Indexação e seleção condicional

Fernando P. Mayer

2022-02-22

Indexação

Existem 6 maneiras diferentes de indexar valores no R. Podemos indexar usando:

- Inteiros positivos
- Inteiros negativos
- Zero
- Espaço em branco
- Nomes
- Valores lógicos

Existem três tipos de operadores que podem ser usados para indexar (e selecionar) **sub-conjuntos** (*subsets*) de objetos no R:

- O operador `[ ]` sempre retorna um objeto da mesma classe que o original. Pode ser usado para selecionar múltiplos elementos de um objeto
- O operador `[[ ]]` é usado para extrair elementos de uma **lista** ou **data frame**. Pode ser usado para extrair um único elemento, e a classe do objeto retornado não precisa necessariamente ser uma lista ou data frame.

- O operador `$` é usado para extrair elementos **nomeados** de uma lista ou data frame. É similar ao operador `[[ ]]`.

Indexação de vetores

Observe o seguinte vetor de contagens

```
cont <- c(8, 4, NA, 9, 6, 1, 7, 9)
cont

# [1]  8  4 NA  9  6  1  7  9
```

Para acessar o valor que está na posição 4, faça:

```
cont[4]

# [1] 9
```

> Os colchetes `[ ]` são utilizados para **extração** (seleção de um intervalo de dados) ou **substituição** de elementos. O valor dentro dos colchetes é chamado de **índice**.

Para acessar os valores nas posições 1, 4 e 8 é necessário o uso da função `c()`:

```
cont[c(1, 4, 8)]

# [1] 8 9 9
```

Ou:

```
ind <- c(1, 4, 8)
cont[ind]

# [1] 8 9 9
```

> Note que os índices no R começam em 1, e não em 0, como algumas outras linguagens.

Para selecionar todos os valores, **excluindo** aquele da posição 4, usamos um índice negativo

```
cont[-4]

# [1]  8  4 NA  6  1  7  9
```

Da mesma forma se quiséssemos todos os valores, menos aqueles das posições 1, 4 e 8:

```
cont[-c(1, 4, 8)]

# [1]  4 NA  6  1  7
```

Também é possível selecionar uma sequência de elementos (com qualquer uma das funções de gerar sequências que já vimos antes):

```
## Seleciona os elementos de 1 a 5
cont[1:5]

# [1]  8  4 NA  9  6


## Seleciona os elementos nas posições ímpar
cont[seq(1, 8, by = 2)]

# [1]  8 NA  6  7
```

Mas note que para selecionar todos menos aqueles de uma sequência, precisamos colocá-la entre parênteses

```
cont[-1:5]

# Error in cont[-1:5]: only 0's may be mixed with negative subscripts


cont[-(1:5)]

# [1] 1 7 9
```

Exercícios 1

1. Crie um vetor com os valores: `88, 5, 12, 13`
2. Selecione o elemento na posição 3
3. Selecione o valor 88
4. Selecione os valores 13 e 5 (nessa ordem)
5. Selecione todos os valores, menos o 88 e o 13
6. Insira o valor 168 **entre** os valores 12 e 13, criando um novo objeto

Para selecionar todos os elementos que sejam `NA` , ou todos menos os `NA` s, precisamos usar a função `is.na()`

```
## Para selecionar os NAs
cont[is.na(cont)]
```

```
# [1] NA


## Para selecionar todos menos os NAs
cont[!is.na(cont)]

# [1] 8 4 9 6 1 7 9
```

Para substituir os `NA` s por algum valor (*e.g.* 0):

```
cont[is.na(cont)] <- 0
cont

# [1] 8 4 0 9 6 1 7 9
```

E para especificar `NA` para algum valor

```
is.na(cont) <- 3
cont

# [1]  8  4 NA  9  6  1  7  9


## Mais seguro do que
# cont[3] <- NA
```

> Note que se utilizarmos o operador de atribuição `<-` em conjunto com uma indexação, estaremos **substituindo** os valores selecionados pelos valores do lado direito do operador de atribuição.

