

Big Data Analytics

Análises de dados simples utilizando R

Parte II



Agenda

- Operações matemáticas, relacionais e lógicas básicas utilizando R;
- Operações temporais utilizando R.



Operações matemáticas, relacionais e lógicas básicas utilizando R



Voltando aos vetores

- Já vimos que o objeto mais simples oferecido pelo R é o vetor.

Criando três vetores

x1 <- 10

x2 = 20

assign("x3", 30)

- Estes objetos são vetores de tamanho 1.



Voltando aos vetores

- Podemos criar vetores concatenando outros objetos.

```
# Cria novo vetor com x1, 2, 3 e x2.
```

```
x3 <- c(x1, 2, 3, x2)
```

```
x3
```

```
# Cria vetor com "a", "b" e "c"
```

```
x4 <- c("a", "b", "c")
```

```
x4
```



Voltando aos vetores

- Lembrando que os vetores podem ser compostos pelos seguintes tipos, dentre outros:
 - *Numeric* (número comum)
 - *Integer* (inteiro)
 - *Complex* (número complexo)
 - *Character* (texto)
 - *Logical* (lógicos, booleanos)



Voltando aos vetores

```
# Numeric
```

```
num <- c(522.34, 23, 456)
```

```
# Integer (atenção ao 'L')
```

```
int <- c(5L, 73L)
```

```
# Complex (atenção ao 'i')
```

```
compl <- c(10i, 3 + 5i)
```

```
# Character (atenção às aspas)
```

```
txt <- c("aspas duplas", 'aspas  
simples', "aspas 'dentro' do texto")
```

```
# Logic (maiúsculas!)
```

```
logi <- c(TRUE, FALSE, TRUE)
```



Classes de vetores

- A função `class ()` é útil para identificar a classe de um objeto.

```
class (num)
```

```
class (int)
```

```
class (compl)
```

```
class (txt)
```

```
class (logi)
```



Teste de classe

- Podemos testar se um vetor é de determinada classe com funções `is."classe"`.

`is.numeric(num)`

`is.character(num)`

`is.character(txt)`

`is.logical(txt)`



Coerção

- Lembrando que um objeto do tipo vetor somente pode ter elementos de uma única classe. Não é possível misturar, por exemplo, textos com números.
- Ao misturar elementos de classes diferentes em um vetor, o R faz a coerção do objeto para uma das classes, obedecendo a ordem de prioridade:

`lógico < inteiro < numérico < complexo < texto`

Estrutura de um objeto

- Podemos ver a estrutura de um objeto no R usando a função `str()`.

`str(x3)`

`str(num)`

`str(int)`

`str(comp1)`

`str(txt)`

`str(logi)`



Nomeando elementos de um objeto

- Objetos podem ter elementos nomeados.
- A função **names ()** pode ser usada para consultar os nomes de um objeto.

```
num
```

```
names (num) <- c ("num1", "num2", "num3")
```

```
num
```

```
names (num)
```



Índices de um vetor

- Podemos acessar elementos de um vetor usando colchetes "[]".

`num[1]` # Primeiro elemento

`num[c(1,2)]` # Elementos 1 e 2

`num[c(1,3)]` # Elementos 1 e 3

`num[c(3,1,2)]` # Alterando a ordem



Selecionando por nomes

- Em vetores nomeados, podemos selecionar elementos pelo nome.

```
num["num1"]
```

```
num["num2"]
```

```
num[c("num1", "num3")]
```



Selecionando por vetores lógicos

- Podemos usar vetores lógicos para selecionar elementos.

```
# Seleciona apenas o 1o elemento
```

```
num[c(TRUE, FALSE, FALSE)]
```

```
# Seleciona apenas o 3o elemento
```

```
num[c(FALSE, FALSE, TRUE)]
```

```
# Seleciona o 1o e o 3o elemento
```

```
num[c(TRUE, FALSE, TRUE)]
```



Alterando elementos

- Podemos usar o operador "<-" junto com os colchetes "[" "]" para alterar elementos específicos do vetor.

```
num
```

```
# Alterando o 1o elemento 100
```

```
num[1] <- 100
```

```
num
```

```
# Alterando o 2o e 3o elementos
```

```
num[2:3] <- c(12.3, -10)
```

```
num
```



Ordenando um vetor

- A função `order()` retorna um vetor com as posições para que um objeto fique em ordem crescente.
- A função `sort()` retorna o vetor ordenado.
- As duas funções tem o parâmetro “**decreasing**” que ao ser configurado como **TRUE**, retornam o vetor em ordem decrescente.

`order(num)` # Índices p/ ordenação crescente

`num[order(num)]` # Ordena o vetor num

`sort(num)`



Sequências e repetições

- Podemos criar uma sequência de inteiros utilizando dois pontos ":".

```
# Criando uma sequência de 1 a 10  
1:10
```

```
# Criando uma sequência de -1 a -10
```

```
# Atenção: não é criada na ordem dos
```

```
# inteiros!
```

```
-1: (-10)
```



Sequências e repetições

- Podemos também criar sequências mais flexíveis com a função `seq()`.
- Para repetições, temos a função `rep()`.

```
seq(from = 1, to = 10, by = 3)
```

```
rep(1, times = 10)
```

```
rep(c(1,2), times = 5)
```



Vetorização

- Diversos operadores da linguagem R são vetorizados, ou seja, os cálculos são realizados elemento a elemento do vetor.
- Veremos diversos exemplos adiante.



Operadores matemáticos básicos

- Todos os operadores abaixo são vetorizados.

$(x + y)$ Soma

$(x - y)$ Subtração

$(x * y)$ Multiplicação

(x / y) Divisão

$(x ^ y)$ Exponenciação

$(x \% / \% y)$ Divisão por inteiro

$(x \% \% y)$ Resto da divisão



Operadores matemáticos básicos

Soma

$1 + 20$

Soma vetorizada

$c(1,2,3) + c(4,5,6)$

Subtração

$200 - 2$

Subtração vetorizada

$c(1,2,3) - c(4,5,6)$



Operadores matemáticos básicos

Divisão

200 / 15

Divisão

c(2,4,6) / c(1,2,3)

Multiplicação

2*10

Multiplicação vetorizada

c(10,9,8) * c(1,2,3)



Operadores matemáticos básicos

Exponenciação

4^2

Exponenciação vetorizada

$c(2,2,2) \wedge c(1,2,3)$

Divisão inteira

$7 \% / \% 3$

Divisão inteira vetorizada

$c(7,7) \% / \% c(3,2)$

Módulo (resto da divisão)

$7 \% \% 3$

Módulo vetorizado

$c(7,7) \% \% c(3,2)$



Operações com vetores de tamanhos diferentes

- Ao fazermos operações com vetores de tamanhos diferentes, o R vai “expandindo” os valores do vetor menor até que este fique com a mesma quantidade de elementos do vetor maior.

Operação

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

```
x * 2
```

Equivale à

```
x * c(2, 2, 2, 2)
```



Operações com vetores de tamanhos diferentes

- E quando o vetor menor possuir dois elementos?

```
# Operação
```

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

```
x * c(2, 3)
```

```
# Equivale à
```

```
x * c(2, 3, 2, 3)
```



Operações com vetores de tamanhos diferentes

- E se tentarmos fazer uma operação onde tamanhos dos vetores não são múltiplos?

```
# Operação
```

```
x * c(2, 3, 1)
```

```
# Warning message:
```

```
In x * c(2, 3, 1) :
```

```
  longer object length is not a  
multiple of shorter object length
```



Outras funções

- Todas as funções abaixo são vetorizadas.

<code>abs(x)</code>	Valor absoluto
<code>log(x)</code>	Logaritmo natural
<code>log10(x)</code>	Logaritmo (base 10)
<code>log2(x)</code>	Logaritmo (base 2)
<code>logb(x, b)</code>	Logaritmo (base b)
<code>exp(x)</code>	Exponencial
<code>sqrt(x)</code>	Raiz quadrada
<code>factorial(x)</code>	Fatorial



Outras funções

```
x <- c(1,2,-3, 4, -20.3)
```

```
abs(x) # Valor absoluto
```

```
log(x) # Logaritmo natural
```

```
Warning message:
```

```
In log(x) : NaNs produced
```

```
exp(x) # Exponencial
```



Outras funções

`sqrt(x)` #Raiz quadrada

Warning message:

In `log(x)` : NaNs produced

`factorial(5)` # Fatorial $(5*4*3*2*1)$



Funções trigonométricas

`sin(pi)` # Seno

`cos(pi)` # Cosseno

`tan(pi)` # Tangente, etc.

Obs.: tente executar `sin(pi)`. Observe que o resultado de não foi exatamente zero (como deveria ser). Como qualquer outra linguagem, o R trabalha com números de ponto flutuante, então é preciso tomar alguns cuidados com a chamada propagação de erro.



Estatísticas descritivas em vetores

Soma e soma acumulada.

`sum(x)` , `cumsum(x)`

Produto e produto acumulado.

`prod(x)` , `cumprod(x)`

Mínimo, mínimo acumulado e mínimo par a par.

`min(x)` , `cummin(x)` , `pmin(x, y)`

Máximo, máximo acumulado e máximo par a par.

`max(x)` , `cummax(x)` , `pmax(x, y)`



Estatísticas descritivas em vetores

Média.

`mean(x)`

Variância e desvio-padrão.

`var(x), sd(x)`

Covariância e correlação.

`cov(x, y), cor(x, y)`

Primeira diferença.

`diff(x)`



Outras funções

```
x <- c(1,2,-3, 4, -20.3)
```

```
mean(x) # Média
```

```
sum(x) # Somatório
```

```
prod(x) # Produto
```

```
cumsum(x) # Somatório acumulado
```

```
cumprod(x) # Produto acumulado
```



Outras funções

```
y <- 1:5
```

```
var(x) # Variância
```

```
sd(x) # Desvio-padrão
```

```
median(x) # Mediana
```

```
cov(x, y) # Covariância
```

```
cor(x, y) # Correlação entre x e y
```



Outras funções

`min(x)` # Mínimo

`max(x)` # Máximo

`cummin(x)` # Mínimo "acumulado"

`cummax(x)` # Máximo "acumulado"

`diff(x)` # Diferença



Operadores relacionais

- São importantes para determinar a relação entre dois vetores.
- Seu resultado é um vetor lógico, sendo também vetorizados.

`x == y` `x` é igual a `y`? (Coerção)

`x != y` `x` é diferente de `y`?

`x > y` `x` é maior do que `y`?

`x >= y` `x` é maior ou igual a `y`?

`x < y` `x` é menor do que `y`?

`x <= y` `x` é menor ou igual a `y`?



Operadores relacionais

```
x <- 10
```

```
y <- 20
```

```
x == y
```

```
x <- c(10, 20, 30)
```

```
y <- c(10, 10, 30)
```

```
x == y
```

```
# Montando um vetor com elementos de x  
# que são iguais a y
```

```
x[x == y]
```



Operadores relacionais

Coerção! Converte x para texto e

compara textualmente.

```
x <- c(10, 20)
```

```
y <- c("10", "20")
```

```
x == y
```

Cuidado com as, pois podem ser

gerados resultados inesperados:

```
20 > "100"
```

Resultado correto pela ordem alfabética.



Operadores relacionais

- Para verificar se dois vetores são exatamente idênticos, melhor utilizar a função `identical()`.

```
identical(x, y)
```

Atenção com os casos abaixo!

O que está acontecendo?

```
identical(sin(pi), 0)
```

```
identical(0.1, 0.3 - 0.2)
```



Operadores relacionais

- Devemos ter cuidado ao fazer comparações de números resultantes de cálculos pois devemos considerar uma tolerância de erro do ponto flutuante.
- O R tem uma função para isso chamada `all.equal()`.

`# Resultado falso, incorreto!`

`sin(pi) == 0`

`# Resultado verdadeiro, correto!`

`all.equal(sin(pi), 0)`



Operadores relacionais

- Curiosidade: vamos testar se a diferença absoluta entre um número e outro é irrelevante (do ponto de vista numérico).

`abs(sin(pi) - 0) < 1e-12`



Operadores relacionais

- Devemos lembrar que os resultados destas operação são vetores lógicos, que podem ser utilizados em operações subsequentes.

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

```
y <- c("1","3","2","4","5") ;
```

```
# Armazena o resultado da comparação em
```

```
# "resultado"
```

```
resultado <- (x >= y)
```

```
resultado
```

```
# Usa o vetor 'resultado' para fazer um
```

```
# subconjunto x
```

```
x[resultado]
```



Operadores relacionais

- Lembrando que vetores lógicos, quando convertidos para numéricos, são transformados em vetores de 0's e 1's.
- Assim, se quisermos saber quantos x são maiores ou iguais a y, basta somar os 1's do vetor 'resultado'.

`as.numeric(resultado)`

`sum(resultado)`



Operadores lógicos

- Operadores lógicos E(AND) ' $\&$ ' e OU(OR) ' $|$ '.

c(TRUE, FALSE) & c(TRUE, TRUE)

c(TRUE, FALSE) | c(TRUE, TRUE)



Funções de operadores lógicos

- Outros comandos são o **all()** e o **any()**.
- **all()** retorna TRUE se todos os elementos forem TRUE.
- **any()** retorna TRUE se ao menos um elemento for TRUE.



Funções de operadores lógicos

```
# Define a semente da simulação
```

```
set.seed(11)
```

```
# simula 1000 observações seguindo
```

```
# distribuição normal(5, 2)
```

```
x <- rnorm(1000, 5, 2)
```

```
# Existe algum x maior que 10?
```

```
any(x > 10)
```

```
# Todos os x estão entre -1 e 20?
```

```
all(x > -20 & x < 20)
```



Funções de operadores lógicos

- A função `which()` retorna a posição dos elementos que são TRUE.

```
which(c(TRUE, FALSE, TRUE))
```

```
# Quais as posições dos elementos de x
```

```
# maiores do que 10?
```

```
which(x > 10)
```



Funções de conjuntos

```
x <- 1:5
```

```
y <- c(1:3, 6:10)
```

```
setdiff(x, y)
```

```
intersect(x, y)
```

```
union(x, y)
```

```
x %in% y
```



Operações temporais utilizando R



Datas no R

- O R possui uma série de funções para tratar datas.
- Se tivermos uma data em formato texto (Ex.: “01 de janeiro de 2014”) podemos transformá-la em um objeto do tipo “Date” que aceita operações como comparação, adição, entre outras.

Datas no R

```
Data <- "11 de outubro de 2016"
```

```
Data <- as.Date(Data, format="%d de %B de  
%Y")  
str(Data)
```

- Para armazenar a data no objeto apropriado, temos que explicar, via o parâmetro **format**, como a data está codificada no texto.
- Especificamos que primeiramente temos o dia (**%d**) seguido da palavra "**de**", depois temos o mês por extenso (**%b**) seguido da palavra "**de**" e finalmente o ano com 4 dígitos (**%Y**).



Operações com datas

`Data + 1`

`Data - 1`

`weekdays(Data)`

`Data > "2016-10-01"`

`months(Data + 31)`

`quarters(Data)`

`seq.Date(from = Data, by = 1, length.out
= 10L)`

Opções do parâmetro format

<code>%Y</code>	Ano com 4 dígitos.
<code>%y</code>	Ano com 1/2 dígitos.
<code>%m</code>	Mês com 4 dígitos.
<code>%B</code>	Mês por extenso (completo).
<code>%b</code>	Mês por extenso (abreviado).
<code>%d</code>	dia com 1/2 dígitos.
<code>%A</code>	Dia da semana (por extenso).
<code>%a</code>	Dia da semana (abreviado).
<code>%w</code>	Dia da semana (número).

POSIXct e POSIXlt

- Podemos adicionar informações sobre hora, minuto e segundo para as datas.
- Existem dois formatos principais que tratam disso:
 - POSIXct (formato estabelecido após 1970)
 - POSIXlt (lista nomeada com objetos de data e hora).



POSIXct e POSIXlt

%H **Hora (00–23)**

%I **Hora (1–12)**

%M **Minutos (00–59)**

%S **Segundos (00–61)**

%p **AM/PM (não vale para o Brasil)**

POSIXct e POSIXlt

```
Data <- "11 de outubro de 2014 às  
19h e 40m"
```

```
ct <- as.POSIXct(Data, format="%d de  
%B de %Y às %Hh e %Mm")
```

```
ct
```

```
lt <- as.POSIXlt(Data, format="%d  
de %B de %Y às %Hh e %Mm")
```

```
lt
```

Operações com datas

- Podemos fazer operações com datas.

`ct + 3600 # Soma uma hora`
`ct`

`lt - 60 # Subtrai um minuto`
`ct`

`months(ct)`

`weekdays(lt)`