numpy-cheatsheet

July 11, 2023

Numpy is a popular Python library for scientific computing and data analysis. It provides a powerful array object, along with many useful functions and methods for manipulating and processing numerical data. In this blog post, I will share some of the essential numpy operations that can help you write faster and cleaner code. Whether you are a beginner or an expert, I hope you will find something useful in this numpy cheat sheet. https://neuronize.dev/numpy-101-learn-numpy-in-10-minutes

1 Installation

1.1 Using pip

[ ]:
```
pip install numpy
```

1.2 Using Conda

[ ]:
```
conda install -c anaconda numpy
```

2 Importing Numpy

[1]:
```
import numpy as np
```

3 Scalar

Let's start from the very basic and learn how to create scalar values of numpy class.

[2]:
```
# int type

python_int = 2
numpy_int = np.int32(2)
print(type(python_int)) # ---> <class 'int'>
print(type(numpy_int)) # ---> <class 'numpy.int32'>

# float type

python_float = 2.5
numpy_float = np.float32(2.5)
```

```
print(type(python_float)) # ---> <class 'float'>
print(type(numpy_float)) # ---> <class 'numpy.float32'>
```

```
<class 'int'>
<class 'numpy.int32'>
<class 'float'>
<class 'numpy.float32'>
```

4 1-D Array

Let's now create a 1-D array using numpy's array() method.

[3]:
```
# creating an 1-D array of int type

one_d_array_int = np.array([1,2,3])
print(one_d_array_int) # ---> output : [1 2 3]

# creating an 1-D array of float type

one_d_array_float = np.array([1.5,2.5,3.5])
print(one_d_array_float) # ---> output : [1.5 2.5 3.5]
```

```
[1 2 3]
[1.5 2.5 3.5]
```

5 2-D Array

Let's now create a 2-D array using numpy's array() method.

[4]:
```
# 2-D int array

two_d_array_int = np.array([
                [1,2,3],
                [4,5,6],
                [7,8,9]
               ])
print(two_d_array_int) # ---> [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# 2-D float array

two_d_array_float= np.array([
                [1.1,2.1,3.0],
                [4.1,5.1,6.1],
                [7.1,8.1,9.1]
               ])
print(two_d_array_float) # ---> [[1.1, 2.1, 3.0], [4.1, 5.1, 6.1], [7.1, 8.1, 9.
  ↪1]]
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[1.1 2.1 3. ]
 [4.1 5.1 6.1]
 [7.1 8.1 9.1]]
```

6 Get The Dimension of an Array

Now, let's learn how do you get the dimension of the any array.

```
[5]: # If the array is a 1-D array, then it'll return 1
     print(one_d_array_int.ndim) # ---> output : 1

     # If the array is a 2-D array, then it'l return 2
     print(two_d_array_int.ndim) # ---> output : 2
```

```
1
2
```

7 Get The Shape of an Array

```
[6]: print(one_d_array_int.shape) # ---> output : (3,)


     print(two_d_array_int.shape) # ---> output : (3, 3)
```

```
(3,)
(3, 3)
```

8 Get the Data Type of an Array

```
[7]: print(one_d_array_int.dtype) # ---> output : dtype('int32')
```

```
int64
```

9 Manipulating Elements of an Array

Let's create a new array and then we will perform operations on this new array.

```
[8]: a = np.array([[1,2,3,4,5,6,7],[8,9,10,11,12,13,14]])
     print(a) # ---> output : [[ 1  2  3  4  5  6  7]
     #                         [ 8  9 10 11 12 13 14]]
```

```
[[ 1  2  3  4  5  6  7]
 [ 8  9 10 11 12 13 14]]
```

10 Access a Specific Element

Let's say we want to access first rows' third element.

```
[9]: # accessing 1st row's 3rd element
     print(a[0,2]) # ---> output : 3
```

```
3
```

11 Access only a specific row or column

```
[10]: # accessing only the first row
      print(a[0,:]) # ---> output : [1 2 3 4 5 6 7]

      # accessing only the second column
      print(a[:,1]) # ---> output : [2 9]
```

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

12 Access elements with skip steps

```
[11]: # [startindex:endindex:stepsize]
      print(a[0, 1:-1:2]) # ---> output : [2 4 6]
```

```
[2 4 6]
```

13 Changing element's value

```
[12]: a[0,2] = 99
      print(a) # ---> output : [[ 1  2 99  4  5  6  7]
      #                         [ 8  9 10 11 12 13 14]]
```

```
[[ 1  2 99  4  5  6  7]
 [ 8  9 10 11 12 13 14]]
```

14 Special Arrays in Numpy

14.1 Zero Matrix

```
[13]: # creating a zero matrix of shape(2,3)

      print(np.zeros((2,3))) # ---> output : [[0. 0. 0.]
       #                                      [0. 0. 0.]
       #                                      [0. 0. 0.]]
```

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

14.2 One Matrix

[14]:
```
# All 1s matrix
print(np.ones((3,2,2))) # ---> output : [[[1. 1.]
#                                         [1. 1.]]
#
#                                         [[1. 1.]
#                                          [1. 1.]]
#
#                                         [[1. 1.]
#                                          [1. 1.]]]

# creating ones matrix int type
print(np.ones((3,2,2), dtype='int32')) # ---> output : [[[1 1]
#                                         [1 1]]
#
#                                         [[1 1]
#                                          [1 1]]
#
#                                         [[1 1]
#                                          [1 1]]]
```

```
[[[1. 1.]
  [1. 1.]]

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

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

 [[1 1]
  [1 1]]

 [[1 1]
  [1 1]]]
```

15 Random Array

```
[15]: print(np.random.rand(3,3)) # ---> output : [[0.10671372 0.31133182 0.56572354]
      #                                            [0.34792672 0.88867917 0.25310353]
      #                                            [0.70052117 0.53243035 0.67948057]]
```

```
[[0.37037094 0.82762615 0.72189694]
 [0.39966545 0.60420349 0.83856636]
 [0.7197862  0.55920717 0.15818768]]
```

16 Random Array (Only Integer Values)

```
[16]: # np.random.randint(start_value,end_value,size)

      print(np.random.randint(-1,5, size=(3,3))) # ---> output : [[0 -1 2]
      #                                                           [4 2 1]
      #                                                           [4 4 0]]
```

```
[[-1  3  3]
 [ 0 -1  3]
 [ 0  0  3]]
```

17 Identity Matrix

```
[17]: print(np.identity(3)) # ---> output : [[1. 0. 0.]
      #                                       [0. 1. 0.]
      #                                       [0. 0. 1.]]
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

18 Transposing Array

```
[18]: a = np.random.randint(-1,5, size = (2,3)) # creating a random int array
      print(a)                                  # output : [[ 2 -1  1]
                                                #           [ 0  1  1]]


      print(a.T)                                # output : [[ 2  0]
                                                #           [-1  1]
                                                #           [ 1  1]]
```

```
[[-1  4  4]
 [-1  1  0]]
[[-1 -1]
```

```
 [ 4  1]
 [ 4  0]]
```

19 Reshape Array

[19]:
```
a = np.random.randint(-1,5, size = (2,3)) # creating a random int array
print(a)                                  # output : [[ 2 -1  1]
                                          #           [ 0  1  1]]

print(a.reshape(3,2))                     # output : [[ 2  0]
                                          #           [-1  1]
                                          #           [ 1  1]]

print(a.reshape(1,6))                     # output : [[1 2 3 2 2 2]]
```

```
[[ 0  0 -1]
 [ 4  4  0]]
[[ 0  0]
 [-1  4]
 [ 4  0]]
[[ 0  0 -1  4  4  0]]
```

20 Fill with Any Number

[20]:
```
print(np.full((3,3), 14)) # ---> output : [[14 14 14]
#                                          [14 14 14]
#                                          [14 14 14]]
```

```
[[14 14 14]
 [14 14 14]
 [14 14 14]]
```

21 Fill Like

[21]:
```
# Any other number (fill_like)
print(np.full_like(a, 4)) # ---> output : [[4 4 4 4 4 4 4]
#                                          [4 4 4 4 4 4 4]]
```

```
[[4 4 4]
 [4 4 4]]
```

22 Copy Array

[22]:
```
a = [1,2,3]             # created an array a
b = [10,20,30]          # created an array b
print(a)                # output : [1 2 3]
a = b                   # I'll explain it below
b[1] = 200              # changed b's 2nd element's value to 200
print(a)                # output : [10 200 30] HOW??????
```

```
[1, 2, 3]
[10, 200, 30]
```

23 Stacking Arrays

[23]:
```
# Vertically stacking vectors
v1 = np.array([1,2,3,4])                # created array v1
v2 = np.array([5,6,7,8])                # create array v2

np.vstack([v1,v2,v1,v2])                # output : [[1 2 3 4]
                                        #           [5 6 7 8]
                                        #           [1 2 3 4]
                                        #           [5 6 7 8]]
print(np.hstack([v1,v2]))               # output : [1 2 3 4 5 6 7 8]
```

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

24 Aggregate Functions

[24]:
```
a = np.array([1, 2, 3])
print(a.sum())                                  # sum of all values of a
print(a.max())                                  # max of all elements of a
print(a.mean())                                 # return mean of a
print(np.median(a))                             # return median of a
print(np.std(a))                                # return standard deviation of a
```

```
6
3
2.0
2.0
0.816496580927726
```

25 Conclusion

This concludes our numpy cheatsheet notebook. We hope you found it useful and learned some new tricks to work with arrays and matrices in Python. Numpy is a powerful and versatile library that can help you perform various numerical computations and data analysis tasks. If you want to learn more about numpy, you can check out the official documentation or some of the online tutorials and courses available. Thank you for reading and happy coding!