NAME:vedant swami

PRN NO:202201030023

ROLL NO:659

DIV:F(F3)

# ASSIGNMENT 3

```
import numpy as np
arr1=np.array([1,2,3])
arr2=np.array([4,5,6])
arr3=np.hstack((arr1,arr2))
print (arr3)
```

[1 2 3 4 5 6]

---

```
import numpy as np
array1=np.array([[1,2,3],[4,5,6],[7,8,9]])
array1
```

```
array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

```
array2=np.array([[11,12,13],[14,15,16],[17,18,19]])
array2
```

```
array([[11, 12, 13], [14, 15, 16], [17, 18, 19]])
```

## #1. Matrix Operation

### 1.1 Addition

```
resultarray=array1 +array2
print("\nUsing Operator:\n",resultarray)
resultarray=np.add(array1,array2)
print("\nUsing Numpy Function:\n",resultarray)
```

```
Using Operator:

 [[12 14 16]
 [18 20 22]
 [24 26 28]]

Using Numpy Function:
 [[12 14 16]
 [18 20 22]
 [24 26 28]]
```

1.2 Substraction

```
resultarray=array1 -array2
print("\nUsing Operator:\n",resultarray)
resultarray=np.subtract(array1,array2)
print("\nUsing Numpy Function:\n",resultarray)
```

```
Using Operator:
 [[-10 -10 -10]
 [-10 -10 -10]
 [-10 -10 -10]]

Using Numpy Function:
 [[-10 -10 -10]
 [-10 -10 -10]
 [-10 -10 -10]]
```

1.3 Multiplication

```
resultarray= array1*array2
print("\nUsing Operator:\n",resultarray)
resultarray=np.multiply(array1,array2)
print("\nUsing Numpy Function:\n",resultarray)
```

```
Using Operator:
 [[ 11  24  39]
 [ 56  75  96]
 [119 144 171]]

Using Numpy Function:
 [[ 11  24  39]
 [ 56  75  96]
 [119 144 171]]
```

1.4 Division

```
resultarray=array1/array2
```

```
print("\nUsing Operator:\n",resultarray)
resultarray=np.divide(array1,array2)
print("\nUsing Numpy Function:\n",resultarray)
```

```
Using Operator:
 [[0.09090909 0.16666667 0.23076923]
 [0.28571429 0.33333333 0.375     ]
 [0.41176471 0.44444444 0.47368421]]

Using Numpy Function:
 [[0.09090909 0.16666667 0.23076923]
 [0.28571429 0.33333333 0.375     ]
 [0.41176471 0.44444444 0.47368421]]
```

1.5 Mode

```
resultarray=array1%array2
print("\nUsing Operator:\n",resultarray)
resultarray=np.mod(array1,array2)
print("\nUsing Numpy Function:\n",resultarray)
```

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

Using Numpy Function:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

1.6 Dot Product

```
resultarray=np.dot(array1,array2)
print("",resultarray)
```

```
[[ 90  96 102]
 [216 231 246]
 [342 366 390]]
```

1.7 Transpose

```
resultarray=np.transpose(array1)
print(resultarray)
```

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

2. Horizontal and vertical stacking of Numpy Arrays

2.1. Horizontal Stacking

```
resultarray=np.hstack((array1,array2))
resultarray
```

```
array([[ 1, 2, 3, 11, 12, 13], [ 4, 5, 6, 14, 15, 16], [ 7, 8, 9, 17,
18, 19]])
```

2.2 Vertical Stacking

```
resultarray=np.vstack((array1,array2))
resultarray
```

```
array([[ 1, 2, 3], [ 4, 5, 6], [ 7, 8, 9], [11, 12, 13], [14, 15, 16],
[17, 18, 19]])
```

3.Custom sequence generation

3.1 Range

```
nparray=np.arange(0,12,1).reshape(3,4)
nparray
```

```
array([[ 0, 1, 2, 3], [ 4, 5, 6, 7], [ 8, 9, 10, 11]])
```

3.2 :Linearly seperable

```
nparray=np.linspace(start=0,stop=24,num=12).reshape(3,4)
nparray
```

```
array([[ 0. , 2.18181818, 4.36363636, 6.54545455],
       [ 8.72727273, 10.90909091, 13.09090909, 15.27272727],
[ 17.45454545, 19.63636364, 21.81818182, 24. ]]
```

3.3 Empty Array

```
nparray=np.empty((3,3),int)
nparray
```

```
array([[ 90, 96, 102], [216, 231, 246], [342, 366, 390]])
```

3.4 Emply Like Some Other Array

```
nparray=np.empty_like(array1)
nparray
```

```
array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

### 3.5. Identity Matrix

```
nparray=np.identity(3)
nparray
```

```
array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])
```

## 4. Arithmetic and Statistical Operations, Mathematical Operations, Bitwise Operators

### 4.1. Arithmetic Operation

```
array1=np.array([1,2,3,4,5])
array2=np.array([11,12,13,14,15])
print(array1)
print(array2)
```

```
[1 2 3 4 5]
[11 12 13 14 15]
```

```
# Addition
print(np.add(array1,array2))
# Subtraction
print(np.subtract(array1,array2))
# Multiplication
print(np.multiply(array1,array2))
# Division
print(np.divide(array1,array2))
```

```
[12 14 16 18 20]
[-10 -10 -10 -10 -10]
[11 24 39 56 75]
[0.09090909 0.16666667 0.23076923 0.28571429 0.33333333
```

### 4.2. Statistical and Mathematical Operations

```
array1=np.array([1,2,3,4,5,9,6,7,8,9,9])
# Standard Deviation
print(np.std(array1))
#Minimum
print(np.min(array1))
#Summation
print(np.sum(array1))
#Median
```

```
print(np.median(array1))
#Mean
print(np.mean(array1))
#Mode
from scipy import stats
print("Most Frequent element=",stats.mode(array1)[0])
print("Number of Occarances=",stats.mode(array1)[1])
# Variance
print(np.var(array1))
```

```
2.7990553306073913
1
63
6.0
5.7272727272727275
Most Frequent element= [9]
Number of Occarances= [3]
```

4.3. Bitwise Operations

```
array1=np.array([1,2,3],dtype=np.uint8)
array2=np.array([4,5,6])
# AND
resultarray=np.bitwise_and(array1,array2)
print(resultarray)
# OR
resultarray=np.bitwise_or(array1,array2)
print(resultarray)
#LeftShift
resultarray=np.left_shift(array1,2)
print(resultarray)
#RightShift
resultarray=np.right_shift(array1,2)
print(resultarray)
```

```
[0 0 2]
[5 7 7]
[ 4 8 12]
[0 0 0]
```

```
print(np.binary_repr(10,8))
resultarray=np.left_shift(10,2)
print(resultarray)
print(np.binary repr(np.left shift(10,2),8))
```

```
00001010
40
00101000
```

5.Copying and viewing arrays

5.1 Copy

```
array1=np.arange(1,10)
print(array1)
newarray=array1.copy()
print(newarray)
#modification in Original Array
array1[0]=100
print(array1)
print(newarray)
```

```
[1 2 3 4 5 6 7 8 9]
[1 2 3 4 5 6 7 8 9]
[100   2   3   4   5   6   7   8   9]
[1 2 3 4 5 6 7 8 9]
```

5.2 View

```
array1=np.arange(1,10)
print(array1)
newarray=array1.view()
print(newarray)
#modification in Original Array
array1[0]=100
print(array1)
print(newarray)
```

```
[1 2 3 4 5 6 7 8 9]
[1 2 3 4 5 6 7 8 9]
[100   2   3   4   5   6   7   8   9]
[100   2   3   4   5   6   7   8   9]
```

6 Searching

```
array1=np.array([[1,2,3,12,5,7],[94,5,6,7,89,44],[7,8,9,11,13,14]])
print(array1)
```

```
[[ 1  2  3 12  5  7]
[94  5  6  7 89 44]
[ 7  8 9 11 13 14]]
```

```
np.sort(array1,axis=0) #Horizontally Sort
```

```
array([[ 1,  2,  3,  7,  5,  7], [ 7,  5,  6, 11, 13, 14], [94,  8,  9, 12, 89,
44]])
```

```
np.sort(array1,axis=1) # Vertically Sort
```

```
array([[ 1,  2,  3,  5,  7, 12], [ 5,  6,  7, 44, 89, 94], [ 7,  8,  9, 11, 13,
14] 7]])
```

7. Searching

```
array1=np.array([1,2,3,12,5,7])
np.searchsorted(array1,7,side="left")#Perform Search After sorting
```

```
3
```

8. Counting

```
array1=np.array([1,2,3,12,5,7,0])
print(np.count_nonzero(array1))#Return total Non Zero element
print(np.nonzero(array1))#Return Index
print(array1.size)#Total Element
```

```
6
(array([0, 1, 2, 3, 4, 5]),)
7
```

---

9. Data Stacking

```
array1=np.array(np.arange(1,5).reshape(2,2))
print(array1)
array2=np.array(np.arange(11,15).reshape(2,2))
print(array2)
```

```
[[1 2]
 [3 4]]
[[11 12]
 [13 14]]
```

```
newarray=np.stack([array1,array2],axis=0)
print(newarray)
```

```
[[[ 1  2]
  [ 3  4]]

 [[11 12]
  [13 14]]]
```

```
newarray=np.stack([array1,array2],axis=1)
print(newarray)
```

```
[[[ 1  2]
  [11 12]]

 [[ 3  4]
  [13 14]]]
```

10. Append

```
array1=np.arange(1,10).reshape(3,3)
print(array1)
array2=np.arange(21,30).reshape(3,3)
print(array2)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[21 22 23]
 [24 25 26]
 [27 28 29]]
```

```
np.append(array1,array2,axis=0)
```

```
array([[ 1,  2,  3], [ 4,  5,  6], [ 7,  8,  9], [21, 22, 23], [24, 25, 26],
[27, 28, 29]])
```

```
np.append(array1,array2,axis=1)
```

```
array([[ 1,  2,  3, 21, 22, 23], [ 4,  5,  6, 24, 25, 26], [ 7,  8,  9, 27,
28, 29]])
```

11. Concat

```
array1=np.arange(1,10).reshape(3,3)
print(array1)
array2=np.arange(21,30).reshape(3,3)
print(array2)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[21 22 23]
 [24 25 26]
 [27 28 29]]
```

## ASSIGNMENT 3B

```
import numpy as np

d1=np. genfromtxt("/content/testmarks1.csv",delimiter=',')
print(d1)
EDS=d1[:,1]
print(type (EDS))
print(max(EDS))
```

```
[[   nan    nan    nan    nan    nan]
 [801.    43.05  27.79  28.7   27.79]
 [802.    43.47  28.52  28.98  27.89]
 [803.    42.24  28.16  28.16  25.63]
 [804.    39.24  26.16  26.16  26.16]
 [805.    40.9   26.03  27.27  25.65]
 [806.    39.47  26.31  26.31  25.21]
 [807.    41.68  25.63  27.79  25.46]
 [808.    42.19  27.61  28.13  26.21]
 [809.    44.75  28.35  29.83  28.21]
 [810.    46.95  28.88  31.3   28.53]]
<class 'numpy.ndarray'>
nan
```

```
import numpy as np
d2=np.genfromtxt("/content/testmarks2.csv",delimiter=',')
print(d2)
EDS=d2[:,1]
print(type (EDS))
print(max(EDS))
```

```
[[   nan    nan    nan    nan    nan]
 [801.    28.48  34.18  30.56  22.23]
 [802.    28.1   33.72  30.68  22.82]
 [803.    26.16  31.39  28.2   22.53]
 [804.    26.16  31.39  28.78  20.93]
 [805.    26.1   31.32  28.22  20.82]
 [806.    25.45  30.54  27.73  21.05]
 [807.    26.16  31.39  28.01  20.51]
 [808.    27.44  32.93  28.83  22.08]
 [809.    28.63  34.35  31.03  22.68]
 [810.    30.35  36.42  31.38  23.1 ]]
<class 'numpy.ndarray'>
Nan
```

Matrix Operator

Addition

```
result=d1+d2
print("/n Using operator",result)
result=np.add(d1,d2)
print("/n Using numpy funtion",result)
```

```
/n Using operator [[nan     nan     nan nan     nan]
 [1602.     56.96   68.36   61.12   44.46]
```

```
 [1604.     56.2    67.44   61.36   45.64]
 [1606.     52.32   62.78   56.4    45.06]
 [1608.     52.32   62.78   57.56   41.86]
 [1610.     52.2    62.64   56.44   41.64]
 [1612.     50.9    61.08   55.46   42.1 ]
 [1614.     52.32   62.78   56.02   41.02]
 [1616.     54.88   65.86   57.66   44.16]
 [1618.     57.26   68.7    62.06   45.36]
 [1620.     60.7    72.84   62.76   46.2 ]]
/n Using numpy funtion [[    nan     nan     nan     nan     nan]
 [1602.     56.96   68.36   61.12   44.46]
 [1604.     56.2    67.44   61.36   45.64]
 [1606.     52.32   62.78   56.4    45.06]
 [1608.     52.32   62.78   57.56   41.86]
 [1610.     52.2    62.64   56.44   41.64]
 [1612.     50.9    61.08   55.46   42.1 ]
 [1614.     52.32   62.78   56.02   41.02]
 [1616.     54.88   65.86   57.66   44.16]
 [1618.     57.26   68.7    62.06   45.36]
 [1620.     60.7    72.84   62.76   46.2 ]]
```

Subtraction

```
result=d1-d2
print("/n Using operator",result)
result=np.subtract(d1,d2)
print("/n Using numpy funtion",result)
```

```
/n Using operator [[nan nan nan nan nan]
 [ 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.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]]
/n Using numpy funtion [[nan nan nan nan nan]
 [ 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.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]]
```

Division

```
result=d1/d2
print("\nUsing Operator:\n",result)
result=np.divide(d1,d2)
print("\nUsing Numpy Function:\n",result)
```

```
Using Operator:
 [[nan nan nan nan nan]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]]

Using Numpy Function:
 [[nan nan nan nan nan]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]]
```

Multiplication

```
result=d1*d2
print("\nUsing Operator:\n",result)
resultarray=np.multiply(d1,d2)
print("\nUsing Numpy Function:\n",result)
```

```
Using Operator:
 [[          nan           nan           nan           nan
nan]
 [6.4160100e+05 8.1111040e+02 1.1682724e+03 9.3391360e+02
4.9417290e+02]
 [6.4320400e+05 7.8961000e+02 1.1370384e+03 9.4126240e+02
5.2075240e+02]
 [6.4480900e+05 6.8434560e+02 9.8533210e+02 7.9524000e+02
5.0760090e+02]
 [6.4641600e+05 6.8434560e+02 9.8533210e+02 8.2828840e+02
4.3806490e+02]
```

```
 [6.4802500e+05 6.8121000e+02 9.8094240e+02 7.9636840e+02
4.3347240e+02]
 [6.4963600e+05 6.4770250e+02 9.3269160e+02 7.6895290e+02
4.4310250e+02]
 [6.5124900e+05 6.8434560e+02 9.8533210e+02 7.8456010e+02
4.2066010e+02]
 [6.5286400e+05 7.5295360e+02 1.0843849e+03 8.3116890e+02
4.8752640e+02]
 [6.5448100e+05 8.1967690e+02 1.1799225e+03 9.6286090e+02
5.1438240e+02]
 [6.5610000e+05 9.2112250e+02 1.3264164e+03 9.8470440e+02
5.3361000e+02]]

Using Numpy Function:
 [[          nan           nan           nan           nan
nan]
 [6.4160100e+05 8.1111040e+02 1.1682724e+03 9.3391360e+02
4.9417290e+02]
 [6.4320400e+05 7.8961000e+02 1.1370384e+03 9.4126240e+02
5.2075240e+02]
 [6.4480900e+05 6.8434560e+02 9.8533210e+02 7.9524000e+02
5.0760090e+02]
 [6.4641600e+05 6.8434560e+02 9.8533210e+02 8.2828840e+02
4.3806490e+02]
 [6.4802500e+05 6.8121000e+02 9.8094240e+02 7.9636840e+02
4.3347240e+02]
 [6.4963600e+05 6.4770250e+02 9.3269160e+02 7.6895290e+02
4.4310250e+02]
 [6.5124900e+05 6.8434560e+02 9.8533210e+02 7.8456010e+02
4.2066010e+02]
 [6.5286400e+05 7.5295360e+02 1.0843849e+03 8.3116890e+02
4.8752640e+02]
 [6.5448100e+05 8.1967690e+02 1.1799225e+03 9.6286090e+02
5.1438240e+02]
 [6.5610000e+05 9.2112250e+02 1.3264164e+03 9.8470440e+02
5.3361000e+02]]
```

Mode

```
result= d1 %d2
print("/n Using Operator",result)
result=np. mod(d1,d2)
print("/nUsing numpy function",result)
```

```
/n Using Operator [[nan nan nan nan nan]
 [ 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.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
```

```
 [ 0.  0.  0.  0.  0.]]
/nUsing numpy function [[nan nan nan nan nan]
 [ 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.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]]
```

Horizontal Stacking

```
result=np.hstack((d1,d2))
result
```

```
array([[ nan, nan, nan, nan, nan, nan, nan, nan, nan, nan],
[801. , 28.48, 34.18, 30.56, 22.23, 801. , 28.48, 34.18, 30.56, 22.23],
[802. , 28.1 , 33.72, 30.68, 22.82, 802. , 28.1 , 33.72, 30.68, 22.82],
[803. , 26.16, 31.39, 28.2 , 22.53, 803. , 26.16, 31.39, 28.2 , 22.53],
[804. , 26.16, 31.39, 28.78, 20.93, 804. , 26.16, 31.39, 28.78, 20.93],
[805. , 26.1 , 31.32, 28.22, 20.82, 805. , 26.1 , 31.32, 28.22, 20.82],
[806. , 25.45, 30.54, 27.73, 21.05, 806. , 25.45, 30.54, 27.73, 21.05],
[807. , 26.16, 31.39, 28.01, 20.51, 807. , 26.16, 31.39, 28.01, 20.51],
[808. , 27.44, 32.93, 28.83, 22.08, 808. , 27.44, 32.93, 28.83, 22.08],
[809. , 28.63, 34.35, 31.03, 22.68, 809. , 28.63, 34.35, 31.03, 22.68],
[810. , 30.35, 36.42, 31.38, 23.1 , 810. , 30.35, 36.42, 31.38, 23.1 ]])
```

Vertical Stacking

```
result=np.vstack((d1,d2))
result
```

```
array([[ nan, nan, nan, nan, nan],
[801. , 43.05, 27.79, 28.7 , 27.79],
 [802. , 43.47, 28.52, 28.98, 27.89],
[803. , 42.24, 28.16, 28.16, 25.63],
[804. , 39.24, 26.16, 26.16, 26.16],
[805. , 40.9 , 26.03, 27.27, 25.65],
[806. , 39.47, 26.31, 26.31, 25.21],
[807. , 41.68, 25.63, 27.79, 25.46],
[808. , 42.19, 27.61, 28.13, 26.21],
```

```
[809. , 44.75, 28.35, 29.83, 28.21],
[810. , 46.95, 28.88, 31.3 , 28.53],
[ nan, nan, nan, nan, nan],
[801. , 28.48, 34.18, 30.56, 22.23],
[802. , 28.1 , 33.72, 30.68, 22.82],
[803. , 26.16, 31.39, 28.2 , 22.53],
[804. , 26.16, 31.39, 28.78, 20.93],
[805. , 26.1 , 31.32, 28.22, 20.82],
[806. , 25.45, 30.54, 27.73, 21.05],
[807. , 26.16, 31.39, 28.01, 20.51],
[808. , 27.44, 32.93, 28.83, 22.08],
[809. , 28.63, 34.35, 31.03, 22.68],
[810. , 30.35, 36.42, 31.38, 23.1 ]])
```

Custon Sequence Generation

Range

```
arr1=np.arange(800,810,1)
print(arr1)
```

```
[800 801 802 803 804 805 806 807 808 809]
```

Linearly seperable

```
nparray=np.linspace(start=0,stop=24,num=12).reshape(3,4)
nparray
```

```
array([[ 0. , 2.18181818, 4.36363636, 6.54545455], [ 8.72727273,
10.90909091, 13.09090909, 15.27272727], [17.45454545, 19.63636364,
21.81818182, 24. ]]
```

Numbers of students who got more than 40 in EDS

```
[[   nan    nan    nan    nan    nan]
 [801.    43.05  27.79  28.7   27.79]
 [802.    43.47  28.52  28.98  27.89]
 [803.    42.24  28.16  28.16  25.63]
 [804.    39.24  26.16  26.16  26.16]
 [805.    40.9   26.03  27.27  25.65]
```

```
 [806.    39.47  26.31  26.31  25.21]
 [807.    41.68  25.63  27.79  25.46]
 [808.    42.19  27.61  28.13  26.21]
 [809.    44.75  28.35  29.83  28.21]
 [810.    46.95  28.88  31.3   28.53]]
<class 'numpy.ndarray'>
```

Arithmetic Operation

```
# Addition
print(np.add(d1,d2))
# Subtraction
print(np.subtract(d1,d2))
# Multiplication
print(np.multiply(d1,d2))
# Division
print(np.divide(d1,d2))
```

```
[[    nan     nan     nan     nan     nan]
 [1602.    71.53   61.97   59.26   50.02]
 [1604.    71.57   62.24   59.66   50.71]
 [1606.    68.4    59.55   56.36   48.16]
 [1608.    65.4    57.55   54.94   47.09]
 [1610.    67.     57.35   55.49   46.47]
 [1612.    64.92   56.85   54.04   46.26]
 [1614.    67.84   57.02   55.8    45.97]
 [1616.    69.63   60.54   56.96   48.29]
 [1618.    73.38   62.7    60.86   50.89]
 [1620.    77.3    65.3    62.68   51.63]]
[[  nan   nan   nan   nan   nan]
 [ 0.   14.57 -6.39 -1.86  5.56]
 [ 0.   15.37 -5.2  -1.7   5.07]
 [ 0.   16.08 -3.23 -0.04  3.1 ]
 [ 0.   13.08 -5.23 -2.62  5.23]
 [ 0.   14.8  -5.29 -0.95  4.83]
 [ 0.   14.02 -4.23 -1.42  4.16]
 [ 0.   15.52 -5.76 -0.22  4.95]
 [ 0.   14.75 -5.32 -0.7   4.13]
 [ 0.   16.12 -6.   -1.2   5.53]
 [ 0.   16.6  -7.54 -0.08  5.43]]
[[          nan           nan           nan           nan
nan]
 [6.4160100e+05 1.2260640e+03 9.4986220e+02 8.7707200e+02
6.1777170e+02]
 [6.4320400e+05 1.2215070e+03 9.6169440e+02 8.8910640e+02
6.3644980e+02]
 [6.4480900e+05 1.1049984e+03 8.8394240e+02 7.9411200e+02
5.7744390e+02]
 [6.4641600e+05 1.0265184e+03 8.2116240e+02 7.5288480e+02
5.4752880e+02]
```

```
 [6.4802500e+05 1.0674900e+03 8.1525960e+02 7.6955940e+02
5.3403300e+02]
 [6.4963600e+05 1.0045115e+03 8.0350740e+02 7.2957630e+02
5.3067050e+02]
 [6.5124900e+05 1.0903488e+03 8.0452570e+02 7.7839790e+02
5.2218460e+02]
 [6.5286400e+05 1.1576936e+03 9.0919730e+02 8.1098790e+02
5.7871680e+02]
 [6.5448100e+05 1.2811925e+03 9.7382250e+02 9.2562490e+02
6.3980280e+02]
 [6.5610000e+05 1.4249325e+03 1.0518096e+03 9.8219400e+02
6.5904300e+02]]
[[       nan        nan        nan        nan        nan]
 [1.         1.51158708 0.81304857 0.93913613 1.25011246]
 [1.         1.54697509 0.84578885 0.94458931 1.22217353]
 [1.         1.6146789  0.89710099 0.99858156 1.13759432]
 [1.         1.5        0.83338643 0.90896456 1.24988055]
 [1.         1.56704981 0.83109834 0.96633593 1.23198847]
 [1.         1.55088409 0.86149312 0.94879192 1.1976247 ]
 [1.         1.59327217 0.81650207 0.99214566 1.24134569]
 [1.         1.53753644 0.83844519 0.97571974 1.1870471 ]
 [1.         1.56304576 0.82532751 0.96132775 1.24382716]
 [1.         1.54695222 0.7929709  0.99745061 1.23506494]]
```

Statistical and Mathematical Operations

```
# Standard Deviation
print(np.std(d1))
#Minimum
print(np.min(d1))
#Summation
print(np.sum(d1))
#Median
print(np.median(d1))
#Mean
print(np.mean(d1))
#Mode
from scipy import stats
print("Most Frequent element=",stats.mode(d1)[0])
print("Number of Occarances=",stats.mode(d1)[1])
# Variance
print(np.var(d1))
```

```
nan
nan
nan
nan
nan
Most Frequent element= [[801.    39.24  25.63  26.16  25.21]]
```

```
Number of Occarances= [[1 1 1 1 1]]
```