□ Aula 02 — Expandindo o Universo: Dominando os DataFrames

□ Olá de novo, explorador de dados!

Na nossa última aula, você conheceu a **Series**, a coluna solitária e poderosa do mundo do Pandas. Se a Series é o átomo fundamental, o **DataFrame** é a molécula complexa. Ele é a evolução natural do seu aprendizado, a estrutura que você usará na maior parte do tempo para manipular dados de forma tabular, como em uma planilha.

Vamos mergulhar de cabeça e construir nosso conhecimento sobre a principal ferramenta de qualquer cientista de dados ou analista de dados.

- **Tema Central**: A estrutura de dados **`DataFrame`**
- **Comando(s) Essencial(is)**: `pd.DataFrame()`, `.columns`, `.index`, `.shape`, `.dtypes`, `.set_index()`, `.rename()`, `.astype()`, `.values`, `.to_dict()`

□ O DataFrame: A Planilha do Python

Se você já trabalhou com planilhas no Excel, no Google Sheets ou com tabelas em bancos de dados SQL, você já conhece a lógica do **DataFrame**. Ele é, em essência, uma coleção de Series, todas compartilhando o mesmo índice (os rótulos das linhas). É por isso que a nossa jornada começou com a Series: para entender o DataFrame, você precisa entender o seu componente básico.

Um DataFrame é uma estrutura bidimensional (2-D) que consiste em:

- **Colunas**: Cada coluna é uma **Series**, com seu próprio nome e tipo de dado.
- **Índice**: Os rótulos das linhas que organizam os dados de forma vertical.

Por que DataFrames são tão importantes? Porque os dados do mundo real quase nunca são unidimensionais. Eles vêm em tabelas, com múltiplos atributos para cada observação. Pense em uma lista de clientes: cada cliente tem um nome, uma idade, um endereço, uma data de cadastro. Cada um desses atributos seria uma coluna (uma Series) dentro de um DataFrame. Dominar o DataFrame significa ter o poder de organizar, limpar, analisar e visualizar qualquer conjunto de dados tabulares.

□ Conteúdo Explicado: Criando e Inspecionando sua Tabela

1. Criando o `DataFrame`: As Múltiplas Faces de `pd.DataFrame()`

A função `pd.DataFrame()` é a porta de entrada para a criação dessa estrutura. Assim como a Series, ela é flexível e aceita diferentes tipos de dados de entrada, mas os mais comuns são dicionários ou listas.

A sintaxe básica é `pd.DataFrame(data, index, columns, ...)`:

- `data`: A fonte de dados que você quer transformar em uma tabela.
- `index`: Rótulos para as linhas.
- `columns`: Rótulos para as colunas.

---

a) A Maneira Mais Comum: Usando um Dicionário de Listas

Essa é a forma mais intuitiva e popular de criar um DataFrame a partir de dados brutos que já estão organizados por "colunas". O dicionário é perfeito para isso, pois suas chaves se tornam os nomes das colunas, e as listas se tornam os valores de cada coluna.

```
import pandas as pd

# Os dados já estão organizados por colunas: 'nome', 'idade', 'cidade'
dados = {
    'nome': ['Ana', 'Bruno', 'Carlos'],
    'idade': [23, 35, 31],
    'cidade': ['SP', 'RJ', 'MG']
}
df = pd.DataFrame(dados)
print(df)
```

Saída Esperada:

```
     nome  idade cidade
0     Ana     23     SP
1   Bruno     35     RJ
2  Carlos     31     MG
```

O que aconteceu? O Pandas leu o dicionário e automaticamente usou as chaves (`'nome'`, `'idade'`, `'cidade'`) como nomes das colunas. Ele também criou um índice numérico padrão (`0, 1, 2...`) para as linhas. Simples e direto.

---

b) Outra Abordagem: Usando uma Lista de Dicionários

Seus dados também podem estar organizados por "linhas". Ou seja, cada dicionário representa um registro (uma linha), e as chaves internas são os nomes das colunas.

```
dados = [
    {'nome': 'Ana', 'idade': 23, 'cidade': 'SP'},
    {'nome': 'Bruno', 'idade': 35, 'cidade': 'RJ'}
]
df = pd.DataFrame(dados)
```

Essa forma é muito útil quando você está recebendo dados de uma API, por exemplo, que geralmente os entrega nesse formato.

---

c) De Series a DataFrame

Lembram-se do método `.to_frame()` da aula passada? Ele serve exatamente para isso: elevar uma Series para a "dimensão" do DataFrame.

```
s1 = pd.Series([10, 20, 30], name='valores')
df = s1.to_frame()
print(df)
```

Saída:

```
   valores
0       10
1       20
2       30
```

Note que o nome da Series (`valores`) se tornou o nome da única coluna do DataFrame. Esse é um excelente exemplo de como os conceitos se conectam.

---

d) Criando a partir do NumPy

Se você já possui dados em um array do NumPy, pode facilmente convertê-lo em um DataFrame, especificando os nomes das colunas.

```
import numpy as np
arr = np.array([[1, 2], [3, 4]])
df = pd.DataFrame(arr, columns=['A', 'B'])
```

Essa abordagem é comum em tarefas de processamento numérico e machine learning, onde a manipulação com arrays é frequente.

---

2. Inspecionando o DataFrame: Os Métodos Essenciais

Após criar seu DataFrame, a primeira coisa a fazer é conhecê-lo. O Pandas oferece um conjunto de atributos e métodos que agem como um "raio-x" para sua estrutura de dados.

- `df.shape`: Retorna uma tupla `(linhas, colunas)`. É o "tamanho" da sua tabela.
- `df.columns`: Exibe os nomes das colunas.
- `df.index`: Exibe os rótulos das linhas.
- `df.dtypes`: Mostra o tipo de dado de cada coluna. **Essa é uma das inspeções mais importantes**, pois o tipo de dado determina quais operações você pode realizar. Lembre-se, o `dtype` é o tipo da **Series** que compõe cada coluna.

□ Demonstração Aplicada: A Análise em Quatro Fases

Vamos aplicar tudo o que aprendemos em um exemplo prático. Nosso objetivo é pegar um DataFrame de vendas, inspecioná-lo, ajustar sua estrutura e preparar os dados para análise.

```
# 1. Fase de Criação: Montando nosso DataFrame de vendas
vendas = pd.DataFrame({
    'dia': ['seg', 'ter', 'qua', 'qui', 'sex'],
    'valor': [250, 300, None, 410, 380],
    'loja': ['A', 'A', 'B', 'B', 'A']
})

print("--- DataFrame Criado ---")
print(vendas)
print("\n")

# 2. Fase de Inspeção: Conhecendo a estrutura
print("--- Inspeção ---")
print(f"Dimensões (linhas, colunas): {vendas.shape}")
print(f"Colunas: {vendas.columns.tolist()}")
print("\nTipos de dados:")
print(vendas.dtypes)
print("\n")

# 3. Fase de Ajuste Estrutural: Organizando a tabela
# Usamos 'dia' como nosso índice para uma melhor organização
vendas_organizadas = vendas.set_index('dia')
print("--- DataFrame com 'dia' como índice ---")
print(vendas_organizadas)

# Agora, vamos renomear a coluna 'valor' para algo mais descritivo
vendas_organizadas = vendas_organizadas.rename(columns={'valor': 'vendas_diarias'})
print("\n--- DataFrame com coluna renomeada ---")
print(vendas_organizadas)
print("\n")

# 4. Fase de Tratamento de Dados: Lidando com valores ausentes
# Lembre-se da aula anterior! Vamos usar fillna() com a média para tratar o valor ausente
media_vendas = vendas_organizadas['vendas_diarias'].mean()
vendas_organizadas['vendas_diarias'] = vendas_organizadas['vendas_diarias'].fillna(media_vendas)
print("--- DataFrame com valores ausentes tratados ---")
print(vendas_organizadas)

# Para garantir, vamos verificar o tipo da coluna 'vendas_diarias'
print("\nTipo de dado da coluna 'vendas_diarias':", vendas_organizadas['vendas_diarias'].dtype)
```

O que aprendemos com a demonstração?

- `set_index()` é um método poderoso para transformar uma coluna em índice.

- `rename()` nos permite manter nossos dados claros e descritivos.
- O tratamento de valores ausentes (`fillna()`) funciona perfeitamente em uma coluna de DataFrame (que, lembre-se, é uma Series!), reforçando a conexão entre as duas estruturas.

□ Dicas e Armadilhas: Evitando Problemas Comuns

- **Tipos de Dados são Cruciais**: Sempre, sempre, **sempre** inspecione os tipos de dados com `.dtypes` ou `.info()` após criar ou carregar um DataFrame. O Pandas é inteligente, mas pode inferir tipos de forma inesperada, especialmente com valores ausentes. Por exemplo, uma coluna de inteiros com um `None` será convertida para `float` para acomodar o valor.
- **`.set_index()` e o Parâmetro `inplace`**: Quando você usa `set_index()`, por padrão, ele retorna um novo DataFrame. Se você quiser que a alteração ocorra no DataFrame original, use `df.set_index('coluna', inplace=True)`. Mas a boa prática é geralmente criar uma nova variável para evitar efeitos colaterais indesejados.
- **Conversão de Tipos com `.astype()`**: Às vezes, você precisará forçar um tipo de dado, como converter uma coluna de `string` para `int`. Use o método `.astype()`, mas seja cauteloso. Ele falhará se houver valores não numéricos em uma conversão para `int` ou `float`.

□ Transições e Conexões

- **Vem de**: A aula anterior sobre **Series**. Lembre-se: um DataFrame é uma coleção de Series. A compreensão de uma facilita a compreensão da outra.
- **Vai para**: A próxima aula, onde exploraremos a magia de **importar dados de arquivos reais**. A partir de agora, não precisaremos mais criar DataFrames manualmente! Aprenderemos a carregar arquivos CSV, Excel e outros, transformando dados brutos em DataFrames prontos para a análise.

□ Recursos Visuais e Mentais (Para Pensar)

1. **DataFrame como uma Tabela**: Pense em um DataFrame como um diagrama de xadrez: as linhas são as fileiras, as colunas são as colunas, e cada célula tem um valor.
2. **Fluxo de Trabalho**: Internalize o fluxo: **Criar (ou Importar) → Inspecionar → Ajustar Estrutura → Limpar/Tratar Dados → Analisar**.
3. **Visualização**: Lembre-se que você pode visualizar um DataFrame rapidamente usando ferramentas como `df.plot()` para ter uma ideia das distribuições dos seus dados.

Você está construindo uma base sólida para se tornar um verdadeiro mestre do Pandas. A transição da Series para o DataFrame é um grande passo, e você já está no caminho certo. Na próxima aula, a aventura fica ainda mais real quando começarmos a trabalhar com dados de verdade.

Pronto para importar e analisar dados do mundo real?

□ Exercícios

□ Parte 1: Criação de DataFrames

1. **Crie um DataFrame com os dados abaixo, usando um dicionário de listas:**

```
{
  'nome': ['Ana', 'Bruno', 'Carlos'],
  'idade': [23, 35, 31],
  'cidade': ['SP', 'RJ', 'MG']
}
```

2. **Crie um DataFrame usando uma lista de dicionários com os seguintes dados:**

```
[
  {'produto': 'Arroz', 'preco': 20},
  {'produto': 'Feijão', 'preco': 8}
]
```

3. **Crie um DataFrame a partir de um array NumPy `[[1, 2], [3, 4]]` com colunas chamadas `'A'` e `'B'`.**

4. **Transforme a Series abaixo em um DataFrame usando `.to_frame()`:**

```
pd.Series([100, 200, 300], name='valores')
```

5. **Crie um DataFrame com os dados de uma loja:**

```
{
  'dia': ['seg', 'ter', 'qua'],
  'vendas': [250, 300, 270],
  'loja': ['A', 'A', 'B']
}
```

□ Parte 2: Inspeção e Estrutura

6. **Exiba o número de linhas e colunas do DataFrame usando `.shape`.**

7. **Liste os nomes das colunas com `.columns`.**

8. **Liste os índices das linhas com `.index`.**

9. **Verifique os tipos de dados de cada coluna com `.dtypes`.**

10. **Use `.values` para exibir os dados como um array NumPy.**

□ Parte 3: Ajustes Estruturais

11. **Use `.set_index('dia')` para transformar a coluna `'dia'` em índice.**

12. **Renomeie a coluna `'vendas'` para `'vendas_diarias'` usando `.rename()`.**

13. **Converta a coluna `'vendas_diarias'` para o tipo `float` usando `.astype(float)`.**

14. **Transforme o DataFrame em um dicionário com `.to_dict()`.**

15. **Crie um novo DataFrame com os dados abaixo e altere o índice para `'nome'`:**

```
{
  'nome': ['João', 'Maria'],
  'idade': [28, 22],
  'cidade': ['SP', 'RJ']
}
```

□ Parte 4: Tratamento de Dados

16. **Crie um DataFrame com valores ausentes (`None`) e use `.isna()` para identificar os dados faltantes.**

17. **Use `.fillna()` para substituir os valores ausentes pela média da coluna.**

18. **Use `.dropna()` para remover linhas com valores ausentes.**

19. **Verifique se a coluna `'idade'` está no tipo correto (`int`). Se não estiver, converta.**

20. **Crie um DataFrame com os dados abaixo e trate o valor ausente:**

```
{
  'produto': ['Arroz', 'Feijão', 'Macarrão'],
  'preco': [20, None, 12]
}
```
