Tip

Compilation note: This Note Is Created By Saurab Gyawali To Help Other Fellows Like Him Who's Struggling To Learn Data Science From Basics Covering All Topics . Feel Free To Contact - saurabgyawali77@gmail.com

1 Basics

Exercise 1: Import the numpy package under the name `np`.

Problem

Import NumPy.

Solution

```
import numpy as np

# quick check
print(np.__version__)
```

Exercise 2: Print the NumPy version and configuration.

Solution

```
import numpy as np
print(np.__version__)
np.show_config()
```

Exercise 3: Create a null vector of size 10.

Solution

```
Z = np.zeros(10)
print(Z)
```

Exercise 4: How to get documentation of `numpy.add` from command line?

Solution

Use Python's `-c` to call `numpy.info`: `python -c "import numpy; numpy.info(numpy.add)"`.

Exercise 5: Create a null vector of size 10 but set the fifth value to 1.

Solution

```
Z = np.zeros(10)
Z[4] = 1
print(Z)
```

Exercise 6: Create a vector with values ranging from 10 to 49.

Solution

```
Z = np.arange(10, 50)
print(Z)
```

Exercise 7: Reverse a vector (first element becomes last).

Solution

```
Z = np.arange(50)
Z = Z[::-1]
```

Exercise 8: Create a 3x3 matrix with values ranging from 0 to 8.

Solution

```
Z = np.arange(9).reshape(3,3)
print(Z)
```

Exercise 9: Find indices of non-zero elements from [1,2,0,0,4,0].

Solution

```
nz = np.nonzero([1,2,0,0,4,0])
print(nz)
```

Exercise 10: Create a 3x3 identity matrix.

Solution

```
Z = np.eye(3)
print(Z)
```

Exercise 11: Create a 1-D array of numbers from 0 to 9.

Solution

```
np.arange(10)
```

Exercise 12: Create a 3x3x3 array with random values.

Solution

```
np.random.random((3,3,3))
```

Exercise 13: Create a 10x10 array with random values and find the minimum and maximum.

Solution

```
Z = np.random.random((10,10))
Z.min(), Z.max()
```

Exercise 14: Create a random vector of size 30 and find the mean value.

Solution

```
Z = np.random.random(30)
Z.mean()
```

Exercise 15: Create a 2d array with 1 on the border and 0 inside.

Solution

```
Z = np.ones((10,10))
Z[1:-1,1:-1] = 0
```

Exercise 16: Multiply a 5x3 matrix by a 3x2 matrix (real matrix product).

Solution

```
A = np.ones((5,3))
B = np.ones((3,2))
A.dot(B)
```

Exercise 17: Create a 1-D array of 10 random integers between 0 and 9.

Solution

```
np.random.randint(0,10,10)
```

Exercise 18: Create a 10x10 matrix with random values and find the minimum and maximum along each row.

Solution

```
Z = np.random.random((10,10))
Z.min(axis=1), Z.max(axis=1)
```

Exercise 19: Create a structured array representing a student record (name, age, marks).

Solution

```
dt = np.dtype([('name','U10'),('age','i4'),('marks','f4')])
students = np.array([('Alice',21,88.5),('Bob',22,75.0)], dtype=
    dt)
```

Exercise 20: Convert a list of tuples to a NumPy array with a specified dtype.

Solution

```
data = [(1,2.0),(3,4.5)]
np.array(data, dtype=[('a','i4'),('b','f4')])
```

Exercise 21: Find common elements between two arrays.

Solution

```
np.intersect1d([1,2,3,4],[3,4,5,6])
```

Exercise 22: Create a random vector and sort it.

Solution

```
v = np.random.random(10)
v.sort()
```

Exercise 23: Create random 2D array and sort by the second column.

Solution

```
Z = np.random.random((5,3))
Z[Z[:,1].argsort()]
```

Exercise 24: Compute the dot product of two arrays.

Solution

```
a = np.arange(5)
b = np.arange(5)
np.dot(a,b)
```

Exercise 25: Create an array of 10 zeros and change variable to view and copy demonstration.

Solution

```
Z = np.zeros(10)
V = Z.view()
C = Z.copy()
V[0] = 1
print(Z[0], C[0])
```

Exercise 26: Create a 4x4 array with values 0..15 and reshape it to 2x8.

Solution

```
Z = np.arange(16).reshape(4,4)
Z.reshape(2,8)
```

Exercise 27: Stack arrays vertically and horizontally.

Solution

```
a = np.arange(6).reshape(2,3)
b = np.arange(6,12).reshape(2,3)
np.vstack([a,b])
np.hstack([a,b])
```

Exercise 28: Create an array with shape (3,4) filled with a given value.

Solution

```
np.full((3,4), 7)
```

Exercise 29: Find indices where elements are equal to a value.

Solution

```
np.where(np.arange(10)%2==0)
```

Exercise 30: Replace non-finite numbers (nan, inf) with zero.

Solution

```
Z = np.array([1, np.nan, np.inf, -np.inf, 5])
Z[~np.isfinite(Z)] = 0
```

Exercise 31: Compute the moving average of a 1D array.

Solution

```
def moving_average(a, n=3):
    return np.convolve(a, np.ones(n)/n, mode='valid')
```

Exercise 32: Create a 2D Gaussian filter kernel.

Solution

```
def gaussian_kernel(k=5, sigma=1.0):
    ax = np.arange(-k//2 + 1., k//2 + 1.)
    xx, yy = np.meshgrid(ax, ax)
    kernel = np.exp(-(xx**2 + yy**2)/(2*sigma**2))
    return kernel / np.sum(kernel)
```

Exercise 33: Count unique values and their frequencies.

Solution

```
values, counts = np.unique([1,2,2,3,3,3], return_counts=True)
```

Exercise 34: Compute pairwise distances between points.

Solution

```
from math import sqrt
X = np.random.random((5,2))
D = np.sqrt(((X[:,None,:] - X[None,:,:])**2).sum(-1))
```

Exercise 35: Create a toeplitz matrix from a vector.

Solution

```
from scipy.linalg import toeplitz
v = np.arange(5)
toeplitz(v)
```

Exercise 36: Compute the rank of a matrix.

Solution

```
np.linalg.matrix_rank(np.random.randn(4,4))
```

Exercise 37: Generate a matrix with random integers and compute row-wise cumulative sums.

Solution

```
Z = np.random.randint(0,10,(4,5))
np.cumsum(Z, axis=1)
```

Exercise 38: Convert polar to cartesian coordinates.

Solution

```
r = np.array([1,2,3])
theta = np.array([0, np.pi/4, np.pi/2])
x = r * np.cos(theta)
y = r * np.sin(theta)
```

Exercise 39: Compute eigenvalues and eigenvectors of a symmetric matrix.

Solution

```
M = np.random.randn(4,4)
S = (M + M.T)/2
w, v = np.linalg.eigh(S)
```

Exercise 40: Solve a system of linear equations Ax = b.

Solution

```
A = np.array([[3,1],[1,2]])
b = np.array([9,8])
x = np.linalg.solve(A,b)
```

Exercise 41: Create an array of dates and compute weekday for each.

Solution

```
dates = np.array(['2025-01-01','2025-01-02'], dtype='datetime64'
    )
dates.astype('datetime64[D]').view('int64') % 7 # weekday-ish
```

Exercise 42: Use fancy indexing to select and modify specific elements.

Solution

```
Z = np.arange(16).reshape(4,4)
rows = [0,1]
cols = [1,3]
Z[rows, cols] = 99
```

Exercise 43: Broadcast a 1D array to a 2D array and add.

Solution

```
a = np.arange(3)
b = np.ones((3,3))
b + a # broadcasts a across rows
```

Exercise 44: Compute correlation coefficient between two variables.

Solution

```
x = np.random.randn(100)
y = 2*x + np.random.randn(100)*0.1
np.corrcoef(x,y)[0,1]
```

Exercise 45: Shuffle rows of a matrix.

Solution

```
Z = np.arange(12).reshape(4,3)
np.random.shuffle(Z)
```

Exercise 46: Compute histogram of an array.

Solution

```
hist, bin_edges = np.histogram(np.random.randn(1000), bins=30)
```

Exercise 47: Construct a Vandermonde matrix from a vector.

Solution

```
x = np.arange(5)
np.vander(x, N=5)
```

Exercise 48: Perform element-wise complex conjugation.

Solution

```
Z = np.array([1+2j, 3-1j])
Z.conj()
```

Exercise 49: Compute cumulative product along axis.

Solution

```
np.cumprod([1,2,3,4])
```

Exercise 50: Apply a function along axis using apply$_a long_a xis$.
Exercise 50: Find the indices of the n largest values in an array.

Solution

```
def nlargest_indices(a, n=3):
    return np.argsort(a)[-n:][::-1]
```

Exercise 51: Use boolean masks to set values conditionally.

Solution

```
Z = np.arange(10)
Z[Z%3==0] = -1
```

Exercise 52: Compute matrix inverse and validate with dot product.

Solution

```
A = np.random.randn(4,4)
invA = np.linalg.inv(A)
np.allclose(A.dot(invA), np.eye(4))
```

Exercise 53: Reshape a flat index to coordinates.

Solution

```
idx = 13
np.unravel_index(idx, (4,4))
```

Exercise 54: Compute element-wise exponentials and logs.

Solution

```
np.exp([0,1,2])
np.log([1, np.e, np.e**2])
```

Exercise 55: Use masked arrays to ignore invalid values in computations.

Solution

```
import numpy.ma as ma
Z = np.array([1, np.nan, 3])
m = ma.masked_invalid(Z)
m.mean()
```

Exercise 56: Compute softmax of a vector in a stable way.

Solution

```
def softmax(x):
    e = np.exp(x - np.max(x))
    return e / e.sum()
```

Exercise 57: Compute outer product of two vectors.

Solution

```
np.outer([1,2,3],[4,5])
```

Exercise 58: Efficient histogram equalization on an image array.

Solution

```
def hist_eq(im):
    hist, bins = np.histogram(im.flatten(), 256, [0,256])
    cdf = hist.cumsum()
    cdf_normalized = cdf * hist.max()/ cdf.max()
    cdf_m = np.ma.masked_equal(cdf,0)
    cdf_m = (cdf_m - cdf_m.min())*255/(cdf_m.max()-cdf_m.min())
    cdf = np.ma.filled(cdf_m,0).astype('uint8')
    return cdf[im]
```

Exercise 59: Compute the Moore-Penrose pseudoinverse of a matrix.

Solution

```
np.linalg.pinv(np.random.randn(3,5))
```

Exercise 60: Generate all combinations (cartesian product) of two vectors.

Solution

```
a = np.array([1,2,3])
b = np.array([4,5])
np.array(np.meshgrid(a,b)).T.reshape(-1,2)
```

Exercise 61: Implement sparse-like operations using masked arrays (toy example).

Solution

```
from scipy import sparse
A = sparse.csr_matrix(np.random.randint(0,2,(5,5)))
A.dot(np.arange(5))
```

Exercise 62: Perform linear interpolation between points.

Solution

```
x = np.linspace(0,10,11)
y = np.sin(x)
xi = np.linspace(0,10,50)
yi = np.interp(xi, x, y)
```

Exercise 63: Compute the cumulative distribution function (CDF) from a PDF array.

Solution

```
pdf = np.abs(np.random.randn(100))
pdf = pdf/pdf.sum()
cdf = pdf.cumsum()
```

Exercise 64: Use stride tricks to create sliding windows (view-only).

Solution

```
from numpy.lib.stride_tricks import sliding_window_view
a = np.arange(10)
sliding_window_view(a, 3)
```

Exercise 65: Compute spectral decomposition of a symmetric matrix and reconstruct it.

Solution

```
S = np.random.randn(5,5)
S = (S + S.T)/2
w, v = np.linalg.eigh(S)
S_rec = (v * w) @ v.T
np.allclose(S, S_rec)
```

Exercise 66: Perform principal component analysis (PCA) for dimensionality reduction (2 PCs).

Solution

```
X = np.random.randn(100,5)
Xc = X - X.mean(axis=0)
U, S, Vt = np.linalg.svd(Xc, full_matrices=False)
X2 = U[:,:2] * S[:2]
```

Exercise 67: Generate random orthonormal matrix via QR decomposition.

Solution

```
Q, R = np.linalg.qr(np.random.randn(5,5))
```

Exercise 68: Compute the logistic regression sigmoid and gradient for a toy dataset.

Solution

```
def sigmoid(z):
    return 1/(1+np.exp(-z))
X = np.random.randn(100,3)
y = (np.random.rand(100) > 0.5).astype(int)
w = np.zeros(3)
preds = sigmoid(X.dot(w))
grad = X.T.dot(preds - y)/len(y)
```

Exercise 69: Use boolean indexing to extract rows matching multiple conditions.

Solution

```
Z = np.random.randint(0,10,(10,3))
Z[(Z[:,0] > 3) & (Z[:,2] < 7)]
```

Exercise 70: Compute the determinant and log-determinant for stability.

Solution

```
A = np.random.randn(4,4)
det = np.linalg.det(A)
sign, logdet = np.linalg.slogdet(A)
```

Exercise 71: Vectorize a Python function using np.vectorize (note: not performance-boosting).

Solution

```
def myfunc(x):
    return x**2 + 1
vfunc = np.vectorize(myfunc)
vfunc(np.arange(5))
```

Exercise 72: Compute row-wise means ignoring NaNs.

Solution

```
Z = np.array([[1,np.nan,3],[4,5,6]])
np.nanmean(Z, axis=1)
```

Exercise 73: Efficiently count occurrences of small integers using bincount.

Solution

```
np.bincount([0,1,1,3,2,1])
```

Exercise 74: Use einsum to compute matrix multiplication and trace compactly.

Solution

```
A = np.random.randn(3,3)
B = np.random.randn(3,3)
C = np.einsum('ij,jk->ik', A, B)
tr = np.einsum('ii->', A)
```

Exercise 75: Create random samples from a multinomial distribution.

Solution

```
np.random.multinomial(10, [0.2,0.3,0.5])
```

Exercise 76: Compute the Kronecker product of two matrices.

Solution

```
np.kron(np.eye(2), np.array([[1,2],[3,4]]))
```

Exercise 77: Implement one-hot encoding for integer class labels.

Solution

```
labels = np.array([0,2,1,3])
onehot = np.eye(labels.max()+1)[labels]
```

Exercise 78: Save and load arrays using np.save and np.load.

Solution

```
a = np.arange(10)
np.save('arr.npy', a)
b = np.load('arr.npy')
```

Exercise 79: Use packbits and unpackbits for bit-level operations.

Solution

```
bits = np.unpackbits(np.array([3,5], dtype=np.uint8))
```

Exercise 80: Implement matrix block extraction using slicing.

Solution

```
M = np.arange(36).reshape(6,6)
block = M[1:4, 2:5]
```

Exercise 81: Compute pairwise outer differences and apply threshold.

Solution

```
a = np.array([1,3,6])
D = np.abs(a[:,None] - a[None,:])
D > 2
```

Exercise 82: Use numpy's random.Generator for new random API.

Solution

```
rng = np.random.default_rng(42)
rng.normal(size=(3,3))
```

Exercise 83: Create a boolean mask for top-k per row.

Solution

```
Z = np.random.rand(5,10)
k = 3
idx = np.argpartition(Z, -k, axis=1)[:, -k:]
mask = np.zeros_like(Z, dtype=bool)
rows = np.arange(Z.shape[0])[:,None]
mask[rows, idx] = True
```

Exercise 84: Compute Spearman correlation by ranking then Pearson.

Solution

```
from scipy.stats import rankdata
x = np.random.randn(100)
y = np.random.randn(100)
rx = rankdata(x)
ry = rankdata(y)
np.corrcoef(rx, ry)[0,1]
```

Exercise 85: Efficiently compute sliding dot-products using FFT (convolution).

Solution

```
from numpy.fft import fft, ifft
def sliding_dot(a, b):
    n = len(a) + len(b) - 1
    return ifft(fft(a, n) * fft(b[::-1], n)).real
```

Exercise 86: Compute element-wise sign and replace zeros with 1.

Solution

```
x = np.array([-2,0,3])
s = np.sign(x)
s[s==0] = 1
```

Exercise 87: Find mode of an array (most frequent value).

Solution

```
vals, counts = np.unique([1,2,2,3,3,3], return_counts=True)
vals[counts.argmax()]
```

Exercise 88: Use np.linspace to create an angle grid and compute sine-cosine map.

Solution

```
theta = np.linspace(0, 2*np.pi, 360)
np.column_stack([np.sin(theta), np.cos(theta)])
```

Exercise 89: Compute least squares solution via np.linalg.lstsq.

Solution

```
A = np.random.randn(100,3)
b = A.dot(np.array([1.5,-2.0,0.5])) + np.random.randn(100)*0.1
x, *_ = np.linalg.lstsq(A, b, rcond=None)
```

Exercise 90: Create an identity matrix and perturb to make it symmetric positive definite.

Solution

```
A = np.eye(5) + np.random.randn(5,5)*0.1
SPD = A.T.dot(A)
```

Exercise 91: Compute convolution between two 1D signals using np.convolve.

Solution

```
np.convolve([1,2,3], [0,1,0.5])
```

Exercise 92: Use masked arrays to compute means along axis while ignoring masked values.

Solution

```
import numpy.ma as ma
a = ma.array([[1,2],[ma.masked,4]])
a.mean(axis=1)
```

Exercise 93: Compute inverse CDF sampling to draw from arbitrary discrete distribution.

Solution

```
probs = np.array([0.1,0.2,0.7])
cdf = probs.cumsum()
r = np.random.rand(1000)
samples = np.searchsorted(cdf, r)
```

Exercise 94: Implement simple k-means initialization (kmeans++ style).

Solution

```
def kmeans_pp_init(X, k):
    n = X.shape[0]
    centers = [X[np.random.randint(n)]]
    for _ in range(1,k):
        D = np.min(np.square(X[:,None]-np.array(centers)[None
            ,:,:]).sum(-1), axis=1)
        probs = D / D.sum()
        centers.append(X[np.searchsorted(probs.cumsum(), np.
            random.rand())])
    return np.array(centers)
```

Exercise 95: Compute blockwise mean of an image (downsampling by averaging blocks).

Solution

```
def block_mean(im, block=(2,2)):
    sh = (im.shape[0]//block[0], block[0], im.shape[1]//block
        [1], block[1])
    return im.reshape(sh).mean(axis=(1,3))
```

Exercise 96: Wrap-up: Short review exercise to combine multiple techniques.

Solution

```
# Example: normalize rows, compute PCA, project and reconstruct
X = np.random.randn(100,10)
X -= X.mean(axis=1, keepdims=True)
U, S, Vt = np.linalg.svd(X, full_matrices=False)
Xproj = (U[:,:3] * S[:3])
Xrec = Xproj.dot(Vt[:3,:])
```

2 Acknowledgements

Converted and improved by **Saurab Gyawali**. For more projects and contact:

- GitHub: https://github.com/soorabcode/
- Instagram: https://www.instagram.com/saurab_gyawalii
- LinkedIn: https://www.linkedin.com/in/saurab-gyawali-420897233