Import NumPy

In [1]:
```
import numpy as np
```

Creating Arrays

Create a 1-Dimesnional numpy array

In [2]:
```
score=np.array([70,63,81,58,76])
print(score)
```

```
[70 63 81 58 76]
```

Create a 4*5 2-D array with elements from 0 to 19 sorted ascending

In [3]:
```
numbers_array=np.arange(20).reshape(4,5)
print(numbers_array)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

In [4]:
```
array_1=np.arange(20).reshape(4,5)
print("Array 1 \n ",array_1)

array_2=np.arange(20).reshape(4,5)*5
print("\nArray 2 \n",array_2)
```

```
Array 1
  [[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]

Array 2
 [[ 0  5 10 15 20]
 [25 30 35 40 45]
 [50 55 60 65 70]
 [75 80 85 90 95]]
```

Create a 2*5 2-Dimesnional numpy array

In [5]:
```
score_height=np.array([([70,63,81,58,76]),(5.1,4.5,6.0,5.3,5.5)])
print(score_height)
```

```
[[70.  63.  81.  58.  76. ]
 [ 5.1  4.5  6.   5.3  5.5]]
```

Create a 4*5 2-Dimesnional 0s numpy array

In [6]:
```
zero=np.zeros((4,5),dtype=np.int16)
print(zero)
```

```
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

Create a 4*5 2-Dimesnional 1s numpy array

In [7]:
```
one=np.ones((4,5),dtype=np.int16)
print(one)
```

```
[[1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]]
```

Create a 5*5 identity matrix

In [8]:
```
identity=np.eye(5,dtype=np.int16)
print(identity)
```

```
[[1 0 0 0 0]
 [0 1 0 0 0]
 [0 0 1 0 0]
 [0 0 0 1 0]
 [0 0 0 0 1]]
```

Inspecting Arrays

Find the number of elements in an array

In [9]:
```
print(score_height)
print(score_height.size)
```

```
[[70.  63.  81.  58.  76. ]
 [ 5.1  4.5  6.   5.3  5.5]]
10
```

Find the dimensions of an array

In [10]:
```
print(score_height.shape)
```

```
(2, 5)
```

Find the data type of elements in the array

In [11]:
```
print(score_height.dtype)
```

```
float64
```

Convert array to Python list

In [12]:
```
print("NumPy array : \n ",score_height)
print("\nPython list : \n",score_height.tolist())
```

```
NumPy array :
  [[70.  63.  81.  58.  76. ]
 [ 5.1  4.5  6.   5.3  5.5]]

Python list :
 [[70.0, 63.0, 81.0, 58.0, 76.0], [5.1, 4.5, 6.0, 5.3, 5.5]]
```

Adding Elements to an array

Append new elements at the end of an array

In [13]:
```
score=np.append(score,90)
print(score)
score=np.append(score,[45,59,94])
print(score)
```

```
[70 63 81 58 76 90]
[70 63 81 58 76 90 45 59 94]
```

Insert new element(s) at a specific position in an array

In [14]:
```
score=np.insert(score,3,20)
print(score)
```

```
[70 63 81 20 58 76 90 45 59 94]
```

Removing Elements from an array

Delete row on index 2 from an array

In [15]:
```
print(score)
score=np.delete(score,2,axis=0)
print("\nRemoved 76 from the original array \n",score)
```

```
[70 63 81 20 58 76 90 45 59 94]

Removed 76 from the original array
 [70 63 20 58 76 90 45 59 94]
```

Delete column on index 2 from an array

In [16]:
```
print(score_height)
score_height=np.delete(score_height,2,axis=1)
print("\nRemove element at index 2 \n",score_height)
```

```
[[70.  63.  81.  58.  76. ]
 [ 5.1  4.5  6.   5.3  5.5]]

Remove element at index 2
 [[70.  63.  58.  76. ]
 [ 5.1  4.5  5.3  5.5]]
```

Copying Array

Copy array

In [17]:
```
new_score_height=np.copy(score_height)
print(new_score_height)
```

```
[[70.  63.  58.  76. ]
 [ 5.1  4.5  5.3  5.5]]
```

