CS 412 Introduction to Machine Learning

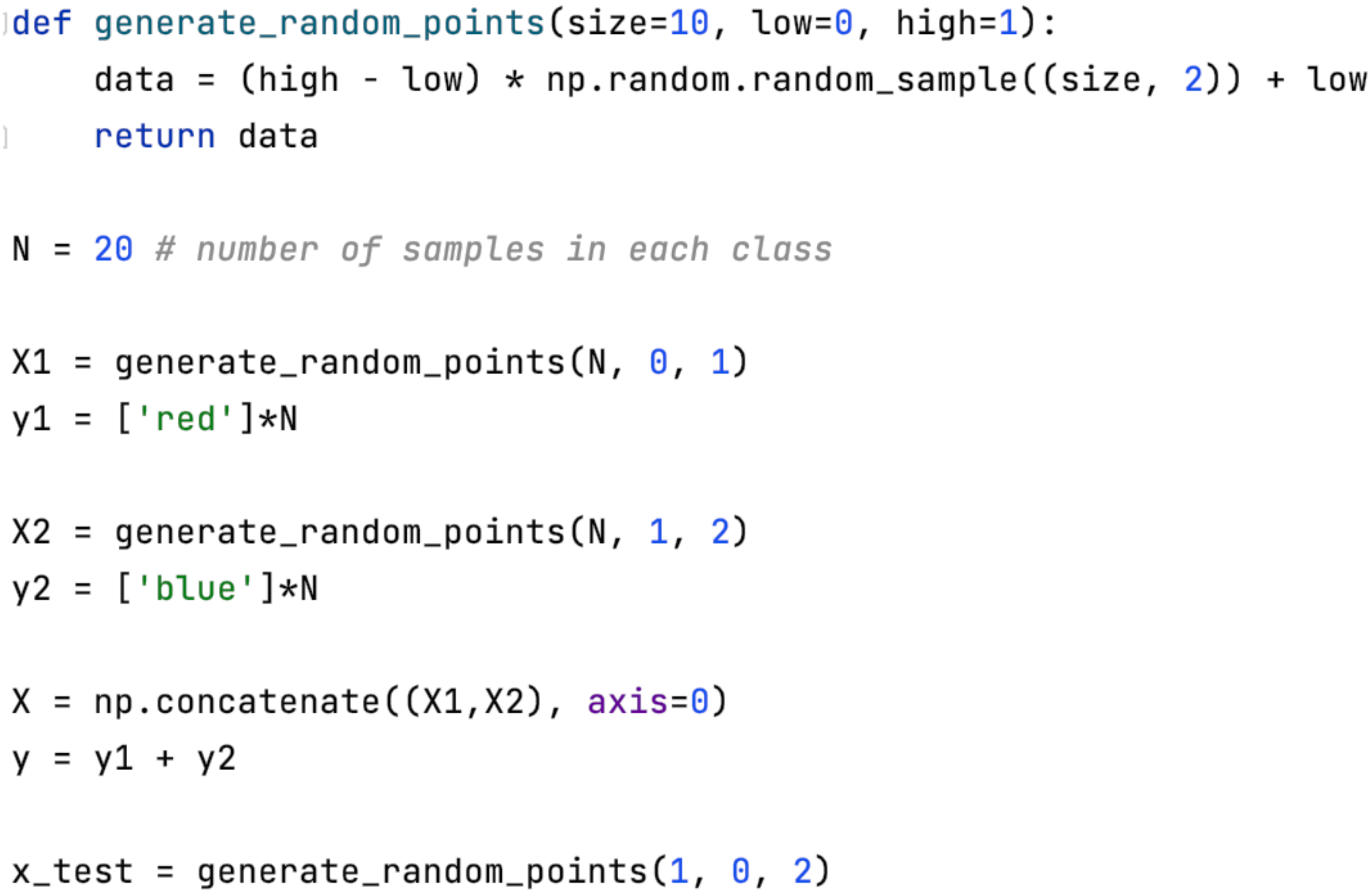
K nearest neighbors – Code Tutorial

Instructor: Wei Tang

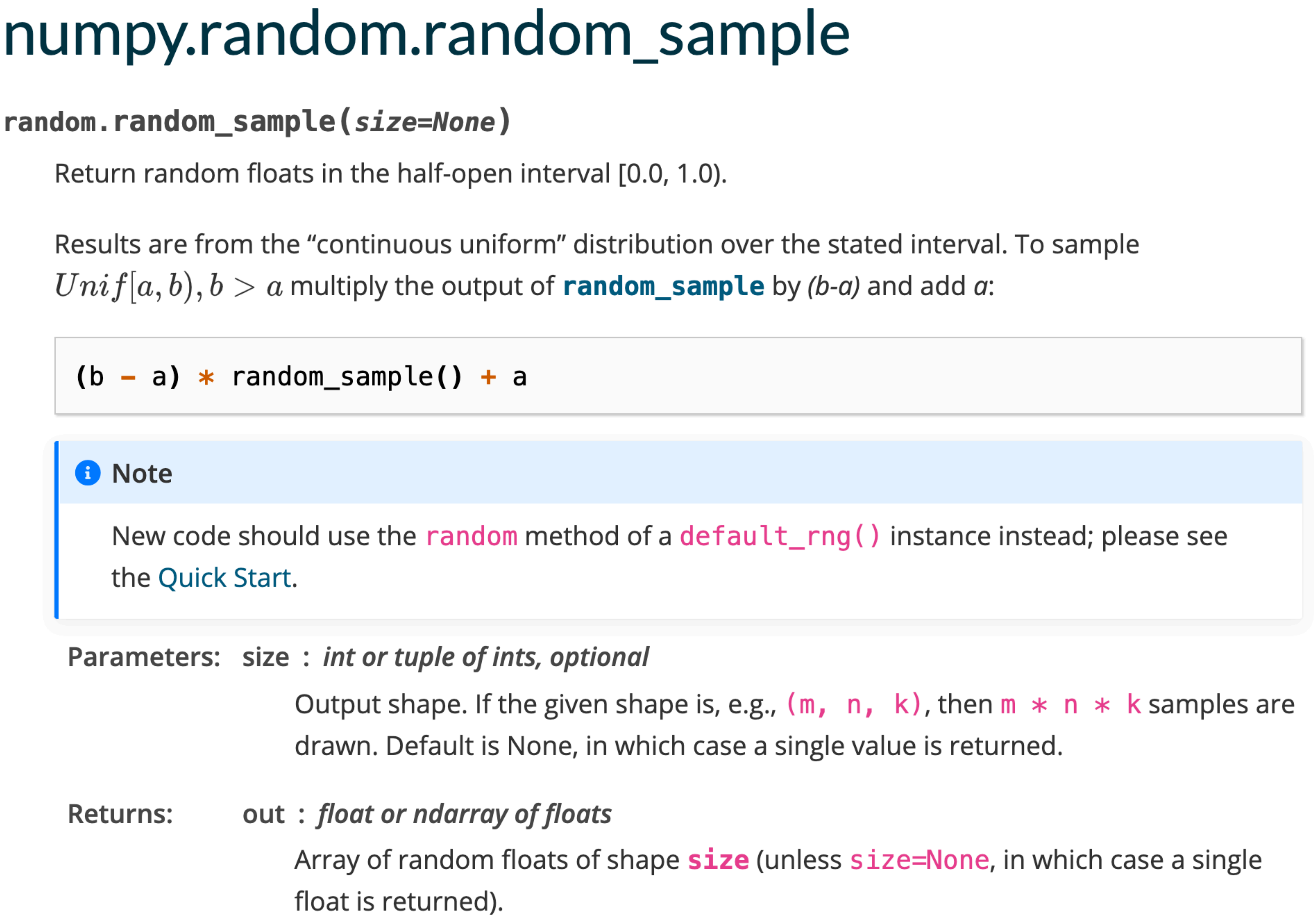
Department of Computer Science
University of Illinois at Chicago
Chicago IL 60607

https://tangw.people.uic.edu
tangw@uic.edu

Data



```python
def generate_random_points(size=10, low=0, high=1):
    data = (high - low) * np.random.random_sample((size, 2)) + low
    return data

N = 20 # number of samples in each class

X1 = generate_random_points(N, 0, 1)
y1 = ['red']*N

X2 = generate_random_points(N, 1, 2)
y2 = ['blue']*N

X = np.concatenate((X1,X2), axis=0)
y = y1 + y2

x_test = generate_random_points(1, 0, 2)
```



numpy.random.random_sample

random.random_sample(*size=None*)

Return random floats in the half-open interval [0.0, 1.0).

Results are from the "continuous uniform" distribution over the stated interval. To sample $Unif[a, b), b > a$ multiply the output of **random_sample** by *(b-a)* and add *a*:

```
(b - a) * random_sample() + a
```

ℹ Note

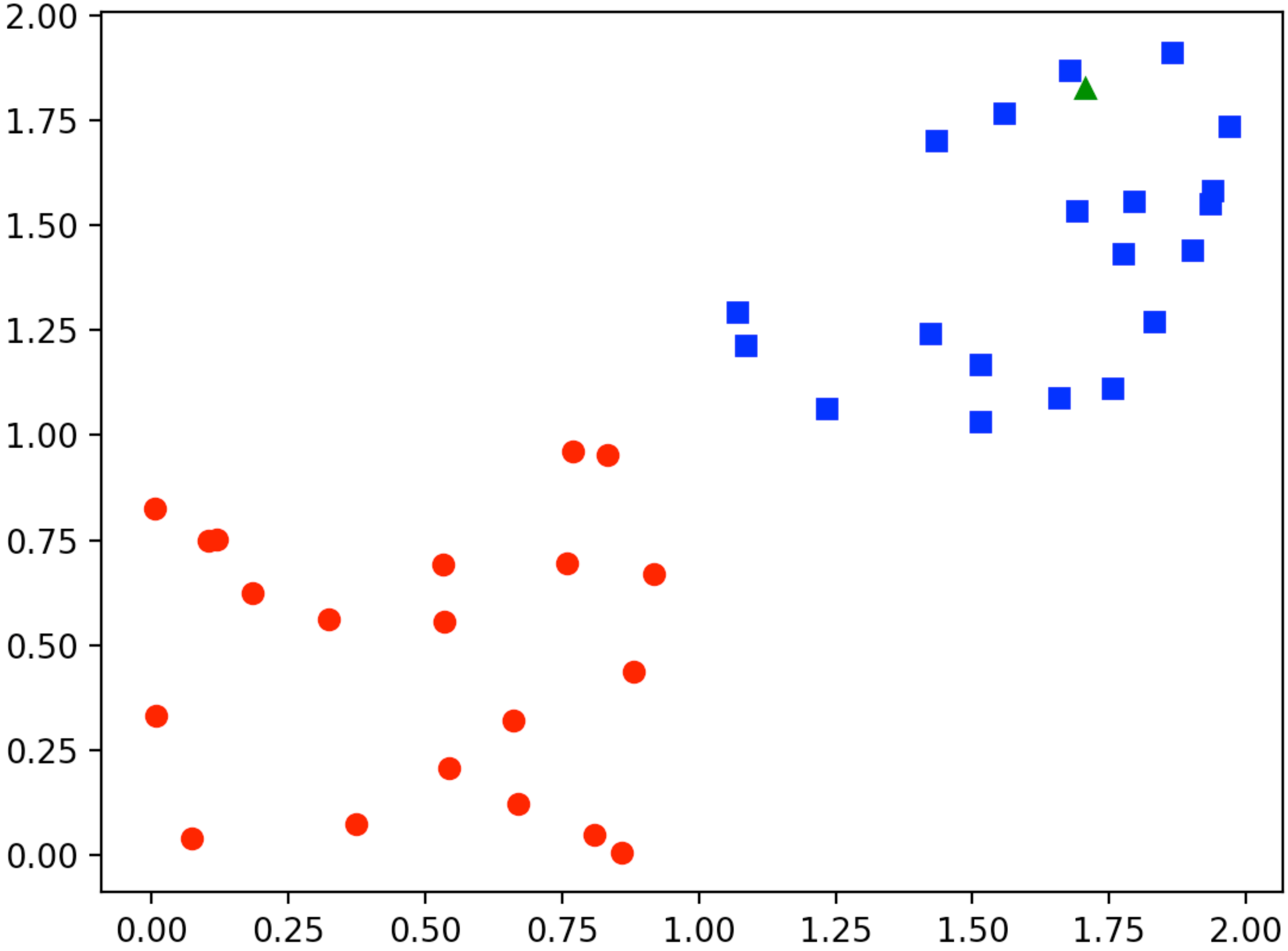
New code should use the `random` method of a `default_rng()` instance instead; please see the Quick Start.

Parameters: **size** : *int or tuple of ints, optional*

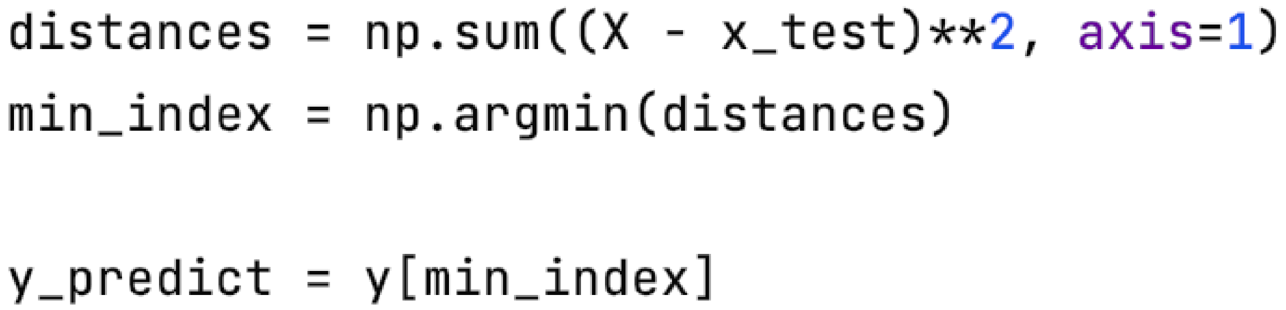
Output shape. If the given shape is, e.g., `(m, n, k)`, then `m * n * k` samples are drawn. Default is None, in which case a single value is returned.

