Introdução ao R

Raimundo Marciano de Freitas Neto

2021-03-22

Contents

Chapter 1

Boas vindas

O curso será composto por módulos teóricos e práticos e terá duração de XX horas, distribuídas em XX semanas, conforme o seguinte cronograma:

- S01 00-00 a 00-00 Introdução
- S02 00-00 a 00-00 Introdução
- S03 00-00 a 00-00 Introdução
- S04 00-00 a 00-00 Introdução
- S05 00-00 a 00-00 Introdução
- S06 00-00 a 00-00 Introdução

O curso é essencialmente assíncrono e a frequência será registrada de acordo com a entrega das atividades nos prazos assinalados.

Nas XX-feiras, das XXh às XXhXX, teremos um encontro síncrono para dúvidas via Google Meet.

Chapter 2

O que é o R?

R é uma linguagem de programação muito utilizada para o desenvolvimento de estatísticas e cálculos matemáticos. Entretanto, enquanto linguagem de programação, ela possui muitos usos, como a raspagem de sites, a automatização de download de arquivos, e a produção de dashboards e relatórios, sendo útil para a coleta de dados e para o reporte das análises, além de ser gratuita.

Mas o que significa ser uma linguagem de programação? Basicamente, você irá dar ordens ao computador por meio de comandos. A vantagem de aprender a estruturar os comandos (ao invés de usar um programa estatístico já consolidado, como o Stata) é que você pode customizar a vontade suas análises. Os programas estatísticos trazem muitas opções pré-prontas. Por exemplo, no Stata é comum que você possa clicar em uma janela e solicitar um quadro com as estatísticas descritivas (média, moda, mediana) de um banco de dados. Entretanto, qualquer funcionalidade fora do padrão oferecido por aqueles pacotes precisará ser gerada pelo próprio usuário (e quando você usa algum recurso como o ssc install é porque alguém gastou um tempinho fazendo isso).

Aliás, o Python (outra linguagem de programação) possui funcionalidades bem similares ao R, embora o modo de escrever os programas seja bem diferente. Imagine que o Português e o Espanhol têm o mesmo propósito - a comunicação - e que mesmo tendo uma mesma raiz (e inclusive diversos vocábulos bem parecidos), possuem algumas regras estruturais bem diferentes, o que inclui, por exemplo, as regras de acentuação e o som produzidos pelas letras: qual a diferença de cajá (PT-BR) para caja (ES); qual a diferença de pastel (PT-BR) para pastel (ES)?

É preferível que o analista iniciante dedique-se a uma das duas (R ou Python). Posteriormente, aprender a outra (ou mais alguma que esteja despontando no mercado, como a Julia) será uma tarefa bem mais simples. Isso se deve ao fato de que diversos elementos da programação se repetem entre as diferentes linguagens, especialmente os conceitos. Um IF tem o mesmo propósito em R ou

em Python ou em Java ou em C++, embora a forma como você deve explicar ao computador o que fazer com esse IF será bem diferente em cada linguagem.

2.1 Scripts

2.2 R Studio

2.3 R nas Nuvens

Chapter 3

Variáveis

Os dados são inseridos no R como variáveis. Eles podem ser importados (trazidos de um arquivo para o R) ou digitados no próprio console. Quando você deseja armazenar um valor na memória, você deve atribuí-lo, usando o sinal = ou <-.

```
x = 1
print(x)
```

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

```
y <- 1+1
y
```

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

```
texto = "Esse é um texto. Observe as aspas duplas"
assim = 'Também podem ser usadas aspas simples'
sugestao = 'Use aspas simples sempre que "possível" para delimitar a variável. Os
textos em português costumam usadas aspas duplas.'
print(sugestao)
```

```
## [1] "Use aspas simples sempre que \"possível\" para delimitar a variável. Os \ntextos em portu
```

```
writeLines(sugestao)
```

```
## Use aspas simples sempre que "possível" para delimitar a variável. Os
## textos em português costumam usadas aspas duplas.
```

O R é uma linguagem que automaticamente reconhece o tipo do dado que foi inserido. Por uma questão de comparação, se você estivesse usando uma linguagem *tipada*, como o C++, você precisaria declarar explicitamente o tipo da variável que está sendo criada. Assim, seria um erro você criar uma variável usando

```
dois = 2
```

O C++ não entende o que você está querendo dizer com isso. Você precisaria informar que existe uma variável do tipo *integer* (simplesmente *int*) chamada *dois* e cujo valor é o número inteiro 2. Outra particularidade do R é que as linhas de código não precisam ser finalizadas com um ponto-e-vírgula (;), como acontece no C++.

```
int dois = 2;
```

Além disso, as variáveis em R são mutáveis, inclusive quanto ao tipo. Isso quer dizer que uma vez definidas (ou atribuídas), podem ter seus valores e tipo modificados. Assim, uma variável que antes tinha um número, pode passar a ter um texto ou um booleano.

```
# se a variavel receber um número (integer)
dois <- 2
# e depois receber um texto
dois <- "dois"
# seu conteúdo e seu tipo estarão de acordo com a última atribuição
print(dois)
```

```
## [1] "dois"
```

Por fim, para o R a indentação não é relevante. Há linguagens de programação em que a posição relativa das linhas de código é essencial para determinar se elas fazem parte de um bloco ou não. No R, os blocos são definidos com o uso de chaves ({ }). Ainda iremos explorar com detalhes o uso das condicionais, mas podemos ilustrar o seguinte caso: se o resultado for maior que zero, então, sim, tivemos lucro; caso contrário, não tivemos.

```
resultado <- 3000

#se o resultado for maior que zero
if (resultado > 0) {
  # informe: "Tivemos Lucro!"
  print("Tivemos Lucro!")
  # caso contrário
```

```
} else {
  # informe: "Deu ruim :( "
  print("Deu ruim :(")
}
```

```
## [1] "Tivemos Lucro!"
```

No R, esse alinhamento é apenas um facilitador de leitura do código. Se o código estiver organizado, as pessoas terão mais facilidade em entendê-lo. O código a seguir tem exatamente a mesma funcionalidade.

```
resultado <- 3000
if (resultado > 0) { print("Tivemos Lucro!") } else {print("Deu ruim :(")}
```

```
## [1] "Tivemos Lucro!"
```

Embora seja verdade que o espaço ocupado está menor, quando o código começa a adquirir um alto grau de complexidade, torna-se muito desejável que ele esteja melhor organizado e, preferencialmente, comentado.

No Python, onde a identação faz diferença, o código precisaria seguir uma estrutura baseada em espaços, sendo desnecessário o uso das chaves.

```
#se o resultado for maior que zero
if (resultado > 0):
  # informe: "Tivemos Lucro!"
  print("Tivemos Lucro!")
else:
  # informe: "Deu ruim :( "
  print("Deu ruim :(")
```

No Python, portanto, o que importa para definir que o print("Tivemos Lucro!") está associado ao if(resultado $>$ 0) é o fato de não haver nenhum espaço entre a margem da página e o if; e o fato de haver quatro espaços entre a margem da página e o print. Isso faz com que esse print esteja subordinado ao if; assim como, o segundo print (que também está mais recuado) é subordinado ao else (que não tem recuo).

3.1 Numéricas

O R possui uma abordagem simplificada para o tratamento de números. Em tese, na computação os números possuem várias classificações, como: inteiro

(integer), que não admitem casas decimais, e ponto-flutuantes (float), que possuem casas decimais. Portanto, 3 pode ser um integer, enquanto 3,5 não pode (tem um dígito após a vírgula). Ainda há mais algumas classificações, como o double e o long, mas o integer e o float são suficientes para o que precisamos discutir.

Basicamente existem os inteiros (integers) e os ponto-flutuantes (float). O integer não admite casas decimais, sendo usado para eventos contáveis, como a quantidade de vezes que algo ocorreu. O float admite casas decimais, sendo indicado para representar valores monetários.

Veja, por exemplo, o caso da idade. Se a idade for calculada como a diferença entre anos (2021 - 1988 = 33), o resultado será um número inteiro, que pode ser representado por um integer. Contudo, se uma idade como 12,5 anos for admitida, então será necessário trabalhar com float. Se você está lendo isso em fevereiro, pense na representação correta da idade de alguém que nasceu em dezembro.

Raramente, isso será uma preocupação. Como dito, o R faz a análise automática e, por padrão, categoriza os números dentro do tipo numeric, que aceita tanto integers como floats. Eventualmente, algum erro pode acontecer, como haver algum canto na planilha que está sendo importada em que foi digitado 2x20 ao invés de 2020. Quando o R tentasse ler esse valor (2x20), ele reconheceria um caractere que não é um número e automaticamente tentaria entender isso como sendo um texto.

3.2 Textuais

Para inserir um caractere, uma palavra ou uma frase, é necessário o uso de aspas simples ou duplas. Como no português é comum o uso de aspas duplas, é interessante o uso de aspas simples no R. Isso se deve ao fato de que se você precisar eventualmente incluir as aspas duplas, elas podem ficar dentro das aspas simples.

Por exemplo, é aceitável:

```
texto1 = 'Ela disse: "Ok".'
texto2 = 'E eu respondi "Certo".'
writeLines(c(texto1, texto2))
```

```
## Ela disse: "Ok".
## E eu respondi "Certo".
```

Perceba que as aspas simples estão funcionando apenas como delimitadores. Estão dizendo que tudo que está dentro delas faz parte de um mesmo texto, inclusive as aspas duplas.

```
texto3 <- "Ela disse: "Ok"."
```

O texto3 seria problemático para o R. Assim que ele identifica as primeiras aspas, ele passa a esperar um texto. Quando aparecem mais aspas, ele entende que o texto foi encerrado e não pretende continuar lendo o que vem depois. Com isso, ele estranha que tenha mais texto logo após (Ok".")

3.2.1 Numérica ou textual?

O CNPJ de uma empresa é composto por 14 dígitos numéricos (0 a 9), separados por alguns marcadores (pontos, barras e traços). Por exemplo, pode ser 12.345.678/0001-19. Para guardar essa informação como uma variável numérica, você precisaria omitir as marcações, digitando 12345678000119. Entretanto, existe aqui um problema: o caso dos CNPJs iniciados por 0. Se há um 0 no início do número, ele é sempre ignorado. Por exemplo, 02 sempre será salvo na memória como 2, porque o zero à esquerda, na Matemática, não tem função.

Pensando nisso, pode ser mais interessante salvar o CNPJ como uma variável textual.

```
cnpj <- "12.345.678/0001-19"
```

Com isso não são perdidos os marcadores. Perceba, por fim, que o tratamento desse "número" como texto não tem implicações sobre a análise estatística dos dados. O CNPJ não é um número sobre o qual você realiza operações matemáticas. Você não calcula a média,a moda ou a mediana dos CNPJs e nem cria um gráfico de CNPJs. Ele é um identificador da empresa, tal qual o nome dela.

No caso de pessoas físicas, o mesmo raciocínio vale para o CPF.

> Exceção: os dois últimos dígitos do CNPJ (e do CPF) são um código verificador. Alguns formulários cadastrais usam esses dois últimos dígitos para conferir se o campo foi preenchido corretamente. Essa conferência é feita a partir de um cálculo matemático; logo, há casos específicos em que mesmo o CNPJ pode ser usado para algum cálculo.

3.3 Lógicas

Em muitos casos, é importante ter variáveis que indiquem VERDADEIRO (TRUE) ou FALSO (FALSE).O tipo de variável que admite apenas esses dois valores lógicos é chamado de Booleana (boolean ou bool). Essas variáveis seguem a lógica de primeira ordem. Por exemplo, TRUE AND FALSE resulta em FALSE, enquanto TRUE OR FALSE resulta em TRUE.

```
# AND ou E é representado por &
print(TRUE & FALSE)
```

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

```
# OR ou OU é representado por |
print(TRUE | FALSE)
```

```
## [1] TRUE
```

Variáveis lógicas podem ter muitos usos, como parâmetros em funções ou filtros para listas de dados. Por exemplo, você pode querer que o usuário informe se ele quer imprimir o relatório dos valores ao final. Nesse caso, ele deveria informar que o valor do parâmetro imprimir_relatorio deveria ser TRUE. Caso ele não queira imprimir o relatório, deveria informar imprimir_relatorio como FALSE.

Também podem ser usados para comparações de igualdade (==) ou desigualdade (!=).

```
dois_numero <- 2
dois_texto <- "dois"
# Vamos testar de o numero dois é o mesmo que o dois "por extenso"
dois_numero == dois_texto
```

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

```
# Vamos testar de o numero dois é diferente do dois "por extenso"
dois_numero != dois_texto
```

```
## [1] TRUE
```

O R não entende que “dois” é a forma “por extenso” do número 2. Para o R, há um número 2 e há uma palavra, cujo significado o R desconhece. Então, na comparação entre esse inteiro e essa string, o R entende que há uma diferença (de tipos). Por isso, ele afirma que são diferentes (ou não iguais).

```
dois_numero <- 2
dois_texto <- "2"
# Vamos testar de o numero dois é o mesmo que o dois "caractere"
# O resultado é verdadeiro porque apesar dos tipos serem diferentes, o conteúdo é o me
dois_numero == dois_texto
```

```
## [1] TRUE
```

Existe outro operador de igualdade (===), mas que não será tratado agora.

3.4 Datas

```
today <- Sys.Date()
print(today)
```

```
## [1] "2021-03-22"
```

```
today <- format(today, "%d/%m/%Y")
print(today)
```

```
## [1] "22/03/2021"
```

```
today2 <- format(Sys.Date(), "%d/%m/%Y")
print(today2)
```

```
## [1] "22/03/2021"
```

Chapter 4

Estruturas de Dados

Pode ser desejável juntar vários valores em um único objeto gravado na memória usada pelo R. Por exemplo, você pode querer criar um objeto que contenha o nome das empresas que tiveram as maiores ofertas públicas iniciais na bolsa de valores brasileira em 2020; ou você pode querer a relação dos nomes e dos valores. Dependendo dos tipos de dados envolvidos, a forma de fazer isso pode mudar.

> O R possui o conceito de objetos, que são estruturas especializadas e que podem conter tanto os valores dos dados quanto atributos. Quando você declara uma variável simples (a = 5) ou (b = "eu"), os atributos não são importantes - é fácil ter certeza que dentro da variável haverá um valor que a representa. Contudo, em estruturas mais complexas, com muitas linhas e colunas, é importante, por exemplo, que as colunas tenham nomes (atributos), a fim de facilitar a identificação.

4.1 Vetores

Quando a sua relação trata de apenas um conjunto de valores de mesmo tipo (todos numéricos, todos textuais ou todos lógicos), pode ser formada como um **vetor**. Por exemplo, poderia haver um vetor com o nome dos CEOs das empresas do setor bancário; ou um vetor com o salário desses CEOs.
Entretanto, se estivéssemos falando de uma relação envolvendo os nomes **e** os salários, seria inadequado usarmos o vetor, porque agora teríamos duas dimensões, além de dois tipos: uma mesma pessoa (o CEO) possui duas características: o nome (textual / string) e o salário (numérica / num).

A criação de vetores é feita com o uso do c( ), em que os valores que devem ser atribuídos ao vetor são incluídos dentro dos parênteses e separados por vírgula.

```
vetorNumerico = c(1,2,3,4,5)
vetorLogico = c(TRUE, TRUE, FALSE,FALSE)
vetorTextual = c("Eu", "Nasci", "há", "10.000", "anos", "atrás")
```

Note que todos os vetores foram criados com dados de mesmo tipo. O vetor numérico tem apenas números inteiros (1 a 5); o vetor lógico tem apenas Verdadeiro (TRUE) e Falso (FALSE); o vetor de texto tem apenas elementos textuais (mesmo o número está entre aspas, o que força que aqueles dígitos sejam lidos como "texto").

Caso você tente misturar tipos, o R irá forçar a padronização. Por exemplo, o que aconteceria se você tentasse criar um vetor com números e textos?

```
vetorNumTex = c(1,2,"três","4",5)
print(vetorNumTex)
```

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

Todos os elementos apresentados estão entre aspas. Como o R não tem uma forma de transformar "três" em um número, ele converte todos os outros números em texto. Por isso, 1 vira "1", 2 vira "2" e 5 vira "5".

> Teste no seu console a seguinte operação: "3" + 1
>
> O R informará um erro. "3" não é um número, mas um texto (por causa das aspas). Não é possível somar um texto com um número. É o mesmo que você tentar somar "banana" com 97. Não dá.

