Preparado para: AT autoridade tributária e aduaneira

REFORM/SC2022/126
DELIVERABLE 4

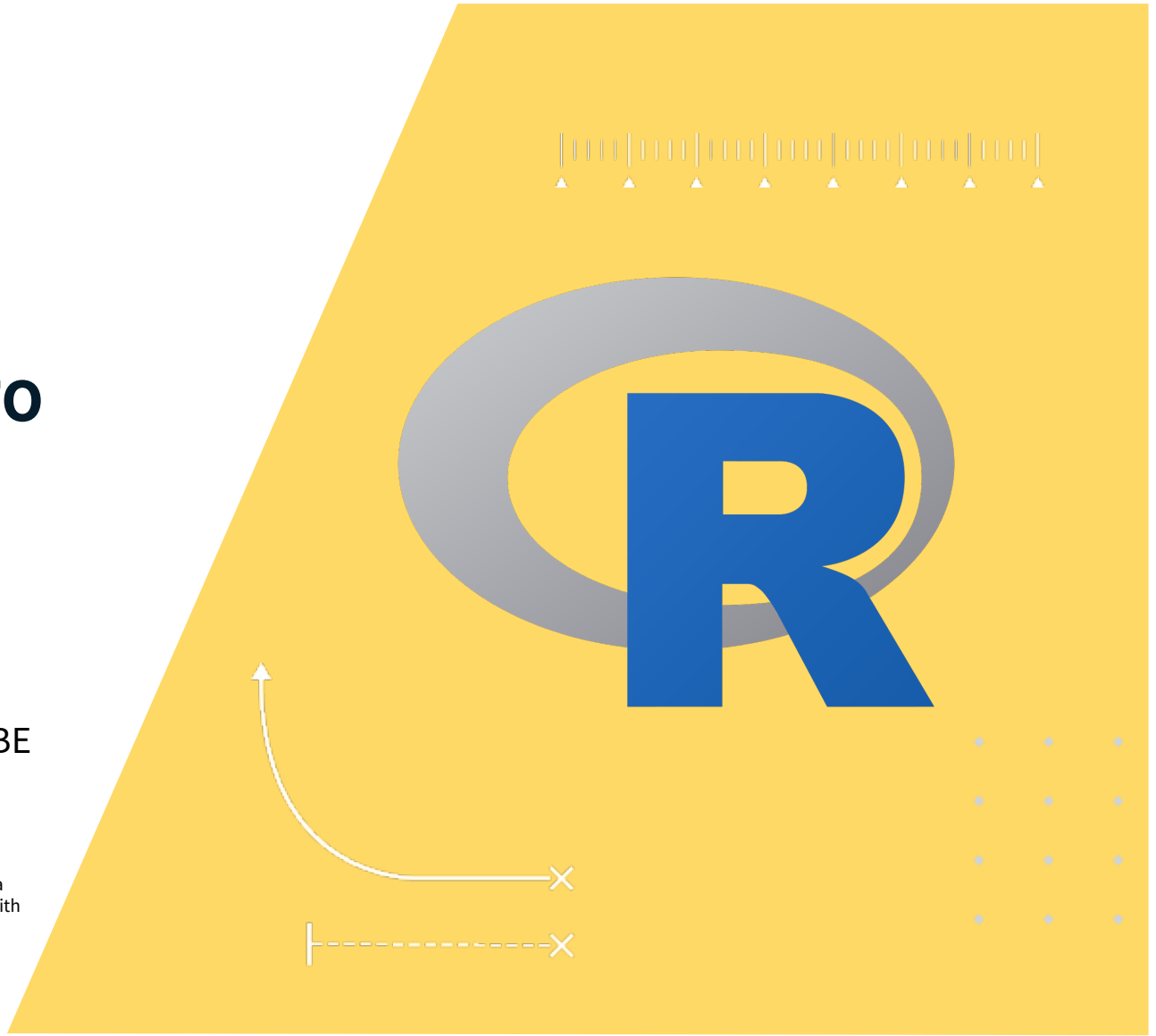
MÓDULO 2
GESTÃO E TRATAMENTO DE DADOS EM R

DESIGNING A NEW VALUATION MODEL
FOR RURAL PROPERTIES IN PORTUGAL

Parte II

Formador: Luís Teles Morais | Nova SBE

Lisboa, 14 junho 2023



This project is carried out with funding by the European Union via the Structural Reform Support Programme and in cooperation with the Directorate General for Structural Reform Support of the European Commission

AARC NOVA SCHOOL OF BUSINESS & ECONOMICS LOBO VASQUES esri INNERLANDS

Programa

Módulos	Duração
Módulo 1 – Introdução ao R: - O que é o R? - Como instalar e configurar o R. - Sintaxe básica e comandos. - Tipos de dados, objetos e classes.	**4 Horas**
Módulo 2 – Gestão e tratamento de dados em R: - Carregar dados no R. - Perceber as estruturas de dados e *subsetting*. - Limpeza de dados: *missing values, outliers* e transformações - Juntar bases de dados	**8 Horas**
Módulo 3 – Estatística básica em R: - Estatísticas descritivas: medidas de dispersão central e variação. - Distribuições probabilísticas: variáveis discretas e contínuas. - Testes de hipóteses.	**8 Horas**

