numpy

July 22, 2024

1 Numpy

[1]:
```python
import numpy as np
```

1.1 Array Creation

[2]:
```python
a=np.array([1,2,3])
print(type(a))                    # Shows type
```

```
<class 'numpy.ndarray'>
```

[3]:
```python
print("Array :\n",np.array([1,2,3]))                # Create a array from␣
 ↪an object
print("Zero Array :\n",np.zeros((2,2)))             # Create a 2X2 matrix␣
 ↪with zeros
print("Ones Array :\n",np.ones((2,2)))              # Create a 2X2 matri␣
 ↪with ones
```

```
Array :
 [1 2 3]
Zero Array :
 [[0. 0.]
 [0. 0.]]
Ones Array :
 [[1. 1.]
 [1. 1.]]
```

[4]:
```python
print("Empty Array :\n",np.empty([2,2],dtype=int))      # Create a array of␣
 ↪shape with out initializing entries
print("Array :\n",np.full((2,2),fill_value=2))          # Returns a array of␣
 ↪given shape with fill_value
print("Array :\n",np.arange(0,7,2))                     # Returns a array of␣
 ↪evenly spaced with given interval
```

```
Empty Array :
 [[ 773104814 -508332684]
 [2141814533 1087262674]]
```

```
Array :
 [[2 2]
 [2 2]]
Array :
 [0 2 4 6]
```

[5]:
```
print('Linspace \t:',np.linspace(1,2,5))     # Returns a evenly spaced numbers␣
 ↪over a specified interval
print('Logspace \t:',np.logspace(2,3,5))     # Returns numbers spaced evenly on␣
 ↪a log scale
```

```
Linspace        : [1.   1.25 1.5  1.75 2.  ]
Logspace        : [ 100.          177.827941    316.22776602  562.34132519 1000.
]
```

1.2 Array Manipulation

Consider a=np.arange(0,10)

[6]:
```
a=np.arange(6)
```

[7]:
```
print('Reshape :\n',np.arange(6).reshape((3, 2)))
print('Transpose : \n',np.arange(6).reshape((3,2)).transpose())
```

```
Reshape :
 [[0 1]
 [2 3]
 [4 5]]
Transpose :
 [[0 2 4]
 [1 3 5]]
```

[8]:
```
print("Concatenate :\n",np.concatenate((a,a),axis=0))
print('Stack :\n',np.stack((a,a),axis=0))
print('Stack :\n',np.stack((a,a),axis=1))
```

```
Concatenate :
 [0 1 2 3 4 5 0 1 2 3 4 5]
Stack :
 [[0 1 2 3 4 5]
 [0 1 2 3 4 5]]
Stack :
 [[0 0]
 [1 1]
 [2 2]
 [3 3]
 [4 4]
 [5 5]]
```

[9]:
```python
print("Split :\n",np.split(a,3))
print("Flip :\n",np.flip(a))
print("Roll :\n",np.roll(a,shift=2))
```

```
Split :
 [array([0, 1]), array([2, 3]), array([4, 5])]
Flip :
 [5 4 3 2 1 0]
Roll :
 [4 5 0 1 2 3]
```

1.3 Mathematical Functions

consider a=np.arange(6)

[10]:
```python
a=np.arange(6)
print("Sum \t\t:",np.sum(a))
print("Squareroot \t:",np.sqrt(a))
print("Sin \t\t:",np.sin(a))
print("Absolute \t:",np.abs(-a))
print("Exponent \t:",np.exp(a))
```

```
Sum             : 15
Squareroot      : [0.         1.         1.41421356 1.73205081 2.
2.23606798]
Sin             : [ 0.          0.84147098  0.90929743  0.14112001 -0.7568025
-0.95892427]
Absolute        : [0 1 2 3 4 5]
Exponent        : [  1.           2.71828183   7.3890561   20.08553692
54.59815003
 148.4131591 ]
```

[11]:
```python
print("Natural Log \t:",np.log(a[1:]))
print("Sin inverse\t:",np.arcsin(np.linspace(0,1,5)))
print("Power \t\t:",np.power(a,2))
```

```
Natural Log     : [0.         0.69314718 1.09861229 1.38629436 1.60943791]
Sin inverse     : [0.         0.25268026 0.52359878 0.84806208 1.57079633]
Power           : [ 0  1  4  9 16 25]
```

1.4 Statistical Function

consider a=np.random.randint(1,100,16).reshape(4,4)

[12]:
```python
a=np.random.randint(1,100,16).reshape(4,4)
print(a)
```

```
[[20 11 65 38]
 [73 89 63  9]
```

```
 [86 47 98 81]
 [38 26 71 25]]
```

[13]:
```python
# Median
print('Median \t\t\t:',np.median(a))
print("Median along 0 axis\t:",np.median(a,axis=0))
print("Median along 1 axis\t:",np.median(a,axis=1))
```

```
Median                  : 55.0
Median along 0 axis     : [55.5 36.5 68.  31.5]
Median along 1 axis     : [29.  68.  83.5 32. ]
```

[14]:
```python
# Standard Deviation
print("Standard Deviation \t\t:",np.std(a))
print("Standard Deviation along 0 axis\t:",np.std(a,axis=0))
print("Standard Deviation along 1 axis\t:",np.std(a,axis=1))
```

```
Standard Deviation              : 28.445122604763018
Standard Deviation along 0 axis : [26.44215385 29.34599632 14.02453208
26.73363986]
Standard Deviation along 1 axis : [20.62159063 30.04579838 18.93409623
18.61451047]
```

[15]:
```python
# Histogram
print("Histogram \t\t:",np.histogram(a))
```

```
Histogram               : (array([2, 3, 0, 2, 1, 0, 3, 1, 3, 1], dtype=int64),
array([ 9. , 17.9, 26.8, 35.7, 44.6, 53.5, 62.4, 71.3, 80.2, 89.1, 98. ]))
```

[16]:
```python
# Mean
print("Mean \t:",np.mean(a))
print("Mean along axis 0",np.mean(a,axis=0))
print("Mean along axis 1",np.mean(a,axis=1))
```

```
Mean    : 52.5
Mean along axis 0 [54.25 43.25 74.25 38.25]
Mean along axis 1 [33.5 58.5 78.  40. ]
```

[17]:
```python
# Percentile
print("Percentile",np.percentile(a,100))
print("Percentile",np.percentile(a,100,axis=0))
print("Percentile",np.percentile(a,100,axis=1))
```

```
Percentile 98.0
Percentile [86. 89. 98. 81.]
Percentile [65. 89. 98. 71.]
```

[18]:
```
# Mean
print("Mean \t\t\t:",np.mean(a))
print("Mean along axis 0 \t:",np.mean(a,axis=0))
print("Mean along axis 1 \t:",np.mean(a,axis=1))
```

```
Mean                    : 52.5
Mean along axis 0       : [54.25 43.25 74.25 38.25]
Mean along axis 1       : [33.5 58.5 78.  40. ]
```

[19]:
```
# Variance
print("Variance \t\t:",np.var(a))
print("Variance along 0 axis\t:",np.var(a,axis=0))
print("Variance along 1 axis\t:",np.var(a,axis=1))
```

```
Variance                : 809.125
Variance along 0 axis   : [699.1875 861.1875 196.6875 714.6875]
Variance along 1 axis   : [425.25 902.75 358.5  346.5 ]
```

[20]:
```
# Corrcoef
print("Corrcoef",np.corrcoef(a))
```

```
Corrcoef [[ 1.         -0.38856205  0.7894713   0.81474844]
 [-0.38856205  1.         -0.39023359  0.19042074]
 [ 0.7894713  -0.39023359  1.          0.70435736]
 [ 0.81474844  0.19042074  0.70435736  1.        ]]
```

1.5 Linear Algebra

Consider A = np.array([[1, 2], [3, 4]]) B = np.array([[5, 6], [7, 8]])

[21]:
```
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
```

[22]:
```
A,B
```

[22]:
```
(array([[1, 2],
        [3, 4]]),
 array([[5, 6],
        [7, 8]]))
```

[23]:
```
# Dot product
print("Dot product :\n",np.dot(A,B))
```

```
Dot product :
 [[19 22]
 [43 50]]
```

[24]:
```python
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
```

[25]:
```python
# Compute the inverse of matrix A
A_inv = np.linalg.inv(A)
print("Inverse of matrix A:")
print(A_inv)
```

```
Inverse of matrix A:
[[-2.   1. ]
 [ 1.5 -0.5]]
```

[26]:
```python
# Compute the determinant of matrix A
det_A = np.linalg.det(A)
print("\nDeterminant of matrix A:", det_A)
```

```

Determinant of matrix A: -2.0000000000000004
```

[27]:
```python
# Solve a system of linear equations Ax = B
B_vector = np.array([5, 6])
x = np.linalg.solve(A, B_vector)
print("\nSolution to Ax = B:")
print(x)
```

```

Solution to Ax = B:
[-4.   4.5]
```

[28]:
```python
# Compute the eigenvalues and eigenvectors of matrix A
eigenvalues, eigenvectors = np.linalg.eig(A)
print("\nEigenvalues of matrix A:")
print(eigenvalues)
print("\nEigenvectors of matrix A:")
print(eigenvectors)
```

```

Eigenvalues of matrix A:
[-0.37228132  5.37228132]

Eigenvectors of matrix A:
[[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
```

[29]:
```python
# Perform Singular Value Decomposition (SVD) on matrix B
U, S, V = np.linalg.svd(B)
print("\nSingular Value Decomposition (SVD) of matrix B:")
print("U:")
```

```
print(U)
print("S (singular values):")
print(S)
print("V (conjugate transpose):")
print(V)
```

```
Singular Value Decomposition (SVD) of matrix B:
U:
[[-0.59206014 -0.80589378]
 [-0.80589378  0.59206014]]
S (singular values):
[13.19003444  0.15162963]
V (conjugate transpose):
[[-0.65212545 -0.75811107]
 [ 0.75811107 -0.65212545]]
```

[30]:
```
# Compute the QR decomposition of matrix B
Q, R = np.linalg.qr(B)
print("\nQR Decomposition of matrix B:")
print("Q:")
print(Q)
print("R:")
print(R)
```

```

QR Decomposition of matrix B:
Q:
[[-0.58123819 -0.81373347]
 [-0.81373347  0.58123819]]
R:
[[-8.60232527 -9.99729693]
 [ 0.          -0.23249528]]
```

[31]:
```
# Norm of a vector
norm_v = np.linalg.norm(A)
print("Norm of vector v:", norm_v)
print()

# Frobenius norm of a matrix
norm_A = np.linalg.norm(A, 'fro')
print("Frobenius norm of Matrix A:", norm_A)
```

```
Norm of vector v: 5.477225575051661

Frobenius norm of Matrix A: 5.477225575051661
```

[32]:
```python
# Matrix Multiplication
print(np.matmul(A,B))
```

```
[[19 22]
 [43 50]]
```

[33]:
```python
# Cross Multiplication
print(np.cross(A,B))
```

```
[-4 -4]
```

[34]:
```python
# Inner Product
print(np.inner(A,B))
```

```
[[17 23]
 [39 53]]
```

1.6 Random Sampling

[35]:
```python
a=np.arange(9)
```

[36]:
```python
print("Rand [0,1]\t:",np.random.rand(4))
print("Randn \'Float\'\t:",np.random.randn(4))
print("Randint 'Integer b/w ranges'\t:",np.random.randint(0,4,4))
print("Random Choice \t:",np.random.choice(a))
print('Random Shuffle',np.random.shuffle(a),a)
```

```
Rand [0,1]      : [0.94489136 0.02063174 0.76720175 0.80372186]
Randn 'Float'   : [-1.41650913 -0.64956613 -0.77400505 -0.32202509]
Randint 'Integer b/w ranges'    : [3 1 1 2]
Random Choice   : 8
Random Shuffle None [8 7 4 0 1 2 3 6 5]
```

[37]:
```python
print("Seed \t:",np.random.seed(1))
print("Permutation :",np.random.permutation(9))
print("Normal Gaussian Distribution \t:",np.random.normal(2))
print("Binomial Distribution :",np.random.binomial(10,0.5,10))
print("Exponential :",np.random.exponential(4))
print("Poisson :",np.random.poisson(a))
```

```
Seed    : None
Permutation : [8 2 6 7 1 0 4 3 5]
Normal Gaussian Distribution    : 1.4502538233401552
Binomial Distribution : [5 6 7 7 4 5 5 4 5 7]
Exponential : 2.4440928352447364
Poisson : [11  6  2  0  2  1  1  4  8]
```

1.7 Bonus

[38]:
```
a=np.arange(9).reshape(3,3)
a
```

[38]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

[39]:
```
print("Broadcast_to :\n",np.broadcast_to([1,2,3],shape=[3,3]))  # Broadcasts an␣
 ↪array to new shape
print("Delete \t\t:",np.delete(a,[1,3,5]))                      # Deletes␣
 ↪elements along an axis of an array
print("Insert \t\t:",np.insert(a,1,9))                          # Inserts value␣
 ↪to the specified index
print("Append \t\t:",np.append(a,a))                            # Appends␣
 ↪values to the end
print("Gradient \t:",np.gradient([1,3]))                        # Computes␣
 ↪numerical gradient of the array
```

```
Broadcast_to :
 [[1 2 3]
 [1 2 3]
 [1 2 3]]
Delete          : [0 2 4 6 7 8]
Insert          : [0 9 1 2 3 4 5 6 7 8]
Append          : [0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8]
Gradient        : [2. 2.]
```

[40]:
```
# Define two arrays
a = np.array([1.0, 2.0, 3.0])
b = np.array([1.0, 2.01, 3.00001])

# Check if elements are close within a default tolerance
print(np.isclose(a,b))
```

```
[ True False  True]
```

[41]:
```
print("Identity Matrix :\n",np.eye(3))
```

```
Identity Matrix :
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

I hope you found this information helpful! Feel free to save this post for future reference. Let's continue to learn and grow together!

Rajendra Prasad