Code Example:

Numpy

Basics of Numpy

- Import numpy and check version, using np.**version**
- Declare an array, using np.array() and check its different attributes.

```
import numpy as np
print("version = ",np.__version__)

a= np.array([1,2,3,4,5])
print('array = ', a)
print('array shape = ',a.shape)
print('array data type = ',a.dtype)
print('number of dimension = ',a.ndim)
print('array size = ',a.size)
print('bytesize of datatype = ',a.itemsize)
```

```
version =  1.22.4
array =  [1 2 3 4 5]
array shape =  (5,)
array data type =  int64
number of dimension =  1
array size =  5
bytesize of datatype =  8
```

- Array Index

```
print('3rd elemnt = ',a[3])
a[3]=9
print('Modified 3rd element = ',a[3])
```

```
9
9
[1 2 3 9 5]
[ 2  6 12 45 30]
```

- Array initialization

```
b=a*np.array([2,3,4,5,6])
print('a = ',a)
print('b = ',b)
```

```
a =  [1 2 3 4 5]
b =  [ 2  6 12 20 30]
```

Array Vs List

- Append

```
l = [1,2,3]
a = np.array([1,2,3])
#appends 4 at the end of list
l.append(4)
print('l = ',l)
#error! no such attribute
a.append(4)
print('a = ',a)
```

```
l =  [1, 2, 3, 4]
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-6-fde5852968b7> in <cell line: 7>()
      5 print('l = ',l)
      6 #error! no such attribute
----> 7 a.append(4)
      8 print('a = ',a)
```

- Same process different output

```
l=[1,2,3]
a=np.array([1,2,3])
l=l+[4]
print('l=l+[4] ==>',l)
a=a+[4]
print('a=a+[4] ==>',a)
l=[1,2,3]
a=np.array([1,2,3])
l=l*2
print('l=l*2 ==>' ,l)
a=a*2
print('a=a*2 ==>',a)
```

```
l=l+[4] ==> [1, 2, 3, 4]
a=a+[4] ==> [5 6 7]
l=l*2 ==> [1, 2, 3, 1, 2, 3]
a=a*2 ==> [2 4 6]
```

Dot Product

```
l1 = [1,2,3]
l2 = [2,3,4]
a1=np.array(l1)
a2=np.array(l2)
dot=0
#using for loop
for i in range(len(l1)):
  dot+=l1[i]*l2[i]
print(dot)

#using dot function
dot = np.dot(a1,a2)
print(dot)

#using multiplication and sum
dot = (a1*a2).sum()
print(dot)
```

```
20
20
20
```

Multidimentional Array

- Declaration

```
a=np.array([[1,2],[4,5]])
print(a)
print(a.shape)
```

```
[[1 2]
 [4 5]]
(2, 2)
```

- Transpose, Inverse, Determinant and Diagonal Matrix

```
print('Transpose \n',a.transpose())
print('Inverse \n',np.linalg.inv(a)) #a must be square matrix
print('Determinant \n', np.linalg.det(a))
print('Diagonal elements \n', np.diag(a))
c=np.diag(a)
print('Diagonal matrix \n', np.diag(c))
```

```
Transpose
 [[1 4]
 [2 5]]
Inverse
 [[-1.66666667  0.66666667]
 [ 1.33333333 -0.33333333]]
Determinant
 -2.9999999999999996
Diagonal elements
 [1 5]
Diagonal matrix
 [[1 0]
 [0 5]]
```

Array Slicing/ Boolean Indexing

- Array slicing

```
print('column 0 of all rows \n', a[:,0])
print('row 0 all elements \n', a[0,:])
```

```
column 0 of all rows
 [1 4]
row 0 all elements
 [1 2]
```

- Boolean Indexing, Check if the elemnts are true of false for a condition

```
a=np.array([[1,2],[3,4],[5,6]])
bool_idx=a>2
print('Boolean Index \n',bool_idx)
print('Output = ',a[bool_idx])
#or
print('Similar output for code, a[a>2] = ',a[a>2])
#for keeping the shape right
b=np.where(a>2, a, -1)
#print elements of a if a[i]>2
#else print -1
print('Matrix view \n',b)
```

```
Boolean Index
 [[False False]
 [ True  True]
 [ True  True]]
Output =  [3 4 5 6]
Similar output for code, a[a>2] =  [3 4 5 6]
Matrix view
 [[-1 -1]
 [ 3  4]
 [ 5  6]]
```

Reshaping

```
a=np.arange(1,7)
print('a = ',a)
print('Shape of a = ',a.shape)
b=a.reshape(3,2)
print('b, reshaped with 3 rows and 2 colums \n',b)
#row fit, column 1
b=a[:, np.newaxis]
print('Row fit ==> column=1\n',b)
#row 1 column fit
b=a[np.newaxis, :]
print('Column fit ==> row=1\n',b)
```

```
a =  [1 2 3 4 5 6]
Shape of a =  (6,)
b, reshaped with 3 rows and 2 colums
 [[1 2]
 [3 4]
 [5 6]]
Row fit ==> column=1
 [[1]
 [2]
 [3]
 [4]
 [5]
 [6]]
Column fit ==> row=1
 [[1 2 3 4 5 6]]
```

Concatenation

- Adding new column or row

```
a=np.array([[1,2],[2,3]])
b=np.array([[5,6]])
print('a\n',a)
print('b\n',b)
#adding as a new row
c=np.concatenate((a,b))
print('c=a+b\n',c)
#axis=0 gives same output
c=np.concatenate((a,b),axis=0)
print('Putting axis = 0\n',c)
#axis=None, makes 1D array
c=np.concatenate((a,b),axis=None)
print('Putting axis = None\n',c)
#axis=1 concatenates new column at end
c=np.concatenate((a,b.T), axis=1)
print('Putting axis = 1\n',c)
```

```
a
 [[1 2]
 [2 3]]
b
 [[5 6]]
c=a+b
 [[1 2]
 [2 3]
 [5 6]]
Putting axis = 0
 [[1 2]
 [2 3]
 [5 6]]
Putting axis = None
 [1 2 2 3 5 6]
Putting axis = 1
 [[1 2 5]
 [2 3 6]]
```

- Stack, using hstack or vstack

```
a=np.array([1,2,3,4])
b=np.array([5,6,7,8])
#hstack=Horizontal
c=np.hstack((a,b))
print('hstack = ',c)
#vstack=vertical
c=np.vstack((a,b))
print('vstack = \n',c)
```

```
hstack =  [1 2 3 4 5 6 7 8]
vstack =
 [[1 2 3 4]
 [5 6 7 8]]
```

Broadcasting

- To work with arrays of different shapes, while performing arithmetic operations.

```
a=np.array([[1,2,3],[4,2,5],[3,2,1]])
b=np.array([1,0,2])
#without describing, np is
#adding b to all rows of a
c=a+b
print(c)
```

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

Functions and Axis

- Sum

```
a=np.array([[7,8,9,10,11,12,13],[17,18,19,20,21,22,23]])
print('a = \n',a)
#sum
print('a.sum() = ',a.sum())
#same as..
print('a.sum(axis = None)\n',a.sum(axis=None))
#sum of each col
print('a.sum(axis = 0)\n',a.sum(axis=0))
#sum of each row
print('a.sum(axis = 1)\n',a.sum(axis=1))
```

```
    a =
     [[ 7  8  9 10 11 12 13]
     [17 18 19 20 21 22 23]]
    a.sum() =  210
    a.sum(axis = None)
     210
    a.sum(axis = 0)
     [24 26 28 30 32 34 36]
    a.sum(axis = 1)
     [ 70 140]
```

- Similarly we can calculate, mean, variance, standerd deviation or min/max of any array.

```
print('Mean = ',a.mean())
print('Variance = ',a.var(axis=None))
print('Standerd deviation = ', a.std(axis=None))
print('Min = ',a.min(axis=None),' Max = ',a.max(axis=None))
```

```
    Mean =  15.0
    Variance =  29.0
    Standerd deviation =  5.385164807134504
    Min =  7  Max =  23
```

Copying

- If we copy an array into another, they share the same location in the memory. So in case of modi- fying one instance, the other will also be modified.

```
a=np.array([2,3,4])
b=a
b[0]=23
print('b = ',b)
print('a = ',a)
```

```
    b =  [23  3  4]
    a =  [23  3  4]
```

- For avoiding this situation and making an actual copy..

```
a=np.array([2,3,4])
b=a.copy()
b[0]=23
print('b = ',b)
print('a = ',a)
```

```
b =  [23  3  4]
a =  [2 3 4]
```

Generating Arrays

- Array innitialization

```
#Generate an array of size 2/3, with all elements=0
a=np.zeros((2,3))
print('2/3 array filled with zeros \n',a)
#all elements=1
a=np.ones((2,3))
print('2/3 array filled with ones \n',a)
#The datatype for zeros and ones are by default set to float
#all elements=x
a=np.full((2,3),5)
print('2/3 array filled with 5 \n',a)
```

```
2/3 array filled with zeros
 [[0. 0. 0.]
 [0. 0. 0.]]
2/3 array filled with ones
 [[1. 1. 1.]
 [1. 1. 1.]]
2/3 array filled with 5
 [[5 5 5]
 [5 5 5]]
```

- Generate Identity Matrix

```
a=np.eye(3)
print('Identity matrix \n',a)
```

- Innitialize with consecutive numbers

```
#from 0 to n
a=np.arange(20)
print('a{0,1,...19} = ',a)
#from l to r
a=np.arange(5,10)
print('a{5,6,7...9} = ',a)
#Equally spaced array
a= np.linspace(0,10,5)
print('5 equally spaced elements between 0 and 10 = ',a)
```

```
a{0,1,...19} =  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
a{5,6,7...9} =  [5 6 7 8 9]
5 equally spaced elements between 0 and 10 =  [ 0.   2.5  5.   7.5 10. ]
```

- Generate an arrray filled with random numbers
- Here the only difference is in how the argu- ments are handled.With numpy. ran- dom. rand(), the length of each dimen- sion of the output array is a separate argument.

```
#uniform distribution beween 0 and 1
a=np.random.random((3,2))
print('a uniformly distributed beween 0 and 1\n',a)

#Gaussian distribution, any real number
a=np.random.randn(3,2)
print('\na any real number\n',a)

#Random integers in between l and r-1
a=np.random.randint(3,10,size=(3,3))
```

```
print('\na is Random Integer\n',a)

#Random Choice in range
a=np.random.choice(5,size=10)
print('\na choosen randomly from 0 to 5\n',a)

#This can also choose from list
a=np.random.choice([2,-4,3,100], size=10)
print('\na is chosen randomly from list [2,-4,3,100]\n',a)
```

```
a uniformly distributed beween 0 and 1
 [[0.78586623 0.3643819 ]
 [0.74900192 0.55220098]
 [0.26042692 0.69599694]]

a any real number
 [[-3.42048306  0.88098196]
 [ 0.71667912 -0.58293655]
 [-0.50064208 -0.42245908]]

a is Random Integer
 [[6 8 4]
 [6 6 8]
 [8 8 8]]

a choosen randomly from 0 to 5
 [3 4 0 0 0 3 2 1 3 1]

a is chosen randomly from list [2,-4,3,100]
 [100  -4   2   3   3 100   3  -4  -4 100]
```

Linear Algebra

- Eigen values

```
#Eigen Value, (used in machine learning, and apply PCA (Principal component anaysis) algorithm)
a=np.array([[1,2],[3,4]])
eigenvalues,eigenvectors=np.linalg.eig(a)
print("eigenvalues",eigenvalues)
print("eigenvectors",eigenvectors) #column vector

#e_vec*e_val=A*e_vec
b=eigenvectors[:,0]*eigenvalues[0]
c=a @ eigenvectors[:,0]
print("b = ",b)
print("c = ",c)
#print(b==c), this is not appropiate style to check
print(np.allclose(b,c))
```

```
eigenvalues [-0.37228132  5.37228132]
eigenvectors [[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
b =  [ 0.30697009 -0.21062466]
c =  [ 0.30697009 -0.21062466]
True
```

- Solving linear system equations

```
suppose,
x1+x2 =2200
1.50*x1+4.0*x2=5050
so if, x = [[x1],[x2]], A = [[1, 1],[1.50, 4.0]], b =[2200,5050]
  x*A = b
=>x = A^-1*b
```

```
#mannual way
A=np.array([[1,1],[1.5,4.0]])
b=np.array([2200,5050])
```

```
x=np.linalg.inv(A).dot(b)
print('x1,x2 = ',x)

#using inverse is slow, and also may give numerical issues
#better way
x= np.linalg.solve(A,b)
print('using solve() x1,x2 = ',x)
#check
print(np.allclose(np.dot(A,x),b))
```

```
x1,x2 =  [1500.  700.]
using solve() x1,x2 =  [1500.  700.]
True
```