Podemos também utilizar mais duas formas de indexação no R: espaços em branco e zero:

```
cont[0]

# numeric(0)


cont[]

# [1]  8  4 NA  9  6  1  7  9
```

Note que o índice zero não existe no R, mas ao utilizá-lo é retornado um vetor "vazio", ou um vetor de comprimento zero. Essa forma de indexação é raramente utilizada no R.

Ao deixar um espaço em branco, estamos simplesmente informando que queremos todos os valores daquele vetor. Essa forma de indexação será particularmente útil para objetos que possuem duas ou mais dimensões, como matrizes e data frames.

Vetores nomeados

Quando vetores possuem seus componentes **nomeados**, a indexação pode ser realizada pelos nomes destes componentes.

```
## Atribui as letras "a", "b", ..., "h" para os componentes de cont
names(cont) <- letters[1:length(cont)]
## Acessando o quarto elemento
cont["d"]
```

```
# d
# 9
```

```
## Acessando o sexto e o primeiro elemento
cont[c("f", "a")]
```

```
# f a
# 1 8
```

Dica: veja `?letters`

Indexação de matrizes

Considere a seguinte matriz

```
mat <- matrix(1:9, nrow = 3)
mat
```

```
#      [,1] [,2] [,3]
# [1,]    1    4    7
# [2,]    2    5    8
# [3,]    3    6    9
```

Acesse o valor que está na linha 2 da coluna 3:

```
mat[2, 3]
```

```
# [1] 8
```

A indexação de matrizes sempre segue a ordem `[linha, coluna]`

Para acessar todas as linhas da coluna 1, deixamos o espaço em branco (que também é uma forma de seleção, como vimos) na posição das linhas e especificamos a coluna desejada

```
mat[, 1]

# [1] 1 2 3
```

Para acessar todas as colunas da linha 1, usamos a mesma lógica

```
mat[1, ]

# [1] 1 4 7
```

Note que o resultado destas extrações trazem os valores internos das matrizes, mas com a dimensão reduzida (nestes casos para uma dimensão). Se o objetivo for, por exemplo, extrair uma parte da matriz, mas mantendo as duas dimensões, então precisamos usar o argumento `drop` da "função" `[` (sim, também é uma função; veja `"?["`). Veja as diferenças:

```
mat[3, 2]

# [1] 6


mat[3, 2, drop = FALSE]

#      [,1]
# [1,]    6


mat[1, ]

# [1] 1 4 7


mat[1, , drop = FALSE]

#      [,1] [,2] [,3]
# [1,]    1    4    7
```

Para acessar as linhas 1 e 3 das colunas 2 e 3:

```
mat[c(1, 3), c(2, 3)]
```

```
#      [,1] [,2]
# [1,]    4    7
# [2,]    6    9
```

E note que aqui a dimensão já é 2 pois naturalmente o resultado preisa ser representado em duas dimensões.

Matrizes nomeadas

Se as matrizes tiverem linhas e/ou colunas nomeados, a indexação também pode ser feita com os nomes.

```
##-------------------------------------------------------------------
---
## Nomes das colunas
colnames(mat) <- LETTERS[1:3]
## Todas as linhas da coluna B
mat[, "B"]
```

```
# [1] 4 5 6
```

```
## Elemento da linha 1 e coluna C
mat[1, "C"]
```

```
# C
# 7
```

```
##-------------------------------------------------------------------
---
## Nomes das linhas
rownames(mat) <- LETTERS[24:26]
## Todas as colunas da linha X
mat["X", ]
```

```
# A B C
# 1 4 7
```

```
## Elemento da linha Y e coluna A
mat["Y", "A"]
```

```
# [1] 2
```

Indexação de listas

Considere a seguinte lista:

```
lis <- list(c(3, 8, 7, 4), mat, 5:0)
lis

# [[1]]
# [1] 3 8 7 4
#
# [[2]]
#   A B C
# X 1 4 7
# Y 2 5 8
# Z 3 6 9
#
# [[3]]
# [1] 5 4 3 2 1 0
```

Podemos acessar um componente da lista da maneira usual

```
lis[1]

# [[1]]
# [1] 3 8 7 4
```

