Operações matemáticas, vetores e matrizes

Leonardo Sangali Barone

March 20, 2017

Começando do zero

No tutorial anterior, começamos "do meio", ou seja, fizemos a primeira incursão na linguagem R aprendendo um pouco sobre *data frames*. Vamos, no presente tutorial, "começar do zero", ou seja, aprender sobre os aspectos elementares da linguagem R para, na etapa seguinte, aprender um pouco mais sobre *data frames* e, então, avançar para aspectos mais complexos da linguagem.

Operações matemáticas e vetores atômicos

R serve como calculadora é bastante simples realizar operações matemáticas.

Soma:

```
42 + 84
```

```
## [1] 126
```

Subtração:

```
84 - 42
```

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

Multiplicação

```
42 * 2
```

```
## [1] 84
```

Divisão:

```
42 / 6
```

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

Potência:

```
2 ^ 5
```

```
## [1] 32
```

Divisão inteira (sem resto):

```
42 %/% 5
```

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

Resto da divisão:

```
42 %% 5
```

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

Nos exemplos acima realizamos operações bastante simples sem usar objetos, ou, como costumamos chamar, variáveis. Note que há dois usos para a palavra "variável". O primeiro deles, que vimos no tutorial anterior, é

o sinônimo de coluna em um *data frame*, e registra uma característica específica para todas as observações (por exemplo, a coluna com idade em um *data frame* com indivíduos).

O outro uso é sinônimo de objeto no R. Nos referimos à "variável x", quando atribuímos algo – um número ou um texto, por exemplo – a "x".

Nos tutoriais, vamos fazer uso de variável com ambos significados. Fique atent@, mas com o tempo você se acostumará.

Voltando às operações matemáticas, vamos criar uma variável "x" com o valor 42. Ao criar uma variável que armazena apenas um número, estamos criando um vetor atômico (pois, vetores atômicos são os vetores de tamanho 1).

```
x <- 42
```

Lembre-se: podemos usar o "<-" ou "=" para fazermos uma atribuição:

```
x = 42
```

No entanto, vale a pena usar "<-" para não confundir os usos de "=", que também é usado para estabelecer valores nos argumentos de uma função.

Podemos imprimir o valor de uma variável no console simplesmente digitando seu nome:

```
x
```

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

Em várias outras linguagens, e em R inclusive, usa-se a função *print* para imprimir os valores de uma variável:

```
print(x)
```

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

Quando usar print? Veremos no futuro que, dependendo da situação (por exemplo, dentro de funções), é preciso explicitar que queremos "imprimir" algo, e, nestes casos, usamos a função *print*.

Vamos criar mais uma variável, y, e fazer operações com variáveis:

```
y <- 5

x + y
```

```
## [1] 47
```

```
x - y
```

```
## [1] 37
```

```
x / y
```

```
## [1] 8.4
```

```
x * y
```

```
## [1] 210
```

Podemos armazenar o resultado de uma operação matemática em uma variável. Veja os exemplos:

```
z1 <- 42 / 3
z2 <- x + y
z3 <- ((x / 5 ) * 9) + 32
```

Veja que na última operação utilizamos diversos parênteses. (Ei! A fórmula acima é conhecida, não?) As regras para o uso de parênteses no R em operações matemáticas são semelhantes às da aritmética "de papel e

caneta". Os parênteses são executados sempre de dentro para fora. Aliás, essa regra vale em geral no R, ou seja, para aplicação de quaisquer funções, e não apenas para as operações matemáticas.

Classes dos vetores atômicos

Há três **classes** fundamentais para os vetores atômicos. Vamos criar três variáveis e examinar suas classes:

```
numero_pi <- 3.14
texto <- "Meu texto"
verdadeiro <- TRUE
```

Usamos a função *class* para examinar a classe de um objeto:

```
class(numero_pi)
```

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

```
class(texto)
```

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

```
class(verdadeiro)
```

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

Auto-explicativo, certo?

Mais um pouco sobre vetores

Aproveitando o embalo dos vetores atômicos, vamos ver um pouco mais sobre vetores de tamanho maior que 1. Alguns exemplos e suas classes:

```
vetor_numerico <- c(42, 7, 999, 3.14)
vetor_texto <- c("texto", "a", 'jota', "Miriam", "4")
vetor_logico <- c(TRUE, FALSE, F, F, T)
class(vetor_numerico)
```

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

```
class(vetor_texto)
```

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

```
class(vetor_logico)
```

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

Detalhes para observarmos:

- No caso do vetor numérico, não importa se usamos números com casas decimais.
- Para vetores do tipo "character", não importa o que há dentro dos parêntese. Tudo é texto.
- Você pode usar TRUE ou T, FALSE ou F, alternadamente. O R entende o que você quer dizer. Lembre-se de sempre usar maiúsculas.

Exercícios:

Qual é a classe dos vetores abaixo? Imprima o vetor com *print* e tente advinhar. Use a função *class* para descobrir a resposta.

```r
v1 <- c(1, 2, TRUE, 4)
v2 <- c("T", "TRUE", "FALSE", "T")
v3 <- c("1", "2", "3", "4")
v4 <- c(1, "4", 4, 1)
v5 <- c(1, 2, "feijao com arroz")
v6 <- c("Beatriz", "Pedro", TRUE)
v7 <- c(T, T, F, T, F, F, 42)
```

Você consegue identificar as regras de combinação de tipos de dados diferentes em um mesmo vetor? Se tiver dúvidas, pergunte.

Sequências

Dá um trabalho danada criar uma sequência de números de, por exemplo, 42 a 66. Ou sequências ainda maiores. Uma maneira simples (mas não a única), de gerar uma sequência de inteiros em R é utilizar ":". Veja o exemplo:

```r
sequencia <- 42:66
print(sequencia)
```

```
##  [1] 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
## [24] 65 66
```

A sequência pode ter ordem reversa:

```r
sequencia_reversa <- 10:1
print(sequencia_reversa)
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1
```

Podemos combinar sequências que contém um intervalo, ou mesmo sequências que se sobrepõe, em um único vetor:

```r
sequencia_intervalo <- c(1:10, 20:30)
print(sequencia_intervalo)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 20 21 22 23 24 25 26 27 28 29 30
```

```r
sequencia_sobreposicao <- c(10:20, 15:25)
print(sequencia_sobreposicao)
```

```
##  [1] 10 11 12 13 14 15 16 17 18 19 20 15 16 17 18 19 20 21 22 23 24 25
```

Operações matemáticas com vetores

Ei, você advinhou a fórmula de conversão de temperatura de celsius para farenheit um pouco acima no tutorial? Bem, vamos usá-la num exemplo.

Comecemos com um vetor de temperaturas médias dos meses de dezembro a abril em um lugar qualquer do hemisfério Norte:

```r
temperatura_celsius <- c(-7, -10, 5, 12, 21)
```

Da mesma maneira que com vetores atômicos, podemos aplicar as operações matemáticas a vetores maiores. Por exemplo, vamos converter os valores do vetor "temperatura_celsius" para farenheit:

```r
temperatura_farenheit <- ((temperatura_celsius / 5) * 9) + 32
```

Veja que as operações são aplicadas a todos os elementos do vetor.

Nomes do vetor

Aproveitando o exemplo, os elementos de um vetor podem ser nomeados. O vetor "temperatura_celsius", por enquanto, não tem nome:

```
names(temperatura_celsius)
```

```
## NULL
```

(Ei, já usamos a função *names* no passado para ver o nome das colunas de um *data frame*).

Se quiseremos atribuir os nomes aos elementos de "temperatura_celsius", atribuimos um vetor a "names(temperatura_celsius)". Esse uso da linguagem é um pouco estranho, pois estamos atribuindo algo a uma função de um objeto, não a um objeto.

Uma maneira de pensar esse uso menos confusa, é imaginar que um objeto tem várias partes e que estamos atribuindo algo a uma parte específica – os nomes dos elementos, no caso – deste objeto. Veja:

```
names(temperatura_celsius) <- c("dezembro", "janeiro", "fevereiro", "março", "abril")
```

Outra forma, mais elegante, de nomear os elementos de um vetor é criar um novo vetor com os nomes, tal como abaixo...

```
meses_experimento <- c("dezembro", "janeiro", "fevereiro", "março", "abril")
```

... e a seguir atribuir aos nomes dos elementos do vetor existente (no nosso caso, temperatura_celsius):

```
names(temperatura_celsius) <- meses_experimento
```

A vantagem deste processo é poder usar mais de uma vez o vetor de nomes sem precisar escrevê-lo novamente, como faremos no exemplo abaixo.

