

tidy_data

April 15, 2020

- José O. Siqueira (siqueira@usp.br)
- Paulo S. P. Silveira (paulo.silveira@fm.usp.br)
- Koichi Sameshima (koichi.sameshima@fm.usp.br)

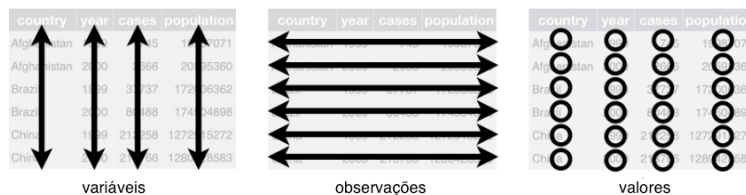
1 Tidy Data / Dados Organizados

1.1 O que é Tidy Data?

R segue um conjunto de convenções que torna um layout de dados tabulares muito mais fácil de trabalhar do que outros. Seus dados serão mais fáceis de trabalhar no R se seguirem três regras:

- Cada **variável** no conjunto de dados é colocada em sua **própria coluna**;
- Cada **observação** é colocada em sua **própria linha**;
- Cada **valor** é colocado em sua **própria célula**.

Os dados que satisfazem essas regras são conhecidos como **tidy data** ou **dados organizados**.



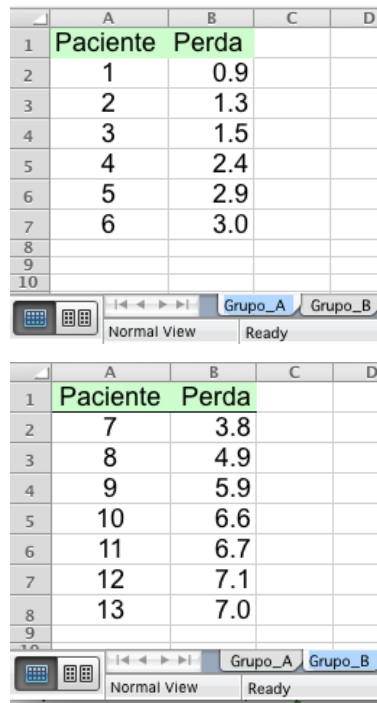
1.2 Conversões entre os formatos WIDE e LONG

1.2.1 Importando dados em planilha Excel

A planilha **anorexigenos.xls** contém dados de um ensaio clínico, em que se comparou dois anorexígenos, A e B, e as perdas de peso foram registradas. Deseja-se testar se a diferença observada nas duas amostras é estatisticamente significativa, assumindo-se $\alpha = 1\%$.

Grupo A		Grupo B	
Paciente	Perda (kg)	Paciente	Perda (kg)
1	0.9	7	3.8
2	1.3	8	4.9
3	1.5	9	5.9
4	2.4	10	6.6
5	2.9	11	6.7
6	3.0	12	7.1
		13	7.0

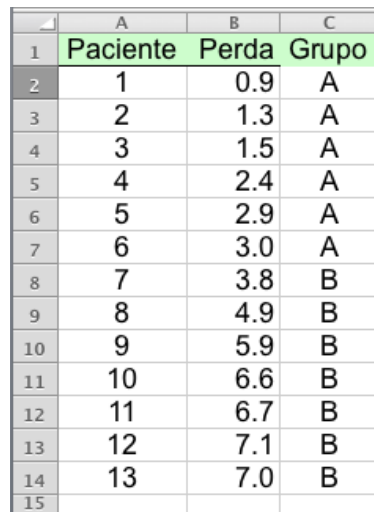
Esse conjunto de dados estão codificados num arquivo Excel, **anorexigenos.xls**, em duas planilhas, **Grupo_A** e **Grupo_B**, na seguinte forma:



	A	B	C	D
1	Paciente	Perda		
2	1	0.9		
3	2	1.3		
4	3	1.5		
5	4	2.4		
6	5	2.9		
7	6	3.0		
8				
9				
10				

	A	B	C	D
1	Paciente	Perda		
2	7	3.8		
3	8	4.9		
4	9	5.9		
5	10	6.6		
6	11	6.7		
7	12	7.1		
8	13	7.0		
9				
10				

Para podermos analisar esses dados no R é necessário rearrumá-los no formato **long**:



	A	B	C
1	Paciente	Perda	Grupo
2	1	0.9	A
3	2	1.3	A
4	3	1.5	A
5	4	2.4	A
6	5	2.9	A
7	6	3.0	A
8	7	3.8	B
9	8	4.9	B
10	9	5.9	B
11	10	6.6	B
12	11	6.7	B
13	12	7.1	B
14	13	7.0	B
15			

```
[1]: if (!require("xlsx")) install.packages("xlsx", repo = "https://vps.fmvz.usp.br/CRAN/",
      dep = TRUE)

library(xlsx)
```

Loading required package: xlsx

Leia a primeira planilha Grupo_A do arquivo anorexigenos.xls.
Primeira linha contém os nomes das variáveis.

```
[2]: mydata_A <- xlsx::read.xlsx("anorexigenos.xls", sheetName="Grupo_A") # "xlsx::"
      ↳ especifica que a função
      ↳ xlsx( ) porém do pacote Xlsx                                     # ou rotina read.
```

```
[3]: head(mydata_A)
```

A data.frame: 6 × 2

	Paciente <dbl>	Perda <dbl>
1	1	0.9
2	2	1.3
3	3	1.5
4	4	2.4
5	5	2.9
6	6	3.0

```
[4]: mydata_B <- xlsx::read.xlsx("anorexigenos.xls", sheetName="Grupo_B")
      head(mydata_B)
```

A data.frame: 6 × 2

	Paciente <dbl>	Perda <dbl>
1	7	3.8
2	8	4.9
3	9	5.9
4	10	6.6
5	11	6.7
6	12	7.1

1.2.2 Crie uma coluna Grupo em cada um dos data frames:

```
[5]: mydata_A$Grupo <- "A"
      mydata_B$Grupo <- "B"
```

```
[6]: head(mydata_A)
```

A data.frame: 6 × 3

	Paciente <dbl>	Perda <dbl>	Grupo <chr>
1	1	0.9	A
2	2	1.3	A
3	3	1.5	A
4	4	2.4	A
5	5	2.9	A
6	6	3.0	A

```
[7]: head(mydata_B)
```

A data.frame: 6 × 3

	Paciente <dbl>	Perda <dbl>	Grupo <chr>
1	7	3.8	B
2	8	4.9	B
3	9	5.9	B
4	10	6.6	B
5	11	6.7	B
6	12	7.1	B

1.2.3 Empilhando dois dataframes com rbind()

```
[8]: ?rbind # e sua companheira cbind
```

```
[9]: mydata_total <- rbind(mydata_A, mydata_B)
```

```
[10]: print(mydata_total)
```

	Paciente	Perda	Grupo
1	1	0.9	A
2	2	1.3	A
3	3	1.5	A
4	4	2.4	A
5	5	2.9	A
6	6	3.0	A
7	7	3.8	B
8	8	4.9	B
9	9	5.9	B
10	10	6.6	B
11	11	6.7	B
12	12	7.1	B
13	13	7.0	B

```
[11]: library(tidyr)
# dados_long <- gather(dadosinicial_wide, condicao, medida, controle:cond2,
#   ↪ factor_key=TRUE)
# dados_long

mydata_wide <- tidyr::spread(mydata_total, Grupo, Perda)
print(mydata_wide)
```

	Paciente	A	B
1	1	0.9	NA
2	2	1.3	NA
3	3	1.5	NA
4	4	2.4	NA
5	5	2.9	NA
6	6	3.0	NA
7	7	NA	3.8
8	8	NA	4.9
9	9	NA	5.9
10	10	NA	6.6
11	11	NA	6.7

```
12      12 NA 7.1
13      13 NA 7.0
```

```
[12]: # ?read.xlsx2
```

```
[13]: options(warn=-1) # Suprime warnings
# Instalando readxl se ainda não presente.
if (!require("readxl")) install.packages("readxl", repo = "https://vps.fmvz.usp.br/CRAN/
↪",
    dep = TRUE)
# Carrega a biblioteca readxl
library(readxl)

options(warn=0)
```

Loading required package: readxl

```
[14]: df_A <- read_excel("anorexigenos.xls",sheet="Grupo_A")
df_A # Agora a função read_excel importa adequadamente os valores de Perda
```

	Paciente	Perda
	<dbl>	<dtm>
	1	1899-12-31 21:36:00
	2	1900-01-01 07:12:00
	3	1900-01-01 12:00:00
	4	1900-01-02 09:36:00
	5	1900-01-02 21:36:00
	6	1900-01-03 00:00:00

A tibble: 6 × 2

Note que a variável Perda foi convertida em formato “date dtm” ao invés de “numeric”, embora no Excel seu formato tenha sido especificado como Number ou Número. Neste caso, as opções **default** dos argumentos adotadas pela função não foram adequadas para a importação este arquivo. Vamos verificar a sintaxe desta função por meio do comando `help(read_excel)` ou `?read_excel`:

```
[15]: ?read_excel #Executando-se este comando obtém-se
```

Facsímile do comando `help(read_excel)` ou `?read_excel`:

Read xls and xlsx files

Description

Read xls and xlsx files `read_excel()` calls `excel_format()` to determine if path is xls or xlsx, based on the file extension and the file itself, in that order. Use `read_xls()` and `read_xlsx()` directly if you know better and want to prevent such guessing.

Usage

```
read_excel(path, sheet = NULL, range = NULL, col_names = TRUE,
  col_types = NULL, na = "", trim_ws = TRUE, skip = 0,
  n_max = Inf, guess_max = min(1000, n_max),
  progress = readxl_progress(), .name_repair = "unique")
```

Arguments

path - Path to the xls/xlsx file.

sheet - Sheet to read. Either a string (the name of a sheet), or an integer (the position of the sheet). Ignored if the sheet is specified via range. If neither argument specifies the sheet, defaults to the first sheet.

range - A cell range to read from, as described in cell-specification. Includes typical Excel ranges like “B3:D87”, possibly including the sheet name like “Budget!B2:G14”, and more. Interpreted strictly, even if the range forces the inclusion of leading or trailing empty rows or columns. Takes precedence over skip, n_max and sheet.

col_names - TRUE to use the first row as column names, FALSE to get default names, or a character vector giving a name for each column. If user provides col_types as a vector, col_names can have one entry per column, i.e. have the same length as col_types, or one entry per unskipped column.

col_types - Either NULL to guess all from the spreadsheet or a character vector containing one entry per column from these options: “skip”, “guess”, “logical”, “numeric”, “date”, “text” or “list”. If exactly one col_type is specified, it will be recycled. The content of a cell in a skipped column is never read and that column will not appear in the data frame output. A list cell loads a column as a list of length 1 vectors, which are typed using the type guessing logic from col_types = NULL, but on a cell-by-cell basis.

na - Character vector of strings to interpret as missing values. By default, readxl treats blank cells as missing data.

trim_ws - Should leading and trailing whitespace be trimmed?

skip - Minimum number of rows to skip before reading anything, be it column names or data. Leading empty rows are automatically skipped, so this is a lower bound. Ignored if range is given.

n_max - Maximum number of data rows to read. Trailing empty rows are automatically skipped, so this is an upper bound on the number of rows in the returned tibble. Ignored if range is given.

guess_max - Maximum number of data rows to use for guessing column types.

progress - Display a progress spinner? By default, the spinner appears only in an interactive session, outside the context of knitting a document, and when the call is likely to run for several seconds or more. See readxl_progress() for more details.

