



Introdução à datascience com R

Cleuton Sampaio

Lição 6: Datasets externos

Datasets externos

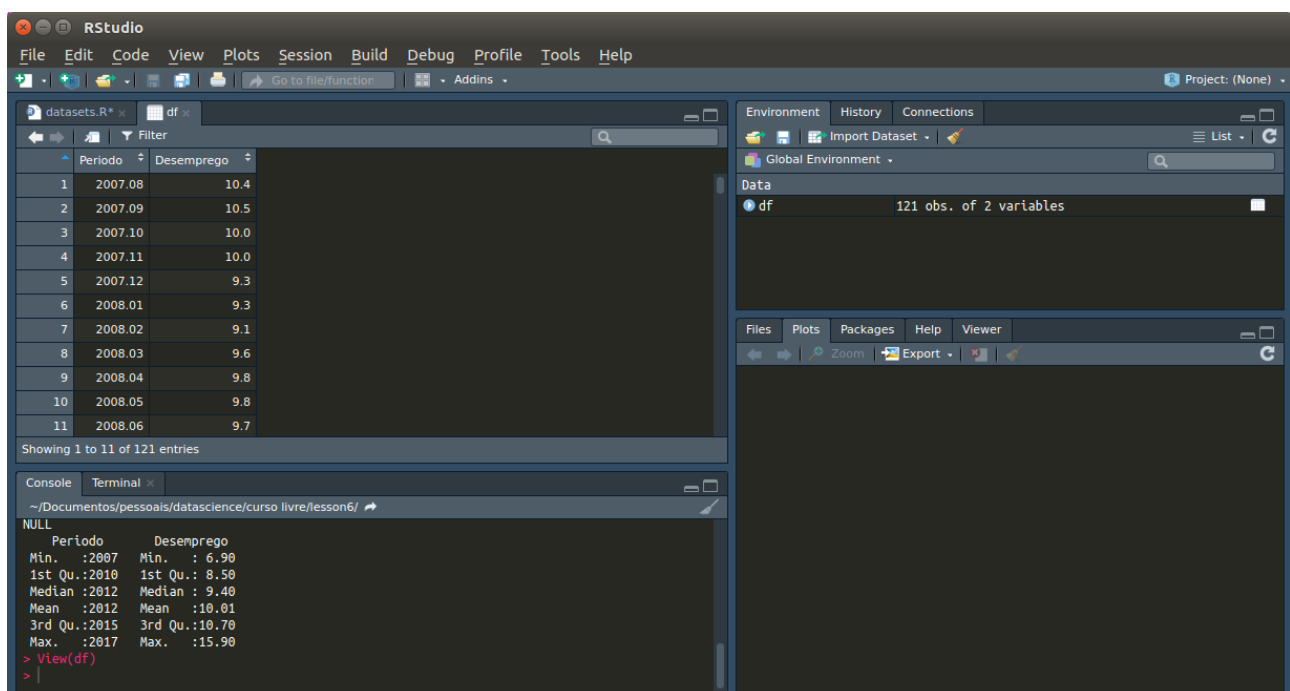
Quase sempre precisaremos ler datasets, como fizemos na primeira aula do curso. A maioria destes datasets está em arquivos CSV – Comma Separated Values:

```
<registro 1, campo 1>,<registro 1, campo 2>,<registro 1, campo 3><CR/LF>
<registro 2, campo 1>,<registro 2, campo 2>,<registro 2, campo 3><CR/LF>
...
<registro n, campo 1>,<registro n, campo 2>,<registro n, campo 3><CR/LF>
```

Você deverá baixar o conteúdo da pasta "datasets" do nosso repositório e colocar na pasta de trabalho do RStudio (menu "Session / Set working directory / Choose directory"). Digite o comando abaixo na console (ou então em um Script):

```
df <- read.csv('desemprego.csv')
```

A partir da execução, no painel "Environment" você verá uma variável "df". Clique nela e o conteúdo do arquivo CSV aparecerá no lugar da tela de edição de código, como na figura abaixo:



Esta função "read.csv()" lê um arquivo CSV, cujo caminho passamos como parâmetro, e retorna uma variável "Data Frame", que possui várias funções para manipulação de

dados. O Data frame contém colunas e linhas. Podemos ver o tipo de dados das colunas com a função "str()" e podemos ver a estatística básica com "summary()":

```
print(str(df))
```

Eis a estrutura deste dataframe:

```
'data.frame': 121 obs. of 2 variables:
 $ Periodo : num 2007 2007 2007 2007 2007 ...
 $ Desemprego: num 10.4 10.5 10 10 9.3 9.3 9.1 9.6 9.8 9.8 ...
```

```
summary(df)
```

Em R podemos ter variáveis: Inteiras, Numéricas, Caracteres, Lógicas, entre outras. As variáveis lógicas só podem conter TRUE ou FALSE (assim mesmo, em maiúsculas e sem aspas).

Eis a estatística deste dataframe:

Periodo	Desemprego
Min. :2007	Min. : 6.90
1st Qu.:2010	1st Qu.: 8.50
Median :2012	Median : 9.40
Mean :2012	Mean :10.01
3rd Qu.:2015	3rd Qu.:10.70
Max. :2017	Max. :15.90

Um dataframe pode conter muitas linhas. Podemos visualizá-lo pelo início ou fim usando as funções "head()" e "tail()". Elas mostram as 6 primeiras ou 6 últimas linhas:

```
print(head(df))
print(tail(df))
```

	Periodo	Desemprego
1	2007.08	10.4
2	2007.09	10.5
3	2007.10	10.0
4	2007.11	10.0
5	2007.12	9.3
6	2008.01	9.3

Datasets não convencionais

Estamos no Brasil, e é nós usamos vírgulas como separador de decimais, por isto, quando exportamos dados do Excel ou do LibreOffice, usamos o ponto e vírgula como separador de colunas. Como faremos?

```
Nome;Pesos;Alturas
"Fulano";74;1,73
"Beltrano";61;1,61
"Cicrano";61;1,61
```

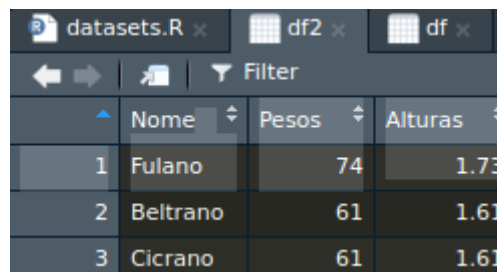
Basta passar alguns parâmetros extras para a função "read.csv()":

- dec : Carácter separador de decimais;
- sep : Carácter separador de colunas.

Por exemplo, para ler o dataset "dataset-nao-convencional.csv", que está na pasta, usamos:

```
df2 <- read.csv('dataset-nao-convencional.csv',dec = ',', sep = ";")
```

(Tanto faz usarmos aspas simples ou duplas nos parâmetros, desde que sejamos coerentes: Abriu aspas simples, tem que fechar aspas simples)



	Nome	Pesos	Alturas
1	Fulano	74	1.73
2	Beltrano	61	1.61
3	Cicrano	61	1.61

Classes das colunas

Podemos especificar as classes, ou domínios das colunas, com o parâmetro "colClasses". Isto pode ajudar a economizar memória em datasets muito grandes:

```
df3 <- read.csv('dataset-nao-convencional.csv',dec = ',', sep = ";",
  colClasses= c('character','integer','numeric'))
```

Lidando com datas

Se tivermos um dataset assim:

```
Nascimento,Pesos,Alturas
"2007/3/5",74,1.73
"2007/1/1",61,1.62
"2007/5/25",61,1.63
"2007/1/15",68,1.68
"2007/2/3",70,1.68
```

Podemos carregá-lo com o comando:

```
df4 <- read.csv('datas.csv',colClasses=c('Date','integer','numeric'))
```

Podemos extrair pedaços da data utilizando a função "format":

```
print(as.numeric(format(df4$Nascimento, "%m")))
```

Neste exemplo, pegamos apenas a coluna "Nascimento", do dataframe df4, e formatamos o mês ("%m") como numérico.

Datas com formatos diferenciados

Notou que alguns arquivos, especialmente os oriundos do IPEA, possuem colunas no formato: ano.mês? Como lemos isso? Uma opção é ler como caracter:

```
df5 <- read.csv('desemprego.csv', colClasses = c('character', 'numeric'))
```

```
'data.frame': 121 obs. of 2 variables:
 $ Período : chr "2007.08" "2007.09" "2007.10" "2007.11" ...
 $ Desemprego: num 10.4 10.5 10 10 9.3 9.3 9.1 9.6 9.8 9.8 ...
```

Não tem como converter diretamente para data. Dá para separar em duas colunas, o que veremos posteriormente.

