Programação Estatística

Introdução ao R - Estrutura de Dados

Rachid Muleia, PhD in Statistics

Universidade Eduardo Mondlane
Faculdade de Ciências
Departamento de Matemática e Informática

2023-03-02

Objectos

Criação de objectos

Forma geral

- `variavel <- valor`

Nomes permitidos

- `meu.objecto`, `meu_objecto`, `meuObjecto`, `a`, `b`, `x1`, `x2`, `data1`, `1data`

Nomes não permitidos

- `.` seguido de um número no início: `.4you`
- Não se pode usar palavras reservadas: `if`, `else`, `repeat`, `for`, `while`, `function`, `TRUE`, ‘`FALSE`, etc.
- Mais descrições sobre palavras reservadas podem ser encontradas executando o código: `help("reserved")`

Tipos de dados

Tipo de dados

- Logical
- Numeric
- Integer
- Character
- Complex

Estruturas de dados

- Vectores
- Factor
- Listas -Matrix
- Data frame

Tipo de dados

Tipo de dados	Descrição	Exemplop
Logical	Verdadeiro ou Falso	TRUE ou FALSE
Numeric	Números reais	2.3, pi, `sqrt(2)`
Integer	Números inteiros	`5L, 4L,-1L`
Character	Sequência de caracteres	`"maria","UEM"`
Complex	Números complexos	2.1+3i, 1+0i

Vectores

Vectores

- Estrutura unidimensional
- Coleccção de valores onde todos têm o mesmo tipo de dados
- Exemplo: `(-2,3.1,2.4,5)`, `(TRUE, FALSE,TRUE,FALSE)`, `("Maria", "Joao","Augusto", "Antonio")`

Vectores

Vectores podem ser criados usando as seguintes funções:

- `c()`: função para combinar valores individuais
- `seq()`: função para criar uma sequência de valores
- `rep()`: função para criar réplica de valores

Vectores

Exemplo de criação de fectores usando a função c()

```
> c(1,2,3,4,5,6,7)
## [1] 1 2 3 4 5 6 7
> c(2:5,11:6)
##  [1]  2  3  4  5 11 10  9  8  7  6
```

```
> x<-c(2,7,8,12,3,25)
> x
## [1]  2  7  8 12  3 25
```

```
y<-c(red="Bob",blue="Dave",green="Jenny")
y
```

```
##     red    blue   green
##   "Bob"  "Dave" "Jenny"
```

Vectores

Exemplo de criação de vectores usando a função `seq()`

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

Vectores

Exemplo de criação de vectores usando a função `seq()`

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

```
> seq(1,8)
## [1] 1 2 3 4 5 6 7 8
> seq(4,10,2)
## [1]  4  6  8 10
> seq(1,10,,4)
## [1]  1  4  7 10
```

Vectores

Criação de vectores usando a função `rep()`

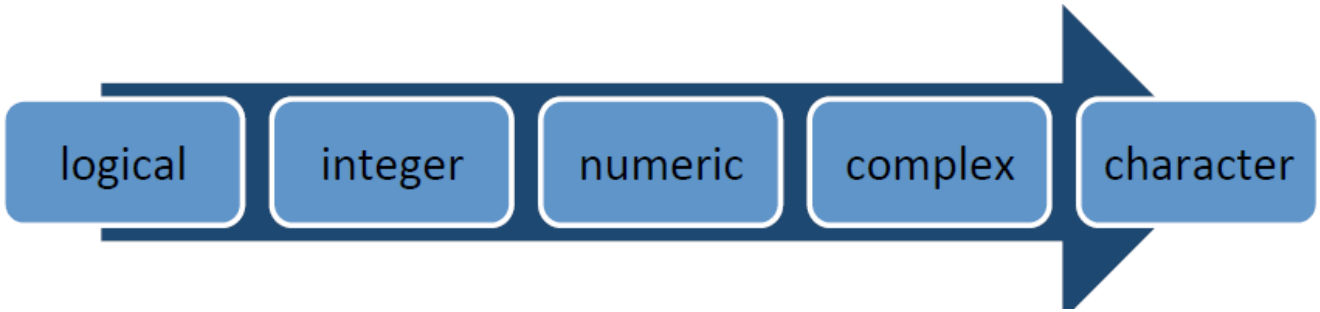
```
> rep(1,4)
## [1] 1 1 1 1
> rep(4:5,3)
## [1] 4 5 4 5 4 5
> rep(1:4,each=2)
## [1] 1 1 2 2 3 3 4 4
> rep(1:4,times=2,each=2)
##  [1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
```

Vectores

Conversão do tipo de dados

- Vectores levam apenas um único tipo de dados
- Ao combinar diferentes tipos de dados, o R irá fazer a coerção para o tipo de dados mais flexível

Regra para coerção:



Vectores

Coerção de dados- exemplo

```
> x<-c(5,'b')    # será convertido para character
> x              # " " indica tipo character
## [1] "5" "b"
> y<-c(FALSE, 3) # será convertido para número
> y
## [1] 0 3
```

Vectores

Coerção de dados

A coerção de dados também pode ser feita usando as funções `as.class_name`.

- Exemplo
  - `as.numeric(x)`
  - `as.logical(x)`
  - `as.character(x)`
  - `as.integer(x)`
  - `as.factor(x)`
  - `as.complex(x)`

O tipo de dados pode ser verificado usando a função `typeof(x)` ou `class(x)`.

Operações com vectores

O acesso aos elementos de um vector pode ser feito usando o operador []

```
> x<-c(2,4,6,8,10)
> x[4]
## [1] 8
> x[3:5]
## [1]  6  8 10
> x[-2]
## [1]  2  6  8 10
```

Operações com vectores

Operações padrão em vectores são feitas elemento a elemento:

```
> c(2,5,3)+c(4,2,7)
## [1]  6  7 10
> c(2,5,3)+2
## [1] 4 7 5
> c(2,5,3)^2
## [1]  4 25  9
```

Operações com vectores

Algumas funções importantes

Operação	Descrição
`class(nome_vector)`	devolve o tipo do vector
`length(nome_vector)`	devolve o número total de elementos
`x[length(x)]`	último elemento do vector
`rev(nome_vector)`	devolve o vector invertido
`sort(nome_vector)`	devolve o vector ordenado
`unique(vector_nome)`	retorna um vector com valores únicos

Factores

- Um Fator é um vetor que representa dados categóricos
- Apenas pode conter categorias predefinidas
- Podem ser ordenados, assim como não

Exemplo: `("sim", "nao", "sim", "nao", "sim")`
`("masculino", "feminino", "masculino", "feminino")`
`("grande", "pequeno", "grande","pequeno")`

Factores

A criação de um factor pode ser feita usando a função `factor()`

```
> sexo <- c("masculino", "feminino", "masculino", "feminino")
> factor(sexo)
## [1] masculino feminino  masculino feminino
## Levels: feminino masculino
```

Factores

Ordenação de factores

- Existem várias formas para ordenar um factor.

```
> sizes <- factor(c("small", "large", "large", "small", "medium"))
> sizes
## [1] small  large  large  small  medium
## Levels: large medium small
```

- Pode-se ordenar um factor especificando as ordem das categorias

```
> sizes <- factor(sizes, levels=c('small', 'medium', 'large'))
> sizes
## [1] small  large  large  small  medium
## Levels: small medium large
> sizes<-factor(sizes,levels=c('large', 'medium', 'small'))
> sizes
## [1] small  large  large  small  medium
## Levels: large medium small
```

Factor

- Uma outra forma de ordenar é usando a função `relevel()`

```
> sizes <- factor(sizes, levels=c('small', 'medium', 'large'))
> sizes <- relevel(sizes, ref = 'medium')
> sizes
## [1] small  large  large  small  medium
## Levels: medium small large
```

- A função `relevel()` é muito importante para definir a categoria de referência

Listas

Listas

- Lista é uma colecção de estruturas de dados
- Uma lista pode armazenar qualquer tipo de dados, incluido lista
- Uma lista pode ser criada usando a função `list()`
- Maior parte das funções em R produzem outputs armazenados numa lista

Lista

Exemplo de criação de lista

```
> minha_Lista<-list(1:3,c("a","b"),c(TRUE,FALSE,TRUE))
>
> str(minha_Lista)
## List of 3
##  $ : int [1:3] 1 2 3
##  $ : chr [1:2] "a" "b"
##  $ : logi [1:3] TRUE FALSE TRUE
```

Assim como vimos na criação de vectores, os elementos da lista podem ter nome

```
> minha_Lista<-list(vector1=1:3,vector2=c("a","b"),vector3=c(TRUE,FALSE,TRUE))
>
> minha_Lista
## $vector1
## [1] 1 2 3
##
## $vector2
## [1] "a" "b"
##
## $vector3
## [1]  TRUE FALSE  TRUE
```

