Agrupamento, Agregação e Consulta Lógica de Dados

Transformando dados brutos em informações valiosas com Pandas

Aula 05 - UC2

Objetivos da Aula

- Compreender o conceito de **agrupamento** e **agregação** de dados e sua importância na análise de grandes volumes de informação.
- Aprender a utilizar as **funções de agregação** do Pandas como **mean()**, **sum()**, **count()**, **max()** e **min()**.
- Desenvolver habilidades para realizar **consultas com operadores lógicos** para filtrar dados de forma precisa e eficiente.
- Aplicar os conceitos em **datasets reais** para extrair insights relevantes e transformar dados brutos em inteligência de negócios.

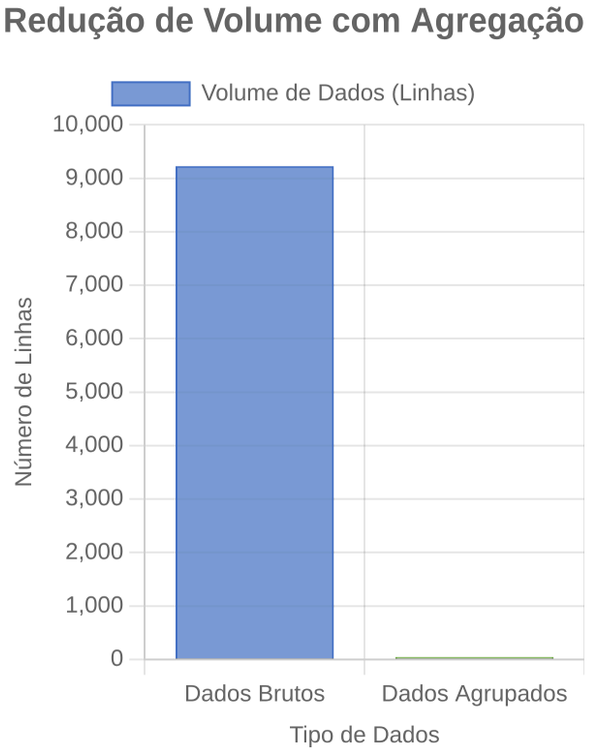
Introdução e Contextualização

> ***"Como transformamos milhares de linhas de dados em uma única conclusão importante?"***

A resposta está na **agregação de dados**, que é a essência da análise de dados. Através da agregação, podemos condensar grandes volumes de informação em métricas significativas que orientam decisões de negócio.

Sem técnicas de agregação, ficaríamos sobrecarregados com milhares ou milhões de linhas de dados sem conseguir extrair conclusões úteis. A agregação nos permite **ver a floresta, não apenas as árvores**.

Nesta aula, aprenderemos como utilizar o Pandas para agrupar, agregar e filtrar dados de forma eficiente, transformando dados brutos em insights valiosos.



Revisão: DataFrame

O **DataFrame** é a estrutura principal de dados no Pandas, semelhante a uma tabela com linhas e colunas. É a base para todas as operações de agrupamento e agregação que aprenderemos hoje.

O Conceito de Agrupamento (groupby)

O **agrupamento** é uma operação fundamental na análise de dados que permite organizar os dados em grupos com base em valores comuns em uma ou mais colunas.