4.2 Matrizes

A matriz é um conjunto de dados do mesmo tipo, ordenados em linhas (filas horizontais) e colunas (filas verticais). É criada pelo matrix(), devendo ser informados os dados que irão compor a matriz (normalmente dispostos em um vetor), o número de linhas (nrow), o número de colunas (ncol) e se o preenchimento deve ser feito por linha (byrow = TRUE) ou por coluna (byrow = FALSE).

```
# Se a matriz dos inteiros de 1 a 6, dispostos em 3 linhas e 2 colunas, preenchida pel
mat = matrix(c(1:6), nrow = 3, ncol = 2, byrow = TRUE)
print(mat)
```

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

Ou seja, os números foram distribuídos até o fim da primeira linha, depois até o fim da segunda linha e depois até o fim da terceira (e última) linha.

```
# Se a matriz dos inteiros de 1 a 6, dispostos em 3 linhas e 2 colunas, preenchida pela ordem das
mat = matrix(c(1:6), nrow = 3, ncol = 2, byrow = FALSE)
print(mat)
```

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

Em contraste, aqui o preenchimento foi por colunas. Foi preenchida toda a primeira coluna e depois toda a segunda coluna.

Se já tivéssemos um vetor salvo na memória, poderíamos usá-lo para compor a matriz.

```
vetorParaMatriz <- c("eu", "nasci", "há", "dez mil", "anos", "atrás")

mat = matrix(vetorParaMatriz, nrow = 3, ncol = 2, byrow = TRUE)
print(mat)
```

```
##      [,1]   [,2]
## [1,] "eu"   "nasci"
## [2,] "há"   "dez mil"
## [3,] "anos" "atrás"
```

Você reparou que aparecem valores entre colchetes e separados por vírgulas? Eles indicam a posição na matriz.

[m,n]	Posição	Elementos
[1,]	Primeira Linha	“eu”, “nasci”
[2,]	Segunda Linha	“há,”dez mil”
[,1]	Primeira Coluna	“eu”, “há”, “anos”
[,2]	Segunda Coluna	“nasci,”dez mil“,”atrás”
[1,1]	Primeira Linha, Primeira Coluna	“eu”
[1,2]	Primeira Linha, Segunda Coluna	“nasci”
[2,1]	Segunda Linha, Primeira Coluna	“há”
[2,2]	Segunda Linha, Segunda Coluna	“dez mil”

Eventualmente, pode ser interessante informar os nomes para as linhas e para as colunas. Por exemplo, se você estiver precisando mapear a distância entre as

unidades de uma empresa, seria importante dizer quais são as filiais.

```
vetorFiliais <- c("Natal", "Parnamirim", "Macaíba")
vetorDistancias <- c(0, 10, 12, 10, 0, 2, 12, 2, 0)
```

Por isso, vamos incluir o argumento

dimnames = list(row_names, col_names)

dimnames significa nome das dimensões, que é informado por meio de dois vetores: o vetor com o nomes das linhas (row_names) e o vetor com o nome das colunas (col_names).

No nosso exemplo, as linhas e as colunas devem possuir os mesmos nomes (porque queremos estabelecer uma matriz com a distância entre cada unidade da empresa). Por isso, trocaremos row_names por vetorFiliais e **também** trocaremos col_names por vetorFiliais.

```
distanciaFiliais = matrix(vetorDistancias, nrow = 3, ncol = 3, byrow = TRUE,
             dimnames = list(vetorFiliais, vetorFiliais))

distanciaFiliais
```

```
##            Natal Parnamirim Macaíba
## Natal          0         10      12
## Parnamirim    10          0       2
## Macaíba       12          2       0
```

Por que o elemento [1,1] da matriz foi 0? A distância de uma unidade para ela mesmo, por definição, é zero. Para solicitar a distância entre a filial Natal e a filial Parnamirim:

```
distanciaFiliais["Natal", "Parnamirim"]
```

```
## [1] 10
```

Perceba que, nessa matriz, a distância entre Natal - Parnamirim é a mesma que Parnamirim - Natal. Se pensarmos em termos de rodovias, isso pode não ser necessariamente verdade, pois em casos específicos pode ser necessário um deslocamento adicional em um dos sentidos (por exemplo, ir adiante x quilômetros para conseguir fazer um retorno). No caso dessa matriz, solicitar o valor como:

distanciaFiliais[“Natal”, “Parnamirim”]

ou

distanciaFiliais[“Parnamirim”, “Natal”]

produziria o mesmo resultado e, nesse caso, é irrelevante. Entretanto, é importante ter em mente que isso nem sempre será verdade. Além disso, se quiséssemos saber a distância da filial Natal para todas as demais, deveríamos informar apenas a linha, omitindo a informação após a vírgula.

```
distanciaFiliais["Natal",]
```

```
##      Natal Parnamirim    Macaíba
##          0         10         12
```

Alternativamente, sabendo que Natal é a primeira linha da matriz, também seria possível executar

```
distanciaFiliais[1,]
```

```
##      Natal Parnamirim    Macaíba
##          0         10         12
```

A segunda opção é particularmente importante quando as filas (linhas ou colunas) não estão nomeadas.

4.3 Data frames

O data frame é similar a matrix, mas admite colunas com tipos diferentes. Nesse caso, é possível incluir uma coluna para o nome de uma empresa (string), uma coluna para o CNPJ (string), uma coluna para o número de empregados (integer), uma coluna para o faturamento (float) e uma coluna para identificar se as demonstrações já foram encerradas ou se ainda estão pendentes (boolean).

O data frame é criado por meio do data.frame(), onde são passados os valores que irão entrar na estrutura. Uma possibilidade é passarmos vetores que representem as colunas. Assim, usaríamos um vetor para a empresa, um vetor para o CNPJ, um vetor para os empregados e um vetor para o faturamento.

Não é necessário informar a quantidade de linhas ou colunas, já que o R não precisará organizar os dados (no caso da matriz, foi necessário informar como ela deveria ser preenchida). Os vetores serão incluídos como colunas na mesma ordem em que forem aparecendo dentro do data.frame().

```
# Ainda discutiremos o que são factors, mas por enquanto insiram esse argumento.
clientes <- data.frame(empresa = c("Marte S.A.", "Deimos S.A.", "Phobos S.A."),
                       cnpj = c("00.000.001/0001-00","00.000.011/0001-10", "99.000.001/
                       empregados = c(10,20, 25),
                       encerramento = c(FALSE, FALSE, TRUE),
                       stringsAsFactors = FALSE)
clientes
```

```
##       empresa               cnpj empregados encerramento
## 1  Marte S.A. 00.000.001/0001-00         10        FALSE
## 2 Deimos S.A. 00.000.011/0001-10         20        FALSE
## 3 Phobos S.A. 99.000.001/0001-55         25         TRUE
```

Por conta dessa permissão para uso de tipos diferentes em uma mesma estrutura, os data frames são consideravalmente importantes. Para solicitar uma fila específica (linha ou coluna) ou um elemento, é possível usar a mesma notação das matrizes.

```
clientes[1,]
```

```
##      empresa               cnpj empregados encerramento
## 1 Marte S.A. 00.000.001/0001-00         10        FALSE
```

```
# para gerar uma relação com os CNPJs:
clientes[,2]
```

```
## [1] "00.000.001/0001-00" "00.000.011/0001-10" "99.000.001/0001-55"
```

Da mesma forma que fizemos com as matrizes, podemos solicitar uma coluna com base em seu nome.

```
clientes[,"cnpj"]
```

```
## [1] "00.000.001/0001-00" "00.000.011/0001-10" "99.000.001/0001-55"
```

Percebea que nesse caso as linhas não foram nomeadas. No caso dos data frames, é bem comum que o identificador seja inserido como um dado

O data frame ainda admite uma forma adicional de solicitar uma coluna, informando o nome do objeto seguido do cifrão ($) e do nome da coluna.

```
clientes$cnpj
```

```
## [1] "00.000.001/0001-00" "00.000.011/0001-10" "99.000.001/0001-55"
```

Perceba como a notação do R evolui. Quando resolvemos solicitar um clientes$cnpj, estamos usando um procedimento equivalente a clientes[,"cnpj"]. Como poderíamos fazer para usando a notação com cifrão, solicitar o terceiro elemento dessa relação?

```
clientes$cnpj[3]
```

```
## [1] "99.000.001/0001-55"
```

Não precisamos informar dois elementos dentro dos colchetes [,] porque um deles foi informado anteriormente (no caso, a coluna *cnpj*)! Assim, quando informarmos o [3] estamos automaticamente nos referindo à linha.

4.4 Tibbles

O tibble é um tipo especial de data.frame, que realiza algumas operações de forma ligeiramente diferente que o data.frame convencional. Eles fazem menos e reclamam mais, induzindo o programador a tratar melhor certos aspectos do código. Essencialmente: não permitem referências parciais ao nome de variáveis contidas em um data.frame; são mais fiéis ao que se espera que as variáveis serão (em várias ocasiões, não produzem factors, ao invés de chars) e avisam se uma coluna que está sendo requerida não faz parte do banco de dados.

Para mais detalhes, acesse a documentação: < https://tibble.tidyverse.org/ >.

Como o tibble não é uma estrutura nativa do R, deve ser acessada por meio do pacote tibble.

```
# chamando o pacote tibble
library(tibble)
primeiroTibble <- tibble(nome = c("raimundo", "marciano"), numeros = c(29,12))
primeiroDataFrame <- data.frame(nome = c("raimundo", "marciano"), numeros = c(29,12))
primeiroTibble
```

```
## # A tibble: 2 x 2
##   nome     numeros
##   <chr>      <dbl>
## 1 raimundo      29
## 2 marciano      12
```

```
primeiroDataFrame
```

```
##       nome numeros
## 1 raimundo      29
## 2 marciano      12
```

```
str(primeiroDataFrame)
```

```
## 'data.frame':    2 obs. of  2 variables:
##  $ nome   : Factor w/ 2 levels "marciano","raimundo": 2 1
##  $ numeros: num  29 12
```

Perceba que o tibble exibido já indica os tipos das variáveis que o compõem. No caso do data frame, foi necessário solicitar, por meio do comando str(), essa informação. Além disso, o primeiroTibble tem um resultado mais previsível que o primeiroDataFrame: as variáveis textuais foram consideradas corretamente como no tibble, mas foram identificadas como no data frame, o que pode ser um comportamento não esperado. Para garantir que o data frame trate a primeira coluna como string, é necessário forçar esse comportamento, incluindo o parâmetro <stringsAsFactors = F>.

```
segundoDataFrame <- data.frame(nome = c("raimundo", "marciano"),
                      numeros = c(29,12), stringsAsFactors = F)
segundoDataFrame
```

```
##       nome numeros
## 1 raimundo      29
## 2 marciano      12
```

```
str(segundoDataFrame)
```

```
## 'data.frame':    2 obs. of  2 variables:
##  $ nome   : chr  "raimundo" "marciano"
##  $ numeros: num  29 12
```

Agora temos um data.frame com os mesmos tipos de variáveis que o tibble. Em resumo, as diferenças são pequenas, mas o tibble é uma versão mais moderna, mais previsível e que gera mais alertas par ao programador que a está usando.

A solicitação de partes do tibble é idêntica à do data.frame.

```
#Para solicitar a colunha chamada de 'nome'
primeiroTibble['nome']
```

```
## # A tibble: 2 x 1
##   nome
##   <chr>
## 1 raimundo
## 2 marciano
```

```
#Para solicitar a segunda coluna
primeiroTibble[,2]
```

```
## # A tibble: 2 x 1
##   numeros
##     <dbl>
## 1      29
## 2      12
```

A principal diferença é que o tibble não aceita referência parcial. Isso significa que poderíamos solicitar ao data.frame uma coluna informando apenas parte do nome dela, ao passo que o tibble exige o nome completo.

```
# O data.frame contém uma coluna chamada 'nome'. Veja o que acontece se, por descuido
# pedirmos a coluna $nom (faltando o 'e')
primeiroDataFrame$nom
```

```
## [1] raimundo marciano
## Levels: marciano raimundo
```

O data.frame retorna os dados normalmente, como se o nome estivesse completo. Isso pode ser indesejável caso haja ambiguidade nos títulos das variáveis. Por exemplo, poderia haver uma variável chamada 'nome' e outra 'nomeCompleto'. Com a informação incompleta, você não garante estar usando a variável correta. O tibble exige que você informe o nome exato do que está sendo solicitado.

```
# Veja o que acontece se você pedir o nome incompleto de uma variável
primeiroTibble$nom
```

```
## Warning: Unknown or uninitialised column: `nom`.
```

```
## NULL
```

O R retorna um aviso que informa que não é possível identificar uma coluna chamada 'nom'. O mesmo aconteceria se você solicitasse qualquer outro nome não presente no banco de dados, como

```
# Aconteceria o mesmo se você usasse um nome 'nada a ver'
primeiroTibble$valhaMeDeus
```

```
## Warning: Unknown or uninitialised column: `valhaMeDeus`.
```

```
## NULL
```

O retorno é NULL (nulo), porque não há uma coluna chamada 'valhaMeDeus'. Não custa lembrar que ele também diferencia letras maiúsculas de minúsculas (diz-se, nesse caso, que ele é 'case sensitive').

Suponha que tenhamos (estranhamente) chamado a variável de 'nUmErOs'. Não adiantar solicitar 'numeros', porque seria uma grafia diferente da esperada.

```
outroTibble <- tibble(nome = c("raimundo", "marciano"), nUmErOs = c(29,12))
outroTibble$nUmErOs
```

```
## [1] 29 12
```

```
outroTibble$numeros
```

```
## Warning: Unknown or uninitialised column: `numeros`.
```

```
## NULL
```

Por padrão, a exibição de um tibble é limitada a 10 colunas e a uma quantidade de colunas compatível com a tela, o que não acontece com o data.frame, que tenta exibir todas as colunas presentes no banco de dados. Essa característica é dispensável em bancos com poucas variáveis, mas é bastante interessante caso ele contenha algumas centenas, porque não gera um output ilegível (raramente será interessante visualizar simultaneamente centenas de colunas ao mesmo tempo). Caso queira exibir todas as colunas, bastar adicionar o parâmetro (width = Inf) dentro do print().

4.5 Listas

4.6 Factors

Chapter 5

Trabalhando com variáveis no R Base

5.1 Manuseio de variáveis

5.2 Funções numéricas básicas

As principais operações matemáticas e de estatísticas descritivas já vêm instaladas no R.

```
# Sejam a = 5 e b = 2 e v um vetor de números inteiros
a = 5
b = 2
v = c(1,2,3,4,5)
```

As operações básicas podem ser representadas por:

```
a + b # soma
```

```
## [1] 7
```

```
a - b # subtração
```

```
## [1] 3
```

```
a * b # multiplicação (asterisco)
```

```
## [1] 10
```

```
a / b # divisão
```

```
## [1] 2.5
```

```
a ^ b # potenciação (acento circunflexo)
```

```
## [1] 25
```

Além disso, podemos estabelecer as operações no vetor, aplicando a elementos específicos:

```
v[1] + v[2] # soma
```

```
## [1] 3
```

```
v[1] - v[2] # subtração
```

```
## [1] -1
```

```
v[1] * v[2] # multiplicação (asterisco)
```

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

```
v[1] / v[2] # divisão
```

```
## [1] 0.5
```

```
v[1] ^ v[2] # potenciação (acento circunflexo)
```

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

Quando queremos aplicar ao vetor inteiro, podemos usar funções instaladas no R base.

```r
sum(v) # soma de todos os elementos de v
```

```
## [1] 15
```

```r
sum(v[1:4]) # soma dos quatro primeiros elementos de v
```

```
## [1] 10
```

```r
sum(v[3:5]) # soma dos três últimos elementos de v
```

```
## [1] 12
```

```r
mean(v) # média aritmérica de todos os elementos de v
```

```
## [1] 3
```

```r
mean(v[1:4]) # média dos quatro primeiros elementos de v
```

```
## [1] 2.5
```

5.3 Funções textuais básicas

Seja o seguinte data.frame:

```r
cias <- data.frame(nome = c("Petrobras", "Petrobras", "Vale", "Banco do Brasil"),
                   ticker = c("PETR3", "PETR4", "VALE3", "BBAS3"),
                   tipo = c("ON", "PN", "ON", "ON"),
                   segmento = c("N2", "N2", "NM", "NM"),
                   stringsAsFactors = FALSE)
cias
```

```
##              nome ticker tipo segmento
## 1       Petrobras  PETR3   ON       N2
## 2       Petrobras  PETR4   PN       N2
## 3            Vale  VALE3   ON       NM
## 4 Banco do Brasil  BBAS3   ON       NM
```

Podemos estar interessados em saber se há uma empresa chamada "Banco do Brasil". Para isso, há as funções grep() e grepl(). A grep() retorna um vetor com todas as posições em que foi encontrada a expressão desejada. A função grepl() retorna um vetor com a análise se cada elemento possui o padrão desejado ou não. Assim, se aplicássemos o grep() com a expressão "Petrobras" à lista de nomes, receberíamos a informação de que "Petrobras" foi localizada nas posições 1 e 2, sem qualquer menção às posições 3 e 4. Se usássemos o grepl(), o retorno seria uma lista composta por (TRUE, TRUE, FALSE, FALSE), que explicita a situação de cada observação.

```
grep("Petrobras", cias$nome)
```

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

```
grep("Banco do Brasil", cias$nome)
```

```
## [1] 4
```

```
grepl("Petrobras", cias$nome)
```

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

```
grepl("Banco do Brasil", cias$nome)
```

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

Chapter 6

Criando sua própria função

No R, você pode criar suas próprias funções da mesma forma como você cria e guarda dados: as funções também são objetos. A função para criar funções é a function(). Vamos começar com um caso simples, produzindo uma função que apenas imprime um texto básico.

```
minhaFuncao <- function(){
  # Esta função imprime a mensagem "Esta é a minha primeira função"
  print("Esta é a minha primeira função.")
}
```

```
minhaFuncao()
```

```
## [1] "Esta é a minha primeira função."
```

Essa função ainda não aceita parâmetros, ou seja, ela ainda não é capaz de receber e processar dados. A função pode receber dados ou usar valores que estão presentes em variáveis anteriormente criadas. O argumento funciona como um apelido que será usado internamente para aquele valor. Por exemplo, posso querer que a função retorne quanto é 8% do salário do empregado. Assim, precisamos que a função receba o valor do salário empregado e que, a partir dele, calcule quanto é 8% e, por fim, exiba esse valor.

```
# a funcao recebera o 'salario'
fgts <- function(salario){
  # variavei INTERNA fgtsFuncionario guarda o valor calculado do FGTS
  # ela não fica automaticamente disponível na memória do R após
  # a função ter terminado a execução
  fgtsFuncionario <- salario * 0.08
```

```
  # vamos gerar uma frase contendo o valor do FGTS
  mensagem <- paste0("O valor do FGTS é R$ ", fgtsFuncionario, ".")
  # para imprimir a mensagem
  print(mensagem)
  # podemos encerrar o código da função especificando uma última variável INTERNA
  # que pode ser guardada em um objeto normal do R, por atribuição
  # Se essa atribuição não ocorrer, todos os valores gerados internamente
  # serão esquecidos
  fgtsFuncionario
}
```

Perceba que se simplesmente rodarmos a função sem guardar seu valor em um objeto

```
fgts(2000)
```

```
## [1] "O valor do FGTS é R$ 160."
```

```
## [1] 160
```

A mensagem será impressa, mas posteriormente não será possível resgatar nem o valor contido em $fgtsFuncionario$ e nem em $mensagem$. Se você quer que o R realmente guarde esses valores, e não apenas os exiba na tela, é necessário que haja a atribuição a uma variável.

```
# estamos guardando o valor da execução de fgts(2000) em fgtsFunc1
fgtsFunc1 <- fgts(2000)
```

```
## [1] "O valor do FGTS é R$ 160."
```

A função $fgts$ foi encerrada com $fgtsFuncionario$. Por isso, o valor que será atribuído para fgtsFunc1 será apenas $fgtsFuncionario$. A $mensagem$ não ficará gravada na memória.

```
# se solicitarmos o valor de fgtsFunc1, será exibido o valor 160.
fgtsFunc1
```

```
## [1] 160
```

6.1 Modularização do código

Em certas situações, seu script pode começar a ficar muito extenso, dificultando a leitura do código e que você encontre problemas específicos. Assim, pode ser uma boa ideia criar scripts auxiliares com funções. Você cria um script separado (por exemplo, funcoes.R) e deixa nele o código com as funções que serão utilizadas.

No script onde você está desenvolvendo a análise, você inclui o comando source(), informando o parâmetro com a localização e o nome do arquivo ("scripts/funcoes.R", por exemplo).

```
# assumindo que o script auxiliar funcoes.R está na pasta scripts
source("scripts/funcoes.R")
```

O source() faz e leitura do script auxiliar e importa todas as funções que constam nele, de forma muito similar ao que é feito pelo library().

6.2 Comentários

Chapter 7

Pacotes / Bibliotecas

Muito do que precisamos usar já foi criado por alguém e disponibilizado na internet. No caso do R, a plataforma "oficial" para isso é o CRAN, mas há outras formas de conseguir esses códigos, como o Github.

Diversas bibliotecas já vêm pré-instaladas no R.

Quando precisamos instalar, usamos o comando

```
install.packages()
```

O nome do pacote deve ser informado entre aspas, como todo bom texto.

```
install.packages("dplyr")
```

A instalação do pacote dplyr permite o acesso a várias funções, como filter(), select() e group_by(). Quando um pacote é instalado, as funções ficam dentro dele. ISso significa que quando queremos acessá-las, precisamos identificar a fonte.

```
dplyr::select()
dplyr::filter()
```

Em outras palavras, primeiro dizemos qual é o pacote (dplyr, no caso), seguido de :: (dois pontos, duas vezes) e concluímos informando a função desejada e que se encontra desse pacote.