Vamos ler o dataset "datas-horas.csv":

```
Nascimento, Pesos, Alturas
"5-3-2005 17:35:00", 5, 0.50
"1-5-2005 12:10:05", 3.5, 0.48
"10-6-2005 23:01:12", 4.1, 0.51
```

```
df6 <- read.csv('datas-horas.csv', colClasses = c('character', 'numeric',
'numeric'))
str(df6)
```

```
'data.frame': 3 obs. of 3 variables:
 $ Nascimento: chr "5-3-2005 17:35:00" "1-5-2005 12:10:05" "10-6-2005 23:01:12"
 $ Pesos : num 5 3.5 4.1
 $ Alturas : num 0.5 0.48 0.51
```

```
df6$Nascimento <- strptime(df6$Nascimento, format='%d-%m-%Y %H:%M:%S')
str(df6)
```

```
'data.frame': 3 obs. of 3 variables:
 $ Nascimento: POSIXlt, format: "2005-03-05 17:35:00" "2005-05-01 12:10:05" ...
 $ Pesos : num 5 3.5 4.1
 $ Alturas : num 0.5 0.48 0.51
```

Nós converteremos a coluna "Nascimento" de carácter para data/hora, utilizando a função "strptime()". Note que usamos uma sintaxe para pegar os valores da coluna: "df6\$Nascimento", e atribuímos à ela mesma.

Lidando com nulos e lacunas nos dados

Nulos podem causar problemas em trabalhos de datascience, logo, precisamos lidar com eles.

Em R temos alguns tipos de valores inválidos:

- Inf: "infinity" ou infinito, é o resultado da divisão de qualquer número (exceto zero) por zero. Pode ser negativo ou positivo;
- NA: O valor não existe;
- NULL: O valor é nulo.

Veja só esses exercícios na Console R:

```
> 1/0
[1] Inf
> 0/0
[1] NaN
> print(xpto)
Error in print(xpto) : objeto 'xpto' não encontrado
> xpto <- NA
> xpto
[1] NA
> xpto2 <- NULL
> xpto2
NULL
> null
Erro: objeto 'null' não encontrado
> NULL
NULL
```

Dividir qualquer número por zero resulta em Inf, porém, dividir zero por zero resulta em NaN (Not a Number).

Se um objeto não foi declarado, seu uso resultará em um erro ("objeto não encontrado"). Agora, se ele existir, mas não contiver valor, ele conterá NA (Not Available). O NULL é um valor e podemos até atribuí-lo a um objeto, mas tem que ser em maiúsculas.

O que acontece se um dataset contiver valores faltando?

O R te protege disto. Veja os exemplos abaixo:

a) Deixar um valor faltando (uma lacuna):

```
> valores <- c(1.0,5.2,,6.7,3.2,4.1)
Error in c(1, 5.2, , 6.7, 3.2, 4.1) : argumento 3 está vazio
> valores
Erro: objeto 'valores' não encontrado
```

b) Declarar um valor como NA:

```
> valores <- c(1.0,5.2,NA,6.7,3.2,4.1)
```

```
> valores
[1] 1.0 5.2 NA 6.7 3.2 4.1
> mean(valores)
[1] NA
```

Ele aceita declarar, mas qualquer cálculo resultará em NA.

c) *Declarar um valor como NAN:*

```
> valores <- c(1.0,5.2,NaN,6.7,3.2,4.1)
> valores
[1] 1.0 5.2 NaN 6.7 3.2 4.1
> mean(valores)
[1] NaN
```

d) *Declarar um valor como Inf:*

```
> valores <- c(1.0,5.2,Inf,6.7,3.2,4.1)
> valores
[1] 1.0 5.2 Inf 6.7 3.2 4.1
> mean(valores)
[1] Inf
```

e) *Declarar como NULL:*

```
> valores <- c(1.0,5.2,NULL,6.7,3.2,4.1)
> valores
[1] 1.0 5.2 6.7 3.2 4.1
> mean(valores)
[1] 4.04
> length(valores)
[1] 5
```

Ele parece ignorar os valores.

Como lidamos com isto? Uma opção é usar as funções "is.na()" e "is.infinite()":

```
> valores <- c(1.0,5.2,NA,6.7,3.2,4.1)
> is.na(valores)
[1] FALSE FALSE TRUE FALSE FALSE FALSE
> is.na(valores[3])
[1] TRUE
> valores <- c(1.0,5.2,NaN,6.7,3.2,4.1)
> is.na(valores)
[1] FALSE FALSE TRUE FALSE FALSE FALSE
> valores <- c(1.0,5.2,Inf,6.7,3.2,4.1)
> is.na(valores)
[1] FALSE FALSE FALSE FALSE FALSE FALSE
> is.infinite(valores)
[1] FALSE FALSE TRUE FALSE FALSE FALSE
> valores <- c(1.0,5.2,NULL,6.7,3.2,4.1)
> is.na(valores)
[1] FALSE FALSE FALSE FALSE FALSE
```

Notou o problema com Inf? A função "is.na()" o considera presente! Então, temos que usar "is.infinite()". E o NULL não é detectado.

Se um dataset está com valores faltando, algumas funções possuem opções para lidar com isso, por exemplo a "mean()":

```
> valores <- c(1.0,5.2,NA,6.7,3.2,4.1)
> mean(valores,na.rm = TRUE)
[1] 4.04
```

O parâmetro "na.rm = TRUE" simplesmente remove o valor NA. Isso funciona para desvio padrão e para Summary:

```
> valores <- c(1.0,5.2,NA,6.7,3.2,4.1)
> mean(valores,na.rm = TRUE)
[1] 4.04
> sd(valores,na.rm = TRUE)
[1] 2.143129
> summary(valores)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  1.00   3.20   4.10   4.04   5.20   6.70     1
```

A função "summary()" já lida com NA e exibe sua quantidade, como você pode ver.

Outras maneiras de lidar são:

1. Substituir o valor faltante pela média dos valores;
2. Remover o valor faltante.

```
> # Substituir pela média:
> valores[is.na(valores)] <- mean(valores, na.rm = TRUE)
> valores
[1] 1.00 5.20 4.04 6.70 3.20 4.10
```

Selecionamos os elementos para os quais a função "is.na()" retorna TRUE, e lhes atribuímos a média (calculada sem contá-los).

Podemos criar uma cópia do datase, na qual removemos os valores faltantes. A função "complete.cases()" pode nos ajudar:

```
> valores <- c(1.0,5.2,NA,6.7,3.2,4.1)
> complete.cases(valores)
[1] TRUE TRUE FALSE TRUE TRUE TRUE
> subval <- valores[complete.cases(valores)]
Erro: não foi possível encontrar a função "valores"
> subval <- valores[complete.cases(valores)]
> subval
[1] 1.0 5.2 6.7 3.2 4.1
```

Selecionamos de valores apenas aqueles em que o resultado da função "complete.cases()" é TRUE!

Agora, com Dataframes

É praticamente a mesma coisa:

```
> a = c(1,2,3)
> b = c("T1", NA, "T3")
> c = c(FALSE, FALSE, TRUE)
> df = data.frame(a,b,c)
>
> df
  a    b    c
1 1  T1 FALSE
2 2 <NA> FALSE
3 3  T3  TRUE
> is.na(df)
      a      b      c
[1,] FALSE FALSE FALSE
[2,] FALSE  TRUE FALSE
[3,] FALSE FALSE FALSE
```

Criamos um dataframe a partir de 3 vetores. Cada vetor virará uma coluna e o seu nome será o nome do próprio vetor. Incluímos propositadamente um valor NA, na segunda linha, segunda coluna.

Substituindo pela média:

```
> a = c(4,5,NA,7,8,9)
> b = c("T1", "T2", "T3", "T4", "T5", "T6")
> c = c(FALSE, FALSE, TRUE, TRUE, TRUE, FALSE)
> df = data.frame(a,b,c)
>
> df
  a    b    c
1 4 T1 FALSE
2 5 T2 FALSE
3 NA T3  TRUE
4 7 T4  TRUE
5 8 T5  TRUE
6 9 T6 FALSE
> df[is.na(df$a),1] <- mean(df$a, na.rm = TRUE)
> df
  a    b    c
1 4.0 T1 FALSE
2 5.0 T2 FALSE
3 6.6 T3  TRUE
4 7.0 T4  TRUE
5 8.0 T5  TRUE
6 9.0 T6 FALSE
```

Pegamos o elemento da coluna 1, de todas as linhas, onde esteja faltando valor e substituímos pela média da coluna "a" (descontando os valores faltantes).

Podemos fazer isso com qualquer dataframe, inclusive aqueles lidos de arquivos CSV. Suponha o arquivo "faltando-valor.csv", que está na nossa pasta:

```
Periodo, Inflacao
2007.09, 0.25
2007.10, 0.3
2007.11,
2007.12, 0.97
2008.01, 0.69
```

Na terceira linha do arquivo, depois do cabeçalho, está faltando o valor da Inflação. Podemos lidar com isso da mesma forma:

```
> df <- read.csv('faltando-valor.csv')
> head(df)
  Periodo Inflacao
1 2007.09    0.25
2 2007.10    0.30
3 2007.11     NA
4 2007.12    0.97
5 2008.01    0.69
6 2008.02    0.48
> summary(df)
  Periodo      Inflacao
Min.   :2007   Min.   :-0.3000
1st Qu.:2010   1st Qu.: 0.2575
Median :2012   Median : 0.5100
Mean   :2012   Mean   : 0.4998
3rd Qu.:2015   3rd Qu.: 0.6675
Max.   :2017   Max.   : 1.5100
      NA's      :1
> df$Inflacao[is.na(df$Inflacao)] <- mean(df$Inflacao, na.rm = TRUE)
> head(df)
  Periodo Inflacao
1 2007.09 0.25000
2 2007.10 0.30000
3 2007.11 0.49975
4 2007.12 0.97000
5 2008.01 0.69000
6 2008.02 0.48000
```

Finalmente, uma opção é retirar as linhas que contenham NA:

```
> df <- na.omit(read.csv('faltando-valor.csv'))
> head(df)
  Periodo Inflacao
1 2007.09    0.25
2 2007.10    0.30
4 2007.12    0.97
5 2008.01    0.69
6 2008.02    0.48
7 2008.03    0.51
```

Análise do script da aula

O script desta aula é: "datasets.R" e está na pasta:

<https://github.com/cleuton/datascience/tree/master/R-course/lesson6>

Lendo e processando arquivos CSV. Vamos ler o seguinte arquivo, que está na mesma pasta:

```
Periodo,Desemprego
2007.08,10.4
2007.09,10.5
2007.10,10
2007.11,10
2007.12,9.3
...
```

```
df <- read.csv('desemprego.csv')

# Estrutura:
str(df)

# Estatística:
print(summary(df))

# Primeiras 6 linhas / últimas 6 linhas:
print(head(df))
print(tail(df))
```

A estrutura do arquivo pode ser exibida com a função "str()":

```
'data.frame': 121 obs. of 2 variables:
 $ Período : num 2007 2007 2007 2007 2007 ...
 $ Desemprego: num 10.4 10.5 10 10 9.3 9.3 9.1 9.6 9.8 9.8 ...
```

Ele considerou duas colunas: "Período" e "Desemprego", ambas numéricas. A função "summary()" permite, entre outras coisas, observarmos estatísticas das colunas:

Periodo	Desemprego
Min. :2007	Min. : 6.90
1st Qu.:2010	1st Qu.: 8.50
Median :2012	Median : 9.40
Mean :2012	Mean :10.01
3rd Qu.:2015	3rd Qu.:10.70
Max. :2017	Max. :15.90

As funções "head()" e "tail()" são auto explicativas.

Para ler arquivos não convencionais, como o que veremos a seguir, é preciso utilizar alguns parâmetros. Eis o arquivo "dataset-nao-convencional.csv", que está na mesma pasta:

```
Nome;Pesos;Alturas
"Fulano";74;1,73
"Beltrano";61;1,61
"Cicrano";61;1,61
```

O que há de diferente neste dataset? Para começar, o separador de colunas é o ponto e vírgula, e, para piorar, estamos utilizando vírgula como separador de decimais. Podemos configurar isso na própria função "read.csv()":

```
# Arquivos CSV não convencionais:
df2 <- read.csv('dataset-nao-convencional.csv',dec = ',', sep = ";")
str(df2)
print(summary(df2))
```

Pelo resultado da função "str()" podemos ver que ele leu corretamente os dados:

```
'data.frame': 29 obs. of 3 variables:
 $ Nome : Factor w/ 29 levels "Antônia","Antônio",...: 12 4 8 14 23 22 18 13 21 9 ...
 $ Pesos : int 74 61 61 68 70 73 67 67 65 57 ...
 $ Alturas: num 1.73 1.61 1.61 1.67 1.69 1.75 1.67 1.67 1.63 1.57 ...
```

Ao ler arquivos muito grandes, podemos diminuir o uso da memória e acelerar a leitura especificando classes para as colunas:

```
# Especificando as classes das colunas:
# logical, integer, numeric, character, raw, "factor", "Date"
df3 <- read.csv('dataset-nao-convencional.csv',dec = ',', sep = ";",
               colClasses= c('character','integer','numeric'))
```

Datas e horas

Podemos ler e processar campos como datas ou horas. Vamos usar o arquivo "datas.csv", que está na mesma pasta:

```
Nascimento,Pesos,Alturas
"2007/3/5",74,1.73
"2007/1/1",61,1.62
"2007/5/25",61,1.63
"2007/1/15",68,1.68
"2007/2/3",70,1.68
```

Como a data está em um formato aceito pelo R, então basta ler o arquivo desta forma:

```
# Datas:
df4 <- read.csv('datas.csv',colClasses=c('Date','integer','numeric'))
str(df4)
```

Vemos que ele considerou a data corretamente:

```
'data.frame':  5 obs. of  3 variables:
 $ Nascimento: Date, format: "2007-03-05" "2007-01-01" ...
 $ Pesos      : int  74 61 61 68 70
 $ Alturas    : num  1.73 1.62 1.63 1.68 1.68
```

E conseguimos até extrair partes dela, como o campo mês:

```
print(as.numeric(format(df4$Nascimento, "%m")))
```

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

Se a data estiver em um formato diferente, podemos usar a função "strptime()", passando o formato:

```
Nascimento,Pesos,Alturas
"5-3-2005 17:35:00",5,0.50
"1-5-2005 12:10:05",3.5,0.48
"10-6-2005 23:01:12",4.1,0.51
```

Nós lemos como carácter e depois convertemos para data e hora (POSIXlt):

```
df6 <- read.csv('datas-horas.csv', colClasses = c('character', 'numeric',
'numeric'))
str(df6)
df6$Nascimento <- strptime(df6$Nascimento,format='%d-%m-%Y %H:%M:%S')
str(df6)
```

```
'data.frame':  3 obs. of  3 variables:
 $ Nascimento: POSIXlt, format: "2005-03-05 17:35:00" "2005-05-01 12:10:05" ...
 $ Pesos      : num  5 3.5 4.1
 $ Alturas    : num  0.5 0.48 0.51
```

O formato POSIXlt contém data e hora e podemos utilizar variáveis deste tipo para cálculo de períodos.

Os formatos utilizados na função "strptime()" podem ser:

- d: Dia do mês (1-31);
- m: Número do mês (1-12);
- Y (maiúsculo): Ano com século (2017);
- H: Hora em formato 24 horas;
- I: Hora em formato 12 horas. Se usar este formato, acrescente um %p para mostrar "AM / PM";
- M: Minutos (0-59);
- S: Segundos (0-59);

Podemos lidar com dados faltantes de duas maneiras:

1. Remover completamente a linha;
2. Substituir os valores faltantes pela média da coluna.

Para remover as linhas podemos usar a função "na.omit()":

```
df <- na.omit(read.csv('faltando-valor.csv'))
```

Para substituir pela média, podemos usar a seleção de colunas, especificando que queremos aquelas em que o valor estiver ausente:

```
df2$Inflacao[is.na(df2$Inflacao)] <- mean(df2$Inflacao, na.rm = TRUE)
```

Ficou confuso? Vamos esclarecer...

Na primeira parte do comando, selecionamos apenas os valores da coluna "Inflação", utilizando a sintaxe de seletor de colunas (<dataframe>\$<nome da coluna>):

```
df2$Inflacao
```

Desta coluna, queremos apenas aquelas em que seu valor estiver ausente. Para isto, usamos a própria função "is.na()". Serão selecionadas apenas as linhas/colunas em que o resultado desta função seja TRUE:

```
df2$Inflacao[is.na(df2$Inflacao)]
```

Atribuímos às linhas/colunas selecionadas, a média daquela coluna (excluindo as que contiverem valores ausentes):

```
<- mean(df2$Inflacao, na.rm = TRUE)
```

Ficou mais claro?