STATISTICS ADVANCE-1

Q1

-->In probability theory, a random variable is a function that assigns numerical values to outcomes of a random experiment. It provides a way to quantify uncertainty by mapping each possible event to a real number.

Types of Random Variables: Discrete Random Variable – Takes on a finite or countably infinite set of values (e.g., the number of heads in 10 coin flips).

Continuous Random Variable – Takes on an uncountable range of values, typically over an interval (e.g., the height of a randomly chosen person)

Q2

-->1. Discrete Random Variables A random variable is discrete if it takes on a finite or countably infinite number of distinct values. These variables usually arise in scenarios where outcomes are counted.

Examples: The number of heads in 10 coin flips (values: 0, 1, 2, ..., 10)

The number of students present in a class on a given day (values: 0, 1, 2, ..., max capacity)

The number of goals scored in a soccer match (values: 0, 1, 2, ...)

Probability Distribution: Represented by a probability mass function (PMF)

Example: If X X represents a fair die roll, the PMF is:

P (X k) 1 6 , k 1 , 2 , 3 , 4 , 5 , 6 P(X=k)= 6 1,k=1,2,3,4,5,6

Continuous Random Variables A random variable is continuous if it takes on an uncountable number of values, typically any value within an interval of real numbers. These arise in measurements where precision can be infinite. Examples: The height of a randomly chosen student (e.g., 150.3 cm, 162.7 cm, etc.)

The time required to complete a task (e.g., 12.45 seconds, 13.98 seconds, etc.)

The temperature in a city at noon (e.g., 22.4°C, 30.1°C, etc.)

Probability Distribution: Represented by a probability density function (PDF)

Probabilities are found over intervals:

P (a ≤ X ≤ b) ∫ a b f (x) d x P(a≤X≤b)=∫ a bf(x)dx where f (x) f(x) is the PDF of X X.

Q3

-->1. Discrete Distributions A discrete probability distribution describes a discrete random variable, which can take on only a countable number of distinct values.

Key Characteristics: The probability of each possible value is given by a probability mass function (PMF):

$P(X\ x)$ P(X=x) The total probability must sum to 1:

$\sum P(X\ x)$ 1 ∑P(X=x)=1 Probabilities are assigned to specific values, meaning $P(X\ x)$ P(X=x) is always nonzero for valid values.

Examples of Discrete Distributions: Binomial Distribution: Models the number of successes in a fixed number of independent trials (e.g., number of heads in 10 coin flips).

Poisson Distribution: Models the number of events occurring in a fixed interval of time or space (e.g., number of emails received per hour).

Geometric Distribution: Models the number of trials until the first success (e.g., number of times you roll a die until you get a 6).

Continuous Distributions A continuous probability distribution describes a continuous random variable, which can take on an infinite number of values within a given range. Key Characteristics: Probabilities are given by a probability density function (PDF):

$f(x)$ f(x) The probability of a specific value is zero:

$P(X\ x)$ 0 P(X=x)=0 Instead, probability is determined over intervals:

$P(a \le X \le b)\ \int a\ b\ f(x)\ d\ x$ P(a≤X≤b)=∫ a bf(x)dx The total probability over all possible values must integrate to 1:

$\int -\infty\ \infty\ f(x)\ d\ x$ 1 ∫ −∞ ∞f(x)dx=1 Examples of Continuous Distributions: Normal (Gaussian) Distribution: Bell-shaped curve often used for natural phenomena (e.g., heights of people, test scores).

Exponential Distribution: Models time until the next event in a Poisson process (e.g., time between customer arrivals).

Uniform Distribution: All values in a given range have equal probability (e.g., randomly choosing a number between 0 and 1).

Q4

-->Probability Mass Function (PMF) A function that gives the probability of a specific value occurring:

$P(X\ x)$ P(X=x) The probabilities must satisfy:

$\sum P(X\ x)$ 1 ∑P(X=x)=1 Example:

Suppose X X is the outcome of rolling a fair six-sided die.

$P(X\ x)\ \{1\ 6\ ,\ x \in \{1\ ,\ 2\ ,\ 3\ ,\ 4\ ,\ 5\ ,\ 6\ \}\ 0\ ,$ otherwise P(X=x)={ 6 1, 0,

x∈{1,2,3,4,5,6} otherwise

Common Discrete Distributions: Binomial Distribution (e.g., number of heads in 10 coin flips)

Poisson Distribution (e.g., number of emails received per hour)

Geometric Distribution (e.g., number of trials before the first success)

Continuous Probability Distribution Function For continuous random variables, the probability distribution is described by a Probability Density Function (PDF). Probability Density Function (PDF) A function $f(x)$ f(x) that represents the likelihood of a random variable taking a particular value within an interval.

Since a continuous variable has an infinite number of possible values, the probability of any specific value is zero:

$P(X\ x)$ 0 P(X=x)=0 Instead, probability is found by integrating over an interval:

$P(a \le X \le b)$ ∫ $a\ b\ f(x)\ d\ x$ P(a≤X≤b)=∫ a bf(x)dx The total probability over all values must be 1:

∫ − ∞ ∞ $f(x)\ d\ x$ 1 ∫ −∞ ∞f(x)dx=1 Example: Normal Distribution (Gaussian) A common continuous distribution with the PDF:

$f(x)$ 1 σ 2 $\pi\ e$ − ($x - \mu$) 2 2 σ 2 f(x)= σ 2π

1e − 2σ 2

(x−μ) 2

where:

μ μ = mean (center)

σ σ = standard deviation (spread)

Common Continuous Distributions: Normal Distribution (e.g., heights of people, IQ scores)

Exponential Distribution (e.g., time between customer arrivals)

Uniform Distribution (e.g., random number between 0 and 1)

Q5

-->1. Probability Distribution Function (PDF/PMF) Describes the instantaneous probability of a random variable taking on a specific value (discrete case) or the likelihood of a value within an interval (continuous case).

Different for discrete and continuous random variables:

For discrete random variables, the function is called the Probability Mass Function (PMF).

For continuous random variables, the function is called the Probability Density Function (PDF).

Example of PMF (Discrete Case) For a fair six-sided die:

$P(X\ x)$ { 1 6 , $x \in \{1, 2, 3, 4, 5, 6\}$ 0 , otherwise P(X=x)={ 6 1, 0,

x∈{1,2,3,4,5,6} otherwise

Here, the PMF gives the probability of getting exactly x x.

Example of PDF (Continuous Case) For a normal distribution:

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

1e − 2σ 2

(x−μ) 2

Since continuous values are infinite, probabilities are determined using integrals over intervals, not at specific points.

Cumulative Distribution Function (CDF) Describes the cumulative probability that a random variable X X takes on a value less than or equal to x x. Defined as:

$F(x)P(X\le x)$ F(x)=P(X≤x) The CDF accumulates probabilities from negative infinity up to x x, giving a running total of probability.

Discrete Case (CDF) For a fair six-sided die:

$F(x)P(X\le x)\sum_{k=-\infty}^{x}P(X=k)$ F(x)=P(X≤x)= k=−∞ ∑ xP(X=k) For example:

$F(3)P(X\le 3)P(X=1)+P(X=2)+P(X=3)\frac{1}{6}+\frac{1}{6}+\frac{1}{6}\frac{3}{6}0.5$ F(3)=P(X≤3)=P(X=1)+P(X=2)+P(X=3)= 6 1+ 6 1+ 6 1 6 3=0.5 Continuous Case (CDF) For a continuous distribution:

$F(x)P(X\le x)\int_{-\infty}^{x}f(t)\,dt$ F(x)=P(X≤x)=∫ −∞ xf(t)dt For example, in a normal distribution, $F(x)$ F(x) gives the probability that a randomly chosen value is less than or equal to x x.

Q6

-->A Discrete Uniform Distribution is a probability distribution where a finite number of outcomes are equally likely. It is one of the simplest probability distributions and is often used when there is no reason to favor one outcome over another.

A discrete random variable X X follows a Discrete Uniform Distribution if it takes on n n distinct values with equal probability. The probability mass function (PMF) is:

$P(X=x)\frac{1}{n}$, for $x\in\{x_1,x_2,\ldots,x_n\}$ P(X=x)= n 1,for x∈{x 1,x 2,...,x n} where:

n n is the total number of possible outcomes

Each outcome has the same probability 1 n n 1

Q7

-->. Key Properties Mean (Expected Value) $E(X)p$ E(X)=p This represents the long-run average value of X X.

Variance Var $(X)p(1-p)$ Var(X)=p(1−p) This measures how much the values of X X vary from the mean.

Probability Mass Function (PMF) $P (X x) p x (1 − p) 1 − x , x ∈ { 0 , 1 }$ P(X=x)=p x (1−p) 1−x ,x∈{0,1} If X 1 X=1, then $P (X 1) p$ P(X=1)=p.

If X 0 X=0, then $P (X 0) 1 − p$ P(X=0)=1−p.

Cumulative Distribution Function (CDF) $F (x) { 0 , x < 0 1 − p , 0 ≤ x < 1 1 , x ≥ 1$ F(x)= ι } r

0, 1−p, 1,

x<0 0≤x<1 x≥1

The CDF shows the probability that X X is less than or equal to a given value.

Moment-Generating Function (MGF) $M X (t) (1 − p) + p e t$ M X(t)=(1−p)+pe t

Used for deriving moments like mean and variance.

Skewness $1 − 2 p p (1 − p)$ p(1−p)

1−2p

Indicates how asymmetric the distribution is.

Kurtosis $1 − 6 p (1 − p) p (1 − p)$ p(1−p) 1−6p(1−p)

Measures how heavy-tailed or peaked the distribution is.

Q8

-->The binomial distribution models the number of successes in a fixed number of independent trials, where each trial has only two possible outcomes: success or failure. It is widely used in probability and statistics for modeling real-world situations where outcomes are binary.

Example Uses

Coin Tossing If you flip a fair coin 10 times (p 10 n=10) and count the number of heads (0.5 p=0.5), the number of heads follows: $X ∼$ Binomial (10 , 0.5) X∼Binomial(10,0.5) For example:

$P (X 5)$ P(X=5) = Probability of getting exactly 5 heads.

$P (X ≤ 5)$ P(X≤5) = Probability of getting at most 5 heads.

Manufacturing Defects A factory produces light bulbs, and each bulb has a 2% chance of being defective (p 0.02 p=0.02). If you randomly inspect 50 bulbs (n 50 n=50), then: $X ∼$ Binomial (50 , 0.02) X∼Binomial(50,0.02) $P (X 0)$ P(X=0) = Probability that none of the 50 bulbs are defective.

$P (X ≥ 2)$ P(X≥2) = Probability that at least 2 bulbs are defective.

Medical Trials If a vaccine has a 90% success rate (n 0.9 p=0.9), and it is given to 20 patients (20 n=20), then: $X ∼$ Binomial (20 , 0.9) X∼Binomial(20,0.9) $P (X 18)$ P(X=18) = Probability that exactly 18 patients develop immunity.

$P (X ≥ 15)$ P(X≥15) = Probability that at least 15 patients develop immunity.

Q9

-->The Poisson distribution models the number of times an event occurs in a fixed interval (time, space, or any unit), assuming the events happen randomly and independently at a constant average rate.

It is often used when dealing with rare or random events occurring over time or space.

Call Center Analysis If a call center receives an average of 5 calls per minute, the number of calls received in one minute follows: X ~ Poisson (5) X~Poisson(5) P (X 3) P(X=3) = Probability of receiving exactly 3 calls in a minute.

P (X ≥ 6) P(X≥6) = Probability of receiving at least 6 calls.

Website Traffic If a website gets 10 visitors per second on average, the number of visitors arriving in a given second follows: X ~ Poisson (10) X~Poisson(10) 3. Disease Occurrence in a Population If a rare disease affects 2 out of every 1,000 people per year, then the number of cases in a town of 10,000 people follows:

X ~ Poisson (20) X~Poisson(20) 4. Accidents & Insurance Claims If a highway experiences 3 accidents per day on average, the number of accidents on a given day follows:

X ~ Poisson (3) X~Poisson(3) 5. Defects in Manufacturing If a factory finds 1 defect per 100 items on average, then the number of defects in a batch of 500 items follows:

X ~ Poisson (5) X~Poisson(5)

Q10

-->The continuous uniform distribution models a random variable that has an equal probability of occurring anywhere within a given interval [a , b] [a,b].** It is one of the simplest continuous probability distributions.

Definition A random variable X X follows a continuous uniform distribution if it is equally likely to take any value in a given range [a , b] [a,b]. $X \sim U$ (a , b) X~U(a,b) where:

a a = lower bound (minimum value)

b b = upper bound (maximum value)

X X takes any real value in the interval [a , b] [a,b]

Q11

-->Key Characteristics of the Normal Distribution

Symmetry & Bell Shape The curve is symmetric around the mean μ. The highest point (peak) occurs at μ μ.

The distribution extends infinitely in both directions but never touches the x-axis.

Mean, Median, and Mode Are Equal In a normal distribution: Mean Median Mode Mean=Median=Mode This means the center of the distribution is also its most frequent value.

Empirical Rule (68-95-99.7 Rule) The percentage of data within standard deviations of the mean is approximately: 68% of data falls within 1 standard deviation ($\mu \pm \sigma$ μ±σ).

95% of data falls within 2 standard deviations ($\mu \pm 2\sigma$ μ±2σ).

99.7% of data falls within 3 standard deviations ($\mu \pm 3\sigma$ μ±3σ).

This is useful in quality control, grading systems, and statistical analysis.

Skewness & Kurtosis Skewness = 0 (perfectly symmetric). Kurtosis = 3 (mesokurtic, meaning it has moderate tails compared to other distributions).

Standard Normal Distribution $N(0,1)$ N(0,1) A special case of the normal distribution is the standard normal distribution: $\sim \sim$ □ N(0,1) Z□ N(0,1) where:

Mean μ 0 μ=0.

Standard deviation σ 1 σ=1.

Any normal distribution can be converted to the standard normal distribution using z-scores:

$Z = \frac{X - \mu}{\sigma}$ Z= σ X−μ

The z-score tells us how many standard deviations a value is from the mean.

Q12

-->The standard normal distribution is a special case of the normal distribution where the mean is 0 and the standard deviation is 1. It is a key concept in probability and statistics because it allows us to compare different normal distributions using z-scores.

Why is the Standard Normal Distribution Important?

Z-Scores Allow Comparison Between Distributions If X X is normally distributed with any mean μ μ and standard deviation σ σ, we can convert it to the standard normal distribution using the z-score formula: $Z = \frac{X - \mu}{\sigma}$ Z= σ X−μ

where:

X X is the original value.

μ μ is the mean.

σ σ is the standard deviation.

Z Z is the transformed value in the standard normal distribution.

This transformation allows us to use the same probability table for any normal distribution.

Q13

-->The Central Limit Theorem (CLT) is one of the most fundamental concepts in statistics and probability theory. It states that:

Regardless of the original population distribution, the distribution of the sample mean (or sum) will approximate a normal distribution as the sample size increases, provided the samples are independent and identically distributed (i.i.d.).

Conditions for the Central Limit Theorem For CLT to hold, the following conditions should be met:

Random Sampling: The samples should be selected randomly and independently.

Sample Size n n Should Be Large:

A general rule of thumb: n ≥ 30 n≥30 is usually enough.

If the original distribution is highly skewed, a larger n n may be needed.

Finite Variance: The population must have a finite variance (σ 2 < ∞ σ 2 <∞).

Q14

-->If a population has mean μ μ and variance σ 2 σ 2 , then the distribution of the sample mean X ¯ X ¯ follows:

X ¯ ∼ N (μ , σ 2 n) X ¯ ∼N(μ, n σ 2

) where:

The mean of the sample means remains μ μ (same as the population mean).

The variance of the sample means is reduced by a factor of n n, making the distribution tighter around the mean.

This means the sample means form a normal distribution, even when the original data does not!

Q15

-->Z-statistics (or Z-tests) are used in hypothesis testing when we want to compare a sample mean to a population mean, assuming the population variance is known or the sample size is large (n ≥ 30 n≥30). The Z-test is based on the standard normal distribution.

