

“Manipulação” de dados usando R

Pedro C. Junger
Doutorando no LMPB
(pedro.junger@gmail.com)

São Carlos, Janeiro 2018

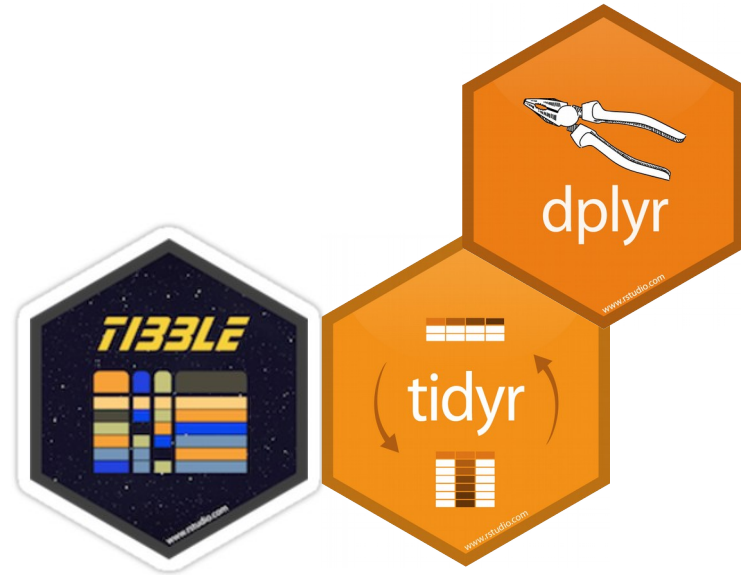
O que veremos nesta aula:

1. Princípios de organização de dados no R
2. Introdução a manipulação de dados no R
3. Funcionamento do tidyr
4. Funcionamento do dplyr
5. Exercícios

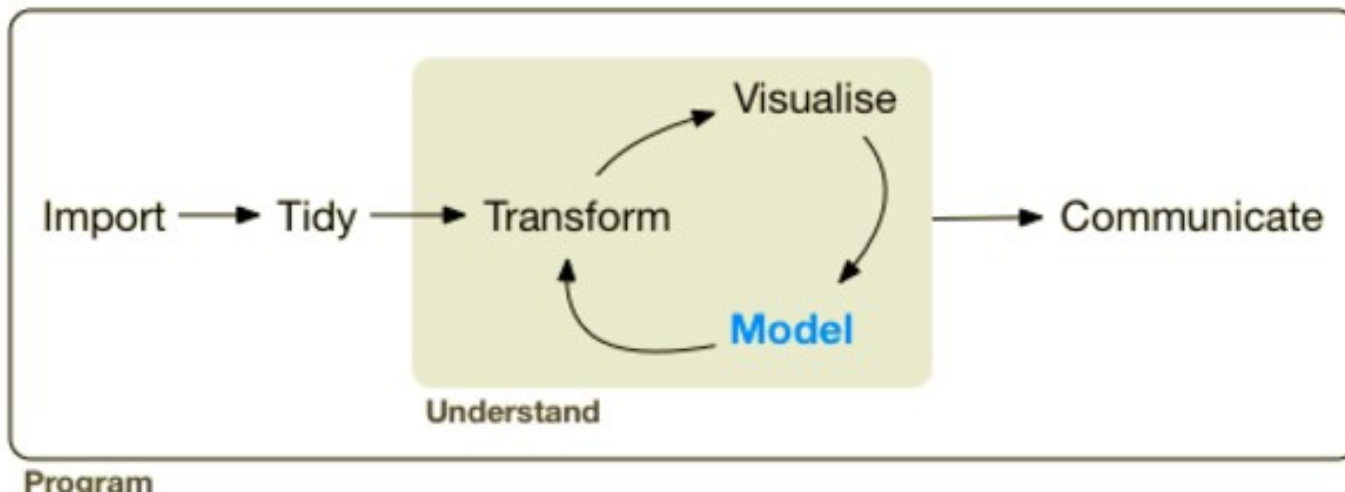
Tidyverse resolve sua vida!



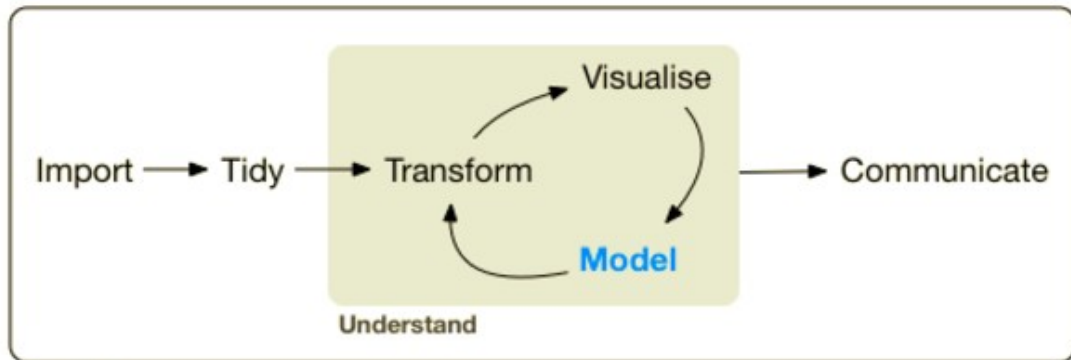
- **tidyr**
- **dplyr**
- *tibble*
- ggplot2
- ggmap
- ...



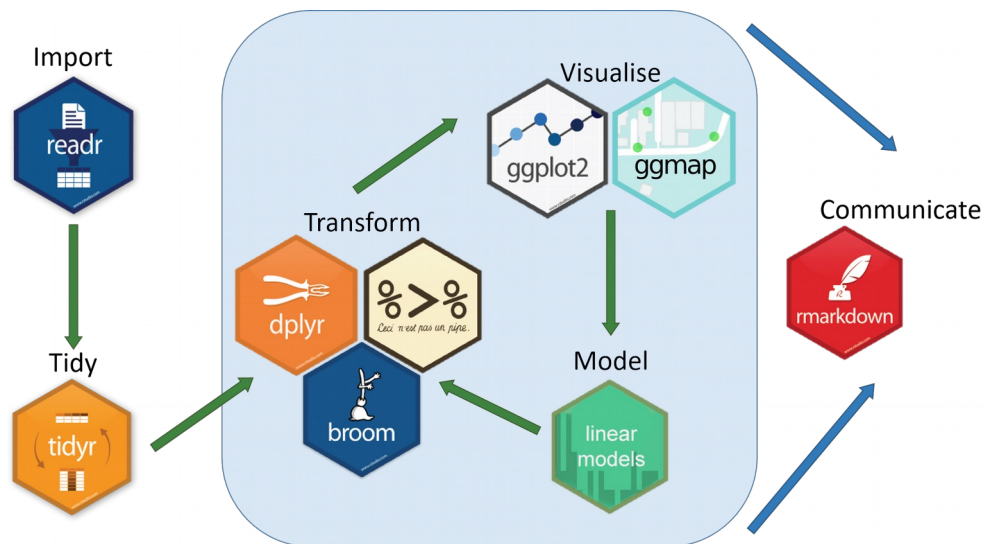
Filosofia do Hadley para análise de dados



