In [ ]:
```
                  " NumPy Random"
```

What is a Random Number?

Random number does NOT mean a different number evry time.Random means somthing that cannot be predicted logically.

"Pseudo Random and True Random "

Computer works on programs,and programs are definitive set of instructions,so it means there must be some algorithm to generate a random number as well

If there is a program to generate random number it can be predicted,thus not truly random

Random numbers are generated through a generation algorithm are called "Pseudo random"

In [ ]:
```
*can we make a truly random numbers?
```

yes,In order to generate a truly random number on our computers we need to get the random data from outside source.This outside source is generally our keystrokes,mouse movements ,data on network etc...

we do not need truly random numbers unless it is related to security (eg.encryption keys)r the basis of application is the randomness (e.g. Digital roulette wheels).

In this tutorial we will be using pseudo random numbers.

In [ ]:
```
" Generate a random number"
```

NumPy offers the "random" module to work with random numbers.

In [ ]:
```
#generate a random number between 0 to 100.

from numpy import random
x=random.randint(100)
print(x)
```

In [ ]:
```
"Generate a random float"
```

the random module's "rand()" method will return a random float between 0 and 1

In [8]:
```
#generate a random float between 0 and 1

from numpy import random
x=random.rand()
print(x)
```

```
0.0660155497032352
```

In [10]:
```
random.random()
```

Out[10]:
```
0.09485102460128592
```

In [18]:
```
x=random.randrange(100,200,2)
print(x)
```

```
---------------------------------------------------------------
AttributeError                Traceback (most recent call last)
Cell In[18], line 1
----> 1 x=random.randrange(100,200,2)
      2 print(x)

AttributeError: module 'numpy.random' has no attribute 'randrange'
```

In [20]:
```
"Generate Random Array"
```

Out[20]:
```
'Generate Random Array'
```

in numpys we work with arrays and you can use the two methods from the above examples to make random arrays

"Integers"

the "randint()" method tajkes a "size" parameter where you can specify the shape of an array

In [24]:
```
# Generate a 1-D array containing 5 random integers from 0 to 100:

from numpy import random
x=random.randint(100,size=(5))
```

```
print(x) #it takes random values bet 1 to 100 haveing 5 values
```

```
[33 64 72 75 58]
```

In [26]:
```
# Generate a 2-D array with 3 rows, each row containing 5 random integers from 0 to 100:

x=random.randint(100,size=(3,5)) #here 3 is rows and 5 is number of elements
print(x)
```

```
[[38  4 41 45  3]
 [11 42 50 14 39]
 [43 75 30  6 89]]
```

"Floats"

The rand() method also allows you to specify the shape of the array.

Example

In [29]:
```
# Generate a 1-D array containing 5 random floats:

x=random.rand(4)  #here 4 defines four random values bet 0 to 1
print(x)
```

```
[0.67005569 0.22833018 0.17409546 0.22400598]
```

In [31]:
```
# Generate a 2-D array with 3 rows, each row containing 5 random numbers:
x=random.rand(3,5)
print(x)
```

```
[[0.27478744 0.42850711 0.3164034  0.79050849 0.27784801]
 [0.63437253 0.83598716 0.29966445 0.01117776 0.32767372]
 [0.21824727 0.04913901 0.68618242 0.85688889 0.48871341]]
```

In [ ]:
```
"Generate random numner from array"
```

the "choice()" method allows you to generate a random value based on an arrays of values.

The "choice " method takes the array as a parametr and randomly returns one of the values.

In [34]:
```
x=random.choice([1,2,3,4,5,5,7])
print(x)
```

```
2
```

the 'choice()' method allows you to return an array of values.

Add a size parameter to specify the shape of the array.

In [37]:
```
# Generate a 2-D array that consists of the values in the array parameter (3, 5, 7, and 9):

x=random.choice([3,5,7,9],size=(3,5))
print(x)
```

```
[[7 9 3 3 7]
 [3 7 3 9 9]
 [5 7 7 3 3]]
```

In [ ]:
```
  "Random Data distribution"
```

what is data distribution?

data distribution is a list of all possible values and how often each vallue occurs.

Such lists are very importtant when working with stastics and datascience.

the random module offers some methods to that returns randomly generated data distributions.

In [ ]:
```
"Random Distribution"
```

A random distribution is set of random numbers that follws a certain probability density function.

"Probability Density Function": A function that descibes a continuous probability i.e probabillity of all values in an array.

we can generate a random numbers based on defined probabilities using the choice() method of the random module.

the choice() method allows us to specify the probability for each value.

the probability set by a number between 0 and 1,where 0 means that the value will never occur and 1 means that the value will always occur

In [ ]:
```
Generate a 1-D array containing 100 values, where each value has to be 3, 5, 7 or 9.
```

```
The probability for the value to be 3 is set to be 0.1

The probability for the value to be 5 is set to be 0.3

The probability for the value to be 7 is set to be 0.6

The probability for the value to be 9 is set to be 0
```

In [43]:
```
x=random.choice([3,5,7,9],p=[0.1,0.3,0.6,0.0],size=(100))
print(x) #values sum must be either 0 or 1
```

```
[7 3 5 7 7 7 5 7 7 3 7 3 7 5 5 7 5 7 5 7 3 7 5 7 5 3 7 7 7 5 3 7 5 5 5 7 5
 7 7 7 7 5 7 5 5 3 5 7 5 5 7 5 7 5 3 5 7 3 7 7 3 5 7 3 3 7 7 7 7 7 7 7 3 7
 7 7 5 7 3 5 7 5 3 7 7 5 3 7 5 7 7 5 3 7 7 5 7 3 5 7]
```

The sum of all probability numbers should be 1.

even if you run 100 times,the value 9 will never occur because it's value is 0.

you can return arrays of any shape and size by specifying the shape in the size parameter

In [48]:
```
# Same example as above, but return a 2-D array with 3 rows, each containing 5 values.
from numpy import random

x = random.choice([3, 5, 7, 9], p=[0.1, 0.3, 0.6, 0.0], size=(3, 5))

print(x)
```

```
[[7 7 7 3 5]
 [5 7 7 5 7]
 [7 5 7 7 7]]
```

In [ ]:
```
"random Permutations"
```

"random Permutations of elements"

A permutations refer to an arrangement of elements e.g [3,2,1,] is a permutaion of [1,2,3] and vice versa.

the numpy random module provides two methods for this: 1.shuffle() 2.permutation().

"Shuffling Arrays"

shuffle means changing arrangement of elements in-place i.e in the array itself.

In [51]:
```
# Randomly shuffle elements of following array:

from numpy import random
import numpy as np
```

In [53]:
```
arr=np.array([1,2,3,4,5])
random.shuffle(arr)
print(arr)
```

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

The shuffle() method makes changes to the original array

In [ ]:
```
"Generating Permutation of Arrays"
```

In [56]:
```
# Generate a random permutation of elements of following array:

arr=np.array([1,2,3,4,5,6,7])
x=random.permutation(arr)
print(x)
```

```
[3 5 2 1 6 7 4]
```

The permutation() method returns a re-arranged array (and leaves the original array un-changed).

In [ ]:
```
 "Seaborn"
```

"Visualise Distributions With Seaborn "

Seaborn is a python library uses matplotlib to plot graphs.It will be used to visulise random distributions.

Install Seaborn.

