ACCEP
Communication and Computer Engineering Program

Random Variables

A python Implementation

Engineering Mathematics 5

EMP305

Under the supervision of

Dr. Lamia Alrefaai

Our full python implementations for the project's random variables and applications are here.

Table of Contents

Introduction

In a wide range of disciplines, from science and engineering to finance and the social sciences, probability and statistics are essential for comprehending and interpreting uncertainty. These mathematical specialties offer methods and instruments for deriving significant inferences and making judgments from data that are intrinsically unpredictable and variable. Random variables are important concepts in probability and statistics because they accurately represent the uncertainty and variability present in real-world phenomena. A random variable is a mathematical function that gives each potential result of a random experiment a numerical value. It acts as a link between the unpredictable character of real-world events and the deterministic realm of mathematics.

Random variables constitute essential elements in statistical modeling, providing a vital framework for comprehending and scrutinizing uncertainty within diverse phenomena. In the domain of probability theory and statistics, these variables hold a pivotal position in depicting and measuring the unpredictable facets of real-world situations.

There are different types of random variables, Different types of random variables are tailored to specific situations, each offering a unique perspective on uncertainty. we will be discussing in detail the following: Discrete random variables (Bernoulli, Binomial, Geometric, Uniform, and Poisson) & Continuous random variables (Uniform, Exponential, and Gaussian). And we will be implementing them using Python coding language and showing the result.

Discrete Random Variables

I. Bernoulli Random Variables:

1. Definitions:

- A Bernoulli distribution is a discrete probability distribution named after Jacob Bernoulli.
- Bernoulli Process: It describes a random variable that can take one of two possible outcomes, typically labeled as 0 and 1, where 1 represents success and 0 represents failure.
- Assumptions:
  - Binary Outcomes: The outcome of each trial is binary.
  - Independent Trials: Each trial is assumed to be independent of others. The outcome of one trial does not influence or affect the outcome of any other trial in the sequence.
  - Identical Probability Distribution: Each trial follows the same probability distribution, specifically a Bernoulli distribution.
  - Fixed Probability of Success: The probability of success (p) is assumed to be constant and does not change from trial to trial.
  - Discreteness: The outcomes are discrete, with only two possible values. This assumption is suitable for situations where the underlying process naturally yields discrete results, such as success or failure in a binary experiment.
  - Simple Structure: The Bernoulli distribution is simple yet foundational, serving as the building block for more complex distributions like the binomial distribution.

2. Equations:

- **Probability Mass Function (PMF):**

➢ The PDF of a Bernoulli distribution describes the likelihood of each possible outcome. For a Bernoulli random variable with probability of success p, the PDF assigns probabilities to the two possible values, 0 and 1.

➢ In the PDF, the bar for 0 is of height $1-p$, and the bar for 1 is of height p. These bars represent the probabilities of the random variable taking on the corresponding values.

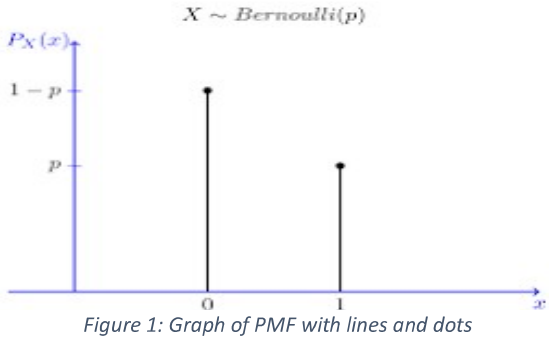


Figure 1: Graph of PMF with lines and dots

$$P(X=k)\begin{cases} p, & \text{if } k=1 \\ 1-p, & if\ k=0 \end{cases}$$

- **Cumulative Distribution Function (CDF):**

➢ The Cumulative Distribution Function (CDF) shows the cumulative probability up to a certain point. For a Bernoulli distribution, the CDF increases in steps.

➢ Starting at 0, the CDF increases to $1-p$ at 0, and then jumps to 1 at 1. This reflects the cumulative probability of observing values less than or equal to 0 and 1, respectively.

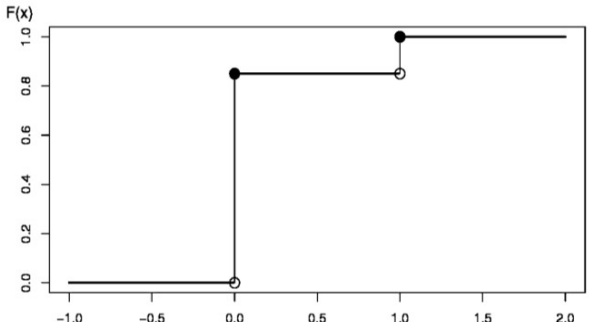


Figure 2: Graph of CDF with lines and dots

$$CDF = F(x,p) = \begin{cases} 0 & if\ x<0 \\ 1-p & if\ 0 \le x < 1 \\ 1 & if\ x \ge 1 \end{cases}$$

- **Expectation and Variance:**

➢ The mean (expected value) of Bernoulli distribution.

$E(X) = \sum_k k \cdot P(X=k)$

Substitute the values from the PMF:

$E(X) = 0 \cdot (1-p) + 1 \cdot p = p$

So, the expectation of a Bernoulli random variable is ***p***.

➢ The variance of a Bernoulli distributed random variable X with probability of success p is given by the formula:

$Var(X) = E((X - E(X))^2)$

Substitute the values:

$Var(X) = \sum_k (k - p)^2 \cdot P(X=k)$

$Var(X) = (0-p)^2 \cdot (1-p) + (1-p)^2 \cdot p$

$Var(X) = p^2 \cdot (1-p) + (1-p)^2 \cdot p$

$Var(X) = p(1-p)$

The standard deviation σ will be calculated by $\sqrt{\sigma^2}$

3. Properties:

- **Parameters:**

- The distribution is characterized by a single parameter ***p***, representing the probability of success. This parameter determines the shape and behavior of the distribution.'

- **Characteristics:**
- There are only two outcomes a 1 or 0, i.e., success or failure each time.
- If the probability of success is *p*, then the probability of failure is 1-*p,* and this remains the same across each successive trial.
- The probabilities are not affected by the outcomes of other trials which means the trials are independent.

4. Applications:

- The success or failure of a medical treatment, Transmission or non-transmission of a disease, A newborn child is male or female, Coin Flipping: Head or Tail, and Modelling if a financial asset will increase (success) or decrease (failure) in value.

5. Python Implementation:

This is only a documentation for the implemented python code. Functions full implementation is here: Bernoulli Random Variable

Generating a Bernoulli random variable:

```
5   def generate_bernoulli(p, size):
```

This Python function, named generate_bernoulli, generates random variables following a Bernoulli distribution.

Where :

- **p**: The probability of success in each Bernoulli trial. This is a parameter you pass to the function.
- **size**: The number of random variables to generate. This is also a parameter you pass to the function.

The function uses NumPy's np.random.binomial function, which simulates a binomial distribution with n=1 (because it's a Bernoulli distribution). The function returns an array (random_variables) containing the generated random variables.

Calculate PMF:

```
12   def bernoulli_pmf(x, p):
```

The function bernoulli_pmf(x, p) calculates the probability mass function (PMF) of a Bernoulli distribution for a given value x and probability of success p.

The formula p^x * (1-p)^(1-x) is the probability mass function of a Bernoulli distribution. It gives the probability of observing a particular outcome x (0 or 1) in a single Bernoulli trial with probability of success p. The term p^x corresponds to the probability of success if x is 1, and (1-p)^(1-x) corresponds to the probability of failure if x is 0.

Calculate CDF:

```
15   def bernoulli_cdf(x, p):
```

The bernoulli_cdf(x, p) function calculates the cumulative distribution function (CDF) of a Bernoulli distribution for a given value x and probability of success p. The CDF represents the probability that the random variable takes a value less than or equal to a given point.

If x is less than 0, the CDF is 0 because the random variable in a Bernoulli distribution cannot take negative values.

If 0 <= x < 1, the CDF is equal to 1 - p. This corresponds to the probability that the random variable is less than or equal to 0, which is the probability of failure (1) minus the probability of success (p).
If x is 1 or greater, the CDF is 1 because the random variable in a Bernoulli distribution can only take values of 0 or 1, and the cumulative probability is 1 when x is 1 or greater.

➢ Calculate Variance:

```
23    def bernoulli_variance(p):
```

The function bernoulli_variance(p) calculates the variance of a Bernoulli distribution, where p is the probability of success in a Bernoulli trial. The variance is a measure of the spread or dispersion of a probability distribution.
The formula for the variance of a Bernoulli distribution is given by:
$\mathrm{Var}(X)=p\cdot(1-p)$

where:

- **Var(X)** is the variance of the Bernoulli distribution.
- **p** is the probability of success in a single Bernoulli trial.

The expression $p\cdot(1-p)p\cdot(1-p)$ represents the product of the probability of success and the probability of failure. It quantifies how much the distribution of the Bernoulli random variable deviates from its mean, which is pp for a Bernoulli distribution.

➢ Calculate Expectation:

```
26    def bernoulli_expectation(p):
```

The function bernoulli_expectation(p) calculates the expected value (or mean) of a Bernoulli distribution, where p is the probability of success in a Bernoulli trial. The expected value represents the average or central tendency of a probability distribution.
For a Bernoulli distribution, the expected value is equal to the probability of success. Mathematically, it can be expressed as:
$E(X)=p$
where:

- **E(X)** is the expected value of the Bernoulli distribution.
- **p** is the probability of success in a single Bernoulli trial.

➢ Plotting PMF:

```
38    def plot_pmf(p):
```

The plot_pmf(p) function is designed to plot the probability mass function (PMF) of a Bernoulli distribution for a given probability of success p. The PMF represents the probabilities of different discrete outcomes of a random variable.

Here's a breakdown of the function:

- **Line 2:** It calculates the PMF values for each possible outcome of the Bernoulli distribution using the bernoulli_pmf(x, p) function, where x is one of the unique values (presumably 0 and 1).
- **Line 3-4:** It creates a bar plot using Matplotlib's plt.bar() function. The x-axis represents the unique values of the random variable (0 and 1), and the height of the bars represents the corresponding PMF values. The color='blue' argument sets the color of the bars and align='center' centers the bars on the x-axis. The alpha=0.7 argument controls the transparency of the bars, and width=0.01 sets the width of the bars. The label='PMF' is used for the legend.
- **Line 5:** It adds scatter points on top of the bars using Matplotlib's plt.scatter() function. This can be useful to visually emphasize the specific data points.
- **Line 6-7:** It sets the x-axis ticks to be at the unique values (0 and 1) and labels the x-axis as 'X (Random variable)' and the y-axis as 'PMF'.
- **Line 8:** It adds a title to the plot indicating that it represents the Probability Mass Function for a Bernoulli distribution with the specified probability of success p.
- **Line 9:** It adds a legend to the plot.
- **Line 10:** It displays the plot using plt.show().

➢ Plotting CDF:

```
50    def plot_cdf(p):
```

The plot_cdf(p) function is designed to plot the cumulative distribution function (CDF) of a Bernoulli distribution for a given probability of success p. The CDF represents the probability that the random variable takes a value less than or equal to a given point.
Here's an explanation of the function:

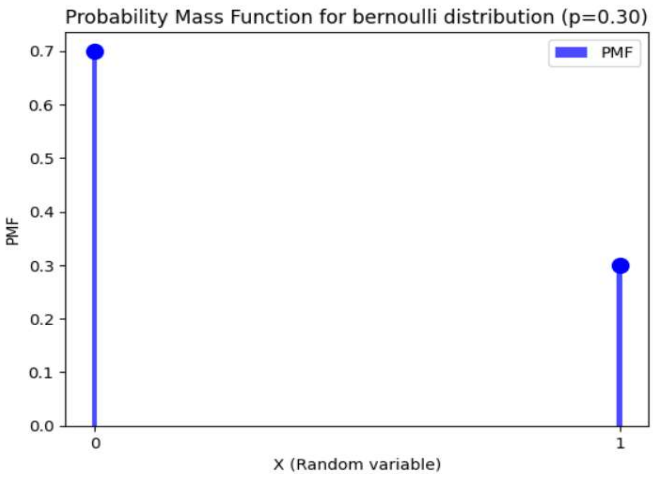
- **Line 2:** It calculates the CDF values for each possible outcome of the Bernoulli distribution using the bernoulli_cdf(x, p) function, where x is one of the unique values (presumably 0 and 1).
- **Line 3-4:** It creates a step plot using Matplotlib's plt.step() function. The x-axis represents the unique values of the random variable (0 and 1), and the y-axis represents the corresponding CDF values. The marker='o' argument adds circular markers at the data points, and color='green' sets the color of the step plot. The where='post' argument ensures that the step function increases at the right endpoint of each step. The label='CDF' is used for the legend.
- **Line 5:** It adds a title to the plot indicating that it represents the Cumulative Distribution Function for a Bernoulli distribution with the specified probability of success p.
- **Line 6-7:** It sets the x-axis ticks to be at the unique values (0 and 1) and labels the x-axis as 'X (Random variable)' and the y-axis as 'CDF'.
- **Line 8:** It adds a legend to the plot.
- **Line 9:** It displays the plot using plt.show().

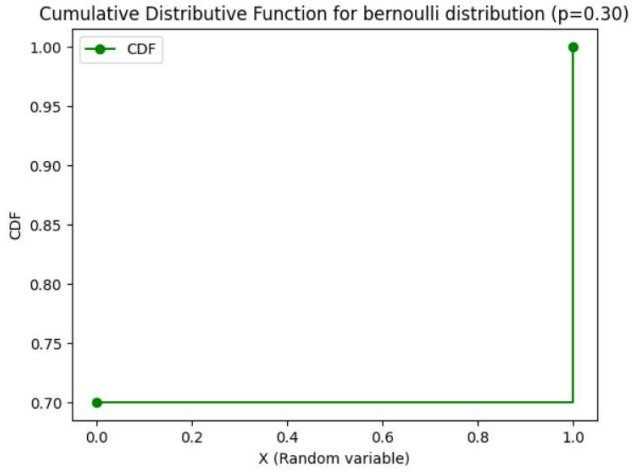
➢ Setting parameters:

```
p = float(input("Enter probability of success: "))
size = int(input("Enter size of random variables: "))
bernoulli_variables = generate_bernoulli(p, size)
unique_values = np.unique(bernoulli_variables)
```

6. *Code Results:*

```
variance =  0.21
expectation =  0.3
```





II. Binomial Random Variables:

1. *Definitions:*

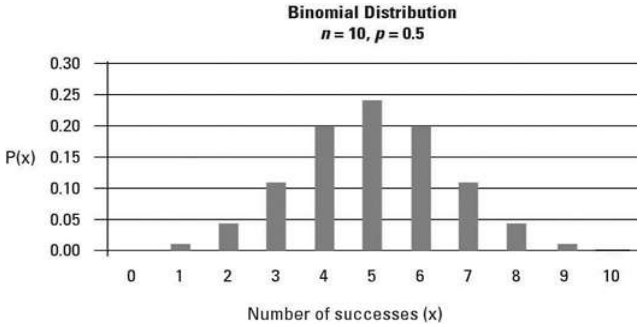
- The binomial distribution is a discrete probability distribution that models the number of successes in a fixed number of independent and identical Bernoulli trials.
- Binomial Process: Each trial results in a success with probability p or a failure with probability $q=1-p$, where p is the probability of success and q is the probability of failure.
- Assumptions: The binomial distribution is based on several key assumptions. These assumptions are crucial for accurately applying the binomial distribution to real-world situations. Here are the main assumptions of the binomial distribution:
  - Fixed Number of Trials (n): The number of trials, denoted as n, is fixed in advance. Each trial is independent of the others, and there are a specified and constant number of trials.
  - Independent Trials: Each trial is independent of the others. The outcome of one trial does not influence the outcome of another. This assumption ensures that the events are not dependent on each other.
  - Two Possible Outcomes (Binary): Each trial results in one of two possible outcomes, often labeled as success (S) and failure (F). These outcomes are mutually exclusive.
  - Constant Probability of Success (p): The probability of success (p) remains constant for each trial. In other words, the likelihood of success is the same for every trial.
  - Fixed Probability of Failure (q): The probability of failure (q), where $q=1-p$, is also constant for each trial. This is complementary to the probability of success.

2. Equations:

- **Probability Mass Function (PMF):**

➢ The Probability Mass Function (PMF) of a binomial distribution describes the probability of obtaining exactly *k* successes in *n* independent Bernoulli trials. The PMF is denoted as $P(X=k)$, where X is the random variable representing the number of successes.

$$P(X = k) = \binom{n}{k} p^k q^{n-k}$$



- **Derivation of PMF:**

➢ The binomial coefficient $\binom{n}{K}$ represents the number of ways to choose *k* successes out of *n* trials. It is defined as $\binom{n}{K} = \frac{n!}{k!(n-k)!}$, where *n*! denotes the factorial of *n*.

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Substitue the Binomial Coefficient

$$P(X = k) = \frac{n!}{k!\,(n-k)!}(1-p)^{n-k}$$

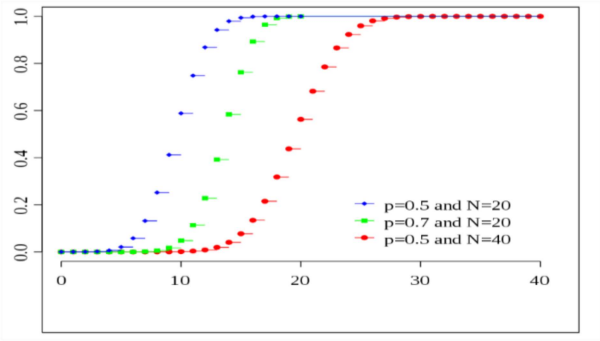
- **Cumulative Distribution Function (CDF):**

➢ The Cumulative Distribution Function (CDF) of a binomial distribution gives the probability that the random variable *X*, representing the number of successes in *n* independent Bernoulli trials, is less than or equal to a specific value *k*.

➢ The CDF is denoted as ***P(X≤k)*** and is calculated by summing the probabilities.

from ***k*=0** to ***k***:

$$P(X \le k) = \sum_{i=0}^{k} \binom{n}{k} p^i q^{n-i}$$



- **Derivation of CDF:**

Definition of CDF

$$F(k) = P(X \le k)$$

Sum of Probabilities

$$F(k) = P(X = 0) + P(X = 1) + \ldots + P(X = k)$$

Expression for $P(X = k)$

$$P(X = k) = \binom{n}{k} . p^k . (1-p)^{n-k}$$

Substitute into summation

$$F(k) = \sum_{i=0}^{k} \binom{n}{k} . p^i . (1-p)^{n-i}$$

Expand Summation

$$F(k) = \binom{n}{0} . p^0 . (1-p)^n + \binom{n}{1} . p^1 . (1-p)^{n-1} + \ldots + \binom{n}{k} . p^k . (1-p)^{n-k}$$

Use Binomial Coefficient formula.

$$F(k) = (1-p)^n + np(1-p)^{n-1} + \ldots + \binom{n}{k}.p^k.(1-p)^{n-k}$$

- **Expectation and Variance:**

- The expectation or expected value of a binomial distribution is the mean or average number of successes in a given number of independent Bernoulli trials.
- For a binomial distribution with parameters n (number of trials) and p (probability of success on each trial), the expected value (μ) is given by the formula: $E(X)=\mu=np$

$$E(X) = \sum_{x\in\Omega_X} xPr(X=x)$$

Thus:

$$E(X) = \sum_{k=0}^{n} k\binom{n}{k}p^k q^{n-k}$$
$$= \sum_{k=1}^{n} k\binom{n}{k}p^k q^{n-k}$$
$$= \sum_{k=1}^{n} n\binom{n-1}{k-1}p^k q^{n-k}$$
$$= np\sum_{k=1}^{n}\binom{n-1}{k-1}p^{k-1}q^{(n-1)-(k-1)}$$
$$= np\sum_{j=0}^{m}\binom{m}{j}p^j q^{m-j}$$
$$= np$$

- The variance of a binomial distribution measures the spread or dispersion of the distribution. For a binomial distribution with parameters n (number of trials) and p (probability of success on each trial), the variance ($\sigma 2$) is given by the formula: $\sigma 2=npq$.

$$\sigma^2 = E(x^2) - [E(x)]^2$$
$$= (np + n^2.p^2 - n.p^2) - (np)^2$$
$$= np + n^2.p^2 - n.p^2 - n^2.p^2$$
$$= np - n.p^2$$
$$= np(1-p)$$
$$= npq$$

The standard deviation σ will be calculated by $\sqrt{\sigma^2}$

3. *Properties:*

- **Parameters:**

- ***p*** is the probability of success on an individual trial.
- ***q*** is the probability of failure on an individual trial (***q***=1−***p***).
- ***n*** is the total number of trials.
- ***k*** is the number of successes.

- **Characteristics:**

- The number of observations n is fixed.
- Each observation is independent.
- Each observation represents one of two outcomes ("success" or "failure").
- The probability of "success" p is the same for each outcome.
- Binomial distribution is a discrete distribution.
- Binomial distribution has two parameters n and p.
- Mean of binomial distribution is np.

- Variance of binomial distribution is npq.
- Standard deviation of binomial distribution √npq.
- The mean is always greater than the variance.

4. *Applications:*

- The binomial random variable finds widespread applications across various fields such as Quality Control and Manufacturing, Biomedical Studies, Finance, Economics and Market Research, Genetic and Biology and Educational Testing.

5. *Python Implementation*:

This is only a documentation for the implemented python code. Functions full implementation is here: Binomial Random Variable

➢ Generating a binomial random variable:

```
7    def generate_binomial(n, p, size):
```

The generate_binomial function generates a list of random variables following a binomial distribution. Here's an explanation of the parameters and the function:

- **n:** This parameter represents the number of trials in each binomial experiment. In a binomial distribution, each experiment consists of a fixed number of trials, and 'n' specifies how many trials are conducted.
- **p:** This parameter is the probability of success in each trial. In a binomial distribution, there are two possible outcomes in each trial – success or failure. 'p' represents the probability of achieving success in a single trial.
- **size:** This parameter determines the number of random variables to generate. The function will produce a list/array of random variables, and 'size' indicates how many elements this list should contain.

The function uses NumPy's random.binomial function, which generates random variables following a binomial distribution. It takes 'n', 'p', and 'size' as arguments and returns a list/array of random variables based on the specified binomial distribution.

➢ Calculate PMF:

```
15    def binomial_pmf(k, n, p):
```

The function binomial_pmf calculates the probability mass function (PMF) for a binomial distribution. Here's an explanation of the parameters and the formula used:

- **k**: This parameter represents the number of successes (or positive outcomes) in the binomial distribution. The PMF is calculated for a specific number of successes 'k' out of 'n' trials.
- **n**: This parameter is the total number of trials in the binomial experiment. It represents the number of times an event is repeated, and each trial has only two possible outcomes (success or failure).
- **p**: This parameter is the probability of success in a single trial. It represents the likelihood of the event being a success in each individual trial.

The formula used to calculate the binomial PMF is:

$P(X=k)=(kn)\cdot pk\cdot(1-p)n-k$

where:

- (kn) is the binomial coefficient, representing the number of ways to choose 'k' successes from 'n' trials. It is often calculated using the combination formula: (nk)=n!k!(n−k)!(kn)=k!(n−k)!n! , where '!' denotes factorial.
- pkpk is the probability of 'k' successes.
  (1−p)n−k(1−p)n−k is the probability of 'n - k' failure

➢ Calculate CDF:

```
19    def binomial_cdf(k, n, p):
```

The function binomial_cdf calculates the cumulative distribution function (CDF) for a binomial distribution. The CDF gives the probability that a random variable X takes a value less than or equal to a specified value kk. Here's an explanation of the parameters and the logic used in the function:

- **k:** This parameter represents the value for which we want to calculate the cumulative probability. The CDF gives the probability that the random variable is less than or equal to kk.
- **n:** The total number of trials in the binomial experiment.
- **p:** The probability of success in a single trial.

The function uses a loop to iterate through all possible values of the random variable from 0 to k (inclusive) and accumulates the probabilities using the binomial_pmf function for each value. The loop adds up the individual probabilities to calculate the cumulative probability up to k.

➢ Calculate Variance:

```
30    def binomial_variance(n, p):
```

The function binomial_variance calculates the variance of a binomial distribution. Variance is a measure of the spread or dispersion of a distribution. Here's an explanation of the parameters and the logic used in the function:

- **n**: This parameter represents the number of trials in the binomial experiment. It is the total number of times the event is repeated.
- **p**: This parameter is the probability of success in a single trial. It represents the likelihood of the event being a success in each individual trial.

The formula for the variance of a binomial distribution is given by:
Var(X)=n·p·(1−p)
where:

- **Var(X)** is the variance of the binomial distribution.
- **n** is the number of trials.
- **p** is the probability of success in a single trial.

The function returns the product of the number of trials (nn), the probability of success (pp), and the probability of failure (1−p1−p). This formula represents the spread or variability in the number of successes in a binomial distribution.

➢ Calculate Expectation:

```
26    def binomial_expectation(n, p):
```

The function binomial_expectation calculates the expected value (or mean) of a binomial distribution. The expected value is a measure of the center or average of the distribution. Here's an explanation of the parameters and the logic used in the function:

- **n**: This parameter represents the number of trials in the binomial experiment. It is the total number of times the event is repeated.
- **p**: This parameter is the probability of success in a single trial. It represents the likelihood of the event being a success in each individual trial.

The formula for the expected value of a binomial distribution is given by:

$E(X)=n \cdot pE(X)=n \cdot p$

where:

- **E(X)** is the expected value of the binomial distribution.
- **n** is the number of trials.
- **p** is the probability of success in a single trial.

The function simply returns the product of the number of trials (n) and the probability of success (p), which gives the expected number of successes in the binomial distribution.

➢ Plotting PMF:

```
34    def plot_pmf(n, p, size):
```

- **PMF Calculation**: The function computes the Probability Mass Function (PMF) values for a binomial distribution with parameters n (number of trials) and p (probability of success in each trial) for a range of unique values.
- **Visualization**: It uses Matplotlib to create a bar plot and scatter plot of the PMF values, showing the probability of each unique value. The x-axis represents the random variable, and the y-axis represents the PMF.
- **Title and Labels**: The function sets appropriate labels and a title for the plot, including the specified probability of success (p). It then displays the plot.

➢ Plotting CDF:

```
47    def plot_cdf(n, p, size):
```

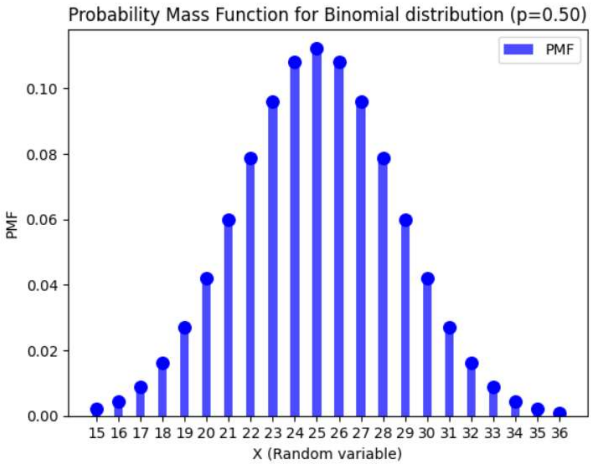
- **CDF Calculation**: The function calculates the Cumulative Distribution Function (CDF) values for a binomial distribution with parameters n (number of trials) and p (probability of success in each trial) for a range of unique values.
- **Visualization:** Using Matplotlib, it creates a step plot of the CDF values, illustrating the cumulative probability up to each unique value. The x-axis represents the random variable, and the y-axis represents the CDF.
- **Title and Labels**: The function sets appropriate labels and a title for the plot, indicating the specified probability of success (p). Finally, it displays the plot.

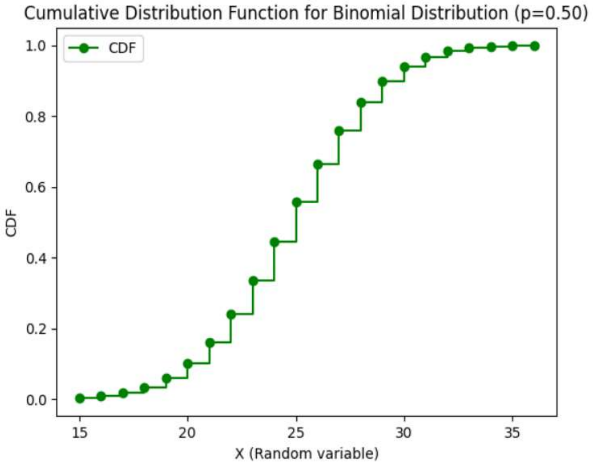
➢ Setting parameters:

```
59    p = 0.5
60    n = 50
61    size = 1000
62    binomial_random_variables = generate_binomial(n, p, size)
63    unique_values, counts = np.unique(binomial_random_variables, return_counts=True)
```

6. *Code results:*

```
variance is: 12.5
expectation is: 25.0
```





III. Geometric Random Variables:

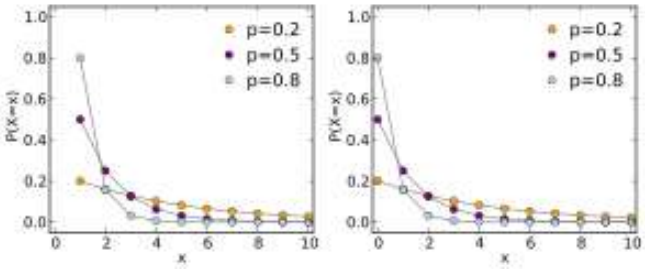
1. *Definitions*:

- The geometric random variable models a series of experiments with two outcomes (success or failure). It signifies the number of trials needed to achieve the first success, taking values of 1, 2, 3, and so on.
- Assumptions: Ensure that the assumptions of the geometric distribution hold, including the independence of trials, constant probability of success, and the memory lessness property. The geometric distribution makes several key assumptions to be applicable in a given scenario. These assumptions include:

➢ Bernoulli Trials: The underlying process consists of a sequence of independent Bernoulli trials, meaning each trial has only two possible outcomes (success or failure).
➢ Constant Probability of Success: The probability of success p, p remains constant across all trials. In other words, the success probability is the same for each trial.
➢ Independence: The trials are independent of each other. The outcome of one trial does not influence the outcome of another.
➢ Discreteness: The random variable representing the number of trials until the first success is a discrete variable. It takes on non-negative integer values.

2. Equations:

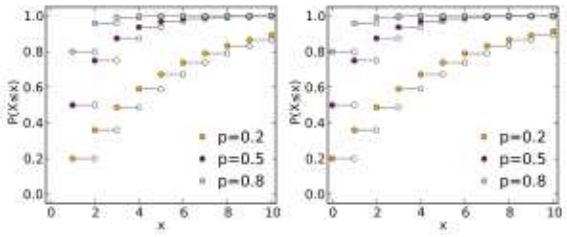
• Probability Mass Function (PMF):



➢ The probability mass function can be defined as the probability that a discrete random variable, X, will be exactly equal to some value, x. The formula for geometric distribution PMF is given as follows:

$$P(X = x) = (1-p)^{x-1} \cdot p$$
$$where, 0 < p \leq 1$$

• Cumulative Distribution Function (CDF):



➢ The cumulative distribution function of a random variable, X, that is evaluated at a point, x, can be defined as the probability that X will take a value that is lesser than or equal to x. It is also known as the distribution function. The formula for the geometric distribution CDF is given as follows:

$$P(X \leq x) = 1 - (1-p)^{x}$$

• Derivation of CDF and PMF:

➢ The PMF can be derived using the probability of having k−1 failures followed by one success:

$$P(X = x) = P(\text{failure})^{k-1} \cdot \text{P(success)}$$
$$P(success) = (1-p)^{k-1} \cdot p$$

➢ The CDF can be derived *by summing the PMF up to k.*

$$P(X \leq k) = \sum_{i=1}^{k} P(X = i) = \sum_{i=1}^{k} (1-p)^{i-1} \cdot p$$

➢ The sum is a geometric series, and the formula for the sum of the first n terms of a geometric series is given by:

$$S_n = \frac{1-(1-p)^n}{p}$$

➢ *Applying this formula to the sum, we get*:

$$P(X \leq k) = \frac{1-(1-p)^k}{p}$$

This is the CDF of the geometric random variable.

• Expectation and Variance:

For a geometric random variable X with probability of success p, the mean (expected value) and variance can be calculated as follows:

➢ The mean is the expected number of trials needed to get the first success. Since each trial has a probability p of success, the expected number of trials is given by the reciprocal of p.

$$\mu = \frac{1}{p}$$

➢ The variance measures the spread of the distribution. For a geometric random variable, the variance is given by:

$$\sigma^2 = \frac{1-p}{p^2}$$

➢ **Derivation:**

This can be derived using the fact that the variance of a random variable X is given by:

$$Var(X) = E(X^2) - [E(X)]^2$$

For the geometric distribution, $E(X^2)$ can be calculated using the formula:

$$E(X^2) = \sum_{x=1} x^2 \cdot P(X = k)$$

Using the PMF of the geometric distribution,

$$P(X=k) = (1-p)$$

you can calculate $E(X^2)$ and then use it in the variance formula to obtain the expression for the variance.

The standard deviation σ will be calculated by $\sqrt{\sigma^2}$

3. *Properties:*

- **parameters:**

➢ p: which represents the probability of success in a Bernoulli trial.

- **Characteristics:**

➢ There are one or more Bernoulli trials with all failures except the last one, which is a success.
➢ In theory, the number of trials could go on forever.
➢ There must be at least one trial.
➢ The probability, p, of a success and the probability, q, of a failure is the same for each trial. p+q=1 and q=1−p.

4. *Applications:*

➢ The geometric random variable is commonly used to model the number of Bernoulli trials needed to achieve the first success in a sequence of independent trials with a constant probability of success. Including Queueing Theory, Reliability and Failure Analysis, Web Search and Information Retrieval and Epidemiology.

5. *Python Implementation*:

This is only a documentation for the implemented python code. Functions full implementation is here: Geometric Random Variable

➢ **Generating a binomial random variable:**

```
6    def generate_geometric(p, size):
```

The function generate_geometric generates a list of random variables following a geometric distribution. Here's an explanation of the parameters and the function:

- p: This parameter represents the probability of success on each trial in the geometric distribution. In a geometric distribution, the random variable represents the number of trials needed to achieve the first success.
- size: This parameter determines the number of random variables to generate. The function will produce a list/array of random variables, and 'size' indicates how many elements this list should contain.

The function uses NumPy's random.geometric function, which generates random variables following a geometric distribution. It takes 'p' as the probability of success and 'size' as the

number of random variables to generate. The resulting list contains integers representing the number of trials needed to achieve the first success in a series of independent Bernoulli trials.

➢ Calculate PMF:

```
21    def geometric_pmf(x, p):
```

The function geometric_pmf calculates the probability mass function (PMF) for a geometric distribution. The geometric distribution models the number of trials needed to achieve the first

success in a sequence of independent Bernoulli trials, where each trial has a probability of success pp. Here's an explanation of the parameters and the logic used in the function:

- x: This parameter represents the number of trials needed to achieve the first success. In a geometric distribution, xx is a positive integer greater than or equal to 1.
- p: This parameter is the probability of success in each trial. It represents the likelihood of the event being a success in each individual trial.

The probability mass function (PMF) for a geometric distribution is given by:

$P(X=x)=(1-p)x-1\cdot pP(X=x)=(1-p)x-1\cdot p$

where:

- $P(X=x)P(X=x)$ is the probability that the first success occurs on the x-th trial.
- $(1-p)x-1(1-p)x-1$ is the probability of having $x-1x-1$ consecutive failures followed by a success.
- p is the probability of success on the x-th trial.

➢ Calculate CDF:

```
25    def geometric_cdf(x, p):
```

The function geometric_cdf calculates the cumulative distribution function (CDF) for a geometric distribution. The geometric distribution models the number of trials needed to achieve the first success in a sequence of independent Bernoulli trials, where each trial has a probability of success p. Here's an explanation of the parameters and the logic used in the function:

- x: This parameter represents the number of trials needed to achieve the first success. In a geometric distribution, xx is a positive integer greater than or equal to 1.
- p: This parameter is the probability of success in each trial. It represents the likelihood of the event being a success in each individual trial.

The cumulative distribution function (CDF) for a geometric distribution is given by:

$P(X\leq x)=1-(1-p)xP(X\leq x)=1-(1-p)x$

where:

- $P(X\leq x)$ is the probability that the first success occurs on or before the xx-th trial.

- (1−p)x is the probability of having xx consecutive failures before the first success.
- 1−(1−p)x is the cumulative probability up to the xx-th trial.

➢ **Calculate Variance:**

```
34    def geometric_variance(p):
```

➢ **Calculate Expectation:**

```
30    def geometric_expectation(p):
```

The function geometric_expectation calculates the expected value (or mean) of a geometric distribution. The geometric distribution models the number of trials needed to achieve the first success in a sequence of independent Bernoulli trials, where each trial has a probability of success p. Here's an explanation of the parameter and the logic used in the function:

- p: This parameter is the probability of success in each trial. It represents the likelihood of the event being a success in each individual trial.

The formula for the expected value of a geometric distribution is given by:

E(X)=1pE(X)=p1

where:

- E(X) is the expected value of the geometric distribution.
- p is the probability of success in each trial.

In simple terms, the expected value represents the average number of trials needed to achieve the first success in the sequence of independent trials.

➢ **Plotting PMF:**

```
39    def plot_pmf(p):
```

- **PMF Calculation:**The function computes the Probability Mass Function (PMF) values for a geometric distribution with parameter pp (probability of success) for a range of unique values.
- **Visualization:** It uses Matplotlib to create a bar plot and scatter plot of the PMF values, showing the probability of each unique value. The x-axis represents the number of trials until the first success, and the y-axis represents the probability.
- **Title and Labels:** The function sets appropriate labels and a title for the plot, including the specified probability of success (pp), and then displays the plot.

➢ **Plotting CDF:**

```
51    def plot_cdf(p):
```

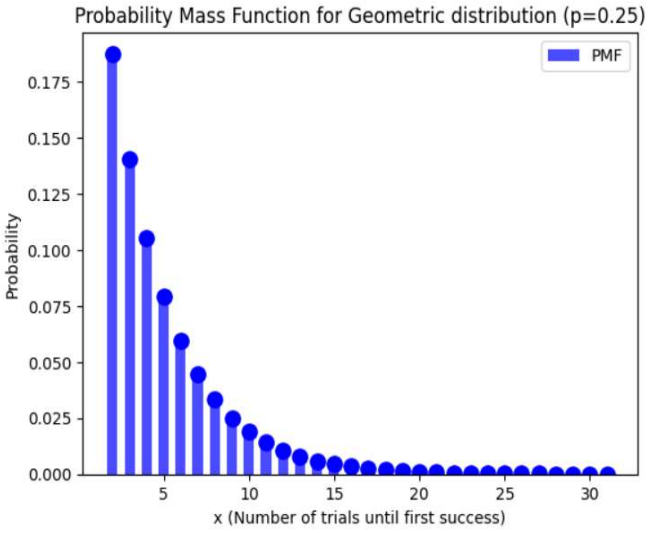
- **CDF Calculation:** The function calculates the Cumulative Distribution Function (CDF) values for a geometric distribution with parameter pp (probability of success) for a range of unique values.
- **Visualization:** Using Matplotlib, it creates a step plot of the CDF values, illustrating the cumulative probability up to each unique value. The x-axis represents the number of trials until the first success, and the y-axis represents the cumulative probability.
- **Title and Labels:** The function sets appropriate labels and a title for the plot, indicating the specified probability of success (pp), and then displays the plot.

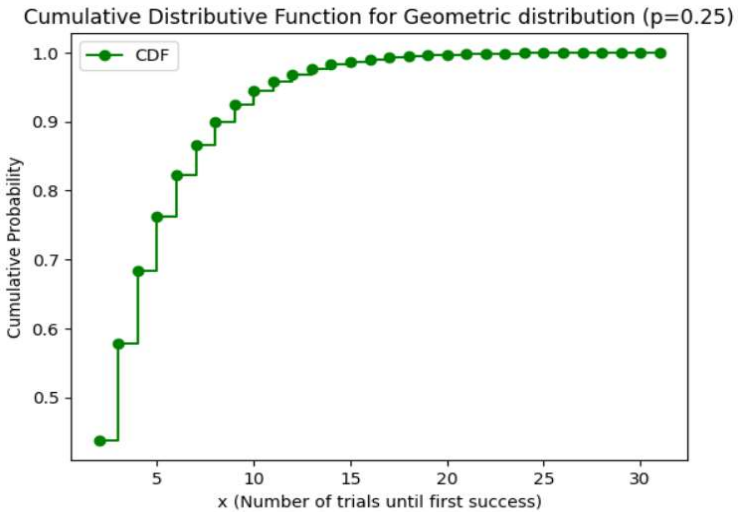
➢ **Setting parameters:**

```
62    p = 0.25
63    size = 1000
64    random_variables = generate_geometric(p, size)
65    unique_values, counts = np.unique(random_variables, return_counts=True)
```

6. *Code results:*

```
Expectation (mean): 4.00
Variance: 12.00
```





IV. Uniform Random Variables:

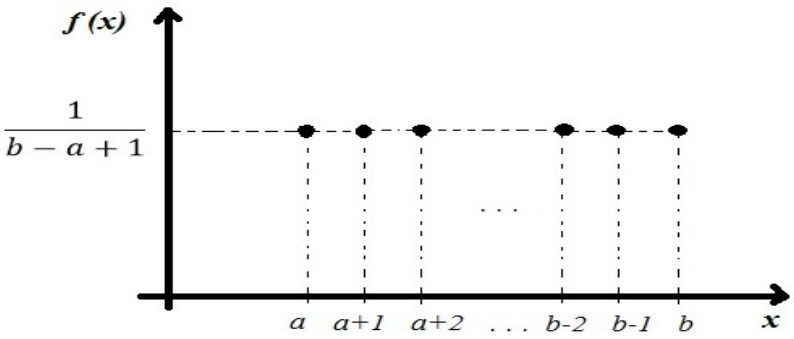
1. *Definitions:*

➢ Discrete uniform distribution is a symmetric probability distribution wherein a finite number of values are equally likely to be observed; every one of n values has equal probability $1/n$. Another way of saying "discrete uniform distribution" would be "a known, finite number of outcomes equally likely to happen".

2. *Equations:*

- **Probability Mass Function (PMF):**

$f_x(x) = \frac{1}{b-a+1}$ *where* $x \in \{a, a+1, \ldots, b-1, b\}$



- **Derivation of PMF:**

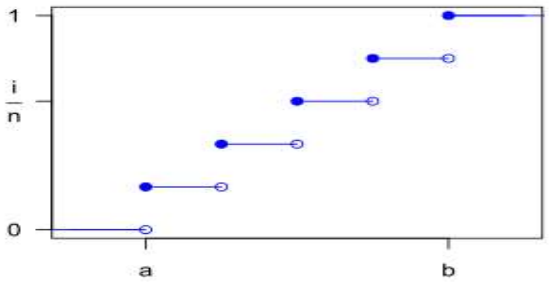
- Suppose we have a random variable *X* that follows a discrete uniform distribution over the values a, a+1, a+2,…., b, where a and b are integers.
- The probability mass function (PMF) is defined as the probability that the random variable takes on a specific value. In this case, since all values are equally likely, the PMF for each value is the same.
- For a uniform distribution, the total probability must sum to 1. Since there are b−a+1 possible values in the range, each individual probability is 1/(b−a+1).
- Therefore, the PMF P(X=x) for a uniform distribution is:

$$P(X = x) = \frac{1}{b - a + 1}$$

- where x is any integer in the range [a, b]. This means that each possible value has an equal probability of 1/(b−a+1), ensuring that the probabilities sum up to 1 over all possible values in the range.

- **Cumulative distribution function (CDF):**

$$F_x(x) = \begin{cases} 0, & if\ x < a \\ \dfrac{|x| - a + 1}{b - a + 1}, & if\ a \le x \le b \\ 1, & if\ x > b \end{cases}$$



- **Derivation of CDF**

- *Let X be a random variable with a discrete uniform distribution over the values a, a+1,a+2,...,b, where a and b are integers.*
- The CDF F(x) for a discrete uniform distribution is given by:

$$F(x) = P(X \le x)$$

For a given x, if x<a, then F(x)=0 because the random variable cannot be less than a.

- If a≤x≤b, then F(x) is the sum of the individual probabilities up to x:

$$F(x) = P(X \le x) = \sum_{k=a}^{x} P(X = k)$$

- Now, since P(X=k) =1/(b−a+1) for each k in the range, *the sum becomes:*

$$F(x) = \sum_{k=a}^{x} \frac{1}{b - a + 1}$$

- This sum is equal to the number of terms (values) in the sum, which is *x*−*a*+1, *multiplied by the individual probability:*

$$F(x) = (x - a + 1) \cdot \frac{1}{b - a + 1}$$

- *Simplifying this expression, we get:*

$$F(x) = \frac{x - a + 1}{b - a + 1}$$

For *x*>*b*, *F*(*x*) reaches its maximum value of 1 since the entire probability space has been covered.

- *So, the final expression for the CDF of a discrete uniform distribution is:*

$$F_x(x) = \begin{cases} 0, & if\ x < a \\ \dfrac{|x| - a + 1}{b - a + 1}, & if\ a \le x \le b \\ 1, & if\ x > b \end{cases}$$

- **Expectation and Variance:**
  - The mean or expected value of a uniform random variable defined over a<x<b is:

$$\mu = E(x) = \frac{a + b}{2}$$

  - Proof:

$$E(x) = \sum_{x=a}^{b} x \cdot P(X = x)$$

For a discrete uniform distribution, $P(X = x) = \frac{1}{b-a+1}$ *for each x in the range.*

$$E(x) = \sum_{x=a}^{b} x \cdot \frac{1}{b - a + 1}$$

Factor out the constant term:

$$E(x) = \frac{1}{b - a + 1} \sum_{x=a}^{b} x$$

Now, use the formula for the sum of consecutive integers:

$$E(x) = \frac{1}{b - a + 1} \cdot \frac{(b + a)(b - a + 1)}{2}$$

Simplify:

$$E(x) = \frac{a + b}{2}$$

  - The variance σ^2 for a discrete uniform distribution is given by: $\sigma^2 = \frac{(b-a+1)^2-1}{12}$

    Which is equal to: $\frac{n^2-1}{12}$
  - Proof:

$$\sigma^2 = \sum_{x=a}^{b} (x - E(x))^2 \cdot P(X = x)$$

Substitute in the values of P(X=x) and E(x)

$$\sigma^2 = \sum_{x=a}^{b} \frac{(x - \frac{a + b}{2})^2}{b - a + 1}$$

Expand and simplify:

$$\sigma^2 = \frac{1}{b-a+1}\sum_{x=a}^{b}(x^2 - 2x\frac{a+b}{2} + (\frac{a+b}{2})^2)$$

Use the formula for the sum of consecutive integers and the sum of consecutive squares:

$$\sigma^2 = \frac{1}{b-a+1}\cdot\frac{(b+a)(b-a+1)(2b+2a+1)}{6}$$

Simplify:

$$\sigma^2 = \frac{(b-a+1)^2-1}{12}$$

The standard deviation σ will be calculated by $\sqrt{\sigma^2}$

3. *Properties:*

- **Parameters:**

➢ For a discrete uniform distribution: *a* (minimum value), *b* (maximum value).

- **Characteristics:**

➢ Equal Probability: All values in the range have an equal probability of occurring.
➢ Rectangular Shape: The probability distribution function resembles a rectangle for both discrete and continuous uniform distributions.

4. *Applications:*

➢ Random Number Generation
➢ Statistical Sampling

5. *Python Implementation:*

This is only a documentation for the implemented python code. Functions full implementation is here: Uniform Random Variable

➢ Calculate PMF:

```
4   def discrete_uniform_pmf(k, a, b):
```

Function Purpose:

- Calculates the Probability Mass Function (PMF) for a discrete uniform distribution.

Parameters:

- k: The specific value for which to calculate the PMF.
- a: The lower limit of the distribution.
- b: The upper limit of the distribution.
- Explanation:
- Returns 1 / (b - a + 1) if k is within the range [a, b]. Otherwise, returns 0.

➢ Calculate CDF:

```
10   def discrete_uniform_cdf(x, a, b):
```

Function Purpose:

- Calculates the Cumulative Distribution Function (CDF) for a discrete uniform distribution.

Parameters:

- x: The specific value for which to calculate the CDF.

- a: The lower limit of the distribution.
- b: The upper limit of the distribution.

Explanation:

- Returns 0 if x is less than a.
- Returns (x - a + 1) / (b - a + 1) if x is within the range [a, b].
- Returns 1 if x is greater than b.

➢ Calculate Variance:

```
18    def discrete_uniform_variance(a, b):
```

- Calculates the variance of the discrete uniform distribution.

➢ Calculate Expectation:

```
21    def discrete_uniform_expectation(a, b):
```

- Calculates the expectation (mean) of the discrete uniform distribution.

Parameters:

- a: The lower limit of the distribution.
- b: The upper limit of the distribution.

Explanation:

- Variance formula: ((b - a + 1)^2 - 1) / 12
- Expectation formula: (a + b) / 2

➢ Plotting PMF:

```
24    def plot_discrete_uniform_pmf(a, b, size=10000):
```

Function Purpose:

- Generates a random sample (X) from the discrete uniform distribution and plots the PMF.

Parameters:

- a: The lower limit of the distribution.
- b: The upper limit of the distribution.
- size: Number of samples in the random sample.

Explanation:

- Generates a random sample of integers between a and b.
- Computes unique values and their counts.
- Calculates the PMF for each unique value and plots the bar chart.

➢ Plotting CDF:

```
36    def plot_discrete_uniform_cdf(a, b, size=10000):
```

Function Purpose:

- Generates a random sample (X) from the discrete uniform distribution and plots the CDF.

Parameters:

- a: The lower limit of the distribution.
- b: The upper limit of the distribution.
- size: Number of samples in the random sample.

Explanation:

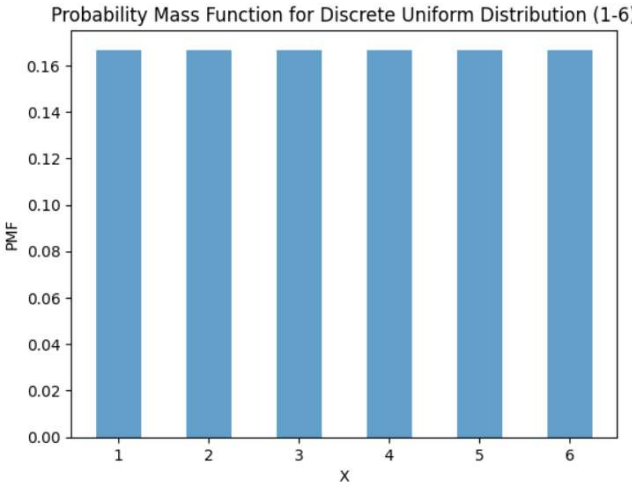
- Generates a random sample of integers between a and b.
- Sorts the sample and calculates the CDF for each value, then plots the step chart.

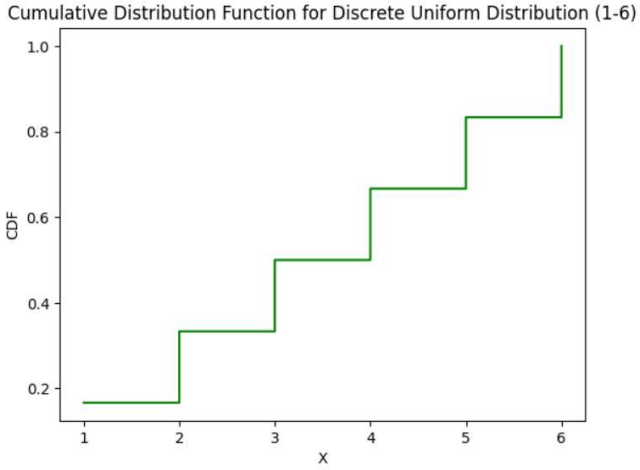
➢ Setting parameters:

```
a = int(input("enter the lower limit : "))
b = int(input("enter the upper limit : "))
```

6. *Code results:*

```
Variance is :  2.92
Expectation is :  3.5
```





V. Poisson Random Variables:

1. *Definitions:*

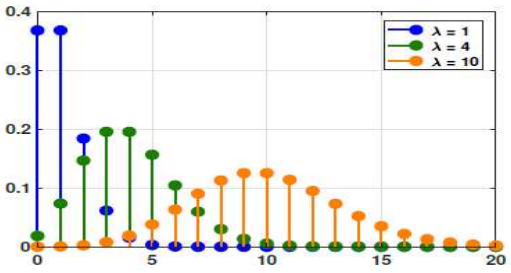
- The Poisson distribution is a probability distribution that describes the number of independent events occurring within a fixed interval of time or space.
- Poisson Process: The distribution is derived from the Poisson process, which is a stochastic process describing a sequence of events occurring in time or space.

- Assumptions:
  - Events occur independently: The occurrence of an event in one interval is independent of the occurrence in any other non-overlapping interval.
  - Constant rate: The average rate at which events occur is constant across time or space intervals.
  - Discreteness: The number of events in each interval is a non-negative integer.

2. *Equations:*

- **Probability Mass Function (PMF):**
  - The PMF of the Poisson distribution gives the probability of observing a specific number of events in a fixed interval.

$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}$$



- **Derivation of PMF**

- The Poisson distribution is often derived from the binomial distribution under certain conditions. The Poisson distribution arises as a limiting case of the binomial distribution when the number of trials (n) becomes large, and the probability of success (p) becomes small, while keeping the product λ = np relatively constant.

$$P(X=k) = \lim_{n\to\infty} \binom{n}{k} P^k (1-p)^{n-k}$$

$$= \lim_{n\to\infty} \binom{n}{k} \frac{\lambda}{n} (1-\frac{\lambda}{n})^{n-k}$$

$$= \lim_{n\to\infty} \frac{n!}{(n-k)!k!} (\frac{\lambda}{n})^k (1-\frac{\lambda}{n})^n (1-\frac{\lambda}{n})^{-k}$$

$$\lim_{n\to\infty} \frac{n!}{(n-k)!} \frac{1}{n^k} = 1$$

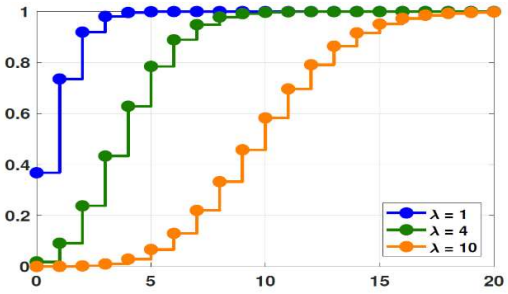
$$= \lim_{n\to\infty} \frac{n!}{(n-k)!} \frac{1}{n^k} \frac{\lambda^k}{k!} e^{-\lambda}$$

$$= \frac{\lambda^k}{k!} e^{-\lambda}$$

- **Cumulative distribution function (CDF)**

- The CDF of a Poisson distribution gives the probability that a random variable X, following a Poisson distribution, is less than or equal to a certain value *k*. The CDF is defined as:

$$f(k,\lambda) = P(0 \le X \le k) = e^{-\lambda} . \sum_{x=0}^{k} \frac{\lambda^x}{x!}$$



- **Derivation of CDF**

- To derive the cumulative distribution function (CDF) for the Poisson distribution, we'll start with the probability mass function (PMF) of the Poisson distribution and then sum up the probabilities for all values up to a given point.

Start with the definition of CDF: $F(x) = P(X \le x)$

Use the PMF for each value up to x: $F(x) = \sum_{k=0}^{x} \frac{e^{-\lambda}.\lambda^k}{k!}$

Simplify the expression: $F(x) = e^{-\lambda} \sum_{k=0}^{x} \frac{\lambda^k}{k!}$

Recognize the sum as the Taylor series expansion of the exponential function: $e^{\lambda} = \sum_{k=0}^{\infty} \frac{\lambda^k}{k!}$

So, we can rewrite the CDF as: $F(x) = 1 - e^{-\lambda} \sum_{k=x+1}^{\infty} \frac{\lambda^k}{k!}$

Simplify further: $F(x) = 1 - e^{-\lambda} \sum_{k=0}^{\infty} \frac{\lambda^{k+x+1}}{(k+x+1)!}$

- This expression involves an infinite sum. In summary, while the derivation involves infinite series, the final expression for the Poisson distribution CDF often relies on mathematical functions beyond basic arithmetic and is usually computed using specialized software or statistical tools.

- **Expectation and Variance**

- The mean (expected value) of a Poisson distribution is denoted by λ, which represents the average rate of events per interval.
- The variance is also λ.
- This means that E(X) = var(X) = λ
- Let us first prove the mean. It can be shown that:

$$\mathbb{E}[X] = \sum_{k=0}^{\infty} k \cdot \frac{\lambda^k}{k!} e^{-\lambda} = \sum_{k=1}^{\infty} \frac{\lambda^k}{(k-1)!} e^{-\lambda}$$
$$= \lambda e^{-\lambda} \sum_{k=1}^{\infty} \frac{\lambda^{k-1}}{(k-1)!} = \lambda e^{-\lambda} \sum_{\ell=0}^{\infty} \frac{\lambda^{\ell}}{\ell!} = \lambda e^{-\lambda} e^{\lambda} = \lambda.$$

- The second moment can be computed as:

$$\begin{aligned}
\mathbb{E}[X^2] &= \sum_{k=0}^{\infty} k^2 \cdot \frac{\lambda^k}{k!} e^{-\lambda} \\
&= \sum_{k=0}^{\infty} k \cdot \frac{\lambda^k}{(k-1)!} e^{-\lambda} \\
&= \sum_{k=0}^{\infty} (k-1+1) \cdot \frac{\lambda^k}{(k-1)!} e^{-\lambda} \\
&= \sum_{k=1}^{\infty} (k-1) \cdot \frac{\lambda^k}{(k-1)!} e^{-\lambda} + \sum_{k=1}^{\infty} \frac{\lambda^k}{(k-1)!} e^{-\lambda} \\
&= \lambda^2 \cdot \underbrace{\sum_{k=2}^{\infty} \frac{\lambda^{k-2} e^{-\lambda}}{(k-2)!}}_{=1} + \lambda \cdot \underbrace{\sum_{k=1}^{\infty} \frac{\lambda^{k-1} e^{-\lambda}}{(k-1)!}}_{=1}.
\end{aligned}$$

- The variance can be computed using:

$$\text{Var}[X] = E[X^2] - [E[X]]^2 = [\lambda^2 + \lambda] - \lambda^2 = \lambda$$

The standard deviation σ will be calculated by $\sqrt{\sigma^2}$

3. *Properties:*

- **Parameters:**

- λ: Average rate of events per interval, $\lambda = \alpha t$, *with t representing time.*
- *k* is a non-negative integer (0, 1, 2, ...) representing the number of events
- *X* represents the values from 0 up to the specified value *k.*

- **Characteristics**:

- Memoryless: The time until the next event is independent of past events.
- Focuses on the number of occurrences, not the timing.

4. *Applications:*

- Common applications include the number of phone calls at a call center in a given hour, the number of accidents at an intersection in a day, or the number of decay events per unit time in radioactive materials.

5. Python Implementation:

This is only a documentation for the implemented python code. Functions full implementation is here: Poisson Random Variable

➢ Generating a Poisson random variable:

```
6   def generate_poisson(lambda_val, size):
```

This Python function, generate_poisson, uses NumPy's random module to generate an array of Poisson-distributed random values with a specified lambda (mean) and size. The lambda_val parameter sets the mean of the Poisson distribution, and size determines the number of random values in the output array. The function returns an array containing Poisson-distributed random values based on the specified parameters.

➢ Calculate PMF:

```
11   def poisson_pmf(k, lambda_val):
```

The function poisson_pmf calculates the probability mass function (PMF) for a Poisson distribution. Here's an explanation of the parameters and the logic used in the function:

- k: This parameter represents the number of events (successes) in a fixed interval for which we want to calculate the probability. It is a non-negative integer.
- lambda_val: This parameter is the average rate or mean number of events in the fixed interval. It determines the shape and central tendency of the Poisson distribution.

The probability mass function (PMF) for a Poisson distribution is given by:
$P(X=k)=k!\lambda ke{-}\lambda$
where:

- P(X=k) is the probability of observing exactly kk events in the fixed interval.
- λ is the average rate of events per interval.
- e is the mathematical constant (approximately 2.71828).
- k! is the factorial of k

➢ Calculate CDF:

```
15   def poisson_cdf(k, lambda_val):
```

The function poisson_cdf calculates the cumulative distribution function (CDF) for a Poisson distribution. Here's an explanation of the parameters and the logic used in the function:

- k: This parameter represents the number of events (successes) for which we want to calculate the cumulative probability. It is a non-negative integer.
- lambda_val: This parameter is the average rate or mean number of events in the fixed interval. It determines the shape and central tendency of the Poisson distribution.

The cumulative distribution function (CDF) for a Poisson distribution is the sum of the individual probabilities up to and including the specified value kk:
$P(X\leq k)=\sum i=0ki!\lambda ie{-}\lambda$
where:

- P(X≤k) is the cumulative probability that the number of events is less than or equal to kk.
- λ is the average rate of events per interval.
- e is the mathematical constant (approximately 2.71828).
- i! is the factorial of ii.

The poisson_cdf function iterates through the values from 0 to kk, accumulating the individual probabilities obtained from the poisson_pmf function. The result is the cumulative probability up to and including kk in the Poisson distribution.

➢ Calculate Variance:

```
26    def poisson_variance(lambda_param):
```

The function poisson_variance return the variance of the poisson distribution

- Var(X) is the variance of the Poisson distribution.
- λ is the average rate of events per interval.

➢ Calculate Expectation:

```
22    def poisson_expectation(lambda_param):
```

The function poisson_expectation calculates the expected value (or mean) of a Poisson distribution. Here's an explanation of the parameter and the logic used in the function:

- lambda_param: This parameter represents the average rate or mean number of events in a fixed interval. It is a key parameter that determines the shape and central tendency of the Poisson distribution.

The expected value of a Poisson distribution is equal to its average rate:

$E(X)=\lambda$

where:

- E(X) is the expected value of the Poisson distribution.
- λ is the average rate of events per interval.

In the context of a Poisson distribution, the expected value represents the average number of events one would expect to observe in a fixed interval based on the specified average rate.

➢ Plotting PMF:

```
31    def plot_poisson_pmf(lambda_val, size):
```

- PMF Calculation: The function computes the Probability Mass Function (PMF) values for a Poisson distribution with parameter λ (average rate) for a range of unique values.
- Visualization: It uses Matplotlib to create a scatter plot and bar plot of the PMF values, showing the probability of each unique value. The x-axis represents the number of events (kk), and the y-axis represents the probability.
- Title and Labels: The function sets appropriate labels and a title for the plot, including the specified average rate (λ), and then displays the plot.

➢ Plotting CDF:

```
43    def plot_poisson_cdf(lambda_val, size):
```

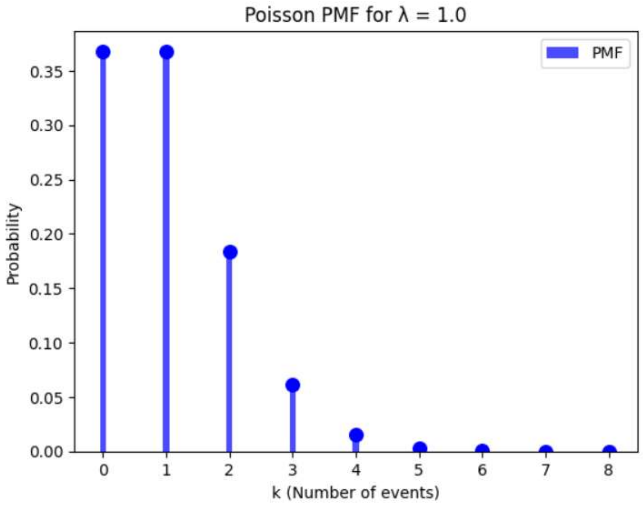
- CDF Calculation: The function calculates the Cumulative Distribution Function (CDF) values for a Poisson distribution with parameter λ (average rate) for a range of unique values.
- Visualization: Using Matplotlib, it creates a step plot of the CDF values, illustrating the cumulative probability up to each unique value. The x-axis represents the number of events (kk), and the y-axis represents the cumulative probability.
- Title and Labels: The function sets appropriate labels and a title for the plot, including the specified average rate (λλ), and then displays the plot.

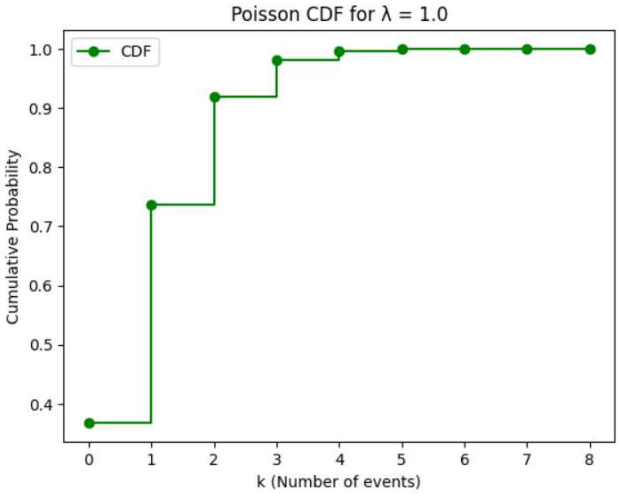
➢ Setting parameters:

```
54    lambda_val = 1.0
55    size = 100000
56    poisson_values = generate_poisson(lambda_val, size)
57    unique_values = np.unique(poisson_values)
```

6. Code results:

```
variance is: 1.0
expectation is: 1.0
```





Continuous random variables:

I. Uniform Random Variables:

1. Definitions:

- Continuous uniform distribution: not all uniform distributions are discrete; some are continuous. A continuous uniform distribution (also referred to as rectangular distribution) is a statistical distribution with an infinite number of equally likely measurable values. Unlike discrete random variables, a continuous random variable can take any real value within a specified range.

2. Equations:

- **Probability Density Function (PDF)**

$$f(x) = \begin{cases} \frac{1}{b-a} & for\ a \le x \le b \\ 0 & for\ x < a\ or\ x > b \end{cases},$$

- **Derivation of PDF**

➢ The continuous uniform distribution is defined over an interval [a,b], where a and b are the lower and upper bounds, respectively.

$$\int_{-\infty}^{\infty} f(x)dx = \int_{a}^{b} \frac{1}{b-a} dx$$

Integrate with respect to x:

$$\int_a^b \frac{1}{b-a}dx = \frac{1}{b-a}\int_a^b dx$$

Evaluate the integral:

$$\frac{1}{b-a}[x]_a^b = \frac{1}{b-a}.(b-a) = 1$$

The integral of the PDF over the entire range is equal to 1, which is a necessary condition for a valid probability density function.

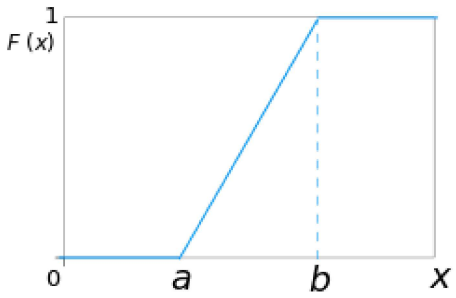
So the PDF for a continuous uniform distribution over the interval [a,b] is:

$$f(x) = \frac{1}{b-a}$$

- This means that the probability of a continuous random variable falling within any subinterval of [a,b] is proportional to the length of that subinterval.

- **Cumulative Distribution Function (CDF)**

$$F(x) = \begin{cases} 0 & for\ x < a, \\ \frac{x-a}{b-a} & for\ a \le x \le b, \\ 1 & for\ x > b. \end{cases}$$



- **Derivation of CDF**
- To derive the CDF F(x), we integrate the PDF from the lower bound (a) to a variable x:

$$F(x) = \int_a^x f(t)\,dt$$

Substitute the PDF into the integral:

$$F(x) = \int_a^x \frac{1}{b-a}\,dt$$

Integrate with respect to t:

$$F(x) = [\frac{t}{b-a}]_a^x$$

Evaluate the upper and lower limits:

$$F(x) = \frac{x-a}{b-a} - \frac{a-a}{b-a}$$

Simplify the expression:

$$F(x) = \frac{x-a}{b-a}$$

- This means that for a≤x≤b, the CDF increases linearly from 0 to 1 as x ranges from the lower bound a to the upper bound b.

- **Expectation and Variance:**

- Expectation:

$$\mathrm{E}(X) = \int_{\mathcal{X}} x \cdot f_X(x)\,\mathrm{d}x\ .$$

with the probability of the continuous uniform distribution, this becomes:

$$\begin{aligned}\mathrm{E}(X) &= \int_a^b x \cdot \frac{1}{b-a}\,\mathrm{d}x \\ &= \left[\frac{1}{2}\,\frac{x^2}{b-a}\right]_a^b \\ &= \frac{1}{2}\,\frac{b^2-a^2}{b-a} \\ &= \frac{1}{2}\,\frac{(b+a)(b-a)}{b-a} \\ &= \frac{1}{2}(a+b)\ .\end{aligned}$$

- The variance of a continuous uniform random variable defined over a<x<b is:

$$\sigma^2 = \frac{(b-a)^2}{12}$$

- **Proof:**
  - The variance is calculated as the integral of $(x-\mu)^2$ multiplied by the PDF over the entire range:

$$\sigma^2 = \int_a^b (x-\mu)^2 \,.\, f(x)\, dx$$

Substitute the expression for μ and f(x):

$$\sigma^2 = \int_a^b (x - \frac{a+b}{2})^2 \,.\, \frac{1}{b-a}\, dx$$

Expand and simplify:

$$\sigma^2 = \frac{1}{b-a}\int_a^b (x^2 - 2x\frac{a+b}{2} + (\frac{a+b}{2})^2)\, dx$$

Use the formulas for the sub of consecutive integers and the sum of consecutive squares:

$$\sigma^2 = \frac{1}{b-a} \,.\, \frac{(b-a)^3}{3}$$

Simplify:

$$\sigma^2 = \frac{(b-a)^2}{12}$$

- *The standard deviation σ will be calculated by* $\sqrt{\sigma^2}$

3. Properties:

- **Parameters:**
  - For a continuous uniform distribution: *a* (lower bound), *b* (upper bound).

- **Characteristics:**
  - Equal Probability: All values in the range have an equal probability of occurring.
  - Constant Probability Density: For continuous uniform distribution, the probability density is constant within the specified interval.
  - Rectangular Shape: The probability distribution function resembles a rectangle for both discrete and continuous uniform distributions.

4. Applications:

- Quality Control
- Finance

5. Python Implementation:

This is only a documentation for the implemented python code. Functions full implementation is here: Uniform Random Variable

Calculate PDF:

```
6    def continuous_uniform_pdf(x, a, b):
```

The continuous_uniform_pdf function defines the Probability Density Function (PDF) for the continuous uniform distribution.
The PDF is defined as 1 / (b - a) for values of x between a and b (inclusive), and 0 elsewhere.
Function Signature:
The function continuous_uniform_pdf takes three parameters:

- x: The variable for which the PDF is evaluated.
- a: The lower limit of the uniform distribution.
- b: The upper limit of the uniform distribution.

Explanation of the PDF Formula:
The PDF of a continuous uniform distribution is a constant value within the interval [a, b] and is zero outside this interval. Implementation using NumPy's np.
where:

- np.where((a <= x) & (x <= b), 1 / (b - a), 0) utilizes NumPy's np.where function to apply the PDF formula conditionally.
- (a <= x) & (x <= b) checks if x falls within the interval [a, b].

If the condition is true, it evaluates to 1 / (b - a), representing the constant PDF within the interval.
If the condition is false, it evaluates to 0, representing the PDF outside the interval.
Return Value:

- The function returns an array of PDF values corresponding to the input values of x.
- The values are 1 / (b - a) within the interval [a, b] and 0 outside this interval.

Calculate CDF:

```
10    def continuous_uniform_cdf(x, a, b):
```

The continuous_uniform_cdf function defines the Cumulative Distribution Function (CDF) for the continuous uniform distribution.

The CDF is defined as 0 for values of x less than a, (x - a) / (b - a) for values between a and b (inclusive), and 1 elsewhere.
Function Signature:
The function continuous_uniform_cdf takes three parameters:

- x: The variable for which the CDF is evaluated.
- a: The lower limit of the uniform distribution.
- b: The upper limit of the uniform distribution.

Explanation of the CDF Formula:
The CDF of a continuous uniform distribution increases linearly within the interval [a, b] and remains constant at 1 outside this interval.
Implementation using NumPy's np.
where:

- np.where(x < a, 0, np.where(x <= b, (x - a) / (b - a), 1)) utilizes NumPy's np.where function to apply the CDF formula conditionally.
- x < a checks if x is less than the lower limit a. If true, it evaluates to 0.
- x <= b checks if x is less than or equal to the upper limit b. If true, it evaluates to (x-a) /(b-a)

If both conditions are false, it evaluates to 1, representing the constant value of the CDF outside the interval.
Return Value:

- The function returns an array of CDF values corresponding to the input values of x.
- The values are 0 for x < a, increase linearly from 0 to 1 with the interval [a, b], and become 1 for x > b.

➢ Calculate Variance:

```
14    def continuous_uniform_variance(a, b):
```

➢ Calculate Expectation:

```
18    def continuous_uniform_expectation(a, b):
```

- The continuous_uniform_variance function calculates the variance of the continuous uniform distribution.
- The continuous_uniform_expectation function calculates the expectation (mean) of the continuous uniform distribution.

Function Signature:
The function continuous_uniform_variance takes two parameters:

- a: The lower limit of the uniform distribution.
- b: The upper limit of the uniform distribution.

Explanation of the Variance Formula:

- The variance of a continuous uniform distribution is calculated using the formula (b-a)**2 / 12.

Implementation:

- The implementation directly applies the formula using Python arithmetic.
- ((b - a)**2) / 12 calculates the variance based on the given lower and upper limits.

Return Value:

- The function returns the calculated variance of the continuous uniform distribution.

➢ Plotting PDF:

```
21    def plot_continuous_uniform_pdf(a, b):
```

Function Signature:

The function plot_continuous_uniform_pdf takes two parameters:

- a: The lower limit of the uniform distribution.
- b: The upper limit of the uniform distribution.

Generate x-values for Plotting:

- x_values = np.linspace(a-2, b+2, 1000) generates a range of 1000 equally spaced x-values for plotting. The range is extended by 2 units on both sides of the interval [a, b] to ensure a clear visualization.

Plotting PDF:

- plt.plot(x_values, continuous_uniform_pdf(x_values, a, b), label='PDF') plots the Probability Density Function (PDF) using the continuous_uniform_pdf function.

Plot Customization:

- plt.title, plt.xlabel, plt.ylabel, and plt.legend are used for customizing the plot with appropriate labels and a legend.

Show the Plot:

- plt.show() displays the generated plot

➢ Plotting CDF:

```
33    def plot_continuous_uniform_cdf(a, b):
```

Function Signature:
The function plot_continuous_uniform_cdf takes two parameters:

- a: The lower limit of the uniform distribution.
- b: The upper limit of the uniform distribution.

Generate x-values for Plotting:

- x_values = np.linspace(a-2, b+2, 1000) generates a range of 1000 equally spaced x-values for plotting. The range is extended by 2 units on both sides of the interval [a, b].

Plotting CDF:

- plt.plot(x_values, continuous_uniform_cdf(x_values, a, b), label='CDF', color='green') plots the Cumulative Distribution Function (CDF) using the continuous_uniform_cdf function. The CDF is plotted in green.

Plot Customization:

- plt.title, plt.xlabel, plt.ylabel, and plt.legend are used for customizing the plot with appropriate labels and a legend.

Show the Plot:

- plt.show() displays the generated plot.

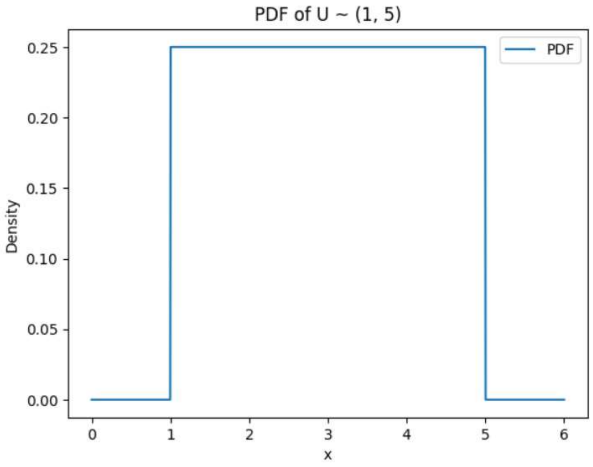
➢ Setting parameters:

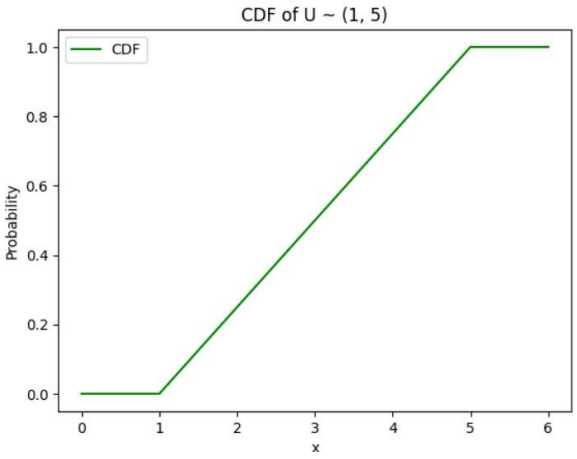
```
a = int(input("Enter the lower limit : "))
b = int(input("Enter the higher limit : "))
```

prompted to enter the lower limit (**a**) and higher limit (**b**) for the continuous uniform distribution.

6. Code results:

```
Variance is :  1.33
Expectation is :  3.0
```





II. Exponential Random Variables:

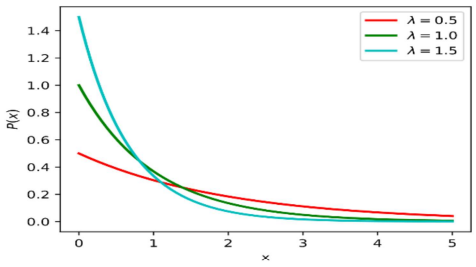
1. Definition:

- The exponential distribution is a fundamental probability distribution that models the time between events in a Poisson process. It is a continuous probability distribution defined for non-negative real numbers.
- Poisson Process: The exponential distribution is closely associated with the Poisson process, which is a stochastic process that models the number of events occurring in fixed intervals of time or space. Key features of the Poisson process include independence of events and a constant average rate of occurrence.

2. Equations:

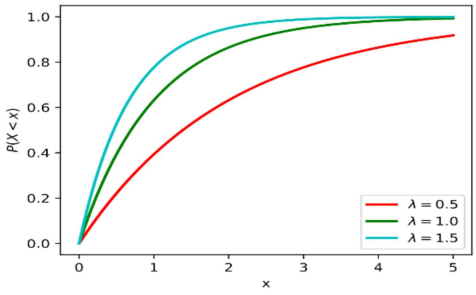
- **Probability Density Function (PDF):**
  - The probability density function of the exponential distribution is given by:

$$\frac{d}{dt}(CDF) = \frac{d}{dt}\left(1 - e^{-\lambda t}\right) = \lambda e^{-\lambda t}$$



- **Cumulative Distribution Function (CDF):**
  - The cumulative distribution function is expressed as:

$$P(T \leq t) = 1 - P(T > t) = 1 - e^{-\lambda t}$$



- **Derivation of PDF & CDF:**
  - To grasp why λ * e$^{(-\lambda t)}$ serves as the probability density function (PDF) for the time until the subsequent event, it is essential to delve into the exponential distribution's definition. The exponential distribution characterizes the probability distribution of the time elapsed between events in a Poisson process.

The concept of the time until the next event implies a period of waiting where no events transpire. This is, in other words, Poisson (X=0).

$$Poisson\ P(X = k) = \frac{\lambda^k}{k!}\ e^{-\lambda}$$

$$\text{P (X = 0)} = \frac{\lambda^0 e^{-\lambda}}{0!} = e^{-\lambda}$$

- One important point about the Poisson PDF is that it models the number of events X occurring in a single unit of time. Given this, how would you model the probability distribution of *"nothing happens during the time duration t"* instead of simply *"during one unit of time"*?
- P (Nothing happens during t time units) = P (X=0 in the first-time unit)

  * P (X=0 in the second time unit)

  * … * P (X=0 in the t-th time unit) = e^−λ * e^−λ * … * e^−λ = e^(-λt)
- The Poisson distribution assumes that events occur independently of one another. So, we can multiply the probability of "no events in a single unit of time (**P (X = 0)**)" **t times** to calculate the probability of "no events (zero success) during a time duration **t**". This results in the expression **e^(-λt)**.
- P(T > t) = P(X=0 during t time units) = e^−λt* T: the random variable of our interest, the waiting time until the first event
  - X: the number of events that follow a Poisson distribution. P (T > t): The probability that the waiting time until the first event is greater than t time units.
  - P(X = 0 in t time units): The probability of observing zero events in t time units.
- A PDF is a derivative of the CDF. So, to find the probability density function (PDF) of an exponential distribution, we can differentiate its cumulative distribution function (CDF), **1 — P (T > t)**.

$$\text{CDF: } P(T \le t) = 1 - P(T > t) = 1 - e^{-\lambda t}$$

$$\text{PDF: } \frac{d}{dt}(CDF) = \frac{d}{dt}\left(1 - e^{-\lambda t}\right) = \lambda e^{-\lambda t}$$

- **Expectation and variance:**
  - Expectation: The mean of the exponential distribution is calculated using the integration by parts.

$$Mean = E[X] = \int_0^\infty x\lambda e^{-\lambda x} dx$$

$$= \lambda\left[\left|\frac{-xe^{-\lambda x}}{\lambda}\right|_0^\infty + \frac{1}{\lambda}\int_0^\infty e^{-\lambda x} dx\right]$$

$$= \lambda\left[0 + \frac{1}{\lambda}\frac{-e^{-\lambda x}}{\lambda}\right]_0^\infty$$

$$= \lambda\frac{1}{\lambda^2}$$

$$= \frac{1}{\lambda}$$

- Variance**:** To find the variance of the exponential distribution, we need to find the second moment of the exponential distribution, and it is given by:

$$E[X^2] = \int_0^\infty x^2 \lambda e^{-\lambda x}\, dx$$

- We can calculate the variance using this formula:

$$Var[X] = E[X^2] - E[X]^2$$

$$Var[X] = \frac{2}{\lambda^2} - \frac{1}{\lambda^2} = \frac{1}{\lambda^2}$$

The standard deviation σ will be calculated by $\sqrt{\sigma^2}$

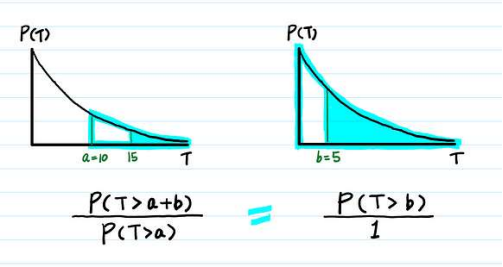
3. Properties:

- **Parameters**:

- x is the random variable representing the time between events,
- λ is the rate parameter, indicating the average number of events per unit time.

- **Memory-lessness:**

- The memoryless property of the exponential distribution can be defined as:
  P (T > a + b | T > a) = P (T > b)
- This means that given an exponential random variable T, the probability that T exceeds a sum of two time periods (a + b) given that it has already exceeded the first period a, is equal to the probability that T exceeds just the second period b.



We already know that $P(T > t) = e^{-\lambda t}$

$$P(T > a + b \mid T > a) = \frac{P(T>a+b)}{P(T>a)} = \frac{e^{-\lambda(a+b)}}{e^{-\lambda a}} = e^{-\lambda b}$$

$$P(T > b) = e^{-\lambda b}$$

- **Exponential Decay:**

- The distribution showcases exponential decay, indicating a constant hazard rate. The probability of an event occurring in the next instant remains constant, regardless of how much time has elapsed.

4. Applications:

- Exponential distribution is widely applied in various fields, including Reliability Engineering, Queuing Theory, Telecommunications and Finance: Modelling.

5. Python Implementation:

This is only a documentation for the implemented python code. Functions full implementation is here: Exponential Random Variable

➢ Calculate PDF:

```
6   def exponential_pdf(x, scale):
```

Function Purpose:

- Defines the Probability Density Function (PDF) for the exponential distribution.

Parameters:

- x: Values at which to evaluate the PDF.
- scale: The scale parameter of the exponential distribution.

Explanation:

- Utilizes the pdf function from the scipy.stats.expon module to calculate the PDF values for given x and scale.

➢ Calculate CDF:

```
10   def exponential_cdf(x, scale):
```

Function Purpose:

- Defines the Probability Density Function (PDF) for the exponential distribution.

Parameters:

- x: Values at which to evaluate the PDF.
- scale: The scale parameter of the exponential distribution.

Explanation:

- Utilizes the pdf function from the scipy.stats.expon module to calculate the PDF values for given x and scale.

➢ Calculate Variance:

```
14   def exponential_variance(scale):
```

Function Purpose:

- Defines the Probability Density Function (PDF) for the exponential distribution.

Parameters:

- x: Values at which to evaluate the PDF.
- scale: The scale parameter of the exponential distribution.

Explanation:

- Utilizes the pdf function from the scipy.stats.expon module to calculate the PDF values for given x and scale.

➢ Calculate Expectation:

```
17  def exponential expectation(lambd):
```

Function Purpose:

- Calculates the expectation (mean) of the exponential distribution.

Parameters:

- lambd: The rate parameter of the exponential distribution (1/scale).

Explanation:

- Expectation formula: 1 / lambd. It ensures that lambd is positive to avoid division by zero.

➢ Plotting PDF:

```
24  def plot_exponential_pdf(scale_param, size=10000):
```

Function Purpose:

- Generates a random sample from an exponential distribution, computes the PDF, and plots the PDF curve.

Parameters:

- scale_param: The scale parameter of the exponential distribution.
- size: Number of samples in the random sample.

Explanation:

- Uses NumPy to generate a random sample from the exponential distribution.
- Computes the PDF values using the previously defined exponential_pdf function.
- Plots the PDF curve.

➢ Plotting CDF:

```
41  def plot_exponential_cdf(scale_param, size=10000):
```

Function Purpose:

- Generates a random sample from an exponential distribution, computes the empirical CDF, and plots the CDF.

Parameters:

- scale_param: The scale parameter of the exponential distribution.
- size: Number of samples in the random sample.

Explanation:

- Uses NumPy to generate a random sample from the exponential distribution.
- Sorts the sample for plotting the empirical CDF.

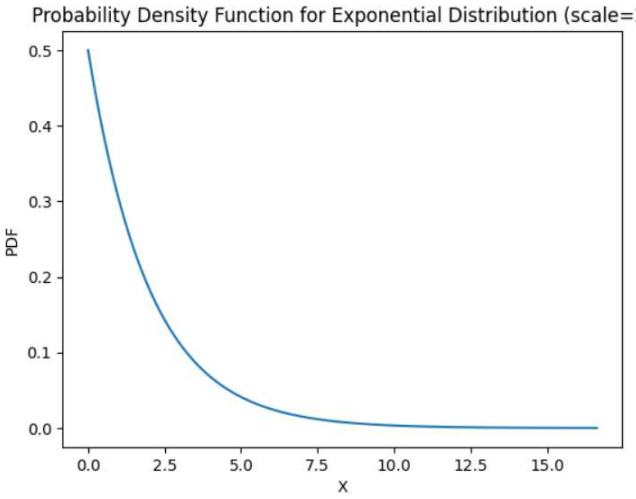
- Computes the CDF values using the previously defined exponential_cdf function.
- Plots the step chart of the CDF.

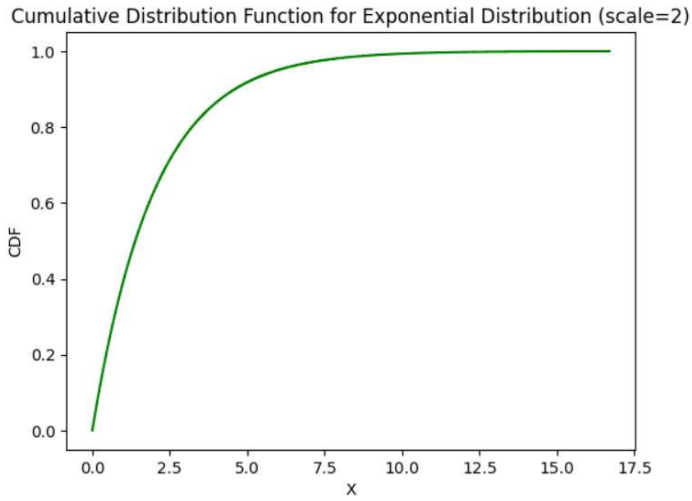
➢ **Setting parameters:**

```
scale_param = int(input("Enter the scale parameter : "))
lambda_param = 1 / scale_param
```

6. *Code results:*

```
Variance is :  4
Expectation is :  2.0
```





IV. Gaussian Random Variables:

1. *Definitions:*

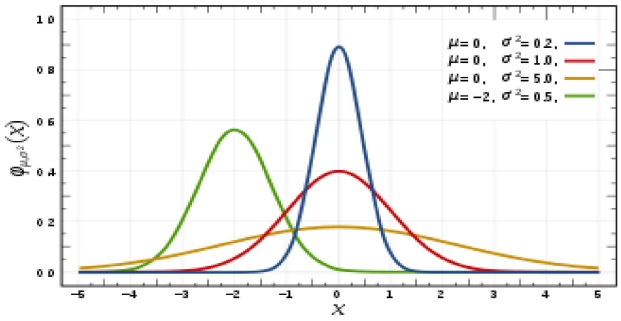
- A random variable with a Gaussian distribution is also called normal distribution. Normal distributions are the most important in statistics and are often used in the natural and social sciences to represent real-valued random variables whose distributions are not known. There are two types of normal distribution: Standard normal distribution and General normal distribution.
- Gaussian Process: A probabilistic model that specifies a distribution over functions is called a Gaussian Process (GP). Put simply, a Gaussian Process represents the uncertainty surrounding predictions by providing a distribution over potential functions rather than a precise estimate for a function at a given input. This makes GPs particularly useful in situations where understanding the uncertainty of predictions is essential.
- Assumptions:

➢ Continuous and Real-Valued Data: The Gaussian distribution is suitable for continuous and real-valued data. It may not be appropriate for discrete or categorical data.

- Symmetry: The distribution is assumed to be symmetric. The mean (μμ) is the central point, and the distribution is symmetric around it. In practice, symmetry may not always hold.
- Single Peak (Unimodal): The Gaussian distribution assumes a single peak (unimodal) shape. It may not accurately model distributions with multiple modes.
- Infinite Range: The Gaussian distribution theoretically extends from negative infinity to positive infinity. For datasets with bounded ranges or specific constraints, alternative distributions might be more appropriate.
- Known Mean and Variance: When using the Gaussian distribution for statistical inference, it is assumed that the mean (μ) and variance (σ^2) are known or estimated accurately from the data.

2. *Equations:*

- **Probability Density Function (PDF):**
  - PDF can be considered as a function which maps each value of the random variable to its frequency.



- **General Normal Distribution:**

The PDF of a normal distribution curve is a mathematical function, and its formula seems daunting, but it is simple as that.

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

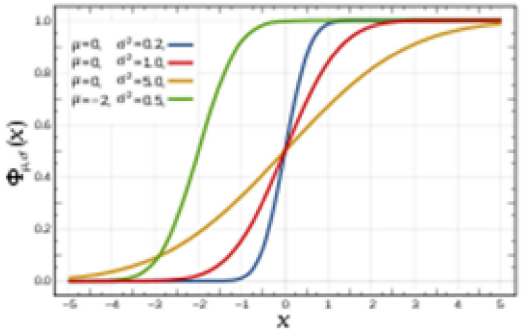
- **Standard Normal Distribution:**

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

- **Derivation of the PDF:**

The PDF is derived from the probability density function of the standard normal distribution (μ=0, σ=1and adjusting for the mean (***μ***) and standard deviation (σ).

- **Cumulative Distribution Function (CDF):**



To find the CDF of the standard normal distribution, we need to integrate the PDF function. In particular, we have

$$\phi(x) = P(Z \le x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-u^2}{2}} du$$

This integral does not have a closed form solution. Nevertheless, because of the importance of the normal distribution, the values of FZ(z) have been tabulated and many calculators and software packages have this function. We usually denote the standard normal CDF by Φ.

$$\phi(x) = P(Z \le x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-u^2}{2}} du$$

- **Derivation of the CDF:**
  - The CDF is derived by integrating the PDF from negative infinity to x. The error function comes into play during this integration. The cumulative distribution function gives the probability that a random variable from the distribution is less than or equal to a given value x.
  - It's important to note that the derivations involve advanced calculus and mathematical concepts. The standard normal distribution is often tabulated or computed using statistical software, and adjustments for different means and standard deviations are made accordingly.
  - If you are interested in the detailed step-by-step derivations, they typically involve techniques from calculus, particularly integration and the normalizing constant for the Gaussian distribution. These derivations are standard components of advanced probability and statistics courses.

- **Expectation and Variance:**
  - **Expectation:**

E[X] = μ

The mean of a normal distribution is equal to its location parameter μ.

  - **Variance:**

Var[X] =σ^2

The variance of a normal distribution is equal to the square of its scale parameter σ .

These formulas hold for any normal distribution, where X is a random variable.

following a normal distribution with mean μ and variance σ^2.

Here, is the location parameter, representing the center of the distribution, and σ^2 is the scale parameter, representing the spread or dispersion of the distribution.

The standard deviation σ will be calculated by $\sqrt{\sigma^2}$.

3. Properties:

- The normal distribution is the only distribution whose cumulants beyond the first two (i.e., other than the mean and variance) are zero. It is also the continuous distribution with the maximum entropy for a specified mean and variance. Geary has shown, assuming that the mean and variance are finite, that the normal distribution is the only distribution where the mean and variance calculated from a set of independent draws are independent of each other.

- The normal distribution is a subclass of the elliptical distributions. The normal distribution is symmetric about its mean, and is non-zero over the entire real line. As such it may not be a suitable model for variables that are inherently positive or strongly skewed, such as the weight of a person or the price of a share. Such variables may be better described by other distributions, such as the log-normal distribution or the Pareto distribution.

- The value of the normal distribution is practically zero when the value xx lies more than a few standard deviations away from the mean (e.g., a spread of three standard deviations covers all but 0.27% of the total distribution). Therefore, it may not be an appropriate model when one expects a significant fraction of outliers—values that lie many standard deviations away from the mean—and least squares and other statistical inference methods that are optimal for normally distributed variables often become highly unreliable when applied to such data. In those cases, a more heavy-tailed distribution should be assumed and the appropriate robust statistical inference methods applied.

- The Gaussian distribution belongs to the family of stable distributions which are the attractors of sums of independent, identically distributed distributions whether or not the mean or variance is finite. Except for the Gaussian which is a limiting case, all stable distributions have heavy tails and infinite variance. It is one of the few distributions that are stable and that have

probability density functions that can be expressed analytically, the others being the Cauchy distribution and the Lévy distribution.

4. Applications:

- The normal (Gaussian) distribution is widely used in business for various purposes, such as: Statistics and Probability Theory, Finance, Physics, Biology and Medicine.

5. Code implementation:

This is only a documentation for the implemented python code. Functions full implementation is here:
Gaussian Random Variable

Calculate PDF:

```
5 ∨ def gaussian_pdf(x, mean, std_dev):
```

Function Purpose:

- Defines the Probability Density Function (PDF) for the Gaussian distribution.

Parameters:

- x: Values at which to evaluate the PDF.
- mean: Mean (expected value) of the Gaussian distribution.
- std_dev: Standard deviation of the Gaussian distribution.

Explanation:

- Utilizes the pdf function from the scipy.stats.norm module to calculate the PDF values for given x, mean, and std_dev

Calculate CDF:

```
8   def gaussian_cdf(x, mean, std_dev):
```

Function Purpose:

- Defines the Cumulative Distribution Function (CDF) for the Gaussian distribution.

Parameters:

- x: Values at which to evaluate the CDF.
- mean: Mean (expected value) of the Gaussian distribution.
- std_dev: Standard deviation of the Gaussian distribution.

Explanation:

- Uses the cdf function from the scipy.stats.norm module to compute the CDF values for given x, mean, and std_dev.

➢ Calculate Variance:

```
11 ∨ def gaussian_variance(std_dev):
```

Function Purpose:

- Calculates the variance of the Gaussian distribution.

Parameters:

- std_dev: Standard deviation of the Gaussian distribution.

Explanation:

- Variance formula: std_dev^2.

➢ Calculate Expectation:

```
14    def gaussian_expectation(mu):
```

Function Purpose:

- Returns the mean of the Gaussian distribution.

Parameters:

- mu: Mean (expected value) of the Gaussian distribution.

Explanation:

- Simply returns the input mu.

➢ Plotting PDF:

```
17    def plot_gaussian_pdf(mean, std_dev, size=10000):
```

Function Purpose:

- Generates a random sample from a Gaussian distribution, computes the PDF, and plots the PDF curve.

Parameters:

- mean: Mean (expected value) of the Gaussian distribution.
- std_dev: Standard deviation of the Gaussian distribution.
- size: Number of samples in the random sample.

Explanation:

- Uses NumPy to generate a random sample from the Gaussian distribution.
- Computes the PDF values using the previously defined gaussian_pdf function.
- Plots the PDF curve.

➢ Plotting CDF:

```
29    def plot_gaussian_cdf(mean, std_dev, size=10000):
```

Function Purpose:

- Generates a random sample from a Gaussian distribution, computes the empirical CDF, and plots the CDF.

Parameters:

- mean: Mean (expected value) of the Gaussian distribution.
- std_dev: Standard deviation of the Gaussian distribution.
- size: Number of samples in the random sample.

Explanation:

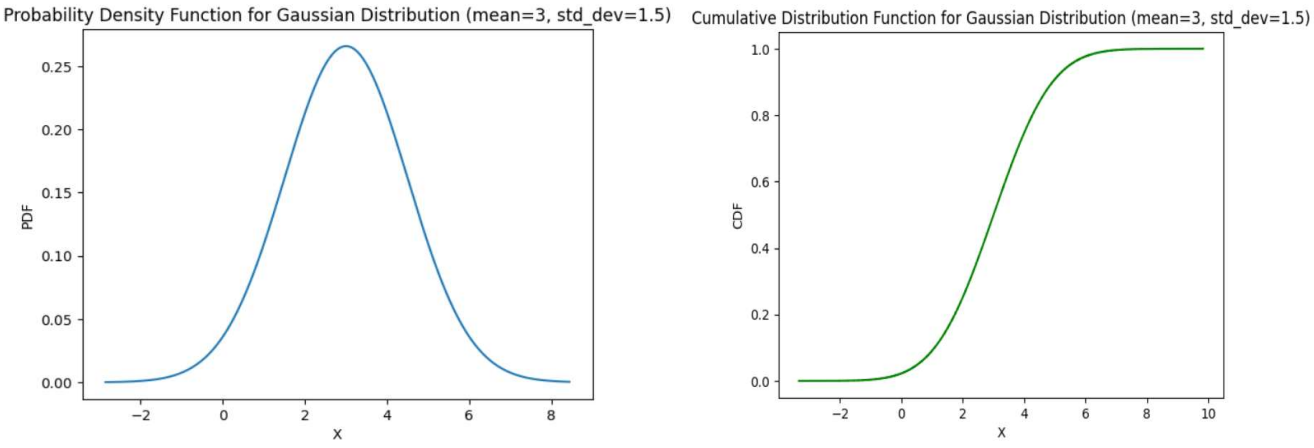
- Uses NumPy to generate a random sample from the Gaussian distribution.
- Sorts the sample for plotting the empirical CDF.
- Computes the CDF values using the previously defined gaussian_cdf function.
- Plots the step chart of the CDF.

➢ Setting parameters:

```
mean = int(input("Enter the mean value of the Gaussian distribution : "))
std_dev = int(input("Enter the standard deviation : "))
```

6. *Code results:*

```
Variance is :  2.25
Expectation is :  3
```



Real Life Applications:

1. Stock analysis:

Introduction:

In the dynamic landscape of financial markets, the ability to predict stock price movements has always been a challenging yet crucial endeavor for investors, traders, and analysts alike. As technology continues to advance, innovative approaches are being employed to gain an edge in the ever-evolving world of finance. One such approach is the utilization of binomial distribution, a powerful statistical tool, to model and analyze stock price movements. This novel application aims to predict whether the closing price of a stock will be positive or negative, providing users with valuable insights to make informed investment decisions.

Why binomial distribution?

The dichotomous nature of the outcomes, represented by 0 for a negative closing price and 1 for a positive closing price, aligns with the fundamentals of binomial distribution. This statistical method is particularly well-suited for scenarios where there are only two possible outcomes, making it an ideal candidate for capturing the binary nature of stock price movements. By leveraging the principles of probability and statistical analysis, this application seeks to enhance the precision of forecasting in the volatile realm of financial markets.

This report will delve into the methodology behind the application, highlighting the key principles of binomial distribution.

Methodology:

Data Collection:

The foundation of this stock prediction application lies in the acquisition of reliable and relevant financial data. To achieve this, we leveraged Yahoo Finance, a reputable source for historical stock data. The chosen stocks' historical closing prices were retrieved for a specified time period, forming the dataset upon which our analysis and predictions are based.

Data Cleaning:

Raw financial data often comes with imperfections, missing values, or outliers that can distort the accuracy of predictions. Therefore, a robust data cleaning process was implemented to ensure the integrity of the dataset. This involved handling missing values, identifying and addressing outliers, and converting the data into a consistent format suitable for subsequent analysis.

Binomial Distribution Parameters:

The core statistical methodology employed in this application is based on binomial distribution. The binomial distribution is characterized by two parameters: probability of success (p) and number of trials (n). In the context of our application, 'success' corresponds to a positive closing price (1), and 'failure' corresponds to a negative closing price (0). To determine these parameters, historical data was

analyzed to estimate the probability of a positive closing price, and the number of trials was set to the total number of historical data points.

Discussion:

Once the binomial parameters were established, The model employs the binomial distribution to calculate the probability of a positive closing price for a given time period. This probability, derived from historical data, serves as the basis for predicting the likelihood of a positive or negative closing price in the future.

By combining data collection, cleaning, exploratory analysis, and the principles of binomial distribution, our methodology forms a comprehensive framework for predicting stock closing prices and contributes to the ongoing exploration of innovative approaches in the realm of financial forecasting.

Results:

The results have offered valuable insights into the potential applications of binomial distribution in forecasting stock closing prices. The predictive model, founded on historical data and statistical principles, demonstrates its ability to discern the likelihood of positive and negative outcomes. As we reflect on the outcomes, several key considerations emerge, shedding light on the significance and limitations of our approach.

The Probability Mass Function (PMF) and Cumulative Distribution Function (CDF) play pivotal roles in interpreting the model's predictions. The PMF provides a probability distribution for discrete outcomes, representing the likelihood of observing a specific closing price (0 or 1) at a given point in time. Meanwhile, the CDF illustrates the cumulative probability of observing a closing price less than or equal to a specified value. These functions offer a nuanced understanding of the probabilities associated with different closing price scenarios, enabling users to make informed decisions based on the risk and reward profiles.

In the context of our application, the variance and expectation further enrich our interpretation of the model's performance. Variance quantifies the dispersion of closing prices around the mean, providing a measure of the model's sensitivity to market fluctuations. On the other hand, the expectation, or mean, provides an average value that serves as a reference point for predicting future closing prices. By analyzing variance and expectation, users can gauge the stability and central tendency of the model's predictions, contributing to a more nuanced understanding of the forecasted outcomes.

It is essential to acknowledge the inherent uncertainties associated with financial markets and the assumptions underlying the binomial distribution model. While our approach leverages historical data to inform predictions, market dynamics can evolve, rendering past patterns obsolete. Additionally, external factors, such as economic events or geopolitical developments, may introduce unforeseen volatility.

Our python implementation:

```
1   import yfinance as yf
2   import matplotlib.pyplot as plt
3   from scipy.stats import binom
4   from datetime import datetime
5   import numpy as np
6   from math import comb
```

- Import libraries.

```
8   start_date = "2019-01-01"
```

- Specify the start date.

```
13   end_date = datetime.today().strftime('%Y-%m-%d')
```

- Set the end date to the current date.

```
16    stock = input("Enter the stock name :")
```

- Choose the stock type you want.

```
18   aapl_data = yf.download(stock, start=start_date, end=end_date, interval="1mo")
```

- Download historical data for AAPL on a monthly basis from the start date to the current date.

```
21   aapl_data.reset_index(inplace=True)
```

- Resetting index for easier manipulation.

```
24   aapl_data.sort_values(by="Date", ascending=False, inplace=True)
```

- Sorting data by date in descending order for data analysis.

```
27   aapl_data['Daily Return'] = aapl_data['Adj Close'].pct_change() * 100
```

- Calculating daily returns (to count number of successes i.e. increasing stock).

```
30   trials = len(aapl_data)
31   success_prob = aapl_data['Daily Return'].gt(0).sum() / len(aapl_data)
```

- Number of days and success probability.

```
34  def generate_binomial(n, p, size):
35      random_variables = np.random.binomial(n=n, p=p, size=size)
36      return random variables  # Return the list of generated random variables
```

- Generate binomial random variables.

```
39  def binomial_pmf(k, n, p):
40      return comb(n, k) * (p ** k) * ((1 - p) ** (n - k))
```

- Function to calculate binomial PMF.

```
43  def binomial_cdf(k, n, p):
44      cdf = 0.0
45      for i in range(k + 1):
46          cdf += binomial_pmf(i, n, p)
47      return cdf
```

- Function to calculate binomial CDF.

```
50  def binomial_expectation(n, p):
51      return n * p
```

- Function to calculate binomial expectation (mean).

```
54  def binomial_variance(n, p):
55      return n * p * (1 - p)
```

- Function to calculate binomial variance.

```
58  def plot_pmf(n, p, size):
59      pmf_values = [binomial_pmf(k, n, p) for k in unique_values]
60      plt.bar(unique_values, pmf_values, color='blue', alpha=0.7, width = 0.4, label='PMF')
61      plt.scatter(unique_values, pmf_values, color='blue', marker='o', s = 70)
62      plt.xticks(unique_values)
63      plt.xlabel('X (Random variable)')
64      plt.ylabel('PMF')
65      plt.title(f'Probability Mass Function for Binomial distribution (p={p:.2f})')
66      plt.legend()
67      plt.show()
```

- Plot PMF.

```
70  def plot_cdf(n, p, size):
71      y_ecdf = [binomial_cdf(k, n, p) for k in unique_values]
72      plt.step(unique_values, y_ecdf, color='green', marker='o', where='post', label='CDF')
73      plt.title(f'Cumulative Distribution Function for Binomial Distribution (p={p:.2f})')
74      plt.xlabel('X (Random variable)')
75      plt.ylabel('CDF')
76      plt.legend()
77      plt.show()
```

- Plot CDF.

```
81  p = success_prob
82  n = trials
83  size = 1000
```

- Set parameters extracted from the data.

```
86  binomial_random_variables = generate_binomial(n, p, size)
```

- Store generated binomial random variables.

```
89  unique_values, counts = np.unique(binomial_random_variables, return_counts=True)
```

- Make an array of generated random variables and take unique values.

```
93    print("variance is:", binomial_variance(n, p))
94    print("expectation is:", binomial_expectation(n, p))
```

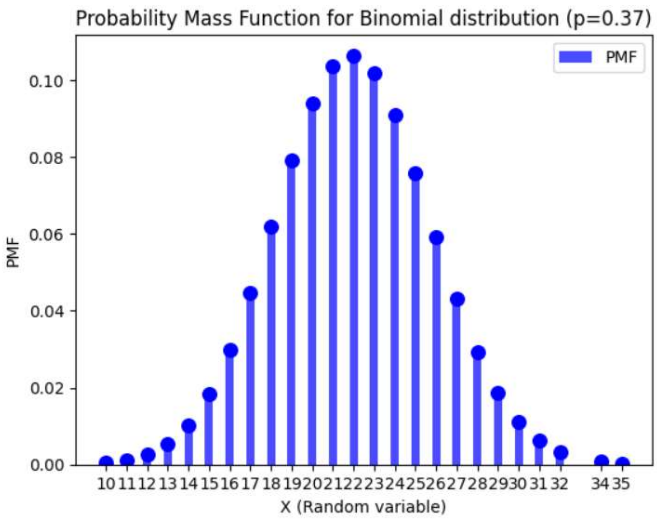
- Print expectation and variance.

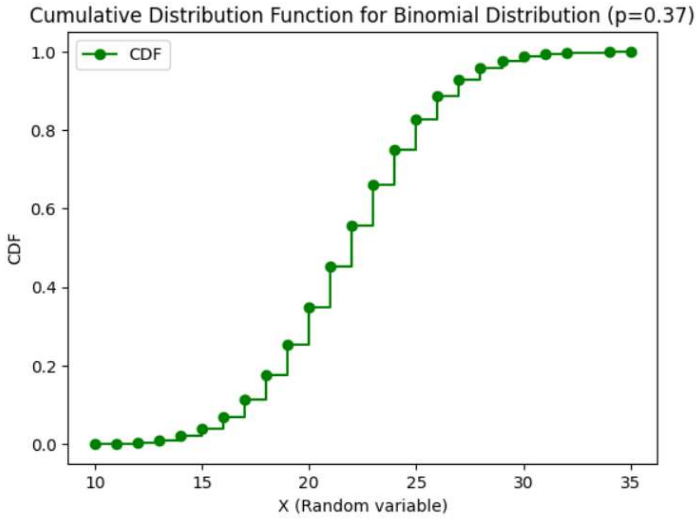
```
97    plot_pmf(n, p, size)
98    plot_cdf(n, p, size)
```

- Show results.

Code results:

```
[**********100%%**********]  1 of 1 completed
variance is: 13.933333333333334
expectation is: 22.0
```





Conclusion:

In conclusion, our stock prediction application, driven by binomial distribution modeling, presents a valuable tool for investors seeking probabilistic insights into stock closing prices. The PMF, CDF, variance, and expectation contribute to a comprehensive analysis of the model's performance, offering users a nuanced understanding of the associated risks and rewards. As we continue to refine and validate our approach, the application stands as a testament to the ongoing exploration of innovative methodologies in the ever-evolving landscape of financial forecasting.

2. Phone tracker:

Introduction:

Background:

In the era of mobile communication dominance, predicting incoming phone call frequencies is crucial for applications like network optimization and resource planning. The application focuses on mathematical modeling, specifically utilizing exponential distribution, to forecast the arrival pattern of phone calls to a mobile device. This distribution is chosen for its versatility in modeling time between successive call arrivals, making it ideal for understanding the temporal dynamics of mobile communication.

Why using exponential distribution in particular:

the exponential distribution is chosen for modeling phone call expectations due to its memoryless property, constant rate of arrival, suitability for independent and identically distributed events, ease of parameterization, and applicability to inter-event times. These characteristics align with the assumption that phone call arrivals exhibit random and independent behavior, making the exponential distribution a practical and effective choice for this application.

Objective:

The primary goal of the application is to empower users to analyze and visualize the expected distribution of incoming phone calls over time. By employing the exponential distribution model, users can optimize various aspects of mobile communication management, such as network capacity planning and call center staffing. The report details the conceptual framework, principles of the exponential distribution, and real-world applications of the model. It also provides a step-by-step guide on using the application to make informed decisions in the realm of mobile communication, aiming to enhance users' understanding of communication patterns for more efficient and responsive systems.

Methodology:

Data Collection:

The foundation of our modeling process lies in the acquisition of relevant data. To obtain a dataset representative of real-world phone call patterns, we sourced call records from publicly available datasets on the internet. These records typically include timestamps for each call received on the designated mobile device.

1	index	date	duration	item	month	network	network_type
2	0	15/10/14 06:58	34.429	data	2014-11	data	data
3	1	15/10/14 06:58	13	call	2014-11	Vodafone	mobile
4	2	15/10/14 14:46	23	call	2014-11	Meteor	mobile
5	3	15/10/14 14:48	4	call	2014-11	Tesco	mobile
6	4	15/10/14 17:27	4	call	2014-11	Tesco	mobile
7	5	15/10/14 18:55	4	call	2014-11	Tesco	mobile
8	6	16/10/14 06:58	34.429	data	2014-11	data	data
9	7	16/10/14 15:01	602	call	2014-11	Three	mobile
10	8	16/10/14 15:12	1050	call	2014-11	Three	mobile
11	9	16/10/14 15:30	19	call	2014-11	voicemail	voicemail
12	10	16/10/14 16:21	1183	call	2014-11	Three	mobile
13	11	16/10/14 22:18	1	sms	2014-11	Meteor	mobile
14	12	16/10/14 22:21	1	sms	2014-11	Meteor	mobile

Model fitting:

We created a model by extracting exponential distribution parameters from the given data so we can expect the probability of getting a call after a certain time. We calculated the time interval between each call and stored it in time_diff list so we can perform our calculations.

Our python implementation:

```
# List of time intervals
intervals = [
    "16/10/14 06:58 - 16/10/14 15:01",
    "16/10/14 15:01 - 16/10/14 15:12",
    "16/10/14 15:12 - 16/10/14 15:30",
    "16/10/14 15:30 - 16/10/14 16:21",
    "16/10/14 16:21 - 16/10/14 22:18",
    "16/10/14 22:18 - 16/10/14 22:21",
    "17/10/14 06:58 - 17/10/14 10:53",
    "17/10/14 10:53 - 17/10/14 11:19",
    "17/10/14 11:19 - 17/10/14 11:20",
    "17/10/14 11:20 - 17/10/14 17:22",
    "17/10/14 17:22 - 17/10/14 17:23",
    "17/10/14 17:23 - 17/10/14 17:26",
    "17/10/14 17:26 - 17/10/14 17:29",
    "17/10/14 17:29 - 17/10/14 17:30",
    "17/10/14 17:30 - 17/10/14 17:42",
    "17/10/14 17:42 - 17/10/14 17:44",
    "17/10/14 17:44 - 17/10/14 17:44",
    "17/10/14 17:44 - 17/10/14 17:44",
    "18/10/14 06:58 - 18/10/14 11:51",
    "18/10/14 11:51 - 18/10/14 12:06",
    "18/10/14 12:06 - 18/10/14 12:06",
    "18/10/14 12:06 - 18/10/14 13:08",
    "18/10/14 13:08 - 18/10/14 13:10",
    "18/10/14 13:10 - 18/10/14 14:01",
    "18/10/14 14:01 - 18/10/14 18:52",
    "18/10/14 18:52 - 18/10/14 20:44",
    "18/10/14 20:44 - 18/10/14 21:04",
    "18/10/14 21:04 - 18/10/14 21:06",
    "18/10/14 21:06 - 18/10/14 21:23",
    "18/10/14 21:23 - 18/10/14 22:37",
    "19/10/14 06:58 - 19/10/14 14:47",
    "19/10/14 14:47 - 19/10/14 15:46",
    "19/10/14 15:46 - 19/10/14 16:21",
    "19/10/14 16:21 - 19/10/14 16:30",
    "19/10/14 16:30 - 19/10/14 20:25",
    "20/10/14 06:58 - 20/10/14 09:43",
    "20/10/14 09:43 - 20/10/14 09:43",
    "20/10/14 09:43 - 20/10/14 13:55",
    "20/10/14 13:55 - 20/10/14 13:56",
    "20/10/14 13:56 - 20/10/14 18:14",
    "20/10/14 18:14 - 20/10/14 18:24",
    "20/10/14 18:24 - 20/10/14 19:59",
    "20/10/14 19:59 - 20/10/14 20:16",
    "21/10/14 06:58 - 21/10/14 16:17",
    "22/10/14 06:58 - 22/10/14 12:04"
]
```

```
def convert_to_datetime(timestamp_str):
    return datetime.strptime(timestamp_str, "%y/%m/%d %H:%M")

# List to store time intervals in minutes
time_intervals = []
```

- "**Convert_to_datetime**" function that takes a string representing a timestamp as its argument and returns a corresponding datetime object. And then declared a list to store the time intervals in minutes.

```
# Iterate through each pair of timestamps and calculate the interval
for interval in intervals:
    start_str, end_str = interval.split(" - ")
    start_time = convert_to_datetime(start_str)
    end_time = convert_to_datetime(end_str)
    interval_minutes = (end_time - start_time).total_seconds() / 60
    time_intervals.append(interval_minutes)
```

- A loop to calculate the time intervals.

```
time_diff_hours = np.array(time_intervals) / 60.0
```

- NumPy library that takes the array time_intervals and divides each element by 60.0 to convert the time intervals from minutes to hours.

```
plot_exponential_pdf(1 / lambda_param)
plot_exponential_cdf(1 / lambda_param)
```

```
print(f"Estimated lambda: {lambda_param}")
```

- Parameters and printing lambda.

* The exponential distribution is defined by a single parameter, the rate (λ), which represents the average number of events (phone calls, in this case) occurring per unit of time. The rate is inversely proportional to the mean time between events.

```
plot_exponential_pdf(1 / lambda_param)
plot_exponential_cdf(1 / lambda_param)
```

- Calling the plotting functions from the main code to draw the graph.

```
Variance = exponential_variance(scale_param)
Expectation = exponential_expectation(lambda_param)

#printing variance and expectation
print("Variance is: ", round(Variance, 2))
print("Expectation is: ", round(Expectation, 2))
```

- Calculating the variance and the expectation then printing them.

Code Results:

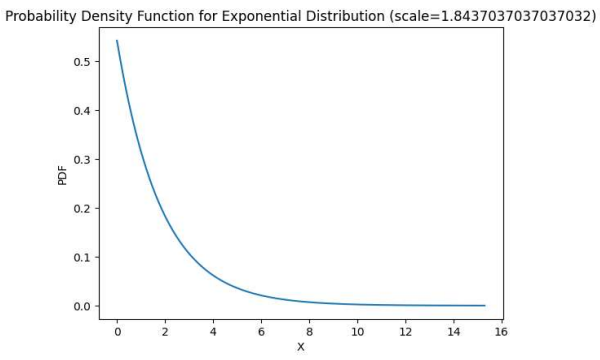
Descriptive Statistics:

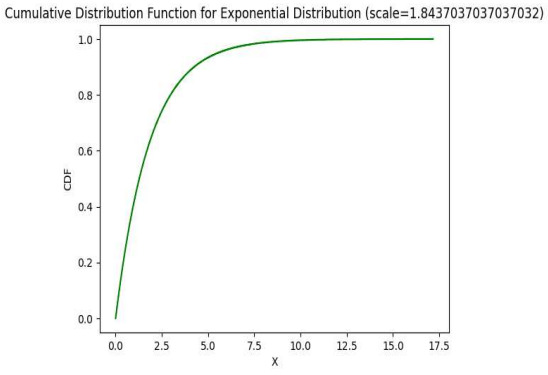
When the code is executed, it will compute mean, expectation, and lambda for the collected data:

```
Estimated lambda: 0.5423865006026518
Variance is:  3.4
Expectation is:  1.84
```

Exponential Distribution Fit:

To visualize the Probability Distribution Function (PDF) and Cumulative Distribution Function (CDF) for the Exponential Distribution in a graph, we employed the matplotlib library, a widely used tool for creating plots. The resulting PDF and CDF plots are displayed as follows:





Prediction:

As illustrated in the graph, the peak likelihood is observed as x approaches 0.5, aligning with the computed λ. The rate parameter, λ, signifies the average number of events per unit time. To evaluate the probability of receiving a call within the next 2.5 hours, we can consult the cumulative distribution function (CDF) graph. The graph indicates that $p(x \leq 2.5)$ equals 0.743, signifying a 74.3% probability of receiving a call in the upcoming 2.5 hours.

Discussion:

Why exponential distribution in particular:

The decision to utilize the exponential distribution in this application is rooted in its inherent characteristics. The distribution is well-suited for modeling processes characterized by constant, independent events occurring over time. In the context of phone call arrivals, the exponential distribution aligns with the assumption that calls arrive independently, and the time between calls follows an exponential pattern. This makes it a natural choice for predicting the inter-arrival times of phone calls.
And here are the key characteristics of the exponential distribution that make it suitable for this application:

1. **Memoryless Property:**
    - One of the defining features of exponential distribution is its memoryless property. This means that the probability of an event occurring in the future is independent of the past. In the context of phone call arrivals, this property implies that the likelihood of receiving a call in the next time interval is not influenced by the time already spent without receiving a call. This characteristic is often observed in scenarios where events occur randomly and independently over time.
2. **Constant Rate of Arrival:**
    - The exponential distribution is characterized by a constant rate of arrival, denoted as λ. This rate represents the average number of events per unit of time. In the context of phone calls, λ corresponds to the average rate of incoming calls. The assumption of a constant rate is reasonable for modeling scenarios where calls are generated by independent and identically distributed processes, and there is no inherent periodicity in the arrivals.
3. **Independent and Identically Distributed Events:**
    - The exponential distribution is well-suited for modeling a sequence of independent and identically distributed events. In the case of phone calls, each call arrival is considered an independent event, and the distribution assumes that the time between calls follows the same exponential pattern throughout the observation period. This assumption simplifies the modeling process and facilitates parameter estimation.
4. **Ease of Parameterization:**
    - The exponential distribution has only one parameter, λ, which represents the rate of arrivals. Estimating this single parameter is relatively straightforward, often involving the reciprocal of the mean time between events. The simplicity of parameterization makes exponential distribution an accessible choice, especially when dealing with

real-world datasets where complex models may be impractical or computationally intensive.

5. **Applicability to Inter-Event Times:**
    - The exponential distribution is particularly well-suited for modeling the time between successive events. In the context of phone call arrivals, it accurately describes the distribution of inter-arrival times, providing a probabilistic framework for understanding the temporal dynamics of incoming calls.

- **Limitations and future considerations:**

Acknowledging the limitations of the model is crucial for a nuanced interpretation of the results. Factors such as changes in user behavior, network conditions, or external events may introduce complexities not accounted for in the current model. Future iterations may explore more sophisticated models or incorporate additional parameters to enhance predictive accuracy.

Conclusion:

In the ever-evolving landscape of mobile communication, understanding and predicting the patterns of incoming phone calls play a pivotal role in optimizing resource allocation, enhancing network efficiency, and facilitating proactive decision-making. This application, centered around modeling phone call expectations, leverages exponential distribution to provide a robust and accessible tool for users seeking insights into call arrival dynamics.

The results obtained from this modeling endeavor offer a valuable perspective on the temporal characteristics of phone call arrivals. Through the estimation of the rate parameter (λ), the exponential distribution successfully captures the memoryless, constant rate of arrival inherent in scenarios where phone calls are considered independent events occurring randomly over time.

The decision to employ exponential distribution is justified by its fundamental characteristics, including the memoryless property, constant rate of arrival, and suitability for modeling independent and identically distributed events. These characteristics align seamlessly with the assumed behavior of phone call arrivals, making the exponential distribution an apt choice for this application.

The practical implications of the modeled phone call expectations are significant, empowering decision-makers to allocate resources efficiently, plan for peak call times, and enhance overall communication infrastructure. By providing a user-friendly interface and transparent documentation, this application aims to bridge the gap between theoretical modeling and actionable insights, enabling users to make informed decisions based on probabilistic predictions.

As with any modeling approach, it is essential to acknowledge the limitations inherent in assumptions and the simplifications made during the modeling process. The discussion section highlights potential avenues for refinement and future considerations, emphasizing the continuous evolution and adaptation of the model to changing real-world conditions.

In conclusion, the application successfully demonstrates the utility of the exponential distribution for modeling phone call expectations. It contributes to the field of mobile communication management by providing a practical and accessible tool that aligns with the assumed characteristics of call arrivals. As technology continues to advance and communication patterns evolve, the application serves as a foundation for ongoing research and development in the optimization of mobile communication systems.

3. Literacy rate analysis:

Introduction:

In this analysis, we employed the normal distribution to model data related to the literacy rate in various districts in India. As part of the Department of Education in a large nation, our goal was to explore the distribution characteristics of district literacy rates and identify potential outliers using z-scores and derive conclusions.

Libraries and Data Import:

We began by importing essential Python libraries, including Numpy, pandas, matplotlib, scipy.stats, and statsmodels. These libraries facilitated data manipulation, visualization, and statistical analysis. The dataset, named 'education_districtwise.csv,' was loaded into a pandas DataFrame, and missing values were handled using the dropna() function.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
from scipy import stats
```

✓ 0.0s

```
# Read CSV file into DataFrame and drop missing values
education_districtwise = pd.read_csv('Districtwise.csv')
education_districtwise = education_districtwise.dropna()
```

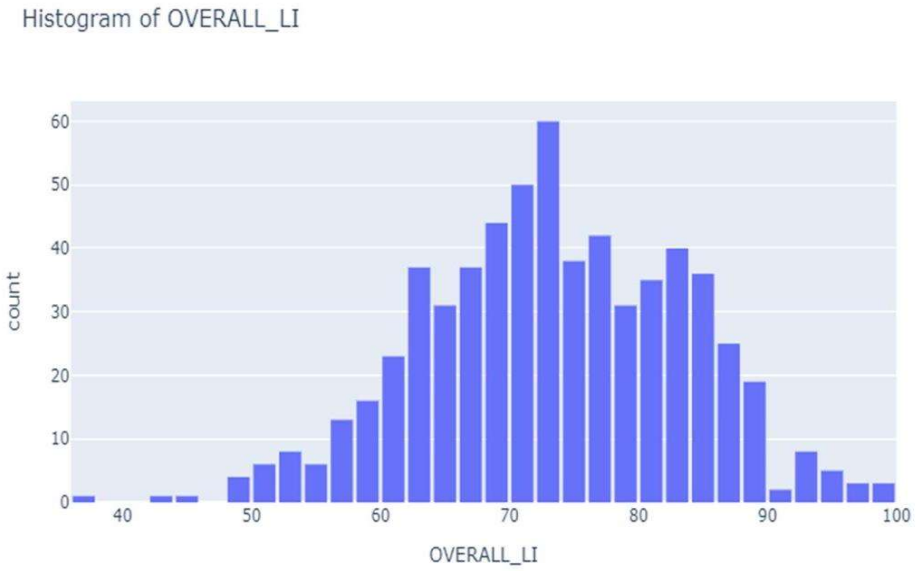
✓ 0.0s

Histogram Plot:

To understand the distribution of district literacy rates, we created a histogram using the 'OVERALL_LI' column. The resulting plot exhibited a bell-shaped curve, indicative of a symmetric and approximately normal distribution. This observation prompted us to consider normal distribution as a potential model for our data.

```
# Plot a histogram of the 'OVERALL_LI' column with styling options using Plotly
hist_fig = px.histogram(education_districtwise, x='OVERALL_LI', nbins=40,
                        labels={'OVERALL_LI': 'OVERALL_LI'}, title='Histogram of OVERALL_LI')
hist_fig.update_layout(bargap=0.1, bargroupgap=0.05)
hist_fig.show()
```

✓ 0.2s

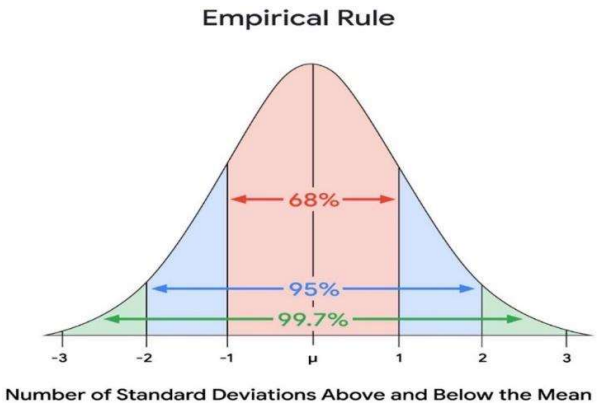


Empirical Rule Verification:

We verified the applicability of the empirical rule to our dataset, which states that for a normal distribution:

- 68% of values fall within +/- 1 standard deviation from the mean.
- 95% of values fall within +/- 2 standard deviations from the mean
- 99.7% of values fall within +/- 3 standard deviations from the mean.

.



Using the mean and standard deviation of district literacy rates, we computed the actual percentages falling within each range and found close.

```
# Calculate mean and standard deviation of 'OVERALL_LI' column
mean_overall_li = education_districtwise['OVERALL_LI'].mean()
std_overall_li = education_districtwise['OVERALL_LI'].std()
print("Mean of OVERALL_LI:", mean_overall_li)
print("Standard Deviation of OVERALL_LI:", std_overall_li)

for i in range(1, 4):
    lower_limit = mean_overall_li - i * std_overall_li
    upper_limit = mean_overall_li + i * std_overall_li
    percentage_within_limit = ((education_districtwise['OVERALL_LI'] >= lower_limit) & (education_districtwise['OVERALL_LI'] <= upper_limit)).mean()
    print(f"\nPercentage of data within {i} standard deviations from the mean: {percentage_within_limit:.2%}")
```

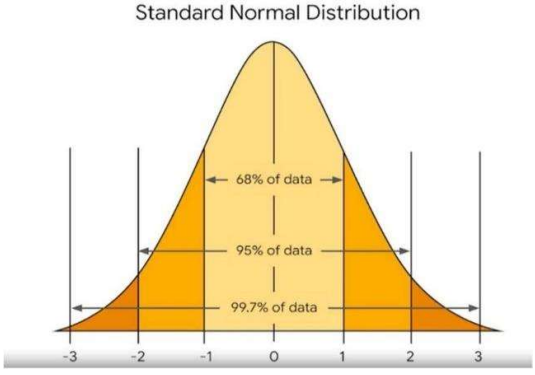
Output:

```
Percentage of data within 1 standard deviations from the mean: 66.56%

Percentage of data within 2 standard deviations from the mean: 95.36%

Percentage of data within 3 standard deviations from the mean: 99.68%
```

agreement with the empirical rule (66.4%, 95.4%, and 99.6%).



Z-Scores and Outlier Detection: Next, we calculated z-scores for each district's literacy rate, providing a measure of how many standard deviations a data point is from the mean. This information was crucial for identifying outliers. Districts with z-scores smaller than -3 or larger than +3 were considered outliers. We identified two such outliers, namely DISTRICT461 and DISTRICT429, both exhibiting significantly lower literacy.

```
# Calculate Z-scores and add a 'Z_SCORE' column to the DataFrame
education_districtwise['Z_SCORE'] = stats.zscore(education_districtwise['OVERALL_LI'])
print("DataFrame with Z-Scores:")
print(education_districtwise)

# Identify and print rows with Z-scores greater than 3 or less than -3 (potential outliers)
outliers = education_districtwise[(education_districtwise['Z_SCORE'] > 3) | (education_districtwise['Z_SCORE'] < -3)]
print("Rows with Z-Scores beyond 3 standard deviations (potential outliers):")
print(outliers)
```

Output:

```
Rows with Z-Scores beyond 3 standard deviations (potential outliers):
      DISTNAME  OVERALL_LI  TOTPOPULAT   Z_SCORE
434  DANTEWADA       42.67    532791.0 -3.030890
494  ALIRAJPUR       37.22    728677.0 -3.569821
```

Conclusion:

In conclusion, this analysis demonstrated the utility of the normal distribution in modeling district literacy rates. The agreement with the empirical rule and the identification of outliers using z-scores enhance our understanding of the distribution characteristics and outliers in the dataset. This information can be valuable for educational policymakers, allowing them to allocate resources more effectively, address specific districts with lower literacy rates, and contribute to overall educational improvements

4. Hospital Admissions:

Introduction:

Background:

Hospital admissions are a critical metric for healthcare institutions to efficiently allocate resources and plan for patient care. Understanding the distribution of expected numbers of people seeking admission to a hospital can aid in optimizing staff and facility management.

Objective:

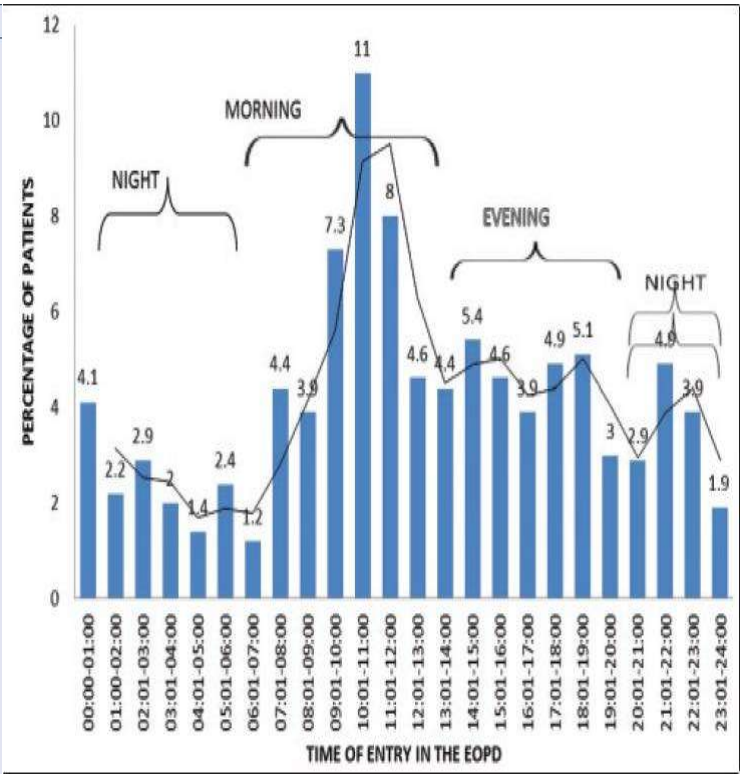
The objective of this study is to model the hospital admissions using the Poisson distribution, which is commonly employed in situations where events occur independently and at a constant average rate.

Methodology:

Data Collection:

The hospital admissions data utilized in this study was sourced from National library of medicine, an official website of the United States government website that regularly publishes relevant healthcare metrics. The website is known for its comprehensive and up-to-date information on hospital admissions, making it a reliable source for our analysis.

Variable	Number (N = 591)	Percentage
Age group (in years)		
0-14	31	5.2
15-29	146	24.7
30-44	125	21.2
45-59	165	27.9
60-74	97	16.4
>75	27	4.6
Gender		
Male	415	70.2
Female	176	29.8
Current place of residence		
Punjab	213	36.0
Chandigarh	114	19.3
Haryana	101	17.1
Himachal Pradesh	89	15.1
Uttar Pradesh	43	7.3
Other states	31	5.2
Health state		
Conscious	502	84.9
Unconscious	82	13.9
Semi-conscious	7	1.2
Referral status		
Health facility referred	377	63.8
Self-referred	214	36.2



Poisson Distribution:

The Poisson distribution is defined by a single parameter, λ (lambda), which represents the average rate of events in a fixed interval. The probability mass function (PMF) of the Poisson distribution is given by: $P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}$

where X is the number of events, k is a non-negative integer, λ is the average rate of events.

Model Fitting:

From the gathered data, we made a function to calculate the average number of events.

```
# Calculate overall average lambda
1 usage
def calculate_overall_average_lambda(people):
    # Initialize an empty list to store average lambdas for each interval
    average_lambdas = []
    # Initialize an empty list to store the generated samples
    generated_samples = []
    # Iterate over each number of people
    for num_people in people:
        # Generate a random sample from a Poisson distribution with the given lambda
        sample = np.random.poisson(num_people, 1000)
        # Calculate the average lambda for the current interval
        average_lambda = np.mean(sample)
        # Append the average lambda to the list
        average_lambdas.append(average_lambda)
        # Append the generated sample to the list
        generated_samples.append(sample)
    # Calculate the overall average of lambdas
    overall_average_lambda = np.mean(average_lambdas)
    # Print the overall average
    print("Overall Average Lambda:", overall_average_lambda)
    # Return the generated samples and the overall average lambda
    return generated_samples, overall_average_lambda
```

- function to calculate the average of given data. And the data goes as follows:

```
# Example using the provided data
people = [
    24.231, 13.002, 17.139, 11.82, 8.274, 14.184,
    7.092, 26.004, 23.049, 43.143, 65.01, 47.28,
    27.186, 26.004, 31.914, 27.186, 23.049, 28.959,
    30.141, 17.73, 17.139, 28.959, 23.049, 11.229
]
```

- Given data When the function is used on the given data, it calculates the average value of λ which is ≈ 24.69.

Code Results:

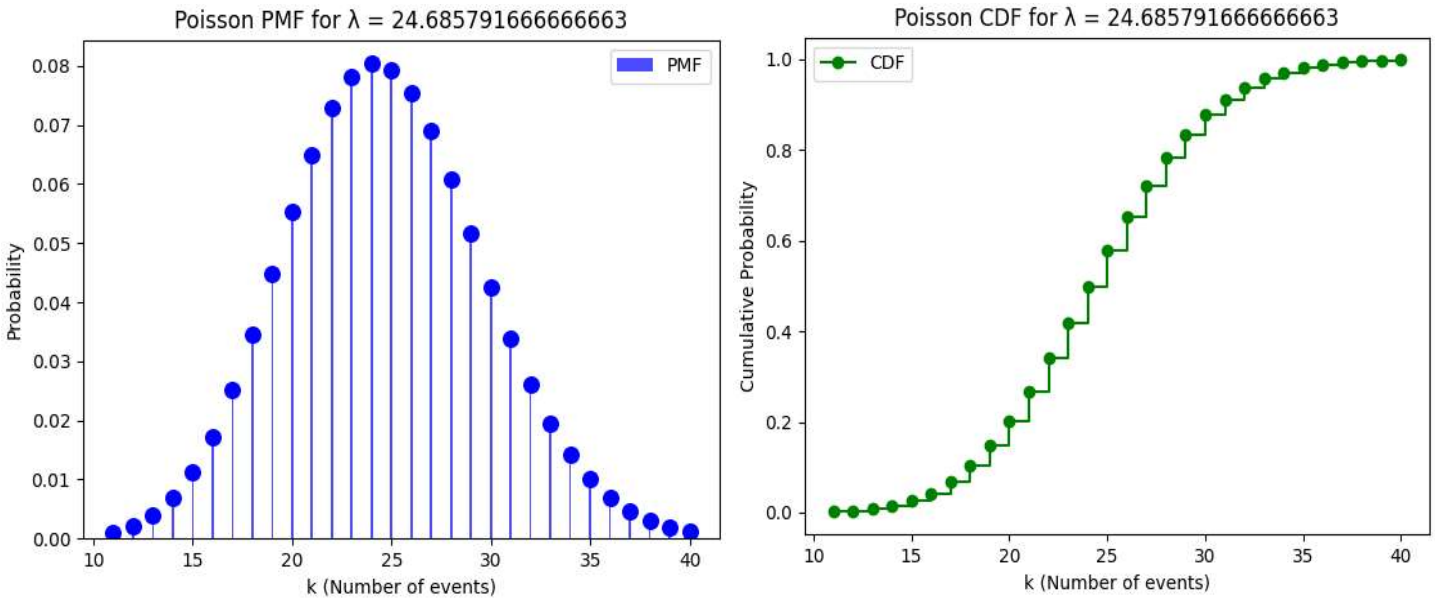
Descriptive Statistics:

When the code is executed, it will compute various statistics for the collected hospital admissions data, such as the mean, variance, and other pertinent metrics. In a Poisson distribution, it is recognized that the mean and the variance are both equal to the rate of events (λ).

```
Overall Average Lambda: 24.685791666666663
Mean and variance = 24.685791666666663
```

Poisson Distribution Fit:

To visualize the Probability Mass Function (PMF) and Cumulative Distribution Function (CDF) for the Poisson Distribution in a graph, we employed the matplotlib library, a widely used tool for creating plots. The resulting PMF and CDF plots are displayed as follows:



Prediction:

As depicted in the graph, the highest probability occurs when K approaches 24, which corresponds to the calculated λ. This λ represents the average number of people admitted to the hospital in any given hour. If the hospital wishes to assess the probability of 40 people being admitted to the hospital during the same hour, they can refer to the graph. The graph illustrates that it is highly unlikely (though not impossible) for 40 people to be admitted simultaneously.

Discussion:

Implications:

- **Probability Estimation:**

The Poisson distribution provides a useful framework for estimating the probability of a specific number of hospital admissions within a fixed time interval.

- **Average Rate Representation:**

The calculated λ (average rate) from the Poisson distribution reflects the expected number of admissions per unit time, aiding in resource planning and management.

- **Identification of Unlikely Events:**

The distribution allows for the identification of events that are highly unlikely, helping hospitals anticipate and prepare for rare occurrences.

- **Suitability for Rare Events:**

Well-suited for modelling rare events, where the probability of multiple admissions within a short time frame is generally low.

4.2 Limitations

- **Constant Rate Assumption:**

Assumes a constant admission rate, which may not accurately reflect real-world variations due to seasonal, temporal, or external factors.

- **Independence Assumption:**

Assumes independence of admissions, overlooking situations where events may be correlated, especially during emergencies or peak hours.

- **Inability to Capture Clustering:**

Fails to capture clustering of events, especially when certain medical conditions lead to increased admissions for specific patient groups.

Conclusion:

This application employed the Poisson distribution to model hospital admissions, utilizing Python with NumPy, Matplotlib, and Math libraries. By simulating realistic scenarios and analysing the generated data, we gained insights into the patterns and frequency of patient arrivals. The application of the Poisson distribution to hospital admissions holds practical value for resource allocation, capacity planning, and overall healthcare management. Visualizations, such as probability mass functions and cumulative distribution functions, aided in representing the distribution effectively. Overall, the project successfully demonstrated the utility of the Poisson distribution in modelling hospital admissions, contributing valuable insights to healthcare analytics.

Used Libraries in Code

- **NumPy:** Numerical Python; a library for numerical operations on large, multi-dimensional arrays and matrices.
- **Matplotlib:** A 2D plotting library for creating static, animated, and interactive visualizations in Python.
- **Math:** A built-in Python library providing mathematical functions and operations.
- **yfinance:** A library for fetching financial data from Yahoo Finance.
- **Pandas:** A data manipulation and analysis library, offering data structures like DataFrame for efficient data handling.
- **Seaborn:** A statistical data visualization library based on Matplotlib, providing a high-level interface for attractive and informative statistical graphics.
- **Plotly Express:** A high-level interface for creating interactive visualizations with Plotly, simplifying the generation of complex plots.
- **SciPy:** An open-source library used for scientific and technical computing, providing tools for optimization, integration, interpolation, eigenvalue problems, and more.

List of References

1. Stanley H. Chan, *Introduction to Probability for Data Science*
2. https://www.simplilearn.com/
3. Casella, George; Roger L. Berger (2001), *Statistical Inference* (2nd ed.)
4. The Book of Statistical Proofs "statproofbook.github.io"
5. https://online.stat.psu.edu/stat414/
6. https://byjus.com/
7. https://towardsdatascience.com/
8. https://www.linkedin.com/
9. https://www.probabilitycourse.com/
10. https://www.investopedia.com/
11. https://github.com/ivanseidel/
12. https://www.qub.ac.uk/schools/SchoolofMathematicsandPhysics/
13. https://www.cuemath.com/en-gb/
14. https://stats.libretexts.org/
15. https://en.wikipedia.org/wiki/Main_Page

Task Assignment

All team participants worked in

- Collecting data about every random variable.
- Implementing each random variable using python coding language.
- Researching real-life applications.
- Implementing each of the four real-life applications.
- Making of the report and presentation.

For Random Variables each one was assigned to the following Random Variable:

Hazem Abuelanin Mohamed	**Binomial** Random Variable
Yehia Mohamed Naeem	**Gaussian** Random Variable
Abdelrahman Afify Hussein	**Bernoulli** Random Variable
Mohamed Hany Mohamed	**Geometric** Random Variable
Amr Rafik Ahmed	**Poisson** Random Variable
Mohanad Mohamed Talaat	**Uniform Continuous** Random Variable
Lujain Ahmed Youssef	**Exponential** Random Variable
Raneem Ahmed Refaat	**Uniform Discrete** Random Variable

For real-life applications each 2 worked on one application:

Hazem Abuelanin Mohamed	**Stock Analysis**
Lujain Ahmed Youssef	
Abdelrahman Afify Hussein	**Phone Tracker**
Mohanad Mohamed Talaat	
Yehia Mohamed Naeem	**Literacy rate analysis**
Mohamed Hany Mohamed	
Amr Rafik Ahmed	**Hospital Admissions**
Raneem Ahmed Refaat	

Given what is mentioned above; each one of us participated on the project with 12.5%