MINERAÇÃO DE DADOS COMPLEXOS

Curso de Aperfeiçoamento

UNICAMP

Análise de Dados

Tipos de Dados

Prof. Zanoni Dias

2020

Instituto de Computação – Unicamp

Roteiro

Apresentação da Linguagem R

R

- R é uma linguagem de programação e um ambiente integrado de desenvolvimento de software.
- É uma das linguagens de programação mais populares para análise de dados (se não a mais popular).
- R foi criada por Ross Ihaka e por Robert Gentleman no departamento de estatística da universidade de Auckland, Nova Zelândia, em 1991.
- O nome da linguagem é derivado parcialmente dos nomes de seus autores e parcialmente como associação à linguagem S, da qual R é derivada.

S

- S foi originalmente desenvolvida por John Chambers e colegas no Bell Labs, em 1976.
- As versões originais da linguagem foram implementadas usando subrotinas em Fortran.
- Em 1988 o sistema foi reescrito em C (versão S3 da linguagem).

S

- A versão S4 foi lançada em 1998 e ainda é usada hoje em dia.
- John Chambers: "We wanted users to be able to begin in an interactive environment, where they did not consciously think of themselves as programming. Then as their needs became clearer and their sophistication increased, they should be able to slide gradually into programming."
- Possui duas implementações mais modernas:
  - S-PLUS (comercial)
  - R (gratuita)

Principais Versões de R

- R foi criada em em 1991, mas anunciada apenas em 1993.
- Se tornou parte do projeto GNU em 1997, na sua versão 0.60.
- Versão 1.0 foi lançada em 2000, considerada a primeira versão estável o suficiente para uso comercial.
- Versão 2.0, lançada em 2004, apresentou melhorias significativas no uso de memória para grandes volumes de dados.
- Versão 3.0, lançada em 2013, permite indexação numérica superior a 2^{31}, para sistemas 64 bits.
- A sua versão mais recente é a 3.6.3 (Holding the Windsock).

Características de R

- Principais vantagens:
  - Possui suporte a um grande número de plataformas e sistemas operacionais (Linux, Mac OS, Windows, etc).
  - Comunidade ativa, com atualizações frequentes.
  - Suporte a análises estatísticas e construção de gráficos.
  - Imensa quantidade de pacotes disponíveis publicamente, sendo cerca de 15.400 no CRAN (The Comprehensive R Archive Network – `https://cran.r-project.org/web/packages`).
- Principais desvantagens:
  - Por ser uma linguagem interpretada, não tem como principal característica a eficiência.
  - Dados precisam ser armazenados em memória para serem analisados, o que pode ser um problema com grandes volumes de dados.

Informações Básicas

- R:
  - `https://www.r-project.org`
- RStudio:
  - `https://www.rstudio.com`
- Tutoriais para instalação do R e do RStudio estão disponíveis no Moodle da disciplina:
  - `https://moodle.lab.ic.unicamp.br/moodle`

Informações Básicas

Atribuição

- Em R, o prompt do interpretador é indicado pelo símbolo >.
- O operador de atribuição usado em R é <- (seta para a esquerda).
- Comentários em R são indicados pelo caractere #. R ignora o resto da linha (todos os caracteres à direita, inclusive ele).
- Exemplo:

```
> x <- 5
> x # impressao implicita
[1] 5
```

Atribuição

- Exemplo:

```
> y <-  # comando incompleto
+       # prompt de um comando incompleto
+ 2
> print(y)
[1] 2
```

- No exemplo acima, ao avaliar a expressão `y`, R indica que trata-se de um vetor, cujo primeiro elemento é o número 2 (`[1] 2`).

Atribuição

- Podemos listar os objetos existentes usando a função `ls()`.

```
> ls()
[1] "x" "y"
```

- Podemos remover um objeto usando a função `rm()`.

```
> rm(x)
> ls()
[1] "y"
```

- Podemos remover todos os objetos existentes de uma só vez.

```
> rm(list = ls())
> ls()
character(0)
```

Tipos Básicos

Tipos Básicos

- Em R existem 5 classes de objetos básicos (ou atômicos):
  - Caractere (character = *string*).
  - Numérico (numeric = número real).
  - Inteiro (integer = número inteiro).
  - Complexo (complex = número complexo).
  - Lógico (logical = booleano).

Tipos Básicos

- Exemplo de objetos com tipos básicos:

```
> class("Ola")
[1] "character"
```

```
> class(1)
[1] "numeric"
```

```
> class(1L)
[1] "integer"
```

```
> class(1 + 4i)
[1] "complex"
```

```
> class(TRUE)
[1] "logical"
```

Números

- Números em R são geralmente tratados como objetos numéricos com precisão dupla (para armazenamento de números reais).
- Podemos explicitamente especificar que queremos um número inteiro usando o sufixo `L`.
- Exemplo:
  - `1` é um objeto numérico (precisão dupla).
  - `1L` é um número inteiro.

Números

- R possui duas constantes numérica importantes:
  - Para representar infinito, R usa o valor `Inf`. Por exemplo, o resultado da expressão `1/0` é `Inf`, assim como a expressão `1/Inf` é igual a `0`.
  - O valor `NaN` ("Not a Number") é usado para representar valores indefinidos. Por exemplo, o resultado da expressão `0/0` é `NaN`.