Sorting Array

Sort array

In [18]:
```
print("Original Array \n", score_height)
print("\nRowise sorted Array \n", score_height.sort(axis=1))
print("\nColumnwise sorted Array \n", score_height.sort(axis=0))
print("\n sorted Array \n", score_height.sort())
```

```
Original Array
 [[70.  63.  58.  76. ]
 [ 5.1  4.5  5.3  5.5]]

Rowise sorted Array
 None

Columnwise sorted Array
 None

 sorted Array
 None
```

Reshaping Array

Resize the array

In [19]:
```
resized_score_height=score_height.resize((5,8))
print(resized_score_height)
```

```
None
```

Reshape the array

In [20]:
```
reshape_score_height=score_height.reshape(4,10)
print(reshape_score_height)
```

```
[[ 4.5  5.1  5.3  5.5 58.  63.  70.  76.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0.   0.   0. ]]
```

Flattens 2-D array to 1-D array

In [21]:
```
flattened_score_height=score_height.flatten()
print(flattened_score_height)
```

```
[ 4.5  5.1  5.3  5.5 58.  63.  70.  76.   0.   0.   0.   0.   0.   0.
  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. ]
```

Transposes Array

In [22]:
```
transposed_score_height=score_height.T
print(score_height)
```

```
[[ 4.5  5.1  5.3  5.5 58.  63.  70.  76. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]]
```

Combining Arrays

Concatenate arrays along specified axis

In [23]:
```
print(np.concatenate((array_1,array_2),axis=0))
print("\n")
print(np.concatenate((array_1,array_2),axis=1))
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [ 0  5 10 15 20]
 [25 30 35 40 45]
 [50 55 60 65 70]
 [75 80 85 90 95]]


[[ 0  1  2  3  4  0  5 10 15 20]
 [ 5  6  7  8  9 25 30 35 40 45]
 [10 11 12 13 14 50 55 60 65 70]
 [15 16 17 18 19 75 80 85 90 95]]
```

Scalar Math Functions

Add 2 to each element in array

In [24]:
```
print(np.add(score_height,2))
```

```
[[ 6.5  7.1  7.3  7.5 60.  65.  72.  78. ]
 [ 2.   2.   2.   2.   2.   2.   2.   2. ]
 [ 2.   2.   2.   2.   2.   2.   2.   2. ]
 [ 2.   2.   2.   2.   2.   2.   2.   2. ]
 [ 2.   2.   2.   2.   2.   2.   2.   2. ]]
```

Remove 2 to each element in array

In [25]:
```
print(np.subtract(score_height,2))
```

```
[[ 2.5  3.1  3.3  3.5 56.  61.  68.  74. ]
 [-2.  -2.  -2.  -2.  -2.  -2.  -2.  -2. ]
 [-2.  -2.  -2.  -2.  -2.  -2.  -2.  -2. ]
 [-2.  -2.  -2.  -2.  -2.  -2.  -2.  -2. ]
 [-2.  -2.  -2.  -2.  -2.  -2.  -2.  -2. ]]
```

Multiply each element in array by 2

In [26]:
```
print(np.multiply(score_height,2))
```

```
[[  9.   10.2  10.6  11.  116.  126.  140.  152. ]
 [  0.    0.    0.    0.    0.    0.    0.    0. ]
 [  0.    0.    0.    0.    0.    0.    0.    0. ]
 [  0.    0.    0.    0.    0.    0.    0.    0. ]
 [  0.    0.    0.    0.    0.    0.    0.    0. ]]
```

Divide each element in array by 2

In [27]:
```
print(np.divide(score_height,2))
```

```
[[ 2.25  2.55  2.65  2.75 29.   31.5  35.   38.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.  ]]
```

Raise each element in array to power 2

In [28]:
```
print(np.power(score_height,2))
```

```
[[  20.25   26.01   28.09   30.25 3364.   3969.   4900.   5776.  ]
 [   0.      0.      0.      0.      0.      0.      0.      0.  ]
 [   0.      0.      0.      0.      0.      0.      0.      0.  ]
 [   0.      0.      0.      0.      0.      0.      0.      0.  ]
 [   0.      0.      0.      0.      0.      0.      0.      0.  ]]
```

Return Square root of each element in the array

In [29]:
```
print(np.sqrt(score_height))
```

```
[[2.12132034 2.25831796 2.30217289 2.34520788 7.61577311 7.93725393
  8.36660027 8.71779789]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]]
```