If you have Python and PIP already installed on a system, install it using this command:

C:\Users\Your Name>pip install seaborn

If you use Jupyter, install Seaborn using this command:

C:\Users\Your Name>!pip install seaborn

In [61]:
```
!pip install seaborn
```

```
Requirement already satisfied: seaborn in c:\users\gadam\anaconda3\lib\site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\gadam\anaconda3\lib\site-packages (from seaborn)
(1.26.4)
Requirement already satisfied: pandas>=1.2 in c:\users\gadam\anaconda3\lib\site-packages (from seaborn) (2.2.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\gadam\anaconda3\lib\site-packages (from seabo
rn) (3.8.4)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\gadam\anaconda3\lib\site-packages (from matplotlib!=
3.6.1,>=3.4->seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\gadam\anaconda3\lib\site-packages (from matplotlib!=3.6.
1,>=3.4->seaborn) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\gadam\anaconda3\lib\site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\gadam\anaconda3\lib\site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\gadam\anaconda3\lib\site-packages (from matplotlib!=3
.6.1,>=3.4->seaborn) (23.2)
Requirement already satisfied: pillow>=8 in c:\users\gadam\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>
=3.4->seaborn) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\gadam\anaconda3\lib\site-packages (from matplotlib!=
3.6.1,>=3.4->seaborn) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\gadam\anaconda3\lib\site-packages (from matplotl
ib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\gadam\anaconda3\lib\site-packages (from pandas>=1.2->sea
born) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\gadam\anaconda3\lib\site-packages (from pandas>=1.2->s
eaborn) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\gadam\anaconda3\lib\site-packages (from python-dateutil>=2.7
->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)
```

"Displots"

Distplots stand for distribution plot,it takes as input an array and plots a curve corresponding to the distributiono of points in the array.

"Import Matplotlib"

Import the pyplot object of the Matplotlib module in your code using the following following statement:

In [64]:
```
import matplotlib.pyplot as plt
```

"import Seaborn"

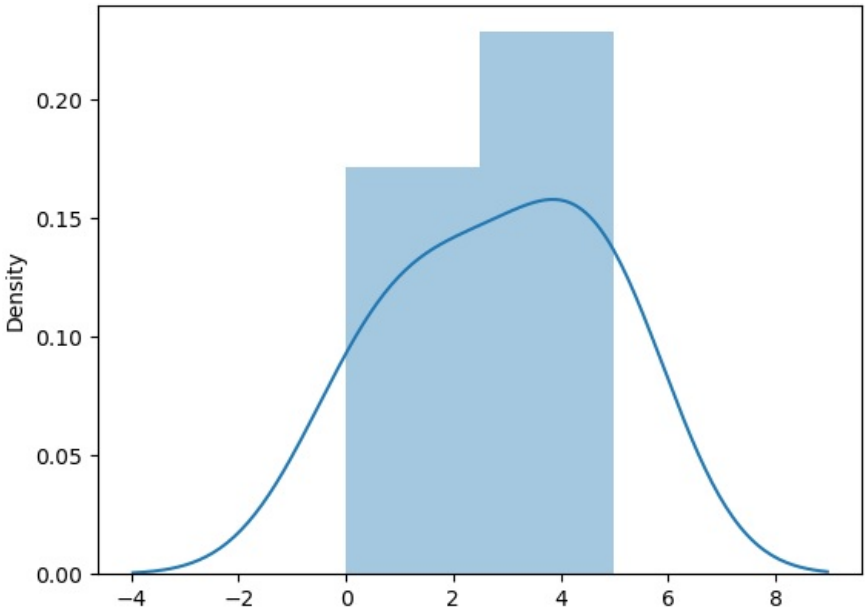
Import the Seaborn module in your code using the following statement:

In [ ]:
```
import seaborn as sns
```

In [ ]:
```
"Plotting a Distplot"
```

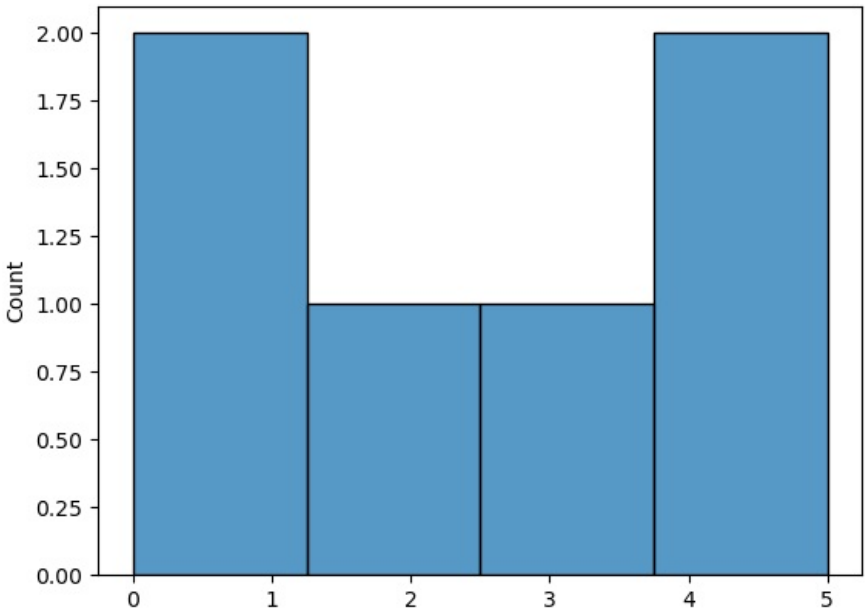
In [67]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot([0,2,1,3,4,5,5])
plt.show()
```

```
C:\Users\gadam\AppData\Local\Temp\ipykernel_20548\898389395.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot([0,2,1,3,4,5,5])
```

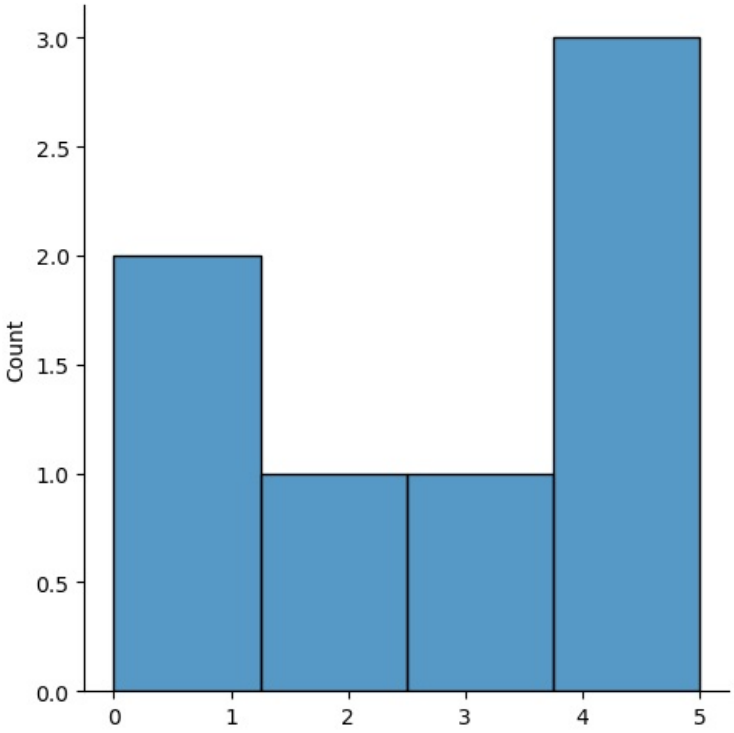


In [ ]:
```
instead of distplot we can use histplot or displot

