



Engenharia de Software
Computacional thinking with Python
Aula 03 –

Estrutura de dados unidimensional e bidimensional

Prof. Dr. Francisco Elânio

Agenda da Aula

- Estrutura de dados com pandas e numpy
- Exemplos de estruturas com pandas e numpy
- Estruturas de dados do tipo vetor (dados homogêneos)
- Estrutura unidimensional e bidimensional
- Exercícios em sala

Estrutura de dados em Python

O Pandas fornece estruturas de dados de alto nível e funções projetadas para tornar o trabalho com dados estruturados ou tabulares rápido, fácil e expressivo. Desde o seu surgimento em 2010, tem ajudado a permitir que o Python seja um ambiente de análise de dados poderoso e produtivo.

Os objetos primários em pandas são o DataFrame, uma estrutura de dados tabular orientada a colunas com rótulos de linha e coluna, e o Series, um objeto de matriz rotulado unidimensional.

```
import pandas as pd
```

```
s = pd.Series([10, 20, 30, 40, 50])
```

```
print(s)
```

Series da biblioteca Pandas

```
import pandas as pd
```

```
# Criando duas Series Pandas
```

```
s1 = pd.Series([10, 20, 30, 40, 50])
```

```
s2 = pd.Series([1, 2, 3, 4, 5])
```

```
# Soma
```

```
soma = s1 + s2
```

```
print("Soma:")
```

```
print(soma)
```

```
# Subtração
```

```
subtracao = s1 - s2
```

```
print("\nSubtração:")
```

```
print(subtracao)
```

```
# Multiplicação
```

```
multiplicacao = s1 * s2
```

```
print("\nMultiplicação:")
```

```
print(multiplicacao)
```

```
# Divisão
```

```
divisao = s1 / s2
```

```
print("\nDivisão:")
```

```
print(divisao)
```

Nome para cada série

```
s1 = pd.Series([10, 20, 30, 40, 50], name='Série 1')
```

```
s2 = pd.Series([1, 2, 3, 4, 5], name='Série 2')
```


Series da biblioteca Pandas

```
import pandas as pd
```

```
# Criando duas Series Pandas com nomes de colunas
```

```
s1 = pd.Series([10, 20, 30, 40, 50], name='Série 1')
```

```
s2 = pd.Series([1, 2, 3, 4, 5], name='Série 2')
```

```
# Criando um DataFrame a partir das Series
```

```
df = pd.DataFrame({'Série 1': s1, 'Série 2': s2})
```

```
print(df)
```

Pandas vs NumPy

Pandas e NumPy são duas bibliotecas muito populares em Python para análise e manipulação de dados. Embora possam parecer semelhantes em certos aspectos, cada uma delas tem suas próprias características e é adequada para diferentes tipos de tarefas.



Pandas	NumPy
pode conter tipos de dados diferentes	tem dados homogêneos
operações tabulares, tarefas de pré-processamento semântico semelhantes a SQL	computação numérica, operações matriciais e vetoriais
duas dimensões	multidimensional (>2 possível)
mais memória	menos memória
Mais devagar	mais rápido



Estrutura de Dados

NumPy trabalha principalmente com arrays multidimensionais homogêneos (ndarrays), que são eficientes para operações numéricas vetorizadas.

Pandas fornece estruturas de dados mais flexíveis, como Series (uma dimensão) e DataFrame (duas dimensões), que são mais adequadas para manipulação de dados tabulares e possuem recursos para lidar com dados heterogêneos, incluindo rótulos de linhas e colunas.

```
import numpy as np
```

```
arr = np.array([[1, 2, 3], [4, 5, 6]])  
result = arr * 2  
print("Array NumPy original:")  
print(arr)  
print("\nArray NumPy após multiplicação por 2:")  
print(result)
```

```
import pandas as pd
```

```
data = {'A': [1, 2, 3], 'B': [4, 5, 6]}  
df = pd.DataFrame(data)  
  
df['C'] = df['A'] + df['B']  
  
print("DataFrame Pandas:")  
print(df)
```

Indexação

NumPy usa indexação baseada em números inteiros para acessar elementos em arrays, semelhante ao que é encontrado em listas Python.

Pandas permite indexação baseada em rótulos, o que torna mais fácil referenciar e manipular dados usando nomes de colunas ou índices específicos.

```
import numpy as np
```

```
arr = np.array([[1, 2, 3], [4, 5, 6]])  
element = arr[1, 2]
```

```
print("Elemento acessado com NumPy:", element)
```

```
import pandas as pd
```

```
data = {'A': [1, 2, 3], 'B': [4, 5, 6]}  
df = pd.DataFrame(data)
```

```
value = df.loc[1, 'B']
```

```
print("Valor acessado com Pandas:", value)
```


Manipulação de Dados (numpy)

```
import numpy as np
```

```
arr = np.array([1, 2, 3, 4, 5])
```

```
# Álgebra linear com NumPy
```

```
matriz = np.array([[1, 2], [3, 4]])
```

```
vetor = np.array([5, 6])
```

```
# Produto escalar
```

```
produto_escalar = np.dot(matriz, vetor)
```

```
# Estatísticas com NumPy
```

```
media = np.mean(arr)
```

```
desvio_padrao = np.std(arr)
```

```
print("Produto escalar com NumPy:", produto_escalar)
```

```
print("Média:", media)
```

```
print("Desvio padrão:", desvio_padrao)
```

NumPy oferece funções e métodos para operações numéricas eficientes em arrays, como álgebra linear, estatísticas e operações matemáticas.

Manipulação de Dados (pandas)

```
import pandas as pd
```

```
data = {'A': [1, 2, 3], 'B': [4, 5, 6]}  
df = pd.DataFrame(data)
```

```
df['C'] = df['A'] + df['B']
```

```
media_coluna_A = df['A'].mean()  
soma_coluna_B = df['B'].sum()
```

```
print("DataFrame Pandas:")  
print(df)  
print("Média da coluna 'A':", media_coluna_A)  
print("Soma da coluna 'B':", soma_coluna_B)
```

Pandas fornece uma ampla gama de funcionalidades para limpeza, transformação e análise de dados tabulares, incluindo operações de agrupamento, junção, pivoteamento, etc.

Integração com outras bibliotecas

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
```

Gerando dados aleatórios com NumPy

```
np.random.seed(0)
data = np.random.normal(loc=0, scale=1, size=1000)
```

Calculando estatísticas com SciPy

```
mean = np.mean(data)
std_dev = np.std(data)
```

Visualizando os dados com Matplotlib

```
plt.hist(data, bins=30, density=True, alpha=0.5, color='blue')
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = stats.norm.pdf(x, mean, std_dev)
plt.plot(x, p, 'k', linewidth=2)
plt.title('Distribuição Normal')
plt.show()
```

NumPy é frequentemente usado em conjunto com outras bibliotecas científicas, como SciPy e Matplotlib, para análise e visualização de dados.

Integração com outras bibliotecas

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target'] = iris.target

# Dividindo os dados em conjunto de treinamento e teste com scikit-learn
X_train, X_test, y_train, y_test = train_test_split(df.drop('target', axis=1),
df['target'], test_size=0.3, random_state=42)

# Treinando um modelo de regressão logística com scikit-learn
model = LogisticRegression()
model.fit(X_train, y_train)

# Fazendo previsões e avaliando a precisão do modelo
predictions = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)

print("Acurácia do modelo:", accuracy)
```

Pandas é frequentemente usado em conjunto com bibliotecas de análise de dados, como scikit-learn para aprendizado de máquina e statsmodels para modelagem estatística.

Desempenho

Tanto NumPy quanto Pandas foram desenvolvidos principalmente em Python, mas incluem extensões de desempenho escritas em linguagens de baixo nível para garantir eficiência.

NumPy: A grande parte da implementação de NumPy é escrita em C e Fortran, proporcionando desempenho otimizado para operações numéricas.

Pandas: escrito em Python, mas utiliza a biblioteca NumPy internamente para operações numéricas eficientes.

