PUC-RIO
INF1036 - Probabilidade Computacional

Material Extra – Noções de R

*Material Adaptado da Professora Ana Carolina Letichevsky

Classes básicas de objetos no R

- Toda variável dentro da linguagem R é interpretada como um objeto que pertence a uma determinada classe.
- 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 básicas são chamados de objetos atômicos.
- Além dessas classes básicas, o R apresenta outras mais complexas que só podem conter objetos de um mesmo tipo, como é o caso das matrizes (classe **matrix**), ou que podem conter objetos de tipos diferentes, como é o caso das listas (classe **list**).
- Para saber qual a classe de um determinado objeto podemos usar a função **class**.

Classes básicas de objetos no R

```
> nome <- "Amanda da Silva"
> class(nome)
[1] "character"
>
> acertou <- TRUE
> class(acertou)
[1] "logical"
>
> x <- 45
> class(x)
[1] "numeric"
>
> y <- 45.2
> class(y)
[1] "numeric"
>
> z <- 45L
> class(z)
[1] "integer"
```

Os objetos do tipo texto fazem parte da classe "character" e irão aparecer entre aspas.

Os objetos do tipo lógico fazem parte da classe "logical" e podem assumir valores TRUE ou FALSE. Também podem ser simplesmente representados pelas letras T ou F.

Os objetos numéricos fazem parte da classe "numeric" sendo tratados como números reais.

Para fazer um número inteiro ser tratado como objeto inteiro, deve-se utilizar a letra L após o número.

Vetores

- Um vetor é uma estrutura de dados que armazena uma **coleção de elementos** de tal forma que cada um dos elementos pode ser identificado por um **índice**.
- No R podemos entender vetor como uma **coleção de objetos**, sendo todos de um **mesmo tipo**, obrigatoriamente.
- Vetores no R são os objetos **mais simples** que podem armazenar **objetos atômicos**.

Instrução para criação de vetores	Descrição	Exemplo
:	Cria sequências numéricas.	v <- 1:5 v <- -3:3
scan	Lê valores digitados.	v <- scan()
c	Combina valores.	v <- c (3, 4, 8, 9) v <- c (“verde”, “amarelo”)
rep	Repete valores.	v <- rep (c (3, 4), 2)
seq	Cria sequências numéricas.	v <- seq (-1, 1, 0.5)

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
```

Exemplo de prioridade da instrução : em relação ao operador *.

Instrução scan

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

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

Instrução c

- A instrução **c** combina valores.

```
> v <- c(3, 4, 8, 9)
> v
[1] 3 4 8 9
>
> v <- c("verde", "amarelo")
> v
[1] "verde" "amarelo"
```

Instrução rep

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

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

Instrução seq

- A instrução **seq** cria sequências numéricas tendo como argumentos valores que definem 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
```

Vetores

- No R, um vetor **tem** sempre a **mesma classe dos objetos** que **armazena**.
- O R **trata qualquer objeto como um vetor**. No caso de um **objeto simples**, e não uma coleção de objetos, o R interpreta o objeto como se fosse um vetor de **tamanho 1**.
- **Length** refere-se ao **número** de elementos do vetor.
- O comando **class** retorna a **classe** dos elementos do vetor.

```
> cor <- c("verde", "amarelo")
> cor
[1] "verde" "amarelo"
> length(cor)
[1] 2
> class(cor)
[1] "character"
>
> v <- 1:5
> v
[1] 1 2 3 4 5
> length(v)
[1] 5
> class(v)
[1] "integer“
>
> x <- 45
> x[1]
[1] 45
```

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"
```

```
> v3 <- c(6, 7, 8)
> v3
[1] 6 7 8
> class(v3)
[1] "numeric"
>
> v <- c (v1, v3)
> v
[1] 1 2 3 4 5 6 7 8
> class(v)
[1] " numeric"
```

A concatenação também funcionaria se v3 <- c (6, 7, 8). Neste caso, os vetores v3 e v seriam da classe “numeric”.

Concatenação de Vetores

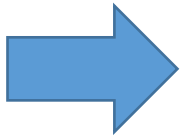
- Podemos concatenar vetores de uma mesma classe com a instrução **c**.

