What is Statistics

Statistics is a branch of mathematics that involves collecting, analysing, interpreting, and presenting data. It provides tools and methods to understand and make sense of large amounts of data and to draw conclusions and make decisions based on the data.

Types of Statistics

- Descriptive
- Inferential

Descriptive Statistics :Descriptive statistics deals with the collection, organization, analysis, interpretation, and presentation of data. It focuses on summarizing and describing the main features of a set of data, without making inferences or predictions about the larger population

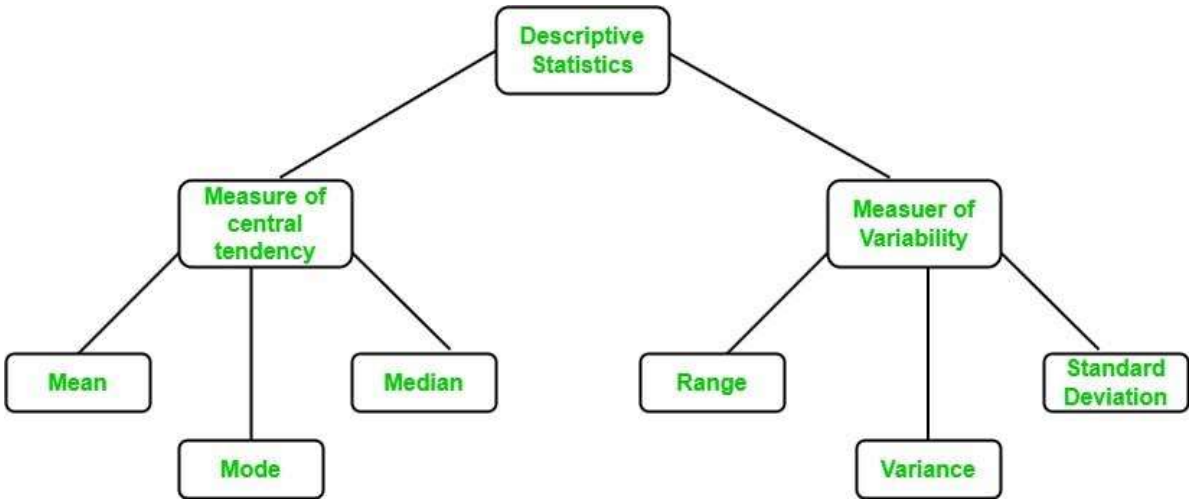
Inferential statistics: Inferential statistics deals with making conclusions and predictions about a population based on a sample. It involves the use of probability theory to estimate the likelihood of certain events occurring, hypothesis testing to determine if a certain claim about a population is supported by the data, and regression analysis to examine the relationships between variables

Population Vs Sample

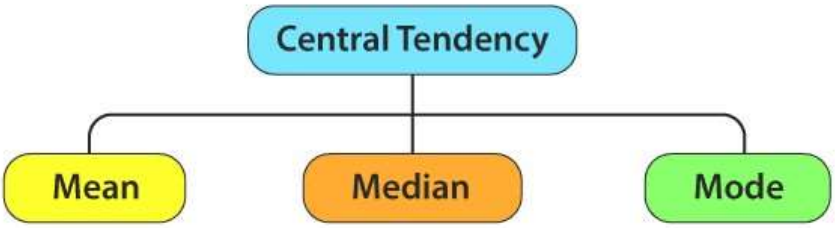
Population :Population refers to the entire group of individuals or objects that we are interested in studying. It is the complete set of observations that we want to make inferences about. For example, the population might be all the students in a particular school or all the cars in a particular city.

Sample : A sample, on the other hand, is a subset of the population. It is a smaller group of individuals or objects that we select from the population to study. Samples are used to estimate characteristics of the population, such as the mean or the proportion with a certain attribute. For example, we might randomly select 100 students.

What is Descriptive Stastistics



- **Measure of Central Tendency** A measure of central tendency is a statistical measure that represents a typical or central value for a dataset. It provides a summary of the data by identifying a single value that is most representative of the dataset as a whole.



- **Mean** : The mean is the sum of all values in the dataset divided by the number of values

Population Mean **Sample Mean**

$$\mu = \frac{\sum x}{N} \qquad \bar{X} = \frac{\sum x}{n}$$

- **Median**: The median is the middle value in the dataset when the data is arranged in order.

$$\text{Median} = \frac{n + 1}{2}$$

- **Mode**: The mode is the value that appears most frequently in the dataset

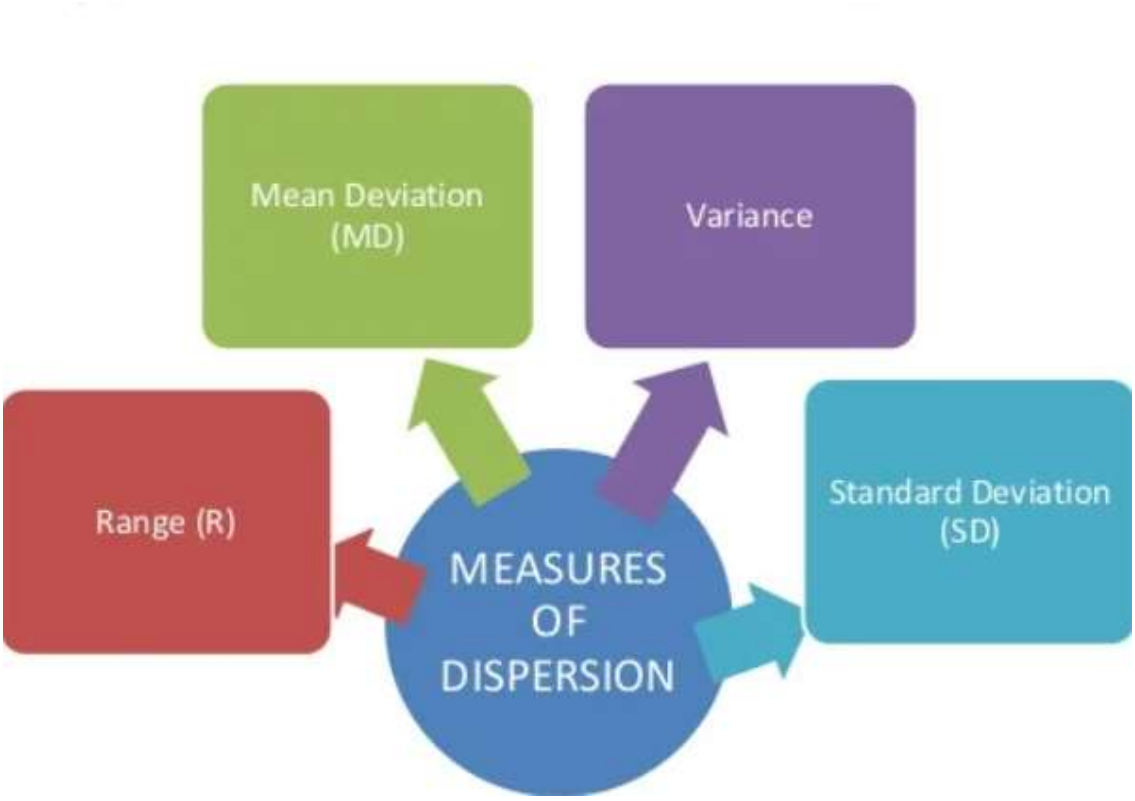
- **Weighted Mean** : : The weighted mean is the sum of the products of each value and its weight, divided by the sum of the weights. It is used to calculate a mean when the values in the dataset have different importance or frequency.

$$\bar{x} = \frac{\sum_{i=1}^{n} (x_i * w_i)}{\sum_{i=1}^{n} w_i}$$

- **Trimmed Mean**: A trimmed mean is calculated by removing a certain percentage of the smallest and largest values from the dataset and then taking the mean of the remaining values. The percentage of values removed is called the trimming percentage.

$$\textbf{Trimmed Mean} = \bar{\chi} = \frac{\sum_{i=p+1}^{n-p} \chi_{(i)}}{n - 2p}$$

- **Measure of Dispersion**: A measure of dispersion is a statistical measure that describes the spread or variability of a dataset. It provides information about how the data is distributed around the central tendency (mean, median or mode) of the dataset.



- **Range**: The range is the difference between the maximum and minimum values in the dataset. It is a simple measure of dispersion that is easy to calculate but can be affected by outliers.

- **Variance**:The variance is the average of the squared differences between each data point and the mean. It measures the average distance of each data point from the mean and is useful in comparing the dispersion of datasets with different means.

$$\sigma^2 = \frac{\sum (x - \mu)^2}{N}$$ **Population Variance**

$$s^2 = \frac{\sum (x - \bar{x})^2}{n - 1}$$ **Sample Variance**

- **Standard Deviation**:The standard deviation is the square root of the variance. It is a widely used measure of dispersion that is useful in describing the shape of a distribution

Population Standard Deviation

$$\sigma = \sqrt{\frac{\sum (x_i - \mu)^2}{N}}$$

Sample Standard Deviation

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{N - 1}}$$

- **Coefficient of Variation (CV)**: The CV is the ratio of the standard deviation to the mean expressed as a percentage. It is used to compare the variability of datasets with different means and is commonly used in fields such as biology, chemistry, and engineering.

The coefficient of variation (CV) is a statistical measure that expresses the amount of variability in a dataset relative to the mean. It is a dimensionless quantity that is expressed as a percentage.

The formula for calculating the coefficient of variation is: CV = (standard deviation / mean) x 100%

$$CV = \frac{\sigma}{\mu}$$

Quantiles and Percentiles

Quantiles are statistical measures used to divide a set of numerical data into equal-sized groups, with each group containing an equal number of observations.

Quantiles are important measures of variability and can be used to: understand distribution of data, summarize and compare different datasets. They can also be used to identify outliers.

There are several types of quantiles used in statistical analysis, including:

- **Quartiles**: Divide the data into four equal parts, Q1 (25th percentile), Q2 (50th percentile or median), and Q3 (75th percentile).
- **Deciles**: Divide the data into ten equal parts, D1 (10th percentile), D2 (20th percentile), ..., D9 (90th percentile).

- **Percentiles**: Divide the data into 100 equal parts, P1 (1st percentile), P2 (2nd percentile), ..., P99 (99th percentile).
- **Quintiles**: Divides the data into 5 equal parts

Percentile

A percentile is a statistical measure that represents the percentage of observations in a dataset that fall below a particular value. For example, the 75th percentile is the value below which 75% of the observations in the dataset fail

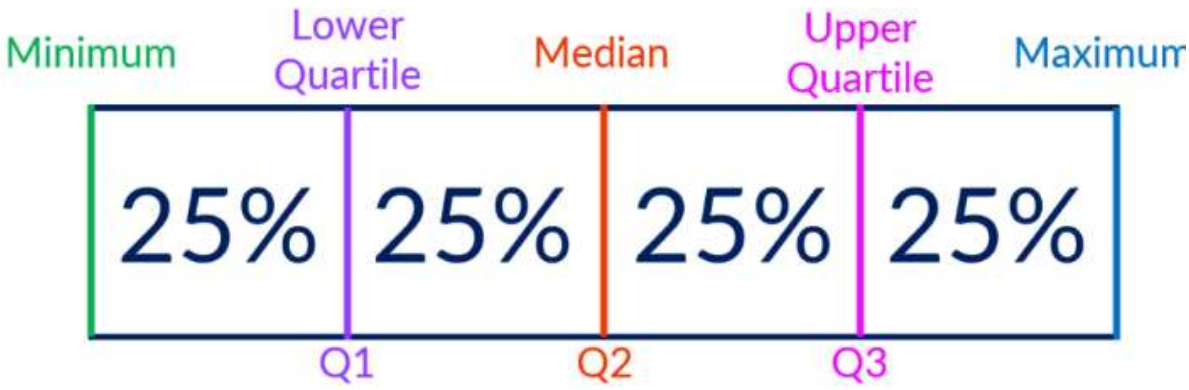
$$P_n = \frac{n^{th}\ percentile \times total\ no\ of\ observations}{100}$$

5 number summary

The five-number summary is a descriptive statistic that provides a summary of a dataset. It consists of five values that divide the dataset into four equal parts, also known as quartiles. The five-number summary includes the following values:

- **1. Minimum value:** The smallest value in the dataset.
- **2.First quartile (Q1)**: The value that separates the lowest 25% of the data from the rest of the dataset.
- **3.Median (Q2):** The value that separates the lowest 50% from the highest 50% of the data.
- **4. Third quartile (Q3):** The value that separates the lowest 75% of the data from the highest 25% of the data.
- **5. Maximum value:** The largest value in the dataset.

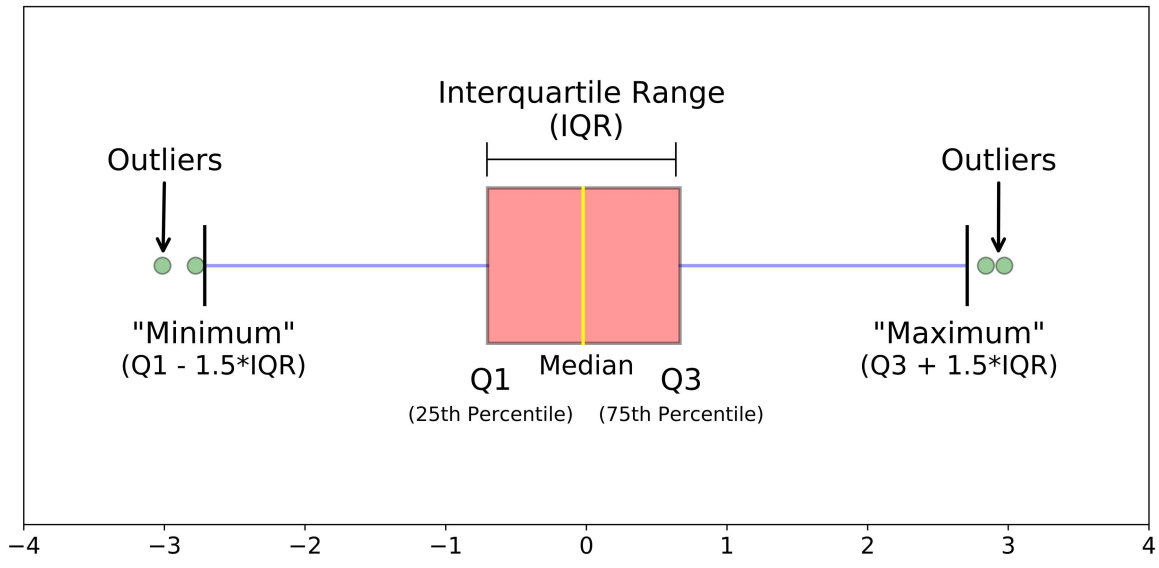
The five-number summary is often represented visually using a box plot, which displays the range of the dataset, the median, and the quartiles. The five-number summary is a useful way to quickly summarize the central tendency, variability, and distribution of a dataset



- **interquartile range**:The interquartile range (IQR) is a measure of variability that is based on the five-number summary of a dataset. Specifically, the IQR is defined as the difference between the third quartile (Q3) and the first quartile (Q1) of a dataset.

1. What is a boxplot

A box plot, also known as a box-and-whisker plot, is a graphical representation of a dataset that shows the distribution of the data. The box plot displays a summary of the data, including the minimum and maximum values, the first quartile (Q1), the median (Q2), and the third quartile (Q3).



Covariance

Covariance measures the direction of **the relationship between two variables**. A positive covariance means that both variables tend to be high or low at the same time. A negative covariance means that when one variable is high, the other tends to be low

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

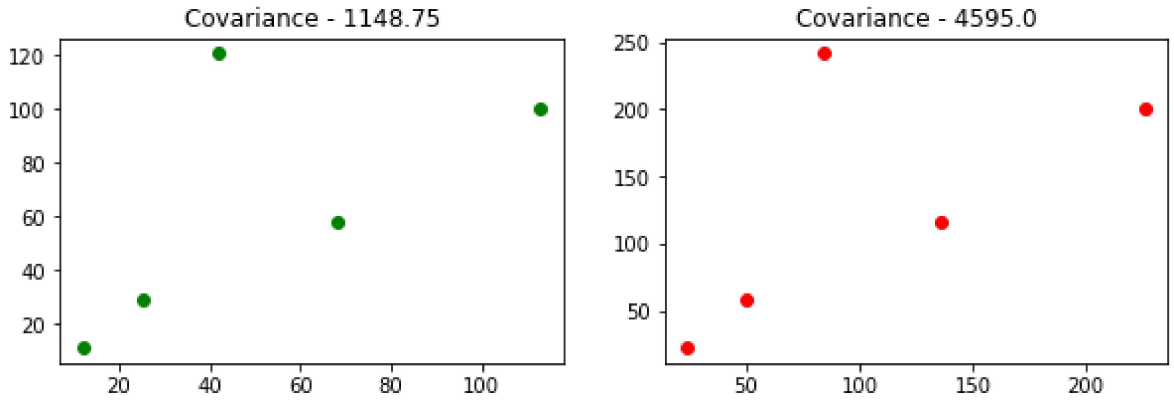
In [2]:
```
df = pd.DataFrame()
```

In [3]:
```
from re import X
x = pd.Series([12,25,68,42,113])
y = pd.Series([11,29,58,121,100])
```

In [4]:
```
df['x'] = x
df['y'] = y
```

In [5]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 3))

# Plot scatterplots on each axes

ax1.scatter(df['x'], df['y'], color ='green')
ax2.scatter(df['x']*2, df['y']*2, color ='red')

ax1.set_title("Covariance - " + str(np.cov(df['x'],df['y'])[0,1]))
ax2.set_title("Covariance - " + str(np.cov(df['x']*2,df['y']*2)[0,1]))
```

Out[5]: Text(0.5, 1.0, 'Covariance - 4595.0')

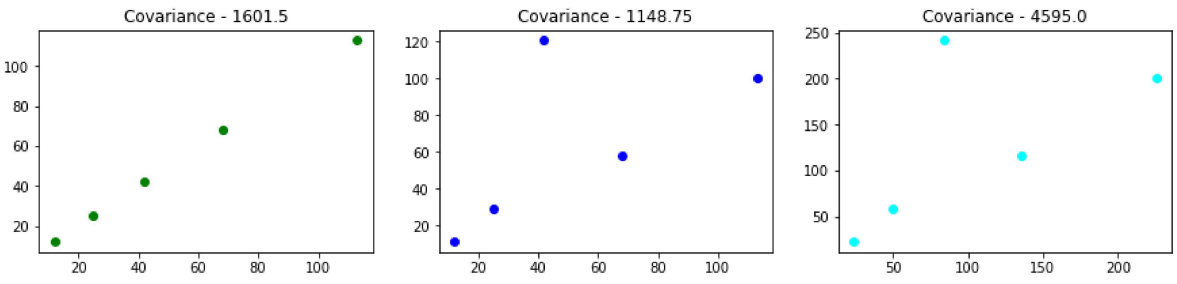


In [6]:
```
# Covariances

print(np.cov(df['x'],df['y'])[0,1])
print(np.cov(df['x']*2,df['y']*2)[0,1])
```

```
1148.75
4595.0
```

In [7]:
```
fig, ax = plt.subplots(1, 3, figsize=(15, 3))

# Plot scatterplots on each axes
ax[0].scatter(df['x'], df['x'],color ='green')
ax[1].scatter(df['x'], df['y'], color ='blue')
ax[2].scatter(df['x']*2, df['y']*2, color ='cyan')

ax[0].set_title("Covariance - " + str(np.cov(df['x'],df['x'])[0,1]))
ax[1].set_title("Covariance - " + str(np.cov(df['x'],df['y'])[0,1]))
ax[2].set_title("Covariance - " + str(np.cov(df['x']*2,df['y']*2)[0,1]))
```

Out[7]: Text(0.5, 1.0, 'Covariance - 4595.0')



Correlation

Correlation is a statistical measure that expresses the extent to which **two variables are linearly related** (meaning they change together at a constant rate). It's a common tool for describing simple relationships without making a statement about cause and effect.

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

Where,

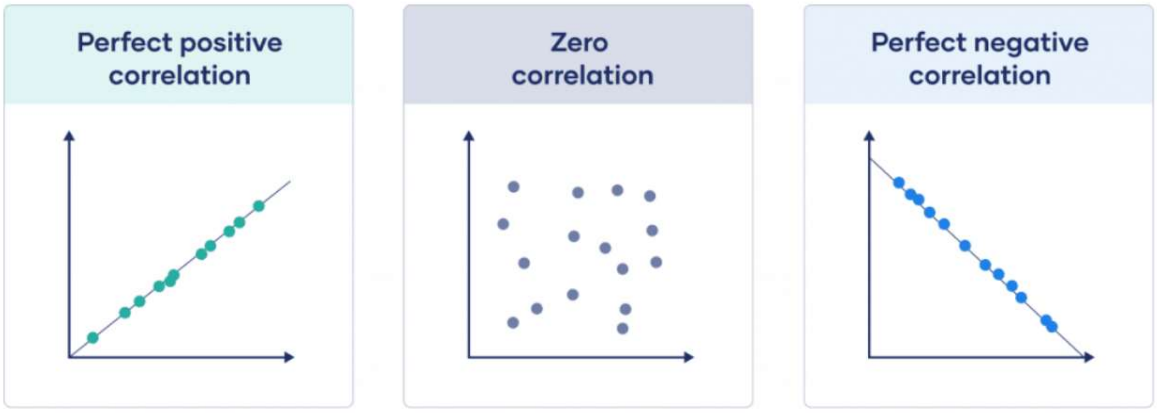
r = Pearson Correlation Coefficient

x_i = x variable samples

y_i = y variable sample

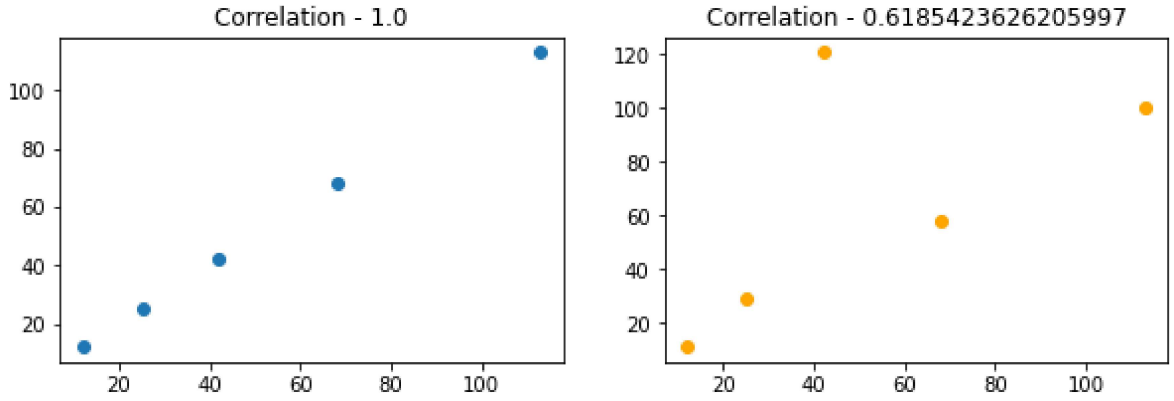
$\bar{x}$ = mean of values in x variable

$\bar{y}$ =mean of values in y variable



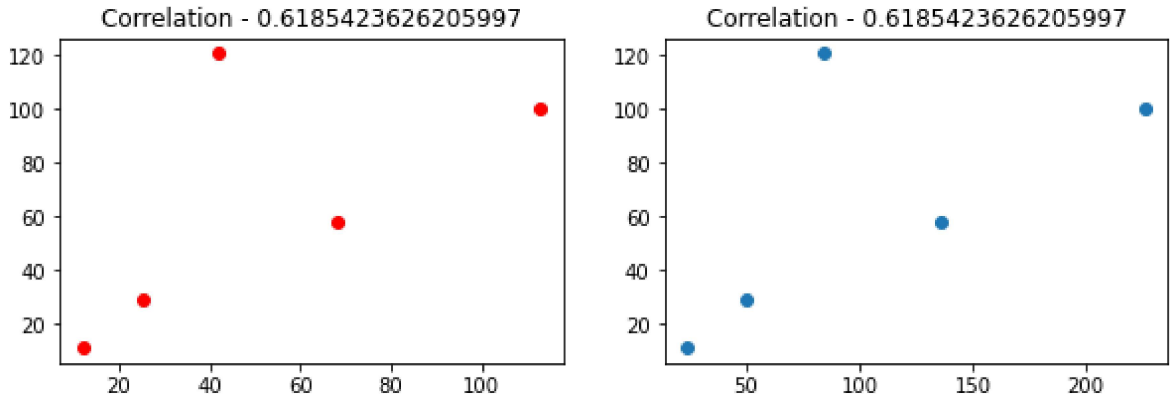
In [8]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 3))

# Plot scatterplots on each axes
ax1.scatter(df['x'], df['x'])
ax2.scatter(df['x'], df['y'],color ='orange')

ax1.set_title("Correlation - " + str(df['x'].corr(df['x'])))
ax2.set_title("Correlation - " + str((df['x']).corr(df['y'])))
```

Out[8]:
```
Text(0.5, 1.0, 'Correlation - 0.6185423626205997')
```



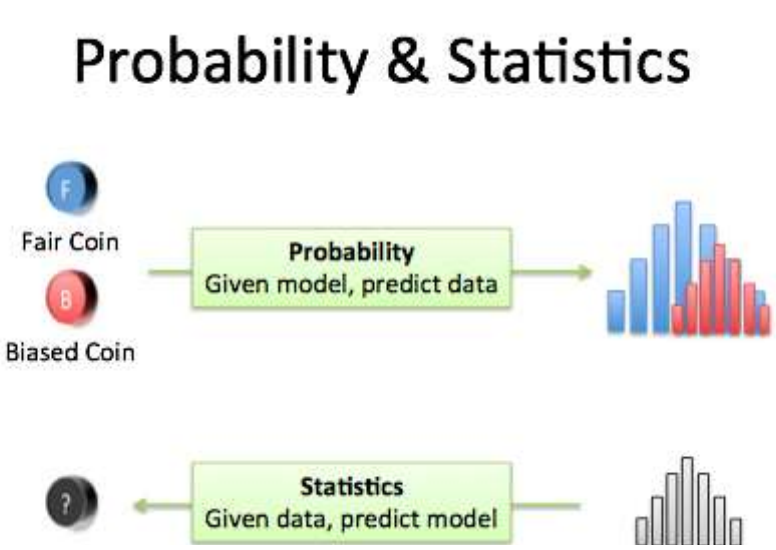
In [9]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 3))

# Plot scatterplots on each axes
ax1.scatter(df['x'], df['y'] , color = 'red')
ax2.scatter(df['x']*2, df['y'])

ax1.set_title("Correlation - " + str(df['x'].corr(df['y'])))
ax2.set_title("Correlation - " + str((df['x']*2).corr(df['y']*2)))
```

Out[9]:
```
Text(0.5, 1.0, 'Correlation - 0.6185423626205997')
```



Probability distribution

A probability distribution is an idealized frequency distribution. A frequency distribution describes a specific sample or dataset. It's the number of times each possible value of a variable occurs in the dataset. The number of times a value occurs in a sample is determined by its probability of occurrence.



Types of Probability Distribution

There are three types of probability distribution which are used for different purposes and various types of the data generation process.

* Probability Mass Function (PMF)
* Probability Density Distribution (PDF)
* Cumulative Probability Function(CDF)

Feature	Probability Density Function (PDF)	Cumulative Distribution Function (CDF)	Probability Mass Function (PMF)
	Describes the likelihood	Describes the probability that	

1. Probability Mass Function (PMF)

PMF stands for Probability Mass Function. It is a mathematical function that describes the probability distribution of categorical a **discrete random variable.**

examples of discrete data

- Nominal (e.g., gender, ethnic background, religious or political affiliation)
- Ordinal (e.g., extent of agreement, school letter grades)
- Quantitative variables with relatively few values (e.g., number of times married)

The PMF of a discrete random variable assigns a probability to each possible value of the random variable.

The probabilities assigned by the PMF must satisfy two conditions:

- a.The probability assigned to each value must be non-negative (i.e., greater than or equal to zero).
- b. The sum of the probabilities assigned to all possible values must equal 1.

In [10]:
```
import pandas as pd
import random
```

In [11]:
```
l = []
for i in range(10000):
    l.append(random.randint(1,6))
# Here we take values from 1 - 6
```

In [12]:
```
len(l)
```

Out[12]:
```
10000
```

In [13]:
```
l[:5]
```

Out[13]:
```
[2, 4, 5, 1, 6]
```

In [14]:
```
pd.Series(l).value_counts()
```

Out[14]:
```
2    1728
5    1682
3    1677
4    1670
6    1657
1    1586
dtype: int64
```

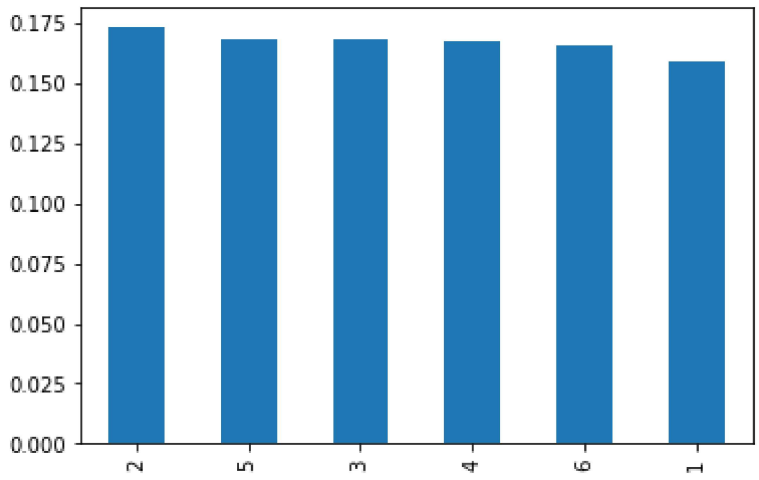
In [15]:
```
pd.Series(l).value_counts()/ pd.Series(l).value_counts().sum()
```

Out[15]:
```
2    0.1728
5    0.1682
3    0.1677
4    0.1670
6    0.1657
1    0.1586
dtype: float64
```

In [16]:
```
s = pd.Series(l).value_counts()/ pd.Series(l).value_counts().sum()
```

In [17]:
```
s.plot(kind ='bar')
```

Out[17]:
```
<AxesSubplot:>
```



In [18]:
```
# Now  rollig two dices and Add the both a and b

l = []
for i in range(10000):
    a = (random.randint(1,6))
    b = (random.randint(1,6))

    l.append(a+b)
```

In [19]:
```
len(l)
```

Out[19]:
```
10000
```

In [20]:
```
l[:5]
```

Out[20]:
```
[6, 11, 10, 10, 3]
```

In [21]:
```
(pd.Series(l).value_counts())
```

Out[21]:
```
7     1724
8     1396
6     1369
9     1098
5     1089
10     836
4      820
11     545
3      537
2      295
12     291
dtype: int64
```

In [22]:
```
(pd.Series(l).value_counts()/pd.Series(l).value_counts().sum())
```

Out[22]:
```
7     0.1724
8     0.1396
6     0.1369
9     0.1098
5     0.1089
10    0.0836
4     0.0820
11    0.0545
3     0.0537
2     0.0295
12    0.0291
dtype: float64
```

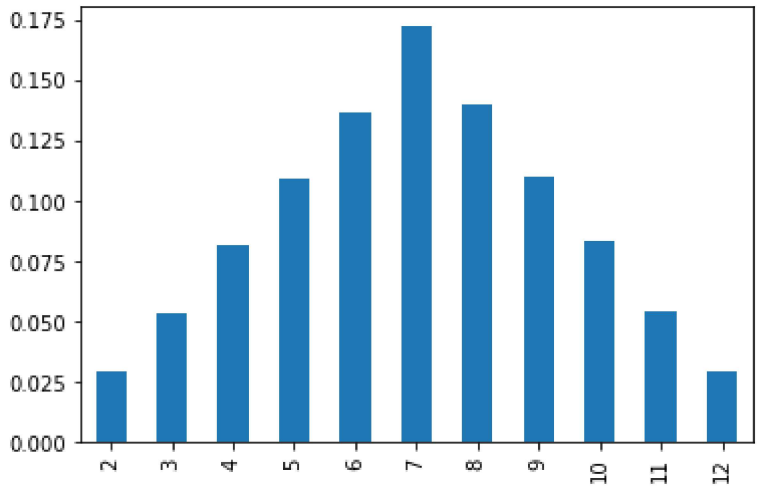
In [23]:
```
(pd.Series(l).value_counts()/pd.Series(l).value_counts().sum()).sort_index()
```

Out[23]:
```
2     0.0295
3     0.0537
4     0.0820
5     0.1089
6     0.1369
7     0.1724
8     0.1396
9     0.1098
10    0.0836
11    0.0545
12    0.0291
dtype: float64
```

In [24]:
```
s = (pd.Series(l).value_counts()/pd.Series(l).value_counts().sum()).sort_index
```

In [25]:
```
s.plot(kind='bar')
```

Out[25]: <AxesSubplot:>



Cumulative Distribution Function(CDF) of PMF

The cumulative distribution function (CDF) F(x) describes the probability that a random variable X with a given probability distribution will be found at a value less than or equal to x

F(x) = P(X <= x)

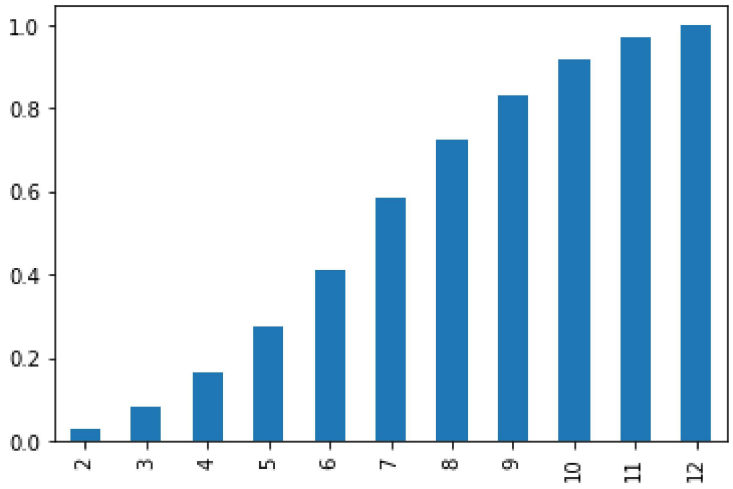
In [26]:
```
import numpy as np
```

In [27]:
```
np.cumsum(s) # Total = 1
```

Out[27]:
```
2     0.0295
3     0.0832
4     0.1652
5     0.2741
6     0.4110
7     0.5834
8     0.7230
9     0.8328
10    0.9164
11    0.9709
12    1.0000
dtype: float64
```

In [28]:
```
np.cumsum(s).plot(kind ='bar')
```

Out[28]: <AxesSubplot:>



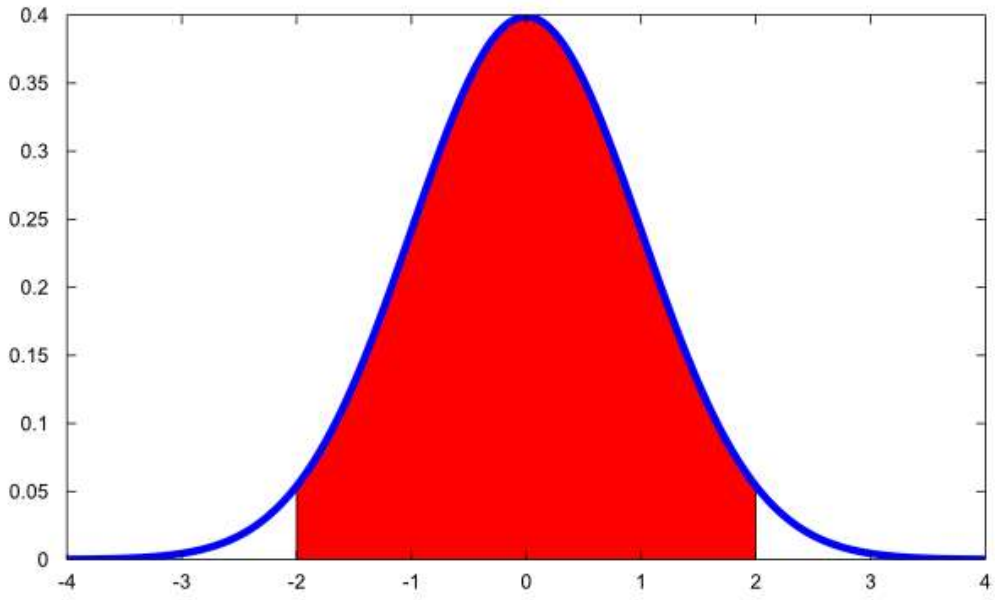
What is the difference between PMF AND CDF

PMF : Gives the probability of the **particular point (X)**

CDF : Gives the probability of the **All the points up to (x)**

Probability Density Function (PDF)

PDF stands for Probability Density Function. It is a mathematical function that describes the probability distribution of a continuous random variable.



In PMF , y axis is known as 'Probability' Where as in PDF , y axis is known as ' Probability density' , because probability will be Zero in PDF And Area under the curve is probabilty for PDF

- How graph is calculated

DENSITY Estimation

Density estimation is a statistical technique used to estimate the probability density function (PDF) of a random variable based on a set of observations or data. In simpler terms, it involves estimating the underlying distribution of a set of data points

- here are various methods for density estimation, including **parametric** and **non□parametric** approaches.
- Parametric methods assume that the data follows a specific probability distribution (such as a normal distribution),
- while non□parametric methods do not make any assumptions about the distribution and instead estimate it directly from the data.

Commonly used techniques for density estimation include kernel density estimation (KDE), histogram estimation, and Gaussian mixture models (GMMs). The choice of method depends on the specific characteristics of the data and the intended use of the density estimate.

Parametric density estimation

Parametric density estimation is a method of estimating the probability density function (PDF) of a random variable by assuming that the underlying distribution belongs to a specific parametric family of probability distributions, such as the normal, exponential, or Poisson distributions.

In [29]:
```
import matplotlib.pyplot as plt
import numpy as np
from numpy.random import normal

sample = normal(loc=50,scale=5,size=1000)
```

In [30]:
```
sample
```

Out[30]:
```
array([61.23026396, 48.88218776, 53.37395358, 48.23859345, 38.82775714,
       50.71547702, 43.55170549, 59.59037734, 44.07646131, 51.2291823 ,
       49.50129918, 41.87482911, 42.89516605, 49.83430018, 50.42984119,
       48.02929317, 43.66796728, 47.26094984, 48.14397298, 45.30814864,
       51.62220736, 44.8594848 , 41.84895438, 47.33320696, 53.85271785,
       41.98071327, 51.7260448 , 52.96282035, 50.05249076, 50.19805956,
       49.07434682, 53.07117544, 53.86269107, 56.04279745, 40.90454341,
       51.77673487, 43.2966207 , 51.75137924, 40.90094009, 54.19609498,
       55.26453117, 55.4609019 , 54.4989116 , 54.82385172, 43.10395288,
       41.8218877 , 46.58957669, 51.41111365, 52.52098661, 49.19831397,
       48.99275682, 62.61191008, 43.65576003, 55.21987601, 44.79056897,
       51.78870203, 46.37564734, 51.40565093, 45.28410534, 45.75391457,
       49.09427446, 53.86831982, 52.71599378, 44.36834434, 50.74665809,
       53.72739067, 49.49842696, 47.32866333, 43.56592226, 46.80359987,
       49.25437035, 56.58358384, 60.41122227, 48.2555872 , 52.34446159,
       58.39865208, 49.96656899, 60.76236236, 47.51901699, 44.65567849,
       49.36043701, 43.655939  , 56.18308073, 47.10868464, 47.8524103 ,
       44.90322859, 48.05274478, 46.16966608, 49.40622809, 39.38872195,
       53.57422306, 50.30220825, 50.92156859, 45.35959403, 44.0693628 ,
```

In [31]:
```
sample.mean() # average
```

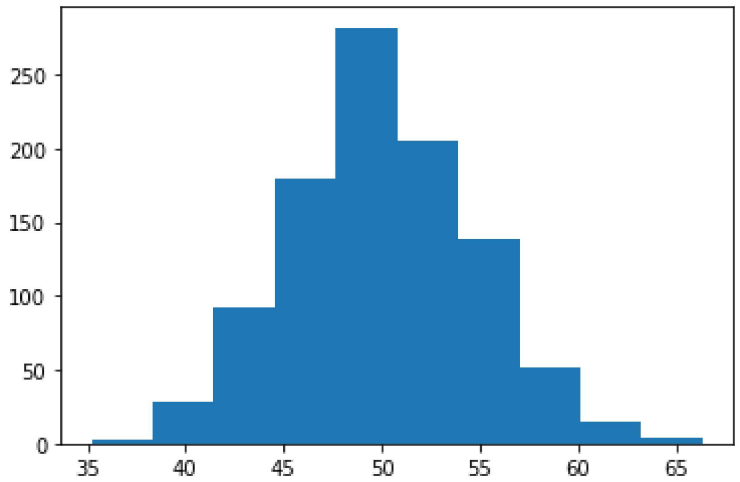
Out[31]:
```
50.06793537132727
```

In [32]:
```
sample.std() # scale -- standard deviation
```

Out[32]:
```
4.70306827374624
```

In [33]:
```
# plot histogram to understand the distribution of data

plt.hist(sample,bins=10)
```

Out[33]:
```
(array([  3.,  28.,  92., 180., 282., 205., 139.,  52.,  15.,   4.]),
 array([35.20388838, 38.31826811, 41.43264784, 44.54702757, 47.6614073 ,
        50.77578703, 53.89016676, 57.0045465 , 60.11892623, 63.23330596,
        66.34768569]),
 <BarContainer object of 10 artists>)
```



In [34]:
```
# calculate sample mean and sample std dev

sample_mean = sample.mean()
sample_std = sample.std()
```

In [35]:
```
# fit the distribution with the above parameters

from scipy.stats import norm

dist = norm(sample_mean, sample_std)
```

In [36]:
```
values = np.linspace(sample.min(),sample.max(),100)
```

In [37]:
```
sample.max()
```

Out[37]:
```
66.34768568938694
```

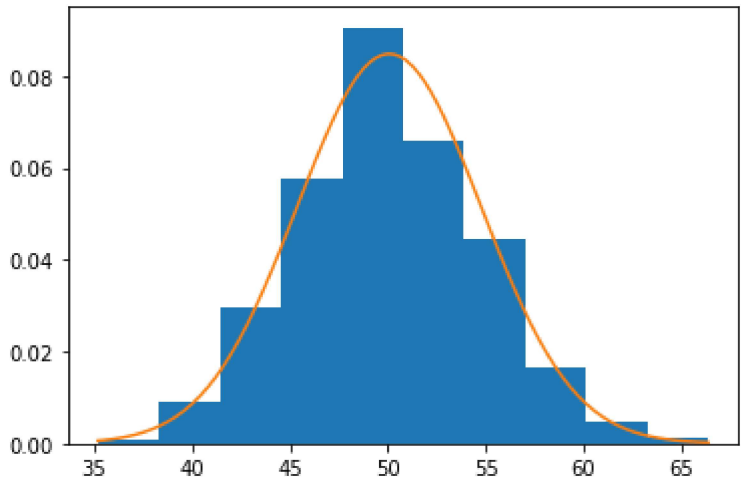
In [38]:
```
sample.min()
```

Out[38]:
```
35.203888376819734
```

In [39]:
```
probability_density = [dist.pdf(value) for value in values]
```

In [40]:
```
# plot the histogram and pdf
plt.hist(sample,bins=10,density=True)
plt.plot(values,probability_density)
```

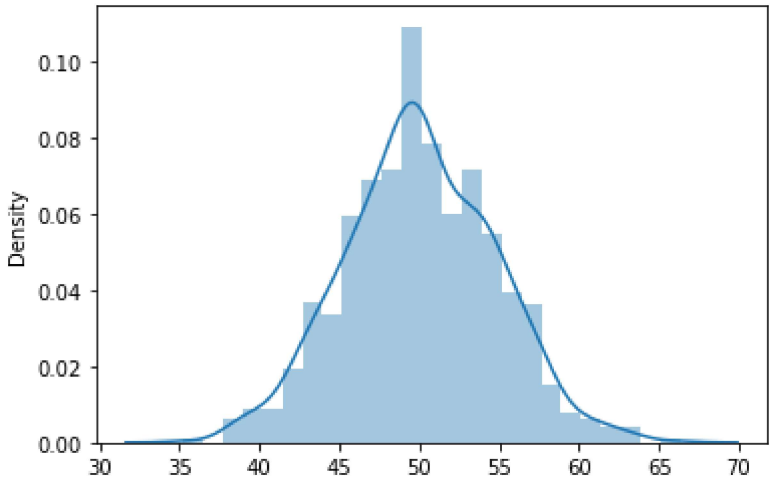
Out[40]:
```
[<matplotlib.lines.Line2D at 0x1e284edafa0>]
```



In [41]:
```
import seaborn as sns

sns.distplot(sample)
```

```
C:\Users\user\anaconda3\lib\site-packages\seaborn\distributions.py:2619: Futu
reWarning: `distplot` is a deprecated function and will be removed in a futur
e version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for his
tograms).
  warnings.warn(msg, FutureWarning)
```

Out[41]: <AxesSubplot:ylabel='Density'>



Non-Parametric Density Estimation (KDE)

But sometimes the distribution is not clear or it's not one of the famous distributions

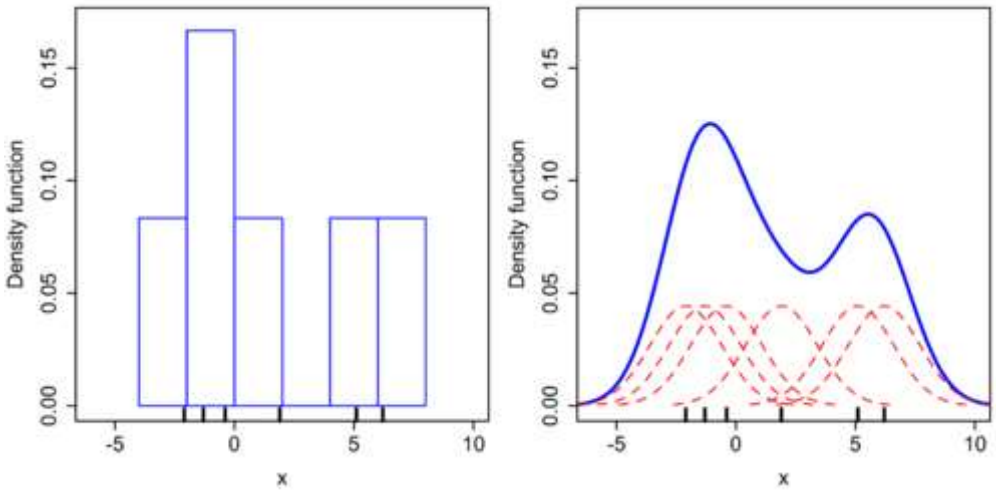
Non-parametric density estimation is a statistical technique used to estimate the probability density function of a random variable without making any assumptions about the underlying distribution. It is also referred to as non-parametric density estimation because it does not require the use of a predefined probability distribution function, as opposed to parametric methods such as the Gaussian distribution.

The non-parametric density estimation technique involves constructing an estimate of the probability density function using the available data. This is typically done by creating a **kernel density estimate**

Non-parametric density estimation has several advantages over parametric density estimation. One of the main **advantages** is that it **does not require the assumption of a specific distribution**, which allows for more flexible and accurate estimation in situations where the underlying distribution is unknown or complex. However, non-parametric density estimation can be **computationally intensive and may require more data to achieve accurate estimates** compared to parametric methods

Kernel Density Estimate (KDE)

The KDE technique involves using a kernel function to smooth out the data and create a continuous estimate of the underlying density function.



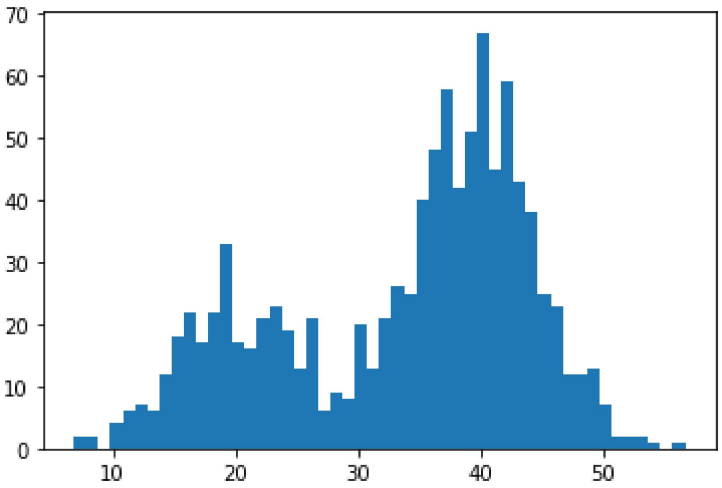
In [42]:
```
# generate a sample
sample1 = normal(loc=20, scale=5, size=300)
sample2 = normal(loc=40, scale=5, size=700)

sample = np.hstack((sample1, sample2))
```

In [43]:
```
sample
```

Out[43]:
```
array([18.74994648, 17.80224222, 18.29800915, 23.42562987, 16.09952685,
       22.33944493, 16.38786433, 21.97125304, 16.72775499, 28.57689684,
       15.66979052, 24.45723695, 15.62960384, 25.88337228, 12.74822236,
       16.56971432, 16.44850038, 26.15707769, 31.25347597, 22.24425542,
       25.87033805, 18.78893963, 22.75752921, 17.78835841, 25.54810048,
       14.25471995, 20.92610389, 30.70721496, 18.97701651, 23.12808417,
       18.66370054, 24.91880322, 23.91945716, 15.68686845, 18.22817138,
       19.73913018, 19.10267034, 16.05492944, 24.91690763, 25.9298788 ,
       15.92662457, 18.2912016 , 27.17933425, 15.83872577, 17.44947828,
       30.30097624, 11.15795971, 30.07166078, 13.42050915, 25.77306299,
       19.56052624, 13.86117965, 18.0353114 , 27.92794143, 22.59782308,
       27.39357026, 19.55656425, 17.40954535, 14.607757  , 19.57608052,
       22.65509317, 14.55056601, 25.84291109, 24.46702315, 20.08455092,
       16.51443889, 16.36934891, 25.43847926, 18.18766906, 19.53279383,
       17.76823545, 15.60928448, 18.67122921, 19.38772615, 18.22732861,
       22.57954358, 20.76040719, 19.90234814, 22.31603646, 17.46979068,
       29.8004973 , 22.98233236, 26.19129825, 19.25758085, 24.73772355,
       17.16142858, 18.75472316, 24.7428767 , 11.92574223, 19.51316978,
       10.90700128, 19.3017756 , 18.4694779 , 20.55078499, 15.68565161,
```

In [44]:
```
# plot histogram bins=50

plt.hist(sample,bins=50)
```

Out[44]:
```
(array([ 2.,  2.,  0.,  4.,  6.,  7.,  6., 12., 18., 22., 17., 22., 33.,
        17., 16., 21., 23., 19., 13., 21.,  6.,  9.,  8., 20., 13., 21.,
        26., 25., 40., 48., 58., 42., 51., 67., 45., 59., 43., 38., 25.,
        23., 12., 12., 13.,  7.,  2.,  2.,  2.,  1.,  0.,  1.]),
 array([ 6.80805431,  7.80548117,  8.80290803,  9.80033489, 10.79776175,
        11.7951886 , 12.79261546, 13.79004232, 14.78746918, 15.78489604,
        16.7823229 , 17.77974976, 18.77717662, 19.77460347, 20.77203033,
        21.76945719, 22.76688405, 23.76431091, 24.76173777, 25.75916463,
        26.75659148, 27.75401834, 28.7514452 , 29.74887206, 30.74629892,
        31.74372578, 32.74115264, 33.7385795 , 34.73600635, 35.73343321,
        36.73086007, 37.72828693, 38.72571379, 39.72314065, 40.72056751,
        41.71799437, 42.71542122, 43.71284808, 44.71027494, 45.7077018 ,
        46.70512866, 47.70255552, 48.69998238, 49.69740924, 50.69483609,
        51.69226295, 52.68968981, 53.68711667, 54.68454353, 55.68197039,
        56.67939725]),
 <BarContainer object of 50 artists>)
```



In [45]:
```
from sklearn.neighbors import KernelDensity

model = KernelDensity(bandwidth=5, kernel='gaussian')

# convert data to a 2D array
sample = sample.reshape((len(sample), 1))

model.fit(sample)
```

Out[45]:
```
KernelDensity(bandwidth=5)
```

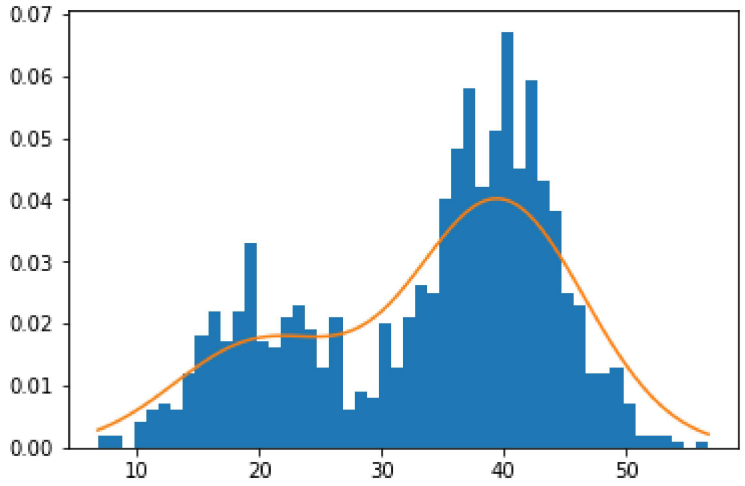
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [46]:
```
values = np.linspace(sample.min(),sample.max(),100)
values = values.reshape((len(values), 1)) # Converting into 2D
```

In [47]:
```
probability_density = model.score_samples(values)
probability_density = np.exp(probability_density)
```

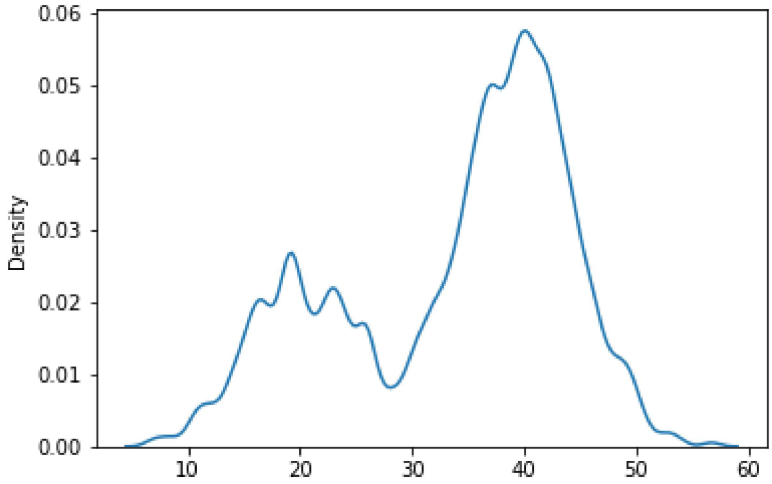
score_samples(values) returns the log-density estimate of the input samples values. This is because the score_samples() method of the KernelDensity class returns the logarithm of the probability density estimate rather than the actual probability density estimate.

In [48]:
```
plt.hist(sample, bins=50, density=True)
plt.plot(values[:], probability_density)
plt.show()
```



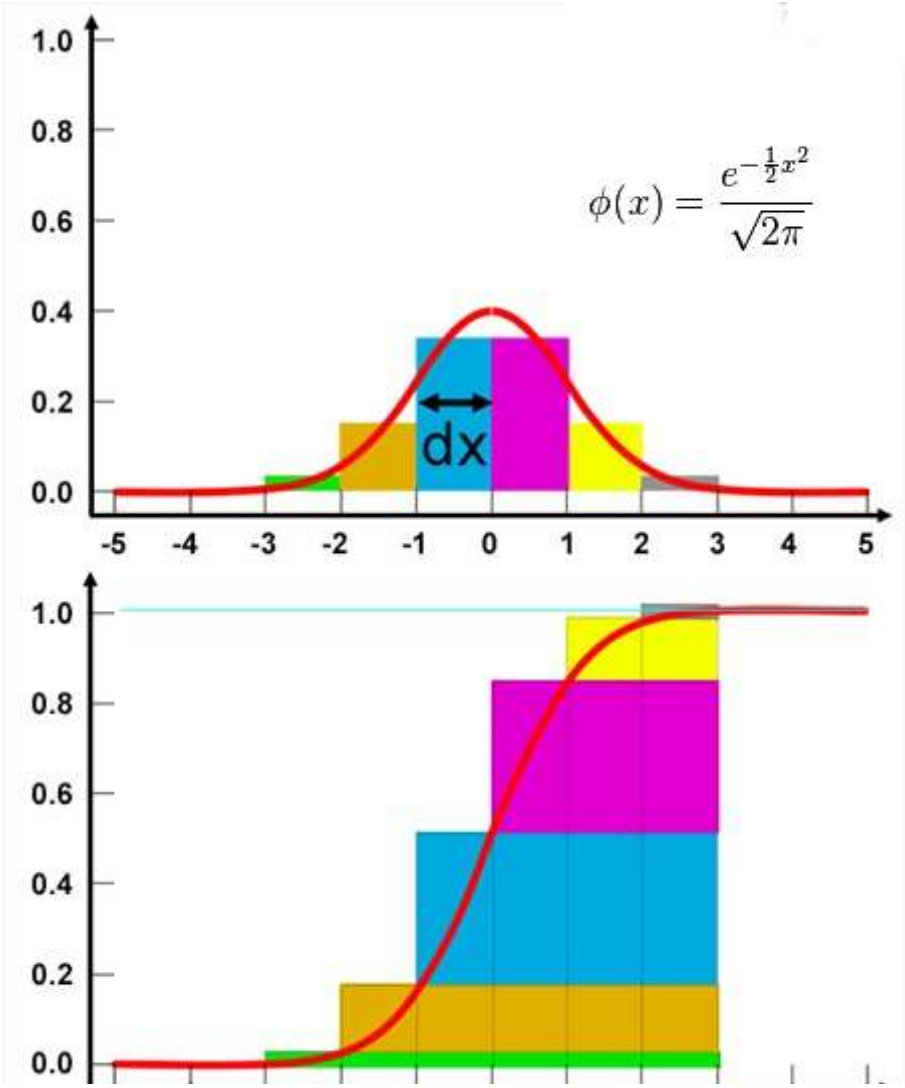
In [49]:
```
sns.kdeplot(sample.reshape(1000),bw_adjust=0.3)
```

Out[49]:
```
<AxesSubplot:ylabel='Density'>
```



Probability Density Function and Cumulative Probability

Function



How to use PDF in Data Science

In [50]:
```
import seaborn as sn
```

In [51]:
```
df = sns.load_dataset('iris')
```

In [52]:
```
df.head()
```

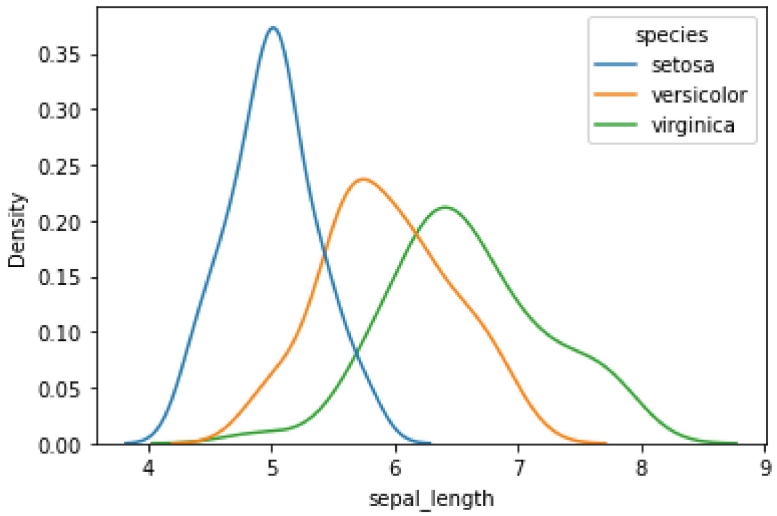
Out[52]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

What are the most important features for analysis?

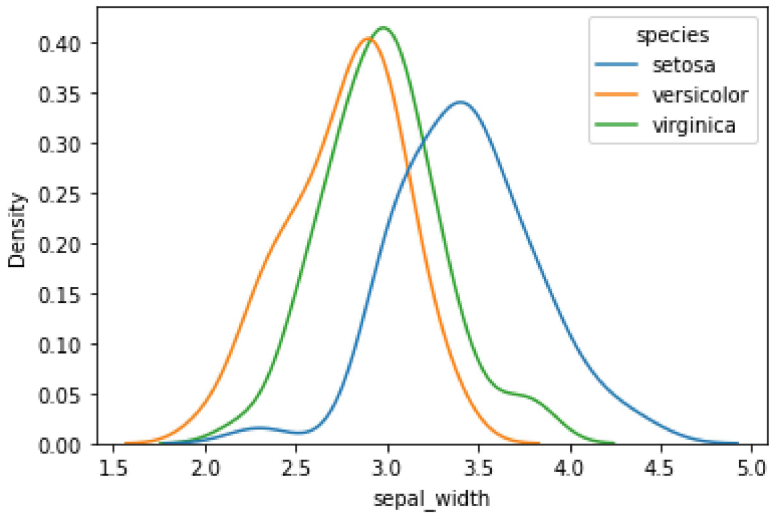
In [53]:
```
sns.kdeplot(df['sepal_length'],hue= df['species'])
```

Out[53]: <AxesSubplot:xlabel='sepal_length', ylabel='Density'>



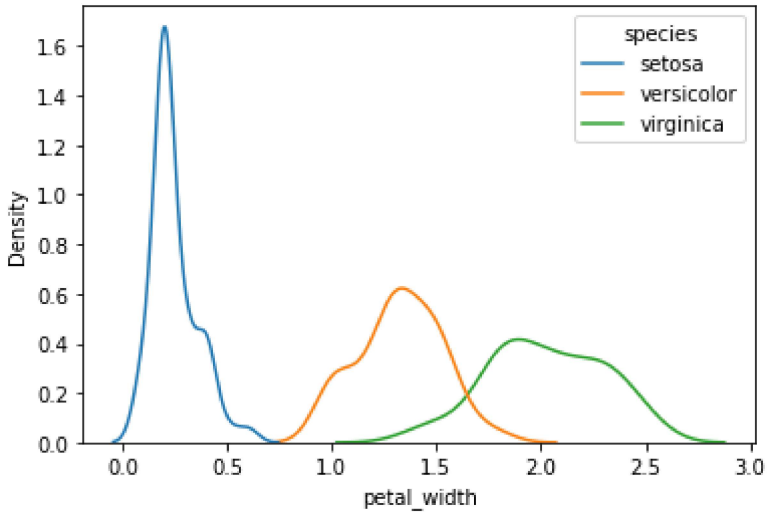
In [54]:
```
sns.kdeplot(df['sepal_width'],hue= df['species'])
```

Out[54]: <AxesSubplot:xlabel='sepal_width', ylabel='Density'>



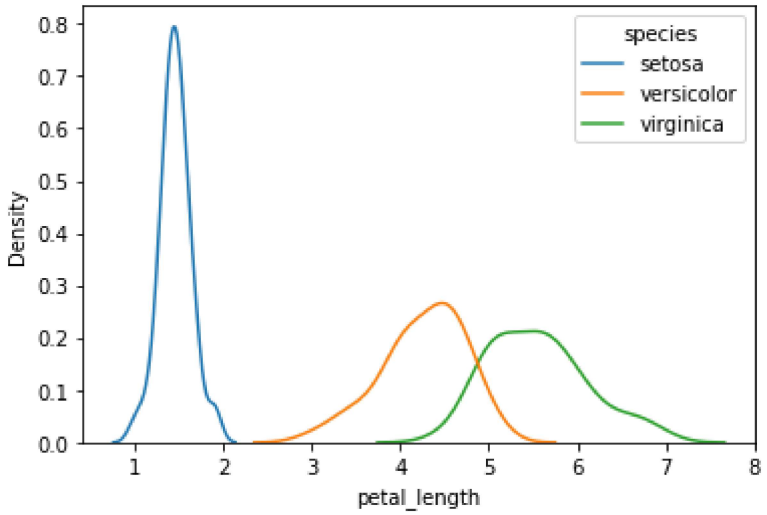
In [55]:
```
sns.kdeplot(df['petal_width'],hue= df['species'])
```

Out[55]: <AxesSubplot:xlabel='petal_width', ylabel='Density'>



In [56]:
```
sns.kdeplot(df['petal_length'],hue= df['species'])
```

Out[56]: <AxesSubplot:xlabel='petal_length', ylabel='Density'>



We can tell from the graph that petal length and petal width are crucial for analysis.

In [57]:
```
titanic = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datas
```

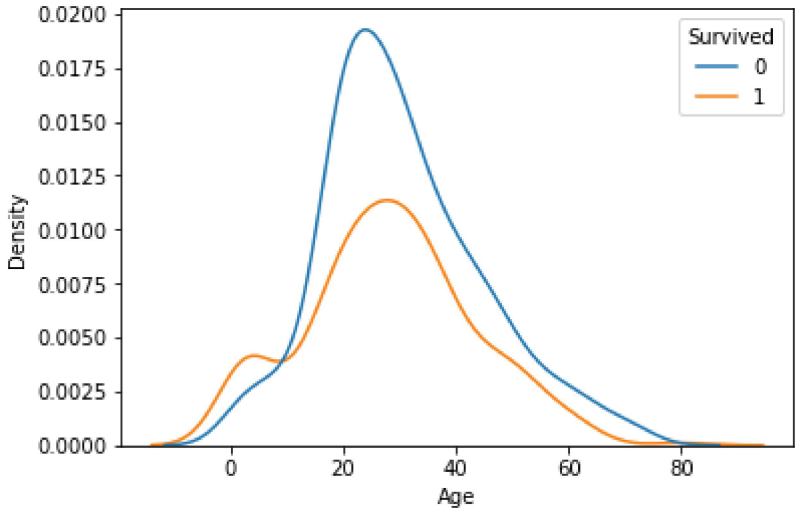
In [58]:
```
titanic.head()
```

Out[58]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Ca
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	I
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	I
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	I

In [59]:
```
#  By using PDF (Probability density distribution) we can analyse the data

sns.kdeplot(data=titanic,x='Age',hue='Survived')

# We can examine the likelihood of survival for children aged 0 to 8 here.

# PDF gives , at particular points Probability
```

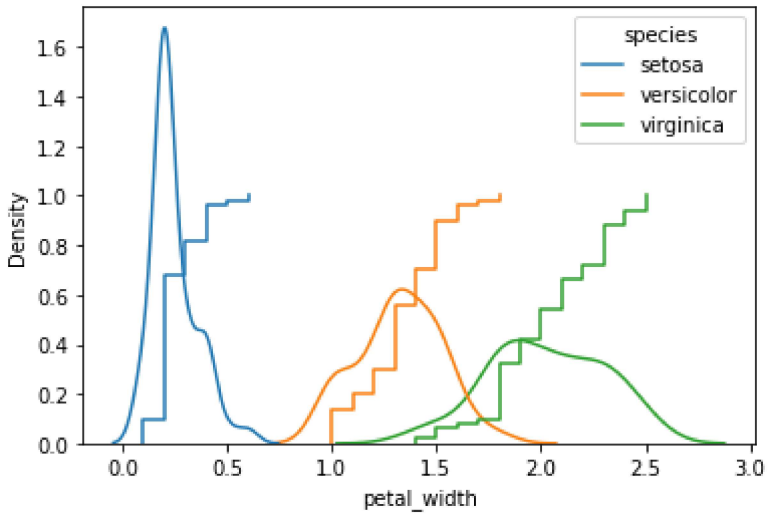
Out[59]:
```
<AxesSubplot:xlabel='Age', ylabel='Density'>
```

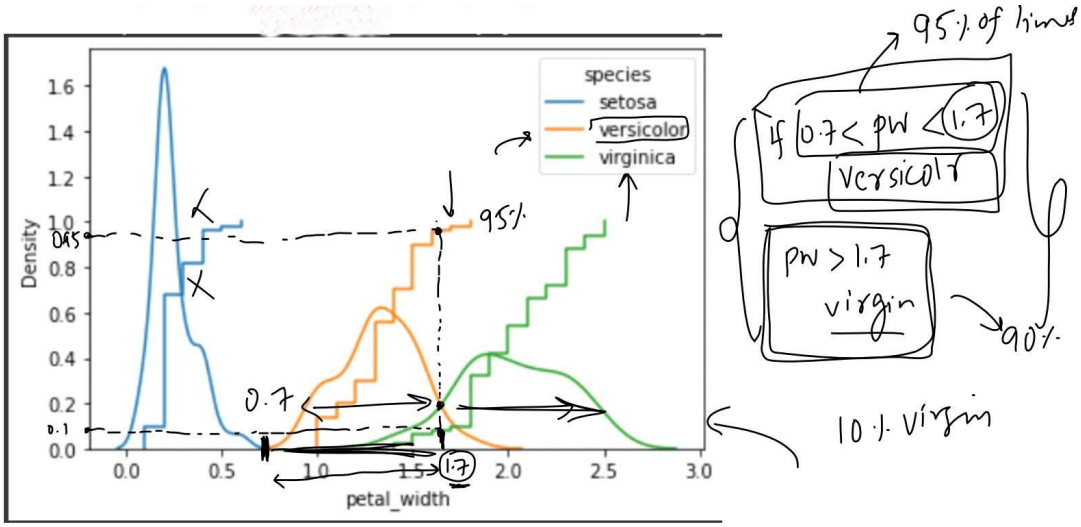


CDF (Cumulative distribution function)

In [60]:
```
sns.kdeplot(df['petal_width'],hue=df['species'])

# ecdfPlot

sns.ecdfplot(data=df,x='petal_width',hue='species')
```

Out[60]:
```
<AxesSubplot:xlabel='petal_width', ylabel='Density'>
```



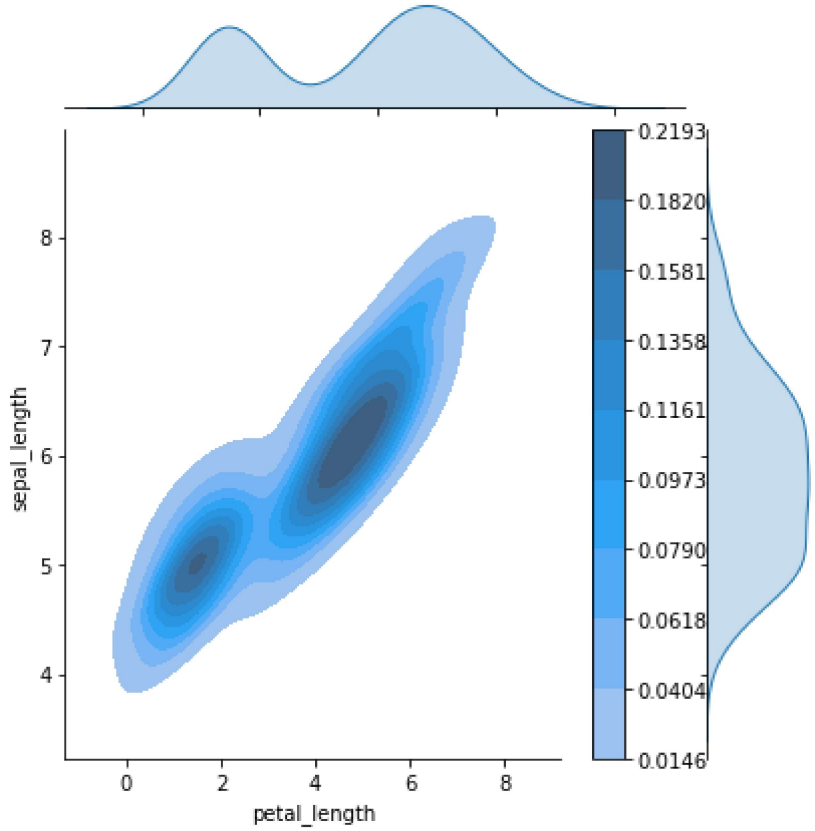


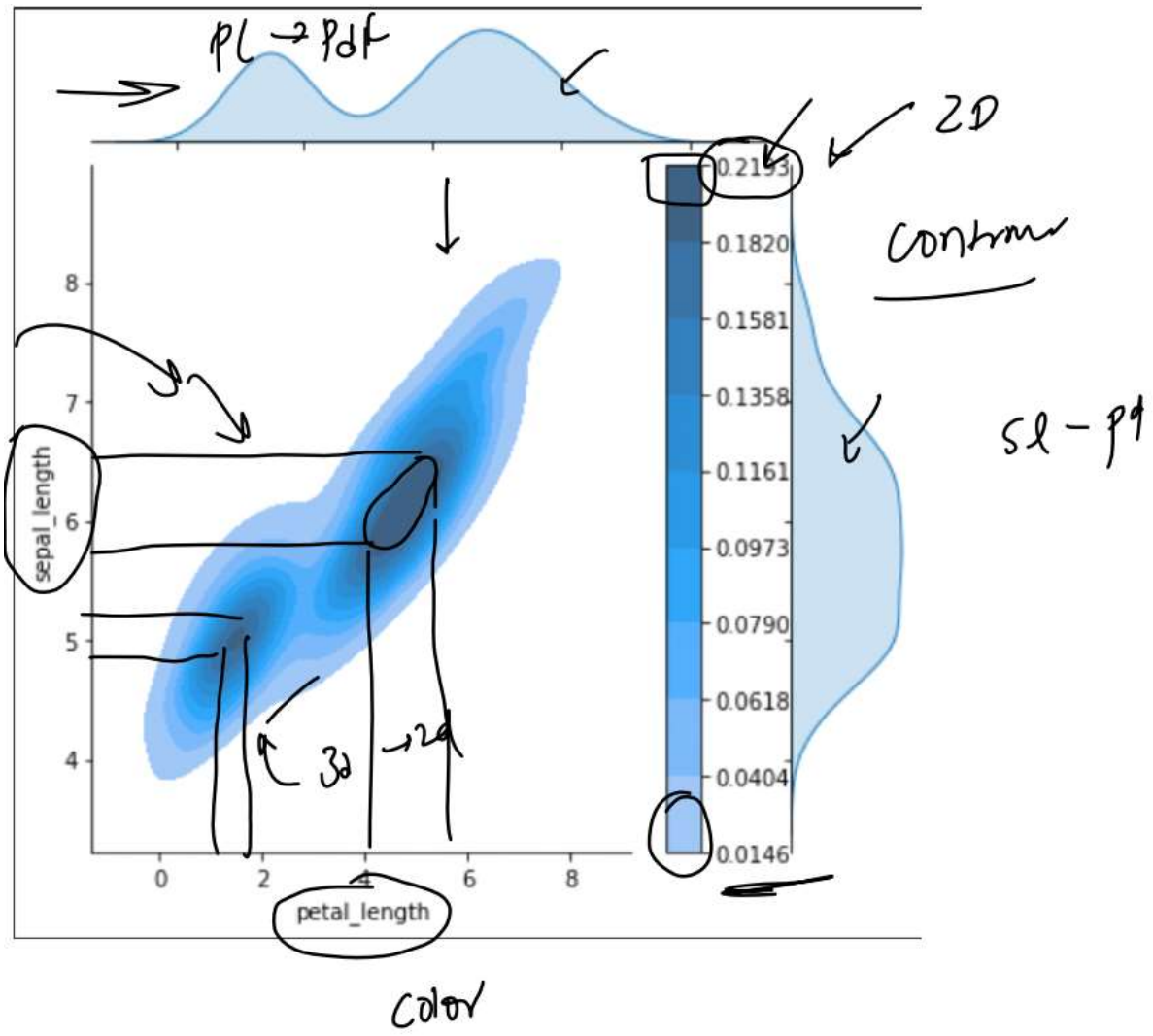
It shows the cumulative density of any data set over time

2D density curve

In [61]:
```
sns.jointplot(data=df, x="petal_length",
              y="sepal_length", kind="kde",fill=True,cbar=True)
```

Out[61]:
```
<seaborn.axisgrid.JointGrid at 0x1e286c37910>
```



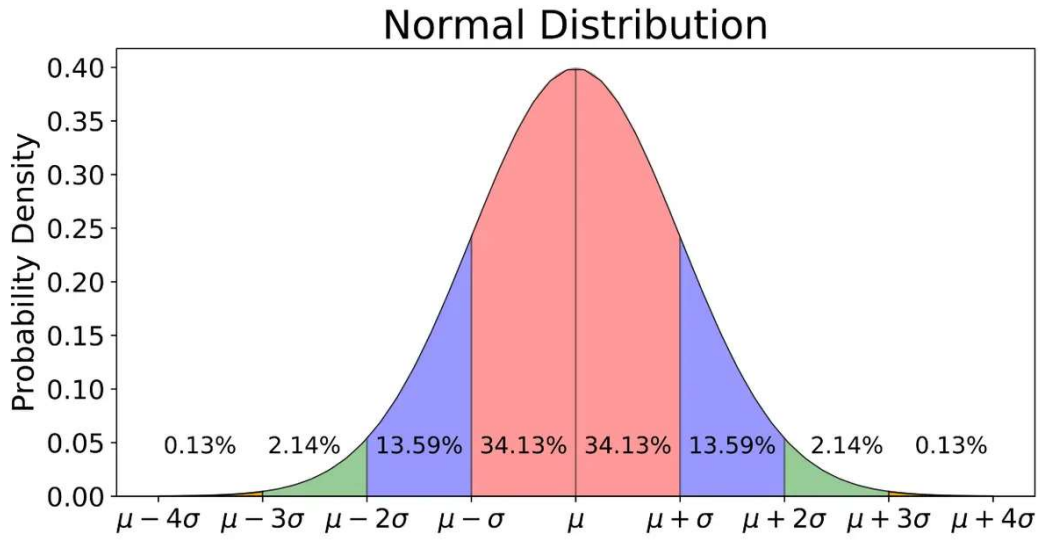


In 2D plot ,color is representing the Third Dimension

Normal Distribution

1.What is normal distribution?

Normal distribution, also known as Gaussian distribution, is a probability distribution that is commonly used in statistical analysis. It is a continuous probability distribution that is symmetrical around the mean, with a bell-shaped curve



The normal distribution is characterized by two parameters:

the mean (μ) and **the standard deviation (σ)**. The mean represents the centre of the distribution, while the standard deviation represents the spread of the distribution. Denoted as:

$$y = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$\begin{aligned}
\mu &= \text{ Mean} \\
\sigma &= \text{ Standard Deviation} \\
\pi &\approx 3.14159\cdots \\
e &\approx 2.71828\cdots
\end{aligned}$$

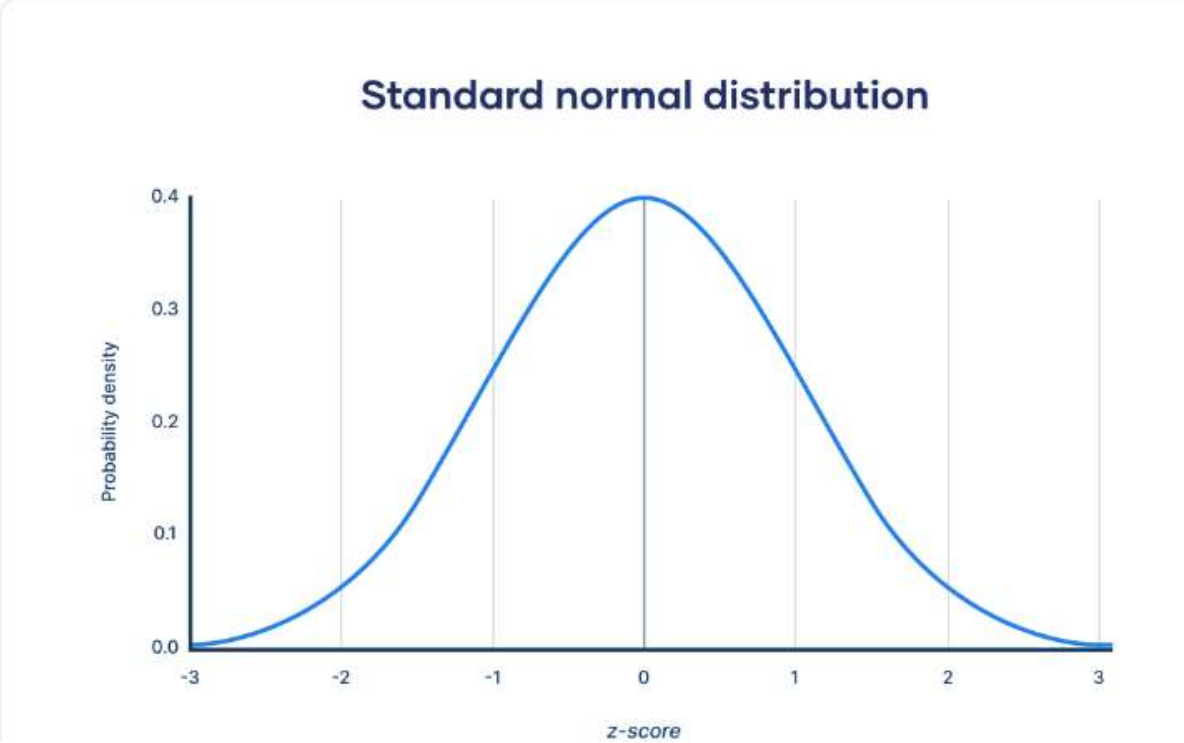
Why is it so important?

Commonality in Nature: Many natural phenomena follow a normal distribution, such as the heights of people, the weights of objects, the IQ scores of a population, and many more. Thus, the normal distribution provides a convenient way to model and analyse such data.

https://samp-suman-normal-dist-visualize-app-lkntug.streamlit.app/ (https://samp-suman-normal-dist-visualize-app-lkntug.streamlit.app/)

Standard Normal Variate(Z)

Standard Normal distribution



- **What is Standard Normal Variate?**

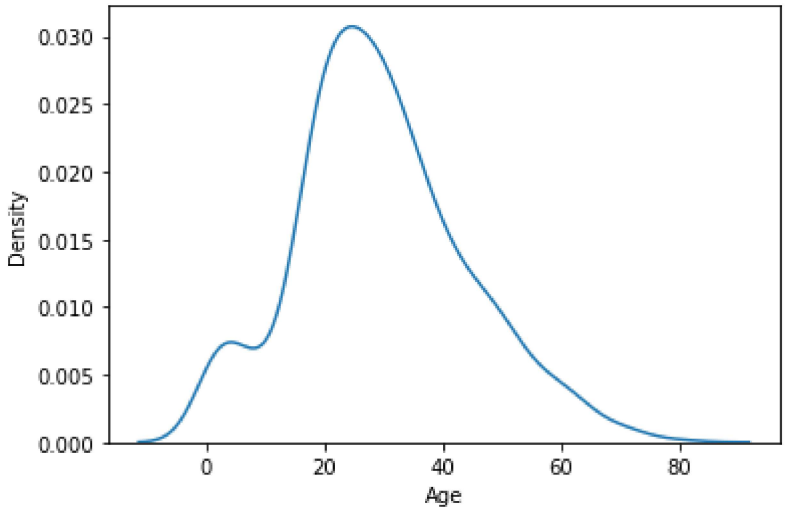
A Standard Normal Variate(Z) is a standardized form of the normal distribution with mean = 0 and standard deviation = 1

$$z \sim N(0,1)$$

Standardizing a normal distribution allows us to compare different distributions with each other, and to calculate probabilities using standardized tables or software.

In [62]:
```
# Example

sns.kdeplot(titanic['Age'])
```

Out[62]:
```
<AxesSubplot:xlabel='Age', ylabel='Density'>
```



In [63]:
```
titanic['Age'].mean()
```

Out[63]:
```
29.69911764705882
```

In [64]:
```
titanic['Age'].std()
```

Out[64]:
```
14.526497332334044
```

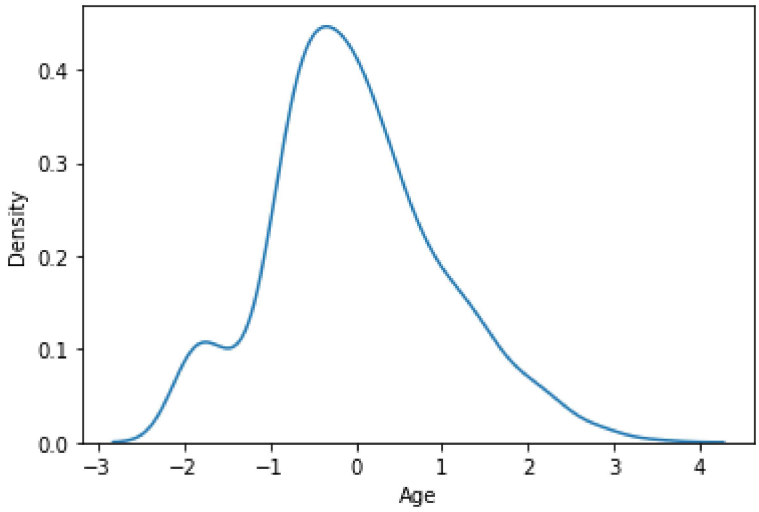
In [65]:
```
(titanic['Age'] - titanic['Age'].mean()) / titanic['Age'].std() # standarizati
```

Out[65]:
```
0     -0.530005
1      0.571430
2     -0.254646
3      0.364911
4      0.364911
         ...
886   -0.185807
887   -0.736524
888         NaN
889   -0.254646
890    0.158392
Name: Age, Length: 891, dtype: float64
```

In [66]:
```
x = (titanic['Age'] - titanic['Age'].mean()) / titanic['Age'].std()
```

In [67]:
```
sns.kdeplot(x) # values changed to 0 ,1,2
```

Out[67]: <AxesSubplot:xlabel='Age', ylabel='Density'>



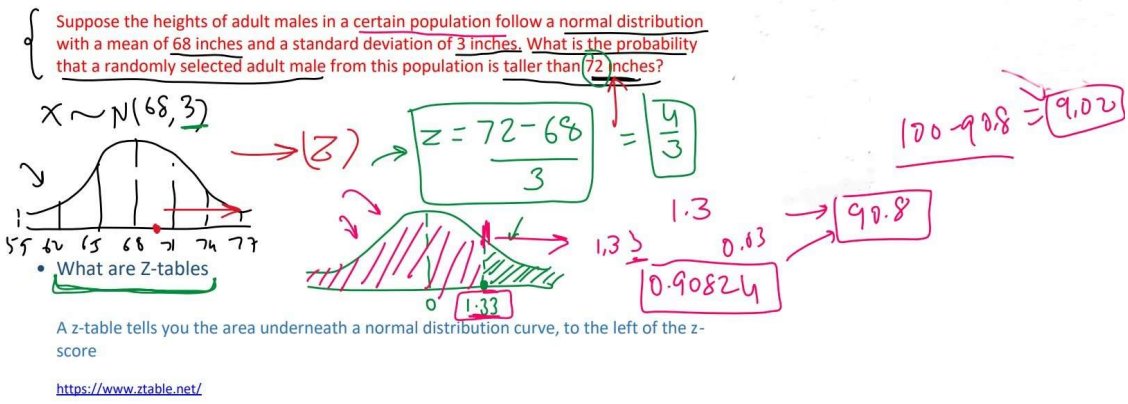
In [68]:
```
x.mean()
```

Out[68]: 2.0039214607642444e-16

In [69]:
```
x.std()
```

Out[69]: 0.9999999999999994

Problem

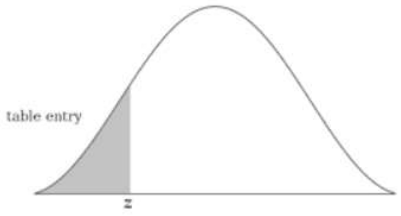


Z table

z	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
+0	.50000	.50399	.50798	.51197	.51595	.51994	.52392	.52790	.53188	.53586
+0.1	.53983	.54380	.54776	.55172	.55567	.55966	.56360	.56749	.57142	.57535
+0.2	.57926	.58317	.58706	.59095	.59483	.59871	.60257	.60642	.61026	.61409
+0.3	.61791	.62172	.62552	.62930	.63307	.63683	.64058	.64431	.64803	.65173
+0.4	.65542	.65910	.66276	.66640	.67003	.67364	.67724	.68082	.68439	.68793
+0.5	.69146	.69497	.69847	.70194	.70540	.70884	.71226	.71566	.71904	.72240
+0.6	.72575	.72907	.73237	.73565	.73891	.74215	.74537	.74857	.75175	.75490
+0.7	.75804	.76115	.76424	.76730	.77035	.77337	.77637	.77935	.78230	.78524
+0.8	.78814	.79103	.79389	.79673	.79955	.80234	.80511	.80785	.81057	.81327
+0.9	.81594	.81859	.82121	.82381	.82639	.82894	.83147	.83398	.83646	.83891
+1	.84134	.84375	.84614	.84849	.85083	.85314	.85543	.85769	.85993	.86214
+1.1	.86433	.86650	.86864	.87076	.87286	.87493	.87698	.87900	.88100	.88298
+1.2	.88493	.88686	.88877	.89065	.89251	.89435	.89617	.89796	.89973	.90147
+1.3	.90320	.90490	.90658	.90824	.90988	.91149	.91308	.91466	.91621	.91774
+1.4	.91924	.92073	.92220	.92364	.92507	.92647	.92785	.92922	.93056	.93189
+1.5	.93319	.93448	.93574	.93699	.93822	.93943	.94062	.94179	.94295	.94408

Z TABLE

Negative Z score table

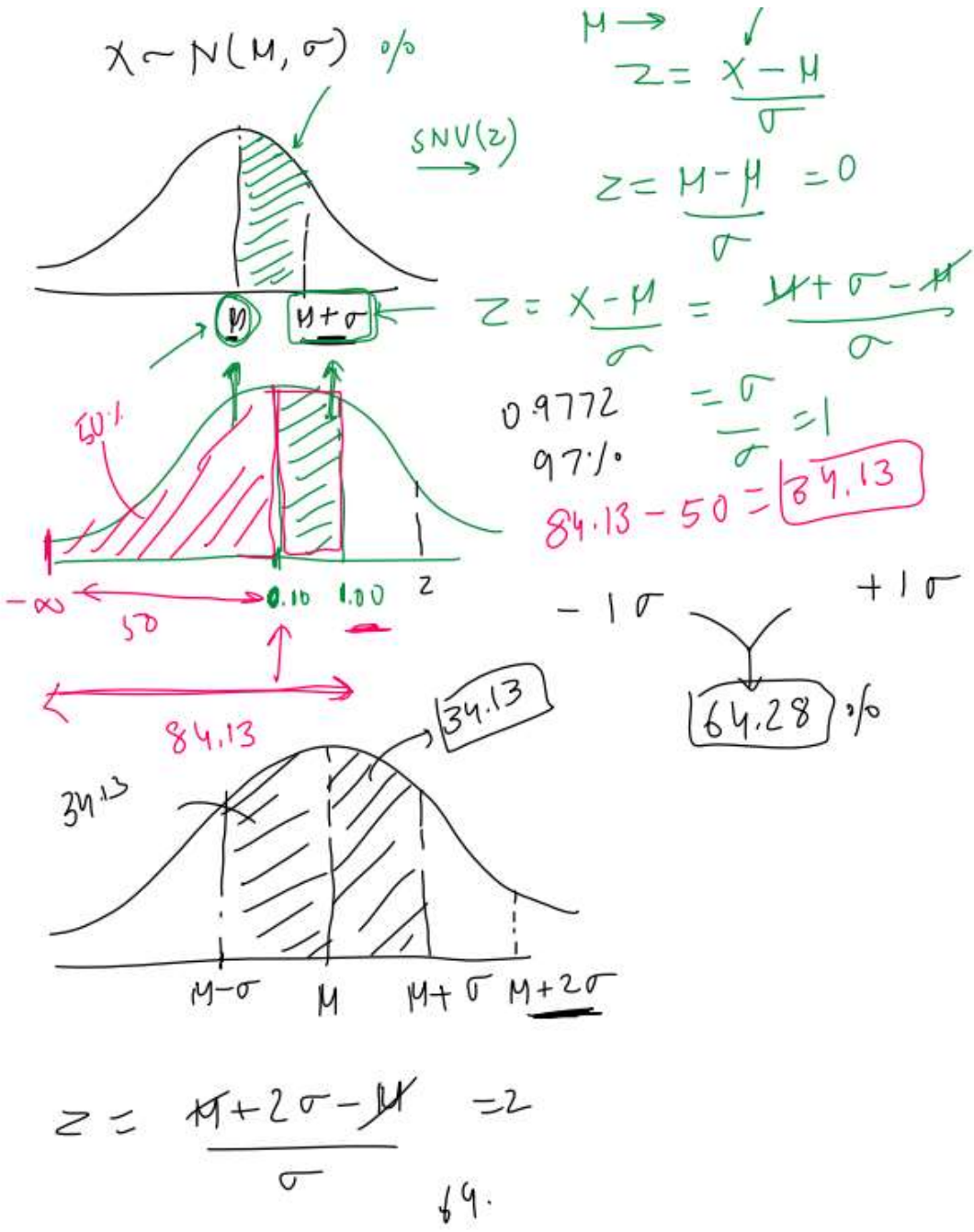


Use the negative Z score table below to find values on the left of the mean as can be seen in the graph alongside. Corresponding values which are less than the mean are marked with a negative score in the z-table and respresent the area under the bell curve to the left of z.

Positive Z score

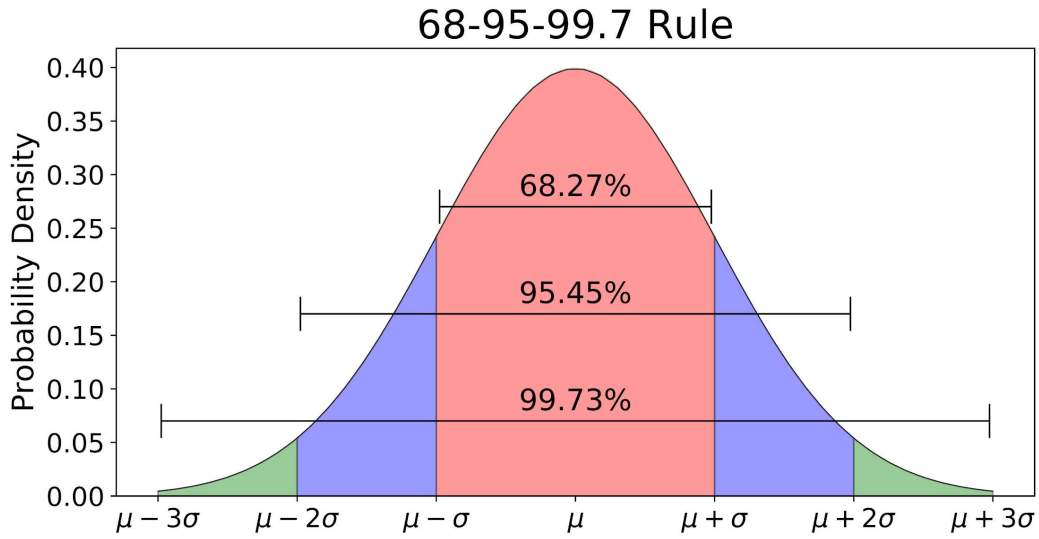
Use the positive Z score table below to find values on the right of the mean as can be seen in the graph alongside. Corresponding values which are greater than the mean are marked with a positive score in the z-table and respresent the area under the bell curve to the left of z.

- **Problem** For a Normal Distribution X~(u,std) what percent of population lie between mean and 1 standard deviation, 2 std and 3 std?



Empirical Rule:

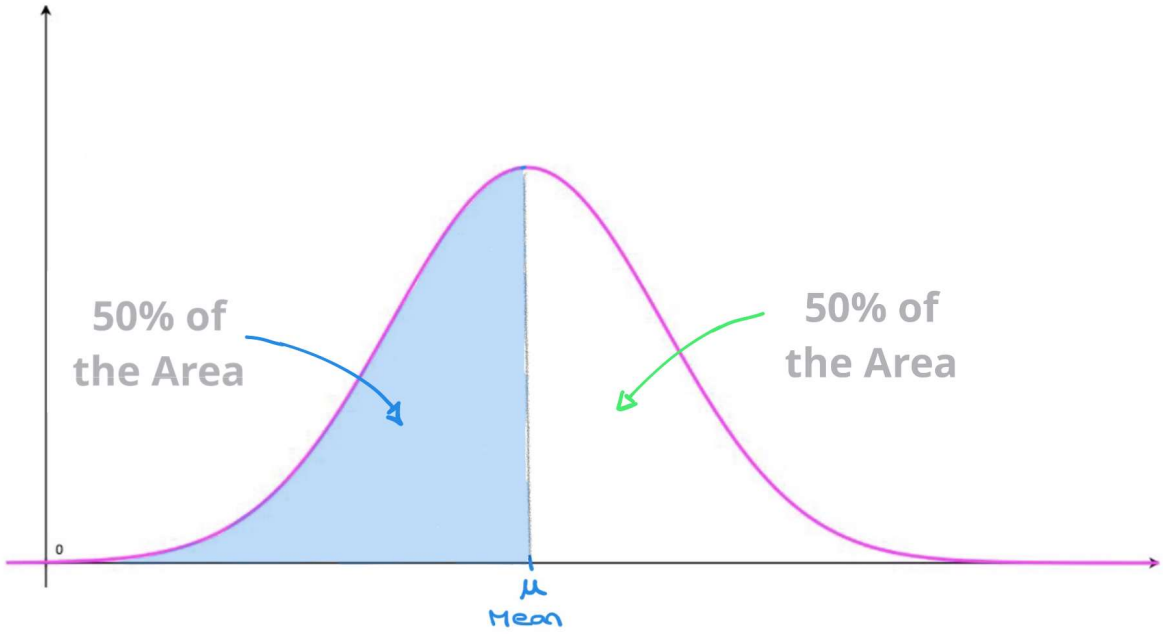
The normal distribution has a well-known empirical rule, also called the **68-95-99.7 rule**, which states that approximately 68% of the data falls within one standard deviation of the mean, about 95% of the data falls within two standard deviations of the mean, and about 99.7% of the data falls within three standard deviations of the mean.



Properties of Normal Distribution

1. Symmetricity

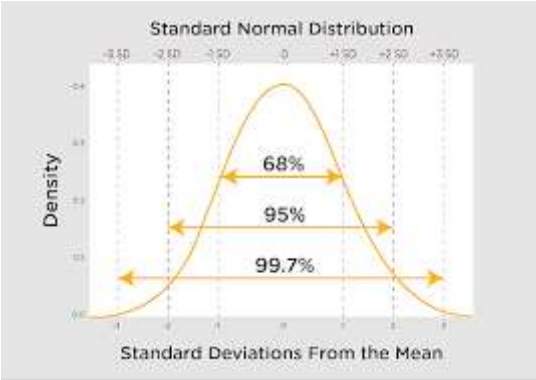
The normal distribution is symmetric about its mean, which means that the probability of observing a value above the mean is the same as the probability of observing a value below the mean. The bell-shaped curve of the normal distribution reflects this symmetry.



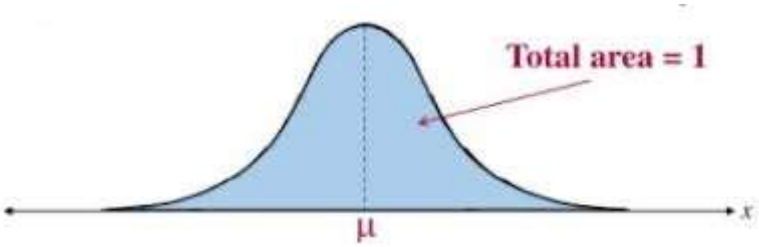
2. Measures of Central Tendencies are equal

Mean , median , mode

3. Empirical Rule



4. The area under the curve

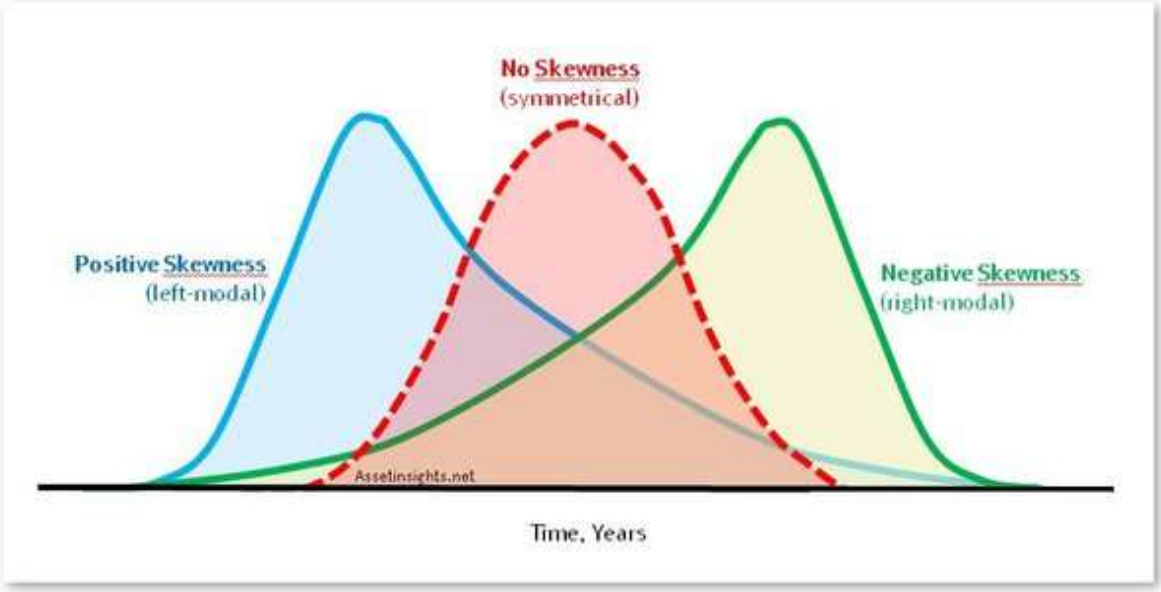


Skewness

What is skewness?

A normal distribution is a bell-shaped, symmetrical distribution with a specific mathematical formula that describes how the data is spread out. **Skewness indicates that the data is not symmetrical, which means it is not normally distributed**.

- Skewness is a measure of the asymmetry of a probability distribution. It is a statistical measure that describes the degree to which a dataset deviates from the normal distribution.
- In a symmetrical distribution, the mean, median, and mode are all equal. In contrast, in a skewed distribution, **the mean, median, and mode are not equal**, and the distribution tends to have a longer tail on one side than the other.
- Skewness can be positive, negative, or zero. A positive skewness means that the tail of the distribution is longer on the right side, while a negative skewness means that the tail is longer on the left side. A zero skewness indicates a perfectly symmetrical distribution



Right skewed: mode < median < mean

Left skewed: mode > median > mean

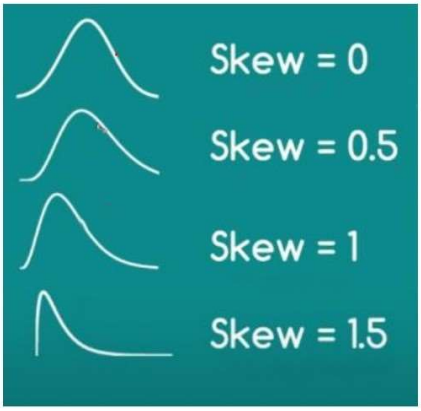
The greater the skew the greater the distance between mode, median and mean

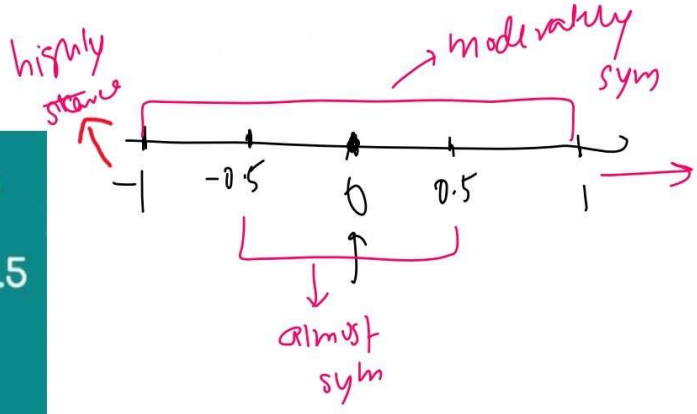
- **How Skewness is Calculated**

Moment number	Name	Measure of	Formula
1	Mean	Central tendency	$\bar{X} = \frac{\sum_{i=1}^{N} X_i}{N}$
2	Variance (Volatility)	Dispersion	$\sigma^2 = \frac{\sum_{i=1}^{N} (X_i - \bar{X})^2}{N}$
3	Skewness	Symmetry (Positive or Negative)	$Skew = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{(X_i - \bar{X})}{\sigma} \right]^3$
4	Kurtosis	Shape (Tall or flat)	$Kurt = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{(X_i - \bar{X})}{\sigma} \right]^4$

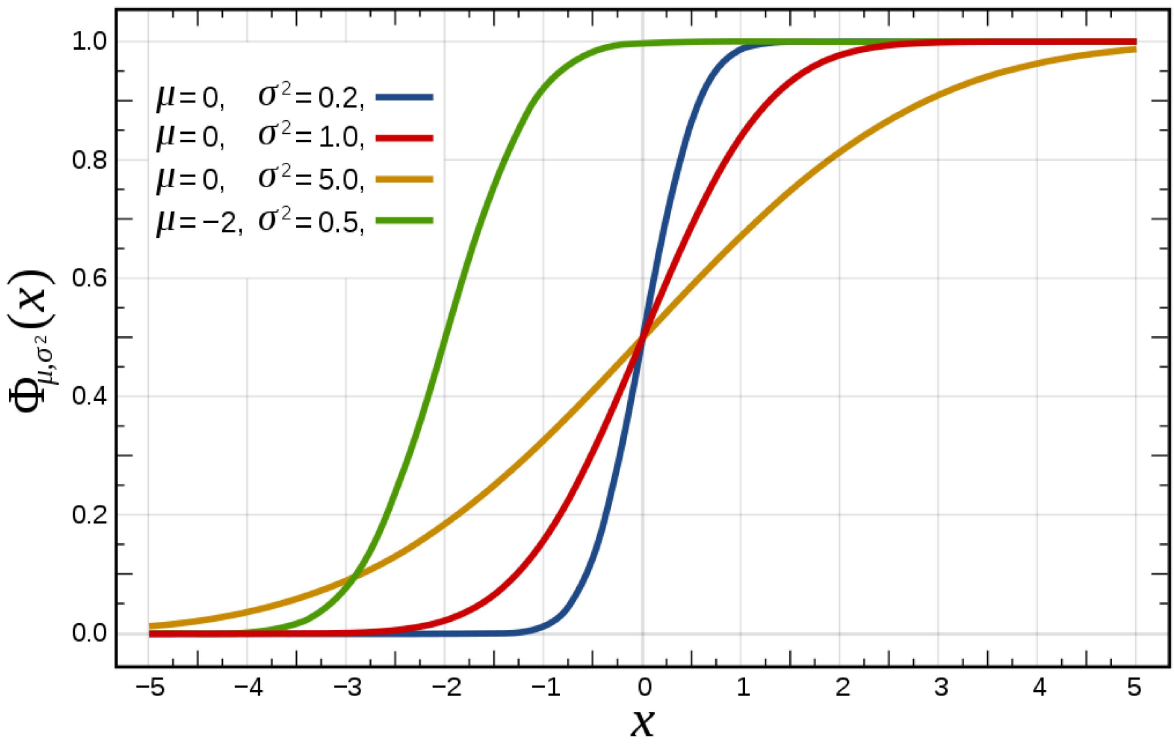
Where X is a random variable having N observations (i = 1,2,…,N).

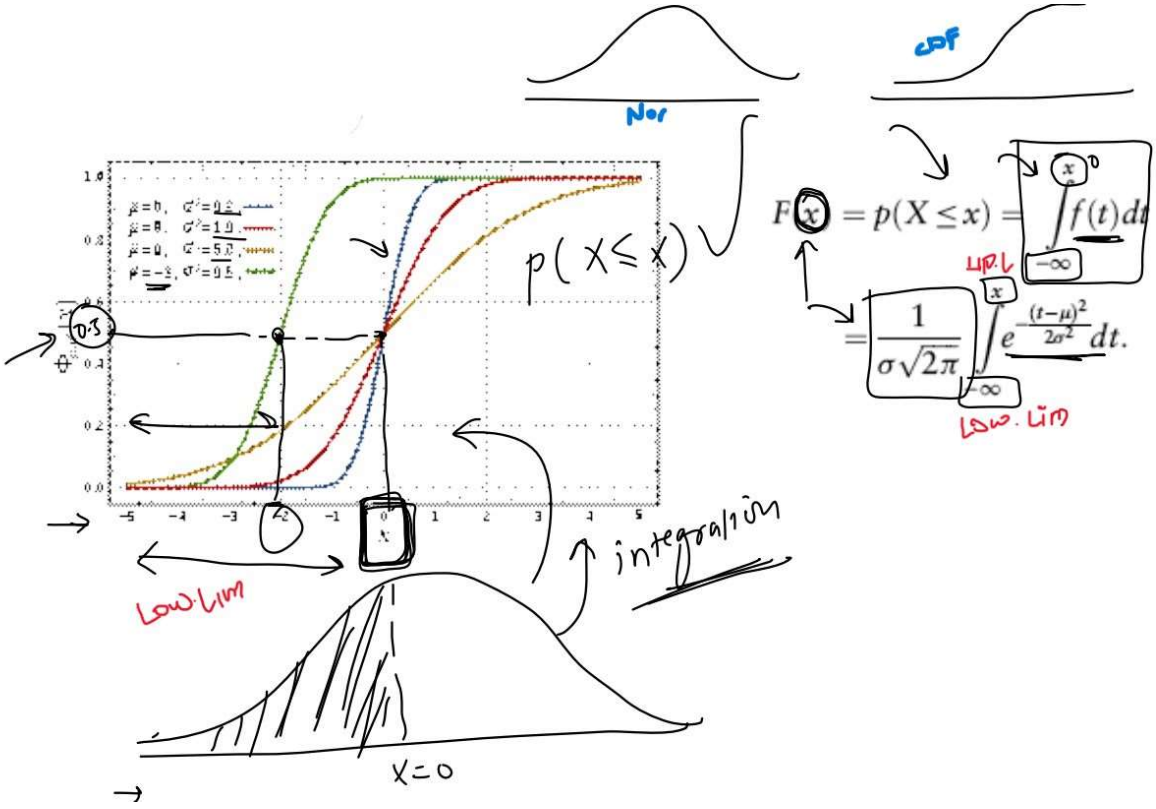
Interpretation





CDF of Normal Distribution





Normalisation vs. Standardisation

Normalisation is suitable to use when the **data does not follow Gaussian Distribution principles**. It can be used in algorithms that do not assume data distribution, such as K-Nearest Neighbors and Neural Networks.

On the other hand, **standardisation** is beneficial in cases where the **dataset follows the Gaussian distribution**. Unlike Normalization, Standardisation is not affected by the outliers in the dataset as it does not have any bounding range.

Applying Normalization or Standardisation depends on the problem and the machine learning algorithm. There are no definite rules as to when to use Normalization or Standardisation. One can fit the normalized or standardized dataset into the model and compare the two.

It is always advisable to first fit the scaler on the training data and then transform the testing data. This would prohibit data leakage during the model testing process, and the scaling of target values is generally not required.

Normalisation	Standardisation
Scaling is done by the highest and the lowest values.	Scaling is done by mean and standard deviation.
It is applied when the features are of separate scales.	It is applied when we verify zero mean and unit standard deviation.
Scales range from 0 to 1	Not bounded
Affected by outliers	Less affected by outliers
It is applied when we are not sure about the data distribution	It is used when the data is Gaussian or normally distributed
It is also known as Scaling Normalization	It is also known as Z-Score

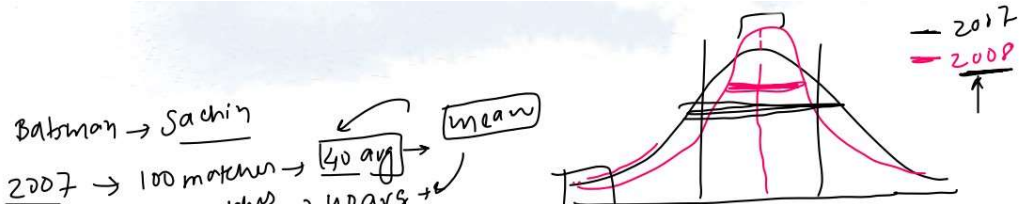
Q:What is Kurtosis.?

Kurtosis is the 4th statistical moment. In probability theory and statistics, kurtosis (meaning "curved, arching") is a **measure of the "tailedness"**of the probability distribution of a real-valued random variable. Like skewness, kurtosis describes a particular aspect of a probability distribution.

Moment in distribution

- **1. Mean**
- **2. standard deviation**
- **3. Skewness**
- **4. Kurtosis**

- **Example:**



- **Inshort**:

Kurtosis is the distribution with a **Fat** tail, and tail Fatness indicates the presence of **outliers**.

Formula:

$$\left[\frac{n(n+1)}{(n-1)(n-2)(n-3)}\sum\left(\frac{x_i-\bar{x}}{s}\right)^4\right]-\frac{3(n-1)^2}{(n-2)(n-3)}$$

Practical Use-case

- In finance, **kurtosis risk** refers to the risk associated with the possibility of extreme outcomes or **"fat tails"** in the distribution of returns of a particular asset or portfolio.
- If a distribution has high kurtosis, it means that there is a higher likelihood of extreme events occurring, either positive or negative, compared to a normal distribution.
- In finance, kurtosis risk is important to consider because it indicates that there is a greater probability of large losses or gains occurring, which can have significant implications for investors. As a result, investors may want to adjust their investment strategies to account for kurtosis risk.

Excess Kurtosis

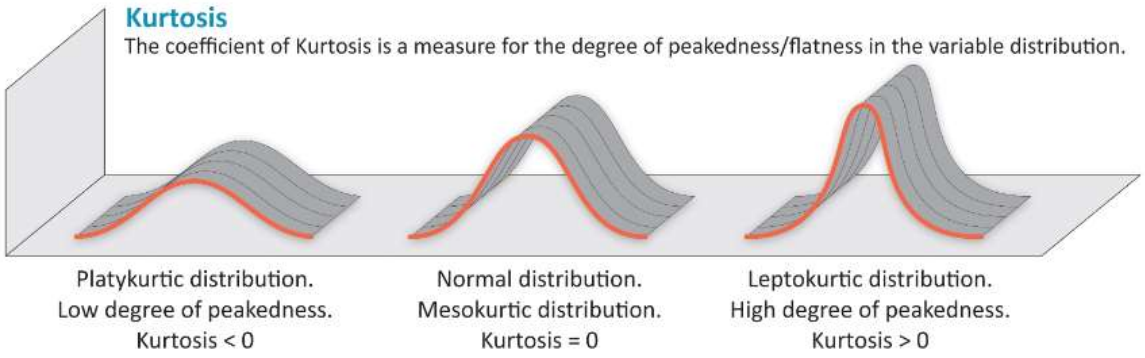
Excess kurtosis is a measure of how much **more peaked or flat a distribution** is compared to a normal distribution, which is considered to have a kurtosis of 0. It is calculated by subtracting 3 from the sample kurtosis coefficient.

Types of Kurtosis:

1. Leptokurtic

A distribution with **positive excess kurtosis** is called leptokurtic. "Lepto-" means "slender". In terms of shape, a leptokurtic distribution has fatter tails. This indicates that there are more extreme values or outliers in the distribution.

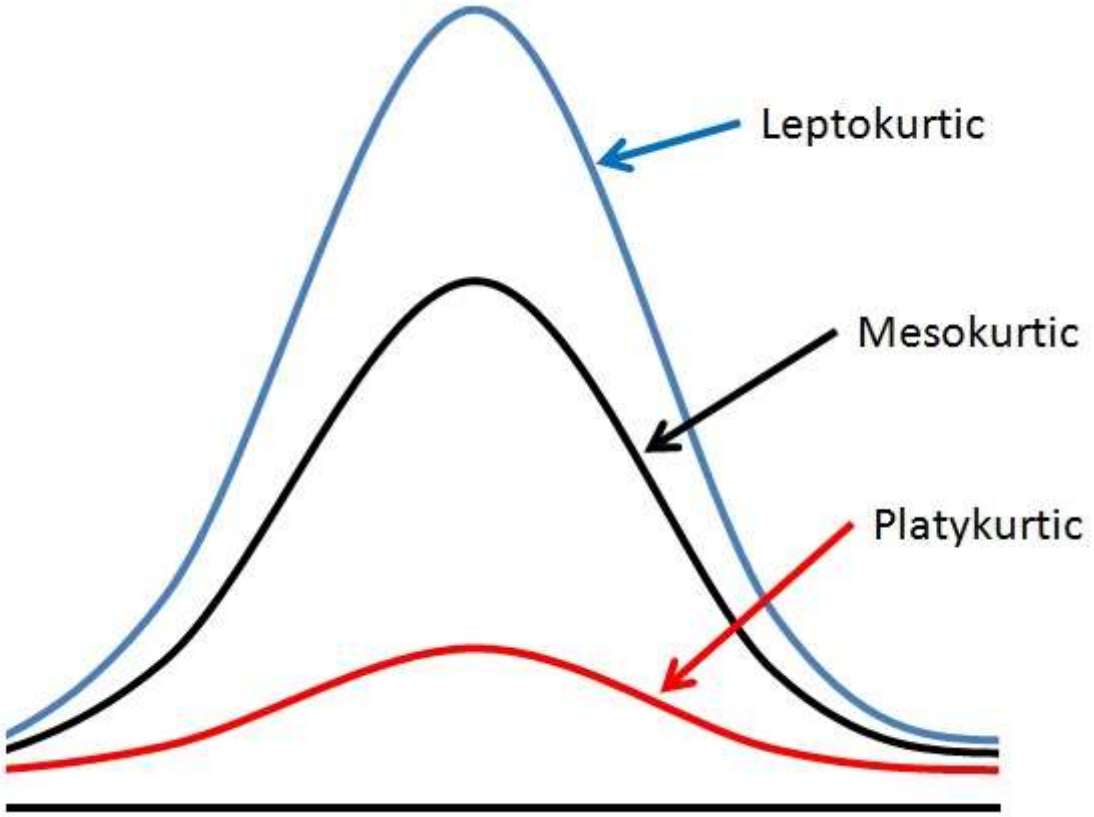
Example - Assets with positive excess kurtosis are riskier and more volatile than those with a normal distribution, and they may experience sudden price movements that can result in significant gains or losses.



2. Platykurtic

A distribution with **negative excess kurtosis** is called platykurtic. "Platy-" means "broad". In terms of shape, a platykurtic distribution has thinner tails. This indicates that there are fewer extreme values or outliers in the distribution.

Assets with negative excess kurtosis are less risky and less volatile than those with a normal distribution, and they may experience more gradual price movements that are less likely to result in large gains or losses



3. Mesokurtic

Distributions with **zero excess kurtosis** are called mesokurtic. The most prominent example of a mesokurtic distribution is the **normal distribution** family, regardless of the values of its parameters.

Mesokurtic is a term used to describe a distribution with a **excess kurtosis of 0**, indicating that it has the same degree of "peakedness" or "flatness" as a normal distribution.

Example - In finance, a mesokurtic distribution is considered to be the ideal distribution for assets or portfolios, as it represents a balance between risk and return

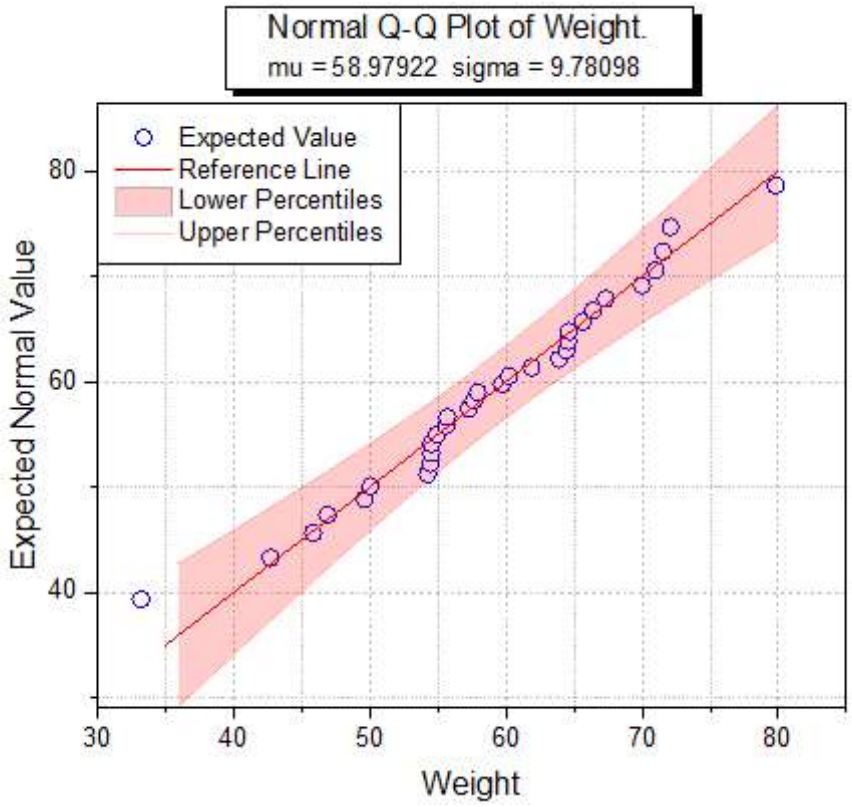
Q: How to find if a given distribution is normal or not?

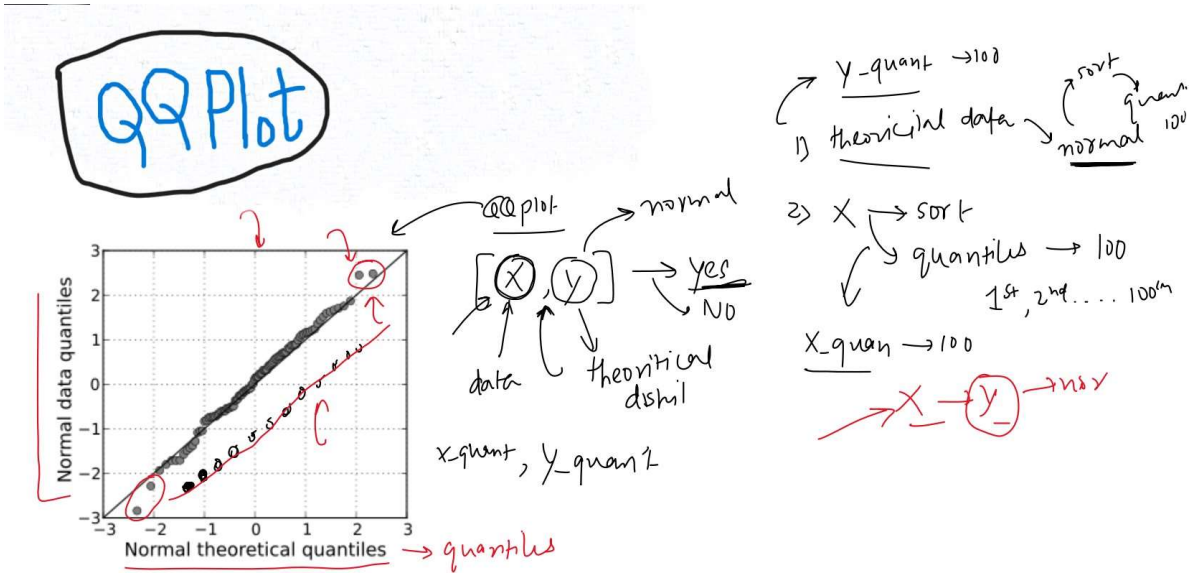
- **Visual inspection**: One of the easiest ways to check for normality is to visually inspect a histogram or a density plot of the data. A normal distribution has a **bell-shaped curve, which means that the majority of the data falls in the middle**, and the tails taper off symmetrically. If the distribution looks approximately bell-shaped, it is likely to be normal.
- **QQ Plot**: Another way to check for normality is to create a normal probability plot (also known as a Q-Q plot) of the data. A normal probability plot plots the observed data against the expected values of a normal distribution. If the data points fall along a straight line, the distribution is likely to be normal.

- **Statistical tests**: There are several statistical tests that can be used to test for normality, such as the Shapiro-Wilk test, the Anderson-Darling test, and the Kolmogorov-Smirnov test. These tests compare the observed data to the expected values of a normal distribution and provide a **p-value** that indicates whether the data is likely to be normal or not. A p-value less than the significance level (usually 0.05) suggests that the data is not normal.

Q: What is QQ Plot and How it is Plotted?

A QQ plot (**quantile-quantile plot**) is a graphical tool used to assess the similarity of the distribution of two sets of data. It is particularly useful for determining whether a set of data follows a normal distribution.



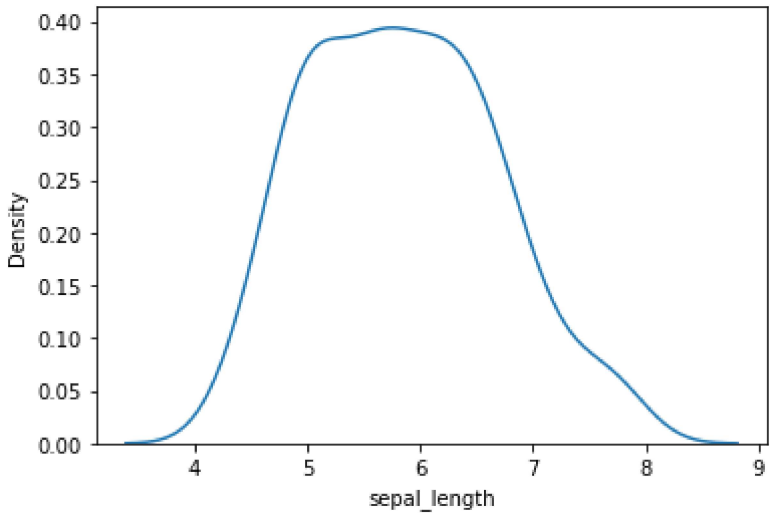


In [70]:
```
import numpy as np
import pandas as pd
import seaborn as sns
```

In [71]:
```
df = sns.load_dataset('iris')
```

In [72]:
```
sns.kdeplot(df['sepal_length'])
```

Out[72]: <AxesSubplot:xlabel='sepal_length', ylabel='Density'>



In [73]:
```
temp = sorted(df['sepal_length'].tolist())
```

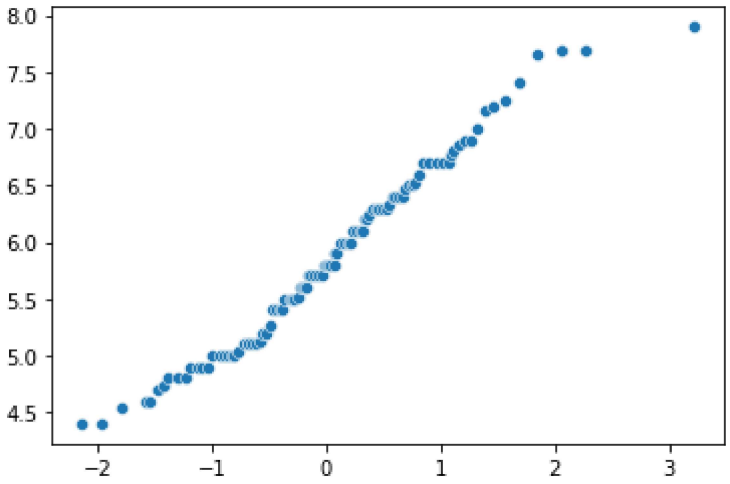
In [74]:
```
y_quant = []

for i in range(1,101):
    y_quant.append(np.percentile(temp,i))
```

In [75]:
```
samples = np.random.normal(loc=0,scale=1,size=1000) # mean =0 , std =1
```

In [76]:
```
x_quant = []

for i in range(1,101):
    x_quant.append(np.percentile(samples,i))
```

In [77]:
```
sns.scatterplot(x=x_quant,y=y_quant)
```

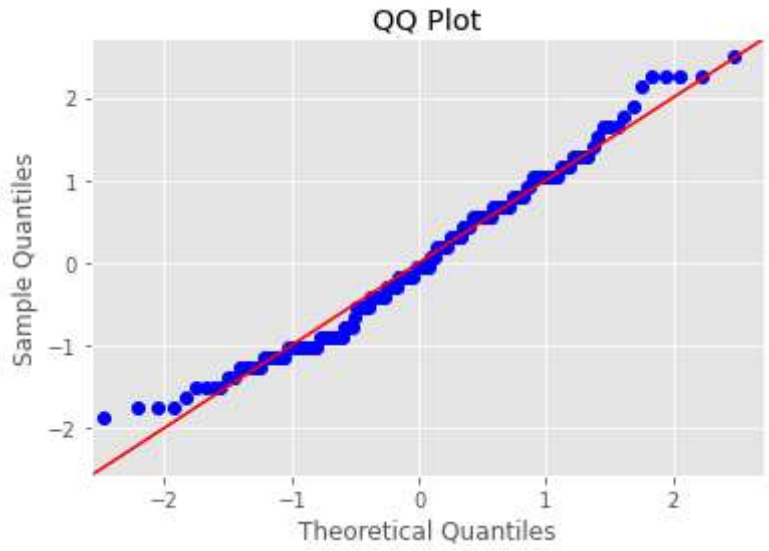
Out[77]: <AxesSubplot:>



using statsmodel

In [78]:
```
 # Style
plt.style.use('ggplot')

# using statsmodel

import statsmodels.api as sm
import matplotlib.pyplot as plt

# Create a QQ plot of the two sets of data

fig = sm.qqplot(df['sepal_length'], line='45', fit=True)

# Add a title and labels to the plot

plt.title('QQ Plot')
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Sample Quantiles')

# Show the plot
plt.show()
```

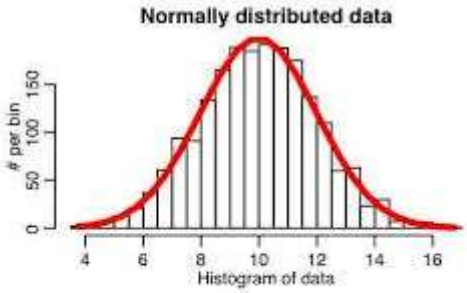
```
C:\Users\user\anaconda3\lib\site-packages\statsmodels\graphics\gofplots.py:993: UserWarning: marker is redundantly defined by the 'marker' keyword argument and the fmt string "bo" (-> marker='o'). The keyword argument will take precedence.
  ax.plot(x, y, fmt, **plot_style)
```

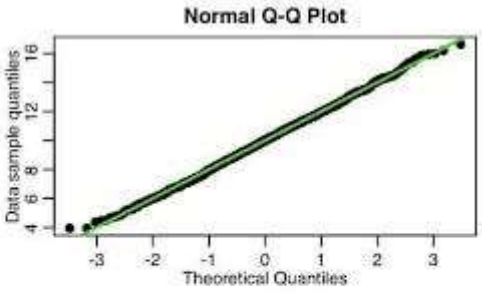


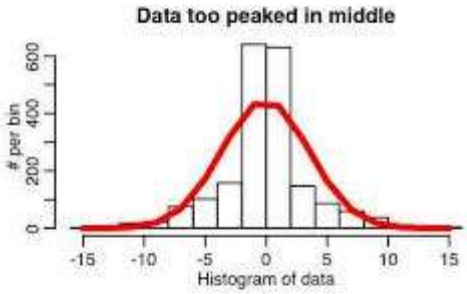
In a QQ plot, the quantiles of the two sets of data are plotted against each other. The quantiles of one set of data are plotted on the x-axis, while the quantiles of the other set of data are plotted on the y-axis. If the two sets of data have the same distribution, the points on the QQ plot will fall on a straight line. If the two sets of data do not have the same distribution, the points will deviate from the straight line.

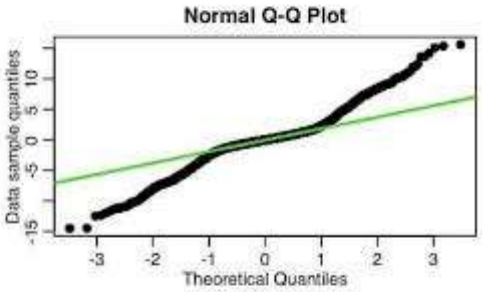
• https://www.statsmodels.org/dev/generated/statsmodels.graphics.gofplots.qqplot.html (https://www.statsmodels.org/dev/generated/statsmodels.graphics.gofplots.qqplot.html)

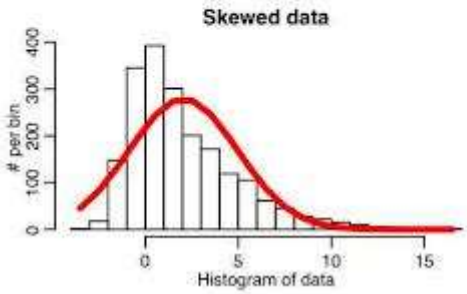
How to interpret QQ plots

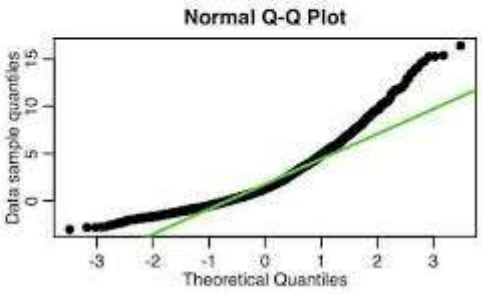


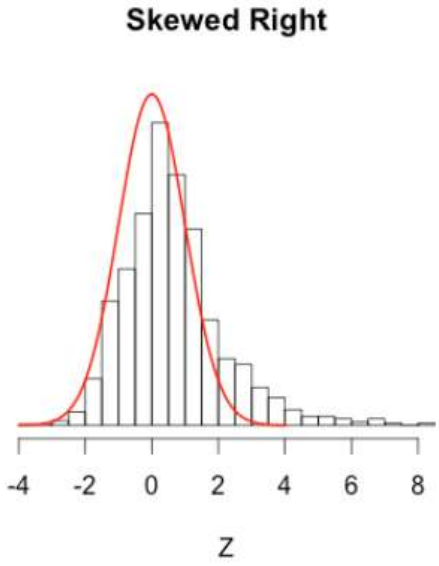


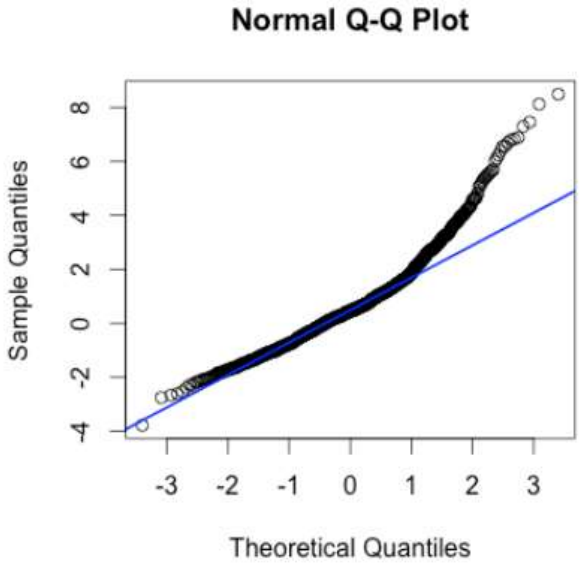


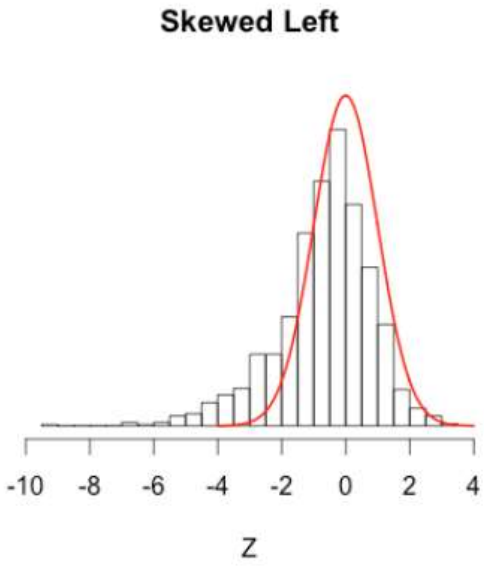


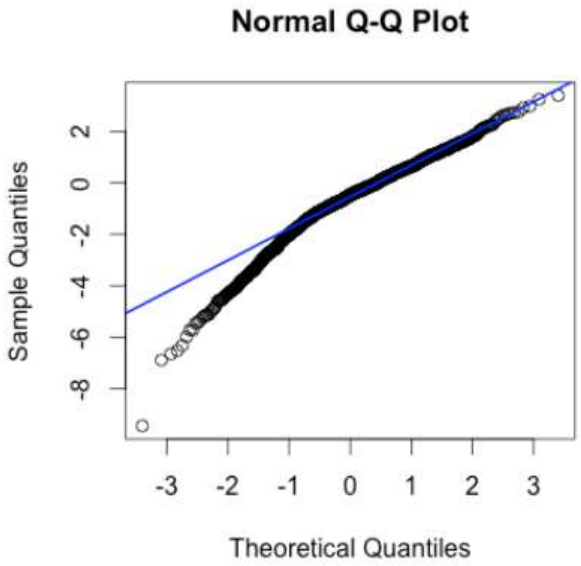


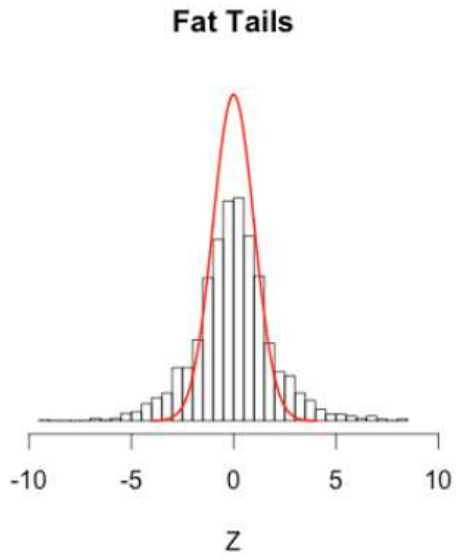


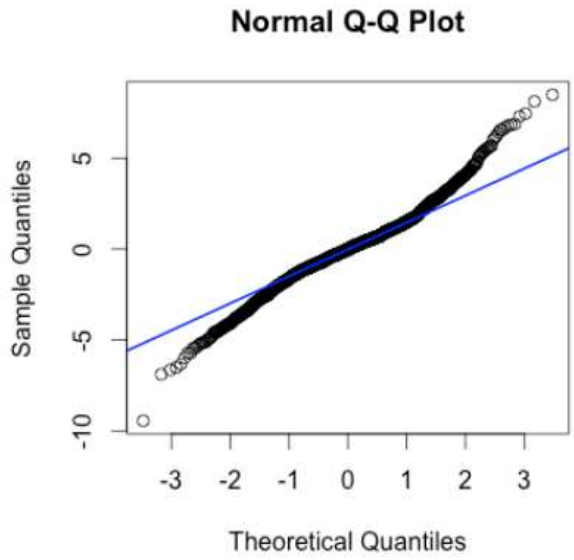

Skewed Right
-4 -2 0 2 4 6 8
Z

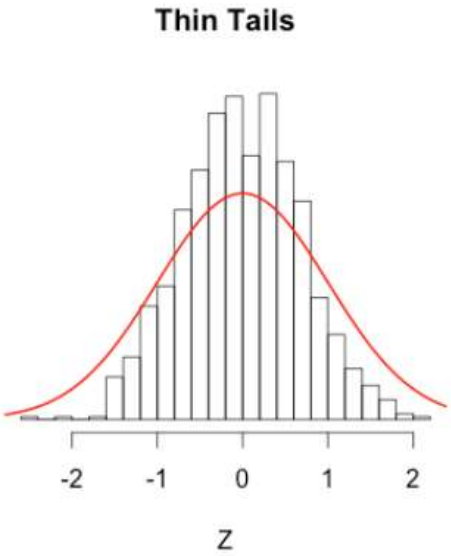

Normal Q-Q Plot
Sample Quantiles
-4 -2 0 2 4 6 8
-3 -2 -1 0 1 2 3
Theoretical Quantiles

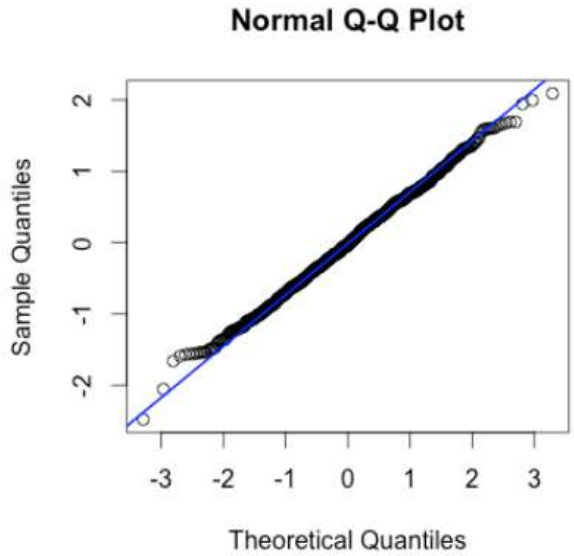

Skewed Left
-10 -8 -6 -4 -2 0 2 4
Z


Normal Q-Q Plot
Sample Quantiles
-8 -6 -4 -2 0 2
-3 -2 -1 0 1 2 3
Theoretical Quantiles





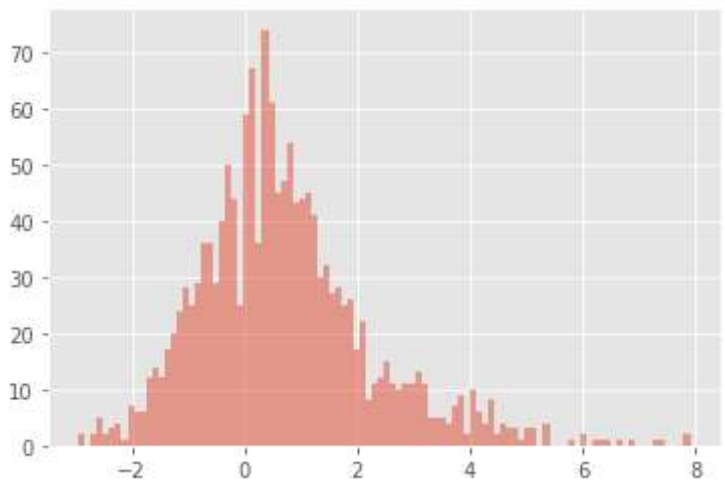


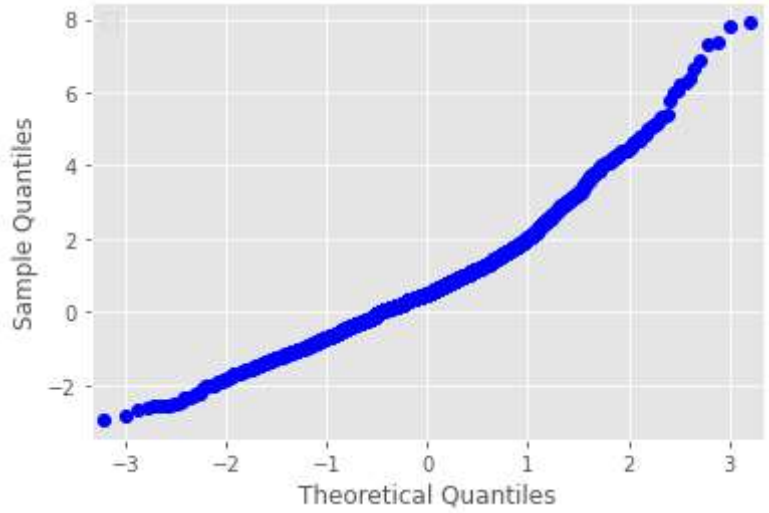


skewed right & skewed left

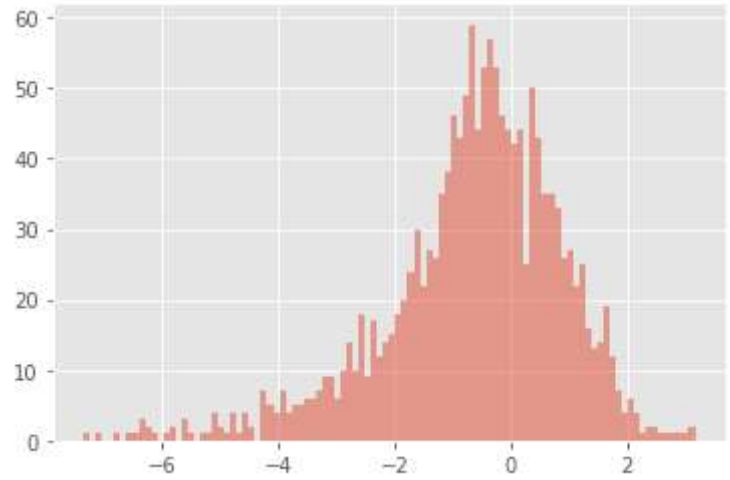
In [79]:
```
import numpy as np
import statsmodels.api as sm

# Generate a Gaussian random variable
gaussian_rv = np.random.normal(loc=0, scale=1, size=1000)

# Create a skewed right dataset
skew_right = np.concatenate((gaussian_rv[gaussian_rv > 0] * 2.5, gaussian_rv))

# Plot the histogram of the skewed right dataset
plt.hist(skew_right, bins=100, alpha=0.5, label="Skewed Right")

# Plot the QQ plot of the skewed right dataset
sm.qqplot(skew_right)
plt.legend()
plt.show()

# Create a skewed left dataset
skew_left = np.concatenate((gaussian_rv[gaussian_rv < 0] * 2.5, gaussian_rv))

# Plot the histogram of the skewed left dataset
plt.hist(skew_left, bins=100, alpha=0.5, label="Skewed Left")

# Plot the QQ plot of the skewed left dataset
sm.qqplot(skew_left)
plt.legend()
plt.show()
```

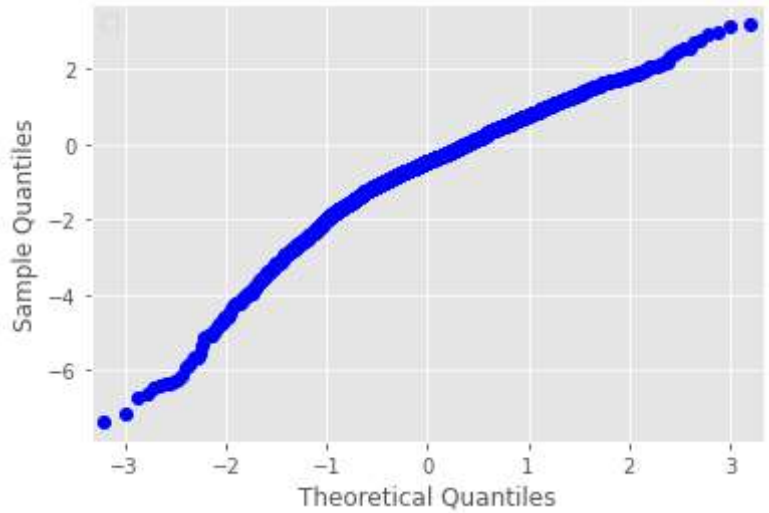
```
C:\Users\user\anaconda3\lib\site-packages\statsmodels\graphics\gofplots.py:99
3: UserWarning: marker is redundantly defined by the 'marker' keyword argumen
t and the fmt string "bo" (-> marker='o'). The keyword argument will take pre
cedence.
  ax.plot(x, y, fmt, **plot_style)
No handles with labels found to put in legend.
```






```
C:\Users\user\anaconda3\lib\site-packages\statsmodels\graphics\gofplots.py:993: UserWarning: marker is redundantly defined by the 'marker' keyword argument and the fmt string "bo" (-> marker='o'). The keyword argument will take precedence.
  ax.plot(x, y, fmt, **plot_style)
No handles with labels found to put in legend.
```

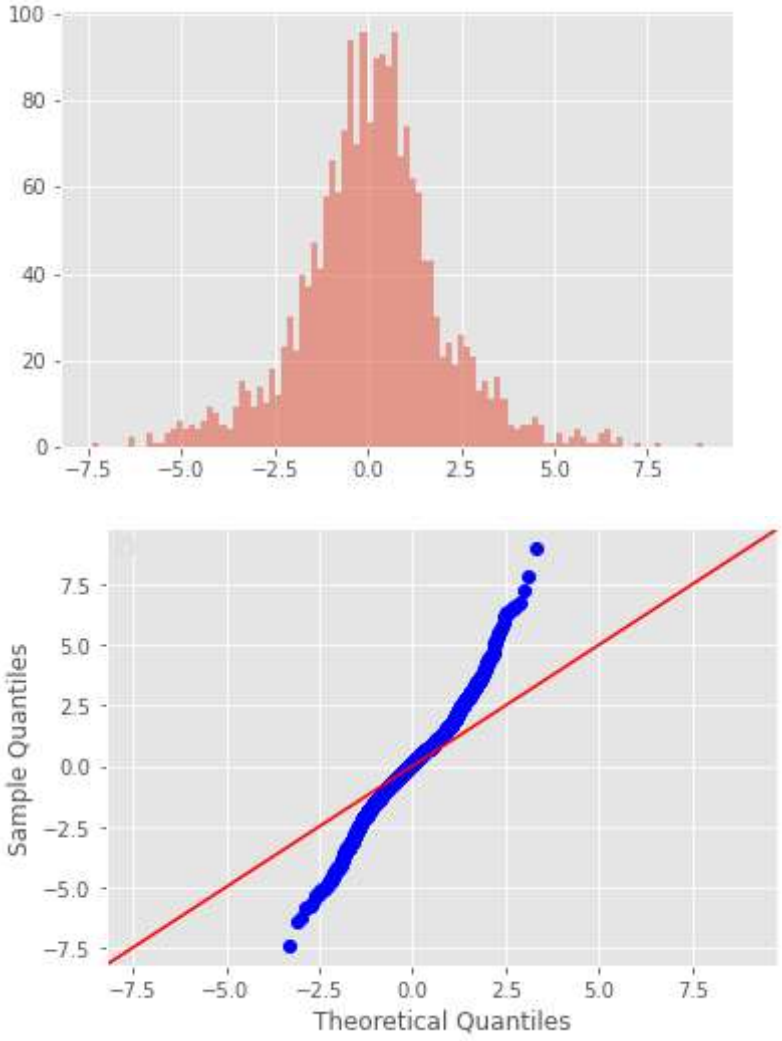




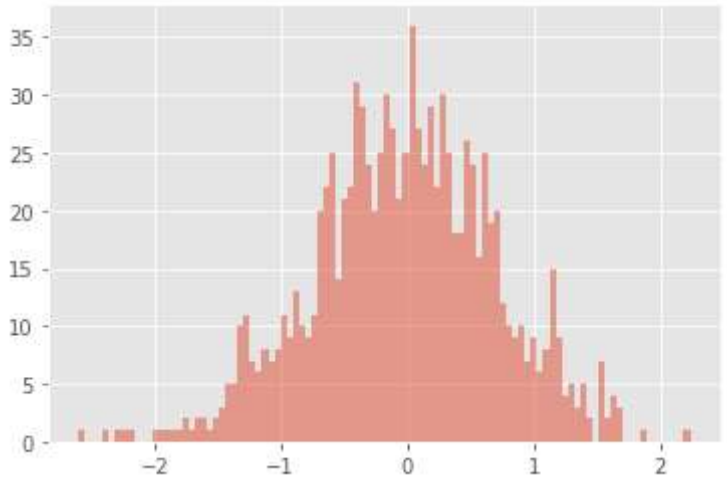
fat-tailed & thin-tailed

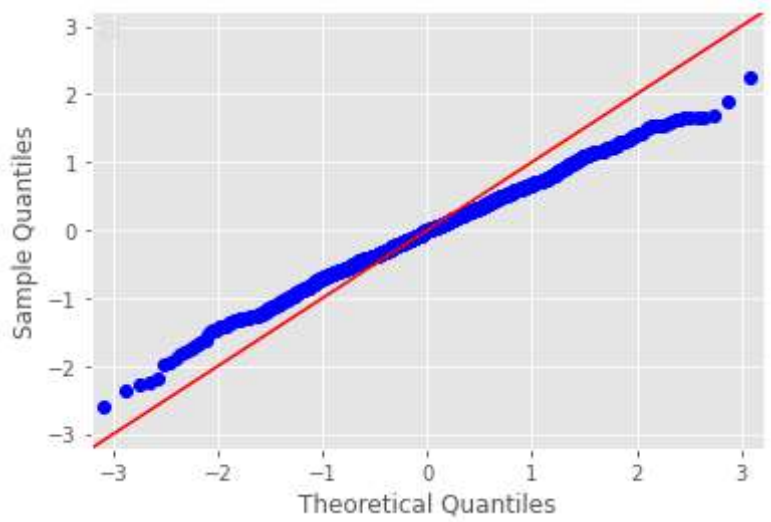
In [80]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from statsmodels.graphics.gofplots import qqplot

# Generate a Gaussian random variable

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

# Create a fat-tailed dataset

fat_tails = np.concatenate((gaussian_rv * 2.5, gaussian_rv))

# Plot the histogram of the fat-tailed dataset

plt.hist(fat_tails, bins=100, alpha=0.5, label="Fat Tails")

# Plot the QQ plot of the fat-tailed dataset

qqplot(fat_tails, line="45")
plt.legend()
plt.show()

# Create a thin-tailed dataset

thin_tails = np.random.normal(loc=0, scale=0.7, size=1000)

# Plot the histogram of the thin-tailed dataset

plt.hist(thin_tails, bins=100, alpha=0.5, label="Thin Tails")

# Plot the QQ plot of the thin-tailed dataset

qqplot(thin_tails, line="45")
plt.legend()
plt.show()
```

```
C:\Users\user\anaconda3\lib\site-packages\statsmodels\graphics\gofplots.py:99
3: UserWarning: marker is redundantly defined by the 'marker' keyword argumen
t and the fmt string "bo" (-> marker='o'). The keyword argument will take pre
cedence.
  ax.plot(x, y, fmt, **plot_style)
No handles with labels found to put in legend.
```




```
C:\Users\user\anaconda3\lib\site-packages\statsmodels\graphics\gofplots.py:99
3: UserWarning: marker is redundantly defined by the 'marker' keyword argumen
t and the fmt string "bo" (-> marker='o'). The keyword argument will take pre
cedence.
  ax.plot(x, y, fmt, **plot_style)
No handles with labels found to put in legend.
```



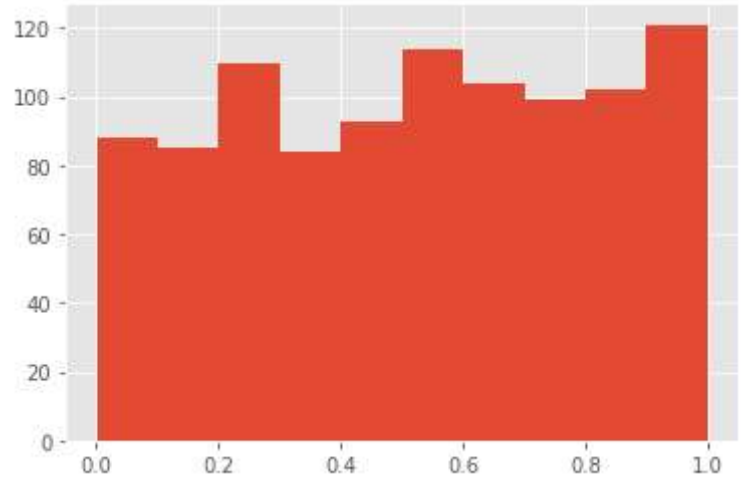


• Does QQ plot only detect normal distribution?

In [81]:
```
# Uniform Distribution with 1000 Points
import numpy as np
import scipy.stats as stats
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Generate a set of random data
x = np.random.uniform(low=0, high=1, size=1000)
```

In [82]:
```
plt.hist(x)
```

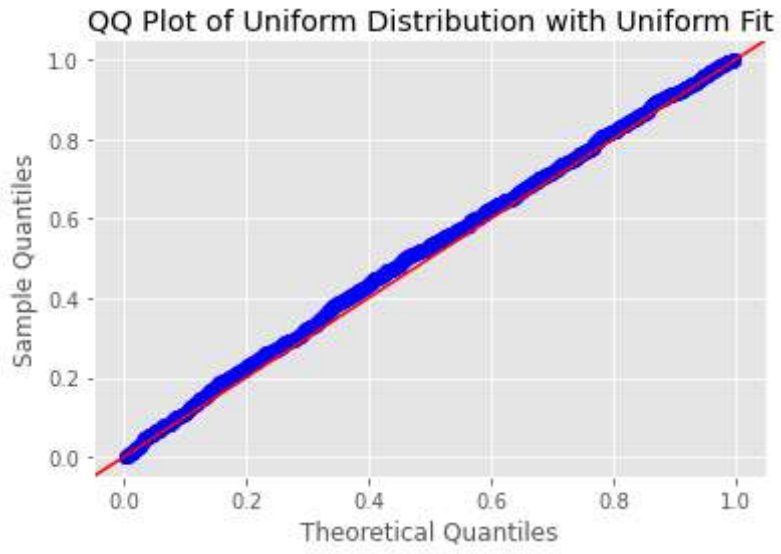
Out[82]:
```
(array([ 88.,  85., 110.,  84.,  93., 114., 104.,  99., 102., 121.]),
 array([0.00154826, 0.10131761, 0.20108696, 0.30085631, 0.40062566,
        0.50039501, 0.60016435, 0.6999337 , 0.79970305, 0.8994724 ,
        0.99924175]),
 <BarContainer object of 10 artists>)
```



In [83]:
```
# Fit a uniform distribution to the data

params = stats.uniform.fit(x)
dist = stats.uniform(loc=params[0], scale=params[1])
```

In [84]:
```
# Create a QQ plot of the data using the uniform distribution

fig = sm.qqplot(x, dist=dist, line='45')

# Add a title and labels to the plot

plt.title('QQ Plot of Uniform Distribution with Uniform Fit')
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Sample Quantiles')

# Show the plot
plt.show()
```

```
C:\Users\user\anaconda3\lib\site-packages\statsmodels\graphics\gofplots.py:993: UserWarning: marker is redundantly defined by the 'marker' keyword argument and the fmt string "bo" (-> marker='o'). The keyword argument will take precedence.
  ax.plot(x, y, fmt, **plot_style)
```



We can comparw two distributions with the Help of Using QQ plot , Not only Normal distribution but also uniform distribution and more , Mostly used to check the Normal distributions.

- QQ plots used to Check the **Normality of data**

Uniform Distribution

Q: What is Uniform Distribution and it's types

In probability theory and statistics, a uniform distribution is a probability distribution where all outcomes are **equally likely within a given range**. This means that if you were to select a random value from this range, any value would be as likely as any other value

Types of Unifrom Distribution

1. Discrete Uniform Distribution
2. Continous Uniform Distribution

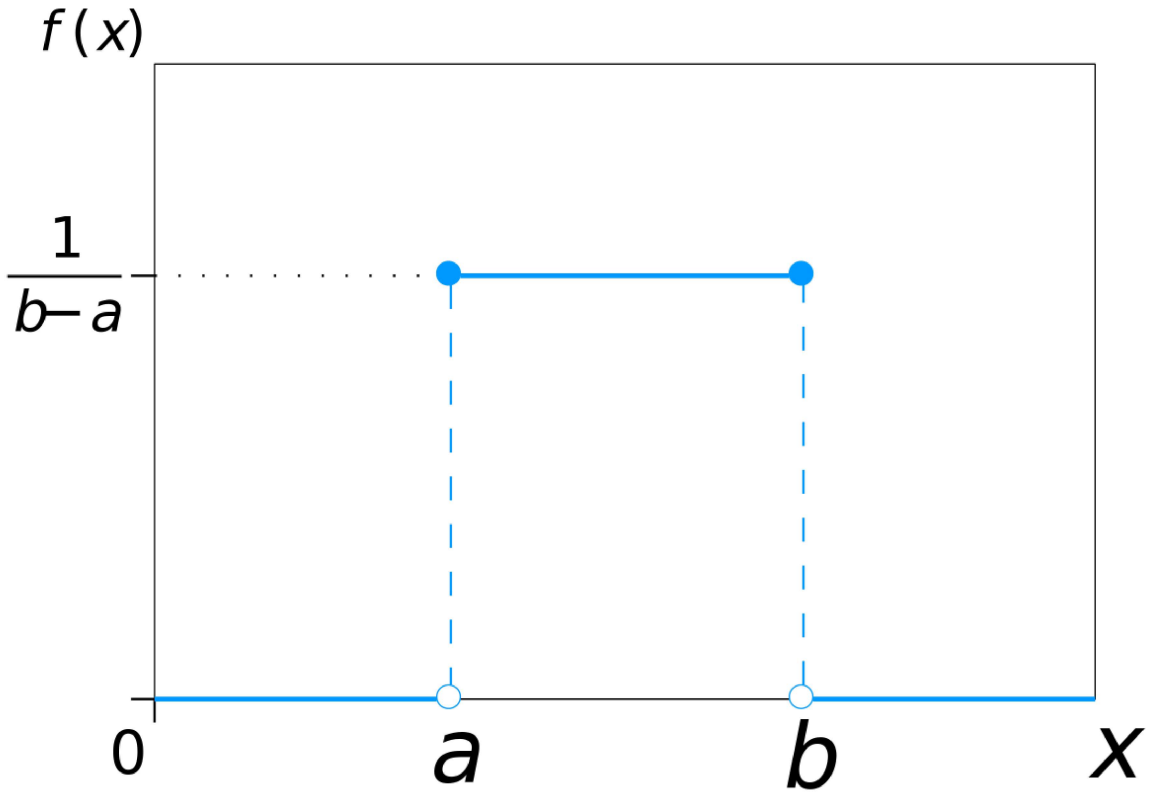
Examples of Continous Uniform Distribution

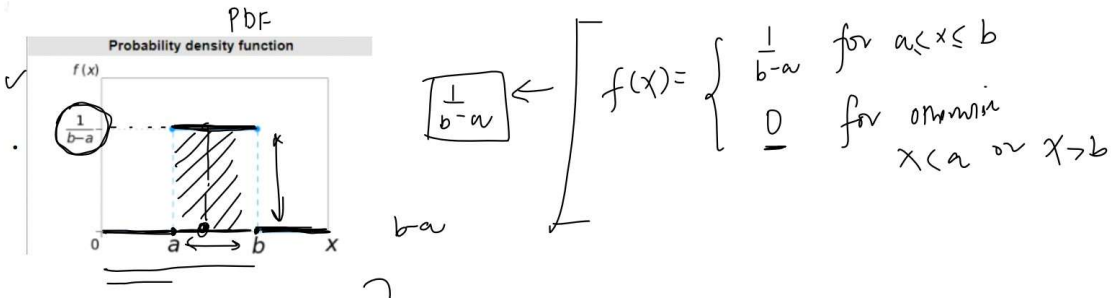
- The height of a person randomly selected from a group of individuals whose heights range from 5'6" to 6'0" would follow a continuous uniform distribution.
- The time it takes for a machine to produce a product, where the production time ranges from 5 to 10 minutes, would follow a continuous uniform distribution.

- The distance that a randomly selected car travels on a tank of gas, where the distance ranges from 300 to 400 miles, would follow a continuous uniform distribution.
- The weight of a randomly selected apple from a basket of apples that weighs between 100 and 200 grams, would follow a continuous uniform distribution.

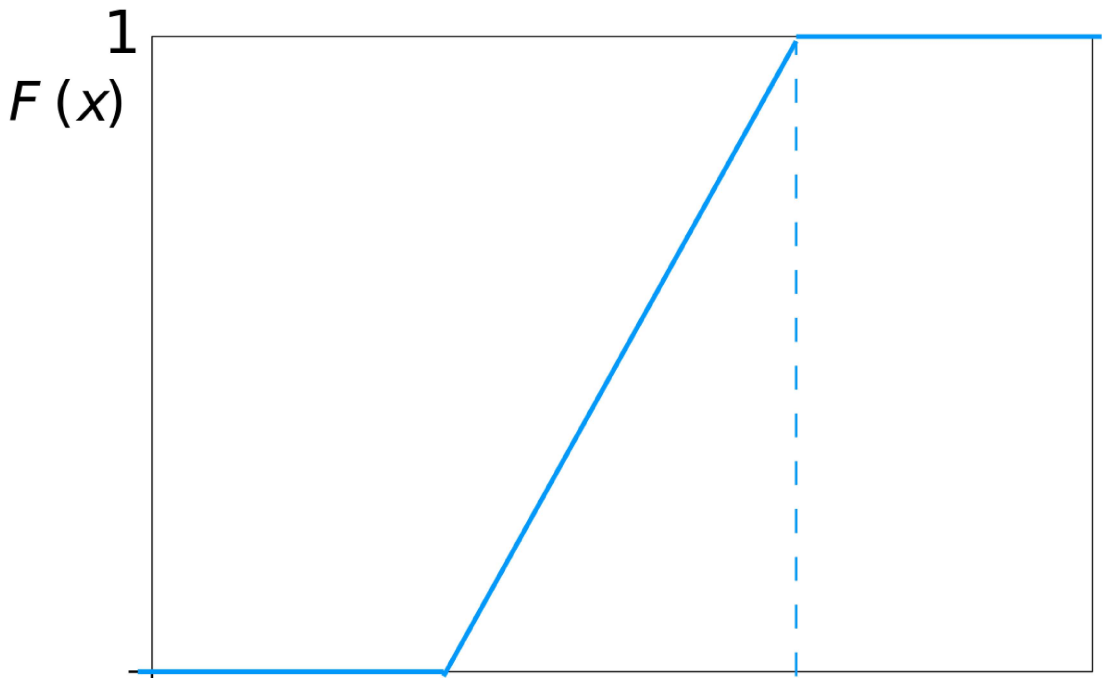
PDF & CDF Graphs for Uniform Distribution

*** Probability Distribution Function (CDF)**





*** Cumulative Distribution Function (CDF)**



https://en.wikipedia.org/wiki/Continuous_uniform_distribution (https://en.wikipedia.org/wiki/Continuous_uniform_distribution)

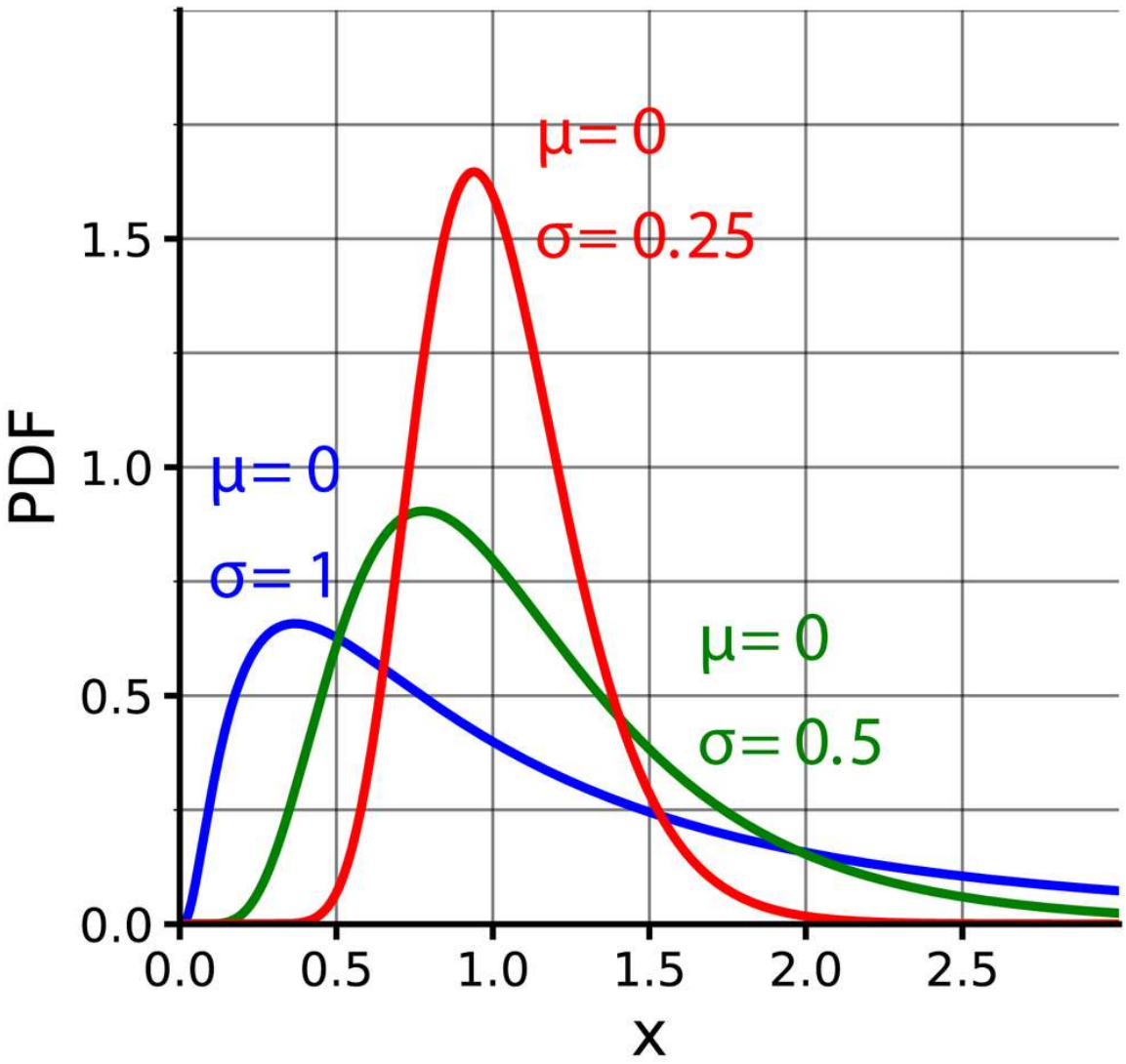
Skewness = O in Continous Uniform Distribution(Symmetric-Normal)

Application in Machine learning and Data Science

- **Random initialization**: In many machine learning algorithms, such as **neural networks and k-means clustering**, the initial values of the parameters can have a significant impact on the final result. Uniform distribution is often used to **randomly initialize the parameters**, as it ensures that all values in the range have an equal probability of being selected
- **Sampling**: Uniform distribution can also be used for sampling. For example, if you have a dataset with an **equal number of samples from each class**, you can use uniform distribution to **randomly** select a subset of the data that is representative of all the classes.
- **Data augmentation**: In some cases, you may want to artificially increase the size of your dataset by **generating new examples** that are similar to original data. Uniform distribution can be used to generate new data points that are within a specified range of the original data
- **Hyperparameter tuning**: Uniform distribution can also be used in hyperparameter tuning, where you need to search for the best combination of hyperparameters for a machine learning model. By defining a uniform prior distribution for each hyperparameter, you can sample from the distribution to explore the hyperparameter space

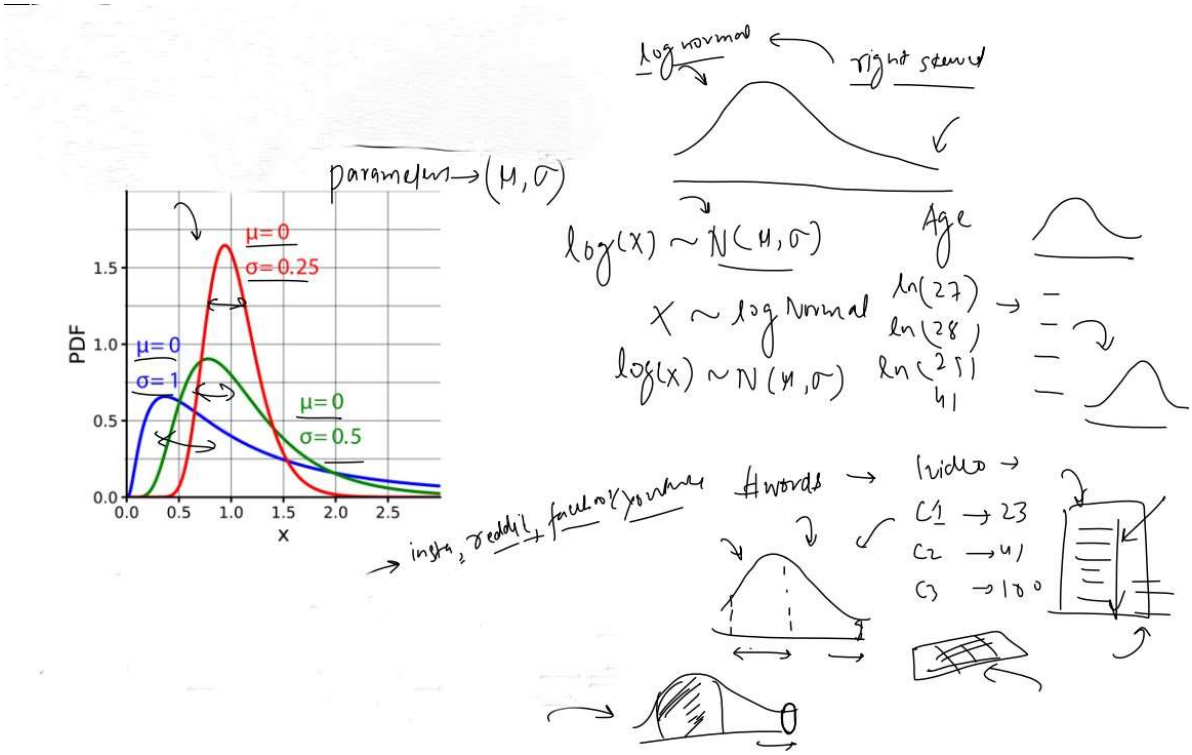
Log Normal Distribution

In probability theory and statistics, a lognormal distribution is a **heavy tailed continuous probability distribution** of a random variable whose logarithm is normally distributed.



Examples

- The length of comments posted in Internet discussion forums follows a log-normal distribution.
- Users dwell time on online articles (jokes, news etc.) follows a log-normal distribution.
- The length of chess games tends to follow a log-normal distribution.In economics, there is evidence that the income of 97%–99% of the population is distributed log-normally.



Formula

Log-Normal Distribution Statistics[1]

Notation	$\ln N\left(\mu, \sigma^2\right)$
Parameter	$-\infty < \mu < \infty$ $\sigma^2 > 0$
Distribution	$x > 0$
Pdf	$\frac{1}{\sqrt{2\pi}\sigma x}\exp\left\{-\frac{\left(\ln\left(x\right)-\mu\right)^2}{2\sigma^2}\right\}$
Cdf	$\frac{1}{2}\left[1+\operatorname{erf}\left(\frac{\ln\left(x-\mu\right)}{\sigma}\right)\right]$

Q: How to check if a random variable is log normally distributed?

- Quantile-Quantile (Q-Q) plot:

Create a Q-Q plot of the transformed data. If the points approximately lie on a straight line, it suggests that the data follows a log-normal distribution.

Pareto Distribution

The Pareto distribution is a type of probability distribution that is commonly used to model the distribution of wealth, income, and other quantities that exhibit a similar **power-law behaviour**

What is Power Law

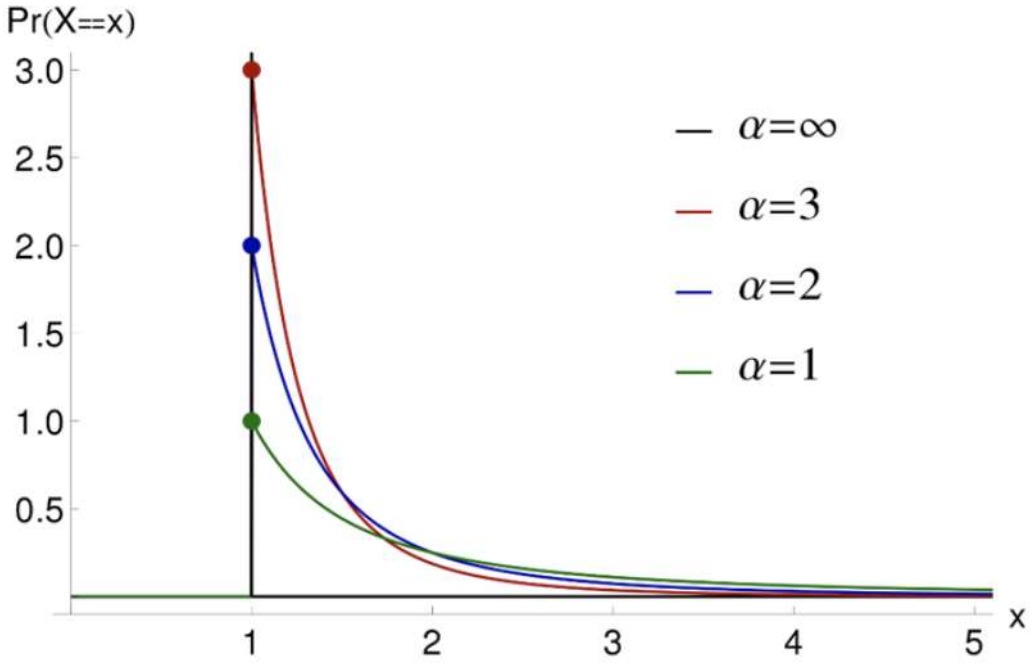
In mathematics, a power law is a functional relationship between two variables, where one variable is proportional to a power of the other. Specifically, if y and x are two variables related by a power law, then the relationship can be written as

```
y = k * x^a
```

The Long Tail offers a great contemporary example of the Pareto probability distribution – a few extreme events or “blockbusters” on the left hand side of the curve and a very long tail of much less popular events on the right hand side of the curve. The Pareto distribution has also been popularized as **the “80/20” rule**.

Vilfredo Pareto originally used this distribution to describe the allocation of wealth among individuals since it seemed to show rather well the way that a larger portion of the wealth of any society is owned by a smaller percentage of the people in that society. He also used it to describe distribution of income. This idea is sometimes expressed more simply as the Pareto principle or the "80-20 rule" **which says that 20% of the population controls 80% of the wealth**

Figure 1: Pareto Distribution (various alpha)



The Pareto distribution is a power-law probability distribution, and has only two parameters to describe the distribution: **α ("alpha") and Xm**. The α value is the shape parameter of the distribution, which determines **how distribution is sloped **(see Figure 1)**. The Xm parameter is the scale parameter, which represents the minimum possible value for the distribution and helps to determine the distribution's spread. The probability density function is given by the following formula:

$$f_X(x) = \frac{\alpha {X_m}^{\alpha}}{x^{\alpha+1}} \qquad x \geq X_m$$

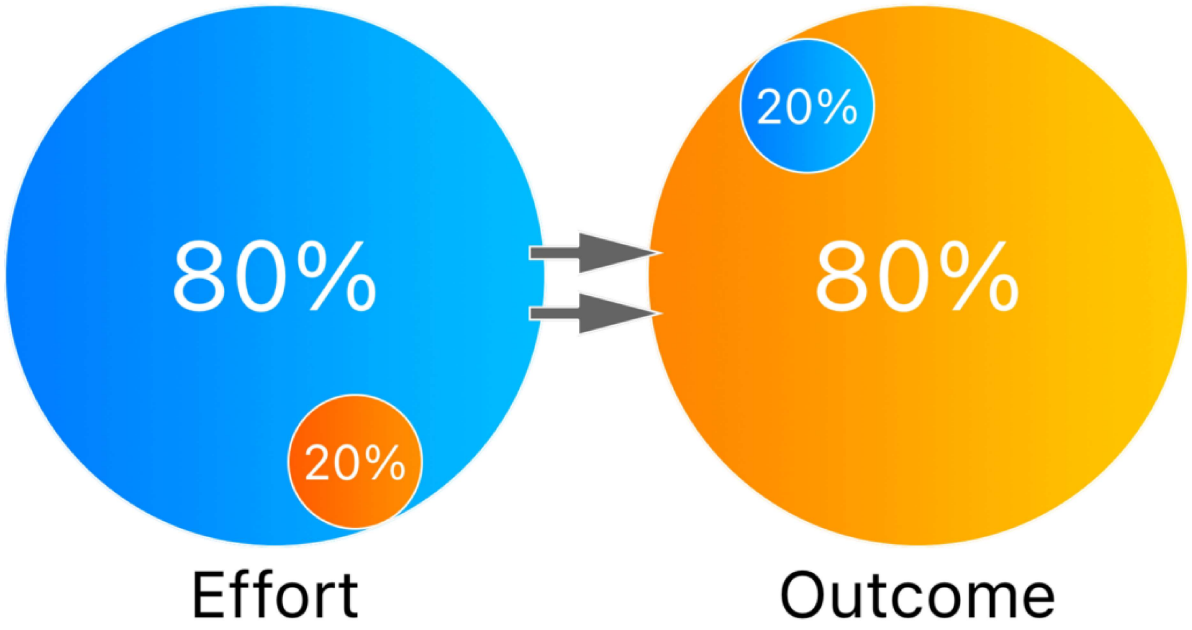
When we plot this function across a range of x values, we see that the distribution slopes downward as x increases. This means that the majority of the distribution's density is concentrated near Xm on the left-hand side, with only a small proportion of the density as we move to the right. For reference, the **"80-20 Rule" is represented by a distribution with alpha equal to approximately 1.16.**

Examples

- The sizes of human settlements (few cities, many hamlets/villages)

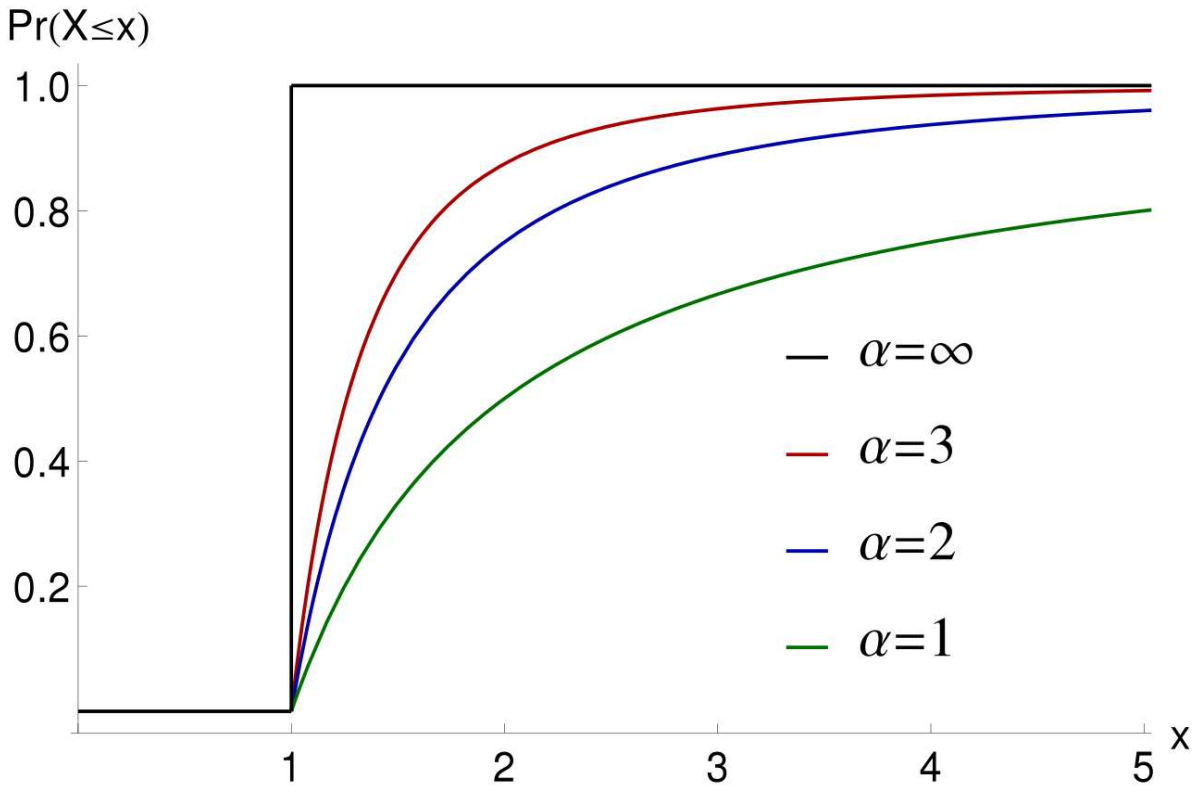
- File size distribution of Internet traffic which uses the TCP protocol (many smaller files, few larger ones)

- **Sales**: In fact, there are companies where only 20% of the salespeople are responsible for 80% of the sales.
- **Warehouses**: It is not uncommon for 20% of the products to take up 80% of the available space.
- **Productivity**: With proper prioritization, as little as 20% of all efforts can often get 80% of the work done.
- **Internet**: 80% of all online traffic is concentrated on 20% of websites.
- **Traffic**: 20% of roads carry 80% of the traffic load.
- **Time Management**: With 20% of the time (properly) spent, 80% of the tasks can be completed.



The Pareto Principle states that 80% of the results are achieved with 20% of the total effort. The remaining 20% of the results require the most work with 80% of the total effort.

CDF:



Cumulative distribution function [edit]

From the definition, the cumulative distribution function of a Pareto random variable with parameters α and x_m is

$$F_X(x) = \begin{cases} 1 - \left(\frac{x_m}{x}\right)^\alpha & x \geq x_m, \\ 0 & x < x_m. \end{cases}$$

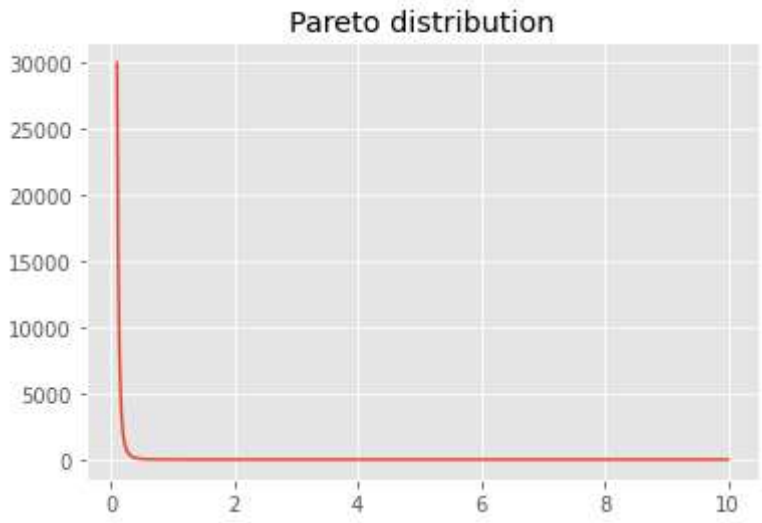
Parameters	$x_{\mathrm{m}} > 0$ scale (real) $\alpha > 0$ shape (real)
Support	$x \in [x_{\mathrm{m}}, \infty)$
PDF	$\frac{\alpha x_{\mathrm{m}}^{\alpha}}{x^{\alpha+1}}$
CDF	$1 - \left(\frac{x_{\mathrm{m}}}{x}\right)^{\alpha}$
Quantile	$x_{\mathrm{m}}(1-p)^{-\frac{1}{\alpha}}$
Mean	$\begin{cases} \infty & \text{for } \alpha \leq 1 \\ \frac{\alpha x_{\mathrm{m}}}{\alpha - 1} & \text{for } \alpha > 1 \end{cases}$
Median	$x_{\mathrm{m}} \sqrt[\alpha]{2}$
Mode	x_{m}

How to detect if a distribution is Pareto Distribution?

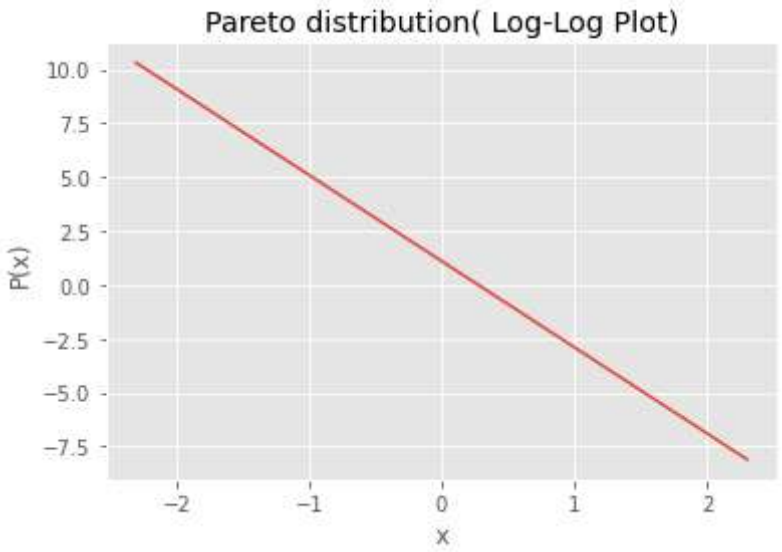
- Log -Log Plot
- QQ Plot

In [85]:
```
# Log Log Plot

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

# Define the parameters of the Pareto distribution
alpha = 3
xm = 1

# Create an array of x values
x = np.linspace(0.1, 10, 1000)

# Calculate the y values of the Pareto distribution
y = alpha * (xm**alpha) / (x**(alpha+1))

plt.title('Pareto distribution')

plt.plot(x,y) #
```

Out[85]:
```
[<matplotlib.lines.Line2D at 0x1e289121040>]
```



In [86]:
```
# Create the log-log plot
plt.plot(np.log(x),np.log(y))

# Add labels and a title
plt.xlabel('x')
plt.ylabel('P(x)')
plt.title('Pareto distribution( Log-Log Plot)')

# Show the plot
plt.show()
```

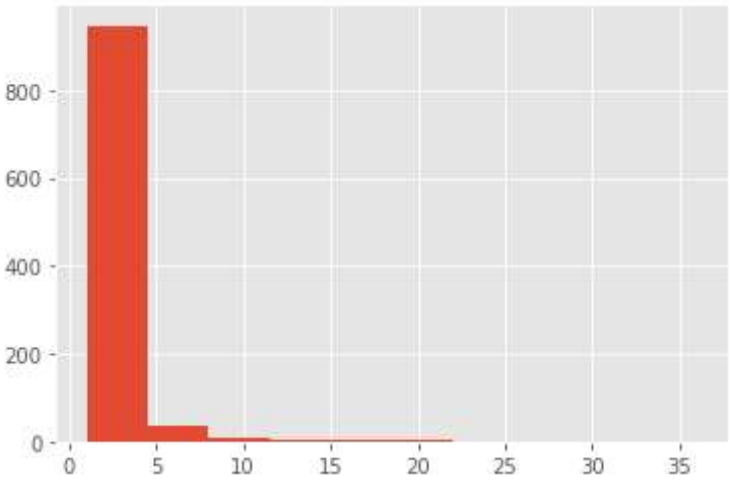


In [87]:
```
import statsmodels.api as sm
```

In [88]:
```
# Generate random data

# Define the parameters of the Pareto distribution
alpha = 2
xm = 1

# Generate a set of random data from the Pareto distribution
x = stats.pareto.rvs(b=alpha, scale=xm, size=1000)
```

In [89]:
```
plt.hist(x)
```

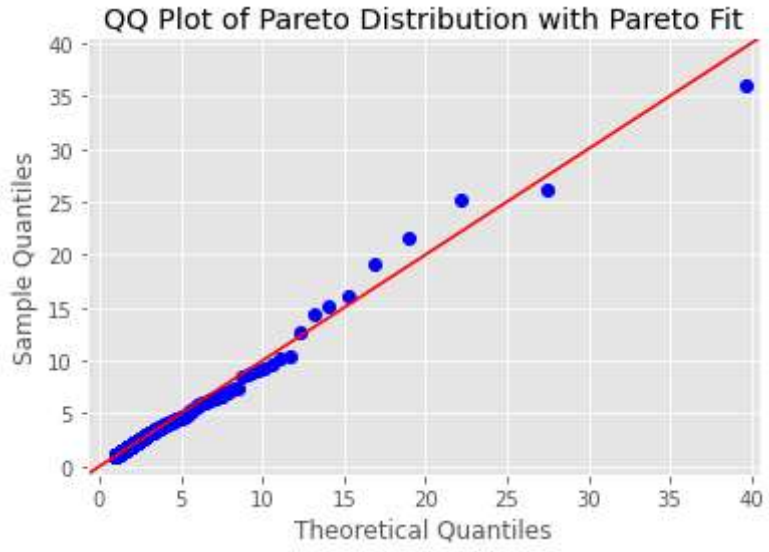
Out[89]:
```
(array([947.,  36.,   8.,   2.,   2.,   2.,   1.,   1.,   0.,   1.]),
 array([ 1.00049542,  4.50522233,  8.00994925, 11.51467616, 15.01940307,
        18.52412999, 22.0288569 , 25.53358381, 29.03831073, 32.54303764,
        36.04776455]),
 <BarContainer object of 10 artists>)
```



In [90]:
```
# Fit a Pareto distribution to the data
params = stats.pareto.fit(x, floc=0)

# Create a Pareto distribution object with the fitted parameters
dist = stats.pareto(b=params[0], scale=params[2])
```

In [91]:
```
# Create a QQ plot of the data using the Pareto distribution

fig = sm.qqplot(x, dist=dist, line='45')

# Add a title and labels to the plot

plt.title('QQ Plot of Pareto Distribution with Pareto Fit')
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Sample Quantiles')

# Show the plot

plt.show()
```

```
C:\Users\user\anaconda3\lib\site-packages\statsmodels\graphics\gofplots.py:993: UserWarning: marker is redundantly defined by the 'marker' keyword argument and the fmt string "bo" (-> marker='o'). The keyword argument will take precedence.
  ax.plot(x, y, fmt, **plot_style)
```



Transformations / Feature Transformations

In sklearn

- Function Transformer

1. Log transform
2. reciprocal
3. square/square_root
4. custom

- Power Transformer

1. boxcox
2. Yeo-Johnson

LOG TRANSFORMATION:

- Generally, these transformations make our data close to a normal distribution but are not able to exactly abide by a normal distribution.
- This transformation is not applied to those features which have negative values.
- This transformation is mostly applied to right-skewed data.
- Convert data from addictive Scale to multiplicative scale i,e, linearly distributed data.

Example

In [92]:
```
import pandas as pd
import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

from sklearn.preprocessing import FunctionTransformer
from sklearn.compose import ColumnTransformer
```

In [93]:
```
df = titanic[['Age','Fare','Survived']]
```

In [94]:
```
df.head()
```

Out[94]:

	Age	Fare	Survived
0	22.0	7.2500	0
1	38.0	71.2833	1
2	26.0	7.9250	1
3	35.0	53.1000	1
4	35.0	8.0500	0

In [95]:
```
# missing data

df.isnull().sum()
```

Out[95]:
```
Age         177
Fare          0
Survived      0
dtype: int64
```

In [96]:
```
# replace missing values to 'mean'

df['Age'].fillna(df['Age'].mean(),inplace=True)
```

```
C:\Users\user\anaconda3\lib\site-packages\pandas\core\generic.py:6392: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  return self._update_inplace(result)
```

In [97]:
```
df.isnull().sum()
```

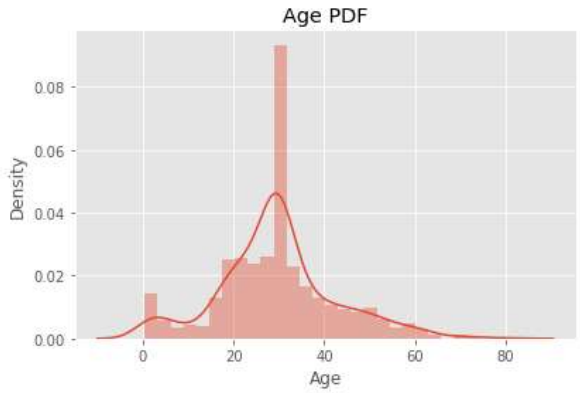
Out[97]:
```
Age         0
Fare        0
Survived    0
dtype: int64
```

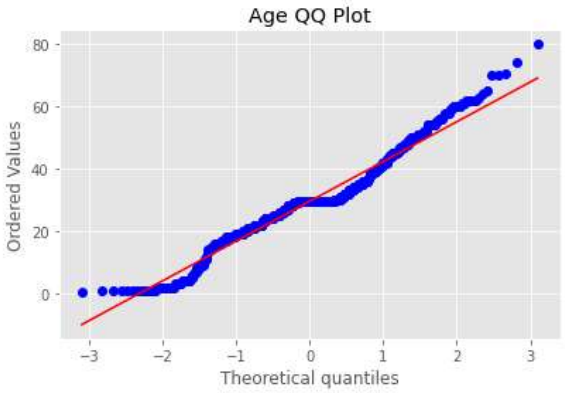
In [98]:
```
# divide data in to two parts

#X = df.iloc[:,1:3] # Age , fare (input)
#y = df.iloc[:,0] # survived (output)

X = df[['Age', 'Fare']]
y = df['Survived']
```

In [99]:
```
# train_test split

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_s
```

In [100]:
```
# Distribution of 'Age' column

plt.figure(figsize=(14,4))

plt.subplot(121)

sns.distplot(X_train['Age'])

plt.title('Age PDF')


# QQ Plot

plt.subplot(122)

stats.probplot(X_train['Age'], dist="norm", plot=plt)

plt.title('Age QQ Plot')

plt.show()
```

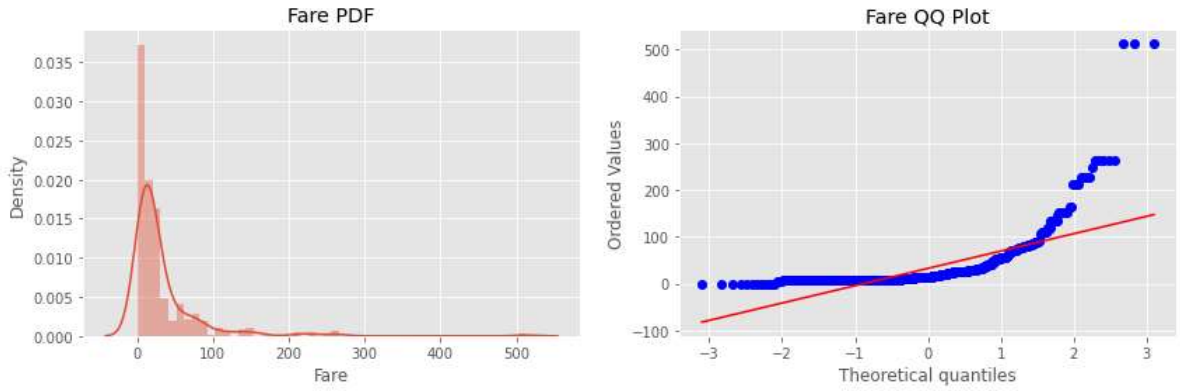
```
C:\Users\user\anaconda3\lib\site-packages\seaborn\distributions.py:2619: Futu
reWarning: `distplot` is a deprecated function and will be removed in a futur
e version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for his
tograms).
  warnings.warn(msg, FutureWarning)
```





In [101]:
```
# Distribution of 'Fare' column

plt.figure(figsize=(14,4))

plt.subplot(121)

sns.distplot(X_train['Fare'])

plt.title('Fare PDF')

# QQ plot

plt.subplot(122)
stats.probplot(X_train['Fare'], dist="norm", plot=plt)
plt.title('Fare QQ Plot')

plt.show()
```

```
C:\Users\user\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [102]:
```
# Modelling

clf = LogisticRegression()

clf2 = DecisionTreeClassifier()
```

In [103]:
```
# Train ,fit ,predict

clf.fit(X_train,y_train)
clf2.fit(X_train,y_train)

y_pred = clf.predict(X_test)
y_pred1 = clf2.predict(X_test)

print("Accuracy LR",accuracy_score(y_test,y_pred))
print("Accuracy DT",accuracy_score(y_test,y_pred1))
```

```
Accuracy LR 0.6480446927374302
Accuracy DT 0.6871508379888268
```

Adding log(Log1p) Transform

What is the difference between NP log and NP log1p?

- np. LOG is only used when we have to take simple log. If we have zeros (0's) in our data so the model will be not that much accurate hence here we use NP. LOG1P and it turns zeros value to x+1 then no values in the data will be zero.

In [104]:
```
trf = FunctionTransformer(func=np.log1p)
```

In [105]:
```
# Build again with transformed model

X_train_transformed = trf.fit_transform(X_train)
X_test_transformed = trf.transform(X_test)
```

In [106]:
```
# Train and test with transformed model

clf = LogisticRegression()
clf2 = DecisionTreeClassifier()

clf.fit(X_train_transformed,y_train) # train
clf2.fit(X_train_transformed,y_train)

y_pred = clf.predict(X_test_transformed) #test
y_pred1 = clf2.predict(X_test_transformed)

print("Accuracy LR",accuracy_score(y_test,y_pred)) # pred
print("Accuracy DT",accuracy_score(y_test,y_pred1))
```

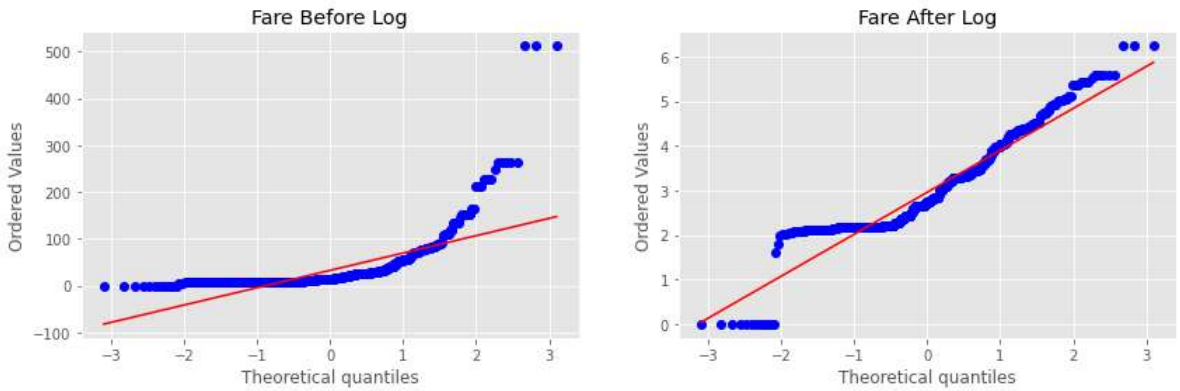
```
Accuracy LR 0.6815642458100558
Accuracy DT 0.6815642458100558
```

In [107]:
```
# Check the Accuracy with 'Cross Validation'

X_transformed = trf.fit_transform(X)

clf = LogisticRegression()
clf2 = DecisionTreeClassifier()

print("LR",np.mean(cross_val_score(clf,
                                    X_transformed,
                                    y,
                                    scoring='accuracy',
                                    cv=10)))


print("DT",np.mean(cross_val_score(clf2,
                                    X_transformed,
                                    y,
                                    scoring='accuracy',
                                    cv=10)))
```

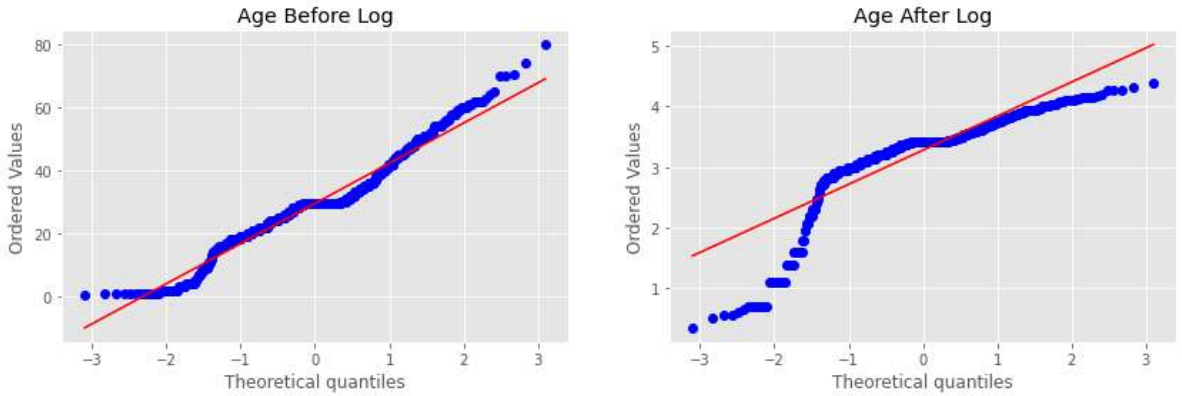
```
LR 0.678027465667915
DT 0.6622471910112359
```

Plot the difference After applying Transformation

In [108]:
```
# Fare Column

plt.figure(figsize=(14,4))

plt.subplot(121) # is used to create a subplot within a larger figure.

stats.probplot(X_train['Fare'], dist="norm", plot=plt)
plt.title('Fare Before Log')

plt.subplot(122)
stats.probplot(X_train_transformed['Fare'], dist="norm", plot=plt)
plt.title('Fare After Log')

plt.show()
```



In [109]:
```
# Age column

plt.figure(figsize=(14,4))

plt.subplot(121)
stats.probplot(X_train['Age'], dist="norm", plot=plt)
plt.title('Age Before Log')

plt.subplot(122)
stats.probplot(X_train_transformed['Age'], dist="norm", plot=plt)
plt.title('Age After Log')

plt.show()
```



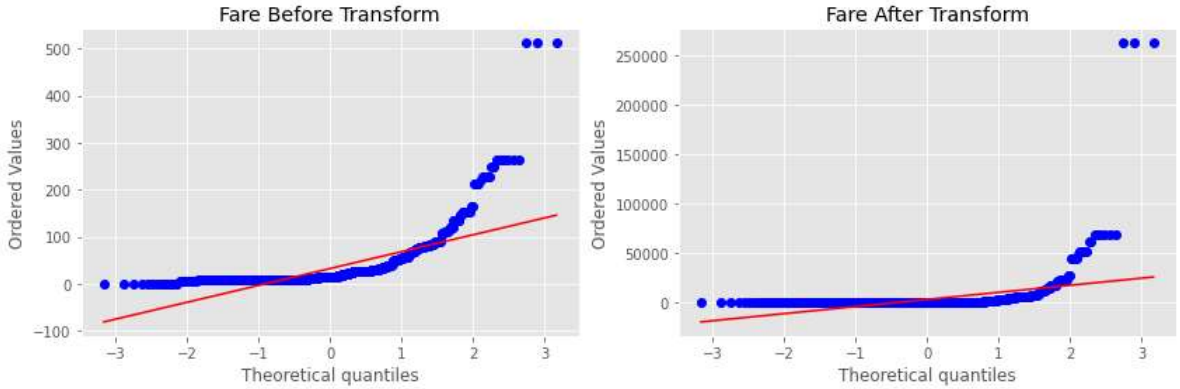
Observation: Here after applying Log transformation to the 'Age' column , it didnt perform well and gives bad results

- The application of a logarithmic transformation to the 'Age' column did not yield favorable results. It is possible that the 'Age' column was already normally distributed,
- while another column, such as 'Fare', exhibited right skewness. This suggests that the logarithmic transformation was unnecessary for the 'Age' column, potentially leading to undesirable changes in the data. Consider the initial distribution and specific requirements of the analysis when deciding on the appropriate data transformation technique.

In [110]:
```
def apply_transform(transform):
    X = df[['Age', 'Fare']]
    y = df['Survived']


    trf = ColumnTransformer([('log',
                              FunctionTransformer(transform),
                              ['Fare'])],
                            remainder='passthrough')



    X_trans = trf.fit_transform(X)


    clf = LogisticRegression()

    print("Accuracy",np.mean(cross_val_score(clf,X_trans,y,
                                              scoring='accuracy',
                                              cv=10)))

    plt.figure(figsize=(14,4))

    plt.subplot(121)
    stats.probplot(X['Fare'], dist="norm", plot=plt)
    plt.title('Fare Before Transform')

    plt.subplot(122)
    stats.probplot(X_trans[:,0], dist="norm", plot=plt)
    plt.title('Fare After Transform')

    plt.show()
```

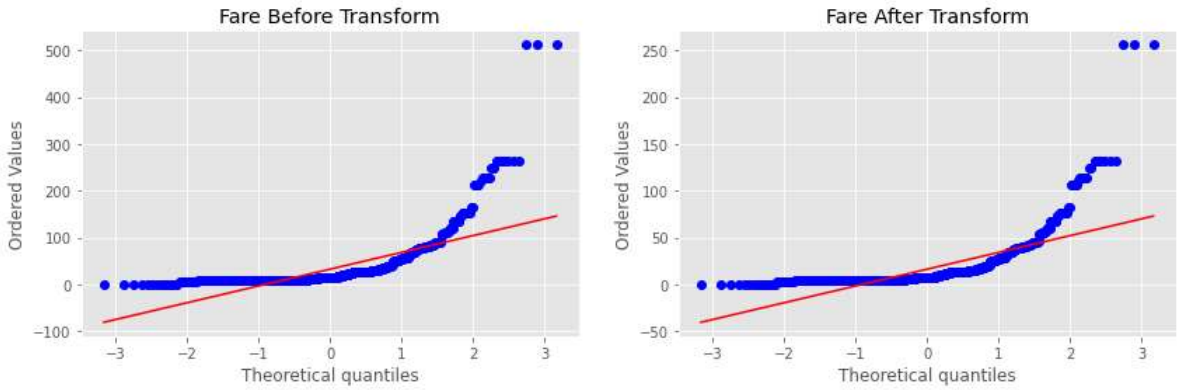
In [111]:
```
# sqaure

apply_transform(lambda x:x**2)
```

```
Accuracy 0.6442446941323345
```



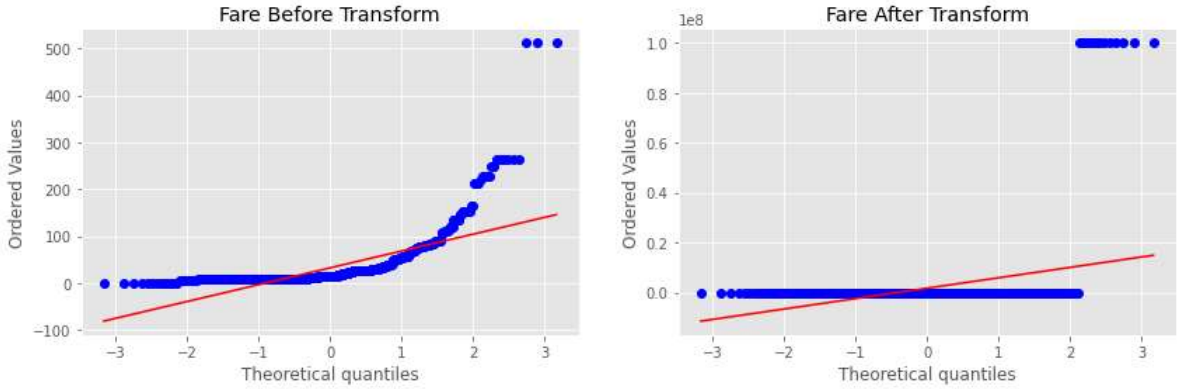
In [112]:
```
# Square_root

apply_transform(lambda x:x**1/2)
```

```
Accuracy 0.6589013732833957
```



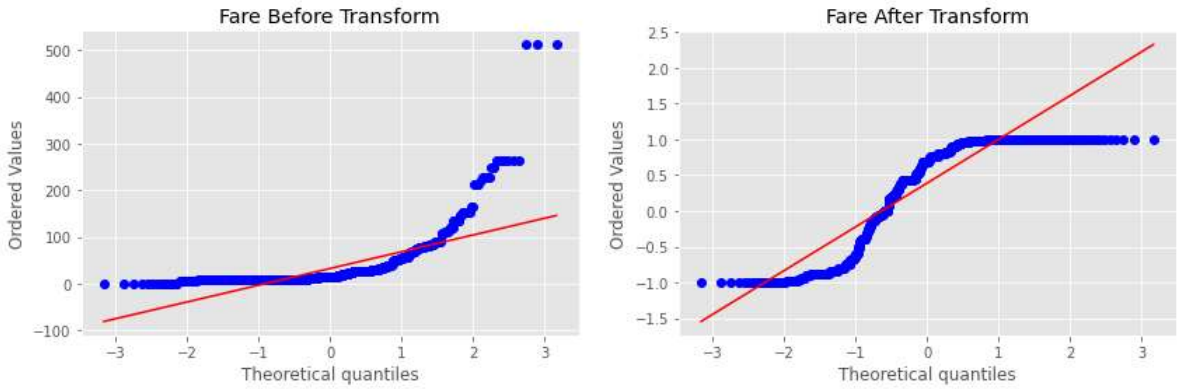
In [113]:
```
# Reciprocal

apply_transform(lambda x:1/(x+0.00000001))
```

```
Accuracy 0.61729088639201
```



In [114]:
```
# Custom logic

apply_transform(np.sin)
```

```
Accuracy 0.6195131086142323
```



Power Transformers

- Box-cox Transformer
- Yeo-Johnson Transformer

Box-Cox Transformation: The Box-Cox transformation is a more flexible power transformation that encompasses both the square root and logarithmic transformations. It introduces a parameter **lambda (λ)** that determines the type and degree of transformation applied to the data. By estimating the optimal lambda value, the Box-Cox transformation can adapt to the specific characteristics of the data.

$$y_i^{(\lambda)} = \begin{cases} \dfrac{y_i^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0, \\ \ln(y_i) & \text{if } \lambda = 0, \end{cases}$$

λ	Transformed Data
-2	y^{-2}
-1	y^{-1}
-0.5	1/√y
0	ln(y)
0.5	√y
1	y
2	y^2

Yeo-Johnson Transformation: Similar to the Box-Cox transformation, the Yeo-Johnson transformation is another power transformation that can **handle both positive and negative values**. It provides a more flexible approach by incorporating a parameter to control the transformation.

$$\psi(y,\lambda) = \begin{cases} \dfrac{(y+1)^\lambda - 1}{\lambda} & y \geq 0 \text{ and } \lambda \neq 0, \\ \log(y+1) & y \geq 0 \text{ and } \lambda = 0, \\ -\dfrac{(-y+1)^{2-\lambda} - 1}{2-\lambda} & y < 0 \text{ and } \lambda \neq 2, \\ -\log(-y+1) & y < 0, \lambda = 2. \end{cases}$$

In [115]:
```
from sklearn.preprocessing import PowerTransformer
```

In [116]:
```
from sklearn.metrics import r2_score
```

In [117]:
```
df =pd.read_csv("concrete_data.csv")
```

In [118]:
```
df.head()
```

Out[118]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Strength
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.99
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.89
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.27
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.05
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.30

In [119]:
```
df.shape
```

Out[119]:
```
(1030, 9)
```

In [120]:
```
df.isnull().sum()
```

Out[120]:
```
Cement                0
Blast Furnace Slag    0
Fly Ash               0
Water                 0
Superplasticizer      0
Coarse Aggregate      0
Fine Aggregate        0
Age                   0
Strength              0
dtype: int64
```

check if any column as '0' values , because 'boxcox' will not work , if any '0'values present in data

In [121]:
```
df.describe()
```

Out[121]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Agg
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.(
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972.918932	773.5
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.753954	80.1
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.(
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.9
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.5
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.(
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.6

In [122]:
```
X = df.drop(columns=['Strength'])
y = df.iloc[:,-1]
```

In [123]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,
                                                    test_size=0.2,
                                                    random_state=42)
```

In [124]:
```
# Applying Regression without any transformation

from sklearn.linear_model import LinearRegression

lr = LinearRegression()

lr.fit(X_train,y_train)

y_pred = lr.predict(X_test)

r2_score(y_test,y_pred)
```

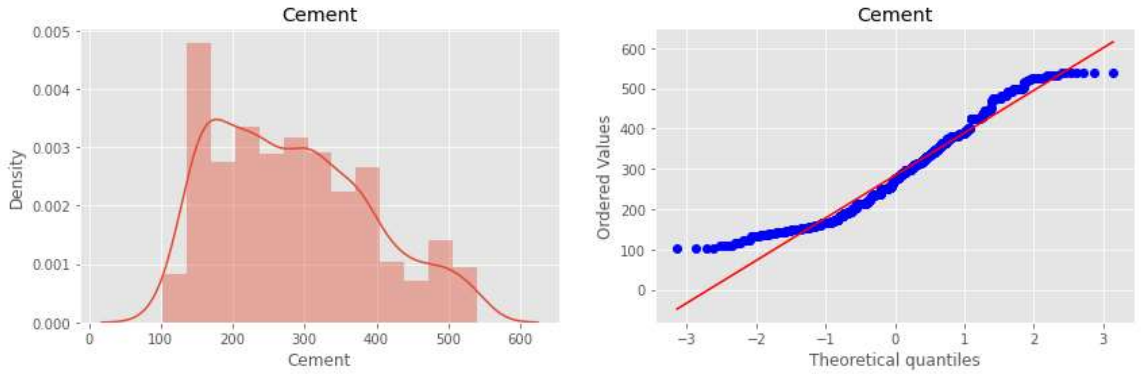
Out[124]:
```
0.627553179231485
```

In [125]:
```
# Cross checking with cross val score

lr = LinearRegression()

np.mean(cross_val_score(lr,X,y,scoring='r2'))
```

Out[125]:
```
0.4609940491662866
```

In [126]:
```
# Plotting the distplots without any transformation , for every column

for col in X_train.columns:
    plt.figure(figsize=(14,4))
    plt.subplot(121)
    sns.distplot(X_train[col])
    plt.title(col)

    plt.subplot(122)
    stats.probplot(X_train[col], dist="norm", plot=plt)
    plt.title(col)

    plt.show()
```

```
C:\Users\user\anaconda3\lib\site-packages\seaborn\distributions.py:2619: F
utureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-l
evel function with similar flexibility) or `histplot` (an axes-level funct
ion for histograms).
  warnings.warn(msg, FutureWarning)
```




```
C:\Users\user\anaconda3\lib\site-packages\seaborn\distributions.py:2619: F
```

Box-Cox Transform

In [127]:
```
# Applying Box-Cox Transform

pt = PowerTransformer(method='box-cox')

X_train_transformed = pt.fit_transform(X_train+0.000001) # to replace '0',we a
X_test_transformed = pt.transform(X_test+0.000001)

pd.DataFrame({'cols':X_train.columns,'box_cox_lambdas':pt.lambdas_})
```

Out[127]:

	cols	box_cox_lambdas
0	Cement	0.177025
1	Blast Furnace Slag	0.025093
2	Fly Ash	-0.038970
3	Water	0.772682
4	Superplasticizer	0.098811
5	Coarse Aggregate	1.129813
6	Fine Aggregate	1.782019
7	Age	0.066631

In [128]:
```
# Applying linear regression on transformed data

lr = LinearRegression()
lr.fit(X_train_transformed,y_train)

y_pred2 = lr.predict(X_test_transformed)

r2_score(y_test,y_pred2)
```

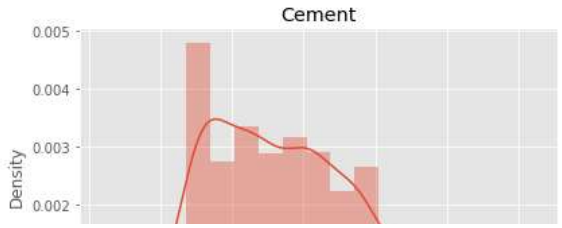
Out[128]:
```
0.8047825006181187
```

In [129]:
```
 # Using Cross validation score

pt = PowerTransformer(method='box-cox')
X_transformed = pt.fit_transform(X+0.0000001)

lr = LinearRegression()
np.mean(cross_val_score(lr,X_transformed,y,scoring='r2'))
```

Out[129]:
```
0.6658537942219862
```

In [130]:
```
# Before and after comparision for Box-Cox Plot

X_train_transformed = pd.DataFrame(X_train_transformed,
                                   columns=X_train.columns)

for col in X_train_transformed.columns:
    plt.figure(figsize=(14,4))
    plt.subplot(121)
    sns.distplot(X_train[col])
    plt.title(col)

    plt.subplot(122)
    sns.distplot(X_train_transformed[col])
    plt.title(col)

    plt.show()
```

```
C:\Users\user\anaconda3\lib\site-packages\seaborn\distributions.py:2619: F
utureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-l
evel function with similar flexibility) or `histplot` (an axes-level funct
ion for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\user\anaconda3\lib\site-packages\seaborn\distributions.py:2619: F
utureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-l
evel function with similar flexibility) or `histplot` (an axes-level funct
ion for histograms).
  warnings.warn(msg, FutureWarning)
```



Yeo-Johnson transform

In [131]:
```
# Apply Yeo-Johnson transform

pt1 = PowerTransformer()

X_train_transformed2 = pt1.fit_transform(X_train)
X_test_transformed2 = pt1.transform(X_test)

lr = LinearRegression()
lr.fit(X_train_transformed2,y_train)

y_pred3 = lr.predict(X_test_transformed2)

print(r2_score(y_test,y_pred3))

pd.DataFrame({'cols':X_train.columns,'Yeo_Johnson_lambdas':pt1.lambdas_})
```

```
0.8161906513339305
```

Out[131]:

	cols	Yeo_Johnson_lambdas
0	Cement	0.174348
1	Blast Furnace Slag	0.015715
2	Fly Ash	-0.161447
3	Water	0.771307
4	Superplasticizer	0.253935
5	Coarse Aggregate	1.130050
6	Fine Aggregate	1.783100
7	Age	0.019885

In [132]:
```
# applying cross val score

pt = PowerTransformer()
X_transformed2 = pt.fit_transform(X)

lr = LinearRegression()
np.mean(cross_val_score(lr,X_transformed2,y,scoring='r2'))
```

Out[132]:
```
0.6834625134285743
```

In [133]:
```
X_train_transformed2 = pd.DataFrame(X_train_transformed2,
                                    columns=X_train.columns)
```

In [134]:
```
# Before and after comparision for Yeo-Johnson

for col in X_train_transformed2.columns:
    plt.figure(figsize=(14,4))
    plt.subplot(121)
    sns.distplot(X_train[col])
    plt.title(col)

    plt.subplot(122)
    sns.distplot(X_train_transformed2[col])
    plt.title(col)

    plt.show()
```

```
C:\Users\user\anaconda3\lib\site-packages\seaborn\distributions.py:2619: F
utureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-l
evel function with similar flexibility) or `histplot` (an axes-level funct
ion for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\user\anaconda3\lib\site-packages\seaborn\distributions.py:2619: F
utureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-l
evel function with similar flexibility) or `histplot` (an axes-level funct
ion for histograms).
  warnings.warn(msg, FutureWarning)
```

In [135]:
```
# Side by side Lambdas

pd.DataFrame({'cols':X_train.columns,'box_cox_lambdas':pt.lambdas_,
              'Yeo_Johnson_lambdas':pt1.lambdas_})
```

Out[135]:

	cols	box_cox_lambdas	Yeo_Johnson_lambdas
0	Cement	0.169544	0.174348
1	Blast Furnace Slag	0.016633	0.015715
2	Fly Ash	-0.136480	-0.161447
3	Water	0.808438	0.771307
4	Superplasticizer	0.264160	0.253935
5	Coarse Aggregate	1.129395	1.130050
6	Fine Aggregate	1.830763	1.783100
7	Age	0.001771	0.019885

Discrete Probability distributions

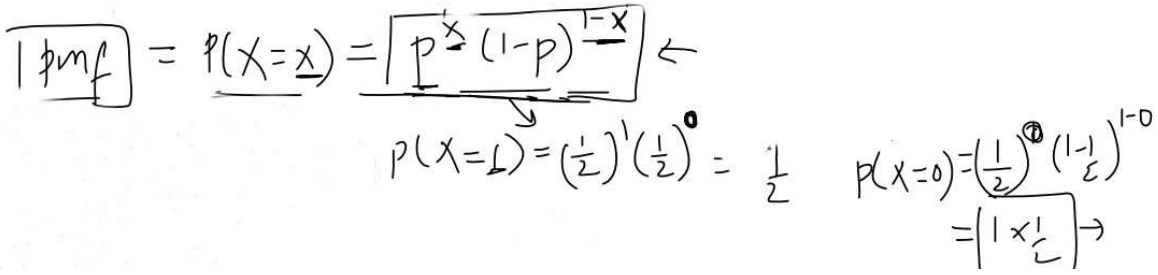
- Bernoulli Distributions
- Binomial Distributions

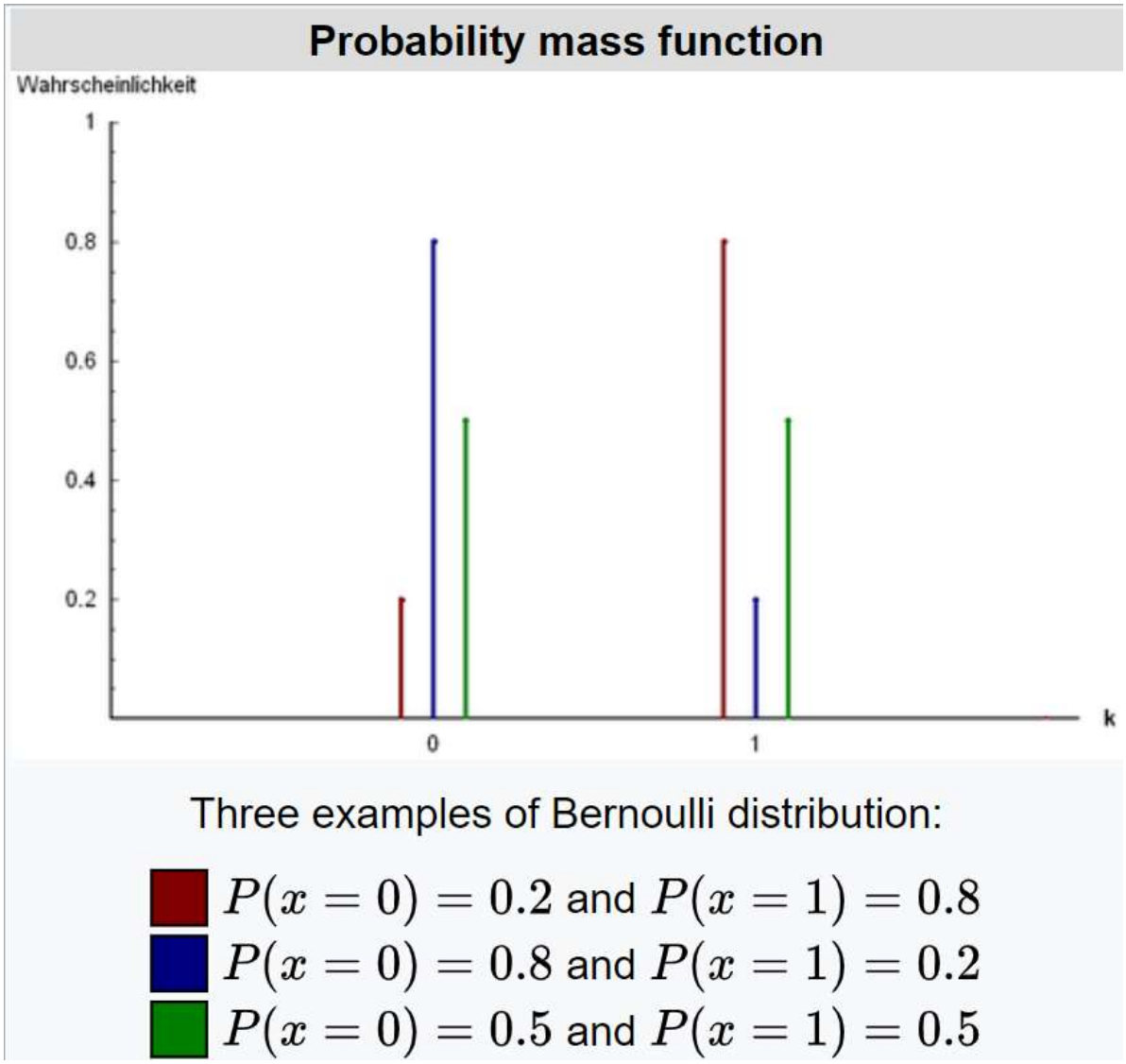
Bernoulli Distribution

it is a probability distribution that models a **binary outcome**, where the outcome can be either success (represented by the value 1) or failure (represented by the value 0). The Bernoulli distribution is named after the Swiss mathematician Jacob Bernoulli, who first introduced it in the late 1600s.

The Bernoulli distribution is characterized by a single parameter, which is the probability of success, denoted by **p**. The **probability mass function (PMF)** of the Bernoulli distribution is:

$$PMF = P(X = x) = px(1-p)1-x$$





The Bernoulli distribution is commonly used in **machine learning for modelling binary outcomes**, such as whether a customer will make a purchase or not, whether an email is spam or not, or whether a patient will have a certain disease or not.

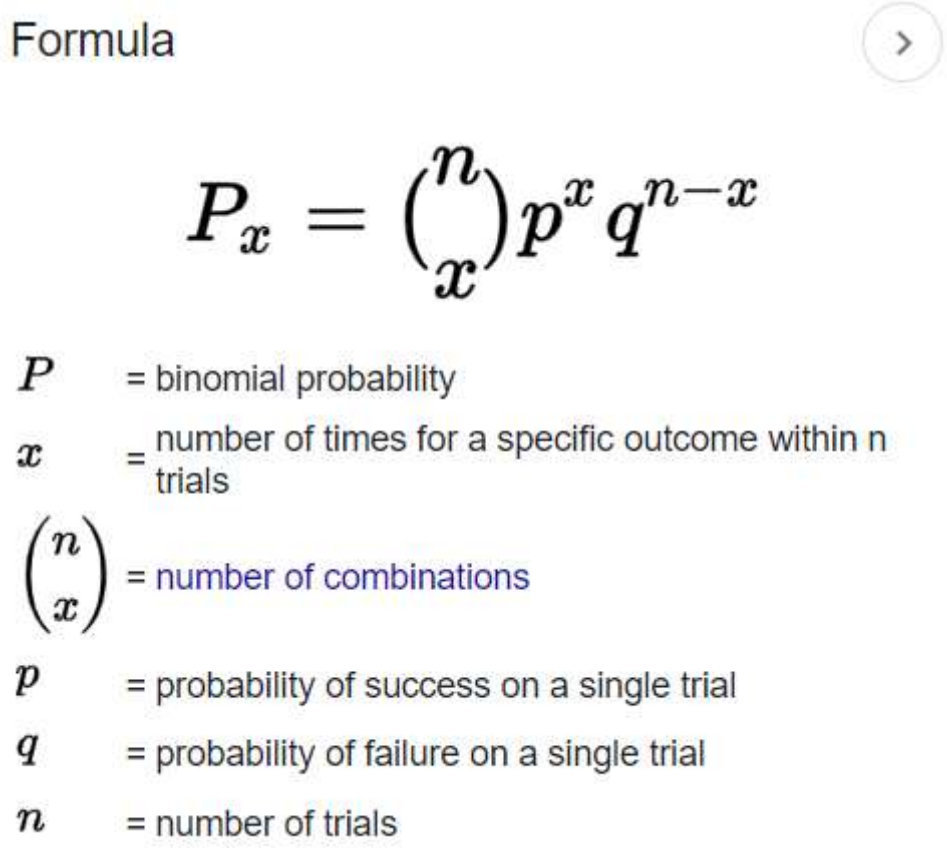
Binomial Distribution

Binomial distribution is a probability distribution that describes the number of successes in a fixed number of **independent** Bernoulli trials with ***two possible outcomes (often called "success" and "failure")**, where the probability of success is constant for each trial. The binomial distribution is characterized by two parameters: **the number of trials n and the probability of success p.**

```
EX: N=1 ( Bernoulli) , N=5 ( Binomial)
```

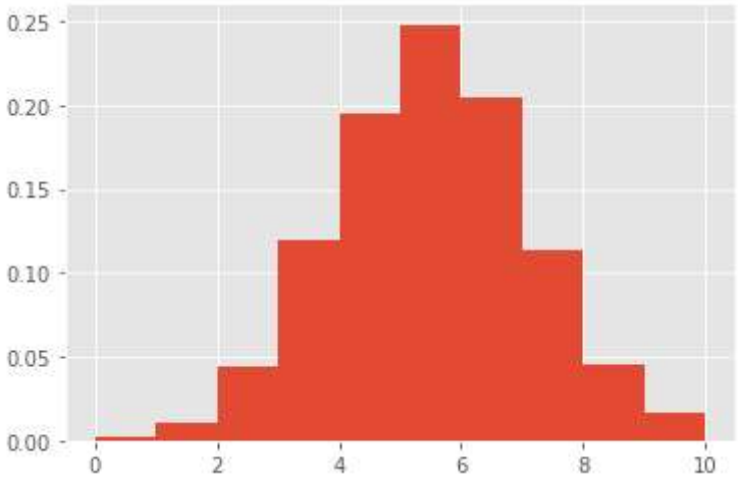
PDF Formula : P(x=x) = nCx px (1-p)n-x

- n - no.of trails
- p - probability of sucess
- x - desired result



Graph of PDF:

In [136]:
```
# code

n = 10  # number of trials
p = 0.5  # probability of success
size = 1000  # number of samples to generate

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

plt.hist(binomial_dist,density=True)
plt.show()
```

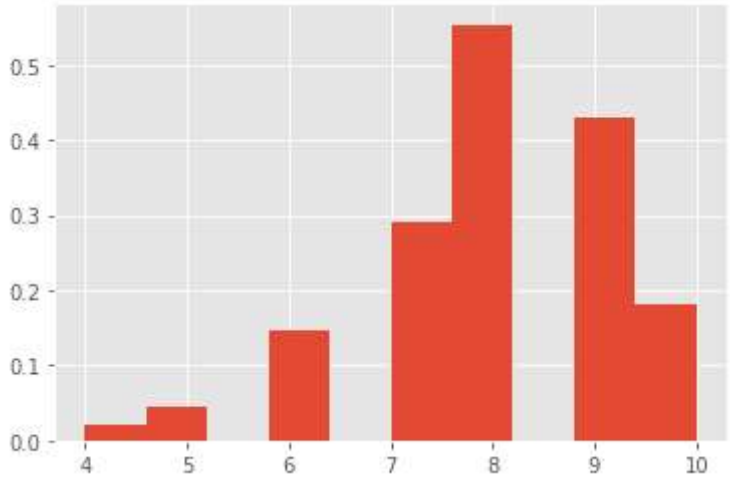


In [137]:
```
# Increased the probability

n = 10  # number of trials
p = 0.8  # probability of success
size = 1000  # number of samples to generate

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

binomial_dist
```

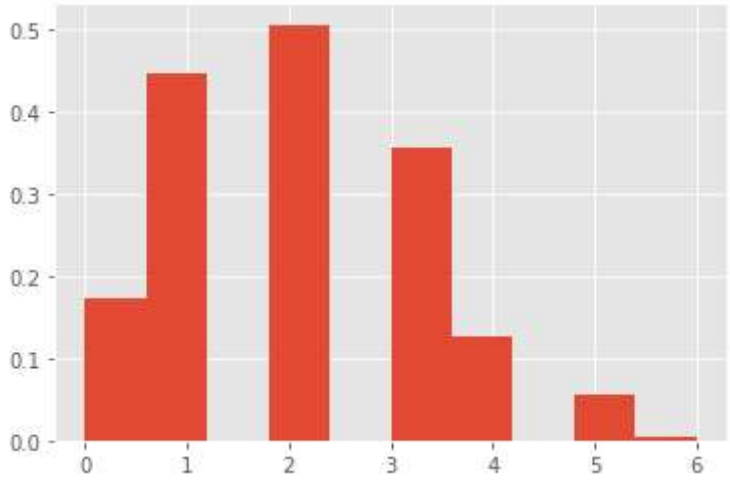
Out[137]:
```
array([ 8,  8,  9,  9,  9,  9, 10,  9, 10,  9,  8,  7,  6,  9,  8,  9,  9,
        7,  9,  9,  8,  6,  8,  8,  8,  8,  8,  7,  6,  7,  8, 10,  9,  9,
        9,  9,  8,  6,  8,  7,  9,  9, 10,  8, 10,  8,  7, 10,  7,  8,  9,
        9,  8,  8,  7,  9,  5,  6,  8,  8,  6,  8,  9,  9,  9,  6,  7,  7,
        8,  8,  9,  9, 10,  9, 10,  9, 10, 10,  7, 10,  9,  7,  9,  9,  6,
        9,  7,  8,  9,  8,  9,  8,  7,  8,  9,  8,  7,  9,  8,  7,  6,  8,
        9,  8,  9,  8,  9,  8,  8,  6,  7,  8,  8,  7, 10, 10,  6, 10,  7,
        6,  6,  8,  6,  8,  6,  7,  7,  7, 10,  8,  9,  8,  8, 10,  7,  8,
        8,  9,  6,  4,  7,  9,  8,  5,  7,  8,  9,  9,  9,  7,  8,  9,  8,
        9,  6,  9,  7,  8, 10,  9,  7,  9,  8,  6,  9,  8,  6,  9,  9,  7,
        8, 10,  7,  8, 10,  7,  8,  7,  5,  8,  9,  8,  8,  7,  8,  9, 10,
        7,  7,  7, 10,  8,  9,  8, 10,  7,  8,  7,  7,  8,  9,  9,  8,  8,
        9,  9,  9,  9, 10,  8, 10, 10,  4,  6,  8,  7,  9,  6,  8,  5,  8,
        7,  9,  9, 10,  9,  7,  7, 10,  8, 10,  9,  7,  9,  9,  8,  9,  7,
        8,  9,  3,  7,  7,  7,  8,  7,  9,  9, 10,  8,  9, 10,  8,  9,  6,
        7, 10,  5,  9,  8,  5,  6,  7,  7,  8, 10,  6,  9,  7,  8,  7,  7,
        7,  8,  7,  6,  7,  6,  8, 10,  7,  9,  7,  9,  7,  9,  7,  8,  8,
        8,  9,  9,  7,  8,  9,  9,  7,  6,  8,  9,  7,  8,  9,  7,  9,  6,
        7, 10,  9,  9,  7,  9,  7,  7,  9,  5,  7,  8,  8,  8,  7,  8,  7,
        8,  9,  6,  8,  8,  9,  8,  8,  7,  8,  8,  8,  8,  8, 10,  7,  8,
        8,  8,  8,  7,  9,  8,  6,  7,  6,  8,  8,  7,  8, 10,  6, 10,  8,
        7,  8,  6,  6,  7,  4,  9,  9,  7,  8,  6, 10,  8,  9,  9,  8,  8,
        7,  8, 10,  8,  9,  6,  9,  8,  7,  9,  9,  9,  9,  9,  8, 10,  8,
        5,  8,  9,  8,  8,  8,  6,  9,  9,  8,  8,  8, 10,  7,  5,  9,  9,
        6,  6, 10, 10,  9, 10,  8,  9,  8,  7,  7,  8,  8,  6,  8,  7,  9,
        8,  8,  6,  7,  5,  8,  8,  9,  7,  9,  5,  8, 10, 10,  8,  8,  9,
        6,  6,  7,  9,  7,  6,  8,  7,  8,  8,  7,  7,  9,  7,  8,  8,  7,
        9,  7,  7,  9,  9,  8,  8,  9,  9,  9,  8,  6,  8,  7,  7,  9,  9,
        9,  7,  9, 10, 10, 10,  8,  9,  8,  8,  8,  8,  8,  8, 10,  9, 10,
        8,  7,  9,  7,  9,  7,  6,  8,  7,  6,  7,  6,  7,  8,  9, 10,  9,
        8, 10,  7, 10,  9,  8,  8,  7,  9,  8,  8,  6,  8,  9,  9,  9, 10,
        6, 10,  9,  6,  9,  9,  8,  7,  6,  7,  7,  6,  8,  9,  8,  9,  9,
        6,  8,  8,  8,  9,  9,  7,  8, 10,  8,  9,  8,  7,  7, 10,  7,  8,
        8,  7,  7,  8, 10,  8, 10,  9, 10,  8,  8,  6,  8,  7,  8, 10,  8,
        4,  7,  9,  8,  9,  7,  9,  9,  5,  8,  8,  8,  9,  9,  8,  9,  6,
        7,  7,  9,  8,  9,  8,  9,  8,  9, 10,  7,  9,  6,  9,  5,  7,  6,
        8,  7,  8,  9,  9, 10,  8,  8,  8,  9,  8, 10,  4,  8, 10,  5,  8,
        9, 10,  9,  8,  9,  9,  8,  8, 10,  9,  7,  9,  7,  6,  7,  6,  9,
        8, 10,  7, 10,  8,  7,  7,  9,  9,  6,  9,  8,  9,  9, 10,  8,  9,
        8,  7,  7,  9,  9,  8,  9,  9,  9,  9,  8, 10,  8,  7,  8, 10,  9,
       10,  9,  8,  8,  6,  9,  9,  7,  7,  6,  8,  7,  9,  8,  6,  7,  8,
        6,  8,  7, 10,  9, 10,  6,  8,  9,  8,  6,  8,  9,  6,  8,  9, 10,
        6,  7,  7,  8,  9,  9,  8,  8,  8,  8,  7,  9,  9, 10,  8,  5,  8,
        8,  8,  7,  8,  9,  8,  7,  9, 10, 10,  7,  7, 10,  8,  9,  8,  7,
        6,  8,  8,  7, 10,  7,  7,  8,  6, 10,  8, 10,  9,  9,  9, 10,  9,
        9,  9,  8, 10,  7,  9,  7, 10,  9,  9,  8, 10,  9, 10,  8,  9,  7,
        8,  7,  8,  8,  6,  7,  8,  7,  8,  8,  9,  8,  9,  8,  7,  5,  5,
        9, 10,  7,  9,  8,  6,  9,  5,  8,  9,  7,  6,  9,  9,  9,  8,  8,
       10,  7,  7,  8,  9,  8,  7,  9,  8,  9,  9,  8,  7,  8,  9,  8,  9,
        9,  7, 10,  9,  9,  8,  9,  8,  8,  9,  8,  6,  9,  9,  6,  9,  8,
        8,  5,  9,  9,  7,  8,  8,  7,  8,  6,  8,  5,  9,  8, 10,  7,  8,
        5,  9,  8,  8,  9,  7,  8,  6, 10, 10,  8,  8, 10,  9,  7,  7,  6,
        7,  8,  7,  7,  7,  5,  9, 10,  9,  7,  7,  9,  7,  5,  5,  5,  7,
       10,  7,  8,  9,  9,  9,  8,  7,  7,  9,  8, 10,  8,  8,  7,  4, 10,
        8,  8,  9,  8,  7,  7, 10,  7,  9,  9,  8,  6,  6,  8,  5,  8,  8,
        8,  9,  7,  9,  9,  8,  7,  8,  7,  6,  8,  6,  7,  8,  9,  8,  9,
        8, 10,  7,  7,  8,  9,  6,  9,  8,  9, 10,  7,  8,  8,  8,  7,  6,
```

```
       9,  9,  8,  7,  9,  7,  9,  5,  9,  7,  6,  8,  7,  7,  9,  9,  7,
       8,  9,  7,  7,  9,  9,  7,  9,  9,  7, 10,  7,  8, 10])
```

In [138]:
```
# Plot

n = 10  # number of trials
p = 0.8  # probability of success
size = 1000  # number of samples to generate

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

plt.hist(binomial_dist,density=True)
plt.show()
```



In [139]:
```
# decreased the probability

n = 10  # number of trials
p = 0.2  # probability of success
size = 1000  # number of samples to generate

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

plt.hist(binomial_dist,density=True)
plt.show()
```



Criteria:

1. The process consists of n trials
2. Only 2 exclusive outcomes are possible, a success and a failure.
3. P(success) = p and P(failure) = 1-p and it is fixed from trial to trial
4. The trials are independent.

USECASES IN Machine Learning

1. **Binary classification problems**: In binary classification problems, we often model the probability of an event happening as a binomial distribution. For example, in a spam detection system, we may model the probability of an email being spam or not spam using a binomial distribution.

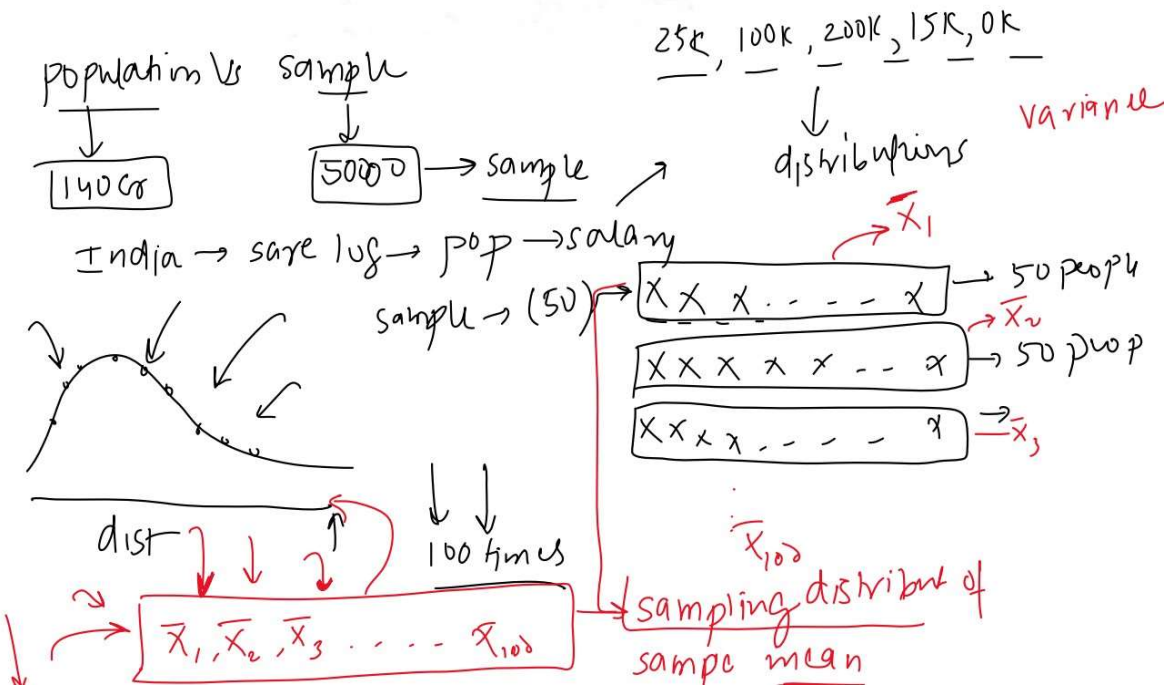
2. **Hypothesis testing**: In statistical hypothesis testing, we use the binomial distribution to calculate the probability of observing a certain number of successes in a given number of trials, assuming a null hypothesis is true. This can be used to make decisions about whether a certain hypothesis is supported by the data or not.

3. **Logistic regression**: Logistic regression is a popular machine learning algorithm used for classification problems. It models the probability of an event happening as a logistic function of the input variables. Since the logistic function can be viewed as a transformation of a linear combination of inputs, the output of logistic regression can be thought of as a binomial distribution.

4. **A/B testing**: A/B testing is a common technique used to compare two different versions of a product, web page, or marketing campaign. In A/B testing, we randomly assign individuals to one of two groups and compare the outcomes of interest between the groups. Since the outcomes are often binary (e.g., click□through rate or conversion rate), the binomial distribution can be used to model the distribution of outcomes and test for differences between the groups.

Sampling Distribution

Sampling distribution is a probability distribution that describes the statistical properties of a sample statistic (such as the sample mean or sample variance) computed from multiple independent samples of the same size from a population.

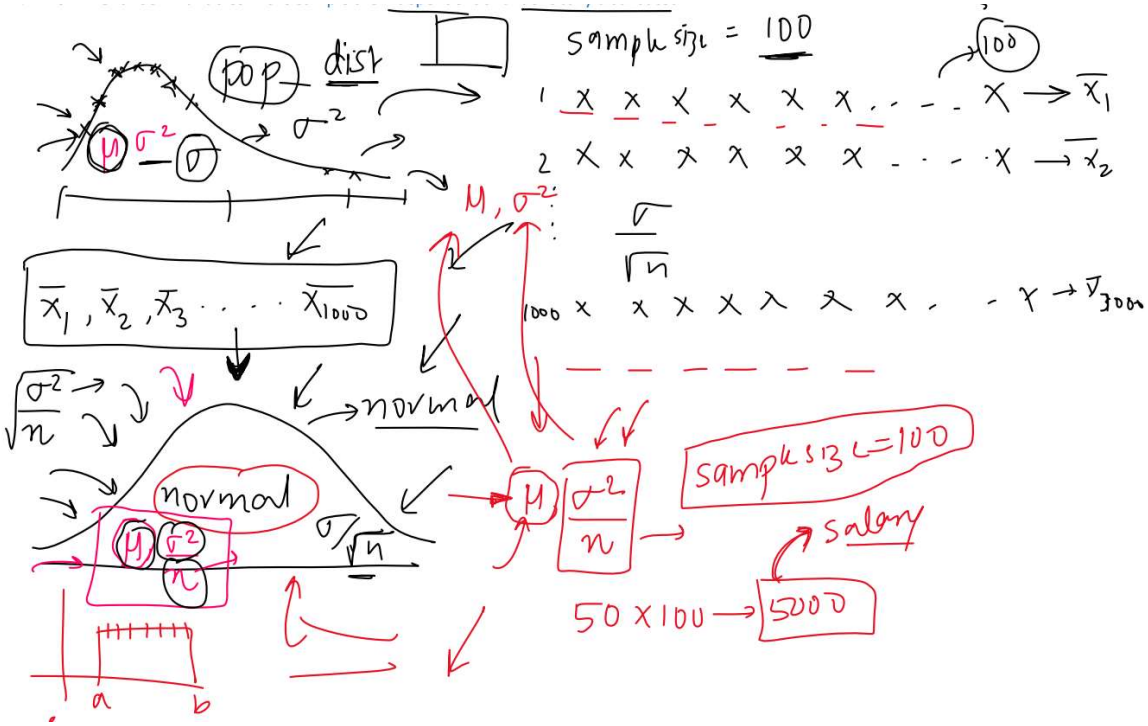


- **Why Sampling Distribution is important?**

Sampling distribution is important in statistics and machine learning because it allows us to estimate the variability of a sample statistic, which is useful for making inferences about the population. By analysing the properties of the sampling distribution, we can compute confidence intervals, perform hypothesis tests, and make predictions about the population based on the sample data.

Central Limit Theorm

The Central Limit Theorem (CLT) states that the distribution of the sample means of a large number of independent and identically distributed random variables will approach a normal distribution, regardless of the underlying distribution of the variables.



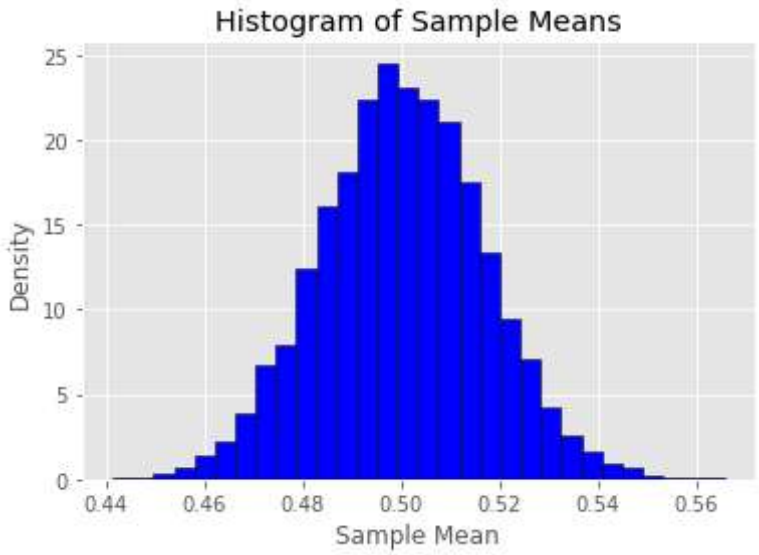
The conditions required for the CLT to hold are:

1. The sample size is large enough, typically greater than or equal to 30.
2. The sample is drawn from a finite population or an infinite population with a inite variance.
3. The random variables in the sample are independent and identically distributed.

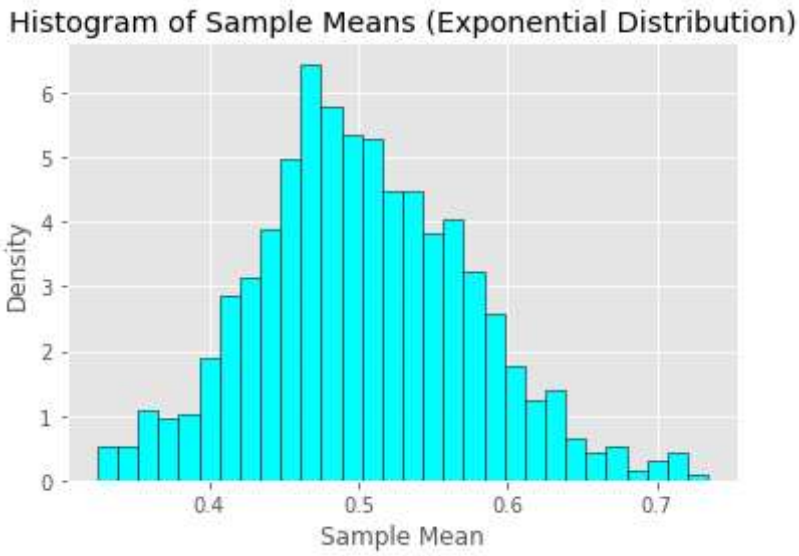
The CLT is important in statistics and machine learning because it allows us to make probabilistic inferences about a population based on a sample of data.

For example, we can use the CLT to construct confidence intervals, perform hypothesis tests, and make predictions about the population mean based on the sample data. The CLT also provides a theoretical justification for many commonly used statistical techniques, such as t-tests, ANOVA, and linear regression

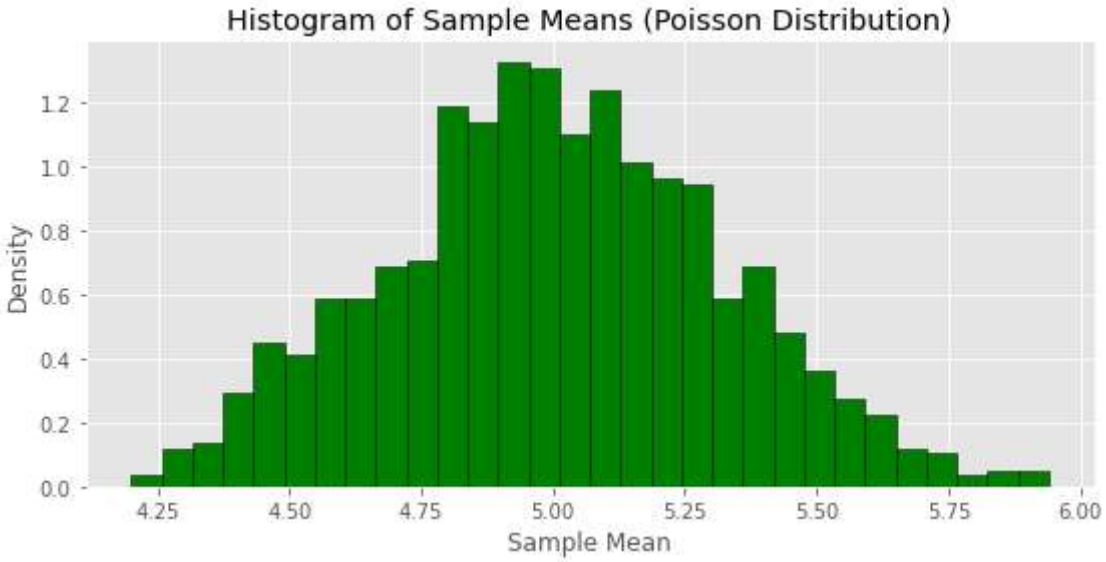
In [140]:
```
# code


# Set the parameters
num_samples = 10000
sample_size = 300
distribution_range = (0, 1)

# Generate samples from a uniform distribution

samples = np.random.uniform(distribution_range[0],
                            distribution_range[1],
                            (num_samples, sample_size))

# Calculate the sample means
sample_means = np.mean(samples, axis=1)

# Plot the histogram of the sample means

plt.hist(sample_means, bins=30, density=True, edgecolor='black',color='blue')
plt.title('Histogram of Sample Means')
plt.xlabel('Sample Mean')
plt.ylabel('Density')
plt.show()
```

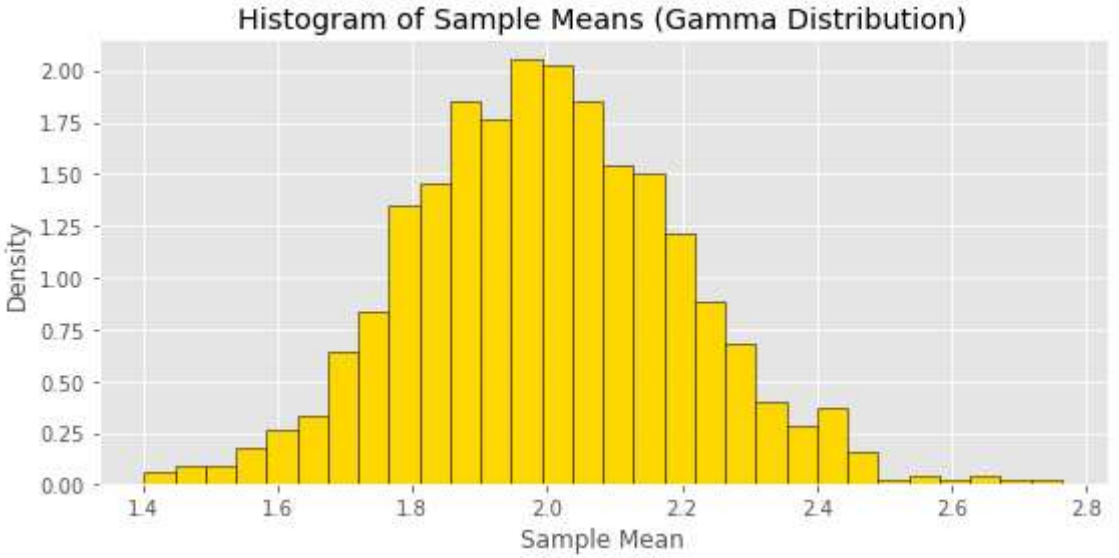


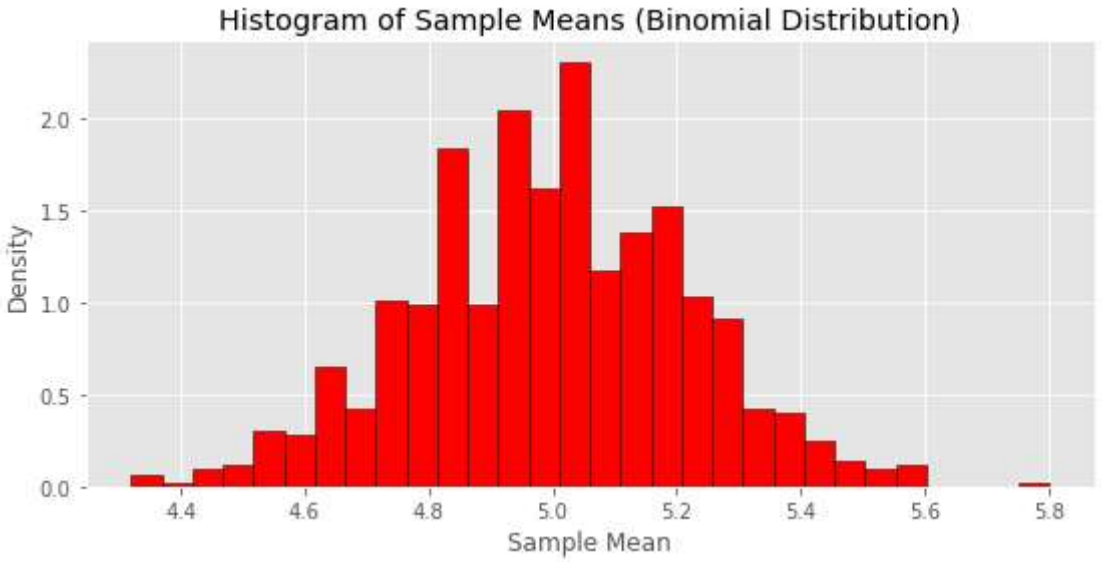
In [141]:
```
# Exponential - normal

# Set the parameters
num_samples = 1000
sample_size = 50
lambda_param = 2

# Generate samples from an exponential distribution
samples = np.random.exponential(scale=1/lambda_param, size=(num_samples, sampl

# Calculate the sample means
sample_means = np.mean(samples, axis=1)

# Plot the histogram of the sample means
plt.hist(sample_means, bins=30, density=True, edgecolor='black',color= 'cyan')
plt.title('Histogram of Sample Means (Exponential Distribution)')
plt.xlabel('Sample Mean')
plt.ylabel('Density')
plt.show()
```



In [142]:
```
# Poisson distribution ,Gamma distribution ,Binomial distribution

# Set the parameters
num_samples = 1000
sample_size = 50

# Poisson distribution parameters
poisson_lambda = 5

# Gamma distribution parameters
gamma_shape = 2
gamma_scale = 1

# Binomial distribution parameters
binomial_n = 10
binomial_p = 0.5

# Generate samples from the distributions
poisson_samples = np.random.poisson(lam=poisson_lambda, size=(num_samples, sam
gamma_samples = np.random.gamma(shape=gamma_shape, scale=gamma_scale, size=(nu
binomial_samples = np.random.binomial(n=binomial_n, p=binomial_p, size=(num_sa

# Calculate the sample means
poisson_means = np.mean(poisson_samples, axis=1)
gamma_means = np.mean(gamma_samples, axis=1)
binomial_means = np.mean(binomial_samples, axis=1)

# Plot the histograms of the sample means
fig, axs = plt.subplots(3, 1, figsize=(8, 12))

# Poisson distribution
axs[0].hist(poisson_means, bins=30, density=True, edgecolor='black',color ='gr
axs[0].set_title('Histogram of Sample Means (Poisson Distribution)')
axs[0].set_xlabel('Sample Mean')
axs[0].set_ylabel('Density')

# Gamma distribution
axs[1].hist(gamma_means, bins=30, density=True, edgecolor='black',color ='gold
axs[1].set_title('Histogram of Sample Means (Gamma Distribution)')
axs[1].set_xlabel('Sample Mean')
axs[1].set_ylabel('Density')

# Binomial distribution
axs[2].hist(binomial_means, bins=30, density=True, edgecolor='black',color ='r
axs[2].set_title('Histogram of Sample Means (Binomial Distribution)')
axs[2].set_xlabel('Sample Mean')
axs[2].set_ylabel('Density')

# Adjust the layout and show the plot
fig.tight_layout()
plt.show()
```


Histogram of Sample Means (Poisson Distribution)
Density
Sample Mean
0.0
0.2
0.4
0.6
0.8
1.0
1.2
4.25
4.50
4.75
5.00
5.25
5.50
5.75
6.00


Histogram of Sample Means (Gamma Distribution)
Density
Sample Mean
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1.4
1.6
1.8
2.0
2.2
2.4
2.6
2.8


Histogram of Sample Means (Binomial Distribution)
Density
Sample Mean
0.0
0.5
1.0
1.5
2.0
4.4
4.6
4.8
5.0
5.2
5.4
5.6
5.8

In [143]:
```
# Theoretical mean and variance'


import numpy as np

# Set the parameters
num_samples = 10000
sample_size = 50

# Gamma distribution parameters
gamma_shape = 2
gamma_scale = 1

# Calculate the theoretical mean and variance
theoretical_mean = gamma_shape * gamma_scale
theoretical_variance = (gamma_shape * gamma_scale ** 2)

# Generate samples from the Gamma distribution
samples = np.random.gamma(shape=gamma_shape, scale=gamma_scale, size=(num_samp

# Calculate the sample means
sample_means = np.mean(samples, axis=1)

# Calculate the empirical mean and variance of the sample means
empirical_mean = np.mean(sample_means)
empirical_variance = np.var(sample_means)

# Compare the theoretical and empirical values
print(f"Theoretical mean: {theoretical_mean:.4f}")
print(f"Empirical mean: {empirical_mean:.4f}")
print(f"\n")
print(f"Theoretical variance: {theoretical_variance:.4f}")
print(f"Empirical variance: {empirical_variance:.4f}")
```

```
Theoretical mean: 2.0000
Empirical mean: 2.0007


Theoretical variance: 2.0000
Empirical variance: 0.0390
```

In [144]:
```
2.0000/50 # Theoretical variance / 50(sample_size) = Empirical variance
```

Out[144]:
```
0.04
```

Case Study - What is the average income of Indians

Step-by-step process:

1. Collect multiple random samples of salaries from a representative group of Indians. Each sample should be large enough (usually, n > 30) to ensure the CLT holds. Make sure the samples are representative and unbiased to avoid skewed results.

2. Calculate the sample mean (average salary) and sample standard deviation for each sample.

3. Calculate the average of the sample means. This value will be your best estimate of the population mean (average salary of all Indians).

4. Calculate the standard error of the sample means, which is the standard deviation of the sample means divided by the square root of the number of samples.

5. Calculate the confidence interval around the average of the sample means to get a range within which the true population mean likely falls. For a 95% confidence interval:

- lower_limit = average_sample_means - 1.96 * standard_error
- upper_limit = average_sample_means + 1.96 * standard_error

6. Report the estimated average salary and the confidence interval.

In [146]:
```
import numpy as np

# Set the parameters
population_size = 100000
sample_size = 50
num_samples = 100

# Generate a random representative sample of salaries (in thousands)
# You should replace this with actual collected salary data
np.random.seed(42)  # Setting a seed for reproducibility
population_salaries = np.random.lognormal(mean=4.5,
                                          sigma=0.8,
                                          size=population_size)

# Generate multiple samples and calculate the sample means and standard deviat
sample_means = []
sample_std_devs = []

for _ in range(num_samples):
    sample_salaries = np.random.choice(population_salaries,
                                       size=sample_size)
    sample_means.append(np.mean(sample_salaries))
    sample_std_devs.append(np.std(sample_salaries))

# Calculate the average of the sample means and the standard error
average_sample_means = np.mean(sample_means)
standard_error = np.std(sample_means) / np.sqrt(num_samples)

# Calculate the 95% confidence interval
margin_of_error = 1.96 * standard_error
lower_limit = average_sample_means - margin_of_error
upper_limit = average_sample_means + margin_of_error

# Report the results
print(f"Estimated average salary (in thousands): {average_sample_means:.2f}")
print(f"95% confidence interval (in thousands): ({lower_limit:.2f}, {upper_lim
```

```
Estimated average salary (in thousands): 124.74
95% confidence interval (in thousands): (121.23, 128.26)
```

****Special Thanks to Krish Naik, sudhanshu kumar, Sunny Savita Sir iNeuron.ai****
