

Usando R: Uma Introdução para Cientistas Sociais

Fernando Meireles

Denisson Silva

Índice

Prefácio	1
Introdução	3
Um exemplo	3
A nossa filosofia de trabalho	3
Para quem é este livro?	3
Como usar o livro	4
Começando	4
1 Básico	5
1.1 Introdução ao R	5
1.2 Instalando o R e RStudio	6
1.3 Sintaxe básica do R	9
1.4 Funções	15
1.5 Objetos	16
1.6 Pacotes	30
1.7 Scripts	32
1.8 Recomendações	34
1.9 Obtendo ajuda	37
References	41

Prefácio

Este é um manuscrito em desenvolvimento. A medida em que os capítulos forem concluídos, serão incluídos aqui. Dúvidas e sugestões podem ser enviadas para nossos e-mails.

Para ver a proposta mais geral do livro, clique [aqui](#).

Obs: bases de dados usadas nos capítulos estão disponíveis [aqui](#).

Introdução

Antes de prosseguir, vamos preparar o terreno: o R é um ambiente de programação, o que significa que não abriremos um banco de dados utilizando um menu de tarefas, nem calcularemos estatísticas clicando em um botão. Em vez disso, precisaremos programar, isto é, escrever código de forma ordenada que será executado sequencialmente pelo computador. Aprender a programar, especialmente no início, pode ser um pouco difícil, mas acho que não preciso reforçar o quanto todo o esforço envolvido valerá à pena.

Um exemplo

Em Desenvolvimento

A nossa filosofia de trabalho

Em Desenvolvimento

Para quem é este livro?

Em Desenvolvimento

Introdução

Como usar o livro

Em Desenvolvimento

Começando

Em Desenvolvimento

1 Básico

1.1 Introdução ao R

Este capítulo é o nosso primeiro encontro com o R. Nele, veremos alguns dos principais conceitos necessários para poder usá-lo para análise de dados – que, afinal de contas, é o nosso principal interesse aqui.

Vale reforçar: esta não é uma introdução formal ao R, comum em outros livros. Este capítulo cobre apenas o fundamental para saltarmos diretamente para o uso de ferramentas mais avançadas, que nos ajudarão a fazer análise de dados e pesquisa acadêmica.

Antes de começar, no entanto, precisaremos instalar o R. Na verdade, precisaremos instalar dois *softwares*: o R e o RStudio. O primeiro é de fato o *software* por detrás da linguagem de programação, mas ele não possui um interface, como o Excel ou outros *softwares* de armazenamento e análise de dados. É por isso que usaremos o R por meio do segundo *software*, o RStudio, que é um interface com um conjunto de funcionalidades que nos ajudará a trabalhar com o R. Depois disso, o restante do capítulo introduz noções de como escrever código em R, como salvar e manipular informações na memória, como usar funções e como instalar pacotes.

1 Básico

1.2 Instalando o R e RStudio

O R é um programa de código aberto¹ que pode ser baixado gratuitamente em <https://cran.r-project.org>, o site oficial do projeto que mantém o R. Uma vez no site, basta buscar pela opção *download* e seguir as instruções específicas para o seu sistema operacional.² A instalação deverá criar um atalho para o R no seu computador que, uma vez acessado, provavelmente mostrará algo como indica a Figura 1.1.

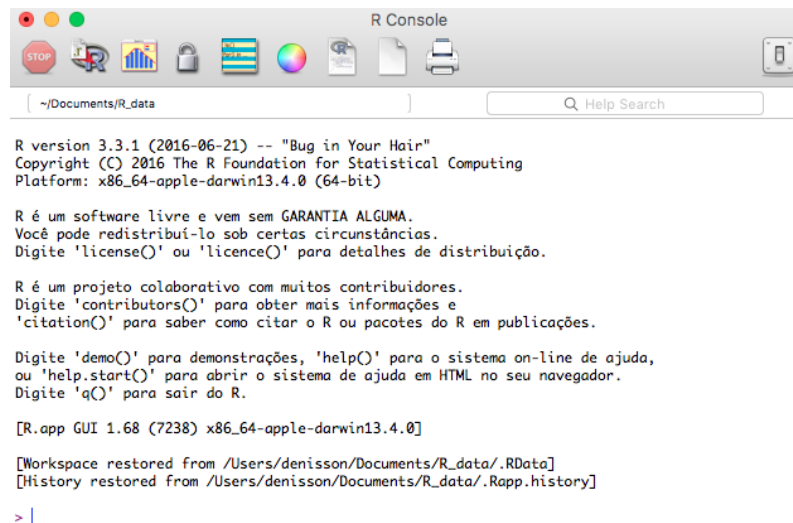


Figura 1.1: Console do R

Sem uma interface, o R nada mais é do que um console, uma tela textual

¹Código aberto é uma expressão usada normalmente para se referir a programas com um tipo de licença que permite que qualquer pessoa os use, modifiquem e compartilhem. O R um desses programas e, portanto, é gratuito.

²Para quem usa Linux, é possível instalar o R diretamente pelo terminal, usando o comando `sudo apt-get install r-base`, por exemplo.

1.2 Instalando o R e RStudio

onde podemos ler e digitar código. Para termos uma interface melhor, podemos agora baixar o RStudio em <https://posit.co/download/rstudio-desktop/>, site da empresa que o mantém – apesar de desenvolverem também outras versões, o RStudio *Desktop – Open Source License* também é gratuito. Novamente, basta buscar a opção mais adequada para o sistema operacional. Para o Windows, basta clicar em Windows e baixar o atalho para o desktop.

