Aggregations with NumPy Arrays

NumPy provides provides various options for calculating summary statistics like min, max, sum, average, standard deviation, median etc.

Sum operations comparison

Lets compare built-in sum() with np.sum() for a big array.

In [1]:

```
import numpy as np
```

In [10]:

```
nparray = np.random.random(100)  #create np array of 100 random values
print("type : ", type(nparray))
sum(nparray)   #apply sum on it
```

```
type :  <class 'numpy.ndarray'>
```

Out[10]:

```
48.79367607670253
```

In [14]:

```
np.sum(nparray)  # apply np.sum() on array
```

Out[14]:

```
48.79367607670251
```

In [12]:

```
%timeit sum(nparray)
```

```
25.1 µs ± 1.31 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

In [13]:

```
%timeit np.sum(nparray)
```

```
6.45 µs ± 291 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

np.sum() performs much better than the built-in sum() function.

Finding mimimum and maximum

In [15]:

```
nparray
```

Out[15]:

```
array([0.93590421, 0.64662614, 0.78043065, 0.3195217 , 0.09856843,
       0.07461706, 0.53835071, 0.38534825, 0.11614307, 0.48195876,
       0.26809185, 0.07647077, 0.70845964, 0.24385294, 0.08436315,
       0.54485213, 0.99366136, 0.30353153, 0.06906205, 0.21052785,
       0.09705089, 0.35816377, 0.36192618, 0.41553726, 0.74889155,
       0.91036602, 0.3865395 , 0.74393723, 0.7767306 , 0.64918206,
       0.4911143 , 0.8249897 , 0.49765776, 0.03279451, 0.71130514,
       0.90291026, 0.09869709, 0.67953225, 0.65387181, 0.84521436,
       0.40202118, 0.75429394, 0.28321498, 0.53004401, 0.72722108,
       0.81737237, 0.8150976 , 0.67350288, 0.06019803, 0.30967881,
       0.46892584, 0.31767079, 0.80798928, 0.92225873, 0.05661287,
       0.87771533, 0.17993659, 0.00140134, 0.49407007, 0.47412474,
       0.21991786, 0.55340462, 0.45036666, 0.55994158, 0.16525676,
       0.92881969, 0.15780931, 0.47306187, 0.51940651, 0.23006248,
       0.7594677 , 0.19150656, 0.27603523, 0.88736462, 0.58580155,
       0.89731996, 0.94062664, 0.34721919, 0.96895167, 0.74420363,
       0.68417724, 0.49073562, 0.34878424, 0.59774055, 0.36745404,
       0.03770034, 0.56450904, 0.50083834, 0.0658192 , 0.84091142,
       0.64608962, 0.5329572 , 0.29416018, 0.05841722, 0.13096685,
       0.52031186, 0.36470414, 0.8362015 , 0.16883867, 0.84570977])
```

In [16]:

```
np.min(nparray)
```

Out[16]:

```
0.0014013386411428908
```

In [17]:

```
np.max(nparray)
```

Out[17]:

```
0.9936613569031882
```

Other way to compute min and max using the array variable is as follows

In [18]:

```
nparray.min()
```

Out[18]:

```
0.0014013386411428908
```

In [19]:

```
nparray.max()
```

Out[19]:

```
0.9936613569031882
```

Multidimensional aggregates

Computing aggregates over the complete array

In [22]:

```
matrix = np.random.random((4, 4))
matrix
```

Out[22]:

```
array([[0.2528745 , 0.00924563, 0.39971661, 0.15279878],
       [0.34349295, 0.98516776, 0.54947196, 0.01427731],
       [0.87725539, 0.21269655, 0.23198681, 0.77600508],
       [0.27561743, 0.04650408, 0.86642111, 0.71214884]])
```

In [23]:

```
matrix.sum()
```

Out[23]:

```
6.705680792203458
```

In [26]:

```
np.sum(matrix)
```

Out[26]:

```
6.705680792203458
```

In [24]:

```
matrix.min()
```

Out[24]:

```
0.009245632402310133
```

In [28]:

```
np.min(matrix)
```

Out[28]:

```
0.009245632402310133
```

In [25]:

```
matrix.max()
```

Out[25]:

```
0.9851677625173139
```

In [27]:

```
np.max(matrix)
```

Out[27]:

```
0.9851677625173139
```

Computing agggregates within each Column

In [29]:

```
matrix
```

Out[29]:

```
array([[0.2528745 , 0.00924563, 0.39971661, 0.15279878],
       [0.34349295, 0.98516776, 0.54947196, 0.01427731],
       [0.87725539, 0.21269655, 0.23198681, 0.77600508],
       [0.27561743, 0.04650408, 0.86642111, 0.71214884]])
```

In [30]:

```
matrix.sum(axis=0)
```

Out[30]:

```
array([1.74924027, 1.25361402, 2.0475965 , 1.65523001])
```

In [33]:

```
np.sum(matrix, axis=0)
```

Out[33]:

```
array([1.74924027, 1.25361402, 2.0475965 , 1.65523001])
```

In [31]:

```
matrix.max(axis=0)
```

Out[31]:

```
array([0.87725539, 0.98516776, 0.86642111, 0.77600508])
```

In [34]:

```
np.max(matrix, axis=0)
```

Out[34]:

```
array([0.87725539, 0.98516776, 0.86642111, 0.77600508])
```

In [32]:

```
matrix.min(axis=0)
```

Out[32]:

```
array([0.2528745 , 0.00924563, 0.23198681, 0.01427731])
```

In [35]:

```
np.min(matrix, axis=0)
```

Out[35]:

```
array([0.2528745 , 0.00924563, 0.23198681, 0.01427731])
```

Computing agggregates within each Row

In [36]:

```
matrix
```

Out[36]:

```
array([[0.2528745 , 0.00924563, 0.39971661, 0.15279878],
       [0.34349295, 0.98516776, 0.54947196, 0.01427731],
       [0.87725539, 0.21269655, 0.23198681, 0.77600508],
       [0.27561743, 0.04650408, 0.86642111, 0.71214884]])
```

In [37]:

```
matrix.sum(axis=1)
```

Out[37]:

```
array([0.81463551, 1.89240998, 2.09794384, 1.90069146])
```

In [38]:

```
np.sum(matrix, axis=1)
```

Out[38]:

```
array([0.81463551, 1.89240998, 2.09794384, 1.90069146])
```

In [39]:

```
matrix.min(axis=1)
```

Out[39]:

```
array([0.00924563, 0.01427731, 0.21269655, 0.04650408])
```

In [40]:

```
np.min(matrix, axis=1)
```

Out[40]:

```
array([0.00924563, 0.01427731, 0.21269655, 0.04650408])
```

In [41]:

```
matrix.max(axis=1)
```

Out[41]:

```
array([0.39971661, 0.98516776, 0.87725539, 0.86642111])
```

In [42]:

```
np.max(matrix, axis=1)
```

Out[42]:

```
array([0.39971661, 0.98516776, 0.87725539, 0.86642111])
```

Other aggregate functions

In [43]:

```
matrix
```

Out[43]:

```
array([[0.2528745 , 0.00924563, 0.39971661, 0.15279878],
       [0.34349295, 0.98516776, 0.54947196, 0.01427731],
       [0.87725539, 0.21269655, 0.23198681, 0.77600508],
       [0.27561743, 0.04650408, 0.86642111, 0.71214884]])
```

In [44]:

```
matrix.prod()  # compute product of all values
```

Out[44]:

```
1.0070370563356953e-10
```

In [45]:

```
matrix.mean()  # compute average of all values
```

Out[45]:

```
0.4191050495127161
```

In [47]:

```
matrix.std() # standard deviation
```

Out[47]:

```
0.31959164166804704
```

In [48]:

```
matrix.var()  # variance
```

Out[48]:

```
0.10213881742407738
```

NaN -safe aggregate functions

In [53]:

```
x = np.array([1, 5, 3, 4, 5])
x
```

Out[53]:

```
array([1, 5, 3, 4, 5])
```

In [54]:

```
np.nansum(x)  # NaN safe sum , ignore missing values
```

Out[54]:

```
18
```

In [55]:

```
np.nanmax(x) # NaN safe max , ignore missing values
```

Out[55]:

```
5
```

In [56]:

```
np.nanmin(x) # NaN safe min , ignore missing values
```

Out[56]:

```
1
```

In [57]:

```
np.nanstd(x) # NaN safe std , ignore missing values
```

Out[57]:

```
1.4966629547095764
```