

# **Introdução à Análise de Dados**

## Cap.5 - Introdução ao Pandas



Prof. MSc. Renzo P. Mesquita

# Objetivos

- Conhecer recursos essenciais de um dos pacotes mais poderosos para Análise de Dados do mercado, o Pandas;
- Explorar suas estruturas fundamentais para lidar com grandes quantidades de dados, como Series e DataFrames;
- Compreender como carregar, manipular, enriquecer e salvar datasets em Python por meio dele.



# Capítulo 5

## Introdução ao Pandas

*5.1. Introdução;*

*5.2. Pandas Series;*

*5.3. Pandas DataFrame;*

*5.4. Entrada e Saída de Dados.*





## 5.1. Introdução

*O Pandas é um dos pacotes open source mais populares em Python e que fornece estruturas de dados rápidas e flexíveis para lidar com dados estruturados e semiestruturados.*

Alguns destaques desta biblioteca:

- Criada sobre a biblioteca NumPy;
- Trabalha com duas estruturas de dados (Series e DataFrames) que são altamente aplicáveis para tratamento de dados voltados às áreas como finanças, estatística, ciência social e engenharia;
- Fornece mecanismos de fácil ajuste de dados faltantes (nulos);
- Novas colunas de dados podem ser facilmente inseridas e excluídas;
- Mecanismos rápidos para ordenação e alinhamento de dados;
- Recursos flexíveis para agrupamento de dados;
- Recursos rápidos para fatiamento, indexação, junção e subseparação de grandes DataSets;
- Lembra o Excel, só que dentro do Python.

*e muito mais..*



## 5.1. Introdução

*Apesar de ser uma biblioteca muito popular para Análise de Dados em Python, é necessário inicialmente instalá-la.*

Como já vimos anteriormente, a instalação por meio do PyCharm é bem simples.

- Basta irmos em File -> Settings;
- No nome do seu projeto, vá em Project Interpreter -> clique em + (Install);
- Procure por NumPy e aperte o botão Install Package;

Uma vez instalada, basta importá-la no projeto:

*Ex:* `import pandas as pd`

Obs: Abreviamos pandas para simplesmente "pd" para pouparmos tempo e também seguirmos um padrão comumente usado por outros desenvolvedores que fazem uso desta biblioteca.



+





## 5.2. Pandas Series

*O Pandas Series é um Array 1-D que trabalha com rótulos (labels) e é capaz de armazenar dados de qualquer tipo (inteiros, strings, float, objetos, etc.).*


Ao se criar uma nova Series, duas informações primordiais precisam ser passadas:

- 1) uma coleção de dados;
- 2) uma coleção de labels;

*Ex:*

```
# Criando os labels
labels = ['a', 'b', 'c']
# Criando os dados
dados = [10, 20, 30]
# Criando a Series
s = pd.Series(data=dados, index=labels)
print(s)
```

Series



a	10
b	20
c	30



## 5.2. Pandas Series

*Observe que como a Series trabalha com a ideia de labels, ela lembra muito um Dicionário.*

Logo, para acessar um valor específico de uma Series, basta utilizar do label como índice:

*Ex:* `print(s['a'])`  10

*Outros detalhes importantes das Series:*

- Tanto os labels como os dados podem ser de qualquer tipo;
- Usamos uma lista tradicional para preencher os dados da Series, mas também poderíamos usar um NumPy Array para fazer isso;
- Como ela se parece muito com um dicionário, podemos criar uma Series simplesmente passando um dicionário como parâmetro;
- Se não passarmos uma coleção de labels, o Pandas criará labels padrões e numéricas para os dados;

*Possui características de operação muito parecidas com o NumPy Array, mas traz como outro diferencial a facilidade de operações baseadas em labels. Vamos ver como isso funciona?*





## 5.2. Pandas Series

*Diferente do NumPy Array que faz operações índice à índice sequenciais entre Arrays, as Series fazem operações se baseando em labels.*

*Ex:*

```
s1 = pd.Series({'a': 10, 'b': 20, 'c': 30})  
s2 = pd.Series({'a': 10, 'c': 50, 'd': 80})  
print(s1 + s2)
```

a 20.0  
b NaN  
c 80.0  
d NaN  
dtype: float64

Veja que apesar de nos oferecer este recurso poderoso, as vezes não queremos perder valores de labels em que as operações não puderam ser aplicadas. Para isso, podemos usar de funções pré-prontas das Series:

```
s1 = pd.Series({'a': 10, 'b': 20, 'c': 30})  
s2 = pd.Series({'a': 10, 'c': 50, 'd': 80})  
print(s1.add(s2, fill_value=0))
```

a 20.0  
b 20.0  
c 80.0  
d 80.0  
dtype: float64

- Observe que nesta abordagem, o parâmetro `fill_value` nos ajuda a preencher valores NaN de cada Series envolvida com um valor default;
- Outras funções populares além do `add()`: `sub()`, `mul()`, `div()`, `mod()`, `pow()`, etc..



## 5.2. Pandas Series

*Da mesma forma que acontece nos NumPy Arrays, nas Series muitas vezes não é necessário varrer seus elementos para se aplicar operações.*

*Ex:* `print(np.mean(s1))` → 20.0

Da mesma forma que vimos nos NumPy Arrays, já podemos facilmente aplicar as funções pré-prontas sobre todos os elementos como `min()`, `max()`, `exp()`, etc..

Uma estratégia também muito interessante nas Series é a possibilidade de filtrar apenas o que se deseja ver:

*Ex:* `print(s1[['a', 'c']])` →

a	10
c	30
dtype: int64	

Obs: Observe a lista de labels como parâmetro.


## 5.2. Pandas Series

*Também conseguimos aplicar o conceito de Slicing nas Series. Observe que os labels também entram no processo.*

*Ex:* `print(s1[1:])` 

b	20
c	30
dtype: int64	

Outro recurso poderoso é a capacidade de se criar condicionais para se retornar apenas o que interessa de uma Series:

*Ex:* `print(s1[s1 > s1.mean()])` 

c	30
dtype: int64	

Retorna apenas os valores que estão acima da média da Series.

`print(s1[s1/2 == 10])` 

b	20
dtype: int64	

Retorna apenas os valores que a metade é igual a 10.

## 5.3. Pandas DataFrame

*O Pandas DataFrame é um Array 2-D que também trabalha com rótulos (labels) mas possui colunas (columns) de dados de qualquer tipo.*

Ao se criar um novo DataFrame, três informações primordiais precisam ser passadas:

- 1) uma coleção 2-D de dados;
- 2) uma coleção de labels;
- 3) uma coleção de colunas;

*Ex:*

```
# Usando seed = 10 para gerar os dados resultantes
np.random.seed(10)
df = pd.DataFrame(data=np.random.randint(1, 50, [5, 4]),
                  index=['A', 'B', 'C', 'D', 'E'],
                  columns=['W', 'X', 'Y', 'Z'])
print(df)
```

Obs: Vale ressaltar que existem outras formas de se criar um DataFrame. Esta é a forma mais tradicional.

DataFrame				
	W	X	Y	Z
A	10	37	16	1
B	29	26	30	49
C	30	9	10	1
D	43	41	37	17
E	37	48	12	25





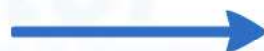
## 5.3. Pandas DataFrame

*Um DataFrame lembra muito uma Planilha de Excel ou Tabela SQL.*

Cada coluna é uma Series.

*Ex:*

```
print(df['W'])  
print(type(df['W']))
```



```
A    10  
B    29  
C    30  
D    43  
E    37  
Name: W, dtype: int32  
<class 'pandas.core.series.Series'>
```

Acessando múltiplas colunas.

*Ex:*

```
print(df[['W', 'Z']])
```



```
   W  Z  
A  10  1  
B  29 49  
C  30  1  
D  43 17  
E  37 25
```

Obs: observe que para alcançar este resultado precisamos passar uma lista de colunas.

Acessando um valor específico do DataFrame.

*Ex:*

```
print(df['W']['A'])
```



10



## 5.3. Pandas DataFrame

*Slicing de dados podem ser facilmente realizados no DataFrame por meio das funções loc() e iloc().*

*Ex:*

```
print(df.loc[['A', 'B'], ['X', 'Y', 'Z']])
```

OU

```
print(df.iloc[0:2, 1:])
```



	X	Y	Z
A	37	16	1
B	26	30	49

Veja que ambas as funções produzem o mesmo resultado no exemplo, porém:

- o loc() leva em consideração os nomes dos índices do DataFrame para realizar o slicing;
- o iloc() leva em consideração índices numéricos, seguindo o padrão NumPy;



## 5.3. Pandas DataFrame

*O Pandas permite que facilmente uma nova coluna de dados seja adicionada ao DataFrame.*

*Ex:*

```
df['N'] = np.random.randint(1, 50, 5)
```

Para exclusão seja de uma linha ou coluna no DataFrame, podemos usar a função `drop()`.

*Ex:*

```
df = df.drop('W', axis=1)
```

Deletando uma coluna: detalhe para axis = 1;

```
df = df.drop('A', axis=0)
```

Deletando uma linha: detalhe para axis = 0;

Obs: observe a importância do df recebendo o resultado do drop para a alteração efetivamente acontecer.





## 5.4. Entrada e Saída de Dados

*São inúmeros os formatos de arquivos que o Pandas consegue ler e já tratá-los como DataFrames.*

Um formato universal e largamente popular é o .csv:

*Ex:*

Lendo o arquivo .csv de países fornecido:

```
dfPaíses = pd.read_csv('países.csv', delimiter=';')  
print(dfPaíses)
```



	Country	Region	...
0	Afghanistan	ASIA (EX. NEAR EAST)	...
1	Albania	EASTERN EUROPE	...
2	Algeria	NORTHERN AFRICA	...
3	American Samoa	OCEANIA	...
4	Andorra	WESTERN EUROPE	...
..	...	...	...

*Ex:*

Salvando um arquivo .csv com apenas os nomes dos países:

```
dfPaíses = pd.read_csv('países.csv')  
dfPaíses['Country'].to_csv('nomepaíses.csv', sep=';', header=False)
```

*Agora que já temos uma noção de como funcionam os recursos mais fundamentais do Pandas, que tal praticarmos?*

## 5.4. Entrada e Saída de Dados

### Exercícios Propostos

1. Carregue o dataset `países.csv`. Em seguida, mostre:

- a) quais os países da OCEANIA;
- b) quantos países são da OCEANIA;

Dica: use do auxílio do metodo `contains` da classe `str`;

2. Mostre a média da taxa de alfabetização (Literacy (%)) do planeta;

Dica: use do auxílio da função `mean`;

3. Encontre o nome e a região do país que possui a maior população;

Dica: use do auxílio da função `max`;

4. Busque o nome de todos os países do dataset que não possuem costa marítima (`Coastline (coast/area ratio) == 0`) e guarde-os em um novo arquivo chamado `noCoast.csv`;

Dica: após realizar os filtros, use do auxílio da função `to_csv`;

#### Dicas Gerais:

- 1) para Datasets grandes, use da função `df.head(0)` para visualizar facilmente quais são as colunas deles;
- 2) para Busca de padrões textuais no Pandas, use métodos da classe `str` da `Series`. Ex: `s.str.contains('texto')`;
- 3) busque sempre modular o problema a ser resolvido, não tente resolver tudo de uma só vez;
- 4) lembre-se que a criação de condicionais no Pandas é um dos seus destaques para resolver problemas mais facilmente.

# **FIM DO CAPÍTULO 5**

