

# Introdução à Análise de Dados

Cap.4 - Introdução ao NumPy



Prof. MSc. Renzo P. Mesquita

# Objetivos

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



# **Capítulo 4**

## **Introdução ao NumPy**

- 4.1. Introdução;*
- 4.2. Criação de NumPy Arrays;*
- 4.3. Ordenando e Concatenando Arrays;*
- 4.4. Remodelamento (Reshape) de Arrays;*
- 4.5. Números Randômicos e Únicos em Arrays;*
- 4.6. Operações Básicas com Arrays;*
- 4.7. Fatiamento (Slicing) de Arrays;*
- 4.8. Trabalhando com Padrões Textuais;*
- 4.9. Salvando e Carregando Dados com NumPy.*



## 4.1. Introdução

O NumPy (Numerical Python) é uma biblioteca *open source* em Python que é largamente utilizada em quase todos os campos da ciência e engenharia.

Ela é o padrão universal para se trabalhar com dados numéricos em Python, e é considerada o núcleo (core) dos ecosistemas PyData (bibliotecas e ferramentas da comunidade de Data Science em Python).

Outros destaques desta biblioteca:

- Contém recursos poderosos para se criar e manipular arrays multidimensionais como vetores e matrizes;
- Usado para realizar uma grande variedade de operações matemáticas sobre estes arrays de forma mais simplificada;
- Biblioteca preparada para lidar com grandes quantidades de dados e eficiente em suas operações, pois sua base é criada na linguagem C;
- Biblioteca mãe de outras bibliotecas poderosas como Pandas, SciPy e muitas outras.



## 4.1. Introdução

*Apesar de ser uma biblioteca fundamental de Ciência de Dados em Python, é necessário inicialmente instalá-la.*

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

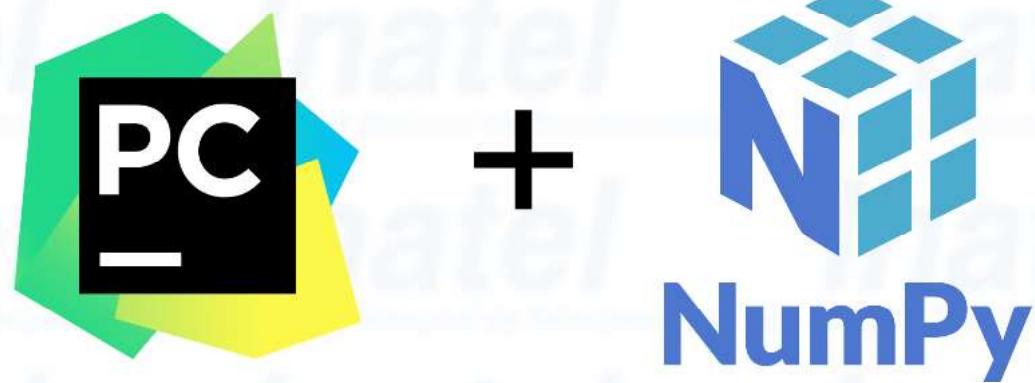
- Basta irmos em File -> Settings;
- No nome do seu projeto, vá em Project Interpreter -> clique em + (Install);
- Procure por NumPy e aperte o botão Install Package;

Uma vez instalada, basta importá-la no projeto:

*Ex:*

```
import numpy as np
```

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



## 4.2. Criação de NumPy Arrays

O Array (ou ndarray) é a estrutura central da biblioteca NumPy, e nada mais é que uma coleção de elementos indexados.

Mas qual a diferença entre uma lista (List) e um Array?

Uma lista pode conter diferentes tipos de dados dentro dela, já em um NumPy Array guarda-se apenas elementos do mesmo tipo. A grande vantagem da homogeneidade do NumPy Array é o maior desempenho em operações sobre seus elementos e o menor uso de memória.



Para se criar um NumPy Array, Ex: podemos utilizar da função np.array().

Command

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



NumPy Array

1
2
3

- Como pode ser visto, precisamos passar uma lista de elementos homogêneos como argumento;
- Para acessar um elemento do array, basta utilizar da notação arr[i], em que i é o índice pelo qual deseja acessar.

## 4.2. Criação de NumPy Arrays

Como vimos, uma forma popular de se criar um NumPy Array é por meio de uma lista, mas também podemos passar um conjunto de listas aninhadas para criação de Arrays com duas ou mais dimensões.

Ex:

```
mtz = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
```

- Atenção! Observe que passamos uma lista de listas como argumento para construirmos nosso Array de 2 dimensões (2D);
- Para acessar um elemento deste array, basta utilizar da notação arr[i][j], em que i é o índice da linha que deseja acessar e j o índice da coluna.



```
[[1 2 3]  
 [4 5 6]  
 [7 8 9]]
```



## 4.2. Criação de NumPy Arrays

*Além de criar um NumPy Array a partir de uma sequência de elementos, podemos facilmente criar novos Arrays a partir de métodos que oferecem diferentes configurações.*

### *Exemplos:*

- *arr = np.zeros(n) -> cria um array unidimensional formado por n 0's;*
- *arr = np.ones(n) -> cria um array unidimensional formado por n 1's;*
- *arr = np.arange(n) -> cria um array unidimensional com uma sequência de elementos, iniciando em 0 e indo até n (exclusive);*
- *arr = np.arange(f,l,s) -> cria um array unidimensional com uma sequência de elementos espaçados. Para isso, deve-se especificar o primeiro elemento (first - f), último (last - l) e o passo (step - s);*
- *arr = np.linspace(f,l,s) -> cria um array unidimensional formado por valores que são linearmente espaçados;*



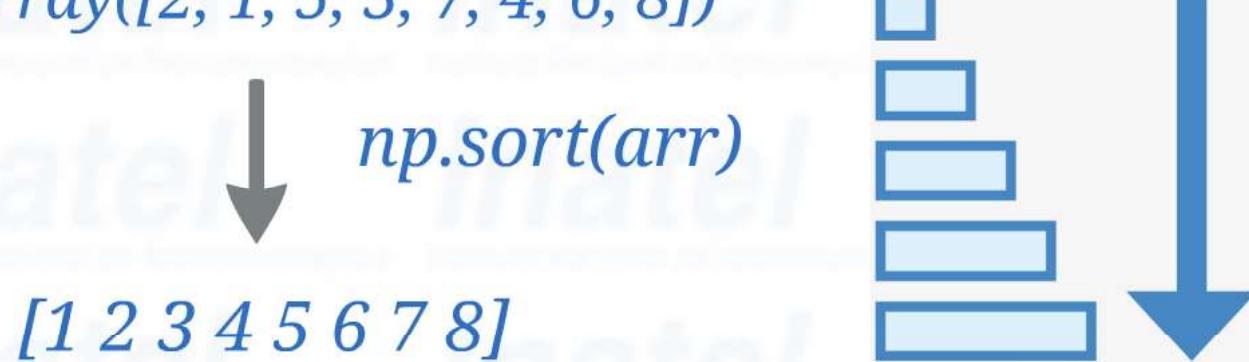
Os exemplos ilustram Arrays unidimensionais. Para criarmos, por exemplo, um Array bidimensional, basta passarmos uma lista com o numero de linhas e colunas como parâmetro para as funções zeros e ones e usar do conceito de reshape nas outras (veremos em breve).

## 4.3. Ordenando e Concatenando Arrays

Ordenar elementos é uma tarefa essencial e podemos fazer isso facilmente com `np.sort()`.

- `np.sort(arr)`

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



É importante ressaltar que só chamar o método `np.sort(arr)` não ordena o array definitivamente. Para se fazer isso, é necessário realizar o processo de atribuição.