because we version is not avaible
```

In [69]:
```
sns.histplot([0,2,1,3,4,5])
plt.show()
```

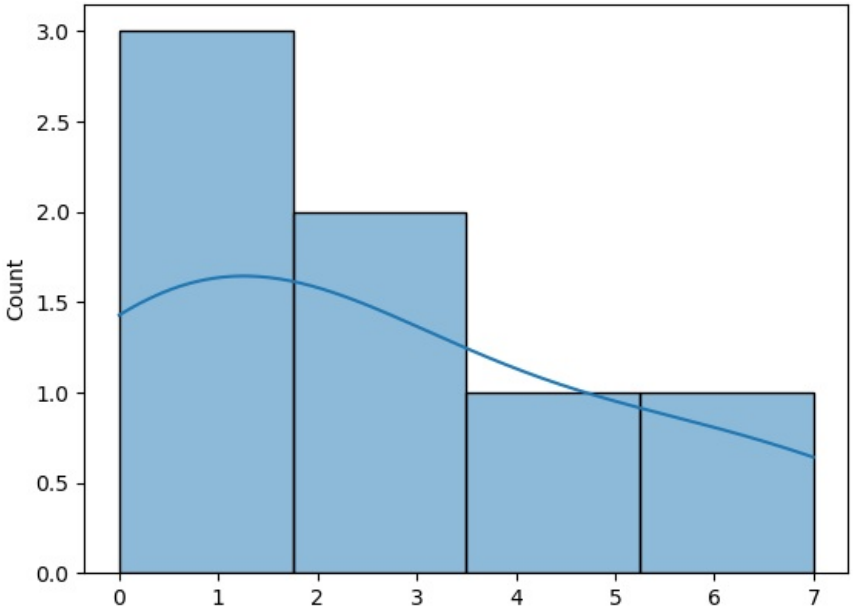


In [71]:
```
sns.displot([0,2,1,3,4,5,5])
plt.show()
```

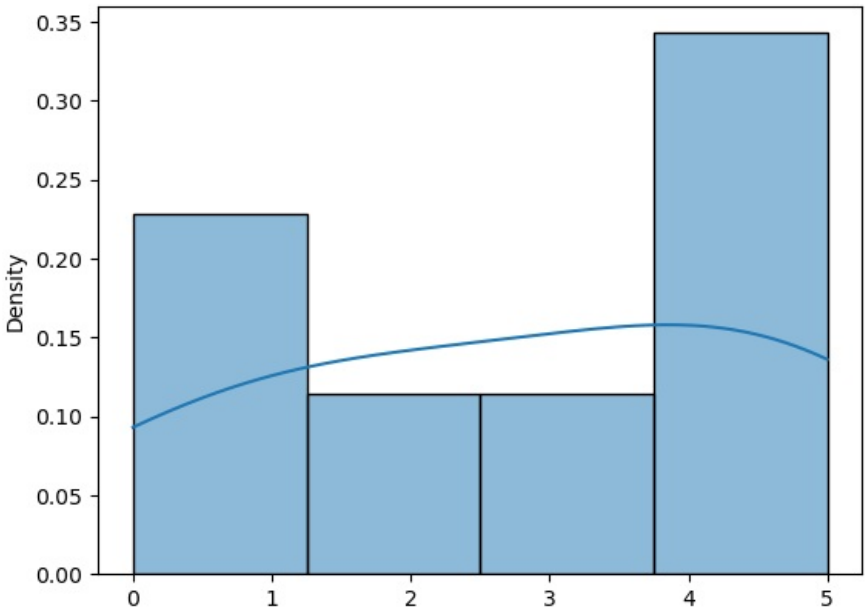


In [ ]:
```
Replicating basic distplot output
Both new functions support the kernel density estimate line, by passing kde=True:
```

In [73]:
```
sns.histplot([1,2,3,5,0,0,7],kde=True)
plt.show()
```



In [75]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.histplot([0,2,1,3,4,5,5],kde=True,stat="density")
plt.show()
```

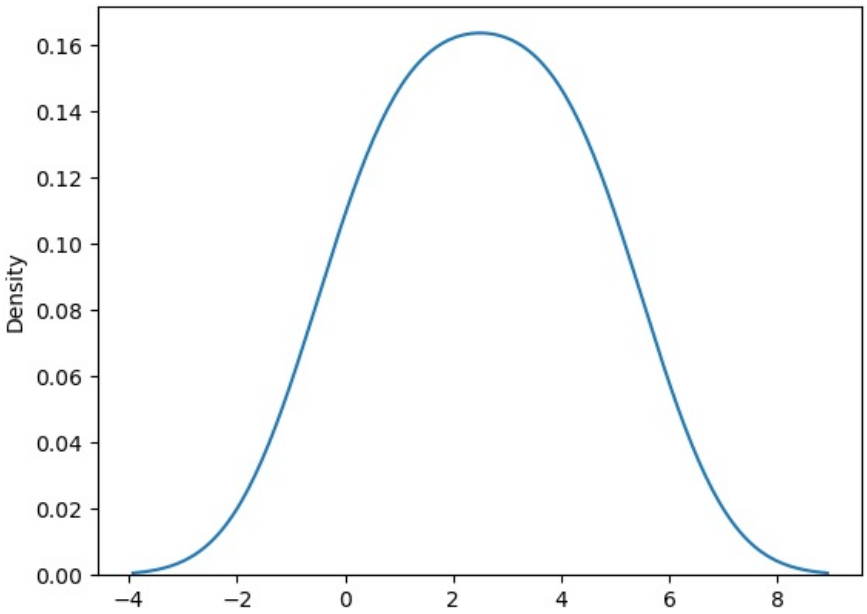


In [ ]:
```
"Plotting a distplot without the Histogram"
```

In [77]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

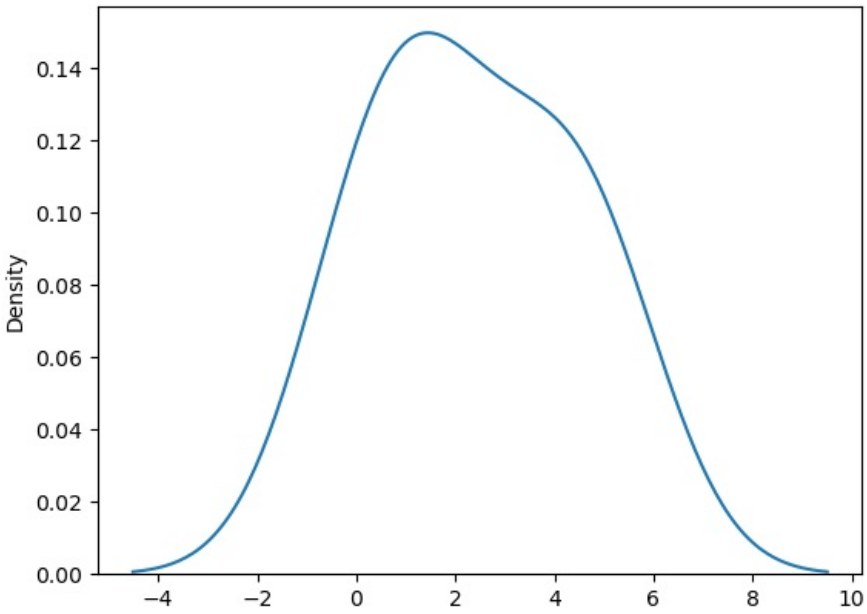
In [79]:
```
sns.distplot([0,1,2,3,4,5],hist=False)
plt.show()
```

```
C:\Users\gadam\AppData\Local\Temp\ipykernel_20548\299229084.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot([0,1,2,3,4,5],hist=False)
```



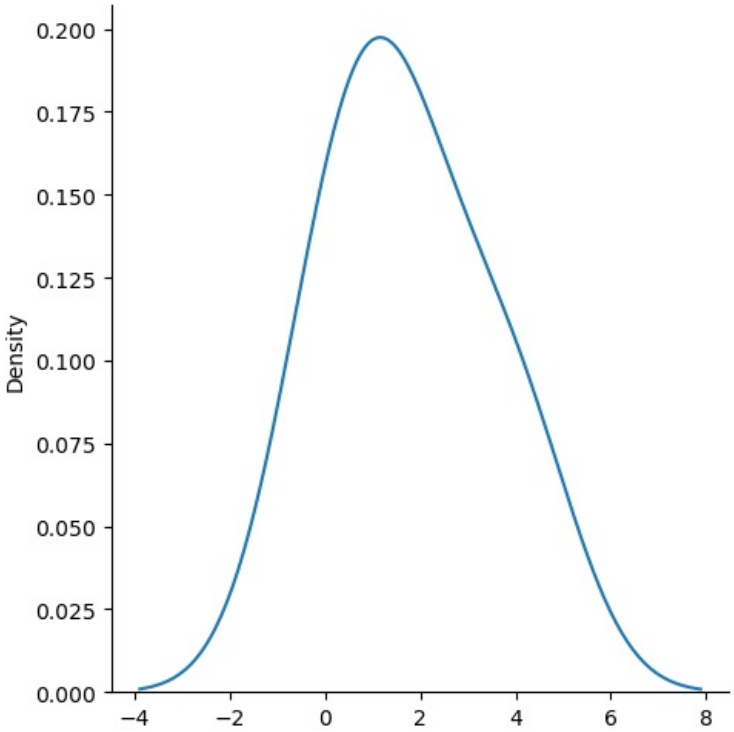
In contrast, the modern approach would call kdeplot directly for an axes-level plot:

In [82]:
```
sns.kdeplot([0,1,2,4,5])  #here updated version
plt.show()
```



In [ ]:
```
Or use kind="kde" in displot:
```

In [84]:
```
sns.displot([0,1,2,4] ,kind='kde')
plt.show()
```



Note: We will be using: sns.distplot(arr, hist=False) to visualize random distributions in this tutorial.

In [ ]:
```
       "Normal (Gaussian ) Distribution"
```

the normal distributions is one of the most important distributions.

It is also called the Gaussian Distribution after the German mathematician Carl Friedrich Gauss.

it fits the probability distributions of many events,eg.IQ Scores,HeartBeat

use the "random.normal() method to get a Normal Data Distribution

it has three parameters:

loc-(Mean) where the peak of the bell exists.

scale-(Standard Deviation) how flat the graph distribution should be.

size-the shape of the returned array.

In [88]:
```
# Generate a random normal distribution of size 2x3:
```

```
from numpy import random
x=random.normal(size=(2,3))
print(x)
```

```
[[2.2500824  0.30611929 0.64496449]
 [0.59353787 0.02840376 2.2629855 ]]
```

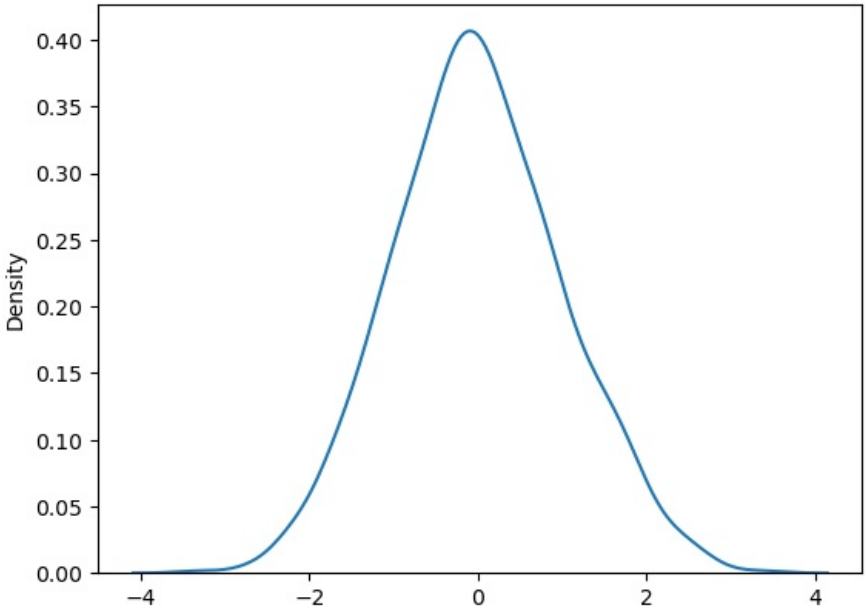
In [90]:
```
# Generate a random normal distribution of size 2x3 with mean at 1 and standard deviation of 2:

x=random.normal(loc=1 ,scale=2,size=(2,3))
print(x)
```

```
[[ 1.92230069  3.36911489  1.13068153]
 [ 0.99759367 -0.93146224  3.47990248]]
```

In [ ]:
```
"Visualization of Normal Distribution"
```

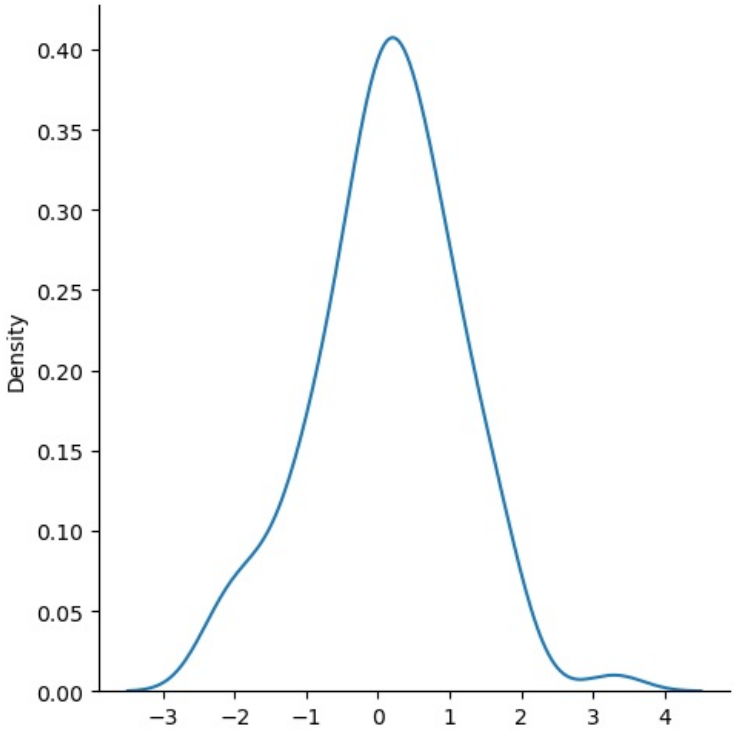
In [92]:
```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns
```

In [96]:
```
# sns.distplot(random.normal(size=1000), hist=False) old version
sns.kdeplot(random.normal(size=1000)) # or use displot kind ='kde'
plt.show()

#kde=kernel density estimate line
```



In [ ]:
```
Note: The curve of a Normal Distribution is also known as the Bell Curve because of the bell-shaped curve.
```

In [98]:
```
sns.displot(random.normal(size=(100)),kind='kde')
plt.show()
```



In [ ]:
```
          "Binomial Distribution"
```

Binomial ditribution is a Discrete Distribution.

It describes the outcomes of binary scenarios ,e.g .tossof coin ,it will either be ahead or tails.

It has three parameters

n-number of trails

p-probability of occuerence of each trail(e.g for toss of a coin 0.5 each)

size-the shape of the returned array.