Returns: **out** : *float or ndarray of floats*

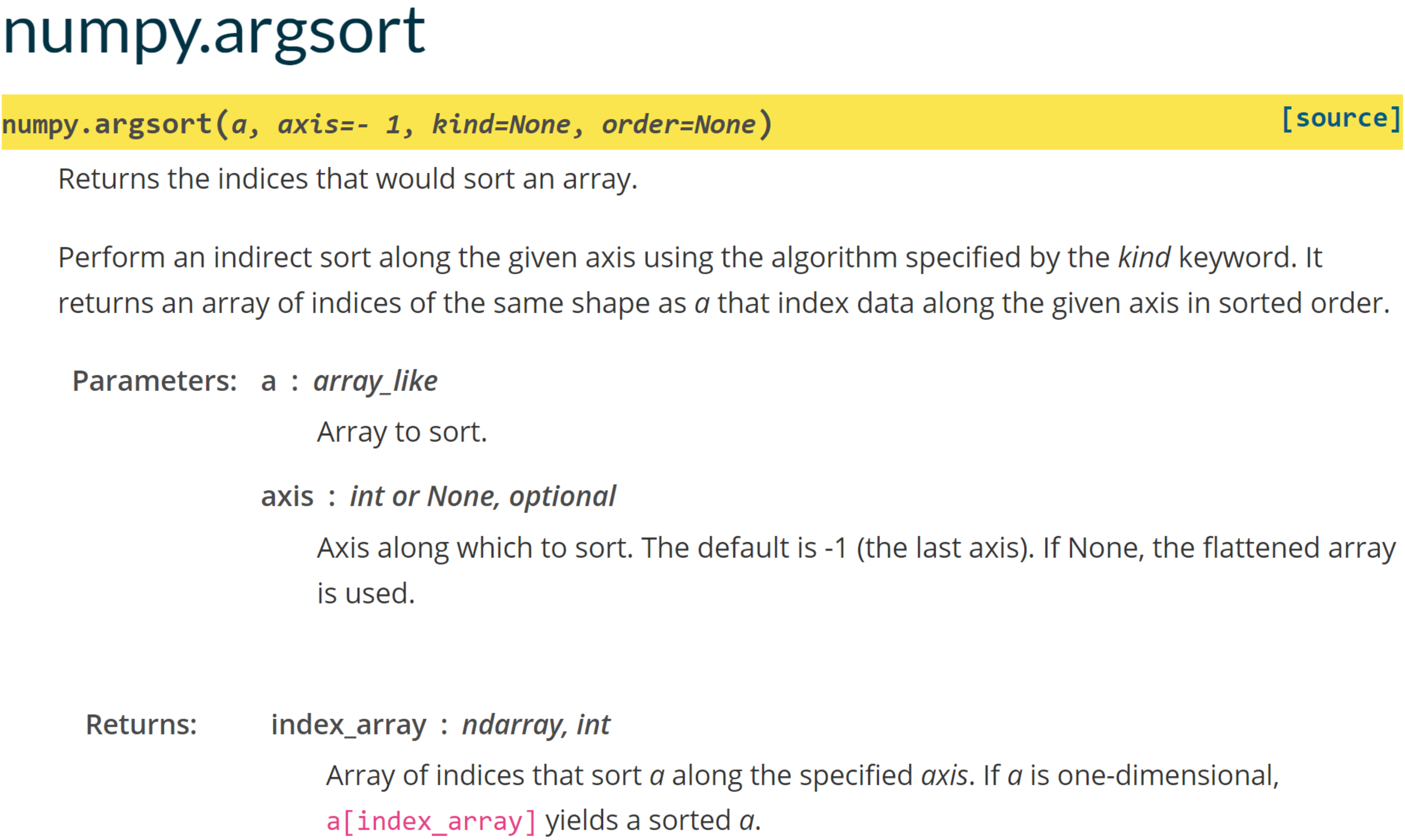
Array of random floats of shape **`size`** (unless `size=None`, in which case a single float is returned).


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Prediction (1 nearest neighbor)



```
distances = np.sum((X - x_test)**2, axis=1)
min_index = np.argmin(distances)

y_predict = y[min_index]
```



numpy.argsort

`numpy.argsort(a, axis=- 1, kind=None, order=None)` [source]

Returns the indices that would sort an array.

Perform an indirect sort along the given axis using the algorithm specified by the *kind* keyword. It returns an array of indices of the same shape as *a* that index data along the given axis in sorted order.

Parameters: **a** : *array_like*

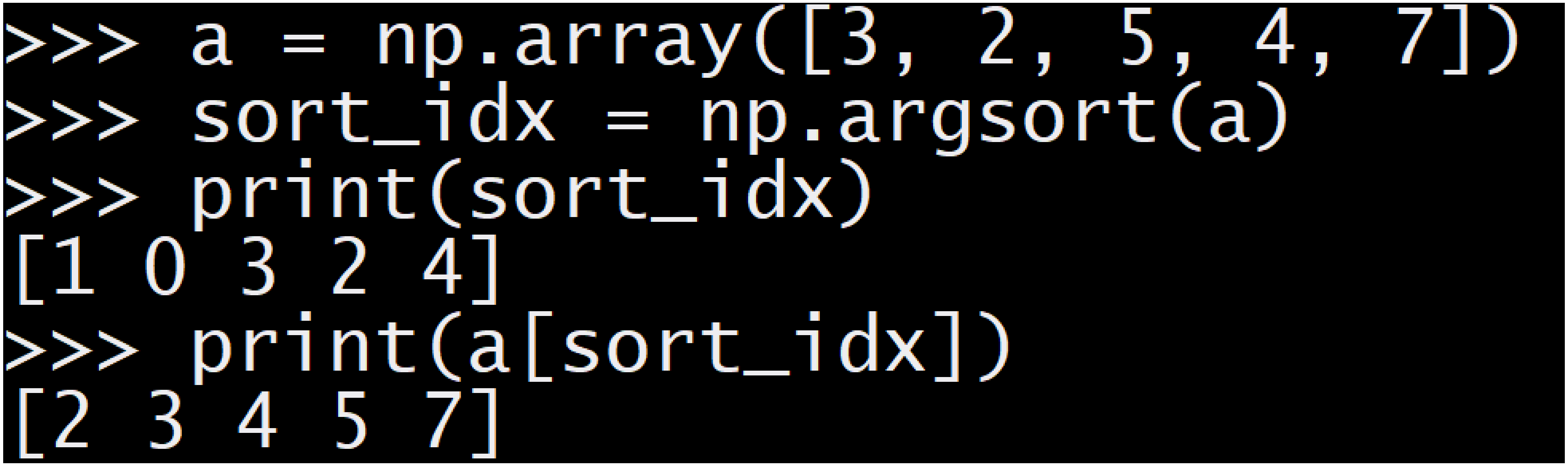
Array to sort.

axis : *int or None, optional*

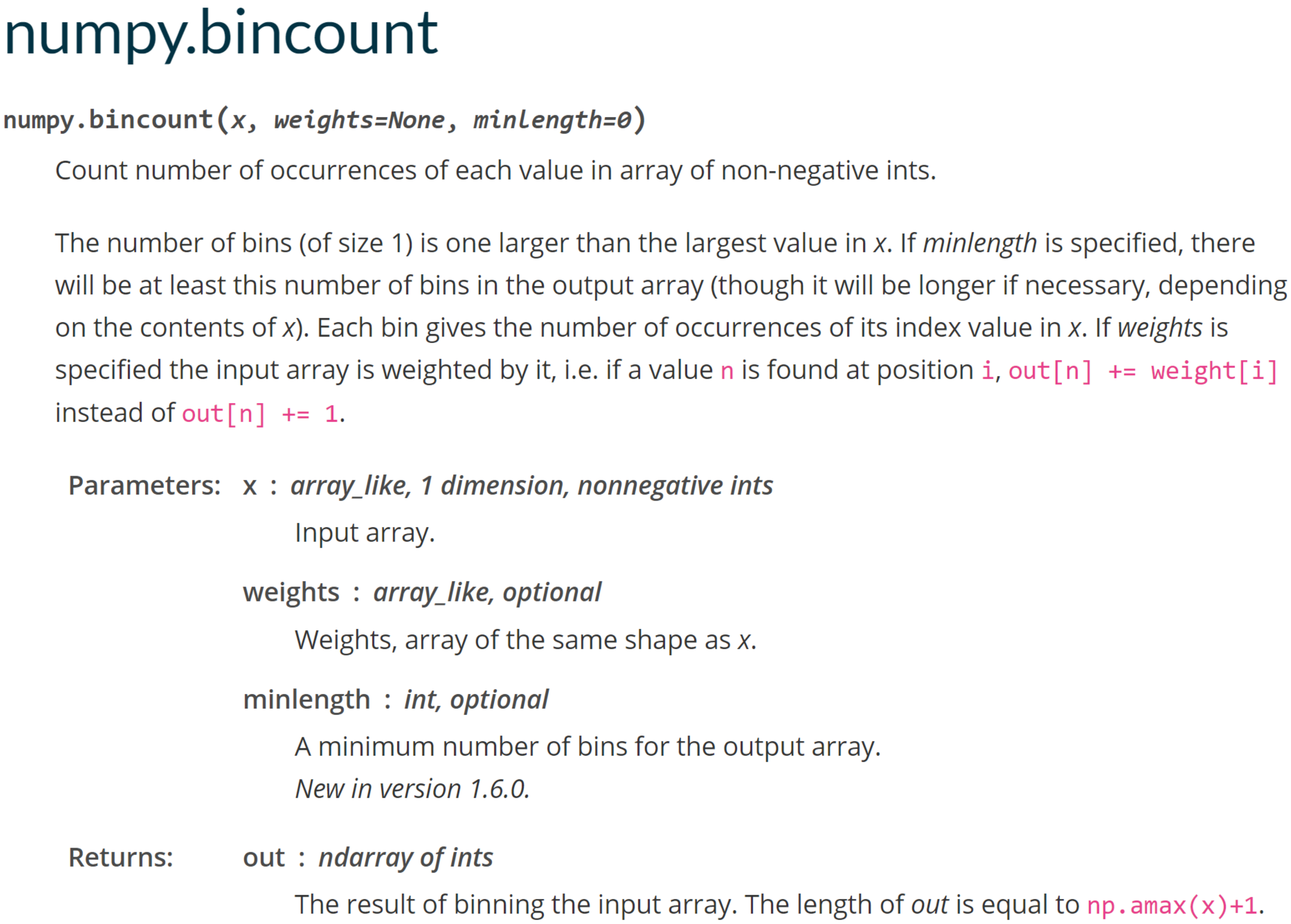
Axis along which to sort. The default is -1 (the last axis). If None, the flattened array is used.

Returns: **index_array** : *ndarray, int*

Array of indices that sort *a* along the specified *axis*. If *a* is one-dimensional, `a[index_array]` yields a sorted *a*.



```
>>> a = np.array([3, 2, 5, 4, 7])
>>> sort_idx = np.argsort(a)
>>> print(sort_idx)
[1 0 3 2 4]
>>> print(a[sort_idx])
[2 3 4 5 7]
```



numpy.bincount

`numpy.bincount(x, weights=None, minlength=0)`

Count number of occurrences of each value in array of non-negative ints.

The number of bins (of size 1) is one larger than the largest value in *x*. If *minlength* is specified, there will be at least this number of bins in the output array (though it will be longer if necessary, depending on the contents of *x*). Each bin gives the number of occurrences of its index value in *x*. If *weights* is specified the input array is weighted by it, i.e. if a value `n` is found at position `i`, `out[n] += weight[i]` instead of `out[n] += 1`.

Parameters:

x : *array_like, 1 dimension, nonnegative ints*

Input array.

weights : *array_like, optional*

Weights, array of the same shape as *x*.

minlength : *int, optional*

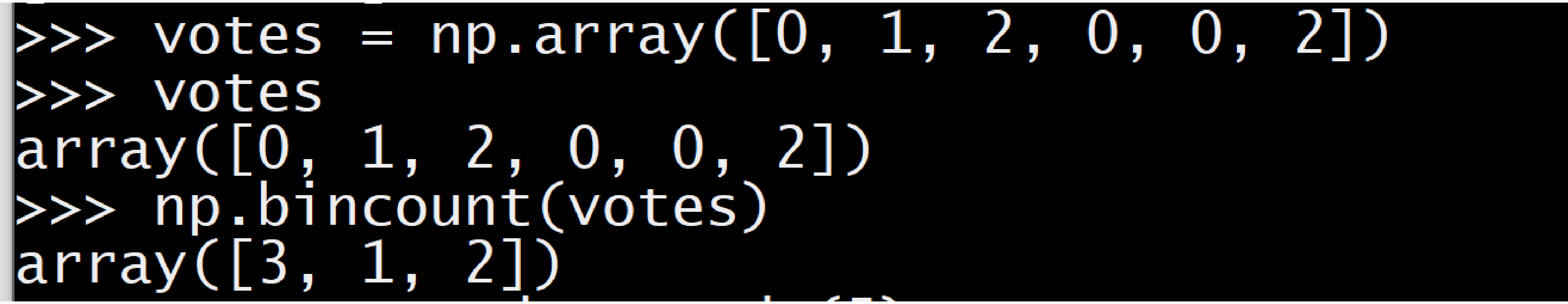
A minimum number of bins for the output array.

New in version 1.6.0.

Returns:

out : *ndarray of ints*

The result of binning the input array. The length of *out* is equal to `np.amax(x)+1`.



```
>>> votes = np.array([0, 1, 2, 0, 0, 2])
>>> votes
array([0, 1, 2, 0, 0, 2])
>>> np.bincount(votes)
array([3, 1, 2])
```