ggplot(aes(x = sexo_biologico, y = altura_em_metros)) +
  geom_boxplot(aes(fill = sexo_biologico), show.legend = FALSE) +
  geom_jitter(show.legend = FALSE, alpha = 0.5) +
  labs(x = "Sexo Biológico",
       y = "Altura (m)") +
  scale_y_continuous(breaks = seq(1.5, 2.1, by = 0.05)) +
  theme_minimal() +
  scale_fill_brewer(palette = "Set2", direction = -1)
```



Outro ponto importante é saber o tamanho da nossa amostra. **E atenção:** neste caso, queremos calcular o intervalo de confiança para a média de dois grupos (sexo biológico), então precisamos calcular a amostra separadamente para cada grupo.

```
tabyl(dados, sexo_biológico) |> # criando uma tabela de frequências
  adorn_totals("row") |> # adicionando a linha de total
  adorn_pct_formatting() # formatando os percentuais
```

sexo_biológico	n	percent
Feminino	13	39.4%
Masculino	20	60.6%
Total	33	100.0%

5.4 Calcular a média, desvio padrão e número de respostas

Para calcular o intervalo de confiança da média das alturas por sexo biológico, precisamos primeiro calcular a média, o desvio padrão e o tamanho da amostra.

Para isso, podemos usar a função `group_by()` do `dplyr` para agrupar os dados por sexo biológico e, em seguida, usar a função `summarise()` para calcular as estatísticas desejadas. [Essas funções foram abordadas na prática anterior.](#)

Para calcular a média, utilizamos a função `mean()`, para o desvio padrão usamos `sd()` e para o tamanho da amostra usamos `n()`. Lembrando que, caso a variável tenha valores ausente (NA), devemos usar o argumento `na.rm = TRUE` para ignorá-los nos cálculos.

```
dados |>
  group_by(sexo_biologico) |>
  summarise(
    media = mean(altura_em_metros, na.rm = TRUE),
    desvio_padrao = sd(altura_em_metros, na.rm = TRUE),
    tamanho_amostra_n = n(),
  )

# A tibble: 2 x 4
  sexo_biologico   media  desvio_padrao tamanho_amostra_n
  <chr>           <dbl>     <dbl>            <int>
1 Feminino         1.67      0.0485          13
2 Masculino        1.77      0.0784          20
```

5.5 Calcular o erro padrão

Com essas informações, podemos calcular o erro padrão (**Standard Error - SE**) amostral da média:

$$SE = \frac{s}{\sqrt{n}}$$

Onde s é o desvio padrão amostral e n é o tamanho da amostra. Lembrando que o erro padrão nos dá uma medida da precisão da média amostral como estimativa da média populacional.

```
dados |>
  group_by(sexo_biologico) |>
  summarise(
    media = mean(altura_em_metros, na.rm = TRUE),
    desvio_padrao = sd(altura_em_metros, na.rm = TRUE),
    tamanho_amostra_n = n(),
    erro_padrao = desvio_padrao / sqrt(tamanho_amostra_n)
  )

# A tibble: 2 x 5
  sexo_biologico   media  desvio_padrao tamanho_amostra_n  erro_padrao
  <chr>           <dbl>     <dbl>            <int>       <dbl>
1 Feminino         1.67      0.0485          13            0.012
2 Masculino        1.77      0.0784          20            0.018
```

	<chr>	<dbl>	<dbl>	<int>	<dbl>
1	Feminino	1.67	0.0485	13	0.0134
2	Masculino	1.77	0.0784	20	0.0175

5.6 Definindo o valor crítico

O intervalo de confiança é dado pela fórmula:

$$IC = \bar{x} \pm (z \times SE)$$

Onde \bar{x} é a média amostral, z é o valor crítico (para um IC de 95%, $z \approx 1.96$, quando a amostra é grande ou a variável tem distribuição aproximadamente normal) e SE é o erro padrão.

Usar um intervalo de confiança de 95% significa que queremos encontrar uma faixa de valores onde acreditamos, com 95% de confiança, que está a média verdadeira da população.

Esse valor de $z \approx 1.96$ é usado apenas quando usamos a distribuição normal. No entanto, como estamos trabalhando com amostras pequenas, utilizamos a distribuição t de Student, e o valor crítico t (que substitui o z) é calculado com a função `qt()`. Neste caso, o valor muda conforme o **tamanho da amostra (n)** e é calculado com base nos graus de liberdade ($n - 1$).

5.6.1 Como calcular o valor crítico para um intervalo de confiança de 95% com a distribuição t de Student?

Podemos usar a função `qt()` do R, que calcula o quantil da distribuição t de Student. Para um intervalo de confiança de 95%, precisamos calcular o quantil para 0.975 (ou seja, $1 - 0.025$, já que estamos considerando os dois lados da distribuição).

