# Aggregations: Min, Max, and Everything In Between

Often when faced with a large amount of data, a first step is to compute summary statistics for the data in question. Perhaps the most common summary statistics are the mean and standard deviation, which allow you to summarize the "typical" values in a dataset, but other aggregates are useful as well (the sum, product, median, minimum and maximum, quantiles, etc.).

NumPy has fast built-in aggregation functions for working on arrays; we'll discuss and demonstrate some of them here.

## Summing the Values in an Array

As a quick example, consider computing the sum of all values in an array. Python itself can do this using the built-in sum function:

import numpy as np

L = np.random.random(100)  
sum(L)

55.61209116604941

The syntax is quite similar to that of NumPy's sum function, and the result is the same in the simplest case:

np.sum(L)

55.612091166049424

However, because it executes the operation in compiled code, NumPy's version of the operation is computed much more quickly:

big\_array = np.random.rand(1000000)  
%timeit sum(big\_array)  
%timeit np.sum(big\_array)

10 loops, best of 3: 104 ms per loop  
1000 loops, best of 3: 442 µs per loop

Be careful, though: the sum function and the np.sum function are not identical, which can sometimes lead to confusion! In particular, their optional arguments have different meanings, and np.sum is aware of multiple array dimensions, as we will see in the following section.

## Minimum and Maximum

Similarly, Python has built-in min and max functions, used to find the minimum value and maximum value of any given array:

min(big\_array), max(big\_array)

(1.1717128136634614e-06, 0.9999976784968716)

NumPy's corresponding functions have similar syntax, and again operate much more quickly:

np.min(big\_array), np.max(big\_array)

(1.1717128136634614e-06, 0.9999976784968716)

%timeit min(big\_array)  
%timeit np.min(big\_array)

10 loops, best of 3: 82.3 ms per loop  
1000 loops, best of 3: 497 µs per loop

For min, max, sum, and several other NumPy aggregates, a shorter syntax is to use methods of the array object itself:

print(big\_array.min(), big\_array.max(), big\_array.sum())

1.17171281366e-06 0.999997678497 499911.628197

Whenever possible, make sure that you are using the NumPy version of these aggregates when operating on NumPy arrays!

### Multi dimensional aggregates

One common type of aggregation operation is an aggregate along a row or column. Say you have some data stored in a two-dimensional array:

M = np.random.random((3, 4))  
print(M)

[[ 0.8967576 0.03783739 0.75952519 0.06682827]  
 [ 0.8354065 0.99196818 0.19544769 0.43447084]  
 [ 0.66859307 0.15038721 0.37911423 0.6687194 ]]

By default, each NumPy aggregation function will return the aggregate over the entire array:

M.sum()

6.0850555667307118

Aggregation functions take an additional argument specifying the *axis* along which the aggregate is computed. For example, we can find the minimum value within each column by specifying axis=0:

M.min(axis=0)

array([ 0.66859307, 0.03783739, 0.19544769, 0.06682827])

The function returns four values, corresponding to the four columns of numbers.

Similarly, we can find the maximum value within each row:

M.max(axis=1)

array([ 0.8967576 , 0.99196818, 0.6687194 ])

The axis keyword specifies the *dimension of the array that will be collapsed*, rather than the dimension that will be returned. So specifying axis=0 means that the first axis will be collapsed: for two-dimensional arrays, this means that values within each column will be aggregated.

### Other aggregation functions

Additionally, most aggregates have a NaN-safe counterpart that computes the result while ignoring missing values, which are marked by the special IEEE floating-point NaN value. Some of these NaN-safe functions were not added until NumPy 1.8, so they will not be available in older NumPy versions.

The following table provides a list of useful aggregation functions available in NumPy:

|  |  |  |
| --- | --- | --- |
| Function Name | NaN-safe Version | Description |
| np.sum | np.nansum | Compute sum of elements |
| np.prod | np.nanprod | Compute product of elements |
| np.mean | np.nanmean | Compute mean of elements |
| np.std | np.nanstd | Compute standard deviation |
| np.var | np.nanvar | Compute variance |
| np.min | np.nanmin | Find minimum value |
| np.max | np.nanmax | Find maximum value |
| np.argmin | np.nanargmin | Find index of minimum value |
| np.argmax | np.nanargmax | Find index of maximum value |
| np.median | np.nanmedian | Compute median of elements |
| np.percentile | np.nanpercentile | Compute rank-based statistics of elements |
| np.any | N/A | Evaluate whether any elements are true |
| np.all | N/A | Evaluate whether all elements are true |