```
> v1 <- c("verde", "amarelo")
> v1
[1] "verde" "amarelo"
> class(v1)
[1] "character"
>
> v2 <- c("azul", "vermelho")
> v2
[1] "azul" "vermelho"
> class(v2)
[1] "character"
>
> v <- c(v1, v2)
> v
[1] "verde" "amarelo" "azul" "vermelho“
> 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. (imc = peso/((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) / length(altura)
[1] 1.76
> sum ((altura - mean(altura))^2) / (length (altura) - 1)
[1] 0.0122
```

Instruções Comparativas

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

Instruções Comparativas

- A instruções comparativas verificam elemento por elemento do vetor.

```
> v <- 1:5
> v
[1] 1 2 3 4 5
>
> v < 2
[1] TRUE FALSE FALSE FALSE FALSE
>
> v <- c(3, 4, 8, 9)
> v
[1] 3 4 8 9
>
> v <= 4
[1] TRUE TRUE FALSE FALSE
>
> v <- rep( c(3, 4), 2)
> v [1] 3 4 3 4
>
> v >= 3
[1] TRUE TRUE TRUE TRUE
```

```
> v <- seq(-1, 1, 0.5)
> v
[1] -1.0 -0.5 0.0 0.5 1.0
>
> v == 1
[1] FALSE FALSE FALSE FALSE TRUE
>
> v <- seq(-1, 1, 0.5)
> v [1] -1.0 -0.5 0.0 0.5 1.0
>
> v != 1
[1] TRUE TRUE TRUE TRUE FALSE
```

Instruções Lógicas

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

Instruções Lógicas

- A instruções lógicas verificam elemento por elemento do vetor.

```
> v <- 1:5
> v
[1] 1 2 3 4 5
>
> (v > 2) & (v < 4)
[1] FALSE FALSE TRUE FALSE FALSE
>
> v <- c(3, 4, 8, 9)
> v
[1] 3 4 8 9
>
> (v <= 4) | (v == 9)
[1] TRUE TRUE FALSE TRUE
>
> 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
TRUE	FALSE	TRUE
FALSE	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
>
> v1 <- v[6]
> v1
[1] 10
>
> v2 <- v[3:5]
> v2
[1] 7 8 9
>
> v3 <- v[c(2, 4, 10)]
> v3
[1] 5 8 9
```

```
> v4 <- v[v > 6]
> v4
[1] 7 8 9 10 12 7 9
>
> v5 <- v[(v > 6) & (v <= 9)]
> v5
[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"
>
> v1 <- as.numeric(v)
> v1
[1] 0 1 2 3 4 5
> class(v1)
[1] "numeric"
```

```
> v2 <- as.logical(v)
> v2
[1] FALSE TRUE TRUE TRUE TRUE TRUE
> class(v2)
[1] "logical"
>
> v3 <- as.character(v)
> v3
[1] "0" "1" "2" "3" "4" "5"
> class(v3)
[1] "character"
```

Classes diferentes em um mesmo Vetor

- Se o R não entender como **coagir** uma **classe na outra**, ele levantará um ***warning*** informado 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
>
> v1 <- as.logical(v)
> v1
[1] FALSE TRUE TRUE TRUE FALSE TRUE
```

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

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"
>
> length(mat)
[1] 12
```

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

Será visto posteriormente.

Criação de Matrizes

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

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 três 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, ncol = 2)
> mat
      [,1] [,2]
[1,]    3    8
[2,]    4    9
```

A matriz terá duas 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
>
> length(mat)
[1] 6
```

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

```
> 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 função *rbind* acrescentou uma nova linha na matriz formada pelos números 18, 20 e 35.