Q16

-->Scenario: A university's exam scores have:

Mean = 75

Standard deviation = 10

A student scores 90

What is the Z-score of 90?

Z 90 − 75 10 15 10 1.5 Z= 10 90−75 10 15=1.5 Interpretation: A score of 90 is 1.5 standard deviations above the mean.

Using a Z-table, we find that P (Z < 1.5) ≈ 0.9332 P(Z<1.5)≈0.9332, meaning 93.32% of students scored lower than 90.

Q17

-->Point Estimates A point estimate is a single value used to estimate a population parameter. It is calculated from a sample and serves as our best guess for the true population value.

Examples of Point Estimates: Sample Mean ($\bar{X}$ X ¯) estimates the population mean (μ μ).

Sample Proportion ($\hat{p}$ ^ p ^) estimates the population proportion (p p).

Sample Standard Deviation (s s) estimates the population standard deviation (σ σ).

Advantage: Simple and easy to compute. Disadvantage: Does not account for sampling variability (margin of error).

Interval Estimates (Confidence Intervals) An interval estimate provides a range of values within which the population parameter is likely to lie, along with a confidence level (e.g., 95%).

Confidence Interval Formula (for a Mean): Confidence Interval $\bar{X} \pm Z \times \frac{\sigma}{n}$ Confidence Interval= X ¯ ±Z× n

σ

where:

$\bar{X}$ X ¯ = Sample mean

Z Z = Z-score corresponding to the confidence level (e.g., 1.96 for 95% confidence)

σ σ = Population standard deviation

n n = Sample size

Example: A researcher wants to estimate the average height of students in a university. A sample of 50 students has a mean height of 170 cm with a standard deviation of 10 cm.

Using a 95% confidence interval (Z 1.96 Z=1.96):

CI 170 ± 1.96 × 10 50 CI=170±1.96× 50

10

= 170 ± 2.77 =170±2.77 (167.23 , 172.77) =(167.23,172.77) Interpretation: We are 95% confident that the true mean height of all students is between 167.23 cm and 172.77 cm.

Q18

-->A confidence interval (CI) is a range of values that estimates an unknown population parameter with a certain level of confidence. It is widely used in statistical analysis for decision-making, hypothesis testing, and estimating population characteristics.

Why Are Confidence Intervals Important? Accounts for Sampling Variability When we collect a sample, the sample statistics (mean, proportion, etc.) may not be exactly equal to the true population value. Confidence intervals account for this uncertainty by providing a range rather than a single estimate.

Provides a Measure of Precision A narrow confidence interval → More precise estimate. A wider confidence interval → Less precise but more conservative.

Increasing sample size narrows the interval, improving precision.

Helps in Hypothesis Testing If a confidence interval includes the hypothesized population value, we fail to reject the null hypothesis. If a confidence interval excludes the hypothesized population value, we reject the null hypothesis.

More Informative than Point Estimates A point estimate (e.g., sample mean) is useful, but it does not tell us how uncertain the estimate is. A confidence interval gives a range, helping in better decision-making.

Used in Medical, Business, and Scientific Research Clinical trials: Estimating the effectiveness of a drug. Market research: Estimating customer satisfaction levels.

Engineering: Assessing reliability of manufactured parts.

Q19

-->Relationship Between a Z-Score and a Confidence Interval A Z-score and a confidence interval (CI) are closely related because Z-scores determine the margin of error in confidence intervals when the population standard deviation is known.

How Are Z-Scores Used in Confidence Intervals? A confidence interval is calculated using the formula: Confidence Interval $\bar{X}$ ± Z × σ n Confidence Interval= X ¯ ±Z× n

σ

where:

$\bar{X}$ X ¯ = Sample mean

Z Z = Z-score corresponding to the confidence level

σ σ = Population standard deviation

n n = Sample size

The Z-score (also called the critical value) depends on the confidence level (1 − α 1−α).

Common Z-Scores for Confidence Levels Confidence Level Z-Score (Critical Value) 90% 1.645 95% 1.96 99% 2.576 A higher confidence level means a larger Z-score, resulting in a wider confidence interval. A lower confidence level means a smaller Z-score, leading to a narrower confidence interval.

Example Calculation Suppose a researcher wants to estimate the average IQ of students. Sample Mean $\bar{X}$ 110 X ¯ =110

Standard Deviation σ 15 σ=15

Sample Size n 100 n=100

95% Confidence Interval (Z 1.96 Z=1.96):

$C\ I$ 110 ± 1.96 × 15 100 CI=110±1.96× 100

15

= 110 ± 1.96 × 1.5 =110±1.96×1.5 110 ± 2.94 =110±2.94 (107.06 , 112.94) =(107.06,112.94) Interpretation: We are 95% confident that the true average IQ of students is between 107.06 and 112.94.

Q20

-->A Z-score (or standard score) standardizes values from different distributions by converting them into a common scale. This allows for meaningful comparisons between datasets with different means and standard deviations.

Why Use Z-Scores for Comparison? Different datasets often have different means and standard deviations. $\bar{x}$-scores transform values into a standard normal distribution (σ 0 μ=0, 1 σ=1). This allows us to compare relative positions of values across different distributions. Q21

-->The Central Limit Theorem (CLT) states that the sampling distribution of the sample mean approaches a normal distribution as the sample size increases, regardless of the shape of the original population distribution. However, certain assumptions must be met for the CLT to apply effectively.

Assumptions of the Central Limit Theorem Random Sampling The sample must be selected randomly to ensure it represents the population without bias. Non-random samples can lead to misleading conclusions.

Independence of Observations The data points in the sample should be independent of each other. This means that selecting one data point should not influence the selection of another.

If sampling without replacement, the sample size should be less than 10% of the population to maintain independence.

Sample Size Should Be Large Enough The larger the sample size, the more the sampling distribution approximates a normal distribution. Rule of Thumb:

If the population is normally distributed, any sample size n n works.

If the population is skewed or not normal, a sample size of n ≥ 30 n≥30 is usually sufficient.

If the population is highly skewed, an even larger sample size may be needed.

Finite Variance (No Extreme Outliers) The population should have a finite mean and variance. Extreme outliers or infinite variance can distort the sampling distribution.

Q22

-->The expected value (EV) of a probability distribution represents the long-term average or mean outcome of a random variable if an experiment is repeated many times. It provides a measure of the central tendency of a probability distribution.

Example 1: Expected Value for a Discrete Random Variable A fair 6-sided die has possible outcomes: 1, 2, 3, 4, 5, 6, each with probability 1 6 6 1.

E (X) (1 × 1 6) + (2 × 1 6) + (3 × 1 6) + (4 × 1 6) + (5 × 1 6) + (6 × 1 6) E(X)=(1× 6 1)+(2× 6 1) +(3× 6 1)+(4× 6 1)+(5× 6 1)+(6× 6 1) 1 + 2 + 3 + 4 + 5 + 6 6 21 6 3.5 6 1+2+3+4+5+6 6 21=3.5 Interpretation: Over many rolls, the average result will be 3.5, even though 3.5 is not an actual die outcome.

Example 2: Expected Value in a Gambling Scenario A lottery ticket costs $10. You can:

Win $100 (probability = 0.1)

Win $50 (probability = 0.2)

Win $0 (probability = 0.7)

$E(X)$ (100 × 0.1) + (50 × 0.2) + (0 × 0.7) E(X)=(100×0.1)+(50×0.2)+(0×0.7) 10 + 10 + 0 20 =10+10+0=20 Since the ticket cost is $10, the net expected value is:

$E(X)$ − 10 20 − 10 10 E(X)−10=20−10=10 Interpretation: On average, a person gains $10 per ticket over many plays.

Q23

-->□ For Discrete Random Variables A probability mass function (PMF) assigns probabilities to each possible value of X X. The expected value is calculated as:

$E(X)$ ∑ x_i $P(X = x_i)$ E(X)=∑x iP(X=x i) ✔ Each value x_i x icontributes to the expected outcome based on its probability $P(X = x_i)$ P(X=x i).

□ For Continuous Random Variables A probability density function (PDF) gives the probability of different values occurring within a range. The expected value is calculated as:

$E(X)$ ∫ − ∞ ∞ x $f(x)$ dx E(X)=∫ −∞ ∞xf(x)dx ✔ The integral weights each value by its probability density $f(x)$ f(x), determining the long-term average.

PRACTICAL SOLUTIONS

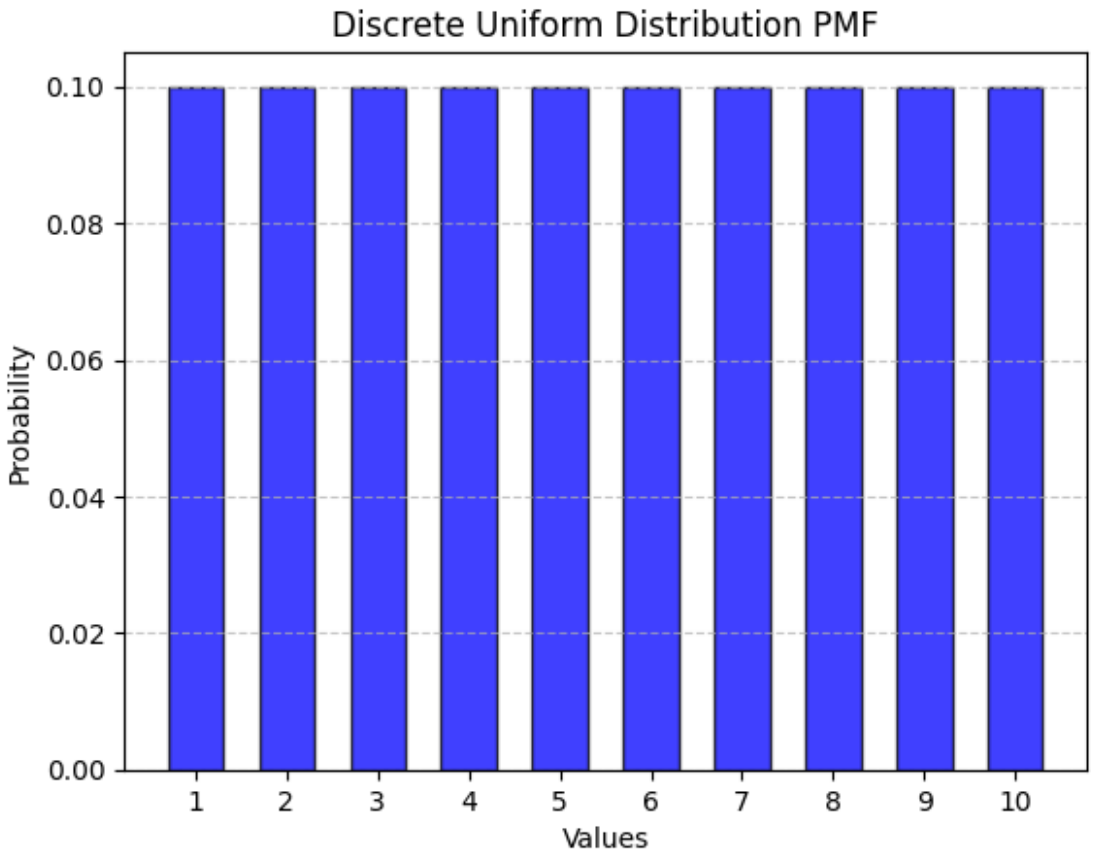
```python
#Q1
import random

# Generate a random integer between 1 and 100
random_value = random.randint(1, 100)

# Display the generated random value
print("Generated Random Value:", random_value)
```

```
Generated Random Value: 45
```

```python
#Q2
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import randint

# Define parameters for discrete uniform distribution (e.g., 1 to 10)
low, high = 1, 10

# Generate a random integer between low and high
random_value = randint.rvs(low, high+1)

# Display the generated random value
```

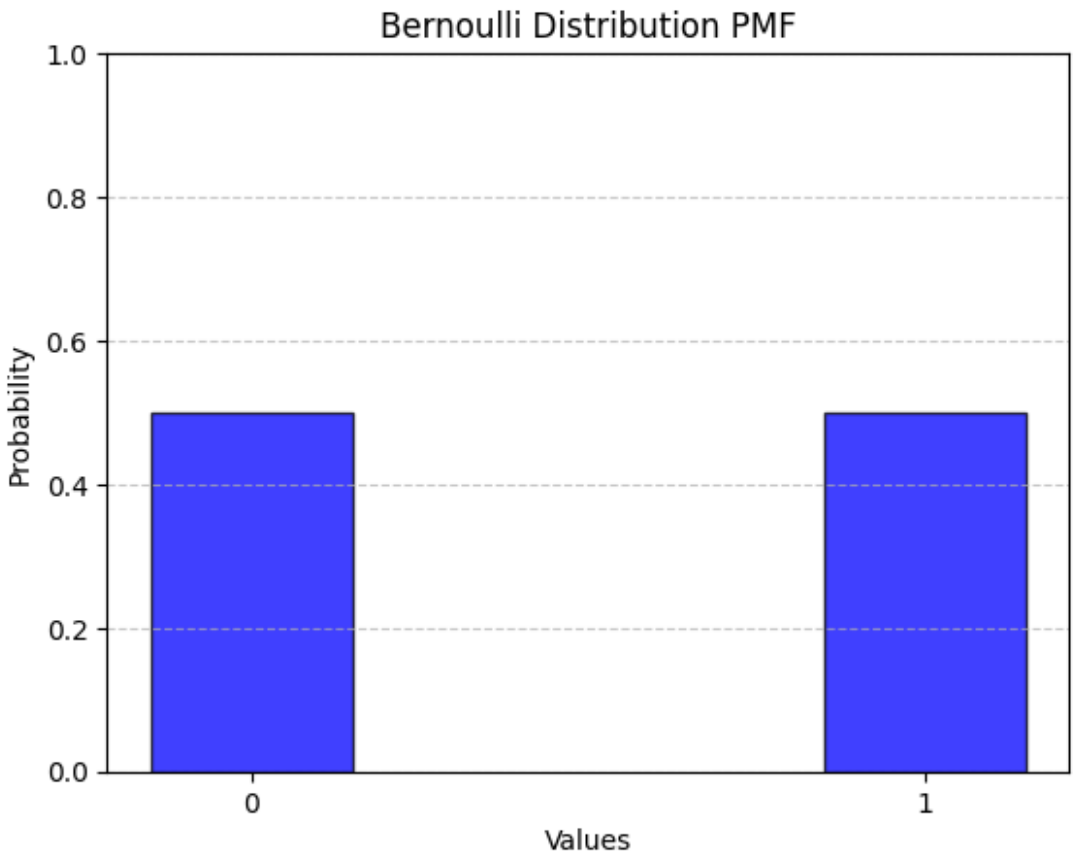
```
print("Generated Random Value:", random_value)

# Generate the probability mass function (PMF)
x = np.arange(low, high+1)
p = randint.pmf(x, low, high+1)

# Plot the PMF
plt.bar(x, p, width=0.6, alpha=0.75, color='blue', edgecolor='black')
plt.xlabel('Values')
plt.ylabel('Probability')
plt.title('Discrete Uniform Distribution PMF')
plt.xticks(x)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
Generated Random Value: 8
```



