In [1]:

```
# The standard deviation of a NumPy array can be found via the std() method.
# Here is an example:

import numpy as np
my_array = np.array([2,4,8,10,12])
print(my_array)
print("std value:")
print(np.std(my_array))
```

```
[ 2  4  8 10 12]
std value:
3.7094473981982814
```

In [2]:

```
# To find the standard deviation across rows and columns in a two-dimensional
# NumPy array, you need to call the std() method. The value of 1 for the axis
# attribute returns the minimum list of minimum values across all columns,
# whereas a value of 0 returns minimum values across all rows.

import numpy as np
my_array = np.random.randint(1,20, size = (3,4))
print(my_array)
print("std-dev:")
print(np.std(my_array, axis = 1))
print(np.std(my_array, axis = 0))
```

```
[[ 3  4 13 16]
 [19  9 19  5]
 [19 17 13 16]]
std-dev:
[5.61248608 6.164414   2.16506351]
[7.54247233 5.35412613 2.82842712 5.18544973]
```

In [5]:

```
# Finally, to find correlations, you can use the correlation() method and pass it
# the two NumPy arrays between which you want to find the correlation.

import numpy as np
a1 = np.array([1, 3, 0, 0.9, 1.2])
a2 = np.array([-1, 0.5, 0.2, 0.6, 5])
print(a1)
print(a2)
print("correlation value:")
print(np.correlate(a1, a2))
```

```
[1.  3.  0.  0.9 1.2]
[-1.   0.5  0.2  0.6  5. ]
correlation value:
[7.04]
```

In [6]:

```
# Oftentimes, you would need to find unique values within a NumPy array and
# to find the count of every unique value in a NumPy array.
# To find unique values in a NumPy array, you need to pass the array to the
# unique() method from the NumPy module. Here is an example:

import numpy as np
my_array =np.array([5,8,7,5,9,3,7,7,1,1,8,4,6,9,7,3])
unique_items = np.unique(my_array)
print(unique_items)
```

```
[1 3 4 5 6 7 8 9]
```

In [7]:

```
# To find the count of every unique item in a NumPy array, you need to pass
# the array to the unique() method and pass True as the value for the
# return_counts attribute. The unique() method in this case returns a tuple,
# which contains a list of all unique items and a corresponding list of counts for
# each unique item. Here is an example:

import numpy as np
my_array =np.array([5,8,7,5,9,3,7,7,1,1,8,4,6,9,7,3])
unique_items, counts = np.unique(my_array, return_counts=True)
print(unique_items)
print(counts)
```

```
[1 3 4 5 6 7 8 9]
[2 2 1 2 1 4 2 2]
```

In [8]:

```
# One way to get the count value against every unique item is to create an array
# of arrays where the first item is the list of unique items and the second item is
# the list of counts, as shown in the script below:

import numpy as np
my_array =np.array([5,8,7,5,9,3,7,7,1,1,8,4,6,9,7,3])
unique_items, counts = np.unique(my_array, return_counts=True)
print(unique_items)
print(counts)
frequencies = np.asarray((unique_items, counts))
print(frequencies)
```

```
[1 3 4 5 6 7 8 9]
[2 2 1 2 1 4 2 2]
[[1 3 4 5 6 7 8 9]
 [2 2 1 2 1 4 2 2]]
```

In [10]:

```
# Next, you can transpose the array that contains the list of unique items and
# their counts. In the output, you will get an array where each item is a list. The
# first item in the list is the unique item itself, while the second is the count of
# the item. For instance, in the output of the script below, you can see that item
# 1 occurs twice in the input array, item 3 also occurs twice, and so on.

import numpy as np
my_array =np.array([5,8,7,5,9,3,7,7,1,1,8,4,6,9,7,3])
unique_items, counts = np.unique(my_array, return_counts=True)
print(unique_items)
print(counts)
frequencies = np.asarray((unique_items, counts)).T
print(frequencies)
```

```
[1 3 4 5 6 7 8 9]
[2 2 1 2 1 4 2 2]
[[1 2]
 [3 2]
 [4 1]
 [5 2]
 [6 1]
 [7 4]
 [8 2]
 [9 2]]
```

In [11]:

```
# There are two main methods to reverse a NumPy array: flipud() and fliplr().
# The flipud() method is used to reverse a one-dimensional NumPy array, as
# shown in the script below:

import numpy as np
print("original")
my_array = np.random.randint(1,20,10)
print(my_array)
print("reversed")
reversed_array = np.flipud(my_array)
print(reversed_array)
```

```
original
[14 16  9  5  9 13  2 13 13  8]
reversed
[ 8 13 13  2 13  9  5  9 16 14]
```

In [12]:

```
# On the other hand, to reverse a two-dimensional NumPy array, you can use
# the fliplr() method, as shown below:

import numpy as np
print("original")
my_array = np.random.randint(1,20, size = (3,4))
print(my_array)
print("reversed")
reversed_array = np.fliplr(my_array)
print(reversed_array)
```

```
original
[[13 10  1 11]
 [11 10 10  8]
 [13 11 17 17]]
reversed
[[11  1 10 13]
 [ 8 10 10 11]
 [17 17 11 13]]
```