.name_repair - Handling of column names. By default, readxl ensures column names are not empty and are unique. If the tibble package version is recent enough, there is full support for .name_repair as documented in tibble::tibble(). If an older version of tibble is present, readxl falls back to name repair in the style of tibble v1.4.2.

Neste caso, a solução é especificar o argumento **col_types**

```
[16]: options(warn=-1)
df_A <- read_excel("anorexigenos.xls", sheet="Grupo_A", col_types=c("text", "numeric"))
df_A # Neste caso a função read_excel não importa adequadamente os valores de Perda
```

A tibble: 6 × 2

	Paciente <chr>	Perda <dbl>
1		0.9
2		1.3
3		1.5
4		2.4
5		2.9
6		3.0

```
[17]: options(warn=-1)
df_B <- read_excel("anorexigenos.xls",sheet="Grupo_B",col_types=c("text","numeric"))
df_B
```

	Paciente <chr>	Perda <dbl>
	7	3.8
	8	4.9
A tibble: 7 × 2	9	5.9
	10	6.6
	11	6.7
	12	7.1
	13	7.0

1.3 Conversão entre formatos WIDE e LONG

Na prática de análises estatísticas é necessário comumente converter dados entre os formatos “wide” e “long”. Muitas funções do R requerem que os dados estejam no formato “long” em vez de formato “wide” (e vice-versa), usado frequentemente em programas de análise estatística comerciais tais como SPSS, Minitab e etc.

O roteiro que segue foi adaptado de [Converting data between wide and long format](#).

No exemplo abaixo, os dois dataframes contêm os mesmos conjuntos de dados em formatos “wide” e “long”. Aqui mostraremos como converter de um formato ao outro e vice-versa.

```
[18]: # Criando data frame no formato wide
dadosinicial_wide <- read.table(header=TRUE, text='
sujeito sexo controle cond1 cond2
  1      M      7.9  12.3  10.7
  2      F      6.3  10.6  11.1
  3      F      9.5  13.1  13.8
  4      M     11.5  13.4  12.9
')
```

```
[19]: print(dadosinicial_wide)
```

	sujeito	sexo	controle	cond1	cond2
1	1	M	7.9	12.3	10.7
2	2	F	6.3	10.6	11.1
3	3	F	9.5	13.1	13.8
4	4	M	11.5	13.4	12.9

```
[20]: # Garanta que a coluna sujeito seja um fator.
dadosinicial_wide$sujeito <- factor(dadosinicial_wide$sujeito)
```

```
[21]: # Criando data frame no formato long
dadosinicial_long <- read.table(header=TRUE, text='
sujeito sexo condicao medida
  1      M controle      7.9
  1      M   cond1     12.3
  1      M   cond2     10.7
  2      F controle      6.3
```

```

      2    F    cond1    10.6
      2    F    cond2    11.1
      3    F controle     9.5
      3    F    cond1    13.1
      3    F    cond2    13.8
      4    M controle    11.5
      4    M    cond1    13.4
      4    M    cond2    12.9
    ')
    # Garanta que a coluna sujeito seja um fator.
    dadosinicial_long$sujeito <- factor(dadosinicial_long$sujeito)

```

1.4 Conversão WIDE → LONG

1.4.1 Usando a rotina gather

```

[22]: if (!require("tidyr")) install.packages("tidyr", repo="https://vps.fmvz.usp.br/CRAN/
      ↪", dep=TRUE)

      library(tidyr)

      # The arguments to gather():
      # - data: Data object
      # - key: Name of new key column (made from names of data columns)
      # - value: Name of new value column
      # - ...: Names of source columns that contain values
      # - factor_key: Treat the new key column as a factor (instead of character vector)

[23]: dados_long <- tidyr::gather(dadosinicial_wide, condicao, medida, controle:cond2,
      ↪ factor_key=TRUE)
      dados_long

```

A data.frame: 12 × 4

	sujeito <fct>	sexo <fct>	condicao <fct>	medida <dbl>
	1	M	controle	7.9
	2	F	controle	6.3
	3	F	controle	9.5
	4	M	controle	11.5
	1	M	cond1	12.3
	2	F	cond1	10.6
	3	F	cond1	13.1
	4	M	cond1	13.4
	1	M	cond2	10.7
	2	F	cond2	11.1
	3	F	cond2	13.8
	4	M	cond2	12.9

Neste exemplo, as colunas a serem juntadas (gathered) foram especificadas por **controle:cond2**, i.e. todas as colunas posicionadas entre **controle** e **cond2**. Observe a ordenação alfabética dos fatores da variável **condição**.

Outra maneira de se executar a mesma tarefa, é listar explicitamente todas as colunas individualmente, via **gather**:


```
[24]: tidyr::gather(dadosinicial_wide, condicao, medida, controle, cond1, cond2)
```

	sujeito	sexo	condicao	medida
	<fct>	<fct>	<chr>	<dbl>
	1	M	controle	7.9
	2	F	controle	6.3
	3	F	controle	9.5
	4	M	controle	11.5
A data.frame: 12 × 4	1	M	cond1	12.3
	2	F	cond1	10.6
	3	F	cond1	13.1
	4	M	cond1	13.4
	1	M	cond2	10.7
	2	F	cond2	11.1
	3	F	cond2	13.8
	4	M	cond2	12.9

Se tens intenção de usar esta função, `gather()`, dentro de um programa e talvez queiras colocar variáveis contendo os nomes das colunas como parâmetro da função, então será necessário usar a função `gather_()`, que aceita “strings” ou lista de strings em vez de nomes de colunas sem aspas ou apóstrofes.

```
[25]: keycol <- "condicao" # coluna chave
valuecol <- "medida" # variável dependente (VD)
gathercols <- c("controle", "cond1", "cond2")

tidyr::gather_(dadosinicial_wide, keycol, valuecol, gathercols)
```

	sujeito	sexo	condicao	medida
	<fct>	<fct>	<chr>	<dbl>
	1	M	controle	7.9
	2	F	controle	6.3
	3	F	controle	9.5
	4	M	controle	11.5
A data.frame: 12 × 4	1	M	cond1	12.3
	2	F	cond1	10.6
	3	F	cond1	13.1
	4	M	cond1	13.4
	1	M	cond2	10.7
	2	F	cond2	11.1
	3	F	cond2	13.8
	4	M	cond2	12.9

É possível ou desejável ocasionalmente renomear os níveis de fator da coluna de variável, e classificá-los em alguma ordem (ascendente ou descendente).

```
[26]: # Renomear nomes de fator de 'cond1' and 'cond2' para 'nivel1' e 'nivel2'
levels(dados_long$condicao)[levels(dados_long$condicao) == "cond1"] <- "nivel1"
levels(dados_long$condicao)[levels(dados_long$condicao) == "cond2"] <- "nivel2"

# Sortear por sujeito, depois por condicao
dados_long <- dados_long[order(dados_long$sujeito, dados_long$condicao, decreasing =
↪ FALSE), ]
dados_long
```

A data.frame: 12 × 4

	sujeito <fct>	sexo <fct>	condicao <fct>	medida <dbl>
1	1	M	controle	7.9
5	1	M	nivel1	12.3
9	1	M	nivel2	10.7
2	2	F	controle	6.3
6	2	F	nivel1	10.6
10	2	F	nivel2	11.1
3	3	F	controle	9.5
7	3	F	nivel1	13.1
11	3	F	nivel2	13.8
4	4	M	controle	11.5
8	4	M	nivel1	13.4
12	4	M	nivel2	12.9

```
[27]: levels(dados_long$condicao) # Listando os níveis do fator condição
```

1. 'controle' 2. 'nivel1' 3. 'nivel2'

```
[28]: ?order # Execute para ver a sintaxe deste comando ou função
```

1.5 Conversão de LONG → WIDE

1.5.1 Usando rotina spread da biblioteca tidyr

```
[29]: library(tidyr)

# os argumentos para spread(data, key, value):
# - data: Objeto de dados
# - key: Nome de coluna contendo os novos nomes de colunas
# - value: Nome da coluna que contém os valores (VD)
dados_wide <- tidyr::spread(dadosinicial_long, condicao, medida)

#>   sujeito sexo cond1 cond2 controle
#> 1      1    M  12.3  10.7      7.9
#> 2      2    F  10.6  11.1      6.3
#> 3      3    F  13.1  13.8      9.5
#> 4      4    M  13.4  12.9     11.5
```

```
[30]: dados_wide
```

A data.frame: 4 × 5

	sujeito <fct>	sexo <fct>	cond1 <dbl>	cond2 <dbl>	controle <dbl>
1	1	M	12.3	10.7	7.9
2	2	F	10.6	11.1	6.3
3	3	F	13.1	13.8	9.5
4	4	M	13.4	12.9	11.5

À semelhança do **formato long** em que se mudou os níveis do fator condição, aqui no **formato wide** pode-se renomear as variáveis coluna:

```
[31]: # Renomear de cond1 para nivel1, e cond2 para nivel2
names(dados_wide)[names(dados_wide)=="cond1"] <- "nivel1"
```

```
names(dados_wide)[names(dados_wide)=="cond2"] <- "nivel2"

# Aqui podemos reordenar as colunas
dados_wide <- dados_wide[, c(1,2,5,3,4)]
#>   sujeito sexo controle nivel1 nivel2
#> 1      1    M      7.9    12.3    10.7
#> 2      2    F      6.3    10.6    11.1
#> 3      3    F      9.5    13.1    13.8
#> 4      4    M     11.5    13.4    12.9
```

```
[32]: dados_wide
```

	sujeito	sexo	controle	nivel1	nivel2
	<fct>	<fct>	<dbl>	<dbl>	<dbl>
A data.frame: 4 × 5	1	M	7.9	12.3	10.7
	2	F	6.3	10.6	11.1
	3	F	9.5	13.1	13.8
	4	M	11.5	13.4	12.9

A ordem dos níveis dos fatores determina a ordem de aparecimento das colunas, agora como nomes de coluna. A ordem dos níveis pode ser alterada antes de “reshaping”, ou as colunas poder ser reordenados depois.

1.6 Biblioteca reshape2

1.6.1 Conversão WIDE => LONG

Uso melt:

```
[33]: options(warn=-1)
if (!require("reshape2")) install.packages("reshape2", repo = "https://vps.fmvz.usp.br/
  ↪CRAN/",
  dep = TRUE)

library(reshape2)
```

Loading required package: reshape2

Attaching package: ‘reshape2’

The following object is masked from ‘package:tidyr’:

smiths

```
[34]: ?reshape2::melt
```

```
[35]: dadosinicial_wide
#>   sujeito sexo controle cond1 cond2
#> 1      1    M      7.9    12.3    10.7
#> 2      2    F      6.3    10.6    11.1
```

```
#> 3      3      F      9.5 13.1 13.8
#> 4      4      M     11.5 13.4 12.9

# Especificar id.vars: as variáveis para manter mas não separá-los
reshape2::melt(dadosinicial_wide, id.vars=c("sujeito", "sexo"))
#>   sujeito sexo variable value
#> 1      1      M controle   7.9
#> 2      2      F controle   6.3
#> 3      3      F controle   9.5
#> 4      4      M controle  11.5
#> 5      1      M   cond1  12.3
#> 6      2      F   cond1  10.6
#> 7      3      F   cond1  13.1
#> 8      4      M   cond1  13.4
#> 9      1      M   cond2  10.7
#> 10     2      F   cond2  11.1
#> 11     3      F   cond2  13.8
#> 12     4      M   cond2  12.9
```

A data.frame: 4 × 5

	sujeito <fct>	sexo <fct>	controle <dbl>	cond1 <dbl>	cond2 <dbl>
1	1	M	7.9	12.3	10.7
2	2	F	6.3	10.6	11.1
3	3	F	9.5	13.1	13.8
4	4	M	11.5	13.4	12.9

A data.frame: 12 × 4

	sujeito <fct>	sexo <fct>	variable <fct>	value <dbl>
1	1	M	controle	7.9
2	2	F	controle	6.3
3	3	F	controle	9.5
4	4	M	controle	11.5
5	1	M	cond1	12.3
6	2	F	cond1	10.6
7	3	F	cond1	13.1
8	4	M	cond1	13.4
9	1	M	cond2	10.7
10	2	F	cond2	11.1
11	3	F	cond2	13.8
12	4	M	cond2	12.9

Note que os nomes para duas colunas à direita foram atribuídas automaticamente pela rotina `melt()`, “variable” e “value”.

Há opções de `melt` que torna a saída talvez um pouco mais adequada para o seu trabalho:

```
[36]: dados_long <- melt(dadosinicial_wide,
  # ID variables - todas as variáveis a preservar mas não separadas.
  id.vars=c("sujeito", "sexo"),
  # Fontes das colunas
  measure.vars=c("controle", "cond1", "cond2" ),
  # Nome de coluna destino que identificará a coluna original
  # de onde a medida proveio
  variable.name="condicao",
  value.name="medida")
```

```
[37]: print(dados_long)
#>      sujeito sexo  condicao medida
#> 1         1    M controle   7.9
#> 2         2    F controle   6.3
#> 3         3    F controle   9.5
#> 4         4    M controle  11.5
#> 5         1    M   cond1  12.3
#> 6         2    F   cond1  10.6
#> 7         3    F   cond1  13.1
#> 8         4    M   cond1  13.4
#> 9         1    M   cond2  10.7
#> 10        2    F   cond2  11.1
#> 11        3    F   cond2  13.8
#> 12        4    M   cond2  12.9
```

```
      sujeito sexo condicao medida
1         1    M controle   7.9
2         2    F controle   6.3
3         3    F controle   9.5
4         4    M controle  11.5
5         1    M   cond1  12.3
6         2    F   cond1  10.6
7         3    F   cond1  13.1
8         4    M   cond1  13.4
9         1    M   cond2  10.7
10        2    F   cond2  11.1
11        3    F   cond2  13.8
12        4    M   cond2  12.9
```

Se o argumento `measure.vars` for omitido, `melt()` usará automaticamente todas as outras variáveis como `id.vars`. O complemento é verdadeiro se `id.vars` for omitido.

Como visto, se não se especificar `variable.name`, `melt()` nomeará a coluna como “variable”, e se `value.name` for omitido, a coluna de VD será denominada “measurement”.

Opcionalmente pode-se renomear os níveis dos fatores da variável coluna.

```
[38]: # Renomear nomes de fator de "cond1" e "cond2" para, por exemplo, "prima" e "seconda"
levels(dados_long$condicao)[levels(dados_long$condicao)=="cond1"] <- "prima"
levels(dados_long$condicao)[levels(dados_long$condicao)=="cond2"] <- "seconda"

# Ordenar primeiro por sujeito, depois pela condição
dados_long <- dados_long[ order(dados_long$sujeito, dados_long$condicao), ]
#>      sujeito sexo  condicao medida
#> 1         1    M controle   7.9
#> 5         1    M   prima  12.3
#> 9         1    M  seconda  10.7
#> 2         2    F controle   6.3
#> 6         2    F   prima  10.6
#> 10        2    F  seconda  11.1
#> 3         3    F controle   9.5
#> 7         3    F   prima  13.1
#> 11        3    F  seconda  13.8
#> 4         4    M controle  11.5
#> 8         4    M   prima  13.4
```

```
#> 12      4      M   seconda  12.9
```

```
[39]: print(dados_long)
```

```
   sujeito sexo condicao medida
1         1    M controle   7.9
5         1    M   prima  12.3
9         1    M  seconda  10.7
2         2    F controle   6.3
6         2    F   prima  10.6
10        2    F  seconda  11.1
3         3    F controle   9.5
7         3    F   prima  13.1
11        3    F  seconda  13.8
4         4    M controle  11.5
8         4    M   prima  13.4
12        4    M  seconda  12.9
```