Operações entre vetores

Criemos dois vetores, cada um registrando os gastos com sorvete e café de um pessoa em cada dia da semana, sábado e domingo, inclusive:

```
semana_1 <- c(32, 20, 15, 20, 18, 19, 40)
semana_2 <- c(32, 21, 12, 12, 24, 21, 50)
```

Antes de seguir com as operações entre os vetores, vamos nomear seus elementos. Vale a penas criar um único vetor, "dias_da_semana", e atribuir aos nomes dos elementos dos vetores de dados de gasto com sorvete e café:

```
dias_da_semana <- c("Domingo", "Segunda", "Terca", "Quarta", "Quinta", "Sexta", "Sabado")
names(semana_1) <- dias_da_semana
names(semana_2) <- dias_da_semana
```

Operações entre vetores seguem a mesma lógica das operações com vetores atômicos e com vetores e números, com a ressalva de que as operações são realizadas "pareando" os elementos dos vetores. Dito de outra forma, ao somarmos dois vetores, por exemplo, o vetor resultante terá na primeira posição a soma dos elementos da primeira posição dos vetores originais, a segunda posição terá a soma dos elementos da segunda posição dos vetores originais e assim por diante. Executando o exemplo:

```
soma_semanas <- semana_1 + semana_2
print(soma_semanas)
```

```
## Domingo Segunda   Terca  Quarta  Quinta   Sexta  Sabado
##      64      41      27      32      42      40      90
```

Note que, como os vetores originais já estavam nomeados e os elementos em cada posição tinham o mesmo nome, o vetor resultante também já está nomeado.

É bastante simples criar um vetor que seja a combinação de dois vetores. Por exemplo, se quisermos juntar as duas semanas em um único vetor, usamos a função *c*:

```
duas_semanas <- c(semana_1, semana_2)
```

Subconjunto de um vetor - parte 1

E se quisermos extrair elementos em apenas uma ou algumas posições de um vetor?

(Ei, esta é uma parte bem importante da linguagem R).

Quando queremos selecionar elementos de um vetor (ou, no futuro, de uma matriz ou de um *data frame*) usamos colchetes [] ao final do objeto. Vetores são objetos com uma única dimensão, então tudo que precisamos fazer é colocar o número da posição que queremos dentro dos colchetes. Chamamos esse procedimento em inglês de "subset" (no português, "selecionar um subconjunto").

Para extrair o primeiro dia do vetor com dados da semana 1 (domingo):

```
semana_1[1]
```

```
## Domingo
##      32
```

Ou, para extrair o final de semana (domingo na posição 1 e sábado na posição 7):

```
semana_1[c(1,7)]
```

```
## Domingo  Sabado
##      32      40
```

Ou ainda, os dias úteis da semana:

```
semana_1[2:6]
```

```
## Segunda   Terca  Quarta  Quinta   Sexta
##      20      15      20      18      19
```

Se os elementos do vetor estiverem adequadamente nomeados, podemos usar seus nomes no lugar de suas posições. Repetindo os dois primeiros exemplos imediatamente acima temos:

```
semana_1["Domingo"]
```

```
## Domingo
##      32
```

```
semana_1[c("Domingo", "Sabado")]
```

```
## Domingo  Sabado
##      32      40
```

Podemos usar um vetor para nos auxíliar a produzir o subconjunto:

```
dias_uteis <- c("Segunda", "Terca", "Quarta", "Quinta", "Sexta")
semana_1[dias_uteis]
```

```
## Segunda   Terca  Quarta  Quinta   Sexta
##      20      15      20      18      19
```

Exercício:

- Crie dois novos vetores. No primeiro, anote (invente) o número de palavras que você escreveu para sua tese/dissertação em cada mês, considerando os últimos seis meses (setembro a fevereiro). No segundo, anote (chute, novamente) quantos litros de café você tomou em cada mês.
- Nomeie os elementos dos 2 vetores.
- Calcule sua produtividade em "palavras por Litro de café". Atribua o resultado a um novo vetor
- Gere um subcojunto do novo vetor com a produtividade no final 2016 e outro com a produtividade no começo de 2017.