Verificação de Tipo

- É possível verificar o tipo do objeto usando as funções `is.numeric()`, `is.integer()`, `is.complex()`, `is.logical()` e `is.character()`.
- Exemplo:

```
> is.numeric(12)
[1] TRUE
```

```
> is.integer(12)
[1] FALSE
```

```
> is.integer("12")
[1] FALSE
```

Tipos Complexos

Tipos Complexos

- Principais tipos complexos em R:
  - Vetores
  - Matrizes
  - Listas
  - Fatores
  - Data frames

Vetores

- Podemos usar a função `c()` para criar vetores de objetos.
- Os objetos de um vetor devem ser todos do mesmo tipo.
- Exemplos de vetores de objetos numéricos:

```
> x <- c(0.5, 3.1, 0.6, 1.5); x
[1] 0.5 3.1 0.6 1.5
```

```
> c(1, 1.5, 2, 2.5, 3, 3.5, 4)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0
```

Vetores

- Exemplos de vetores de objetos lógicos:

```
> x <- c(TRUE, FALSE, TRUE, TRUE, FALSE); x
[1] TRUE FALSE TRUE TRUE FALSE
```

```
> y <- c(T, F, F, F, F, F); y
[1] TRUE FALSE FALSE FALSE FALSE FALSE
```

- Exemplos de vetores de objetos caracteres:

```
> c("a", "b", "c")
[1] "a" "b" "c"
```

```
> c("R", "C", "Java", "C++", "Python")
[1] "R" "C" "Java" "C++" "Python"
```

Vetores

- Exemplos de vetores de objetos inteiros:

```
> c(12L, -3L, 7L, -2L)
[1] 12 -3 7 -2
```

```
> c(1L, 2L, 3L, 4L, 5L)
[1] 1 2 3 4 5
```

- Exemplos de vetores de objetos complexos:

```
> c(1+0i, 2+4i)
[1] 1+0i 2+4i
```

```
> c(1+1i, 2-2i, -3+3i, -4-4i)
[1] 1+1i 2-2i -3+3i -4-4i
```

Sequências

- Podemos criar uma sequência (um vetor) de valores numéricos com o operador : (dois pontos).
- Exemplos:

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

```
> 10:20
[1] 10 11 12 13 14 15 16 17 18 19 20
```

```
> -3:3
[1] -3 -2 -1  0  1  2  3
```

Sequências

- Exemplos (continuação):

```
> 3.2:10
[1] 3.2 4.2 5.2 6.2 7.2 8.2 9.2
```

```
> 3:3
[1] 3
```

```
> 7:2
[1] 7 6 5 4 3 2
```

Sequências

- Também podemos criar sequências utilizando a função `seq()`.
- Exemplos:

```
> seq(from = 1, to = 10, by = 1)
[1] 1 2 3 4 5 6 7 8 9 10
```

```
> seq(from = 1, to = 10, by = 2)
[1] 1 3 5 7 9
```

```
> seq(from = 3, by = 3, length.out = 10)
[1] 3 6 9 12 15 18 21 24 27 30
```

Sequências

- É possível ainda criar sequências utilizando a função de repetição `rep()`.
- Exemplos:

```
> rep(x = 1, times = 5)
[1] 1 1 1 1 1
```

```
> rep(x = c(1,2,3), times = 2)
[1] 1 2 3 1 2 3
```

```
> rep(x = c(1,2,3), times = 3, each = 2)
[1] 1 1 2 2 3 3 1 1 2 2 3 3 1 1 2 2 3 3
```

Vetores

- Podemos concatenar vetores usando a função `c()`:

```
> x <- c(1, 2, 3, 2, 1)
> y <- 6:8
> z <- rep(x = 7, times = 3)
```

```
> c(x, y)
[1] 1 2 3 2 1 6 7 8
```

```
> c(y, z)
[1] 6 7 8 7 7 7
```

```
> c(y, 4, z, 5, 5, x)
[1] 6 7 8 4 7 7 7 5 5 1 2 3 2 1
```

Vetores

- Podemos usar a função `c()` para adicionar elementos no começo ou no fim de um vetor:

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

```
> x <- c(x, 7); x
[1] 1 2 3 2 1 7
```

```
> x <- c(x, 5); x
[1] 1 2 3 2 1 7 5
```

```
> x <- c(0, x); x
[1] 0 1 2 3 2 1 7 5
```

Vetores

- Um vetor vazio é representado por `NULL`:

```
> rm(x)
> x <- c(x, 1)
Error: object 'x' not found
```

```
> x <- NULL; x
NULL
```

```
> x <- c(x, 1); x
[1] 1
```

```
> x <- c(x, 1); x
[1] 1 1
```

Coerção Implícita

- Como mencionado anteriormente, os objetos de um vetor devem ser todos do mesmo tipo.
- Sendo assim, R sempre faz coerção (implícita) dos elementos de um vetor, para garantir a propriedade acima.
- Exemplos:

```
> c(1.3, 2.1, 1.7, T, 1.2)
[1] 1.3 2.1 1.7 1.0 1.2
```

```
> c(1.3, 2.1, 1.7, "a", 1.2)
[1] "1.3" "2.1" "1.7" "a" "1.2"
```

Coerção Implícita

- Exemplos (continuação):

```
> c(TRUE, 2, FALSE, TRUE)
[1] 1 2 0 1
```

```
> c("a", TRUE, 2, "b", FALSE, 1)
[1] "a" "TRUE" "2" "b" "FALSE" "1"
```

```
> c(1+2i, 1.2, TRUE, 1-1i, -0.7, FALSE)
[1] 1.0+2i 1.2+0i 1.0+0i 1.0-1i -0.7+0i 0.0+0i
```

Coerção Explícita

- É possível fazer coerção explícita de tipo usando as funções `as.numeric()`, `as.integer()`, `as.complex()`, `as.logical()` e `as.character()`.
- Exemplo:

```
> x <- -1.3:3; x
[1] -1.3 -0.3 0.7 1.7 2.7
> as.integer(x)
[1] -1 0 0 1 2
> as.logical(x)
[1] TRUE TRUE TRUE TRUE TRUE
> as.logical(as.integer(x))
[1] TRUE FALSE FALSE TRUE TRUE
```

Coerção Explícita

- Exemplo (continuação):

```
> x <- -1.3:3; x
[1] -1.3 -0.3 0.7 1.7 2.7
> as.complex(x)
[1] -1.3+0i -0.3+0i 0.7+0i 1.7+0i 2.7+0i
> y <- as.character(x); y
[1] "-1.3" "-0.3" "0.7" "1.7" "2.7"
> as.complex(y)
[1] -1.3+0i -0.3+0i 0.7+0i 1.7+0i 2.7+0i
```

Vetores

- É possível criar um vetor com a função `vector()`, especificando o tipo dos objetos e número de elementos (objetos).
- Exemplos (continuação):

```
> vector(mode = "numeric", length = 10)
[1] 0 0 0 0 0 0 0 0 0 0
```

```
> vector(length = 8, mode = "integer")
[1] 0 0 0 0 0 0 0 0
```

Vetores

- Exemplos (continuação):

```
> vector("logical", length = 5)
[1] FALSE FALSE FALSE FALSE FALSE
```

```
> vector("complex", 6)
[1] 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i
```

```
> vector("character", 12)
[1] "" "" "" "" "" "" "" "" "" "" "" ""
```

Valores Desconhecidos

- Os valores `NA` (Not Available) e `NaN` (Not a Number) são usados para indicar valores desconhecidos.
- As funções `is.na()` e `is.nan()` são usadas, respectivamente, para testar se um valor é `NA` ou `NaN`.
- Um valor `NaN` também é um `NA`, mas o contrário não é verdade.
- Exemplos:

```
> x <- c(1, 2, NaN, NA, 4)
> is.na(x)
[1] FALSE FALSE TRUE TRUE FALSE
> is.nan(x)
[1] FALSE FALSE TRUE FALSE FALSE
```

Matrizes

- Matrizes são vetores com um atributo especial `dim` (dimensões).
- Assim como vetores, todos os objetos de uma matriz devem ser do mesmo tipo.
- O atributo `dim` (dimensões) é um vetor de tamanho (pelo menos) 2.

Matrizes

- Exemplo:

```
> m <- matrix(nrow = 2, ncol = 3); m
     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
> dim(m)
[1] 2 3
> attributes(m)
$dim
[1] 2 3
```

Matrizes

- É possível especificar o conteúdo inicial de uma matriz (`data`).
- Exemplos:

```
> matrix(data = 2, nrow = 2, ncol = 3)
     [,1] [,2] [,3]
[1,]    2    2    2
[2,]    2    2    2
```

```
> matrix(data = 1:2, nrow = 2, ncol = 3)
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
```

Matrizes

- Exemplos (continuação):

```
> matrix(data = 1:3, nrow = 2, ncol = 3)
     [,1] [,2] [,3]
[1,]    1    3    2
[2,]    2    1    3
```

```
> matrix(data = 1:6, nrow = 2, ncol = 3)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Matrizes

- Por padrão, matrizes são construídas coluna a coluna.
- No entanto, é possível especificar que uma matriz seja construída linha a linha (`byrow`).
- Exemplos:

```
> matrix(data = 1:2, nrow = 2, ncol = 3,
+        byrow = TRUE)
     [,1] [,2] [,3]
[1,]    1    2    1
[2,]    2    1    2
```

Matrizes

- Exemplos (continuação):

```
> matrix(data = 1:3, nrow = 2, ncol = 3,
+        byrow = TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    1    2    3
```

```
> matrix(data = 1:6, nrow = 2, ncol = 3,
+        byrow = TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Matrizes

- Matrizes também podem ser construídas diretamente de vetores, adicionando um atributo `dim` (dimensões).
- Exemplos:

```
> m <- 1:24; m
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
[13] 13 14 15 16 17 18 19 20 21 22 23 24
> dim(m) <- c(4,6)
> m
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
[4,]    4    8   12   16   20   24
```

Matrizes

- Exemplos (continuação):

```
> dim(m) <- c(2,4,3); m
, , 1
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
, , 2
     [,1] [,2] [,3] [,4]
[1,]    9   11   13   15
[2,]   10   12   14   16
, , 3
     [,1] [,2] [,3] [,4]
[1,]   17   19   21   23
[2,]   18   20   22   24
```

Matrizes

- Matrizes podem ser criadas juntando vetores, linha a linha (`rbind()`) ou coluna a coluna (`cbind()`).
- Exemplos:

```
> x <- 1:3
> y <- 7:9
> rbind(x, y)
  [,1] [,2] [,3]
x    1    2    3
y    7    8    9
> cbind(x, y)
     x y
[1,] 1 7
[2,] 2 8
[3,] 3 9
```

Matrizes

- Exemplos (continuação):

```
> rbind(1:4, 5:8, 3:6)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    3    4    5    6
```

```
> cbind(1:3, 9:7, 5:7, 6:4)
     [,1] [,2] [,3] [,4]
[1,]    1    9    5    6
[2,]    2    8    6    5
[3,]    3    7    7    4
```

Matrizes

- Ao criar uma matriz utilizando vetores, caso o tamanho de algum dos vetores não seja múltiplo do tamanho do maior vetor o R irá gerar um alerta.
- Exemplos:

```
> x <- 1:3; y <- 9:6; z <- 4:5
> rbind(x, y)
  [,1] [,2] [,3] [,4]
x    1    2    3    1
y    9    8    7    6
Warning message:
 In rbind(x, y):
 number of columns of result is not a multiple
    of vector length (arg 1)
```

Matrizes

- Exemplos (continuação):

```
> x <- 1:3; y <- 9:6; z <- 4:5
> cbind(x, y, z)
     x y z
[1,] 1 9 4
[2,] 2 8 5
[3,] 3 7 4
[4,] 1 6 5
Warning message:
 In cbind(x, y, z):
 number of rows of result is not a multiple of
    vector length (arg 1)
```

Matrizes

- Matrizes podem ser criadas juntando matrizes, uma de baixo da outra (`rbind()`) ou lado a lado (`cbind()`).
- Exemplos:

```
> a <- matrix(data = 1:6, nrow = 2, ncol = 3)
> a
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> rbind(a, a)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
[3,]    1    3    5
[4,]    2    4    6
```

Matrizes

- Exemplos (continuação):

```
> a <- matrix(data = 1:6, nrow = 2, ncol = 3)
> a
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> cbind(a, a)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    5    1    3    5
[2,]    2    4    6    2    4    6
```

Listas

- Em R é possível criar uma lista (`list`) onde cada elemento é um objeto, um vetor, uma matriz ou mesmo uma outra lista.
- Exemplos:

```
> list("a", 2.3, FALSE, 1+2i)
[[1]]
[1] "a"
[[2]]
[1] 2.3
[[3]]
[1] FALSE
[[4]]
[1] 1+2i
```

Listas

- Exemplos (continuação):

```
> list(c(FALSE, 2), matrix(ncol = 3, nrow = 2),
+      list("b", 3:5))
[[1]]
[1] 0 2
[[2]]
     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
[[3]]
[[3]][[1]]
[1] "b"
[[3]][[2]]
[1] 3 4 5
```

Fatores

- Fatores são usados para representar categorias.
- A função `factor()` é usada para criar uma coleção de fatores.
- Exemplos:

```
> factor(c("yes", "yes", "no", "yes", "no"))
[1] yes yes no yes no
Levels: no yes
```

```
> factor(c("male", "female", "female", "male"))
[1] male female female male
Levels: female male
```

Fatores

- Exemplos (continuação):

```
> factor(c("M", "G", "G", "P", "G", "M"))
[1] M G G P G M
Levels: G M P
```

```
> factor(c("M", "G", "G", "P", "G", "M"),
+        levels = c("P", "M", "G"),
+        ordered = TRUE)
[1] M G G P G M
Levels: P < M < G
```

Data Frames

- Data frames são usados para armazenar dados tabulares.
- Data frames são tipos especiais de listas onde cada um dos seus elementos são vetores (cada um deles representando uma coluna), todos com o mesmo número de subelementos (todas as colunas possuem o mesmo número de elementos).
- Ao contrário das matrizes (e similar às listas), cada coluna pode armazenar objetos de um tipo diferente.
- Data frames são criados usando a função `data.frame()`.
- Dados tabulares podem ser lidos e armazenados em data frames usando as funções `read.table()` e `read.csv()`.

Data Frames

- Exemplos:

```
> name <- c("Alice", "Bob", "Julia")
> age <- c(19, 21, 20)
> dados <- data.frame(nome = name, idade = age,
+                     stringsAsFactors = FALSE)
> dados
   nome idade
1 Alice    19
2   Bob    21
3 Julia    20
```

```
> dados[[1]]    # acessando a primeira coluna
[1] "Alice" "Bob" "Julia"
```

Data Frames

- Exemplos (continuação):

```
> name <- c("Alice", "Bob", "Julia")
> age <- c(19,21,20)
> dados <- data.frame(nome = name, idade = age)
> dados
   nome idade
1 Alice    19
2   Bob    21
3 Julia    20
```

```
 > dados[[1]]     # acessando a primeira coluna
[1] Alice Bob Julia
Levels: Alice Bob Julia
```

Data Frames

- Exemplos (continuação):

```
> cliente <- c(T, F, T)
> dados <- cbind(dados, cliente)
> dados
   nome idade cliente
1 Alice    19    TRUE
2   Bob    21   FALSE
3 Julia    20    TRUE
```

