

INF 1514

Introdução à Análise de Dados

Material 6



Valores reservados do R

- Existem valores reservados para representar dados faltantes, infinitos e indefinições matemáticas.
- **NA** (Not Available) significa **dado faltante/indisponível**. É o **null** de outras linguagens. O **NA** tem uma classe, ou seja, podemos ter **NA** numeric, **NA** character, etc.
- **NaN** (Not a Number) representa **indefinições matemáticas**, como $0/0$ e $\log(-1)$. Um **NaN** é um **NA**, mas a recíproca não é verdadeira.
- **Inf** (Infinito) é um **número muito grande** ou o **limite matemático**, por exemplo, $1/0$ e 10^{310} . Aceita sinal negativo **-Inf**.
- **NULL** representa a ausência de informação. Conceitualmente, a diferença entre **NA** e **NULL** é sutil, mas, no R, o **NA** está mais alinhado com os conceitos de estatística (ou como gostaríamos que os dados faltantes se comportassem em análise de dados), e o **NULL** está em sintonia com comportamentos de lógica de programação.
- Use as funções **is.na()**, **is.nan()**, **is.infinite()** e **is.null()** para testar se um objeto possui algum desses valores.

Classes básicas de objetos no R

- Existem cinco classes básicas de objetos no R:
 - character (“aula”)
 - numeric (1.)
 - integer (3)
 - logical (TRUE or FALSE)
 - complex : números complexos no formato $a + bi$
- Os objetos pertencentes a estas classes são chamados de **objetos atômicos**.
- A função interna **class()** é utilizada para se identificar a classe de um objeto.
- No R encontramos outras classes de objetos que podem ser agrupadas em dois grupos:
 - Contém apenas objetos de uma única classe básica de objetos: vector, matrix, array, factor, ts.
 - Podem conter objetos de todos os tipos de classe: data.frame, list.

Vetores

- Vetores no R são os objetos **mais simples** que podem armazenar **objetos atômicos**.
- Um vetor **tem** sempre a **mesma classe dos objetos** que **armazena**.
- A função **length()** retorna o **número** de elementos do vetor.

```
> sequencia <- 2:5
> sequencia
[1] 2 3 4 5
> length(sequencia)
[1] 4
> class(sequencia)
[1] "integer"
>
> cor <- c("verde", "azul")
> cor
[1] "verde" "azul"
> length(cor)
[1] 2
> class(cor)
[1] "character"
```

Criação de vetores

Instrução	Descrição	Exemplo
:	Cria sequências numéricas.	<code>v <- 1:5</code> <code>v <- -3:3</code>
scan	Lê valores digitados.	<code>v <- scan()</code>
c	Combina valores.	<code>v <- c (3, 4, 8, 9)</code> <code>v <- c ("verde", "amarelo")</code>
rep	Repete valores.	<code>v <- rep (c (3, 4), 2)</code>
seq	Cria sequências numéricas.	<code>v <- seq (-1, 1, 0.5)</code>

Instrução :

- A instrução : cria sequências numéricas, podendo ter prioridade sobre outras instruções.

```
> v <- 1:5
```

```
> v
```

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

```
>
```

```
> v <- -3:3
```

```
> v
```

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

```
>
```

```
> v <- 2*(1:5)
```

```
> v
```

```
[1] 2 4 6 8 10
```

```
>
```

```
> v <- rep(c(3, 4), 2)
```

```
> v
```

```
[1] 3 4 3 4
```

Instrução **scan**

- A instrução **scan** lê valores digitados.

```
> v <- scan()
1: 2.3 4.7 6.8 9.1
5:
Read 4 items
> v
[1] 2.3 4.7 6.8 9.1
>
> v <- scan(what = " ", sep = ",")
1: verde,amarelo,azul turquesa
4:
Read 3 items
> v
[1] "verde"          "amarelo"         "azul turquesa"
```

Instrução **c**

- A instrução **c** combina valores.

```
> v <- c(3, 4, 8, 9)
```

```
> v
```

```
[1] 3 4 8 9
```

```
>
```

```
> v <- c("verde", "azul")
```

```
> v
```

```
[1] "verde" "azul"
```


Instrução **rep**

- A instrução **rep** retorna o primeiro argumento, repetido o número de vezes indicado pelo segundo argumento.

```
> v <- rep(2, 5)
> v
[1] 2 2 2 2 2
>
> v <- rep(c(3, 4), 2)
> v
[1] 3 4 3 4
>
> v <- rep("A", 3)
> v
[1] "A" "A" "A"
```

Instrução **seq**

- A instrução **seq** cria sequências numéricas tendo como argumentos o início, o fim e os passos da sequência.

```
> v <- seq(-1, 1, 0.5)
> v
[1] -1.0 -0.5  0.0  0.5  1.0
>
> v <- seq(-1, 1, 0.3)
> v
[1] -1.0 -0.7 -0.4 -0.1  0.2  0.5  0.8
>
> v <- seq(-1, 1, length = 6)
> v
[1] -1.0 -0.6 -0.2  0.2  0.6  1.0
>
> v <- seq(-1, by = 0.4, length = 8)
> v
[1] -1.0 -0.6 -0.2  0.2  0.6  1.0  1.4  1.8
```

Concatenação de vetores

- Podemos concatenar vetores de uma mesma classe com a instrução `c`.

```
> v1 <- 1:5
> v1
[1] 1 2 3 4 5
> class(v1)
[1] "integer"
>
> v2 <- c(6L, 7L, 8L)
> v2
[1] 6 7 8
> class(v2)
[1] "integer"
>
> v <- c(v1, v2)
> v
[1] 1 2 3 4 5 6 7 8
> class(v)
[1] "integer"
```

A concatenação também funcionaria se `v2 <- c(6, 7, 8)`. Neste caso, os vetores `v1` e `v` seriam da classe "numeric".

```
> v1 <- c("verde", "azul")
> v1
[1] "verde" "azul"
> class(v1)
[1] "character"
>
> v2 <- c("amarelo", "branco")
> v2
[1] "amarelo" "branco"
> class(v2)
[1] "character"
>
> v <- c(v1, v2)
> v
[1] "verde"      "azul"      "amarelo" "branco"
> class(v)
[1] "character"
```

Operações aritméticas com vetores

- De forma bastante intuitiva, você pode fazer **operações** com vetores.

```
> v <- 1:5
> v
[1] 1 2 3 4 5
> v <- v - 1
> v
[1] 0 1 2 3 4
```

```
> v <- v * 2
> v
[1] 0 2 4 6 8
```

- Caso os vetores tenham a **mesma dimensão**, as operações são feitas elemento a elemento. Vamos calcular o índice de massa corporal de pessoas a partir dos vetores peso e altura. ($\text{imc} = \text{peso} / ((\text{altura})^2)$)

```
> peso <- c(57, 63, 68, 78, 96)
> altura <- c(1.65, 1.90, 1.85, 1.67, 1.73)
> imc <- peso / (altura^2)
> imc
[1] 20.93664 17.45152 19.86852 27.96802 32.07591
```

Outras funções aplicadas a vetores

```
> altura <- c(1.65, 1.90, 1.85, 1.67, 1.73)
> min(altura)
[1] 1.65
> max(altura)
[1] 1.9
> range(altura)
[1] 1.65 1.90
> mean(altura)
[1] 1.76
> var(altura)
[1] 0.0122
> sum(altura)
[1] 8.8
> length(altura)
[1] 5
> sum(altura) / length(altura)
[1] 1.76
```

Instruções comparativas

Instrução	Descrição	Exemplo
<	Menor que.	<code>v <- 1:5</code> <code>v < 2</code>
>	Maior que.	<code>v <- 1:5</code> <code>v > 3</code>
<=	Menor ou igual a.	<code>v <- c(3, 4, 8, 9)</code> <code>v <= 4</code>
>=	Maior ou igual a.	<code>v <- rep(c(3, 4), 2)</code> <code>v >= 3</code>
==	Igual a.	<code>v <- seq(-1, 1, 0.5)</code> <code>v == 1</code>
!=	Diferente de.	<code>v <- seq(-1, 1, 0.5)</code> <code>v != 1</code>

Instruções comparativas

- As instruções comparativas verificam elemento por elemento do vetor.

```
> v <- 1:5
> v
[1] 1 2 3 4 5
> comparacao <- v < 2
> comparacao
[1] TRUE FALSE FALSE FALSE FALSE
> comparacao <- v > 3
> comparacao
[1] FALSE FALSE FALSE TRUE TRUE
> comparacao <- v <= 4
> comparacao
[1] TRUE TRUE TRUE TRUE FALSE
> comparacao <- v == 1
> comparacao
[1] TRUE FALSE FALSE FALSE FALSE
> comparacao <- v != 1
> comparacao
[1] FALSE TRUE TRUE TRUE TRUE
```

Instruções lógicas

Instrução	Descrição	Exemplo
&	E.	<code>v <- 1:5</code> <code>(v > 2) & (v < 4)</code>
	Ou.	<code>v <- c(3, 4, 8, 9)</code> <code>(v <= 4) (v == 9)</code>
xor	Ou exclusivo.	<code>v <- seq(-1, 1, 0.5)</code> <code>xor((v == 1), (v > 0))</code>

Instruções lógicas

- As instruções lógicas verificam elemento por elemento do vetor.

```
> v <- 1:5
> v
[1] 1 2 3 4 5
> comparacao <- ((v > 2) & (v < 4))
> comparacao
[1] FALSE FALSE TRUE FALSE FALSE
> v <- c(3, 4, 8, 9)
> v
[1] 3 4 8 9
> comparacao <- ((v <= 4) | (v == 5))
> comparacao
[1] TRUE TRUE FALSE FALSE
> v <- seq(-1, 1, 0.5)
> v
[1] -1.0 -0.5 0.0 0.5 1.0
> xor((v == 1), (v > 0))
[1] FALSE FALSE FALSE TRUE FALSE
```

x	y	&		xor(x,y)
FALSE	FALSE	FALSE	FALSE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE
FALSE	TRUE	FALSE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	FALSE

Subconjuntos de vetores

- Podemos montar **subconjuntos** de um vetor usando: os índices do vetor (número inteiro que indica a posição do elemento no vetor); funções comparativas e lógicas; etc..

```
> v <- c(2, 5, 7, 8, 9, 10, 12, 3, 7, 9)
> v
[1] 2 5 7 8 9 10 12 3 7 9
> a <- v[6]
> a
[1] 10
> b <- v[3:5]
> b
[1] 7 8 9
> c <- v[c(2, 4, 10)]
> c
[1] 5 8 9
```

```
> d <- v[v > 6]
> d
[1] 7 8 9 10 12 7 9
> e <- v[(v > 6) & (v <= 9)]
> e
[1] 7 8 9 7 9
```

Classes diferentes em um mesmo vetor

- Se colocarmos duas ou mais **classes diferentes** dentro de um **mesmo vetor**, o R vai **forçar** que todos os elementos **passem** a pertencer à **mesma classe**.
- A ordem de **precedência** é: character > complex > numeric > integer > logical.

```
> v <- c(0, 1.8, "verde")
> v
[1] "0"      "1.8"    "verde"
> v <- c(TRUE, FALSE, 2)
> v
[1] 1 0 2
> v <- c(TRUE, FALSE, "verde")
> v
[1] "TRUE"   "FALSE"  "verde"
```

Classes diferentes em um mesmo vetor

- Pode-se **coagir** um objeto a ser de uma **classe específica** com as funções **as.character()**, **as.numeric()**, **as.integer()** e **as.logical()**. Isso equivale a algumas funções de conversão de tipo de dados presente em outras linguagens.

```
> v <- 0:5
> v
[1] 0 1 2 3 4 5
> class(v)
[1] "integer"
>
> a <- as.numeric(v)
> a
[1] 0 1 2 3 4 5
> class(a)
[1] "numeric"
```

```
> b <- as.logical(v)
> b
[1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
> class(b)
[1] "logical"
>
> c <- as.character(v)
> c
[1] "0" "1" "2" "3" "4" "5"
> class(c)
[1] "character"
```

Classes diferentes em um mesmo vetor

- Se o R não entender como **coagir** uma **classe na outra**, ele levantará um ***warning*** informando que colocou **NA** no lugar.

```
> v <- c(0, 0.5, 1, 2, FALSE, TRUE)
> v
[1] 0.0 0.5 1.0 2.0 0.0 1.0
> a <- as.logical(v)
> a
[1] FALSE TRUE TRUE TRUE FALSE TRUE
>
```

```
> v <- c(1, 3.2, TRUE, "c")
> v
[1] "1" "3.2" "TRUE" "c"
> b <- as.numeric(v)
Warning message:
NAs introduced by coercion
> b
[1] 1.0 3.2 NA NA
>
```

Exercício 1

- Defina um vetor “v” com a sequência de números ímpares maiores que zero e menores que 50. A seguir:
 - apresente um vetor “v.produto” que contém o produto por elemento do vetor “v” por 2.
 - apresente um vetor “v.produto.filtrado” com somente os elementos do vetor “v.produto” com valor maior ou igual a 30.
 - armazene em um objeto “v.produto.filtrado.soma” a soma dos elementos do vetor “v.produto.filtrado”.

Exercício 1 – Resolução

```
> v <- seq(1, 49, 2)
> v
[1] 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
> v.produto <- v * 2
> v.produto
[1] 2 6 10 14 18 22 26 30 34 38 42 46 50 54 58 62 66 70 74 78 82 86 90 94 98
> v.produto.filtrado <- v.produto[(v.produto >= 30)]
> v.produto.filtrado
[1] 30 34 38 42 46 50 54 58 62 66 70 74 78 82 86 90 94 98
> v.produto.filtrado.soma <- sum(v.produto.filtrado)
> v.produto.filtrado.soma
[1] 1152
```

Matriz

- Matrizes são **vetores** com **duas dimensões**.
- Somente possuem elementos de uma **mesma classe**.
- As matrizes apresentam **quatro atributos**: mode, length, dim e dimnames.
 - **mode** informa a **natureza** dos seus elementos: logical, numeric, complex e character.
 - **length** informa o **número** de elementos da matriz.
 - **dim** informa o número de linhas e colunas.
 - **dimnames** informa os nomes das linhas e colunas.

```
> v <- 1:12
> mat <- matrix(v, ncol = 3)
> mat
```

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

```
> mode(mat)
[1] "numeric"
> class(mat)
[1] "matrix" "array"
> length(mat)
[1] 12
```

```
> dim(mat)
[1] 4 3
> dimnames(mat)
NULL
```


Criação de matrizes

Instrução	Descrição	Exemplo
:	Cria sequências numéricas.	<pre>v <- 1:12 mat <- matrix (v, ncol = 3)</pre>
scan	Lê valores digitados.	<pre>v <- scan() mat <- matrix (v, ncol = 2)</pre>
c	Combina valores.	<pre>v <- c (3, 4, 8, 9) mat <- matrix (v, nrow = 2)</pre>
rep	Repete valores.	<pre>v <- rep (c (3, 4, 5), 2) mat <- matrix (v, nrow = 3)</pre>
seq	Cria sequências numéricas.	<pre>v <- seq (-1, 1.5, 0.5) mat <- matrix (v, ncol = 3)</pre>

Instrução :

- A instrução : cria sequências numéricas.

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

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

A matriz terá 3
colunas.

Por padrão a matriz é
preenchida por
colunas.

byrow = TRUE (por linhas) informa que a
matriz deve ser preenchida por linhas.

```
> v <- 1:12
> v
[1] 1 2 3 4 5 6 7 8 9 10 11 12
> mat <- matrix(v, ncol = 3, byrow = TRUE)
> mat
```

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

Instrução scan

- A instrução **scan** lê valores digitados.

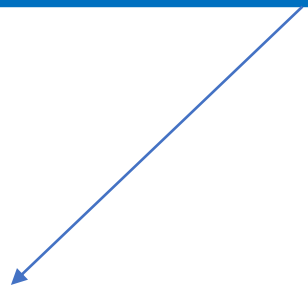
```
> v <- scan()  
1: 1 2 3 4 5 6  
7:
```

Read 6 items

```
> mat <- matrix(v, ncol = 2)  
> mat
```

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

“what” informa o tipo de dado a ser lido (logical, integer, numeric, complex, character).



```
> v <- scan(what = character(), sep = ",")  
1: Gol,Civic,Kicks,Prata,Cinza,Branco  
7:
```

Read 6 items

```
> v  
[1] "Gol"      "Civic"    "Kicks"    "Prata"  
"Cinza"    "Branco"  
> mat <- matrix(v, ncol = 2)  
> mat
```

	[,1]	[,2]
[1,]	"Gol"	"Prata"
[2,]	"Civic"	"Cinza"
[3,]	"Kicks"	"Branco"

Instrução c

- A instrução **c** combina valores.

```
> v <- c(3, 4, 8, 9)
```

```
> v
```

```
[1] 3 4 8 9
```

```
> mat <- matrix(v, nrow = 2)
```

```
> mat
```

```
      [,1] [,2]  
[1,]    3    8  
[2,]    4    9
```

A matriz terá 2 linhas.

O vetor possui 8 elementos, mas a matriz somente 6. Um *warning* é levantado pelo R informando possível problema.

```
> v <- c(3, 4, 8, 9, 11, 15, 18, 33)
```

```
> v
```

```
[1] 3 4 8 9 11 15 18 33
```

```
> mat <- matrix(v, nrow = 2, ncol = 3)
```

Warning message:

In matrix(v, nrow = 2, ncol = 3) :

data length [8] is not a sub-multiple or multiple of the number of columns [3]

```
> mat
```

```
      [,1] [,2] [,3]  
[1,]    3    8   11  
[2,]    4    9   15
```


Instrução **rep**

- A instrução **rep** retorna o primeiro argumento, repetido o número de vezes indicado pelo segundo argumento.

```
> v <- rep(c(3, 4, 5), 2)
> v
[1] 3 4 5 3 4 5
> mat <- matrix(v, nrow = 3)
> mat
```

	[,1]	[,2]
[1,]	3	3
[2,]	4	4
[3,]	5	5

A função **summary()** opera em cada coluna da matriz como se fossem vetores apresentando um resumo de cada uma.



```
> summary(mat)
```

	V1	V2
Min.	:3.0	Min. :3.0
1st Qu.:	3.5	1st Qu.:3.5
Median	:4.0	Median :4.0
Mean	:4.0	Mean :4.0
3rd Qu.:	4.5	3rd Qu.:4.5
Max.	:5.0	Max. :5.0

Instrução **seq**

- A instrução **seq** cria sequências numéricas tendo como argumentos o início, o fim e os passos da seqüência.

```
> v <- seq(-1, 1.5, 0.5)
> v
[1] -1.0 -0.5  0.0  0.5  1.0  1.5
> mat <- matrix(v, ncol = 3)
> mat
      [,1] [,2] [,3]
[1,] -1.0  0.0  1.0
[2,] -0.5  0.5  1.5
```

Obtendo informações da
matriz e de seus elementos.



```
> dim(mat)
[1] 2 3
> length(mat)
[1] 6
> min(mat)
[1] -1
> max(mat)
[1] 1.5
> mean(mat)
[1] 0.25
> sd(mat)
[1] 0.9354143
```

Adicionando linhas e colunas

A função **rbind()** acrescentou uma nova linha na matriz formada pelos números 18, 20 e 35.

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

```
> mais.linha <- rbind(mat, c(18, 20, 35))
```

```
> mais.linha
```

```
      [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
[5,]   18   20   35
```

A função **cbind()** acrescentou uma nova coluna na matriz formada por uma sequência de 1 a 4.

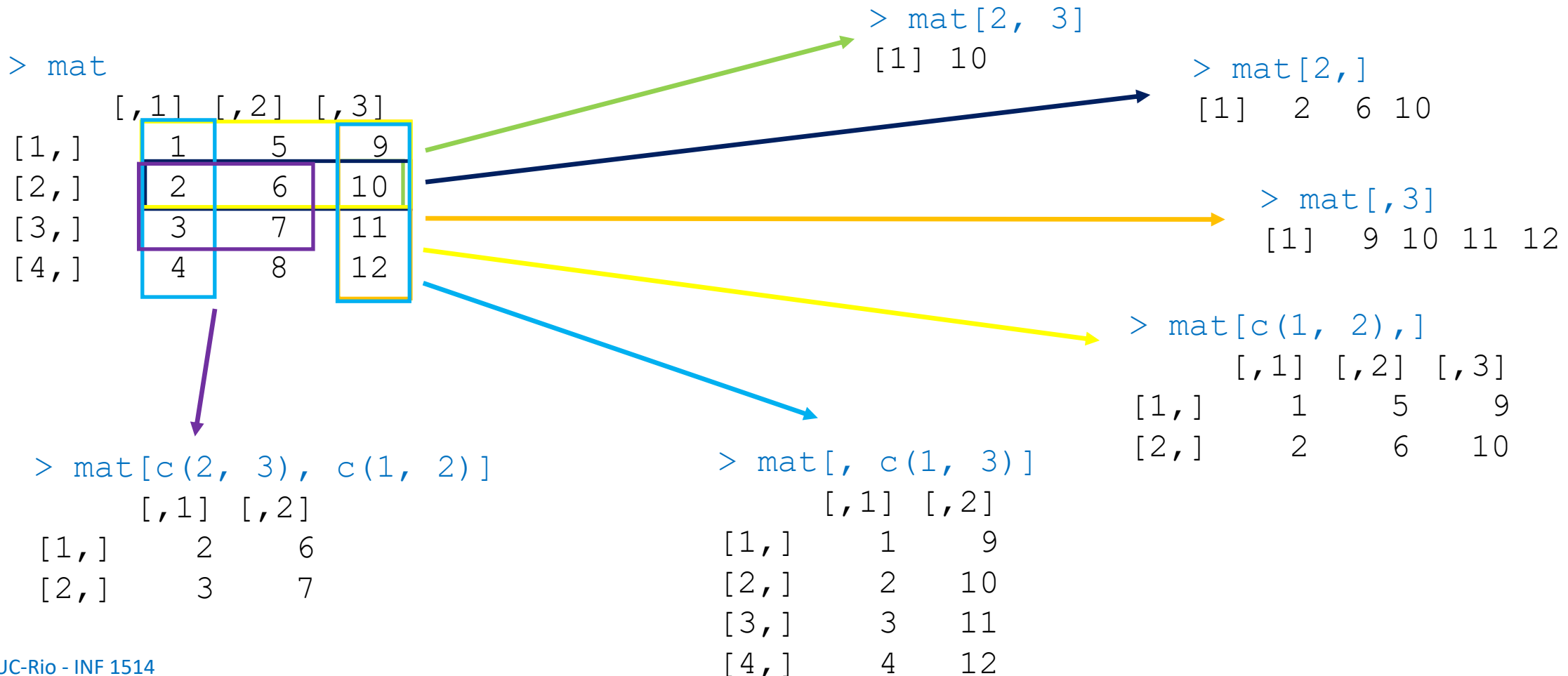
```
> mais.coluna <- cbind(mat, 1:4)
```

```
> mais.coluna
```

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

Índices de matrizes

- Como nos vetores, os **colchetes [linha, coluna]** podem ser utilizados para extrair partes de uma matriz.



Instruções comparativas

Instrução	Descrição	Exemplo
<	Menor que.	mat <- matrix (1:12, ncol = 3) mat < 5
>	Maior que.	mat <- matrix (1:12, ncol = 3) mat [, 2] > 3
<=	Menor ou igual a.	mat <- matrix (1:12, ncol = 3) mat [1, 3] <= 4
>=	Maior ou igual a.	mat <- matrix (1:12, ncol = 3) mat [1,] >= 3
==	Igual a.	mat <- matrix (1:12, ncol = 3) mat == 1
!=	Diferente de.	mat <- matrix (1:12, ncol = 3) mat != 1

Instruções comparativas

- A instruções comparativas verificam elemento por elemento da matriz.

```
> mat <- matrix(1:12, ncol = 3)
```

```
> mat
```

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

```
> ma <- mat < 5
```

```
> ma
```

	[,1]	[,2]	[,3]
[1,]	TRUE	FALSE	FALSE
[2,]	TRUE	FALSE	FALSE
[3,]	TRUE	FALSE	FALSE
[4,]	TRUE	FALSE	FALSE

```
> mb <- mat[, 2] > 3
```

```
> mb
```

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

```
> mc <- mat[1, 3] <= 4
```

```
> mc
```

```
[1] FALSE
```

```
> md <- mat[1, ] >= 3
```

```
> md
```

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

```
> me <- mat == 1
```

```
> me
```

	[,1]	[,2]	[,3]
[1,]	TRUE	FALSE	FALSE
[2,]	FALSE	FALSE	FALSE
[3,]	FALSE	FALSE	FALSE
[4,]	FALSE	FALSE	FALSE

Instruções lógicas

Instrução	Descrição	Exemplo
&	E.	<pre>mat <- matrix (1:12, ncol = 3) (mat > 2) & (mat < 4) (mat [, 3] <= 10) & (mat [, 2] == 9)</pre>
	Ou.	<pre>mat <- matrix (1:12, ncol = 3) (mat [, 2] <= 6) (mat [, 2] == 9)</pre>
xor	Ou exclusivo.	<pre>mat <- matrix (1:12, ncol = 3) xor((mat == 1), (mat > 0))</pre>

Instruções lógicas

- As instruções lógicas verificam elemento por elemento da matriz.

```
> mat <- matrix(1:12, ncol = 3)
```

```
> mat
```

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

```
> ma <- (mat > 2)
```

```
> ma
```

	[,1]	[,2]	[,3]
[1,]	FALSE	TRUE	TRUE
[2,]	FALSE	TRUE	TRUE
[3,]	TRUE	TRUE	TRUE
[4,]	TRUE	TRUE	TRUE

```
> mb <- (mat > 2) & (mat < 4)
```

```
> mb
```

	[,1]	[,2]	[,3]
[1,]	FALSE	FALSE	FALSE
[2,]	FALSE	FALSE	FALSE
[3,]	TRUE	FALSE	FALSE
[4,]	FALSE	FALSE	FALSE

```
> mc <- (mat[, 2] <= 6) | (mat[, 2] == 9)
```

```
> mc
```

[1]	TRUE	TRUE	FALSE	FALSE
-----	------	------	-------	-------

```
> md <- (mat[, 3] <= 11) & (mat[, 3] > 9)
```

```
> md
```

[1]	FALSE	TRUE	TRUE	FALSE
-----	-------	------	------	-------

Outras operações com matrizes

```
> mat1 <- matrix(c(1,1,1,0,3,2,0,1,0), ncol = 3)
```

```
> mat1
```

	[,1]	[,2]	[,3]
[1,]	1	0	0
[2,]	1	3	1
[3,]	1	2	0

```
> transposta <- t(mat1)
```

```
> transposta
```

	[,1]	[,2]	[,3]
[1,]	1	1	1
[2,]	0	3	2
[3,]	0	1	0

```
> inversa <- solve(mat1)
```

```
> inversa
```

	[,1]	[,2]	[,3]
[1,]	1.0	0	0.0
[2,]	-0.5	0	0.5
[3,]	0.5	1	-1.5

As operações matemáticas são aplicadas a todos os elementos da matriz.

Matriz transposta - função `t()` - e matriz inversa – função `solve()`.

Produto de matrizes (`%*%`).

```
> divisao <- mat1 / 2
```

```
> divisao
```

	[,1]	[,2]	[,3]
[1,]	0.5	0.0	0.0
[2,]	0.5	1.5	0.5
[3,]	0.5	1.0	0.0

```
> mat2 <- matrix(1:9, nrow = 3)
```

```
> mat2
```

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

```
> produto <- mat1 %*% mat2
```

```
> produto
```

	[,1]	[,2]	[,3]
[1,]	1	4	7
[2,]	10	25	40
[3,]	5	14	23

Exercício 2

- Crie um objeto da classe `matriz` chamado `matriz.normal` com 3 linhas e 5 colunas contendo uma amostra de uma distribuição normal de média 10 e desvio padrão 1,5. A seguir:
 - apresente a linha 2 e, a seguir, a coluna 3 da matriz.
 - apresente as dimensões da matriz.
 - calcule a soma dos elementos da matriz.
 - calcule a soma dos elementos da primeira linha da matriz.
 - calcule o produto da matriz por sua matriz transposta.
 - calcule a inversa da matriz resultante do item anterior.

Exercício 2 – Resolução

- Crie um objeto da classe matriz chamado `matriz.normal` com 3 linhas e 5 colunas contendo uma amostra de uma distribuição normal de média 10 e desvio padrão 1,5. A seguir:
 - apresente a linha 2 e, a seguir, a coluna 3 da matriz.
 - apresente as dimensões da matriz.
 - calcule a soma dos elementos da matriz.
 - calcule a soma dos elementos da primeira linha da matriz.
 - calcule o produto da matriz por sua matriz transposta.
 - calcule a inversa da matriz resultante do item anterior.

A função `rnorm()` gera amostras de uma distribuição normal dados a sua média e desvio padrão.

```
> matriz.normal <- matrix(rnorm(15, mean = 10, sd = 1.5), 3, 5)
```

```
> matriz.normal
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	9.430857	12.245638	10.760206	8.145814	12.21183
[2,]	11.317613	9.836326	13.359592	8.173003	11.90331
[3,]	8.840698	6.741280	9.379789	11.498002	11.20609

Exercício 2 – Resolução

- Crie um objeto da classe `matriz` chamado `matriz.normal` com 3 linhas e 5 colunas contendo uma amostra de uma distribuição normal de média 10 e desvio padrão 1,5. A seguir:
 - apresente a linha 2 e, a seguir, a coluna 3 da matriz.
 - apresente as dimensões da matriz.
 - calcule a soma dos elementos da matriz.
 - calcule a soma dos elementos da primeira linha da matriz.
 - calcule o produto da matriz por sua matriz transposta.
 - calcule a inversa da matriz resultante do item anterior.

