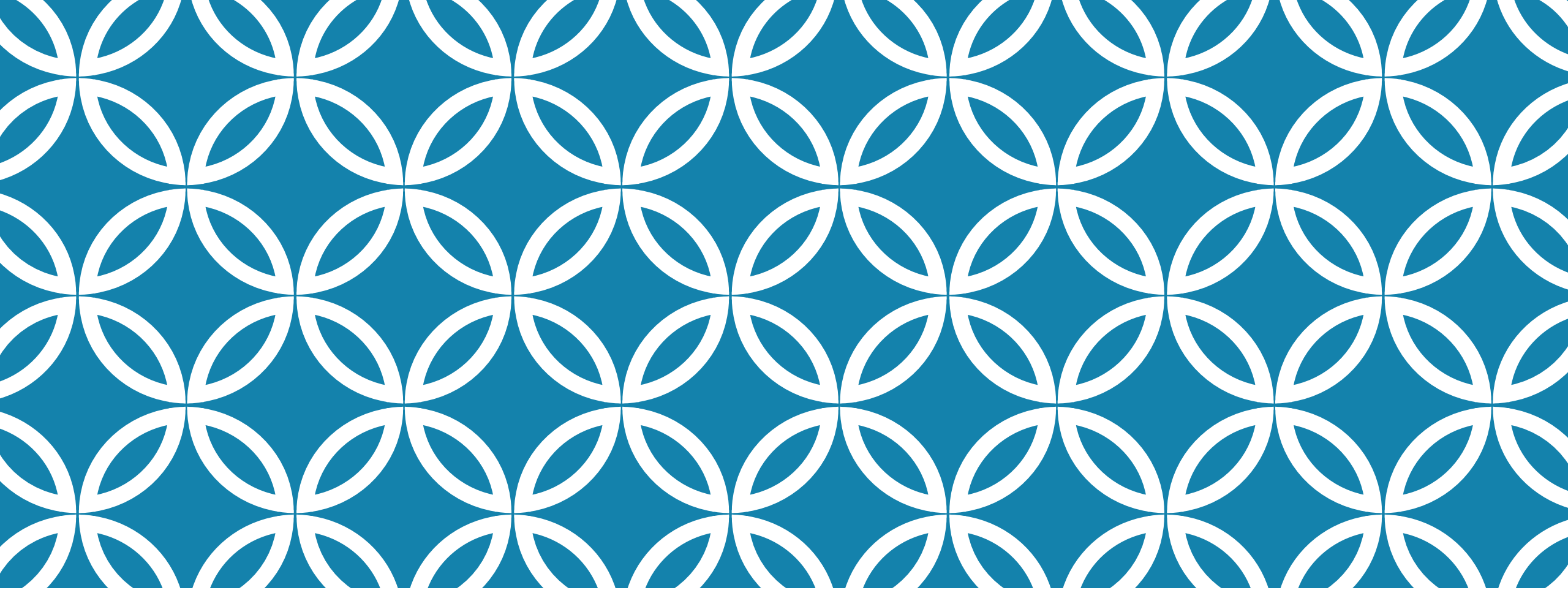


LAB 1 – INTRODUÇÃO A LINGUAGEM R E RSTUDIO

André Gustavo Adami
Daniel Luis Notari

PORQUE A LINGUAGEM R?

Linguagem de programação (orientada a objetos) grátis e *open-source*

- Grande legado no mundo da estatística (linguagens S e S+)

A disponibilidade de material e cursos na internet tem favorecido o aprendizado desta linguagem

A linguagem R oferece uma abstração maior dos detalhes da programação que facilitam o seu aprendizado

PORQUE A LINGUAGEM R?

Fornece uma ferramenta muito poderosa para manipulação de dados em memória: *data frames*. Uma estrutura de dados que permite organizar múltiplos tipos de dados em um único elemento

- Esta estrutura é muito importante para o desenvolvimento de sistemas de aprendizado de máquina (conjunto de dados)

Disponibilidade de diversos pacotes (comunidade) que facilitam o processo de análise/visualização de dados e desenvolvimento de modelos para o aprendizado de máquina

Possui diversas IDEs, mas a mais popular é o R Studio

PORQUE A LINGUAGEM R?