Soma, média e estatísticas descritivas dos elementos de um vetor

Ao longo do tempo, nosso repertório de funções de R aumentará rapidamente. Há um conjunto de funções fáceis de lembrar que são muito úteis para calcular estatísticas descritivas de um vetor (ou de uma variável em um *data frame*). Exemplo: meu consumo de litros de café por mês em 2016.

```
litros_cafe <- c(4.3, 3.1, 5.3, 5.5, 6.9, 8.3, 9.7, 9.9, 9.1, 7.0, 6.2, 5.6)
```

Observe as funções de soma, media, desvio padrão, variância, mediana, máximo, mínimo e quantil, na respectiva ordem:

```
sum(litros_cafe)
```

```
## [1] 80.9
```

```
mean(litros_cafe)
```

```
## [1] 6.741667
```

```
sd(litros_cafe)
```

```
## [1] 2.158475
```

```
var(litros_cafe)
```

```
## [1] 4.659015
```

```
median(litros_cafe)
```

```
## [1] 6.55
```

```
max(litros_cafe)
```

```
## [1] 9.9
```

```
min(litros_cafe)
```

```
## [1] 3.1
```

```
quantile(litros_cafe, probs = c(0, 0.25, 0.5, 0.75, 1))
```

```
##   0%  25%  50%  75% 100%
## 3.10 5.45 6.55 8.50 9.90
```

Veja que, com a exceção de *quantile*, todas as funções retornam vetores atômicos. *quantile* retorna um vetor do tamanho do vetor de probabilidades, que é o segundo argumento da função, e que indica os quantis correspondentes a cada valor.

Exercício

Com os seus próprios exemplos do exercício acima (palavras e litros de café por mês), use 6 funções acima.

Subconjunto de um vetor - parte 2

Finalmente, vamos usar operadores relacionais (ao qual voltaremos no início do próximo tutorial) para produzir um exemplo de subconjunto mais interessante. A "Organização Mundial de Bebedores de Café", OMBC, recomenda que o consumo de café não ultrapasse o limite de até 7 litros por mês (inclusive). Vamos observar em quais meses de 2016 eu tomei mais café do que deveria.

Nomeando o vetor:

```
meses <- c("Janeiro", "Fevereiro", "Marco", "Abril", "Maio", "Junho",
           "Julho", "Agosto", "Setembro", "Outubro", "Novembro", "Dezembro")
names(litros_cafe) <- meses
```

Criando um vetor lógico (TRUE ou FALSE) que indique em quais meses meu consumo ultrapassou o limite recomendado:

```
selecao <- litros_cafe > 7
print(selecao)
```

```
##   Janeiro Fevereiro     Marco     Abril      Maio     Junho     Julho
##     FALSE     FALSE     FALSE     FALSE     FALSE      TRUE      TRUE
##    Agosto  Setembro   Outubro  Novembro  Dezembro
##      TRUE      TRUE     FALSE     FALSE     FALSE
```

Usamos o vetor "selecao" para fazer o subconjunto do vetor de dados de consumo de café:

```
litros_cafe[selecao]
```

```
##    Junho    Julho   Agosto Setembro
##      8.3      9.7      9.9      9.1
```

Para vetores pequenos, o procedimento adotado para gerar subconjuntos parece desnecessariamente trabalhos. Mas imagine agora que você queira observar os votos de todos os candidatos que atendam a determinada condição – por exemplo, teremo gastos de campanha acima de determinado patarmar. Com uma variável (gasto de campanha) você pode gerar um vetor de seleção que permite gerar o subconjunto desejado.

Voltaremos em variações desse assunto em diversos momentos no futuro.

Factors e variáveis categóricas no R

No tutorial anterior, trabalhamos com tipos de dados e deixamos de lado classe. Neste tutorial, falamos rapidamente de classe e o assunto pareceu relativamente simples. Vamos complicá-lo com um dos aspectos mais confusos da linguagem R: a classe de vetores **factor**.

factor é, basicamente, a classe de vetores em R utilizada para lidar com dados categóricos, nominais ou ordinais. Em vez de gastarmos tinta tentando entendê-los de forma abstrata, vamos a um exemplo.

Suponhamos que temos um vetor de texto que representa uma variável categórica que pode receber dois valores, "sim" e "nao".

```
yes_no <- c("sim", "nao", "nao", "nao", "sim", "nao")
print(yes_no)
```

```
## [1] "sim" "nao" "nao" "nao" "sim" "nao"
```

```r
class(yes_no)
```

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

Vamos usar a função *factor* para gerar o vetor "f_yes_no" e observar o resultado:

```r
f_yes_no <- factor(yes_no)
print(f_yes_no)
```

```
## [1] sim nao nao nao sim nao
## Levels: nao sim
```

Note que não temos mais aspas nos valores impressos do vetor de fatores. Além disso, ele vem acompanhado de uma nova informação: "levels", ou niveis.

Basicamente, "factors" são vetores numéricos cujos valores estão associados a um rótulo. Os "levels" são esses pares de código numérico + rótulo.

Se tentamos transformar em números o vetor de fatores, veja o que obtemos:

```r
as.numeric(f_yes_no)
```

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

1 e 2 são os códigos numéricos gerados autormaticamente para "nao" e "sim", respectivamente. O critério para atribuir valores foi a ordem alfabética dos textos transformados em fatores.

Podemos investigar os níveis de um vetor de fatores com a função *levels*:

```r
levels(f_yes_no)
```

```
## [1] "nao" "sim"
```

E também podemos alterá-los, tal como fazemos com os nomes dos elementos de um vetor:

```r
levels(f_yes_no) <- c("No", "Sim")
print(f_yes_no)
```

```
## [1] Sim No  No  No  Sim No
## Levels: No Sim
```

Tudo bem até agora?

E se os níveis de uma variável forem ordenados? Vejamos um exemplo:

```r
tamanho <- c("alto", "baixo", "baixo", "medio", "alto", "baixo", "medio")
f_tamanho <- factor(tamanho)
print(f_tamanho)
```

```
## [1] alto  baixo baixo medio alto  baixo medio
## Levels: alto baixo medio
```

Como o R segue o ordenamento alfabético, os códigos numéricos 1, 2 e 3 estão associados a "alto", "baixo" e "medio", respectivamente. Mas qual é o problema de não ordenarmos? Veja abaixo:

```r
f_yes_no[1]
```

```
## [1] Sim
## Levels: No Sim
```

```r
f_yes_no[2]
```

```
## [1] No
## Levels: No Sim
```

```
f_yes_no[1] > f_yes_no[2]
```

```
## Warning in Ops.factor(f_yes_no[1], f_yes_no[2]): '>' not meaningful for
## factors
```

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

Sem ordenarmos, não podemos comparar os níveis e estamos assumindo a variável como sendo nominal.

Para ordernar os níveis de um vetor de fatores, temos que informar alguns parâmetros adicionais – *order* e *levels* – ao criá-lo:

```
f_tamanho <- factor(tamanho, order = T, levels <- c("baixo", "medio", "alto"))
print(f_tamanho)
```

```
## [1] alto  baixo baixo medio alto  baixo medio
## Levels: baixo < medio < alto
```

Note que a informação sobre os "levels" acompanha a ordem informada, que, neste caso, é diferente da alfabética. Comparações entre os níveis fazem sentido se a variável for ordinal:

```
f_tamanho[1]
```

```
## [1] alto
## Levels: baixo < medio < alto
```

```
f_tamanho[2]
```

```
## [1] baixo
## Levels: baixo < medio < alto
```

```
f_tamanho[1] > f_tamanho[2]
```

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

```
f_tamanho > "medio"
```

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

Voltaremos aos "factors" em momento adequado. O importante agora é saber que eles existem e que é uma classe de vetores em R. Atenção especial deve ser dada ao fato de que diversas vezes, ao importarmos bases de dados para o "workspace", o R considera variáveis de texto como sendo "factors", mesmo de maneira inadequada. Para evitar este problema, devemos adotar o argumento "stringAsFactors = F" em diversas funções de importação. Como avisei no início deste tópico, "factors" é um dos aspectos mais confusos em R (em conjunto com "missing values", que vamos adiar até um momento adequado).

Exercício

- Crie um vetor de texto com categorias não ordenáveis.
- Crie um vetor de fatores a partir do vetor do item anterior.
- Traduza os níveis para o inglês (ou para o português se já estiverem em inglês)
- Crie um vetor de texto com categorias ordenáveis.
- Crie um vetor de fatores a partir do vetor do item anterior.
- Compare dois elementos do vetor criado no item anterior

Matrizes em R

Durante o curso, utilizaremos poucas ou nenhuma vez matrizes. Há uma razão para isso: estamos interessad@s sorbetudo em dados no formato de *data frame*, que é um caso específico de matriz. Ainda assim, convém rapidamente aprender sobre matrizes para, advinhe, entender um pouco mais sobre *data frames*.

Para matemáticos, matrizes são objetos com uma álgebra própria e parte de uma área denominada Álgebra Linear. Por exemplo, a multiplicação de matrizes, se você lembra dela de seu período de escola ou graduação, segue regras e tem propriedades diferentes da multiplicação de números. Apesar do R ter uma "gramática" para Álgebra Linear (que lembra o MATLAB e Octave), ela não nos interessa agora e a deixaremos de lado.

Para criarmos um matriz, precisamos de um vetor que contenha o número de elementos a serem inseridos em uma matriz. Para uma matriz de 3 linhas e 3 colunas, precisamos de um vetor de 9 elementos, a exemplo do vetor gerado pelo comando "1:9".

Há duas maneiras de "preencher" a matriz: pelas linhas ("byrow = T") ou pelas colunas ("byrow = F"). Veja os exemplos abaixo e tente compreender o funcionamento de *matrix*:

```
matrix(1:9, byrow = TRUE, nrow = 3)
```

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

```
matrix(1:9, byrow = FALSE, nrow = 3)
```

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

Vamos a um exemplo de matriz criada a partir de 3 vetores. Os vetores abaixo representam o gasto de café de 3 pessoas de segunda a sexta-feira em uma semana determinada.

```
beatriz <- c(4, 5, 0, 3, 5)
pedro <- c(2, 2, 2, 2, 2)
mateus <- c(0, 0, 12, 0, 0)
```

Vamos juntar todas os vetores em um só...

```
vetor_cafe <- c(beatriz, pedro, mateus)
```

... e criar uma matriz de 5 linhas e 3 colunas denominada café:

```
cafe <- matrix(vetor_cafe, byrow = FALSE, nrow = 5)
print(cafe)
```

```
##      [,1] [,2] [,3]
## [1,]    4    2    0
## [2,]    5    2    0
## [3,]    0    2   12
## [4,]    3    2    0
## [5,]    5    2    0
```

Veja que as margens da matriz não tem nomes. Vamos proceder como fizemos com vetores. Entretanto, não vamos nomear cada elemento, mas as linhas e colunas, e utilizaremos as funções *rownames* e *colnames* para a tarefa:

```
fregueses <- c("Beatriz", "Pedro", "Mateus")
dias_uteis <- c("Segunda", "Terca", "Quarta", "Quinta", "Sexta")
```

```r
rownames(cafe) <- dias_uteis
colnames(cafe) <- fregueses

print(cafe)
```

```
##         Beatriz Pedro Mateus
## Segunda       4     2      0
## Terca         5     2      0
## Quarta        0     2     12
## Quinta        3     2      0
## Sexta         5     2      0
```

O processo de nomear as linhas e colunas poderia ser realizado no momento em que a matriz foi gerada com o usdo do argumento *dimnames*, como no exemplo:

```r
cafe <- matrix(vetor_cafe, byrow = FALSE, nrow = 5,
               dimnames = list(dias_uteis, fregueses))
```

E se quisermos trocar as linhas pelas colunas – processo conhecido como transposição da matriz? Usamos a função *t*. Dica: *c* e *t* são funções em R. Evite criar objetos com esse nome.

```r
t(cafe)
```

```
##         Segunda Terca Quarta Quinta Sexta
## Beatriz       4     5      0      3     5
## Pedro         2     2      2      2     2
## Mateus        0     0     12      0     0
```

Se quisermos juntar os três vetores originais como se fosse colunas, podemos usar a função *cbind*.

```r
cbind(beatriz, pedro, mateus)
```

```
##      beatriz pedro mateus
## [1,]       4     2      0
## [2,]       5     2      0
## [3,]       0     2     12
## [4,]       3     2      0
## [5,]       5     2      0
```

rbind, função bastante útil e que voltará várias vezes no curso, faz o mesmo tratando os vetores como linhas:

```r
rbind(beatriz, pedro, mateus)
```

```
##         [,1] [,2] [,3] [,4] [,5]
## beatriz    4    5    0    3    5
## pedro      2    2    2    2    2
## mateus     0    0   12    0    0
```

As funções *rowSums* e *colSums*, como é de se esperar, calculam as somas de todos os elementos de cada linha e de cada coluna, respectivamente:

```r
rowSums(cafe)
```

```
## Segunda   Terca  Quarta  Quinta   Sexta
##       6       7      14       5       7
```

```r
colSums(cafe)
```

```
## Beatriz   Pedro  Mateus
##      17      10      12
```

Combinando as funções de soma com as de combinação (sic), podemos gerar os totais nas margens da matriz:

```
Total_Coluna <- colSums(cafe)
cafe2 <- rbind(cafe, Total_Coluna)

Total_Linha <- rowSums(cafe2)
cafe2 <- cbind(cafe2, Total_Linha)
print(cafe2)
```

```
##              Beatriz Pedro Mateus Total_Linha
## Segunda            4     2      0           6
## Terca              5     2      0           7
## Quarta             0     2     12          14
## Quinta             3     2      0           5
## Sexta              5     2      0           7
## Total_Coluna      17    10     12          39
```

Como com vetores, podemos fazer operações aritméticas (veja que não estamos falando de Álgebra Linear) com matrizes. Por exemplo, para transformar os gastos com café em doláres (cotação = 3.2):

```
cafe / 3.2
```

```
##         Beatriz Pedro Mateus
## Segunda  1.2500 0.625   0.00
## Terca    1.5625 0.625   0.00
## Quarta   0.0000 0.625   3.75
## Quinta   0.9375 0.625   0.00
## Sexta    1.5625 0.625   0.00
```

Podemos também relizar operações entre matrizes de mesmas dimensões que considerem os elementos de forma pareada, tal como com vetores. Usando as duas matrizes do começo deste tópico:

```
matrix(1:9, byrow = TRUE, nrow = 3) + matrix(1:9, byrow = FALSE, nrow = 3)
```

```
##      [,1] [,2] [,3]
## [1,]    2    6   10
## [2,]    6   10   14
## [3,]   10   14   18
```

Tanto sobre matriz para chegarmos ao que realmente importa: subconjuntos de matrizes. Com vetores, usamos colchetes para produzirmos subconjuntos. As regras que aprendemos para vetores valem para matrizes. A diferença é que matrizes – e também *data frames* – são objetos com duas dimensões (linha e coluna). Portanto, ao extrairmos um subconjunto, precisamos informar as duas dimensões no colchetes, primeiro linha, depois coluna, separadas por vírgula. Se deixamos uma das dimensões em branco, estamos selecionando todos os elementos daquela dimensão.

Vamos a uma série de exemplos para compreender as regras de subconjunto de matrizes.

Selecionar toda a segunda coluna:

```
cafe[, 2]
```

```
## Segunda   Terca  Quarta  Quinta   Sexta
##       2       2       2       2       2
```

Selecionar toda a terceira linha:

```
cafe[3, ]
```

```
## Beatriz   Pedro  Mateus
##       0       2      12
```

Selecionar o elemento da linha 1 e coluna 3:

```
cafe[1, 3]
```

```
## [1] 0
```

Selecionar os elementos 4 e 5 da coluna 1:

```
cafe[4:5, 1]
```

```
## Quinta  Sexta
##      3      5
```

Selecionar os elementos 1, 3 e 5 da coluna 3:

```
cafe[c(1,3,5), 3]
```

```
## Segunda  Quarta   Sexta
##       0      12       0
```

Selecionar o elemento 4 das colunas 2 e 3:

```
cafe[4, 2:3]
```

```
##  Pedro Mateus
##      2      0
```

Selecionar a segunda e terça-feira de Pedro:

```
cafe[c("Segunda", "Terca"), "Pedro"]
```

```
## Segunda   Terca
##       2       2
```

Selecionar a quarta-feira de todos:

```
cafe["Quarta",]
```

```
## Beatriz   Pedro  Mateus
##       0       2      12
```

Exercício

Subconjunto de matrizes são fundamentais para o futuro do curso. Seja criativ@ e faça 6 exemplos seus com a matriz "cafe" com a qual trabalhamos.

Paramos por aqui

Vimos nesse tutorial aspectos básicos da linguagem R e uma variedade de objetos e funções que utilizaremos no futuro. Deixamos de lado *data frames*, pois já trabalhamos com eles anteriormente e voltaremos a eles a seguir, e listas, que, dada sua complexidade, convém deixar para um estágio mais avançado do curso.