Alternativamente, podemos usar library(dplyr). Isso deixa, durante a sessão ativa, as funções do pacote dplyr prontamente disponíveis. Quando você fechar o R (ou o R Studio), isso será desfeito. Portanto, da próxima vez que abrir o programa, precisará executar novamente o comando library(dplyr).

É possível que existam duas funções com o mesmo nome, o que gera um conflito. Por exemplo, na minha máquina, há um pacote chamado *stats* que contém uma função chamada *lag*. Quando carrego o *dplyr*, que também possui uma função chamada lag(), recebo a seguinte mensagem:

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

Isso significa que a função lag() do *stats* foi ocultada pelo *dplyr*. Em outras palavras, se eu for usar a função lag() sem dizer a que pacote ela pertence, o R automaticamente entenderá que eu estou usando a do *dplyr*, que está sendo dominante em relação ao *stats*.

Sempre que você estiver confuso sobre qual função está dominando e qual está oculta, você pode incluir o nome do pacote no comando. Se você usar dplyr::lag(), com certeza será o lag() do dplyr. Se você usar stats::lag(), com certeza será o lag() do stats.

Se o pacote que você deseja instalar está disponível no Github, você deve usar a função install_github() do pacote remotes.

```
remotes::install_github()
```

Chapter 8

Obtendo dados

O R oferece muitas opções de pacotes para importar os dados presentes em um arquivo. Além disso, alguns pacotes permitem que você colete dados de fontes oficiais facilmente.

8.1 Planilhas (csv / Excel)

```
dados <- read.csv("data/df_info.csv", encoding = "UTF-8", colClasses = c("cnpj" = "character"))
```

Solicitamos as cinco primeiras colunas das cinco primeiras linhas:

```
dados[1:5, 1:5]
```

```
##                                       name.company id.company           cnpj
## 1                              524 PARTICIPAÇOES SA      16284  1851771000155
## 2 ADVANCED DIGITAL HEALTH MEDICINA PREVENTIVA S.A.      21725 10345009000198
## 3                              AES TIETÊ ENERGIA S.A      18970  4128563000110
## 4     AFLUENTE TRANSMISSÃO DE ENERGIA ELETRICA S/A      22179 10338320000100
## 5                                          ALEF SA      16705  2217319000107
##   date.registration date.constitution
## 1        1997-05-30        1997-04-02
## 2        2009-06-24        2008-08-18
## 3        2001-06-29        2000-11-06
## 4        2010-09-24        2008-08-18
## 5        1997-12-08        1997-10-06
```

O CNPJ está parecendo um pouco estranho. O nosso conhecimento prévio nos diz que todos os CNPJs possuem 14 dígitos. Uma simples inspeção visual revela que o número de caracteres parece estar diferente ao longo das linhas. Vamos ver se é verdade usando a função nchar().

```
nchar(dados$cnpj[1:5])
```

```
## [1] 13 14 13 14 13
```

Isso significa que o CNPJ realmente está com um número diferente de caracteres: o primeiro tem 13 (errado); o segundo, 14 (certo); o terceiro, 13 (errado); e assim por diante. Não é interessante que deixemos nossos dados assim, então precisamos investigar o que está errado. Nosso conhecimento prévio também indica uma possível falha: vários CNPJs são iniciados com o dígito zero. Como a AES Tietê Energia S.A. está listada na bolsa de valores, é fácil achar seu CNPJ fazendo uma busca no Google: 04.128.563/0001-10. Com isso, temos uma evidência do problema e é interessante analisar mais alguns casos similares (o que, aliás, se confirma).

Feito isso, precisamos saber se o problema está na importação ou no arquivo original que continha os dados. Como o arquivo é pequeno (menos de 100kb), é fácil inspecioná-lo, abrindo em uma planilha eletrônica convencional ou até mesmo no Bloco de Notas (foi a minha opção). Com isso, constatamos que os "zeros do começo" realmente estão faltando no arquivo e podemos concluir que não há erro na nossa importação.

Entretanto, não significa que o dado está como queríamos, devendo ser tratado. Primeiro, vamos avaliar se há algum com menos de 13 dígitos. Afinal, não é porque nas primeiras cinco linhas o menor valor fio 13 que isso seja representativo de todo o conjunto, podendo haver algum CNPJ iniciado por 00, por exemplo.

```
caracteres <- nchar(dados$cnpj)
table(caracteres)
```

```
## caracteres
##   3   9  10  11  12  13  14
##   1   1   1   1  10 117 199
```

Logo, há um ou mais CNPJs em que o primeiro dígito diferente de 0 é o 12º. Assim, precisaremos preencher com zeros até termos 14 dígitos.

```
zeros <- NA
for (i in 1:nrow(dados)){
  numeroZeros <- 14 - nchar(dados$cnpj[i])
```

```
  if(numeroZeros == 0){
    next
  } else {
  zeros <- rep("0", numeroZeros)
  zeros <- paste0(zeros, collapse = "")
  dados$cnpj[i] <- paste0(zeros, dados$cnpj[i], collapse = "")
  }
}

# Vamos atualizar o vetor de caracteres
caracteres <- nchar(dados$cnpj)
# E solicitar a frequência
table(caracteres)
```

```
## caracteres
##  14
## 330
```

8.2 TXT (bloco de notas) e Word

8.3 XML

8.4 Dados coletados por meio de pacotes

8.4.1 GetDFPData

8.4.2 BCBData

8.5 Juntando bases

Chapter 9

Final Words

We have finished a nice book.