Data Frames

- Exemplos (continuação):

```
> jack <- data.frame(nome = "Jack", idade = 22,
+                    cliente = F)
> jack
  nome idade cliente
1 Jack    22   FALSE
> dados <- rbind(dados, jack)
> dados
   nome idade cliente
1 Alice    19    TRUE
2   Bob    21   FALSE
3 Julia    20    TRUE
4  Jack    22   FALSE
```

Data Frames

- Exemplos (continuação):

```
> data.frame(idade = c(25, 36, 24, 33),
+            cliente = c(F, T, F, F),
+            row.names = c("Aline", "Bianca",
+                          "Carlos", "Daniel"))
        idade cliente
Aline      25   FALSE
Bianca     36    TRUE
Carlos     24   FALSE
Daniel     33   FALSE
```

Nomes

- Objetos em R podem ter nomes (`names`), que são muito úteis na escrita de códigos legíveis e na definição de objetos auto-descritivos.
- Exemplos:

```
> x <- c(24, 32, 28); x
[1] 24 32 28
> names(x)
NULL
> names(x) <- c("ana", "beatriz", "carlos")
> x
ana beatriz carlos
 24      32     28
> names(x)
[1] "ana" "beatriz" "carlos"
```

Nomes

- Exemplos (continuação):

```
> x <- list(digitos = 0:9,
+           letras = c("a", "b", "c"),
+           alternativas = factor(c("V", "F")))
> x
$digitos
[1] 0 1 2 3 4 5 6 7 8 9

$letras
[1] "a" "b" "c"

$alternativas
[1] V F
Levels: F V
```

Nomes

- Exemplos (continuação):

```
> m <- matrix(1:12, nrow = 3, ncol = 4); m
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> dimnames(m) <- list(c("SP", "RJ", "MG"),
+                     c("A", "B", "C", "D"))
> m
   A  B  C  D
SP 1  4  7 10
RJ 2  5  8 11
MG 3  6  9 12
```

Subconjuntos

Subconjuntos

- Existem alguns operadores em R que podem ser usados para extrair um subconjunto de objetos (de um vetor, de uma matriz, de uma lista, etc):
  - `[]` retorna um objeto da mesma classe do objeto original. Pode ser usado para extrair mais do que um elemento.
  - `[[]]` é usado para extrair elementos de listas ou data frames. Extrai um único elemento, sendo que este elemento não necessariamente será uma lista ou um data frame.
  - `$` é usado para extrair elementos de uma lista ou de um data frame pelo nome do objeto. Semanticamente similar a `[[]]`.

Vetores

- Exemplos:

```
> x <- c("e", "b", "a", "d", "a", "f", "d"); x
[1] "e" "b" "a" "d" "a" "f" "d"
```

```
> x[1]
[1] "e"
```

```
> x[2:4]
[1] "b" "a" "d"
```

```
> x[c(1, 4, 3, 6, 3)]
[1] "e" "d" "a" "f" "a"
```

Vetores

- Exemplos (continuação):

```
> x <- c(3, 7, 4, 1, 6, 2, 1); x
[1] 3 7 4 1 6 2 1
```

```
> x[-2]
[1] 3 4 1 6 2 1
```

```
> x[c(-3, -1, -6)]
[1] 7 1 6 1
```

```
> x[-3:-5]
[1] 3 7 2 1
```

Vetores

- Exemplos (continuação):

```
> x
[1] 3 7 4 1 6 2 1
```

```
> x > 2
[1] TRUE TRUE TRUE FALSE TRUE FALSE FALSE
```

```
> x[x > 2]
[1] 3 7 4 6
```

```
> y <- (x < 5)
> x[y]
[1] 3 4 1 2 1
```

Vetores

- Exemplos (continuação):

```
> x
[1] 3 7 4 1 6 2 1
```

```
> r <- x %% 2; r
[1] 1 1 0 1 0 0 1
```

```
> p <- (r == 0); p
[1] FALSE FALSE TRUE FALSE TRUE TRUE FALSE
```

```
> pares <- x[p]; pares
[1] 4 6 2
```

Vetores

- Exemplos (continuação):

```
> qtd <- c(20, 25, 42, 18, 12)
> names(qtd) <- c("PP", "P", "M", "G", "GG")
```

```
> qtd
PP  P  M  G GG
20 25 42 18 12
```

```
> qtd["G"]
 G
18
```

```
> qtd[c("G", "M", "P")]
 G  M  P
18 42 25
```

Fatores

- Exemplos:

```
> x <- factor(c("M", "F", "M", "M")); x
[1] M F M M
Levels: F M
```

```
> x[2]
[1] F
Levels: F M
```

```
> x[c(2, 4)]
[1] F M
Levels: F M
```

Fatores

- Exemplos (continuação):

```
> names(x) <- c("João", "Taís", "Luís", "José")
> x
João Taís Luís José
   M    F    M    M
Levels: F M
```

```
> x[c("Luís", "Taís")]
Luís Taís
   M    F
Levels: F M
```

Matrizes

- Exemplos:

```
> m <- matrix(nrow = 3, ncol = 4, data = 1:12)
> m
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

```
> m[2, 2]
[1] 5
```

```
> m[1, 3]
[1] 7
```

Matrizes

- Exemplos (continuação):

```
> m
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

```
> m[2,  ]
[1]  2  5  8 11
```

```
> m[ , 3]
[1] 7 8 9
```

