

4 Tipos de Variáveis Comuns

Estamos prontos agora para pôr R em operação. Vamos iniciar com alguns exercícios e terminar esta capítulo com a manipulação e análise exploratório de um dataset interessante. Seria um boa prática para você replicar estes exemplos no seu R.

4.1 O Que São Dados?

Parece que esta deve ser uma pergunta simples. Mas, vamos a esclarecer no contexto dos dados em R. No primeiro capítulo expliquei o conceito de atribuição e mostrei o símbolo `<-` que usamos invés de sinal de igualdade (`=`) para associar um nome com alguns valores. Esses valores podem ser numéricos ou texto, podem ser até listas ou matrizes. Mas, sempre usamos a convenção

```
nome do objeto <- definição do objeto
```

onde a definição é os valores são o conteúdo do objeto.

4.2 Exemplos de *Assignment*

```
x <- 5
```

```
x
```

```
## [1] 5
```

```
class(x)
```

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

Aqui a letra `x` vai assumir o valor 5. Ou seja, `x` vai ser definido por 5. Podemos colocar na tela simplesmente colocando o nome do objeto. Este objeto nós podemos também chamar uma *variável*. Utilizando a função `class()`, nós podemos ver que esta variável é do tipo ou classe *numeric*. Variáveis da classe *numeric* também podem ter valores decimais como `5.5`.

Existe um outro tipo importante de classe das variáveis numéricas: números inteiros. Números inteiros são representados dentro do computador por menos bytes por número mas não têm diferenças em operação importantes. Um número inteiro está marcado em R com o sufixo `L`. Porque L e não outra letra? Quem quer especular? Aqui é um exemplo.

```
var_integer <- 5L  
var_integer
```

```
## [1] 5
```

```
class(var_integer)
```

```
## [1] "integer"
```

Se você não inclui o “L” como sufixo, R vai interpretar o número como *numeric*.

Podemos definir nossa variável numérica `x` também através de uma operação aritmética:

```
x <- 3 + 2  
x
```

```
## [1] 5
```

Também, nós podemos usar funções para definir o valor de `x`. Aqui usei o raiz quadrado. Mas, têm muitas outras funções que você pode utilizar.

```
x <- sqrt(675.3)
```

```
x
```

```
## [1] 25.98653
```

Agora, deixamos usar um teste lógico para definir o valor de uma variável.

```
var_logical <- 1 == 0
```

```
var_logical
```

```
## [1] FALSE
```

```
class(var_logical)
```

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

Esta vez, demos a `var_logical` o valor que é um valor lógico: 1 não igual a 0, então o resultado deve ser `FALSE`. Valores lógicos só podem assumir os valores `TRUE` ou `FALSE`.

Finalmente, entre os tipos de variáveis mais comuns, temos texto, chamado aqui *character*.

```
text_var <- "Hello, world!"
```

```
text_var
```

```
## [1] "Hello, world!"
```

```
class(text_var)
```

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

Qualquer caracteres entre aspas (seja simples ou dupla, como aqui) vai ser visto como texto e ter a classe `character`. Existem funções e pacotes especializados que trabalham com essa classe de variável que aprenderemos mais tarde.

A definição de uma variável pode ser um pouco mais complicado ainda. Vamos fazer um cálculo usando nossos dados do primeiro capítulo. Aqui é o código (no formato levemente alterado):

```
set.seed(1)
dados <- runif(100, min = 0, max = 1000)
m <- sum(dados)/length(dados)
m
```

```
## [1] 517.8471
```

Aqui, temos três linhas que fazem operações e a quarta que relata o resultado. Vamos olhar em cada uma e ver o que faz.

`set.seed(1)` : `set.seed()` é um comando que fornece às funções que calculam valores aleatórios a instrução de sempre iniciar o cálculo na mesma posição. Pode aceitar qualquer número inteiro como argumento. Aqui usei `1`. Mas, podia ter usado `43` ou `2019`. Qualquer número inteiro que você usa produzirá uma série de números aleatórios diferentes.

`dados <- runif(100, min = 0, max = 1000)` : `dados` é uma variável receberá 100 valores aleatórios entre 0 e 1000 (`min` e `max`). Estes valores são baseados na distribuição de números “uniforme”, que quer dizer que todos os números entre os 2 limites têm uma chance igual de ser selecionados. O resultado `dados` vai ser um vetor com os 100 números.

`m <- sum(dados)/length(dados)` : para calcular `m`, nós vamos somar os 100 valores e dividir (`/`) o resultado pelo número de valores na variável (neste caso 100). Este é a

mesma coisa que pedir a média de uma série de números, utilizando o cálculo que você vai lembrar de seu curso de estatística:

$$\mu = \frac{\sum_{i=1}^n x_i}{n}$$

Você se lembrou desta formula, não?

Também, podemos usar a função `mean(dados)` para conseguir o mesmo resultado.

`m` : finalmente, indicando o nome da variável que cuja valor vai ser colocado na tela.

Se você quer explorar mais esses conceitos e as funções específicas, posso recomendar que você guarda o *cheatsheet* **Base R** aberta numa tela para consultas. Lembre que pode fazer um download do site de RStudio (<https://www.rstudio.com/resources/cheatsheets/>). É o segundo cheatsheet na lista dos “Contributed Cheatsheets”.

Base R
Cheat Sheet

Getting Help

Accessing the help files

`?mean`
Get help of a particular function.
`help.search('weighted mean')`
Search the help files for a word or phrase.
`help(package = 'dplyr')`
Find help for a package.

More about an object

`str(iris)`
Get a summary of an object's structure.
`class(iris)`
Find the class an object belongs to.

Using Packages

`install.packages('dplyr')`
Download and install a package from CRAN.

`library(dplyr)`
Load the package into the session, making all its functions available to use.

Vectors

Creating Vectors

<code>c(2, 4, 6)</code>	<code>2 4 6</code>	Join elements into a vector
<code>2:6</code>	<code>2 3 4 5 6</code>	An integer sequence
<code>seq(2, 3, by=0.5)</code>	<code>2.0 2.5 3.0</code>	A complex sequence
<code>rep(1:2, times=3)</code>	<code>1 2 1 2 1 2</code>	Repeat a vector
<code>rep(1:2, each=3)</code>	<code>1 1 1 2 2 2</code>	Repeat elements of a vector

Vector Functions

<code>sort(x)</code> Return x sorted.	<code>rev(x)</code> Return x reversed.
<code>table(x)</code> See counts of values.	<code>unique(x)</code> See unique values.

Selecting Vector Elements

By Position	
<code>x[4]</code>	The fourth element.
<code>x[-4]</code>	All but the fourth.
<code>x[2:4]</code>	Elements two to four.

Programming

For Loop

```
for (variable in sequence){  
  Do something  
}
```

Example

```
for (i in 1:4){  
  j <- i + 10  
  print(j)  
}
```

While Loop

```
while (condition){  
  Do something  
}
```

Example

```
while (i < 5){  
  print(i)  
  i <- i + 1  
}
```

If Statements

```
if (condition){  
  Do something  
} else {  
  Do something different  
}
```

Example

```
if (i > 3){  
  print('Yes')  
} else {  
  print('No')  
}
```

Functions

```
function_name <- function(var){  
  Do something  
  return(new_variable)  
}
```

Example

```
square <- function(x){  
  squared <- x*x  
  return(squared)  
}
```

Reading and Writing Data

Also see the `readr` package.

(#fig:base_cheat)Base R Cheatsheet

4.3 Projetos e Pastas

Antes de continuar a consideração dos dados, variáveis e demais, vamos considerar projetos em R, algo que vai facilitar nossa vida de pesquisa bastante.