Numpy:

It is predefined module used to create and manipulate ndarray, it has wide variety of functions used to perform the operations like

1. Mathematical ,logical operations
2. Linear Algebra, random number generation
3. Statistics
4. used for scientific calculations
5. It is very fast because it is built in or associated c programming

*** Numpy is used with the packages like SciPy(Scientific Python) and Mat-Plotlib(Plotting Library)

Using Numpy we can create following dimensional arrays.

1. 1D array also called as Vector
2. 2D array also called as Matrix
3. 3D array also called as Tensor

Operations using ndarray:

```
import numpy as np
```

1. Basics

Create:

```
a=np.array([1,2,3,4,5],dtype='int32')
for x in a:
   print(x)


a.dtype
a.ndim
a.shape
a.size
a.nbytes
a.itemsize
```

Accessing:

```
print(a)

a[2]
```

2D:

```
a = np.array([[1,2,3,4,5,6,7],[8,9,10,11,12,13,14],[11,21,31,41,51,61,71]])

for x in a:
    for y in x:
        print(y)
```

3D:

```
b = np.array([[[1,2],[3,4]],[[5,6],[7,8]]])

print(b[0,1,1]) # 4

print(b[1,0,1]) #


for x in b:
    for y in x:
        for z in y:
            print(z)
```

2. Intializing arrays:

```
# All 0s matrix

x=np.zeros((2,3))

print(x)


# All 1s matrix

y=np.ones((4,2,2), dtype='int32')

print(y)
```

```
# Any other number
np.full((2,2), 99)



# Any other number (full_like)
a=np.array([1,2,3,4,5])
np.full_like(a, 4)

# Random decimal numbers
np.random.rand(4,2)
array([[0.07805642, 0.53385716],
      [0.02494273, 0.99955252],
      [0.48588042, 0.91247437],
      [0.27779213, 0.16597751]])



# Random Integer values
np.random.randint(-4,-1, size=(3,3))
array([[-4, -4, -2],
      [-2, -2, -3],
      [-3, -2, -4]])





# The identity matrix
np.identity(5)
array([[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.]])
```

```
# Repeat an array-- rows
arr = np.array([[1,2,3]])
r1 = np.repeat(arr,3, axis=0)
print(r1)

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

```
# Repeat an array -- columns
arr = np.array([[1,2,3],[5,6,7]])
r1 = np.repeat(arr,3, axis=1)
print(r1)
[[1 1 1 2 2 2 3 3 3]
 [5 5 5 6 6 6 7 7 7]]
```

Copying the arrays:

```
a = np.array([1,2,3])
b = a.copy()
b[0] = 100
print(a)
print(b)
[1 2 3]
[100   2   3]
```

3. Mathematics

```
a = np.array([1,2,3,4])
print(a)


a+2


array([3,4,5,6])



a-2
array([-1,  0,  1,  2])


a*2
array([2, 4, 6, 8])
a/2
array([0.5, 1. , 1.5, 2. ])


a**2
array([ 1,  4,  9, 16], dtype=int32)



b = np.array([1,0,1,0])
a+b
array([2, 2, 4, 4])



np.cos(a)
array([ 0.54030231, -0.41614684, -0.9899925 , -0.65364362])
```

```
np.sin(a)

array([0.84147098, 0.90929743, 0.14112001])
```

Linear Algebra

```
a = np.ones((2,3))

print(a)


b = np.full((3,2), 2)

print(b)


np.matmul(a,b)


# Find the determinant

c = np.identity(3)

print(c)

np.linalg.det(c)
```

Statistics:

```
stats = np.array([[1,2,3],[4,5,6]])
```

1. min

```
np.min(stats)
```

2. max

```
np.max(stats)
```

3. sum by columns

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

# array([5, 7, 9])
```

3. sum by rows

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

# array([ 6, 15])
```

4. Vertically stacking arrays

```
v1 = np.array([1,2,3,4])
v2 = np.array([5,6,7,8])

np.vstack([v1,v2,v1,v2])

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

5. Horizontal stacking arrays

```
h1 = np.ones((2,4))
h2 = np.zeros((2,2))
```

```
np.hstack((h1,h2))
```

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

Re-organizing the arrays:

```
before = np.array([[1,2,3,4],[5,6,7,8]])
print(before)
```

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

```
after = before.reshape((4,2))
print(after)
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
```

More Statistics:

these are measures of central location

1. Mean: the average value
2. Median: the middle value
3. Mode: The most common value
4. Standard Deviation: σ = √(∑(x−¯x)2 /n)

1. Mean:

```
x=np.array([1,2,3,4,5,6,7,2,6,2,1,4,2,2,6])
print(np.mean(x))
3.533333333333333
```

2. Median: To calculate the median, we should sort

the array

to sort: np.sort(x)

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

by seeing this we can say the middle value as 3

So the median is 3

by using numpy:

```
np.median(x)
3.0
```

Mode: It is the most repeated value in an array

x=[1 1 2 2 2 2 2 3 4 4 5 6 6 6 7]

in this array the most repeated value is 2, So mode is 2

By Using numpy:

```
np.mode(x)
```

It throws an error, because mode is not the method of numpy, it is the method of scipy

```
from scipy import stats
stats.mode(x)
```

Result:

```
ModeResult(mode=array([2]), count=array([5]))
```

Standard Deviation:

1.9618585292749546

Lets see this whether it is correct or not manually

To calculate we do the following steps

```
z=np.array([2, 6, 5, 3, 2, 3])
```

1. calculate mean:

$\bar{x}$ =3.5

2. Construct a table

$x1$	$x1 - \bar{x}$	$(x1 - \bar{x})2$
2	-1.5	2.25
6	2.5	6.25
5	1.5	2.25
3	-0.5	0.25
2	-1.5	2.25
3	-0.5	0.25

= 13.5

Standard Deviation =>σ = √(∑(x−¯x)2 /n)

=√(13.5/[6-1])

=√[2.7]

=1.643

Load Data from File

```
filedata = np.genfromtxt('data.txt', delimiter=',')
filedata = filedata.astype('int32')
print(filedata)
```

data.txt:

```
1,13,21,11,196,75,4,3,34,6,7,8,0,1,2,3,4,5
```

3,42,12,33,766,75,4,55,6,4,3,4,5,6,7,0,11,12

1,22,33,11,999,11,2,1,78,0,1,2,9,8,7,1,76,88

Assignment:

create a textdocument

```
# Stats using numpy
temp = {
"day1":[32,34,36,34,35],
"day2":[34,33,36]
}
humidity = {
"day1":[1,2,4],
"day2":[2,3,5]
}
weather = {
"temp":temp,
"humidity":humidity
}
```

1.Min (Minimum):

Real-life Application: Finding the minimum temperature in weather data to predict cold spells.

32

2.Max (Maximum):

Real-life Application: Identifying the highest daily stock price for investment decisions.

36

3.Mean:

Real-life Application: Calculating the average test scores of students in a class to assess performance.

34.2

4.Median:

Real-life Application: Determining the median income in a population to study income distribution.

36

5.Mode:

Real-life Application: Finding the most commonly purchased item in a store's inventory.

34

6.Variance:

Real-life Application: Assessing the risk associated with different investment portfolios.

Calculate the squared differences from the mean for each data point: Day1

(32 - 34)^2 = (-2)^2 = 4

(34 - 34)^2 = 0

(36 - 34)^2 = 2^2 = 4

Calculate the variance by finding the average of the squared differences:

Variance = (4 + 0 + 4) / 3 = 8 / 3 ≈ 2.67 (rounded to two decimal places)

7.Standard Deviation:

Real-life Application: Measuring the variability in the heights of individuals in a population.

Standard Deviation = √(Variance) ≈ √(2.67) ≈ 1.63 (rounded to two decimal places)

8.Scalars:

Real-life Application: Representing a single temperature reading in a weather report.

```
[32,34,36,34,35]
```

9.Vectors:

Real-life Application: Modeling the velocity and direction of an object's movement in physics.

```
temp = {
"day1":[32,34,36,34,35],
"day2":[34,33,36]
}
```

10.Tensors:

Real-life Application: Utilized in deep learning for image and speech recognition tasks.

```
weather = {
"temp":temp,
"humidity":humidity
}
```

11.Correlations:

Real-life Application: Analyzing the relationship between apples and mangoes in fruits.

$2x+3y=8$

$3x+y=5$

12.Probability: It is calculated on vector

Real-life Application: Calculating the probability of winning in a game of chance like roulette.

$P(A|B) = P(B|A)P(A) / P(B)$

13.Differentiation:

Real-life Application: Calculating rates of change in physics to predict motion.

14.Integration:

Real-life Application: Finding the area under a curve to calculate total revenue in economics.

15.Arrays:

Real-life Application: Storing multiple data points like daily temperatures in a weather dataset.

16.Matrices:

Real-life Application: Representing network connections in computer science for routing algorithms.

```
import numpy as np

from scipy import stats


#1.Min (Minimum):

min_value=np.min(temp['day1'])

print("min of day1",min_value)


#2.Max (Maximum):

max_value=np.max(temp['day1'])

print("max of day1",max_value)


#3.Mean:

mean_value = np.mean(temp['day1'])

print("mean of day1",mean_value)


#4.Median:

median_value = np.median(temp['day1'])

print("median of day1",median_value)


#5.Mode:

data = np.array([12, 45, 23, 56, 34, 23, 67, 12, 34, 23, 12])

mode_value = stats.mode(data)

print("Mode:", mode_value.mode[0], "with a count of", mode_value.count[0])
```

```
#6.Variance:
variance = np.var(temp['day1'])
print("Variance",variance)

#7.Standard Deviation:
std_deviation = np.std(temp['day1'])
print("Standard Deviation",std_deviation)

output = np.ones((5,5))
print(output)

z = np.zeros((3,3))
z[1,1] = 9
print(z)


# matrix manipulations

import numpy as np

# Define two example matrices
matrix1 = np.array([[1, 2], [3, 4]])
matrix2 = np.array([[5, 6], [7, 8]])

# Addition
result_add = matrix1 + matrix2

# Subtraction
result_sub = matrix1 - matrix2
```

```
# Multiplication (element-wise)
result_mul_element = matrix1 * matrix2


# Matrix multiplication (dot product)
result_dot_product = np.dot(matrix1, matrix2)


# Transposition
result_transpose1 = np.transpose(matrix1)
result_transpose2 = matrix2.T



# Inversion (for square matrices)
#If A is a square matrix of size n, and its determinant is denoted as |A|, and its adjugate matrix as adj(A), then the formula for the inverse of A is:


#A^(-1) = (1 / |A|) * adj(A)


result_inverse1 = np.linalg.inv(matrix1)
result_inverse2 = np.linalg.inv(matrix2)


#det(A) = (a * d) - (b * c)
#det(A) = a(ei − fh) - b(di − fg) + c(dh − eg)


determinant = np.linalg.det(matrix1)



print("Matrix 1:\n", matrix1)
print("Matrix 2:\n", matrix2)
print("Addition:\n", result_add)
print("Subtraction:\n", result_sub)
print("Element-wise Multiplication:\n", result_mul_element)
print("Matrix Multiplication:\n", result_dot_product)

print("Transpose of Matrix 1:\n", result_transpose1)

print("Transpose of Matrix 2:\n", result_transpose2)

print("Inverse of Matrix 1:\n", result_inverse1)

print("Inverse of Matrix 2:\n", result_inverse2)

print(f"Determinant:{determinant}")
```