01 - Introduction To NumPY

Importing NumPY

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

Initialization of Array in NumPY

From Python Lists

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

```
[1 2 3]
```

Array with Evenly Spaced Numbers

Syntax -> numpy.linspace(start, stop, num = 50, endpoint = True, retstep = False, dtype = None, axis = 0)

In [3]:
```
arr = np.linspace(3.5, 10, 3, dtype = np.float64)
print(arr)
```

```
[ 3.5   6.75 10.  ]
```

Array with Garbage Value

In [4]:
```
arr = np.empty([4, 3], dtype = np.int32, order = 'f')
print(arr)
```

```
[[0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]]
```

Array with Zeros

In [5]:
```
arr = np.zeros([2, 3], dtype = np.int32, order = 'f') # [Dimension of Array]
print(arr)
```

```
[[0 0 0]
 [0 0 0]]
```

Array with Ones

In [6]:
```
arr = np.ones((2, 2), dtype = np.int32, order = 'f') # [Dimension of Array]
print(arr)
```

```
[[1 1]
 [1 1]]
```

Array with All Values Equal

In [7]:
```
arr = np.full((2, 2), 7)
print(arr)
```

```
[[7 7]
 [7 7]]
```

Array with Range Values

In [8]:
```
arr = np.arange(0, 10, 2, dtype = np.int64) # (start, end, steps), end is excl
print(arr)
```

```
[0 2 4 6 8]
```

Indexing in NumPY

In 1D Array

In [9]:
```
arr = np.array([10, 20, 30, 40, 50])
              #  0,  1,  2,  3,  4
              # -5, -4, -3, -2, -1
print(arr[2])
print(arr[-1])
```

```
30
50
```

In 2D Array

In [10]:
```
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr[1, 0]) # [Row, Column]
```

```
4
```

In [11]:
```
arr = np.array([[1, 2, 3], [4, 5, 6]])

# all rows, second column
print(arr[:, 1])
# [2 5]
print(arr[:, [1]])
# [[2]
#  [5]]

# all, rows, second and third column
print(arr[:, [1, 2]])
# [[2 3]
#  [5 6]]
```

```
[2 5]
[[2]
 [5]]
[[2 3]
 [5 6]]
```

[start:stop(excluded):steps]

In [12]:
```
arr = np.array([0, 1, 2, 3, 4, 5])
print(arr[1:6:2])
```

```
[1 3 5]
```

In [13]:
```
arr = np.random.rand(4, 4, 4)
print(arr[..., 0])
```

```
[[0.11768385 0.8764094  0.22217558 0.38894211]
 [0.10770664 0.04327252 0.16304568 0.75131819]
 [0.79017186 0.95736669 0.31046103 0.8610755 ]
 [0.87284114 0.69506407 0.35112079 0.49079746]]
```

In [14]:
```
arr = np.array([1, 2, 0, 4, 5, 3])

idx = np.array([1, 3, 5])
print (arr[idx])

print (arr[(arr != 0) | (arr == 0)])
```

```
[2 4 3]
[1 2 0 4 5 3]
```

In [15]:
```
arr = np.array([1, 2, 3])
print(arr[:, np.newaxis])
```

```
[[1]
 [2]
 [3]]
```

In [16]:
```
arr = np.array([1, 2, 3, 4])
arr[1:3] = 99
print(arr)
```

```
[ 1 99 99  4]
```

Arithmetic Operations in NumPY Arrays

In [17]:
```
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])

print(x + y)
print(x - y)
print(x * y)
print(x / y)
```

```
[5 7 9]
[-3 -3 -3]
[ 4 10 18]
[0.25 0.4  0.5 ]
```

Absolute in NumPY

In [18]:
```
arr = np.array([-3, -1, 0, 1, 3])
res = np.absolute(arr)

print(res)
```

```
[3 1 0 1 3]
```

Add in Numpy

In [19]:
```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
res = np.add(arr1, arr2)

print(res)
```

```
[5 7 9]
```

Exponentiation in NumPY

In [20]:
```
arr1 = np.array([5, 72, 13, 100])
arr2 = np.array([2, 5, 10, 30])

res = np.power(arr1, arr2)
print(res)
```

```
[                 25          1934917632        137858491849
 1152921504606846976]
```

Modulus in NumPY

In [21]:
```
arr1 = np.array([5, 72, 13, 100])
arr2 = np.array([2, 5, 10, 30])

res = np.mod(arr1, arr2)
print(res)
```

```
[ 1  2  3 10]
```

Sine / Exponential / Square Root in NumPy

In [22]:
```
a = np.array([0, np.pi/2, np.pi])
print(np.sin(a))
print(np.exp(a))
print(np.sqrt(a))
```

```
[0.0000000e+00 1.0000000e+00 1.2246468e-16]
[ 1.          4.81047738 23.14069263]
[0.         1.25331414 1.77245385]
```

Sorting NumPY Arrays

In [23]:
```
dtypes = [('name', 'S10'), ('grad_year', int), ('cgpa', float)]

values = [('Hrithik', 2009, 8.5), ('Ajay', 2008, 8.7),
          ('Pankaj', 2008, 7.9), ('Aakash', 2009, 9.0)]

arr = np.array(values, dtype = dtypes)

print (np.sort(arr, order = 'name'))
print (np.sort(arr, order = ['grad_year', 'cgpa']))
```

```
[(b'Aakash', 2009, 9. ) (b'Ajay', 2008, 8.7) (b'Hrithik', 2009, 8.5)
 (b'Pankaj', 2008, 7.9)]
[(b'Pankaj', 2008, 7.9) (b'Ajay', 2008, 8.7) (b'Hrithik', 2009, 8.5)
 (b'Aakash', 2009, 9. )]
```

Type Of NumPY Arrays

In [24]:
```
arr = np.array([1, 2, 3])
print(type(arr))
```

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

Shape / Size / DataType of NumPY Arrays

In [25]:
```
arr = np.array([[1, 2, 3], [4, 5, 6]])
print("Shape :", arr.shape)
print("Size :", arr.size)
print("D Type :", arr.dtype)
```

```
Shape : (2, 3)
Size : 6
D Type : int64
```