```
dados |>
  group_by(sexo_biologico) |>
  summarise(
    media = mean(altura_em_metros, na.rm = TRUE),
    desvio_padrao = sd(altura_em_metros, na.rm = TRUE),
    tamanho_amostra_n = n(),
    erro_padrao = desvio_padrao / sqrt(tamanho_amostra_n),
    valor_t = qt(p = 0.975, df = tamanho_amostra_n - 1) # valor crítico t para IC de 95%
  )
```

```
# A tibble: 2 x 6
  sexo_biológico media desvio_padrao tamanho_amostra_n erro_padrao valor_t
  <chr>          <dbl>      <dbl>           <int>      <dbl>      <dbl>
1 Feminino        1.67       0.0485            13       0.0134     2.18
2 Masculino       1.77       0.0784            20       0.0175     2.09
```

5.7 Calcular o intervalo de confiança

```
ic_altura <- dados |>
  group_by(sexo_biológico) |>
  summarise(
    media = mean(altura_em_metros, na.rm = TRUE),
    desvio_padrao = sd(altura_em_metros, na.rm = TRUE),
    tamanho_amostra_n = n(),
    erro_padrao = desvio_padrao / sqrt(tamanho_amostra_n),
    valor_t = qt(p = 0.975, df = tamanho_amostra_n - 1),
    ic_inferior = media - (valor_t * erro_padrao),
    ic_superior = media + (valor_t * erro_padrao)
  )

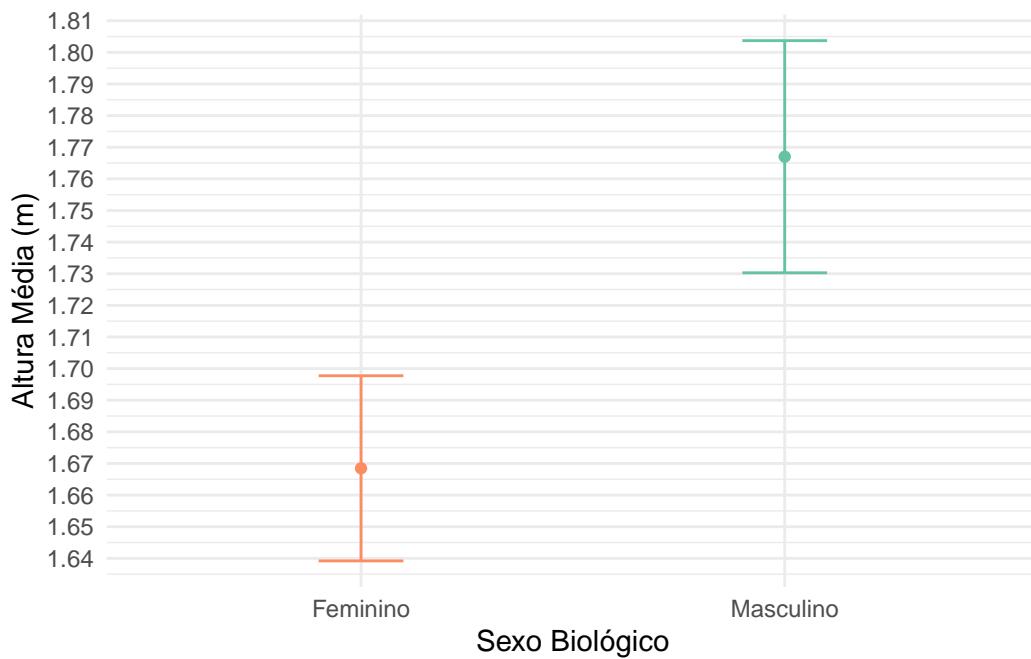
knitr::kable(ic_altura)
```

sexo_biológico	media	desvio_padrao	tamanho_amostra_n	erro_padrao	valor_t	ic_inferior	ic_superior
Feminino	1.668462	0.0484503	13	0.0134377	2.178813	1.639183	1.697740
Masculino	1.767000	0.0784119	20	0.0175334	2.093024	1.730302	1.803698

Com o intervalo de confiança calculado, podemos visualizar os resultados em um gráfico. Vamos criar um gráfico de pontos com barras de erro para representar o intervalo de confiança.

```
ggplot(ic_altura,
       aes(x = sexo_biológico, y = media, color = sexo_biológico)) +
  geom_point(show.legend = FALSE) +
  geom_errorbar(aes(ymin = ic_inferior, ymax = ic_superior),
                width = 0.2,
                show.legend = FALSE) +
  labs(x = "Sexo Biológico", y = "Altura Média (m)") +
  scale_color_brewer(palette = "Set2", direction = -1) +
```

```
scale_y_continuous(breaks = seq(1.5, 2.1, by = 0.01)) +
theme_minimal()
```



Para discussão:

- 1) O intervalo de confiança dos dois grupos se sobrepõe? O que isso pode significar?
- 2) Como o tamanho da amostra (n) afetou o intervalo de confiança?
- 3) Como a variação nas alturas (desvio padrão) influenciou o intervalo de confiança? ##
Materiais complementares
 - Materiais da disciplina Ciência de Dados Aplicada ao Direito II:
 - Testes de hipóteses

6 Prática - Correlação

Explorando a relação linear entre duas variáveis numéricas

i Nota

Os conceitos foram apresentados na aula teórica. A seguir, temos uma prática para aplicar esses conceitos. É importante que você tenha assistido à aula antes de realizar esta prática, pois ela se baseia nos conceitos discutidos.

i Nota

Essa prática foi muito inspirada pela prática preparada pelo [Luis Felipe Bortolatto da Cunha](#) para edições passadas dessa disciplina.

6.1 Carregando os pacotes

Os pacotes necessários para essa prática são:

- `tidyverse`: para manipulação e visualização de dados;
- `corrplot` e `GGally`: para visualização de matrizes de correlação;

```
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    3.3.0
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 non-conflicting

library(corrplot)

corrplot 0.95 loaded

library(GGally)

Registered S3 method overwritten by 'GGally':
  method from
  +.gg   ggplot2

```

6.2 Importando os dados

Para executar a análise de correlação vamos utilizar dados demográficos e de consumo de água de 2010, para uma amostra de municípios, extraídos do Censo Demográfico (IBGE) e do Sistema Nacional de Informações sobre Saneamento (SNIS), para investigar se **o consumo de água está correlacionado com a renda**, conforme análise apresentada por [Carmo et al., 2013](#).

Download: [Clique aqui caso queira fazer o download dos dados](#)

Atenção: os dados estão no formato CSV, que é um formato de texto separado por vírgulas. O `read_csv2` é usado para ler arquivos CSV que usam ponto e vírgula (;) como separador de campos, o que é o caso deste arquivo.

```

dados <- read_csv2("https://raw.githubusercontent.com/beatrizmilz/ESHT011-21-analise-dados-pi

i Using ',', '' as decimal and '\"' as grouping mark. Use `read_delim()`` for more control.

Rows: 5566 Columns: 19
-- Column specification -----
Delimiter: ;"
chr (4): nome_mun, uf, regiao, idh_class
dbl (15): id_ibge, domicil, rede, proprede, id_snis, pib, rendapita, gini, i...
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 dar uma olhada para conhecer a estrutura dos dados e as variáveis disponíveis.

```
glimpse(dados)
```

```
Rows: 5,566
Columns: 19
$ id_ibge    <dbl> 3548807, 4214904, 3303302, 3205309, 3547304, 4205407, 314480~
$ domicil    <dbl> 50492, 1312, 169237, 108515, 31610, 147437, 24203, 508456, 7~
$ rede        <dbl> 50472, 412, 164768, 107715, 28728, 137984, 22436, 505149, 73~
$ proprede   <dbl> 0.9996038976, 0.3140243902, 0.9735932450, 0.9926277473, 0.90~
$ id_snis    <dbl> 354880, 421490, 330330, 320530, 354730, 420540, 314480, 4314~
$ nome_mun   <chr> "São Caetano do Sul", "Rio Fortuna", "Niterói", "Vitória", "~"
$ uf          <chr> "SP", "SC", "RJ", "ES", "SP", "SC", "MG", "RS", "DF", "SP", ~
$ regiao      <chr> "Sudeste", "Sul", "Sudeste", "Sudeste", "Sudeste", "Sul", "S~
$ pib         <dbl> 6694384.00, 39035.65, 5831066.00, 9270129.00, 1085714.00, 42~
$ rendapita   <dbl> 2008.98, 1570.54, 1951.11, 1820.97, 1798.50, 1770.29, 1709.8~
$ gini         <dbl> 0.5480, 0.5638, 0.5983, 0.6124, 0.6858, 0.5474, 0.6914, 0.61~
$ idh         <dbl> 0.919, 0.822, 0.886, 0.856, 0.853, 0.875, 0.821, 0.865, 0.84~
$ idh_class   <chr> "Muito alto", "Alto", "Alto", "Alto", "Alto", "Alto", "Alto"~
$ ge012       <dbl> 149263, 4446, 487562, 327801, 108813, 421240, 80998, 1409351~
$ ag001       <dbl> 149263, 1774, 487562, 327801, 102006, 413263, 79232, 1409351~
$ ag020       <dbl> 9666.74, 58.94, 31499.85, 19825.98, 6207.20, 18464.15, 4185.~
$ ag022       <dbl> 56873, 501, 154431, 108174, 30583, 156279, 25012, 518953, 85~
$ consumo1    <dbl> 64.763136, 13.256860, 64.606860, 60.481756, 57.044655, 43.83~
$ consumo2    <dbl> 64.76314, 33.22435, 64.60686, 60.48176, 60.85132, 44.67893, ~
```

A tabela abaixo apresenta uma descrição de cada variável.

Código	Descrição
id_ibge	Código IBGE (7 dígitos)
domicil	Quantidade de Domicílios
rede	Quantidade de Domicílios com Acesso à Rede Geral de Água
proprede	Proporção de Domicílios com com Acesso à Rede Geral de Água (REDE/DOMICIL)
id_snins	Código IBGE (6 dígitos)
NOME_MUN	Nome do Município
uf	Unidade da Federação
regiao	Região do País
pib	Produto Interno Bruto 2010
rendapita	Renda per Capita 2010

Código	Descrição
gini	Índice GINI 2010
idh	Índice de Desenvolvimento Humano 2010
idh_class	Classificação do Índice de Desenvolvimento Humano 2010: Muito Alto $\geq 0,9$; Alto $\geq 0,8$; Médio $\geq 0,5$; Baixo $< 0,5$.
ge012	População Total Residente no Município
ag001	População Total Atendida com Abastecimento de Água
ag020	Volume Micromedido nas Economias Residenciais Ativas de Água - 1.000 m ³ /ano
ag022	Quantidade de Economias Residenciais Ativas Micromedidas
consumo1	Consumo de Água per capita - População Total - m ³ /ano (AG020/GE012)
consumo2	Consumo de Água per capita - População Atendida - m ³ /ano (AG020/AG001)

6.3 Pergunta norteadora

A pergunta que vamos investigar é: **o consumo de água (per capita) está correlacionado com a renda (per capita)?**

As variáveis que vamos utilizar para responder a essa pergunta são:

- **consumo1**: Consumo de água per capita - População Total - m³/ano (AG020/GE012);
- **rendapita**: Renda per capita.