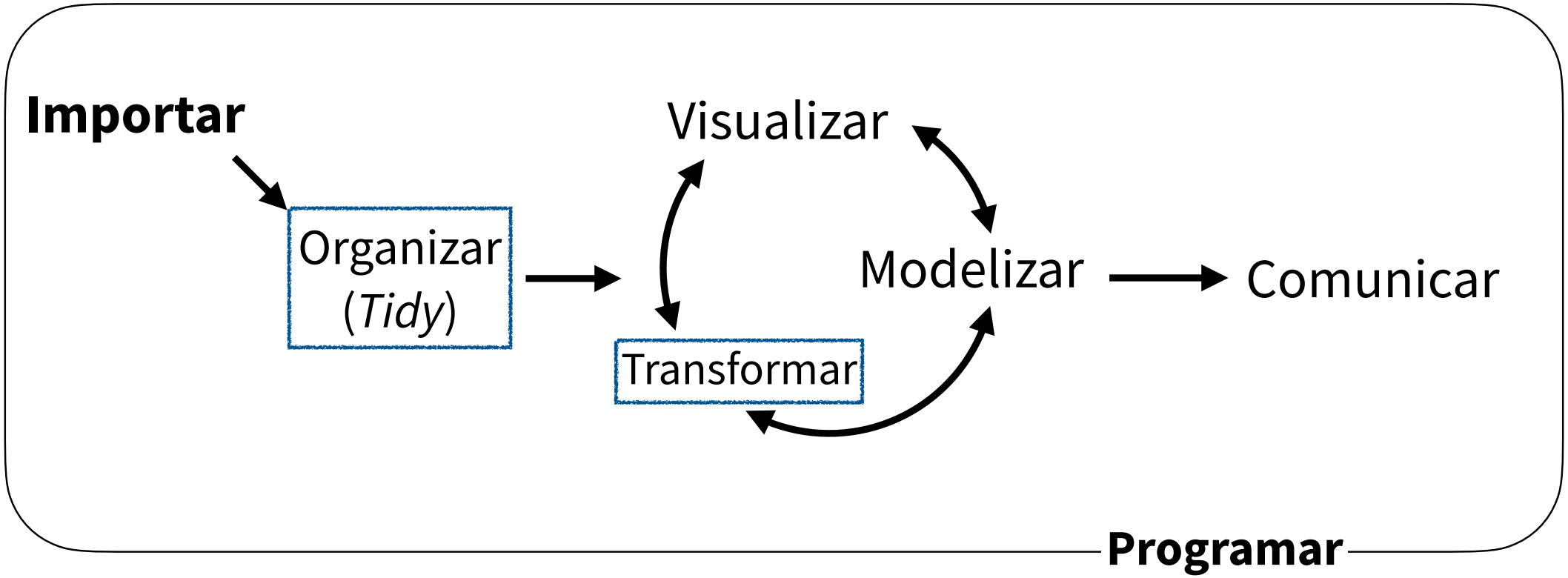
Módulos	Duração
Módulo 4 – Regressão Linear: - O modelo classico linear. - Estimação de parametros segundo o MMQ. - Testes de hipóteses: significância estatística e ajuste do modelo. - Modelo de regressão múltipla. - Testar as premissas: multicolinearidade, heteroscedasticidade e normalidade dos resíduos. - Critérios de seleção dos modelos.	**12 Horas**
Módulo 5 – O modelo: - Estrutura do modelo e premissas – Perceber o modelo (4 Hours). - Uso e tratamento dos dados (4 Hours). - Descrição do modelo (4 Hours). - Aplicação do modelo a cada piloto (12 Hours). - Aplicação autónoma do modelo a uma região (8 Hours).	**32 Horas**

Slides CC BY-SA RStudio

Transformação de dados (I)

Arrumar - *tidy*

Ciência de dados



Vamos a isso

Aceda a este link para começar já

https://posit.cloud/content/5906356

Back to **babynames**

	year	sex	name	n	prop
1	1880	F	Mary	7065	0.0724
2	1880	F	Margar	1578	0.0162
3	1880	F	Martha	1040	0.0107
4	1880	F	Marie	471	0.00483
5	1880	F	Maria	125	0.00128
1	1881	F	Mary	6919	0.0700
2	1881	F	Margar	1658	0.0168
3	1881	F	Martha	1044	0.0106
4	1881	F	Marie	499	0.00505
5	1881	F	Maria	120	0.00121
	1881	M	Gideon	7	0.0001

year	sex	name	n	prop
1880	F	Maria	125	0.00128
1881	F	Maria	120	0.00121
…	…	Maria	…	…

filter()

Extrai observações que cumpram todos os critérios fornecidos

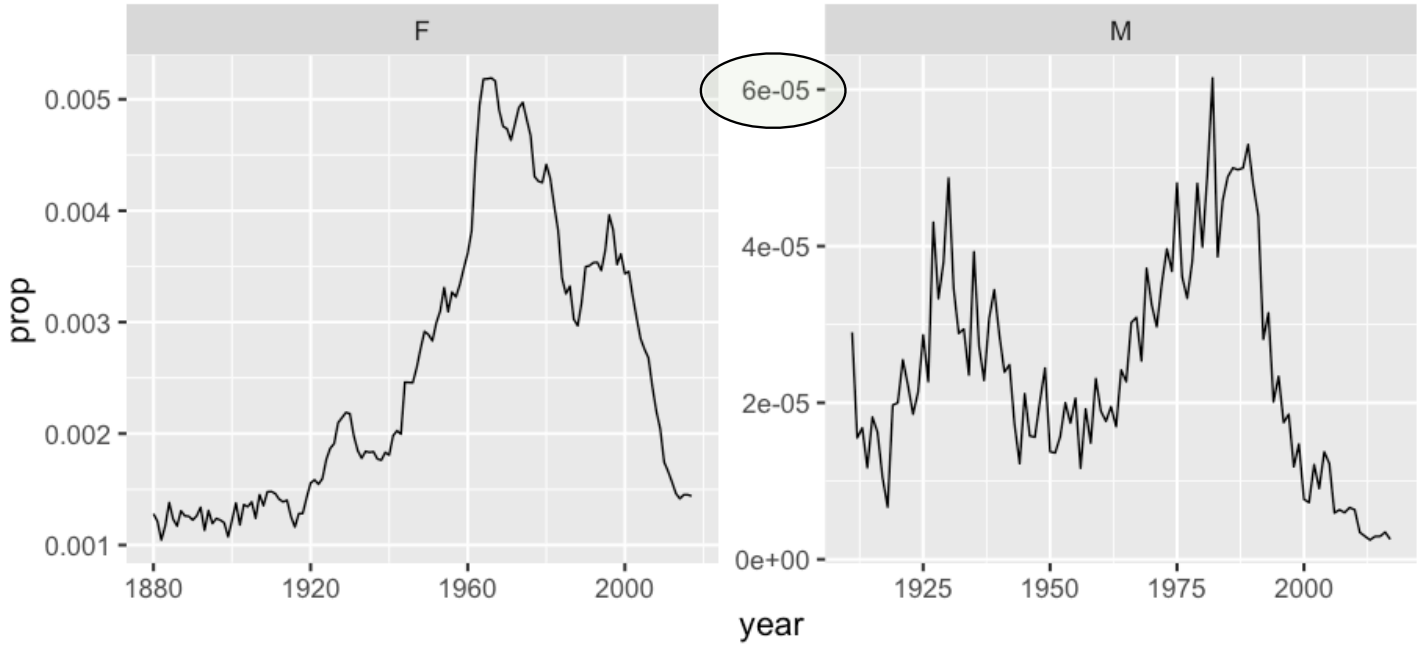
```
filter(babynames, name == "Maria" & year == 1880)
```

babynames

year	sex	name	n	prop
1880	F	Mary	7065	0.0724
1880	F	Margaret	1578	0.0162
1880	F	Martha	1040	0.0107
1880	F	Marie	471	0.00483
1880	F	Maria	125	0.00128
1881	F	Mary	6919	0.0700

year	sex	name	n	prop
1880	F	Maria	125	0.00128

```
babynames %>% filter(name == "Michael") %>%
  ggplot() +
    geom_line(mapping = aes(year, prop)) +
    facet_wrap(~ sex)
```



Quais os nomes mais populares?