```
> matriz.normal[2, ]  
[1] 11.317613  9.836326 13.359592  8.173003 11.903312  
> matriz.normal[, 3]  
[1] 10.760206 13.359592  9.379789  
> dim(matriz.normal)  
[1] 3 5
```


Exercício 2 – Resolução

- Crie um objeto da classe matriz chamado `matriz.normal` com 3 linhas e 5 colunas contendo uma amostra de uma distribuição normal de média 10 e desvio padrão 1,5. A seguir:
 - apresente a linha 2 e, a seguir, a coluna 3 da matriz.
 - apresente as dimensões da matriz.
 - calcule a soma dos elementos da matriz.
 - calcule a soma dos elementos da primeira linha da matriz.
 - calcule o produto da matriz por sua matriz transposta.
 - calcule a inversa da matriz resultante do item anterior.

```
> soma <- sum(matriz.normal)
> soma
[1] 155.05
> soma.primeira.linha <- sum(matriz.normal[1, ])
> soma.primeira.linha
[1] 52.79435
```

Exercício 2 – Resolução

- Crie um objeto da classe `matriz` chamado `matriz.normal` com 3 linhas e 5 colunas contendo uma amostra de uma distribuição normal de média 10 e desvio padrão 1,5. A seguir:
 - apresente a linha 2 e, a seguir, a coluna 3 da matriz.
 - apresente as dimensões da matriz.
 - calcule a soma dos elementos da matriz.
 - calcule a soma dos elementos da primeira linha da matriz.
 - calcule o produto da matriz por sua matriz transposta.
 - calcule a inversa da matriz resultante do item anterior.

```
> produto <- matriz.normal %*% t(matriz.normal)
```

```
> produto
```

```
      [,1]      [,2]      [,3]  
[1,] 570.1619 582.8759 497.3625  
[2,] 582.8759 611.8072 519.0379  
[3,] 497.3625 519.0379 469.3636
```

Exercício 2 – Resolução

- Crie um objeto da classe `matriz` chamado `matriz.normal` com 3 linhas e 5 colunas contendo uma amostra de uma distribuição normal de média 10 e desvio padrão 1,5. A seguir:
 - apresente a linha 2 e, a seguir, a coluna 3 da matriz.
 - apresente as dimensões da matriz.
 - calcule a soma dos elementos da matriz.
 - calcule a soma dos elementos da primeira linha da matriz.
 - calcule o produto da matriz por sua matriz transposta.
 - calcule a inversa da matriz resultante do item anterior.

```
> inversa <- solve(produto)
> inversa
```

	[, 1]	[, 2]	[, 3]
[1,]	0.068654513	-0.05965132	-0.006785533
[2,]	-0.059651325	0.07825739	-0.023329926
[3,]	-0.006785533	-0.02332993	0.035119862

Data frames


- Um **data frame** é constituído pela concatenação de várias **colunas** assim como as **matrizes**.
- Assim como uma **matriz**, todas as **colunas** de um **data frame** têm que ter o **mesmo número de elementos**, mas, ao contrário de uma matriz, um data frame pode conter colunas de **tipos diferentes**.
- Um **data frame** é semelhante a uma **tabela** de um banco de dados relacional ou uma **planilha** do Excel, sendo objetos muito importantes na solução de diversos tipos de problemas de análise de dados.

```
> acao <- c("RADL3", "HYPE3", "USIM5", "BRAP4", "VALE3")
> quantidade <- c(100, 20, 35, 40, 90)
> valor <- c(77.28, 27.71, 6.98, 32.03, 54.97)
> investimento <- data.frame(acao, quantidade, valor)
```

```
> investimento
  acao quantidade  valor
1 RADL3         100  77.28
2 HYPE3          20  27.71
3 USIM5          35   6.98
4 BRAP4          40  32.03
5 VALE3          90  54.97
```


Acesso aos dados

- Os elementos de um **data frame** podem ser **acessados e modificados** como em uma **matriz**.



```
> investimento
  acao quantidade valor
1 RADL3         100  77.28
2 HYPE3          20  27.71
3 USIM5          35   6.98
4 BRAP4          40  32.03
5 VALE3          90  54.97
```

```
> investimento[, 2]
[1] 100  20  35  40  90
>
> investimento[1, ]
  acao quantidade valor
1 RADL3         100  77.28
>
> investimento[2, 3]
[1] 27.71
>
> investimento[2, c("acao", "quantidade")]
  acao quantidade
2 HYPE3          20
```




```
> investimento[2, 3] <- 15.0
> investimento
  acao quantidade valor
1 RADL3         100  77.28
2 HYPE3          20  15.00
3 USIM5          35   6.98
4 BRAP4          40  32.03
5 VALE3          90  54.97
```

Acesso aos dados

- As colunas também podem ser selecionadas usando os símbolos \$ e [[]]. Ambos retornam um vetor como resultado.

```
> investimento
  acao quantidade valor
1 RADL3         100  77.28
2 HYPE3          20  27.71
3 USIM5          35   6.98
4 BRAP4          40  32.03
5 VALE3          90  54.97
```



```
> investimento$quantidade
[1] 100  20  35  40  90
```



```
> investimento[\"valor\"]
[1] 77.28 15.00  6.98 32.03 54.97
```

Quais são os tipos de objetos retornados por `investimento$valor`, `investimento[\"valor\"]` e `investimento[\"valor\"]` ?

Acesso aos dados

- Tanto o `$` quanto o `[[]]` sempre retornam um **vetor** como resultado. Se você quiser garantir que o resultado da seleção seja um **data frame**, use **drop = FALSE** ou selecione sem a vírgula.

```
> investimento
```

	acao	quantidade	valor
1	RADL3	100	77.28
2	HYPE3	20	27.71
3	USIM5	35	6.98
4	BRAP4	40	32.03
5	VALE3	90	54.97

```
> investimento[, "quantidade", drop = FALSE]
```

	quantidade
1	100
2	20
3	35
4	40
5	90

```
> investimento["valor"]
```

	valor
1	77.28
2	15.00
3	6.98
4	32.03
5	54.97

Adicionando linhas

```
> investimento
  acao quantidade valor
1 RADL3         100  77.28
2 HYPE3          20  27.71
3 USIM5          35   6.98
4 BRAP4          40  32.03
5 VALE3          90  54.97
```



```
> investimento <- rbind(investimento, data.frame(acao = "ITUB4", quantidade = 55, valor = 42.09))
> investimento
```

```
  acao quantidade valor
1 RADL3         100  77.28
2 HYPE3          20  27.71
3 USIM5          35   6.98
4 BRAP4          40  32.03
5 VALE3          90  54.97
6 ITUB4          55  42.09
```

A função **rbind()** acrescentou uma nova linha no data frame.

Removendo linhas

Remove as linhas 2 e 4 do data frame.

```
> investimento
  acao quantidade valor
1 RADL3         100 77.28
2 HYPE3          20 27.71
3 USIM5          35  6.98
4 BRAP4          40 32.03
5 VALE3          90 54.97
```

Remove as linhas com valor ≤ 7 (ou seleciona somente as linhas com valor > 7).

```
> investimento <- investimento[c(-2, -4), ]
> investimento
```

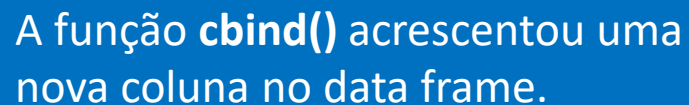
```
  acao quantidade valor
1 RADL3         100 77.28
3 USIM5          35  6.98
5 VALE3          90 54.97
```

```
> investimento <- investimento[valor > 7, ]
> investimento
```


```
  acao quantidade valor
1 RADL3         100 77.28
2 HYPE3          20 27.71
4 BRAP4          40 32.03
5 VALE3          90 54.97
```

Adicionando colunas

A função **cbind()** acrescentou uma nova coluna no data frame.



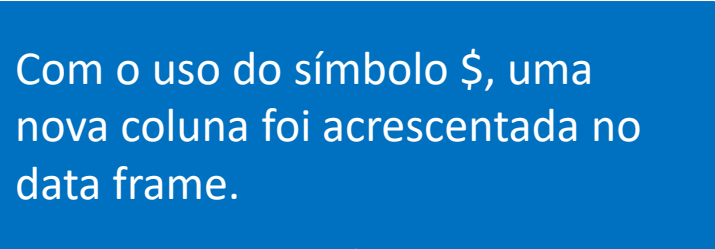
```
> investimento
  acao quantidade valor
1 RADL3         100 77.28
2 HYPE3          20 27.71
3 USIM5          35  6.98
4 BRAP4          40 32.03
5 VALE3          90 54.97
```




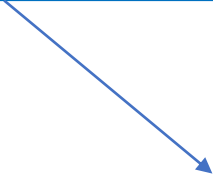
```
> investimento <- cbind(investimento, variacao = c(0.2, -0.3, 0.5, -0.8, 0.1))
> investimento
  acao quantidade valor variacao
1 RADL3         100 77.28      0.2
2 HYPE3          20 27.71     -0.3
3 USIM5          35  6.98      0.5
4 BRAP4          40 32.03     -0.8
5 VALE3          90 54.97      0.1
```

Adicionando colunas

Com o uso do símbolo \$, uma nova coluna foi acrescentada no data frame.



```
> investimento
  acao quantidade valor
1 RADL3         100 77.28
2 HYPE3          20 27.71
3 USIM5          35  6.98
4 BRAP4          40 32.03
5 VALE3          90 54.97
```



```
> investimento$variacao <- c(0.2, -0.3, 0.5, -0.8, 0.1)
> investimento
```

```
  acao quantidade valor variacao
1 RADL3         100 77.28      0.2
2 HYPE3          20 27.71     -0.3
3 USIM5          35  6.98      0.5
4 BRAP4          40 32.03     -0.8
5 VALE3          90 54.97      0.1
```

Removendo colunas

Remove a coluna valor do data frame.

```
> investimento
  acao quantidade valor
1 RADL3         100 77.28
2 HYPE3          20 27.71
3 USIM5          35  6.98
4 BRAP4          40 32.03
5 VALE3          90 54.97
```

Remove a coluna valor do data frame.

```
> investimento$valor <- NULL
```

```
> investimento
```

```
  acao quantidade
1 RADL3         100
2 HYPE3          20
3 USIM5          35
4 BRAP4          40
5 VALE3          90
```

```
> investimento <- investimento[, -3]
```

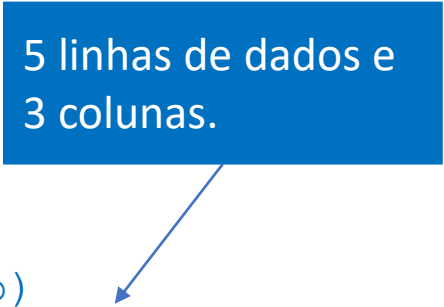
```
> investimento
```

```
  acao quantidade
1 RADL3         100
2 HYPE3          20
3 USIM5          35
4 BRAP4          40
5 VALE3          90
```

Estrutura

- A função **str** permite visualizar a **estrutura** de um **data frame**, e a função **names**, o nome das colunas.

5 linhas de dados e
3 colunas.



```
> str(investimento)
'data.frame': 5 obs. of  3 variables:
 $ acao      : chr  "RADL3" "HYPE3" "USIM5" "BRAP4" ...
 $ quantidade: num  100 20 35 40 90
 $ valor     : num  77.28 27.71 6.98 32.03 54.97
>
> names(investimento)
[1] "acao"          "quantidade" "valor"
```

Aplicando funções no data frame

- A função **sapply** aplica uma função nas **colunas** de um **data frame**.

```
> investimento
  acao quantidade valor
1 RADL3         100  77.28
2 HYPE3          20  27.71
3 USIM5          35   6.98
4 BRAP4          40  32.03
5 VALE3          90  54.97
```

```
> minimo <- sapply(investimento["quantidade"], min)
> minimo
quantidade
          20
> minimo[[1]]
[1] 20
>
> maximo <- sapply(investimento[3], max)
> maximo
valor
77.28
>
> media <- sapply(investimento[2:3], mean)
> media
quantidade      valor
      57.000      39.794
> media[[1]]
[1] 57
> media[[2]]
[1] 39.794
```

Unindo data frames

- A função **merge** permite unir **data frames**.

```
> investimento
```

	acao	quantidade	valor
1	RADL3	100	77.28
2	HYPE3	20	27.71
3	USIM5	35	6.98
4	BRAP4	40	32.03
5	VALE3	90	54.97

```
> acao <- c("ITUB4", "PTRE2")
```

```
> quantidade <- c(10, 65)
```

```
> valor <- c(22.28, 10.71)
```

```
>
```

```
> novo <- data.frame(acao, quantidade, valor)
```

```
> novo
```

	acao	quantidade	valor
1	ITUB4	10	22.28
2	PTRE2	65	10.71

```
> investimento.final <- merge(investimento, novo, all = TRUE)
```

```
> investimento.final
```

	acao	quantidade	valor
1	BRAP4	40	32.03
2	HYPE3	20	27.71
3	ITUB4	10	22.28
4	PTRE	65	10.71
5	RADL3	100	77.28
6	USIM5	35	6.98
7	VALE3	90	54.97

O parâmetro `all = TRUE` faz com que o `merge` una os dois data frames. Se existirem linhas iguais nos dois data frames, somente uma será considerada.


Se o parâmetro `all` não for declarado, somente as linhas que aparecem em ambos os data frames serão consideradas.

Unindo data frames

- A função **merge** permite unir **data frames**.

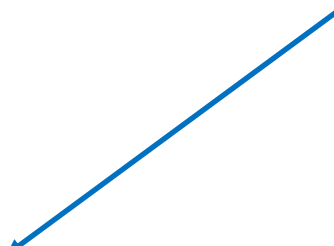
```
> codigo <- c("GFX6", "ELT6", "GLO5", "KLM1")
> compra <- c(1.01, 0.30, 1.26, 1.82)
>
> acao.compra <- data.frame(codigo, compra)
> acao.compra
```

	codigo	compra
1	GFX6	1.01
2	ELT6	0.30
3	GLO5	1.26
4	KLM1	1.82



```
> identificador <- c("ELT6", "GLO5", "GFX6", "KLM1")
> venda <- c(0.98, 0.78, 1.23, 1.95)
>
> acao.venda <- data.frame(identificador, venda)
> acao.venda
```

	identificador	venda
1	ELT6	0.98
2	GLO5	0.78
3	GFX6	1.23
4	KLM1	1.95



```
> acao <- merge(acao.compra, acao.venda, by.x = "codigo", by.y = "identificador")
> acao
```

	codigo	compra	venda
1	ELT6	0.30	0.98
2	GFX6	1.01	1.23
3	GLO5	1.26	0.78
4	KLM1	1.82	1.95

Os data frames `acao.compra` e `acao.venda` foram unidos pela coluna `codigo` e `identificador`, mas somente a coluna `codigo` foi incluída no data frame `acao`.

Os vetores `codigo` e `identificador` não precisam estar na mesma ordem.

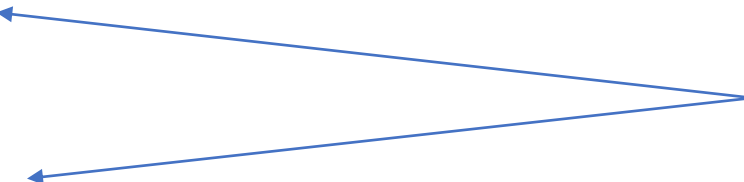
Filtrando data frames

- Mecanismo padrão.

```
> filtrado <- investimento[investimento$quantidade >= 70, ]  
> filtrado  
  acao quantidade valor  
1 RADL3         100  77.28  
5 VALE3          90  54.97
```

- A função **filter** do pacote **dplyr** que permite **selecionar linhas** específicas dos **data frames**.

```
> library(dplyr)  
> filtrado <- filter(investimento, quantidade >= 70)  
> filtrado  
  acao quantidade valor  
1 RADL3         100  77.28  
2 VALE3          90  54.97  
> filtrado <- filter(investimento, (quantidade >= 70) & (valor < 60))  
> filtrado  
  acao quantidade valor  
1 VALE3          90  54.97
```



A função **filter()** gera um data frame como retorno.

Ordenando data frames

- Mecanismo padrão com **order**.

```
> ordenado <- investimento[order (investimento$acao), ]
> ordenado
  acao quantidade valor
4 BRAP4          40 32.03
2 HYPE3          20 27.71
1 RADL3         100 77.28
3 USIM5          35  6.98
5 VALE3          90 54.97
```

Para colocar em ordem decrescente, use o argumento `decreasing = TRUE`.

- A função **arrange** do pacote **dplyr** permite **ordenar os data frames**.

```
> library(dplyr)
> ordenado <- arrange(investimento, quantidade)
> ordenado
  acao quantidade valor
1 HYPE3          20 27.71
2 USIM5          35  6.98
3 BRAP4          40 32.03
4 VALE3          90 54.97
5 RADL3         100 77.28
```

Para ordenar por quantidade e valor, use: `arrange(investimento, quantidade, valor)`.

Para colocar em ordem decrescente, use: `arrange(investimento, - quantidade)`.

Outras funções aplicadas a data frames

- A função **select** do pacote **dplyr** permite **selecionar colunas dos data frames**.

```
> library(dplyr)
> selecionado <- select(investimento, acao, quantidade)
> selecionado
```

	acao	quantidade
1	RADL3	100
2	HYPE3	20
3	USIM5	35
4	BRAP4	40
5	VALE3	90

Somente as colunas ação e quantidade são selecionadas.

Todas as colunas de ação a valor são selecionadas.

```
> selecionado <- select(investimento, acao:valor)
> selecionado
```

	acao	quantidade	valor
1	RADL3	100	77.28
2	HYPE3	20	27.71
3	USIM5	35	6.98
4	BRAP4	40	32.03
5	VALE3	90	54.97

```
> selecionado <- select(investimento, - quantidade)
```

```
> selecionado
```

	acao	valor
1	RADL3	77.28
2	HYPE3	27.71
3	USIM5	6.98
4	BRAP4	32.03
5	VALE3	54.97


Todas as colunas, exceto quantidade, são selecionadas.

Outras funções aplicadas a data frames

- A função **mutate** do pacote **dplyr** permite **criar** novas **colunas** nos **data frames**.

```
> library(dplyr)
> investimento <- mutate(investimento, investido = quantidade * valor)
> investimento
```

	acao	quantidade	valor	investido
1	RADL3	100	77.28	7728.0
2	HYPE3	20	27.71	554.2
3	USIM5	35	6.98	244.3
4	BRAP4	40	32.03	1281.2
5	VALE3	90	54.97	4947.3



Para incluir no data frame a coluna investido, use:
`investimento <- mutate (investimento, valorinvestido = quantidade * valor).`

Outras funções aplicadas a data frames

- A função **mutate** do pacote **dplyr** permite **criar novas colunas** nos **data frames**.

```
> library(dplyr)
> investimento <- mutate(investimento, a.investir = quantidade * valor,
investir = ifelse(quantidade < 50, "Sim", "Não"))
> investimento
```

	acao	quantidade	valor	a.investir	investir
1	RADL3	100	77.28	7728.0	Não
2	HYPE3	20	27.71	554.2	Sim
3	USIM5	35	6.98	244.3	Sim
4	BRAP4	40	32.03	1281.2	Sim
5	VALE3	90	54.97	4947.3	Não

A coluna a.investir foi adicionada ao data frame e a coluna investir foi criada a partir de uma instrução ifelse aplicada à coluna quantidade.

Exercício 3

- Uma empresa possui duas filiais cujos gastos ao longo dos seis primeiros meses do ano com pessoal e com aluguel estão armazenados em dois arquivos: `filial01.csv` e `filial02.txt`. Para os dados das filiais:
 - Crie um data frame (`filial01` e `filial02`) para cada uma das filiais contendo as informações de mês, pessoal e aluguel.
 - Para cada uma das filiais imprima na tela o menor e o maior valor gasto com pessoal.
 - Para cada uma das filiais imprima na tela a média de gastos com aluguel.

Exercício 3 – Resolução

- Uma empresa possui duas filiais cujos gastos ao longo dos seis primeiros meses do ano com pessoal e com aluguel estão armazenados em dois arquivos: filial01.csv e filial02.txt. Para os dados das filiais:
 - Crie um data frame (filial01 e filial02) para cada uma das filiais contendo as informações de mês, pessoal e aluguel.
 - Para cada uma das filiais imprima na tela o menor e o maior valor gasto com pessoal.
 - Para cada uma das filiais imprima na tela a média de gastos com aluguel.

```
setwd('c:/PUC/INF1514')
```

```
library("descr")  
file.head("filial01.CSV")  
file.head("filial02.TXT")
```



Análise dos arquivos.

Exercício 3 – Resolução

- Uma empresa possui duas filiais cujos gastos ao longo dos seis primeiros meses do ano com pessoal e com aluguel estão armazenados em dois arquivos: filial01.csv e filial02.txt. Para os dados das filiais:
 - Crie um data frame (filial01 e filial02) para cada uma das filiais contendo as informações de mês, pessoal e aluguel.
 - Para cada uma das filiais imprima na tela o menor e o maior valor gasto com pessoal.
 - Para cada uma das filiais imprima na tela a média de gastos com aluguel.

```
filial01 <- read.csv2("filial01.CSV", dec = ".")
filial01
```

```
filial02 <- read.fwf("filial02.TXT", col.names = c("mes", "pessoal", "aluguel"),
widths = c(3, 5, 6), colClasses = c("character", "numeric", "numeric"))
filial02$aluguel <- filial02$aluguel / 100
filial02
```


Exercício 3

- Uma empresa possui duas filiais cujos gastos ao longo dos seis primeiros meses do ano com pessoal e com aluguel estão armazenados em dois arquivos: filial01.csv e filial02.txt. Para os dados das filiais:
 - Crie um data frame (filial01 e filial02) para cada uma das filiais contendo as informações de mês, pessoal e aluguel.
 - Para cada uma das filiais imprima na tela o menor e o maior valor gasto com pessoal.
 - Para cada uma das filiais imprima na tela a média de gastos com aluguel.