Importante: a correlação é utilizada para investigar a relação entre duas **variáveis numéricas**.

6.4 Gráfico de dispersão

O gráfico de dispersão é uma representação gráfica que mostra a relação entre duas variáveis numéricas. Cada ponto no gráfico representa um par de valores das duas variáveis.

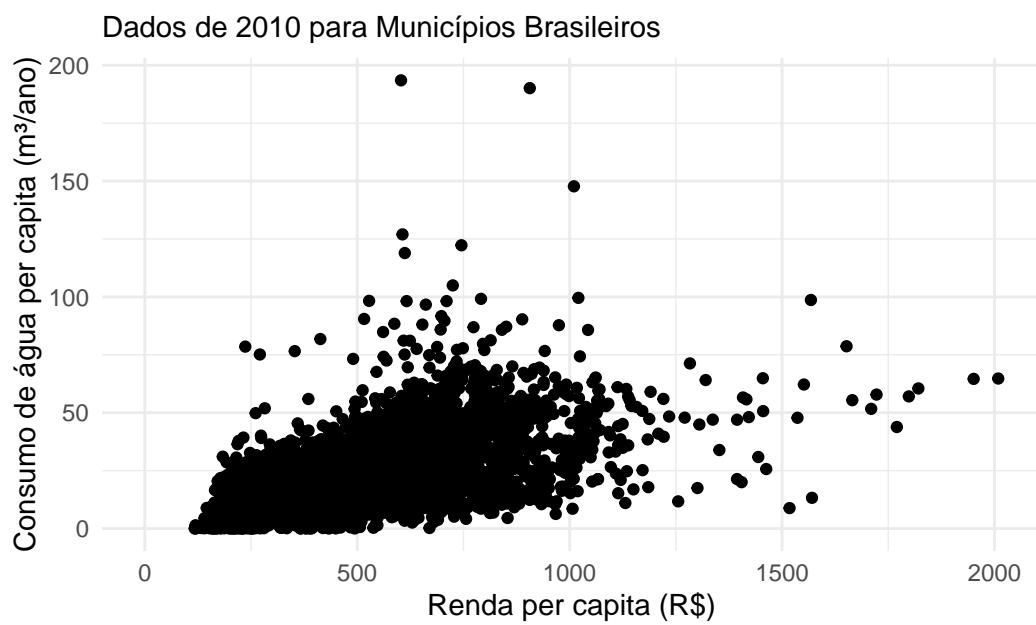
Por exemplo, podemos representar

```

dados |>
  ggplot(aes(x = rendapita, y = consumo1)) +
  geom_point() +
  labs(
    title = "Relação entre Renda per Capita e Consumo de Água per Capita",
    subtitle = "Dados de 2010 para Municípios Brasileiros",
    caption = "Fonte: SNIS e IBGE",
    x = "Renda per capita (R$)",
    y = "Consumo de água per capita (m³/ano)"
  ) +
  theme_minimal()

```

Warning: Removed 1149 rows containing missing values or values outside the scale range (geom_point()).



! Aviso

Veja que o `ggplot` apresentou um aviso (*warning*):

Warning: Removed 1149 rows containing missing values or values outside the scale range (geom_point()).

Quando estamos criando um gráfico de dispersão, precisamos que ambas as variáveis

estejam presentes para cada ponto. Se uma das variáveis tiver valores ausentes (NA) ou fora do intervalo esperado, esses pontos serão removidos do gráfico. Isso é normal e esperado, pois o `ggplot` não pode plotar pontos com valores ausentes.

Interpretando o gráfico, podemos observar que há uma **tendência** de que, à medida que a renda per capita aumenta, o consumo de água per capita também tende a aumentar.

No entanto, essa relação não é perfeita e existem pontos que se desviam dessa tendência. Por exemplo: alguns municípios apresentam maiores valores de consumo de água per capita, mesmo com rendas per capita mais baixas, enquanto outros municípios com rendas mais altas apresentam consumos de água per capita mais baixos.

Porém, o foco neste momento é visualizar as tendências.

Para quantificar essa relação, podemos calcular a correlação entre as duas variáveis.

