In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
```

Bhavani Prasad V's Notes

NumPy

> **NumPy**, short for "Numerical Python," is a fundamental Python library for numerical and **array**-based operations. It provides support for large, multi-dimensional arrays and matrices, as well as a vast collection of mathematical functions to operate on these arrays. NumPy is widely used in scientific, mathematical, and data-related computing tasks and serves as the foundation for many other libraries in the data science and machine learning ecosystems.

In [2]:
```python
# Create a NumPy array
n = np.array([1, 2])
n
```
Out[2]:
```
array([1, 2])
```

In [3]:
```python
n[n < 3]
```
Out[3]:
```
array([1, 2])
```

In [4]:
```python
# Check the data type of the NumPy array 'n'
n.dtype
```
Out[4]:
```
dtype('int64')
```

In [5]:
```python
# Create a 2D NumPy array
n2 = np.array([[2, 3], [5, 6]])
n3 = np.array([[2, 3], [5, 6]])
n2, n3
```
Out[5]:
```
(array([[2, 3],
        [5, 6]]),
 array([[2, 3],
        [5, 6]]))
```

In [6]:
```python
print((n2 <= n3).sum())
n2 == n3
```
```
4
```
Out[6]:
```
array([[ True,  True],
       [ True,  True]])
```

In [7]:
```python
# Create a 3D NumPy array 'nnn' with two 2x2x3 sub-arrays
nnn = np.array([
    [[11, 12, 13],
     [13, 14, 15]],

    [[15, 16, 17],
     [17, 18, 19]]
])
print(nnn)
nnn.shape
```
```
[[[11 12 13]
  [13 14 15]]

 [[15 16 17]
  [17 18 19]]]
```
Out[7]:
```
(2, 2, 3)
```

In [8]:
```python
# Slicing
nnn[1, ...]
```
Out[8]:
```
array([[15, 16, 17],
       [17, 18, 19]])
```

In [9]:
```python
# Check the number of dimensions in the NumPy array 'nnn'
nnn.ndim
```
Out[9]:
```
3
```

In [10]:
```python
# Compute the dot product of two NumPy arrays
dot_product = np.dot(n2, n3)
dot_product
```
Out[10]:
```
array([[19, 24],
       [40, 51]])
```

In [11]:
```python
# Calculate the matrix multiplication of 'n2' and 'n'
np.matmul(n2, n)
```
Out[11]:
```
array([ 8, 17])
```

In [12]:
```python
import timeit

# Create two NumPy arrays for comparison
np_arr1 = np.arange(1000, 2000)
np_arr2 = np.arange(1000, 2000)

# Define the pure Python function for dot product
def pure_python_dot_product():
    arr1 = list(range(1000, 2000))
    arr2 = list(range(1000, 2000))
    return sum(x * y for x, y in zip(arr1, arr2))

# Measure the execution time of the NumPy approach
numpy_time = timeit.timeit(lambda: np.dot(np_arr1, np_arr2), number=100000)

# Measure the execution time of the pure Python approach
python_time = timeit.timeit(pure_python_dot_product, number=100000)

# Compare execution times
print(f"NumPy execution time: {numpy_time:.6f} seconds")
print(f"Pure Python execution time: {python_time:.6f} seconds")
```
```
NumPy execution time: 0.462690 seconds
Pure Python execution time: 23.875412 seconds
```

In [13]:
```python
# Several operations with numpy
np.sum(n2)
np.prod(n2)
np.mean(n2)
np.min(n2)
np.max(n2)
np.median(n2)
np.std(n2)
np.var(n2)
np.argmax(n2)
np.argmin(n2)
```
Out[13]:
```
0
```

In [14]:
```python
n2
print(n2.T)
```
```
[[2 5]
 [3 6]]
```

In [15]:
```python
# Calculate the sum of elements along axis 0 (columns)
print(np.sum(n2, axis = 0))

# Calculate the sum of elements along axis 1 (rows)
print(np.sum(n2, axis = 1))
```
```
[7 9]
[ 5 11]
```

In [16]:
```python
# Create a NumPy array 'n4' containing values from 0 to 15
n4 = np.arange(0, 16)

# Reshape 'n4' into a 4x4 array, 'n4_reshaped'
n4_reshaped = n4.reshape(4, 4)

# Transpose 'n4_reshaped' to create 'n4_transposed'
n4_transposed = np.transpose(n4_reshaped)

r1, r2 = np.vsplit(n4_reshaped, 2)
r3, r4 = np.hsplit(n4_reshaped, 2)
print(r2)
r4
```
```
[[ 8  9 10 11]
 [12 13 14 15]]
```
Out[16]:
```
array([[ 2,  3],
       [ 6,  7],
       [10, 11],
       [14, 15]])
```

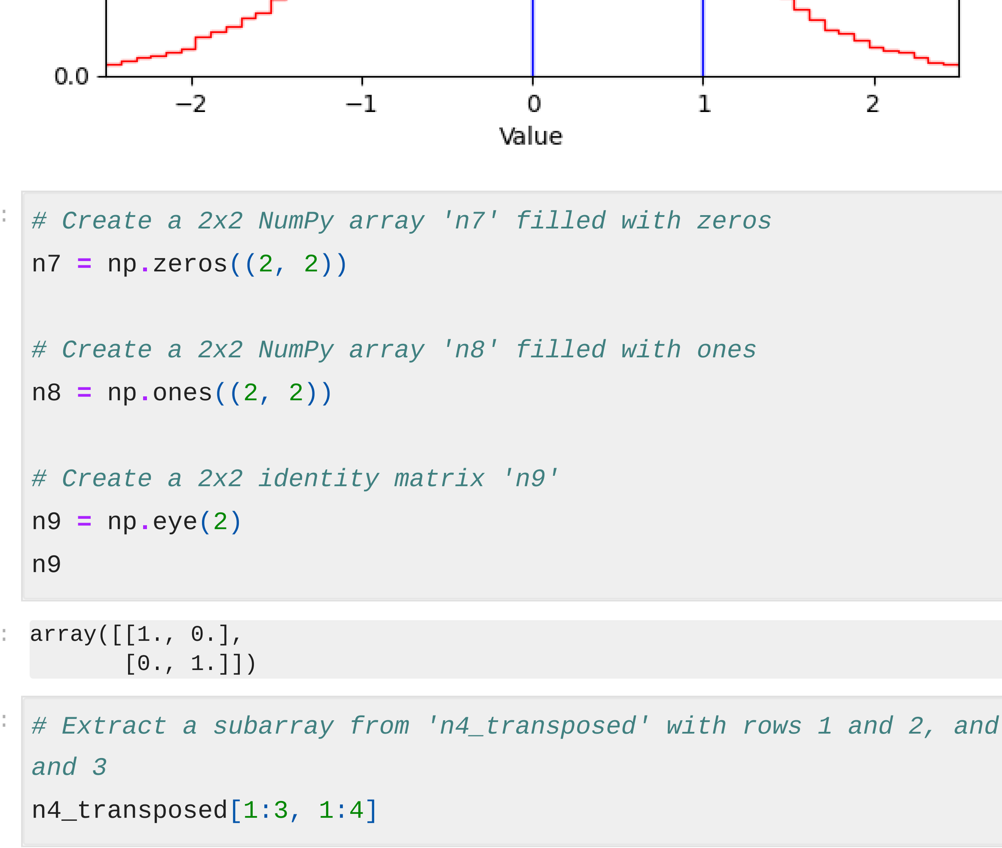
In [17]:
```python
  # Create a 2x3 NumPy array 'n5' with random integers ranging from 0 to 99
n5 = np.random.randint(0, 100, (2, 3))
# np.full([2, 3], 42)
n5
```
Out[17]:
```
array([[54,  2, 27],
       [ 1, 69, 61]])
```

In [18]:
```python
# Create an array with 5 equally spaced values between 3 and 27
np.linspace(3, 27, 5)
```
Out[18]:
```
array([ 3.,  9., 15., 21., 27.])
```

In [19]:
```python
# Create a 2x2 NumPy array 'n6' with random values from a uniform distribution (0 to 1)
n6 = np.random.rand(2, 2)
n6
```
Out[19]:
```
array([[0.2251781 , 0.17626304],
       [0.7226445 , 0.4356652 ]])
```

In [20]:
```python
# Create a 2x2 NumPy array 'R' with random integers ranging from 3 to 9
R = np.random.randint(3, 10, size = (2, 2))
# np.random.choice([1, 2, 3, 4, 5], size = 8)
R
```
Out[20]:
```
array([[7, 9],
       [5, 3]])
```

In [21]:
```python
plt.hist(np.random.rand(100000), density=True, bins=100, histtype="step", color="blue", label="rand")
plt.hist(np.random.randn(100000), density=True, bins=100, histtype="step", color="red", label="randn")
plt.axis([-2.5, 2.5, 0, 1.1])
plt.legend(loc = "upper left")
plt.title("Random distributions")
plt.xlabel("Value")
plt.ylabel("Density")
plt.show()
```



In [22]:
```python
# Create a 2x2 NumPy array 'n7' filled with zeros
n7 = np.zeros((2, 2))

# Create a 2x2 NumPy array 'n8' filled with ones
n8 = np.ones((2, 2))

# Create a 2x2 identity matrix 'n9'
n9 = np.eye(2)
n9
```
Out[22]:
```
array([[1., 0.],
       [0., 1.]])
```

In [23]:
```python
# Extract a subarray from 'n4_transposed' with rows 1 and 2, and columns 1, 2, and 3
n4_transposed[1:3, 1:4]
```
Out[23]:
```
array([[ 5,  9, 13],
       [ 6, 10, 14]])
```

In [24]:
```python
# Stack two NumPy arrays 'x' and 'y' horizontally
x = np.array([[1, 2, 3], [4, 5, 6]])
y = np.array([[1, 2, 3], [4, 5, 6]])
arr_hstacked = np.hstack((x, y))
arr_hstacked
```
Out[24]:
```
array([[1, 2, 3, 1, 2, 3],
       [4, 5, 6, 4, 5, 6]])
```

In [25]:
```python
# Stack two NumPy arrays 'x' and 'y' vertically
# con = np.concatenate((n2, n3), axis = 0)
arr_vstacked = np.vstack((x, y))
arr_vstacked
```
Out[25]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [1, 2, 3],
       [4, 5, 6]])
```

In [26]:
```python
# Sort the NumPy array 'b' in ascending order
b = np.array([1, 4, 2, 8, 5, 3, 4, 9, 1, 5, 3, 8, 5, 2])
np.sort(b)
#np.where(b ==2)
```
Out[26]:
```
array([1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 5, 8, 8, 9])
```

In [27]:
```python
# Calculate the determinant of the 2x2 NumPy array 'a' using NumPy's linalg.det() function
a = np.array([[1, 2], [3, 4]])
np.linalg.det(a)
```
Out[27]:
```
-2.0000000000000004
```

In [28]:
```python
# Calculate the inverse of the 2x2 NumPy array 'a' using NumPy's linalg.inv() function
np.linalg.inv(a)
```
Out[28]:
```
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

In [29]:
```python
# Create a NumPy array 'A' as a 3x3 matrix
A = np.array([[3, 4, 1], [2, 2, 2], [4, 1, 1]])

# Create a NumPy array 'x' with variables 'x', 'y', and 'z'
x = np.array([['x'], ['y'], ['z']])

# Create a NumPy array 'z' with values 7, 14, and 21
z = np.array([[7], [14], [21]])
```

In [30]:
```python
# Solve the equation A.X = Z for X
# Calculate the inverse of matrix A
invA = np.linalg.inv(A)

# Compute the solution X by multiplying the inverse of A with Z
X = np.dot(invA, z)
X
```
Out[30]:
```
array([[ 4.66666667],
       [-3.11111111],
       [ 5.44444444]])
```

In [32]:
```python
# Load data from the 'climate.txt' file using NumPy's genfromtxt
# The data is assumed to be comma-separated with a header row that is skipped
climate_data = np.genfromtxt('climate.txt', delimiter = ',', skip_header = 1)
climate_data
```
Out[32]:
```
array([[25., 76., 99.],
       [39., 65., 70.],
       [59., 45., 77.],
       ...,
       [99., 62., 58.],
       [70., 71., 91.],
       [92., 39., 76.]])
```