Quiz

Será que conseguimos:

1. calcular o número total de bebés de cada nome?

Transformação de dados (II)

Transformar para derivar nova informação

Derivar nova informação

summarise() - resumir **variáveis**
group_by() - agrupar por **casos**
mutate() - criar novas **variáveis**

summarise()

e.g. medidas de tendência central (média, mediana) ou dispersão (desvio padrão)
+ na próxima aula!

Calcular tabela de medidas sumárias:

```
babynames %>% summarise(total = sum(n), max = max(n))
```

babynames

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

total	max
348120517	99686

Medidas sumárias em summarise()

Input: vector

Output: valor

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

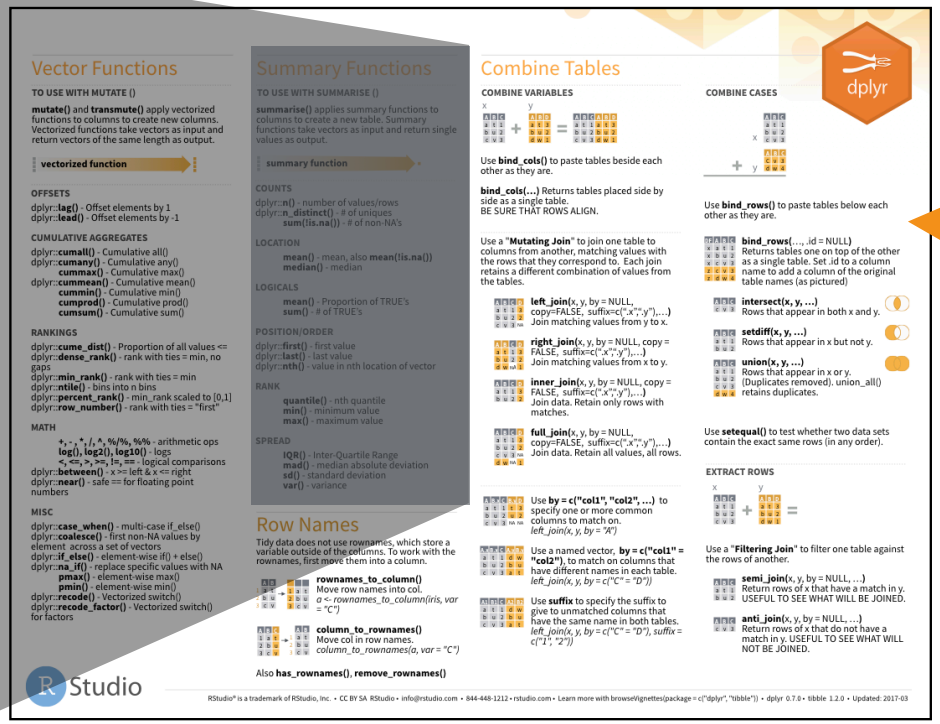
dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance



on back

n()

Número de observações num conjunto de dados

```
babynames %>% summarise(n = n())
```

babynames

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

n
1924665

n_distinct()

Número de casos únicos contidos numa variável (sem repetições)

```
babynames %>% summarise(n = n(), nname = n_distinct(name))
```

babynames

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

n	nname
1924665	97310

Experimente

Numa única operação, encontre as observações com o nome **“Melo”** e utilize **summarise(), sum()** e **min()** para encontrar:

1. O número total de bebés com o nome Melo
2. O primeiro ano (**year**) em que nasceu um bebé com este nome

```
babynames %>%
  filter(name == "Melo") %>%
  summarise(total = sum(n), first = min(year))
# A tibble: 1 × 2
  total first
  <dbl> <dbl>
1    73  2012
```

Quais são os nomes mais populares?

Nomes mais populares = elevado número de recém-nascidos

1. **Soma** - número de bebés com esse nome, somando as observações de todos os anos disponíveis

```
babynames %>%
  filter(name == “Melo")
# A tibble: 6 × 5
   year sex   name      n       prop
  <dbl> <chr> <chr> <dbl>      <dbl>
1  2012 M     Melo      5 0.00000247
2  2013 M     Melo     11 0.00000545
3  2014 M     Melo      8 0.00000391
4  2015 M     Melo     15 0.00000736
5  2016 M     Melo     16 0.00000793
6  2017 M     Melo     18 0.00000917
```

Não parece muito popular?

```
babynames %>%
  filter(name == “Maria”, sex == 'F')
# A tibble: 138 × 5
    year sex   name      n    prop
   <dbl> <chr> <chr> <dbl>   <dbl>
 1  1880 F     Maria   125 0.00128
 2  1881 F     Maria   120 0.00121
 3  1882 F     Maria   121 0.00105
 4  1883 F     Maria   141 0.00117
 5  1884 F     Maria   190 0.00138
# … with 133 more rows
# ℹ Use `print(n = ...)` to see more rows
```

Como comparar com outros?

```
babynames %>%
  filter(name == "Melo" & sex == "M") %>%
  summarise(total = sum(n))
  total
  <dbl>
1    73
babynames %>%
  filter(name == "Maria" & sex == "F") %>%
  summarise(total = sum(n))
   total
   <dbl>
1 543324
```

group_by()

Agrupa observações em função dos diferentes **casos** de uma ou mais **variável/is**

```
babynames %>%
  group_by(sex) %>%
  summarise(total = sum(n))
```

sex	total
F	172371079
M	175749438

group_by()

```
babynames %>%
  group_by(name) %>%
  summarise(total = sum(n))
```

Também ordena pela variável grupo

sex	total
Aaban	107
Aabha	35
Aabid	10
Aabir	5
Aabriella	22

ungroup()

