Numpy for tutoral

Wednesday, October 29, 2025 11:56 PM

Numpy là thư viện py cốt lõi để tính toán số, xử lý các mảng và ma trận lớn hiệu quả: array manipulation (thao tác mảng), mathematical, numpy linear algebra, random array generation

Fact: NumPy arrays are homogeneous. Vectorized operations in Numpy can be 10 to 100 times faster than equivalent Py loops

Numpy có các tính năng: toán tử, add, sort
.shape: Number of elements along with each axis and is returned as a tuple.
axis: axis of an array describes the order of the indexing into the array.
.dtype: is an example of numpy.dtype class. It describes how the bytes in the fixed-size block of memory corresponding to an array item should be interpreted.

Different Ways of Creating Numpy Array
Numpy.array(): called ndarry
numpy.fromiter(): The fromiter() function create a new one-dimensional array from an iterable object.
Syntax: numpy.fromiter(iterable, dtype, count=-1)
numpy.arange(): This is an inbuilt NumPy function that returns evenly spaced values within a given interval.
Syntax: numpy.arange(start , stop, step , dtype=None)
numpy.linspace(): This function returns evenly spaced numbers over a specified between two limits.
Syntax: numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)
numpy.empty(): This function create a new array of given shape and type without initializing value.
Syntax: numpy.empty(shape, dtype=float, order='C')
numpy.ones(): This function is used to get a new array of given shape and type filled with ones (1).
Syntax: numpy.ones(shape, dtype=None, order='C')
numpy.zeros(): This function is used to get a new array of given shape and type filled with zeros (0).
Syntax: numpy.ones(shape, dtype=None)
Lưu ý: nên dùng **order = "f"**
Numpy.full(): full 1 số
Systax: np.full(shape, value)

Using Random number generation
Np.random.rand(shape): [0,1]
Np.random.randn(shape)
Np.random.randint(shape)

Using Matrix creation routines
Np.eye(shape): đường chéo là 1
Np.diag(shape): đường chéo tăng tiến
Np.zeros_like(array): tạo ma trận 0 dựa trên ma trận có sẵn
Np.ones_like(array)

Indexing:
Ellipsis() in Indexing:
Cube = np.random.rand(4,4,4)
Print(cube[…,0])

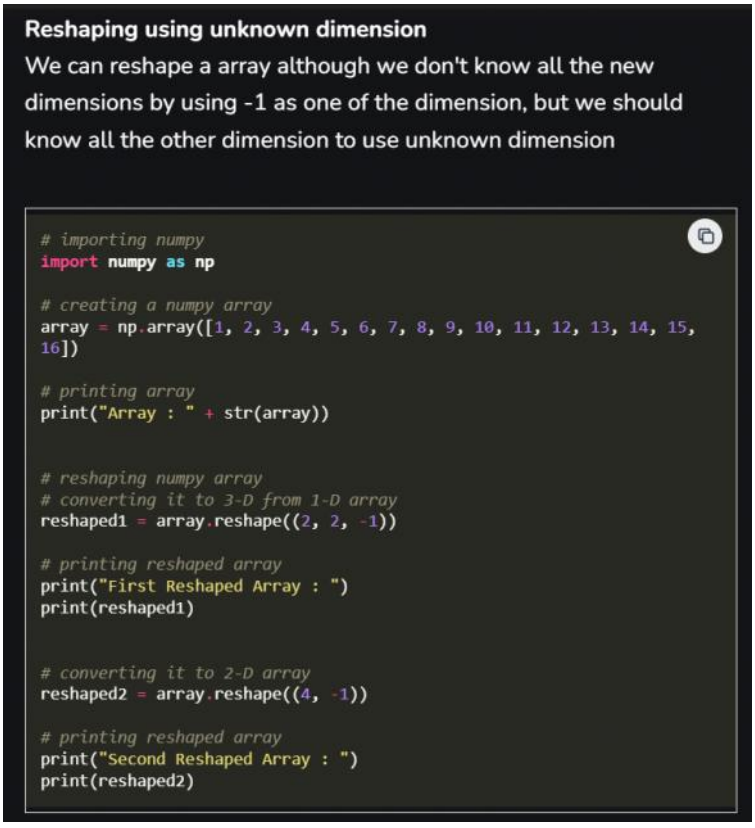
Using np.newaxis to Add New Dimensions
The np.newaxis keyword adds a new axis to the array which helps in converting a 1D array into a row or column vector.

Numpy tập 2

Thursday, October 30, 2025 11:19 AM

⚖️ 🟦 **Khi nào khác nhau**

Tình huống	`np.reshape()`	`array.reshape()`
Bạn có **biến không phải NumPy array** (ví dụ list)	✅ vẫn dùng được: `np.reshape([1,2,3,4], (2,2))`	❌ list không có `.reshape()`
Gọi **theo phong cách hàm toán học**	phù hợp trong pipeline kiểu functional	tiện hơn khi làm việc trực tiếp với mảng
Tính năng bổ sung (như `order='F'`)	cả hai đều hỗ trợ như nhau	cả hai đều hỗ trợ



Reshaping using unknown dimension

We can reshape a array although we don't know all the new dimensions by using -1 as one of the dimension, but we should know all the other dimension to use unknown dimension

```
# importing numpy
import numpy as np

# creating a numpy array
array = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16])

# printing array
print("Array : " + str(array))


# reshaping numpy array
# converting it to 3-D from 1-D array
reshaped1 = array.reshape((2, 2, -1))

# printing reshaped array
print("First Reshaped Array : ")
print(reshaped1)


# converting it to 2-D array
reshaped2 = array.reshape((4, -1))

# printing reshaped array
print("Second Reshaped Array : ")
print(reshaped2)
```

.reize(): 1x6 to 2x3
Với 1D
Np.sum(arr, axis = 0): giữ hàng gộp cột
Np.sum(arr, axis = 1): giữ cột gộp hàng
Axis = 2: stack theo cột
Stack: gộp các mảng numpy

So sánh trực quan

Cú pháp	Giải thích	Kết quả	Shape
`np.stack((a,b), axis=0)`	Thêm chiều mới **bên ngoài**, ghép **theo hàng**	`[[1 2 3],[4 5 6]]`	`(2,3)`
`np.stack((a,b), axis=1)`	Thêm chiều mới **bên trong**, ghép **theo cột**	`[[1 4],[2 5],[3 6]]`	`(3,2)`

Tóm tắt cách hiểu

Cấp mảng	Trước stack	Sau stack	Ý nghĩa `axis`
1D	(3,)	(2,3) hoặc (3,2)	`0` = thêm chiều ngoài, `1` = thêm chiều trong
2D	(2,3)	(2,2,3) hoặc (2,3,2)	`0` = chồng theo lớp, `1` = chồng theo hàng, `2` = chồng theo cột
3D	(2,2,2)	(2,2,2,2)	thêm trục 4D, vị trí tùy `axis`

Ghi nhớ ngắn gọn:

> `np.stack` = *nối các mảng có cùng shape, thêm* ***một chiều mới*** *tại vị trí* `axis`.
> `axis=0` thêm ở ngoài cùng (stack lên–xuống),
> `axis=-1` thêm ở trong cùng (stack sang–trái/phải).

Split

Cho mảng 1D

Np.split(array, n): khi này axis = 0, cho mảng 1 chiều

Np.array_split(array, n): it allows for uneven splitting of arrays

Cho mảng 2D

Np.split(array, n, axis = …)

Np.vsplit (matrix, 2): khi này axis =0: chia theo trục dọc

Np.hsplit (array,2): khi này axis =1: chia theo trục ngang

Cho mảng 3D

Np.dsplit(array, 2): khi này axis=2: trục thứ 3

<u>NumPy Array Broadcasting</u>

Các phép tính: arr1 +-*/ arr2. với * thì áp dụng như nhân ma trận. Tương tự add(), sub(), multiply(), divide(), power(), mod()

Hàm np.where(condition[,x,y]): quét điều kiện các phần tử của array đi tuần tự

Array.std(axis=0): tính độ lệch chuẩn tính theo cột. Đáp án là array2

Array.mean(axis=0): tính tb theo cột. Đáp án là array2

.sum(arr): tính tổng toàn bộ phần tử

.sum(arr, axis = …)

Tương tự min, max, mean

Linear Algebra with NumPy

Thursday, October 30, 2025 1:38 PM

ufunc's Trigonometric Functions in NumPy	
Function	**Description**
sin, cos, tan	compute the sine, cosine, and tangent of angles
arcsin, arccos, arctan	calculate inverse sine, cosine, and tangent
hypot	calculate the hypotenuse of the given right triangle
sinh, cosh, tanh	compute hyperbolic sine, cosine, and tangent
arcsinh, arccosh, arctanh	compute inverse hyperbolic sine, cosine, and tangent
deg2rad	convert degree into radians
rad2deg	convert radians into degree

Function	Description
amin, amax	returns minimum or maximum of an array or along an axis
ptp	returns range of values (maximum-minimum) of an array or along an axis
percentile(a, p, axis)	calculate the pth percentile of the array or along a specified axis
median	compute the median of data along a specified axis
mean	compute the mean of data along a specified axis
std	compute the standard deviation of data along a specified axis
var	compute the variance of data along a specified axis
average	compute the average of data along a specified axis

ufunc's Bit-twiddling functions in NumPy	
Function	**Description**
bitwise_and	performs bitwise and operation on two array elements
bitwies_or	performs bitwise or operation on two array elements
bitwise_xor	performs bitwise xor operation on two array elements
invert	performs bitwise inversion of an array of elements
left_shift	shift the bits of elements to the left
right_shift	shift the bits of elements to the left

MATHEMATICAL FUNCTION

Một số lệnh trọng tâm:
Np.around(arr, decimals = …): làm tròn
Np.round_(arr, decimals = ..): y chang cái trên
Np.divide(arr1, arr2)
Np.reciprocal(x,/,out=None,*,where=True): nghịch đảo
Np.isreal(array)
Np.conj(): lieen hợp số phức
Np.cbrt(arr): căn bậc 3
Np.clip(arr, min =…, max=…): lọc ra mảng mới khoảng giá trị

tan()	Compute tangent element-wise.
arcsin()	Inverse sine, element-wise.
arccos()	Trigonometric inverse cosine, element-wise.
arctan()	Trigonometric inverse tangent, element-wise.
arctan2()	Element-wise arc tangent of x1/x2 choosing the quadrant correctly.
degrees()	Convert angles from radians to degrees.
rad2deg()	Convert angles from radians to degrees.
deg2rad	Convert angles from degrees to radians.
radians()	Convert angles from degrees to radians.
hypot()	Given the “legs” of a right triangle, return its hypotenuse.
unwrap()	Unwrap by changing deltas between values to 2*pi complement.

FUNCTION	DESCRIPTION
tanh()	Compute hyperbolic tangent element-wise.
arcsinh()	Inverse hyperbolic sine element-wise.
arccosh()	Inverse hyperbolic cosine, element-wise.
arctanh()	Inverse hyperbolic tangent element-wise.

FUNCTION	DESCRIPTION
rint()	Round to nearest integer towards zero.
fix()	Round to nearest integer towards zero.
floor()	Return the floor of the input, element-wise.
ceil()	Return the ceiling of the input, element-wise.
trunc()	Return the truncated value of the input, element-wise.

FUNCTION	DESCRIPTION
expm1()	Calculate exp(x) – 1 for all elements in the array.
exp2()	Calculate 2**p for all p in the input array.
log10()	Return the base 10 logarithm of the input array, element-wise.
log2()	Base-2 logarithm of x.
log1p()	Return the natural logarithm of one plus the input array, element-wise.
logaddexp()	Logarithm of the sum of exponentiations of the inputs.
logaddexp2()	Logarithm of the sum of exponentiations of the inputs in base-2.

FUNCTION	DESCRIPTION
add()	Add arguments element-wise.
positive()	Numerical positive, element-wise.
negative()	Numerical negative, element-wise.
multiply()	Multiply arguments element-wise.
power()	First array elements raised to powers from second array, element-wise.
subtract()	Subtract arguments, element-wise.
true_divide()	Returns a true division of the inputs, element-wise.
floor_divide()	Return the largest integer smaller or equal to the division of the inputs.
float_power()	First array elements raised to powers from second array, element-wise.
mod()	Return the element-wise remainder of division.
remainder()	Return element-wise remainder of division.
divmod()	Return element-wise quotient and remainder simultaneously.

FUNCTION	DESCRIPTION
convolve()	Returns the discrete, linear convolution of two one-dimensional sequences.
sqrt()	Return the non-negative square-root of an array, element-wise.
square()	Return the element-wise square of the input.
absolute()	Calculate the absolute value element-wise.
fabs()	Compute the absolute values element-wise.
sign()	Returns an element-wise indication of the sign of a number.
interp()	One-dimensional linear interpolation.
maximum()	Element-wise maximum of array elements.
minimum()	Element-wise minimum of array elements.
real_if_close()	If complex input returns a real array if complex parts are close to zero.
nan_to_num()	Replace NaN with zero and infinity with large finite numbers.
heaviside()	Compute the Heaviside step function.

Sign, logdet = np.linalg.slogdet(arr): trả về sign (dấu của định thức), logdet (ln của absolute của det). Nhằm tránh lỗi tràn số

Sau đó sign * np.exp(logdet) => định thức

Np.linalg.det(array): tính det (phù hợp MT nhỏ)

Np.linalg.inv(arr): nghịch đảo ma trận

Np.dot(arr1, arr2): dot(a, b)[i,j,k,m] = sum(a[i,j,:] * b[k,:,m]): tích vô hướng (áp dụng cho ma trận hoặc vector). **Có thể trả về vector hoặc mảng**

Np.vdot(vector1,vector2): tích vô hướng dùng riêng cho vector. DÙng được cho số phức (lấy liên hợp phức của mảng thứ 1 trước khi nhân). Dot không liên hợp được. Vdot là hàm duy nhất làm phẳng mảng. **Luôn trả về 1 số**

Np.inner(arr1, arr2, out=None): tích vô hướng theo chiều ngang. **Có thể trả về mảng**

Np.outer(arr1, arr2, out=None): nhân từng phần tử mảng 1 với toàn bộ mảng 2 tạo thành ma trận 2D. O(n^2). **Luôn trả về ma trận**
Np.cross(arr1, arr2): tính tích có hướng

Random Number Generation and Statistics

Thursday, October 30, 2025 2:55 PM

Np.random.randint(low = …, high = …, size = …)

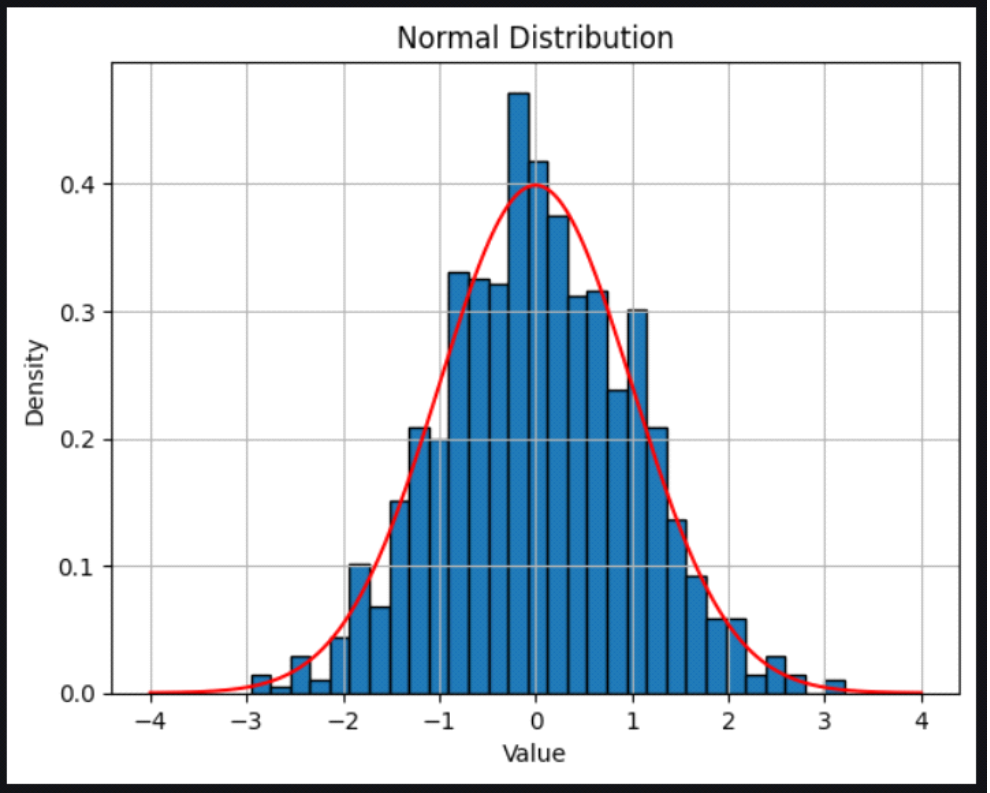
Normal/Gaussian Distribution: It is widely used to model real-world phenomena such as IQ scores, heart rates, test results and many other naturally occurring events.
Np.random.normal(loc, scale, size)

Visualizing the Normal Distribution

Visualizing the generated numbers helps in understanding their behavior. Below is an example of plotting a histogram of random numbers generated using `numpy.random.normal`.

```
import numpy as np
import matplotlib.pyplot as plt

data = np.random.normal(loc=0, scale=1, size=1000)

plt.hist(data, bins=30, edgecolor='black', density=True)
pdf = norm.pdf(x, loc=loc, scale=scale)
plt.plot(x, pdf, color='red', label='Theoretical PDF')
plt.title("Normal Distribution")
plt.xlabel("Value")
plt.ylabel("Density")
plt.grid(True)
plt.show()
```

Output:



Normal Distribution

The histogram represents the frequency of the generated numbers and the curve shows the theoretical pattern for comparison. The curve of a Normal Distribution is also known as the Bell Curve because of the bell-shaped curve.

Binomial Distribution: It models the number of successes in a fixed number of independent trials where each trial has only two possible outcomes: success or failure. This distribution is widely used in scenarios like coin flips, quality control and surveys.

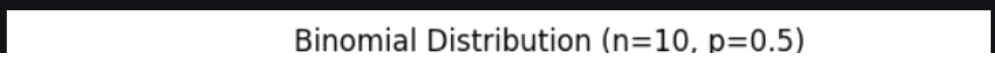
Np.random.binomial (n, p, size =None)

Visualizing the Binomial Distribution

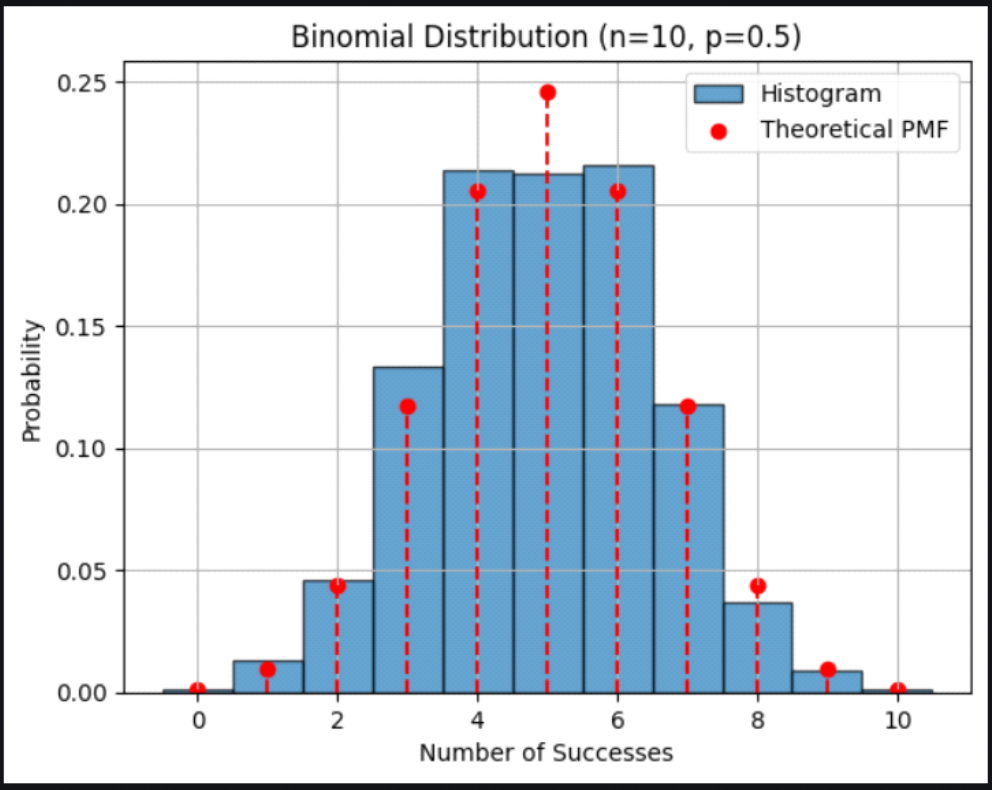
Visualizing the generated numbers helps in understanding their behavior. Below is an example of plotting a histogram of random numbers generated using `numpy.random.binomial`.

```
import numpy as np
import matplotlib.pyplot as plt

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='black',
alpha=0.7, label='Histogram')

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()
```

Output:



Output:



Binomial Distribution

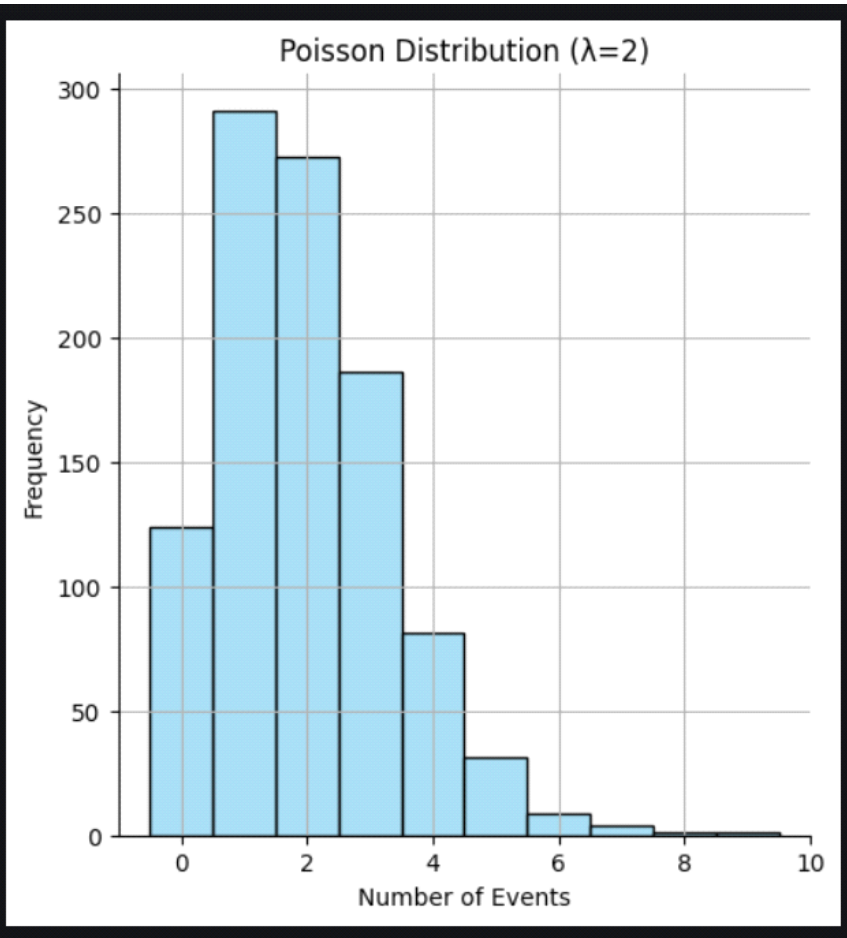
The image shows a **Binomial Distribution** with 10 trials (n=10) and a 50% success rate (p=0.5). The blue bars represent simulated data and the red dots show the expected probabilities. The distribution is symmetric, centered around 5 successes.

Poisson Distribution: The Poisson Distribution model the number of times an event happens within a fixed time or space when we know the average number of occurrences. It is used for events that occur independently such as customer arrivals at a store, Website clicks where events happen independently.
Numpy.random.poisson(lam = 1.0, size = None)

Visualizing the Poisson Distribution

To understand the distribution better we can visualize the generated numbers. Here is an example of plotting a histogram of random numbers generated using `numpy.random.poisson`.

```
import numpy as np
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

lam = 2
size = 1000

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

sns.displot(data, kde=False, bins=np.arange(-0.5,
max(data)+1.5, 1), color='skyblue', edgecolor='black')

plt.title(f"Poisson Distribution (λ={lam})")
plt.xlabel("Number of Events")
plt.ylabel("Frequency")
plt.grid(True)

plt.show()
```



Poisson Distribution

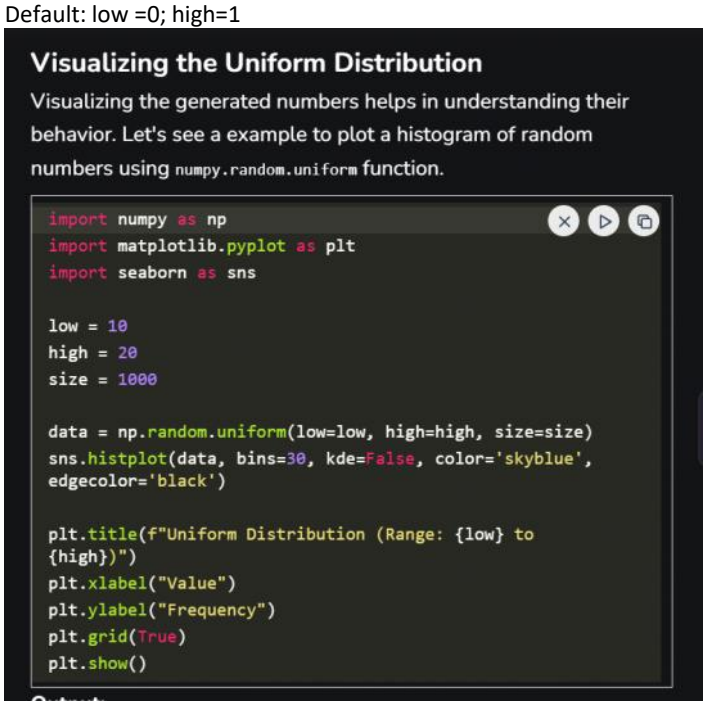
The image shows a Poisson Distribution with λ=2 displaying the frequency of events. The histogram represents simulated data highlighting the peak at 0 and 1 events, with frequencies decreasing as the number of events increases.

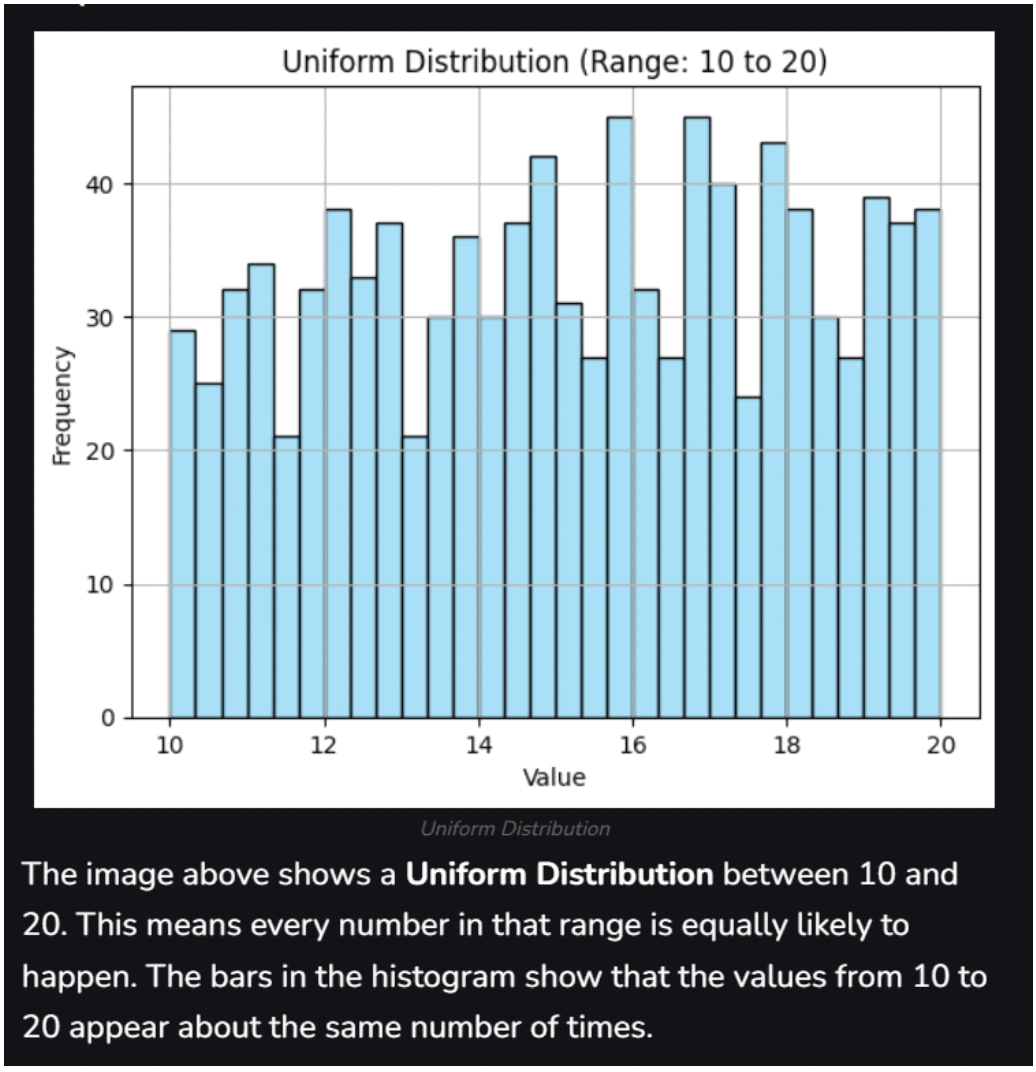
Uniform Distribution

A Uniform Distribution is used when all the numbers in a range have the same chance of being picked. For example, if we choose a number between 10 and 20 and every number in that range is just as likely as any other.

Np.random.uniform(low, high, size)

Default: low =0; high=1





Uniform Distribution

The image above shows a **Uniform Distribution** between 10 and 20. This means every number in that range is equally likely to happen. The bars in the histogram show that the values from 10 to 20 appear about the same number of times.

Exponential Distribution

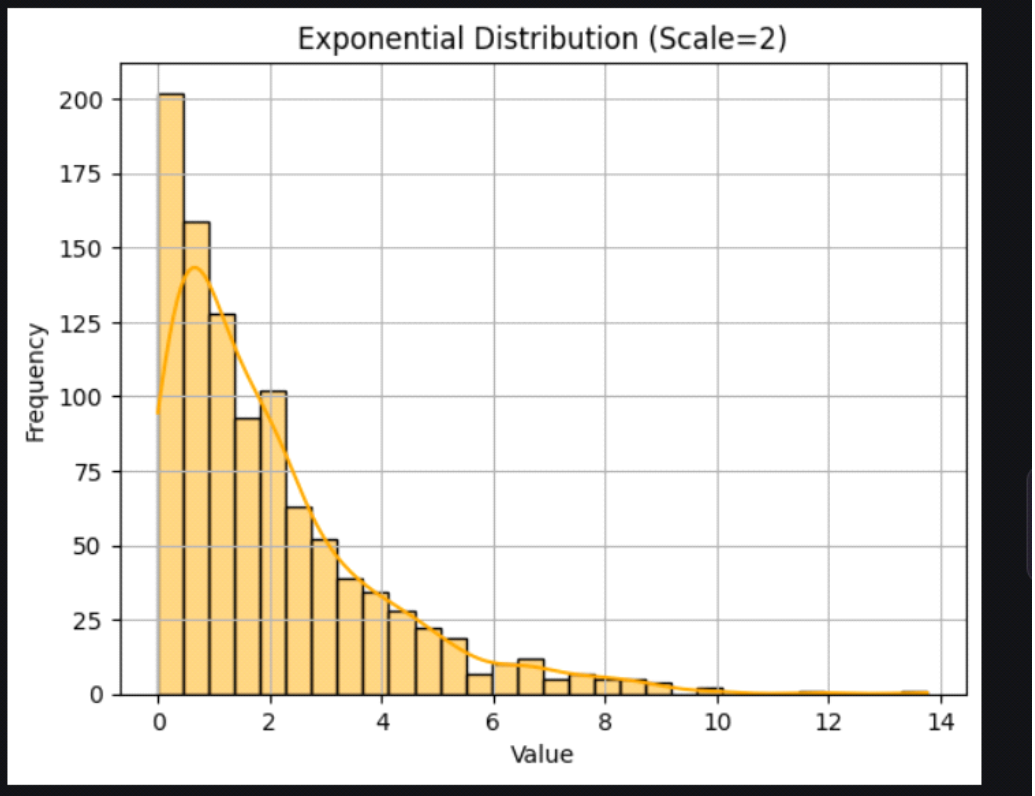
The Exponential Distribution is a fundamental concept in probability and statistics. It describe the time between events in a Poisson process where events occur continuously and independently at a constant average rate.

Np.random.exponential(scale = 1.0, size = None)

Visualizing the Exponential Distribution

Visualizing the generated numbers helps in understanding their behavior. Below is an example of plotting a histogram of random numbers generated using `numpy.random.exponential`.

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

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()
```



Exponential Distribution

The above image shows an Exponential Distribution with a scale parameter of 2. The histogram represents simulated data while the orange curve depicts the theoretical distribution.

Chi-Square Distribution in Numpy

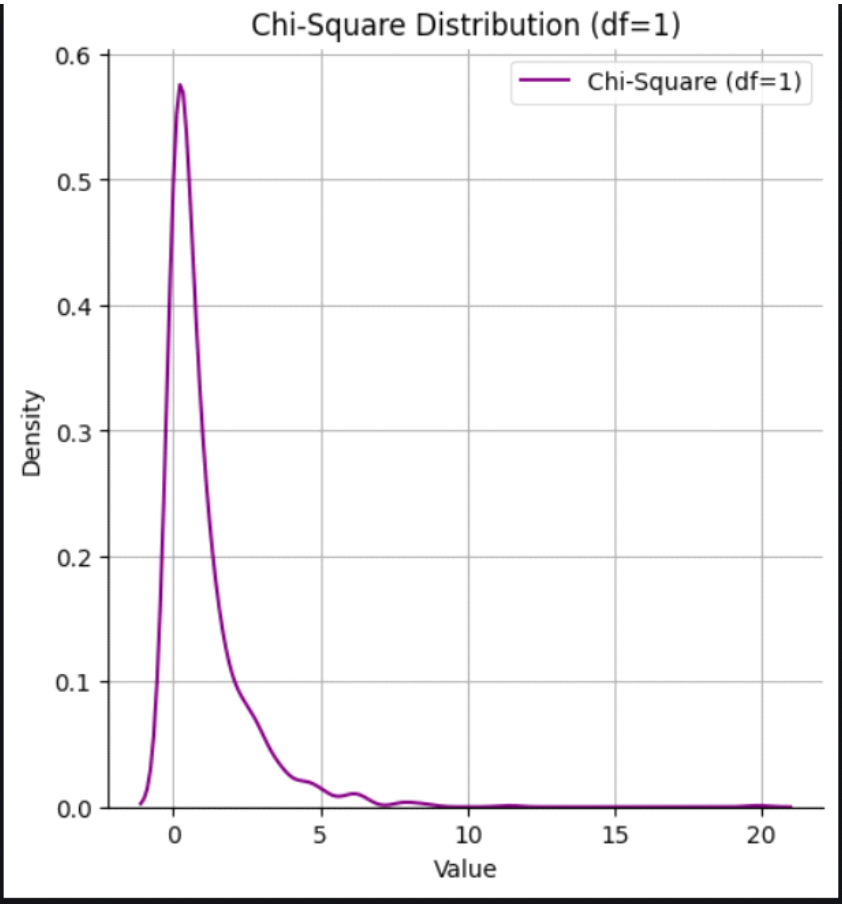
The Chi-Square Distribution is used in statistics when we add up the squares of independent random numbers that follow a standard normal distribution. It is used in hypothesis testing to check whether observed data fits a particular distribution or not

Np.random.chisquare(df, size = None)

Visualizing the Chi-Square Distribution

Visualizing the generated numbers helps to understand the behavior of the Chi-Square distribution. You can plot a histogram or a density plot using libraries like Matplotlib and Seaborn.

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = 1
size = 1000

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

sns.displot(data, kind="kde", color='purple', label=f'Chi-
Square (df={df})')

plt.title(f"Chi-Square Distribution (df={df})")
plt.xlabel("Value")
plt.ylabel("Density")
plt.legend()
plt.grid(True)

plt.show()
```



Chi-Square Distribution

The above chart shows the **shape** of the Chi-Square distribution for `df = 1`:

- The **x-axis** represents the values generated.
- The **y-axis** shows the **density** (how often values occur).
- With `df = 1` the curve is **skewed to the right** meaning lower values occur more frequently and higher values become rarer.