



SÉRIES DO PANDAS (PYTHON)

Guia completo de Series: estrutura 1D do pandas, indexação (loc, iloc), operações vetorizadas, métodos (mean, sum, value_counts), casos práticos (estatísticas, filtros), Series vs List.

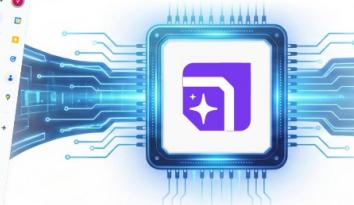
Gere ebooks como este com



Ebookr

em <https://ebookr.ai>

Crie ebooks profissionais incríveis em minutos com IA



Chega de formatar texto no Google Docs, Word ou ferramentas que só te fazem perder tempo...

E deixe que nossa IA faça o trabalho pesado!



Capas gerados por IA



Infográficos feitos por IA



Edite em Markdown em Tempo Real



Adicione Banners Promocionais

TESTE AGORA



PRIMEIRO CAPÍTULO 100% GRÁTIS

 **Atualizado para Python 3.13** (Dezembro 2025) Pandas 2.2+ com Series otimizada, operações vetorizadas e casos práticos.

Pandas, Pandas **everywhere!**

Series é a estrutura 1D do pandas - como uma lista turbinada com índices e operações vetorizadas!

Neste guia:

-  **Criação** - A partir de listas, dicts, arrays
-  **Indexação** - loc, iloc, booleana
-  **Operações** - Vetorização, agregadores
-  **Casos práticos** - Estatísticas, filtros

Hoje vamos continuar nossa **Jornada** no mundo da Ciência de Dados com Python!

Prontos pra conhecer as Séries do Pandas???



O, produção?? De novo??? 😞

Esse é um post **muito** importante!

Fizemos com muito carinho para você saia daqui entendendo os principais componentes do Pandas: séries.

E não perca o [post de dataframes](#) que vem na sequência!!

Conhecendo bem esses conceitos, o caminho para o tratamento de dados fica muito mais suave.

Já fizemos outro post de Pandas, mostrando como importar arquivos CSV! [Acesse o post clicando aqui!](#)

Como eu sei que vocês são apressados, **vamos nessa!**

Definição de Séries

Começando pelo começo, o componente basilar do Pandas são as **Séries**.

Uma série é uma espécie de arranjo unidimensional, como uma lista, mas possui algumas características diferentes.

Uma série tem 4 partes importantes:

- Os elementos em si
- O índice que contém a referência para acessar os elementos
- O tipo dos elementos
- Um nome

Como jack, o estripador, vamos por partes.

Elementos e Tipos

Os elementos podem ser de qualquer tipo, ou seja, podemos ter uma série com números e strings, por exemplo.

Como o pandas é implementado utilizando `numpy`, colocar tipos diferentes numa mesma série não é recomendado, porque perdemos muitas das vantagens de performance quando são do mesmo tipo.

Até aqui, nenhum segredo.

Abaixo, criamos duas séries de exemplo de forma bem parecida como criamos a lista, com a exceção de que as criamos a partir da classe `Series` do pandas:

```
>>> import pandas as pd
>>> serie = pd.Series([42, 99, -1])
>>> serie
0    42
1    99
2    -1
dtype: int64
>>> serie2 = pd.Series(['radiohead', 2.3, True])
>>> serie2
0    radiohead
1        2.3
2      True
dtype: object
```

Percebam que a primeira série tem o tipo `int64`, pois todos os elementos são inteiros.

Já a segunda é uma salada, com string, um número racional e um valor booleano.

O tipo da `serie2` é `object`, ou seja, um tipo bem genérico para abranger a bagunça que fizemos com tipos diferentes de elementos.

Assim, concluimos a primeira parte:

Elementos de uma série

Agora, vamos para uma parte muito importante, como acessar os elementos.

Mas primeeeeiro...

Acessando elementos

Numa lista, acessamos os elementos por meio de índices posicionais, numéricos, certo?

Acessar o primeiro elemento: lista[0], o terceiro elemento: lista[2], e assim por diante.

Nas séries podemos acessar da mesma forma! Vamos testar:

```
>>> serie[0]  
42  
>>> serie[2]  
-1
```

Muita atenção, agora!

Nesse exemplo acessamos os elementos com um índice posicional, mas não precisa ser assim, podemos criar um índice próprio que nem precisa ser numérico!

Nós mesmos podemos criar os índices do jeito que bem entendermos: números, strings, tuplas. Meio estranho? Vai ficar fácil com exemplos.

Imagine que queremos guardar as calorias de cada alimento (tô de dieta 😊).

Podemos criar uma série com as calorias de uma banana, um prato feito e um big mac (nham, que fome):

```
>>> import pandas as pd  
>>> serie = pd.Series([200, 350, 550])  
>>> serie  
0    200  
1    350  
2    550  
dtype: int64
```

Quando não passamos quais devem ser os índices de uma série, o pandas cria um objeto do tipo `RangeIndex` que vai de 0 até o número de elementos menos um: no exemplo acima, de 0 a 2.

Mas, e agora? Qual caloria é do big mac? Se eu comer uma banana, quantas calorias estou ingerindo?

Com esse índice, para acessar as calorias do prato feito basta executar `serie[1]`:

```
>>> serie[1]  
350
```

Vamos criar um índice mais legal, com o nome dos alimentos?

```
>>> import pandas as pd  
>>> serie = pd.Series([200, 350, 550], index=['banana', 'prato feito',  
        'big mac'])  
>>> serie  
banana      200  
prato feito  350  
big mac     550  
dtype: int64
```

Agora sim! Será que posso comer um big mac na minha dieta?

```
>>> serie['big mac']  
550
```

Acho que não 😬

Maneiro? É bem importante entendermos como funcionam os índices, porque o acesso aos dados em séries e dataframes se dá por meio deles!

Vamos a mais um exemplo, esse pra mostrar como os índices são flexíveis:

```
>>> import pandas as pd  
>>> serie = pd.Series([200, 350, 550], index=[(0, 0), (0, 1), (0, 2)])  
>>> serie  
(0, 0)    200  
(0, 1)    350  
(0, 2)    550  
dtype: int64
```

Sim, tuplas como índices! E como acessar os elementos? Só passar a tupla certa e pronto:

```
>>> serie[(0,1)]  
350
```

 *Estou construindo o [Ebookr.ai](#), uma plataforma onde você cria ebooks profissionais com IA sobre qualquer assunto — do zero ao PDF pronto, com capas e infográficos gerados automaticamente. Dá uma olhada!*

Crie **Ebooks profissionais incríveis** em minutos com IA



Chega de formatar texto no Google Docs, Word ou ferramentas que só te fazem perder tempo...

... e deixe que nossa IA faça o trabalho pesado!

TESTE AGORA! PRIMEIRO CAPÍTULO 100% GRÁTIS 

 Capas gerados por IA

 Adicione Banners Promocionais

 Edite em Markdown em Tempo Real

 Infográficos feitos por IA

Um nome pra chamar de seu

Last, but not least: as séries do pandas podem ter um nome!

```
>>> import pandas as pd
>>> serie = pd.Series([200, 350, 550], index=['banana', 'prato feito',
   'big mac'], name='calorias')
>>> serie
banana      200
prato feito  350
big mac     550
Name: calorias, dtype: int64
```

Vejam que criamos explicitamente um nome, ‘calorias’, para nossa série.

Casos Práticos

1. Análise Estatística Rápida

```
import pandas as pd

notas = pd.Series([7.5, 8.0, 6.5, 9.0, 7.0, 8.5], name='notas')

print(f'Média: {notas.mean():.2f}')
print(f'Mediana: {notas.median():.2f}')
print(f'Desvio padrão: {notas.std():.2f}')
print(f'Mínimo: {notas.min()}, Máximo: {notas.max()}')

# Estatísticas completas
print(notas.describe())
```

2. Filtros Complexos

```
import pandas as pd

precos = pd.Series([100, 250, 50, 300, 150], index=['A', 'B', 'C', 'D', 'E'])

# Filtrar preços entre 100 e 200
filtro = (precos >= 100) & (precos <= 200)
produtos_filtrados = precos[filtro]
print(produtos_filtrados)

# A      100
# E      150
```

3. Operações Vetorizadas

```
import pandas as pd

vendas = pd.Series([1000, 1500, 2000, 1200])

# Aplicar desconto de 10%
vendas_com_desconto = vendas * 0.9

# Adicionar taxa fixa
vendas_com_taxa = vendas + 50

# Operações complexas
vendas_final = (vendas * 0.9) + 50
print(vendas_final)
```

4. Value Counts

```
import pandas as pd

categorias = pd.Series(['A', 'B', 'A', 'C', 'B', 'A', 'D', 'B'])

# Contar ocorrências
contagem = categorias.value_counts()
print(contagem)
# A    3
# B    3
# C    1
# D    1

# Percentuais
percentuais = categorias.value_counts(normalize=True)
print(percentuais)
```

Series vs List

```
import pandas as pd
import numpy as np

# Lista Python
lista = [10, 20, 30, 40, 50]
result_lista = [x * 2 for x in lista] # Loop necessário

# Series Pandas (vetorizada!)
serie = pd.Series([10, 20, 30, 40, 50])
result_serie = serie * 2 # Operação vetorizada!

# Series tem métodos úteis
print(serie.mean()) # ❌ lista.mean() não existe!
print(serie.std())
print(serie.describe())
```

Quando Usar Series?

✓ Use Series quando:

- Dados **numéricos** ou categóricos
- Precisa **operações** estatísticas
- Quer **indexação** flexível
- Trabalha com **DataFrames** (colunas são Series)

✓ Use List quando:

- Dados **heterogêneos**
- Operações **simples**
- Não precisa pandas

- **Performance crítica** em loops

Conclusão

Neste guia de **Series**, você aprendeu:

- ✓ **Criação** - A partir de listas, dicts, arrays
- ✓ **Indexação** - loc (rótulos), iloc (posições)
- ✓ **Operações** - Vetorização automática
- ✓ **Métodos** - mean(), sum(), value_counts()
- ✓ **Casos práticos** - Estatísticas, filtros, contagens

Principais lições:

- Series = **array 1D com índices**
- Operações são **vetorizadas** (rápidas!)
- Métodos **estatísticos** integrados
- Colunas de DataFrame são **Series**
- Mais poderosa que **listas** para análise

Próximos passos:

- Explore [DataFrames](#) (Series 2D)
- Aprenda [read_csv\(\)](#)
- Pratique indexação avançada
- Estude métodos de string (`.str`)

Este foi mais um post da **Série Pandas** (trocadilho intencional 😜)!

Daqui vocês podem seguir direto para o [post de Dataframes](#)!

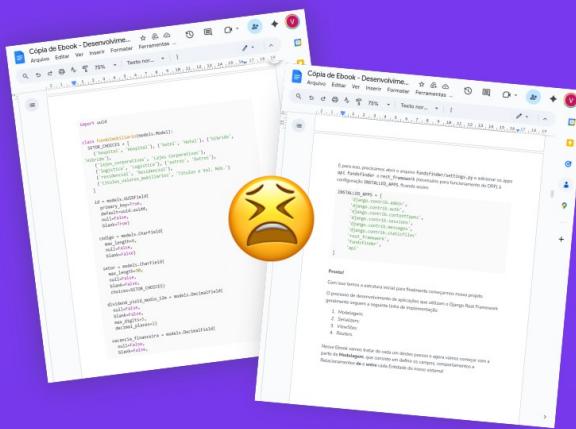
Até a próxima!

Não se esqueça de conferir!



Ebookr

Crie Ebooks profissionais em minutos com IA



Chega de formatar código no Google Docs ou Word



Capas gerados por IA



• Infográficos feitos



Adicione Banners Promocionais

Deixe que nossa IA faça o trabalho pesado



Edite em Markdown em Tempo Real

TESTE AGORA



 PRIMEIRO CAPÍTULO 100% GRÁTIS