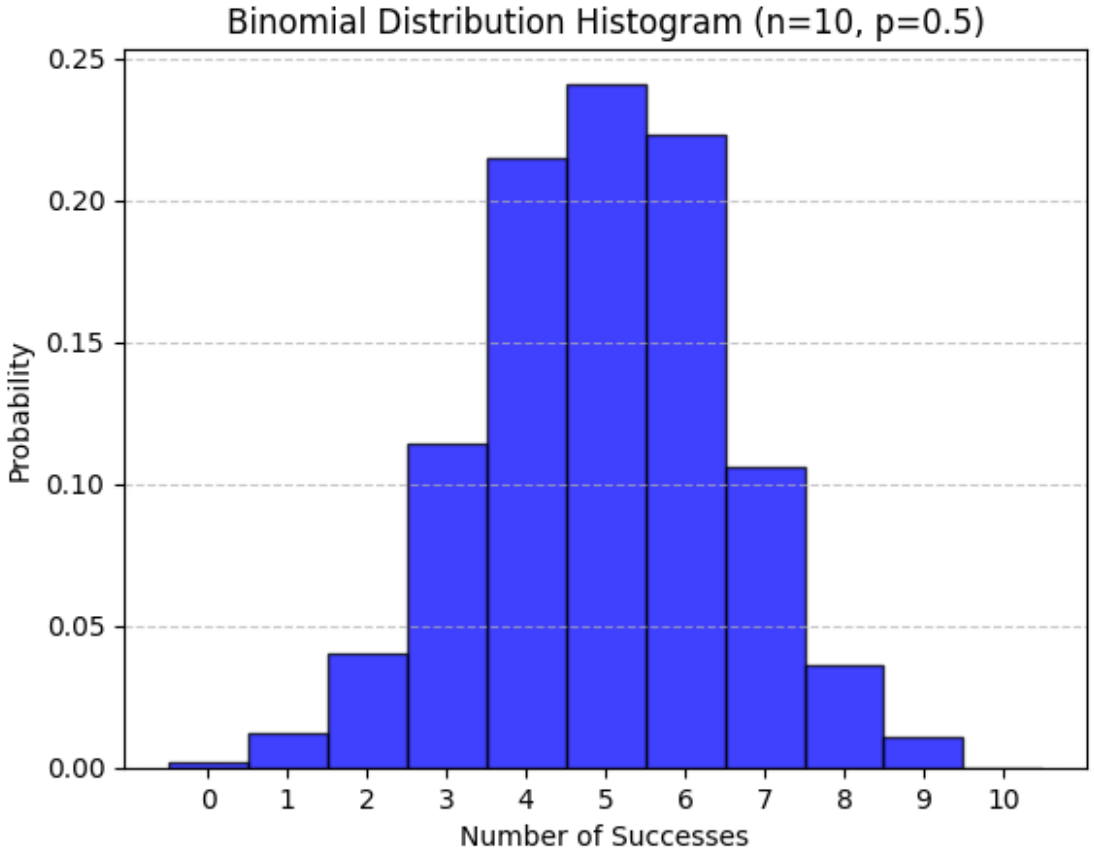

```
#Q3
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import bernoulli

def bernoulli_pmf(p, x):
```

```
    """
    Calculate the probability mass function (PMF) of a Bernoulli
distribution.
    :param p: Probability of success (0 <= p <= 1)
    :param x: Value (0 or 1)
    :return: Probability mass function value for given x
    """
    if x not in [0, 1]:
        raise ValueError("x must be 0 or 1 for a Bernoulli
distribution.")
    return p if x == 1 else 1 - p

# Example usage
p = 0.5  # Probability of success
x_values = [0, 1]
pdf_values = [bernoulli_pmf(p, x) for x in x_values]

# Plot the PMF of Bernoulli Distribution
plt.bar(x_values, pdf_values, width=0.3, alpha=0.75, color='blue',
edgecolor='black')
plt.xlabel('Values')
plt.ylabel('Probability')
plt.title('Bernoulli Distribution PMF')
plt.xticks(x_values)
plt.ylim(0, 1)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



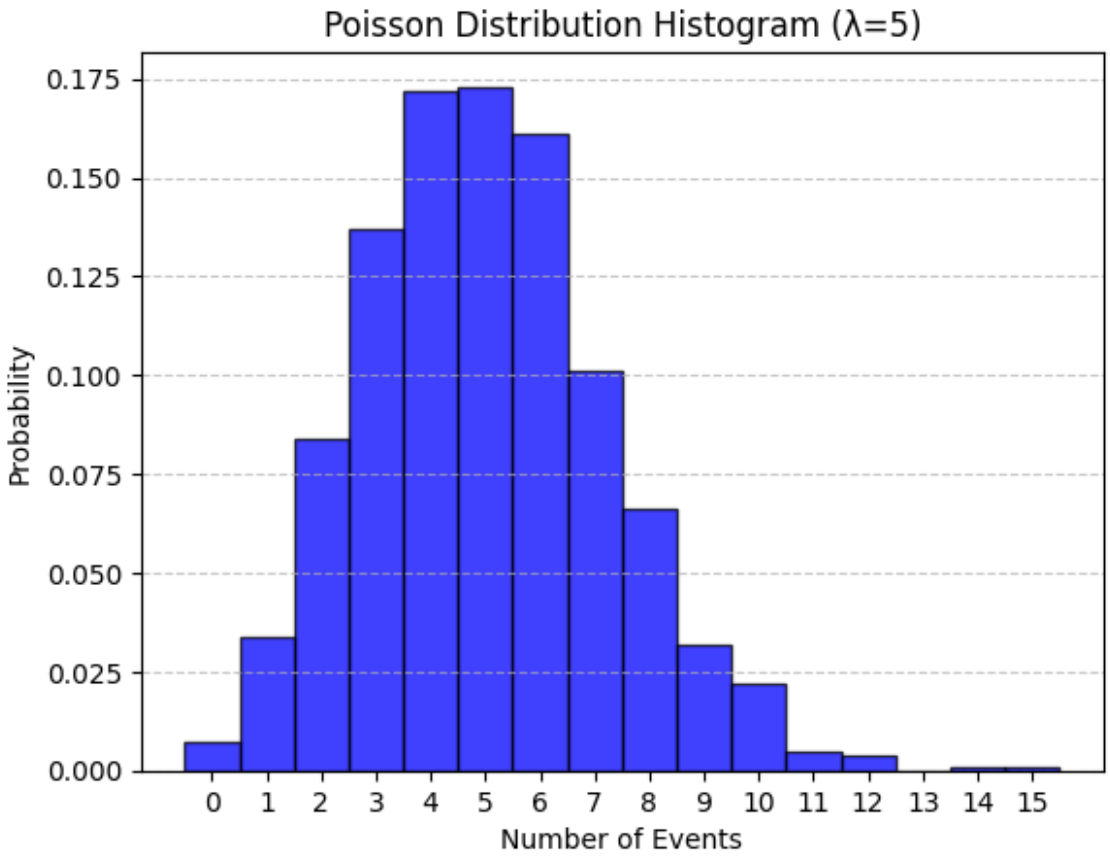

```python
#Q4
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import binom

def simulate_binomial(n, p, size=1000):
    """
    Simulate a binomial distribution and return random samples.
    :param n: Number of trials
    :param p: Probability of success in each trial
    :param size: Number of simulations
    :return: Array of binomial samples
    """
    return binom.rvs(n, p, size=size)

# Parameters
n = 10  # Number of trials
p = 0.5  # Probability of success
size = 1000  # Number of simulations

# Simulate binomial distribution
samples = simulate_binomial(n, p, size)
```

```
# Plot histogram
plt.hist(samples, bins=np.arange(n+2)-0.5, density=True, alpha=0.75,
color='blue', edgecolor='black')
plt.xlabel('Number of Successes')
plt.ylabel('Probability')
plt.title('Binomial Distribution Histogram (n=10, p=0.5)')
plt.xticks(range(n+1))
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



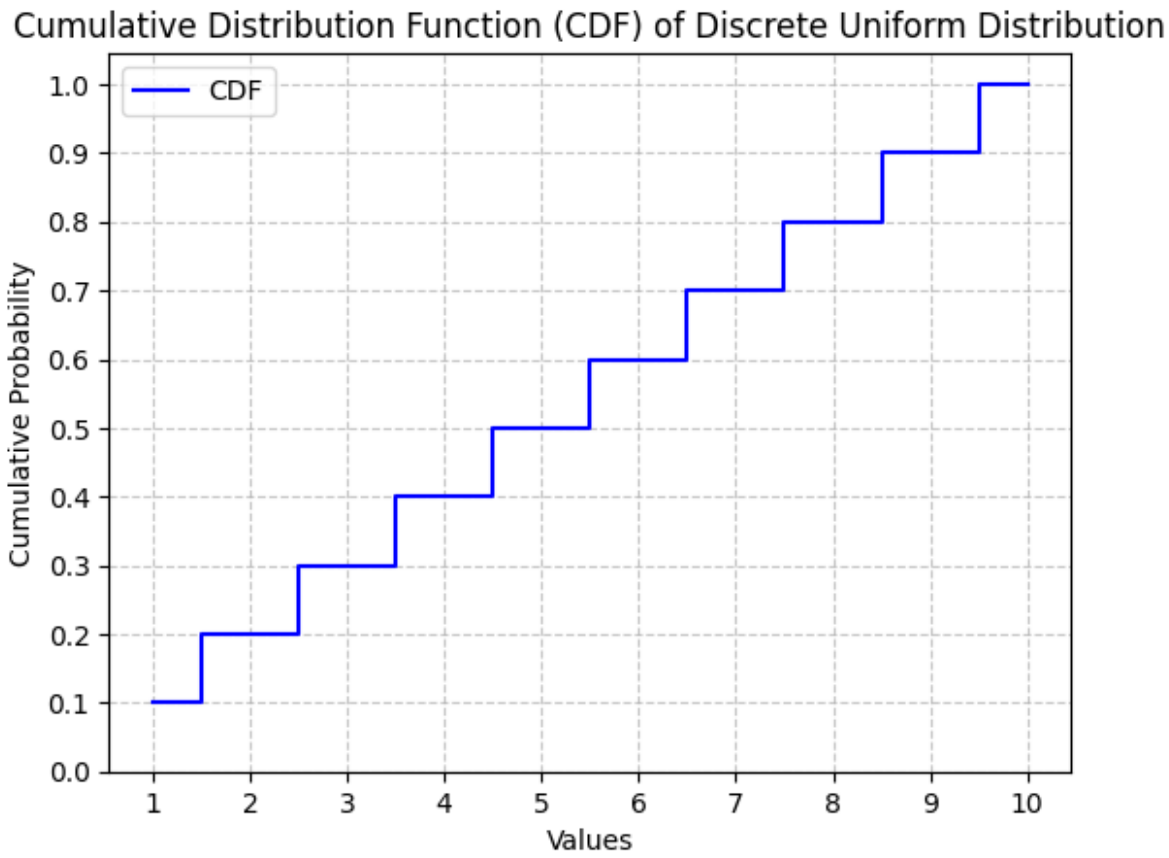

```
#Q5
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import poisson

def simulate_poisson(lam, size=1000):
    """
    Simulate a Poisson distribution and return random samples.
    :param lam: Expected number of events (λ)
    :param size: Number of simulations
    :return: Array of Poisson samples
    """
```

```
    return poisson.rvs(lam, size=size)

# Parameters
lam = 5  # Expected number of events (λ)
size = 1000  # Number of simulations

# Simulate Poisson distribution
samples = simulate_poisson(lam, size)

# Plot histogram
plt.hist(samples, bins=np.arange(min(samples), max(samples) + 1.5) -
0.5, density=True, alpha=0.75, color='blue', edgecolor='black')
plt.xlabel('Number of Events')
plt.ylabel('Probability')
plt.title('Poisson Distribution Histogram (λ=5)')
plt.xticks(range(min(samples), max(samples) + 1))
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



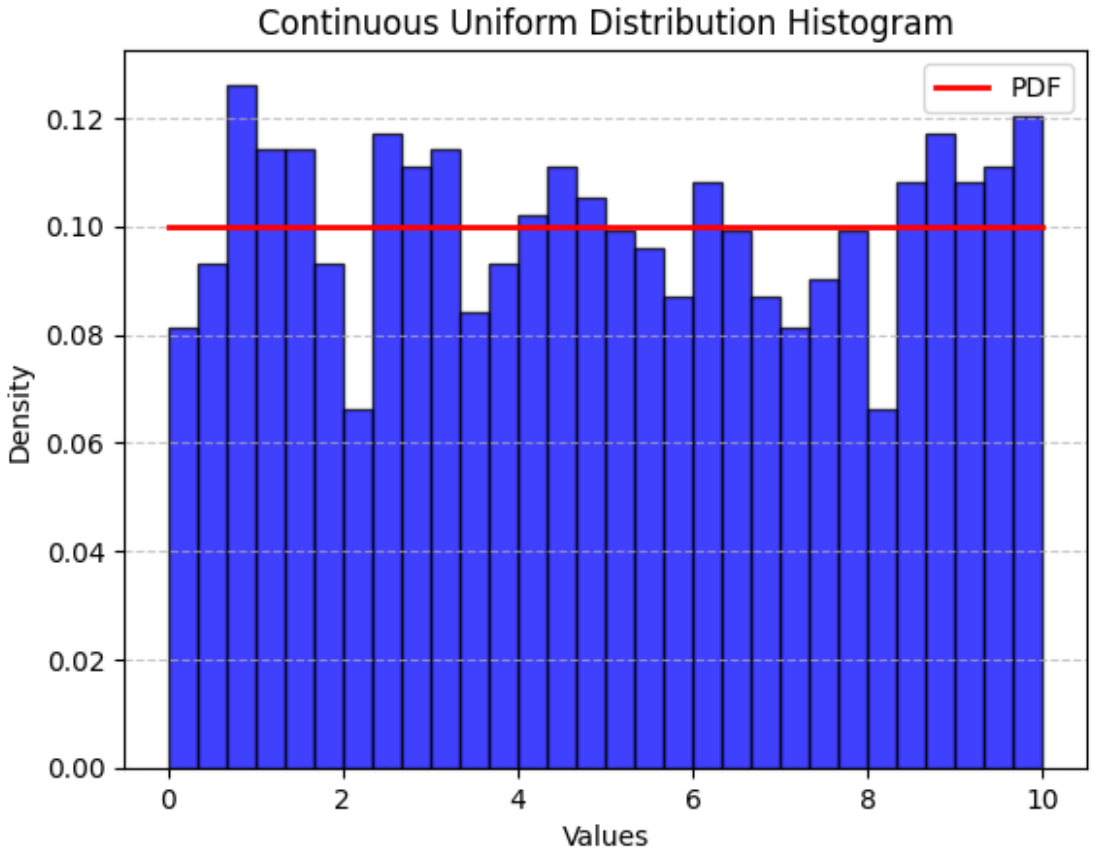

```
#Q6
import numpy as np
import matplotlib.pyplot as plt
```

```
from scipy.stats import randint

def discrete_uniform_cdf(low, high):
    """
    Calculate the cumulative distribution function (CDF) of a discrete
uniform distribution.
    :param low: Lower bound (inclusive)
    :param high: Upper bound (inclusive)
    :return: Arrays of x values and corresponding CDF values
    """
    x = np.arange(low, high + 1)
    cdf_values = randint.cdf(x, low, high + 1)
    return x, cdf_values

# Parameters
low, high = 1, 10  # Range of discrete uniform distribution

# Compute CDF
x, cdf_values = discrete_uniform_cdf(low, high)

# Plot CDF
plt.step(x, cdf_values, where='mid', color='blue', label='CDF')
plt.xlabel('Values')
plt.ylabel('Cumulative Probability')
plt.title('Cumulative Distribution Function (CDF) of Discrete Uniform
Distribution')
plt.xticks(x)
plt.yticks(np.linspace(0, 1, high - low + 2))
plt.grid(axis='both', linestyle='--', alpha=0.7)
plt.legend()
plt.show()
```



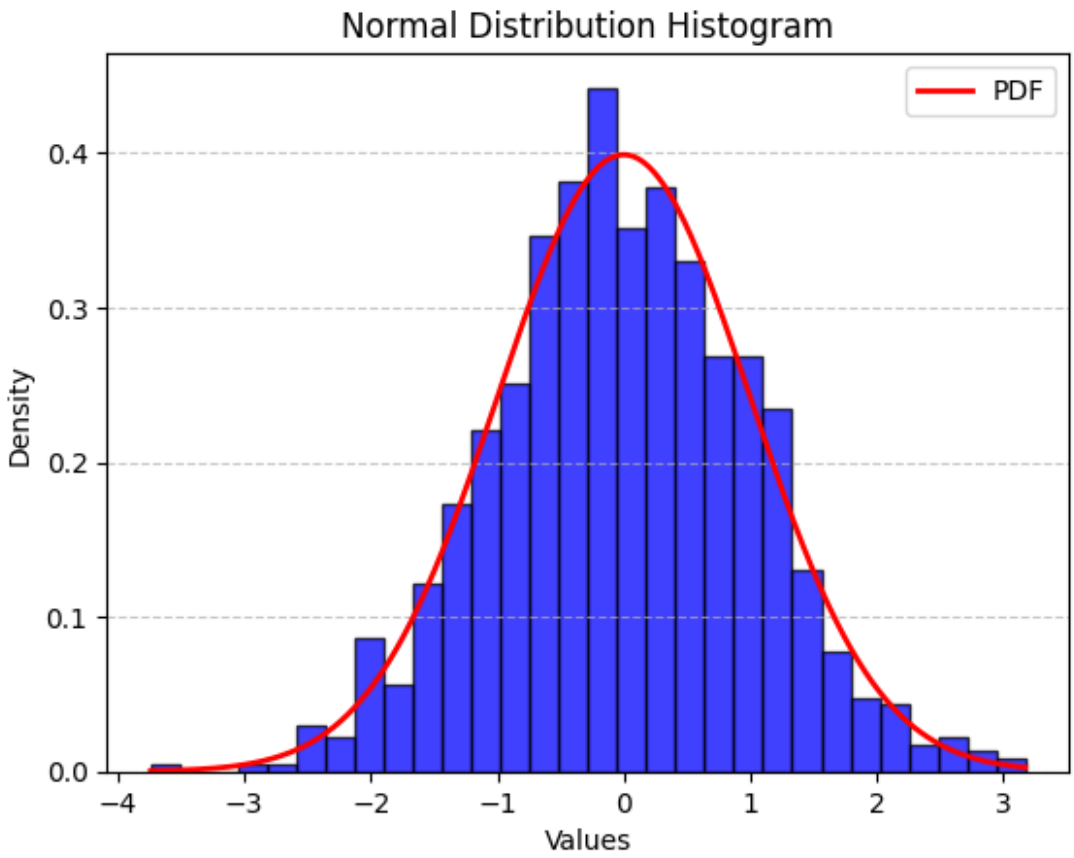

```python
#Q7
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import uniform

def continuous_uniform_distribution(low, high, size=1000):
    """
    Generate a continuous uniform distribution.
    :param low: Lower bound
    :param high: Upper bound
    :param size: Number of samples
    :return: Array of uniform samples
    """
    return np.random.uniform(low, high, size)

# Parameters
low, high = 0, 10  # Range of uniform distribution
size = 1000  # Number of samples

# Generate samples
samples = continuous_uniform_distribution(low, high, size)

# Plot histogram
```

```
plt.hist(samples, bins=30, density=True, alpha=0.75, color='blue',
edgecolor='black')
plt.xlabel('Values')
plt.ylabel('Density')
plt.title('Continuous Uniform Distribution Histogram')
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Plot theoretical PDF
x = np.linspace(low, high, 1000)
pdf = uniform.pdf(x, low, high - low)
plt.plot(x, pdf, 'r-', lw=2, label='PDF')
plt.legend()
plt.show()
```



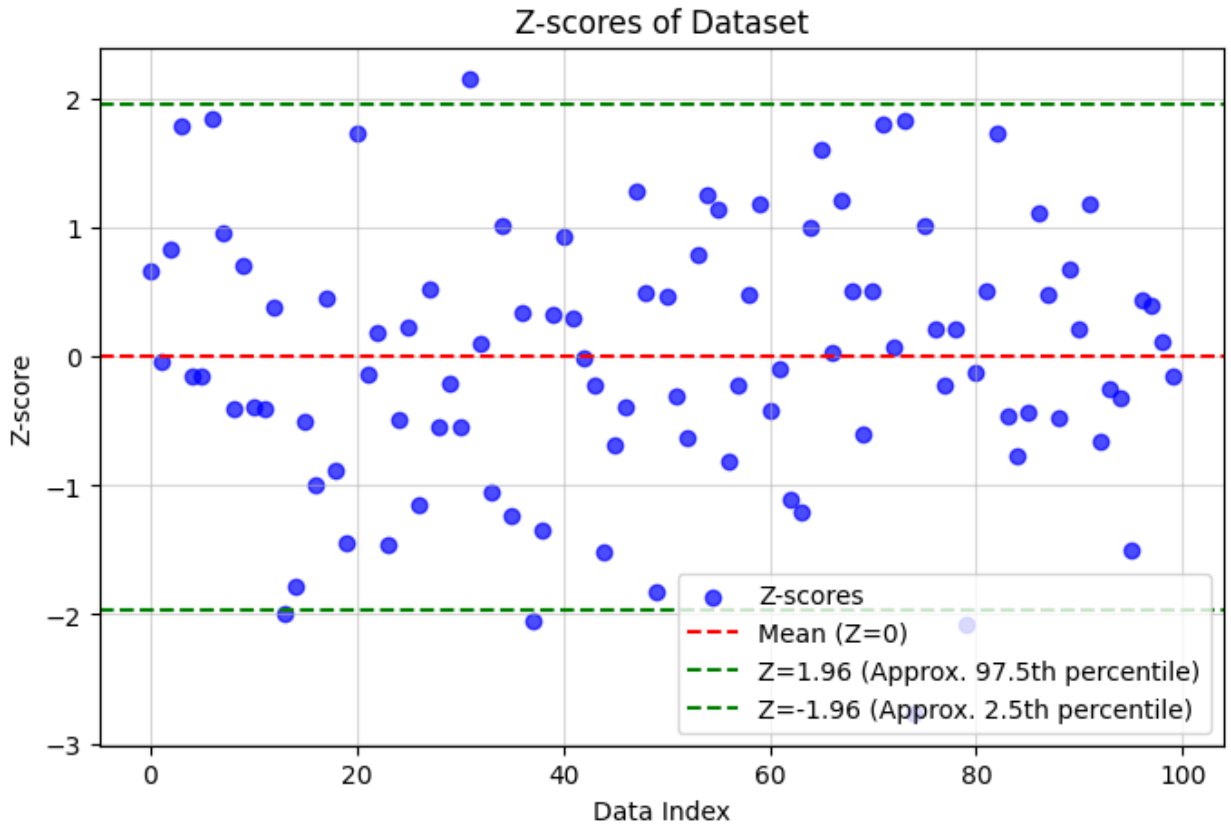

```
#Q8
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

def simulate_normal_distribution(mean, std_dev, size=1000):
    """
    Generate random samples from a normal distribution.
```

```
    :param mean: Mean of the normal distribution
    :param std_dev: Standard deviation of the normal distribution
    :param size: Number of samples
    :return: Array of normal distribution samples
    """
    return np.random.normal(mean, std_dev, size)

# Parameters
mean, std_dev = 0, 1  # Mean and standard deviation of normal
distribution
size = 1000  # Number of samples

# Generate samples
samples = simulate_normal_distribution(mean, std_dev, size)

# Plot histogram
plt.hist(samples, bins=30, density=True, alpha=0.75, color='blue',
edgecolor='black')
plt.xlabel('Values')
plt.ylabel('Density')
plt.title('Normal Distribution Histogram')
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Plot theoretical PDF
x = np.linspace(min(samples), max(samples), 1000)
pdf = norm.pdf(x, mean, std_dev)
plt.plot(x, pdf, 'r-', lw=2, label='PDF')
plt.legend()
plt.show()
```



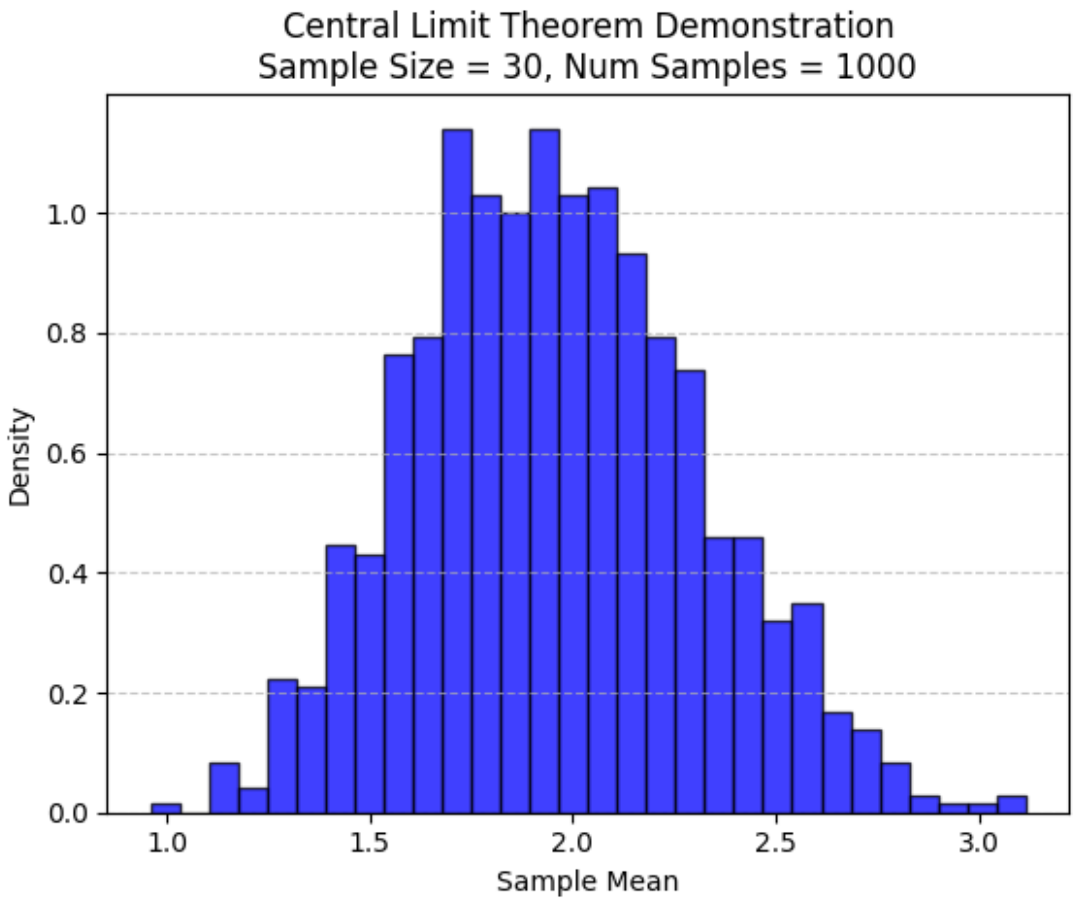

```python
#Q9
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

def calculate_z_scores(data):
    """
    Calculate Z-scores from a dataset.
    :param data: List or array of numerical values
    :return: Array of Z-scores
    """
    mean = np.mean(data)
    std_dev = np.std(data, ddof=1)  # Use sample standard deviation
    return (data - mean) / std_dev

# Generate a random normal dataset
np.random.seed(42)
data = np.random.normal(loc=50, scale=10, size=100)

# Calculate Z-scores
z_scores = calculate_z_scores(data)

# Plot Z-scores
```

```
plt.figure(figsize=(8, 5))
plt.scatter(range(len(z_scores)), z_scores, color='blue', alpha=0.7,
label='Z-scores')
plt.axhline(y=0, color='r', linestyle='--', label='Mean (Z=0)')
plt.axhline(y=1.96, color='g', linestyle='--', label='Z=1.96 (Approx.
97.5th percentile)')
plt.axhline(y=-1.96, color='g', linestyle='--', label='Z=-1.96
(Approx. 2.5th percentile)')
plt.xlabel('Data Index')
plt.ylabel('Z-score')
plt.title('Z-scores of Dataset')
plt.legend()
plt.grid(alpha=0.5)
plt.show()
```



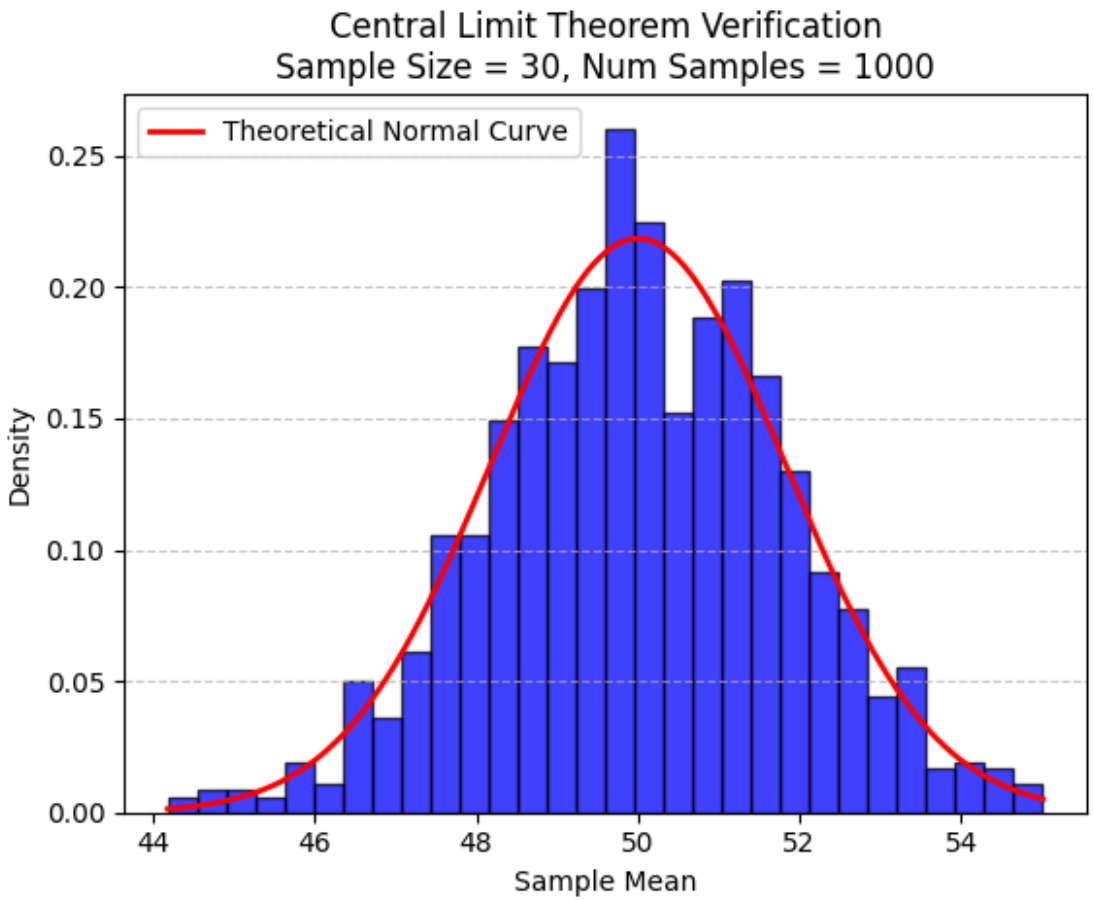

```
#Q10
import numpy as np
import matplotlib.pyplot as plt

def central_limit_theorem_demo(sample_size, num_samples):
    """
    Demonstrate the Central Limit Theorem (CLT) using a non-normal
distribution.
```

```
    :param sample_size: Number of elements in each sample
    :param num_samples: Number of samples to draw
    """
    # Generate a non-normal distribution (exponential)
    population = np.random.exponential(scale=2, size=10000)

    # Draw multiple samples and compute their means
    sample_means = [np.mean(np.random.choice(population, sample_size,
replace=True)) for _ in range(num_samples)]

    # Plot histogram of sample means
    plt.hist(sample_means, bins=30, density=True, alpha=0.75,
color='blue', edgecolor='black')
    plt.xlabel('Sample Mean')
    plt.ylabel('Density')
    plt.title(f'Central Limit Theorem Demonstration\nSample Size =
{sample_size}, Num Samples = {num_samples}')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()

# Parameters
sample_size = 30  # Size of each sample
num_samples = 1000  # Number of samples

# Run CLT demonstration
central_limit_theorem_demo(sample_size, num_samples)
```



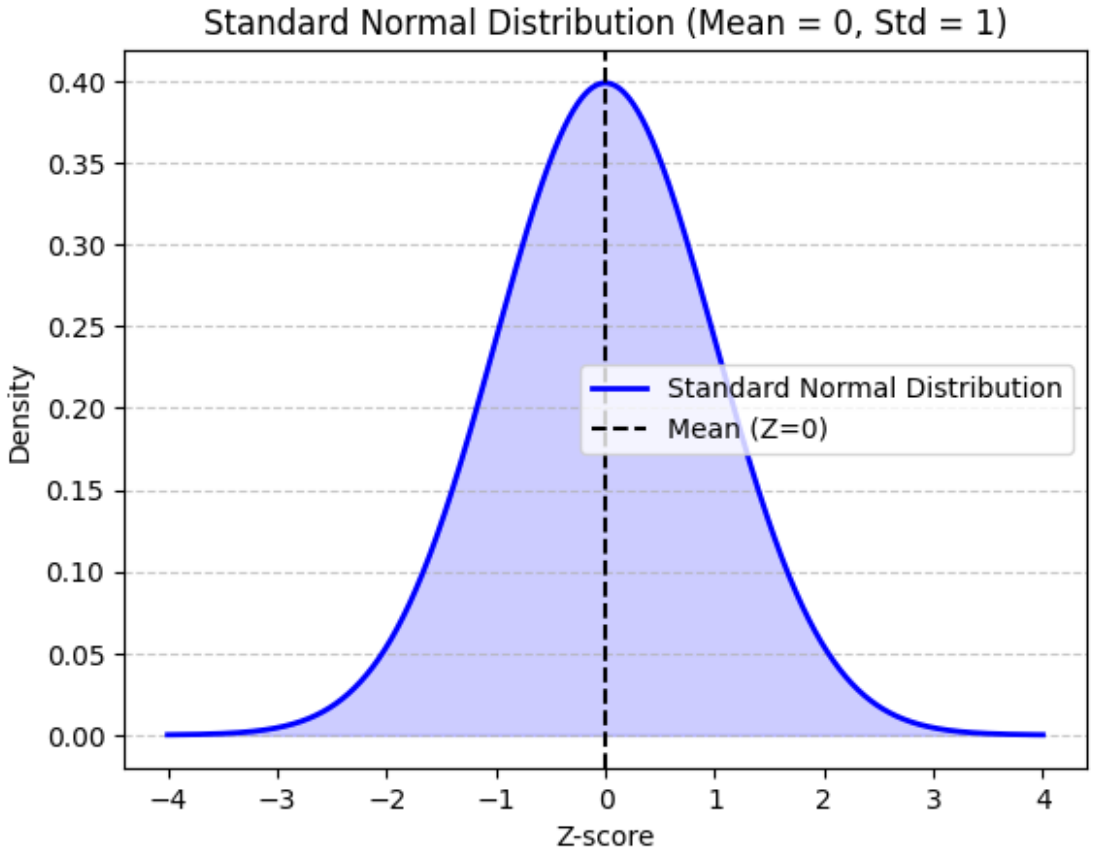

```python
#Q11
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

def central_limit_theorem_normal_demo(population_mean, population_std,
sample_size, num_samples):
    """
    Simulate multiple samples from a normal distribution and verify
the Central Limit Theorem (CLT).
    :param population_mean: Mean of the normal distribution
    :param population_std: Standard deviation of the normal
distribution
    :param sample_size: Number of elements in each sample
    :param num_samples: Number of samples to draw
    """
    # Generate a normal population
    population = np.random.normal(loc=population_mean,
scale=population_std, size=10000)

    # Draw multiple samples and compute their means
    sample_means = [np.mean(np.random.choice(population, sample_size,
```

```
replace=True)) for _ in range(num_samples)]

    # Plot histogram of sample means
    plt.hist(sample_means, bins=30, density=True, alpha=0.75,
color='blue', edgecolor='black')

    # Plot theoretical normal curve
    x = np.linspace(min(sample_means), max(sample_means), 1000)
    pdf = stats.norm.pdf(x, loc=population_mean, scale=population_std
/ np.sqrt(sample_size))
    plt.plot(x, pdf, 'r-', lw=2, label='Theoretical Normal Curve')

    plt.xlabel('Sample Mean')
    plt.ylabel('Density')
    plt.title(f'Central Limit Theorem Verification\nSample Size =
{sample_size}, Num Samples = {num_samples}')
    plt.legend()
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()

# Parameters
population_mean = 50  # Mean of the population
population_std = 10  # Standard deviation of the population
sample_size = 30  # Size of each sample
num_samples = 1000  # Number of samples

# Run CLT verification
central_limit_theorem_normal_demo(population_mean, population_std,
sample_size, num_samples)
```



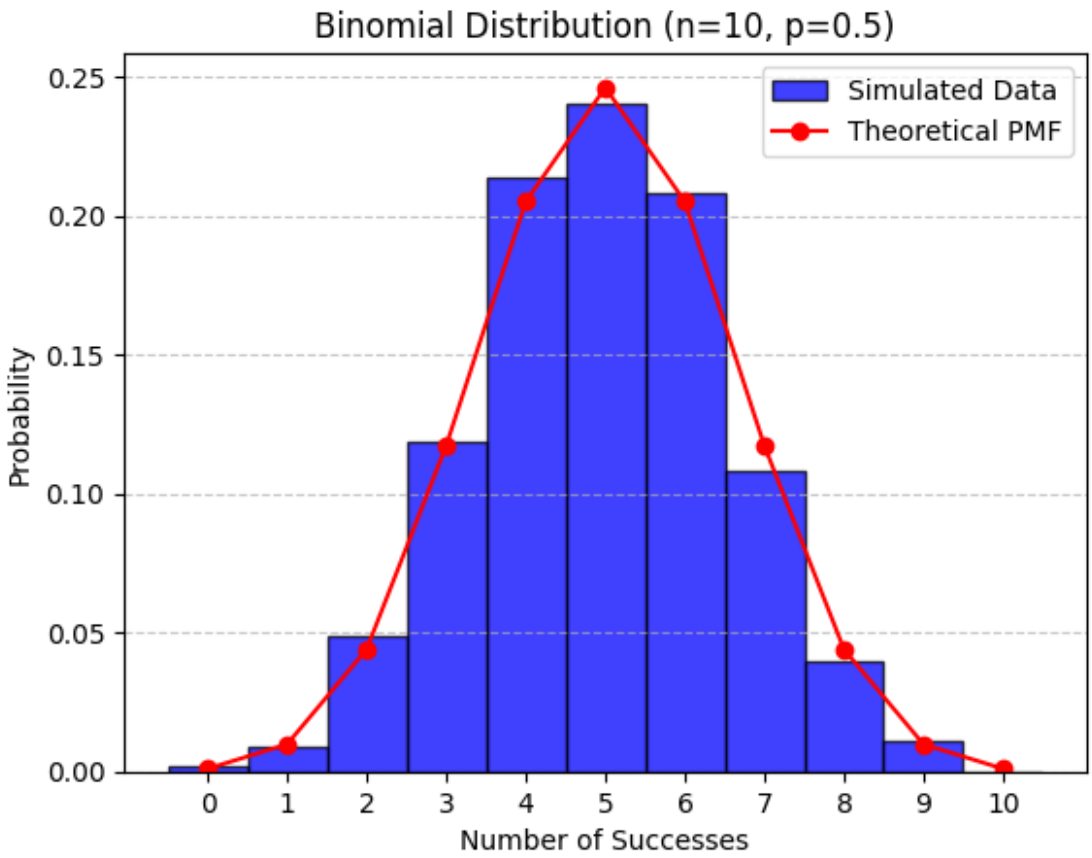

```python
#Q12
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

def standard_normal_distribution():
    """
    Calculate and plot the standard normal distribution (mean = 0, std
= 1).
    """
    x = np.linspace(-4, 4, 1000)  # Define the range for standard
normal distribution
    pdf = stats.norm.pdf(x, loc=0, scale=1)  # Calculate PDF

    plt.plot(x, pdf, 'b-', lw=2, label='Standard Normal Distribution')
    plt.fill_between(x, pdf, alpha=0.2, color='blue')

    plt.xlabel('Z-score')
    plt.ylabel('Density')
    plt.title('Standard Normal Distribution (Mean = 0, Std = 1)')
    plt.axvline(0, color='black', linestyle='--', label='Mean (Z=0)')
    plt.legend()
```

```
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()

# Run function to plot standard normal distribution
standard_normal_distribution()
```



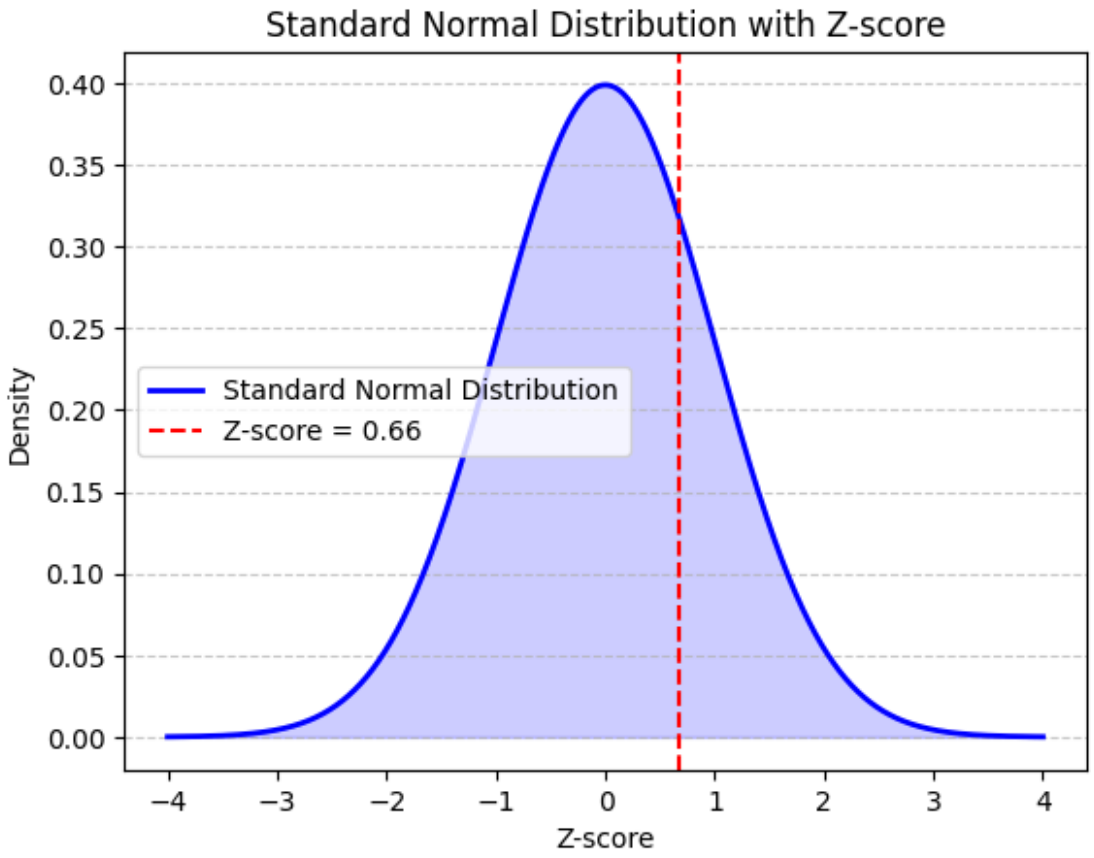

```
#Q13
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import binom

def binomial_distribution(n, p, size=1000):
    """
    Generate random variables and calculate their corresponding
probabilities using the binomial distribution.
    :param n: Number of trials
    :param p: Probability of success
    :param size: Number of random samples
    :return: Array of binomial random variables
    """
    return binom.rvs(n, p, size=size)
```

```
# Parameters
n = 10  # Number of trials
p = 0.5  # Probability of success
size = 1000  # Number of random samples

# Generate binomial random variables
samples = binomial_distribution(n, p, size)

# Compute probability mass function (PMF)
k_values = np.arange(0, n + 1)
probabilities = binom.pmf(k_values, n, p)

# Plot histogram of generated samples
plt.hist(samples, bins=np.arange(n+2)-0.5, density=True, alpha=0.75,
color='blue', edgecolor='black', label='Simulated Data')

# Plot theoretical PMF
plt.plot(k_values, probabilities, 'ro-', label='Theoretical PMF')

plt.xlabel('Number of Successes')
plt.ylabel('Probability')
plt.title('Binomial Distribution (n=10, p=0.5)')
plt.xticks(k_values)
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```




```python
#Q14
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

def calculate_z_score(data_point, data):
    """
    Calculate the Z-score for a given data point.
    :param data_point: The specific value for which to calculate the
Z-score
    :param data: List or array of numerical values
    :return: Z-score of the data point
    """
    mean = np.mean(data)
    std_dev = np.std(data, ddof=1)  # Sample standard deviation
    return (data_point - mean) / std_dev

# Generate a random normal dataset
np.random.seed(42)
data = np.random.normal(loc=50, scale=10, size=100)
data_point = 55  # Example data point

# Calculate Z-score
```

```
z_score = calculate_z_score(data_point, data)
print(f"Z-score for data point {data_point}: {z_score:.2f}")

# Plot standard normal distribution
x = np.linspace(-4, 4, 1000)
pdf = stats.norm.pdf(x, loc=0, scale=1)
plt.plot(x, pdf, 'b-', lw=2, label='Standard Normal Distribution')
plt.fill_between(x, pdf, alpha=0.2, color='blue')

# Mark the Z-score on the plot
plt.axvline(z_score, color='red', linestyle='--', label=f'Z-score =
{z_score:.2f}')
plt.xlabel('Z-score')
plt.ylabel('Density')
plt.title('Standard Normal Distribution with Z-score')
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

Z-score for data point 55: 0.66
```




```
#Q15
import numpy as np
```

```python
import scipy.stats as stats

def z_test(sample_data, population_mean, population_std, alpha=0.05):
    """
    Perform hypothesis testing using Z-statistics.
    :param sample_data: List or array of sample values
    :param population_mean: Mean of the population (null hypothesis)
    :param population_std: Standard deviation of the population
    :param alpha: Significance level (default = 0.05)
    :return: Z-score, p-value, and test result (reject or fail to
reject H0)
    """
    sample_mean = np.mean(sample_data)
    sample_size = len(sample_data)
    standard_error = population_std / np.sqrt(sample_size)

    # Compute Z-score
    z_score = (sample_mean - population_mean) / standard_error

    # Compute p-value (two-tailed test)
    p_value = 2 * (1 - stats.norm.cdf(abs(z_score)))

    # Determine whether to reject the null hypothesis
    result = "Reject H0" if p_value < alpha else "Fail to reject H0"

    return z_score, p_value, result

# Generate a sample dataset
np.random.seed(42)
sample_data = np.random.normal(loc=52, scale=10, size=30)  # Sample
with mean 52, std 10

# Population parameters
population_mean = 50  # Hypothesized population mean
population_std = 10  # Known population standard deviation

# Perform Z-test
z_score, p_value, result = z_test(sample_data, population_mean,
population_std)

# Print results
print(f"Z-score: {z_score:.2f}")
print(f"P-value: {p_value:.4f}")
print(f"Hypothesis Test Result: {result}")
```

```
Z-score: 0.06
P-value: 0.9482
Hypothesis Test Result: Fail to reject H0
```

```python
#Q16
import numpy as np
```

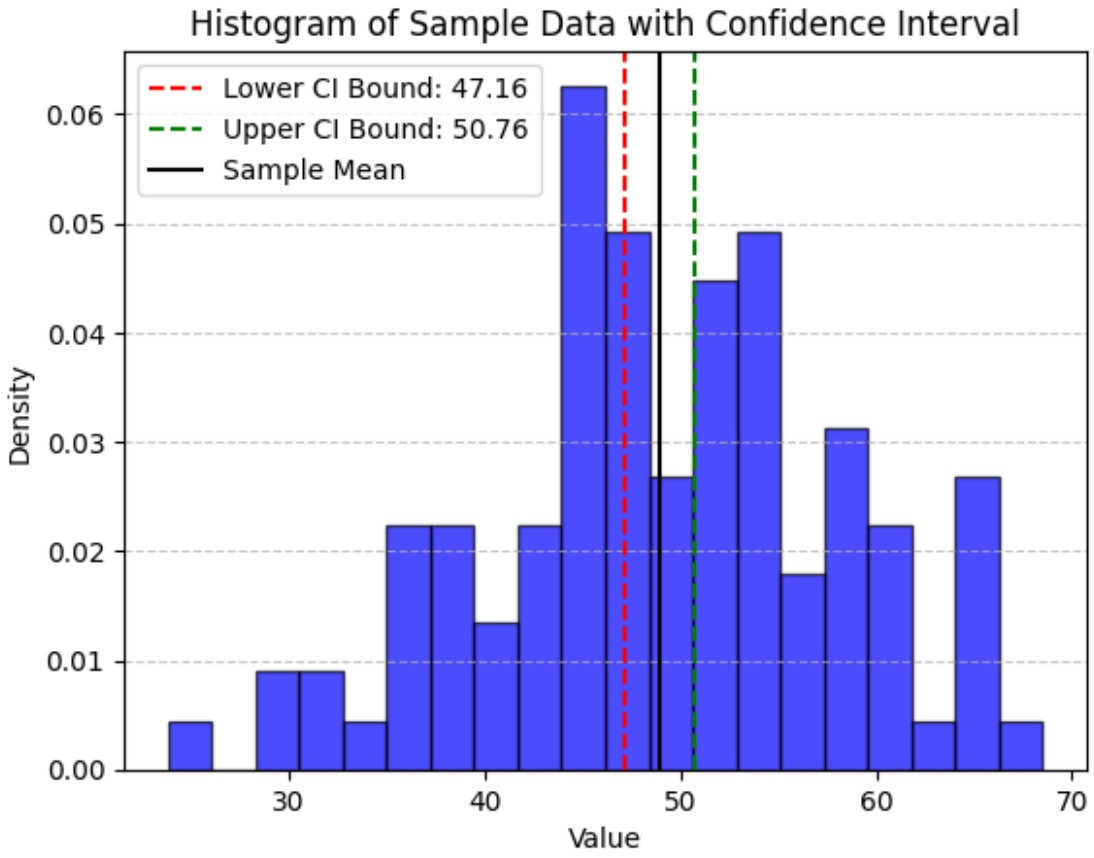
```
import scipy.stats as stats

def confidence_interval(data, confidence=0.95):
    """
    Calculate the confidence interval for a dataset.
    :param data: List or array of numerical values
    :param confidence: Confidence level (default = 0.95)
    :return: Tuple containing (lower bound, upper bound)
    """
    sample_mean = np.mean(data)
    sample_std = np.std(data, ddof=1)  # Sample standard deviation
    sample_size = len(data)

    # Compute the margin of error
    margin_of_error = stats.t.ppf((1 + confidence) / 2, df=sample_size
- 1) * (sample_std / np.sqrt(sample_size))

    # Compute confidence interval
    lower_bound = sample_mean - margin_of_error
    upper_bound = sample_mean + margin_of_error

    return lower_bound, upper_bound

# Generate a sample dataset
np.random.seed(42)
data = np.random.normal(loc=50, scale=10, size=30)  # Sample with mean
50, std 10

# Compute confidence interval
confidence_level = 0.95
ci_lower, ci_upper = confidence_interval(data, confidence_level)

# Print results
print(f"{confidence_level*100}% Confidence Interval: ({ci_lower:.2f},
{ci_upper:.2f})")
print("Interpretation: We are 95% confident that the true population
mean lies within this interval.")
```

```
95.0% Confidence Interval: (44.76, 51.48)
Interpretation: We are 95% confident that the true population mean
lies within this interval.
```

```
#Q17
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

def confidence_interval(data, confidence=0.95):
    """
    Calculate the confidence interval for the mean of a normal
```

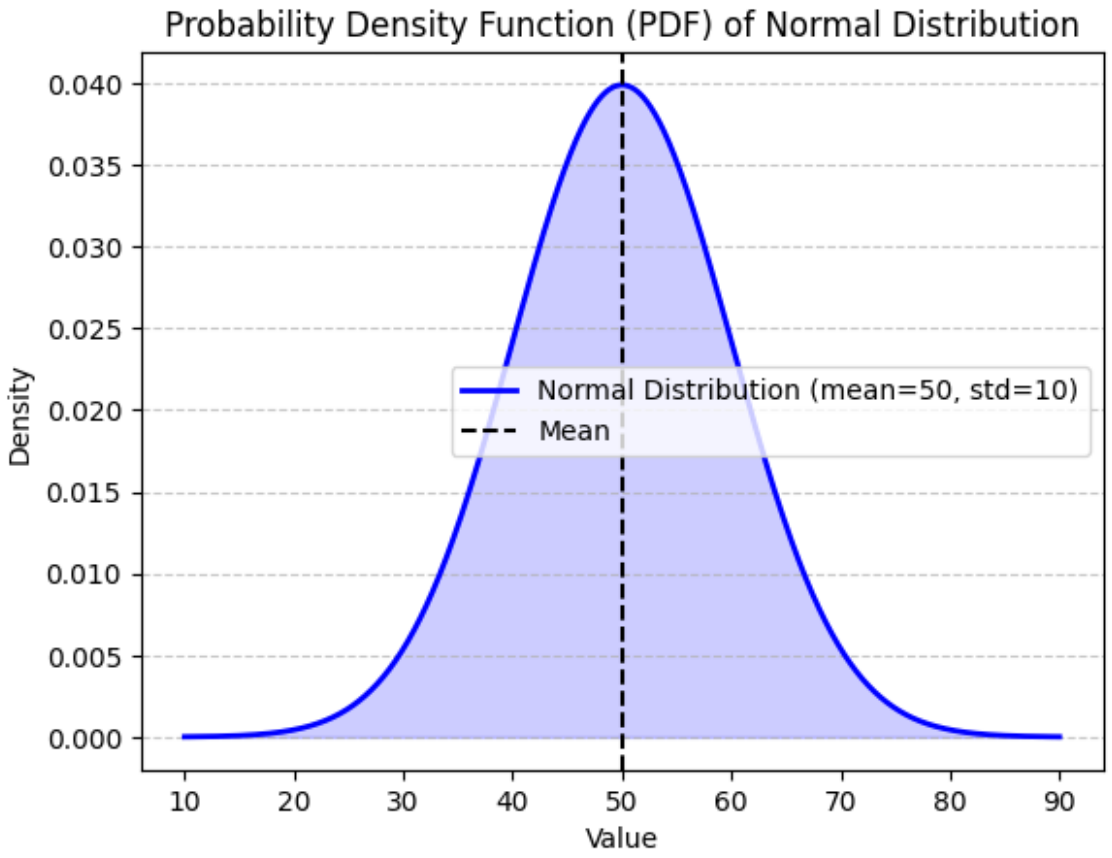
```
distribution dataset.
    :param data: List or array of numerical values
    :param confidence: Confidence level (default = 0.95)
    :return: Tuple containing (lower bound, upper bound)
    """
    sample_mean = np.mean(data)
    sample_std = np.std(data, ddof=1)  # Sample standard deviation
    sample_size = len(data)

    # Compute the margin of error
    margin_of_error = stats.t.ppf((1 + confidence) / 2, df=sample_size
- 1) * (sample_std / np.sqrt(sample_size))

    # Compute confidence interval
    lower_bound = sample_mean - margin_of_error
    upper_bound = sample_mean + margin_of_error

    return lower_bound, upper_bound

# Generate data from a normal distribution
np.random.seed(42)
data = np.random.normal(loc=50, scale=10, size=100)  # Sample with
mean 50, std 10

# Compute confidence interval
confidence_level = 0.95
ci_lower, ci_upper = confidence_interval(data, confidence_level)

# Print results
print(f"{confidence_level*100}% Confidence Interval for the Mean:
({ci_lower:.2f}, {ci_upper:.2f})")
print("Interpretation: We are 95% confident that the true population
mean lies within this interval.")

# Plot histogram with confidence interval
plt.hist(data, bins=20, alpha=0.7, color='blue', edgecolor='black',
density=True)
plt.axvline(ci_lower, color='red', linestyle='--', label=f'Lower CI
Bound: {ci_lower:.2f}')
plt.axvline(ci_upper, color='green', linestyle='--', label=f'Upper CI
Bound: {ci_upper:.2f}')
plt.axvline(np.mean(data), color='black', linestyle='-', label='Sample
Mean')
plt.xlabel('Value')
plt.ylabel('Density')
plt.title('Histogram of Sample Data with Confidence Interval')
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
95.0% Confidence Interval for the Mean: (47.16, 50.76)
Interpretation: We are 95% confident that the true population mean
lies within this interval.
```



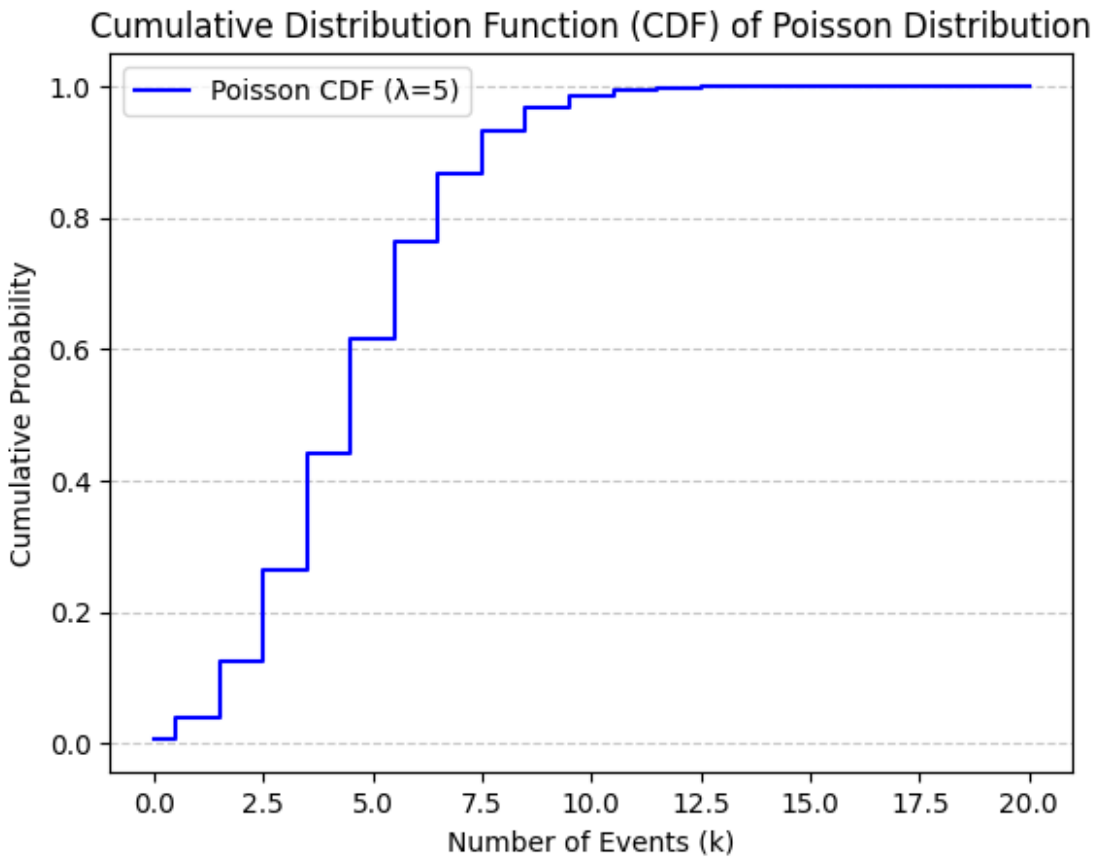

```python
#Q18
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

def normal_distribution_pdf(mean=0, std=1, num_points=1000):
    """
    Calculate and visualize the probability density function (PDF) of
a normal distribution.
    :param mean: Mean of the normal distribution
    :param std: Standard deviation of the normal distribution
    :param num_points: Number of points for plotting
    """
    x = np.linspace(mean - 4*std, mean + 4*std, num_points)
    pdf = stats.norm.pdf(x, loc=mean, scale=std)

    plt.plot(x, pdf, 'b-', lw=2, label=f'Normal Distribution
(mean={mean}, std={std})')
```

```
    plt.fill_between(x, pdf, alpha=0.2, color='blue')

    plt.xlabel('Value')
    plt.ylabel('Density')
    plt.title('Probability Density Function (PDF) of Normal
Distribution')
    plt.axvline(mean, color='black', linestyle='--', label='Mean')
    plt.legend()
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()

# Example usage
normal_distribution_pdf(mean=50, std=10)
```



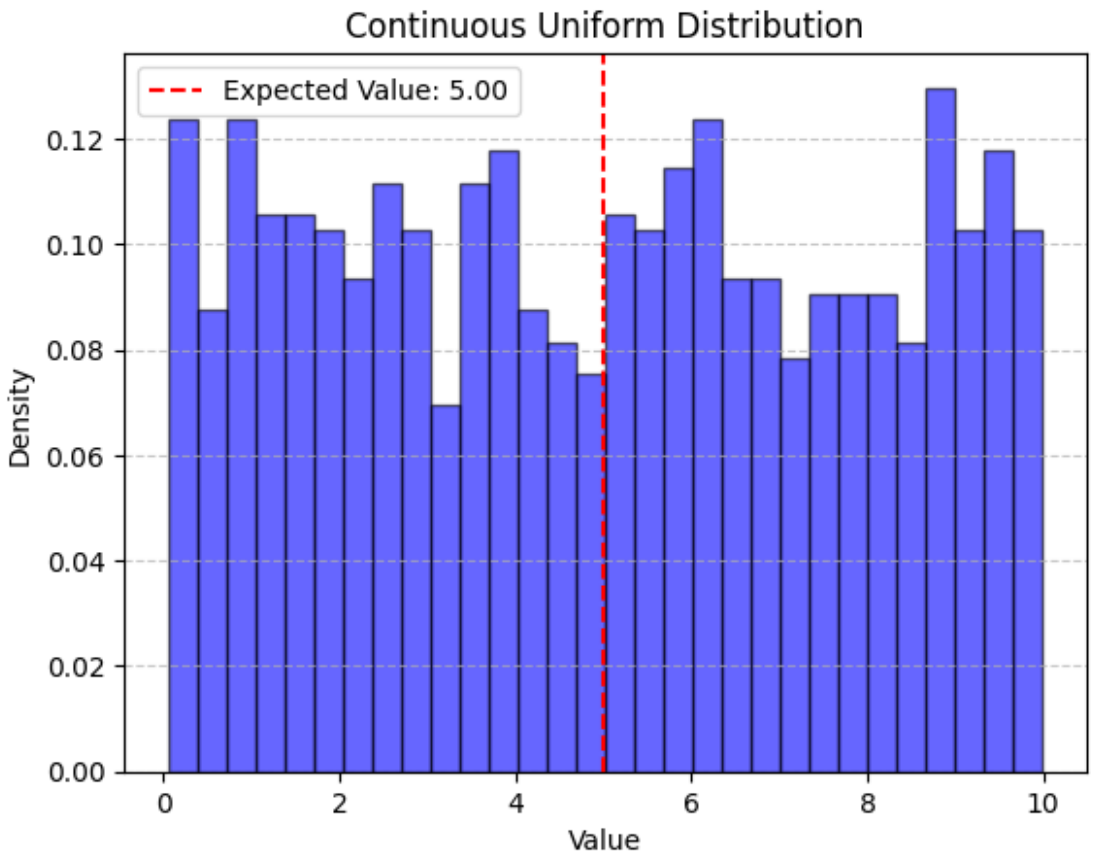

```
#Q19
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

def poisson_cdf(lam=5, max_k=20):
    """
    Calculate and visualize the cumulative distribution function (CDF)
```

```
of a Poisson distribution.
    :param lam: Lambda (mean) of the Poisson distribution
    :param max_k: Maximum value for plotting the CDF
    """
    k_values = np.arange(0, max_k + 1)
    cdf_values = stats.poisson.cdf(k_values, mu=lam)

    plt.step(k_values, cdf_values, where='mid', color='b',
label=f'Poisson CDF (λ={lam})')
    plt.xlabel('Number of Events (k)')
    plt.ylabel('Cumulative Probability')
    plt.title('Cumulative Distribution Function (CDF) of Poisson
Distribution')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.legend()
    plt.show()

    print("Interpretation: The CDF provides the probability that the
Poisson-distributed random variable is at most k.")
    for k, cdf_val in zip(k_values, cdf_values):
        print(f"P(X ≤ {k}) = {cdf_val:.4f}")

# Example usage
poisson_cdf(lam=5, max_k=20)
```




```
Interpretation: The CDF provides the probability that the Poisson-
distributed random variable is at most k.
P(X ≤ 0) = 0.0067
P(X ≤ 1) = 0.0404
P(X ≤ 2) = 0.1247
P(X ≤ 3) = 0.2650
P(X ≤ 4) = 0.4405
P(X ≤ 5) = 0.6160
P(X ≤ 6) = 0.7622
P(X ≤ 7) = 0.8666
P(X ≤ 8) = 0.9319
P(X ≤ 9) = 0.9682
P(X ≤ 10) = 0.9863
P(X ≤ 11) = 0.9945
P(X ≤ 12) = 0.9980
P(X ≤ 13) = 0.9993
P(X ≤ 14) = 0.9998
P(X ≤ 15) = 0.9999
P(X ≤ 16) = 1.0000
P(X ≤ 17) = 1.0000
P(X ≤ 18) = 1.0000
P(X ≤ 19) = 1.0000
P(X ≤ 20) = 1.0000
```

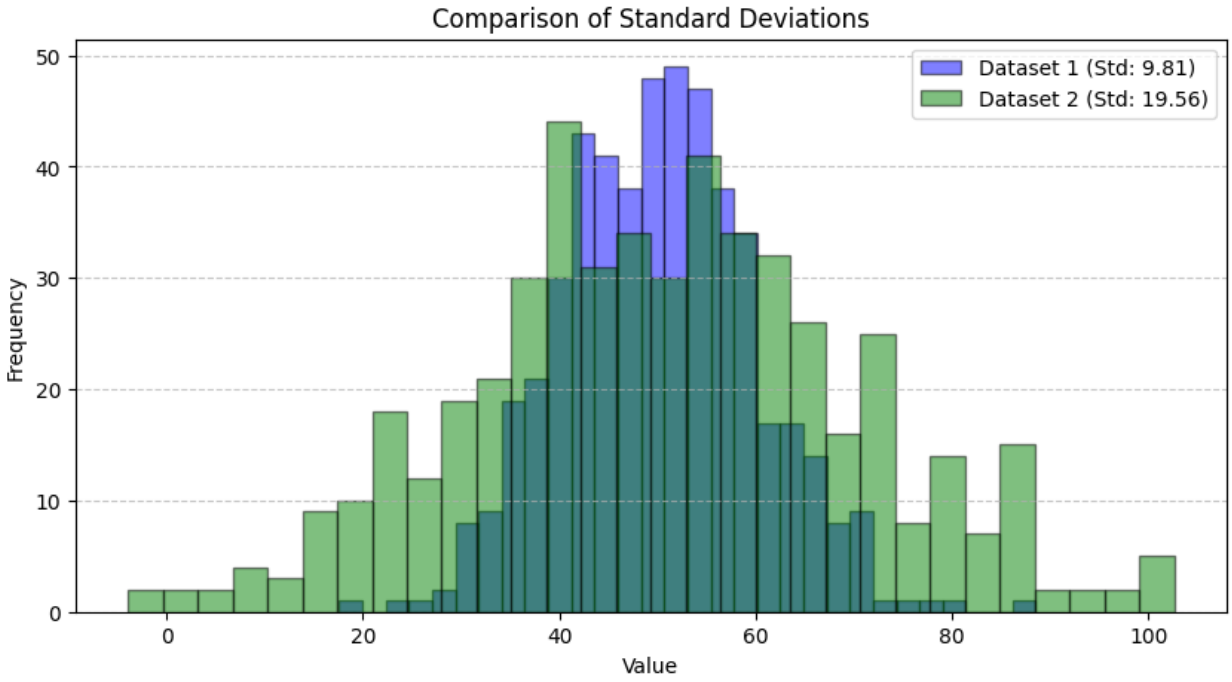
```python
#Q20
import numpy as np
import matplotlib.pyplot as plt

def simulate_uniform_distribution(a=0, b=10, size=1000):
    """
    Simulate a random variable using a continuous uniform
distribution,
    calculate its expected value, and visualize the distribution.
    :param a: Lower bound of the uniform distribution
    :param b: Upper bound of the uniform distribution
    :param size: Number of random samples
    """
    samples = np.random.uniform(a, b, size)
    expected_value = (a + b) / 2  # Expected value of Uniform(a, b)

    # Plot histogram
    plt.hist(samples, bins=30, density=True, alpha=0.6, color='blue',
edgecolor='black')
    plt.axvline(expected_value, color='red', linestyle='--',
label=f'Expected Value: {expected_value:.2f}')
    plt.xlabel('Value')
    plt.ylabel('Density')
    plt.title('Continuous Uniform Distribution')
    plt.legend()
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()

    print(f"Expected Value (Theoretical): {expected_value:.2f}")
    print(f"Sample Mean (Empirical): {np.mean(samples):.2f}")

# Example usage
simulate_uniform_distribution(a=0, b=10, size=1000)
```




```
Expected Value (Theoretical): 5.00
Sample Mean (Empirical): 4.98

#Q21
import numpy as np
import matplotlib.pyplot as plt

def compare_standard_deviations(data1, data2):
    """
    Compare the standard deviations of two datasets and visualize the
difference.
    :param data1: First dataset (array-like)
    :param data2: Second dataset (array-like)
    """
    std1 = np.std(data1, ddof=1)
    std2 = np.std(data2, ddof=1)

    print(f"Standard Deviation of Dataset 1: {std1:.2f}")
    print(f"Standard Deviation of Dataset 2: {std2:.2f}")

    # Visualization
    plt.figure(figsize=(10, 5))
    plt.hist(data1, bins=30, alpha=0.5, label=f'Dataset 1 (Std:
```

```
{std1:.2f})', color='blue', edgecolor='black')
    plt.hist(data2, bins=30, alpha=0.5, label=f'Dataset 2 (Std:
{std2:.2f})', color='green', edgecolor='black')

    plt.xlabel('Value')
    plt.ylabel('Frequency')
    plt.title('Comparison of Standard Deviations')
    plt.legend()
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()

# Generate sample datasets
np.random.seed(42)
data1 = np.random.normal(loc=50, scale=10, size=500)  # Mean 50, Std
10
data2 = np.random.normal(loc=50, scale=20, size=500)  # Mean 50, Std
20

# Compare standard deviations
compare_standard_deviations(data1, data2)
```

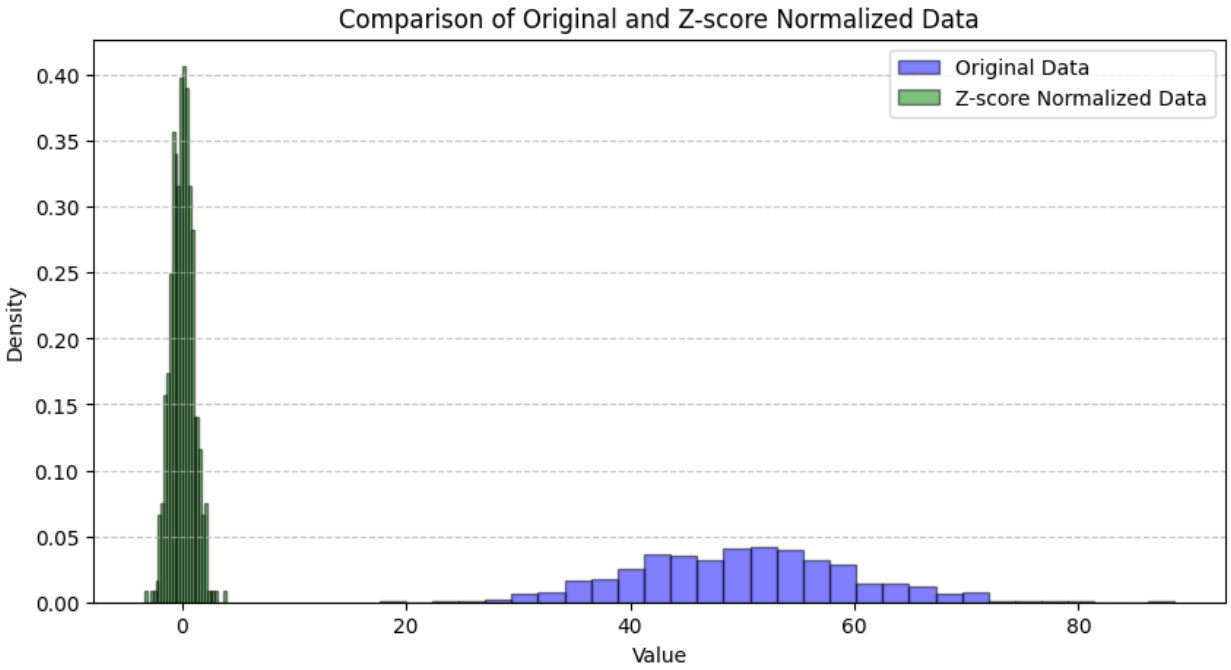
```
Standard Deviation of Dataset 1: 9.81
Standard Deviation of Dataset 2: 19.56
```




```
#Q23
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
```

```
def z_score_normalization(data):
    """
    Perform Z-score normalization on a dataset and visualize the
transformation.
    :param data: List or array of numerical values
    :return: Normalized dataset
    """
    mean = np.mean(data)
    std = np.std(data, ddof=1)
    normalized_data = (data - mean) / std

    print(f"Original Mean: {mean:.2f}, Original Std Dev: {std:.2f}")
    print(f"Normalized Mean: {np.mean(normalized_data):.2f},
Normalized Std Dev: {np.std(normalized_data, ddof=1):.2f}")

    # Visualization
    plt.figure(figsize=(10, 5))
    plt.hist(data, bins=30, alpha=0.5, label='Original Data',
color='blue', edgecolor='black', density=True)
    plt.hist(normalized_data, bins=30, alpha=0.5, label='Z-score
Normalized Data', color='green', edgecolor='black', density=True)
    plt.xlabel('Value')
    plt.ylabel('Density')
    plt.title('Comparison of Original and Z-score Normalized Data')
    plt.legend()
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()

    return normalized_data

# Generate a dataset from a normal distribution
np.random.seed(42)
data = np.random.normal(loc=50, scale=10, size=500)  # Mean 50, Std 10

# Perform Z-score normalization
z_score_normalization(data)
```

```
Original Mean: 50.07, Original Std Dev: 9.81
Normalized Mean: -0.00, Normalized Std Dev: 1.00
```




```
array([ 4.99235197e-01, -1.47874462e-01,  6.53093934e-01,
1.54515857e+00,
       -2.45595487e-01, -2.45578756e-01,  1.60241490e+00,
7.75127865e-01,
       -4.85412285e-01,  5.45956969e-01, -4.79239879e-01, -
4.81596111e-01,
        2.39616305e-01, -1.95680192e+00, -1.76484086e+00, -
5.79998614e-01,
       -1.03914980e+00,  3.13282365e-01, -9.32340425e-01, -
1.44625427e+00,
        1.48668122e+00, -2.37058370e-01,  6.18496910e-02, -
1.45893650e+00,
       -5.61751740e-01,  1.06073122e-01, -1.17995184e+00,
3.75907060e-01,
       -6.19082470e-01, -3.04235166e-01, -6.20170795e-01,
1.88069715e+00,
       -2.07237218e-02, -1.08488703e+00,  8.31290923e-01, -
1.25113639e+00,
        2.05885281e-01, -2.00407807e+00, -1.36052955e+00,
1.93653618e-01,
        7.45606282e-01,  1.67673623e-01, -1.24826366e-01, -
3.13824888e-01,
       -1.51373765e+00, -7.40565399e-01, -4.76407866e-01,
1.07034981e+00,
        3.43214451e-01, -1.80369151e+00,  3.23306930e-01, -
3.99407876e-01,
       -6.96823166e-01,  6.16393674e-01,  1.04372804e+00,
9.42103506e-01,
       -8.62219330e-01, -3.22088484e-01,  3.30623555e-01,
```

```
9.87214193e-01,
       -4.95297451e-01, -1.96174608e-01, -1.13444004e+00, -
1.22602867e+00,
        8.21080419e-01,  1.37518224e+00, -8.03545021e-02,
1.01573667e+00,
        3.61576414e-01, -6.64413342e-01,  3.61331401e-01,
1.56045198e+00,
       -4.34791261e-02,  1.58756739e+00, -2.67676373e+00,
8.30636242e-01,
        8.17414606e-02, -3.11688492e-01,  8.65452239e-02, -
2.03250987e+00,
       -2.30837333e-01,  3.56966540e-01,  1.49916044e+00, -
5.35140357e-01,
       -8.30908432e-01, -5.18311699e-01,  9.25922157e-01,
3.28063235e-01,
       -5.46849857e-01,  5.16104726e-01,  9.19635731e-02,
9.80182230e-01,
       -7.22434387e-01, -3.40890735e-01, -4.06567977e-01, -
1.49844390e+00,
        2.94808994e-01,  2.59074075e-01, -1.75748508e-03, -
2.46037533e-01,
       -1.44937990e+00, -4.35650347e-01, -3.56230680e-01, -
8.24573336e-01,
       -1.71335694e-01,  4.04801577e-01,  1.91525267e+00,
1.70944472e-01,
        2.55502233e-01, -8.28368320e-02, -1.96239780e+00, -
3.39890544e-02,
        5.44122687e-02,  2.50333349e+00, -2.03004637e-01,
3.00339743e-01,
       -4.23435687e-02, -1.19797416e+00,  1.15768770e+00,
7.59330010e-01,
        7.99175905e-01, -9.33729852e-01,  1.42262593e+00, -
1.43560193e+00,
        5.91100311e-01,  2.22533544e+00, -1.01642906e+00, -
5.84085429e-01,
        9.45865614e-02, -5.20063144e-01, -1.58725735e+00,
6.29042303e-02,
       -1.08956756e+00,  4.75671736e-01, -9.43958383e-01,
1.57257712e+00,
       -8.05185908e-01, -3.35183106e-01,  8.22090755e-01, -
1.26134850e+00,
        2.24836902e-01,  1.32514696e+00, -1.64516269e+00,
1.81192637e-01,
        2.57879197e-01,  7.89790892e-01, -1.26755117e+00, -
1.35265245e+00,
        5.24944577e-01,  2.95689904e-01,  2.48309859e-01,
3.46098437e-01,
       -6.99985165e-01,  2.29722249e-01,  2.91702962e-01, -
7.34967670e-01,
```

```
        1.89445133e+00,  4.75916821e-01, -1.22103187e+00,
6.62128371e-01,
       -1.00027151e+00,  7.95153149e-01,  1.17376181e+00, -
8.43330012e-01,
        9.74812707e-01,  4.13698435e-01,  8.30796910e-01,
1.92606240e+00,
       -2.57044867e-01, -7.75104858e-01, -9.13477154e-01, -
8.38364899e-01,
       -8.55433643e-02,  3.40701018e-01,  2.75008318e-01,
8.36017875e-01,
        6.28165798e-03,  1.47433508e+00, -2.76681711e-01,
2.76516911e+00,
        6.30652031e-01, -8.80502106e-01, -1.09832043e+00,
4.84721372e-01,
       -2.34700655e-01,  7.20672774e-01,  4.75310152e-01, -
8.11889361e-02,
       -8.69940268e-01, -1.55075687e+00, -4.62014213e-01,
8.65791580e-01,
        2.11215351e-01, -1.27650714e+00,  1.69520898e-01,
3.85710199e-01,
       -9.07712085e-01,  1.49693376e-01,  5.23521568e-02, -
1.17177527e+00,
        3.57654221e-01,  5.64529629e-01,  1.09677420e+00,
1.06696621e+00,
       -1.41095825e+00, -9.62710735e-01,  5.17906335e-01,
5.16633150e-01,
        5.17918991e-01,  3.91936894e+00,  5.74828687e-01,
1.15029188e+00,
        9.65259245e-01,  6.56867387e-01, -3.28261068e-01,
7.66500623e-01,
       -7.94558603e-01, -2.48311638e-01, -5.01605007e-01,
7.64697034e-02,
        2.35191127e+00, -1.90990776e+00,  6.92402494e-01, -
1.65049529e+00,
       -4.87916714e-01,  1.10278626e+00,  5.85394491e-02, -
1.10530363e+00,
       -7.35938154e-01,  6.85612767e-01, -7.51288853e-01,
2.13625377e-01,
        3.94738519e-02, -6.71017746e-01,  2.17793531e+00,
6.39061353e-01,
       -2.07080138e+00,  1.83047873e-01, -6.81398468e-01,
8.61750361e-01,
       -8.14630408e-01, -1.23897104e-01,  5.07666380e-01,
8.75326733e-01,
       -1.23019659e+00, -3.47860485e-01, -4.90987731e-01, -
6.72779661e-01,
        1.79221445e+00,  4.05750215e-01, -1.29194166e+00,
9.28428981e-01,
        2.15573116e+00,  1.04522178e+00, -1.55536602e+00, -
```

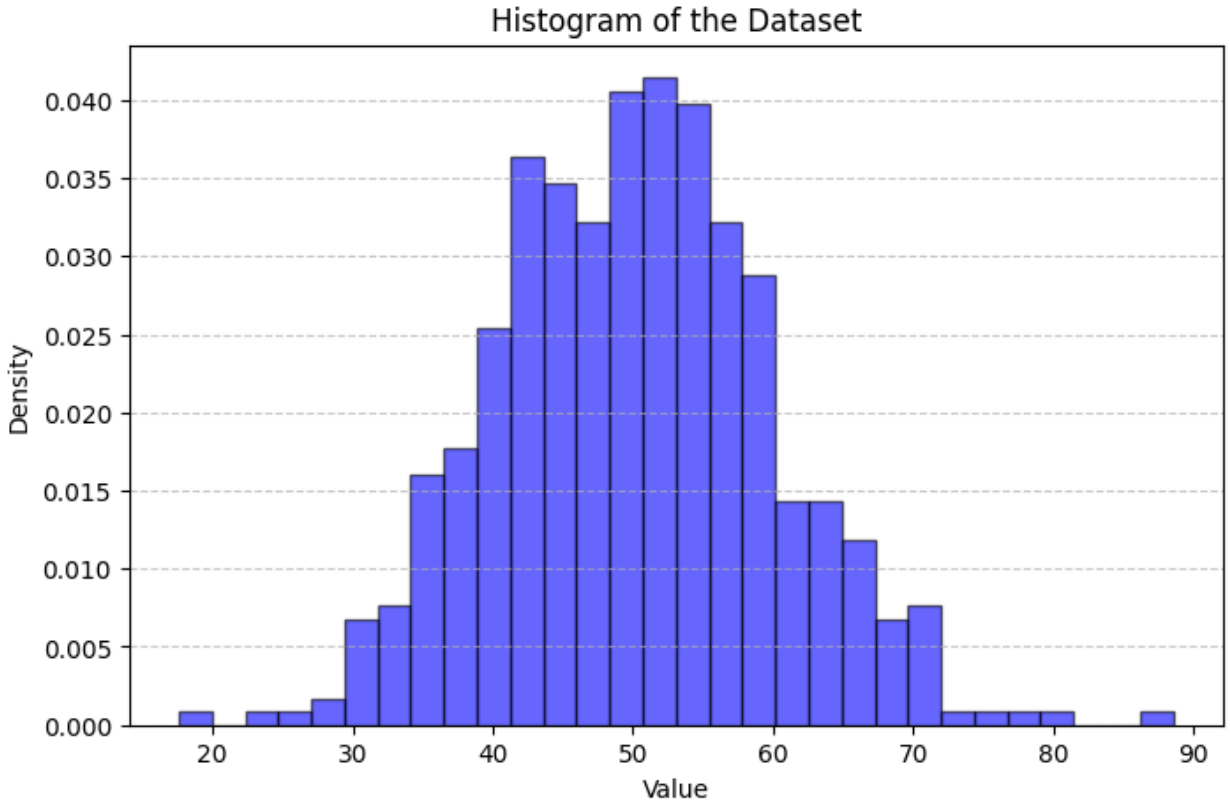
```
5.00453954e-01,
        1.28414674e+00, -7.28158059e-01,  4.45329923e-01,
7.82464732e-01,
       -9.51608026e-01, -6.76312163e-02, -3.31016009e+00, -
1.05092711e+00,
       -2.64362076e-01, -1.27859060e+00,  1.65662974e+00, -
1.46443273e+00,
       -4.55420130e-01,  1.26269730e-01,  1.46184005e+00, -
1.47026280e+00,
        1.17841725e+00,  3.45992886e-03, -1.00722892e+00,
4.63963285e-01,
        1.95894079e-01, -6.18652599e-01,  6.41670135e-02, -
3.99643611e-01,
        1.08717450e-01,  6.67811986e-01,  1.60934889e+00, -
1.26843248e+00,
        2.16681615e+00, -1.99635089e+00, -1.61653569e-01,
5.92588319e-01,
        2.79391558e-01, -6.41564770e-01, -2.19067041e-01, -
5.09388306e-01,
       -6.07593150e-01,  8.58865033e-01,  3.56867600e-01, -
7.13116203e-01,
        9.09818014e-01,  3.06201816e-01,  8.21423141e-01,
6.34689209e-01,
       -8.51801518e-01, -5.77851881e-01,  7.54601947e-01,
6.15062699e-01,
       -2.82695508e-02,  1.12600278e-01,  1.29510593e+00, -
6.09841939e-01,
        5.50580992e-01, -2.13024158e-01, -2.28808616e-01,
1.11280025e+00,
        8.34217218e-01,  8.22083029e-01,  1.32345123e+00,
1.44364842e-02,
        6.88012986e-01, -3.23163008e-01,  3.23390887e-01, -
1.39598059e-01,
        9.18804301e-02,  5.99558812e-01, -8.40821348e-01,
2.12539351e+00,
       -1.03220588e+00, -1.24435421e+00,  1.17326784e+00,
7.99818703e-01,
        6.29074935e-01,  6.33381358e-01, -1.94493802e-02, -
9.21364967e-01,
        7.02841634e-02, -6.97067457e-01,  9.86780672e-01, -
1.56835533e-01,
       -8.48236878e-01, -3.34494522e-01,  4.13851838e-01, -
5.81463092e-01,
       -8.44897474e-01,  2.41374199e-01,  2.42678001e-01, -
5.23596912e-01,
       -4.87006083e-01,  2.29514596e-01, -1.48271849e+00, -
1.44132187e+00,
       -7.39138666e-01, -2.24493674e-01,  3.09878792e-01,
1.49657413e+00,
```

```
        8.67076497e-01, -1.69962775e-01, -2.63481446e-02, -
1.02865123e+00,
       -2.58354616e-02, -3.01142069e-01,  3.21915435e-01, -
8.50003748e-01,
        5.22299948e-01,  1.55505311e+00, -1.17806635e-01,
4.02417754e-01,
        6.96360495e-01, -4.15854386e-01,  2.21405114e-01,
5.86434359e-03,
        9.25735575e-02, -7.94746698e-01,  1.80098050e-02,
5.00543869e-01,
        1.47189894e+00,  9.70628973e-01,  2.18735018e+00, -
7.88976301e-01,
        8.82017608e-01,  1.79876104e-01,  2.22467028e+00, -
8.30709383e-01,
       -8.62733284e-01, -6.17812620e-01, -2.17144119e+00, -
5.42768157e-01,
       -7.80604455e-01,  1.46298412e-01,  3.41316558e-01,
1.90504628e+00,
        9.61612964e-01, -5.94893981e-01, -9.22547434e-01,
4.94348608e-01,
       -1.35242477e+00,  1.85947998e+00,  1.19500458e+00, -
4.85107843e-01,
       -1.75283244e+00,  1.37276935e+00, -1.23696752e-01,
1.25449604e+00,
       -1.63185769e+00, -6.17794661e-01, -1.62475373e-03,
4.09095198e-02,
       -4.65632565e-01,  6.27780789e-01, -1.09498585e+00, -
1.52068266e-01,
        1.15625235e-01,  5.17298507e-01,  7.18241581e-01, -
1.15309691e+00,
       -1.57039191e+00,  1.29511809e+00,  3.31694207e-01, -
7.69754937e-01,
        1.57381796e+00,  1.10915954e-01,  1.19485891e+00,
6.18397819e-02,
        2.09314969e+00,  1.78190783e+00, -2.60689219e-01,
9.83164090e-01,
        6.50737164e-01,  1.38781050e+00, -9.90326868e-01,
6.92189776e-01,
        1.07167696e+00, -1.79930868e+00, -1.21283319e+00, -
2.08516015e+00,
       -2.81522461e-01,  7.24282200e-01,  1.52409081e+00,
6.85417205e-02,
        1.65276146e+00, -1.41343680e+00, -1.74289404e+00, -
6.35775664e-02,
        3.84434350e-01, -4.02880121e-02, -2.11390902e+00, -
9.77913035e-02,
       -1.33635990e+00,  6.75497947e-01,  3.66633438e-01, -
9.64804737e-01,
       -5.30652931e-01, -1.08641833e+00, -7.08452095e-02,
```

```
9.66421592e-01,
       -1.01152689e+00,  5.06707644e-01, -5.47356775e-01, -
8.14989228e-01,
       -1.16043799e-01, -1.06198917e+00, -5.71195358e-01, -
1.22773187e+00,
        1.99529239e+00,  2.89686250e-02, -7.20062333e-01,
2.11099343e-01,
       -1.21442701e-01, -2.32159837e-01,  6.18931664e-01,
7.65011191e-01,
       -5.47604957e-01, -5.93787829e-01, -2.87275168e-01, -
2.35286779e+00,
       -1.55110728e+00,  1.38601964e+00,  1.66942604e+00, -
2.60762484e-01,
        5.80603397e-01,  3.10227926e-01,  3.13073391e+00,
1.13399565e+00,
       -1.37330079e-01, -9.80764586e-01, -1.64410596e+00,
2.00382156e-01,
       -7.77769390e-01, -1.45639437e+00, -6.65894233e-01, -
1.10917951e+00,
        1.71240569e+00,  8.91514769e-01, -1.50935917e-02,
1.50124970e+00,
        7.18777882e-02, -8.84707590e-01,  1.54525459e+00,
5.42237236e-01,
       -1.06403128e+00, -2.00943715e-01, -8.99315493e-01, -
1.41618663e+00])

#Q24
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

def calculate_skewness_kurtosis(data):
    """
    Calculate the skewness and kurtosis of a dataset.
    :param data: List or array of numerical values
    :return: Tuple containing (skewness, kurtosis)
    """
    skewness = stats.skew(data)
    kurtosis = stats.kurtosis(data)

    print(f"Skewness: {skewness:.2f}")
    print(f"Kurtosis: {kurtosis:.2f}")

    # Visualization
    plt.figure(figsize=(8, 5))
    plt.hist(data, bins=30, color='blue', alpha=0.6,
edgecolor='black', density=True)
    plt.xlabel('Value')
    plt.ylabel('Density')
    plt.title('Histogram of the Dataset')
```

```
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()

    return skewness, kurtosis

# Generate a dataset from a normal distribution
np.random.seed(42)
data = np.random.normal(loc=50, scale=10, size=500)  # Mean 50, Std 10

# Calculate skewness and kurtosis
calculate_skewness_kurtosis(data)
```

```
Skewness: 0.18
Kurtosis: 0.26
```

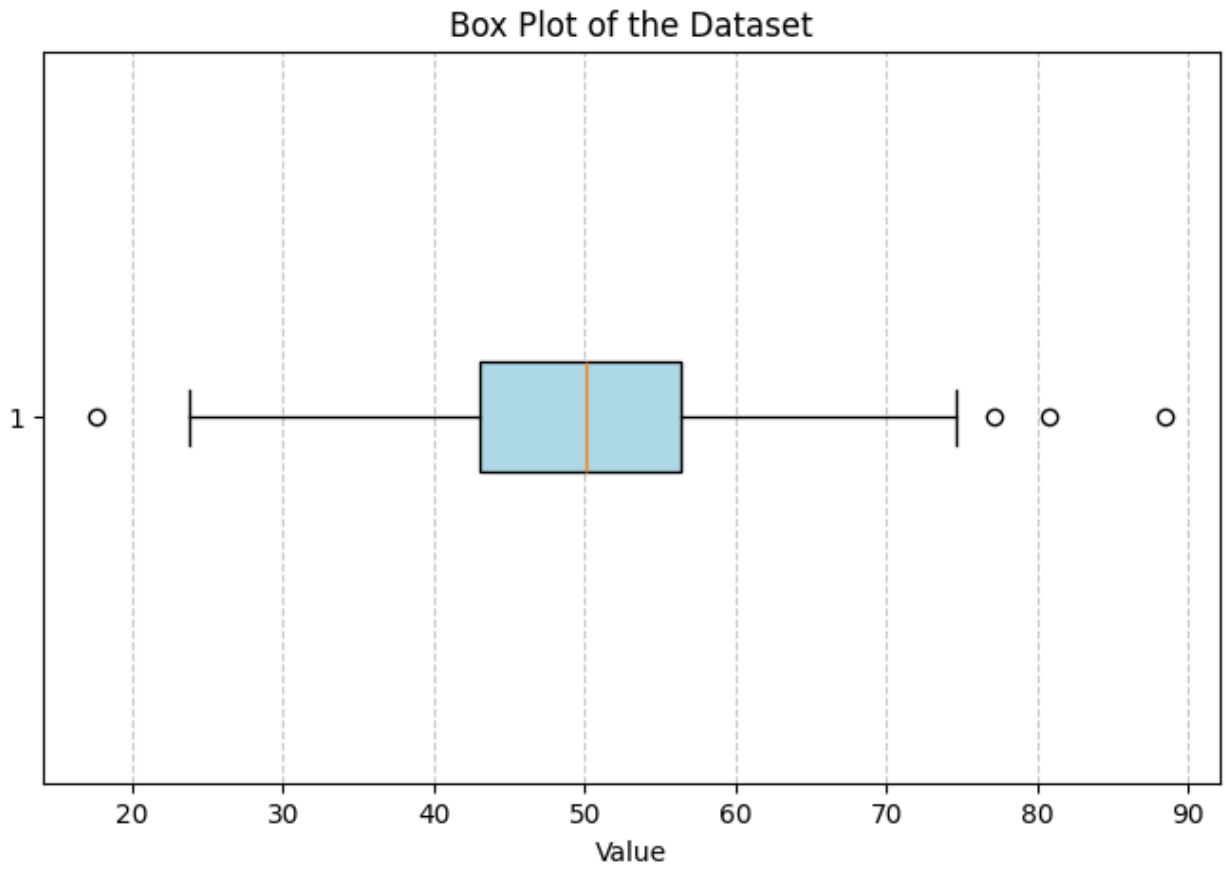



```
(np.float64(0.17962295069666998), np.float64(0.2563808006991657))
```

```
#Q22
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

def calculate_range_iqr(data):
    """
    Calculate the range and interquartile range (IQR) of a dataset.
```

```
    :param data: List or array of numerical values
    :return: Tuple containing (range, IQR)
    """
    data_range = np.max(data) - np.min(data)
    q1 = np.percentile(data, 25)
    q3 = np.percentile(data, 75)
    iqr = q3 - q1

    print(f"Range: {data_range:.2f}")
    print(f"Interquartile Range (IQR): {iqr:.2f}")

    # Visualization
    plt.figure(figsize=(8, 5))
    plt.boxplot(data, vert=False, patch_artist=True,
boxprops=dict(facecolor='lightblue'))
    plt.xlabel('Value')
    plt.title('Box Plot of the Dataset')
    plt.grid(axis='x', linestyle='--', alpha=0.7)
    plt.show()

    return data_range, iqr

# Generate a dataset from a normal distribution
np.random.seed(42)
data = np.random.normal(loc=50, scale=10, size=500)  # Mean 50, Std 10

# Calculate range and IQR
calculate_range_iqr(data)
```

```
Range: 70.94
Interquartile Range (IQR): 13.37
```




```
(np.float64(70.93998830723794), np.float64(13.370906585985722))
```