



PYTHON
ACADEMY

MANIPULAÇÃO DE DADOS COM DATAFRAMES DO PANDAS

Guia completo de DataFrames: estrutura 2D do pandas, seleção (loc, iloc, at, iat), filtros booleanos, groupby, merge/join, casos práticos (análise vendas, agregações), DataFrame vs Series.

Gere ebooks como este com



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 DataFrames otimizados e casos práticos essenciais.

Pandas, Pandas mais uma vez!



Agora que você é craque em Séries do Pandas, vamos ao complemento dessa dupla inseparável: Séries e Dataframes Pandas!

Definição de DataFrame

Depois de aprender tudo sobre Séries, chegou a hora de aprender tudo sobre **Dataframes**.

Como assim? Não sabe de Séries?? Pode parar por aqui então e ler nosso último post sobre as Séries do Pandas.

Como a gente é bonzinho, ele tá [aqui ó](#).

Um jeito fácil de entender: Séries são colunas, Dataframes são tabelas!

Na verdade, um Dataframe é formado por um conjunto de séries, cada uma delas sendo uma coluna da ‘tabela’.

Para criar um Dataframe, é simples! Só precisamos instanciar um objeto a partir da classe `DataFrame` do Pandas:

```
>>> import pandas as pd
>>> df = pd.DataFrame([200, 350, 550])
>>> df
   0
0  200
1  350
2  550
```

Exemplo simples, mas já mostra que alguns componentes de um Dataframe.

Assim como as séries, Dataframes possuem índices, mas também possui colunas, cada uma delas com um nome.

Como não passamos explicitamente o nome das colunas, o pandas cria... adivinhem: um `RangeIndex` de 0 até o número de colunas menos um.

Mas, vamos devagar. No exemplo anterior criamos um Dataframe a partir de uma lista.

Outra forma é criar a partir de um dicionário, onde a chave é o nome da coluna e o valor, uma lista de elementos:

```
>>> import pandas as pd
>>> df = pd.DataFrame({'calorias':[200, 350, 550], 'gordura (%)':[0, 6, 15]})
>>> df
```

	calorias	gordura (%)
0	200	0
1	350	6
2	550	15

Agora, podemos alterar os índices para algo mais mnemônico, que facilite nossa vida:

```
>>> import pandas as pd
>>> df = pd.DataFrame({'calorias':[200, 350, 550], 'gordura (%)':[0, 6, 15]}, index=['banana', 'prato feito', 'big mac'])
>>> df
```

	calorias	gordura (%)
banana	200	0
prato feito	350	6
big mac	550	15

Vale lembrar que tudo que funcionava com o índice das séries também vale para os Dataframes!

Agora vamos aprender a acessar os dados de um Dataframe.

Acesso aos Dados

Essa parte é um capítulo à parte de termos um post detalhando várias formas de acessar os dados de um Dataframe.

Alguns dos métodos, como `loc`, `iloc`, `at` e `iat` merecem bastante detalhe e exemplos, mas traremos isso pra vocês em breve!

A verdade é que existem MUITAS formas de acessar os dados de um Dataframe, mas não vamos deixar os padawans na mão: a Python Academy sempre ao serviços de vocês ❤️

Uma forma simples de acessar todos os valores de uma coluna de um Dataframe é utilizar o `df[coluna]`, dessa forma:

```
>>> df['gordura (%)']
banana          0
prato feito     6
big mac        15
Name: gordura (%), dtype: int64
```

Que estranho?! Parece uma série!

Certíssimo!

Lembra que dissemos que uma coluna de um Dataframe é uma série? Vamos testar:

```
>>> type(df['gordura (%)'])
<class 'pandas.core.series.Series'>
```

E para acessar os valores de uma linha?

Vamos recorrer aos índices!


```
>>> df = pd.DataFrame({'calorias':[200, 350, 550], 'gordura (%)':[0, 6, 15]})
>>> df
banana          0
prato feito     6
big mac        15
Name: gordura (%), dtype: int64
```

Nesse caso, como não criamos o índice, ele é numérico e corresponde à posição de cada linha.

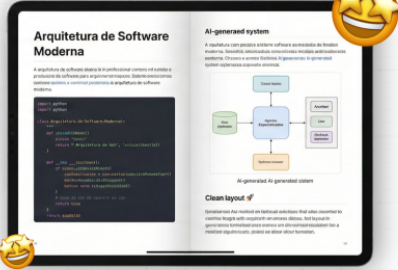

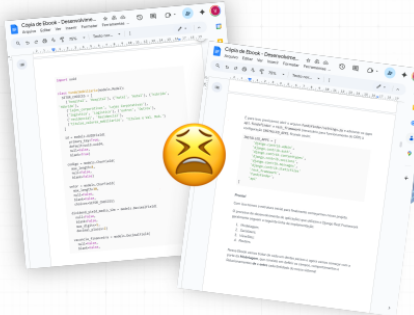
Uma forma é acessar como acessamos elementos de uma lista, utilizando *slice*:

```
>>> df[1:2]
   calorias  gordura (%)
1       350         6
```

💡 Estou desenvolvendo o [Ebookr.ai](https://ebookr.ai), uma plataforma que transforma suas ideias em ebooks profissionais usando IA — com geração de capa, infográficos e exportação em PDF. Te convido a conhecer!




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

Outros tipos de acesso (`loc`)

Adiantando um assunto importante, podemos também utilizar o método `loc`, que acessa por meio do índice:

```
>>> df.loc[0]
calorias      200
gordura (%)    0
Name: 0, dtype: int64
```

Percebam que a apresentação dos dois resultados é bem diferente!

Isso porque utilizando `slice`, o resultado é um novo Dataframe com as linhas selecionadas.

No segundo caso, o `loc` retorna uma série com os valores da linha selecionada.

Outro assunto que traremos pra vocês em mais detalhes: quando temos Dataframes, Séries e Cópias como resultados.

Séries são a base de tudo

Peraí, peraí: a Série não era uma coluna, agora é uma linha?

Ótima pergunta, Pythonista!

Olha que legal, o Pandas retorna tanto uma coluna como uma linha completa (com o respectivo índice) como uma Série!

Uma série “coluna” possui um nome (o título da coluna), os valores (elementos da coluna) bem como o tipo.

De forma similar, uma série “linha” possui um nome (o índice que referencia a linha), os valores (elementos de uma linha) bem como tipo.

As séries são tão flexíveis e são uma mão na roda para representar os dados, não é à toa que os Cientistas de Dados utilizam Dataframes sempre que possível.

Casos Práticos Essenciais

1. Análise de Vendas

```
import pandas as pd

df = pd.DataFrame({
    'produto': ['Mouse', 'Teclado', 'Mouse', 'Monitor'],
    'quantidade': [10, 5, 15, 2],
    'preco': [50, 150, 50, 800]
})

# Total por produto
df['total'] = df['quantidade'] * df['preco']

# Agrupar e somar
vendas_por_produto = df.groupby('produto')['total'].sum()
print(vendas_por_produto)
```

2. Filtros e Agregações

```
import pandas as pd

df = pd.read_csv('vendas.csv')

# Filtrar vendas > 1000
grandes_vendas = df[df['valor'] > 1000]

# Média por categoria
media_categoria = df.groupby('categoria')['valor'].agg(['mean', 'sum',
                                                         'count'])
print(media_categoria)
```

3. Merge de DataFrames

```
import pandas as pd

clientes = pd.DataFrame({'id': [1, 2, 3], 'nome': ['Alice', 'Bob',
                                                  'Carol']})
pedidos = pd.DataFrame({'cliente_id': [1, 2, 1], 'valor': [100, 200,
                                                         150]})

# Inner join
result = pd.merge(clientes, pedidos, left_on='id',
                  right_on='cliente_id')
print(result)
```

Conclusão

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

✓ **Estrutura 2D** - Tabelas com linhas e colunas ✓ **Seleção** - loc (rótulos), iloc (posições) ✓ **Filtros** - Condições booleanas ✓ **Agregações** - groupby(), sum(), mean() ✓ **Merge** - Combinar DataFrames

Próximos passos:

- Aprenda [read_csv\(\)](#)
- Explore [Séries](#)
- Pratique `pivot_table()`
- Estude merge vs join vs concat

É isso, galera! Esse post + [post de Séries](#) dão uma boa base para conhecer a base de Pandas!

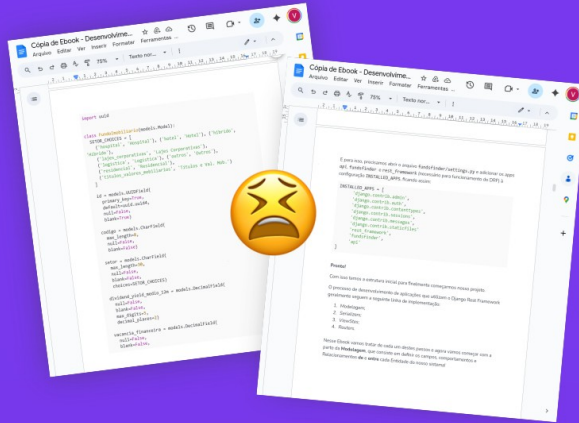
Até a próxima!

Não se esqueça de conferir!

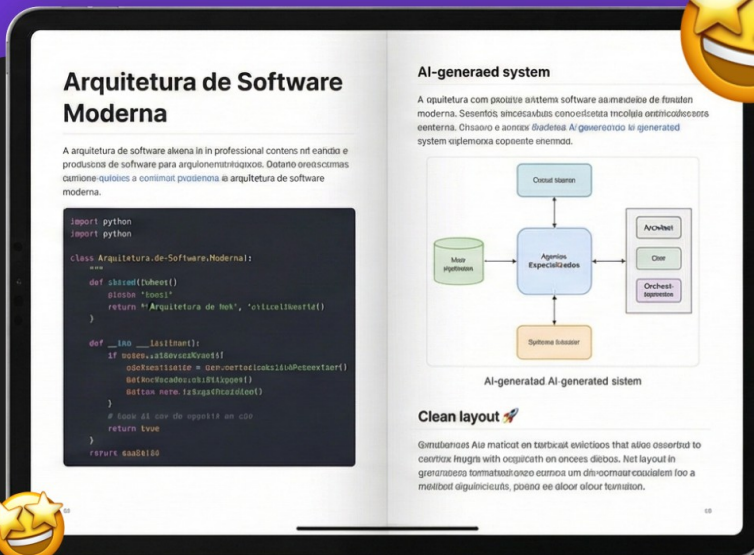


Ebookr

Crie Ebooks profissionais em minutos com IA



Chega de formatar código no Google Docs ou Word



Capas gerados por IA



Infográficos feitos por IA



Adicione Banners Promocionais

Deixe que nossa IA faça o trabalho pesado



Edite em Markdown em Tempo Real

TESTE AGORA



PRIMEIRO CAPÍTULO 100% GRÁTIS