Mas note que esse objeto continua sendo uma lista

```
class(lis[1])

# [1] "list"
```

Geralmente o que queremos é acessar os elementos que estão **contidos** nos componentes da lista, e para isso precisamos usar `[[` no lugar de `[`:

```
lis[[1]]

# [1] 3 8 7 4


class(lis[[1]])

# [1] "numeric"
```

Isso é importante, por exemplo, se quisessemos aplicar uma função qualquer a um componente da lista. No primeiro caso isso é possível, embora o resultado continue dentro de uma lista, enquanto que no segundo caso os valores retornados são os próprios números:

```
mean(lis[1])

# Warning in mean.default(lis[1]): argument is not numeric or logica
l: returning
# NA

# [1] NA


mean(lis[[1]])

# [1] 5.5
```

Para acessar o segundo **componente** da lista:

```
lis[[2]]

#   A B C
# X 1 4 7
# Y 2 5 8
# Z 3 6 9
```

Para acessar o terceiro valor do primeiro componente:

```
lis[[1]][3]

# [1] 7
```

Note que o segundo elemento da lista é uma matriz, portanto a indexação da matriz *dentro da lista* também segue o mesmo padrão

```
lis[[2]][2, 3]

# [1] 8
```

Se a lista tiver componentes **nomeados**, eles podem ser acessados com o operador `$` :

```
lis <- list(vetor1 = c(3, 8, 7, 4), mat = mat, vetor2 = 5:0)
## Ou
## names(lis) <- c("vetor1", "mat", "vetor2")
lis
```

```
# $vetor1
# [1] 3 8 7 4
#
# $mat
#   A B C
# X 1 4 7
# Y 2 5 8
# Z 3 6 9
#
# $vetor2
# [1] 5 4 3 2 1 0
```

```
## Acesando o segundo componente
lis$mat
```

```
#   A B C
# X 1 4 7
# Y 2 5 8
# Z 3 6 9
```

```
## Linha 2 e coluna 3
lis$mat[2, 3]
```

```
# [1] 8
```

```
## Terceiro elemento do primeiro componente
lis$vetor1[3]
```

```
# [1] 7
```

Ou então

```
lis[["mat"]]
```

```
#   A B C
# X 1 4 7
# Y 2 5 8
# Z 3 6 9
```

```
lis[["vetor1"]][3]
```

```
# [1] 7
```

O símbolo `$` é utilizado para acessar componentes **nomeados** de listas ou data frames.

Indexação de data frames

Considere o seguinte data frame

```
da <- data.frame(A = 4:1, B = c(2, NA, 5, 8))
da
```

```
#   A  B
# 1 4  2
# 2 3 NA
# 3 2  5
# 4 1  8
```

Para acessar o segundo elemento da primeira coluna (segue a mesma lógica da indexação de matrizes pois também possui duas dimensões):

```
da[2, 1]
```

```
# [1] 3
```

Acesse todas as linhas da coluna B:

```
da[, 2]
```

```
# [1]  2 NA  5  8
```

Ou usando o nome da coluna:

```
da[,"B"]
```

```
# [1]  2 NA  5  8
```

Todas as colunas da linha 1:

```
da[1, ]
```

```
#   A B
# 1 4 2
```

Ou usando o “nome” da linha:

```
da["1", ]
```

```
#   A B
# 1 4 2
```

Note que o argumento `drop=` também é válido se quiser manter a dimensão do objeto

```
da[, "B"]
```

```
# [1]  2 NA  5  8
```

```
da[, "B", drop = FALSE]
```

```
#    B
# 1  2
# 2 NA
# 3  5
# 4  8
```

Como o data frame é um caso particular de uma lista (onde todos os componentes tem o mesmo comprimento), as colunas de um data frame também podem ser acessadas com `$` :

```
da$A
```

```
# [1] 4 3 2 1
```

Para acessar o terceiro elemento da coluna B:

```
da$B[3]
```

```
# [1] 5
```

Para acessar os elementos nas posições 2 e 4 da coluna B:

```
da$B[c(2, 4)]
```

```
# [1] NA  8
```

Assim como nas listas, podemos indexar um data frame usando apenas um índice, que nesse caso retorna a coluna (componente) do data frame:

```
da[1]
```

```
#   A
# 1 4
# 2 3
# 3 2
# 4 1


class(da[1])

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

Note que dessa forma a classe é mantida. Também podemos indexar os data frames usando `[[` da mesma forma como em listas

```
da[[1]]

# [1] 4 3 2 1


da[["A"]]

# [1] 4 3 2 1


da[["A"]][2:3]

# [1] 3 2
```

Para lidar com `NA` s em data frames, podemos também usar a função `is.na()`

```
da[is.na(da), ] # nao retorna o resultado esperado

#     A  B
# NA NA NA


## Deve ser feito por coluna
da[is.na(da$A), ]

# [1] A B
# <0 rows> (or 0-length row.names)


da[is.na(da$B), ]

#   A  B
# 2 3 NA
```

```
## De maneira geral
is.na(da)

#          A     B
# [1,] FALSE FALSE
# [2,] FALSE  TRUE
# [3,] FALSE FALSE
# [4,] FALSE FALSE


is.na(da$A)

# [1] FALSE FALSE FALSE FALSE


is.na(da$B)

# [1] FALSE  TRUE FALSE FALSE
```

Para remover as linhas que possuem `NA` , note que será necessário remover todas as colunas daquela linha, pois data frames não podem ter colunas de comprimentos diferentes

```
da[!is.na(da$B), ]

#   A B
# 1 4 2
# 3 2 5
# 4 1 8
```

A função `complete.cases()` facilita esse processo. Veja o resultado

```
complete.cases(da)

# [1]  TRUE FALSE  TRUE  TRUE


da[complete.cases(da), ]

#   A B
# 1 4 2
# 3 2 5
# 4 1 8
```

A função `with()`

Para evitar fazer muitas indexações de um mesmo data frame, por exemplo, podemos utilizar a função `with()`

```
with(da, A)

# [1] 4 3 2 1
```

é o mesmo que

```
da$A

# [1] 4 3 2 1
```

Também é útil para acessar elementos específicos dentro de data frames. Por exemplo, o terceiro elemento da coluna B

```
with(da, B[3])

# [1] 5
```

E também para aplicar funções nas colunas do data frame

```
with(da, mean(A))

# [1] 2.5
```

Exercícios 2

1. Crie a seguinte matriz

$$\begin{bmatrix} 4 & 16 & 2 \\ 10 & 5 & 11 \\ 9 & 9 & 5 \\ 2 & 0 & NA \end{bmatrix}$$

2. Acesse o elemento na quarta linha e na segunda coluna
3. Acesse todos os elementos, com exceção da segunda coluna e da terceira linha
4. Crie uma lista (nomeada) com 3 componentes: um vetor numérico de comprimento 10, um vetor de caracteres de comprimento 7, e a matriz do exercício anterior
5. Acesse os elementos nas posições de 5 a 3 do primeiro componente da lista
6. Acesse os caracteres de todas as posições, menos na 2 e 6
7. Acesse todas as linhas da coluna 3 da matriz dentro desta lista
8. "Crie" um novo componente nessa lista (usando `$`), contendo 30 valores aleatórios de uma distribuição normal $\mathrm{N}(12, 4)$ (veja `?rnorm`)

9. Crie um data frame que contenha duas colunas: a primeira com as letras de “A” até “J”, e outra com o resultado de uma chamada da função `runif(7, 1, 5)`
10. Extraia as duas primeiras linhas desse data frame
11. Extraia as duas últimas linhas desse data frame
12. Qual é o valor que está na linha 6 e coluna 1?
13. Qual é o valor que está na linha 4 da coluna 2?
14. Como você faria para contar quantos valores perdidos (`NA` s) existem nesse data frame?
15. Elimine os `NA` s deste data frame.
16. Crie uma nova coluna neste data frame, com valores numéricos (quaisquer)
17. Crie mais um componente na lista anterior, que será também uma lista com dois componentes: `A` com os valores `1:5` , e `B` com as letras de “a” a “f”
18. Acesse o número 4 de `A`
19. Acesse a letra “c” de `B`

Seleção condicional

Seleção condicional em vetores

> A **seleção condicional** serve para extrair dados que satisfaçam algum critério, usando **expressões condicionais** e **operadores lógicos**.

Considere o seguinte vetor

```
dados <- c(5, 15, 42, 28, 79, 4, 7, 14)
```

Selecione apenas os valores maiores do que 15:

```
dados[dados > 15]

# [1] 42 28 79
```

Selecione os valores maiores que 15 E menores ou iguais a 35:

```
dados[dados > 15 & dados <= 35]

# [1] 28
```

Para entender como funciona a seleção condicional, observe apenas o resultado da condição dentro do colchetes:

```
## Usando & (e)
dados > 15 & dados <= 35
```

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

```
## Usando | (ou)
dados > 15 | dados <= 35
```

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

Os valores selecionados serão aqueles em que a condição for `TRUE`, no primeiro caso apenas o quarto elemento do vetor `dados`.

A seleção condicional também é muito útil para selecionar elementos de um vetor, baseado em uma condição de outro vetor.

Considere o seguinte vetor de caracteres

```
cara <- letters[1:length(dados)]
```

Considere que de alguma forma, os objetos `dados` e `cara` possuem alguma relação. Sendo assim, podemos selecionar elementos de `dados`, baseados em alguma condição de `cara`

```
## Elemento de dados onde cara é igual a "c"
dados[cara == "c"]
```

```
# [1] 42
```

Se quisermos selecionar mais de um elemento de `dados`, baseado em uma condição de `cara`?

```
## Elementos de dados onde cara é igual a "a" e "c"
cara == "a" & cara == "c" # porque não funciona?
```

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

```
cara == "a" | cara == "c"
```

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

```
dados[cara == "a" | cara == "c"]
```

```
# [1]  5 42
```

Uma solução melhor seria se pudessemos usar

```
dados[cara == c("a", "c")]
```

```
# [1] 5
```

mas nesse caso só temos o primeiro elemento. Um operador muito útil nestes casos é o `%in%`

```
dados[cara %in% c("a", "c")]
```

```
# [1]  5 42
```

```
cara %in% c("a", "c")
```

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

Veja a diferença

```
cara == c("a", "c")
```

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

```
cara %in% c("a", "c")
```

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

Também é possível fazer a seleção de `cara` , baseado em uma condição em `dados`

```
## Elemento de cara onde dados é igual a 15
cara[dados == 15]
```

```
# [1] "b"
```

```
## Elemento de cara onde dados for maior do que 30
cara[dados > 30]
```

```
# [1] "c" "e"
```

```
## Elemento de cara onde dados for igual a 4 ou 14
cara[dados %in% c(4, 14)]
```

```
# [1] "f" "h"
```

A função `which()`

Até agora vimos seleções condicionais que nos retornavam o resultado de uma expressão condicional em vetores. No entanto, muitas vezes estamos interessados em saber a **posição** do resultado de uma expressão condicional, ao invés do resultado em si.

A fução `which()` retorna as posições dos elementos que retornarem `TRUE` em uma expressão condicional.

```
## Elementos maiores de 15
dados[dados > 15]

# [1] 42 28 79


which(dados > 15)

# [1] 3 4 5


## Elementos maiores de 15 e menores ou iguais a 35
dados[dados > 15 & dados <= 35]

# [1] 28


which(dados > 15 & dados <= 35)

# [1] 4


## Elementos de dados onde cara igual a "c"
dados[cara == "c"]

# [1] 42


which(cara == "c")

# [1] 3


## Elementos de dadaos onde cara igual a "a" ou "c"
dados[cara %in% c("a", "c")]

# [1]  5 42


which(cara %in% c("a", "c"))