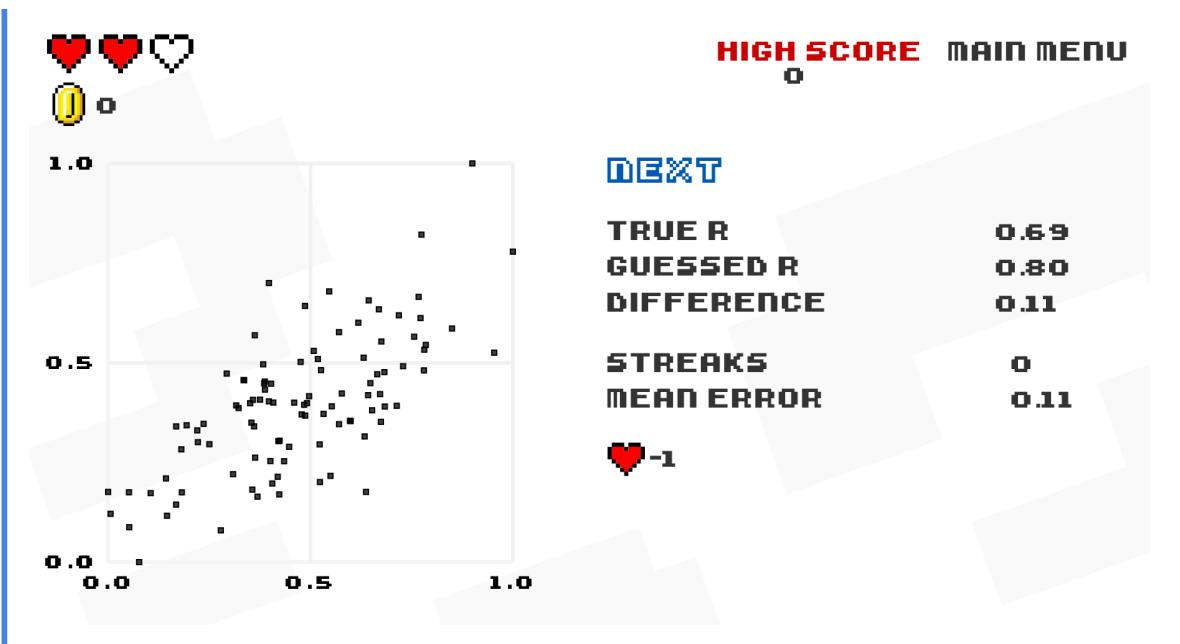
6.5 Correlação

A correlação é uma medida estatística que indica a força e a direção de uma **relação linear** entre duas **variáveis numéricas**. O **coeficiente de correlação de Pearson** apresenta valores entre -1 e 1, onde:

- 1 indica uma correlação positiva perfeita (quando uma variável aumenta, a outra também aumenta);
- -1 indica uma correlação negativa perfeita (quando uma variável aumenta, a outra diminui);
- 0 indica que não há correlação linear entre as variáveis.

i Nota

Um site divertido para brincar com correlações é o [Guess the Correlation](#). Ele apresenta gráficos de dispersão e você deve adivinhar o coeficiente de correlação!



Para calcular a correlação entre as variáveis `consumo1` e `rendapita`, podemos usar a função `cor()` do R, que calcula o coeficiente de correlação, e podemos indicar alguns argumentos:

- `x` e `y`: as duas variáveis numéricas que queremos correlacionar;
- `method`: o método de correlação a ser utilizado (por padrão, é o coeficiente de correlação de Pearson, porém também é possível calcular utilizando os métodos "`spearman`" e "`kendall`");
- `use`: como lidar com valores ausentes (NA). Por exemplo, podemos usar `use = "complete.obs"` para remover os casos com valores ausentes antes de calcular a correlação.

```
correlacao <- cor(x = dados$rendapita, # variável
                    y = dados$consumo1, # variável
                    method = "pearson", # método de correlação
                    use = "complete.obs" # usar apenas observações completas
                  )

correlacao
```

[1] 0.6012537

A correlação entre as variáveis renda e o consumo de água per capita é de 0.6, indicando a mesma **correlação positiva** que o gráfico de dispersão permitiu visualizar.

6.6 Matriz de correlação

Até agora, calculamos a correlação entre duas variáveis específicas. No entanto, podemos calcular a correlação entre várias variáveis numéricas de uma vez, criando uma matriz de correlação.

Vamos recapitular alguns cuidados: - Correlação deve ser usada apenas com **variáveis numéricas**; - Não devemos usar variáveis de identificação, como códigos IBGE, nomes de municípios, etc;

Para construir uma matriz de correlação, primeiro é necessário selecionar as variáveis que desejamos investigar (com a função `select()`), para depois calcular o coeficiente de correlação entre cada dupla de variáveis (com a função `cor()`).

Vamos criar um objeto com as variáveis que queremos investigar:

```
dados_selecionados <- dados |>
  select(renderapita, pib, gini, idh, consumo1, consumo2, proprede)
```

Agora podemos calcular a matriz de correlação entre essas variáveis, utilizando a função `cor()`:

```
dados_selecionados |>
  cor(method = "pearson", use = "complete.obs")
```

```
      renderapita      pib        gini       idh    consumo1    consumo2
rendapita  1.0000000 0.22623371 -0.21476760  0.7415117  0.6012537  0.49269163
pib        0.2262337 1.00000000  0.09083846  0.1190256  0.1176077  0.06518107
gini      -0.2147676  0.09083846  1.00000000 -0.3259986 -0.2680246 -0.25814509
idh        0.7415117  0.11902559 -0.32599859  1.0000000  0.5143510  0.42554311
consumo1   0.6012537  0.11760770 -0.26802459  0.5143510  1.0000000  0.85700512
consumo2   0.4926916  0.06518107 -0.25814509  0.4255431  0.8570051  1.00000000
proprede   0.4012507  0.11643021 -0.17370841  0.3731320  0.6117510  0.35458719
                           proprede
rendapita  0.4012507
pib        0.1164302
gini      -0.1737084
idh        0.3731320
consumo1   0.6117510
consumo2   0.3545872
proprede  1.0000000
```

Esse resultado pode ser mais fácil de ser interpretado caso os valores sejam arredondados. Podemos fazer isso com a função `round()`:

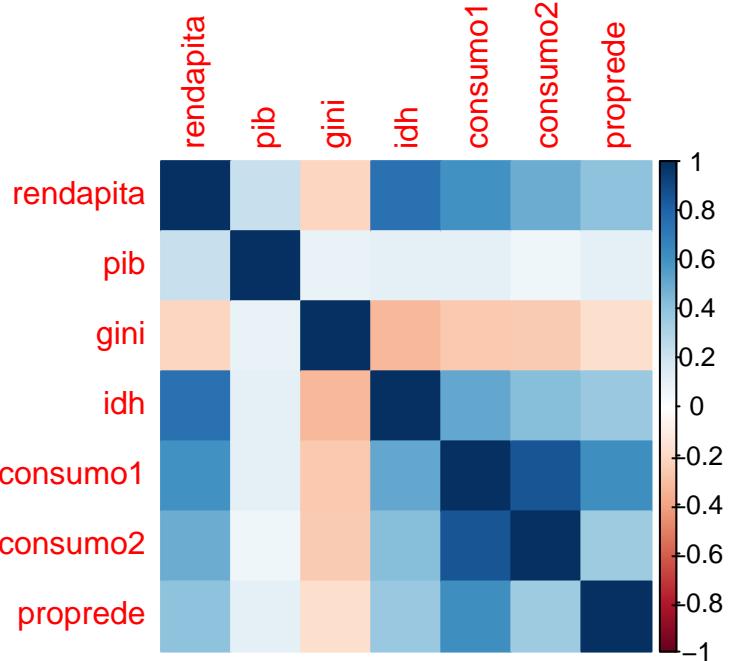
```
dados_selecionados |>
  cor(method = "pearson", use = "complete.obs") |>
  round(2)
```

	rendapita	pib	gini	idh	consumo1	consumo2	proprede
rendapita	1.00	0.23	-0.21	0.74	0.60	0.49	0.40
pib		1.00	0.09	0.12	0.12	0.07	0.12
gini			1.00	-0.33	-0.27	-0.26	-0.17
idh				1.00	0.51	0.43	0.37
consumo1					1.00	0.86	0.61
consumo2						1.00	0.35
proprede							1.00

6.6.1 Visualizando a matriz de correlação

Também é possível visualizar a matriz de correlação construindo um gráfico, usando a função `corrplot()` do pacote `corrplot`.

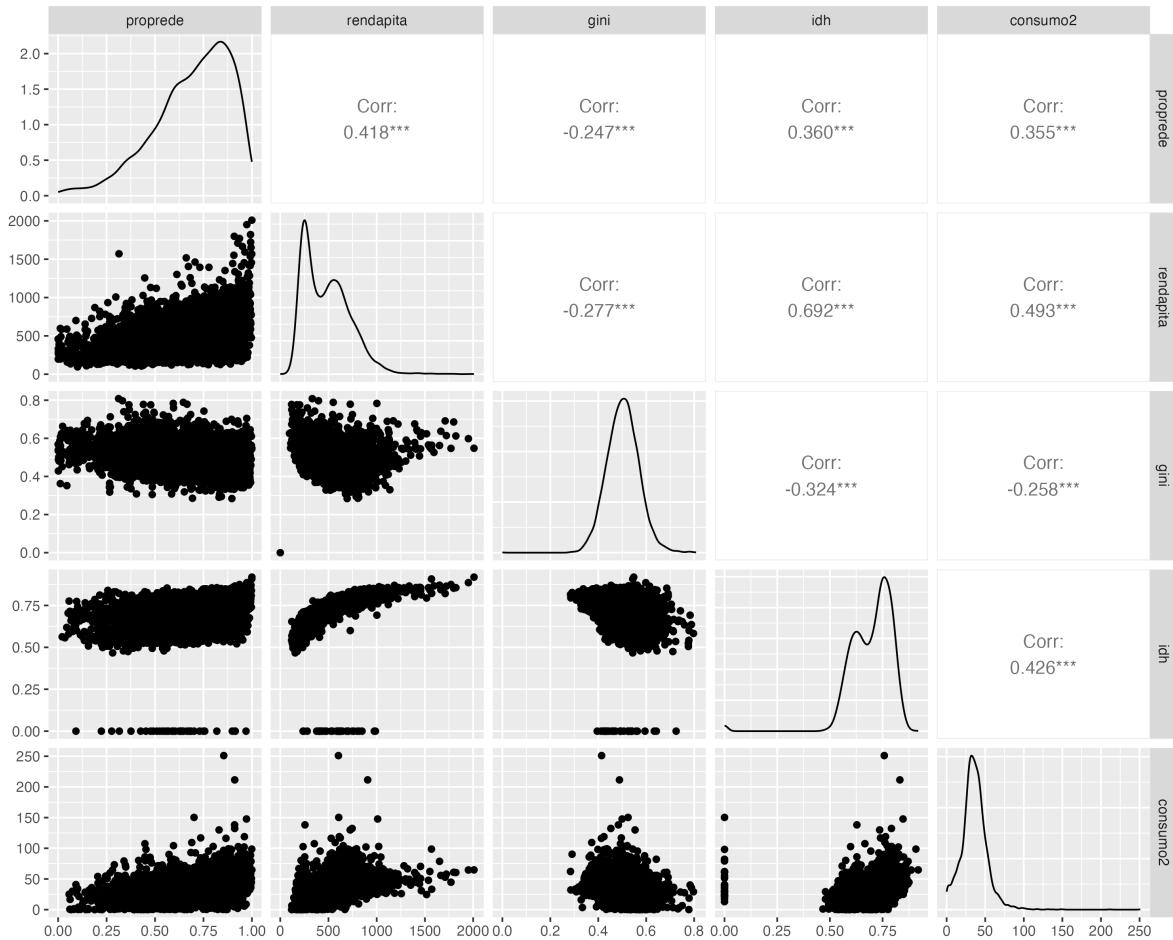
```
dados_selecionados |>
  cor(method = "pearson",
       use = "complete.obs") |>
  corrplot(method = "color")
```



Outra forma de visualizar a matriz de correlação é utilizando a função `ggpairs()` do pacote `GGally`, que cria uma matriz de gráficos de dispersão, histogramas e correlações entre as variáveis selecionadas:

```
plot_ggpairs <- dados |>
  select(rendapita, gini, idh, consumo2) |>
  GGally::ggpairs()

plot_ggpairs
```



6.7 EXTRA: Teste de significância

Será que 0.6 é significativo? Se fosse -0.3, seria significativo? E se fosse 0.1? Uma forma de verificar isso é através de um teste de hipótese. Nesse teste, avaliamos se a correlação observada é tão próxima de zero que podemos considerar que o que observamos é fruto do acaso.

Para testar a significância da correlação entre duas variáveis, podemos usar o teste de hipótese para o coeficiente de correlação de Pearson. Esse teste nos dá um guia para avaliar se a correlação observada é significativa.

É importante definir antes de realizar o teste de significância qual é a hipótese nula e a hipótese alternativa:

- **Hipótese nula (H0):** Não há correlação entre as duas variáveis (coeficiente de correlação é igual a zero).
- **Hipótese alternativa (H1):** Há correlação entre as duas variáveis (coeficiente de correlação é diferente de zero, podendo ser positivo ou negativo).

Para testar a significância do coeficiente de correlação de Pearson podemos usar a função `cor.test()`, especificando os argumentos:

- `x` e `y`: as duas variáveis numéricas que queremos correlacionar;
- `method`: o método de correlação a ser utilizado. Por padrão, é o coeficiente de correlação de Pearson, mas também é possível calcular utilizando os métodos "`spearman`" e "`kendall`". Ao mudar o método, também mudamos o teste de hipótese realizado;
- `alternative`: o tipo de teste de hipótese a ser realizado. Os valores possíveis são "`two.sided`" (bilateral), "`less`" (unilateral à esquerda: valor negativo) e "`greater`" (unilateral à direita: valor positivo).
- `conf.level`: o nível de confiança para o intervalo de confiança do coeficiente de correlação (por padrão, é 0.95, ou seja, 95% de confiança).

```
resultado_cor_test <- cor.test(x = dados$rendapita, # variável
                                y = dados$consumo1, # variável
                                method = "pearson", # método de correlação
                                alternative = "two.sided", # teste bilateral
                                conf.level = 0.95, # nível de confiança
                                )
```

`resultado_cor_test`

```
Pearson's product-moment correlation

data: dados$rendapita and dados$consumo1
t = 49.997, df = 4415, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
0.5820832 0.6197561
sample estimates:
cor
0.6012537
```

```
broom::tidy(resultado_cor_test)

# A tibble: 1 x 8
  estimate statistic p.value parameter conf.low conf.high method      alternative
    <dbl>      <dbl>     <dbl>      <int>      <dbl>      <dbl> <chr>      <chr>
1     0.601      50.0       0        4415      0.582      0.620 Pearson's~ two.sided
```

Os resultados de testes estatísticos no R são confusos no início. O objeto criado é uma lista, e podemos acessar os componentes dessa lista para obter informações específicas utilizando o operador \$.

```
names(resultado_cor_test)
```

```
[1] "statistic"    "parameter"    "p.value"      "estimate"     "null.value"
[6] "alternative"  "method"       "data.name"    "conf.int"
```

6.7.1 statistic

O componente **statistic** é o valor do teste estatístico usado para calcular o valor-p. No caso da correlação de Pearson, esse valor é calculado com base no valor de t , por conta das propriedades da estatística de Pearson. Esses valores não costumam ser interpretados isoladamente neste contexto.

```
resultado_cor_test$statistic
```

```
  t
49.99709
```

6.7.2 parameter

O componente **parameter** indica os parâmetros associados ao teste. No caso da correlação de Pearson, são os graus de liberdade, que é calculado como $n - 2$, onde n é o número de observações completas (ou seja, o número de pares de valores das duas variáveis). Isso acontece porque estamos calculando duas variâncias, o que diminui os graus de liberdade disponíveis para o teste em 2.

```
resultado_cor_test$parameter
```

```
df  
4415
```

Podemos conferir isso contando o número de observações completas (ou seja, sem valores ausentes) nas duas variáveis que estamos correlacionando:

```
dados |>  
  drop_na(rendapita, consumo1) |>  
  nrow()
```

```
[1] 4417
```

6.7.3 p.value ou valor-p

O valor-p (ou p-value) indica a probabilidade de observar um valor tão ou mais extremo quanto o observado, assumindo que a hipótese nula (correlação = 0) seja verdadeira. Em uma linguagem mais simples, o valor-p nos diz a probabilidade de obtermos o resultado observado ao acaso. Neste contexto, é usado como guia para decidir se a correlação observada é estatisticamente significante.

Aviso

ALERTA: O R arredonda os valores por padrão para algumas casas. Por isso, eu prefiro interpretar o valor-p a partir do resultado completo do teste apresentado no console (e não usando o valor da lista, ex. `resultado_cor_test$p.value`).

Neste caso, o resultado encontrado foi muito baixo, $p < 2.2e-16$, indicando que é altamente improvável obter esse resultado se a correlação verdadeira fosse zero (ou seja, se não houvesse correlação entre as variáveis).

Isso significa que **podemos rejeitar a hipótese nula (H_0) de que não há correlação entre as duas variáveis**.

6.7.4 estimate

É a estimativa do parâmetro de interesse. No caso do teste de correlação de Pearson, é o coeficiente de correlação calculado entre as duas variáveis. Esse valor é o mesmo que calculamos anteriormente com a função `cor()`.

```
resultado_cor_test$estimate
```

```
cor  
0.6012537
```

6.7.5 null.value

O componente `null.value` indica o valor do parâmetro sob a hipótese nula. No caso do teste de correlação de Pearson, esse valor é zero, pois a hipótese nula afirma que não há correlação entre as duas variáveis. Em usos mais avançados, poderíamos alterar esse valor, para verificar se a correlação é diferente de um valor específico.

```
resultado_cor_test$null.value
```

```
correlation  
0
```

6.7.6 alternative

O componente `alternative` indica qual foi a hipótese alternativa testada. No caso do teste de correlação de Pearson, o valor padrão é "`two.sided`", o que significa que estamos testando se a correlação é diferente de zero (ou seja, se há uma correlação positiva ou negativa).

```
resultado_cor_test$alternative
```

```
[1] "two.sided"
```

6.7.7 method

O componente `method` indica o método utilizado para calcular a correlação. No caso do teste de correlação de Pearson, o valor é "`Pearson`".

```
resultado_cor_test$method
```

```
[1] "Pearson's product-moment correlation"
```

6.7.8 data.name

O componente `data.name` indica o nome dos dados utilizados no teste. No caso do teste de correlação de Pearson, é uma string que contém os nomes das duas variáveis que foram correlacionadas.

```
resultado_cor_test$data.name
```

```
[1] "dados$rendapita and dados$consumo1"
```

6.7.9 conf.int

O componente `conf.int` contém o intervalo de confiança para o coeficiente de correlação. Esse intervalo é calculado com base no nível de confiança especificado (por padrão, 95%) e fornece uma faixa de valores que esperamos que contenham o verdadeiro coeficiente de correlação (lembrando que temos uma amostra de municípios, já que não temos dados para todos eles).

Além disso, esse elemento também indica qual foi o nível de confiança (`conf.level`) utilizado para calcular o intervalo de confiança.

```
resultado_cor_test$conf.int
```

```
[1] 0.5820832 0.6197561  
attr(,"conf.level")  
[1] 0.95
```

6.8 Materiais complementares

- Materiais da disciplina [Ciência de Dados Aplicada ao Direito II](#):
 - [Correlação e regressão](#)
- Vídeo: [Correlação linear bivariada no R](#) por Fernanda Peres:

A Erros e *warnings* frequentes

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

A.1 Instalação

A.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 A.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 4.5.1 (2025-06-13)"
```

A.2 Conceitos básicos

A.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")
```

A.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)
```

A.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
```

A.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).

A.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
```

A.2.4.2 Erro de digitação

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

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

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

Para corrigir, precisamos escrever a função corretamente:

```
length(letters)
```

```
[1] 26
```

B Sugestão de vídeos, posts, e outros materiais

Nesta página, reunimos alguns vídeos, posts e outros materiais que podem ser úteis para complementar os estudos!

B.1 Estatística básica

- Como calcular variância, desvio padrão e coeficiente de variação (manualmente e no Excel) - por [Fernanda Peres](#). A Fernanda também escreveu [um texto](#) sobre o assunto.
- O segredo da MERITOCRARIA - por [Atila Iamarino](#). Este vídeo aborda o tabuleiro de Galton (até cerca de 5 min), que conecta com os conceitos de distribuição normal, média, valores extremos.
- Será que você nasceu inteligente? - por [Atila Iamarino](#). Este vídeo complementa os conceitos apresentados no vídeo anterior: tabuleiro de Galton, distribuição normal, média, valores extremos.
- Uma doutora toma chá: entendendo teste de hipóteses, valor de p e nível de significância alfa - por [Fernanda Peres](#). A Fernanda também escreveu [um texto](#) sobre o assunto, e alguns outros posts complementares: [Valor de p, nível de significância e testes uni vs. bicaudais, As falácias do valor de p](#).
- Texto: [Como interpretar o intervalo de confiança?](#) - por [Fernanda Peres](#).
- Playlist: Estatística Aplicada no R - por [Fernanda Peres](#).

B.2 Organização dos dados

- Como organizar sua base de dados para análises estatísticas? - por [Fernanda Peres](#).

B.3 Gráficos

- Entendendo o gráfico boxplot - por [Fernanda Peres](#). A Fernanda também escreveu [um texto](#) sobre o assunto.

B.4 Sobre os canais citados

B.4.1 Dr. Atila Iamarino

- [Youtube](#)
- [Instagram](#)

B.4.2 Dra. Fernanda Peres

- [Youtube](#)
- [Instagram](#)
- [Blog](#)