Remove critérios de group_by

```
babynames %>%
  group_by(sex) %>%
  ungroup() %>%
  summarise(total = sum(n))
```

total
348120517

Experimente

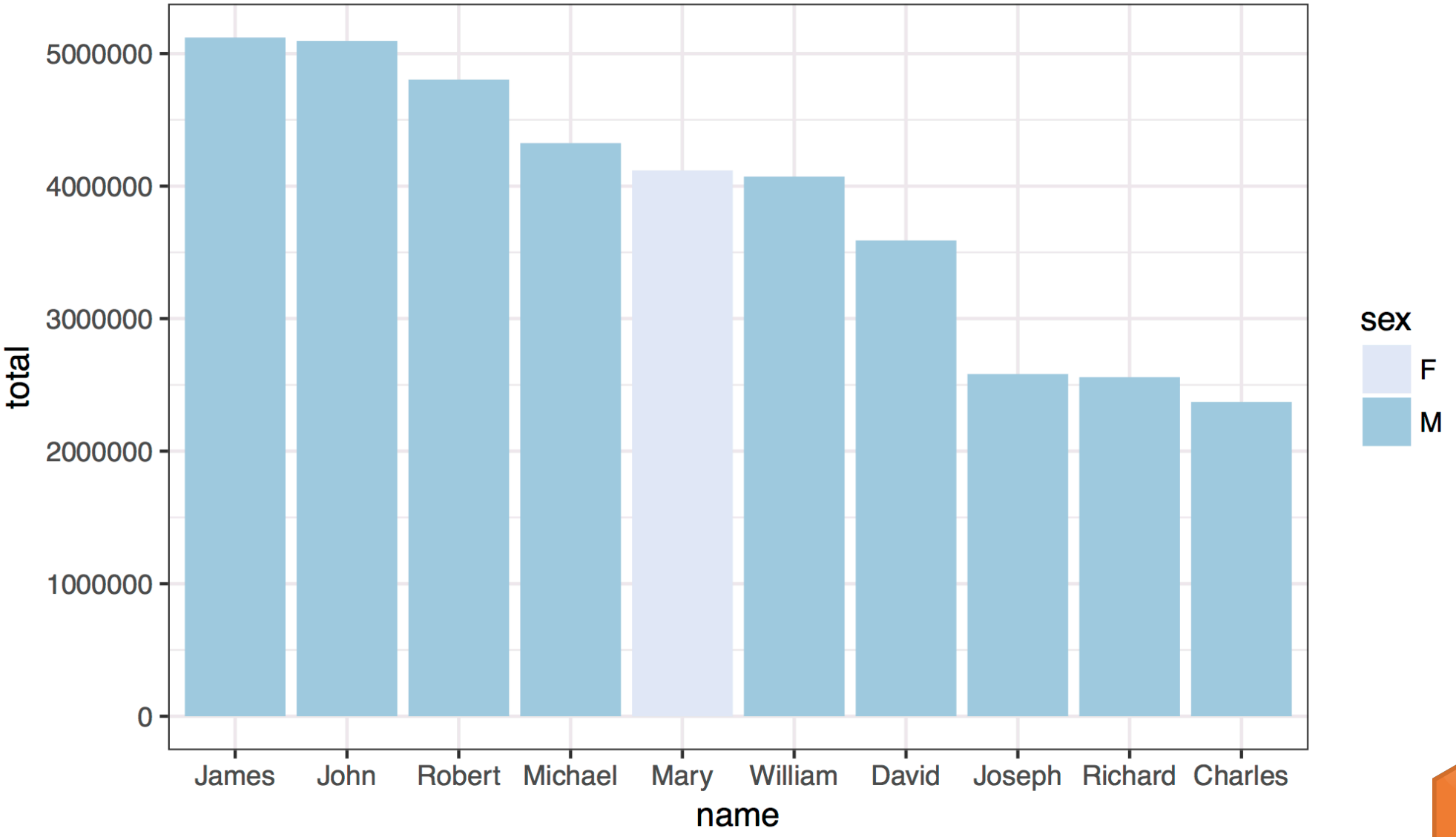
Utilize uma combinação de **group_by()**, **summarise()**, e **arrange()** para ver as 10 combinações mais populares de nome (name) e sexo (**sexo**). Calcule a popularidade como o número *total* de bebés com cada par nome/sexo.

```
babynames %>%
   group_by(name, sex) %>%
   summarise(total = sum(n)) %>%
   arrange(desc(total))
# A tibble: 107,973 × 3
# Groups:   name [97,310]
   name    sex     total
   <chr>   <chr>   <dbl>
 1 James   M     5150472
 2 John    M     5115466
 3 Robert  M     4814815
 4 Michael M     4350824
 5 Mary    F     4123200
# … with 107,968 more rows
```

10+?

```
babynames %>%
  group_by(name, sex) %>%
  summarise(total = sum(n)) %>%
  arrange(desc(total)) %>%
  ungroup %>% slice(1:10)
```

```
# A tibble: 10 × 3
   name    sex     total
   <chr>   <chr>   <dbl>
 1 James   M     5150472
 2 John    M     5115466
 3 Robert  M     4814815
 4 Michael M     4350824
 5 Mary    F     4123200
 6 William M     4102604
...
```


5000000
4000000
3000000
2000000
1000000
0
total
James
John
Robert
Michael
Mary
William
David
Joseph
Richard
Charles
name
sex
F
M

```
babynames %>%
  group_by(name, sex) %>%
  summarise(total = sum(n)) %>%
  arrange(desc(total)) %>%
  ungroup() %>%
  slice(1:10) %>%
  ggplot() +
    geom_col(mapping = aes(x = fct_reorder(name,
      desc(total)), y = total, fill = sex)) +
    theme_bw() +
    scale_fill_brewer() +
    labs(x = "name")
```

Quais são os nomes mais populares?

Nomes mais populares = elevado número de recém-nascidos

1. **Soma** - número de bebés com esse nome, somando as observações de todos os anos disponíveis

Enviesado? Porquê?

```
babynames %>%
  filter(year == 2017) %>%
  summarise(total = sum(n))
##     total
## 1 3546301
```

3,546,301

```
babynames %>%
  filter(year == 1880) %>%
  summarise(total = sum(n))
##     total
## 1  201484
```

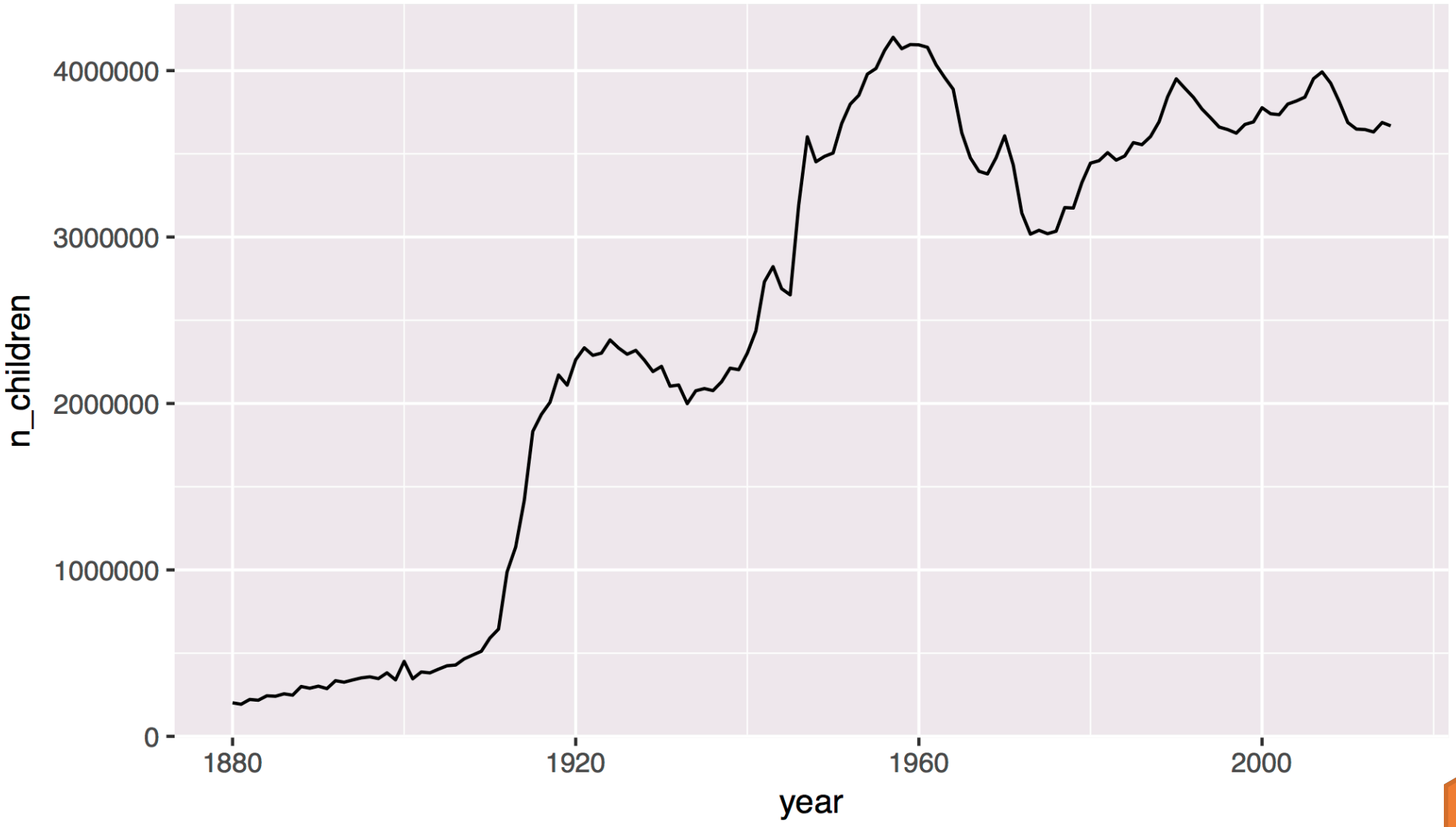
201,484

Experimente

Use group_by() para calcular o número total de bebés registados **em cada ano**.

Mostre os resultados como um gráfico de linhas, uma série temporal

```
babynames %>%
  group_by(year) %>%
  summarise(n_children = sum(n)) %>%
  ggplot() +
    geom_line(mapping = aes(x = year, y = n_children))
```


4000000
3000000
2000000
1000000
0
n_children
1880
1920
1960
2000
year

Qual o nome mais popular em cada ano?

Quais são os nomes mais populares?

Nomes mais populares = elevado número de recém-nascidos

1. **Soma** - número de bebés com esse nome, somando as observações de todos os anos disponíveis
2. **Ordem** - ranking/posição entre os nomes mais populares *em cada ano*

Será que temos informação suficiente para **ordenar os nomes pela sua popularidade em cada ano?**

mutate()

Comando geral - cria novas variáveis/colunas

```
babynames %>%
  mutate(percent = round(prop*100, 2))
```

babynames

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

year	sex	name	n	prop	percent
1880	M	John	9655	0.0815	8.15
1880	M	William	9532	0.0805	8.05
1880	M	James	5927	0.0501	5.01
1880	M	Charles	5348	0.0451	4.51
1880	M	Garrett	13	0.0001	0.01
1881	M	John	8769	0.081	8.1

mutate()

```
babynames %>%
  mutate(percent = round(prop*100, 2), nper = round(percent))
```

babynames

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

year	sex	name	n	prop	percent	nper
1880	M	John	9655	0.0815	8.15	8
1880	M	William	9532	0.0805	8.05	8
1880	M	James	5927	0.0501	5.01	5
1880	M	Charles	5348	0.0451	4.51	5
1880	M	Garrett	13	0.0001	0.01	0
1881	M	John	8769	0.081	8.1	8

```
babynames %>%
  mutate(rank =              ?              )
```

min_rank()

Função dá a posição ordinal de cada valor

```
min_rank(c(50, 100, 100, 1000))
# [1] 1 2 2 4
```

(em caso de empate, posição seguinte é partilhada)

```
min_rank(desc(c(50, 100, 100, 1000)))
# [1] 4 2 2 1
```

Experimente

Utilize **mutate()** e **min_rank()** para ordenar as observações pela sua frequência (**n**), da maior para a mais pequena

```
babynames %>%
  mutate(rank = min_rank(desc(prop)))


## A tibble: 1,924,665 x 6
#    year sex   name           n   prop  rank
# 1  1880 F     Mary        7065 0.0724    14
# 2  1880 F     Anna        2604 0.0267   709
# 3  1880 F     Emma        2003 0.0205  1131
# 4  1880 F     Elizabeth   1939 0.0199  1192
# 5  1880 F     Minnie      1746 0.0179  1427
# 6  1880 F     Margaret    1578 0.0162  1683
## … with 1,924,659 more rows
```

Experimente

Agrupe os dados por ano e volte a ordená-los por frequência. Filtre os resultados para manter apenas os nomes mais populares em cada ano

```
babynames %>%
  group_by(year) %>%
  mutate(rank = min_rank(desc(prop))) %>%
  filter(rank == 1)


# A tibble: 138 x 6
# Groups:   year [138]
    year sex   name       n   prop  rank
 1  1880 M     John    9655 0.0815     1
 2  1881 M     John    8769 0.0810     1
 3  1882 M     John    9557 0.0783     1
# … with 135 more rows
```

Funções vetoriais

Input: vector

Output: vector da mesma dimensão (*length*)

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

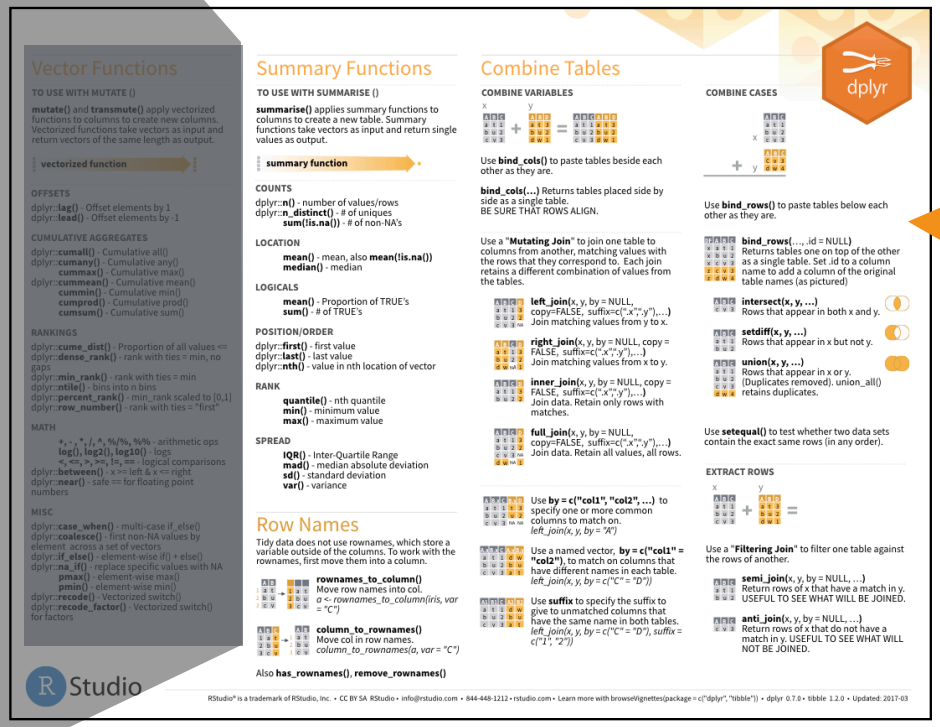
dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

MISC

dplyr::**case_when()** - multi-case if_else()
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors



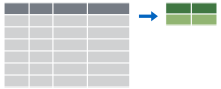
Resumo: verbos dplyr

(Verbos para uma tabela única)

Selecionar variáveis: **select()**

Extrair casos específicos: **filter()**

Reordenar tabela: **arrange()**



Obter medidas sumárias: **summarise()**

Criar novas variáveis: **mutate()**

Transformação de dados (III)

Reorientar dados (*pivot*)

sales.xlsx

```
clientes <- readxl::read_xlsx("sales_data/sales.xlsx", sheet = 'clientes')
clientes
```

```
## # A tibble: 2 × 4
##   id_cliente item_1 item_2          item_3
##        <dbl> <chr>  <chr>           <chr>
## 1          1 pao    leite           banana
## 2          2 leite  papel higienico <NA>
```

```
precos <- readxl::read_xlsx("sales_data/sales.xlsx", sheet = 'precos')
precos
```

```
## # A tibble: 5 × 2
##   item            price
##   <chr>           <dbl>
## 1 abacate           2
## 2 banana            0.5
## 3 pao               1.5
## 4 leite             1
## 5 papel higienico   3
```

clientes

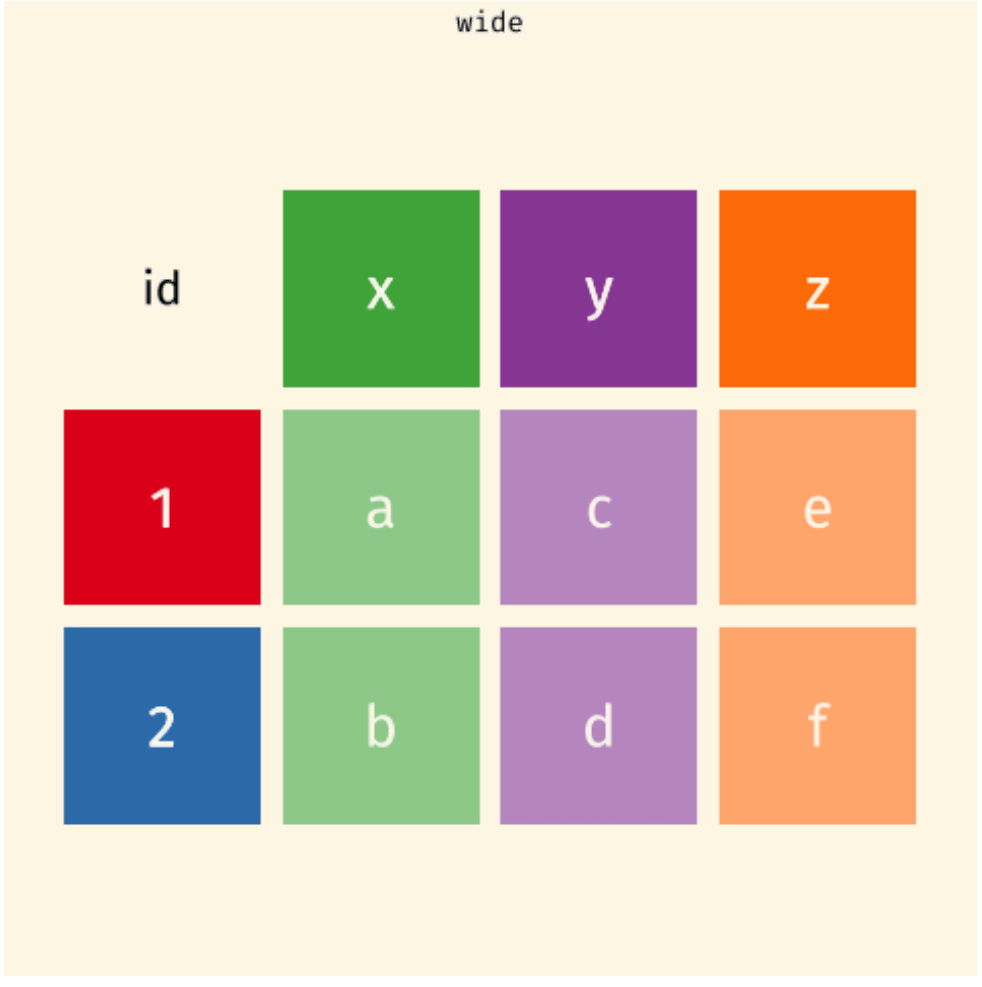
Temos...

```
## # A tibble: 2 × 4
##   id_cliente item_1 item_2          item_3
##        <dbl> <chr>  <chr>          <chr>
## 1          1 pao    leite          banana
## 2          2 leite  papel higienico <NA>
```

Queremos...

```
## # A tibble: 6 × 3
##   id_cliente item_no item
##        <dbl> <chr>   <chr>
## 1          1 item_1  pao
## 2          1 item_2  leite
## 3          1 item_3  banana
## 4          2 item_1  leite
## 5          2 item_2  papel higienico
## 6          2 item_3  <NA>
```

O objetivo



Wide vs. long

wide

mais colunas / variáveis

```
## # A tibble: 2 × 4
##   id_cliente item_1 item_2          item_3
##        <dbl> <chr>  <chr>          <chr>
## 1          1 pao    leite          banana
## 2          2 leite  papel higienico <NA>
```

long

mais linhas / observações

```
## # A tibble: 6 × 3
##   id_cliente item_no item
##        <dbl> <chr>   <chr>
## 1          1 item_1  pao
## 2          1 item_2  leite
## 3          1 item_3  banana
## 4          2 item_1  leite
## 5          2 item_2  papel higienico
## 6          2 item_3  <NA>
```

pivot_longer()

- data

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  values_to = "value"
  )
```

`pivot_longer()`

- `data`
- `cols`: colunas a transpor para formato long

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  values_to = "value"
  )
```

`pivot_longer()`

- `data`
- `cols`: colunas a transpor para formato long
- `names_to`: nome da variável que vai receber os nomes das colunas a transpor para long, como valores

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  values_to = "value"
  )
```

`pivot_longer()`

- `data`
- `cols`: colunas a transpor para formato long
- `names_to`: nome da variável que vai receber os nomes das colunas a transpor para long, como valores
- `values_to`: nome da variável que vai receber os valores atualmente dispersos por várias colunas (string)

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  values_to = "value"
  )
```

clientes → compras

```
compras <- clientes %>%
  pivot_longer(
    cols = item_1:item_3,  # variables item_1 to item_3
    names_to = "item_no",  # column names -> new column called item_no
    values_to = "item"     # values in columns -> new column called item
    )

compras
```

```
## # A tibble: 6 × 3
##   id_cliente item_no item
##        <dbl> <chr>   <chr>
## 1          1 item_1  pao
## 2          1 item_2  leite
## 3          1 item_3  banana
## 4          2 item_1  leite
## 5          2 item_2  papel higienico
## 6          2 item_3  <NA>
```

Exemplo da importância de dados tidy

Várias operações de transformação requerem-no (e.g. *join* - prox. aula)

```
precos
```

```
## # A tibble: 5 × 2
##   item            price
##   <chr>           <dbl>
## 1 abacate           2
## 2 banana            0.5
## 3 pao               1.5
## 4 leite             1
## 5 papel higienico   3
```

```
compras %>%
  left_join(precos)
```

```
## # A tibble: 6 × 4
##   id_cliente item_no item            price
##        <dbl> <chr>   <chr>           <dbl>
## 1          1 item_1  pao               1.5
## 2          1 item_2  leite             1
## 3          1 item_3  banana            0.5
## 4          2 item_1  leite             1
## 5          2 item_2  papel higienico   3
## 6          2 item_3  <NA>             NA
```

compras → clientes

- `data`
- `names_from`: variável em formato long a transpor para nomes de novas colunas
- `values_from`: variável em formato long que contêm valores a dispersar por várias colunas no formato wide

```
compras %>%
  pivot_wider(
    names_from = item_no,
    values_from = item
  )
```

```
## # A tibble: 2 × 4
##   id_cliente item_1 item_2          item_3
##        <dbl> <chr>  <chr>          <chr>
## 1          1 pao    leite          banana
## 2          2 leite  papel higienico <NA>
```

Transformação de dados (IV)

Juntar bases de dados

band & instrument

```
band <- tribble(
   ~name,     ~band,
  "Mick",  "Stones",
  "John", "Beatles",
  "Paul", "Beatles"
)
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

```
instrument <- tribble(
    ~name,   ~plays,
   "John", "guitar",
   "Paul",   "bass",
  "Keith", "guitar"
)
```

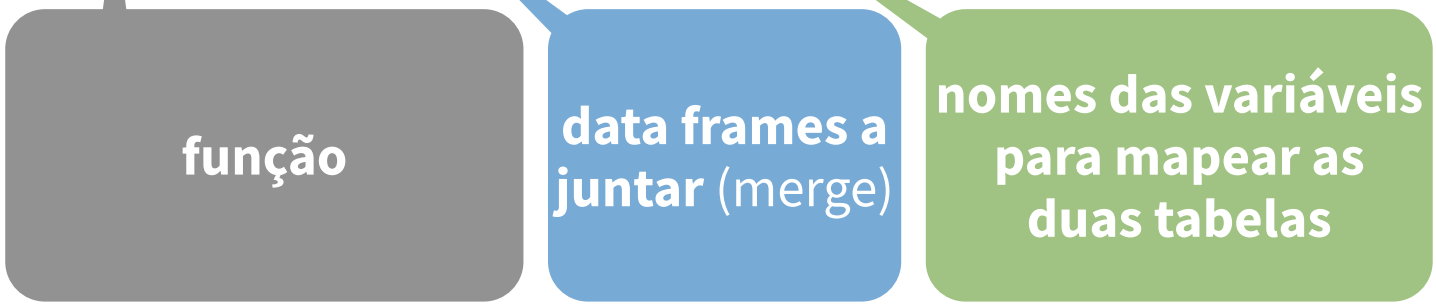
instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

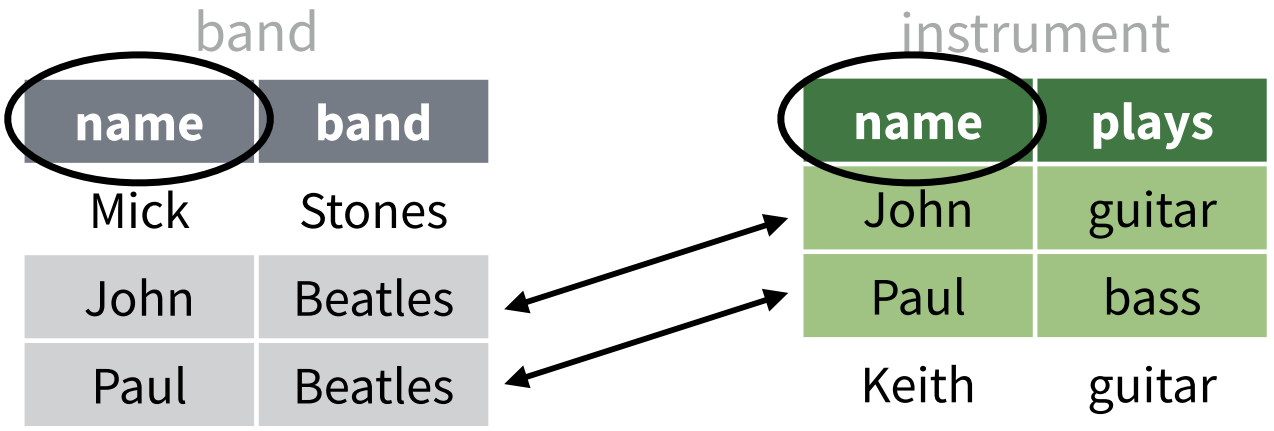
_join - sintaxe

Cada função **join** tem como input duas tabelas, e output uma única

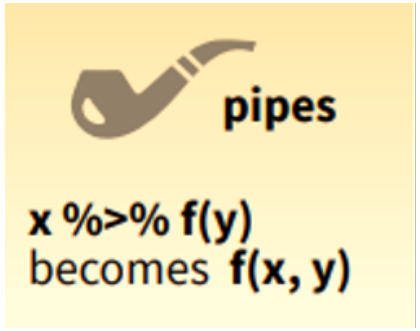
```
left_join(x, y, by = NULL, … )
```

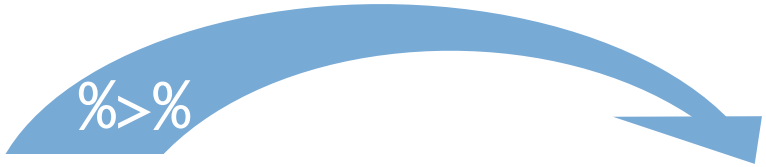


Juntar por nome



Operador pipe %>%






```
instrument   %>%   left_join(_____, by = "name")
```

Passa resultado da esquerda à função da direita, para o seu primeiro argumento. Experimente:

```
left_join(instrument, by = "name")
instrument %>% left_join(by = "name")
```

left

```
band %>% left_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
Mick	Stones	<NA>
John	Beatles	guitar
Paul	Beatles	bass

right

```
band %>% right_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
John	Beatles	guitar
Paul	Beatles	bass
Keith	<NA>	guitar

full

```
band %>% full_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
Mick	Stones	<NA>
John	Beatles	guitar
Paul	Beatles	bass
Keith	<NA>	guitar

inner

```
band %>% inner_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
John	Beatles	guitar
Paul	Beatles	bass

Verbos join

left_join() mantém todos os casos da tabela da **esquerda**

right_join() mantém todos os casos da tabela da **direita**

full_join() mantém todos os casos de **qualquer** tabela

inner_join() mantém apenas os casos em comum de **ambas**

semi_join() extrai os cases em que **há correspondência**

anti_join() extrai os cases em que **não há correspondência**

Mini-teste I

Hip.: os bebés de 1969 com o nome de um destes ícones da música tocarão o respetivo instrumento. (Os outros não tocarão nenhum.)

1. Use um **join** para combinar as tabelas **babynames** e **instruments**.
2. Calcule a percentagem de recém-nascidos nesse ano que tocará algum instrumento.

Mini-teste II

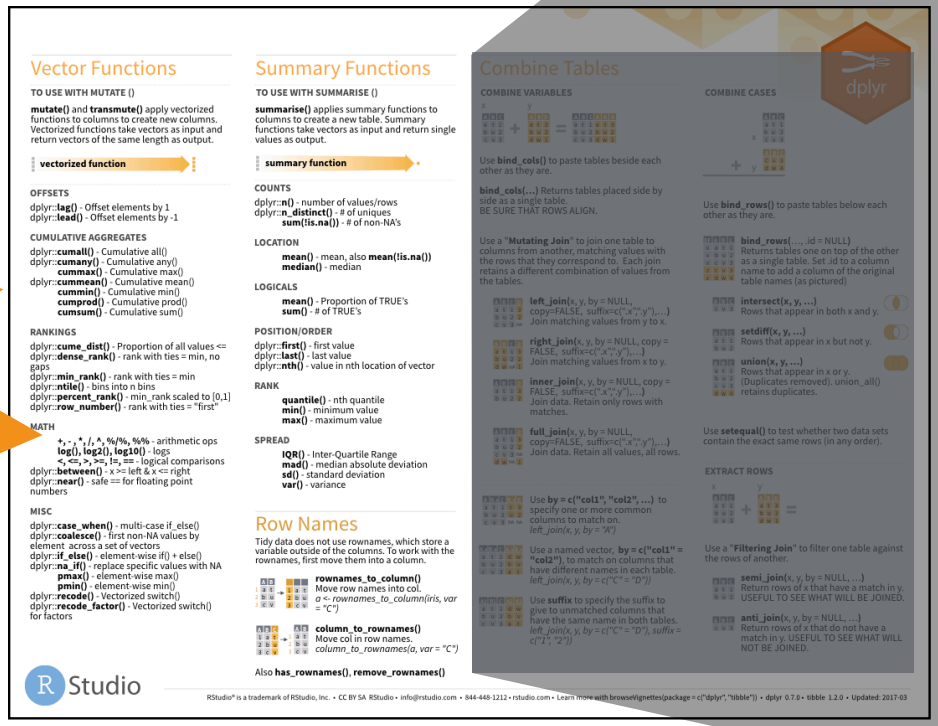
Quanto se gastou em cada categoria de bens?

1. Utilize **read_excel** (do pacote **readxl**) e importe o ficheiro Excel **sales2.xlsx** (ambas as folhas).
2. Reoriente a tabela de **clientes** para formato ***long***, de forma a ficar com 4 variáveis: **id_cliente, item, categoria e num_item**.
   - *Dica: vai precisar da opção **names_sep***
3. Junte as tabelas de **preços** e **clientes** numa única.
4. Calcule o montante gasto por categoria. Cada cliente comprou apenas 1 unidade de cada item.

join

Data Transformation cheatsheet

on back



Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , * , /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

MISC

dplyr::**case_when()** - multi-case if_else()
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

column_to_rownames()
Move col in row names.
column_to_rownames(a, var = "C")

Also **has_rownames()**, **remove_rownames()**

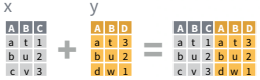
RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.7.0 • tibble 1.2.0 • Updated: 2017-03

Combine Tables

dplyr

COMBINE VARIABLES



Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(…) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…)
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…)
Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…)
Join data. Retain only rows with matches.

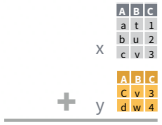
full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…)
Join data. Retain all values, all rows.

Use **by = c("col1", "col2", …)** to specify one or more common columns to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES



Use **bind_rows()** to paste tables below each other as they are.

bind_rows(…, .id = NULL)
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

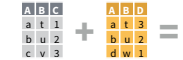
intersect(x, y, …)
Rows that appear in both x and y.

setdiff(x, y, …)
Rows that appear in x but not y.

union(x, y, …)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS



Use a "**Filtering Join**" to filter one table against the rows of another.

semi_join(x, y, by = NULL, …)
Return rows of x that have a match in y.
USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, …)
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

dplyr

Quiz 2

Obrigado
e até amanhã!

luis.morais@novasbe.pt