Discrete Distribution: The distribution is defiend at seperate set of events e,g a coin toss's result is discrete as it can be only head or tails whereas height of people is continous as it can be 170,170.1,170.11 and so on.

finite numbers of outcomes is called dicreate distribution

infinite number of outcomes is called continuous distribution

In [102…
```
# Given 10 trials for coin toss generate 10 data points:

from numpy import random
x=random.binomial(n=10,p=0.5,size=10)
print(x)
```

```
[4 8 5 3 5 4 3 6 5 5]
```

In [ ]:
```
"sample space"
the range of all posibile outcomes of an experiment is known as "sample space"

A random variable is afunction which can take on any value from the sample space & having rangeof some set of r

example:- let's take an example of (experiment) tossing 3 coins at the same time.
s={hhh,hht,hth,thh,htt,tht,tth,ttt}->sample space

here let's suppose the no.of tails is the random variable x.
"discrete variable->finite"

x={x1,x2,x3,x4,xx5,x6,x7,x8}
 tails ={0,1,1,1,2,2,2,3}
```

"continuous random varaibles->" a random variable takes an infinite number of values is known as continuous random values. Many physical systems (experiments) can produce infinte no.of outcomes i afinite time of observation.in such cases we use continous random variables to define outcomes of such system

ex-1: A random variable that measures the time taken in completing a job is continuous random variable.

ex-2> noise voltage that is generated by an elctronic has a continuous amplified therefore sample space (s) & ranodm variable both are continuous

"what is Cdf:" the cummulative distribution function (cdf) of a random varaible 'x' may be defined as the probablity that the random variable 'x' takes a value "less then or equal to x"

mathematically CDF (Fx(x)] may be defined as CDF:Fx=p(X<=x)

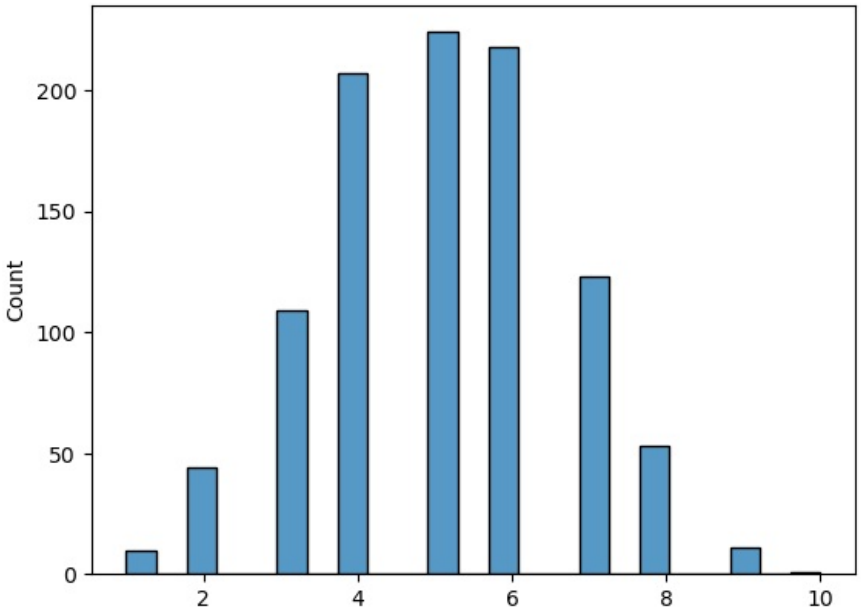
cdf may be defined for- continuous random vraible & discrete random variables

"other names of cdf" -probabillity distribution function of the random varaible -distribution function of the random variables -continuous probability distribution function""

properties of cdf:

In [ ]:
```
"visualisation of Binomial Distribution"
```

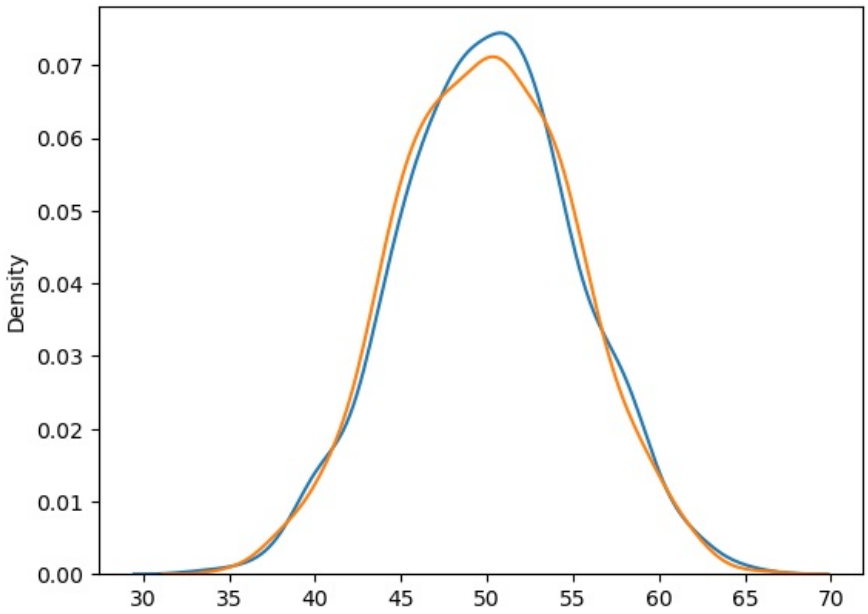
In [106…
```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.histplot(random.binomial(n=10,p=0.5,size=1000))
plt.show()
```



In [ ]:
```
"Difference Between Normal and Binomial Distribution"
```

the main difference is that normal distribution is continuous whereas binomail is discrete ,but if there are enough data points it will be quite similar to normal distribution with certain loc and scale.

In [109…
```
sns.kdeplot(random.normal(loc=50, scale=5, size=1000),  label='normal')
sns.kdeplot(random.binomial(n=100, p=0.5, size=1000), label='binomial')

plt.show()
```



In [ ]:
```
"Poisson Distribution"
```

Poisson distribution is discrete distribution

It estimates how many times an event can happen in specified time. ex: If a person eats twice a day what is the probability he will eat thrice?

It has two parameters:

lam-rate or known number of occurences

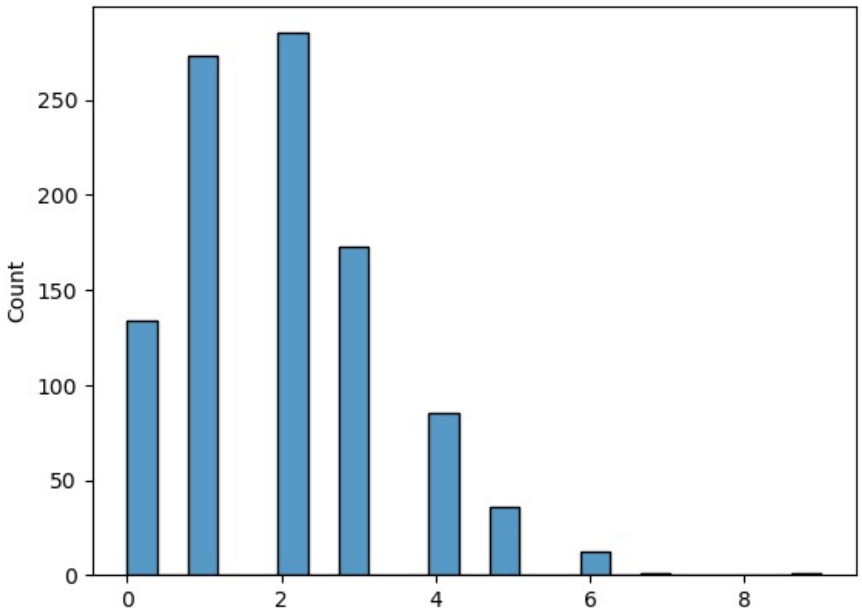
size-the shape of the returned array.

In [112…
```
# Generate a random 1x10 distribution for occurrence 2:
from numpy import random

x=random.poisson(lam=2,size=10)
print(x)
```

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

In [ ]:
```
"Visualization of Poisson Distribution"
```

In [114…
```
sns.histplot(random.poisson(lam=2,size=1000))
plt.show()
```



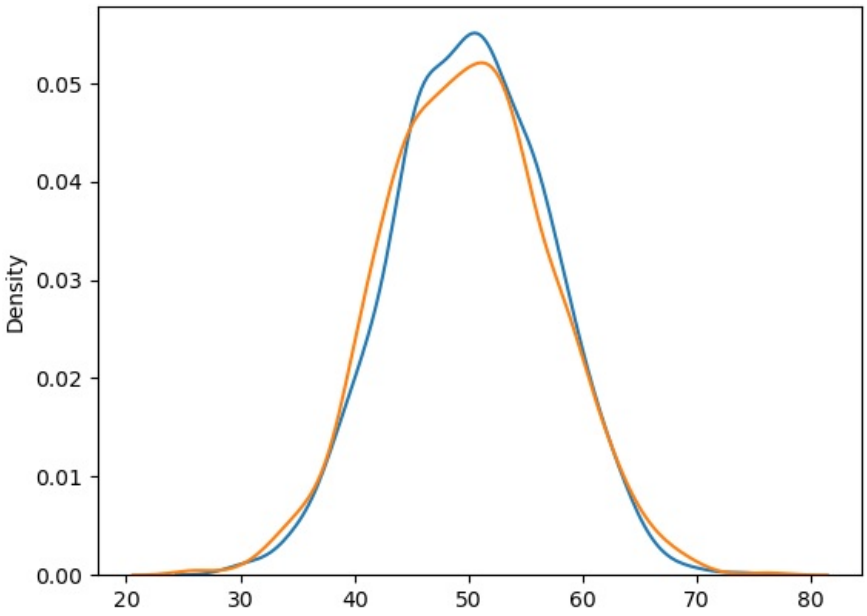
In [ ]:
```
"Difference Between Normal and Poisson Distribution"
```

normal distribution is continuous whereas poisson is discrete

but we can see that similar for binomail a large enough poisson distribution it will become similar to normal distribution with certain std

dev and mean

In [117…
```
sns.kdeplot(random.normal(loc=50,scale=7,size=1000),label="normal")
sns.kdeplot(random.poisson(lam=50, size=1000),label="poisson")
plt.show()
```

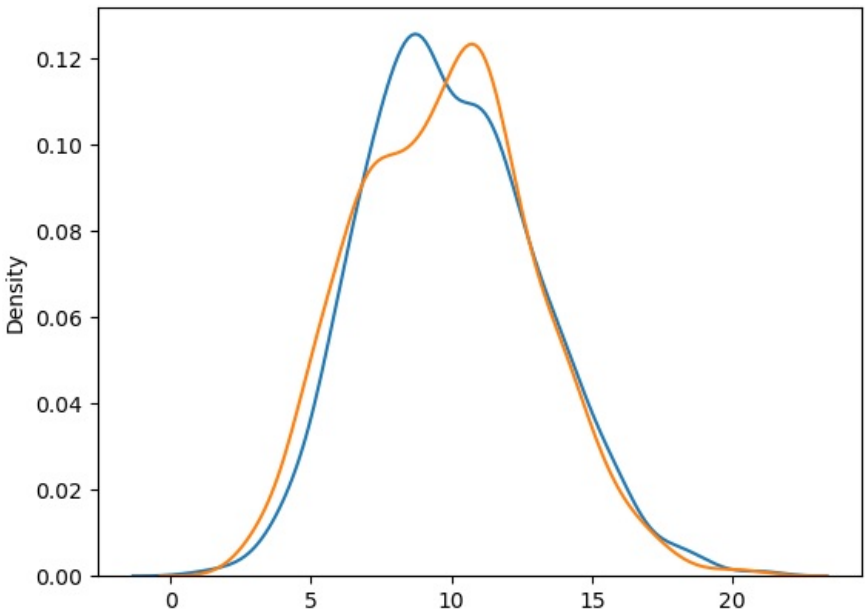


In [ ]:
```
"Difference Between Binomial and Poisson Distribution"
```

binomail distribution onlu has two possible outcomes ,whereas poisson distribution have unlimited possible outcomes.

but for very large "n" and near-zero "p" binomial distribution is near identical to poisson distribution such that n*p is nearly equal to "lam"

In [121…
```
sns.kdeplot(random.binomial(n=1000,p=0.01,size=1000),label="binomail")
sns.kdeplot(random.poisson(lam=10,size=1000),label="poisson")
plt.show()
```



In [ ]:
```
"Uniform Distribution"
```

used to descibe the probability where every even has equal chances of occuring

e.g: Generation of random numbers

It has three parameters :

a-lower bound - default 0.0

b- upper bound -default 1.0

size-shape of the returned array

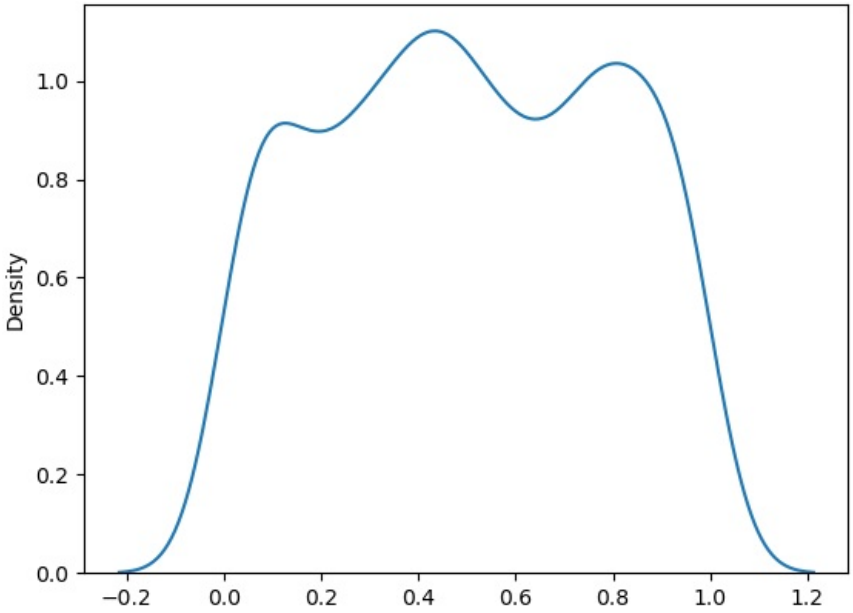
In [124…
```
# Create a 2x3 uniform distribution sample:
```

```
x=random.uniform(size=(2,3))
print(x)
```

```
[[0.30948164 0.86690953 0.47743948]
 [0.18639205 0.2406905  0.30260831]]
```

In [ ]:
```
"Visualization of Uniform Distribution"
```

In [126…
```
sns.kdeplot(random.uniform(size=1000))
plt.show()
```



In [ ]:
```
"logistic Distribution "
```

logistic distribution describes growth

used extensively in machine learning in logistic regression,neural networks

It has three parameters:

loc- mean,where the peak is default 0.

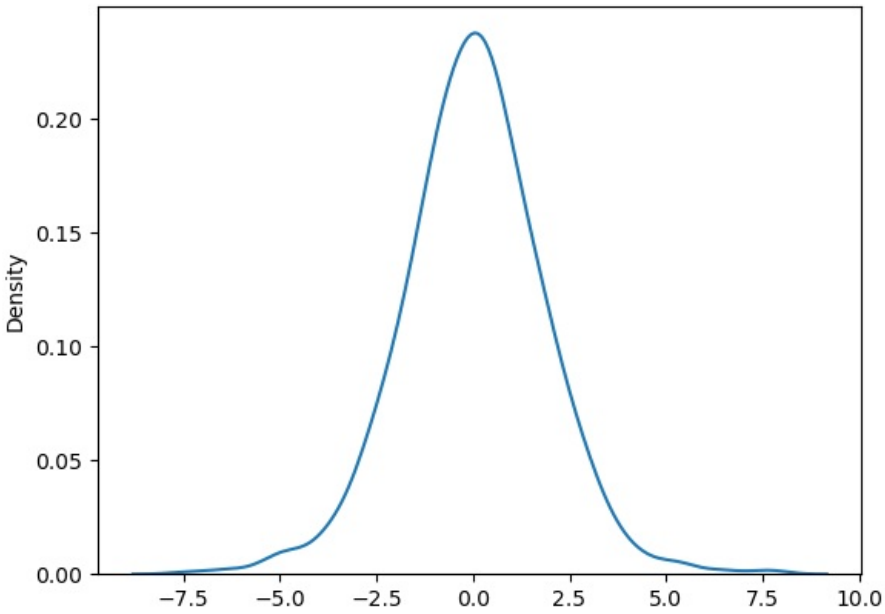
scale -std deviation ,the flatness of distribution .default 1

size-the shape of returned array

In [129…
```
# Draw 2x3 samples from a logistic distribution with mean at 1 and stddev 2.0:

x=random.logistic(loc=1,scale=2,size=(2,3))
print(x)
```

```
[[-0.10440253 -5.05074938  3.98946139]
 [ 1.45633336  3.58513195 -2.61345665]]
```

In [131…
```
sns.kdeplot(random.logistic(size=1000))
plt.show()
```

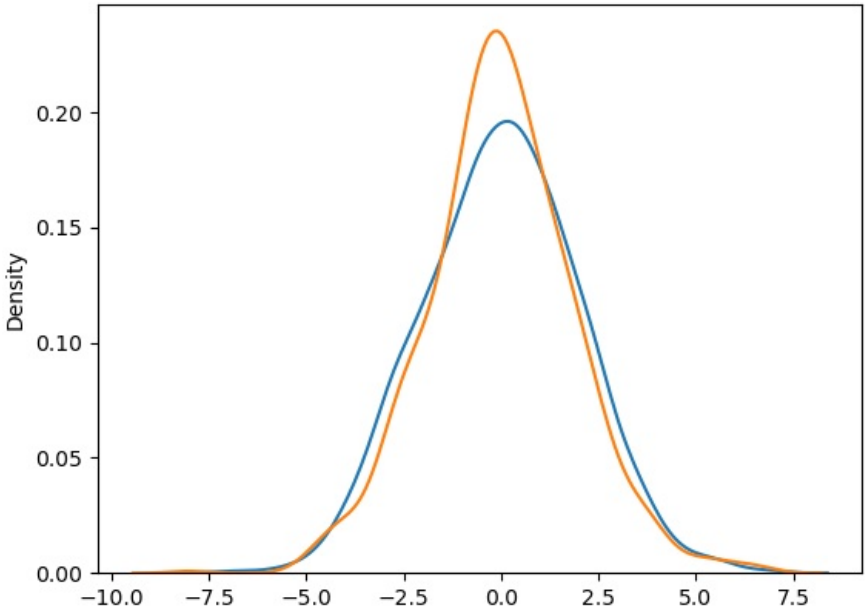


In [ ]:
```
"Difference Between Logistic and Normal Distribution"
```

both distributions are near identical,but logistic distribution has more are under the tails,meaning it represents more possibility of occurences of an even further away from mean.

for higher value of scale (std deviation) the normal and logistic distribution are near identical apart from the peak

In [134…
```
sns.kdeplot(random.normal(scale=2,size=1000))
sns.kdeplot(random.logistic(size=1000))
plt.show()
```



In [ ]:
```
"multinomial Distribution"
```

multinomail distribution is a generalisation of binomial distribution

It describes outcomes of multi-nomial scenarios unlike binomial where scenarios must be only one of two ,

eg : Blood type of apopulation, dice roll outcome.

n- number of possible outcomes(e.g 6 for dice roll)

pvals-list of probabilities of outcomes (eg.[1/6,1/6,1/6,1/6,1/6,1/6] for dice roll

size- shape of returned array

In [137…
```
x=random.multinomial(n=6,pvals=[1/6,1/6,1/6,1/6,1/6,1/6])
print(x)
```

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

Note: Multinomial samples will NOT produce a single value! They will produce one value for each pval.

Note: As they are generalization of binomial distribution their visual representation and similarity of normal distribution is same as that of multiple binomial distributions.

In [ ]:
```
"exponential distribution"
```

exponential distribution is used to describing time till next even

eg: failur/success

It has two parameters:

scale- inverse of rate(see lam in poisson distribution) defaults to 1.0

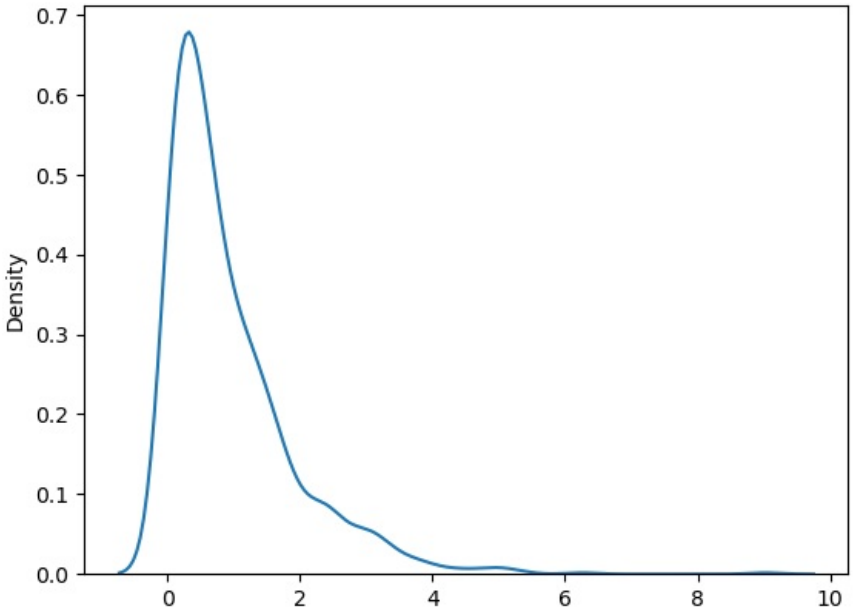
size-shape of the returned array.

In [142…
```
x=random.exponential(scale=2,size=(2,3))
print(x)
```

```
[[2.94531356 2.71970923 3.76810995]
 [0.03069814 0.31762821 2.1517275 ]]
```

In [ ]:
```
"Visualization of Exponential Distribution"
```

In [144…
```
sns.kdeplot(random.exponential(size=1000))
plt.show()
```



In [ ]:
```
"Relation Between Poisson and Exponential Distribution"
```

poisson distribution deals with number of occureneces of an event in a time period whereas exponential distribution deal with the time between these events.

In [ ]:
```
"Chi Square Distribution"
```

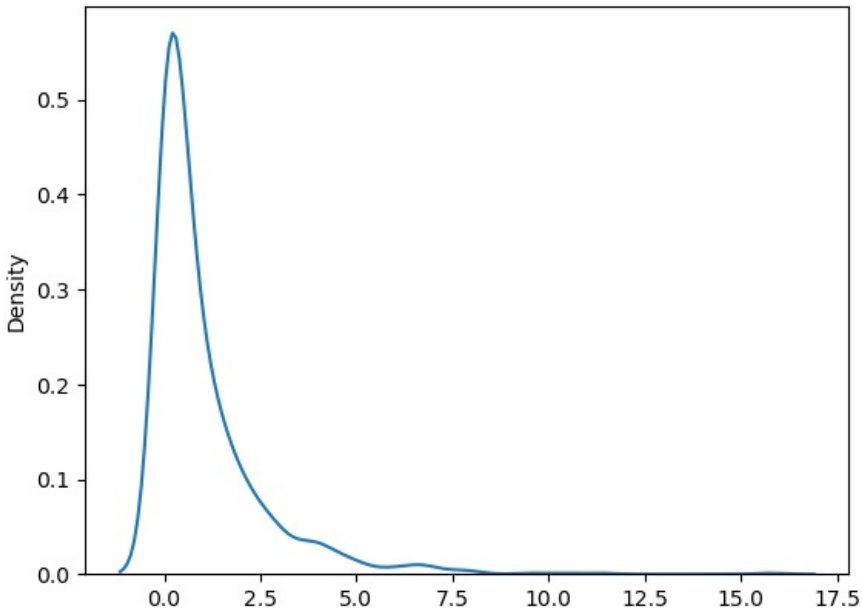
Chi square distribution is used as a basis of verify the hypothesis

it has two parametrs: df-(degree of freedom) size-shape of the returned array.

In [148…
```
# Draw out a sample for chi squared distribution with degree of freedom 2 with size 2x3:
from numpy import random

x = random.chisquare(df=2, size=(2, 3))

print(x)
```

```
[[1.25541842 4.95335898 3.71684166]
 [2.53232325 8.75785893 7.66729335]]
```

In [150…
```
sns.kdeplot(random.chisquare(df=1,size=1000))
plt.show()
```



In [ ]:
```
"Rayleigh Distribution"
```

Rayleigh Distribution is used in signal processing.

It has two parameters:

scale - (standard deviation) decides how flat the distribution will be default 1.0).

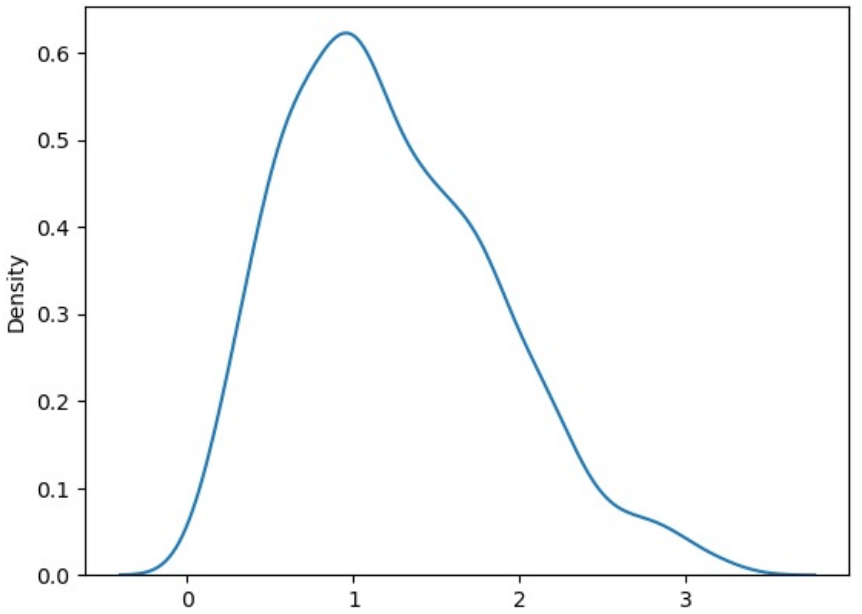
size - The shape of the returned array.

In [153…
```
# Draw out a sample for rayleigh distribution with scale of 2 with size 2x3:

x=random.rayleigh(scale=2,size=(2,3))
print(x)
```

```
[[2.83566999 3.66600605 3.90273745]
 [1.73551902 2.39999119 2.52178808]]
```

In [ ]:
```
"Visualization of Rayleigh Distribution"
```

In [155…
```
sns.kdeplot(random.rayleigh(scale=1,size=1000))
plt.show()
```



In [ ]:
```
"Similarity Between Rayleigh and Chi Square Distribution"
```

At unit stddev and 2 degrees of freedom rayleigh and chi square represent the same distributions.

In [ ]:
```
"Pareto Distribution"
```

A distribution follows paretos's law i.e 80-20 distribution (20% factors cause 80% outcome).

It has two parameter:

a - shape parameter.

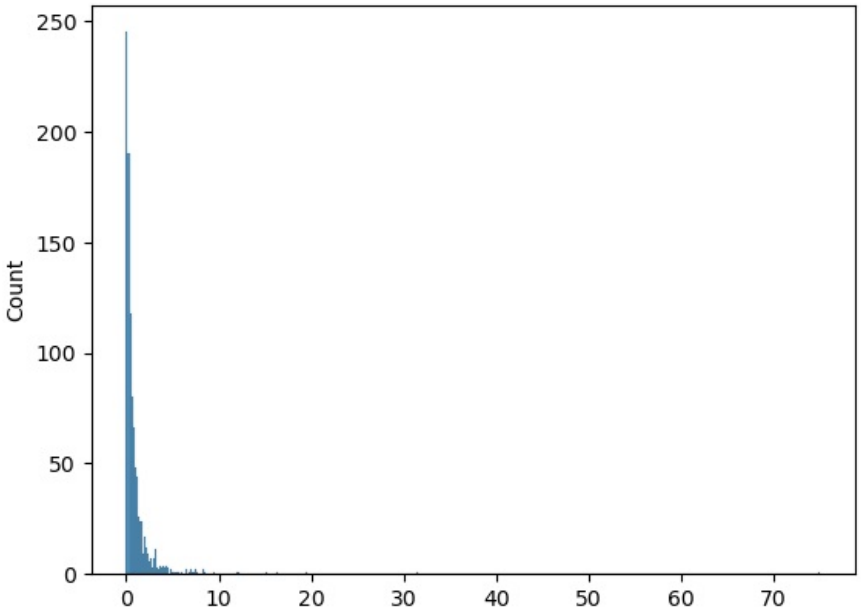
size - The shape of the returned array.

In [159…
```
# Draw out a sample for pareto distribution with shape of 2 with size 2x3:

x=random.pareto(a=2,size=(2,3))
print(x)
```

```
[[0.9943856  0.66353796 0.34820001]
 [0.11104681 2.85937723 1.76300048]]
```

In [ ]:
```
"Visualization of Pareto Distribution"
```

In [161…
```
sns.histplot(random.pareto(a=2,size=1000))
plt.show()
```



In [ ]:
```
"Zipf Distribution"
```

Zipf distributions are used to sample data based on zipf's law.

zipf's laws: In a collection,the nth common term is 1/n times of the most common term E.g 5th most common word in english occurs nearly 1/5 times often as the most common word.

It has two parameters:

a - distribution parameter.

size - The shape of the returned array.

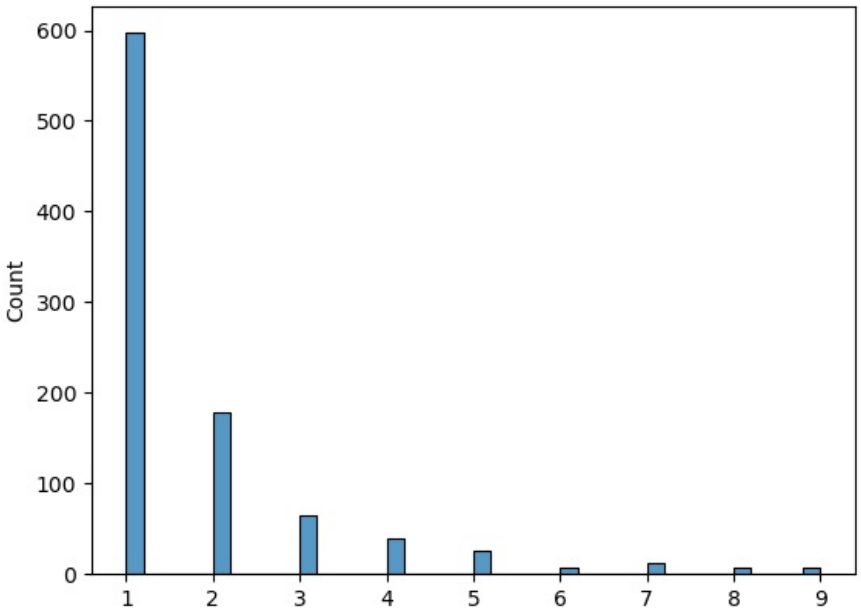
In [165…
```
# Draw out a sample for zipf distribution with distribution parameter 2 with size 2x3:
x=random.zipf(a=2,size=(2,3))

print(x)
```

```
[[1 3 2]
 [5 1 1]]
```

In [ ]:
```
"Visualization of Zipf Distribution"
```

Sample 1000 points but plotting only ones with value < 10 for more meaningful chart.

In [168…
```
x=random.zipf(a=2,size=1000)
sns.histplot(x[x<10])
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: