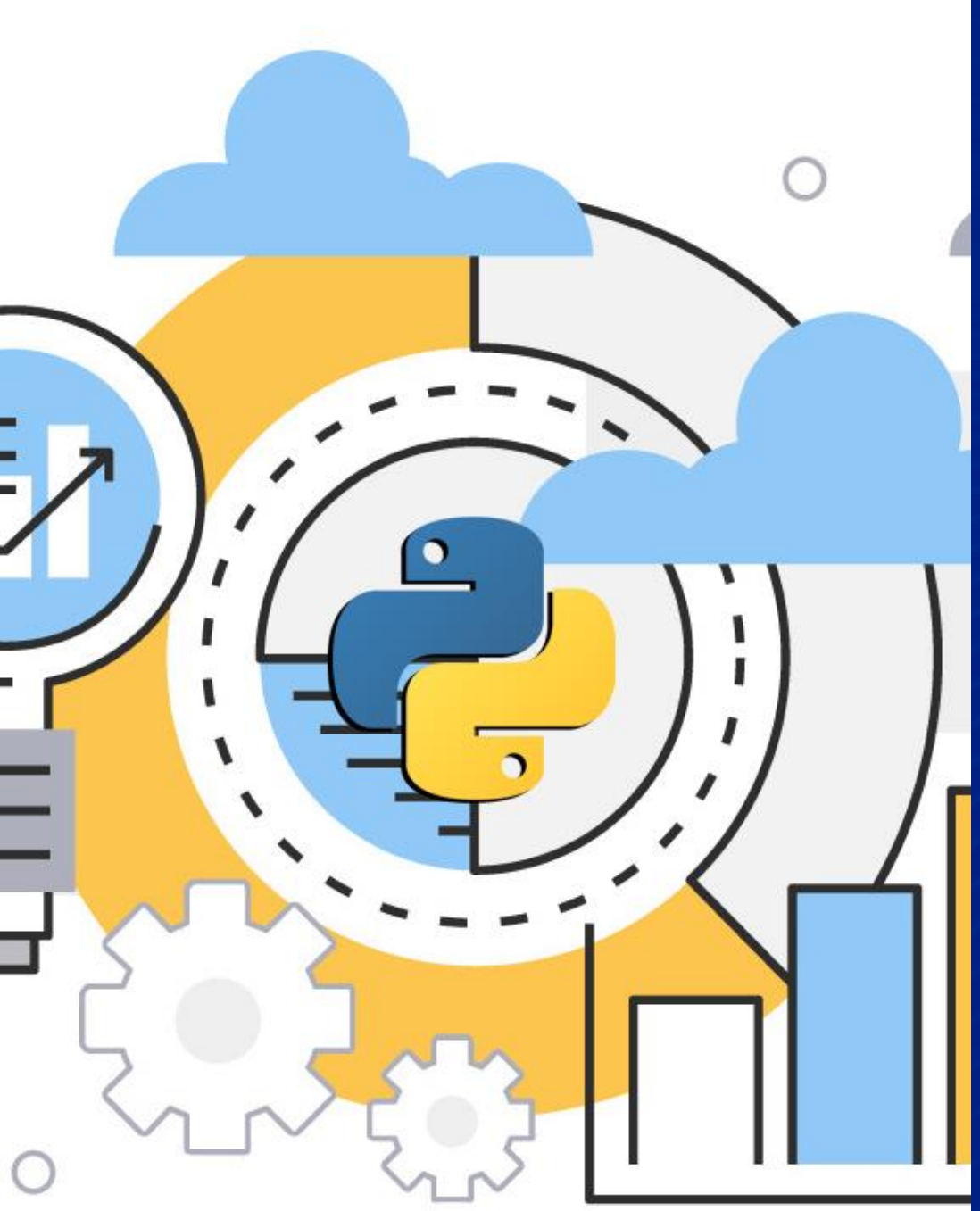
CIÊNCIA DE DADOS COM PYTHON

CAP.4 – INTRODUÇÃO AO NUMPY

Prof. Renzo Paranaíba Mesquita



OBJETIVOS

- Conhecer recursos essenciais do pacote mais fundamental para Ciência de Dados em Python, o NumPy
- Explorar suas funções mais populares e recursos de álgebra linear, como o NumPy Array de 1 e 2 dimensões
- Começar a realizar o carregamento e tratamento de *Datasets* por meio desta biblioteca

TÓPICOS

1. Introdução
2. Criação de NumPy Arrays
3. Operações Básicas com Arrays
4. Números aleatórios e Únicos em Arrays
5. Fatiamento de NumPy Arrays e Condicionais
6. Trabalhando com Padrões Textuais
7. Salvando e Carregando Dados com NumPy

Inatel

4.1. INTRODUÇÃO

- O NumPy (Numerical Python) é uma biblioteca *open source* em Python, largamente utilizada em quase todos os campos da ciência e engenharia
- É considerada o núcleo (*core)* dos ecossistemas PyData (bibliotecas e ferramentas da comunidade de *Data Science* em Python)
- Outros destaques desta biblioteca:
  - Manipulação de arrays multidimentsionais como vetores e matrizes
  - Grande variedade de operações matemáticas disponíveis
  - Preparada para lidar com grandes quantidades de dados (Sua base é criada na linguagem C)
  - Biblioteca mãe de outras, como Pandas e MatplotLib
- Documentação oficial disponível em: https://numpy.org/doc/2.1/reference/

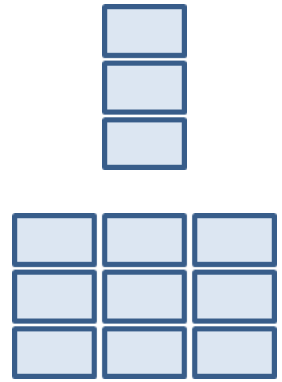
4.2. CRIAÇÃO DE NUMPY ARRAYS

- O ndarray (NumPy Array) é a estrutura de dados fundamental do NumPy
- O NumPy Array guarda apenas elementos do mesmo tipo
  - Isso fornece maior desempenho nas operações
  - Menor uso de memória
- Aqui, utilizaremos NumPy Arrays Unidimensionais e Bidimensionais
- **EXEMPLOS:**

```
# importando o numpy
import numpy as np

# array unidimensional
arr = np.array([1, 2, 3])

# array bidimensional (matriz)
mtz = np.array[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```



- **OBSERVAÇÕES:**
  - Observe o NumPy Array recebendo uma lista (arr) ou uma lista de listas (mtz) como argumento
  - Para quem estiver usando uma IDE, provavelmente precisará fazer *download* da biblioteca inicialmente antes de usá-la

4.2. CRIAÇÃO DE NUMPY ARRAYS

- Podemos facilmente criar novos NumPy Arrays a partir de métodos que oferecem diferentes configurações

- **EXEMPLOS:**

```
# cria um array uni ou bidimensional formado por 0`s
arr = np.zeros(n)
mtz = np.zeros([n,n])

# cria um array uni ou bidimensional formado por 1`s
arr = np.ones(n)
mtz = np.ones([n,n])

# cria um array unidimensional formado por elementos espaçados
arr = np.arange(n)

# (f – first element, l – last element, s - step)
arr = np.arange(f, l, s)
```

- **OBSERVAÇÕES:**
  - Veremos como transformar um array uni em bidimensional em breve com o conceito de **reshape**

4.3. OPERAÇÕES BÁSICAS COM ARRAYS

- Operações populares em arrays individuais ou entre arrays
- **EXEMPLOS:**

```
arr = np.array([1, 2, 3, 4])
arr2 = np.array([5, 6, 7, 8])

# pegando o menor valor do array e seu índice
print(arr.min(), arr.argmin())

# pegando o maior valor do array e seu índice
print(arr.max(), arr.argmax())

# pegando a soma e a média dos elementos do array
print(arr.sum(), arr.mean())

# realizando operações índice à índice nos NumPy Arrays
print(arr + arr2)
print(arr * arr2)

# concatenando dois numpy arrays
print(np.concatenate((arr, arr2))
```

- **OBSERVAÇÕES:**
  - Observe como a operação índice a índice facilita as operações entre NumPy Arrays
  - Observe a necessidade dos arrays terem o mesmo tamanho para realizar operações entre eles
  - Observe a função correta para se concatenar NumPy Arrays

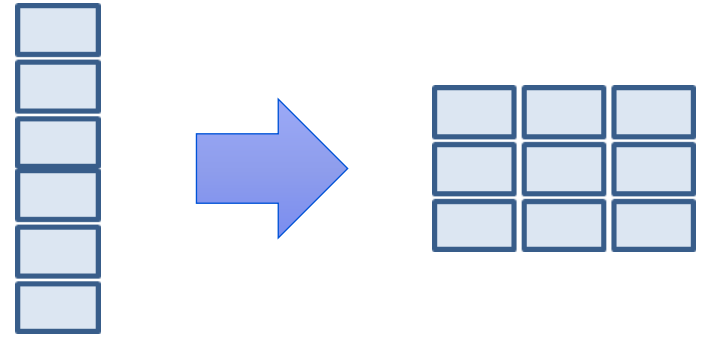
4.3. OPERAÇÕES BÁSICAS COM ARRAYS

- Uma tarefa fundamental na Ciência de Dados é a possibilidade de realizar o **Remodelamento (*Reshape)*** de estruturas como o Array.

- **EXEMPLOS:**

```
# criando um array unidimensional com 6 elementos
arr = np.arange(6)

# remodelando o array para uma matriz 3 x 2
mtz = arr.reshape(3, 2)
```



- Para compreendermos com precisão o tamanho, dimensão e formato de um NumPy Array, podemos utilizar de algumas propriedades importantes:

```
print(mtz.size) # retorna o tamanho de um NumPy Array
print(mtz.ndim) # retorna o número de dimensões de um NumPy Array
print(mtz.shape) # retorna uma tupla com o tamanho de cada dimensão do NumPy Array
```

- **OBSERVAÇÕES:**
  - Observe que quando se usa o **reshape()**, o array que se deseja produzir deve ter o mesmo número de elementos do array original

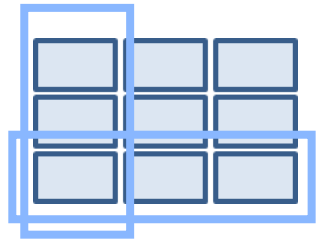
4.3. OPERAÇÕES BÁSICAS COM ARRAYS

- Operações básicas sobre matrizes também são simples, mas a especificação do eixo (linha ou coluna) dependendo do caso pode ser importante

- **EXEMPLOS:**

```
# aplica uma operação de soma sobre as colunas do NumPy Array
print(mtz.sum(axis=0))

# aplica uma operação de soma sobre as linhas do NumPy Array
print(mtz.sum(axis=1))
```

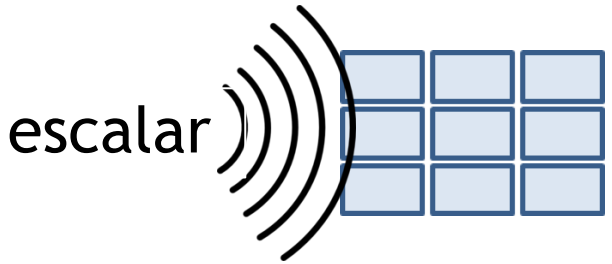


- O NumPy também permite a operação entre um vetor e um escalar (número avulso). Isso se chama **Broadcasting**

EXEMPLOS:

```
# multiplica todos os elementos da matriz por 5
print(mtz*5)

# divide todos os elementos da matriz por 2
Print(mtz/2)
```



- **OBSERVAÇÕES:**
  - Veja que as operações sobre colunas e linhas retornam uma lista de valores como resultado. Para acessar um elemento específico, basta realizar um ***Slicing*** de dados nela;
  - Outras operações que vimos como min(), max(), mean(), etc. também podem ser utilizadas para se realizar operações sobre linhas e colunas;

4.4. NÚMEROS ALEATÓRIOS E ÚNICOS COM ARRAYS

- O uso de números aleatórios é uma importante parte da configuração e avaliação de vários algoritmos numéricos
- No NumPy, eles podem ser gerados por meio do subpacote **random**

- **EXEMPLOS:**

```
# permite plantar uma Semente Aleatória para que os números aleatórios sejam produzidos iguais, independente da máquina que estão sendo gerados
np.random.seed(n)

# gera um Array Unidimensional de s (size) números inteiros positivos no intervalo entre l (low – inclusive) e h (high – exclusive)
print(np.random.randint(l, h, s))

# gera um Array Bidimensional de [s, s] de números inteiros positivos no intervalo entre l (low – inclusive) e h (high – exclusive)
print(np.random.randint(l, h, [s, s]))
```

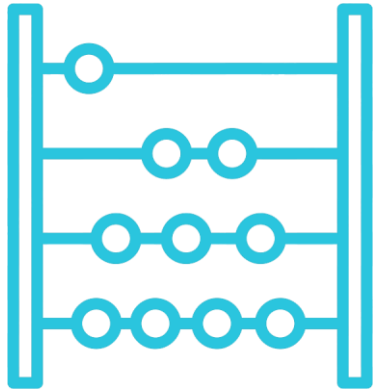
4.4. NÚMEROS ALEATÓRIOS E ÚNICOS COM ARRAYS

- É comum em vários momentos buscarmos por elementos únicos dentro de um NumPy Array, ou seja, sem repetições

- **EXEMPLOS:**

```
# retorna  elementos únicos em um NumPy Array, independente da dimensão
print(np.unique(mtz))

# retorna a frequência de cada elemento único no NumPy Array
print(np.unique(mtz, return_counts = True))
```



- **OBSERVAÇÕES:**
  - Observe como o unique com return_counts=True pode poupar um grande tempo na contagem de elementos únicos
  - Os exemplos foram aplicados sobre uma matriz, mas também funcionam normalmente em um NumPy Array unidimensional

EXERCÍCIOS (PARTE 1)

1. Crie dois NumPy Arrays unidimensionais de tamanho 8: um formado apenas por 1's e outro formado por números aleatórios entre 0 e 9. Some estes dois NumPy Arrays e guarde o resultado dentro de um terceiro NumPy Array. Por fim, faça o seguinte:
   a. Se a soma de todos os elementos do Array resultante for >= 40, remodele este NumPy Array para se tornar uma matriz com mais linhas do que colunas. Senão, remodele para que se torne uma matriz com mais colunas do que linhas.

2. Crie dois NumPy Arrays unidimensionais: um de números pares de 0 à 51 e outro também de números pares de 100 até 50. Em seguida, os concatene e mostre os resultados ordenados

3. Mini Campo Minado
   a) Crie um NumPy Array 2x2 formado apenas por 0's
   b) Em seguida, adicione um número 1 em uma posição aleatória desta matriz;
   c) Faça uma entrada de dados para solicitar o usuário que faça uma jogada (selecione uma posição da matriz)
      I. Se ele selecionar todas as posições em que o número 1 não se encontra, mostre a mensagem *"Congratulations! You beat the game! :)"*
      II. Senão, se dentro das 3 primeiras jogadas ele achar o número 1, mostre a mensagem *"Game Over! :(Try Again!"*

EXERCÍCIOS (PARTE 2)

4. Crie uma matriz de tamanho qualquer. Extraia seu número de linhas e colunas, multiplique-os, e diga se esta matriz poderia se tornar um vetor unidimensional com número par ou ímpar de elementos

5. Crie uma matriz de tamanho 4x4 formada por números aleatórios inteiros entre 1 e 50 (use seed = 10 antes)

a) Mostre o resultado da média de cada linha e cada coluna da matriz gerada

b) Apresente o maior valor das médias das linhas e também das colunas

c) Mostre a quantidade de aparições de cada um dos números gerados na matriz. Em seguida, mostre apenas os números que aparecem 2 vezes

4.5. FATIAMENTO DE NUMPY ARRAYS E CONDICIONAIS

- Podemos realizar o *Slicing* de NumPy Arrays unidimensionais da mesma forma que fizemos com as Tuplas e Listas
- Utilizaremos o *Slicing* de NumPy Arrays bidimensionais com frequência nesta disciplina

- **EXEMPLOS:**

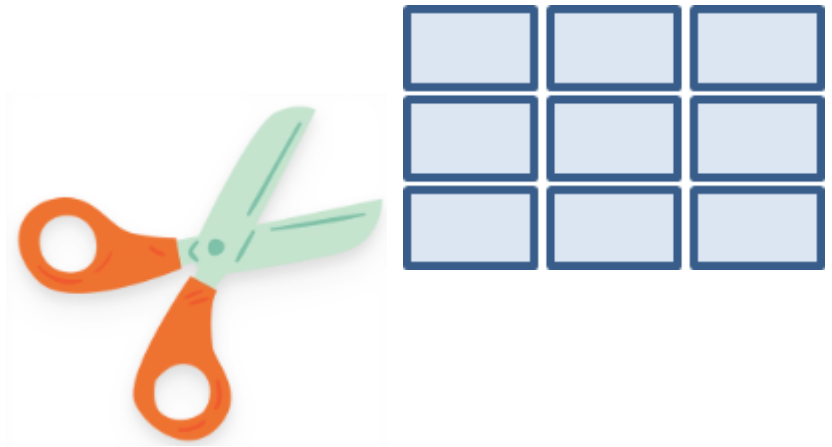
```
# array bidimensional (matriz)
mtz = np.array[[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]]

# fatiando uma única linha
print(mtz[2]) # [7, 8, 9]

# fatiando múltiplas linhas
print(mtz[0:2]) #[[1, 2, 3],
                 [4, 5, 6]]

# fatiando uma coluna
print(mtz[:,1]) # [2, 5, 8]

# fatiando linhas e colunas
print(mtz[0:2, 1:]) # [[2, 3],
                     [5, 6]]
```



Inatel

4.5. FATIAMENTO DE NUMPY ARRAYS E CONDICIONAIS

- Um dos recursos mais importantes para se usar no *Slicing* de arrays é o uso de condicionais
- Ele poupa o programador de realizar estruturas de repetição para se filtrar elementos de um NumPy Array

- **EXEMPLOS:**

```
# retorna True se a condição for atendida
print(mtz > 5) # [[False False False]
                  [False False True]
                   [True True True]]

# usa o condicional como um filtro no próprio Array para retornar os elementos que são True
print(mtz[mtz > 5]) # [6, 7, 8, 9]

# usando uma variável para guardar o condicional
cond = mtz%2==0
print(mtz[cond]) # [2, 4, 6, 8]
```

- **OBSERVAÇÕES:**
  - Observe que todo condicional que pode ser criado por meio de um if-else também pode ser utilizado aqui, porém, de forma mais simplificada

4.6. TRABALHANDO COM PADRÕES TEXTUAIS

- O subpacote **numpy.char** fornece um conjunto de operações para serem realizadas em elementos do tipo texto

- **EXEMPLOS:**

```
# para cada elemento do array, retorna o índice em que a substring inserida se encontra
# retorna -1 para textos em que a substring não aparece
np.char.find(arr, sub)

# retorna True para cada elemento do array iniciado com uma substring específica
np.char.startswith(arr, sub)

# retorna um array com textos colocados em letras maiúsculas
np.char.upper(arr)

# retorna True para cada texto do array formado apenas por letras
np.char.isalpha(arr)

etc..
```

4.7. SALVANDO E CARREGANDO DADOS COM NUMPY

- Com o NumPy, em muitos momentos será necessário carregarmos *datasets* a fim de analisá-los
- Poderemos também alterá-los e salvá-los para consultas futuras, se necessário
- Existem duas formas de se fazer este procedimento no NumPy

- **EXEMPLOS:**

```
# carregando um arquivo de texto tradicional ou .csv
dataset = np.loadtxt('arquivo.txt')
# salvando um arquivo de texto tradicional ou .csv
np.savetxt('arquivo.txt', dataset)

# carregando um arquivo binário NumPy (.npy)
dataset = np.load('arquivo.npy')
# salvando um arquivo binário NumPy (.npy)
np.save('arquivo.npy')
```

- **OBSERVAÇÕES:**
  - A fim de facilitar, os arquivos devem ser inseridos dentro da pasta raiz do seu projeto antes de serem lidos
  - Caso o arquivo seja adicionado em outra localidade, deve-se passar o endereço completo da localização como argumento
  - Arquivos .txt e .csv são altamente portáveis, já arquivos .npy são mais rápidos em suas operações

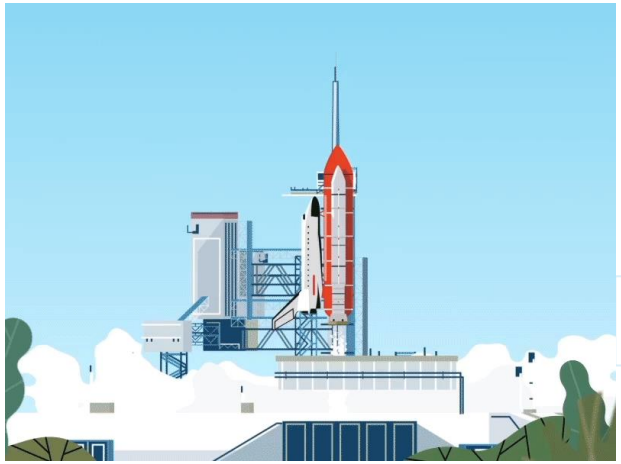
4.7. SALVANDO E CARREGANDO DADOS COM NUMPY

- *Datasets* são arquivos que contêm centenas ou até milhares de dados sobre um determinado assunto
- Como vimos no Cap.1, estes arquivos podem ser de diferentes formatos e oriundos de diferentes fontes
- Neste momento, aplicaremos nossos conhecimentos de NumPy Arrays para responder perguntas sobre o *Dataset* space.csv, que possui dados de missões espaciais de 1957 até meados de 2020
- Adicione o arquivo em seu Projeto e o importe fazendo uso da função loadtxt() que vimos:

- **EXEMPLOS:**

```
# carregando o dataset space.csv
dataset = np.loadtxt('space.csv', delimiter = ';', dtype=str,
encoding='utf-8')
```

- **OBSERVAÇÕES:**
  - delimiter: indica qual o símbolo do arquivo .csv utilizado para separar os campos do *Dataset*;
  - dtype: indica que todos os campos serão tratados como String por padrão;
  - encoding: indica a codificação padrão que os dados do *Dataset* serão lidos (às vezes necessário ao carregar um *Dataset*);

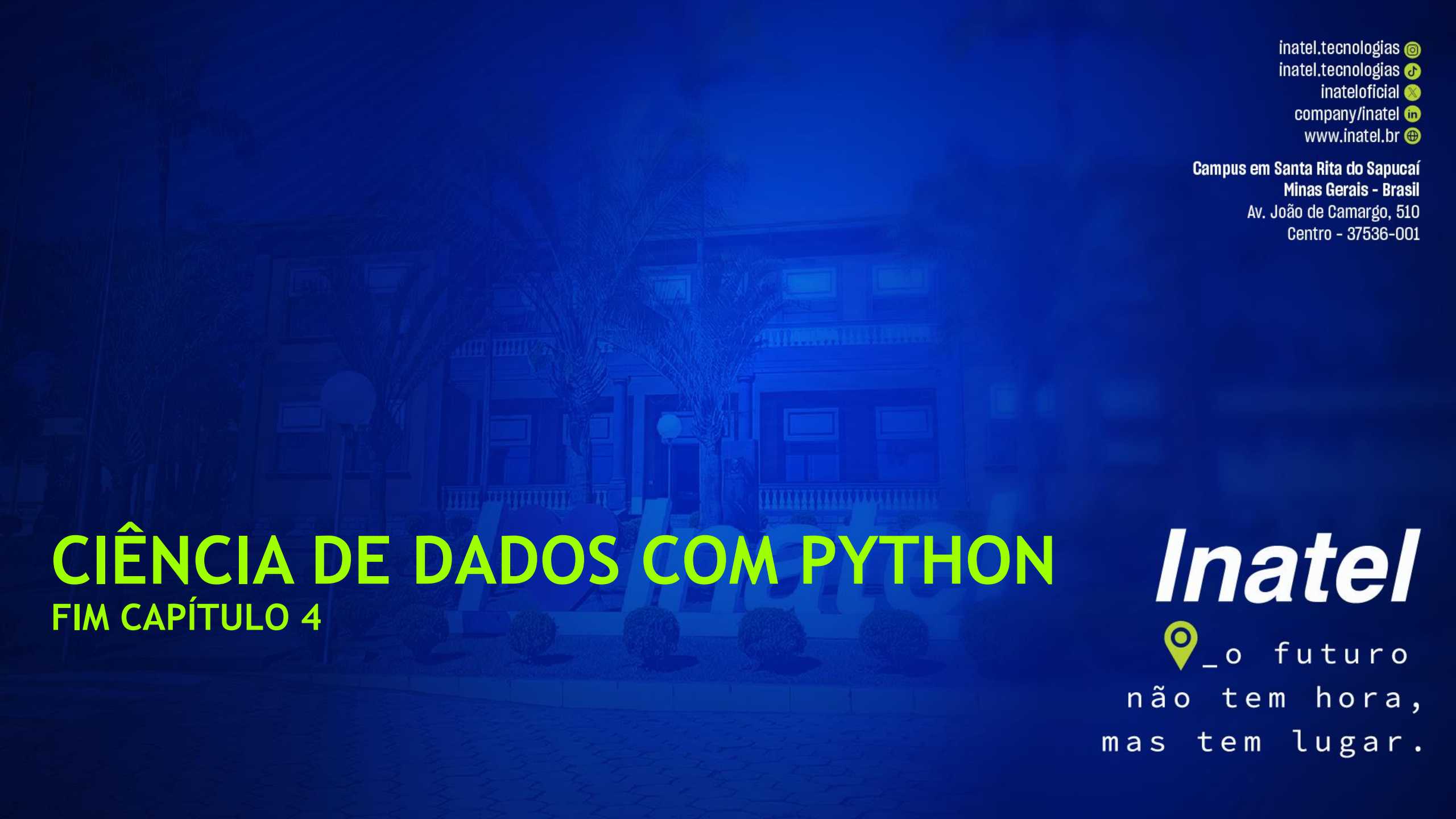


Inatel

EXERCÍCIOS (PARTE 3)

Baseado nos commandos que vimos até o momento e no Dataset fornecido, crie scripts em Python que respondam às seguintes perguntas:

1. Apresente a porcentagem de missões que deram certo
2. Qual a media de gastos de uma missão especial se baseando em missões que possuam valores disponíveis (> 0)?
3. Encontre quantas missões espaciais neste Dataset foram realizadas pelos Estados Unidos (EUA)
4. Encontre qual foi a missão mais cara realizada pela empresas “SpaceX”
5. Mostre o nome das empresas que já realizaram missões espaciais, juntamente com suas respectivas quantidades de missões (use o for no final para mostrar as informações)



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 4

Inatel

_o futuro
não tem hora,
mas tem lugar.