Estrutura unidimensional

Estrutura unidimensional

Pandas

```
import pandas as pd
```

```
s = pd.Series([10, 20, 30, 40, 50])
```

```
print("Pandas Series:")  
print(s)
```

Estrutura unidimensional

Numpy

```
import numpy as np
```

```
arr = np.array([10, 20, 30, 40, 50])
```

```
print("NumPy ndarray:")  
print(arr)
```

Estrutura bidimensional

Estrutura unidimensional

Pandas

```
import pandas as pd
```

```
data = {'A': [1, 2, 3], 'B': [4, 5, 6], 'C': [7, 8, 9]}  
df = pd.DataFrame(data)
```

```
print("Pandas DataFrame:")  
print(df)
```

Estrutura unidimensional

Numpy

```
import numpy as np
```

```
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

```
print("NumPy ndarray:")  
print(arr)
```

Estrutura tridimensional

Não existe uma Estrutura tridimensional no Pandas

Estrutura tridimensional

Numpy

```
import numpy as np
```

```
arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
```

```
print("NumPy ndarray (tridimensional):")
```

```
print(arr)
```


Exercícios Pandas Series

Estrutura Unidimensional

1. Exercício de Criação: Crie uma Pandas Series contendo os números de 1 a 10.
2. Exercício de Acesso: Acesse o terceiro elemento de uma Pandas Series e imprima seu valor.
3. Exercício de Soma: Some duas Pandas Series contendo os números de 1 a 5 e os números de 6 a 10, respectivamente.
4. Exercício de Filtragem: Crie uma Pandas Series contendo os números de 1 a 10 e exiba apenas os valores maiores que 5.
5. Exercício de Estatísticas: Calcule a média e o desvio padrão de uma Pandas Series contendo os números de 1 a 10.
6. Exercício de Concatenação: Concatene duas Pandas Series contendo os números de 1 a 5 e 6 a 10, respectivamente, em uma única Series.
7. Exercício de Subtração: Subtraia uma Pandas Series contendo os números de 1 a 10 por uma outra Series contendo os números de 10 a 1.
8. Exercício de Ordenação: Ordene uma Pandas Series contendo os números de 1 a 10 em ordem decrescente.
9. Exercício de Verificação de Valores Únicos: Verifique se há valores duplicados em uma Pandas Series contendo os números de 1 a 10.
10. Exercício de Slicing: Utilize slicing para selecionar os elementos de uma Pandas Series do terceiro ao sétimo elemento.

Exercícios Pandas Series

Resolução

1. Exercício de Criação: Crie uma Pandas Series contendo os números de 1 a 10.

```
import pandas as pd
```

```
s = pd.Series(range(1, 11))
```

```
print(s)
```

2. Exercício de Acesso: Acesse o terceiro elemento de uma Pandas Series e imprima seu valor.

```
print("Terceiro elemento:", s[2])
```

3. Exercício de Soma: Some duas Pandas Series contendo os números de 1 a 5 e os números de 6 a 10, respectivamente.

```
s1 = pd.Series(range(1, 6))
```

```
s2 = pd.Series(range(6, 11))
```

```
soma = s1 + s2
```

```
print("Soma:")
```

```
print(soma)
```

4. Exercício de Filtragem: Crie uma Pandas Series contendo os números de 1 a 10 e exiba apenas os valores maiores que 5.

```
print("Valores maiores que 5:")
```

```
print(s[s > 5])
```

Exercícios Pandas Series

Resolução

5. Exercício de Estatísticas: Calcule a média e o desvio padrão de uma Pandas Series contendo os números de 1 a 10.

```
print("Média:", s.mean())  
  
print("Desvio padrão:", s.std())
```

6. Exercício de Concatenação: Concatene duas Pandas Series contendo os números de 1 a 5 e 6 a 10, respectivamente, em uma única Series.

```
s_concat = pd.concat([s1, s2])  
  
print("Concatenação:")  
  
print(s_concat)
```

7. Exercício de Subtração: Subtraia uma Pandas Series contendo os números de 1 a 10 por uma outra Series contendo os números de 10 a 1.

```
import pandas as pd  
  
s1 = pd.Series(range(1, 11))  
  
s2 = pd.Series(range(10, 0, -1))  
  
resultado = s1 - s2  
  
print("Resultado da Subtração:", resultado)
```

8. Exercício de Ordenação: Ordene uma Pandas Series contendo os números de 1 a 10 em ordem decrescente.

```
s_ordenada = s2.sort_values(ascending=True)  
  
print("Série Ordenada:")  
  
print(s_ordenada)
```


Exercícios Pandas Series

Resolução

9. Exercício de Verificação de Valores Únicos: Verifique se há valores duplicados em uma Pandas Series contendo os números de 1 a 10.

```
s = pd.Series([10, 20, 20, 30, 40, 50])  
print("Valores Duplicados:", s.duplicated().any())
```

10. Exercício de Slicing: Utilize slicing para selecionar os elementos de uma Pandas Series do terceiro ao sétimo elemento.

```
print("Slicing do terceiro ao sétimo elemento:")  
print(s[2:7])
```


Exercícios Pandas Series

Resolução

9. Exercício de Verificação de Valores Únicos: Verifique se há valores duplicados em uma Pandas Series contendo os números de 1 a 10.

```
s = pd.Series([10, 20, 20, 30, 40, 50])  
print("Valores Duplicados:", s.duplicated().any())
```

10. Exercício de Slicing: Utilize slicing para selecionar os elementos de uma Pandas Series do terceiro ao sétimo elemento.

```
print("Slicing do terceiro ao sétimo elemento:")  
print(s[2:7])
```

Exercícios Desafio

Pandas Series

Exercício Prático: Análise de Dados de Vendas de uma Loja de Eletrônicos

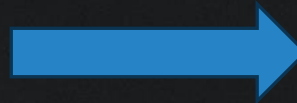
Descrição do Problema:

Uma loja de eletrônicos deseja analisar seus dados de vendas para entender melhor o desempenho de seus produtos ao longo do tempo. Eles possuem um arquivo CSV com informações sobre as vendas de vários produtos nos últimos meses. O objetivo é usar o Pandas Series para realizar algumas análises básicas e responder a algumas perguntas sobre os dados de vendas.

Exercícios Desafio

Pandas Series

Produto,Mês,Vendas
Celular,Jan-2023,150
Celular,Fev-2023,180
Celular,Mar-2023,200
Tablet,Jan-2023,100
Tablet,Fev-2023,120
Tablet,Mar-2023,130
Notebook,Jan-2023,80
Notebook,Fev-2023,90
Notebook,Mar-2023,100



**Resolver todos exercícios
utilizando pandas series**

Exercícios Desafio

Pandas Series

Tarefas

1. Carregar os dados do arquivo CSV em um Pandas Series, onde cada entrada representa o número de vendas de um produto em um determinado mês.
2. Identificar o produto mais vendido e o menos vendido ao longo de todo o período.
3. Calcular a média, mediana, máximo e mínimo de vendas mensais.
4. Visualizar um gráfico de linha mostrando a tendência de vendas ao longo do tempo.
5. Calcular a variação percentual entre as vendas de um mês para o próximo e identificar os meses com maior e menor variação percentual.

Exercícios Numpy

Estrutura Unidimensional

1. Exercício de Criação: Crie um NumPy array contendo os números de 1 a 10.
2. Exercício de Acesso: Acesse o quarto elemento de um NumPy array e imprima seu valor.
3. Exercício de Soma: Some dois NumPy arrays contendo os números de 1 a 5 e os números de 6 a 10, respectivamente.
4. Exercício de Filtragem: Crie um NumPy array contendo os números de 1 a 10 e exiba apenas os valores maiores que 5.
5. Exercício de Estatísticas: Calcule a média e o desvio padrão de um NumPy array contendo os números de 1 a 10.
6. Exercício de Concatenação: Concatene dois NumPy arrays contendo os números de 1 a 5 e 6 a 10, respectivamente, em um único array.
7. Exercício de Subtração: Subtraia um NumPy array contendo os números de 1 a 10 por outro array contendo os números de 10 a 1.
8. Exercício de Ordenação: Ordene um NumPy array contendo os números de 1 a 10 em ordem decrescente.
9. Exercício de Verificação de Valores Únicos: Verifique se há valores duplicados em um NumPy array contendo os números de 1 a 10.
10. Exercício de Slicing: Utilize slicing para selecionar os elementos de um NumPy array do terceiro ao sétimo elemento.

Exercícios Numpy

Resolução

5. Exercício de Estatísticas: Calcule a média e o desvio padrão de um NumPy array contendo os números de 1 a 10.

