□ Aula 01 — A Jornada Começa: Dominando a `Series`

□ Olá, Futuro Cientista de Dados!

Bem-vindo à primeira etapa da sua jornada com **Pandas**! Nesta aula, vamos desvendar os segredos da estrutura de dados mais básica e, ao mesmo tempo, mais poderosa da biblioteca: a **Series**. Pense nela como a célula-tronco do Pandas — a partir dela, tudo se desenvolve.

Nosso objetivo aqui é ir além da sintaxe. Quero que você entenda o **porquê** por trás de cada comando, construindo uma base sólida para manipular dados como um profissional.

- **Tema Central**: A estrutura de dados **`Series`**
- **Comando(s) Essencial(is)**: `pd.Series()`, `.head()`, `.tail()`, `.describe()`, `.mean()`, `.sum()`, `.isna()`, `.fillna()`, `.dropna()`, `.rolling()`

□ A Series: O Coração Unidimensional do Pandas

Imagine que você tem uma única coluna de uma planilha: os valores de vendas diárias, a idade de um grupo de pessoas, as notas de um aluno em diferentes disciplinas. Esses dados, por si só, já carregam um significado. A **Series** do Pandas é exatamente isso: uma estrutura de dados de uma dimensão, como uma lista, mas com uma superpotência.

Essa superpotência é o seu **índice** (ou rótulo).

Enquanto uma lista comum do Python usa posições numéricas (0, 1, 2...), a Series permite que você nomeie esses "endereços" de forma significativa, como `'segunda-feira'`, `'nota_do_aluno'`, ou `'vendas_de_março'`.

Por que isso é tão importante? Porque permite que o Pandas execute operações de forma **inteligente**. Ele não se importa apenas com os valores, mas também com os rótulos. Isso evita erros e torna a manipulação de dados muito mais intuitiva, como veremos mais adiante. A Series é a fundação para a estrutura de dados mais complexa e famosa do Pandas: o **DataFrame**, que é basicamente uma coleção de Series.

□ Conteúdo Explicado: Mergulhando Fundo

1. Criando a `Series`: O Gênesis dos Dados

A função `pd.Series()` é o seu portal para criar essa estrutura de dados mágica. Ela é bastante flexível e aceita diversos tipos de dados como entrada. Vamos explorar os principais.

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

- `data`: Os valores que você quer armazenar (uma lista, um array, um dicionário...).
- `index`: Os rótulos que você quer dar a esses valores. **Se você não fornecer um, o Pandas criará um índice numérico padrão (`0, 1, 2...`)**.
- `name`: Uma ótima prática é nomear sua Series, especialmente quando ela for se tornar uma coluna de um DataFrame.

---

a) O Modo Mais Simples: A partir de uma lista

Começamos com o básico. O Pandas, por padrão, já entende o que fazer.

```
import pandas as pd
s = pd.Series([10, 20, 30])
print(s)
```

Saída Esperada:

```
0    10
1    20
2    30
dtype: int64
```

O que aconteceu? O Pandas pegou sua lista de valores e, como não especificamos um índice, ele criou um para nós. Note que ele também detectou o tipo de dado (`dtype`) automaticamente. Simples assim!

---

b) Adicionando Rótulos e Nome

Aqui, a mágica começa a acontecer. Vamos dar nomes aos nossos dados, tornando-os mais significativos.

```
s = pd.Series([10, 20, 30], index=['a', 'b', 'c'], name='valores')
print(s)
```

Saída:

```
a    10
b    20
c    30
Name: valores, dtype: int64
```

Agora, cada valor está associado a um rótulo personalizado. O nome da Series (`valores`) torna a sua intenção ainda mais clara.

---

c) O Atalho Inteligente: Usando um Dicionário

Se seus dados já vêm como um dicionário, o Pandas é esperto o suficiente para saber que as **chaves** devem se tornar os **índices**.

```
s = pd.Series({'jan': 100, 'fev': 150, 'mar': 120})
print(s)
```

Saída:

```
jan    100
fev    150
mar    120
dtype: int64
```

Esse é um dos recursos mais convenientes do Pandas para iniciar a manipulação de dados de forma rápida.

---

d) Outras Maneiras de Criar

A Series é compatível com outras estruturas de dados muito comuns:

- **Do NumPy**:

```
import numpy as np
arr = np.array([1.5, 2.0, 3.7])
s = pd.Series(arr, index=['x', 'y', 'z'])
```

- **A partir de um valor único (Scalar)**:

```
s = pd.Series(5, index=['a', 'b', 'c'])
```

Isso cria uma Series onde todos os valores são `5`, um truque útil para inicializar dados.

---

2. Acessando os Dados: A Arte da Seleção

A grande vantagem de ter índices é a flexibilidade para acessar seus dados. Você pode usar tanto a posição quanto o rótulo.

- **Seleção simples**: `s[0]` (posição) ou `s['a']` (rótulo).
- **Seleção explícita com `.loc` e `.iloc`**: Essa é a forma mais recomendada, pois evita ambiguidades.
  - `.loc` (de "location") usa **rótulos**. Use `s.loc['a']`.
  - `.iloc` (de "integer location") usa **posições inteiras**. Use `s.iloc[0]`.

---

3. A Matemática Vetorizada: Eficiência Máxima

Um dos maiores benefícios do Pandas é a capacidade de realizar operações matemáticas em todos os elementos de uma Series de uma só vez, sem precisar de loops.

```
s = pd.Series([10, 20, 30])
s_dobrada = s * 2
# Saída:
# 0    20
# 1    40
# 2    60
```

Isso é incrivelmente mais rápido e eficiente do que usar um loop `for`.

A Armadilha do Alinhamento de Índices: A inteligência da Series vai ainda mais longe. Quando você soma duas Series, o Pandas alinha os dados pelos índices antes de somar, gerando `NaN` (Not a Number) onde não há correspondência.

```
s1 = pd.Series([1, 2, 3], index=['a', 'b', 'c'])
s2 = pd.Series([10, 20], index=['a', 'b'])
s1 + s2
```

Saída:

```
a    11.0
b    22.0
c     NaN
dtype: float64
```

Note que o resultado para o índice `'c'` foi `NaN`. Isso é um comportamento intencional e muito poderoso, que previne erros de dados desalinhados. O Pandas prefere te avisar com um `NaN` do que somar dados de forma incorreta.

---

4. Métodos Utilitários: Seu Kit de Ferramentas de Análise

A Series vem com uma série de métodos já prontos para a análise. Eles são a sua caixa de ferramentas para rapidamente inspecionar e resumir os dados.

- `s.head(n)`: Mostra as primeiras `n` linhas (ótimo para dar uma espiada).
- `s.tail(n)`: Mostra as últimas `n` linhas.
- `s.describe()`: Calcula estatísticas descritivas (média, desvio padrão, mínimo, quartis...).
- `s.mean()` e `s.sum()`: Cálculos rápidos e diretos.
- `s.value_counts()`: Contagem de ocorrências únicas de cada valor.
- `s.unique()`: Retorna os valores únicos.
- `s.to_frame()`: Converte sua Series em um DataFrame (o que veremos na próxima aula!).
- `s.to_numpy()`: Transforma sua Series em um array NumPy.

---

5. Lidando com o Lado Escuro dos Dados: Valores Ausentes

Dados do mundo real raramente são perfeitos. Quase sempre haverá valores ausentes, que o Pandas representa como `NaN`. Saber lidar com eles é uma habilidade fundamental.

- `s.isna()`: Retorna uma Series de `True`/`False` indicando onde os valores estão ausentes.
- `s.dropna()`: Remove as linhas com valores ausentes.
- `s.fillna(valor)`: Preenche os valores ausentes com um valor que você escolher (por exemplo, a média da Series).

□ Hora de Praticar: Um Exemplo do Dia a Dia

Vamos unir tudo o que vimos até agora em um exemplo prático. Imagine que você está analisando as vendas de uma semana.

```
import pandas as pd
import numpy as np

# Vamos criar uma Series de vendas, simulando um dia sem registro (NaN)
vendas = pd.Series([250, 300, np.nan, 410, 380], index=['seg', 'ter', 'qua', 'qui', 'sex'], name='vendas')
print("--- Vendas Brutas ---")
print(vendas)
print("\n--- Descrição Estatística ---")
print(vendas.describe())
```

O que fizemos? Criamos a Series com os dados e índices rotulados. Usamos `np.nan` (do NumPy) para representar o dia sem registro. Em seguida, usamos `.describe()` para ter uma visão geral dos nossos dados de forma instantânea.

□ Demonstração Aplicada: A Análise em 4 Passos

Agora, vamos aplicar um pequeno fluxo de análise de dados, como se estivéssemos em uma situação real.

```
# 1. Criação: Nossa Series de vendas já com um valor ausente
vendas = pd.Series({'seg': 250, 'ter': 300, 'qua': None, 'qui': 410, 'sex': 380}, name='vendas')

# 2. Inspeção: Usamos describe para entender a distribuição dos dados
print("--- Estatísticas iniciais ---")
print(vendas.describe())
```

```
# 3. Tratamento de Dados Ausentes: Preenchemos o dia que faltou com a média
vendas_tratadas = vendas.fillna(vendas.mean())
print("\n--- Vendas após preencher 'qua' com a média ---")
print(vendas_tratadas)

# 4. Cálculo de Indicadores: A magia da análise!
total_vendas = vendas_tratadas.sum()
media_vendas = vendas_tratadas.mean()
dia_de_pico = vendas_tratadas.idxmax() # idxmax retorna o índice do valor máximo
media_movel_3dias = vendas_tratadas.rolling(window=3).mean() # Média móvel de 3 dias

print(f"\nTotal de vendas na semana: {total_vendas}")
print(f"Média diária de vendas: {media_vendas:.2f}")
print(f"Dia com mais vendas: {dia_de_pico}")
print("\n--- Média Móvel de 3 Dias ---")
print(media_movel_3dias)
```

O que aprendemos aqui?

- `fillna()` com `mean()` é uma técnica comum para preencher dados ausentes de forma inteligente.
- Métodos como `idxmax()` e `rolling()` são atalhos poderosos para encontrar informações e tendências rapidamente.

□ Dicas Práticas e Armadilhas Comuns para Evitar

- **Alinhamento de Índices**: Sempre se lembre que o Pandas fará o alinhamento de índices antes de uma operação. Fique atento aos `NaN`s que isso pode gerar.
- **Tipos de Dados e `NaN`**: Se a sua Series for de tipo inteiro (`int64`) e você adicionar um `NaN`, ela será automaticamente convertida para `float64` porque `NaN` não é um inteiro. Isso é um comportamento normal, mas é algo que você deve estar ciente.
- **`.loc` vs `.iloc`**: A prática recomendada é usar `.loc` para seleções por rótulo e `.iloc` para seleções por posição. A diferença é sutil, mas evita comportamentos inesperados.

□ Transições e Conexões

- **Vem de**: Sua experiência com listas e dicionários em Python. A Series é a próxima evolução dessas estruturas, otimizada para análise de dados.
- **Vai para**: A próxima aula, onde vamos desmistificar o **DataFrame**. Agora que você entende o que é uma `Series`, será muito mais fácil entender o DataFrame, que é, essencialmente, uma coleção de Series que compartilham um mesmo índice.

□ Recursos Visuais e Mentais (Para Pensar)

1. **Series vs. DataFrame**: Pense na Series como uma única coluna, e no DataFrame como uma planilha inteira. O DataFrame é a união de várias Series.
2. **O Fluxo da Análise**: Mantenha este fluxo mental para qualquer análise de dados: **Criar → Inspecionar → Tratar** (dados ausentes, erros) **→ Calcular Indicadores → Visualizar**.
3. **Diagrama de Alinhamento**: Imagine duas Series, cada uma com seus índices. Ao somá-las, o Pandas desenha uma linha imaginária entre os índices correspondentes e só então realiza a soma. Onde não há correspondência, ele escreve `NaN`.

Espero que esta nova abordagem tenha tornado a compreensão da **Series** mais clara e envolvente! Agora que você domina essa estrutura fundamental, a transição para o DataFrame será muito mais tranquila.

Pronto para o próximo passo?

□ Exercícios

□ Parte 1: Criação e Estrutura

1. **Crie uma Series com os números `5`, `10`, `15`, `20`. Exiba-a na tela.**
2. **Crie uma Series com os valores `[100, 200, 300]` e índices personalizados `['A', 'B', 'C']`. Dê o nome “valores”.**
3. **Crie uma Series a partir de um dicionário com os meses `'jan': 120`, `'fev': 150`, `'mar': 130`.**

4. **Crie uma Series com valor escalar `7` e índices `['x', 'y', 'z']`.**
5. **Crie uma Series a partir de um array NumPy com os valores `[1.1, 2.2, 3.3]` e índices `['a', 'b', 'c']`.**

□ Parte 2: Acesso e Seleção

6. **Crie uma Series com os valores `[10, 20, 30, 40, 50]`. Exiba o terceiro valor usando `.iloc`.**
7. **Utilize `.loc` para acessar o valor com índice `'B'` em uma Series com índices `['A', 'B', 'C']`.**
8. **Exiba os dois últimos elementos de uma Series usando `.tail()`.**
9. **Utilize fatiamento para exibir os valores entre os índices `'seg'` e `'qua'` em uma Series de dias da semana.**
10. **Crie uma Series com 5 valores e exiba apenas os valores maiores que `25`.**

□ Parte 3: Operações e Estatísticas

11. **Crie duas Series com índices parcialmente sobrepostos e some-as. Observe o alinhamento automático.**
12. **Utilize `.mean()` para calcular a média de uma Series de notas.**
13. **Utilize `.sum()` para calcular o total de vendas semanais.**
14. **Utilize `.describe()` para obter estatísticas de uma Series de temperaturas.**
15. **Utilize `.value_counts()` para contar quantas vezes cada valor aparece em uma Series.**

□ Parte 4: Tratamento de Dados Ausentes

16. **Crie uma Series com valores `[10, np.nan, 30, np.nan, 50]`. Use `.isna()` para identificar os ausentes.**
17. **Utilize `.fillna()` para substituir os valores ausentes pela média da Series.**
18. **Utilize `.dropna()` para remover os valores ausentes.**
19. **Crie uma Series com dados de vendas e calcule a média móvel com janela de 3 dias usando `.rolling().mean()`.**
20. **Converta uma Series para DataFrame com `.to_frame()` e depois para NumPy array com `.to_numpy()`.**