# E-BOOK
## DO ESTUDANTE

# Dataframes, Series e Arrays

Dentro da caixa!

RESILIA

# SUMÁRIO

# Dataframes, Series e Arrays

## Dentro da caixa!


Dataframes, Series e Arrays são estruturas de dados muito importantes em Python para manipulação, análise e visualização de informações de maneira eficiente. Os **Arrays** permitem armazenar valores do mesmo tipo em uma ou várias dimensões, possibilitando a realização de cálculos matemáticos e a manipulação de dados.

As **Series** e **Dataframes** são estruturas bidimensionais que permitem armazenar informações em tabelas e manipulá-las de maneira eficiente, possibilitando a análises de dados em larga escala. A biblioteca **Pandas** é amplamente utilizada em Python para criar e manipular essas estruturas. Ela oferece estruturas de dados flexíveis para trabalhar com dados, como o **DataFrame**, que permite armazenar dados em uma estrutura tabular semelhante a uma planilha de Excel.

Neste material, serão apresentados alguns fundamentos essenciais sobre Dataframes, Series e Arrays, e também como construir essas estruturas com a linguagem Python.

## CONTEXTUALIZANDO

Python é uma das linguagens de programação mais utilizadas atualmente, e ela oferece diversas **estruturas** de dados que permitem a manipulação e a análise de informações de maneira eficiente. Entre as principais estruturas estão Dataframes, Series e Arrays.

Os **Dataframes** são estruturas de dados bidimensionais que organizam informações em tabelas. Eles são muito utilizados em análise de dados e ciência de dados.

As **Series** são estruturas de dados unidimensionais que representam uma sequência de valores associados a um rótulo. Elas podem ser criadas a partir de listas, dicionários e outras estruturas de dados, e oferecem diversas funções para manipulação e análise de dados.

Os **Arrays** são estruturas de dados unidimensionais ou multidimensionais que armazenam valores do mesmo tipo. Em Python, é possível criar Arrays utilizando a biblioteca NumPy, que oferece diversas funções para manipulação de Arrays e cálculos matemáticos.

Ao trabalhar com grandes conjuntos de dados, a utilização dessas estruturas de dados torna-se fundamental para facilitar sua análise e manipulação. Por exemplo, ao importar um grande arquivo de dados, os usuários podem armazenar os dados em um **dataframe** para visualizá-los, manipulá-los e executar análises estatísticas e de **machine learning**.

## ARRAYS

Arrays em Python são uma estrutura de dados que permite armazenar e manipular coleções de elementos do mesmo tipo. Elas são úteis para trabalhar com matrizes, vetores e sequências numéricas, entre outros.

Em Python, existem várias bibliotecas que oferecem suporte a Arrays, como NumPy e Array. A biblioteca NumPy é amplamente utilizada para trabalhar com Arrays multidimensionais e oferece muitas funcionalidades para a manipulação de dados numéricos.

Para criar um Array em **NumPy**, você pode usar a função array(), passando uma lista ou tupla de elementos como argumento:

```
import numpy as np

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

Nesse exemplo, a variável a é um Array NumPy com os elementos 1, 2, 3, 4 e 5. Você pode acessar os elementos do Array usando a notação de colchetes, como em a[0], para acessar o primeiro elemento.

NumPy também oferece muitas funções para criar Arrays preenchidos com valores padrão, como zeros(), ones(), full() e arange(). Por exemplo, para criar um Array de zeros com 5 elementos, você pode usar o seguinte código:

```
b = np.zeros(5)
print(b)
```

Para criar um Array com valores sequenciais, você pode usar a função arange(), como em np.arange(0, 10, 2) para criar um array com valores de 0 a 10 (exclusivo) com um intervalo de 2.

Além disso, NumPy oferece muitas funções para manipular e processar Arrays, como sum(), mean(), std(), max() e min(). Essas funções permitem realizar **operações matemáticas** em Arrays e obter **estatísticas** sobre seus valores.

Por exemplo, para calcular a média dos valores de um Array, você pode usar a função mean(), como em np.mean(a). Para obter o valor máximo do array, você pode usar a função max(), como em np.max(a).

NumPy também oferece muitas funcionalidades para operações de **álgebra linear**, como multiplicação de matrizes, inversão de matrizes e decomposição de valores singulares.

Como vimos, essas são apenas algumas das funcionalidades oferecidas pelo NumPy para trabalhar com Arrays em Python. A biblioteca é amplamente utilizada em ciência de dados, processamento de imagens, aprendizado de máquina e outras áreas que exigem manipulação de dados numéricos. Vamos dar uma olhada em outra estrutura fundamental para nossos estudos: Series!

## SERIES

Series em Python é uma estrutura de dados que permite armazenar e manipular coleções de dados unidimensionais e rotulados. Ela é útil para trabalhar com dados de séries temporais, dados financeiros, dados de ações, entre outros.

Em Python, a biblioteca Pandas é amplamente utilizada para trabalhar com Series. Para criar uma **Series** em Pandas, você pode usar a função Series(), passando uma lista de elementos como argumento:

```
import pandas as pd

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

Nesse exemplo, a variável s é uma Series com os elementos 1, 2, 3, 4 e 5. A diferença entre uma lista e uma Series é que esta tem um índice associado a cada elemento, que pode ser usado para acessar os elementos individualmente.

Você pode acessar os elementos da Series usando a notação de colchetes, como em s[0], para acessar o primeiro elemento. Também é possível acessar vários elementos de uma vez, como em s[0:3] para acessar os elementos de índice 0 a 2.

Pandas também oferece muitas funcionalidades para criar e manipular Series de forma eficiente. Por exemplo, você pode criar uma Series com valores sequenciais usando a função range() e a função Series():

```
s = pd.Series(range(1, 6))
print(s)
```

Para criar uma Series com **rótulos personalizados**, você pode usar a função Series() e passar uma lista de rótulos como argumento:

```
s = pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e'])
print(s)
```

Nesse exemplo, a Series tem rótulos personalizados para cada elemento: 'a' para 1, 'b' para 2, 'c' para 3, 'd' para 4 e 'e' para 5. Você pode acessar os elementos usando os rótulos, como em s['a'] para acessar o elemento com rótulo 'a'.

Pandas também oferece muitas funções para manipular e processar Series, como sum(), mean(), std(), max() e min(). Essas funções permitem realizar operações matemáticas em Series e obter **estatísticas** sobre seus valores.

Por exemplo: para calcular a média dos valores de uma Series, você pode usar a função mean(), como em s.mean(). Para obter o valor máximo da Series, você pode usar a função max(), como em s.max().

Pandas também oferece muitas funcionalidades para operações de **agregação**, como agrupamento de dados, agregação de dados e resumos estatísticos.

Como vimos, essas são apenas algumas das funcionalidades oferecidas pelo Pandas para trabalhar com Series em Python. Vamos conferir agora os dataframes, estrutura semelhante, mas em duas dimensões.

## DATAFRAMES

Dataframes em Python são uma estrutura de dados bidimensional organizada em linhas e colunas, semelhante a uma tabela de banco de dados ou a uma planilha do Excel. É uma das estruturas de dados mais utilizadas na análise e na manipulação de dados em Python.

Existem várias bibliotecas em Python que oferecem suporte à criação e à manipulação de dataframes, mas a mais comum é a biblioteca Pandas.

Para criar um dataframe em Pandas, você pode usar a função DataFrame(), que permite criar um dataframe a partir de **diferentes tipos de dados**, como listas, dicionários e arquivos CSV.

Por exemplo, para criar um dataframe a partir de uma lista de dicionários, você pode usar o seguinte código:

```
import pandas as pd

dados = [
    {'nome': 'João', 'idade': 30, 'cidade': 'São Paulo'},
    {'nome': 'Maria', 'idade': 25, 'cidade': 'Rio de Janeiro'},
    {'nome': 'Pedro', 'idade': 40, 'cidade': 'Belo Horizonte'},
]

df = pd.DataFrame(dados)
```

Nesse exemplo, a lista dados contém três dicionários, cada um representando uma linha do dataframe. Cada chave do dicionário representa uma coluna do dataframe.

Você pode acessar as colunas do dataframe usando a notação de colchetes, como em df['nome'] ou df['idade']. Você também pode usar a função head() para ver as primeiras linhas do dataframe:

```
print(df['nome'])

print(df.head())
```

Além disso, Pandas oferece muitas funções úteis para manipular dataframes, como groupby(), merge() e pivot_table(). Essas funções permitem agrupar dados, combinar dataframes e criar tabelas dinâmicas, respectivamente.

Por exemplo, você pode agrupar os dados por cidade e calcular a média da idade para cada cidade, usando o seguinte código:

```
df_por_cidade = df.groupby('cidade').mean()

print(df_por_cidade)
```

Como vimos, essas são apenas algumas das funcionalidades oferecidas pelo Pandas para trabalhar com dataframes em Python. A biblioteca também oferece suporte a operações de seleção, filtragem, ordenação e agregação de dados, bem como leitura e gravação de arquivos CSV, Excel e outros formatos.

### Para refletir

- Qual é a principal diferença entre Series e Arrays em Python, e em que situações cada uma dessas estruturas de dados seria mais útil em um projeto prático?
- Quais são algumas das principais bibliotecas em Python usadas para trabalhar com Dataframes e Series, e quais são suas principais funcionalidades? Não vale citar Pandas, hein?!
- Como você pode usar Dataframes e Series em Python para realizar análise exploratória de dados, limpeza e manipulação de dados, e como essas estruturas de dados podem ser usadas em conjunto com outras ferramentas em Python, como  NumPy, por exemplo?

### Atividade: Refletindo sobre dados

Pesquise sobre como as estruturas de dados DataFrame, Series e Arrays são usadas em projetos de ciência de dados e responda às seguintes questões:

- Como essas estruturas de dados podem facilitar a análise de grandes conjuntos de dados e ser usadas com outras ferramentas em Python, como NumPy, por exemplo?
- Quais são os principais desafios encontrados ao trabalhar com grandes conjuntos de dados e como o uso dessas estruturas de dados pode ajudar a superar esses desafios?

### Para ir além

- Vimos que os Arrays são estruturas capazes de agrupar várias informações. Nos sites abaixo, você encontrará mais formas de manipular Arrays com Python.
  <https://celsokitamura.com.br/o-que-e-array-em-python/>
  <https://www.w3schools.com/python/python_arrays.asp>

- Vimos, neste e-book, como funcionam as estruturas de Series. No site abaixo, você descobrirá como usar o Pandas para construção dessas estruturas.
<https://www.codingame.com/playgrounds/52723/programacao-python-parte-3---prof--marco-vaz/pacote-pandas-series>

- Vimos, neste e-book, como funcionam as estruturas de dados Dataframe. No site abaixo, você aprenderá um pouco mais sobre como manipular dataframes com Python.
<https://pythonacademy.com.br/blog/dataframes-do-pandas>

## RESUMO DE DATAFRAMES, SERIES E ARRAYS

**ARRAYS**
**Arrays em Python são uma estrutura de dados que permite armazenar e manipular coleções de elementos do mesmo tipo.**

**Arrays em Python**

```
import numpy as np

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

```
b = np.zeros(5)
print(b)
```

**SERIES**
**Series em Python são uma estrutura de dados que permitem armazenar e manipular coleções de dados unidimensionais e rotulados.**

**Series em Python**

```
import pandas as pd

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

```
s = pd.Series(range(1, 6))
print(s)
```

```
s = pd.Series([1, 2, 3, 4, 5], index=['a',
'b', 'c', 'd', 'e'])
print(s)
```

**DATAFRAMES**
**Dataframes em Python são uma estrutura de dados bidimensional organizada em linhas e colunas.**

**Dataframes em Python**

```
import pandas as pd

dados = [
    {'nome': 'João', 'idade': 30, 'cidade':
'São Paulo'},
    {'nome': 'Maria', 'idade': 25, 'cidade':
'Rio de Janeiro'},
    {'nome': 'Pedro', 'idade': 40, 'cidade':
'Belo Horizonte'},
]

df = pd.DataFrame(dados)
```

```
print(df['nome'])
print(df.head())
```

**Até a próxima e**
**#confianoprocesso**

RESILIA