```
print("Média:", arr.mean())
```

```
print("Desvio padrão:", arr.std())
```

6. Exercício de Concatenação: Concatene dois NumPy arrays contendo os números de 1 a 5 e 6 a 10, respectivamente, em um único array.

```
arr_concat = np.concatenate([arr1, arr2])
```

```
print("Concatenação:")
```

```
print(arr_concat)
```

7. Exercício de Subtração: Subtraia um NumPy array contendo os números de 1 a 10 por outro array contendo os números de 10 a 1.

```
import numpy as np
```

```
arr1 = np.arange(1, 11)
```

```
arr2 = np.arange(10, 0, -1)
```

```
resultado = arr1 - arr2
```

```
print("Resultado da Subtração:", resultado)
```

8. Exercício de Ordenação: Ordene um NumPy array contendo os números de 1 a 10 em ordem decrescente.

```
arr1 = np.arange(1, 11)
```

```
arr_ordenado = np.sort(arr1)[::-1]
```

```
print(arr_ordenado)
```

Exercícios Numpy

Resolução

9. Exercício de Verificação de Valores Únicos: Verifique se há valores duplicados em um NumPy array contendo os números de 1 a 10.

```
arr = np.array([0, 4, 2, 6, 5, 9, 6])
```

```
print("Valores Duplicados:", np.unique(arr).size != arr.size)
```

10. Exercício de Slicing: Utilize slicing para selecionar os elementos de um NumPy array do terceiro ao sétimo elemento.