Mas nem tudo é alegria...

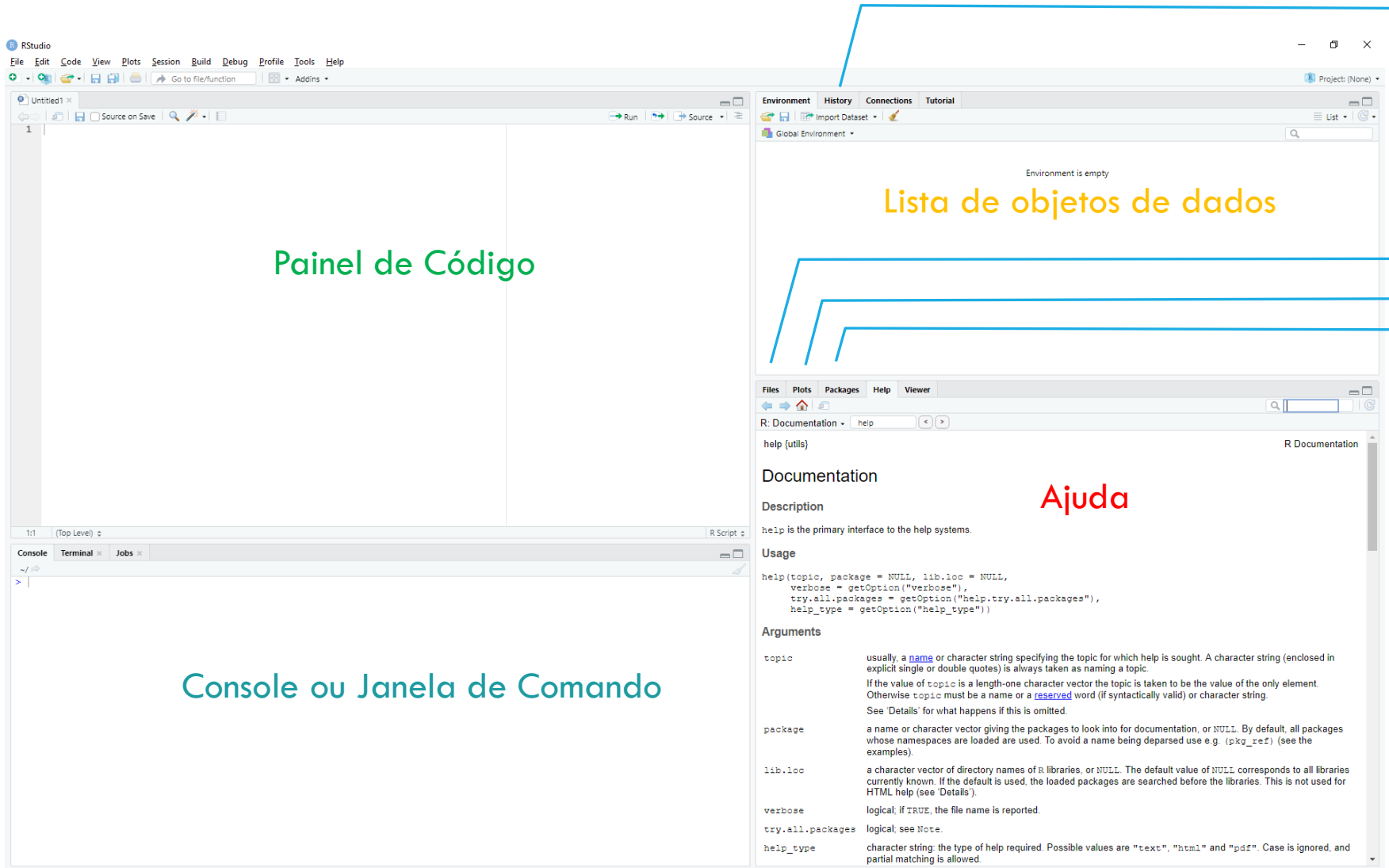
Os pacotes oferecidos por terceiros ou comunidades podem ter alguma inconsistência entre eles e outras ferramentas

- É importante conhecer maiores detalhes sobre os pacotes (algoritmos, suposições, objetivo)

A documentação nem sempre é farta e compreensível (sendo até inconsistente)

E sendo uma linguagem de programação, determinadas características podem ser estranhas para um programador Em algumas situações

RSTUDIO


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Untitled1
Source on Save
Run
Source
1
1:1
(Top Level)
R Script
Console
Terminal
Jobs
~/
>
Environment
History
Connections
Tutorial
Import Dataset
List
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
R: Documentation
help
help {utils}
R Documentation
Documentation
Description
help is the primary interface to the help systems.
Usage
help(topic, package = NULL, lib.loc = NULL,
verbose = getOption("verbose"),
try.all.packages = getOption("help.try.all.packages"),
help_type = getOption("help_type"))
Arguments
topic
usually, a name or character string specifying the topic for which help is sought. A character string (enclosed in explicit single or double quotes) is always taken as naming a topic.
If the value of topic is a length-one character vector the topic is taken to be the value of the only element. Otherwise topic must be a name or a reserved word (if syntactically valid) or character string.
See 'Details' for what happens if this is omitted.
package
a name or character vector giving the packages to look into for documentation, or NULL. By default, all packages whose namespaces are loaded are used. To avoid a name being deparsed use e.g. (pkg_ref) (see the examples).
lib.loc
a character vector of directory names of R libraries, or NULL. The default value of NULL corresponds to all libraries currently known. If the default is used, the loaded packages are searched before the libraries. This is not used for HTML help (see 'Details').
verbose
logical; if TRUE, the file name is reported.
try.all.packages
logical; see Note.
help_type
character string: the type of help required. Possible values are "text", "html" and "pdf". Case is ignored, and partial matching is allowed.
Painel de Código
Lista de objetos de dados
Histórico dos Comandos
Gerenciador de Arquivos/Pastas
Ajuda
Painel de Gráficos
Console ou Janela de Comando
Gerenciador de Pacotes

RSTUDIO – PACOTES

O CRAN (cran.r-project.org) é um repositório público de pacotes (conjunto de funções e dados)

Se você não sabe o nome do pacote

- É possível buscar um pacote pela descrição no painel de gerenciamento de pacotes

Para atualizar os pacotes

- No console, utilize a função update.packages()
- Clique no menu **Tools**, em seguida selecione a opção **Check for Package Updates**.

Se você sabe o nome do pacote

- Clique no menu **Tools**, em seguida selecione a opção **Install Packages**.

- No console, deve-se utilizar a função **install.packages()** (a opção **dependencies** define que deve ser instalada qualquer outro pacote que seja necessário)

```
install.packages("caret",
                 dependencies = TRUE)
```

Para carregar um pacote

```
library(caret)
```

RSTUDIO BÁSICO

Para uma melhor organização dos scripts e dados, recomenda-se criar um diretório de trabalho para cada tarefa

- getwd() retorna o diretório de trabalho atual

Para definir o diretório de trabalho como o seu diretório atual

```
setwd("C:/Users/Adami/Documents/MachineLearning")
setwd("C:\\Users\\Adami\\Documents\\MachineLearning")
```

No caso de ter uma dúvida do funcionamento de uma função, utilize o caractere "?" antes da função (a ajuda vai aparecer na aba Help)

```
?getwd
```

CONSTANTES

Numéricas (integer, double e complex)

```
> typeof(5)
[1] "double"
> typeof(5L)
[1] "integer"
> typeof(5i)
[1] "complex“
```

Lógica (TRUE e T ou FALSE e F)

```
> typeof(TRUE)
[1] "logical“
> typeof(F)
[1] "logical"
```

