ME315

MANIPULAÇÃO DE BANCO DE DADOS

GABRIELLA ARGENTON - 255677

Filter - FILTRAR LINHAS

FUNÇÃO: mantém apenas as linhas que satisfazem uma condição lógica

- (ex.: grupo == "A", renda > 3000).
- Em R (dplyr): df |> filter(condição 1, condição 2, ...)
- Usa os nomes das colunas diretamente dentro das condições.
- Em Julia (DataFrames.jl): filter (row -> condicao(row), df) ou df[vetor_logico, :]
- A condição é uma função que recebe cada linha (row) ou um vetor lógico criado com df.coluna .== valor.

FILTER

```
# Pessoas do grupo A com renda_2024 > 3500
df_filtrado = filter(row -> row.grupo == "A" && row.renda_2024 > 3500, df)
```

	id	nome	grupo	idade	sexo	renda_2023	renda_2024
	Int64	String	String	Int64	String	Int64	Int64
1	2	Bruno	A	35	M	4800	5000
2	8	Hugo	A	30	M	3500	3800

Select - SELECIONAR COLUNAS

FUNÇÃO: Escolhe quais colunas manter (e pode reordená-las), deixando a tabela mais enxuta.

- Em R (dplyr): df |> select(col1, col2, ...)
- Também aceita helpers como starts_with(), ends_with(), etc.
- Em Julia (DataFrames.jl): select(df, :col1, :col2, ...)
- Usa símbolos (:coluna) ou vetores de símbolos para indicar as colunas.

SELECT

```
# Manter apenas id, nome, grupo e renda_2024
df_select = select(df, :id, :nome, :grupo, :renda_2024)
```

	id	nome	grupo	renda_2024
	Int64	String	String	Int64
1	1	Ana	A	2700
2	2	Bruno	A	5000
3	3	Carla	A	3400
4	4	Diego	B	6200
5	5	Eva	B	2000
6	6	Felipe	B	4400
7	7	Gabi	B	3300
8	8	Hugo	A	3800

Rename - RENOMEAR COLUNAS

FUNÇÃO: muda o nome das colunas para algo mais claro ou padronizado.

- Em R (dplyr): df |> rename(novo_nome = nome_antigo)

  O nome novo fica à esquerda, o antigo à direita.
- Em Julia (DataFrames.jl): rename(df, :antigo => :novo) ou rename!(df, :antigo => :novo)

  Sem ! devolve um novo DataFrame; com ! altera o original.

RENAME

```
# Renomear renda_2024 para renda_atual (sem alterar o original)
df_renomeado = rename(df, :renda_2024 => :renda_atual)

# Ou alterando o próprio df
# rename!(df, :renda_2024 => :renda_atual)
```

	id	nome	grupo	idade	sexo	renda_2023	renda_atual
	Int64	String	String	Int64	String	Int64	Int64
1	1	Ana	A	23	F	2500	2700
2	2	Bruno	A	35	M	4800	5000
3	3	Carla	A	29	F	3200	3400
4	4	Diego	B	41	M	6000	6200
5	5	Eva	B	19	F	1800	2000
6	6	Felipe	B	33	M	4200	4400
7	7	Gabi	B	27	F	3100	3300
8	8	Hugo	A	30	M	3500	3800

Mutate - CRIAR/ALTERAR COLUNAS

FUNÇÃO: cria novas variáveis ou modifica variáveis existentes a partir de expressões envolvendo as colunas.

- Em R (dplyr): df |> mutate(nova_col = expressao(colunas), ...)
- Se usar o mesmo nome, sobrescreve a coluna original.
- Em Julia (DataFrames.jl): transform(df, :col => funcao => :nova_col) ou transform!
- Para operar linha a linha usa ByRow: :col => ByRow(x -> ...) => :nova_col.

MUTATE

```
# Criar renda_anual_2024 e faixa_idade
df_mut = transform(
    df,
    :renda_2024 => (x -> x .* 12) => :renda_anual_2024,
    :idade      => ByRow(x -> x < 30 ? "jovem" : "adulto") => :faixa_idade
)
```

	id	nome	grupo	idade	sexo	renda_2023	renda_2024	renda_anual_2024
	Int64	String	String	Int64	String	Int64	Int64	Int64
1	1	Ana	A	23	F	2500	2700	32400
2	2	Bruno	A	35	M	4800	5000	60000
3	3	Carla	A	29	F	3200	3400	40800
4	4	Diego	B	41	M	6000	6200	74400
5	5	Eva	B	19	F	1800	2000	24000
6	6	Felipe	B	33	M	4200	4400	52800
7	7	Gabi	B	27	F	3100	3300	39600
8	8	Hugo	A	30	M	3500	3800	45600

Arrange - ORDENAR LINHAS

FUNÇÃO: reordena as linhas com base em uma ou mais colunas (crescente ou decrescente).

- Em R (dplyr): df |> arrange(col1, col2) ou df |> arrange(desc(col))
- Em Julia (DataFrames.jl): sort(df, :col; rev = true) ou sort(df, [:col1, :col2])
- O argumento rev = true indica ordem decrescente.

ARRANGE

```julia
# Ordenar pela renda_2024 em ordem decrescente
df_ord = sort(df, :renda_2024; rev = true)

# Ordenar por grupo e idade crescente
df_ord2 = sort(df, [:grupo, :idade])
```

	id	nome	grupo	idade	sexo	renda_2023	renda_2024
	Int64	String	String	Int64	String	Int64	Int64
1	1	Ana	A	23	F	2500	2700
2	3	Carla	A	29	F	3200	3400
3	8	Hugo	A	30	M	3500	3800
4	2	Bruno	A	35	M	4800	5000
5	5	Eva	B	19	F	1800	2000
6	7	Gabi	B	27	F	3100	3300
7	6	Felipe	B	33	M	4200	4400
8	4	Diego	B	41	M	6000	6200

group_by - AGRUPAR OBSERVAÇÕES

- FUNÇÃO: define grupos dentro do data frame (por exemplo, por grupo, sexo, ano) para fazer operações por grupo.
- Em R (dplyr): df |> group_by(col1, col2, ...)
- Normalmente usado antes do summarise().
- Em Julia (DataFrames.jl): g = groupby(df, :col) ou groupby(df, [:col1, :col2])
- Cria um GroupedDataFrame, usado depois em combine().

Summarise - RESUMIR DADOS

- FUNÇÃO: calcula estatísticas resumidas (médias, contagens, etc.) por grupo ou no total.
- Em R (dplyr): df |> group_by(grupo) |> summarise(media = mean(var), n = n(), .groups = "drop")
- Em Julia (DataFrames.jl): combine(groupby(df, :grupo), :var => mean => :var_media, nrow => :n_obs)

Também pode usar combine(df, ...) para resumo global (sem grupos).

GROUP_BY + SUMMARISE

```
g = groupby(df, :grupo)

resumo_grupo = combine(
    g,
    :renda_2024 => mean => :renda_media,
    :idade      => mean => :idade_media,
    nrow        => :n_pessoas
)
```

	grupo	renda_media	idade_media	n_pessoas
	String	Float64	Float64	Int64
1	A	3725.0	29.25	4
2	B	3975.0	30.0	4

Pivot_wider - LONG → WIDE

- FUNÇÃO: transforma linhas em colunas, saindo de formato “long” para “wide” (uma coluna para cada categoria).
- Em R (tidyr): df_wide <- df_long |> pivot_wider(names_from = coluna_que_vira_coluna, values_from = coluna_de_valores)
- Em Julia (DataFrames.jl): df_wide = unstack(df_long, :id, :chave, :valor)
- Onde :chave vira nomes de colunas e :valor são os valores preenchidos.

PIVOT_WIDER

```
# :id → permanece como identificador das linhas
# :ano → vira o nome das novas colunas (ex.: renda_2023, renda_2024)
# :renda → são os valores preenchidos nessas colunas
# Cada ano que estava em linhas vira uma coluna nova, reorganizando a tabela.

df_wide = unstack(df_long, :id, :ano, :renda)
```

	id	renda_2023	renda_2024
	Int64	Int64?	Int64?
1	1	2500	2700
2	2	4800	5000
3	3	3200	3400
4	4	6000	6200
5	5	1800	2000
6	6	4200	4400
7	7	3100	3300
8	8	3500	3800

Pivot_longer - WIDE→ LONG

- FUNÇÃO: faz o inverso do pivot_wider: empilha colunas em duas colunas (“nome” da coluna e “valor”).
- Em R (tidyr):

df_long <- df_wide |> pivot_longer(cols = cols_escolhidas, names_to = "nome", values_to = "valor")

- Em Julia (DataFrames.jl):

df_long = stack(df_wide, cols, variable_name = :nome, value_name = :valor)

As colunas em cols são empilhadas em duas: :nome e :valor.

PIVOT_LONGER

```
# Deixar rendas em formato long
df_long = stack(
    df,
    [:renda_2023, :renda_2024],
    variable_name = :ano,
    value_name    = :renda
)
```

	id	nome	grupo	idade	sexo	ano	renda
	Int64	String	String	Int64	String	String	Int64
1	1	Ana	A	23	F	renda_2023	2500
2	2	Bruno	A	35	M	renda_2023	4800
3	3	Carla	A	29	F	renda_2023	3200
4	4	Diego	B	41	M	renda_2023	6000
5	5	Eva	B	19	F	renda_2023	1800
6	6	Felipe	B	33	M	renda_2023	4200
7	7	Gabi	B	27	F	renda_2023	3100
8	8	Hugo	A	30	M	renda_2023	3500
9	1	Ana	A	23	F	renda_2024	2700
10	2	Bruno	A	35	M	renda_2024	5000
11	3	Carla	A	29	F	renda_2024	3400
12	4	Diego	B	41	M	renda_2024	6200
13	5	Eva	B	19	F	renda_2024	2000
14	6	Felipe	B	33	M	renda_2024	4400
15	7	Gabi	B	27	F	renda_2024	3300
16	8	Hugo	A	30	M	renda_2024	3800

- FUNÇÃO: junta duas tabelas mantendo todas as linhas da tabela da esquerda e trazendo colunas da direita quando as chaves batem.
- Em R (dplyr):

df_left <- left_join(df1, df2, by = "id")

Ou by = c("id_esq" = "id_dir") se os nomes forem diferentes.

- Em Julia (DataFrames.jl):

df_left = leftjoin(df1, df2, on = :id)

Ou on = [:id_esq => :id_dir] quando os nomes diferem.

Left_join - COMBINAR TABELAS

LEFT_JOIN

```
# Juntar df com df_bonus pela coluna id
df_left = leftjoin(df, df_bonus, on = :id)
```

	id	nome	grupo	idade	sexo	renda_2023	renda_2024	bonus
	Int64	String	String	Int64	String	Int64	Int64	Int64?
1	2	Bruno	A	35	M	4800	5000	500
2	4	Diego	B	41	M	6000	6200	1000
3	8	Hugo	A	30	M	3500	3800	700
4	1	Ana	A	23	F	2500	2700	*missing*
5	3	Carla	A	29	F	3200	3400	*missing*
6	5	Eva	B	19	F	1800	2000	*missing*
7	6	Felipe	B	33	M	4200	4400	*missing*
8	7	Gabi	B	27	F	3100	3300	*missing*

COMPARAÇÃO R X PYTHON

Para comparar o desempenho de R e Python na manipulação de dados, utilizamos a mesma tarefa em ambas as linguagens:
gerar um conjunto de dados com 1.000.000 de linhas
com as colunas: grupo (A, B, C, D), x (1 a 100) e y (1 a 1000)
calcular a média de y por grupo (grupo)

Analogia:

Imagine uma escola com 1 milhão de alunos:
Cada aluno pertence a uma das 4 salas: A, B, C ou D → (coluna grupo)
Cada aluno tem um número de chamada entre 1 e 100 → (coluna x)
E cada aluno tem uma nota entre 1 e 1000 → (coluna y)

Nosso objetivo:

Calcular a média das notas (y) de cada sala (A, B, C e D).

Facilidade de uso (R x Python)

R (com dplyr)

Sintaxe extremamente clara para manipulação de dados.

group_by() + summarise() são intuitivos e fáceis de ler.

O pipeline (|>) deixa o fluxo do código natural e direto.

Muito familiar para quem vem de estatística e análise de dados.

Python (com pandas)

Sintaxe um pouco mais verbosa, mas muito poderosa.

A estrutura df.groupby(...)[col].mean() é direta, mas exige familiaridade com o estilo do pandas.

Excelente integração com bibliotecas de machine learning e aplicações de produção.

Comparação geral:

R tende a ser mais legível e natural para manipulação de dados tabulares.

Python é mais versátil no ecossistema geral de ciência de dados.

Para esta tarefa específica, as duas soluções ficaram compactas e foram simples de entender.

RSULTADOS R X PYTHON

R: (11.9362 milissegundos)

```
: milliseconds
                    expr    min      lq     mean  median      uq     max ne
umo_por_grupo(df_big) 9.5845 10.9482 11.83474 11.9362 12.6181 13.7454
```

PYTHON: (24,6 milissegundos)

```
print("Tempo médio por execução:", tempo/10, "segundos")
```

```
Tempo médio por execução: 0.02464951999718323 segundos
```

Conclusão - Python x R

- O R apresentou o menor tempo mediano (~12 ms), sendo aproximadamente duas vezes mais rápido que o Python nesta tarefa.
- O Python apresentou desempenho estável (~25 ms por execução).
- Para a operação avaliada, não há diferença prática significativa: ambas são eficientes e adequadas para manipulação de grandes volumes de dados.

Conclusão: para cálculos agrupados simples, R e Python apresentam desempenho muito semelhante, com uma leve vantagem para R na mediana — mas ambas são rápidas o suficiente para uso cotidiano em datasets grandes.

Obrigada!