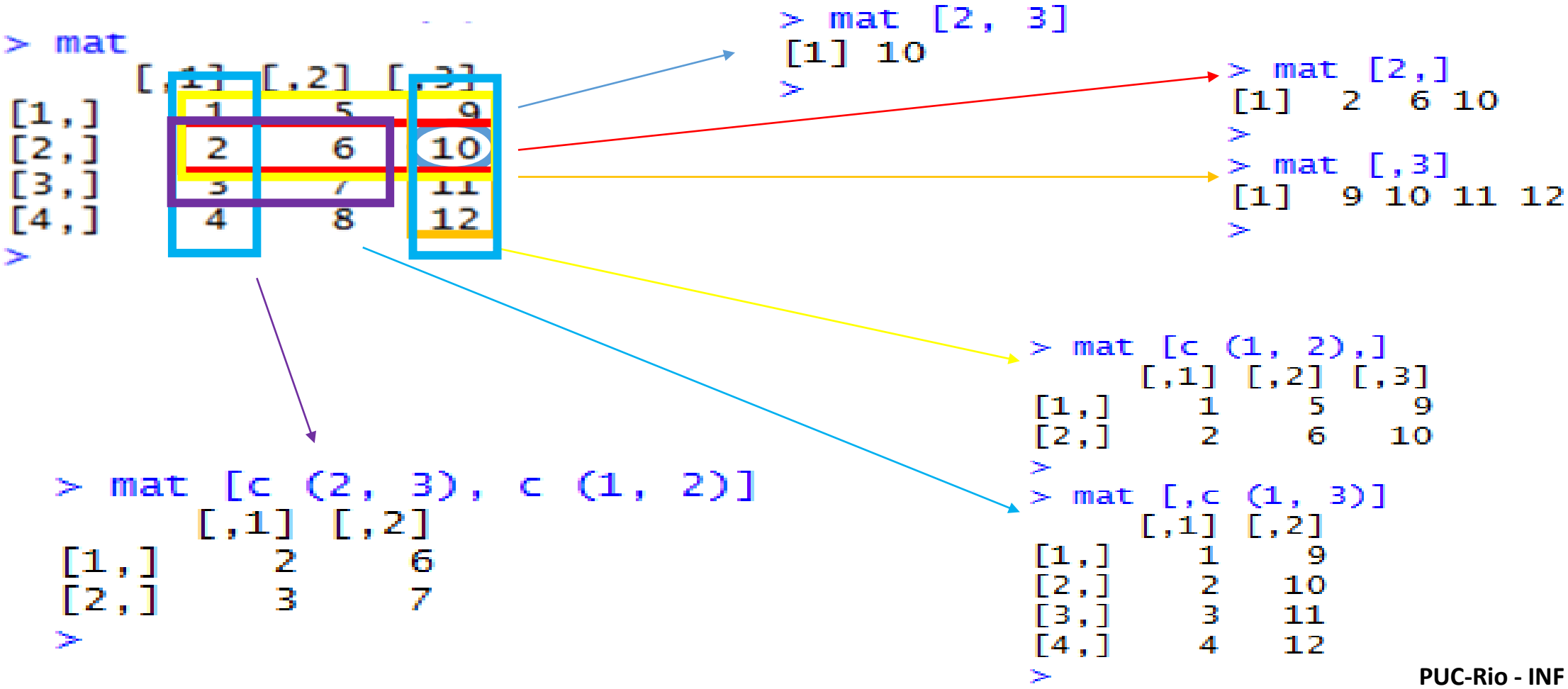
```
> mat.mais.uma.linha <- rbind(mat, c(18, 20, 35))
> mat.mais.uma.linha
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
[5,]   18   20   35
```

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

```
> mat.mais.uma.coluna <- cbind(mat, 1:4)
> mat.mais.uma.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
>
> mat < 5
      [,1]  [,2]  [,3]
[1,]  TRUE FALSE FALSE
[2,]  TRUE FALSE FALSE
[3,]  TRUE FALSE FALSE
[4,]  TRUE FALSE FALSE
>
> mat[, 2] > 3
[1] TRUE TRUE TRUE TRUE
```

```
> mat[1, 3] <= 4
[1] FALSE
>
> mat[1, ] >= 3
[1] FALSE TRUE TRUE
>
> mat == 1
      [,1]  [,2]  [,3]
[1,]  TRUE FALSE FALSE
[2,] FALSE FALSE FALSE
[3,] FALSE FALSE FALSE
[4,] FALSE FALSE FALSE
>
> mat != 1
      [,1]  [,2]  [,3]
[1,] FALSE  TRUE  TRUE
[2,]  TRUE  TRUE  TRUE
[3,]  TRUE  TRUE  TRUE
[4,]  TRUE  TRUE  TRUE
```

Instruções Lógicas

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

Instruções Lógicas

- A 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
>
> (mat > 2) & (mat < 4)
      [,1]  [,2]  [,3]
[1,] FALSE FALSE FALSE
[2,] FALSE FALSE FALSE
[3,]  TRUE FALSE FALSE
[4,] FALSE FALSE FALSE
>
> (mat[, 2] <= 6 | mat[, 2] == 9)
[1] TRUE TRUE FALSE FALSE
>
> (mat[, 3] <= 10 | mat[, 2] == 9)
[1] TRUE TRUE FALSE FALSE
>
> xor((mat == 1), (mat > 0))
      [,1]  [,2]  [,3]
[1,] FALSE  TRUE  TRUE
[2,]  TRUE  TRUE  TRUE
[3,]  TRUE  TRUE  TRUE
[4,]  TRUE  TRUE  TRUE
```

```
> v1 <- (mat[, 2] <= 6)
> v1
[1] TRUE TRUE FALSE FALSE
> v2 <- (mat[, 2] == 9)
> v2
[1] FALSE FALSE FALSE FALSE
> v1 | v2
[1] TRUE TRUE FALSE FALSE
```

```
> v1 <- (mat[, 3] <= 10)
> v1
[1] TRUE TRUE FALSE FALSE
> v2 <- (mat[, 2] == 9)
> v2
[1] FALSE FALSE FALSE FALSE
> v1 & v2
[1] FALSE FALSE FALSE FALSE
```

Outras Operações com Matrizes

```
> v <- c(1, 1, 1, 0, 3, 2, 0, 1, 0)
> mat1 <- matrix(v, ncol = 3)
> mat1
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    1    3    1
[3,]    1    2    0
>
> t(mat1)
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    0    3    2
[3,]    0    1    0
>
> solve(mat1)
     [,1] [,2] [,3]
[1,]  1.0  0  0.0
[2,] -0.5 0  0.5
[3,]  0.5  1  -1.5
> > mat1 / 2
     [,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
>
> mat1 %*% mat2
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]   10   25   40
[3,]    5   14   23
```

Produto de matriz.

Matriz transposta.
Matriz inversa.
Matriz divida por 2.

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.

```
> 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)
```

A função *data.frame* cria um data frame a partir de vetores.

O data frame investimento possui 3 colunas e 5 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
```

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
```

Qual é o retorno de 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
```

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

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

Removendo 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
6 ITUB4          55    42.09
```

Remove as linhas 2 e 4 do data frame.

```
> investimento <- investimento [ c(-2, -4), ]
> investimento
   acao quantidade valor
1 RADL3         100 77.28
3 USIM5          35  6.98
5 VALE3          90 54.97
6 ITUB4          55 42.09
```

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

```
> investimento <- investimento [ investimento$valor > 7, ]
> investimento
   acao quantidade valor
1 RADL3         100 77.28
5 VALE3          90 54.97
6 ITUB4          55 42.09
```

Adicionando Colunas

```
> investimento
   acao quantidade    valor
1 RADL3        100    77.28
5 VALE3         90    54.97
6 ITUB4         55    42.09
```

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

```
> investimento <- cbind (investimento, variacao = c (0.2, -0.3, 0.5))
> investimento
   acao quantidade valor variacao
1 RADL3        100 77.28      0.2
5 VALE3         90 54.97     -0.3
6 ITUB4         55 42.09      0.5
```

Adicionando Colunas

```
> investimento
   acao quantidade valor variacao
1 RADL3        100 77.28      0.2
5 VALE3         90 54.97     -0.3
6 ITUB4         55 42.09      0.5
```

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

```
> investimento$variacao2 <- c (0.2, -0.3, 0.5)
> investimento
   acao quantidade valor variacao variacao2
1 RADL3        100 77.28      0.2       0.2
5 VALE3         90 54.97     -0.3      -0.3
6 ITUB4         55 42.09      0.5       0.5
```

Removendo Colunas

```
> investimento
   acao quantidade valor variacao variacao2
1 RADL3        100 77.28      0.2       0.2
5 VALE3         90 54.97     -0.3      -0.3
6 ITUB4         55 42.09      0.5       0.5
```

Remove a coluna valor do data frame.

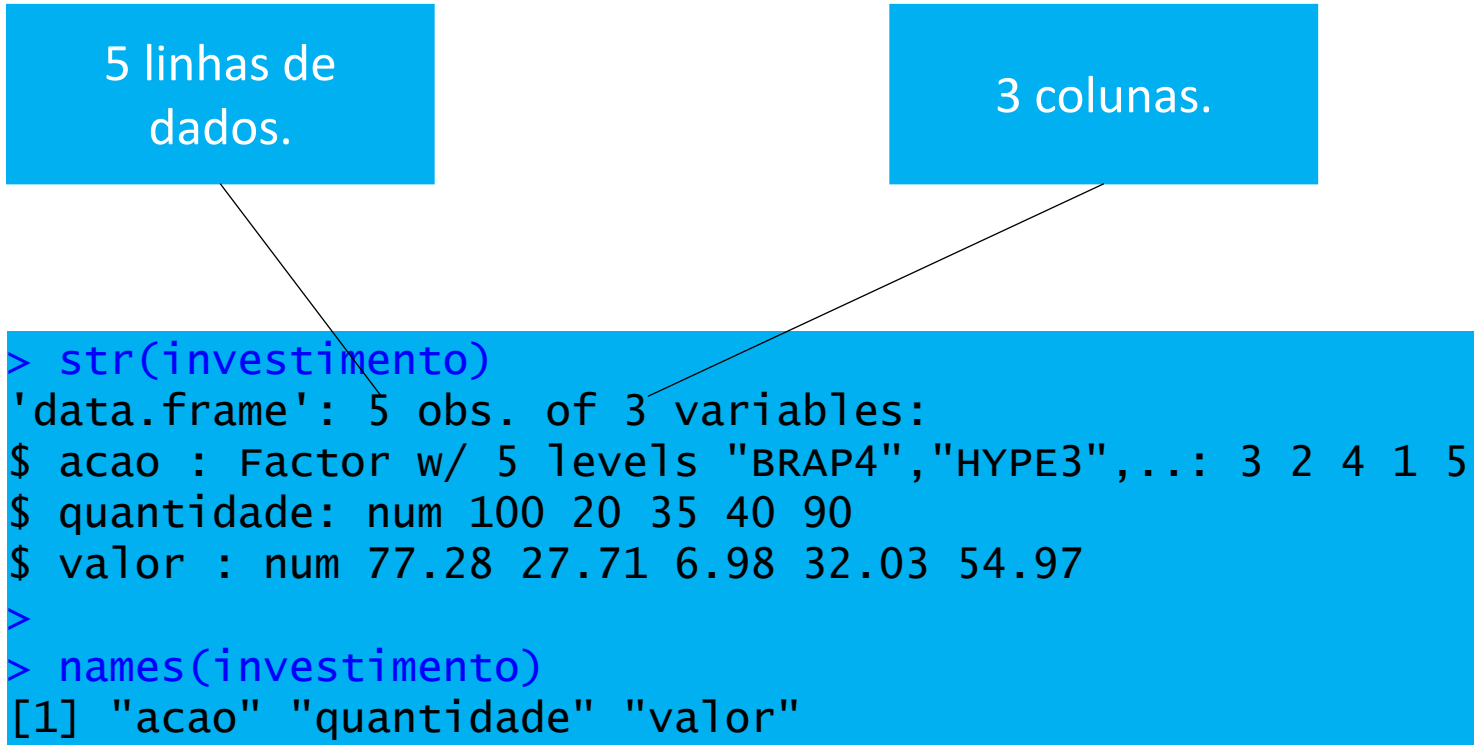
```
> investimento$valor <- NULL
> investimento
   acao quantidade variacao variacao2
1 RADL3        100      0.2       0.2
5 VALE3         90     -0.3      -0.3
6 ITUB4         55      0.5       0.5
```

Remove a coluna variacao do data frame.

```
> investimento <- investimento [, -3]
> investimento
   acao quantidade variacao2
1 RADL3        100       0.2
5 VALE3         90      -0.3
6 ITUB4         55       0.5
```

Estrutura

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



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
```

```
> sapply (investimento ["quantidade"], min)
quantidade 20
>
> sapply (investimento [3], max)
valor 77.28
>
> sapply (investimento [2:3], mean)
quantidade  valor
    57.000 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)
>
> investimento2 <- data.frame (acao, quantidade, valor)
> investimento2
   acao quantidade valor
1 ITUB4         10 22.28
2 PTRE2         65 10.71
```

```
> investimento.final <- merge (investimento, investimento2, all = TRUE)
> investimento.final
acao quantidade valor
1 BRAP4 40 32.03
2 HYPE3 20 27.71
3 RADL3 100 77.28
4 USIM5 35 6.98
5 VALE3 90 54.97
6 ITUB4 10 22.28
7 PTRE2 65 10.71
```

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**.

```
> acao.compra
  codigo compra
1   GFX6    1.01
2   ELT6    0.30
3   GLO5    1.26
4   KLM1    1.82
```

```
codigo <- c("GFX6", "ELT6", "GLO5", "KLM1")
compra <- c(1.01, 0.30, 1.26, 1.82)

acao.compra <- data.frame(codigo, compra)
acao.compra
```

```
> acao.venda
  identificador venda
1          ELT6  0.98
2          GLO5  0.78
3          GFX6  1.23
4          KLM1  1.95
```

```
identificador <- c("ELT6", "GLO5", "GFX6", "KLM1")
venda <- c (0.98, 0.78, 1.23, 1.95)

acao.venda <- data.frame(identificador, venda)
acao.venda
```

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

```
> 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.

```
> investimento.f <- investimento[investimento$quantidade >= 70, ]
> investimento.f
   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) #Permite usar o pacote dplyr
> filter (investimento, quantidade >= 70)
acao quantidade valor
1 RADL3 100 77.28
2 VALE3 90 54.97
```

A função filter gera um data frame como retorno.

```
> library (dplyr) #Permite usar o pacote dplyr
> filter (investimento, quantidade >= 70 & valor < 60)
   acao quantidade valor
1 VALE3         90 54.97
```

Ordenando Data Frames

- Mecanismo padrão com **order**.

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

```
> investimento [order (investimento$acao), ]
   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
```

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

```
> library (dplyr)
> arrange (investimento, quantidade)
   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)
> select (investimento, acao, quantidade)
   acao quantidade
1 RADL3        100
2 HYPE3         20
3 USIM5         35
4 BRAP4         40
5 VALE3         90
```

Somente as colunas acao e quantidade são selecionadas.

```
> library (dplyr)
> select (investimento, - quantidade)
   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 (sinal -) quantidade, são selecionadas.

```
> library (dplyr)
> select(investimento, acao:valor)
   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
```

Todas as colunas de acao a valor são selecionadas.

Outras Funções Aplicadas a Data Frames

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

```
> library (dplyr)
> 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 <- mutate (investimento, valorinvestido = quantidade * valor)
   acao quantidade valor valorinvestido
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 valorinvestido, 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, valorinvestido = quantidade * valor, investir =
ifelse(quantidade < 50, "Sim", "Não"))
> investimento
   acao quantidade valor valorinvestido 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 valorinvestido foi adicionada ao data frame, e a coluna investir foi criada a partir de uma instrução *ifelse* aplicada à coluna quantidade.

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),
+                     operacao = "Credito",
+                     filial = "01")
> lancamento
$mes
[1] "Jan"

$valores
$valores$pessoal
[1] 3340

$valores$aluguel
[1] 256

$operacao
[1] "Credito"

$filial
[1] "01"
```

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

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 (1:8, 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[1, 1]
[1] 1
```

```
> 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
```