```
print("Slicing do terceiro ao sétimo elemento:")
```

```
print(arr[2:7])
```


Exercício Desafio com Numpy

Desafio: Análise de Estoque e Vendas em um Supermercado

Descrição do Problema: Um supermercado deseja realizar uma análise do estoque e das vendas de seus produtos para otimizar o gerenciamento de estoque e melhorar o desempenho de vendas. Eles possuem dados armazenados em arquivos CSV, incluindo informações sobre o estoque inicial, vendas mensais e reposições de estoque de vários produtos. O desafio é utilizar o NumPy para realizar análises avançadas e responder a algumas perguntas específicas sobre o estoque e as vendas dos produtos.

Exercício Desafio com Numpy

Estoque inicial

Produto,Estoque_Inicial

Arroz,500

Feijão,400

Óleo,300

Macarrão,600

Leite,700

Vendas Mensais

Produto,Jan-2023,Fev-2023,Mar-2023

Arroz,100,120,90

Feijão,80,100,110

Óleo,70,80,85

Macarrão,110,130,120

Leite,150,170,160

Vendas Mensais

Produto,Jan-2023,Fev-2023,Mar-2023

Arroz,50,60,70

Feijão,40,50,55

Óleo,30,35,40

Macarrão,60,70,80

Leite,80,90,100

Exercício Desafio com Numpy

Tarefas

- Carregar os dados do estoque inicial, vendas mensais e reposições de estoque de cada produto em arrays NumPy.
- Calcular o estoque final de cada produto para cada mês, levando em consideração as vendas e reposições de estoque.
- Identificar os produtos que tiveram o maior e menor número de vendas ao longo do período.
- Calcular a média, mediana, máximo e mínimo de vendas mensais para cada produto.
- Identificar os meses com maior e menor variação de estoque em relação ao mês anterior.

Estrutura bidimensional

Representação de dados tabulares

Armazenar dados tabulares, como planilhas, bancos de dados, registros de transações, etc.

Por exemplo, informações de alunos em uma escola, onde cada linha representa um aluno e cada coluna representa atributos como nome, idade, notas etc.

Estrutura bidimensional

Processamentos de Imagens

Em processamento de imagens, uma matriz bidimensional é frequentemente usada para representar pixels.

Cada elemento da matriz bidimensional representa a intensidade do pixel em uma determinada posição na imagem.

Estrutura bidimensional

Análise de dados

Em análise de dados, você pode usar uma estrutura bidimensional para representar dados que são organizados em linhas e colunas.

Por exemplo, você pode ter dados de vendas onde as linhas representam diferentes produtos e as colunas representam diferentes meses.

Estrutura bidimensional

Modelagem de Jogos

Em jogos de tabuleiro, uma matriz bidimensional pode ser usada para representar o tabuleiro do jogo.

Cada elemento da matriz representa uma célula no tabuleiro e pode conter informações como peças, obstáculos, etc.

Estrutura bidimensional

Algoritmos de Matriz

Muitos algoritmos e operações matemáticas envolvendo matrizes são bidimensionais.

Por exemplo, operações de multiplicação de matriz, decomposição de matriz, etc.

Estrutura bidimensional

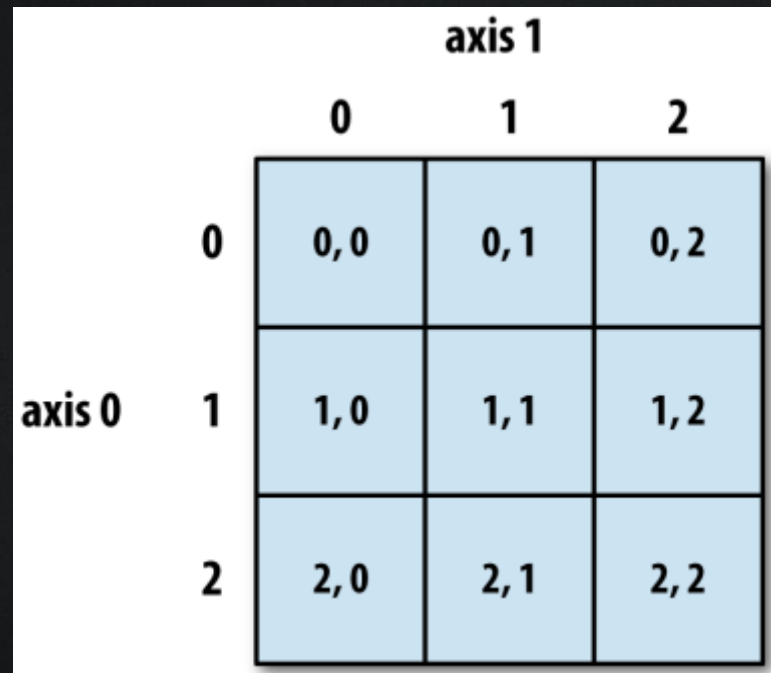
Visualização de Dados

Em visualização de dados, você pode usar uma estrutura bidimensional para armazenar dados que serão exibidos em um gráfico ou mapa.

Cada linha pode representar um ponto de dados e cada coluna pode representar uma dimensão ou atributo.

Estrutura bidimensional

A figura abaixo mostra uma ilustração da indexação em uma matriz bidimensional, na qual eixo 0 é chamado de "linhas" da matriz e no eixo 1 "colunas".



		axis 1		
		0	1	2
axis 0	0	0,0	0,1	0,2
	1	1,0	1,1	1,2
	2	2,0	2,1	2,2

*“O que sabemos é uma gota; o
que ignoramos é um oceano.”
(Issac Newton)*

Referências

- **ASCENCIO, A. F. G, CAMPOS, E. A. V. Fundamentos da Programação de Computadores: algoritmos, Pascal, C/C++ e Java, 2ª Edição, São Paulo: Pearson 2007.**
- **FURGERI, Sérgio. Introdução à Programação em Python. São Paulo: Editora Senac, 2021.**
- **MENEZES, Nilo. Introdução à Programação em Python. São Paulo: Novatec, 2019**
- **SALVETTI, Dirceu Douglas; BARBOSA, Lisbete Madson. Algoritmos. São Paulo: Pearson, 2004.**