

# Big Data Analytics

Transformações de dados



## Agenda

- Transformações de dados



# Transformações de dados



## Transformações de dados

- Focaremos nosso estudo nesta etapa nas funções de transformação de dados.
- São funções que manipulam dataframes, talvez o mais importante objeto da linguagem R!



## Funções *apply*

- **Aplicar função nas dimensões:** a função `apply()` aplica funções nas linhas, colunas ou outras dimensões de uma matriz, *dataframe* ou *array*.



## Funções *apply*

- **Aplicar função nos elementos:** para aplicar uma função a cada elemento de um objeto, podemos usar `lapply()` ou `sapply()`. A diferença entre elas é o formato do resultado.
  - `lapply()` retorna uma lista;
  - `sapply()` tenta simplificar o resultado para um objeto mais simples (como um vetor ou matriz).



## Funções *apply*

- **Aplicar funções em múltiplos elementos:** a função `mapply()` pode ser considerada uma versão multivariada do `sapply()`.
- A `mapply()` aplica uma função em todos elementos de múltiplos objetos ao mesmo tempo.



## Funções *apply*

- **Repetir código em simulações:** para isso temos a função `replicate()`, que replica uma expressão diversas vezes.



## Quando usar?

- O `sapply()` é para uso interativo. Quando você está explorando uma base de dados, o `sapply()` facilita seu trabalho tentando simplificar o resultado da operação.
- No entanto, o `sapply()` não te dá sempre o mesmo resultado (às vezes pode ser um vetor, às vezes uma lista), e isso pode ser perigoso para usar em funções.



## Quando usar?

- A função `lapply()` é mais previsível que o `sapply()`, ela sempre vai te retornar uma lista.
- Neste caso, você vai ter o trabalho de simplificar o resultado manualmente, mas não terá a surpresa de vir um resultado em um formato diferente do que você esperava.



## Função *replicate*

- A função **replicate()** é uma função de conveniência para repetir a execução de uma expressão diversas vezes no R.
- É bastante utilizada para simulações, como por exemplo a simulação de Monte Carlo.



## Função *mapply*

- A função **mapply()** pode ser considerada uma **sapply()** “multivariada”.
- Exemplo: a instrução **mapply(funcao, x, y, z)** é equivalente a:

```
funcao(x[1], y[1], z[1])
funcao(x[2], y[2], z[2])
funcao(x[3], y[3], z[3])
funcao(x[4], y[4], z[4])
...
funcao(x[n], y[n], z[n])
funcao(x[n], y[n], z[n])
```



## Resumo

- **apply()** : aplica função nas dimensões (linha, coluna, etc.) do objeto.
- **lapply()** : aplica função em todos elementos do objeto. Retorna uma lista.
- **sapply()** : similar a **lapply()** mas tenta simplificar resultado.
- **mapply()** : versão multivariada do **sapply()**. Aplica função a todos elementos de vários objetos.
- **replicate()** : replica expressão um número pré-estabelecido de vezes.



## Funções *apply*

- Exemplos no RStudio.

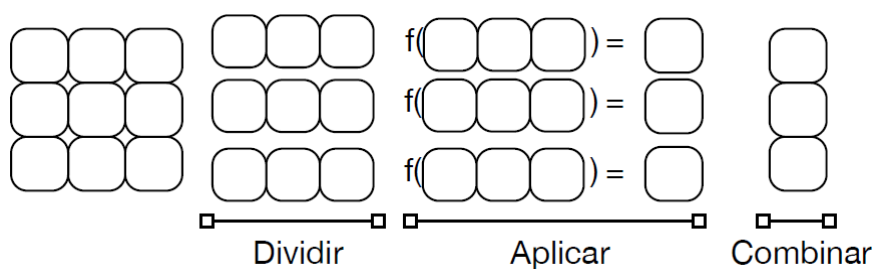


## Manipulação de *dataframes*

- Vamos ver agora mais instruções de manipulação de dataframes.
- Para isso, vamos carregar uma base de dados de valores de imóveis no Rio de Janeiro.
- Passemos ao RStudio para executar as operações.



## DAC ou SAC





## DAC ou SAC

- Funções `apply()` também trabalham assim.
- Ao aplicar uma função por linhas, estamos dividindo a matriz por uma das dimensões, aplicando funções a cada uma das partes e combinando os resultados em um vetor.
- Voltando ao RStudio.



## Outliers

- O que está estranho com a resposta abaixo:

```
> lapply(alug_venda, summary)
```

```
$aluguel
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0	1300	2300	9419	5500	7000000

```
$venda
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0	485000	980000	1634000	1800000	828600000



## Outliers

- A grande diferença entre média e mediana pode indicar a presença de *outliers*.
- Vamos então limpar o dataframe, retirando os valores muito discrepantes de preço por metro quadrado:
  - Venda, valores abaixo de 3000 ou acima de 20000.
  - Aluguel, valores abaixo de 25 ou acima de 60.
- Voltando ao RStudio.



## O pacote *dplyr*

- As funções da família **apply** e similares nos permite fazer praticamente todas as operações para preparação, exploração e análise de dados.
- Entretanto, essas funções podem deixar a desejar em termos de performance.
- Existe um pacote bastante rápido para manipulação de *dataframes*, de sintaxe simples, chamado **dplyr**.
- É provável que, na grande maioria dos casos, o pacote **dplyr** seja a solução mais rápida e eficiente.



## Resumo das funções principais

- **select**: seleciona uma ou mais colunas de um *dataframe*. Por exemplo, selecionar a coluna de preços.
- **filter**: filtra um *dataframe* com vetores lógicos. Por exemplo, filtrar valores de pm2 menores ou maiores que determinado nível.
- **arrange**: ordena o *dataframe* com base em uma coluna. Por exemplo, ordenar do maior para o menor pm2.



## Resumo das funções principais

- **mutate**: cria uma nova coluna. Por exemplo, criar a coluna pm2 como preco/m2.
- **group\_by**: agrupa um *dataframe* por índices. Por exemplo, agrupar os dados de imóveis por bairro e número de quartos.
- **summarise**: utilizado normalmente após o **group\_by** para calcular valores por grupo. Por exemplo, tirar a média ou mediana do preço por bairro.



## Operador %>%

- Além das funções anteriores (e outras), o pacote `dplyr` inclui também o uso do operador “*pipe*”, tão conhecido no ambiente Linux, indicado na linguagem R por `%>%`.
- Este operador faz com que você possa escrever `x %>% f()` ao invés de `f(x)`.
- Na prática, você vai poder escrever o código de manipulação dos dados da mesma forma que você pensa nas atividades.



## Exemplo 1

- Vamos pegar a base de dados “unicos”, filtrar apenas os dados de venda de “apartamento”. Vamos então agrupar os dados por “bairro”, calcular as medianas do “preço”, “m2” e “pm2” e o número de observações. Finalmente, vamos filtrar apenas os grupos com mais de 30 observações e ordenar de forma decrescente com base na mediana de “pm2”.



## Exemplo 2

- Agora, vamos usar a base de dados “unicos” para fazer uma busca de apartamentos. Estamos procurando apartamento nos bairros A e D, para aluguel, com preço menor que R\$2.200,00.



## Exemplo 3

- Finalmente, vamos pegar a base de dados “unicos”, filtrar apenas os dados coletados de “apartamento”, selecionar as colunas “bairro” e “preco”, criar uma coluna “pm2=preco/m2” e ordenar os dados de forma decrescente por “pm2”.



## A função *merge*

- A função `merge()` serve para combinar *dataframes*.
- Ela tenta identificar quais são as colunas identificadoras em comum entre dois *dataframes* para realizar a combinação.
- É similar ao `join` do SQL.

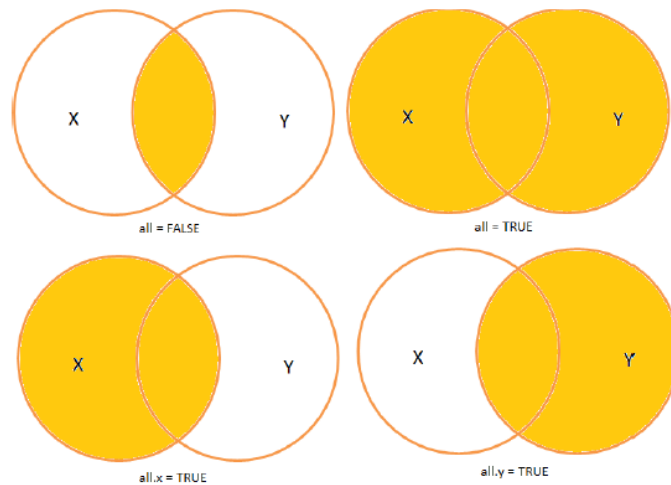


## A função *merge*

- Por meio dos parâmetros `all`, `all.x`, `all.y` define-se o tipo do `merge`.
  - O default é `FALSE` e é equivalente ao “natural join” do SQL;
  - “all” é equivalente ao “outer join”;
  - “all.x” é equivalente ao “left outer join”;
  - “all.y” é equivalente ao “right outer join”.



## A função *merge*



## Exemplo

- Vamos usar a instrução `aggregate()` para calcular a mediana do `pm2` separadamente para aluguel e para venda. Depois vamos usar o `merge()` para juntar os dados.



## Fazendo *merge* no *dplyr*

- O pacote **dplyr** também vem com funções de **merge** (neste caso, chamadas **join**).

x1	x2
A	1
B	2
C	3

+

x1	x3
A	T
B	F
D	T

=

### Mutating Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NA

**dplyr::left\_join(a, b, by = "x1")**  
Join matching rows from b to a.

x1	x3	x2
A	T	1
B	F	2
D	T	NA

**dplyr::right\_join(a, b, by = "x1")**  
Join matching rows from a to b.

x1	x2	x3
A	1	T
B	2	F

**dplyr::inner\_join(a, b, by = "x1")**  
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NA
D	NA	T

**dplyr::full\_join(a, b, by = "x1")**  
Join data. Retain all values, all rows.



## Fazendo *merge* no *dplyr*

- Exemplos no RStudio.





## Formatos *wide* x *long*

- A tabela gerada pela instrução `tidyapply()` é um exemplo de tabela no formato *wide*, onde as colunas representam grupos.
- Já o formato gerado pela `aggregate()` ou pelo `dplyr` é um exemplo de formato *long*, onde temos várias colunas identificadoras e apenas uma coluna de valores.
- É bastante comum usar dados no formato *long* em pacotes de visualização, como o `ggplot2` ou `lattice`.
- O formato *wide*, por sua vez, é bastante utilizado em apresentação de tabelas.
- Então, muitas vezes precisaremos transformar nossos dados de um formato para o outro.



## O pacote *reshape2*

- A função `melt()` transforma dados no formato *wide* para o formato *long*.
- As funções `acast()` e `dcast()` transformam dados no formato *long* para o formato *wide*. A primeira retorna um *array* e a segunda um *dataframe*.
- Exemplos no RStudio.



## O pacote *tidyr*

- O **tidyr** é um pacote novo que executa diversas tarefas de manipulação de dados, entre elas transformar do formato *wide* para o formato *long* e vice-versa.
- No entanto, este pacote só opera em *dataframes*.
- Exemplos no RStudio.