Return Sine of each element in the array

In [30]:
```
print(np.sin(score_height))
```

```
[[-0.97753012 -0.92581468 -0.83226744 -0.70554033  0.99287265  0.1673557
   0.77389068  0.56610764]
 [ 0.          0.          0.          0.          0.          0.
   0.          0.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.          0.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.          0.        ]
 [ 0.          0.          0.          0.          0.          0.
   0.          0.        ]]
```

Return Natural log of each element in the array

In [31]:
```
print(np.log(score_height))
```

```
[[1.5040774  1.62924054 1.66770682 1.70474809 4.06044301 4.14313473
  4.24849524 4.33073334]
 [      -inf       -inf       -inf       -inf       -inf       -inf
        -inf       -inf]
 [      -inf       -inf       -inf       -inf       -inf       -inf
        -inf       -inf]
 [      -inf       -inf       -inf       -inf       -inf       -inf
        -inf       -inf]
 [      -inf       -inf       -inf       -inf       -inf       -inf
        -inf       -inf]]

C:\Users\soongaya\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarning: divide by zero encountered in
log
  """Entry point for launching an IPython kernel.
```

Return Absolute value of each element in the array

In [32]:
```
print(np.abs(score_height))
```

```
[[ 4.5  5.1  5.3  5.5 58.  63.  70.  76. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]]
```

Rounds up to the nearest whole number

In [33]:
```
print(np.ceil(score_height))
```

```
[[ 5.  6.  6.  6. 58. 63. 70. 76.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]]
```

Rounds down to the nearest whole number

In [34]:
```
print(np.floor(score_height))
```

```
[[ 4.  5.  5.  5. 58. 63. 70. 76.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]]
```

Rounds to the nearest whole number

In [35]:
```
print(np.round(score_height))
```

```
[[ 4.  5.  5.  6. 58. 63. 70. 76.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]]
```

Vector Math Functions

Elementwise Addition

In [36]:
```
print("\nArray_1 + Array_2")
print(np.add(array_1,array_2))
```

```
Array_1 + Array_2
[[  0   6  12  18  24]
 [ 30  36  42  48  54]
 [ 60  66  72  78  84]
 [ 90  96 102 108 114]]
```

Elementwise Subtraction

In [37]:
```
print(np.subtract(array_1,array_2))
```

```
[[  0  -4  -8 -12 -16]
 [-20 -24 -28 -32 -36]
 [-40 -44 -48 -52 -56]
 [-60 -64 -68 -72 -76]]
```

Elementwise Multiplication

In [38]:
```
print(np.multiply(array_1,array_2))
```

```
[[   0    5   20   45   80]
 [ 125  180  245  320  405]
 [ 500  605  720  845  980]
 [1125 1280 1445 1620 1805]]
```

Elementwise Divition

In [39]:
```
print(np.divide(array_1,array_2))
```

```
[[nan 0.2 0.2 0.2 0.2]
 [0.2 0.2 0.2 0.2 0.2]
 [0.2 0.2 0.2 0.2 0.2]
 [0.2 0.2 0.2 0.2 0.2]]

C:\Users\soongaya\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarning: invalid value encountered in t
rue_divide
  """Entry point for launching an IPython kernel.
```

Elementwise Power

In [40]:
```
print(np.power(array_1,array_2))
```

```
[[          1           1        1024    14348907           0]
 [  167814181  1073741824   613813847           0 -1054898967]
 [          0  2138032771           0  -476037875           0]
 [  511147439           0    72095569           0 -1801488741]]
```

Inner product

In [41]:
```
print(np.inner(array_1,array_2))
```

```
[[ 150  400  650  900]
 [ 400 1275 2150 3025]
 [ 650 2150 3650 5150]
 [ 900 3025 5150 7275]]
```

NumPy Statistics functions

Sum all elements in an array using sum() function

sum() function offers better performance than manually iterating through each element