```
filial01.menor.gasto <- sapply(filial01["pessoal"], min)
print(filial01.menor.gasto[[1]])
filial01.maior.gasto <- sapply(filial01["pessoal"], max)
print(filial01.maior.gasto[[1]])
filial02.menor.gasto <- sapply(filial02["pessoal"], min)
print(filial02.menor.gasto[[1]])
filial02.maior.gasto <- sapply(filial02["pessoal"], max)
print(filial02.maior.gasto[[1]])
```

Exercício 3

- Uma empresa possui duas filiais cujos gastos ao longo dos seis primeiros meses do ano com pessoal e com aluguel estão armazenados em dois arquivos: filial01.csv e filial02.txt. Para os dados das filiais:
 - Crie um data frame (filial01 e filial02) para cada uma das filiais contendo as informações de mês, pessoal e aluguel.
 - Para cada uma das filiais imprima na tela o menor e o maior valor gasto com pessoal.
 - Para cada uma das filiais imprima na tela a média de gastos com aluguel.

```
filial01.media.gasto <- sapply(filial01["aluguel"], mean)
print(filial01.media.gasto[[1]])
```

```
filial02.media.gasto <- sapply(filial02["aluguel"], mean)
print(filial02.media.gasto[[1]])
```

Exercício 4

- Para a empresa do exercício anterior:
 - Crie uma nova coluna chamada filial em cada um dos dois data frames e atribua para a nova coluna filial o valor “01” para todos os registros do data frame filial01 e o valor “02” para todos os registros do data frame filial02. A seguir, crie um data frame chamado empresa unindo os registros dos data frames filial01 e filial02.
 - Aumente em 10% o valor dos aluguéis.
 - Crie no data frame empresa uma nova coluna chamada desvio e atribua o valor 40000 - (pessoal + aluguel), caso a filial seja “01” e o valor 15000 - (pessoal + aluguel), caso a filial seja “02”.
 - Imprima na tela a soma dos valores da coluna desvio.

Exercício 4 – Resolução

- Para a empresa do exercício anterior:
 - Crie uma nova coluna chamada filial em cada um dos dois data frames e atribua para a nova coluna filial o valor “01” para todos os registros do data frame filial01 e o valor “02” para todos os registros do data frame filial02. A seguir, crie um data frame chamado empresa unindo os registros dos data frames filial01 e filial02.
 - Aumente em 10% o valor dos aluguéis.
 - Crie no data frame empresa uma nova coluna chamada desvio e atribua o valor 40000 - (pessoal + aluguel), caso a filial seja “01” e o valor 15000 - (pessoal + aluguel), caso a filial seja “02”.
 - Imprima na tela a soma dos valores da coluna desvio.

```
filial01$filial <- rep ("01", 6)
filial02$filial <- rep ("02", 6)
empresa <- merge(filial01, filial02, all = TRUE)
```

Exercício 4

- Para a empresa do exercício anterior:
 - Crie uma nova coluna chamada filial em cada um dos dois data frames e atribua para a nova coluna filial o valor “01” para todos os registros do data frame filial01 e o valor “02” para todos os registros do data frame filial02. A seguir, crie um data frame chamado empresa unindo os registros dos data frames filial01 e filial02.
 - Aumente em 10% o valor dos aluguéis.
 - Crie no data frame empresa uma nova coluna chamada desvio e atribua o valor 40000 - (pessoal + aluguel), caso a filial seja “01” e o valor 15000 - (pessoal + aluguel), caso a filial seja “02”.
 - Imprima na tela a soma dos valores da coluna desvio.

```
empresa$aluguel <- empresa$aluguel * 1.1  
empresa$aluguel
```

Exercício 4

- Para a empresa do exercício anterior:
 - Crie uma nova coluna chamada filial em cada um dos dois data frames e atribua para a nova coluna filial o valor “01” para todos os registros do data frame filial01 e o valor “02” para todos os registros do data frame filial02. A seguir, crie um data frame chamado empresa unindo os registros dos data frames filial01 e filial02.
 - Aumente em 10% o valor dos aluguéis.
 - Crie no data frame empresa uma nova coluna chamada desvio e atribua o valor 40000 - (pessoal + aluguel), caso a filial seja “01” e o valor 15000 - (pessoal + aluguel), caso a filial seja “02”.
 - Imprima na tela a soma dos valores da coluna desvio.

```
library(dplyr)
empresa <- mutate(empresa, desvio = ifelse(filial == "01", 40000 - (pessoal +
aluguel), 15000 - (pessoal + aluguel)))
empresa
```

Exercício 4

- Para a empresa do exercício anterior:
 - Crie uma nova coluna chamada filial em cada um dos dois data frames e atribua para a nova coluna filial o valor “01” para todos os registros do data frame filial01 e o valor “02” para todos os registros do data frame filial02. A seguir, crie um data frame chamado empresa unindo os registros dos data frames filial01 e filial02.
 - Aumente em 10% o valor dos aluguéis.
 - Crie no data frame empresa uma nova coluna chamada desvio e atribua o valor 40000 - (pessoal + aluguel), caso a filial seja “01” e o valor 15000 - (pessoal + aluguel), caso a filial seja “02”.
 - Imprima na tela a soma dos valores da coluna desvio.

```
valor.desvio <- sapply(empresa["desvio"], sum)
print(valor.desvio)
```

Listas

- **Listas** são um tipo especial de **vetor** que aceita elementos de classes diferentes.
- Uma **lista** é criada com a função **list()**, que aceita um número arbitrário de elementos, sendo que uma **lista** pode também conter outra **lista**.

```
lancamento <- list(  
  mes = "Jan",  
  valores = list(  
    pessoal = 3340.00,  
    aluguel = 256.00  
  ),  
  filial = "01"  
)
```

```
> lancamento  
$mes  
[1] "Jan"  
  
$valores  
$valores$pessoal  
[1] 3340  
  
$valores$aluguel  
[1] 256  
  
$filial  
[1] "01"
```


Listas

- **Listas** são um tipo especial de **vetor** que aceita elementos de classes diferentes.
- Uma **lista** é criada com a função **list()**, que aceita um número arbitrário de elementos, sendo que uma **lista** pode também conter outra **lista**.

```
lancamento <- list(
  mes = "Jan",
  valores = list(
    pessoal = 3340.00,
    aluguel = 256.00
  ),
  filial = "01"
)
```

```
> lancamento$mes
[1] "Jan"
> lancamento$valores
$pessoal
[1] 3340
$aluguel
[1] 256
> lancamento$valores$pessoal
[1] 3340
> lancamento$valores$aluguel
[1] 256
```

Arrays

- **Arrays** são a classe generalizada das **matrizes** e **vetores** e possuem até 3 dimensões que podem ser *numeric*, *character*, *complex* ou *logical*.
- Para acessar um elemento de um **array**, caso ele tenha 3 dimensões, a primeira coordenada representa a **linha**, a segunda coordenada representa a **coluna** e a terceira, a **página**.

```
> dados.array <- array(1:6, c(3, 2))
```

```
> dados.array
      [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

```
> dados.array[1, ]
```

```
[1] 1 4
```

```
> dados.array[, 1]
```

```
[1] 1 2 3
```

```
> dados.array[3, 2]
```

```
[1] 6
```

```
> dados.array[1, ][2]
```

```
[1] 4
```

```
> dados.array[3, 2][1]
```

```
[1] 6
```

Arrays

```
> dados.array <- array(1:24, c(3, 4, 2))
```

```
> dados.array
```

```
, , 1
```

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

```
, , 2
```

	[,1]	[,2]	[,3]	[,4]
[1,]	13	16	19	22
[2,]	14	17	20	23
[3,]	15	18	21	24

```
> dados.array[1, , ]
```

	[,1]	[,2]
[1,]	1	13
[2,]	4	16
[3,]	7	19
[4,]	10	22

```
> dados.array[, 1, ]
```

	[,1]	[,2]
[1,]	1	13
[2,]	2	14
[3,]	3	15

```
> dados.array[, , 1]
```

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

```
> dados.array[2, 3, 1]
```

```
[1] 8
```