Lista

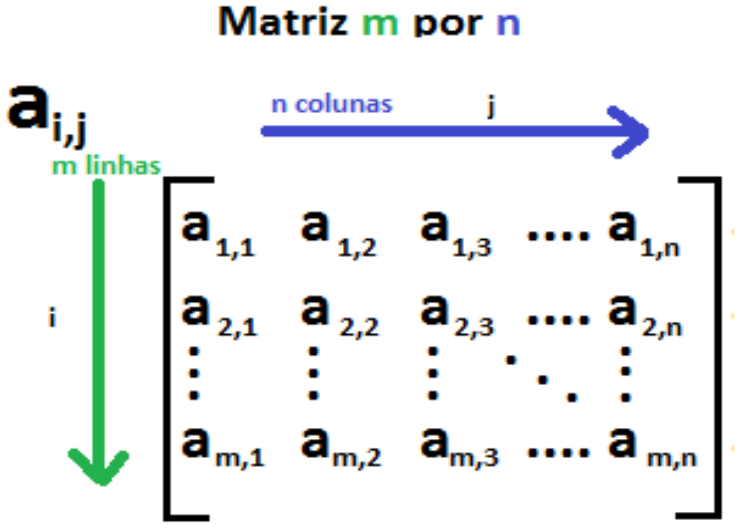
Uma outra forma de criar uma lista é usando a função `vector()`

```
> lista_vazia<-vector(mode='list', length=5)
>
> lista_vazia
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL
```

Matriz

Matriz/ Matrix

- Uma colecção de valores com mesmo tipo de dados
- Estrutura bidimensional disposta em linhas e colunas.



Matriz

Uma matriz pode ser criada usando as seguintes funções:

- `matrix()` cria uma matrix especificando as linhas e as colunas
- `dim()` cria uma matrix definindo a dimensão do vector
- `cbind()` ou `rbind` cria uma matriz combinando colunas ou linhas, respectivamente

Matriz

```
> matrix(data=1:6,nrow=2,ncol=3,byrow=FALSE)
> matrix(data=1:6, # dados para  popular a matriz
+        nrow=2,   # numero de linhas
+        ncol=3,   # numero de colunas
+        byrow=FALSE) # preencher a matriz por linha
```

Exemplo de criação de matriz

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

Matriz

Criação de matriz usando a função `dim()`

```
> x<-1:6
> dim(x)<-c(2,3)
> x
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
> y<-1:6
> dim(y)<-c(3,2)
> y
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

Matriz

Criação de matriz usando as funções `cbind()` e `rbind()`

```
> x<-1:3
> y<-10:12
>
> cbind(x,y)
##       x  y
## [1,] 1 10
## [2,] 2 11
## [3,] 3 12
> rbind(x,y)
##   [,1] [,2] [,3]
## x    1    2    3
## y   10   11   12
```

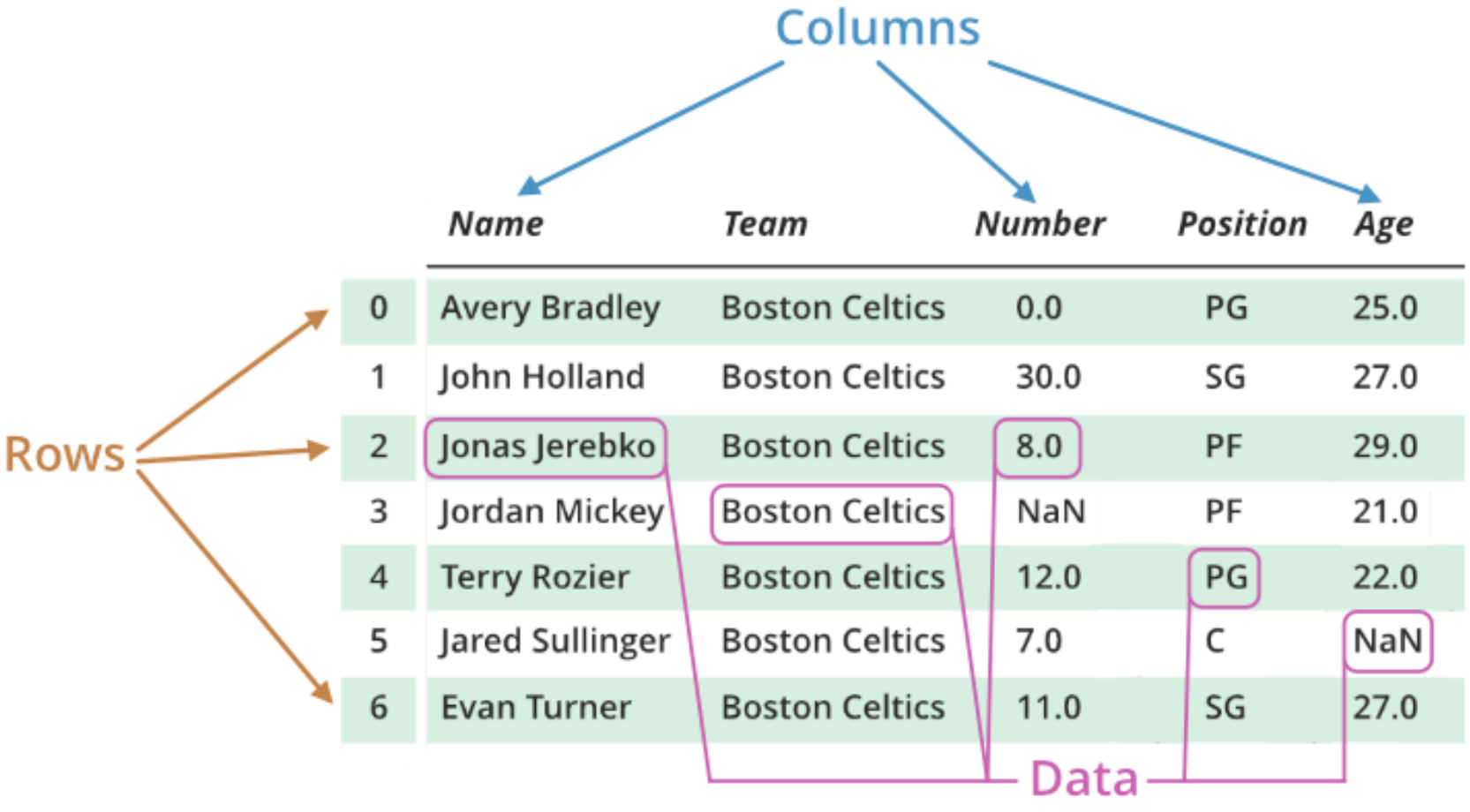
Data frame

Data frame

Colecção de vectores com igual tamanho

- Estrutura bidimensional disposta em linhas e colunas
  - **MAS**: DENTRO de uma coluna, cada célula deve ter o mesmo tipo de dados!
- `data frames` são usadas para representar todo conjunto de dados
- As colunas contêm vectores com diferentes tipos de dados

Data frame-layout



Uma data frame pode ser criada usando a função `data.frame()`.

Data frame

Exemplo de criação de data frame

```
> df <- data.frame(ID = 1:3, Sexo = c("F", "F", "M"),
+ Peso = c(71, 60, 65))
>
> df <- data.frame(
+ ID = 1:3,                    # elementos da primeira coluna
+ Sexo = c("F", "F", "M"), # elementos da segunda coluna
+ Peso = c(71, 60, 65))     # elementos da terceira coluna
```

Data frame

Exemplo de criação de data frame

```
> df <- data.frame(ID = 1:3, Sexo = c("F", "F", "M"),
+ Peso = c(71, 60, 65))
>
> df
  ID Sexo Peso
1  1    F   71
2  2    F   60
3  3    M   65
```

Data frame

Nota: Data frame, automaticamente, converte strings em factores.

```
> df_2<-data.frame(x=1:3,y=c("a","b","c"))
>
>
> str(df_2)  #  exibe a extrutura interna dos dados
'data.frame':   3 obs. of  2 variables:
 $ x: int  1 2 3
 $ y: chr  "a" "b" "c"
```

O argumento `stringsAsFactors = FALSE` impede esse comportamento

Data frame

Nota: Data frame, automaticamente, converte strings em factores.

```
> df_2<-data.frame(x=1:3,y=c("a","b","c"),
+                  stringsAsFactors = FALSE)
>
>
> str(df_2)
'data.frame':   3 obs. of  2 variables:
 $ x: int  1 2 3
 $ y: chr  "a" "b" "c"
```

Data frame

- Criar uma `data frame` manualmente leva muito tempo
- Além disso, a digitação convida erros
- Deve evitar digitar grandes conjuntos de dados no R manualmente

Data frame

- Criar uma `data frame` manualmente leva muito tempo
- Além disso, a digitação convida erros
- Deve evitar digitar grandes conjuntos de dados no R manualmente

As `data frames`, geralmente, são importadas usando as funções `read.csv()` e `read.table()`

Indexação

Indexação/subset- vectores

Para se aceder aos elementos de um vector pode-se usar seguinte notação `x[]`

```
> x<-10:22
> x
 [1] 10 11 12 13 14 15 16 17 18 19 20 21 22
> x[7]
[1] 16
> x[1:4]
[1] 10 11 12 13
> x[c(1,4,6,5)]
[1] 10 13 15 14
```

Para excluir alguns numeros do vector usamos números negativos

```
> x<-10:22
> x
 [1] 10 11 12 13 14 15 16 17 18 19 20 21 22
> x[-2:-7]  # excluir do segundo até ao sétimo elemento
[1] 10 17 18 19 20 21 22
```

Selecção condicional- vectores

- Serve para extrair elementos que satisfaçam uma determinada condição

```
> dados <- c(5, 15, 42, 28, 79, 4, 7, 14)
>
> # seleccione elementos maiores do que  15
> dados[dados>15]
[1] 42 28 79
> # Seleccione os valores maiores do que 15
> # menores ou iguais a 35:
>
> dados[dados > 15 & dados <= 35]
[1] 28
```

Indexação- função `which()`

- Muitas vezes estamos interessados em saber a posição do resultado de uma expressão condicional, ao invés do resultado em si
- A fução `which()` retorna as posições dos elementos que retornarem `TRUE` em uma expressão condicional

```
> dados <- c(5, 15, 42, 28, 79, 4, 7, 14)
> dados[dados > 15]
[1] 42 28 79
> which(dados>15)
[1] 3 4 5
> dados[dados > 15 & dados <= 35]
[1] 28
> which(dados > 15 & dados <= 35)
[1] 4
```

Indexação de matrizes

Em estruturas de dados multidimensionais (por exemplo, matrizes e quadros de dados), um elemento na m-ésima linha, n-ésima coluna pode ser acessado pelo expressão `x[m, n]`

```
> mat <- matrix(1:9, nrow = 3,ncol=3)
> mat
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> #Acesse o valor que está na linha 2 da coluna 3
>
> mat[2,3]
[1] 8
```

Indexação de matrizes

A m-ésima linha inteira pode ser extraída pela expressão `x[m, ]`

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

A n-ésima coluna inteira pode ser extraída pela expressão `x[,n]`

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

Indexação de matriz

Pode-se também aceder a uma determinada linha e coluna em simultâneo

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

Indexação de listas

Considere a seguinte lista:

```
> lis <- list(c(3, 8, 7, 4), mat, 5:0)
> lis
[[1]]
[1] 3 8 7 4

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

[[3]]
[1] 5 4 3 2 1 0
```

Acessar o segundo elemento da lista

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

Indexação de listas

Aceder o segundo valor do primeiro componente da lista

```
> lis[[1]][2]
[1] 8
```

Listas nomeadas

- Componentes de uma lista nomeada podem ser acessados usando o operador $

```
> lis <- list(vetor1 = c(3, 8, 7, 4), mat = mat, vetor2 = 5:0)
> lis
$vetor1
[1] 3 8 7 4

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

$vetor2
[1] 5 4 3 2 1 0
> lis$mat
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

Indexação de listas

Alternativamente, podemos aceder a lista da seguinte forma:

```
> lis[["mat"]]
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

O símbolo $ é utilizado para acessar componentes nomeados de listas ou data frames

Indexação de data frame

Considere a seguinte data frame

```
> df1 <- data.frame(A = 4:1, B = c(2, NA, 5, 8))
> df1
  A  B
1 4  2
2 3 NA
3 2  5
4 1  8
```

Para acessar o segundo elemento da primeira coluna (seguindo a mesma lógica das matrizes, viste que tem duas dimensões)

```
> df1[2,1]
[1] 3
> # acesse todas as linhas da coluna B
> df1[,2]
[1]  2 NA  5  8
> # ou
>
> df1[,'B']
[1]  2 NA  5  8
> # ou
>
> df1$B
[1]  2 NA  5  8
```

Indexação de data frame

Aceder todos elementos da primeira linha

```
> df1[1,]
  A B
1 4 2
```

Selecção condicional de data frame

```
> dados <- data.frame(ano = c(2001, 2002, 2003, 2004, 2005),
+                     captura = c(26, 18, 25, 32, NA),
+                     porto = c("SP", "RS", "SC", "SC", "RN"))
>
> #Extraia deste objeto apenas a linha correspondente ao ano 2004:
>
> dados[dados$ano == 2004, ]
   ano captura porto
4 2004      32    SC
> #Mostre as linhas apenas do porto "SC":
>
> dados[dados$porto == "SC", ]
   ano captura porto
3 2003      25    SC
4 2004      32    SC
> #Observe as linhas onde a captura seja maior que 20, selecionando apenas a coluna captura
>
> dados[dados$captura > 20, "captura"]
[1] 26 25 32 NA
```

Programação Estatística

Introdução ao R -Importação de dados

Rachid Muleia, PhD in Statistics

Universidade Eduardo Mondlane
Faculdade de Ciências
Departamento de Matemática e Informática

2023-03-16

1 Importação & Exportação de dados

Importação & Exportação de dados

Objectivos da sessão

No final desta sessão deverá:

- Conhecer o formato ideal de uma `data frame`.
- Saber importar uma `data frame` para o R.
- Saber inspeccionar uma `data.frame`.
- Identificar anomalias na `data frame`.
- Exportar objectos em R para outro ambiente.

Passos para importação e exportação de `data frames`

1. Importar seus dados.
2. Inspeccionar, limpar e preparar os dados.
3. Fazer as análises.
4. Exportar os seus resultados
5. Limpar o ambiente R e fechar a sessão.

Como devem se apresentar os seus dados?

- As colunas devem representar variáveis.
- As linhas devem representar observações.
- Use a primeira linha para os nomes das variáveis.
- Todas as células em branco devem ser preenchidas com **NA**
- Armazene os dados no formato `.csv` e `.txt`, pois podem ser facilmente importados para o R.

NOTA IMPORTANTE: Todos os valores da mesma variável DEVEM estar na mesma coluna!

Exemplo: Tratamento para redução do peso.

Tratamento	Controlo	Placebo
65	70	80
70	80	86
55	90	78
68	100	98

Formato ideal de uma data frame.

Resposta	Método
65	Tratamento
70	Tratamento
55	Tratamento
68	Tratamento
70	Controlo
80	Controlo
90	Controlo
100	Controlo
80	Placebo
86	Placebo
78	Placebo
98	Placebo

Importação dados

Importe dados usando as funções `read.table()` ou `read.csv`

```
> meus_dados <- read.csv(file = 'MeusDados,txt')
> meus_dados <- read.csv(file = 'MeusDados.csv')
>
> # criam uma data frame meus_dados
```

Importação dados

Importe dados usando as funções `read.table()` ou `read.csv`

```
> meus_dados <- read.csv(file = 'MeusDados.csv')
> Error in file(file, "rt") : cannot open the connection
> In addition: Warning message:
> In file(file, "rt") :
>   cannot open file 'MeusDados.csv': No such file or directory
```

NOTA:

- Especificar o directório de trabalho usando a função `setwd()`.
- `setwd()` permite que o R saiba em que pasta está a base de dados.

Importação de dados - Argumentos

Veja a documentação da função `?read.table` e `?read.csv`

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
           row.names, col.names, as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = FALSE,
           fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
```

Importação de dados - Argumentos importantes

Algumas dicas para reduzir possíveis erros na importação de dados.

- `header = TRUE` - informa ao R que a primeira linha da base de dados representa as variáveis.
- `sep = ','` informa ao R que os campos são separados por vírgulas.
- `strip.white = TRUE` remove o espaço em branco antes ou após caracteres que foram inseridos por engano durante a digitação de dados (EX: “pequeno” vs "pequeno ").
- `na.strings = ''` permite substituir células vazias por `NA`.

Importação de dados

Primeiro especifique o directório de trabalho

```
setwd('C:/Users/Rachid/Dropbox/Analise de Dados DMI/Slides Aulas')
```

Agora vamos importar a base de dados `snail_feeding.csv` usando a função `read.csv()`

```
> snail_data <- read.csv(file = "Snail_feeding.csv",
+ header = TRUE, strip.white = TRUE,
+ na.strings = " ")
```

Inspeccionando os dados

Após a importação dos dados, podemos usar as funções `str()`, `head()` e `tail()` para inspeccionar os dados.

```
> str(snail_data)
'data.frame':   769 obs. of  10 variables:
 $ Snail.ID: int  1 1 1 1 1 1 1 1 1 1 ...
 $ Sex     : chr  "male" "male" "males" "male" ...
 $ Size    : chr  "small" "small" "small" "small" ...
 $ Feeding : logi  FALSE FALSE FALSE FALSE FALSE TRUE ...
 $ Distance: num  0.17 0.87 0.22 0.13 0.36 0.84 0.69 0.6 0.85 0.59 ...
 $ Depth   : num  1.66 1.26 1.43 1.46 1.21 1.56 1.62 162 1.96 1.93 ...
 $ Temp    : int  21 21 18 19 21 21 20 20 19 19 ...
 $ X       : logi  NA NA NA NA NA NA ...
 $ X.1     : logi  NA NA NA NA NA NA ...
 $ X.2     : logi  NA NA NA NA NA NA ...
```

Inspecção de dados

- No slide anterior o execução do comando `str` mostra-nos que temos 10 variáveis, o que não é verdade.
- As colunas indesejadas podem ser excluídas.

```
> snail_data <- snail_data[ , 1:7]
> str(snail_data)
'data.frame':   769 obs. of  7 variables:
 $ Snail.ID: int  1 1 1 1 1 1 1 1 1 1 ...
 $ Sex     : chr  "male" "male" "males" "male" ...
 $ Size    : chr  "small" "small" "small" "small" ...
 $ Feeding : logi  FALSE FALSE FALSE FALSE FALSE TRUE ...
 $ Distance: num  0.17 0.87 0.22 0.13 0.36 0.84 0.69 0.6 0.85 0.59 ...
 $ Depth   : num  1.66 1.26 1.43 1.46 1.21 1.56 1.62 162 1.96 1.93 ...
 $ Temp    : int  21 21 18 19 21 21 20 20 19 19 ...
```

Inspecção de dados

A execução do comando `str` mostra-nos que a variável `Sex` tem 4 categorias. Esquisito !!!

```
> unique(snail_data$Sex)
[1] "male"   "males"  "Male"   "female"
```

Pode se também usar as seguintes funções para fazer uma breve inspecção nos dados:

- `summary()` : estatísticas sumárias para cada variável
- `head()` : retorna as 6 primeiras linhas da base de dados
- `tail()` : retorna as seis últimas linhas da base de dados
- `names()`: para aceder aos nomes das variáveis

Inspecção de dados

Para transformar “males” ou “Male” em “male correto, pode usar o o operador `[ ]` acompanhado da função `which()`:

```
snail_data$Sex[which(snail_data$Sex == "males")]<- "male"

snail_data$Sex[which(snail_data$Sex == "Male")]<- "male"
```

Inspecção de dados

Para transformar “males” ou “Male” em “male correto, pode usar o o operador `[ ]` acompanhado da função `which()`:

```
snail_data$Sex[which(snail_data$Sex =="males")]<- "male"

snail_data$Sex[which(snail_data$Sex == "Male")]<- "male"
```

ou

```
snail_data$Sex[which(snail_data$Sex == "males" |
snail_data$Sex == "Male")] <- "male"
```

Inspecção de dados

Funções que podem ser úteis para preparar seus dados: `sort()` e `order()`

Ordenar os dados em ordem decrescente em função da variável `Temp`.

```
snail_data[order(snail_data$Depth, snail_data$Temp,decreasing=TRUE), ]
    Snail.ID    Sex  Size Feeding Distance  Depth Temp
8          1   male small    TRUE  0.60000 162.00   20
762       16 female large   FALSE  0.92000   2.00   21
412        9 female small    TRUE  0.48000   2.00   19
37         1   male small   FALSE  0.67000   2.00   18
155        4   male small   FALSE  0.38000   2.00   18
434       10 female small   FALSE  0.49000   2.00   18
644       14 female large   FALSE  0.79000   1.99   21
692       15 female large   FALSE  0.87000   1.99   21
217        5   male large   FALSE  0.86000   1.99   20
568       12 female small   FALSE  0.48000   1.99   20
675       15 female large   FALSE  0.08000   1.99   20
397        9 female small   FALSE  0.48000   1.99   19
116        3   male small   FALSE  0.24000   1.99   18
320        7   male large   FALSE  0.50000   1.99   18
411        9 female small    TRUE  0.45000   1.99   18
299        7   male large    TRUE  0.46000   1.98   21
749       16 female large    TRUE  0.98000   1.98   21
290        7   male large   FALSE  0.44000   1.98   20
345        8   male large    TRUE  0.37000   1.98   20
752       16 female large   FALSE  0.75000   1.98   18
```

Inspecção de dados

Indentificação e remoção de duplicados usando a função `duplicated()`

```
duplicated(snail_data)
  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [13] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [49] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [73] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [85] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [97] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[109] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[121] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[145] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[157] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[169] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[181] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[193] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[205] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[217] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[229] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[241] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[253] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[265] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[277] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[289] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[301] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Inspecção de dados

Para remover linhas duplicadas, pode usar o operador `[ ]` acompanhado a função `duplicated()`:

```
snail_data<- snail_data[!duplicated(snail_data), ]
```

OU

```
snail_data<- unique(snail_data)
```

Importação de dados armazenados na web

O R também pode ler dados armazenados usando uma `url`.

```
> data_web = read.table("http://www.sthda.com/upload/decathlon.txt",
+                       header=TRUE, sep='\t',strip.white = TRUE)
> head(data_web)
     name X100m Long.jump Shot.put High.jump X400m X110m.hurdle Discus
1  SEBRLE 11.04      7.58    14.83      2.07 49.81        14.69  43.75
2    CLAY 10.76      7.40    14.26      1.86 49.37        14.05  50.72
3  KARPOV 11.02      7.30    14.77      2.04 48.37        14.09  48.95
4 BERNARD 11.02      7.23    14.25      1.92 48.93        14.99  40.87
5  YURKOV 11.34      7.09    15.19      2.10 50.42        15.31  46.26
6 WARNERS 11.11      7.60    14.31      1.98 48.68        14.23  41.10
  Pole.vault Javeline X1500m Rank Points Competition
1       5.02    63.19  291.7    1   8217           1
2       4.92    60.15  301.5    2   8122           1
3       4.92    50.31  300.2    3   8099           1
4       5.32    62.77  280.1    4   8067           1
5       4.72    63.44  276.4    5   8036           1
6       4.92    51.77  278.1    6   8030           1
```

Importação de dados armazenados na web

- Ler dados a partir da web não é uma boa prática.
- O aconselhável é baixar os dados e ler localmente.
- O R possui uma função que permite baixar os dados, a função `download.file()`

```
download.file(url = "http://www.sthda.com/upload/decathlon.txt",destfile='decathlon')
```

- A função download pode baixar ficheiros com outras extensões.

Importação de dados no formato SPSS, SAS e STATA

- O R oferece recursos para ler ficheiros SPSS, SAS e STATA.
- Os recursos estão disponíveis na livraria `haven`. Como aceder??
- Precisamos instalar a livraria `haven` usando a função `install.packages()`.

Importação de dados no formato SPSS, SAS e STATA

- `read_sas()`: lê ficheiros `.sas7bdat` e `.sas7bcat` do SAS.
- `read_sav` ou `read_spss`: lê ficheiros `.sav` do SPSS.
- `read_dta`: lê ficheiros `.dta` do Stata.

```
> install.packages(haven)
> library(haven)
> dados_spss <- read_spss(file ='MZAR72FL.SAV')
```

Importação de dados no formato SPSS, SAS e STATA

Após ler os dados no formato spss, podemos aceder os nomes das variáveis.

```
> #install.packages(haven)
> library(haven)
> dados_spss <- read_spss(file ='MZAR72FL.SAV')
> attr(dados_spss$HIV03, 'label') # descrição da variável
[1] "Blood test result"
> attr(dados_spss$HIV03, 'labels') # labels dos valores assumido pela variável
                          HIV negative                           HIV  positive
                                     0                                      1
                         HIV2 positive                  HIV1 & HIV2 positive
                                     2                                      3
                   ERROR : V-, W+, M+                    ERROR : V-, W+, M-
                                     4                                      5
                   ERROR : V-, W-, M+                         Indeterminate
                                     6                                      7
Not enough samples to complete protocol                          Inconclusive
                                     8                                      9
```

Exportação de Dados

- O R permite exportar/gravar dados em vários formatos.
- Para exportação de dados usa-se a função `write.table()` ou `write.csv()`.

```
write.table(snail_data,  # Objecto que pretendemos exportar
            file='snail_inspecionado.txt', # nome do ficheiro
            sep=",", # separador de campos
           row.names=FALSE) # exlcuir os nomes das linhas.
```

- Para exportar dados no formato SPSS, SAS ou STATA usamos as funções, `write_sav()`, `write_sas()` e `write_dta()`, respectivamente.