NumPy

1. Introdução

Segundo o site https://numpy.org/, o NumPy é o pacote fundamental para a computação científica com Python. Ele contém entre outras coisas:

- um poderoso objeto de matriz N-dimensional;
- funções sofisticadas (transmissão);
- ferramentas para integrar códigos escritos em C / C ++ e Fortran;
- recursos úteis de álgebra linear, transformação de Fourier e números aleatórios.

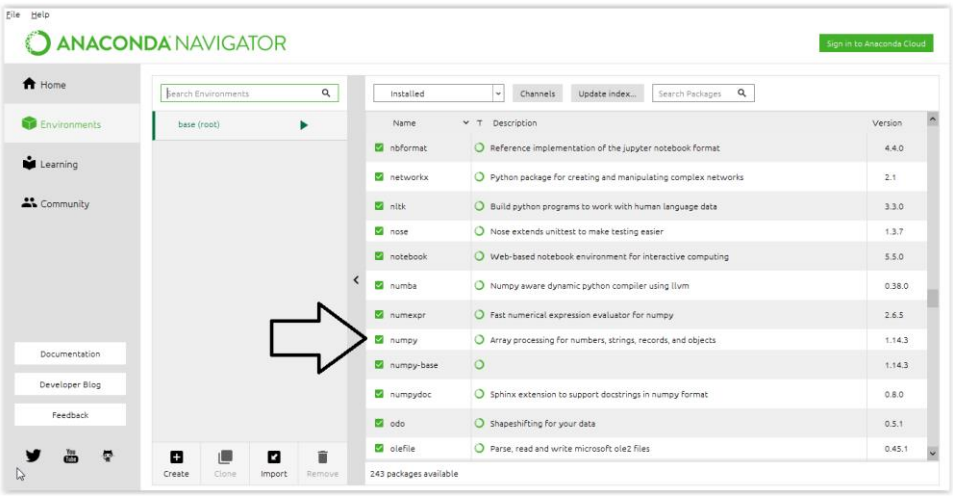
Além de seus óbvios usos científicos, o NumPy também pode ser usado como um eficiente recipiente multidimensional de dados genéricos. Tipos de dados arbitrários podem ser definidos. Isso permite que o NumPy se integre de maneira fácil e rápida a uma ampla variedade de bancos de dados. O NumPy é licenciado sob a licença BSD , permitindo a reutilização com poucas restrições.

Compreender o uso do NumPy pode ajudar a usar ferramentas de análise de dados como o Pandas, cujas principais operações são as seguintes operações, por exemplo:

1.1. remoção e limpeza de dados;
1.2. subconjunto e filtragem;
1.3. transformação;
1.4. classificação, únicas e definidas;
1.5. estatísticas descritivas eficientes e agregação;
1.6. mesclar e unir conjuntos de dados heterogêneos.

2. Instalação do NumPy

Certifique-se que o NumPy está disponível em seu ambiente de desenvolvimento. No caso do Anaconda, na interface do Anaconda Navigator, selecione Environments no menu lateral esquerdo. E certifique-se que o NumPy está instalado.



3. Começando a usar NunPy

Usaremos a convenção “import numpy como np”, de modo que você precisa apenas incluir np. antes do nome do método.

3.1. Utilizando arrays

Em Python podemos usar tanto Listas quanto Arrays para armazenar dados, e ambos podem ser indexados e iterados. Os arrays precisam ser declaradas, enquanto as listas não precisam, pois fazem parte da sintaxe nativa do Python. Portanto, as listas são mais usadas que os arrays. Entretanto, se precisar executar alguma função aritmética deve utilizar arrays.

```
minhaLista = [3,5,7,9]
print(minhaLista/2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-9da9af4ed6c6> in <module>()
      1 minhaLista = [3,5,7,9]
----> 2 print(minhaLista/2)

TypeError: unsupported operand type(s) for /: 'list' and 'int'
```

```
import numpy as np
minhaLista = [3,5,7,9]
meuArray = np.array(minhaLista)
print(meuArray/2)
```

```
[1.5 2.5 3.5 4.5]
```

3.2. Principais funções com arrays

Um ndarray é um contêiner multidimensional genérico para dados homogêneos, ou seja, todos os elementos devem ser do mesmo tipo.

- array( ): converte dados de entrada (lista, tupla, array ou outro tipo de sequência) em um ndarray;
- arange( ): como o intervalo interno, mas retorna um ndarray em vez de uma lista;
- Ones( ): produza uma matriz de todos os 1s com a forma e o tipo especificados. Já a função ones_like leva outra array e produz um array ones da mesma forma e tipo;
- Zeros( ): produz arrays de zer;
- Empty(): produz um array vazio, não necessariamente com zeros.

```python
import numpy as np
dados = [3,5,7,9]
print("dados")
print(dados)

array = np.array(dados)
print("\narray")
print(array)

arranjo = np.arange(10)
print("\narranjo")
print(arranjo)

uns = np.ones((2,2))
print("\nuns")
print(uns)

zeros = np.zeros(10)
print("\nzeros")
print(zeros)

zeros2 = np.zeros((2,3))
print("\nzeros2")
print(zeros2)

vazios = np.empty((2,3))
print("\nvazios")
print(vazios)
```

```
dados
[3, 5, 7, 9]

array
[3 5 7 9]

arranjo
[0 1 2 3 4 5 6 7 8 9]

uns
[[1. 1.]
 [1. 1.]]

zeros
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

zeros2
[[0. 0. 0.]
 [0. 0. 0.]]

vazios
[[0. 0. 0.]
 [0. 0. 0.]]
```

3.3. Operações com arrays

Outra vantagem do array é poder realizar operações em lote com os dados, sem precisar utilizar um loop, também conhecido com vetorização. Inclusive muitas operações simples como sqrt ou exp.

```python
import numpy as np
a1 = np.array([[1,2,3],[4,5,6]])
print("\narray1")
print(a1)

print("\narray2")
a2 = np.ones((2,3))
print(a2)

print("\narray1 + array2")
print(a1+a2)

print("\narray1 x 2")
print(a1*2)

print("\narray1 elevado a 2")
print(a1**2)

print("\narray1 - array1")
print(a1-a1)
```

```
array1
[[1 2 3]
 [4 5 6]]

array2
[[1. 1. 1.]
 [1. 1. 1.]]

array1 + array2
[[2. 3. 4.]
 [5. 6. 7.]]

array1 x 2
[[ 2  4  6]
 [ 8 10 12]]

array1 elevado a 2
[[ 1  4  9]
 [16 25 36]]

array1 - array1
[[0 0 0]
 [0 0 0]]
```

3.4. Outras operações com arrays

Observe que existe uma importante diferença entre lista e arrays. Fatias de um array são visualizações do array original. Portanto, os dados não são copiados e quaisquer modificações na exibição serão refletidas no array original.

```
minhaLista =  [3,4,5,7]
print("lista")
print(minhaLista)

pedacoLista = minhaLista[1:3]
print("parte da lista")
print(pedacoLista)

pedacoLista[1]= 23
print("modificando a parte da lista")
print(pedacoLista)

print("voltando à lista original")
print(minhaLista)#não mudou a lista original
```

```
lista
[3, 4, 5, 7]
parte da lista
[4, 5]
modificando a parte da lista
[4, 23]
voltando à lista original
[3, 4, 5, 7]
```

```
import numpy as np
minhaLista =  [3,4,5,7]
meuArray = np.array(minhaLista)
print("\n\nmeu array")
print(meuArray)

pedacoArray = meuArray[1:3]
print("parte do array")
print(pedacoArray)

pedacoArray[1]= 23
print("modificando a parte do array")
print(pedacoArray)

print("voltando ao array original")
print(meuArray)#mudou o array original
```

```
meu array
[3 4 5 7]
parte do array
[4 5]
modificando a parte do array
[ 4 23]
voltando ao array original
[ 3  4 23  7]
```

Em matrizes multidimensionais, se você omitir índices posteriores, o objeto retornado será uma dimensão dimensional ndarray que consiste em todos os dados ao longo das dimensões mais altas.

```python
import numpy as np

array3d = np.array([[[1, 2, 3], [4, 5, 6]],
                    [[7, 8, 9], [10, 11, 12]]])
print("\narray 3d")
print(array3d)

print("\nprimeiro array")
print(array3d[0])

print("\ncopiando os dados do primeiro array")
old_values = array3d[0].copy()
print(old_values)

print("\nmudando o valor do primeiro array")
array3d[0] = 25
print(array3d[0])

print("\nvoltando o valor do primeiro array")
array3d[0] = old_values
print(array3d[0])
```

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

 [[ 7  8  9]
  [10 11 12]]]

primeiro array
[[1 2 3]
 [4 5 6]]

copiando os dados do primeiro array
[[1 2 3]
 [4 5 6]]

mudando o valor do primeiro array
[[25 25 25]
 [25 25 25]]

voltando o valor do primeiro array
[[1 2 3]
 [4 5 6]]
```