# [1] 1 3
```

Exercícios 3

1. Crie um vetor (`x`) com os valores 3, 8, 10, 4, 9, 7, 1, 9, 2, 4.
2. Selecione os elementos maiores ou iguais a 5
3. Selecione todos os elementos menos o 4
4. Selecione os elementos maiores que 4 e menores que 8
5. Crie um vetor (`a`) com as letras de A até J
6. Selecione os elementos de x onde a for igual a “F”
7. Selecione os elementos de x onde a for igual a “B”, “D”, e “H”
8. Qual a posição do número 10 em x?
9. Quais as posições dos valores maiores ou iguais a 8 e menores ou iguais a 10 em x?
10. Quais as posições das letras “A”, “B”, “D” em a?

Seleção condicional em data frames

Considere o seguinte data frame

```
dados <- data.frame(ano = c(2001, 2002, 2003, 2004, 2005),
                    processos = c(26, 18, 25, 32, NA),
                    estado = c("SP", "RS", "SC", "SC", "RN"))
dados
```

```
#    ano processos estado
# 1 2001        26     SP
# 2 2002        18     RS
# 3 2003        25     SC
# 4 2004        32     SC
# 5 2005        NA     RN
```

Extraia deste objeto apenas a linha correspondente ao ano 2004:

```
dados[dados$ano == 2004, ]
```

```
#    ano processos estado
# 4 2004        32     SC
```

Mostre as linhas apenas do estado “SC”:

```
dados[dados$estado == "SC", ]
```

```
#    ano processos estado
# 3 2003        25     SC
# 4 2004        32     SC
```

Observe as linhas onde a processos seja maior que 20, selecionando apenas a coluna `processos`:

```
dados[dados$processos > 20, "processos"]

# [1] 26 25 32 NA
```

Também exclua as linhas com `NA`s (agora com todas as colunas):

```
dados[dados$processos > 20 & !is.na(dados$processos), ]

#    ano processos estado
# 1 2001        26     SP
# 3 2003        25     SC
# 4 2004        32     SC


dados[dados$processos > 20 & complete.cases(dados), ]

#    ano processos estado
# 1 2001        26     SP
# 3 2003        25     SC
# 4 2004        32     SC
```

A condição pode ser feita com diferentes colunas:

```
dados[dados$processos > 25 & dados$estado == "SC", ]

#    ano processos estado
# 4 2004        32     SC
```

A função `subset()` serve para os mesmos propósitos, e facilita todo o processo de seleção condicional:

```
dados[dados$estado == "SC", ]

#    ano processos estado
# 3 2003        25     SC
# 4 2004        32     SC


subset(dados, estado == "SC")
```

```
#    ano processos estado
# 3 2003        25     SC
# 4 2004        32     SC
```

```
dados[dados$processos > 20, ]
```

```
#     ano processos estado
# 1  2001        26     SP
# 3  2003        25     SC
# 4  2004        32     SC
# NA   NA        NA   <NA>
```

```
subset(dados, processos > 20)
```

```
#    ano processos estado
# 1 2001        26     SP
# 3 2003        25     SC
# 4 2004        32     SC
```

```
dados[dados$processos > 20 & !is.na(dados$processos), ]
```

```
#    ano processos estado
# 1 2001        26     SP
# 3 2003        25     SC
# 4 2004        32     SC
```

```
dados[dados$processos > 20, "processos"]
```

```
# [1] 26 25 32 NA
```

```
subset(dados, processos > 20, select = processos)
```

```
#   processos
# 1        26
# 3        25
# 4        32
```

```
subset(dados, processos > 20, select = processos,  drop = TRUE)
```

```
# [1] 26 25 32
```

A grande vantagem é que a função `subset()` já lida com os `NA` s (se isso for o que você precisa).

Exercícios 4

1. Você observou 42 processos em SC, 34 no PR, 59 no RN, e 18 no AM. Crie um data frame para armazenar estas informações (número de processos observados e estado).
2. Com o data frame criado no exercício anterior, mostre qual o estado onde foram observados menos de 30 processos (usando seleção condicional!).
3. Crie uma nova coluna (região) neste data frame indicando a qual região do país pertence cada estado.
4. Você está interessado em saber em qual dos estados da região sul o número de processos observados foi maior do que 40. Usando a seleção condicional, mostre essa informação na tela.
5. Qual região do país possui mais de 50 processos observados?