importing numpy

```
import numpy as np
```

General operation

```
temp=np.array([22,34,5,4,23,76,423,78,9,5,678,4,3,123,4567])
avg=np.mean(temp)

avg
```

```
np.float64(403.6)
```

Creating 1d array

```
Array_1d=np.array([10,20,30,40,50])
print(Array_1d)
```

```
[10 20 30 40 50]
```

Creating 2d array

```
#The rows and columns are forming an matrix

Array_2d=np.array([[1,2,3],
                   [4,5,6],
                   [7,8,9]
                   ])
print(Array_2d)
```

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

Creating Multidimensional array

```
# Basically a matrix is created as 3d array
Multidimensional=np.array([[[1,2,3],
                            [4,5,6]
                             ]])
print(Multidimensional)
```

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

Different operations

```
# Default values array in numpy
# In array the size is passed for 1d array
```

```
zero_array=np.zeros(3)
print(zero_array)
```

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

```
# Default values array in numpy
# In array the size is passed for 2d array
zero_array1=np.zeros((3,3))
print(zero_array1)
```

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

```
# Insted of zero we place one

one_array1=np.ones((3,3))
print(one_array1)
```

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

```
# specific value
# first twos are size of array and last element is that specific
element that has been passed as value


filled_Array=np.full((2,2),7)
print(filled_Array)
```

```
[[7 7]
 [7 7]]
```

```
# creating sequesces of numbers
# values passed in array-first- start , second-stop , third-step(jump)
Array=np.arange(1,10,2)
print(Array)

#starts from one stop at 9 jump (gap) is 2
```

```
[1 3 5 7 9]
```

```
# creating identity matrices
# Use .eye function to print diagonal matrix

identity_matrix=np.eye(3)   # 3 is  3-rows and 3-columns
print(identity_matrix)
```

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

```
identity_matrix=np.eye(4)   # 4 is  4-rows and 4-columns
print(identity_matrix)
```

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

Numpy array Properties

```
# shape , size , dimension , type

arr_2d=np.array([[1,2,3],
                 [4,5,6],
                 [7,8,9]])

print(arr_2d.shape)  #  rows,columns-(3, 3)

print(arr_2d.size)   # Total number of arrays-9

print(arr_2d.ndim)   # How many dimensions are in array-2

print(arr_2d.dtype)  # Data type of array-int64
```

```
(3, 3)
9
2
int64
```

```
# 3d array
arr_d=np.array([[[1,2,3],
                 [4,5,6],
                 [7,8,9]]]
                 )
print(arr_d.ndim)
```

```
3
```

```
# float array

float=np.array([1,2,3,4,5.5,6,7])
print(float.dtype)
```

```
float64
```

```
# change datatype

#conversion of float into int-float64=int64

# Actual values are change-[1 2 3 4 5]
```

```
arr=np.array([1.1,2.5,3.4,4.5,5.6])
print(arr.dtype)
int_arr=arr.astype(int)
print(int_arr)
print(int_arr.dtype)
```

```
float64
[1 2 3 4 5]
int64
```

Operators in Numpy

```
# Array
arr=np.array([10,20,30])

# Addition
print(arr+5)

# Substraction
print(arr-5)

# Multiplication
print(arr*2)

# power
print(arr**2)

# Division
print(arr/2)

# Last digit finding using modulus
print(arr%10)
```

```
[15 25 35]
[ 5 15 25]
[20 40 60]
[100 400 900]
[ 5. 10. 15.]
[0 0 0]
```

Aggregation function in numpy (summery)

```
# Array
arr=np.array([10,20,30])
# sum
print(np.sum(arr))

# Mean
print(np.mean(arr))
```

```
# Minimun
print(np.min(arr))

# Maximum
print(np.max(arr))

# Standard Deviation
print(np.std(arr))

# Varience
print(np.var(arr))
```

```
60
20.0
10
30
8.16496580927726
66.66666666666667
```

Indexing and Slicing

```
# Indexing

arr=np.array([100,200,300,400,500])
print(arr[0])  # access 0th element (first to last)

print(arr[-1]) # negative indexing from last to first
```

```
100
500
```

```
# Slicing Extracting subset or (subarray) of data
# accessing the portion of an array

print(arr[0:6:2]) #start stop and step   output fron 0th index to
last+1

print(arr[:6]) # 0th index to last

print(arr[::2]) # choose every second element

print(arr[::-1]) # -ve step reverse the array
```

```
[100 300 500]
[100 200 300 400 500]
[100 300 500]
[500 400 300 200 100]
```

Fancy indexing

```
arr=np.array([100,200,300,400,500])

# accessing specific indexe elements

print(arr[[0,1,4]])

# it creates the copy any chnges are not done with original array


[100 200 500]
```

Boolean masking

```
# It is basically the conditional function

arr=np.array([100,200,300,400,500])

print(arr[arr>250])


[300 400 500]
```

Reshaping and Manipulating Arrays

```
# reshaping is the changing the dimension of the array without
modifying the original data(array)
# 1d to 2d
# 2d to 3d
# only reshape when dimension match
# It does not creates the copy it returns the view
arr=np.array([100,200,300,400])

#reshape the 1d array into 2d array

arr.reshape(2,2)




array([[100, 200],
       [300, 400]])
```

Flattening array

```
# Used to convert Multidimensional array into 1d array
# There are two methods ravel which makes changes in actual arrays
# flatten which makes the copy does not make change in original array

arr=np.array([[1,2,3],
```

```
              [4,5,6],
              [7,8,9]])

print(arr.ndim)


print(arr.ravel())  # creates the view

# check the dimension

print(arr.ravel().ndim)

print(arr.flatten()) # creates the copy
```

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

Advance Numpy

```
# inserting element in 1d-array at specific index
arr=np.array([100,200,300,400])
print(np.insert(arr,1,10,axis=0))


# or we can write
newarr=np.insert(arr,1,3,axis=0)
print(newarr)
```

```
[100  10 200 300 400]
[100   3 200 300 400]
```

```
# inserting in 2d-array
arr2d=np.array([[1,2],
                [5,6]])
print(arr2d)

# operation
#axis=0 rowwise
new2darr=np.insert(arr2d,1,[3,4],axis=0)

print(new2darr)

#axis=1 columnwise
new2darr=np.insert(arr2d,1,[3,4],axis=1)

print(new2darr)

#axis=None flatten
```

```
new2darr=np.insert(arr2d,1,[3,4],axis=None)

print(new2darr)
```

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

```
# Adding data in array use .append use to add at the last of an array
arr=np.array([100,200,300,400])
newarr=np.append(arr,[500,600,700])
print(newarr)
```

```
[100 200 300 400 500 600 700]
```

```
# Adding 2 or more arrays together using concatinate

# row=0--vertical stacking
# column=1--Horizontal stacking

arr1=np.array([100,200,300,400])
arr2=np.array([500,600,700,800])
new=np.concatenate((arr1,arr2),axis=0)
print(new)
```

```
[100 200 300 400 500 600 700 800]
```

```
# Remove element from specific index in 1d array
arr=np.array([100,200,300,400])

np.delete(arr,0,axis=0)
```

```
array([200, 300, 400])
```

```
# Remove element from specific index in 2d array

arr=np.array([[100,200],
              [300,400]])

# Remove Specific element
print(np.delete(arr,0))

# Remove all row
print(np.delete(arr,0,axis=0))
```

```
[200 300 400]
[[300 400]]
```

```
arr1=np.array([1,2,3])
arr2=np.array([4,5,6])


# vertically stack-vstack

print(np.vstack((arr1,arr2)))


# horizontally stack-hstack

print(np.hstack((arr1,arr2)))
```

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

```
# spliting -arrays into subarrays

arr1=np.array([1,2,3,4,5,6])

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

# equal split
print(np.split(arr1,2))

# vsplit
print(np.vsplit(arr,2))

#hsplit
print(np.hsplit(arr,3))
```

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

Broadcasting

```
# This is the numpy way in which we perform different operations on
different shape of array

# Definition:
# Broadcasting is a set of rules NumPy uses to perform operations on
arrays of different shapes, without copying data.
```

```
# It makes automatic alignment of smaller arrays with larger ones for
arithmetic operations.

prices=np.array([100,200,300])

# we make a discount prices of 10%

discount=10
final_price=prices-(prices*discount/100)  # [100, 200, 300] - [10, 20,
30]
print(final_price)
```

```
[ 90. 180. 270.]
```

Matching dimension

```
# addition of two 1d-arrays
arr1=np.array([1,2,3])
arr2=np.array([4,5,6])
result=arr1+arr2
print(result)

# addition of two unequal arrays

array1=np.array([[10,20,30],
                 [40,50,60]])
array2=np.array([10,20,30])

print(array1+array2)
```

```
[5 7 9]
[[20 40 60]
 [50 70 90]]
```

Incompatable type error dimension not match

```
arr1=np.array([[10,20,30],
      [40,50,60]])
arr2=np.array([1,2])
print(arr1+arr2)
```

```
---------------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
Cell In[40], line 4
      1 arr1=np.array([[10,20,30],
      2       [40,50,60]])
      3 arr2=np.array([1,2])
----> 4 print(arr1+arr2)
```

```
ValueError: operands could not be broadcast together with shapes (2,3)
(2,)
```

Vectorization

```
# Definition:
# Vectorization means replacing explicit loops (like for loops) with
array operations that are performed in
# compiled code under the hood — making them faster and more efficient

arr = np.array([1, 2, 3, 4])
result = []
for x in arr:
    result.append(x * 2)
print(result)  # Output: [2, 4, 6, 8]
```

```
[np.int64(2), np.int64(4), np.int64(6), np.int64(8)]
```

Handling Missing values

```
# np.isnan-detect missing values
# NAN- Not a Number
# It returns boolean value
# we cannot directly compare the nan values
arr=np.array([1,2,np.nan,4])

print(np.isnan(arr))

# we cannot compare it directly
print(np.nan==np.nan)
```

```
[False False  True False]
False
```

```
# replace with another function

# replace with a specific number nan value

arr=np.array([1,2,np.nan,4])

print(np.isnan(arr))
clean_arr=np.nan_to_num(arr,nan=1000) # if not pass nan=1000 then
default is 0
print(clean_arr)
```

```
[False False  True False]
[   1.    2. 1000.    4.]
```

```
# used to detect infinite valeues
# 1/0 is infinite
```

```
# return boolean value
arr=np.array([1,2,3,np.inf,4,-np.inf,5])
print(np.isinf(arr))

# for replace
clean_arr=np.nan_to_num(arr,posinf=1000,neginf=-1000)
print(clean_arr)
```

```
[False False False  True False  True False]
[    1.     2.     3.  1000.     4. -1000.     5.]
```

```
# Importing dastset we required to import pandas

import pandas as pd

file_path="D:/Data Science/Amazon 1 Final.csv"
Amazon=pd.read_csv(file_path)
Amazon
```

```
            Order ID  Order Date   Ship Date   Status   Customer Name
\
0     CA-2013-138688  06/13/2013  06/17/2013  On time   DarrinVanHuff

1     CA-2011-115812  06-09-2011  06/14/2011    Delay  BrosinaHoffman

2     CA-2011-115812  06-09-2011  06/14/2011    Delay  BrosinaHoffman

3     CA-2011-115812  06-09-2011  06/14/2011    Delay  BrosinaHoffman

4     CA-2011-115812  06-09-2011  06/14/2011    Delay  BrosinaHoffman

...              ...         ...         ...      ...             ...

3198  CA-2013-125794  09/30/2013  10-04-2013  On time     MarisLaWare

3199  CA-2014-121258  02/27/2014  03-04-2014    Delay      DaveBrooks

3200  CA-2014-121258  02/27/2014  03-04-2014    Delay      DaveBrooks

3201  CA-2014-121258  02/27/2014  03-04-2014    Delay      DaveBrooks

3202  CA-2014-119914  05-05-2014  05-10-2014    Delay     ChrisCortes


             Country         City       State     Category  \
0     United States  Los Angeles  California       Labels
1     United States  Los Angeles  California  Furnishings
2     United States  Los Angeles  California          Art
3     United States  Los Angeles  California       Phones
4     United States  Los Angeles  California      Binders
...             ...          ...         ...          ...
3198  United States  Los Angeles  California  Accessories
```

```
3199  United States   Costa Mesa  California  Furnishings
3200  United States   Costa Mesa  California       Phones
3201  United States   Costa Mesa  California        Paper
3202  United States  Westminster  California   Appliances

                                                Product Name    Sales
Quantity  \
0     Self-Adhesive Address Labels for Typewriters b...   14.620
2.0
1     Eldon Expressions Wood and Plastic Desk Access...   48.860
7.0
2                                             Newell 322    7.280
4.0
3                         Mitel 5320 IP Phone VoIP phone  907.152
4.0
4     DXL Angle-View Binders with Locking Rings by S...   18.504
3.0
...                                                  ...      ...
...
3198  Memorex Mini Travel Drive 64 GB USB 2.0 Flash ...   36.240
1.0
3199  Tenex B1-RE Series Chair Mats for Low Pile Car...   91.960
2.0
3200                                Aastra 57i VoIP phone  258.576
2.0
3201  It's Hot Message Books with Stickers, 2 3/4" x 5"   29.600
4.0
3202  Acco 7-Outlet Masterpiece Power Center, Wihtou...  243.160
2.0

       Profit
0      6.8714
1     14.1694
2      1.9656
3     90.7152
4      5.7825
...       ...
3198  15.2208
3199  15.6332
3200  19.3932
3201  13.3200
3202  72.9480

[3203 rows x 13 columns]
```

```
# finding missing values in each coulmn
print('Missing values is-')
print(Amazon.isnull().sum())
```

```
Missing values is-
Order ID         0
Order Date       0
Ship Date        0
Status           0
Customer Name    0
Country          0
City             0
State            0
Category         0
Product Name     0
Sales            0
Quantity         5
Profit           0
dtype: int64
```

```
Amazon['Quantity'].fillna(Amazon['Quantity'].mean())
```

```
0       2.0
1       7.0
2       4.0
3       4.0
4       3.0
       ...
3198    1.0
3199    2.0
3200    2.0
3201    4.0
3202    2.0
Name: Quantity, Length: 3203, dtype: float64
```