numpy-3

November 16, 2025

[2]:
```
# Inverse
```

[ ]:
```
# Q. Make a matrix, A = 3x3
# Make a Indentity matrix, I = 3x3

# Check for

# A * I = A
# A * invA = I
```

[7]:
```
import numpy as np


A = np.array([1, 2, 3, 4, 5, 3, 1, 7, 9]).reshape(3, 3)
I = np.eye(3)

print(A)
print(I)
```

```
[[1 2 3]
 [4 5 3]
 [1 7 9]]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

[8]:
```
np.matmul(A, I)
```

[8]:
```
array([[1., 2., 3.],
       [4., 5., 3.],
       [1., 7., 9.]])
```

[ ]:
```
# Determinant of matrix

np.linalg.det(A)
```

[ ]:
```
np.float64(27.0)
```

[10]:
```
invA = np.linalg.inv(A)
invA
```

[10]:
```
array([[ 0.88888889,  0.11111111, -0.33333333],
       [-1.22222222,  0.22222222,  0.33333333],
       [ 0.85185185, -0.18518519, -0.11111111]])
```

[11]:
```
np.matmul(A, invA)
```

[11]:
```
array([[ 1.00000000e+00,  2.77555756e-17,  4.16333634e-17],
       [-1.11022302e-16,  1.00000000e+00,  4.16333634e-17],
       [-3.33066907e-16, -2.77555756e-17,  1.00000000e+00]])
```

[ ]:
```
# Transpose of Matrix
A.T
```

[ ]:
```
array([[1, 4, 1],
       [2, 5, 7],
       [3, 3, 9]])
```

[13]:
```
A.sum()
```

[13]:
```
np.int64(35)
```

[14]:
```
A.mean()
```

[14]:
```
np.float64(3.888888888888889)
```

[15]:
```
A.std()
```

[15]:
```
np.float64(2.5579698740491863)
```

[16]:
```
A.min()
```

[16]:
```
np.int64(1)
```

[17]:
```
A.max()
```

[17]:
```
np.int64(9)
```

[18]:
```
# Row sum

A.sum(axis=0)
```

[18]:
```
array([ 6, 14, 15])
```

[19]:
```
# Col sum

A.sum(axis=1)
```

[19]:
```
array([ 6, 12, 17])
```

[ ]:
```
np.linalg.eig(A)

# A.v = lemda.v
```

[ ]:
```
EigResult(eigenvalues=array([13.09569004+0.j        ,  0.95215498+1.07477808j,
        0.95215498-1.07477808j]), eigenvectors=array([[-0.28447549+0.j        ,
0.2940061 -0.27326013j,
         0.2940061 +0.27326013j],
       [-0.45354298+0.j        , -0.71082121+0.j        ,
        -0.71082121-0.j        ],
       [-0.8446138 +0.j        ,  0.5670899 +0.10968848j,
         0.5670899 -0.10968848j]]))
```

[ ]:
```
arr = np.array([1, 2, 3, 4])

# Cumulative prod
np.cumprod(arr)
```

[ ]:
```
array([ 1,  2,  6, 24])
```

[ ]:
```
# Comulative sum
np.cumsum(arr)
```

[ ]:
```
array([ 1,  3,  6, 10])
```

[23]:
```
np.nan
```

[23]:
```
nan
```

[24]:
```
np.inf
```

[24]:
```
inf
```

[25]:
```
arr = np.array([1, 2, 3, np.nan, 4, 5, np.nan])
arr
```

[25]:
```
array([ 1.,  2.,  3., nan,  4.,  5., nan])
```

[27]:
```
# Handling missing data

mask = np.isnan(arr)

arr[mask] = np.nanmean(arr)
```

[28]:
```
arr
```

[28]:
```
array([1., 2., 3., 3., 4., 5., 3.])
```

[29]:
```
# Removing missing data

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

mask = ~np.isnan(arr)

arr[mask]
```

[29]:
```
array([1., 2., 3., 4., 5.])
```

[30]:
```
# 3  1 + 5 2 - 9  3 = 6
# -3  1 + 7  3 = -2
#  -   2  + 4  3  = 8
```

[32]:
```
A = np.array([[3, 5, -9], [-3, 0, 7], [0, -1, 4]])
B = np.array([[6], [-2], [8]])
```

[33]:
```
invA = np.linalg.inv(A)
```

[34]:
```
np.matmul(invA, B)
```

[34]:
```
array([[6.37037037],
       [1.77777778],
       [2.44444444]])
```

[35]:
```
np.linalg.solve(A, B)
```

[35]:
```
array([[6.37037037],
       [1.77777778],
       [2.44444444]])
```

1 Statistics and Probability

1.0.1 Types of stats

1. Descriptive stats
   - summarize data (mean, median, mode)
2. Inferential stats
   - Make predictions or inference (hypothesis testing, confidence interval)
3. Parametric stats
   - Assumes data follows a distribution like normal (t-test, ANOVA)
4. Non-parametric stats
   - No assumption of data distribution (median test, Wilcoxon test)

[38]:
```
# Measure of central tendency
```

```
# mean -> if there is outlier we dont use mean,
# median,
# mode

data = np.array([1, 2, 1, 4, 5, 6, 5, 5, 9])

mean = np.mean(data)
median = np.median(data)
mode = np.bincount(data).argmax()
```

[39]:
```
print(mean)
print(median)
print(mode)
```

```
4.222222222222222
5.0
5
```

[ ]: