Random

- Random means something that can not be predicted logically.

Pseudo Random vs Truly Random

- **Pseudo Random** - Function that generates random numbers, which means there is a **logic to generate** random numbers.
- **Truly Random** - Based random data from outside like based on **mouse click, random data from network**. (we dont need it unless its for security)

[ ]:
```
# Install Numpy
!pip install Numpy
```

[3]:
```
import numpy as np
from numpy import random
```

[4]:
```
# Generate a Random Number
a = random.randint(100)

print(a)

x = random.rand() # float between 0 to 1

print(x)
```

```
43
0.6645911483103041
```

[5]:
```
# 5 numbers from the array
x=random.randint(100, size=(5))

print(x)
```

```
[ 2 65 33 45 17]
```

[6]:
```
# 2D Array of 3 rows & 5 Cols
x = random.randint(100, size=(3, 5))

print(x)
```

```
[[13 69 42 88 49]
 [80 94 18 54 94]
 [ 9  7 97 60 11]]
```

[7]:
```
#choice
x = random.choice([3, 5, 7, 9])

print(x)
```

```
5
```

[8]:
```
# 2D Array with follwing values of specific size
x = random.choice([3, 5, 7, 9], size=(3, 5))

print(x)
```

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

Random Data Distribution

[9]:
```
# 1-D array containing 100 values, where each value has to be 1, 2, 17 or 186.
# sum of all probability numbers should be 1
from numpy import random

x = random.choice([1, 2, 17,186], p=[0.1, 0.3, 0.6, 0.0], size=(100))
print("Generated numbers : ")
print(x, "\n")

print("Count of the Occurrence of the Number: ")
# Count the occurrence
from collections import Counter
Counter(x)
```

```
Generated numbers :
[17  2  1 17 17  1 17 17  1 17 17 17 17 17 17  2 17  2 17 17  1 17  2  1
 17 17 17  1  2 17 17 17 17  1 17  1 17 17  2  1 17 17  2  2 17 17 17  1
 17  1 17  2 17 17 17  2 17 17 17 17  2 17 17 17  2  2  2 17  2 17 17 17
  2 17  1 17  2  2  2 17  2 17 17  1 17 17 17 17 17 17  2 17  2  1  2 17 17
 17 17 17 17]

Count of the Occurrence of the Number:
```

[9]:
```
Counter({17: 63, 2: 23, 1: 14})
```

[10]:
```
#### Permute / Shuffle
# Shuffle() - Makes changes to original Array
# Permute() - Re-arranged array (and leaves the original array un-changed).
arr = np.array([1, 2, 3, 4, 5])

random.shuffle(arr)

print(arr)
```

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

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

print(random.permutation(arr))
```

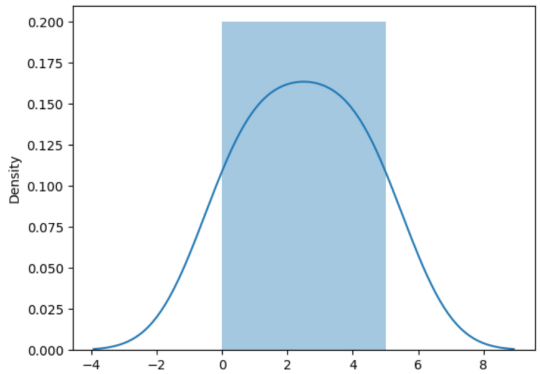
```
[2 3 5 4 1]
```

[ ]:
```
!pip install seaborn
!pip install matplotlib
```

[16]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

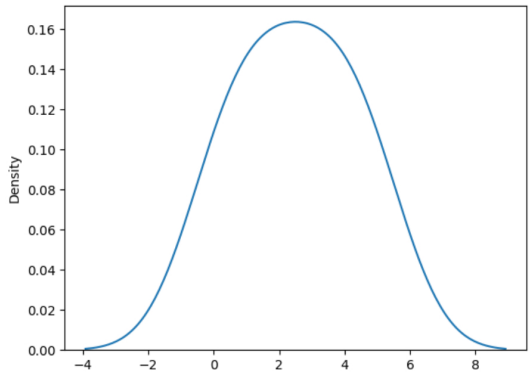
[17]:
```
# A histogram is a graph that displays the frequency of numerical data using rectangles
# with histogram
sns.distplot([0, 1, 2, 3, 4, 5])

plt.show()
```

```
C:\Users\Gowtham\AppData\Local\Temp\ipykernel_15824\3490525831.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, 1, 2, 3, 4, 5])
```



[18]:
```
#without histogram
sns.distplot([0, 1, 2, 3, 4, 5], hist=False)

