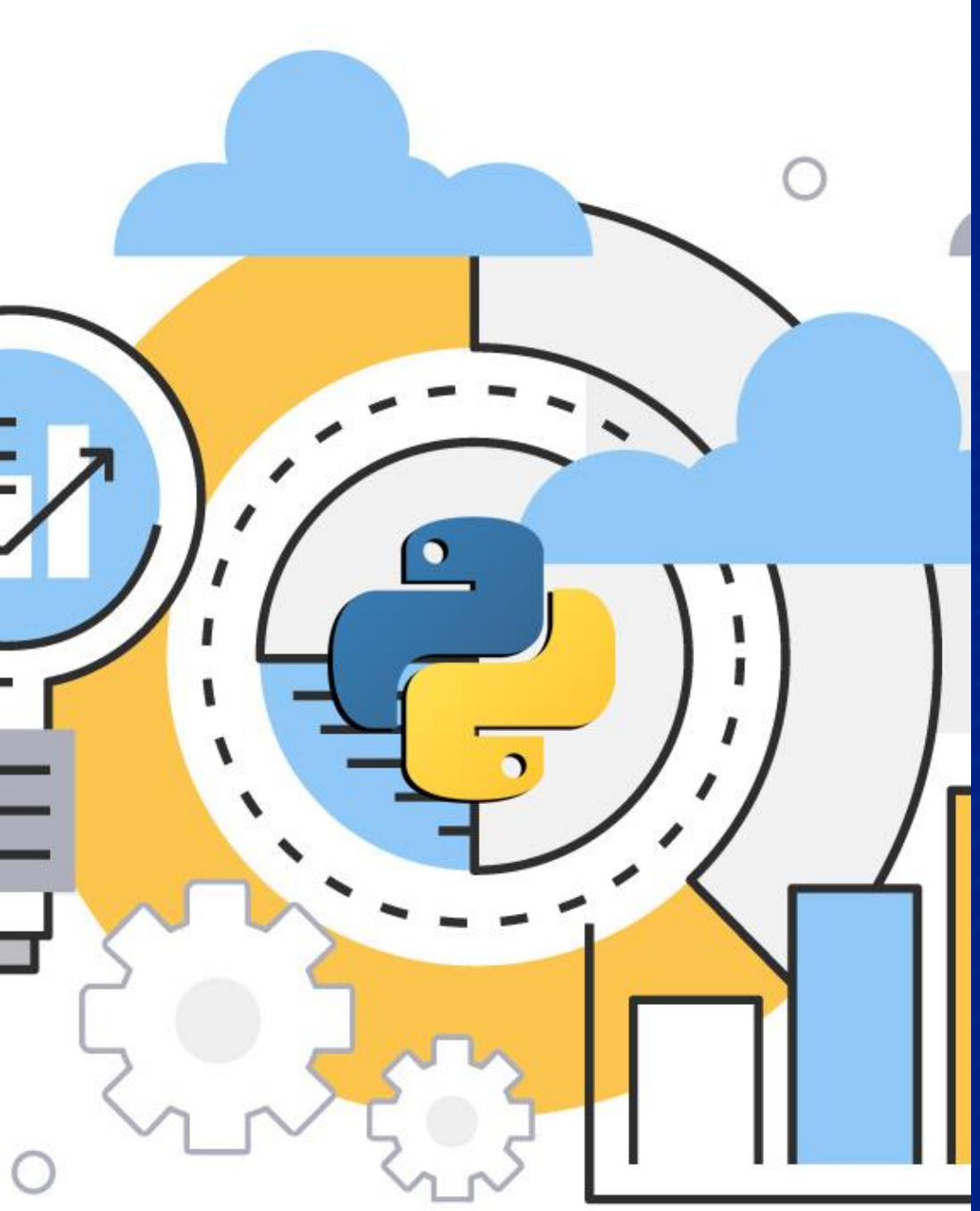
# CIÊNCIA DE DADOS COM PYTHON

## CAP.5 – INTRODUÇÃO AO PANDAS

Prof. Renzo Paranaíba Mesquita



# OBJETIVOS

- Conhecer recursos essenciais de um dos pacotes mais poderosos para Ciência 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.

Inatel

# TÓPICOS

1. Introdução
2. Pandas Series
3. Pandas DataFrame
4. Carregando Dados com Pandas
5. Salvando Dados com Pandas
6. Agrupamento de Dados
7. Operações com Métodos Customizados
8. Dados Ausentes

Inatel

# 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
- É um pacote criado sobre o NumPy. Lembra o Excel, mas dentro do Python
- Outros destaques desta biblioteca:
  - Trabalha com duas estruturas de dados: Series e DataFrames;
  - Fornece mecanismos de fácil ajuste de dados faltantes (nulos);
  - Recursos flexíveis para agrupamentos de dados;
  - Recursos rápidos para realização de Fatiamento, Indexação, e Junção de dados;
- Documentação oficial disponível em: https://pandas.pydata.org/docs/user_guide/index.html


Inatel

# 5.2. PANDAS SERIES

- O Pandas Series é um Array 1-D que trabalha com rótulos (*labels*) e é capaz de armazenar dados heterogêneos (de qualquer tipo);

- Ao se criar uma nova Series, duas informações primordiais precisam ser passadas:
  - Uma coleção de *labels*
  - Uma coleção de dados

- **EXEMPLOS:**

```
# importando o pandas
import pandas as pd

# criando uma coleção de labels
labels = ['a', 'b', 'c']

# criando uma coleção de dados
dados = [10, 20, 30]

# criando a Series
s = pd.Series(index=labels, data=dados)
```

Series

| | |
|---|---|
| a | 10 |
| b | 20 |
| c | 30 |

Inatel

# 5.2. PANDAS SERIES

- Observe que, como as Series trabalham com a ideia de *labels*, elas lembram muito um Dicionário

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

- **EXEMPLOS:**

```
print(s[‘a’]) #10
```

- **OBSERVAÇÕES:**
  - Tanto os *labels* quanto os dados podem ser de qualquer tipo;
  - NumPy Arrays e Dicionários também podem ser usados para se preencher os dados de uma Series;
  - Se não passarmos uma coleção de *labels*, o Pandas criará *labels* padrões e numéricas para os dados;
  - Possui operações muito similares às oferecidas pelo NumPy, porém, traz como diferencial a facilidade de **operações baseadas em *labels***. Vamos ver com isso funciona?

Inatel

# 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*

- **EXEMPLOS:**

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

print(s1 + s2) #realiza a operação apenas em labels com pares

print(s1.add(s2, fill_value=0)) #realiza a operação com os pares de labels e seta um valor padrão para realizar a operação com labels sem pares
```

- **OBSERVAÇÕES:**
  - Observe que na abordagem com funções pré-prontas populares, o parâmetro *fill_value* nos ajuda a preencher valores NaN com um valor *default*
  - Relembrando algumas funções populares: add(), mul(), div(), mod(), pow()…

Inatel

# 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 como min(), max(), mean(), etc.

- **EXEMPLOS:**

```
print(np.mean(s1)) #observe não só a aplicação da operação como a compatibilidade com o NumPy ;)
```

- Outra estratégia interessante nas Series é a possibilidade de filtrar apenas o que se deseja ver:

- **EXEMPLOS:**

```
# observe a lista de labels como parâmetro
print(s1[['a', 'c']])
```

```
a    10
c    30
dtype: int64
```

- Também conseguimos aplicar o conceito de Slicing nas Series utilizando índices númericos:

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

```
b    20
c    30
dtype: int64
```

Inatel

# 5.2. PANDAS SERIES

- Outro recurso poderoso é a capacidade de se criar condicionais (igual vimos no NumPy) para se retornar apenas o que interessa de uma Series:

- **EXEMPLOS:**

```
# retorna apenas os valores maiores que 25
print(s1[s1 > 25]) #30

# retorna apenas os valores que estão acima da média da Series
print(s1[s1 > s1.mean()]) #30

# retorna apenas os valores em que a metade deles é igual a 10
print(s1[s1/2 == 10]) #20
```

- **OBSERVAÇÕES:**
  - Lembre-se que a criação de condicionais é um recurso primordial para se aplicar filtros em um conjunto de valores, trazendo não só praticidade mas também desempenho.

Inatel

# 5.3. PANDAS DATAFRAME

- Os 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:
  - Uma coleção de *labels*
  - Uma coleção de columns
  - Uma coleção 2-D (matriz) de dados

- **EXEMPLOS:**

```
# importando o numpy e o pandas
import numpy as np
import pandas as pd

# plantando uma semente aleatória
np.random.seed(10)

# criando um dataframe
df = pd.DataFrame(
index=['A', 'B', 'C', 'D', 'E'],
columns=['W', 'X', 'Y', 'Z'],
data=np.random.randint(1, 50, [5, 4]))
```

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 |

row

column

- **OBSERVAÇÕES:**
  - Vale ressaltar que existem outras formas de se criar um DataFrame. Esta é a forma mais tradicional;

Inatel

# 5.3. PANDAS DATAFRAME

- Um DataFrame lembra muito uma Planilha do Excel ou Tabela SQL
- Cada coluna de um DataFrame é uma Series

- **EXEMPLOS:**

```
# acessando uma coluna
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
# observe o parâmetro – uma lista de colunas
print(df[['W', 'Z']])
```

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

```
# acessando um valor específico do DataFrame
print(df['W']['A']) #10
```

Inatel

# 5.3. PANDAS DATAFRAME

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

- **EXEMPLOS:**

```
# realizando um slicing com loc()
print(df.loc[['A', 'B'], ['X', 'Y', 'Z']])
```

| | X | Y | Z |
|---|---|---|---|
| A | 37 | 16 | 1 |
| B | 26 | 30 | 49 |

```
# realizando o mesmo slicing com iloc()
print(df.iloc[0:2, 1:])
```

- **OBSERVAÇÕES:**
  - 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

Inatel

# EXERCÍCIOS (PARTE 1)

1. Crie duas Series com os seguintes valores:
- seriesAno1: {‘Java’: 16.25, ‘C’: 16.04, ‘Python’: 9.85}
- seriesAno2: {‘C’: 16.21, ‘Python’: 12.12, ‘Java’: 11.68}

2. Os valores das Series criadas na Questão 1 representam as fatias de mercado (porcentagem) de 3 linguagens de programação populares em dois anos consecutivos. Para cada ano, apresente a porcentagem total que elas juntas representam no mercado;

3. Apresente o crescimento/declínio no mercado de cada linguagem do primeiro ano para o segundo ano;

4. Baseado nos resultados da Questão 3, mostre apenas os dados das linguagens que tiveram crescimento;

5. Se estas porcentagens de crescimento/declínio se mantivessem iguais para os próximos 2 anos, qual seria a linguagem mais popular?
*Dica: use o método nlargest(1) no final para retornar rapidamente a label e maior valor de uma Series.*

# EXERCÍCIOS (PARTE 2)

6. Utilizando do DataFrame exemplo do tópico 5.3 deste material, calcule a média dos elementos da coluna X que são menores que 30;

7. Utilizando do mesmo DataFrame, apresente a média dos elementos da linha D usando a função loc() como base e a soma dos elementos da linha E usando a função iloc() como base;

8. Faça um *Slicing* na matriz mostrando apenas as linhas A, C e E juntamente com as colunas X e Y. Em seguida, mostre qual seria a soma dos elementos de cada uma destas linhas e cada uma destas colunas.

# 5.4. CARREGANDO DADOS COM PANDAS

- São inúmeros os formatos de arquivos que o Pandas consegue ler e já trata-los como DataFrames
- Explore a coletânea de funções que iniciam com pd.read_(formato)

- **EXEMPLOS:**

```
# lendo o arquivo .csv de países fornecido:
dfPaises = pd.read_csv('paises.csv',
          delimiter=';')
print(dfPaises)

# acessando com facilidade as colunas do DataFrame
print(dfPaises.columns)

# acessando com facilidade os primeiros n registros do dataset
Print(dfPaises.head(3))

# acessando com facilidade os últimos n registros do dataset
Print(dfPaises.tail(3))
```

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

Inatel

# 5.5. SALVANDO DADOS COM PANDAS

- Ao carregar o DataSet e modificá-lo, o processo para salvá-lo com as modificações é simples
- **EXEMPLOS:**

```
#Vamos verificar a % de quanto a população de um país representa no mundo

#calculando a população mundial
total_population = np.sum(dfPaises['Population’])

#fazendo um broadcasting para calcular a % da população de cada país
seriesPercPopulation = (dfPaises['Population']/total_population)*100

# adicionando uma nova coluna no Dataset contendo esta informação
# veja como é simples adicionar uma nova coluna no DataFrame
dfPaises['% Population'] = np.round(seriesPercPopulation, 3)

# salvando um novo arquivo com esta atualização
dfPaises.to_csv('paises_v2.csv', sep=';’)
```

- **OBSERVAÇÕES:**
  - para exclusão seja de uma linha ou coluna no DataFrame, podemos usar a função drop(). Ex: dfPaises.drop(‘Region’, axis=1)
  - parâmetro axis = 1 para exclusão de colunas e axis = 0 para exclusão de linhas.

Inatel

# 5.6. AGRUPAMENTO DE DADOS

- Agrupar dados é uma tarefa importante em Ciência de Dados e o DataFrame possui um ótimo recurso para facilitar esta tarefa
- O **groupby** permite que agrupemos os dados com base em uma ou mais colunas categóricas e aplique funções de agregação como contagem, soma, média, entre outras

- **EXEMPLOS:**

```
# agrupando os países por região e mostrando as quantidades de cada região
group_region = dfPaises.groupby('Region')
print(group_region.count()['Country’])

# agrupando os países por região e a soma da população de cada região
group_region = dfPaises.groupby('Region')
print(group_region.sum()['Population’]

# agrupando os países por região e descrevendo estatísticas importantes
group_region = dfPaises.groupby('Region’)
print(group_region.describe())
```

Inatel

# 5.7. OPERAÇÕES COM MÉTODOS CUSTOMIZADOS

- Apesar do DataFrame oferecer uma coletânea diversificada de métodos pré-prontos, às vezes também é necessário se criar e aplicar métodos customizados

- **EXEMPLOS:**

```
# criando um método para aplicá-lo valor a valor em um conjunto de dados
# o método retorna o valor com 10% de desconto
def tenpercent(x):
    return x*0.9

# mostrando a taxa de mortalidade dos países
deathrate1 = dfPaises['Deathrate']

# aplicando o método por meio da função apply()
# diminuindo a taxa de mortalidade dos países em 10%
deathrate2 = dfPaises['Deathrate'].apply()

# concatenando e mostrando as duas series
pd.concat([deathrate1, deathrate2], axis=1)
```

Inatel

# 5.8. DADOS AUSENTES

- Pode acontecer de DataSets apresentarem dados faltantes (Ex: NaN ou valores não preenchidos)
- O Pandas DataFrame possui soluções interessantes para lidar com este caso

- **EXEMPLOS:**

```
# 1. remove linhas que possuam dados ausentes usando o método dropna()
novoDataFrame = dataFrame.dropna()

# 2. preenche dados ausentes com um valor padrão por meio do método fillna()
# neste exemplo estamos preenchendo os valores ausentes com o valor 0
novoDataFrame = dataFrame.fillna(0)
```

- **OBSERVAÇÕES:**
  - Tanto o método dropna() quanto fillna() possuem parâmetros que permitem preencher os valores faltantes de diferentes formas. Para mais detalhes, acesse: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.dropna.html

Inatel

# EXERCÍCIOS (PARTE 3)

1. Carregue o Dataset paises.csv e em seguida mostre:
   a. Quais são os países da OCEANIA;
   b. Quantos países são da OCEANIA;

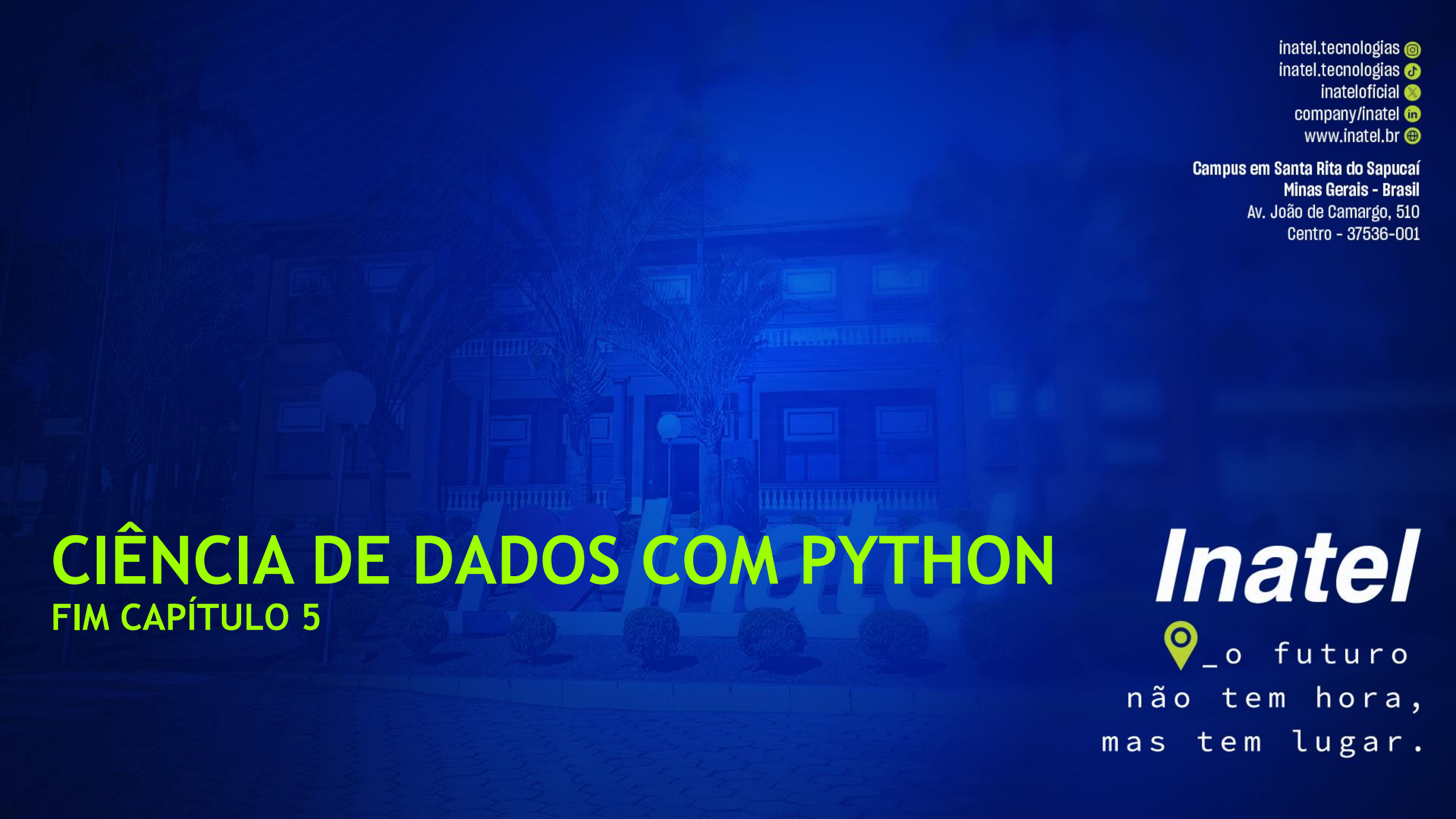
*Dica: para busca de padrões textuais no Pandas, use métodos da subclasse str da Series. Ex: series.str.contains('texto')*

2. Encontre o nome e a região do país que possui a maior população segundo este *Dataset*;

3. Agrupe os países por Regiões. Em seguida, mostre a média de alfabetização (Literacy (%)) de cada região do planeta;

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;

5. Faça uma função que receba a taxa de mortalidade de cada país (Deathrate) e retorne o texto 'Balanced' caso o valor seja < 9 e 'Urgent' caso contrário. Em seguida, crie um campo no *Dataset* chamado 'Humanitarian Help' que receba estes valores para cada país. No final, mostre o *Dataset* para verificar se a inserção da nova coluna foi feita com sucesso.



inatel.tecnologias
inatel.tecnologias
inateloficial
company/inatel
www.inatel.br

**Campus em Santa Rita do Sapucaí**
**Minas Gerais - Brasil**
Av. João de Camargo, 510
Centro - 37536-001

# CIÊNCIA DE DADOS COM PYTHON

## FIM CAPÍTULO 5

Inatel

_o futuro
não tem hora,
mas tem lugar.