In [42]:
```
print("1. Original array \n",score_height)
print("\n2. Sum of columns \n ",np.sum(score_height,axis=0))
print("\n3. Sum of rows \n ",np.sum(score_height,axis=1))
print("\n4. Sum of all elements \n",np.sum(score_height))
```

```
1. Original array
 [[ 4.5  5.1  5.3  5.5 58.  63.  70.  76. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]]

2. Sum of columns
  [ 4.5  5.1  5.3  5.5 58.  63.  70.  76. ]

3. Sum of rows
  [287.4   0.    0.    0.    0. ]

4. Sum of all elements
 287.4
```

Cumulative sum

In [43]:
```
print("1. Original array \n",score_height)
print("\n2. Cumulative sum of columns \n ",np.cumsum(score_height,axis=0))
print("\n3. Cumulative sum of rows \n ",np.cumsum(score_height,axis=1))
print("\n4. Cumulative sum of all elements \n",np.cumsum(score_height))
```

```
1. Original array
 [[ 4.5  5.1  5.3  5.5 58.  63.  70.  76. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]]

2. Cumulative sum of columns
  [[ 4.5  5.1  5.3  5.5 58.  63.  70.  76. ]
 [ 4.5  5.1  5.3  5.5 58.  63.  70.  76. ]
 [ 4.5  5.1  5.3  5.5 58.  63.  70.  76. ]
 [ 4.5  5.1  5.3  5.5 58.  63.  70.  76. ]
 [ 4.5  5.1  5.3  5.5 58.  63.  70.  76. ]]

3. Cumulative sum of rows
  [[  4.5   9.6  14.9  20.4  78.4 141.4 211.4 287.4]
 [  0.    0.    0.    0.    0.    0.    0.    0. ]
 [  0.    0.    0.    0.    0.    0.    0.    0. ]
 [  0.    0.    0.    0.    0.    0.    0.    0. ]
 [  0.    0.    0.    0.    0.    0.    0.    0. ]]

4. Cumulative sum of all elements
 [  4.5   9.6  14.9  20.4  78.4 141.4 211.4 287.4 287.4 287.4 287.4 287.4
 287.4 287.4 287.4 287.4 287.4 287.4 287.4 287.4 287.4 287.4 287.4 287.4
 287.4 287.4 287.4 287.4 287.4 287.4 287.4 287.4 287.4 287.4 287.4 287.4
 287.4 287.4 287.4 287.4]
```

Return the product of array elements

In [44]:
```
print("1. Original array \n",score_height)
print("\n2. Product of columns \n ",np.prod(score_height,axis=0))
print("\n3. Product of rows \n ",np.prod(score_height,axis=1))
print("\n4. Product of all elements \n",np.prod(score_height))
```

```
1. Original array
 [[ 4.5  5.1  5.3  5.5 58.  63.  70.  76. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]]

2. Product of columns
  [0. 0. 0. 0. 0. 0. 0. 0.]

3. Product of rows
  [1.30047325e+10 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00]

4. Product of all elements
 0.0
```

Find max element in a numpy array

In [45]:
```
print("1. Original array \n",score_height)
print("\n2. Max of columns \n ",np.max(score_height,axis=0))
print("\n3. Max of rows \n ",np.max(score_height,axis=1))
print("\n4. Max of all elements \n",np.max(score_height))
```

```
1. Original array
 [[ 4.5  5.1  5.3  5.5 58.  63.  70.  76. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]]

2. Max of columns
  [ 4.5  5.1  5.3  5.5 58.  63.  70.  76. ]

3. Max of rows
  [76.  0.  0.  0.  0.]

4. Max of all elements
 76.0
```

Find min element in a numpy array

In [46]:
```
print("1. Original array \n",score_height)
print("\n2. Min of columns \n ",np.min(score_height,axis=0))
print("\n3. Min of rows \n ",np.min(score_height,axis=1))
print("\n4. Min of all elements \n",np.min(score_height))
```

```
1. Original array
 [[ 4.5  5.1  5.3  5.5 58.  63.  70.  76. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]]

2. Min of columns
  [0. 0. 0. 0. 0. 0. 0. 0.]

3. Min of rows
  [4.5 0.  0.  0.  0. ]

4. Min of all elements
 0.0
```

Find mean of a numpy array

In [47]:
```
print("1. Original array \n",score_height)
print("\n2. Mean of columns \n ",np.mean(score_height,axis=0))
print("\n3. Mean of rows \n ",np.mean(score_height,axis=1))
print("\n4. Mean of all elements \n",np.mean(score_height))
```

```
1. Original array
 [[ 4.5  5.1  5.3  5.5 58.  63.  70.  76. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]]

2. Mean of columns
  [ 0.9   1.02  1.06  1.1  11.6  12.6  14.   15.2 ]

3. Mean of rows
  [35.925  0.     0.     0.     0.   ]

4. Mean of all elements
 7.185
```

Find variance of a 1-Dimensional numpy array

Variance refers to a statistical measure of the spread between values in a data set. It shows how values/variables varies with respect to each other. It is the square of Standard Deviation

In [48]:
```
print("1. Original array \n",score_height)
print("\n2. Variance of columns \n ",np.var(score_height,axis=0))
print("\n3. Variance of rows \n ",np.var(score_height,axis=1))
print("\n4. Variance of all elements \n",np.var(score_height))
```

```
1. Original array
 [[ 4.5  5.1  5.3  5.5 58.  63.  70.  76. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]]

2. Variance of columns
  [  3.24     4.1616   4.4944   4.84   538.24   635.04   784.     924.16  ]

3. Variance of rows
  [973.594375   0.         0.         0.         0.      ]

4. Variance of all elements
 401.21577499999995
```

Find standard deviation of a numpy array

Standard Deviation is a statistic that measures the dispersion/spread of a value with respect to the mean. A low standard deviation shows that the values are closser to the mean of the dataset, while a high standard deviation shows that the values are widely spread out.

In [49]:
```
print("1. Original array \n",score_height)
print("\n2. Standard deviation of columns \n ",np.std(score_height,axis=0))
print("\n3. Standard deviation  of rows \n ",np.std(score_height,axis=1))
print("\n4. Standard deviation  of all elements \n",np.std(score_height))
```

```
1. Original array
 [[ 4.5  5.1  5.3  5.5 58.  63.  70.  76. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0. ]]

2. Standard deviation of columns
  [ 1.8   2.04  2.12  2.2  23.2  25.2  28.   30.4 ]

3. Standard deviation  of rows
  [31.20247386  0.          0.          0.          0.        ]

4. Standard deviation  of all elements
 20.030371314581263
```

Find correlation coefficient of a numpy array

Correlation coefficients is a measure of the strength of relationship between two variables. It shows the magnitude and direction of the relationship between two variables.

In [50]:
```
print(np.corrcoef(score_height))
```

```
[[ 1. nan nan nan nan]
 [nan nan nan nan nan]
 [nan nan nan nan nan]
 [nan nan nan nan nan]
 [nan nan nan nan nan]]
```

```
C:\Users\soongaya\Anaconda3\lib\site-packages\numpy\lib\function_base.py:2559: RuntimeWarning: invalid value encounte
red in true_divide
  c /= stddev[:, None]
C:\Users\soongaya\Anaconda3\lib\site-packages\numpy\lib\function_base.py:2560: RuntimeWarning: invalid value encounte
red in true_divide
  c /= stddev[None, :]
```

Read Data to NumPy

Read data from text file

In [51]:
```
students_data=np.loadtxt('students.txt')
print(students_data)
```

```
[[78.   5.1]
 [56.   4.5]
 [67.   5.3]
 [73.   6. ]]
```

Read data from csv file

In [52]:
```
titanic_data=np.genfromtxt('titanic.csv',delimiter=',') # add ,dtype='str' to display text data
print(titanic_data)
```

```
[[    nan     nan     nan ...     nan     nan     nan]
 [ 0.      3.        nan ...  1.      0.      7.25  ]
 [ 1.      1.        nan ...  1.      0.     71.2833]
 ...
 [ 0.      3.        nan ...  1.      2.     23.45  ]
 [ 1.      1.        nan ...  0.      0.     30.    ]
 [ 0.      3.        nan ...  0.      0.      7.75  ]]
```

Store Data to File

Save data to a text file

In [53]:
```
np.savetxt('students_saved_array.txt',students_data,delimiter=' ')
```

Save data to a csv file

In [54]:
```
np.savetxt('titanic_saved_array.csv',titanic_data,delimiter=',')
```

@Sammy Ongaya https://data2ml.com (https://data2ml.com) https://github.com/SammyOngaya (https://github.com/SammyOngaya) sammiongaya@gmail.com