plt.show()
```

```
C:\Users\Gowtham\AppData\Local\Temp\ipykernel_15824\1074436991.py:2: 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)
```



Normal / Gaussian Distribution

It's characterized by a **bell-shaped curve** where most values cluster around the **mean** (average), and the further away you move from the mean, the fewer values you encounter.

Imagine you're measuring the heights of students in a classroom. If most students are around the average height, with only a few being significantly taller or shorter, their heights would follow a **Gaussian distribution**.

It's a handy mathematical model for describing how data is spread out around a central value.

[27]:
```
#random normal distribution of size 2x3
x = random.normal(size=(2, 3))
print(x)
print("next")
# loc - (Mean) where the peak of the bell exists.
# scale - (Standard Deviation) how flat the graph distribution should be.
x = random.normal(loc=1, scale=2, size=(2, 3))

print(x)
```

```
[[ 0.10208513  1.53001522 -1.55693939]
 [-0.89140773 -0.92905223 -0.20313126]]
next
[[1.01074621 1.4428591  1.299793  ]
 [2.17960629 4.84752901 0.53968406]]
```

- Each value represents a random observation drawn from a normal distribution with a **mean of 1 and a standard deviation of 2**.
- The mean of this distribution is **shifted to 1**, so the values tend to be **closer to 1 on average.**
- The **standard deviation of 2** determines how spread out the values are around the mean. **Larger** standard deviation values result in a **wider spread** of values.

Discrete Distribution vs Continuous Distribution

- **Discrete Distribution** - Represents probabilities of outcomes in a finite or countably infinite set of possible values.
- Examples include rolling a die, counting occurrences of words in a text, or recording the number of defects in a batch of products.
- Possible outcomes: {1, 2, 3, 4, 5, 6} of Rolling a Dice
- **Continuous Distribution** - Represents probabilities of outcomes in a continuous range of values.
- Can take any value within a given interval.
- Examples include measuring heights or weights of individuals, recording reaction times, or estimating temperatures.

Binomial Distribution:

- Discrete Distrubition
- outcome of binary scenarios, e.g. toss of a coin, it will either be head or tails.

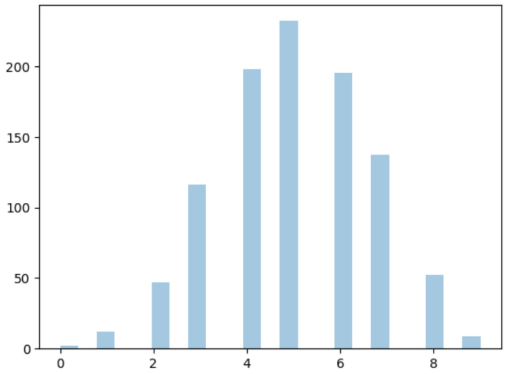
[20]:
```
# n - number of trials.
# p - probability of occurence of each trial (e.g. for toss of a coin 0.5 each).
# size - The shape of the returned array.

x = random.binomial(n=10, p=0.5, size=10)

print(x)
```

```
[6 8 2 8 6 4 2 5 5 3]
```

- In the first experiment, there were 6 successes (heads) out of 10 trials.
- In the second experiment, there were 8 successes (heads) out of 10 trials.
- In the third experiment, there were 2 successes (heads) out of 10 trials.
- In the fourth experiment, there were 8 successes (heads) out of 10 trials.

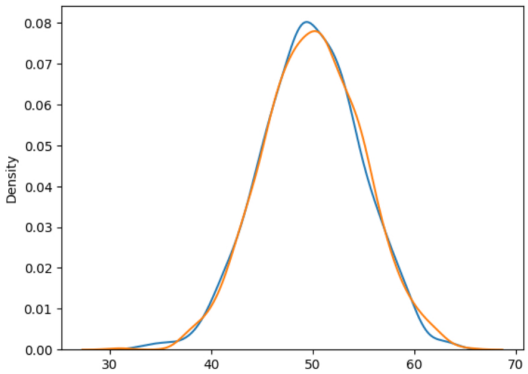
[21]:
```
sns.distplot(random.binomial(n=10, p=0.5, size=1000), hist=True, kde=False)

plt.show()
```

```
C:\Users\Gowtham\AppData\Local\Temp\ipykernel_15824\661670757.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 `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(random.binomial(n=10, p=0.5, size=1000), hist=True, kde=False)
```



[22]:
```
# Normal vs Binomial
# if there are enough data points it will be quite similar to normal distribution with certain loc and scale.
sns.distplot(random.normal(loc=50, scale=5, size=1000), hist=False, label='normal')
sns.distplot(random.binomial(n=100, p=0.5, size=1000), hist=False, label='binomial')

plt.show()
```

```
C:\Users\Gowtham\AppData\Local\Temp\ipykernel_15824\3334510217.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 `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(random.normal(loc=50, scale=5, size=1000), hist=False, label='normal')
C:\Users\Gowtham\AppData\Local\Temp\ipykernel_15824\3334510217.py:4: 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(random.binomial(n=100, p=0.5, size=1000), hist=False, label='binomial')
```



Poisson Distribution

- Estimates **how many times an event can happen** in a specified time ?
- In a call center, the Poisson distribution can model the **number of incoming calls per minute**, where λ (lambda) represents the average rate of calls.

[28]:
```
# Lambda - avg rate, here lets assume 5
x = random.poisson(lam=5, size=5)
```

```
print(x)
```

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

- The generated value stored in variable x represents the **number of incoming calls** that occurred in that minute, based on the **Poisson distribution with an average rate of 5 calls per minute.**

Understand the Output [3 5 1 7 4]:

- **3**, means that in one particular observation or interval, there were 3 occurrences of the event (e.g., 3 phone calls in a minute, 3 accidents in an hour, etc.).
- **5** ,indicates that in **another observation or interval**, there were **5 occurrences of the event.**
- **1**, suggests that in yet **another observation or interval**, there was only **1 occurrence of the event.**
- **7**, implies that in **another observation or interval**, there were **7 occurrences of the event.**
- **4**, shows that in **another observation or interval**, there were **4 occurrences of the event.**

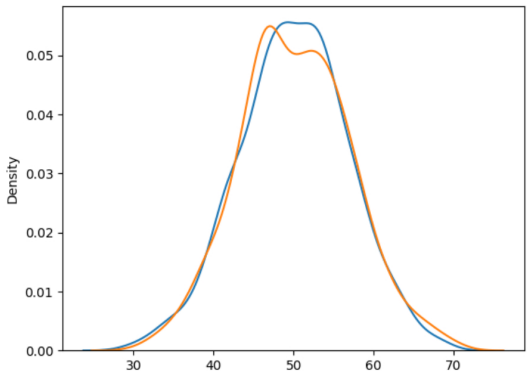
Normal vs Poisson:

- Similar to binomial for a **large enough poisson distribution** it will become similar to **normal distribution with certain std dev and mean.**

[30]:
```
sns.distplot(random.normal(loc=50, scale=7, size=1000), hist=False, label='normal')
sns.distplot(random.poisson(lam=50, size=1000), hist=False, label='poisson')
```

```
C:\Users\Gowtham\AppData\Local\Temp\ipykernel_15824\2196936898.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(random.normal(loc=50, scale=7, size=1000), hist=False, label='normal')
C:\Users\Gowtham\AppData\Local\Temp\ipykernel_15824\2196936898.py:2: 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(random.poisson(lam=50, size=1000), hist=False, label='poisson')
```

[30]:
```
<Axes: ylabel='Density'>
```



Uniform Distribution:

- Describe probability where **every event has equal chances of occuring.**

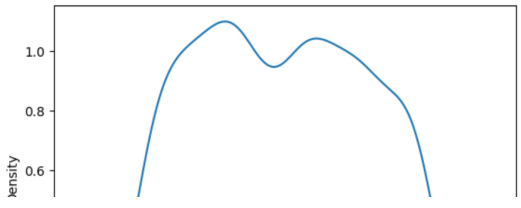
[31]:
```
# a - lower bound - default 0 .0.
# b - upper bound - default 1.0.
# size - The shape of the returned array.
x = random.uniform(size=(2, 3))

print(x)
```

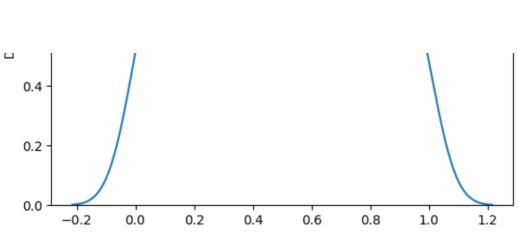
```
[[0.45081039 0.73389115 0.12742365]
 [0.05490672 0.83972275 0.43572517]]
```

- Each value in the output array x represents a **randomly generated number** drawn from a uniform distribution within the **range [0, 1].**

[37]:
```
#generates 1000 random numbers from a uniform distribution between 0 and 1
sns.distplot(random.uniform(size=1000), hist=False)

plt.show()
```

```
C:\Users\Gowtham\AppData\Local\Temp\ipykernel_15824\873973027.py:2: 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(random.uniform(size=1000), hist=False)
```





Logistic Distribution:

- Used to **describe growth.**
- **Logistic & Normal Distribution** are nearly identical.

[38]:
```
# loc - mean, where the peak is. Default 0.
# scale - standard deviation, the flatness of distribution. Default 1.
# size - The shape of the returned array.
```

[40]:
```
x = random.logistic(loc=1, scale=2, size=(2, 3))
print(x)
```

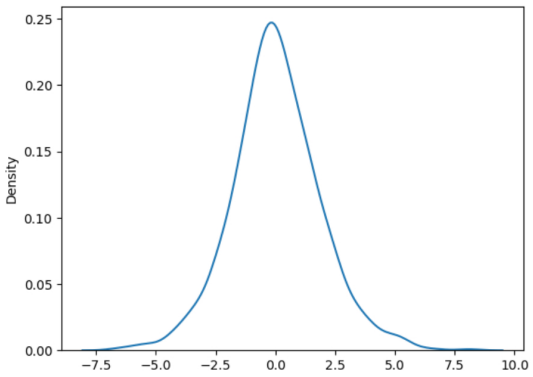
```
[[ 3.05237959  0.79660699  1.73357064]
 [-0.20501045  6.28039953 -0.96282702]]
```

- **Each value** in the output array represents a random observation drawn from the **logistic distribution with a location parameter of 1** and a **scale parameter of 2.**

[41]:
```
sns.distplot(random.logistic(size=1000), hist=False)
```

```
C:\Users\Gowtham\AppData\Local\Temp\ipykernel_15824\2288425481.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(random.logistic(size=1000), hist=False)
```

[41]:
```
<Axes: ylabel='Density'>
```



Multi-Nomial Distribution:

- Outcomes of multi-nomial scenarios unlike binomial where scenarios must be only one of two.
- Blood type of a population - Many types

[46]:
```
# n - number of possible outcomes (e.g. 6 for dice roll).
# pvals - list of probabilties of outcomes (e.g. [1/6, 1/6, 1/6, 1/6, 1/6, 1/6] for dice roll).
# size - The shape of the returned array.
x = random.multinomial(n=6, pvals=[1/6, 1/6, 1/6, 1/6, 1/6, 1/6])
print(x)
```

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

- Outcome 1 Occured 1 time
- Outcome 2 Occured 1 time

Exponential Distribution:

- Describes time till next event **e.g. failure/success etc.**

[45]:
```
# scale - inverse of rate ( see lam in poisson distribution ) defaults to 1.0.
# size - The shape of the returned array.
x = random.exponential(scale=2, size=(2, 3))
print(x)
```

```
[[3.5710446  4.96320306 2.96054734]
 [2.03933716 0.11304665 0.3392997 ]]
```

- Each value in the output array represents a random observation drawn from the exponential distribution with a scale parameter of 2. These values represent durations or intervals between consecutive events, where the likelihood of observing shorter intervals increases as the interval becomes smaller.

Chi Square Distribution:

- Basis to verify the **hypothesis**
- Example - checking if the number of red, green, and blue candies in a bag matches what was expected.

[47]:
```
# df - (degree of freedom).
# size - The shape of the returned array.
x = random.chisquare(df=2, size=(2, 3))
x
```

[47]:
```
array([[0.19795771, 0.1080954 , 1.79874725],
       [2.27961498, 0.09918375, 0.08850529]])
```

Rayleigh Distribution:

- Used in **signal processing** to models signal amplitudes in scenarios like communication noise or radar signal reflections.

[49]:
```
x = random.rayleigh(scale=2, size=(2, 3))
```

```
x
```

[49]:
```
array([[2.99794597, 2.50224527, 3.90538088],
       [1.61604312, 1.55906398, 2.23914444]])
```

Pareto Distribution:

- Pareto's law i.e. 80-20 distribution (20% factors cause 80% outcome).
- Model the distribution of wealth, where a **small percentage of the population holds the majority of the wealth**, such as in the distribution of income among individuals.

[50]:
```
x = random.pareto(a=2, size=(2, 3))
x
```

[50]:
```
array([[0.03139624, 1.91942264, 0.48232991],
       [0.03655805, 0.917247  , 0.29902159]])
```

Zipf Distribution:

- The Zipf distribution is observed in phenomena where the **frequency of occurrence of certain events follows a power-law distribution**, such as the distribution of word frequencies in natural languages like English, where a **few words occur very frequently while most words occur rarely.**

[52]:
```
# a - distribution parameter.
# size - The shape of the returned array.
x = random.zipf(a=2, size=(2, 3))
x
```

[52]:
```
array([[ 1,  2,  1],
       [ 3, 12,  1]])
```

[ ]:
```

```