Caracteres (delimitadas por ‘ ou “)

```
> "string"
[1] "string"
> 'a'
[1] "a"
> typeof('a')
[1] "character"
> typeof("string")
[1] "character"
```

OPERADORES

Operadores Aritméticos	
+	adição
-	subtração
*	multiplicação
/	divisão
^ ou **	exponenciação
x %% y	Resto da divisão inteira x / y
x %/% y	Divisão inteira de x por y

Operadores Lógicos Vetores	
x | y	x ou-lógico y
x & y	x e-lógico y

Operadores Relacionais	
<	Menor que
<=	Menor ou igual que
>	Maior que
>=	Maior ou igual que
==	Igual
!=	Diferente

Operadores Lógicos	
!x	Inverso de x
x || y	x ou-lógico y
x && y	x e-lógico y

Operador de Atribuição	
=	`x = 1`
<-	`str <- "string"`
<<-	`flag <<- TRUE`

Operador de Atribuição	
:	Cria uma série de números inteiros em sequência para um vetor `vet = 6:9 # cria um vetor de iteiros com os valores 6, 7, 8 e 9`

ESTRUTURAS DE DADOS: VETOR

Vetor contém elementos do mesmo tipo (`character`, `logical`, `double` e `integer`)

```
> vetor = c(1*pi, 2*pi, 3*pi, 4*pi)
> print(vetor)
[1]  3.141593  6.283185  9.424778 12.566371
> typeof(vetor)
[1] "double“
```

No caso de combinar tipos, a função `c()` converterá em um único tipo

```
> vetor <- c(1, 2, 3, "a", "b", "c")
> vetor
[1] "1" "2" "3" "a" "b" "c"
```

ESTRUTURAS DE DADOS: VETOR

Criar um vetor usando uma sequência

```
> vetor = 1:8
> print(vetor)
 [1]  1  2  3  4  5  6  7  8

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

Obter o tamanho de um vetor

```
> vetor = 1:8
> length(vetor)
[1] 8
```

Acessar um elemento (índice começa em 1)

```
> vetor = 1:8
> print(vetor[3])
[1] 3
> print(vetor[6:8])
[1] 6 7 8
```

Omitir posições (índices negativos)

```
> print(vetor[-6:-8])
[1] 1 2 3 4 5
```

ESTRUTURAS DE DADOS: VETOR

É possível selecionar elementos de um vetor utilizando expressões condicionais (utiliza indexação lógica)

- Recomendado para situações onde o tamanho dos vetores (condição e dados) são iguais

```
> vetor = 5:10
> vetor[vetor>6 & vetor<9]
[1] 7 8
> vetor>6 & vetor<9
[1] FALSE FALSE  TRUE  TRUE FALSE
FALSE
```

Note que o operador lógico utilizado é `&` (vetores)

No caso de selecionar elementos utilizando índices, é possível utilizar a função `which()`, que retornar os índices que satisfazem a condição passada como parâmetro

```
> vetor = 5:10
> vetor[which(vetor>6 & vetor<9)]
[1] 7 8
> which(vetor>6 & vetor<9)
[1] 3 4
```

ESTRUTURAS DE DADOS: VETOR

Dois vetores de mesmo tamanho podem ser somados, subtraídos, multiplicados e divididos

```
> v1 = 2:4
> v2 = 5:7
> v1+v2
[1]  7  9 11
```

Caso os vetores tiverem tamanhos diferentes (mas o tamanho do maior deve ser múltiplo do tamanho do menor), o menor vetor será repetido até igualar o tamanho do maior

```
> v1 = 1:6
> v2 = 3:4
> v1+v2
[1]  4  6  6  8  8 10
```

Para ordenar um vetor, deve-se utilizar a função `sort()`

```
> sort(c(2,5,3,7))
[1] 2 3 5 7
> sort(c(2,5,3,7), decreasing = T)
[1] 7 5 3 2
```

Para acrescentar (append) um elemento no vetor, basta atribuir um valor no respectivo índice (caso o índice não for o próximo, o R insere NA no vetor)