1.6.2 Conversão LONG => WIDE

O seguinte código usa `dcast` para remodelar (reshape) os dados. Esta função é destinada para variáveis do tipo **data frames**; se estiver lidando com variáveis **arrays** ou **matrizes**, use `acast`.

```
[40]: library(reshape2)
```

```
[41]: dadosinicial_long
```

```
#>   sujeito sexo   condicao medida
#> 1         1    M controle   7.9
#> 2         2    F controle   6.3
#> 3         3    F controle   9.5
#> 4         4    M controle  11.5
#> 5         1    M   cond1  12.3
#> 6         2    F   cond1  10.6
#> 7         3    F   cond1  13.1
#> 8         4    M   cond1  13.4
#> 9         1    M   cond2  10.7
#> 10        2    F   cond2  11.1
#> 11        3    F   cond2  13.8
#> 12        4    M   cond2  12.9
```

A data.frame: 12 × 4

	sujeito <fct>	sexo <fct>	condicao <fct>	medida <dbl>
	1	M	controle	7.9
	1	M	cond1	12.3
	1	M	cond2	10.7
	2	F	controle	6.3
	2	F	cond1	10.6
	2	F	cond2	11.1
	3	F	controle	9.5
	3	F	cond1	13.1
	3	F	cond2	13.8
	4	M	controle	11.5
	4	M	cond1	13.4
	4	M	cond2	12.9

```
[42]: # Da fonte:
# "sujeito" e "sexo" são colunas que desejamos manter na mesma forma
# "condicao" é a coluna que contém os nomes das novas colunas destinados para colocar as
# ↳ coisas da
# "medida" para conter as medidas (VD)
dados_wide <- reshape2::dcast(dadosinicial_long, sujeito + sexo ~ condicao, value.
# ↳ var="medida")
#>   sujeito sexo controle cond1 cond2
#> 1      1    M      7.9  12.3  10.7
#> 2      2    F      6.3  10.6  11.1
#> 3      3    F      9.5  13.1  13.8
#> 4      4    M     11.5  13.4  12.9
```

```
[43]: print(dados_wide)
```

```
   sujeito sexo cond1 cond2 controle
1      1    M  12.3  10.7      7.9
2      2    F  10.6  11.1      6.3
3      3    F  13.1  13.8      9.5
4      4    M  13.4  12.9     11.5
```

Se conveniente, pode-se modificar a nomeação das colunas na forma adequada para as suas necessidades.

```
[44]: # Renomear cond1 para prima, e cond2 para segunda
names(dados_wide)[names(dados_wide)=="cond1"] <- "prima"
names(dados_wide)[names(dados_wide)=="cond2"] <- "segunda"

# Reordenando as colunas
dados_wide <- dados_wide[, c(1,2,5,3,4)]

#>   sujeito sexo controle prima segunda
#> 1      1    M      7.9  12.3  10.7
#> 2      2    F      6.3  10.6  11.1
#> 3      3    F      9.5  13.1  13.8
#> 4      4    M     11.5  13.4  12.9
```

```
[45]: print(dados_wide)
```

```
   sujeito sexo controle prima segunda
```

1	1	M	7.9	12.3	10.7
2	2	F	6.3	10.6	11.1
3	3	F	9.5	13.1	13.8
4	4	M	11.5	13.4	12.9

[]: