Section A: Theory

1. What is a NumPy array, and how is it different from a Python list?

Q.1] what is Numpy Array & how is it different from Python list?

Numpy :- it stands for numerical python it is a porfurfull python library which is used to perform scientic computing & perform numerical computing task.

- It provides efficient multidimensional array object.
- it can store elements of same data type

Diff	Numpy Array	List
Data Type	① Homogeneous (all elements must be of the same type)	① Heterogeous (all elements can be different type
MIM effiency	② boz of fixed size & type memory effici is more	② It has less memory efficency boz of dynamic size
Performance	③ It faster than list. due to optimised C implemention	④ Slower, especially for large data
Funktality	④ it supports advance mathematical fun^n	④ Des not have built in fun for elementwise operation.

3. What are ufuncs in NumPy? Provide an example.

What are ufunc in Numpy? Provide example.

- Stands for universal function.
- It perform element-wise opn on arrays
- It allows you to apply opn on entire arrays without using explicit loops

Types of ufunction

① Unary ufunction → operate on single array (eg. square square root) eg np.sqrt(array)

② Binary ufunction → operate on two array element-wise (e.g addition, sub, mul)

③ Other ufuncs → more complex funn that might involve conditionals or other advance logic like user defined their own ufunc with help of np. frompy fun()

Note: you can check type of fun which is ufunc or not

eg. type(np.add) → ufunc

eg:-

```
x = [1, 2, 34, 4]
y = [10, 10, 20, 10]
print(np.add(x, y))
print(type(np.add))
```

o/p. [11, 12, 54, 14]
<class numpy.ufunc>

eg:- mathemtical functio

① np.abs() ② np.log ③ np.add()
④ np.substract() ⑤ np.divide
⑥ np.sin() etc

Statical functions

① np.mean ② np.median ③ np.percentile
etc

Comparison fun

① np.equal() ② np.less

you can define your own ufunc by using np.frompyfunc()
It take no. of i/p arguments & no. of output arguments as parameter.

eg:-

```
def each_add_10(x, y):
    return x+y+10
arr = np.frompyfunc(each_add_10, 2, 1)
```

fun name no. of arguments input no. of arguments output

```
x=[1,2,34,4]
y=[10,10,20,10]
print(np.add(x,y))
print(type(np.add))
print(np.sqrt(x))
print(type(np.sqrt))
print(np.less(x,y))

def each_add_10(x,y):
    return x+y+10
arr=np.frompyfunc(each_add_10,2,1)
print(arr(x,y))
```

2. Explain the concept of broadcasting in NumPy with an example.

Broadcasting

- The term broadcasting describes how numpy treats arrays with different shapes during arithmetic operations
- The smaller array is brodcast across the larger array so that they have compatible shapes.

Broadcasting Rules :-

(1) Make the two arrays have the same number of dimensions
- If the no. of dimensions of the two array are different, add new dimensions with size 1 to the head of the array within the smaller ~~arr~~ dimension

eg - 1

arr 1	arr 2	→
(3, 2)	(3,)	(1, 3)
2D	1D	(head) added ~~to~~ smaller dimen 1

eg - 2

(3, 3, 3) (3,) - (1, 1, 3)

(2) Make each dimension of the two arrays the same size

If sizes of each dimension of the two array do not match, dimesions with size 1 are stretched to the size of the other array.

(3, 3) (3) (1, 3)

not same

(3, 3) (3, 3)

If there is a dimension whose size is not 1 in either of the two arrays, it cannot be broadcasted, and an error is raised

eg :- arr1 = [[0, 1, 2]
[3 4 5]
[9 10 11]]
arr 2 = [0 1 2]

(4, 3) (3,) → (1, 3)
→ (4, 3)

(4, 3) (4, 3)

o/p [[0 2 4]
[3 5 7]
[6 8 10]
[9 11 13]]

eg 2 :- arr 1 shape (3, 4)
arr 2 shap (3,)

step 1 (3, 4) (3,) → (1, 3)
step 2 stretch 1 → (3, 3)

(3, 4) × (3, 3)
broadcasting is not paible
it shows Broadcasting error

eg 2 :- a1 (1, 3) a2 (3, 1)

step 1 (1, 3) (3, 1)
step 2 strech 1 by same dimension
(3, 3) (3, 3)

eg 3 :- a1 [0, 1, 2] a2 [[0]
[1]
[3]] - (3, 1)
(1, 3)

~~out~~ o/p

[[0 1 2]
[1 2 3]
[2 3 4]]

eg 3 - (1, 3) (4, 1)
(4, 3) (4, 3)

eg : (2, 2) (1) → (1, 1)
Stretch (2, 2)

4. What is the purpose of the numpy.random module? Name three functions it provides.

o/p [21, 22, 64, 24]

.4] what is purpose of the numpy. random module ?
- It provides fonns for generate random numbers.
- also it can provide random gerated array from range.
- It also enables of data shuffly.

1] np. random. rand () → Generate random number between 0 — 1
ey: np.random.rand () → 0.8756
ey% -
np. random. rand (2, 2) → [[0.081, 0.57], [0.61, 0.03]]
dimentions

2] np. random. randint () — Generate random integers from specified range (inclusive of low, exclusive of high) by default start from 0
from numpy import random as rd.
ey 1 → ① rd. randint (1, 10) → 5
rd. randint (1, 10, (2, 2)) →
[[3, 2]
[5 8]]
dimentions

3] np. random. randn () - Generates random numbers from a standard normal distribution (mean = 0, std = 1) - by ~~defaut~~
ey
np. random. randn (2, 2)
dimention
o/p [[-0.4, 0.8]
[-0.66, -1.63]]

4] np random. choice (a, size = None, replace = True)
- Generates a random sample from a given 1-D array. you can specify sample size and whethe sample is with or without duplicate or replacement.
replace - by defaut is True means duplicat element.
false - unique.
if you given size is bigger than choice array and replace is false you will get an error.
note - If your array having repeted elements and you give replace = false then also you can get duplicate elements of that array.
ey: rd. choice ([1, 2, 3, 4, 5])
o/p
4

[1, 2, 1]

5]. Shuffle - it shuffles the original array.
return, None
arr = [10, 20, 30, 40]
rd. shuffle (arr)
print (arr) → [30, 20, 10, 40]

6] random. permutation (x) - it return random shuffled (permutation of x rather than changing entire original array, (x)

⑦ ⑧ ⑨ rd. normal - (loc = 0.0, scale = 0.0, size
- generates random numbers from normal distribution with given mean (loc) and standard deviation (scale)
rd. binomial (n, p, size = None)
- Draws samples from binomial distribution, where n is the number of trials & p is the probality of success
rd. poisson (lam = 1.0, size = None)
Generate random samples from a poisson distribution with rate lam

10] uniform (low = 0.0, high 1.0, size = None)
- Generates random numbers from uniform distribution over [low, high]. - each element in range has equally chances for being selected. It returns random number in float

11] Seed function → we know that. when we generate random number or array each time when we executing random number get changed. But when if we want to fix the generation of this random number. in that case we use seed, this generates fix set of random numbers.
eg:-
rd. seed (12) It can be any number
rd. rand ()
o/p. 0.15 — 1st execution
0.15 — 2nd execution.

```
from numpy import random as rd
# rand
# print(rd.rand(2,2))
# [[0.08217302 0.5734172 ]
#  [0.61766962 0.03358355]]


#randint
# print(rd.randint(1,10)) ->5
# print(rd.randint(1,10,(2,2)))
# [[3 2]
#  [5 8]]

# print(rd.randint(1,10,size=5))
# [3 4 1 2 1]

# randn
# print(rd.randn(2,2))
# [[ 2.24035776  1.23595522]
#  [-0.39494653 -0.6499075 ]]

# choice
# print(rd.choice([1,2,3,4]))#2
print(rd.choice([1,1,4,5,6],size=6))#[6 1 6 5 1 1]
print(rd.choice([1,2,3,4],size=3,replace=False))#[2 1 4]
print(rd.choice([1,2,1,3,4],size=4,replace=False))#[2 4 1 1]
# print(rd.choice([1,2,3,4,1],size=6,replace=False))#Error


# Shuffle
arr=[10,20,30,40]
rd.shuffle(arr)
print(arr)
arr2=np.array([[10,20,30],[11,22,33],[1,2,3]])
rd.shuffle(arr2)
print(arr2)


# Permutation
print(rd.permutation([1,2,3,7,8,9,0]))
```

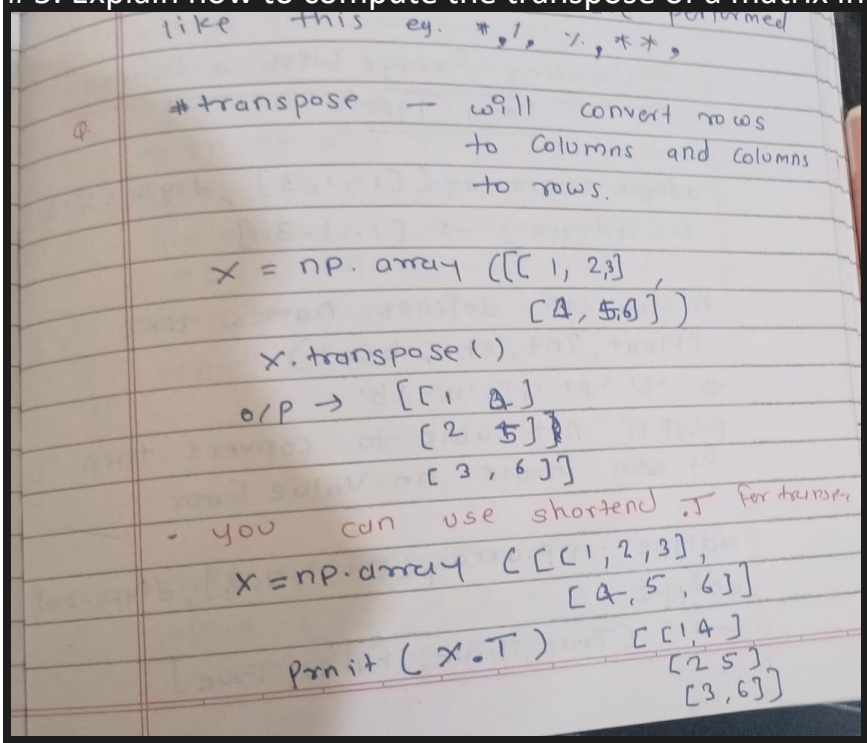
```
#uniform
print("Uniform",rd.uniform(1,100))

#normal
print(rd.normal(0,1,(2,2)))

#binomal
print(rd.binomial(10,0.5,3))


#seed
rd.seed(12)#it can be any number
print(rd.rand())
```

5. Explain how to compute the transpose of a matrix in NumPy.




```
arr=np.array([[1,2,3],[4,5,6]])
print(arr)
trans=np.transpose(arr)
print(trans)
witht=trans.T
print(witht)
# output:
# [[1 2 3]
```

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

6. Define the inverse of a matrix. When does a matrix not have an inverse?

6] inverse of Matrix → Inverse of matrix denot by A^{-1} (A) matrix should be square mat

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

change of sign

$$A^{-1} = \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

inter chang

Inverse of d is $A^{-1} = \frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$

~~eg~~ Rules for inverse

① only square matrix can have an inverse (a matrix has same no. of rows & coloums.

② The determinant is non-zero

$\det(A) = ad - bc$

$\det(A) \neq 0$

eg:- ① $A = \begin{bmatrix} 4 & 7 \\ 2 & 6 \end{bmatrix}$ - is square matrix.

step ① calculate the determinant

$(6 \times 4) - (7 \times 2) = 10$

step ③ Inverse the computale

$A^{-1} = \frac{1}{10}\begin{bmatrix} 6 & -7 \\ -2 & 4 \end{bmatrix}$

$A^{-1} \begin{bmatrix} 0.6 & -0.7 \\ -0.2 & 0.4 \end{bmatrix}$

code

```
arr = [[4,7], [2,6]]
if np.linalg.det(arr):
    P(np.linalg.det(arr)
    P(np.linalg.inv(arr)
```

O/P 10.000

```
[ [0.6  -0.7]
  [-0.2  0.4]]
```

7. What are the primary features of a pandas Series?

PAGE NO.
DATE

Q. 7] what is the primary features of a Pandas Series?

Series - is a 1D Array that can hold homogenous type elements. like int float, string.

- A Series is essentially a labeled
- All elements in Series are of the same type you can have mixed-type - but it is not recommended.
- It support vectorization (element-wise opⁿ),
- you can access series with label-based.
- elements in series are associated with index that labels the data.

8. How does a pandas Series differ from a NumPy array?

Q. 8] How does pandas Series Different from numpy array.

Series	Array numpy
① In Series is a Label based indexing	① In Array integer-based indexing.
② It can be store mixed type data	② It is homogeneous data.
③ In Series you can handle missing data like nan with funⁿ like isnan() fillna()	③ Array don't have built in support for missing data
④ In Series has metadata such as custom "name" eg. Series([1,2], name="Num")	④ Numpy array doesn't have built-in metadat
⑤ Slower that Array	⑤ faster than array
⑥ Use case is for Data Analysis & handling labele data	⑥ use case is Numerical & scientic computution

9. What is the role of the dtype parameter in a pandas Series?

Q.9] what is the role of the dtype parameter in pandas Series?

- It defines how the data is stored, processed & interpreted which affects both M/M usage & performance.

- The dtype allows you to explicitly set the data type of Series.
- if you don't specify the dtype pandas will automatically infer the data type based on the input data
- if the data is mixed or ambiguous, you might get an undesirable type eg (object).
- It improves performance & memory efficiency
eg. using int32 or float32 insted of int64 or float64 if your data doesn't require such large ranges.

eg → pd.Series([1,2,3]) → o/p int64
pd.Series([1,2,"Hello"]) → o/p object
pd.Series([1,1,2],dtype='float32') → float32

you can change after creation using astype()

s= pd.Series([1,2,3,"Hello"])
print(s) → type object

s1 = s.astype('string')
print(s1)
→ string type.
print(s1[0])type → dtype.string

```
print("Q-9----------")
s=pd.Series([1,2,3,"hello"])
print(s)
s1=s.astype('string')
print(s1)
```

```
print(type(s1[0]))

s2=pd.Series([1,1,2],dtype='float32')
print(s2)
```

10. Discuss the advantages of using pandas over NumPy for data analysis.

Q. 10] Discus the advantages of using pandas over numpy for data analysis.

→ pandas — is specifically designed for data manipulation & analysis especially with labeled heterogenous & real-world data.

Numpy — is better suitable for efficient numerical computation, homogeneous data & scientific computing tasks.

pandas — supports label axis. to make work with Real world data where labels are important to understand the context of the data.

numpy . numpy stores multidimenial data without labels which makes it less understandable for analysis.

pandas - it handles hetrogeus data.
numpy - it handles homogenus data.

Pandas - provides built-in method for handling missing data.
Numpy - There no built in function for handling missing data.

Pandas - works seamlessly with libraris like matplotlib & Seaborn. enables plotting & visualizing of data. you can directly Plot from DFs
Numpy - is used for data manipulation before plotting.

Section B: Correct the Error

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


# 2. result = np.random.randint(10, 20, size=5, replace=False)
# ANS:
# result = np.random.randint(10, 20, size=5)  #replace is present choice function


# 3. np.add([1, 2, 3], [4, 5])
# ANS
# print(np.add([1, 2], [4, 5]) )


# 4. matrix = np.array([[1, 2], [3, 4]]) print(matrix.T()
matrix = np.array([[1, 2], [3, 4]])
# ANS: print(matrix.T)

# 5. inv_matrix = np.linalg.inv([[1, 2], [3, 4]])
# inv_matrix = np.linalg.inv([[1, 2], [3, 4]])
# ANS
# print(inv_matrix)
# [[-2.   1. ]
#  [ 1.5 -0.5]]


# 6. series = pd.Series[1, 2, 3]
# ANS
series = pd.Series([1, 2, 3])
print(series)

# 7. data = {'a': 1, 'b': 2, 'c': 3} pd.Series(data, index=data.keys)
ANS
data = {'a': 1, 'b': 2, 'c': 3}
print(pd.Series(data))

# 8. random_arr = np.random.random(2.5)
ANS
random_arr = np.random.rand(2,5)
```

```
print(random_arr)
print(np.random.random(2))#random takes only 1 positional argument

# 9. matrix = np.array([1, 2], [3, 4])
ANS
matrix = np.array([[1, 2], [3, 4]])

# 10. print(pd.Series([1, 2, 3, 4], dtype=int64))
ANS
print(pd.Series([1, 2, 3, 4], dtype='int64'))#you  have to give it in string form if you giving with
bytes other wise int is ok
```

Section C: Write a Program

```
from numpy import random as rd
# 1. Create a 3x3 NumPy array filled with random integers between 1 and 10.
# arr=rd.randint(1,10,(3,3))
# print(arr)
# output:
# [[3 2 1]
#  [9 1 5]
#  [4 8 4]]


# 2. Write a program to find the transpose of a 4x4 matrix.
# arr=np.arange(1,17).reshape(4,4)
# print(arr.T)
# output:
# [[ 1  5  9 13]
#  [ 2  6 10 14]
#  [ 3  7 11 15]
#  [ 4  8 12 16]]


# 3. Generate a NumPy array of 10 random floats between 0 and 1.
# print(rd.rand(10))
# output:
# [0.57443462 0.19231697 0.25399767 0.39233393 0.26131864 0.12850515
#  0.24464098 0.6318173  0.26389712 0.31208749]
```

```
# 4. Write a program to compute the inverse of a 2x2 matrix.
# arr=rd.randint(1,10,(2,2))
# print(np.linalg.inv(arr))
# output:
# [[ 1.16666667 -0.33333333]
#  [-0.66666667  0.33333333]]


# 5. Create a NumPy array and apply the np.sqrt ufunc on all its elements.
# arr=np.array([1,2,3,4,5,8,16])
# print(np.sqrt(arr))
# output:
# [1.        1.41421356 1.73205081 2.        2.23606798 2.82842712
#  4.        ]


# 6. Write a program to create a pandas Series from a Python dictionary.
d={'Nitis':78,'Lavanya':90,'Samarth':98,'Kirana':34}
# print(pd.Series(d))
# output:
# Nitis      78
# Lavanya    90
# Samarth    98
# Kirana     34
# dtype: int64


# 7. Create a pandas Series with custom indices and retrieve values using indices.
# s=pd.Series([78,89,90,56],index=['nitis','samarth','lavanya','kiran'])
# print(s['lavanya'])
# output:90

# 8. Generate a 5x5 identity matrix using NumPy.
# print(np.eye(5,5,dtype=int))
# output:
# [[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]]
# 9. Write a program to create a NumPy array of even numbers between 10 and 30.
# arr=np.arange(10,30,2)
# print(arr)
```

```
# output:
# [10 12 14 16 18 20 22 24 26 28]


# 10. Create a pandas Series from a NumPy array and display its dtype.
# nprray=np.array([1,2,3,4,5])
# arr=pd.Series(nprray)
# print(arr.dtype)
# print(nprray.dtype)
# output:
# 0    1
# 1    2
# 2    3
# 3    4
# 4    5
# dtype: int64
# int64
# int64
```