Formiter in NumPY

In [26]:
```
var = "RafatAlam"
arr = np.fromiter(var, dtype = 'U2')

print(arr)
```

```
['R' 'a' 'f' 'a' 't' 'A' 'l' 'a' 'm']
```

Random In NumPY

Fills values with random values between [0, 1)

In [27]:
```
arr = np.random.rand(2, 3)
print(arr)
```

```
[[0.36980815 0.56430882 0.31254843]
 [0.93595332 0.05877026 0.13532889]]
```

Fill values with standard Normal Distribution

In [28]:
```
arr = np.random.randn(2, 2)
print(arr)
```

```
[[-1.37944518 -0.63128077]
 [ 0.72492623 -0.18402523]]
```

Fills values with random integers between [a, b)

In [29]:
```
arr = np.random.randint(1, 10, size = [2, 3])
print(arr)
```

```
[[1 6 5]
 [7 1 9]]
```

Matrix creation in NumPY

Identity Matrix

In [30]:
```
arr = np.eye(3, dtype = np.int64)
print(arr)
```

```
[[1 0 0]
 [0 1 0]
 [0 0 1]]
```

Diagonal Matrix

In [31]:
```
arr = np.diag([1, 2, 3])
print(arr)
```

```
[[1 0 0]
 [0 2 0]
 [0 0 3]]
```

Zero Array

In [32]:
```
arr = np.zeros_like(arr) # Pass array it will take shape of that
print(arr)
```

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
```

One's Array

In [33]:
```
arr = np.ones_like(arr) # Pass array it will take shape of that
print(arr)
```

```
[[1 1 1]
 [1 1 1]
 [1 1 1]]
```

Reshaping Arrays

In [34]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16])
print(str(arr))

res = arr.reshape((2, 8))
print(res)
res = np.reshape(arr, [4, -1])
print(res)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16]
[[ 1  2  3  4  5  6  7  8]
 [ 9 10 11 12 13 14 15 16]]
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
```

Flatten vs Ravel

In [35]:
```
arr = np.array([[1, 2, 3, 4],
                [5, 6, 7, 8],
                [9, 10, 11, 12],
                [13, 14, 15, 16]])

res = arr.flatten() # Creates Copy
print(res)
print(arr.ravel())  # Not a Copy
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16]
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16]
```

Resizing Arrays

Permanent Array Reshaping

In [36]:
```
arr = np.array([1, 2, 3, 4, 5, 6])
arr.resize(2, 3)
print(arr)
```

```
[[1 2 3]
 [4 5 6]]
```

Type-casting in NumPY

In [37]:
```
a = np.array([1.1, 2.2, 3.3])
a = a.astype(int)

print(a)
```

```
[1 2 3]
```

Inserting Values in NumPY

In [38]:
```
arr = np.array([1, 2, 3, 4])
res = np.insert(arr, 3, 33, axis = None)
print(res)
```

```
[ 1  2  3 33  4]
```

In [39]:
```
arr = np.array([[1, 2, 3], [4, 5, 6]])

res1 = np.insert(arr, 2, 33, axis = 0)
print(res1)

res2 = np.insert(arr, 2, [33, 44], axis = 1)
print(res2)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [33 33 33]]
[[ 1  2 33  3]
 [ 4  5 44  6]]
```

Deleting Values in NumPY

In [40]:
```
arr = np.array([1, 2, 3, 4])
res = np.delete(arr, 2, axis = None)
print(res)
```

```
[1 2 4]
```

Identify missing values in Data Sets

In [41]:
```
arr = np.array([1, 2, np.nan, 4, np.nan])
print(np.isnan(arr))

res = np.nan_to_num(arr, nan = 0)
print(res)
```

```
[False False  True False  True]
[1. 2. 0. 4. 0.]
```

Identifying Infinite values in Data Set

In [42]:
```
arr = np.array([1, 2, np.inf, 4, -np.inf])
print(np.isinf(arr))

res = np.nan_to_num(arr, posinf = 0, neginf = 0)
print(res)
```

```
[False False  True False  True]
[1. 2. 0. 4. 0.]
```

---

02 - Stacking and Splitting Arrays

Importing NumPY

In [43]:
```
import numpy as np
```

Stacking 2 [1-D Arrays]

In [44]:
```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

res = np.stack((arr1, arr2), axis = 0)
print(res)
```

```
[[1 2 3]
 [4 5 6]]
```

In [45]:
```
res = np.stack((arr1, arr2), axis = 1)
print(res)
```

```
[[1 4]
 [2 5]
 [3 6]]
```

In [46]:
```
res = np.stack((arr1, arr2), axis = -1) # -1 represents 'last dimension-wise'.
print(res)
```

```
[[1 4]
 [2 5]
 [3 6]]
```

Splitting Arrays in NumPY

In [47]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
res = np.array_split(arr, 3)
print(res)
```

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

In [48]:
```
arr = np.array([[3, 2, 1], [8, 9, 7], [4, 6, 5]])
res = np.split(arr, 3, axis = 1)
print(res)
```

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

V-Split is for axis = 0

In [49]:
```
arr = np.array([[1, 2, 3, 4],
                [5, 6, 7, 8],
                [9, 10, 11, 12],
                [13, 14, 15, 16]])

res = np.vsplit(arr, 2)
print(res)
res = np.split(arr, 2, axis = 0)
print(res)
```

```
[array([[1, 2, 3, 4],
       [5, 6, 7, 8]]), array([[ 9, 10, 11, 12],
       [13, 14, 15, 16]])]
[array([[1, 2, 3, 4],
       [5, 6, 7, 8]]), array([[ 9, 10, 11, 12],
       [13, 14, 15, 16]])]
```

H-Split is for axis = 1

In [50]:
```
res = np.hsplit(arr, 2)
print(res)
res = np.split(arr, 2, axis = 1)
print(res)
```

```
[array([[ 1,  2],
       [ 5,  6],
       [ 9, 10],
       [13, 14]]), array([[ 3,  4],
       [ 7,  8],
       [11, 12],
       [15, 16]])]
[array([[ 1,  2],
       [ 5,  6],
       [ 9, 10],
       [13, 14]]), array([[ 3,  4],
       [ 7,  8],
       [11, 12],
       [15, 16]])]
```

D-Split is for axis = 2

In [51]:
```
arr = np.array([[[0, 1, 2, 3],
                 [4, 5, 6, 7],
                 [8, 9, 10, 11]],

                [[12, 13, 14, 15],
                 [16, 17, 18, 19],
                 [20, 21, 22, 23]]])

res = np.dsplit(arr, 2)
print(res)
```

```
[array([[[ 0,  1],
        [ 4,  5],
        [ 8,  9]],

       [[12, 13],
        [16, 17],
        [20, 21]]]), array([[[ 2,  3],
        [ 6,  7],
        [10, 11]],

       [[14, 15],
        [18, 19],
        [22, 23]]])]
```

---

03 - Array Broadcasting

Importing NumPY

In [52]:
```
import numpy as np
```

Operation always happens row-wise

In [53]:
```
a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([1, 2, 3])

print(a + b)
```

```
[[2 4 6]
 [5 7 9]]
```

By using np.newaxis, we can add elements column-wise

In [54]:
```
a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([1, 2])

print(a + b[:, np.newaxis])
```

```
[[2 3 4]
 [6 7 8]]
```

Broadcasting in Conditional Operations

In [55]:
```
ages = np.array([12, 24, 35, 45, 60, 72])
age_group = np.array(["Adult", "Minor"])

res = np.where(ages >= 18, age_group[0], age_group[1])
print(res)
```

```
['Minor' 'Adult' 'Adult' 'Adult' 'Adult' 'Adult']
```

Normalizing Data in ML

- centers data and it has zero mean.
- By dividing the standard deviation, its ensures unit variance.

In [56]:
```
arr = np.array([
  [100, 120, 130],
  [90, 110, 140],
  [80, 100, 120]
])

mean = arr.mean(axis = 0)
std = arr.std(axis = 0)

normalized_arr = (arr - mean) / std
print(normalized_arr)
```

```
[[ 1.22474487  1.22474487  0.        ]
 [ 0.          0.          1.22474487]
 [-1.22474487 -1.22474487 -1.22474487]]
```

Centering Data in ML

In [57]:
```
arr = np.array([
  [10, 20],
  [15, 25],
  [20, 30]
])

mean = arr.mean(axis = 0)
centered_arr = arr - mean
print(centered_arr)
```

```
[[-5. -5.]
 [ 0.  0.]
 [ 5.  5.]]
```

---

04 - Aggregation / Universal Functions

Importing NumPY

In [58]:
```
import numpy as np
```

Sum in NumPY

In [59]:
```
arr = [20, 2, .2, 10, 4]

print("Sum :", np.sum(arr))
```

```
print("Data Type :", np.sum(arr).dtype)

print("Sum (int32) :", np.sum(arr, dtype = np.int32))
```

```
Sum : 36.2
Data Type : float64
Sum (int32) : 36
```

In [60]:
```
arr = [[14, 17, 12, 33, 44],
             [15, 6, 27, 8, 19],
             [23, 2, 54, 1, 4,]]

print("Sum :", np.sum(arr))
print("Data Type :", np.sum(arr).dtype)
```

```
Sum : 279
Data Type : int64
```

In [61]:
```
arr = [[14, 17, 12, 33, 44],
             [15, 6, 27, 8, 19],
             [23, 2, 54, 1, 4,]]

print(np.sum(arr, axis = 0))
print(np.sum(arr, axis = 1))
print(np.sum(arr, axis = 1, keepdims = True))
```

```
[52 25 93 42 67]
[120  75  84]
[[120]
 [ 75]
 [ 84]]
```

Mean in NumPY

In [62]:
```
arr = [20, 2, 7, 1, 34]
print(np.mean(arr))
```

```
12.8
```

In [63]:
```
arr = [[14, 17, 12, 33, 44],
       [15, 6, 27, 8, 19],
       [23, 2, 54, 1, 4]]

print(np.mean(arr))
print(np.mean(arr, axis = 0))
print(np.mean(arr, axis = 1))
```

```
18.6
[17.33333333  8.33333333 31.         14.         22.33333333]
[24.  15.  16.8]
```

Max / Min in NumPY

In [64]:
```
num1 = 11
num2 = 21

res = np.maximum(num1, num2)
print(res)
res = np.minimum(num1, num2)
print(res)
```

```
21
11
```

In [65]:
```
num1 = [2, 8, 125]
num2 = [3, 3, 15]

res = np.maximum(num1, num2)
print(res)
res = np.minimum(num1, num2)
print(res)
```

```
[  3   8 125]
[ 2  3 15]
```

In [66]:
```
num1 = [np.nan, 0, np.nan]
num2 = [np.nan, np.nan, 0]

res = np.maximum(num1, num2)
print(res)
res = np.minimum(num1, num2)
print(res)
```

```
[nan nan nan]
[nan nan nan]
```

Trignometric Functions

In [67]:
```
angles = np.array([0, 30, 45, 60, 90, 180])

radians = np.deg2rad(angles)

# sine of angles
```

```
sine = np.sin(radians)
print(np.sin(radians))

# inverse sine of sine values
print(np.rad2deg(np.arcsin(sine)))

# hyperbolic sine of angles
sineh = np.sinh(radians)
print(np.sinh(radians))

# inverse sine hyperbolic
print(np.sin(sineh))

# hypotenuse
print(np.hypot(3, 4))
```

```
[0.00000000e+00 5.00000000e-01 7.07106781e-01 8.66025404e-01
 1.00000000e+00 1.22464680e-16]
[0.0000000e+00 3.0000000e+01 4.5000000e+01 6.0000000e+01 9.0000000e+01
 7.0167093e-15]
[ 0.          0.54785347  0.86867096  1.24936705  2.3012989  11.54873936]
[ 0.          0.52085606  0.76347126  0.94878485  0.74483916 -0.85086591]
5.0
```

Statical Functions

In [68]:

```
arr = np.array([50.7, 52.5, 50, 58, 55.63, 73.25, 49.5, 45])

# minimum and maximum
print(np.amin(arr), np.amax(arr))

# range of arr i.e. max - min
print(np.ptp(arr))

# percentile -> Value below which 70 % student fall
print(np.percentile(arr, 70))

# mean
print(np.mean(arr))

# median
print(np.median(arr))

# standard deviation
print(np.std(arr))

# variance
print(np.var(arr))

# average
print(np.average(arr))
```

```
45.0 73.25
28.25
55.317
54.3225
51.6
8.052773978574091
64.84716875
54.3225
```

Bit-twiddling Functions

In [69]:
```
even = np.array([0, 2, 4, 6, 8, 16, 32])
odd = np.array([1, 3, 5, 7, 9, 17, 33])

# bitwise_and
print(np.bitwise_and(even, odd))

# bitwise_or
print(np.bitwise_or(even, odd))

# bitwise_xor
print(np.bitwise_xor(even, odd))

# invert or not
print(np.invert(even))

# left_shift
print(np.left_shift(even, 1))

# right_shift
print(np.right_shift(even, 1))
```

```
[ 0  2  4  6  8 16 32]
[ 1  3  5  7  9 17 33]
[1 1 1 1 1 1 1]
[ -1  -3  -5  -7  -9 -17 -33]
[ 0  4  8 12 16 32 64]
[ 0  1  2  3  4  8 16]
```

---

05 - Linear Algebra with NumPY

Importing NumPY

In [70]:
```
import numpy as np
```

Transpose of Matrix

In [71]:
```
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr.T)
```

```
[[1 4]
 [2 5]
 [3 6]]
```

Matrix Multiplication / Dot Product

In [72]:
```
arr1 = np.array([[1, 2], [4, 5]])
arr2 = np.array([[7, 8], [9, 10]])

print(np.dot(arr1, arr2))
print(np.vdot(arr1, arr2)) # It assumes array to be flattened

vector_a = 2 + 3j
vector_b = 4 + 5j

print(np.dot(vector_a, vector_b))
print(np.vdot(vector_a, vector_b)) # It will take conjugate of complex numbers
```

```
[[25 28]
 [73 82]]
109
(-7+22j)
(23-2j)
```

Inner Product

In [73]:
```
arr1 = np.array([2, 6])     # 1 x 2
arr2 = np.array([3, 10])    # 2 x 1

print(np.inner(arr1, arr2)) # 1 x 1

arr1 = np.array([[2, 3, 4], [3, 2, 9]])  # 2 x 3
arr2 = np.array([[1, 5, 0], [5, 10, 3]]) # 3 x 2

print(np.inner(arr1, arr2))              # 2 x 2
```

```
66
[[17 52]
 [13 62]]
```

Outer Product

In [74]:
```
arr1 = np.array([2, 6])
arr2 = np.array([3, 10])

print(np.outer(arr1, arr2))

arr1 = np.array([[3, 6, 4], [9, 4, 6]])
arr2 = np.array([[1, 15, 7], [3, 10, 8]])

print(np.outer(arr1, arr2))
```

```
[[ 6 20]
 [18 60]]
[[  3  45  21   9  30  24]
 [  6  90  42  18  60  48]
 [  4  60  28  12  40  32]
 [  9 135  63  27  90  72]
 [  4  60  28  12  40  32]
 [  6  90  42  18  60  48]]
```

Cross Product

In [75]:
```
arr1 = np.array([3, 6, 0])
arr2 = np.array([9, 10, 0])

print(np.cross(arr1, arr2))

arr1 = np.array([[2, 6, 9], [2, 7, 3]])
arr2 = np.array([[7, 5, 6], [3, 12, 3]])

print(np.cross(arr1, arr2))
```

```
[  0   0 -24]
[[ -9  51 -32]
 [-15   3   3]]
```

Dterminant of a Matrix

Using log of determinant

In [76]:
```
arr = np.array([[50, 29], [30, 44]])
sign, logdet = np.linalg.slogdet(arr)
res = sign * np.exp(logdet)
print(res)
```

```
1330.0000000000002
```

Using simple Determinant (Used for small values)

In [77]:
```
arr = np.array([[1, 2], [3, 4]])
res = np.linalg.det(arr)
print(res)
```

```
-2.0000000000000004
```

Using LU Decomposition : LU decomposition can also be used to calculate the determinant by decomposing the matrix into lower (L) and upper (U) triangular matrices. The determinant is the product of the diagonal elements of the U matrix.

In [78]:
```
import scipy.linalg
arr = np.array([[1, 2], [3, 4]])
P, L, U = scipy.linalg.lu(arr)
res = np.prod(np.diag(U))
print(res)
```

```
2.0
```

Inverse of a Matrix

In [79]:
```
arr = np.array([[6, 1, 1],
                [4, -2, 5],
                [2, 8, 7]])
print(np.linalg.inv(arr))
```

```
[[ 0.17647059 -0.00326797 -0.02287582]
 [ 0.05882353 -0.13071895  0.08496732]
 [-0.11764706  0.1503268   0.05228758]]
```

---

06 - Statistics in ML

Importing Modules

In [80]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from scipy.stats import binom
import seaborn as sns
```

Random Sampling in NumPY

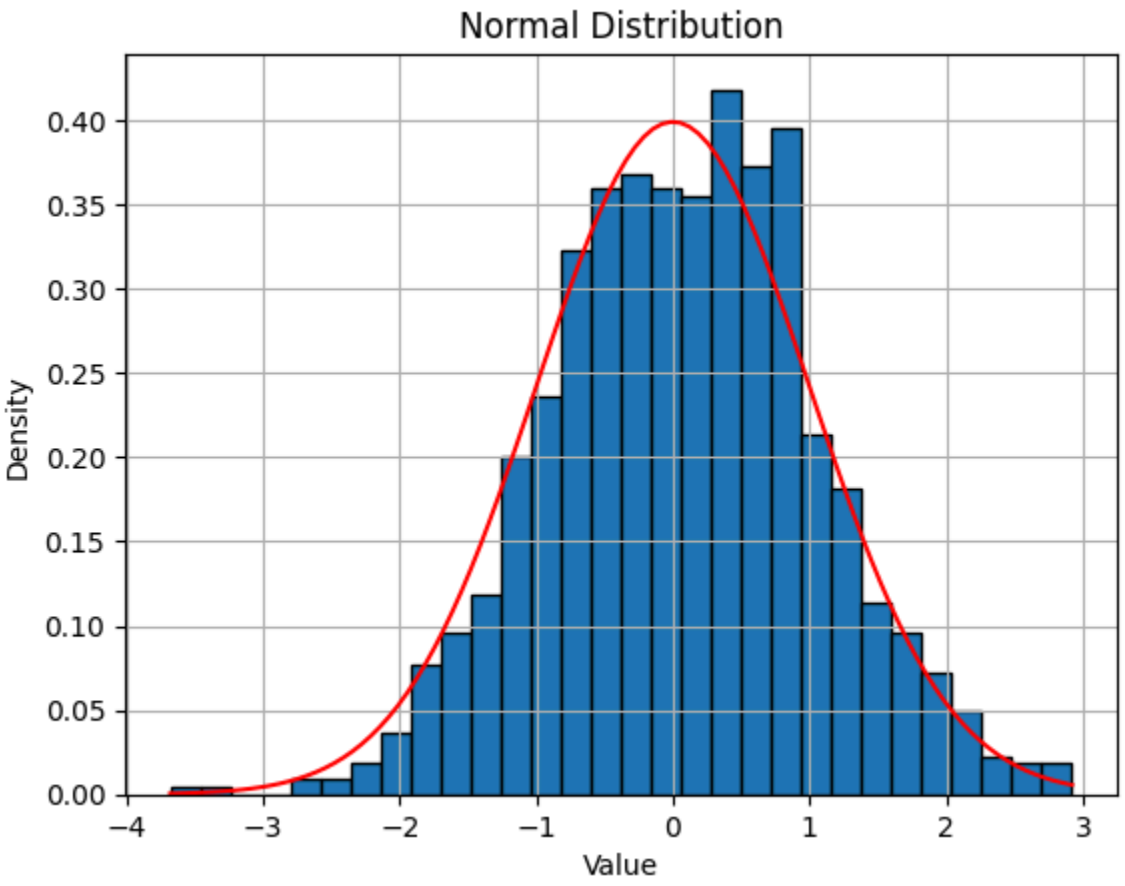
In [81]:
```
res = np.random.randint(low = 0, high = 3, size = [2, 3])
print(res)
```

```
[[0 2 2]
 [0 0 2]]
```

Normal Distribution

In [82]:
```
data = np.random.normal(loc = 0, scale = 1, size = 1000)

x = np.linspace(min(data), max(data), 100)
plt.hist(data, bins = 30, edgecolor = 'black', density = True)
pdf = norm.pdf(x, loc = 0, scale = 1)
plt.plot(x, pdf, color = 'red')
plt.title("Normal Distribution")
plt.xlabel("Value")
plt.ylabel("Density")
plt.grid(True)
plt.show()
```

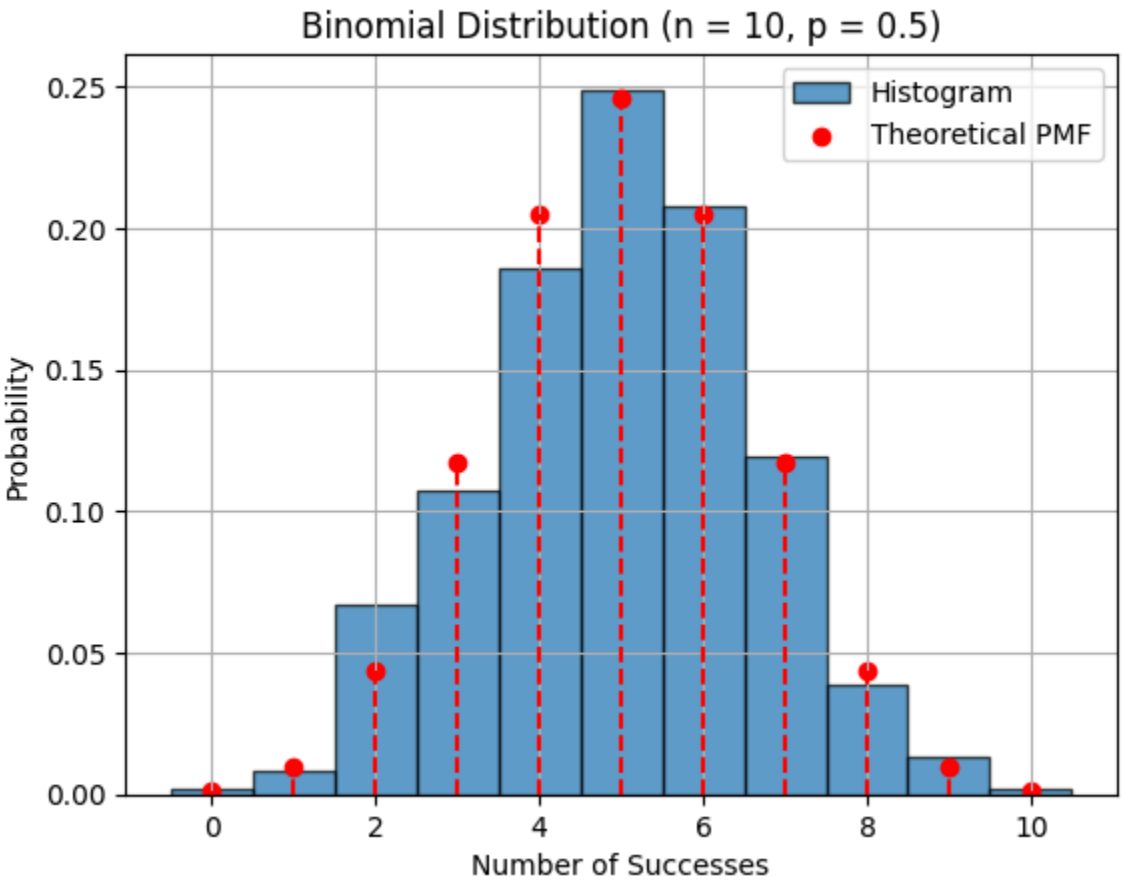


Binomial Distribution

In [83]:
```
n = 10
p = 0.5
size = 1000

data = np.random.binomial(n = n, p = p, size = size)

plt.hist(data, bins = np.arange(-0.5, n + 1.5, 1), density = True, edgecolor =
x = np.arange(0, n + 1)
pmf = binom.pmf(x, n = n, p = p)
plt.scatter(x, pmf, color = 'red', label = 'Theoretical PMF')
plt.vlines(x, 0, pmf, colors = 'red', linestyles = 'dashed')
plt.title("Binomial Distribution (n = 10, p = 0.5)")
plt.xlabel("Number of Successes")
plt.ylabel("Probability")
plt.legend()
plt.grid(True)
plt.show()
```

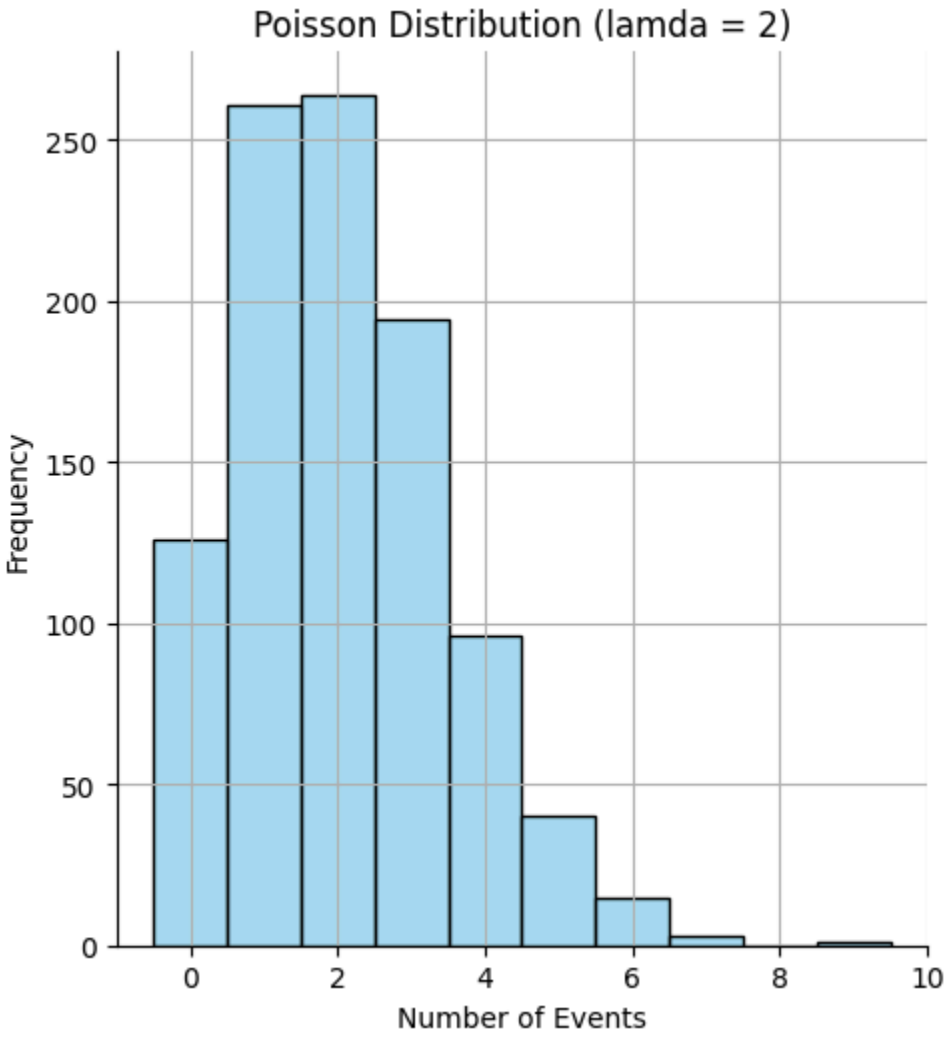


Poission Distribution

In [84]:
```
lam = 2
size = 1000

data = np.random.poisson(lam = lam, size = size)

sns.displot(data, kde = False, bins = np.arange(-0.5, max(data) + 1.5, 1), col
plt.title(f"Poisson Distribution (lamda = {lam})")
plt.xlabel("Number of Events")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()
```



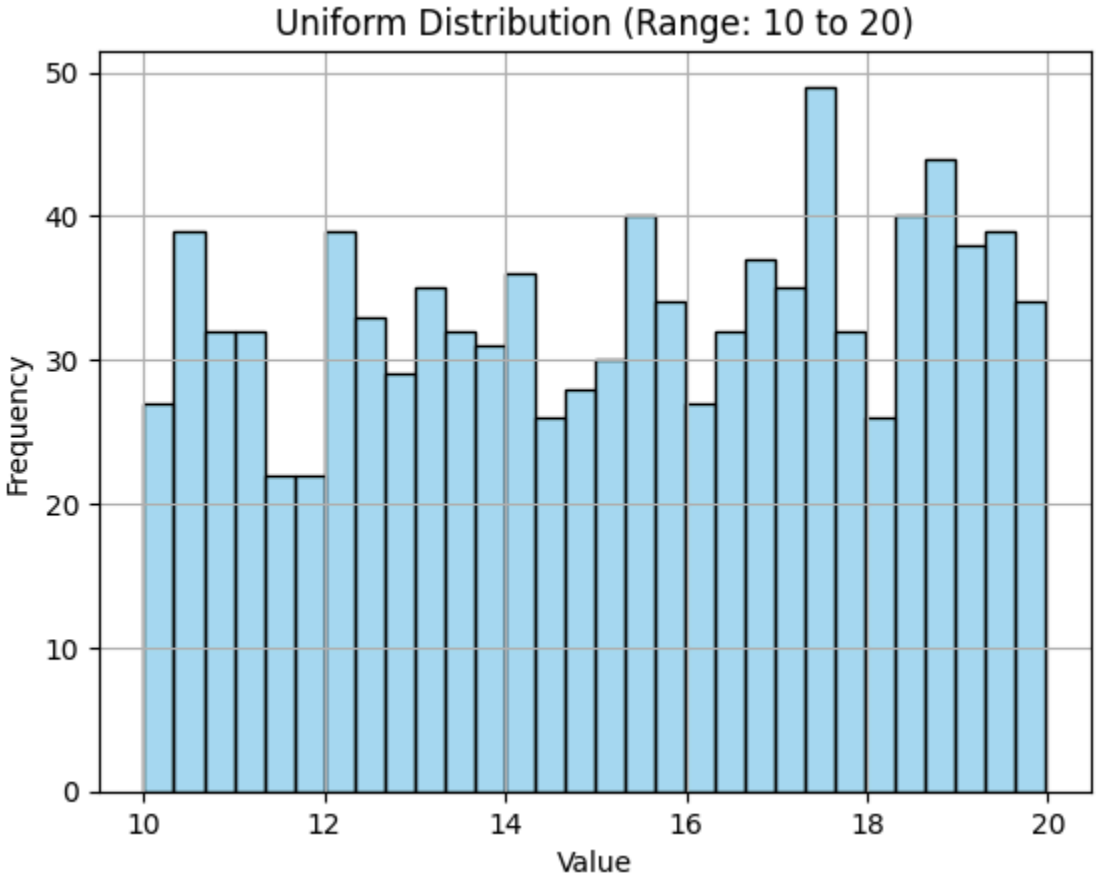
Uniform Distribution

In [85]:
```
low = 10
```

```
high = 20
size = 1000

data = np.random.uniform(low = low, high = high, size = size)

sns.histplot(data, bins = 30, kde = False, color = 'skyblue', edgecolor = 'bla
plt.title(f"Uniform Distribution (Range: {low} to {high})")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()
```



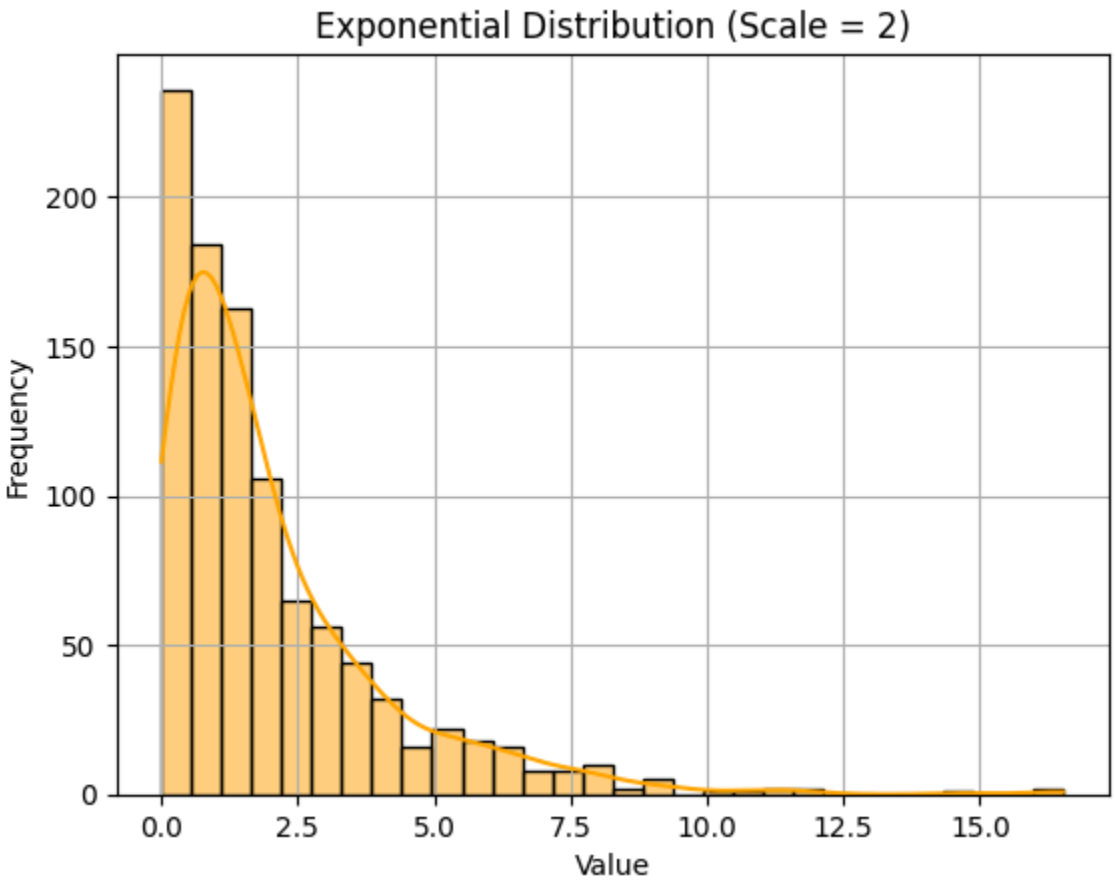
Exponential Distribution

In [86]:
```
scale = 2
size = 1000

data = np.random.exponential(scale = scale, size = size)

sns.histplot(data, bins = 30, kde = True, color = 'orange', edgecolor = 'black
plt.title(f"Exponential Distribution (Scale = {scale})")
plt.xlabel("Value")
plt.ylabel("Frequency")
```

```
plt.grid(True)
plt.show()
```

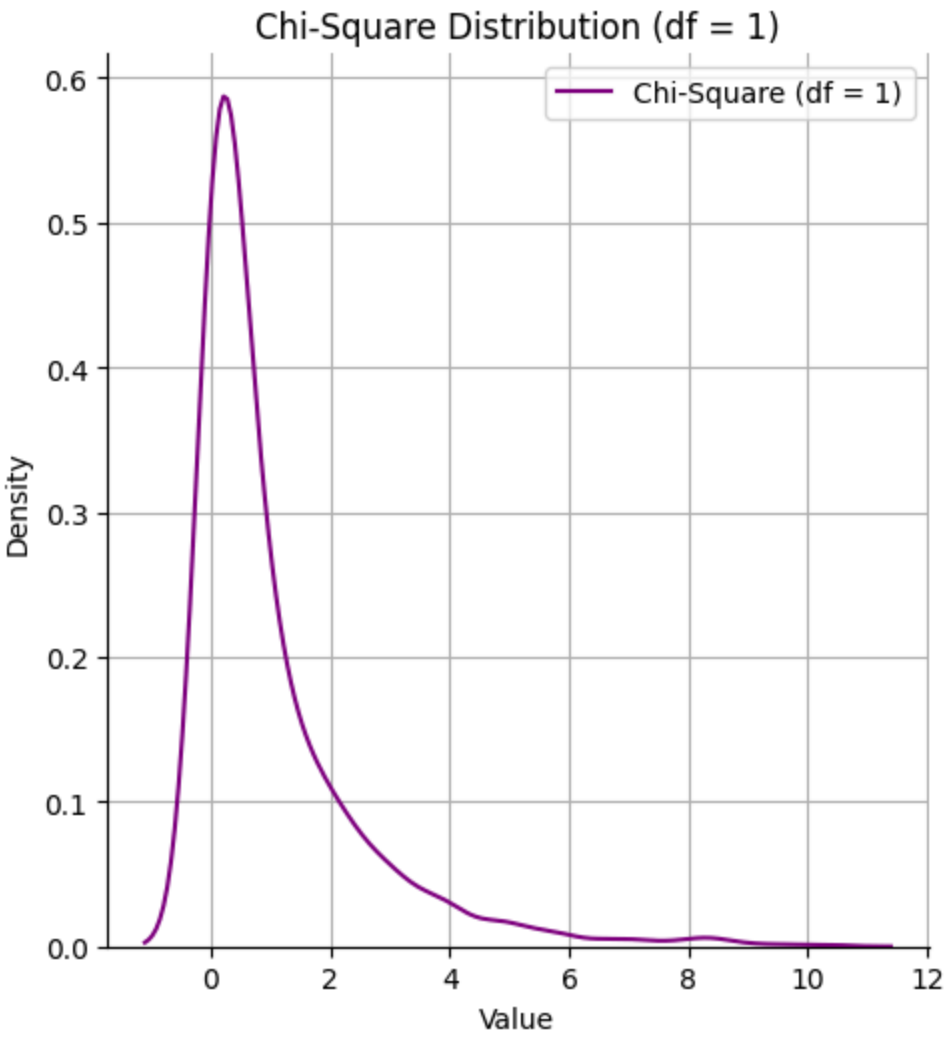


Chi-Square Distribution

In [87]:
```
df = 1
size = 1000

data = np.random.chisquare(df = df, size = size)

sns.displot(data, kind = "kde", color = 'purple', label = f'Chi-Square (df = {
plt.title(f"Chi-Square Distribution (df = {df})")
plt.xlabel("Value")
plt.ylabel("Density")
plt.legend()
plt.grid(True)
plt.show()
```



---

07 - Sparse Matrix with SciPY

Importing Modules

In [88]:
```
import numpy as np
from scipy.sparse import csr_matrix, csc_matrix, coo_matrix, lil_matrix, dok_m
```

CSR MATRIX

Compressed Sparse Row good for arithmetic and row access.

In [89]:
```
d = np.array([3, 4, 5, 7, 2, 6])     # data
r = np.array([0, 0, 1, 1, 3, 3])     # rows
```

```
c = np.array([2, 4, 2, 3, 1, 2])     # cols

csr = csr_matrix((d, (r, c)), shape = [4, 5])
print(csr.toarray())
```

```
[[0 0 3 0 4]
 [0 0 5 7 0]
 [0 0 0 0 0]
 [0 2 6 0 0]]
```

CSC MATRIX

Compressed Sparse Column efficient for column-based ops.

In [90]:
```
d = np.array([3, 4, 5, 7, 2, 6])     # data
r = np.array([0, 0, 1, 1, 3, 3])     # rows
c = np.array([2, 4, 2, 3, 1, 2])     # cols

csr = csc_matrix((d, (r, c)), shape = [4, 5])
print(csr.toarray())
```

```
[[0 0 3 0 4]
 [0 0 5 7 0]
 [0 0 0 0 0]
 [0 2 6 0 0]]
```

COO MATRIX

Coordinate format using (row, col, value) triples.

In [91]:
```
d = np.array([3, 4, 5, 7, 2, 6]) # data
r = np.array([0, 0, 1, 1, 3, 3]) # rows
c = np.array([2, 4, 2, 3, 1, 2]) # cols

coo = coo_matrix((d, (r, c)), shape = [4, 5])
print(coo.toarray())
```

```
[[0 0 3 0 4]
 [0 0 5 7 0]
 [0 0 0 0 0]
 [0 2 6 0 0]]
```

LIL MATRIX

List of Lists, modify rows easily before converting.

In [92]:
```
lil = lil_matrix((4, 5))
lil[0, 2] = 3
lil[0, 4] = 4
lil[1, 2] = 5
lil[1, 3] = 7
lil[3, 1] = 2
lil[3, 2] = 6

print(lil.toarray())
```

```
[[0. 0. 3. 0. 4.]
 [0. 0. 5. 7. 0.]
 [0. 0. 0. 0. 0.]
 [0. 2. 6. 0. 0.]]
```

DOK MATRIX

Dictionary-like, ideal for random updates.

In [93]:
```
dok = dok_matrix((4, 5))
dok[0, 2] = 3
dok[0, 4] = 4
dok[1, 2] = 5
dok[1, 3] = 7
dok[3, 1] = 2
dok[3, 2] = 6

print(dok.toarray())
```

```
[[0. 0. 3. 0. 4.]
 [0. 0. 5. 7. 0.]
 [0. 0. 0. 0. 0.]
 [0. 2. 6. 0. 0.]]
```

DIA (Diagonal) MATRIX

Stores only diagonals, saves space.

In [94]:
```
d = np.array([[3, 0, 0, 0, 0], [0, 5, 0, 0, 0]])
offsets = np.array([0, -1])
dia = dia_matrix((d, offsets), shape = [4, 5])

print(dia.toarray())
```

```
[[3 0 0 0 0]
 [0 0 0 0 0]
 [0 5 0 0 0]
 [0 0 0 0 0]]
```