Numpy Advanced Topics

Transposing and swap axes

Transpose is special form of reshaping

e.g. arr = np.arange(15).reshape((3, 5)) is array with 3 rows and 5 columns to make it array of 5 rows and 3 columns we can transpose it with arr.T

Swap axes is also works as reshaping for higher dimensional arrays

e.g. arr1 = np.arange(8).reshape((2,4))

```
print(arr1)
print(arr1.swapaxes(0,1))
```

Universal functions

- ufunc, is a function that performs element-wise operations on data in ndarrays.
- e.g. arr = np.arange(10)
  - np.sqrt(arr) #Square root
  - np.exp(arr) # Exponential
  - **These are unary ufuncs**
  - **There are binary ufuncs too**
  - x = np.random.randn(8)
  - y = np.random.randn(8)
  - np.maximum(x, y)

Mathematical and Statistical methods

- Create Numpy array np_height, that is equal to first column of np_baseball.
- Print out the mean of np_height.
- Print out the median of np_height.
- Print out the sum, cumulative sum and cumulative product of np_weight
- Use np.std() on the np_weight to calculate std dev
- Do big players tend to be heavier? Use np.corrcoef() to store the correlation between the first and second column of np_baseball in corr.

Solution

```
# Import numpy import numpy as np

# Create np_height from np_baseball np_height=np_baseball[:,0]

# Create np_height from np_baseball np_weight=np_baseball[:,1]

# Print out the mean of np_height print(np.mean(np_height))

# Print out the median of np_height print(np.median(np_height))

# Print out the median of np_height print(np.sum(np_weight))
```

_VOIS


```
# Print out the median of np_height print(np.cumsum(np_weight))

# Print out the median of np_height print(np.cumprod(np_weight))

# Average of np_height avg = np.mean(np_baseball[:,0])

# Print median height.med = np.median(np_baseball[:,0])

# Print out the standard deviation on height

stddev = np.std(np_baseball[:,0])

# Print out correlation between first and second column.

corr = np.corrcoef(np_baseball[:,0],np_baseball[:,1])
```

Methods for Boolean Arrays

e.g. **arr = np.random.randn(100)**
(arr > 0).sum() # Number of positive values

- There are two additional methods, any and all ,any tests whether one or more values in an array is True , while all checks if every value is True :

e.g. **bools = np.array([False, False, True, False])**
bools.any()
bools.all()

Sorting

- NumPy arrays can be sorted in-place with the sort method:
- e.g. **arr = np.random.randn(6)**

  arr_sorted = np.sort(arr)

Unique Function

- np.unique returns the sorted unique values in an array
- e.g. **names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe']**

  np.unique(names)

File Input and Output with Arrays

- NumPy is able to save and load data to and from disk either in text or binary format.
- np.save and np.load are the two workhorse functions for efficiently saving and loading array data on disk. Arrays are saved by default in an uncompressed raw binary format with file extension .npy:
- e.g. **arr = np.arange(10)**

  np.save('some_array', arr)

  np.load('some_array.npy')

Flattening and Raveling

- opposite operation of reshape from one-dimensional to a higher dimension is typically known as flattening or raveling

  e.g. **arr = np.arange(15).reshape((5, 3))**

  arr.ravel()

- ravel does not produce a copy of the underlying values if the values in the result were contiguous in the original array. The flatten method behaves like ravel except it always returns a copy of the data

  e.g. **arr = np.arange(15).reshape((5, 3))**

  arr.flatten()

Concatenating and Splitting Arrays

- numpy.concatenate takes a sequence (tuple, list, etc.) of arrays and joins them together in order along the input axis
  e.g. **arr1 = np.array([[1, 2, 3], [4, 5, 6]])**
  arr2 = np.array([[7, 8, 9], [10, 11, 12]])
  np.concatenate([arr1, arr2], axis=0)
  np.concatenate([arr1, arr2], axis=1)
  np.vstack((arr1, arr2))
  np.hstack((arr1, arr2))

- split , on the other hand, slices apart an array into multiple arrays along an axis
  e.g. **arr = np.random.randn(5, 2)**
  first, second, third = np.split(arr, [1, 3])

Repeating Elements: tile and repeat

- useful tools for repeating or replicating arrays to produce larger arrays are the repeat and tile functions. repeat replicates each element in an array some number of times, producing a larger array
  e.g. arr = np.arange(3)
  arr.repeat(3)
- you pass an array of integers, each element can be repeated a different number of times
  e.g. arr.repeat([2, 3, 4])
- Multidimensional arrays can have their elements repeated along a particular axis
  e.g. arr = np.random.randn(2, 2)
  arr.repeat(2, axis=0)
- tile is a shortcut for stacking copies of an array along an axis
  e.g. np.tile(arr, 2)
  np.tile(arr, (2, 1))
  np.tile(arr, (3, 2))

Writing New ufuncs in Python

- numpy.frompyfunc accepts a Python function along with a specification for the number of inputs and outputs.

  e.g. **def add_elements(x, y):**
  return x + y

  add_them = np.frompyfunc(add_elements, 2, 1) **#arguments function , inputs , outputs**

  add_them(np.arange(8), np.arange(8))

References -

1. Zed A Shaw , *Learn Python 3 the hard way* , Addison Wesley
2. Erric Matthes , *Python Crash Course* , No starch press
3. Wes McKinney , *Python for data analysis* , O'Reilly Media, Inc.


thank you