Filosofia do Hadley para análise de dados



Program



Tidy data

Como organizar seus dados para análise?

Seal.Length	Seal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.3	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa

"The principles of tidy data provide a
standard way to organise data values
within a dataset"

Estrutura formal de um banco de dados

A tabela organizada de forma padrão: (i) cada variável ocupa uma coluna; (ii) cada observação ocupa uma linha.

Tabela com 3 variáveis, 6 observações e um total de 18 valores.

name	trt	result
John Smith	a	—
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

Estrutura formal de um banco de dados

A tabela organizada de forma padrão: (i) cada variável ocupa uma coluna; (ii) cada observação ocupa uma linha

18 valores

name	trt	result
John Smith	a	—
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

Estrutura formal de um banco de dados

A tabela organizada de forma padrão: (i) cada variável ocupa uma coluna; (ii) cada observação ocupa uma linha

3 Variáveis

name	trt	result
John Smith	a	—
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

Estrutura formal de um banco de dados

A tabela organizada de forma padrão: (i) cada variável ocupa uma coluna; (ii) cada observação ocupa uma linha

6 observações

name	trt	result
John Smith	a	—
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

Tidy data

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20095360
Brazil	1999	37737	17206362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	210766	128042583

variables

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20095360
Brazil	1999	37737	17206362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	210766	128042583

observations

country	year	cases	population
Afghanistan	99	745	19987071
Afghanistan	00	2666	20095360
Brazil	99	37737	17206362
Brazil	00	80488	174504898
China	99	212258	1272915272
China	00	210766	128042583

values

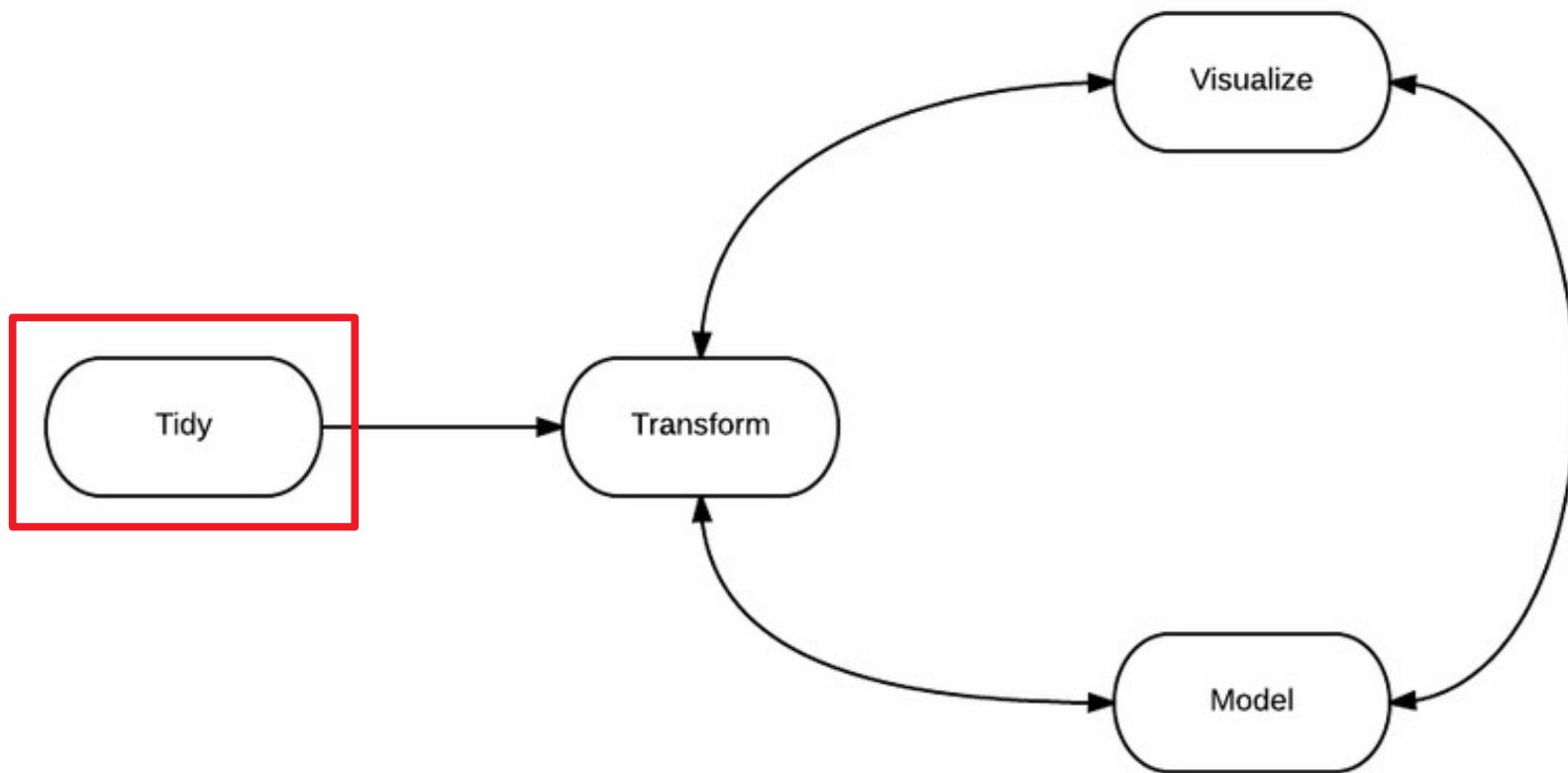
Este banco de dados é tidy?

```
# A tibble: 12 x 13
  country `1952` `1957` `1962` `1967` `1972` `1977` `1982` `1987`
  <chr>    <int> <int> <int> <int> <int> <int> <int> <int>
1 Albania      -9      -9      -9      -9      -9      -9      -9      -9
2 Argentina    -9      -1      -1      -9      -9      -9      -8       8
3 Armenia      -9      -7      -7      -7      -7      -7      -7      -7
4 Australia    10      10      10      10      10      10      10      10
5 Austria      10      10      10      10      10      10      10      10
6 Azerbaijan   -9      -7      -7      -7      -7      -7      -7      -7
7 Belarus      -9      -7      -7      -7      -7      -7      -7      -7
8 Belgium      10      10      10      10      10      10      10      10
9 Bhutan      -10     -10     -10     -10     -10     -10     -10     -10
10 Bolivia     -4      -3      -3      -4      -7      -7       8       9
11 Brazil       5       5       5      -9      -9      -4      -3       7
12 Bulgaria    -7      -7      -7      -7      -7      -7      -7      -7
# ... with 4 more variables: `1992` <int>, `1997` <int>,
#   `2002` <int>, `2007` <int>
```

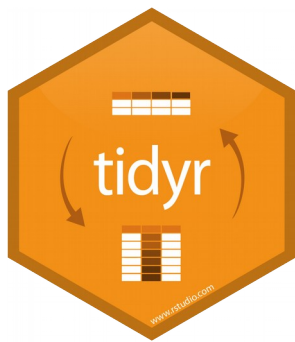
Para o bom funcionamento de muitos programas (i.e.:ggplot2), temos que ter os dados organizados desta maneira. **Cada variável em uma coluna** e cada observação em uma linha (tidy data.frame).

name	trt	result
John Smith	a	—
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

Filosofia do Hadley para análise de dados

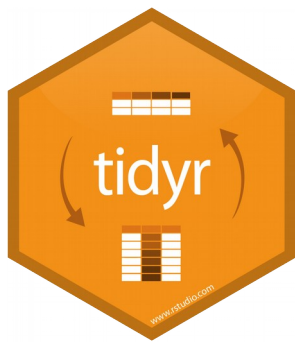


Introdução ao Tidy



O pacote **tidyr** dispõe de funções úteis para deixar os seus dados no formato que você precisa para a análise. Na maioria das vezes, utilizamos para deixá-los **tidy**. Outras, precisamos “bagunça-los” um pouco para poder aplicar alguma função

Introdução ao Tidyr




Principais funções:

- **gather()**
- **spread()**
- **separate()** and **unite()**
- **extract()**

Função **gather()** do tidyr

A função **gather()** “empilha” o banco de dados. Ela é utilizada principalmente quando as colunas da base não representam nomes de variáveis, mas sim seus valores. Com esta função pode-se transformar a tabela de um **formato largo para longo** (“wide to long format”), o que é desejável na maioria das vezes!



ID	Product1	Product2	Product3	Product4
1	1	NA	1	1
2	1	1	NA	1
3	1	1	NA	NA
4	1	1	1	1

ID	Product	value
1	Product1	1
1	Product3	1
1	Product4	1
2	Product1	1
2	Product2	1
2	Product4	1
3	Product1	1
3	Product2	1
4	Product1	1
4	Product2	1
4	Product3	1
4	Product4	1

wide vs long

ID	Product	value
1	Product1	1
1	Product3	1
1	Product4	1
2	Product1	1
2	Product2	1
2	Product4	1
3	Product1	1
3	Product2	1
4	Product1	1
4	Product2	1
4	Product3	1
4	Product4	1

Desafio!

Qual o formato de dataframe? E como podemos transformá-lo? Faça no papel!

```
# A tibble: 12 x 13
  country    `1952` `1957` `1962` `1967` `1972` `1977` `1982` `1987`
  <chr>      <int>  <int>  <int>  <int>  <int>  <int>  <int>  <int>
1 Albania      -9     -9     -9     -9     -9     -9     -9     -9
2 Argentina    -9     -1     -1     -9     -9     -9     -8      8
3 Armenia      -9     -7     -7     -7     -7     -7     -7     -7
4 Australia    10     10     10     10     10     10     10     10
5 Austria      10     10     10     10     10     10     10     10
6 Azerbaijan   -9     -7     -7     -7     -7     -7     -7     -7
7 Belarus      -9     -7     -7     -7     -7     -7     -7     -7
8 Belgium      10     10     10     10     10     10     10     10
9 Bhutan     -10    -10    -10    -10    -10    -10    -10    -10
10 Bolivia      -4     -3     -3     -4     -7     -7      8      9
11 Brazil        5      5      5     -9     -9     -4     -3      7
12 Bulgaria     -7     -7     -7     -7     -7     -7     -7     -7
# ... with 4 more variables: `1992` <int>, `1997` <int>,
#   `2002` <int>, `2007` <int>
```

Estrutura básica da função **gather()**:

```
gather(dados, key = "key", value = "value")
```

Argumentos:

key = “Os atributos desejados são colocados na coluna “key” (pode chamar como quiser)

value = Os respectivos valores desses novos atributos (também pode ser chamado como quiser)

Pode-se escolher as colunas que não se quer (ou não) “empilhar” com o **gather()**:

```
gather(dados, key = "key", value = "value", -nomedacoluna)
```

Argumentos:

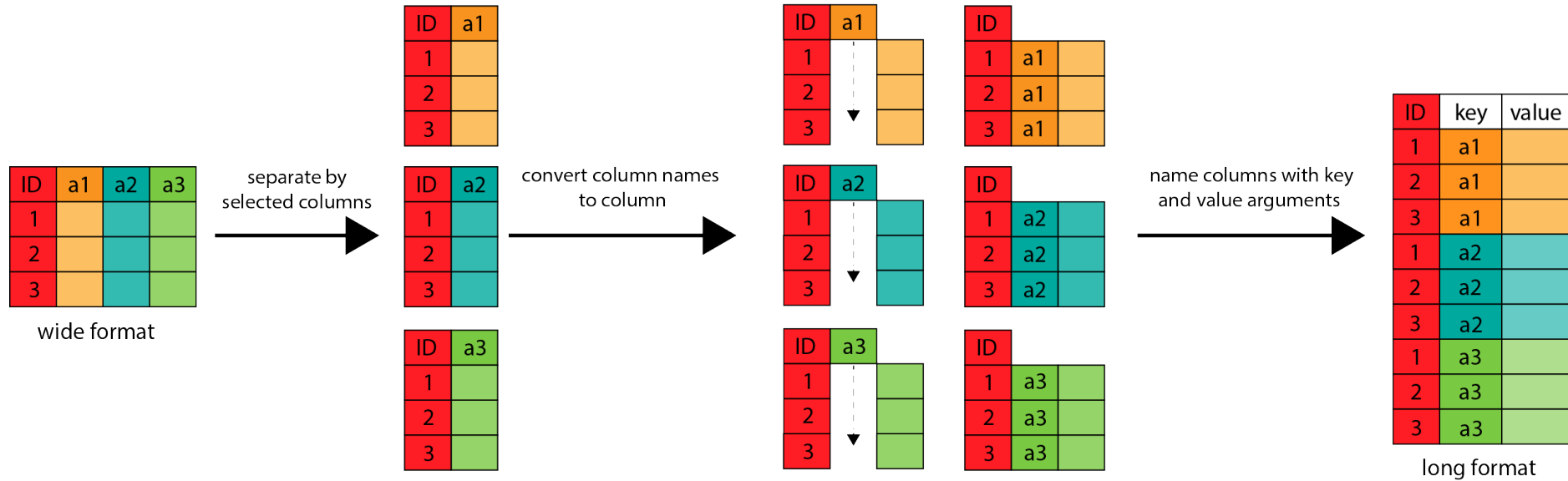
key = Os atributos desejados são colocados na coluna “key” (pode chamar como quiser)

value = Os respectivos valores desses novos atributos (também pode ser chamado como quiser)

... = Seleção das colunas. Se ficar vazio seleciona todas.

Como opera a função **gather()**:

`gather(data, key = "key", value = "value", c("a1", "a2", "a3"))`



Função **spread()** do tidyR

A função **spread()** é essencialmente o inverso da **gather()**. Ela espalha uma determinada variável nas colunas, **transformando do formato longo para amplo** (“long to wide format”).

ID	Product	value
1	Product1	1
1	Product3	1
1	Product4	1
2	Product1	1
2	Product2	1
2	Product4	1
3	Product1	1
3	Product2	1
4	Product1	1
4	Product2	1
4	Product3	1
4	Product4	1



ID	Product1	Product2	Product3	Product4
1	1	NA	1	1
2	1	1	NA	1
3	1	1	NA	NA
4	1	1	1	1

spread()



Estrutura básica da função **spread()**:

```
spread(dados, key = "key", value = "value")
```

Argumentos:

key = coluna chave que deseja transformar em atributos (colunas espalhadas na nova tabela)

value = os valores dos atributos que deseja “espalhar” na nova tabela

Função **separate()** do tidyr

A função **separate()** transforma uma coluna em várias usando um separador ('_', por exemplo).

```
separate(data, col, into, sep = " ")
```

Argumentos principais:

col = coluna que deseja separar

into = nome das novas variáveis (colunas) que serão criadas a partir da separação

sep= especifica o separador entre colunas

Exemplo de aplicação do `separate()` do `tidyr`

```
separate(data=dados,col= Quarter, into= c("Time_Interval", "Interval_ID"))
```

##	Group	Year	Quarter	Revenue
## 1	1	2006	Qtr.1	15
## 2	1	2007	Qtr.1	12
## 3	1	2008	Qtr.1	22
## 4	1	2009	Qtr.1	10
## 5	2	2006	Qtr.1	12
## 6	2	2007	Qtr.1	16



	Group	Year	Time_Interval	Interval_ID	Revenue
1	1	2006	Qtr	1	15
2	1	2007	Qtr	1	12
3	1	2008	Qtr	1	22
4	1	2009	Qtr	1	10
5	2	2006	Qtr	1	12
6	2	2007	Qtr	1	16

Função **unite()** do tidyrr

A função **unite()** faz o inverso do `separate()`, ou seja, junta duas ou mais colunas usando algum separador como referência (`_`, por exemplo).

```
unite(data, col, ..., sep = "_")
```

Argumentos principais:

col = nome da nova coluna que deseja criar

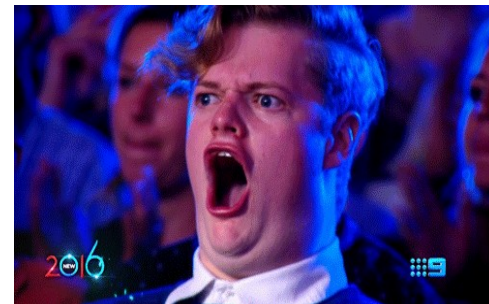
... = seleção das colunas que deseja unificar. Se ficar vazia, todas as variáveis (colunas) da sua tabela serão selecionadas.

sep= especifica o separador entre valores

Exemplo de aplicação do **unite()** do tidyr:

```
unite(data=dados,col= Quarter, into= c("Time_Interval", "Interval_ID"))
```

	Group	Year	Time_Interval	Interval_ID	Revenue
1	1	2006	Qtr	1	15
2	1	2007	Qtr	1	12
3	1	2008	Qtr	1	22
4	1	2009	Qtr	1	10
5	2	2006	Qtr	1	12
6	2	2007	Qtr	1	16



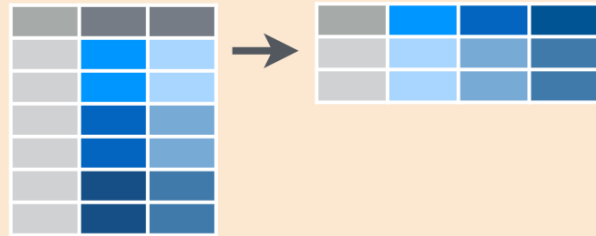
##	Group	Year	Quarter	Revenue
## 1	1	2006	Qtr.1	15
## 2	1	2007	Qtr.1	12
## 3	1	2008	Qtr.1	22
## 4	1	2009	Qtr.1	10
## 5	2	2006	Qtr.1	12
## 6	2	2007	Qtr.1	16

Resumo da opera



tidyr::gather(cases, "year", "n", 2:4)

Gather columns into rows.



tidyr::spread(pollution, size, amount)

Spread rows into columns.



tidyr::separate(storms, date, c("y", "m", "d"))

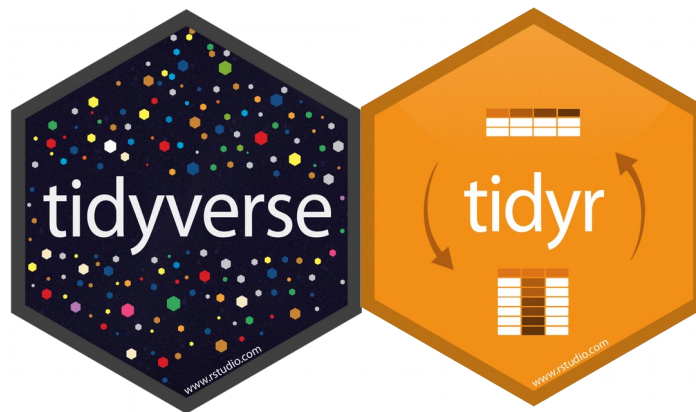
Separate one column into several.



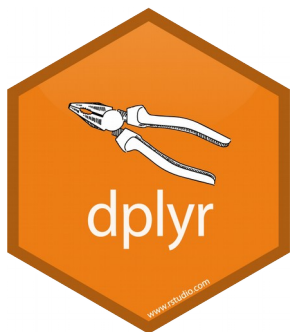
tidyr::unite(data, col, ..., sep)

Unite several columns into one.

Vamos aos exercícios!

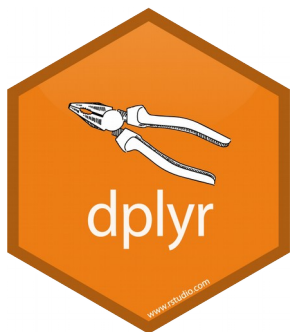


Introdução ao dplyr



O pacote **dplyr** é o pacote mais útil para realizar manipulação e transformação de base de dados, aliando simplicidade e eficiência de uma forma elegante.

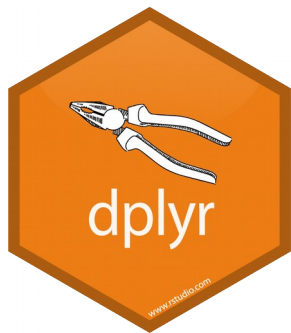
Introdução ao dplyr



Principais funções:

- **filter()** - filtra linhas
- **select()** - seleciona colunas
- **mutate()** - cria/modifica colunas
- **arrange()** - ordena a base
- **summarise()** - sumariza a base

Introdução ao dplyr



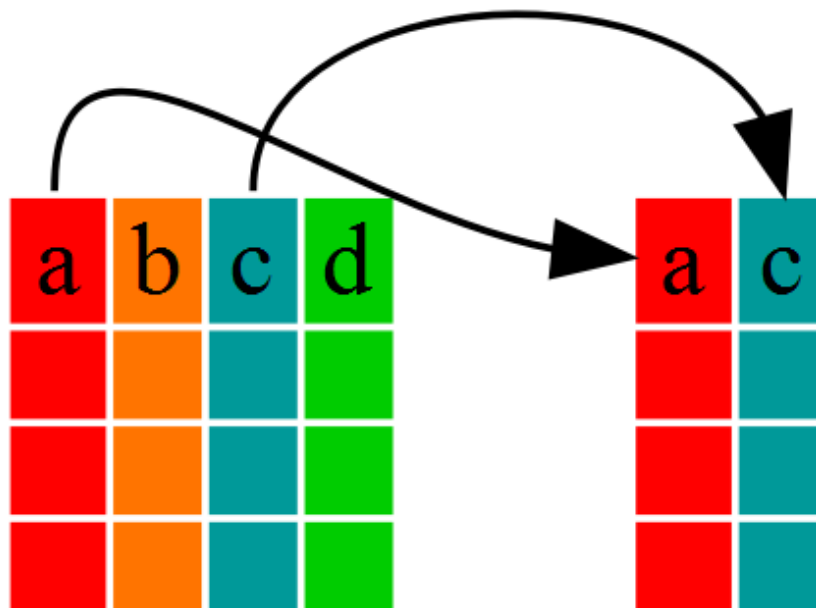
Vantagens em relação em detrimento das funções do R base:

- Manipular dados se torna muito mais simples;
- O código fica mais intuitivo de ser escrito e simples de ser lido
- Código mais eficiente (linguagem C e C++)

Função `select()`

A função **`select()`** seleciona colunas (variáveis). É possível utilizar nomes, índices ou intervalos de variáveis.

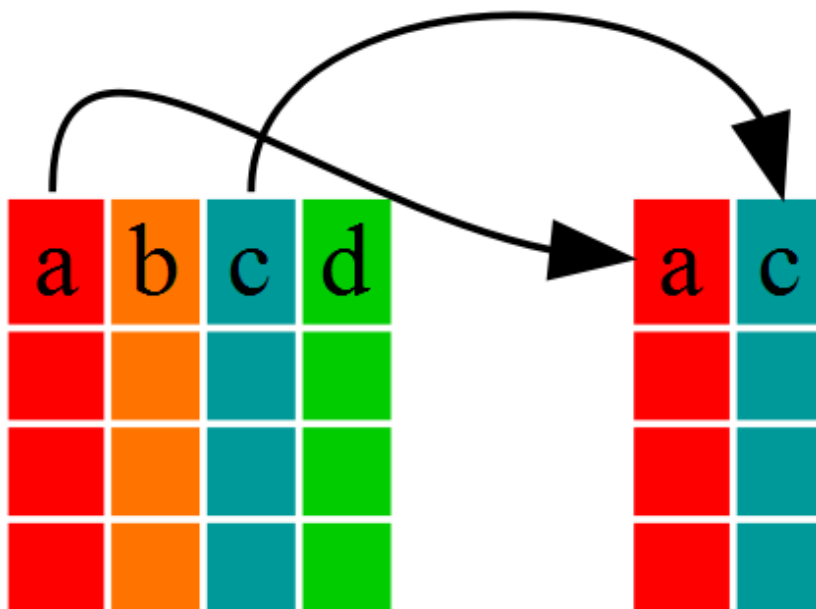
`select(data, ...)`



Função `select()`

Como seria o código para selecionar as colunas abaixo?

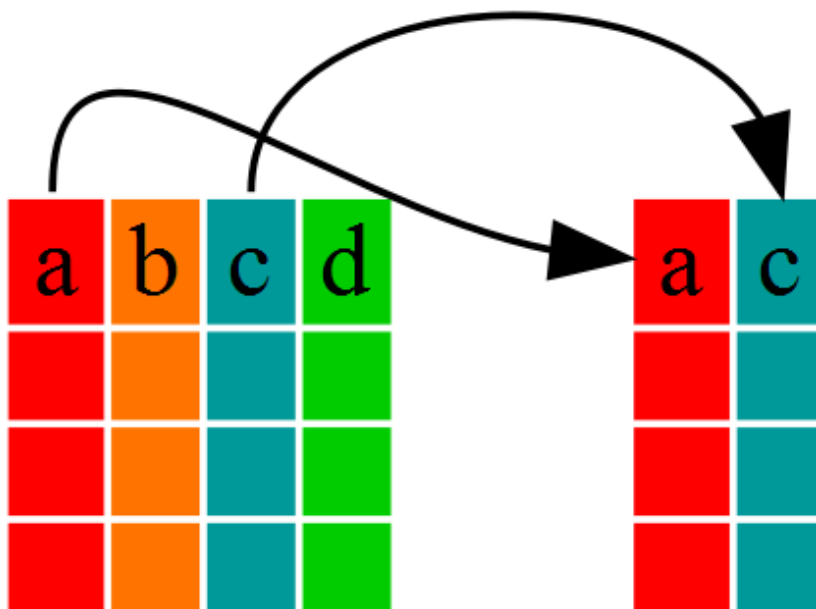
`select(data, ...)`



Função select()

Como seria o código para selecionar as colunas abaixo?

```
select(data.frame,a,c)
```



Função `select()`

Existem algumas funções auxiliares (“helper functions”) da função `select()` que são muito úteis:

- **`starts_with(x)`** = seleciona as colunas que iniciam com...
- **`ends_with(x)`** = seleciona as colunas que terminam com...
- **`contains(x)`** = seleciona as colunas que contém...
- **`matches(x)`** = seleciona as colunas que combinam com...
- **`one_of(x)`** = seleciona as colunas que de um grupo de nomes...

Exemplo: select(data, starts_with())

A tibble: 6 x 6

	country	continent	year	lifeExp	pop	gdpPercap
	<fct>	<fct>	<int>	<dbl>	<int>	<dbl>
1	Afghanistan	Asia	1952	28.8	8425333	779.
2	Afghanistan	Asia	1957	30.3	9240934	821.
3	Afghanistan	Asia	1962	32.0	10267083	853.
4	Afghanistan	Asia	1967	34.0	11537966	836.
5	Afghanistan	Asia	1972	36.1	13079460	740.
6	Afghanistan	Asia	1977	38.4	14880372	786.

```
select (gapminder, starts_with("c"))
```

A tibble: 6 x 2

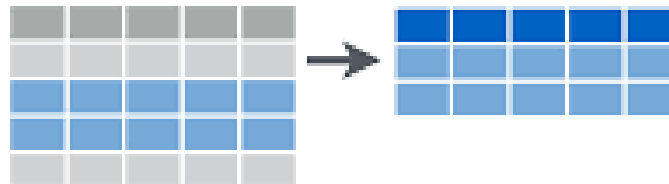
	country	continent
	<fct>	<fct>
1	Afghanistan	Asia
2	Afghanistan	Asia
3	Afghanistan	Asia
4	Afghanistan	Asia
5	Afghanistan	Asia
6	Afghanistan	Asia

Função `filter()`

A função **`filter()`** filtra linhas. Neste caso, utiliza-se muitas condições lógicas.

```
filter(dataframe, [condicoes logicas])
```


Subset Observations (Rows)




Função `filter()`

A função `filter()` filtra linhas. Neste caso, utiliza-se muitas condições lógicas.

```
filter(dataframe, [condicoes logicas])
```



Base de
dados que
você quer
filtrar



Condições
lógicas
especificando
que linhas
você quer ter
(>, <, ==, !=)
(&, |)


Exemplo função filter()

Quais condições lógicas foram utilizadas para subselecionar essa tabela?

filter()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Ana	40	1013	1997-07-01

Exemplo função filter()

Quais condições lógicas foram utilizadas para subselecionar essa tabela?

```
filter(storms, storm == alberto & storm ==Ana)
```

filter()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Ana	40	1013	1997-07-01

Função `mutate()`

A função **`mutate()`** cria ou modifica colunas. Novas variáveis devem ter o mesmo número de linhas da base original.

`mutate(data, ...)`

storm	wind	pressure	date		storm	wind	pressure	date	ratio
Alberto	110	1007	2000-08-12		Alberto	110	1007	2000-08-12	9.15
Alex	45	1009	1998-07-30		Alex	45	1009	1998-07-30	22.42
Allison	65	1005	1995-06-04		Allison	65	1005	1995-06-04	15.46
Ana	40	1013	1997-07-01		Ana	40	1013	1997-07-01	25.32
Arlene	50	1010	1999-06-13		Arlene	50	1010	1999-06-13	20.20
Arthur	45	1010	1996-06-21		Arthur	45	1010	1996-06-21	22.44

Exemplo função mutate()

Como seria o código para criar essa nova coluna?

mutate(data, ...)

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	wind	pressure	date	ratio
Alberto	110	1007	2000-08-12	9.15
Alex	45	1009	1998-07-30	22.42
Allison	65	1005	1995-06-04	15.46
Ana	40	1013	1997-07-01	25.32
Arlene	50	1010	1999-06-13	20.20
Arthur	45	1010	1996-06-21	22.44

Exemplo função mutate()

Como seria o código para criar essa nova coluna?

```
> mutate(storms, ratio = pressure/wind)
```

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	wind	pressure	date	ratio
Alberto	110	1007	2000-08-12	9.15
Alex	45	1009	1998-07-30	22.42
Allison	65	1005	1995-06-04	15.46
Ana	40	1013	1997-07-01	25.32
Arlene	50	1010	1999-06-13	20.20
Arthur	45	1010	1996-06-21	22.44

Exemplo função mutate()

Também se pode criar mais de uma coluna nova

```
> mutate(storms, ratio=pressure/wind, inverse=ratio^-1)
```

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	wind	pressure	date	ratio	inverse
Alberto	110	1007	2000-08-12	9.15	0.11
Alex	45	1009	1998-07-30	22.42	0.04
Allison	65	1005	1995-06-04	15.46	0.06
Ana	40	1013	1997-07-01	25.32	0.04
Arlene	50	1010	1999-06-13	20.20	0.05
Arthur	45	1010	1996-06-21	22.44	0.04

Exemplo função mutate()

Quais condições lógicas foram utilizadas para subselecionar essa tabela?

```
filter(storms, storm == alberto & storm ==Ana)
```

filter()

storm

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21




storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Ana	40	1013	1997-07-01

Função `arrange()`

A função **`arrange()`** ordena a base. O argumento *desc*= pode ser utilizado para gerar uma ordem decrescente.

```
> arrange(storms, wind)
```




	storm (chr)	wind (int)	pressure (int)	date (date)
1	Alberto	110	1007	2000-08-03
2	Alex	45	1009	1998-07-27
3	Allison	65	1005	1995-06-03
4	Ana	40	1013	1997-06-30
5	Arlene	50	1010	1999-06-11
6	Arthur	45	1010	1996-06-17

	storm (chr)	wind (int)	pressure (int)	date (date)
1	Ana	40	1013	1997-06-30
2	Alex	45	1009	1998-07-27
3	Arthur	45	1010	1996-06-17
4	Arlene	50	1010	1999-06-11
5	Allison	65	1005	1995-06-03
6	Alberto	110	1007	2000-08-03

Função `arrange()`

A função **`arrange()`** ordena a base. O argumento *desc* pode ser utilizado para gerar uma ordem decrescente.

```
> arrange(storms, desc(wind))
```



	storm (chr)	wind (int)	pressure (int)	date (date)
1	Alberto	110	1007	2000-08-03
2	Alex	45	1009	1998-07-27
3	Allison	65	1005	1995-06-03
4	Ana	40	1013	1997-06-30
5	Arlene	50	1010	1999-06-11
6	Arthur	45	1010	1996-06-17


	storm (chr)	wind (int)	pressure (int)	date (date)
1	Alberto	110	1007	2000-08-03
2	Allison	65	1005	1995-06-03
3	Arlene	50	1010	1999-06-11
4	Alex	45	1009	1998-07-27
5	Arthur	45	1010	1996-06-17
6	Ana	40	1013	1997-06-30

Função `arrange()`

Também se pode ordenar a partir de mais de uma coluna (variável):

```
> arrange(storms, wind, date)
```

	storm (chr)	wind (int)	pressure (int)	date (date)
1	Alberto	110	1007	2000-08-03
2	Alex	45	1009	1998-07-27
3	Allison	65	1005	1995-06-03
4	Ana	40	1013	1997-06-30
5	Arlene	50	1010	1999-06-11
6	Arthur	45	1010	1996-06-17



	storm (chr)	wind (int)	pressure (int)	date (date)
1	Ana	40	1013	1997-06-30
2	Arthur	45	1010	1996-06-17
3	Alex	45	1009	1998-07-27
4	Arlene	50	1010	1999-06-11
5	Allison	65	1005	1995-06-03
6	Alberto	110	1007	2000-08-03

Função `summarize()`

A função **`summarize()`** sumariza a base. Ela aplica uma função às variáveis, retornando um vetor. Geralmente é utilizada em conjunto com `group_by()`.

summarise()

city	particle size	amount ($\mu\text{g}/\text{m}^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56



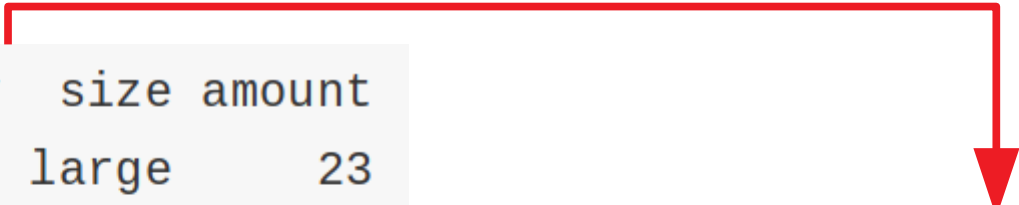
median	variance
22.5	1731.6

Função summarise()

Como funciona?

```
summarise(pollution, mediana = median(amount), variance = var(amount))
```

##		city	size	amount
##	1	New York	large	23
##	2	New York	small	14
##	3	London	large	22
##	4	London	small	16
##	5	Beijing	large	121
##	6	Beijing	small	56



##		mediana	variance
##	1	22.5	1731.6

Função `summarize()`

Pode operar com função do R-base:

```
summarise(pollution, mediana = median(amount), variance = var(amount))
```

	base
<code>min(), max()</code>	Valores max y min
<code>mean()</code>	media
<code>median()</code>	mediana
<code>sum()</code>	suma de los valores
<code>var, sd()</code>	varianza y desviación típica

Função summarize()

Mas também com funções do próprio dplyr:

```
summarise(pollution, mediana = median(amount), variance = var(amount))
```

	dplyr
first()	primer valor en un vector
last()	el último valor en un vector
n()	el número de valores en un vector
n_distinct()	el número de valores distintos en un vector
nth()	Extraer el valor que ocupa la posición n en un vector

Função `group_by()`

A função **`group_by()`** agrupa um conjunto de linhas selecionadas em um conjunto de linhas de resumo de acordo com os valores de uma ou mais colunas ou expressões.

```
> group_by(pollution, city)
```

city	particle size	amount ($\mu\text{g}/\text{m}^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56



city	particle size	amount ($\mu\text{g}/\text{m}^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

Função `group_by()`

A função **`group_by()`** agrupa um conjunto de linhas selecionadas em um conjunto de linhas de resumo de acordo com os valores de uma ou mais colunas ou expressões.

```
> group_by(pollut
```

city	particle size	amount ($\mu\text{g}/\text{m}^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56



A função `group_by()` trabalha bem com a `summarize()`

A função `group_by()` é extremamente útil trabalhando conjuntamente com a função `summarize()`.

```
> pollution %>% group_by(city) %>%  
+   summarise(mean = mean(amount), sum = sum(amount), n = n())
```

city	particle size	amount ($\mu\text{g}/\text{m}^3$)
New York	large	23
New York	small	14



city	mean	sum	n
New York	18.5	37	2

London	large	22
London	small	16



London	19.0	38	2
--------	------	----	---

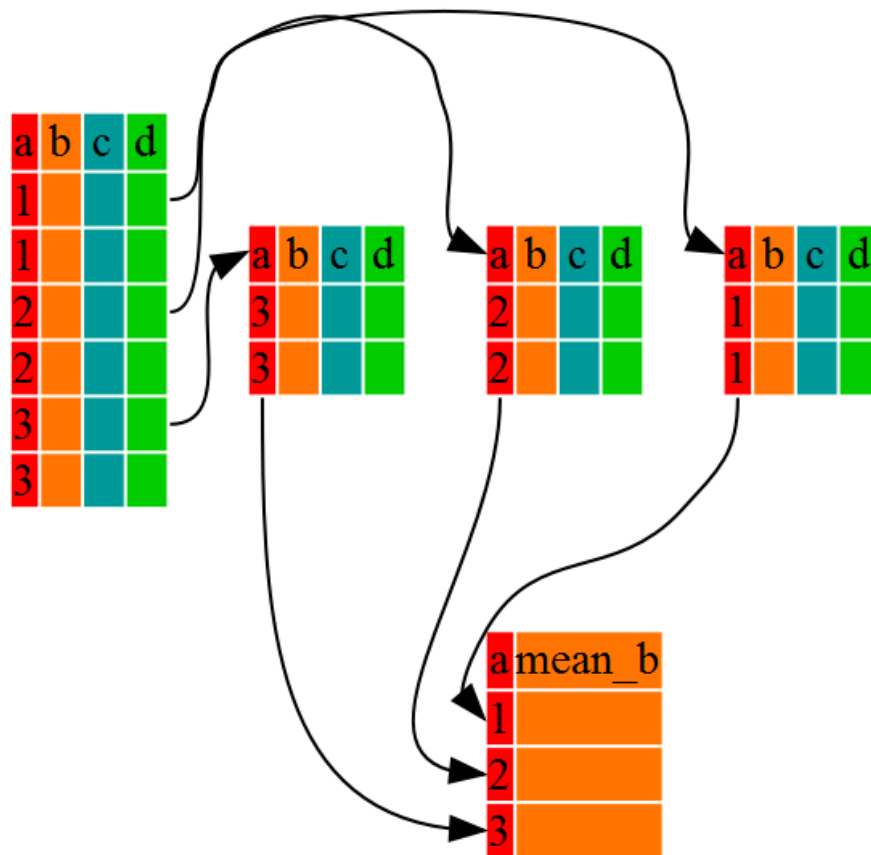
Beijing	large	121
Beijing	small	56



Beijing	88.5	177	2
---------	------	-----	---

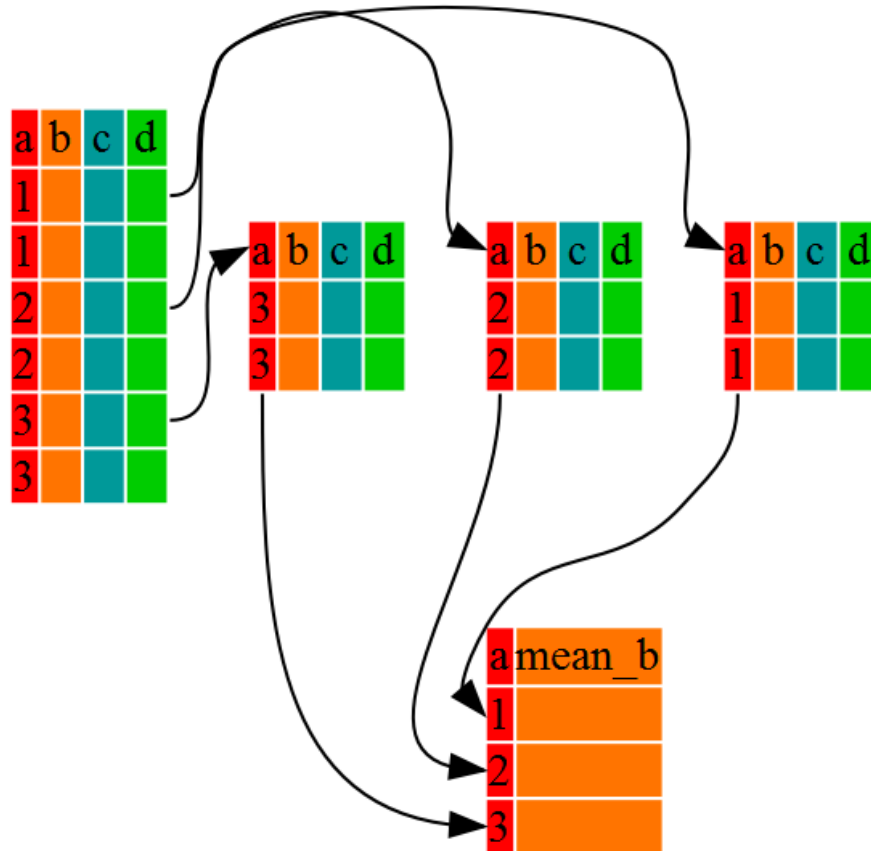
Como opera a função `group_by()`?

```
gapminder %>% group_by(a) %>% summarize(mean_b=mean(b))
```



Como opera a função group_by()?

gapminder %>% group_by(a) %>% summarize(mean_b = mean(b)) ???



Operador “pipe” %>%

- O operador %>% (pipe) foi uma das grandes revoluções recentes do R, tornando a leitura de códigos mais lógica, fácil e compreensível;
- Idéia simples: usa o valor resultante da expressão do lado esquerdo como 1o argumento da função do lado direito;
- Pipe = cano; condutor. Faz sentido?

```
# As duas linhas abaixo são equivalentes.
```

```
f(x, y)
```

```
x %>% f(y)
```

Operador “pipe” %>%

- Qual é mais lógico e fácil de entender?

```
third(second(first(x)))
```



```
first(x) %>% second(x) %>% third(x)
```

Operador “pipe” %>%

O operador pipeline %>% é útil para concatenar múltiplas operações do dplyr.

```
storms %>%  
  filter(wind>=50) %>%  
  select(storm, pressure)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	pressure
Alberto	1007
Allison	1005
Arlene	1010

Atalhos operador “pipe” %>%

Cmd + Shift + M

(Mac)

Ctrl + Shift + M

(Windows)

Vamos aos exercícios!