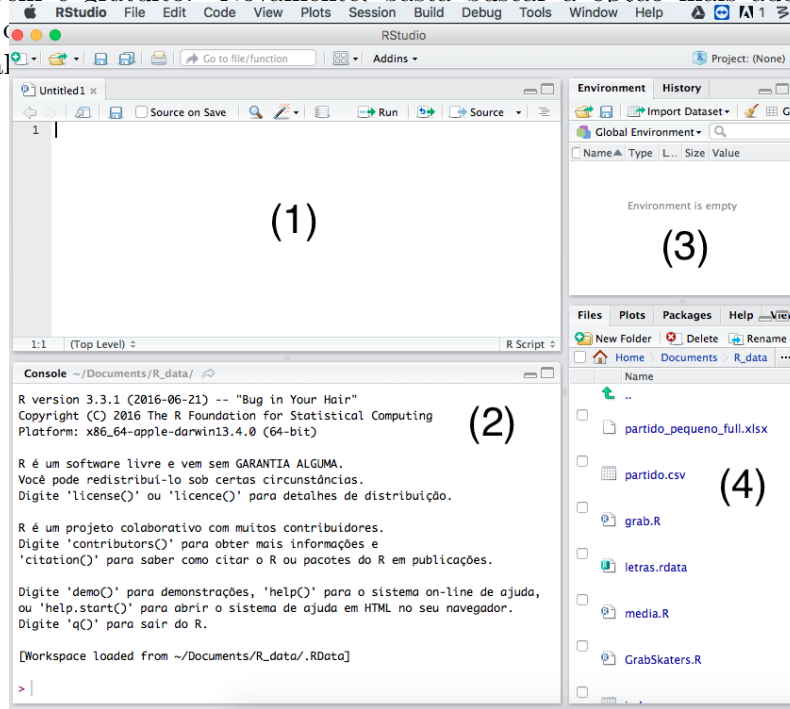


Figura 1.2: RStudio

1 Básico

1.2.1 O RStudio

No RStudio, temos 4 sub-janelas por padrão, isto é, a janela do *software* é dividida em quatro áreas diferentes, como ilustra a Figura. Um resumo da utilidade de cada uma:

1. Janela de *script* na qual escrevemos e documentamos nossos códigos;
2. Console do R, onde podemos executar código e, também, ter retorno de mensagens de erro e avisos;
3. Nesta sub-janela temos duas abas principais:
 - i. *Environment*, na qual visualizamos quais objetos estão na memória do R (e.g., vetores, banco de dados, listas);
 - ii. *History*, na qual podemos ver o histórico dos códigos que já executamos;
4. Aqui temos cinco abas principais:
 - i. *Files*, onde é possível visualizar a lista de arquivo da pasta (área de trabalho, no jargão do R) em que você está trabalhando no seu computador;
 - ii. *Plots*, na qual podemos visualizar gráficos criado no R;
 - iii. *Packages*, que exibi pacotes de funções instalados no R; e, (d) *Help*; onde será visualizadas as ajudas solicitadas dentro do próprio programa;
 - iv. *View*, usada para visualizar o resultado da execução de certas funções.

De início, o mais importante é pensar no RStudio como uma espécie de pacote Office, mas para o R: é nele que escreveremos nossos códigos, executaremos e visualizaremos os seus resultados.

1.2.2 Usando o R e o RStudio pelo navegador

Para quem tem problemas ao instalar o R, ou não pode instalá-los por qualquer razão, há uma alternativa simples pela nuvem: o Posit Cloud, uma plataforma mantida pela mesma empresa do RStudio que permite o seu uso diretamente pelo navegador, sem a necessidade de instalação. Para usá-lo, basta criar uma conta no site, selecionar o plano gratuito (*Free forever*) e começar a usar o R de lá. A tela do seu navegador deverá mostrar algo como na Figura 1.3.

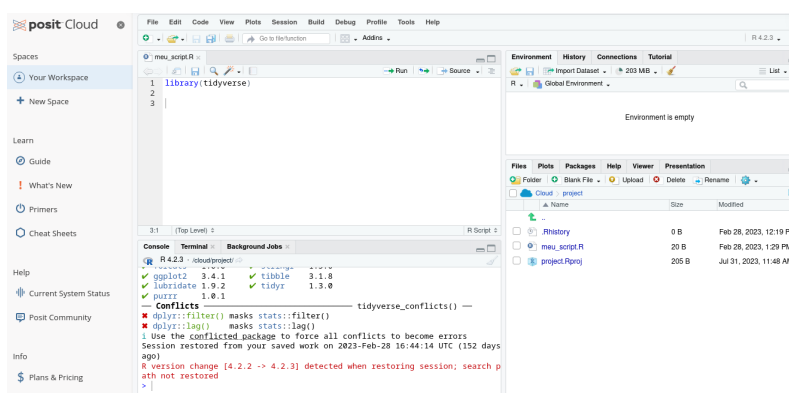


Figura 1.3: Usando o RStudio do navegador

1.3 Sintaxe básica do R

A partir de agora, começaremos a aprender R do jeito mais direto possível: escrevendo e executando códigos. Para tanto, as próximas seções começarão a introduzir exemplos de código que, a princípio, podem parecer confusos. Mas não se preocupe: o objetivo é aprendermos R de forma prática, sem memorizações, entendendo o que cada parte de um código faz.

1 Básico

Daqui até o final do livro, o seguinte se aplicará:

- Tudo o que estiver em caixa cinza, com texto destacado por cores, é código e pode ser executado no R (basta copiar e colar o código no console do RStudio, janela (2) na Figura 1.1, e apertar *enter*);
- Tudo o que estiver logo após precedido de um [1] ou algo do tipo é *output* do R, isto é, o resultado da execução de um código;
- Alguns códigos dependem de códigos anteriores; caso encontre algum erro ao rodar um código de exemplo neste livro, tente voltar atrás e rodar os códigos anteriores.

1.3.1 R como uma calculadora

Assim como em outras linguagens de programação, podemos usar o R como uma calculadora. Experimento digitar $2 + 2$ no console do RStudio e apertar *enter*:

```
2 + 2
```

```
[1] 4
```

O R reproduzirá o resultado da soma antecedido por [1]. Aproveitando a deixa, # indica um comentário: tudo o que vem sucedido de # o R não executará.

```
# 2 + 2
```

Nada acontece. Comentários são úteis para documentar nossos códigos, algo que veremos em seguida. Por enquanto, experimente usar o console como uma calculadora (logo veremos usos mais interessantes do R):

```
8 + 7 # Adição (depois do #, nada é executado)
```

1.3 Sintaxe básica do R

```
[1] 15
```

```
8 - 7 # Subtração (depois do #, nada é executado)
```

```
[1] 1
```

Para resolver expressões numéricas, usamos ().³

```
2 / (3 + 5)
```

```
[1] 0.25
```

```
4 * ((2 ^ 5) / 3)
```

```
[1] 42.66667
```

1.3.2 Operadores

Anteriormente, usamos operadores aritméticos, como + e * (você deve ter percebido que * é o operador de multiplicação no R, e não x). No R, existem vários outros (tente adaptar os exemplos):

```
3^2
```

```
[1] 9
```

³No R, [] e {} são reservados para outros usos, como veremos em breve.

1 Básico

```
11 / 5
```

```
[1] 2.2
```

```
11 %/% 5
```

```
[1] 2
```

```
11 %% 5
```

```
[1] 1
```

Caso você não tenha entendido algum apenas pelo seu uso, a Tabela 1.1 apresenta uma descrição dos principais operadores matemáticos comuns em R.

Tabela 1.1: Operadores matemáticos no R

Operação	Símbolo
Adição	+
Subtração	-
Divisão	/
Multiplicação	*
Exponenciação	^
Divisão inteira	%/%
Resto da divisão	%%

Além dos operandos matemáticos, existem também operadores lógicos, que usamos para saber se algo é verdadeiro ou falso. Para sermos mais concretos, podemos usar == (dois =) para testar se um número é igual a outro:

1.3 Sintaxe básica do R

```
1 == 1
```

```
[1] TRUE
```

O que o código anterior faz é testar se 1 é igual a 1, retornando TRUE. Um exemplo falso:

```
2 == 1
```

```
[1] FALSE
```

Testes lógicos também nos permitem fazer operações mais complexas. Por exemplo, podemos testar se um número é maior ou menor que outro:

```
10 > 5
```

```
[1] TRUE
```

```
3 > 1
```

```
[1] TRUE
```

E, indo além, podemos combinar dois testes usando o operador `&` (que significa E, ou AND):

```
(10 > 5) & (3 > 1)
```

```
[1] TRUE
```

1 Básico

```
(10 > 5) & (5 < 2)
```

```
[1] FALSE
```

No caso acima, o resultado de cada expressão só será **TRUE** se ambos os testes forem verdadeiros. Se quisermos que o resultado seja **TRUE** se pelo menos um dos testes for verdadeiro, usamos o operador **|** (ou):

```
(10 > 5) | (5 < 2)
```

```
[1] TRUE
```

Testes lógicos

Testes lógicos sempre retornam **TRUE** ou **FALSE**, em maiúsculo.

A Tabela 1.2 apresenta os operadores lógicos mais comuns:

Tabela 1.2: Operadores lógicos comuns no R

Operação	Símbolo	Exemplo
Igualdade	==	1 == 1
Diferença	!=	1 != 1
Maior que	>	1 > 1
Menor que	<	1 < 1
Maior ou igual	>=	1 >= 1
Menor ou igual	<=	1 <= 1
E	&	(1 == 1) & (2 == 2)
OU		(1 == 1) (2 == 2)
NÃO	!	!(1 == 1)

Todos esses operadores são úteis – mas certamente não é por causa deles que o R é tão utilizado.

1.4 Funções

Parte da potencialidade do R advém do fato dele conter uma série de funções nativas para realizar as mais diversas tarefas de pesquisa. É por isso que ele é considerado um ambiente, e não apenas uma linguagem de programação.⁴ Dito de forma simples, funções são códigos que executam uma tarefa específica. A função `sqrt()`, por exemplo, calcula a raiz quadrada de um número:

```
sqrt(4) # Raiz quadrada do número 4
```

[1] 2

Em R, funções têm uma anatomia específica: o nome da função, seguido de parênteses, dentro dos quais estão os argumentos da função – o *input* que a função recebe e processa. No caso da função `sqrt()`, o argumento é o número cuja raiz quadrada queremos calcular.⁵ Vale memorizar: uma função nada mais é do que uma espécie de ferramenta que recebe uma determinada informação e a transforma em outra.⁶

⁴Embora ele também seja uma linguagem de programação.

⁵Em R, argumentos são os valores que uma função recebe para executar uma tarefa e, como veremos em seguida, há funções que recebem vários argumentos, alguns deles nomeados.

⁶Há funções que não recebem *inputs*, assim como outros que não retornam *outputs*, mas esses não são os usos mais comuns de funções.

1 Básico

1.4.1 Usando funções

No R, as informações que passamos para determinada função vão dentro de parêntesis. A função `sum`, por exemplo, recebe e soma dois ou mais números, todos separados por vírgula. Se esquecermos de fazer essa separação, obtemos um erro.

```
sum(2 2) # retorna erro
```

```
Error: <text>:1:7: unexpected numeric constant
1: sum(2 2
      ^
```

Erros

Quando executamos um código que o R não consegue interpretar, ele retorna um erro no console.

Para corrigir o código anterior, basta separar os números por vírgula:

```
sum(2, 2) # retorna 4
```

```
[1] 4
```

1.5 Objetos

Para além de executar código, o R nos permite salvar informações na memória do programa. Essas informações são armazenadas em objetos, que podem ser usados posteriormente. De forma bem simples, objetos são como locais na memória do programa que armazenam valores quaisquer. No R esses valores podem ser: números, textos, um vetor de números (isto

1.5 Objetos

é, uma sequência de números), um banco de dados e, até mesmo, uma função.

Podemos armazenar objetos no R com o operador `<-` (menor que, seguido de hífen). Basicamente, ele diz ao R para armazenar um valor em um objeto para podermos acessá-lo posteriormente. Exemplo: vamos salvar o número 2 em um objeto chamado `x`.

```
x <- 2
```

Tocamos em algo extremamente importante: agora, podemos digitar `x` no lugar de 2 para realizar outras operações.

```
x
```

```
[1] 2
```

```
x + 1
```

```
[1] 3
```

```
x / 2
```

```
[1] 1
```

E como fazemos para salvar o resultado de uma nova operação, como `x + 10`, por exemplo? Simples: basta criar um novo objeto.

```
y <- x + 10  
y
```

1 Básico

```
[1] 12
```

Algo que ainda não vimos, também é possível armazenar texto em um objeto – note que, para o R reconhecer algo como texto, precisamos colocá-lo entre aspas:

```
texto <- "um texto"
texto
```

```
[1] "um texto"
```

No R elementos entre aspas, simples ou duplas, são considerados textos.

i Criação de objetos

No R também é possível criar objetos usando o símbolo de igualdade, =, como em `x = 1`. No entanto, não usaremos essa sintaxe neste livro e, por razões de consistência de código, também não recomendamos seu uso.

1.5.1 Tipos de objeto

Números são diferentes de textos e, em R, essa diferença também existe: ela é dada pelas classes de objetos. Classes são como categorias de objetos, isto é, grupos de objetos que compartilham de uma mesma estrutura e, portanto, podem ser manipulados de forma semelhante. O número 1 é um objeto da classe **integer** (inteiro), assim como os números 2 e 10, que também são inteiros. O número 1,5, ao contrário, é um objeto da classe **numeric**, dado que não é um número inteiro (por conta da casa decimal). Para saber a classe de um objeto, usamos a função `class()`:

```
class(1)
```

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

```
class(1.5)
```

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

Diferentes funções podem exigir diferentes classes de objetos. Por exemplo, a função `sum()` exige que os objetos que ela soma sejam da classe `numeric` ou `integer`. Se tentarmos somar um objeto da classe `character`, o R retornará um erro:

```
sum("1", "2")
```

```
Error in sum("1", "2"): 'type' inválido (character) do argumento
```

Para resumir, classes determinam o tipo de informação que diferentes objetos armazenam e o que podemos fazer com elas. Entendido isso, podemos começar a aprender sobre as classes mais comuns no R: `integer`, `numeric`, `character`, `factor`, `matrix`, `data.frame` e `list`.

1.5.1.1 Números, textos e categorias

1.5.1.1.1 integer

`integer` é uma classe de objeto específica para números inteiros.

```
exemplo_inteiro <- 20  
class(exemplo_inteiro)
```

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

1 Básico

Até agora, só criamos objetos com um elemento, mas, quando estamos analisando muitos dados, podemos combiná-los em vetores, ou seja, objetos com mais de um elementos (mais de um caso). Uma forma elementar de criar um vetor é por meio da função *combine*, *c*:

```
# Cria um vetor de números  
x <- c(18, 20, 19, 25, 21)  
x
```

```
[1] 18 20 19 25 21
```

1.5.1.1.2 numeric

A classe **numeric** também é composta por números, mas, diferentemente de **integer**, armazenam números decimais.

```
exemplo_decimal <- 20.5  
class(exemplo_decimal)
```

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

Por padrão, o R já atribui classe aos objetos quando os criamos, deduzindo o tipo adequado a partir do nosso código. No caso de **integer** ou **numeric**, a escolha está atrelada à quantidade de memória reservada no programa para armazenar as informações: quando temos números decimais, a classe sempre será **numeric** pois é necessário mais espaço para guardar informações das casas decimais, e todos os números do vetor passaram a ter um decimal, mesmo aqueles que foram declarados (inseridos) sem decimal:⁷

⁷Na verdade, em R, vetores de inteiros são armazenados como **numeric**, o que você pode ver por conta própria rodando `class(c(1, 2, 3))`.


```
y <- c(50, 65.5, 55.8, 70, 85.6)
class(y)
```

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

Decimal

O R adota o sistema de casas decimais americano, com ponto. Por isso, ao declarar um número decimal no R, usamos o ponto, e não a vírgula.

1.5.1.1.3 character

Como já dito, `character` é a classe usada no R para armazenar informações textuais, que devem estar contidas entre aspas.

```
w <- c("superior", "médio", "fundamental", "superior")
class(w)
```

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

1.5.1.1.4 factor

Similar a `character`, `factor` é uma classe que guarda simultaneamente uma informação textual com uma numérica associada – o que costumamos chamar de variável categórica nas Ciências Sociais e similares.

```
z <- factor(c("Feminino", "Masculino", "Feminino", "Masculino", "Feminino"))
class(z)
```

1 Básico

```
[1] "factor"
```

```
z
```

```
[1] Feminino Masculino Feminino Masculino Feminino  
Levels: Feminino Masculino
```

Como podemos ver pelo retorno do R anterior, um vetor da classe **factor** nos mostra seus *levels*, ou seja, as categorias da nossa variável: **Feminino** e **Masculino**. Mas, como podemos ver, o R não nos mostra os valores numéricos associados a cada categoria. Para isso, podemos usar a função **as.numeric()**, que converte objetos de outras classes para **numeric** (quando essa conversão for possível):

```
as.numeric(z)
```

```
[1] 1 2 1 2 1
```

1.5.1.2 Matrizes e bancos de dados

1.5.1.2.1 matrix

A classe **matrix** é um tipo de objeto bidimensional utilizada principalmente para representar linhas e colunas. De forma geral, matrizes são espécies de tabelas ou planilhas como as que vemos no Excel, mas com uma diferença essencial: todos os elementos devem ser do mesmo tipo, isto é, todos **numeric**, **integer**, **character**, e assim por diante.

Podemos criar uma matriz com a função **matrix**, declarando argumentos que indicam quantas linhas e quantas colunas essa matriz deverá ter. Um exemplo de matriz:

```
matriz <- matrix(c(1, 3, 4, 5, 6, 7), nrow = 2, ncol = 3)
matriz
```

```
      [,1] [,2] [,3]
[1,]    1    4    6
[2,]    3    5    7
```

```
class(matriz)
```

```
[1] "matrix" "array"
```

Note que, no exemplo anterior, criamos uma matriz com 2 linhas e 3 colunas e passamos a ela um vetor com os elementos `c(1, 3, 4, 5, 6, 7)`. Em outras palavras, os argumentos `nrow` (i.e., número de linhas) e `ncol` (i.e., número de colunas) determinam como o conteúdo da matriz será dividido entre linhas e colunas.

1.5.1.2.2 data.frame

Já que matrizes salvam apenas informações da mesma classe, naturalmente precisamos de outra classe se quisermos analisar variáveis, ou colunas, de classes diferentes. `data.frame` é exatamente a classe que nos permite fazer isso. Especificamente, `data.frame` também é bidimensional e tabular, como a `matrix`, mas é mais versátil.

Vamos criar aqui um banco de dados a partir de vetores com a função `data.frame`:

```
x <- c("Superior", "Médio", "Médio")
y <- c(23, 45, 63)
z <- c("Feminino", "Masculino", "Masculino")
```

1 Básico

```
banco <- data.frame(escolaridade = x, idade = y, sexo = z)
class(banco)
```

```
[1] "data.frame"
```

Com o banco criado, podemos ver suas informações com a função `print`, que serve para mostrar no console o conteúdo de um objeto:

```
print(banco)
```

	escolaridade	idade	sexo
1	Superior	23	Feminino
2	Médio	45	Masculino
3	Médio	63	Masculino

Para o caso de bancos maiores, podemos usar a função `View()`, que abrirá uma nova janela no RStudio com o conteúdo do banco de dados.⁸

data.frames

Para criar matrizes e bancos de dados a partir de vetores, todos eles precisam ter o mesmo número de elementos, caso contrário o R retornará um erro.

1.5.1.3 Listas

Finalmente, os objetos da classe `list` são um dos mais complexos que veremos – eles são multidimensionais. Em particular, com eles armazenamos

⁸Note que, para usar a função `View()` adequadamente, precisamos que o RStudio esteja instalado no computador.

1.5 Objetos

objetos de diferentes classes, mas não só vetores do mesmo tamanho como em um `data.frame`. Ou seja, em um objeto tipo `list` podemos armazenar vetores de diversos tamanhos, `matrix` e `data.frame`, ou mesmo outras listas. Vejamos um exemplo:

```
# Cria uma lista chamada 'guarda_trecos'
guarda_trecos <- list(x, y, z, banco)
class(guarda_trecos)
```

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

```
print(guarda_trecos)
```

```
[[1]]
```

```
[1] "Superior" "Médio"    "Médio"
```

```
[[2]]
```

```
[1] 23 45 63
```

```
[[3]]
```

```
[1] "Feminino" "Masculino" "Masculino"
```

```
[[4]]
```

	escolaridade	idade	sexo
1	Superior	23	Feminino
2	Médio	45	Masculino
3	Médio	63	Masculino

Como podemos ver cada item (objeto) foi armazenado na lista `guarda_trecos`, na ordem em que foram colocado dentro da função `list()`.

1.5.2 Manipulando objetos

Criamos alguns objetos de distintas classes e exibimos eles por completo no console. Mas e se quisermos apresentar no console apenas um elemento de um objeto? Para isso precisamos nos mover pelos objetos usando índices. Ao exibir elementos de um objeto no console, o R há nos dá uma dica de como fazer isso: o [1] sempre indica o conteúdo do primeiro elemento. Se quisermos acessá-lo, basta executar:

```
x <- c(1, 2, 3, 4, 5)
x[1]
```

```
[1] 1
```

De forma geral, em objetos unidimensional basta usar `objeto[índice]`, com a posição desejada em colchetes, para acessar determinado elemento, como o quarto e o quinto, digamos:

```
x[4]
```

```
[1] 4
```

```
x[5]
```

```
[1] 5
```

```
x[c(4, 5)] # Podemos outro vetor para acessar mais de um elemento
```

```
[1] 4 5
```

1.5 Objetos

Em objetos multidimensionais como um `data.frame` o modo de acesso de um elemento é um pouco diferente. Por exemplo, no nosso objeto `banco` criado anteriormente precisamos indexar linhas e colunas, `objeto[linhas, colunas]`. Para acessar a célula da primeira linha e da terceira coluna, usamos:

```
banco[1, 3]
```

```
[1] "Feminino"
```

No exemplo acima, estamos selecionando o elemento (caso) número 1 que está na coluna (variável) 3 que é o sexo. É importante fixar: em objeto bidimensional como um `data.frame`, antes da vírgula nos colchetes temos as linhas e, só depois da vírgula, as colunas. Outro caso:

```
banco[, 3]
```

```
[1] "Feminino" "Masculino" "Masculino"
```

Quando deixamos o do lado esquerdo do colchete vazio, estamos dizendo ao R que retorne um vetor com todas as linhas (casos) da coluna (variável) identificada no lado direito da vírgula. Nesse exemplo, temos o sexo de todas as pessoas no `banco`. Já aqui, pegamos todas as informações da pessoa na segunda linha do `banco`:

```
banco[2, ]
```

```
  escolaridade idade      sexo  
2         Médio   45 Masculino
```

1 Básico

Se quisermos selecionar mais um caso ou variável podemos usar um vetor, também podemos usar vetores usando a função `c` ou dois pontos, para criar uma sequência de inteiros entre dois números:

```
banco[3:5, c(1, 3)]
```

	escolaridade	sexo
3	Médio	Masculino
NA	<NA>	<NA>
NA.1	<NA>	<NA>

No exemplo acima estamos selecionando os casos de 3 a 5 (o código `3:5` cria uma sequência de inteiros de 3 a 5) da base de dados e as variáveis 1 e 3.

Indexadores também funcionam em listas, mas com uma diferença: como listas são objetos multidimensionais, precisamos usar dois conjuntos de colchetes para acessar elementos. Por exemplo, para acessar o primeiro elemento da lista `guarda_trecos`, usamos:

```
guarda_trecos[[1]]
```

```
[1] "Superior" "Médio"    "Médio"
```

O primeiro conjunto de colchetes indica que queremos acessar um elemento da lista, enquanto o segundo indica qual elemento queremos acessar. Se quisermos acessar um valor dentro do primeiro elemento da lista, basta adicionar um colchete simples logo depois dos colchetes duplos indicando o índice do elemento desejado:

```
guarda_trecos[[1]][2]
```



```
[1] "Médio"
```

Com isso, selecionamos o segundo elemento do vetor armazenado na sublista 1. E se o conteúdo da sublista for um `data.frame`, como acesso um valor dentro dele? Assim:

```
guarda_trecos[[4]][1, 3]
```

```
[1] "Feminino"
```

Para `data.frames`, há um jeito mais simples de se acessar o conteúdo inteiro de uma variável: por meio do cifrão (`$`). Por exemplo, para acessar a variável `sexo` do `banco`, basta executar:

```
banco$sexo
```

```
[1] "Feminino" "Masculino" "Masculino"
```

Como dá para notar, é preciso saber o nome da coluna que queremos acessar para usar esse meio de indexação. Um jeito simples de fazer isso é usando a função `names()`, que retorna os nomes das colunas de um `data.frame`:

```
names(banco)
```

```
[1] "escolaridade" "idade"          "sexo"
```

Assim sabemos que a primeira variável se chama “escolaridade”, a segunda “idade”, e assim por diante.

Combinando o `$` com os indexadores que vimos há pouco, é fácil obter, por exemplo, o terceiro elemento da variável `sexo` no objeto `banco`:

1 Básico

```
banco$sexo[3]
```

```
[1] "Masculino"
```

Manipular objetos no R pode parecer bastante complicado, mas, com o devido tempo e prática, tudo se torna mais simples. Ao final deste capítulo, sugerimos alguns exercícios que ajudarão no processo.

1.6 Pacotes

O R já vem com uma série de funcionalidades embutidas nele – como as funções `sqrt` e `sum`, que já vimos. Mas, como já dito, uma das grandes vantagens do R é a sua comunidade, que desenvolve novas funcionalidades para a linguagem e, normalmente, as disponibilizam por meio de pacotes, ou bibliotecas. Estes são como extensões do R, que adicionam novas funcionalidades ao programa – pense em um pacote como um livro de receitas ou um manual de instruções, que ensina o R como fazer coisas novas.

Em R, a principal fonte de pacotes é a *CRAN* (*The Comprehensive R Archive Network*), que é uma comunidade de desenvolvedores que mantém o código base do R e os seus pacotes oficiais, aqueles que passaram por uma série de testes e que seguem uma série de protocolos que garantem o seu funcionamento estável e harmônico com outras ferramentas no R.⁹

⁹Enquanto escrevemos este livro, o CRAN se aproxima de 20 mil pacotes oficiais mantidos em seu *website*. Informação disponível em: <https://cran.r-project.org/web/packages/>.

1.6.1 Instalando Pacotes

Para instalar pacotes que está no CRAN, basta sabermos o seu nome e usar a função `install.packages`:

```
install.packages("electionsBR")
```

No exemplo acima, instalamos o pacote `electionsBR` e, com ele, damos ao R a capacidade de se conectar ao repositório de dados eleitorais do TSE (Tribunal Superior Eleitoral) para obter informações eleitorais.

1.6.2 Instalação de pacotes do GitHub

Apesar da imensidade de pacotes no CRAN, encontramos outro grande volume de pacotes em outros repositórios não-oficiais, a maioria em desenvolvimento. A principal fonte destes pacotes, depois do CRAN, é o GitHub.¹⁰

Para instalar pacotes do GitHub, precisamos instalar outro pacote antes, o `remotes`:

```
install.packages("remotes")
```

Este pacote contém uma função, `install_github`, que permite ao R se conectar ao GitHub, obter de lá o código fonte de um pacote e realizar a sua instalação. Para usar esta função, precisamos antes carregar o pacote `remotes`, isto é, tornar ela acessível ao R, o que fazemos por meio da função `library`:

¹⁰GitHub é uma plataforma de armazenamento e versionamento de *software* criado em cima do Git, um *software* de código aberto de controle de versões. Atualmente, o GitHub é um dos maiores repositórios de código aberto no mundo. Para acessá-lo, visite <https://github.com>.

1 Básico

```
library(remotes) # Carrega o pacote  
install_github("silvadenisson/electionsBR") # Instala o pacote
```

Instalar e carregar pacotes são duas tarefas similares, mas suas diferenças são importantes: no primeiro caso, estamos incorporando novas funções no nosso R, assim como instalar o Office no computador nos permite usar o processador de texto Word; no segundo caso, estamos carregando o pacote instalado, assim como quando abrimos o Word pelo seu atalho no computador.

i Pacotes

Pacotes só precisam ser instalados uma vez, mas precisam ser carregados (abertos) no R em cada seção em que precisarmos de suas funções.

No exemplo anterior, usamos a função `library` para carregar o pacote `remotes`; com este, usamos a função `install_github` para instalarmos a versão de desenvolvimento do pacote `electionsBR`.

1.7 Scripts

Trabalhar no console, digitando e executando código diretamente de lá, é algo rápido para tarefas simples, mas inviável para análises mais complicadas. Pior que isso, sem poder salvar nossos códigos em algum lugar, não temos como reproduzir uma análise, ou compartilhar nossos passos com outras pessoas. Justamente para evitar isso, usamos *scripts*, documentos de texto que servem para documentar e armazenar códigos.

No Rstudio, podemos criar um *script* clicando, no menu superior esquerdo, em *File > New File > R Script*, ou, também no canto superior esquerdo, no símbolo de uma folha em branco acompanhada de um símbolo de mais em verde. Feito isso, uma nova janela será aberta, na qual podemos escrever

1.7 Scripts

nosso códigos. Para salvá-los, basta clicar em *File > Save* e escolher um nome para o arquivo, ou clicar no ícone de disquete ligeiramente acima, ou, ainda, teclar *ctrl/command + s*. O *script* salvo aparecerá na sub-janela de gestão de arquivos do RStudio, indicada no item 4 da Figura 1.2.

Para acompanhar o restante deste capítulo e os próximos, acostume-se a criar *scripts* e use comentários para descrever o que cada linha faz – isso será muito útil para documentar o que estamos aprendendo. A título de exemplo, um *script* de acompanhamento deste capítulo poderia ter o seguinte início:

```
# Capítulo 1: Introdução ao R

# Este é um comentário. Ele não é executado pelo R, mas serve para documentar o que estamos
# Para executar um código, basta clicar na linha e teclar ctrl/command + enter.
# Para executar várias linhas, basta selecioná-las e teclar ctrl/command + enter.

# Criando objetos
x <- 2
y <- x + 10

# ...
```

Documentar o script é uma das tarefas mais importantes do desenvolvimento do seu código. Primeiro porque podes voltar em um outro momento e saber o que exatamente estas tentando fazer com seu script. Isso pode parecer tolice, mas tenha certeza que não é, principalmente quando chegamos no nível de trabalhar com muitos scripts.

Esse motivo acima já seria suficiente, no entanto, há outro mais importante para o desenvolvimento de pesquisas científicas que é a replicabilidade. Pois, quando documenta teu código aumenta a capacidade replicativa dele. E replicabilidade é a palavra que chave na ciência, porque não fazemos ciência para ficar na gaveta, ou melhor, em pasta perdida dentro do

1 Básico

computador, e sim para que outra pessoa saibam o que fizemos e possam replicar, vamos abordar mais sobre replicação no capítulo 8.

1.8 Recomendações

Podemos criar objetos e realizar operações no R de forma simples, como vimos. No entanto, algumas coisas devem ser evitadas quando escrevemos nosso código, seja para evitar erros ou para facilitar a leitura dele por outras pessoas.

A recomendação mais básica neste sentido é: evite criar objetos com nomes que comecem com números, caracteres especiais ou nomes de funções. Algumas destas coisas produzirão erros imediatos; outras, podem complicar códigos inteiros. Alguns exemplos.

```
# Exemplos de nomes de objetos que produzem erros
2x <- 1
_x <- 1
&x <- 1
```

```
Error: <text>:2:2: unexpected symbol
```

```
1: # Exemplos de nomes de objetos que produzem erros
2: 2x
   ^
```

```
# Exemplos de nomes de objetos que não produzem erros imediatos
sum <- 1
sqrt <- 1
```

Também note que o R é *case sensitive*, A (maiúsculo) não é a mesma coisa que a (minúsculo).

```
A <- 1  
print(a)
```

Error in eval(expr, envir, enclos): objeto 'a' não encontrado

Sempre que criar um objeto armazenando texto, não esqueça das aspas (outra forma de cometer erros no R bastante comum).

```
x <- Texto
```

Error in eval(expr, envir, enclos): objeto 'Texto' não encontrado

Por fim, quando abrir parênteses, não esqueça de fechá-los (caso contrário, aparecerá um + no console, indicando que o R espera mais conteúdo). Caso esteja executando um código e não saiba porque apareceu um + no console, opte por cancelar a operação e volte ao código para ver o que há de errado.¹¹

1.8.1 Estilo

Não é algo obrigatório, mas algumas noções de estilo nos ajudam a compreender e partilhar códigos, tanto nossos quanto os de outras pessoas. Resumidamente, as principais considerações aqui são:

- Use espaços entre objetos, operadores e chamadas a funções;
- Use quebra de linhas para separar blocos de códigos;
- Sempre que possível, crie objetos apenas com letras minúsculas;
- Se precisar separar o nome de um objeto, use `_` (underscore);

¹¹Estas e outras recomendações comuns para evitar erros podem ser vistas em: <http://www.alex-singleton.com/R-Tutorial-Materials/common-error-msg.pdf> (em inglês).

1 Básico

- Prefira nomes curtos para objetos;
- Prefira o atribuidor `<-` a `=` (eles fazem a mesma coisa).

```
y<-1 # Ruim  
y <- 1 # OK  
  
y+y+y+y # Ruim
```

```
[1] 4
```

```
y + y + y + y # OK
```

```
[1] 4
```

```
print (y) # Ruim
```

```
[1] 1
```

```
print(y) # OK
```

```
[1] 1
```

```
y = 1 # Ruim  
y <- 1 # OK  
  
OBJETO <- 1 # Ruim  
objeto <- 1 # OK
```



```
meu.objeto <- 1 # Ruim
meu_objeto <- 1 # OK

objeto_com_nome_excessivamente_grande <- 1 # Ruim
objeto <- 1 # OK
```

1.9 Obtendo ajuda

Para a nossa sorte a comunidade em torno do R cresceu muito nos últimos anos, o que aumentou a quantidade de material disponível na internet sobre a linguagem. De toda forma, a maneira mais simples de obter ajuda no R sobre alguma função ou operador é consultando a documentação – em geral, muito boa. Para isso, podemos usar a função `help`:

```
help(sum)
```

Esse recurso só é útil quando sabemos o nome exato da função que queremos consultar e, como é possível imaginar, quando essa função pertence a algum pacote e este pacote está instalado e carregado devidamente. Outra forma de consultar documentação é usando um ponto de interrogação antes do nome de uma função:

```
?sum
```

Quando não sabemos o nome da função que queremos usar, ou até mesmo para saber se existe no R uma função específica para a tarefa que queremos executar, temos recorrer a outras fontes. Antes mesmo de ir para o Google, no entanto, há no próprio R um pacote que faz uma busca por palavras-chave do que há no R, o `sos`. Para usá-lo, precisamos instalá-lo e, depois, carregá-lo:

1 Básico

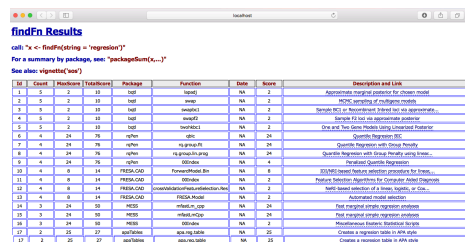
```
install.packages("sos")
```

```
library(sos)
```

E, então, usar a função `findFn`, que tem como argumento principal um texto (*string*) que será pesquisado. Exemplo:

```
findFn('regresion')
```

Quando executada, a função irá abrir seu navegador em uma página com o resultado, como a Figura 1.4 ilustra.



The screenshot shows a web browser window titled "findFn Results". The page contains the following text:

```
cmd: "s <- findFn(string = 'regresion')"
```

For a summary by package, use: "packageFn(s,...)"

See also: vignette("sos")

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

Figura 1.4: Pacote sos

Que nos apresenta o nome do pacote, o nome da função que tem algum relação com o termo pesquisado, e uma breve descrição onde podemos clicar e ver mais detalhes da função.

Alem dessas opções a internet está cheia de material, e acada dia cresce mais, por exemplo nos ultimos anos tem almentado exponencialmente a quantidade de tutoriais no yourtube. Mas, o que vamos destar a aqui é o stackoverflow, R-bloggers, R Brasil - Programadores (Facebook), rbloggersbr (twitter).

O Primeiro deles é um fórum onde porgramadores de todos os níveis e linguagens publicam problemas e soluções. Originalmente em inglês, mas que agora conta com uma versão para que os problemas e soluções

1.9 Obtendo ajuda

sejam postado em português o que facilita na expansão das linguagens de programções em especial, R. O endereço é <https://pt.stackoverflow.com>. Para refinar as busca dentro do fórum é necessario antes do termo buscado inserir o nome da linguagem dentro de conchetes.

Ex.: [R] data.frame

O R-bloggers é um site que reuni postagens de varios blogs sobre R, ou seja, é concentrador de postagens, assim podemos acompanhar o desenvolvimento da linguagem por um unico canal, é inglês. <https://www.r-bloggers.com/tag/rblogs/>

Em português há uma iniciativa no Twitter para agregar as postagens do blogs brasileiros cadastrados, <https://twitter.com/rbloggersbr>.

Há também no facebook grupos sobre R, em português o principal deles é o R Brasil - Programadores. Como podemos ver temos varios canais para buscar informações sobre o R. Mas claro caso seja necessario o google está ai para nos ajudar também, para facilitar podemos buscar algo tipo “regressão linear r”, e variações.

O próximo capítulo começamos aprofundar o que vimos por até aqui. Mas especificamente sobre dados, criar objetos, importação e etc.

References