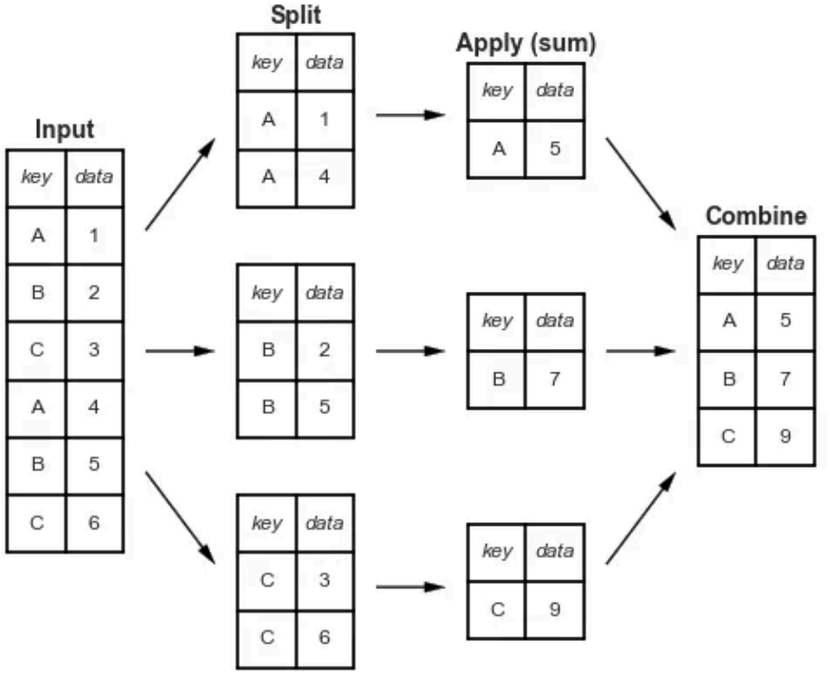
Analogia: Organizando Papéis em Pastas

Imagine que você tem uma pilha de documentos (linhas de dados) e precisa organizá-los em pastas (grupos). Cada pasta representa uma categoria específica, como "Vendas por Região" ou "Clientes por Faixa Etária". O agrupamento é como separar esses documentos nas pastas corretas para depois analisá-los em conjunto.

No Pandas, o agrupamento é realizado através do método **.groupby()**, que divide os dados em grupos com base em uma ou mais colunas e permite aplicar funções de agregação a cada grupo.

```
# Sintaxe básica do groupby
df.groupby('coluna_de_agrupamento')
```

O resultado do **.groupby()** não é imediatamente útil até que seja aplicada uma função de agregação, como **.sum()**, **.mean()** ou **.count()**.



Processo de agrupamento: dados são organizados por uma coluna-chave e depois agregados

Funções de Agregação

Após agrupar os dados com **groupby()**, precisamos aplicar funções de agregação para resumir os dados de cada grupo:

Média (.mean())

Calcula a média de valores dentro de cada grupo.

```
df.groupby('Regiao')['Vendas'].mean()
```

Soma (.sum())

Soma todos os valores dentro de cada grupo.

```
df.groupby('Produto')['Vendas'].sum()
```

Contagem (.count() ou .size())

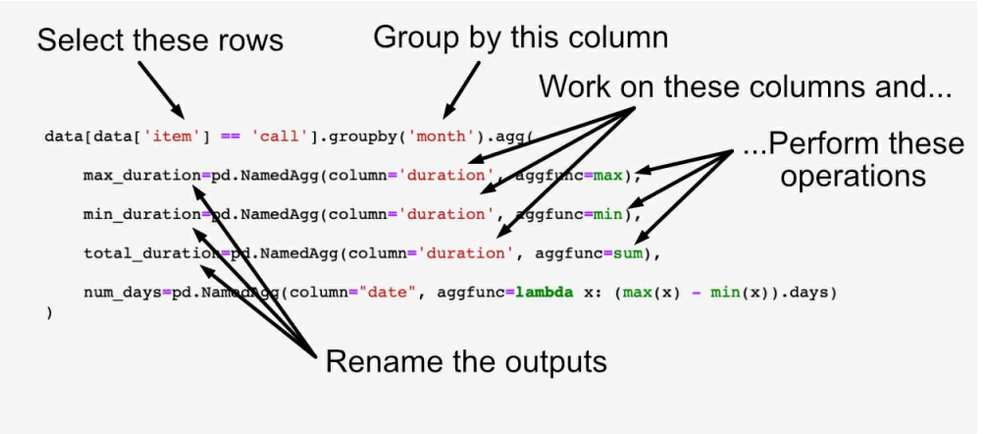
Conta o número de ocorrências em cada grupo.

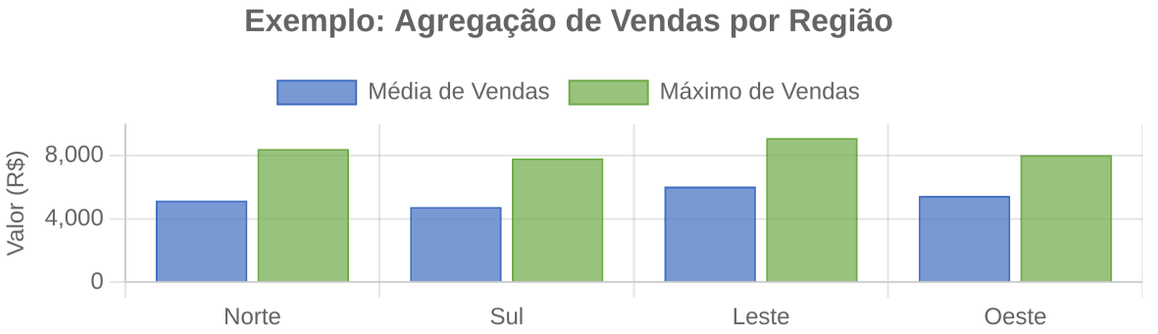
```
df.groupby('Cidade')['Cliente'].count()
```

Máximo/Mínimo (.max() / .min())

Encontra o maior ou menor valor em cada grupo.

```
df.groupby('Vendedor')['Vendas'].max()
```





Demonstração Guiada - Agrupamento Básico

</> Passo 1: Criar um DataFrame de Exemplo

```
# Importar pandas
import pandas as pd
import numpy as np

# Criar dados de exemplo
dados = {
  'Produto': ['Notebook', 'Mouse', 'Monitor',
'Notebook', 'Teclado', 'Monitor'],
  'Regiao': ['Sul', 'Norte', 'Sul', 'Nordeste',
'Norte', 'Nordeste'],
  'Vendas': [5200, 120, 1800, 4800, 250, 1950]
}

# Criar DataFrame
df = pd.DataFrame(dados)
```

Passo 2: Agrupar e Somar Vendas por Região

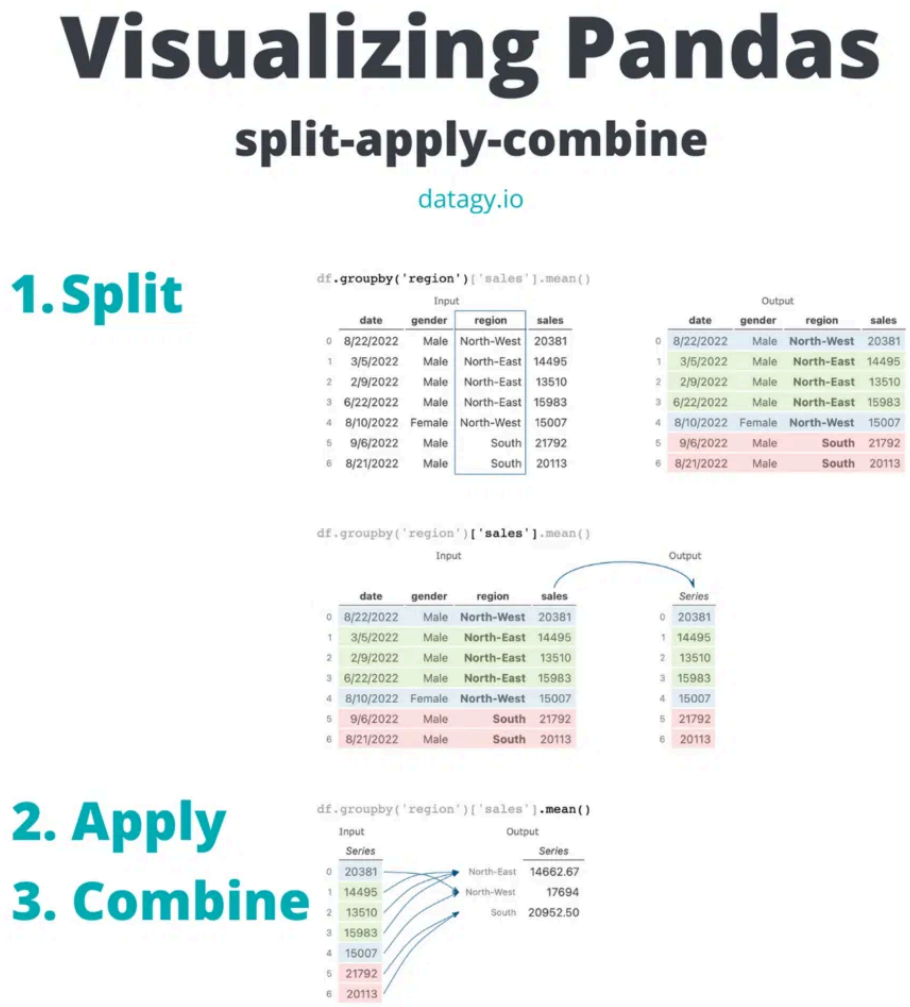
```
# Agrupar por região e somar vendas
vendas_por_regiao = df.groupby('Regiao')
['Vendas'].sum()

# Visualizar resultado
print(vendas_por_regiao)
```

```
Regiao
Nordeste 6750
Norte 370
Sul 7000
Name: Vendas, dtype: int64
```

Passo 3: Calcular a Média de Vendas por Produto

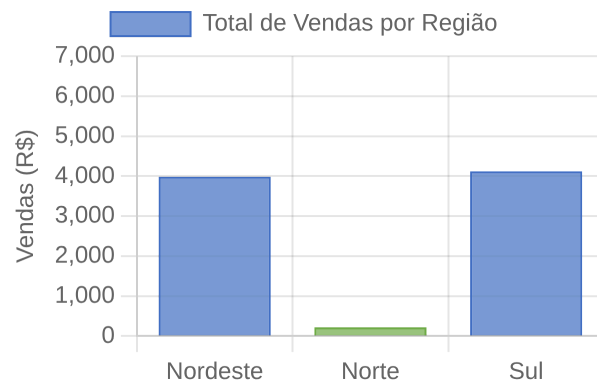
```
# Agrupar por produto e calcular média
media_por_produto = df.groupby('Produto')
['Vendas'].mean()

# Visualizar resultado
print(media_por_produto)
```

```
Produto
Monitor 1875.0
Mouse 120.0
Notebook 5000.0
Teclado 250.0
Name: Vendas, dtype: float64
```



Observações Importantes:

- O método **groupby()** sempre é seguido por uma função de agregação como **sum()**, **mean()**, etc.
- O resultado de um agrupamento é um novo DataFrame ou Series com os índices baseados na coluna de agrupamento.
- É possível agrupar por múltiplas colunas: **df.groupby(['Regiao', 'Produto'])**
- Para aplicar múltiplas funções de agregação, use **agg()**: **df.groupby('Regiao')['Vendas'].agg(['sum', 'mean'])**



Atividade Prática 1: Agrupamento e Agregação em Dataset Real

Objetivo: Aplicar técnicas de agrupamento e agregação em um dataset real.
Formato: Atividade em grupos usando Google Colab.
Dataset: Dados de vendas, notas de alunos ou dados demográficos.

1 Carregamento do Dataset

Importe o dataset no notebook do Google Colab:

```
import pandas as pd
df = pd.read_csv('nome_do_arquivo.csv')
```

2 Agrupamento por Categoria

Escolha um campo categórico e calcule a média:

```
media_por_cidade = df.groupby('Cidade')['Nota'].mean()
```

3 Agregação de Contagem

Agrupe por outro campo e conte ocorrências:

```
contagem_por_genero = df.groupby('Genero').size()
```

4 Agrupamento Múltiplo

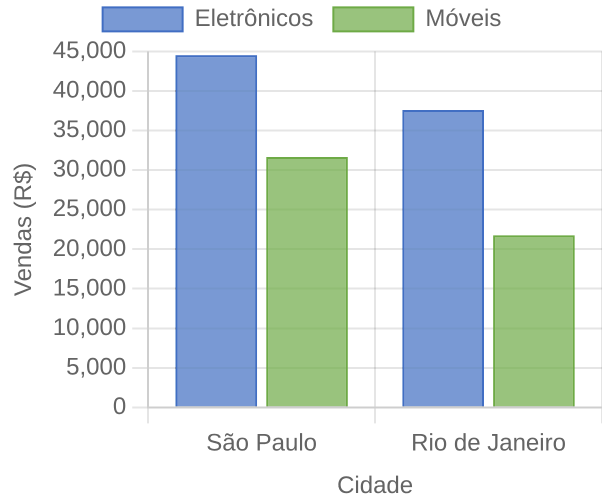
Agrupe por dois campos simultaneamente:

```
vendas_cidade_produto = df.groupby(['Cidade', 'Produto'])
['Vendas'].sum()
```

5 Apresentação dos Resultados

Cada grupo apresenta o resultado mais interessante, justificando sua relevância.

Exemplo: Vendas por Cidade e Categoria




```
# Exemplo de resultado de agrupamento múltiplo
Cidade      Produto
São Paulo   Eletrônicos     45000
            Móveis          32000
Rio         Eletrônicos     38000
            Móveis          22000
```

Tempo estimado: **30 minutos**

Operadores Lógicos em DataFrames

Os **operadores lógicos** são ferramentas essenciais que permitem filtrar dados com base em condições específicas, ajudando a selecionar exatamente os registros necessários.

Operadores Simples de Comparação

Usados para comparar valores: **>** (maior), **<** (menor), **==** (igual), **!=** (diferente), **>=** (maior ou igual), **<=** (menor ou igual)

```
df[df['idade'] > 30] # Seleciona pessoas com mais de 30 anos
```

Operadores Combinados (Lógicos)

E (&)

As duas condições devem ser verdadeiras simultaneamente.

```
df[(df['idade'] > 30) & (df['cidade'] == 'São Paulo')]
```

OU (|)

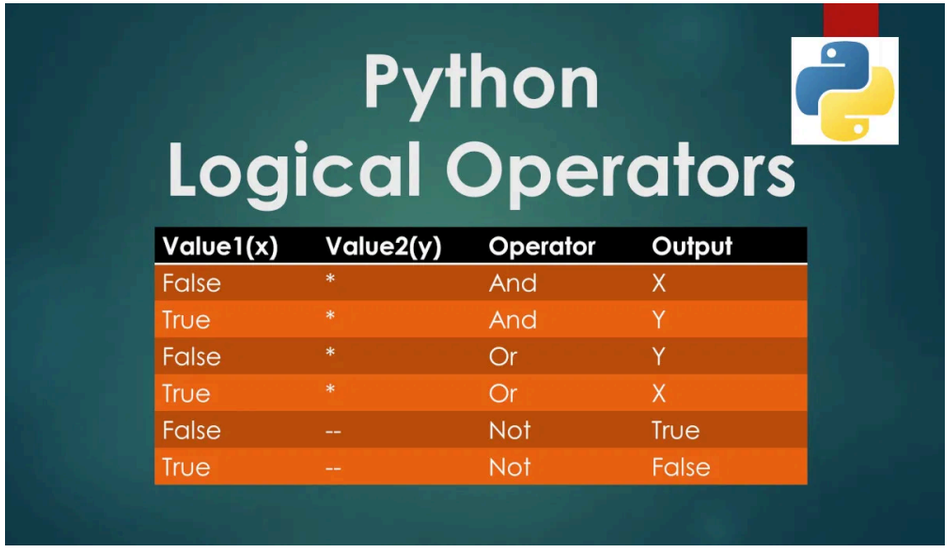
Pelo menos uma das condições deve ser verdadeira.

```
df[(df['cidade'] == 'São Paulo') | (df['cidade'] == 'Rio')]
```

NÃO (~)

Inverte a condição, selecionando o oposto.

```
df[~(df['país'] == 'Brasil')] # Todos que não são do Brasil
```



⚠ Atenção aos Parênteses!

Sempre use parênteses ao combinar condições com operadores lógicos:

```
Incorreto: df[df['idade'] > 30 & df['salario'] > 5000]
Correto: df[(df['idade'] > 30) & (df['salario'] > 5000)]
```

Atividade Prática 2: Consulta com Operadores Lógicos

Objetivo: Aplicar operadores lógicos para filtrar dados de forma precisa.
Formato: Continuação da atividade em grupos usando o mesmo dataset.

1 Filtro Simples

Filtre o DataFrame usando apenas uma condição:

```
# Exemplo: Clientes com idade acima de 25 anos
clientes_25_mais = df[df['Idade'] > 25]
```

2 Filtro Combinado (E)

Combine duas condições usando o operador &:

```
# Vendas de um produto específico acima de um valor
vendas_filtradas = df[(df['Produto'] == 'Notebook') & (df['Valor'] > 3000)]
```

3 Filtro Combinado (OU)

Combine duas condições usando o operador |:

```
# Clientes de duas cidades
clientes_sp_rj = df[(df['Cidade'] == 'São Paulo') | (df['Cidade'] == 'Rio')]
```

4 Agrupamento + Filtro

Combine filtros com agrupamento:

```
# Média de vendas por região, apenas para produtos premium
media_por_regiao = df[df['Categoria'] == 'Premium'].groupby('Regiao')['Vendas'].mean()
```

5 Justificativa de Negócio

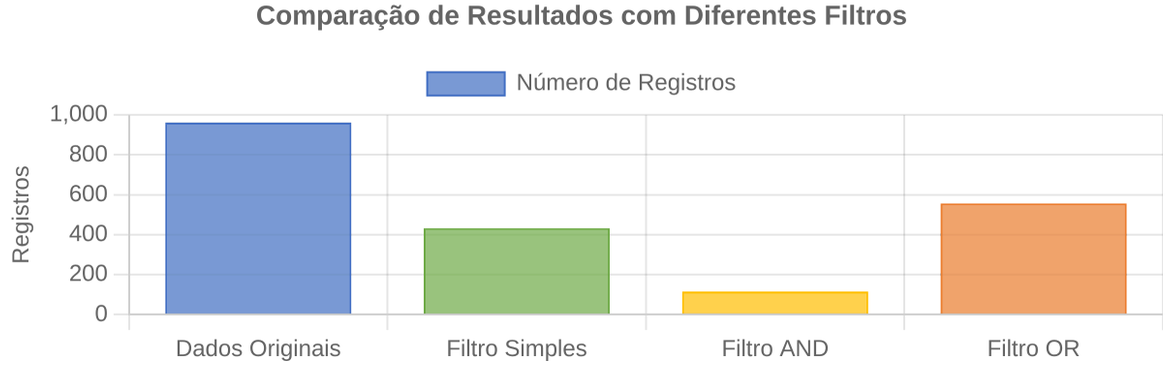
Cada grupo deve explicar como os resultados obtidos podem ajudar na tomada de decisões.

```
In [26]: import pandas as pd

In [27]: d = {'col1': ['abc', 'abcd','abcde','def','bbi']}

In [28]: df = pd.DataFrame(data=d)

In [29]: df

Out[29]:
```

	col1
0	abc
1	abcd
2	abcde
3	def
4	bbi

```
In [30]: df[df['col1'].str.contains('abc',regex=False)|df['col1'].str.contains('bbi',regex=False)]

Out[30]:
```

	col1
0	abc
1	abcd
2	abcde
4	bbi



Tempo estimado: **25 minutos**

Combinando Agrupamento e Filtros

A combinação de **filtros lógicos** com **agrupamento** permite análises mais sofisticadas e direcionadas:

Abordagem 1: Filtrar e depois agrupar

```
# Filtrar produtos 'Premium'
produtos_premium = df[df['Categoria'] == 'Premium']
# Agrupar por região
vendas_premium_por_regiao = produtos_premium.groupby('Regiao')
['Vendas'].sum()
```

Abordagem 2: Agrupar e depois filtrar

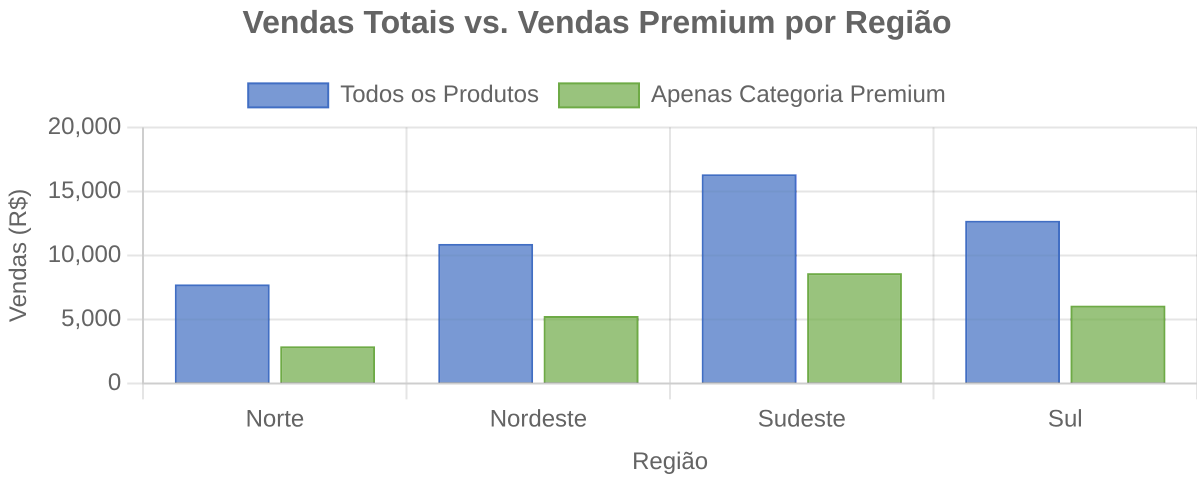
```
# Agrupar todos os dados por região
vendas_por_regiao = df.groupby('Regiao')['Vendas'].sum()
# Filtrar regiões com vendas > 10.000
regioes_alto_desempenho = vendas_por_regiao[vendas_por_regiao >
10000]
```

Dica de Eficiência

Sempre filtre os dados **antes** de agrupá-los quando possível. Isso reduz a quantidade de dados a serem processados.

```
df[['name', 'fiber', 'calories']][(df['calories']==90) & (df['fiber']<4)]
```

	name	fiber	calories
50	Nutri-grain Wheat	3.0	90
60	Raisin Squares	2.0	90
65	Shredded Wheat spoon size	3.0	90
68	Strawberry Fruit Wheats	3.0	90



Boas Práticas e Dicas

Encerramento e Desafio Final

O que Aprendemos Hoje

- O conceito de **agrupamento** e como usar o método **.groupby()** para organizar dados
- Funções de **agregação** como **.sum()**, **.mean()**, **.count()**, **.max()** e **.min()**
- Uso de **operadores lógicos** (**&**, **|**, **~**) para filtrar dados com precisão
- Como **combinar filtros e agrupamentos** para análises mais sofisticadas
- **Boas práticas** para organização de scripts e atenção aos detalhes

Na próxima aula, exploraremos técnicas avançadas de visualização de dados e como apresentar insights de forma eficaz.

Desafio Final

Crie uma pergunta de análise que só possa ser respondida usando agrupamento e pelo menos um operador lógico. Pense em uma questão relevante para um contexto de negócio real.

Exemplo de Desafio:

"Qual o total de vendas de produtos Premium (categoria 'A') OU de produtos Essenciais (categoria 'B') por estado do Nordeste (Região 'Nordeste')?"

Solução:

```
# Filtrar produtos das categorias A ou B na região Nordeste
produtos_filtrados = df[((df['Categoria'] == 'A') | (df['Categoria'] == 'B')) &
                        (df['Regiao'] == 'Nordeste')]

# Agrupar por estado e somar vendas
resultado = produtos_filtrados.groupby('Estado')['Vendas'].sum()
```