Ex:

`arr = np.sort(arr)`

Para ordenar em ordem decrescente, uma alternativa é usando o `flip()`:

Ex:

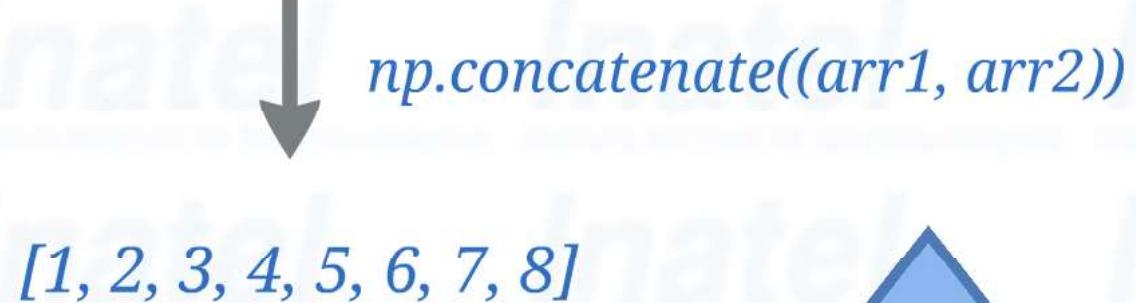
`np.flip(np.sort(arr))`

## 4.3. Ordenando e Concatenando Arrays

*Muitas vezes elementos não se encontram em apenas um Array, mas em vários deles. Podemos concatená-los facilmente com np.concatenate().*

- `np.concatenate(arr1,arr2)`

*Ex:* `arr1 = np.array([1,2,3,4]) arr2 = np.array([5,6,7,8])`



Atenção! Observe que os arrays como argumento do concatenate são passados como uma Tupla.

## 4.4. Remodelamento (Reshape) de Arrays

*Como entender de forma precisa o formato de um NumPy Array?*

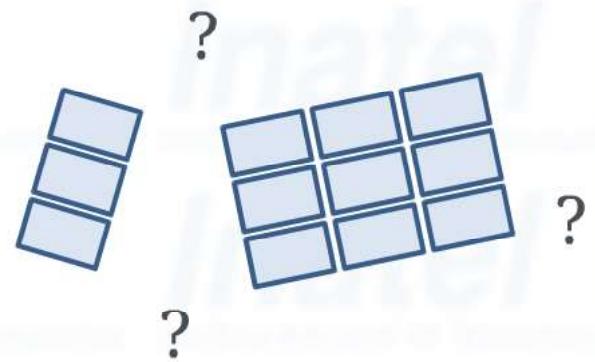
- `arr.size` -> Retorna o número de elementos de um Array;
- `arr.ndim` -> Retorna o número de dimensões de um Array;
- `arr.shape` -> Retorna uma tupla de inteiros que indica o número de elementos armazenados em cada dimensão do Array;

*Ex:*

```
mtz = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
```

```
print(mtz.size)  
print(mtz.ndim) →  
print(mtz.shape)
```

12  
2  
(3,4)



## 4.4. Remodelamento (Reshape) de Arrays

*Uma tarefa importante e fundamental da Ciência de Dados é a possibilidade de remodelamento (Reshape) de estruturas, como o Array.*

- `arr.reshape()` -> permite dar um novo formato ao Array sem perder seus dados.

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



`print(arr)` → [1 2 3 4 5 6]



`arr = arr.reshape(3,2)`



`print(arr)` →

```
[[1 2]
 [3 4]
 [5 6]]
```



Atenção! Observe que quando se usa do `reshape()`, o array que se deseja produzir deve ter o mesmo número de elementos do array original.

## 4.4. Remodelamento (Reshape) de Arrays

### Exercícios Propostos

*Baseado nos comandos que vimos até o momento, crie scripts em Python que resolvam os seguintes problemas:*

1. Crie um Array de tamanho 21 com valores linearmente espaçados entre 0 e 1;
2. Crie dois Arrays: um de números pares de 0 até 51 e outro também de número pares de 100 até 50. Em seguida, os concatene e mostre os resultados ordenados;
3. Ordene os resultados do array resultante da Questão 2 em ordem decrescente;
4. Crie uma matriz formada somente por uns de tamanho 3x4. Em seguida, transforme-a em um Array 1-D;
5. 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 com número par ou ímpar de elementos.

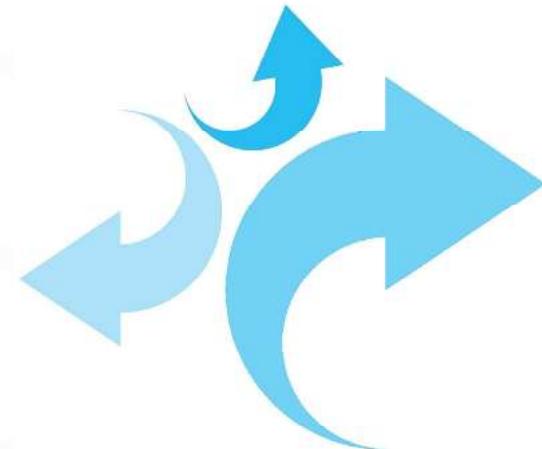


## 4.5. Números Randômicos e Únicos em Arrays

O uso de números aleatórios é uma importante parte da configuração e avaliação de vários algoritmos numéricos.

*Exemplos Fundamentais no NumPy:*

- `np.seed(n)` -> permite usar um número como "semente" para sempre gerar um conjunto de números aleatórios iguais;
- `np.random.rand(n)` -> gera um Array de  $n$  números de ponto flutuante (float) aleatórios entre 0 e 1;
- `np.random.rand(i,j)` -> gera um Array Bidimensional de  $n$  números de ponto flutuante (float) aleatórios entre 0 e 1;
- `np.random.randn(n)` -> gera um Array de  $n$  números de ponto flutuante (float) aleatórios (inclui negativos) sobre uma distribuição normal;
- `np.random.randint(l,h,s)` -> gera um Array de  $s$  números inteiros positivos aleatórios no intervalo entre  $l$  (inclusive) e  $h$  (exclusive);
- `np.random.shuffle(arr)` -> embaralha os valores de um Array;



## 4.5. Números Randômicos e Únicos em Arrays

*Em um Array é comum em alguns momentos buscarmos apenas elementos únicos, sem repetições.*

- `np.unique(arr)` -> busca elementos únicos em um NumPy Array;

Ex:

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



```
print(np.unique(arr))
```

```
[1 3 4 5 7 8]
```

Alguns parâmetros interessantes que podem ser usados na função unique:

- `return_index=True` -> retorna o índice dos primeiros elementos únicos que aparecem no Array;
- `return_counts=True` -> retorna a frequência de aparição de cada elemento único do Array;

Lembrando que estes usos também servem para matrizes, mas respeitando suas particularidades.

## 4.6. Operações Básicas com Arrays

*Uma vez que nossos Arrays estejam criados, podemos realizar operações sobre eles.*

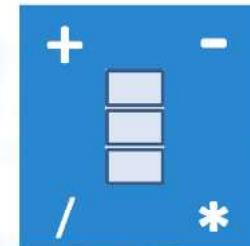
*Ex:*

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

`print(arr + arr2) → [ 6 8 10 12]`

`print(arr * arr2) → [ 5 12 21 32]`

...para outras operações básicas, a ideia é a mesma.



Se quisermos realizar operações sobre os elementos do array (como uma soma), podemos usar de uma série de funções fundamentais do NumPy.

*Ex:*

`print(arr.sum()) → 10`

Lembrando outras funções fundamentais que já vimos da Biblioteca Math e que por padrão também estão aqui:

- `min()` e `max()`: pegam o menor e maior valor respectivamente;
- `argmin()` e `argmax()`: pegam o índice do menor e maior valor respectivamente;
- `mean()`: faz a média dos elementos do Array;
- `astype(int)`: pega apenas a parte inteira de um float...

## 4.6. Operações Básicas com Arrays

*Operações básicas sobre matrizes também são simples, mas a especificação do eixo (linha ou coluna) em alguns casos pode ser importante.*

*Ex:*

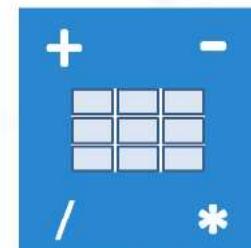
```
mtz = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
```

```
print(mtz.sum(axis=0)) → [15 18 21 24]
```

Eixo 0 = aplica uma operação sobre as colunas do array;

```
print(mtz.sum(axis=1)) → [10 26 42]
```

Eixo 1 = aplica uma operação sobre as linhas do array;



Veja que nos exemplos acima, é retornado um array de resultados. Um exemplo para retornar um resultado específico da soma de uma única coluna, por exemplo, seria:

*Ex:*

```
print(mtz.sum(axis=0)[1]) → 18
```

## 4.6. Operações Básicas com Arrays

*Existem momentos que desejamos realizar uma operação entre um array (vetor) e um único número (escalar). Este processo é chamado de broadcasting.*

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

`print(5*arr)` → [ 5 10 15 20]



`mtz = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])`

`print(mtz/5)` → [[0.2 0.4 0.6 0.8]  
[1. 1.2 1.4 1.6]  
[1.8 2. 2.2 2.4]]

## 4.6. Operações Básicas com Arrays

### Exercícios Propostos 2

*Baseado nos comandos que vimos até o momento, crie scripts em Python que resolvam os seguintes problemas:*

1. Crie um array de floats com 10 elementos positivos e negativos entre 0 e 1. Em seguida, multiplique seus valores por 100 e crie um novo vetor apenas com a parte inteira destes números; (use seed(5) antes)
2. Crie uma matriz de tamanho 4x4 formada por números aleatórios inteiros entre 1 e 50; (use seed(10) antes)
3. Mostre o resultado da média de cada linha e cada coluna da matriz gerada pela questão 2, e em seguida, apresente o maior valor das médias para as linhas e também para as colunas;
4. Baseado na matriz gerada na questão 2, mostre a quantidade de aparições de cada um dos números na mesma. Em seguida, mostre apenas os números que aparecem 2 vezes.



## 4.7. Fatiamento (Slicing) de Arrays

Podemos realizar o fatiamento de Arrays da mesma forma que fazemos com Strings e Listas.

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

`print(arr[0:2])` → [2 1]

`print(arr[1:])` → [1 5 3 7 4 6 8]

`print(arr[-2:])` → [6 8]

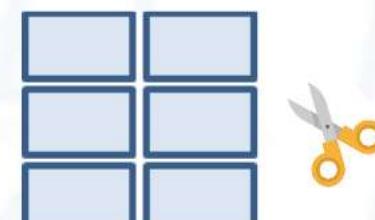


`mtz = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])`

`print(mtz[2])` → [ 9 10 11 12]

`print(mtz[0:2])` → [[1 2 3 4]  
[5 6 7 8]]

`print(mtz[0:2,2:])` → [[3 4]  
[7 8]]



## 4.7. Fatiamento (Slicing) de Arrays

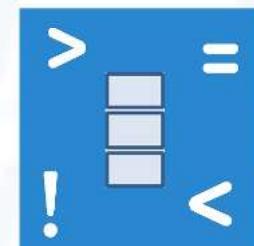
*Outro recurso importante para fatiamento de Arrays são as criações de condicionais*

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

`print(arr[arr <= 5])` → [2 1 5 3 4]

`print(arr[arr%2==0])` → [2 4 6 8]

`cond = (arr%2==0) & (arr%3==0)` → [6]  
`print(arr[cond])`



`print(arr > 5)` → [False False False False True False True True]

## 4.7. Fatiamento (Slicing) de Arrays

*Um detalhe muito importante que devemos nos atentar quando realizamos Slicing de Arrays é sobre a cópia dos mesmos.*

Após um Slicing, se guardamos o resultado em um novo Array e não mostrarmos ao Python que o mesmo se tornará uma nova cópia, alterações futuras no novo Array vão alterar o Array original.

Para isso, deveremos usar da função `copy()`.

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



*arr2 = arr[4:].copy()*



- Agora que a função `copy()` foi utilizada, o `arr2` será um Array independente, ou seja, não terá mais relação com o Array original;
- Faça este Slicing mas sem utilizar da função `copy()`. Altere os elementos do novo Array e mostre os elementos do Array original. Observe o que acontece;
- O Python faz isso pois o NumPy é uma biblioteca feita para tratar grandes quantidades de dados. Se ele automaticamente fizesse cópia de todos os Slicing que fizéssemos, teríamos problemas com alocação de memória.

## 4.8. Trabalhando com Padrões Textuais

*O Módulo numpy.char fornece um conjunto de operações para serem realizadas em elementos do tipo textos.*

Inclusive, muitas das funções aqui presentes já vimos no Capítulo de Fundamentos de Python. A diferença é que aqui as funções se aplicam a todos elementos de um NumPy Array.

*Alguns Exemplos:*

- *np.char.find(arr,sub) -> para cada elemento do array, retorna o índice em que a substring aparece (retorna -1 para textos onde a substring não aparece);*
- *np.char.startswith(arr,sub) -> retorna True para cada elemento do array iniciado por uma substring específica;*
- *np.char.upper(arr) -> retorna um array com os textos colocados em letras maiúsculas;*
- *np.char.isalpha(arr) -> retorna True para cada texto do array formado apenas por letras;*

substring

etc..

## 4.9. Salvando e Carregando Dados com NumPy

*Em algum momento, vamos querer salvar nossos Arrays em disco e carregá-los em outra oportunidade.*

Duas formas simplificadas de se fazer isso no NumPy é usando as funções:

- *savetxt() e loadtxt() -> para lidar com arquivos de texto tradicionais ou .csv.*

*Ex: np.savetxt('arquivos/array1.txt',arr) # salvando um arquivo*

*arr = np.loadtxt('arquivos/array1.txt') # carregando um arquivo*

- *save() e load() -> para lidar com arquivos binários NumPy (.npy).*

*Ex: np.save('arquivos/array1',arr) # salvando um arquivo*

*arr = np.load('arquivos/array1.npy') # carregando um arquivo*

Apesar de arquivos de textos (.txt e .csv) serem altamente portáveis, arquivos NumPy (.npy) são mais rápidos em suas operações. Estas funções são adequadas para Arrays homogêneos e simples.

## 4.8. 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 em NumPy Arrays para responder perguntas sobre o Dataset `space.csv` (extraído do website Next Spaceflight), que mapeou dados de missões espaciais que aconteceram desde 1957 até meados de 2020.

Para isso, adicione o arquivo em seu Projeto no PyCharm e o importe fazendo uso da função `loadtxt()` que vimos:

```
arr = np.loadtxt('arquivos/space.csv', delimiter=';',  
dtype=str, encoding='utf-8')
```

Observe os parâmetros:

- `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 tratados;



## 4.8. Salvando e Carregando Dados com NumPy

### *Exercícios Propostos 3*

*Baseado nos comandos que vimos até o momento e no DataSet fornecido, crie scripts em Python que resolvam os seguintes problemas:*

1. Apresente a porcentagem de quantas missões deram certo;
2. Qual a média de gastos de uma missão espacial 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 Empresa "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).



**FIM  
DO  
CAPÍTULO 4**



*Próximo Capítulo*  
Análise de Dados com  
Pandas