Matrizes

- Exemplos (continuação):

```
> m
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

```
> m[ , 3, drop = FALSE]
     [,1]
[1,]    7
[2,]    8
[3,]    9
```

Matrizes

- Exemplos (continuação):

```
> m
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

```
> m[2:3, 2:4]
     [,1] [,2] [,3]
[1,]    5    8   11
[2,]    6    9   12
```

Matrizes

- Exemplos (continuação):

```
> m
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

```
> m[c(1,3), c(3, 2, 4)]
     [,1] [,2] [,3]
[1,]    7    4   10
[2,]    9    6   12
```

Matrizes

- Exemplos (continuação):

```
> dimnames(m) <- list(c("SP", "RJ", "MG"),
+                     c("A", "B", "C", "D"))
> m
    A  B  C  D
SP  1  4  7 10
RJ  2  5  8 11
MG  3  6  9 12
```

```
> m["RJ", c("A", "C")]
[1] 2 8
```

Listas

- Exemplos:

```
> x <- list(foo = 1:4, bar = 0.6,
+           msg = c("hello", "world"),
+           quiz = list(5, 7, 3))
```

```
> x[1]
$foo
[1] 1 2 3 4
```

```
> x[[1]]
[1] 1 2 3 4
```

Listas

- Exemplos (continuação):

```
> x <- list(foo = 1:4, bar = 0.6,
+           msg = c("hello", "world"),
+           quiz = list(5, 7, 3))
```

```
> x$bar
[1] 0.6
```

```
> x["bar"]
$bar
[1] 0.6
```

```
> x[["bar"]]
[1] 0.6
```

Listas

- Exemplos (continuação):

```
> x <- list(foo = 1:4, bar = 0.6,
+           msg = c("hello", "world"),
+           quiz = list(5, 7, 3))
```

```
> x[c(1, 3)]
$foo
[1] 1 2 3 4

$msg
[1] "hello" "world"
```

Listas

- Exemplos (continuação):

```
> x <- list(foo = 1:4, bar = 0.6,
+           msg = c("hello", "world"),
+           quiz = list(5, 7, 3))
```

```
> x[[4]][[2]]
[1] 7
```

```
> x[[c(4, 2)]]
[1] 7
```

```
> x[[c(3, 2)]]
[1] "world"
```

Listas

- Exemplos (continuação):

```
> x <- list(foo = 1:4, bar = 0.6,
+           msg = c("hello", "world"),
+           quiz = list(5, 7, 3))
```

```
> name <- "msg"
> x[[name]]
[1] "hello" "world"
```

```
> x$name
NULL
```

```
> x$msg
[1] "hello" "world"
```

Listas

- Exemplos (continuação):

```
> x <- list(foo = 1:4, bar = 0.6,
+           msg = c("hello", "world"),
+           quiz = list(5, 7, 3))
```

```
> x$m
[1] "hello" "world"
```

```
> x[["m"]]
NULL
```

```
> x[["m", exact = FALSE]]
[1] "hello" "world"
```

Data Frames

- Exemplos:

```
> d <- data.frame(foo = 6:8, bar = c(T, F, T))
> d
  foo   bar
1   6  TRUE
2   7 FALSE
3   8  TRUE
```

```
> d$fo
[1] 6 7 8
```

```
> d[["ba"]]
NULL
```

Data Frames

- Exemplos (continuação):

```
> d
  foo   bar
1   6  TRUE
2   7 FALSE
3   8  TRUE
```

```
> d[2]
    bar
1  TRUE
2 FALSE
3  TRUE
```

```
> d[["bar"]]
[1]  TRUE FALSE  TRUE
```

Data Frames

- Exemplos (continuação):

```
> d
  foo   bar
1   6  TRUE
2   7 FALSE
3   8  TRUE
```

```
> names(d)
[1] "foo" "bar"
```

```
> d$quiz <- c("cat", "dog", "rat"); d
  foo   bar quiz
1   6  TRUE  cat
2   7 FALSE  dog
3   8  TRUE  rat
```

Data Frames

- Exemplos (continuação):

```
> d
  foo   bar quiz
1   6  TRUE  cat
2   7 FALSE  dog
3   8  TRUE  rat
```

```
> names(d)
[1] "foo" "bar" "quiz"
```

```
> d[ , 1:2] <- d[ , 2:1]; d
    foo bar quiz
1  TRUE   6  cat
2 FALSE   7  dog
3  TRUE   8  rat
```

Data Frames

- Exemplos (continuação):

```
> d
    foo  bar quiz
1  TRUE    6  cat
2 FALSE    7  dog
3  TRUE    8  rat
```

```
> d$bar <- NULL
> d
    foo quiz
1  TRUE  cat
2 FALSE  dog
3  TRUE  rat
```

Data Frames

- Exemplos (continuação):

```
> d
    foo quiz
1  TRUE  cat
2 FALSE  dog
3  TRUE  rat
```

```
> d[ , c("foo", "quiz")] <-
+       d[ , c("quiz", "foo")]; d
   foo  quiz
1 cat  TRUE
2 dog FALSE
3 rat  TRUE
```

Data Frames

- Exemplos (continuação):

```
> d
  foo  quiz
1 cat  TRUE
2 dog FALSE
3 rat  TRUE
```

```
> names(d)[1] <- "tmp"
> d
  tmp  quiz
1 cat  TRUE
2 dog FALSE
3 rat  TRUE
```

Data Frames

- Exemplos (continuação):

```
> d
  tmp  quiz
1 cat  TRUE
2 dog FALSE
3 rat  TRUE
```

```
> names(d)[1:2] <- names(d)[2:1]
> d
  quiz   tmp
1  cat  TRUE
2  dog FALSE
3  rat  TRUE
```

Data e Hora

Datas e Horários

- Datas e horários são representados por classes especiais em R.
- Datas são representadas pela classe `Date`.
- Horários são representados pelas classes `POSIXct` e `POSIXlt`.
- Datas são representadas internamente pelo número de dias desde 01/01/1970.
- Horários são representados internamente pelo número de segundos desde 01/01/1970.

Datas

- Exemplos:

```
> nascimento <- as.Date("1975-09-19")
> nascimento
[1] "1975-09-19"
```

```
> hoje <- Sys.Date(); hoje
[1] "2020-02-15"
```

```
> hoje - nascimento
Time difference of 16220 days
```

```
> hoje - as.Date("1970/01/01")
Time difference of 18307 days
```

```
> unclass(hoje)
[1] 18307
```

Horários

- A classe `POSIXct` é de fato apenas um número, útil para armazenar horários de forma compacta (por exemplo, em data frames).
- A classe `POSIXlt` é de fato uma lista, que armazena várias informações úteis como dia da semana, dia do ano, mês, etc.
- Existem algumas funções genéricas que funcionam tanto com datas, quanto com horários:
  - `weekdays()`: retorna o dia da semana.
  - `months()`: o retorna o mês do ano.
  - `quarters()`: retorna o trimestre ("Q1", "Q2","Q3" ou "Q4").

Horários

- Exemplos:

```
> agora <- Sys.time(); agora
[1] "2020-02-15 12:27:00 -02"
```

```
> unclass(agora)
[1] 1581776821
```

```
> agora + 3600
[1] "2020-02-15 13:27:00 -02"
```

```
> weekdays(agora)
[1] "Saturday"
```

Horários

```
> agora
[1] "2020-02-15 12:27:00 -02"
```

```
> agora <- as.POSIXlt(agora)
> names(unclass(agora))
[1] "sec"  "min"  "hour"  "mday"  "mon"  "year"
[7] "wday" "yday" "isdst" "zone" "gmtoff"
```

```
> unclass(agora)
$sec
[1] 0.616215
$min
[1] 27
$hour
...
```

Horários

- Se um horário não estiver no formato padrão adotado por R ("YYYY-MM-DD HH:MM:SS TZN"), ele pode convertido usado a função `strptime()`, que retorna um objeto da classe `POSIXlt`.
- Exemplo:

```
> s1 <- "15/08/2015-17:10"
> t1 <- strptime(s1, "%d/%m/%Y-%H:%M")
> t1
[1] "2015-08-15 17:10:00 -03"
```

```
> s2 <- "09/19/1994 04:13:47 PM"
> t2 <- strptime(s2, "%m/%d/%Y %I:%M:%S %p")
> t2
[1] "1994-09-19 16:13:47 -03"
```

Operações Vetorizadas

Operações Básicas

- Os operadores aritméticos básicos em R são os seguintes:
  - `+` Soma
  - `-` Subtração
  - `*` Multiplicação
  - `/` Divisão
  - `%%` Resto
  - `%/%` Divisão inteira
  - `^` Exponenciação
  - `**` Exponenciação

Operações Básicas

- Os operadores de comparação em R são os seguintes:
  - `==` Igual
  - `!=` Diferente
  - `>` Maior
  - `>=` Maior ou igual
  - `<` Menor
  - `<=` Menor ou igual
- Os operadores lógicos em R são os seguintes:
  - `!` Negação
  - `|` Ou lógico
  - `&` E lógico
  - `||` Ou lógico ("preguiçoso")
  - `&&` E lógico ("preguiçoso")

Operações Básicas

- Em R, os operandos das operações mencionadas anteriormente são vetores, eventualmente unitários, por isso dizemos que estas operações são vetorizadas.
- Exemplos:

```
> 3 + 4
[1] 7
```

```
> a <- 1:5; a
[1] 1 2 3 4 5
```

```
> a + 10
[1] 11 12 13 14 15
```

Operações Básicas

- Exemplos (continuação):

```
> a
[1] 1 2 3 4 5
```

```
> a - 3
[1] -2 -1 0 1 2
```

```
> 3 * a
[1] 3 6 9 12 15
```

```
> a / 4
[1] 0.25 0.50 0.75 1.00 1.25
```

```
> 4 / a
[1] 4.00000 2.00000 1.33333 1.00000 0.80000
```

Operações Básicas

- Exemplos (continuação):

```
> a
[1] 1 2 3 4 5
```

```
> a ^ 2
[1] 1 4 9 16 25
```

```
> 2 ** a
[1] 2 4 8 16 32
```

```
> a %% 4
[1] 1 2 3 0 1
```

```
> a %/% 2
[1] 0 1 1 2 2
```

Operações Básicas

- Exemplos (continuação):

```
> 3 > 4
[1] FALSE
```

```
> a
[1] 1 2 3 4 5
```

```
> a > 4
[1] FALSE FALSE FALSE FALSE TRUE
```

```
> a != 3
[1] TRUE TRUE FALSE TRUE TRUE
```

```
> 1 < a
[1] FALSE TRUE TRUE TRUE TRUE
```

Operações Básicas

- Exemplos (continuação):

```
> a
[1] 1 2 3 4 5
```

```
> a == 2
[1] FALSE TRUE FALSE FALSE FALSE
```

```
> a >= 0
[1] TRUE TRUE TRUE TRUE TRUE
```

```
> (a ^ 2) > 10
[1] FALSE FALSE FALSE TRUE TRUE
```

```
> 2 ^ (a < 4)
[1] 2 2 2 1 1
```

Operações Básicas

- Exemplos (continuação):

```
> b <- c(TRUE, FALSE, FALSE, TRUE,
+         FALSE, TRUE, FALSE); b
[1] TRUE FALSE FALSE TRUE FALSE TRUE FALSE
```

```
> c <- c(FALSE, FALSE, TRUE,
+         TRUE, TRUE, FALSE, FALSE); c
[1] FALSE FALSE TRUE TRUE TRUE FALSE FALSE
```

```
> b | c
[1] TRUE FALSE TRUE TRUE TRUE TRUE FALSE
```

```
> b & c
[1] FALSE FALSE FALSE TRUE FALSE FALSE FALSE
```

Operações Básicas

- Exemplos (continuação):

```
> b
[1] TRUE FALSE FALSE TRUE FALSE TRUE FALSE
> c
[1] FALSE FALSE TRUE TRUE TRUE FALSE FALSE
```

```
> !b
[1] FALSE TRUE TRUE FALSE TRUE FALSE TRUE
```

```
> b || c
[1] TRUE
```

```
> b && c
[1] FALSE
```

Operações Básicas

- Exemplos (continuação):

```
> b
[1] TRUE FALSE FALSE TRUE FALSE TRUE FALSE
> c
[1] FALSE FALSE TRUE TRUE TRUE FALSE FALSE
```

```
> b || (teste > 0)
[1] TRUE
```

```
> b | (teste > 0)
Error: object 'teste' not found
```

- Exemplos (continuação):

```
> x <- c(TRUE, FALSE, NA)
```

```
> names(x) <- c("TRUE", "FALSE", "NA"); x
 TRUE FALSE NA
 TRUE FALSE NA
```

```
> outer(x, x, "&")
      TRUE FALSE    NA
TRUE    TRUE FALSE    NA
FALSE FALSE FALSE FALSE
NA        NA FALSE    NA
```

Operações Básicas

- Exemplos (continuação):

```
> x
 TRUE FALSE NA
 TRUE FALSE NA
```

```
> outer(x, x, "|")
      TRUE FALSE   NA
TRUE  TRUE  TRUE TRUE
FALSE TRUE FALSE   NA
NA    TRUE    NA   NA
```

```
> !x
 TRUE FALSE NA
FALSE  TRUE NA
```

- Exemplos (continuação):

```
> x
 TRUE FALSE NA
 TRUE FALSE NA
```

```
> outer(x, x, "==")
      TRUE FALSE NA
TRUE    TRUE FALSE NA
FALSE FALSE  TRUE NA
NA         NA    NA NA
```

Operações Básicas

- Exemplos (continuação):

```
> a <- c(1, 2, 3, 4)
> b <- c(5, 6, 7, 8)
> c <- c(3, 4)
```

```
> a / b
[1] 0.2000000 0.3333333 0.4285714 0.5000000
```

```
> a / c
[1] 0.3333333 0.5000000 1.0000000 1.0000000
```

Operações Básicas

- Exemplos (continuação):

```
> a <- c(1, 2, 3, 4)
> b <- c(5, 6, 7, 8)
> d <- c(2, 3, 4)
```

```
> a / d
[1] 0.5000000 0.6666667 0.7500000 2.0000000
Warning message:
In a/d : longer object length is not a multiple
    of shorter object length
```

```
> a + b
[1] 6 8 10 12
```

Operações Básicas

- Exemplos (continuação):

```
> b <- c(5, 6, 7, 8)
> c <- c(3, 4)
> d <- c(2, 3, 4)
```

```
> b * c
[1] 15 24 21 32
```

```
> (b - c) * d
[1] 4 6 16 8
Warning message:
In (b - c) * d : longer object length is not a
    multiple of shorter object length
```

Operações Básicas

- Exemplos (continuação):

```
> x <- matrix(1:4, nrow =  2, ncol = 2); x
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> y <- matrix(10:7, nrow =  2, ncol = 2); y
     [,1] [,2]
[1,]   10    8
[2,]    9    7
```

```
> y - x
     [,1] [,2]
[1,]    9    5
[2,]    7    3
```

Operações Básicas

- Exemplos (continuação):

```
> x
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> y
     [,1] [,2]
[1,]   10    8
[2,]    9    7
```

```
> x * y
     [,1] [,2]
[1,]   10   24
[2,]   18   28
```

Operações Básicas

- Exemplos (continuação):

```
> x
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> y
     [,1] [,2]
[1,]   10    8
[2,]    9    7
```

```
> x %*% y          # multiplicacao de matrizes
     [,1] [,2]
[1,]   37   29
[2,]   56   44
```