```
> v = 1:6
> v[7] = 9
> v
[1] 1 2 3 4 5 6 9
```

ESTRUTURAS DE DADOS: LISTA

Diferente de vetores, **listas** podem armazenar elementos de tipos heterogêneos

```
> lista = list(1, "abc", 1.23, TRUE)
> str(lista)
List of 4
 $ : num 1
 $ : chr "abc"
 $ : num 1.23
 $ : logi TRUE
```

A função `str()` fornece uma apresentação mais compacta da estrutura interna de qualquer objeto R

Duas formas de acessar os elementos

- [] : retorna uma lista

```
> lista[1]
[[1]]
[1] 1
> typeof(lista[1])
[1] "list"
```

- [[]] : retorna o elemento da lista

```
> lista[[1]]
[1] 1
> typeof(lista[[1]])
[1] "double"
```

As mesmas operações de acessar os elementos que nem vetor podem ser utilizadas

ESTRUTURAS DE DADOS: FACTORS

Existem informações do mundo real que podem ser categorizadas

- Os rótulos/classes de um problema de aprendizado de máquina são um exemplo

Tais categorias se repetem e por isso podem ser armazenadas de maneira compacta utilizando a estrutura de dados **factors**

Um fator (fator) é uma lista ordenada de itens, onde os valores são chamados de níveis (levels)

```
> classes = c("C1", "C2", "C3",
        "C2", "C1", "C3",
"C2", "C1", "C2")
> typeof(classes)
[1] "character"
> print(classes)
[1] "C1" "C2" "C3" "C2" "C1" "C3"
"C2" "C1" "C2"

> classes = factor(classes)
> typeof(classes)
[1] "integer"
> print(classes)
[1] C1 C2 C3 C2 C1 C3 C2 C1 C2
Levels: C1 C2 C3
```

ESTRUTURAS DE DADOS: FACTORS

Em algumas situações, é possível que não exista todas as categorias (então a conversão automática não é recomendada)

Na geração dos fatores, pode-se definir os níveis existentes

```
> classes = factor(classes,
+          levels=c("C1", "C2", "C3", "C4"))
> print(classes)
[1] C1 C2 C3 C2 C1 C3 C2 C1 C2
Levels: C1 C2 C3 C4
```

Para definir o rótulos dos níveis

```
> classes = factor(classes,
+       levels=c("C1", "C2", "C3", "C4"),
+       labels=c("1", "2", "3", "4"))
> print(classes)
[1] 1 2 3 2 1 3 2 1 2
Levels: 1 2 3 4
```

O acesso a um elemento é similar a um vetor

```
> classes = factor(c("C1", "C2", "C3","C2", "C1", "C3", "C2", "C1", "C2"))
> classes[1]
[1] C1
Levels: C1 C2 C3
> classes[1:3]
[1] C1 C2 C3
Levels: C1 C2 C3
```

ESTRUTURAS DE DADOS: FACTORS

Alterando um factor

```
> classes
[1] C1 C2 C3 C2 C1 C3 C2 C1 C2
Levels: C1 C2 C3

> classes[1] = "C2"
> classes
[1] C2 C2 C3 C2 C1 C3 C2 C1 C2
Levels: C1 C2 C3
```

Identificar os níveis de um fator

```
> levels(classes)
[1] "C1" "C2" "C3"
```

Como inspecionar os conteúdos de um factor

```
> summary(classes)
C1 C2 C3
 3  4  2
> table(classes)
classes
C1 C2 C3
 3  4  2
```

ESTRUTURAS DE DADOS: DATA FRAME

Uma estrutura que vem facilitar a manipulação de conjunto de dados heterogêneos é o **data frame**

- O data frame é semelhante a uma lista de vetores de mesmo tamanho

```
> data("iris") # carrega os dados iris
> str(iris)
'data.frame': 150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Cada linha é chamada de observação (medição) e as colunas são as variáveis

ESTRUTURAS DE DADOS: DATA FRAME

Para obter os nomes das colunas, deve-se utilizar a função names()

```
> names(iris)
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

Informações acerca das dimensões da estrutura

```
> ncol(iris)
[1] 5
> nrow(iris)
[1] 150
> dim(iris)
[1] 150   5
```

ESTRUTURAS DE DADOS: DATA FRAME

Para acessar um elemento, o data frame deve ser acessado como se fosse uma matriz

```
> x = iris[1,1]
> typeof(x)
[1] "double“
> class(x)
[1] "numeric"
```

ou pelo nome da coluna (retorna um vetor)

```
> x = iris$Sepal.Length
> class(x)
[1] "numeric"
> typeof(x)
[1] "double"
```

Acessar como se fosse uma lista

```
> x = iris["Sepal.Width"]
> typeof(x)
[1] "list"
> class(x)
[1] "data.frame"
```

```
> x = iris[["Sepal.Width"]]
> typeof(x)
[1] "double"
> class(x)
[1] "numeric"
```

ESTRUTURAS DE DADOS: DATA FRAME

Acessar um subconjunto do data frame

```
> sub = iris[1:2,]
> typeof(sub)
[1] "list"
> class(sub)
[1] "data.frame"
```

Selecionar linhas com base em uma condição

```
> sub = iris[iris$Sepal.Width<3.0,]
> typeof(sub)
[1] "list"
> class(sub)
[1] "data.frame"
```

Remover colunas (para remover linhas, deve-se realizar a operação na dimensão das linhas)

```
> dim(iris)
[1] 150   5
> sub = iris[,-1] # Remove a coluna 1
> dim(sub)
[1] 150   4
```

Ou remover no próprio data frame

```
> data(iris)
> iris$Sepal.Length = NULL
> ncol(iris)
[1] 4
```

ESTRUTURAS DE DADOS: DATA FRAME

Adicionar uma coluna

```
> iris$novacoluna = iris$Sepal.Width + iris$Petal.Length
> str(iris)
'data.frame': 150 obs. of  6 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ novacoluna  : num  8.6 7.9 7.9 7.7 8.6 9.3 8 8.4 7.3 8 ...
```

ESTRUTURAS DE CONTROLE: IF ELSE

```
if (condição) {
  // bloco de comandos
} else {
 // bloco de comandos
}
```

A condição deve produzir um valor booleano (`TRUE` ou `FALSE`)

O `else` é opcional

```
if (x>y) {
  print("x é maior que y")
}
```

```
if (x>5) {
  print("x é maior que 5")
} else {
  print("x é menor ou igual a 5")
}
```

ESTRUTURAS DE CONTROLE: LAÇOS

```
while (condição) {
 // bloco de comandos
}
```

```
x = 1
while (x<10) {
  print(x)
  x = x + 2
}
```

```
for (variável in vetor) {
// bloco de comandos
}
```

```
for (i in 1:6) {
  print(i)
}
```

```
vetor = c(2, 4, 6, 8)
for(elemento in vetor) {
  print(elemento)
}
```

TAREFA

Para investigar se a infertilidade secundária feminina estaria associada com abortos prévios, um estudo caso-controle com mulheres com e sem fertilidade foi realizado. As variáveis consideradas foram: idade, número de gestações (incluindo abortos) e anos de escolaridade

Responda as perguntas a seguir

TAREFA

1. Carregue os dados (`data("infert")`) e mostre a sua estrutura.
2. Quantas variáveis o data frame possui? Mostre o nome delas.
3. Quantas observações o data frame possui?
4. Uma das variáveis define o status do caso, isto é, se os dados referem-se a uma mulher com infertilidade (status = 1) ou sem infertilidade (status = 0). Como esta variável é categórica, transforme-a em *factor*, com os rótulos "Caso" para status=1 e "Controle" para status=0. Em seguida, mostre a quantidade de observações por nível do *factor* gerado.
5. Quantas mulheres possuem 12 anos ou mais de educação (*education*)?
6. Quantas mulheres que possuem 12 anos ou mais de educação (*education*) e são do grupo de controle (case) tiveram aborto espontâneo (*spontaneous*)?