Theory Questions (Statistics Part - 2)

---

1. What is hypothesis testing in statistics?

Ans: Hypothesis testing is like a scientific experiment for your data. It's a formal procedure for determining whether there's enough evidence in a sample of data to infer that a certain condition is true for the entire population from which the sample was drawn. You start with an assumption about the population (your "null hypothesis") and use sample data to see if that assumption is likely to be false.

2. What is the null hypothesis, and how does it differ from the alternative hypothesis?

Ans:

Null Hypothesis (H_0): This is the statement of "no effect," "no difference," or "no relationship." It's the default assumption that you're trying to disprove. For example, H_0 might be "The new drug has no effect on blood pressure."

Alternative Hypothesis (H_1 or Ha): This is the statement you're trying to prove. It contradicts the null hypothesis and suggests that there is an effect, a difference, or a relationship. For example, H_1 might be "The new drug does reduce blood pressure." The goal of hypothesis testing is to decide whether to reject H_0 in favor of H_1.

3. What is the significance level in hypothesis testing, and why is it important?

Ans: The significance level (α) is a threshold you set before conducting your test. It represents the maximum probability of making a Type 1 error (rejecting a true null hypothesis) that you are willing to accept.

Importance: It determines how much evidence you need to consider a result "statistically significant." Common values are 0.05 (5%) or 0.01 (1%). If your p-value is less than α, you reject the null hypothesis.

4. What does a P-value represent in hypothesis testing?

Ans: The P-value (or probability value) is the probability of observing test results at least as extreme as the ones you got, assuming that the null hypothesis is true. It's a measure of the strength of evidence against the null hypothesis.

5. How do you interpret the P-value in hypothesis testing?

Ans:

Small P-value ($P \leq \alpha$): If the P-value is less than or equal to your chosen significance level (e.g., $P \leq 0.05$), it means that observing your data (or more extreme data) would be very unlikely if the null hypothesis were true. This provides strong evidence against the null hypothesis, so you reject the null hypothesis. You conclude there is a statistically significant effect or difference.

Large P-value ($P > \alpha$): If the P-value is greater than your significance level (e.g., $P > 0.05$), it means that your observed data is not unusual if the null hypothesis were true. This suggests there isn't enough evidence to reject the null hypothesis, so you fail to reject the null hypothesis. You conclude there is no statistically significant effect or difference based on your data.

6.What are Type 1 and Type 2 errors in hypothesis testing?

Ans: These are the two types of mistakes you can make in hypothesis testing:

Type 1 Error (False Positive): This occurs when you reject the null hypothesis (H_0) when it is actually true. The probability of making a Type 1 error is equal to your significance level (α).

Analogy: Convicting an innocent person.

Type 2 Error (False Negative): This occurs when you fail to reject the null hypothesis (H_0) when it is actually false. The probability of making a Type 2 error is denoted by β (beta).

Analogy: Letting a guilty person go free.

7. What is the difference between a one-tailed and a two-tailed test in hypothesis testing?

Ans: This refers to the directionality of your alternative hypothesis:

One-tailed test: Used when you are interested in a difference in only one direction (e.g., the new drug increases blood pressure, or the new drug decreases blood pressure). The critical region for rejecting H_0 is entirely in one tail of the distribution.

Two-tailed test: Used when you are interested in any difference, regardless of direction (e.g., the new drug changes blood pressure, either increasing or decreasing it). The critical region is split between both tails of the distribution.

8. What is the Z-test, and when is it used in hypothesis testing?

Ans: The Z-test is a type of hypothesis test used to compare means.

When to use: You use a Z-test when you know the population standard deviation (σ), or when your sample size is large ($n \geq 30$). In the latter case, due to the Central Limit Theorem, the sample standard deviation can be used as a good estimate for the population standard deviation, and the sample mean distribution approximates a normal distribution.

9. How do you calculate the Z-score, and what does it represent in hypothesis testing?

Ans: In hypothesis testing, the Z-score (or Z-statistic) measures how many standard errors a sample mean (or other statistic) is away from the hypothesized population mean.

Formula:

$$Z=\bar{x}-\mu0\sigma n$$

- $\bar{x}$
- : Sample mean
- $\mu0$
- : Hypothesized population mean (from the null hypothesis)
- σ
- : Population standard deviation

- n
- : Sample Size
- σn
- : Standard error of the mean

Representation: It represents the "standardized difference" between your sample observation and what you would expect if the null hypothesis were true. A larger absolute Z-score means your sample is further away from the hypothesized mean, providing stronger evidence against H_0.

10. What is the T-distribution, and when should it be used instead of the normal distribution?

Ans: The T-distribution (Student's t-distribution) is similar to the normal distribution but has fatter tails, meaning it accounts for more variability.

When to use: You use the T-distribution instead of the normal distribution when:

The population standard deviation (σ) is unknown.

The sample size (n) is small (typically $n < 30$).

The data are approximately normally distributed (or the sample size is sufficiently large by CLT).

11. What is the difference between a Z-test and a T-test?

Ans: The key difference lies in what you know about the population standard deviation and the sample size:

Z-test: Used when the population standard deviation (σ) is known, or when the sample size is large ($n \geq 30$), allowing the sample standard deviation to approximate σ. It uses the standard normal distribution.

T-test: Used when the population standard deviation (σ) is unknown and must be estimated from the sample standard deviation, especially with small sample sizes ($n < 30$). It uses the T-distribution, which adjusts for the added uncertainty of estimating σ.

12. What is the T-test, and how is it used in hypothesis testing?

Ans: The T-test is a hypothesis test used to determine if there is a significant difference between the means of two groups or between a sample mean and a hypothesized population mean, particularly when the population standard deviation is unknown and/or the sample size is small.

Usage:

One-sample T-test: Compares a sample mean to a known population mean (when σ is unknown).

Independent samples T-test: Compares the means of two independent groups.

Paired samples T-test: Compares the means of two related groups (e.g., before-after measurements).

13. What is the relationship between Z-test and T-test in hypothesis testing?

Ans: The T-test can be seen as a more generalized version of the Z-test. As the sample size (n) gets larger, the T-distribution approaches the standard normal (Z) distribution. When n is large (typically > 30), the T-test results will be very similar to the Z-test results, and the sample standard deviation becomes a very good estimate for the population standard deviation. So, for large samples, a T-test is practically equivalent to a Z-test.

14. What is a confidence interval, and how is it used to interpret statistical results?

Ans: A confidence interval is a range of values that is likely to contain the true population parameter (e.g., the population mean or proportion) with a certain level of confidence (e.g., 95% confidence).

Interpretation: If you construct a 95% confidence interval, it means that if you were to repeat the sampling and interval calculation many times, 95% of those intervals would contain the true population parameter. It gives you a sense of the precision of your estimate. If a hypothesized value falls outside the confidence interval, it's considered statistically unlikely and would lead to rejecting the null hypothesis.

15. What is the margin of error, and how does it affect the confidence interval?

Ans:The margin of error is the "plus or minus" part of a confidence interval. It's the maximum expected difference between the true population parameter and the sample estimate.

Formula (for mean): Margin of Error = (Critical Value) * (Standard Error)

Effect on CI: A larger margin of error results in a wider confidence interval, indicating less precision in your estimate. A smaller margin of error results in a narrower confidence interval, indicating more precision. The margin of error is influenced by the confidence level (higher confidence = larger margin), the sample size (larger sample size = smaller margin), and the variability of the data.

16. How is Bayes' Theorem used in statistics, and what is its significance?

Ans: Bayes' Theorem describes the probability of an event, based on prior knowledge of conditions that might be related to the event[cite: 15]. It's a fundamental concept in Bayesian statistics.

Formula: $P(A|B)=[P(B|A)*P(A)]/P(B)$

$P(A|B)$: Posterior probability (probability of A given B)

$P(B|A)$: Likelihood (probability of B given A)

$P(A)$: Prior probability (initial probability of A)

$P(B)$: Marginal probability of B

Significance: Updating Beliefs: It provides a formal way to update your beliefs about a hypothesis as new evidence becomes available. Incorporating Prior Knowledge: Unlike frequentist statistics (which often ignores prior beliefs), Bayesian statistics explicitly incorporates prior knowledge or beliefs into the analysis. Applications: Spam filtering, medical diagnosis, machine learning (e.g., Naive Bayes classifiers), financial modeling.

17. What is the Chi-square distribution, and when is it used?

Ans: The Chi-square (

x2

) distribution is a family of distributions that arise in hypothesis testing, particularly when dealing with categorical data. Its shape depends on its degrees of freedom.

When used:

Chi-square Goodness-of-Fit Test: To test if observed frequencies of categorical data match expected frequencies.

Chi-square Test of Independence: To test if there is a significant association between two categorical variables in a contingency table.

Confidence Intervals for Variance/Standard Deviation: In some cases, for normally distributed data.

18. What is the Chi-square goodness of fit test, and how is it applied? The Chi-square goodness-of-fit test is used to determine whether a sample of categorical data comes from a population with a hypothesized distribution. It checks if the observed frequencies of categories differ significantly from the expected frequencies under a given hypothesis.

Application:

1. State Hypotheses: H_0: The observed distribution matches the expected distribution. H_1: The observed distribution does not match the expected distribution.

2. Calculate Expected Frequencies: Determine how many observations you would expect in each category if H_0 were true.
3. Calculate Chi-square Statistic:
4. χ2=∑(Oi−Ei)2Ei
5. where
6. Oi
7. are observed frequencies and
8. Ei
9. are expected frequencies.
4. Determine P-value: Compare the calculated
5. x2
6. statistic to the Chi-square distribution with appropriate degrees of freedom to find the P-value.
5. Make Decision: If P-value < α, reject H_0.

19. What is the F-distribution, and when is it used in hypothesis testing?

Ans: The F-distribution (Fisher-Snedecor distribution) is a continuous probability distribution that arises in the context of comparing variances, particularly in ANOVA. It is defined by two degrees of freedom parameters.

When used:

ANOVA (Analysis of Variance): The primary use is in ANOVA tests to compare the means of three or more groups. The

F-statistic is the ratio of two variances. Comparing Variances of Two Populations: To test if two population variances are equal.

Regression Analysis: To test the overall significance of a regression model.

20. What is an ANOVA test, and what are its assumptions? ANOVA (Analysis of Variance) is a statistical test used to compare the means of three or more groups simultaneously. Instead of performing multiple t-tests (which increases the chance of Type 1 error), ANOVA uses variance to determine if there are significant differences between group means.

Assumptions:

Independence: The samples from each group must be independent.

Normality: The data within each group should be approximately normally distributed.

Homoscedasticity (Homogeneity of Variances): The variances of the populations from which the samples are drawn must be equal (or very similar).

21. What are the different types of ANOVA tests?

Ans: The common types of ANOVA tests include:

One-Way ANOVA: Used when you have one categorical independent variable (factor) with three or more levels (groups) and one continuous dependent variable. It tests if there's a significant difference in means across the different levels of the single factor.

Two-Way ANOVA: Used when you have two categorical independent variables and one continuous dependent variable. It examines the main effects of each independent variable and their interaction effect on the dependent variable.

MANOVA (Multivariate Analysis of Variance): Used when you have one or more categorical independent variables and two or more continuous dependent variables. It tests for differences in means across multiple dependent variables simultaneously.

Repeated Measures ANOVA: Used when the same subjects are measured multiple times under different conditions or at different time points.

What is the F-test, and how does it relate to hypothesis testing?

Ans: The F-test is a statistical test that compares the variances of two or more populations, or compares a model with more parameters to a model with fewer parameters to see if the additional parameters significantly improve the model's fit.

Relationship to Hypothesis Testing: In ANOVA, the F-test is the primary test used. The F-statistic is the ratio of the "between-group variability" (variance between group means) to the "within-group variability" (variance within each group).

If the F-statistic is large, it suggests that the variability between groups is much greater than the variability within groups, leading to a rejection of the null hypothesis (that all group means are equal).

The P-value from the F-test helps determine if the observed differences in means are statistically significant.

Practical Questions Part - 1

---

1. Write a Python program to generate a random variable and display its value.

In [3]:

```
import numpy as np

# Generate a single random integer between 1 and 100 (inclusive)
random_integer = np.random.randint(1, 101) #
print(f"Generated Random Integer: {random_integer}")

# Generate a single random float between 0.0 and 1.0
random_float = np.random.rand() #
print(f"Generated Random Float: {random_float}")

# Generate a single random value from a standard normal distribution
random_normal = np.random.randn() #
print(f"Generated Random Value from Standard Normal Distribution:
{random_normal}")
```

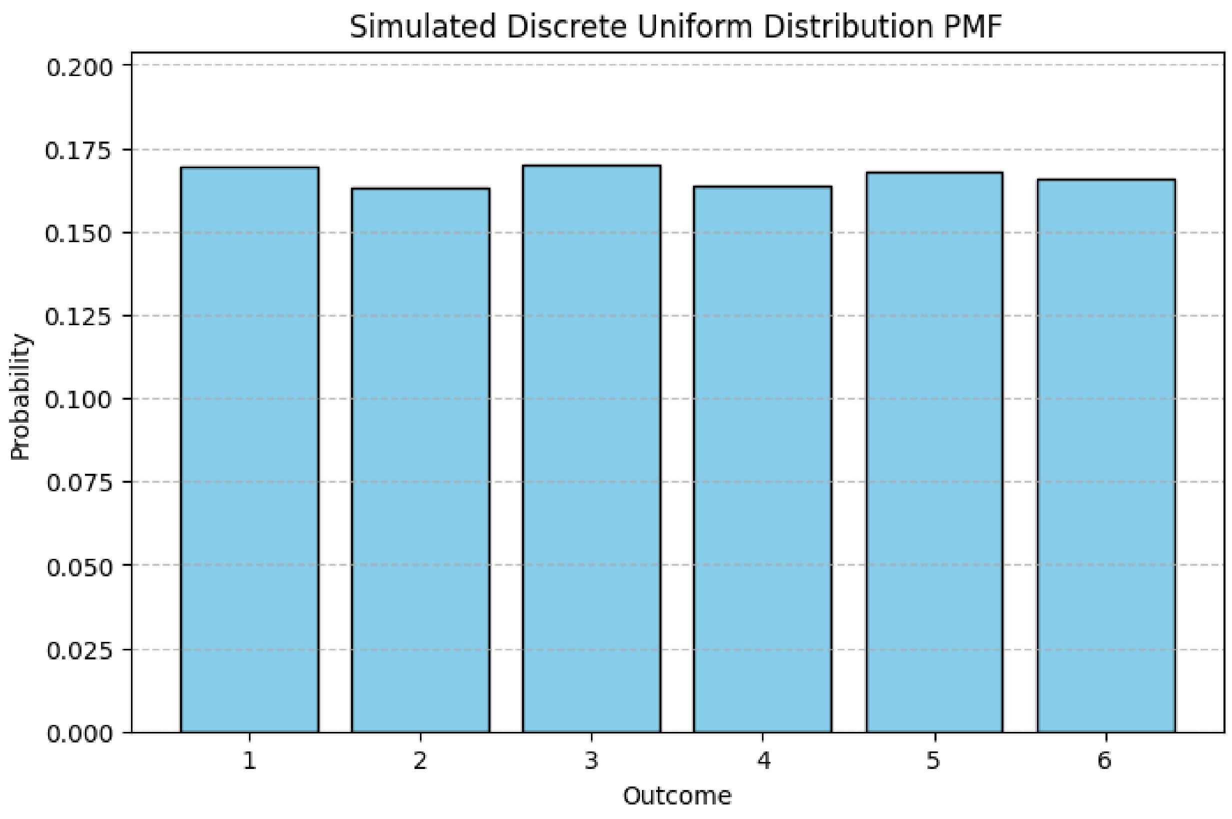
```
Generated Random Integer: 73
Generated Random Float: 0.7287459938199851
Generated Random Value from Standard Normal Distribution: 0.9491565673807842
```

2. Generate a discrete uniform distribution using Python and plot the probability mass function (PMF).

In [4]:

```
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
```

```python
# Define the range of possible outcomes (e.g., rolling a fair die)
outcomes = np.arange(1, 7) # Numbers 1 to 6
num_trials = 10000

# Simulate discrete uniform distribution
# Each outcome has an equal chance
simulated_rolls = np.random.choice(outcomes, size=num_trials, replace=True) #

# Calculate observed probabilities (PMF)
counts = Counter(simulated_rolls)
total_count = sum(counts.values())
pmf_observed = {k: v / total_count for k, v in counts.items()}

# Plotting the PMF
plt.figure(figsize=(8, 5))
plt.bar(list(pmf_observed.keys()), list(pmf_observed.values()),
color='skyblue', edgecolor='black') #
plt.title('Simulated Discrete Uniform Distribution PMF') #
plt.xlabel('Outcome')
plt.ylabel('Probability')
plt.xticks(outcomes)
plt.ylim(0, max(pmf_observed.values()) * 1.2) # Adjust y-axis limit for better
visualization
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

print(f"Observed PMF: {pmf_observed}")
print(f"Expected Probability for each outcome: {1/len(outcomes):.4f}")
```



```
Observed PMF: {np.int64(3): 0.1701, np.int64(4): 0.1638, np.int64(1): 0.1693,
np.int64(6): 0.1658, np.int64(5): 0.1678, np.int64(2): 0.1632}
Expected Probability for each outcome: 0.1667
```

3. Write a Python function to calculate the probability distribution function (PDF) of a Bernoulli distribution.

In [5]:

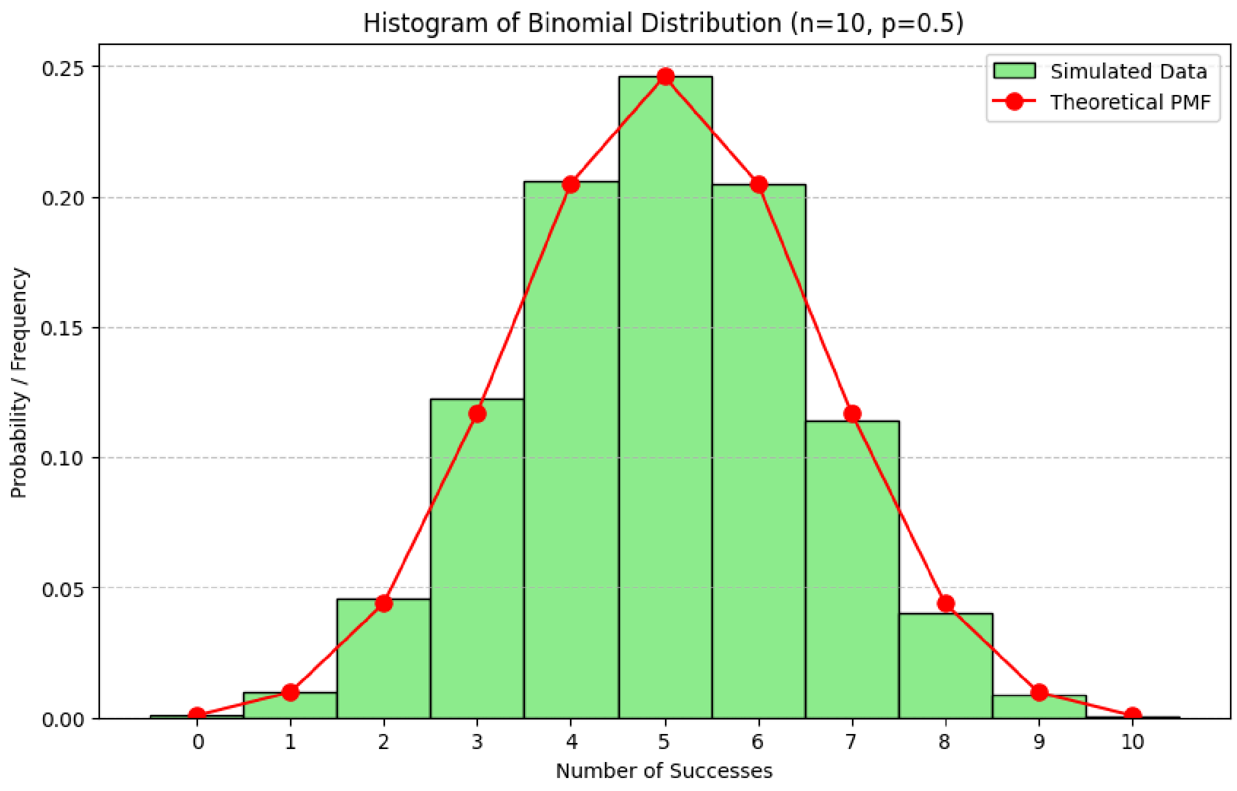
```
def bernoulli_pmf(k, p):
    """
    Calculates the Probability Mass Function (PMF) for a Bernoulli
distribution.
    Args:
        k (int): The outcome (0 for failure, 1 for success).
        p (float): The probability of success (between 0 and 1).
    Returns:
        float: The probability of outcome k.
    """
    if not (0 <= p <= 1):
        raise ValueError("Probability 'p' must be between 0 and 1.")
    if k == 1:
        return p #
    elif k == 0:
        return 1 - p #
    else:
        return 0 # For any other outcome, probability is 0

# Example Usage:
p_success = 0.7
print(f"Probability of success (k=1) with p={p_success}: {bernoulli_pmf(1,
p_success)}") #
print(f"Probability of failure (k=0) with p={p_success}: {bernoulli_pmf(0,
p_success)}") #

p_success = 0.2
print(f"\nProbability of success (k=1) with p={p_success}: {bernoulli_pmf(1,
p_success)}")
print(f"Probability of failure (k=0) with p={p_success}: {bernoulli_pmf(0,
p_success)}")
```

```
Probability of success (k=1) with p=0.7: 0.7
Probability of failure (k=0) with p=0.7: 0.30000000000000004

Probability of success (k=1) with p=0.2: 0.2
Probability of failure (k=0) with p=0.2: 0.8
```

4. Write a Python script to simulate a binomial distribution with n=10 and p=0.5, then plot its histogram.

In [6]:

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

n_trials = 10     # Number of trials
p_success = 0.5   # Probability of success on each trial
num_experiments = 10000 # Number of times we run the 10-trial experiment

# Simulate binomial distribution (number of successes in n_trials)
simulated_data = np.random.binomial(n=n_trials, p=p_success,
size=num_experiments) #

# Plotting the histogram
plt.figure(figsize=(10, 6))
plt.hist(simulated_data, bins=np.arange(n_trials + 2) - 0.5, density=True,
color='lightgreen', edgecolor='black', label='Simulated Data') #
plt.title(f'Histogram of Binomial Distribution (n={n_trials}, p={p_success})')
#
plt.xlabel('Number of Successes')
plt.ylabel('Probability / Frequency')
plt.xticks(np.arange(n_trials + 1))
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Overlay theoretical PMF for comparison
x = np.arange(0, n_trials + 1)
pmf = binom.pmf(x, n_trials, p_success)
plt.plot(x, pmf, 'ro-', markersize=8, label='Theoretical PMF') #
plt.legend()
plt.show()

# Print mean and variance for binomial distribution
mean_binomial = n_trials * p_success
variance_binomial = n_trials * p_success * (1 - p_success)
print(f"Theoretical Mean: {mean_binomial}")
print(f"Theoretical Variance: {variance_binomial}")
print(f"Simulated Mean: {np.mean(simulated_data):.2f}")
print(f"Simulated Variance: {np.var(simulated_data):.2f}")
```

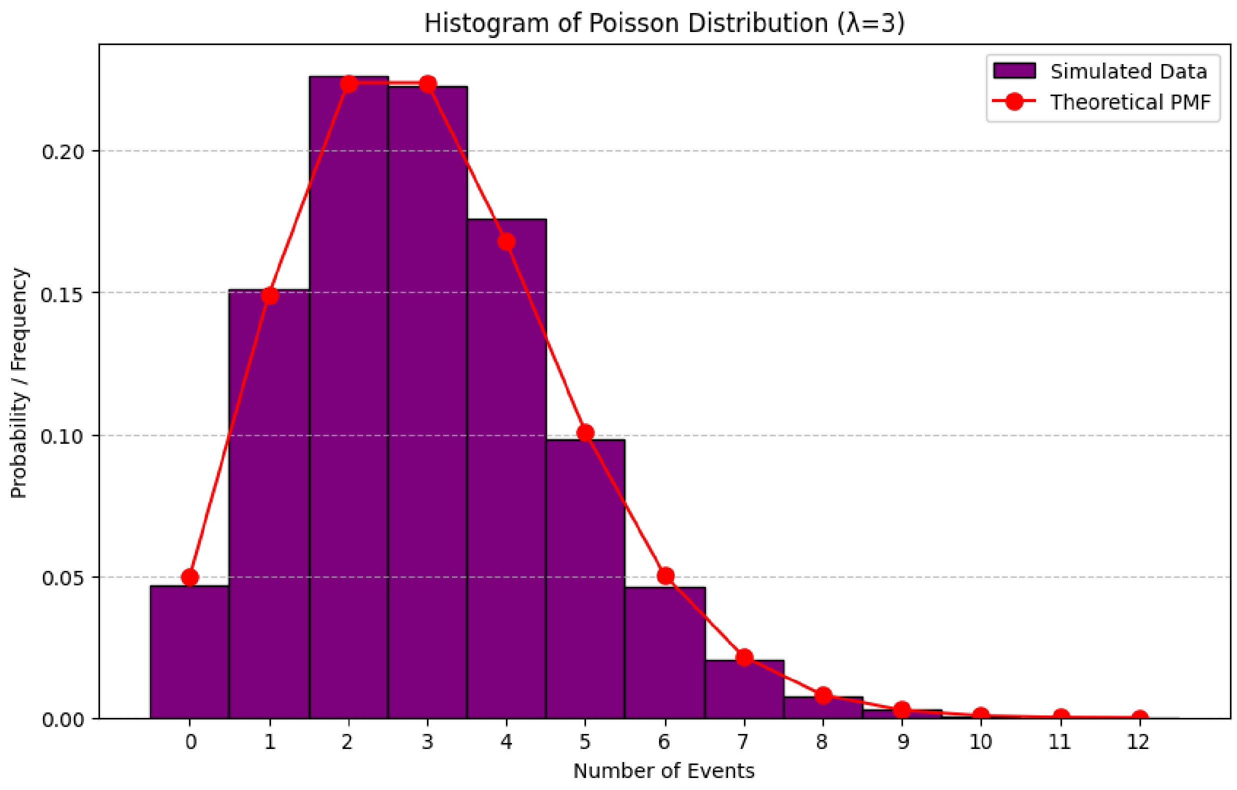


```
Theoretical Mean: 5.0
Theoretical Variance: 2.5
Simulated Mean: 4.96
Simulated Variance: 2.46
```

5. Create a Poisson distribution and visualize it using Python.

In [7]:

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

lambda_param = 3 # Average rate of events (e.g., 3 events per hour)
num_samples = 10000 # Number of times we observe the process

# Simulate Poisson distribution
simulated_data = np.random.poisson(lam=lambda_param, size=num_samples) #

# Plotting the histogram
plt.figure(figsize=(10, 6))
plt.hist(simulated_data, bins=np.arange(np.max(simulated_data) + 2) - 0.5,
density=True, color='purple', edgecolor='black', label='Simulated Data') #
plt.title(f'Histogram of Poisson Distribution (λ={lambda_param})') #
plt.xlabel('Number of Events')
plt.ylabel('Probability / Frequency')
plt.xticks(np.arange(np.max(simulated_data) + 1))
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Overlay theoretical PMF for comparison
x = np.arange(0, np.max(simulated_data) + 1)
pmf = poisson.pmf(x, lambda_param)
plt.plot(x, pmf, 'ro-', markersize=8, label='Theoretical PMF') #
plt.legend()
plt.show()

# Print theoretical mean and variance (for Poisson, mean = variance = lambda)
print(f"Theoretical Mean (λ): {lambda_param}")
print(f"Theoretical Variance (λ): {lambda_param}")
print(f"Simulated Mean: {np.mean(simulated_data):.2f}")
print(f"Simulated Variance: {np.var(simulated_data):.2f}")
```

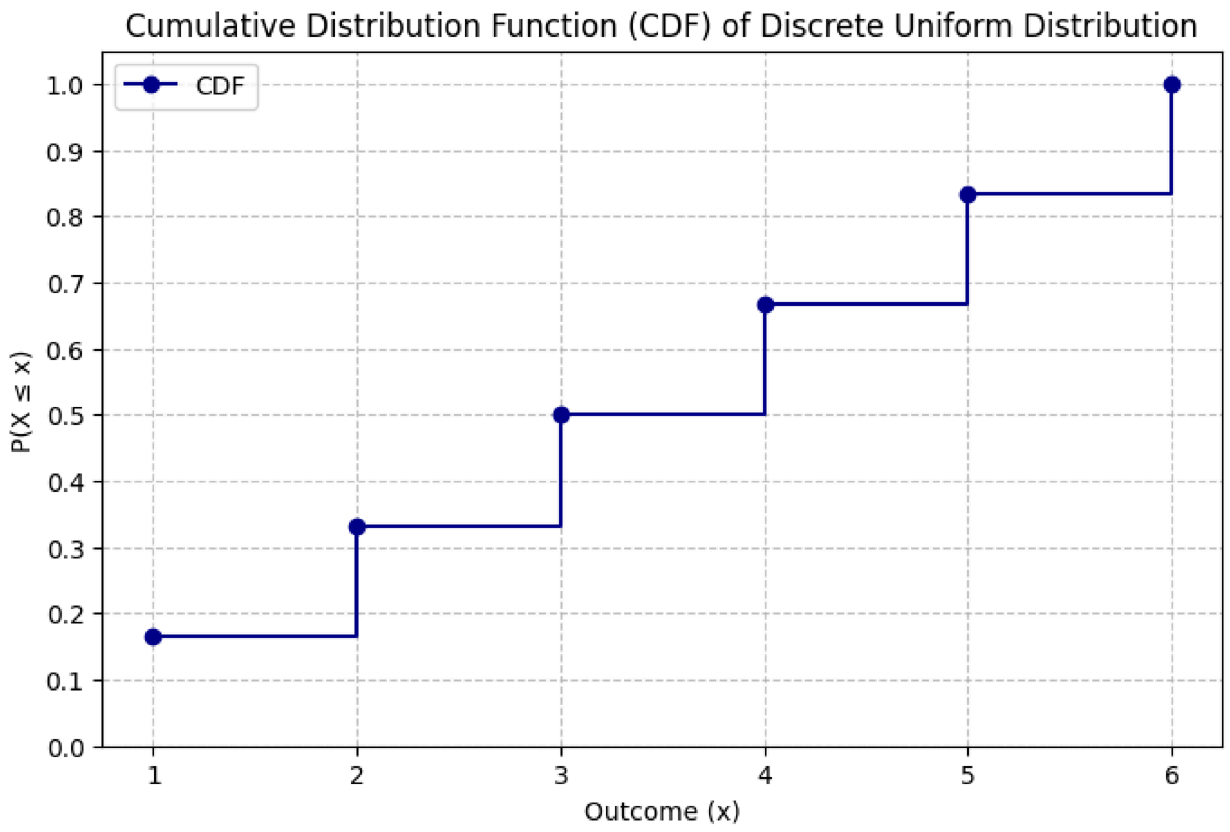


```
Theoretical Mean (λ): 3
Theoretical Variance (λ): 3
Simulated Mean: 2.99
Simulated Variance: 2.90
```

6. Write a Python program to calculate and plot the cumulative distribution function (CDF) of a discrete uniform distribution.

In [8]:

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

# Define the range of possible outcomes (e.g., rolling a fair die)
outcomes = np.arange(1, 7) # Numbers 1 to 6
n_outcomes = len(outcomes)

# Probability of each outcome in a discrete uniform distribution
pmf_value = 1 / n_outcomes

# Calculate CDF
cdf_values = []
cumulative_prob = 0
for i in range(1, n_outcomes + 1):
    cumulative_prob += pmf_value
    cdf_values.append(cumulative_prob)

# Plotting the CDF (step function for discrete distribution)
plt.figure(figsize=(8, 5))
plt.step(outcomes, cdf_values, where='post', color='darkblue', marker='o',
linestyle='-', label='CDF') #
plt.title('Cumulative Distribution Function (CDF) of Discrete Uniform
Distribution') #
plt.xlabel('Outcome (x)')
plt.ylabel('P(X ≤ x)')
plt.xticks(outcomes)
plt.yticks(np.linspace(0, 1, 11))
plt.grid(True, linestyle='--', alpha=0.7)
plt.ylim(0, 1.05)
plt.legend()
plt.show()

print(f"Outcomes: {outcomes}")
print(f"CDF Values: {cdf_values}")
```



```
Outcomes: [1 2 3 4 5 6]
CDF Values: [0.16666666666666666, 0.3333333333333333, 0.5, 0.6666666666666666,
0.8333333333333333, 0.9999999999999999]
```

7. Generate a continuous uniform distribution using NumPy and visualize it.

In [ ]:

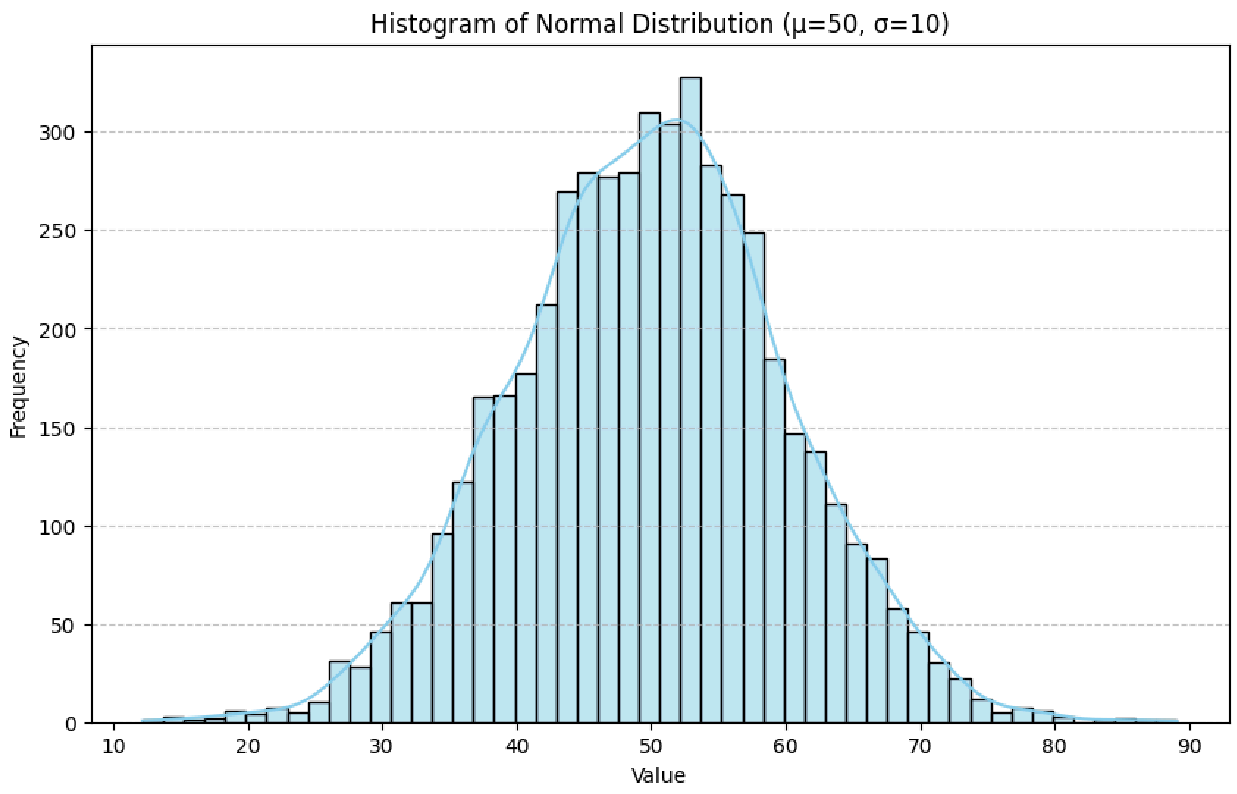
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Define the interval [a, b]
a = 5    # Lower bound
b = 15   # Upper bound
num_samples = 10000

# Generate random numbers from a continuous uniform distribution
uniform_data = np.random.uniform(low=a, high=b, size=num_samples) #

# Visualize with a histogram (approximating the PDF)
plt.figure(figsize=(10, 6))
sns.histplot(uniform_data, bins=50, stat='density', color='teal',
edgecolor='black') #
plt.title(f'Continuous Uniform Distribution (a={a}, b={b})') #
plt.xlabel('Value')
plt.ylabel('Probability Density')
plt.xlim(a - 1, b + 1)
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Plot the theoretical PDF (a flat line)
x_pdf = np.linspace(a, b, 100)
y_pdf = [1 / (b - a)] * len(x_pdf)
plt.plot(x_pdf, y_pdf, 'r--', linewidth=2, label=f'Theoretical PDF (1/({b}-{a})
= {1/(b-a):.2f})') #
plt.legend()
plt.show()

print(f"Mean of simulated data: {np.mean(uniform_data):.2f}")
print(f"Theoretical Mean: {(a + b) / 2}")
```

8. Simulate data from a normal distribution and plot its histogram.

In [9]:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

mean = 50        # Mean of the distribution
std_dev = 10     # Standard deviation of the distribution
num_samples = 5000 # Number of data points to simulate
```

```
# Simulate data from a normal distribution
normal_data = np.random.normal(loc=mean, scale=std_dev, size=num_samples) #

# Plotting the histogram
plt.figure(figsize=(10, 6))
sns.histplot(normal_data, bins=50, kde=True, color='skyblue',
edgecolor='black') #
plt.title(f'Histogram of Normal Distribution (μ={mean}, σ={std_dev})') #
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

print(f"Simulated Data Mean: {np.mean(normal_data):.2f}")
print(f"Simulated Data Standard Deviation: {np.std(normal_data):.2f}")
```



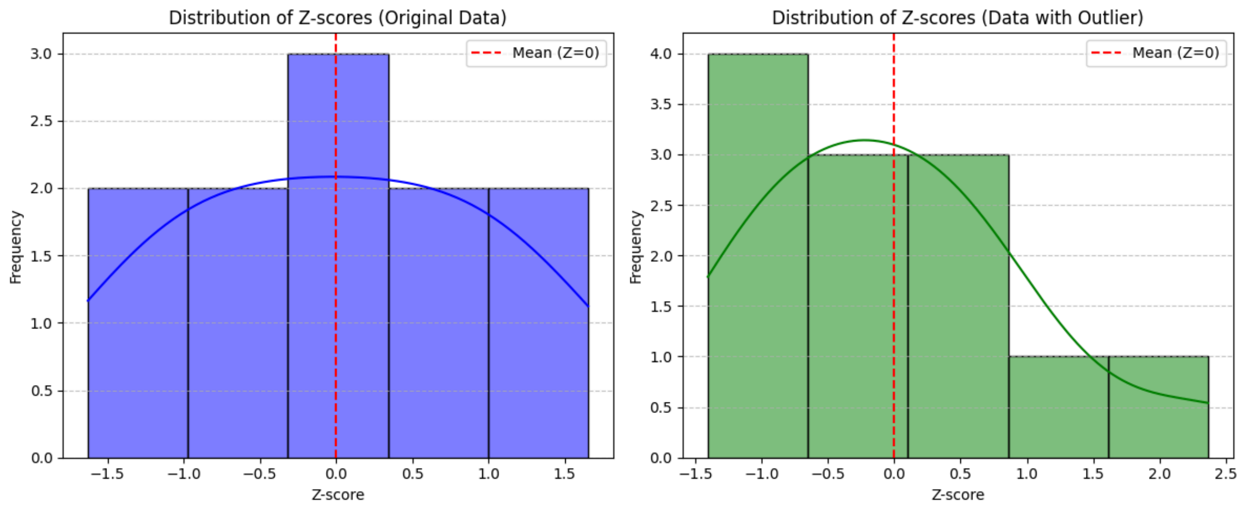
```
Simulated Data Mean: 49.85
Simulated Data Standard Deviation: 10.04
```

9. Write a Python function to calculate Z-scores from a dataset and plot them.

In [10]:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm

def calculate_z_scores(data):
    """
    Calculates the Z-scores for each data point in a dataset.
    Z-score = (x - mean) / standard deviation
    """
    mean = np.mean(data)
    std_dev = np.std(data)
    if std_dev == 0: # Handle case of zero standard deviation
        return np.zeros_like(data)
    z_scores = (data - mean) / std_dev #
    return z_scores

# Generate some sample data (e.g., test scores)
data = np.array([65, 70, 72, 75, 80, 82, 85, 90, 92, 95, 100])
# Add an outlier to see its Z-score
data_with_outlier = np.append(data, 120)

z_scores_original = calculate_z_scores(data) #
z_scores_outlier = calculate_z_scores(data_with_outlier) #

print(f"Original Data: {data}")
print(f"Z-scores for Original Data: {z_scores_original.round(2)}")

print(f"\nData with Outlier: {data_with_outlier}")
print(f"Z-scores for Data with Outlier: {z_scores_outlier.round(2)}")

# Plotting Z-scores
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.histplot(z_scores_original, bins=5, kde=True, color='blue',
edgecolor='black') #
plt.title('Distribution of Z-scores (Original Data)') #
plt.xlabel('Z-score')
plt.ylabel('Frequency')
plt.axvline(0, color='red', linestyle='--', label='Mean (Z=0)')
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.subplot(1, 2, 2)
sns.histplot(z_scores_outlier, bins=5, kde=True, color='green',
edgecolor='black') #
plt.title('Distribution of Z-scores (Data with Outlier)') #
plt.xlabel('Z-score')
plt.ylabel('Frequency')
plt.axvline(0, color='red', linestyle='--', label='Mean (Z=0)')
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()

print("\nSignificance of Z-scores:")
print("- Z-scores standardize data, allowing comparison of observations from
different scales.")
print("- A Z-score tells you how many standard deviations an observation is
from the mean.")
print("- Values with Z-scores typically outside ±2 or ±3 are often considered
outliers.")
```

```
Original Data: [ 65  70  72  75  80  82  85  90  92  95 100]
Z-scores for Original Data: [-1.63 -1.16 -0.97 -0.69 -0.22 -0.03  0.25  0.72
0.9   1.19  1.65]

Data with Outlier: [ 65  70  72  75  80  82  85  90  92  95 100 120]
Z-scores for Data with Outlier: [-1.41 -1.06 -0.93 -0.72 -0.38 -0.24 -0.03
0.31  0.45  0.65  0.99  2.37]
```



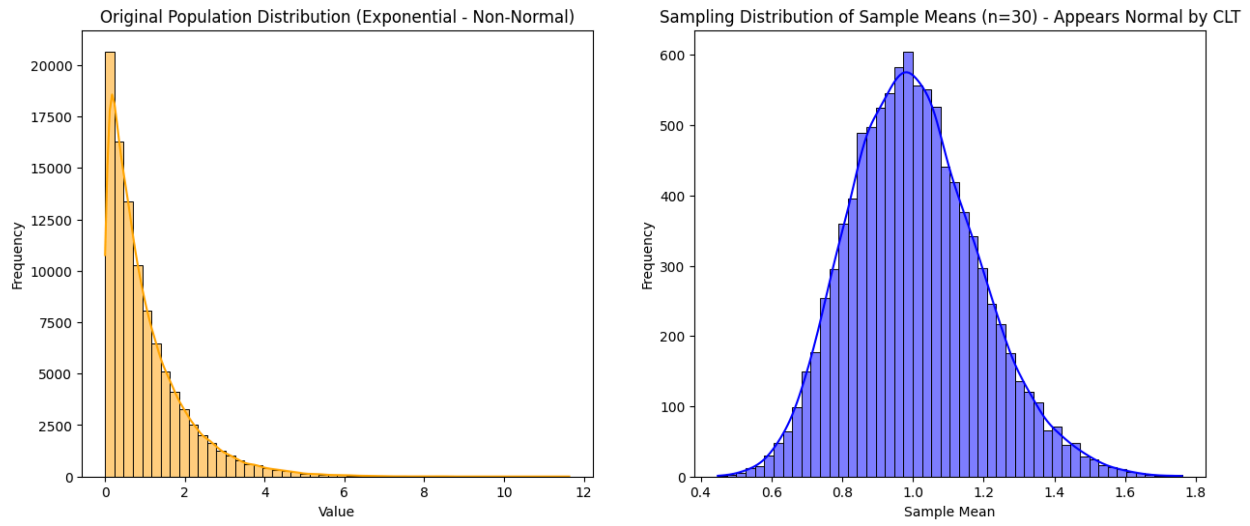
```
Significance of Z-scores:
- Z-scores standardize data, allowing comparison of observations from different
scales.
- A Z-score tells you how many standard deviations an observation is from the
mean.
- Values with Z-scores typically outside ±2 or ±3 are often considered
outliers.
```

10. Implement the Central Limit Theorem (CLT) using Python for a non-normal distribution.

In [11]:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Define a non-normal population distribution (e.g., Exponential
Distribution)
# This distribution is skewed.
lambda_param = 1.0
population_data = np.random.exponential(scale=lambda_param, size=100000)

plt.figure(figsize=(15, 6))

plt.subplot(1, 2, 1)
sns.histplot(population_data, bins=50, kde=True, color='orange',
edgecolor='black') #
plt.title('Original Population Distribution (Exponential - Non-Normal)') #
plt.xlabel('Value')
plt.ylabel('Frequency')

# 2. Take many samples of a certain size from this population
sample_size = 30 # A 'sufficiently large' sample size for CLT
num_samples = 10000 # Number of samples to draw
sample_means = []

for _ in range(num_samples):
    sample = np.random.choice(population_data, size=sample_size, replace=False)
# Draw random sample
    sample_means.append(np.mean(sample)) # Calculate and store sample mean

# 3. Plot the distribution of the sample means
plt.subplot(1, 2, 2)
sns.histplot(sample_means, bins=50, kde=True, color='blue', edgecolor='black')
#
plt.title(f'Sampling Distribution of Sample Means (n={sample_size}) - Appears
Normal by CLT') #
plt.xlabel('Sample Mean')
plt.ylabel('Frequency')
plt.show()

print(f"Mean of original population: {np.mean(population_data):.2f}")
print(f"Mean of sample means: {np.mean(sample_means):.2f} (Should be close to
population mean)")

# Standard Deviation of original population
pop_std = np.std(population_data)
# Theoretical Standard Error of the Mean (SEM) = pop_std / sqrt(sample_size)
theoretical_sem = pop_std / np.sqrt(sample_size)
# Actual Standard Deviation of Sample Means
```

```
actual_std_of_sample_means = np.std(sample_means)

print(f"Theoretical Standard Error of Mean (SEM): {theoretical_sem:.2f}")
print(f"Actual Standard Deviation of Sample Means:
{actual_std_of_sample_means:.2f} (Should be close to Theoretical SEM)")

print("\nCentral Limit Theorem (CLT) Demonstration:")
print("Even though the original population is exponentially distributed
(skewed),")
print(f"the distribution of sample means (with n={sample_size}) is
approximately normal,")
print("and its mean is close to the population mean.")
```



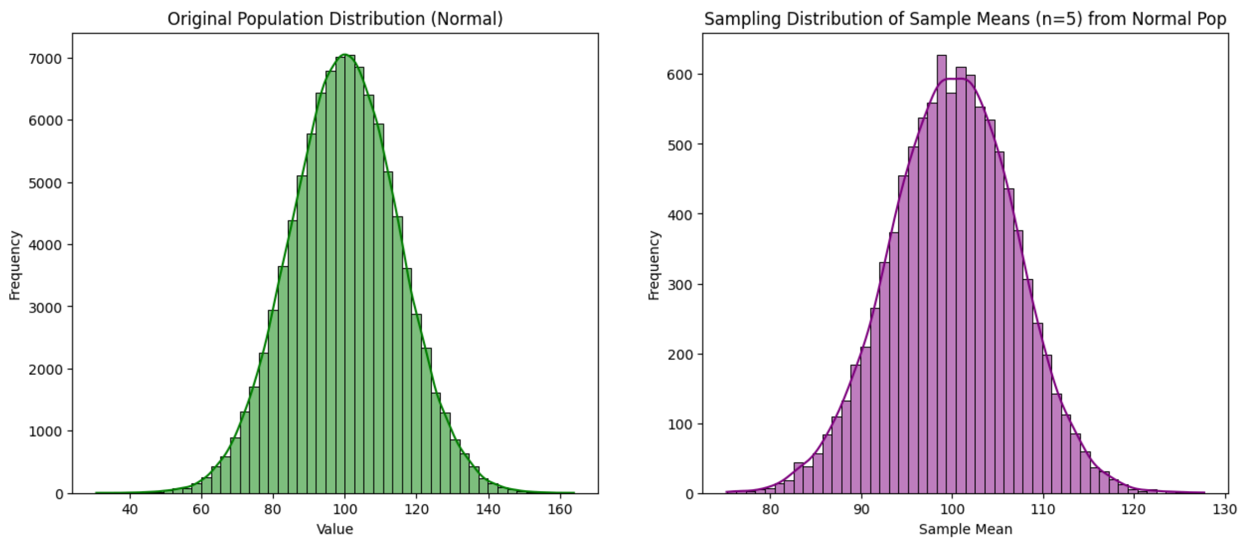
```
Mean of original population: 1.00
Mean of sample means: 1.00 (Should be close to population mean)
Theoretical Standard Error of Mean (SEM): 0.18
Actual Standard Deviation of Sample Means: 0.18 (Should be close to Theoretical
SEM)

Central Limit Theorem (CLT) Demonstration:
Even though the original population is exponentially distributed (skewed),
the distribution of sample means (with n=30) is approximately normal,
and its mean is close to the population mean.
```

11. Simulate multiple samples from a normal distribution and verify the Central Limit Theorem.

In [12]:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# 1. Define a normal population distribution
pop_mean = 100
pop_std_dev = 15
population_data = np.random.normal(loc=pop_mean, scale=pop_std_dev,
size=100000)

plt.figure(figsize=(15, 6))

plt.subplot(1, 2, 1)
sns.histplot(population_data, bins=50, kde=True, color='green',
edgecolor='black') #
plt.title('Original Population Distribution (Normal)') #
plt.xlabel('Value')
plt.ylabel('Frequency')

# 2. Take many samples of a certain size from this normal population
sample_size_clt = 5 # Small sample size
num_samples_clt = 10000 # Number of samples
sample_means_clt = []

for _ in range(num_samples_clt):
    sample = np.random.choice(population_data, size=sample_size_clt,
replace=False)
    sample_means_clt.append(np.mean(sample))

# 3. Plot the distribution of the sample means
plt.subplot(1, 2, 2)
sns.histplot(sample_means_clt, bins=50, kde=True, color='purple',
edgecolor='black') #
plt.title(f'Sampling Distribution of Sample Means (n={sample_size_clt}) from
Normal Pop') #
plt.xlabel('Sample Mean')
plt.ylabel('Frequency')
plt.show()

print(f"Mean of original normal population: {np.mean(population_data):.2f}")
print(f"Mean of sample means: {np.mean(sample_means_clt):.2f} (Should be close
to population mean)")

pop_std_clt = np.std(population_data)
theoretical_sem_clt = pop_std_clt / np.sqrt(sample_size_clt)
actual_std_of_sample_means_clt = np.std(sample_means_clt)

print(f"Theoretical Standard Error of Mean (SEM): {theoretical_sem_clt:.2f}")
print(f"Actual Standard Deviation of Sample Means:
{actual_std_of_sample_means_clt:.2f}")

print("\nCLT Verification with Normal Population:")
print("When the population is already normal, the sampling distribution of the means is normal")
print("regardless of the sample size. The CLT still applies, confirming this behavior.")
print("The mean of sample means converges to the population mean, and the standard deviation of sample means (SEM)")
print("is approximately the population standard deviation divided by the square root of the sample size.")
```



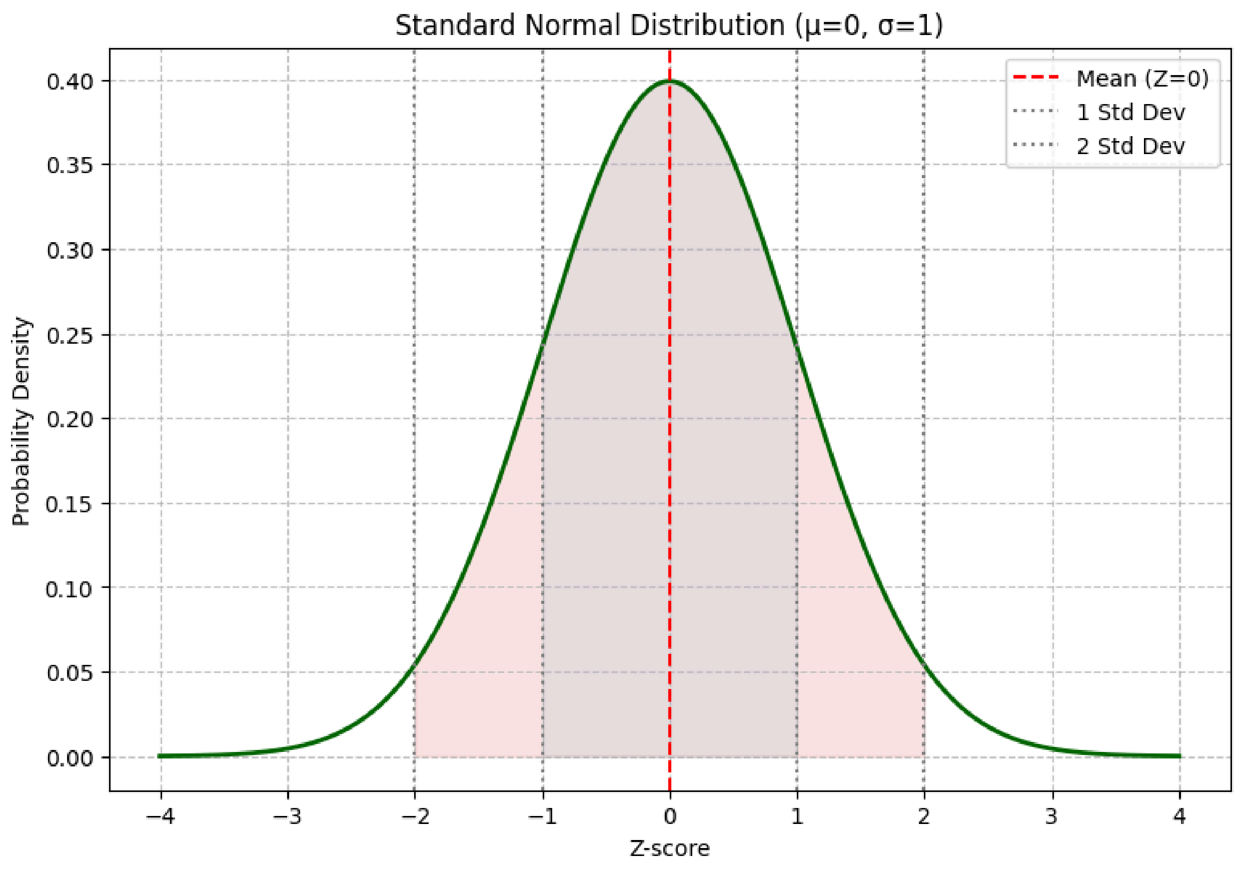
```
Mean of original normal population: 100.01
Mean of sample means: 100.09 (Should be close to population mean)
Theoretical Standard Error of Mean (SEM): 6.72
Actual Standard Deviation of Sample Means: 6.81

CLT Verification with Normal Population:
When the population is already normal, the sampling distribution of the means
is normal
regardless of the sample size. The CLT still applies, confirming this behavior.
The mean of sample means converges to the population mean, and the standard
deviation of sample means (SEM)
is approximately the population standard deviation divided by the square root
of the sample size.
```

12. Write a Python function to calculate and plot the standard normal distribution (mean=0, std=1).

In [13]:

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

```
def plot_standard_normal_distribution():
    """
    Calculates and plots the Probability Density Function (PDF)
    of the standard normal distribution (mean=0, std=1).
    """
    # Generate x values (a range around the mean for visualization)
    x = np.linspace(-4, 4, 1000) # From -4 to +4 standard deviations

    # Calculate the PDF values for the standard normal distribution
    # For standard normal, mean=0, std_dev=1
    pdf_values = norm.pdf(x, loc=0, scale=1) #

    # Plotting the PDF
    plt.figure(figsize=(9, 6))
    plt.plot(x, pdf_values, color='darkgreen', linewidth=2) #
    plt.title('Standard Normal Distribution (μ=0, σ=1)') #
    plt.xlabel('Z-score')
    plt.ylabel('Probability Density')
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.axvline(0, color='red', linestyle='--', label='Mean (Z=0)')
    plt.axvline(1, color='gray', linestyle=':', label='1 Std Dev')
    plt.axvline(-1, color='gray', linestyle=':')
    plt.axvline(2, color='gray', linestyle=':', label='2 Std Dev')
    plt.axvline(-2, color='gray', linestyle=':')
    plt.legend()
    plt.fill_between(x, 0, pdf_values, where=(x >= -1) & (x <= 1),
color='lightblue', alpha=0.3, label='68%')
    plt.fill_between(x, 0, pdf_values, where=(x >= -2) & (x <= 2),
color='lightcoral', alpha=0.2, label='95%')
    plt.show()

# Call the function to plot
plot_standard_normal_distribution()
```



13. Generate random variables and calculate their corresponding probabilities using the binomial distribution.

In [16]:

```
import numpy as np
from scipy.stats import binom

n_trials = 20    # Number of trials (e.g., 20 coin flips, 20 items inspected)
p_success = 0.6  # Probability of success on each trial (e.g., prob of heads,
prob of item being good)

# Generate a random number of successes from this binomial distribution
# This simulates one "experiment" of n_trials
random_successes = np.random.binomial(n=n_trials, p=p_success, size=1)[0] #
print(f"Randomly generated number of successes in {n_trials} trials:
{random_successes}")

# Calculate the probability of observing exactly this many successes
prob_exact_successes = binom.pmf(k=random_successes, n=n_trials, p=p_success) #
print(f"Probability of getting exactly {random_successes} successes:
{prob_exact_successes:.4f}")
```

```
# Calculate the probability of getting at most this many successes (CDF)
prob_at_most_successes = binom.cdf(k=random_successes, n=n_trials, p=p_success)
#
print(f"Probability of getting at most {random_successes} successes:
{prob_at_most_successes:.4f}")

# Calculate the probability of getting at least this many successes (1 -
CDF(k-1))
prob_at_least_successes = 1 - binom.cdf(k=random_successes - 1, n=n_trials,
p=p_success) #
print(f"Probability of getting at least {random_successes} successes:
{prob_at_least_successes:.4f}")

# Example: Probability of exactly 5 successes
k_specific = 5
prob_k_specific = binom.pmf(k=k_specific, n=n_trials, p=p_success)
print(f"\nProbability of getting exactly {k_specific} successes:
{prob_k_specific:.4f}")
```

```
Randomly generated number of successes in 20 trials: 9
Probability of getting exactly 9 successes: 0.0710
Probability of getting at most 9 successes: 0.1275
Probability of getting at least 9 successes: 0.9435

Probability of getting exactly 5 successes: 0.0013
```

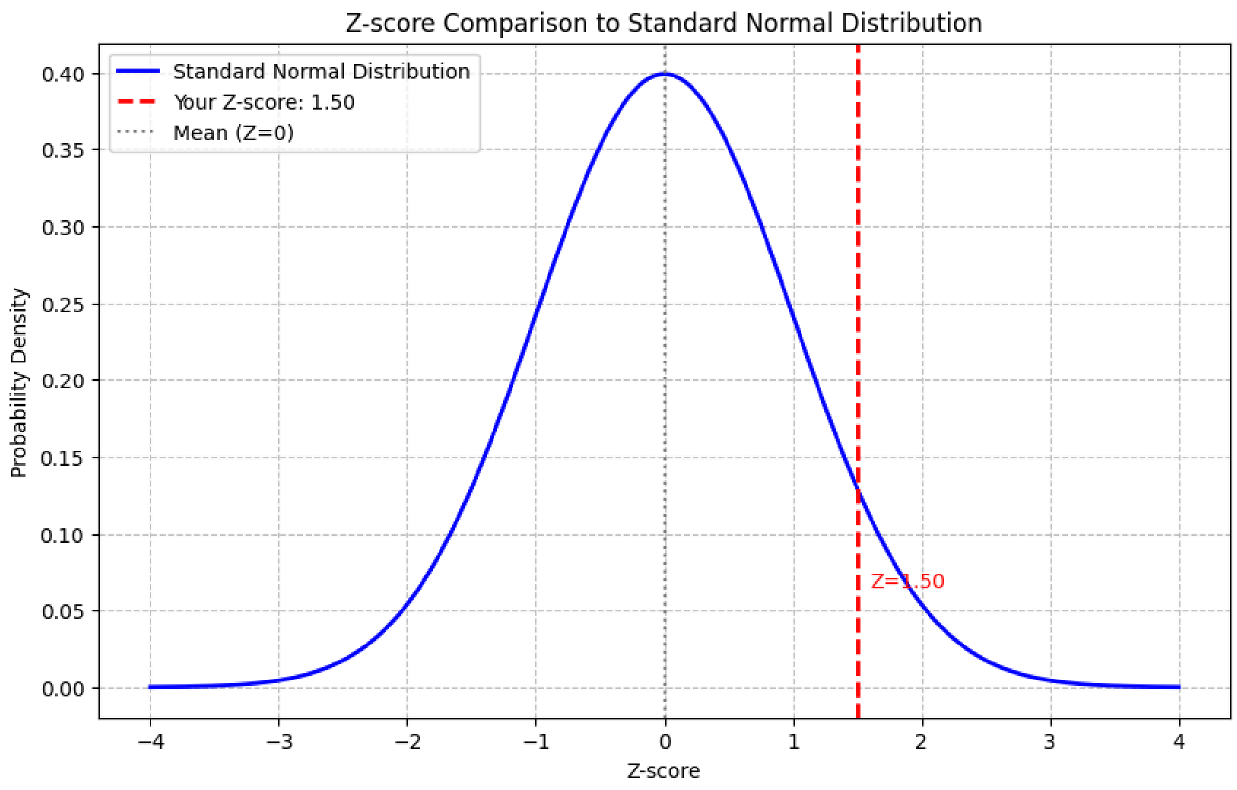
14. Write a Python program to calculate the Z-score for a given data point and compare it to a standard normal distribution.

In [17]:

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

# Given data point
data_point = 85

# Population parameters (assume known for Z-score calculation)
population_mean = 70
population_std_dev = 10

# Calculate the Z-score
z_score = (data_point - population_mean) / population_std_dev #
print(f"Data Point: {data_point}")
print(f"Population Mean: {population_mean}")
print(f"Population Standard Deviation: {population_std_dev}")
print(f"Calculated Z-score: {z_score:.2f}")
```

```
# Compare to a standard normal distribution
# Generate x values for the standard normal distribution
x = np.linspace(-4, 4, 500)
pdf_values = norm.pdf(x, loc=0, scale=1)

plt.figure(figsize=(10, 6))
plt.plot(x, pdf_values, color='blue', linewidth=2, label='Standard Normal
Distribution') #
plt.title('Z-score Comparison to Standard Normal Distribution') #
plt.xlabel('Z-score')
plt.ylabel('Probability Density')
plt.grid(True, linestyle='--', alpha=0.7)

# Mark the calculated Z-score on the plot
plt.axvline(z_score, color='red', linestyle='--', linewidth=2, label=f'Your
Z-score: {z_score:.2f}') #
plt.text(z_score + 0.1, norm.pdf(z_score, 0, 1) * 0.5, f'Z={z_score:.2f}',
color='red') #
plt.axvline(0, color='gray', linestyle=':', label='Mean (Z=0)')
plt.legend()
plt.show()

# Interpretation
print(f"\nInterpretation:")
print(f"- Your data point of {data_point} is {z_score:.2f} standard deviations
above the population mean of {population_mean}.")
if z_score > 2 or z_score < -2:
    print("- This Z-score is relatively high/low, suggesting the data point is
somewhat unusual or an outlier.")
elif z_score > 1 or z_score < -1:
    print("- This Z-score indicates the data point is reasonably far from the
mean but still within typical range.")
else:
    print("- This Z-score indicates the data point is close to the mean.")
```

```
Data Point: 85
Population Mean: 70
Population Standard Deviation: 10
Calculated Z-score: 1.50
```



```
Interpretation:
- Your data point of 85 is 1.50 standard deviations above the population mean
of 70.
- This Z-score indicates the data point is reasonably far from the mean but
still within typical range.
```

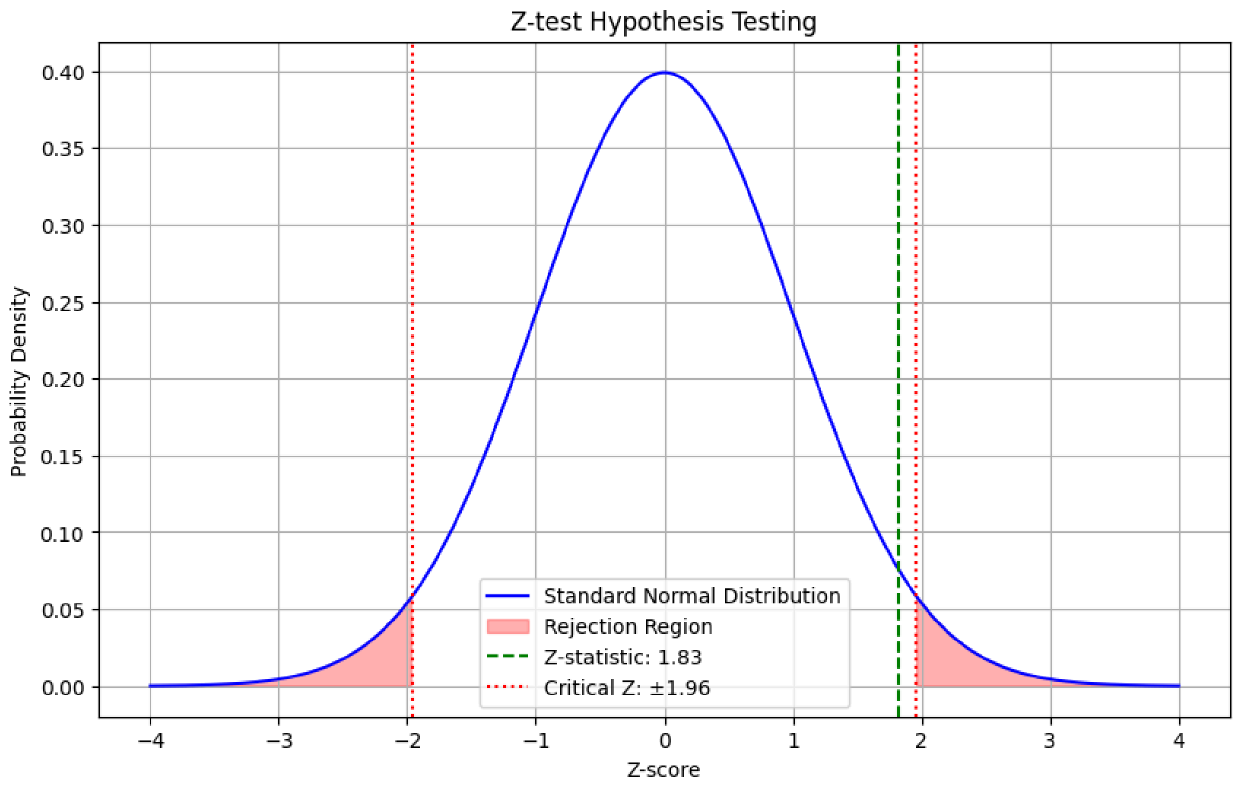
15. Implement hypothesis testing using Z-statistics for a sample dataset.

In [30]:

```python
import numpy as np
from scipy.stats import norm

# Scenario: A school claims its students' average IQ is 100 with a standard
deviation of 15.
# We take a sample of 30 students and find their average IQ is 105.
# Is this sample mean significantly different from 100?

# 1. Define Hypotheses
# Null Hypothesis (H0): The true mean IQ of the school's students is 100 (μ =
100).
# Alternative Hypothesis (H1): The true mean IQ of the school's students is
not 100 (μ ≠ 100).
# This is a two-tailed test.
# 2. Set Significance Level (alpha)
alpha = 0.05 # Commonly used 5% significance level

# 3. Collect Sample Data
sample_mean = 105      # Sample mean IQ
population_std_dev = 15 # Known population standard deviation (assumption for
Z-test)
sample_size = 30       # Sample size (n >= 30, so Z-test is appropriate)
hypothesized_mean = 100 # Hypothesized population mean under H0

# 4. Calculate the Test Statistic (Z-score)
standard_error = population_std_dev / np.sqrt(sample_size)
z_statistic = (sample_mean - hypothesized_mean) / standard_error #
print(f"Sample Mean: {sample_mean}")
print(f"Hypothesized Mean: {hypothesized_mean}")
print(f"Population Standard Deviation: {population_std_dev}")
print(f"Sample Size: {sample_size}")
print(f"Calculated Z-statistic: {z_statistic:.3f}")

# 5. Determine the P-value
# For a two-tailed test, we find the area in both tails
p_value = 2 * (1 - norm.cdf(abs(z_statistic))) #
print(f"P-value: {p_value:.3f}")

# 6. Make a Decision and Interpret
print(f"Significance Level (alpha): {alpha}")
if p_value < alpha:
    print(f"P-value ({p_value:.3f}) < alpha ({alpha}), so we REJECT the Null
Hypothesis.")
    print("Conclusion: There is sufficient evidence to conclude that the true
mean IQ of the school's students is significantly different from 100.")
else:
    print(f"P-value ({p_value:.3f}) > alpha ({alpha}), so we FAIL TO REJECT the
Null Hypothesis.")
    print("Conclusion: There is NOT enough evidence to conclude that the true
mean IQ of the school's students is significantly different from 100.")

# Visualize the decision (optional but good for understanding)
import matplotlib.pyplot as plt
x = np.linspace(-4, 4, 1000)
pdf_values = norm.pdf(x, 0, 1)
plt.figure(figsize=(10, 6))
plt.plot(x, pdf_values, color='blue', label='Standard Normal Distribution')
plt.fill_between(x, 0, pdf_values, where=(x <= -norm.ppf(1 - alpha/2)) | (x >=
norm.ppf(1 - alpha/2)),
                 color='red', alpha=0.3, label='Rejection Region')
plt.axvline(z_statistic, color='green', linestyle='--', label=f'Z-statistic:
{z_statistic:.2f}')
```

```
plt.axvline(-norm.ppf(1 - alpha/2), color='red', linestyle=':',
label=f'Critical Z: ±{norm.ppf(1 - alpha/2):.2f}')
plt.axvline(norm.ppf(1 - alpha/2), color='red', linestyle=':')
plt.title('Z-test Hypothesis Testing')
plt.xlabel('Z-score')
plt.ylabel('Probability Density')
plt.legend()
plt.grid(True)
plt.show()
```

```
Sample Mean: 105
Hypothesized Mean: 100
Population Standard Deviation: 15
Sample Size: 30
Calculated Z-statistic: 1.826
P-value: 0.068
Significance Level (alpha): 0.05
P-value (0.068) > alpha (0.05), so we FAIL TO REJECT the Null Hypothesis.
Conclusion: There is NOT enough evidence to conclude that the true mean IQ of
the school's students is significantly different from 100.
```



16. Create a confidence interval for a dataset using Python and interpret the result.

In [19]:

```python
import numpy as np
from scipy.stats import norm, t

# Scenario: We have a sample of 40 exam scores.
# We want to estimate the true average exam score for all students.

# Sample Data
np.random.seed(42) # for reproducibility
sample_scores = np.random.normal(loc=75, scale=8, size=40) # Simulate 40 scores

sample_mean = np.mean(sample_scores)
sample_std = np.std(sample_scores, ddof=1) # Use ddof=1 for sample standard
deviation
sample_size = len(sample_scores)

print(f"Sample Mean: {sample_mean:.2f}")
print(f"Sample Standard Deviation: {sample_std:.2f}")
print(f"Sample Size: {sample_size}")

# Choose a Confidence Level
confidence_level = 0.95 # 95% Confidence Interval
alpha = 1 - confidence_level

# Determine Critical Value
# Since sample size is >= 30, we can use Z-distribution (norm.ppf)
# If sample_size < 30 and population std dev unknown, use t-distribution
(t.ppf)
# critical_value = norm.ppf(1 - alpha/2) # For Z-distribution
# print(f"Z-critical value for {confidence_level*100}% CI:
{critical_value:.2f}")

# Using t-distribution (safer for unknown population std dev, even with large
n)
degrees_freedom = sample_size - 1
critical_value = t.ppf(1 - alpha/2, df=degrees_freedom) #
print(f"T-critical value for {confidence_level*100}% CI with {degrees_freedom}
df: {critical_value:.2f}")

# Calculate Margin of Error
standard_error = sample_std / np.sqrt(sample_size)
margin_of_error = critical_value * standard_error #
print(f"Margin of Error: {margin_of_error:.2f}")

# Calculate Confidence Interval
lower_bound = sample_mean - margin_of_error #
upper_bound = sample_mean + margin_of_error #

print(f"\n{confidence_level*100}% Confidence Interval: ({lower_bound:.2f},
{upper_bound:.2f})") #
```

```
# Interpretation
print("\nInterpretation:")
print(f"We are {confidence_level*100}% confident that the true population mean
exam score")
print(f"lies between {lower_bound:.2f} and {upper_bound:.2f}.")
print("This means if we were to repeat this sampling process many times,")
print(f"{confidence_level*100}% of the confidence intervals constructed would
contain the true population mean.")
```

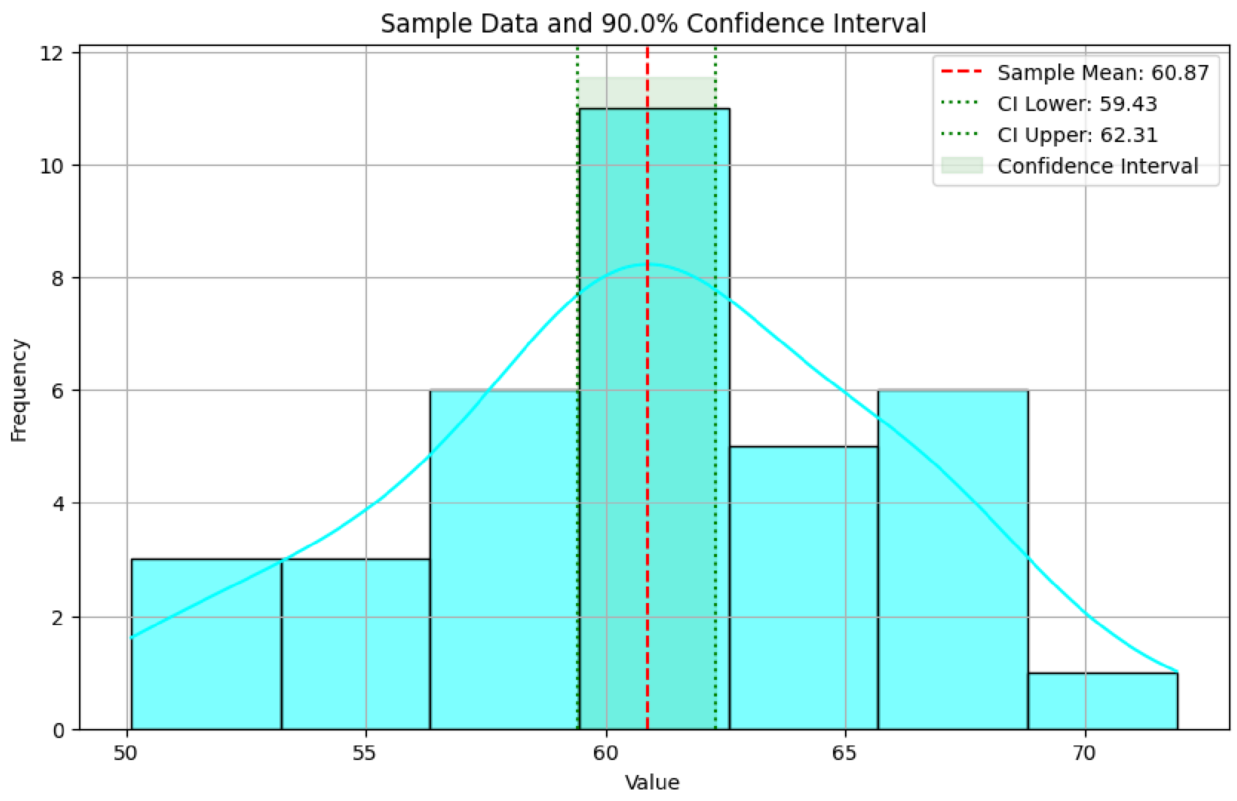
```
Sample Mean: 73.25
Sample Standard Deviation: 7.62
Sample Size: 40
T-critical value for 95.0% CI with 39 df: 2.02
Margin of Error: 2.44

95.0% Confidence Interval: (70.81, 75.69)

Interpretation:
We are 95.0% confident that the true population mean exam score
lies between 70.81 and 75.69.
This means if we were to repeat this sampling process many times,
95.0% of the confidence intervals constructed would contain the true population
mean.
```

17. Generate data from a normal distribution, then calculate and interpret the confidence interval for its mean.

In [20]:

```
import numpy as np
from scipy.stats import t # Using t-distribution as population std dev is
unknown for the *sample*
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Generate data from a normal distribution
population_mean = 60
population_std = 5
sample_size = 35 # Choose a reasonable sample size

np.random.seed(10) # for reproducibility
data_from_normal_dist = np.random.normal(loc=population_mean,
scale=population_std, size=sample_size) #

print(f"Generated sample from normal distribution (first 5):
{data_from_normal_dist[:5].round(2)}")
print(f"Actual Sample Mean: {np.mean(data_from_normal_dist):.2f}")
print(f"Actual Sample Standard Deviation: {np.std(data_from_normal_dist,
ddof=1):.2f}")
print(f"Sample Size: {len(data_from_normal_dist)}")

# 2. Calculate Confidence Interval for its Mean
sample_mean_gen = np.mean(data_from_normal_dist)
sample_std_gen = np.std(data_from_normal_dist, ddof=1)
n_gen = len(data_from_normal_dist)

confidence_level_gen = 0.90 # Let's use 90% CI for this example
alpha_gen = 1 - confidence_level_gen

# Critical value from t-distribution (since population std is unknown for the
sample)
degrees_freedom_gen = n_gen - 1
critical_value_gen = t.ppf(1 - alpha_gen/2, df=degrees_freedom_gen) #

standard_error_gen = sample_std_gen / np.sqrt(n_gen)
margin_of_error_gen = critical_value_gen * standard_error_gen #

lower_bound_gen = sample_mean_gen - margin_of_error_gen #
upper_bound_gen = sample_mean_gen + margin_of_error_gen #

print(f"\n{confidence_level_gen*100}% Confidence Interval for Mean:
({lower_bound_gen:.2f}, {upper_bound_gen:.2f})") #

# 3. Interpretation
print("\nInterpretation:")
print(f"We are {confidence_level_gen*100}% confident that the true population
mean")
print(f"from which this data was drawn lies between {lower_bound_gen:.2f} and
{upper_bound_gen:.2f}.")
print(f"(Note: The actual population mean was {population_mean}, which falls
within this interval.)")

# Optional: Visualize the sample data and the confidence interval
plt.figure(figsize=(10, 6))
sns.histplot(data_from_normal_dist, kde=True, color='cyan', edgecolor='black',
bins=7)
plt.axvline(sample_mean_gen, color='red', linestyle='--', label=f'Sample Mean:
{sample_mean_gen:.2f}')
plt.axvline(lower_bound_gen, color='green', linestyle=':', label=f'CI Lower:
{lower_bound_gen:.2f}')
plt.axvline(upper_bound_gen, color='green', linestyle=':', label=f'CI Upper:
{upper_bound_gen:.2f}')
plt.fill_betweenx([0, plt.gca().get_ylim()[1]], lower_bound_gen,
upper_bound_gen, color='green', alpha=0.1, label='Confidence Interval')
plt.title(f'Sample Data and {confidence_level_gen*100}% Confidence Interval')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
plt.grid(True)
plt.show()
```

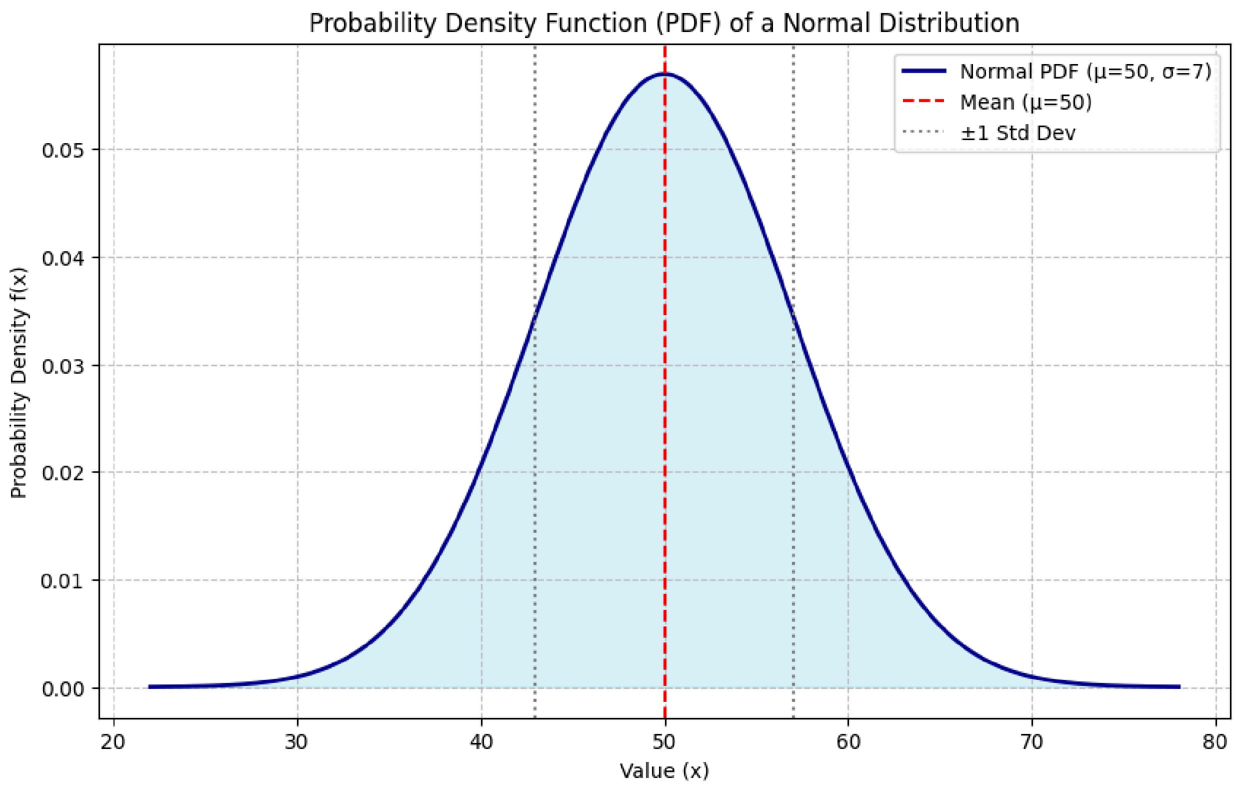
```
Generated sample from normal distribution (first 5): [66.66 63.58 52.27 59.96
63.11]
Actual Sample Mean: 60.87
Actual Sample Standard Deviation: 5.05
Sample Size: 35

90.0% Confidence Interval for Mean: (59.43, 62.31)

Interpretation:
We are 90.0% confident that the true population mean
from which this data was drawn lies between 59.43 and 62.31.
(Note: The actual population mean was 60, which falls within this interval.)
```



18. Write a Python script to calculate and visualize the probability density function (PDF) of a normal distribution.

In [21]:

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

# Define parameters for the normal distribution
mean = 50        # Mean (μ)
std_dev = 7      # Standard Deviation (σ)

# Generate x values (range of data for the plot)
# Typically 3-4 standard deviations around the mean to cover most of the
distribution
x = np.linspace(mean - 4 * std_dev, mean + 4 * std_dev, 1000)

# Calculate the PDF values for each x
pdf_values = norm.pdf(x, loc=mean, scale=std_dev) #

# Plotting the PDF
plt.figure(figsize=(10, 6))
plt.plot(x, pdf_values, color='darkblue', linewidth=2, label=f'Normal PDF
(μ={mean}, σ={std_dev})') #
plt.title('Probability Density Function (PDF) of a Normal Distribution') #
plt.xlabel('Value (x)')
plt.ylabel('Probability Density f(x)')
plt.grid(True, linestyle='--', alpha=0.7)
plt.axvline(mean, color='red', linestyle='--', label=f'Mean (μ={mean})')
plt.axvline(mean + std_dev, color='gray', linestyle=':', label='±1 Std Dev')
plt.axvline(mean - std_dev, color='gray', linestyle=':')
plt.legend()
plt.fill_between(x, 0, pdf_values, color='skyblue', alpha=0.3) # Shade the area
under the curve
plt.show()

print(f"Mean: {mean}")
print(f"Standard Deviation: {std_dev}")
print(f"Peak (max density) at x = {x[np.argmax(pdf_values)]:.2f}")
```



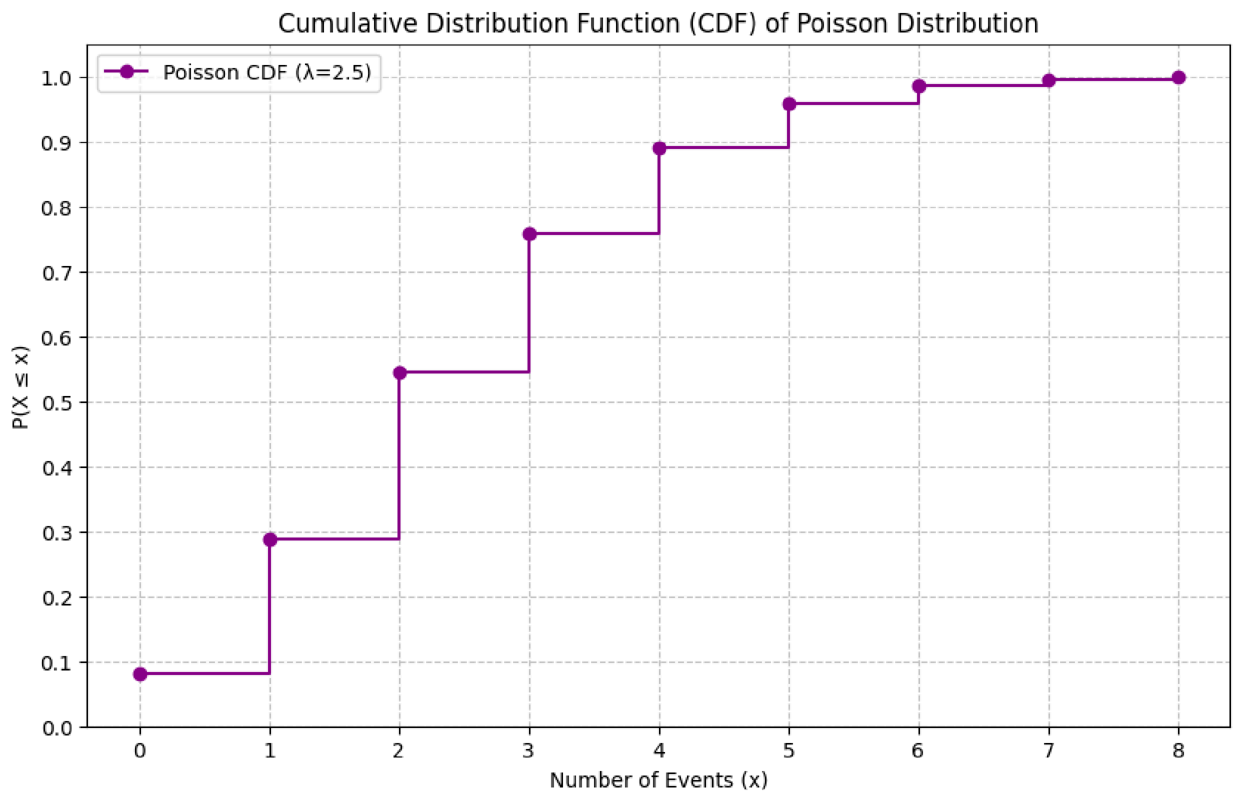
```
Mean: 50
Standard Deviation: 7
Peak (max density) at x = 49.97
```

19. Use Python to calculate and interpret the cumulative distribution function (CDF) of a Poisson distribution.

In [22]:

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

# Define lambda (average rate of events) for the Poisson distribution
lambda_param = 2.5

# Generate x values (number of events)
# Consider a range up to a few standard deviations beyond lambda
x = np.arange(0, int(lambda_param + 4 * np.sqrt(lambda_param)) + 1) # Ensure we
cover enough range

# Calculate the CDF values for each x
cdf_values = poisson.cdf(x, mu=lambda_param) #

# Plotting the CDF (step function for discrete distribution)
```

```
plt.figure(figsize=(10, 6))
plt.step(x, cdf_values, where='post', color='darkmagenta', marker='o',
linestyle='-', label=f'Poisson CDF (λ={lambda_param})') #
plt.title('Cumulative Distribution Function (CDF) of Poisson Distribution') #
plt.xlabel('Number of Events (x)')
plt.ylabel('P(X ≤ x)')
plt.xticks(x)
plt.yticks(np.linspace(0, 1, 11))
plt.grid(True, linestyle='--', alpha=0.7)
plt.ylim(0, 1.05)
plt.legend()
plt.show()

print(f"Lambda (average rate): {lambda_param}")
print(f"X values (number of events): {x}")
print(f"CDF Values (P(X <= x)): {cdf_values.round(4)}")

# Interpretation Examples:
print("\nInterpretation Examples:")
# Probability of 0 events
prob_0 = poisson.cdf(0, mu=lambda_param)
print(f"- Probability of observing 0 events: P(X <= 0) = {prob_0:.4f}")

# Probability of at most 2 events
prob_at_most_2 = poisson.cdf(2, mu=lambda_param)
print(f"- Probability of observing at most 2 events: P(X <= 2) =
{prob_at_most_2:.4f}")

# Probability of exactly 3 events (CDF(3) - CDF(2))
prob_exact_3 = poisson.pmf(3, mu=lambda_param)
print(f"- Probability of observing exactly 3 events: P(X = 3) =
{prob_exact_3:.4f}")
# Or using CDF: prob_exact_3 = poisson.cdf(3, mu=lambda_param) -
poisson.cdf(2, mu=lambda_param)

# Probability of at least 5 events (1 - CDF(4))
prob_at_least_5 = 1 - poisson.cdf(4, mu=lambda_param)
print(f"- Probability of observing at least 5 events: P(X >= 5) =
{prob_at_least_5:.4f}")
```



```
Lambda (average rate): 2.5
X values (number of events): [0 1 2 3 4 5 6 7 8]
CDF Values (P(X <= x)): [0.0821 0.2873 0.5438 0.7576 0.8912 0.958  0.9858
0.9958 0.9989]

Interpretation Examples:
- Probability of observing 0 events: P(X <= 0) = 0.0821
- Probability of observing at most 2 events: P(X <= 2) = 0.5438
- Probability of observing exactly 3 events: P(X = 3) = 0.2138
- Probability of observing at least 5 events: P(X >= 5) = 0.1088
```

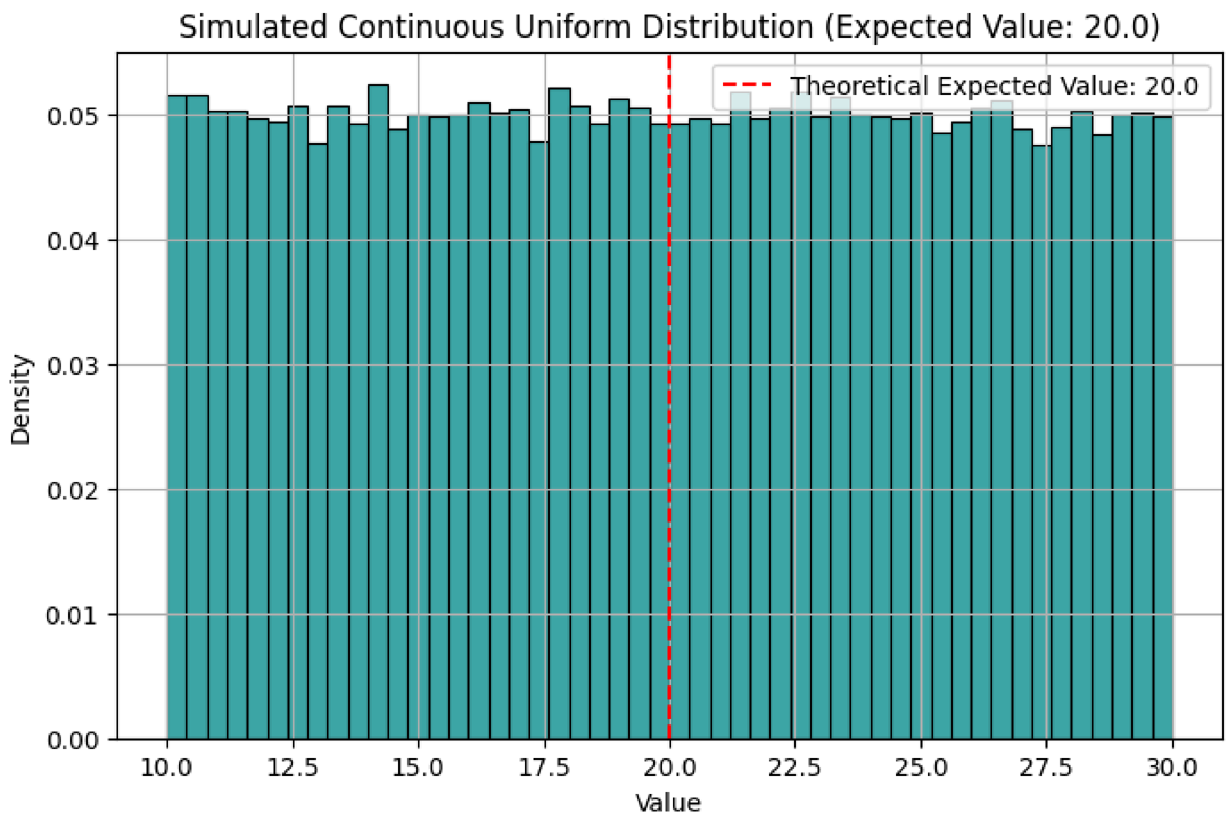
20. Simulate a random variable using a continuous uniform distribution and calculate its expected value.

In [23]:

```python
import numpy as np

# Define the interval [a, b] for the continuous uniform distribution
a = 10
b = 30
num_samples = 100000 # Large number of samples to approximate expected value

# Simulate random variables from the continuous uniform distribution
simulated_data = np.random.uniform(low=a, high=b, size=num_samples) #
```

```
# Calculate the theoretical Expected Value (Mean) for a continuous uniform
distribution
# E[X] = (a + b) / 2
theoretical_expected_value = (a + b) / 2 #

# Calculate the observed Expected Value (Mean) from the simulated data
observed_expected_value = np.mean(simulated_data) #

print(f"Interval for Uniform Distribution: [{a}, {b}]")
print(f"Theoretical Expected Value: {theoretical_expected_value}") #
print(f"Observed Expected Value from simulated data:
{observed_expected_value:.2f}") #

# Plotting a histogram to visualize the distribution
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 5))
sns.histplot(simulated_data, bins=50, stat='density', color='darkcyan',
edgecolor='black') #
plt.axvline(theoretical_expected_value, color='red', linestyle='--',
label=f'Theoretical Expected Value: {theoretical_expected_value}')
plt.title(f'Simulated Continuous Uniform Distribution (Expected Value:
{theoretical_expected_value})')
plt.xlabel('Value')
plt.ylabel('Density')
plt.legend()
plt.grid(True)
plt.show()

print("\nInterpretation:")
print("The expected value of a random variable from a continuous uniform
distribution")
print("is simply the midpoint of its interval. As the number of simulations
increases,")
print("the observed mean of the simulated data will converge to this
theoretical expected value.")
```

```
Interval for Uniform Distribution: [10, 30]
Theoretical Expected Value: 20.0
Observed Expected Value from simulated data: 19.97
```



```
Interpretation:
The expected value of a random variable from a continuous uniform distribution
is simply the midpoint of its interval. As the number of simulations increases,
the observed mean of the simulated data will converge to this theoretical
expected value.
```

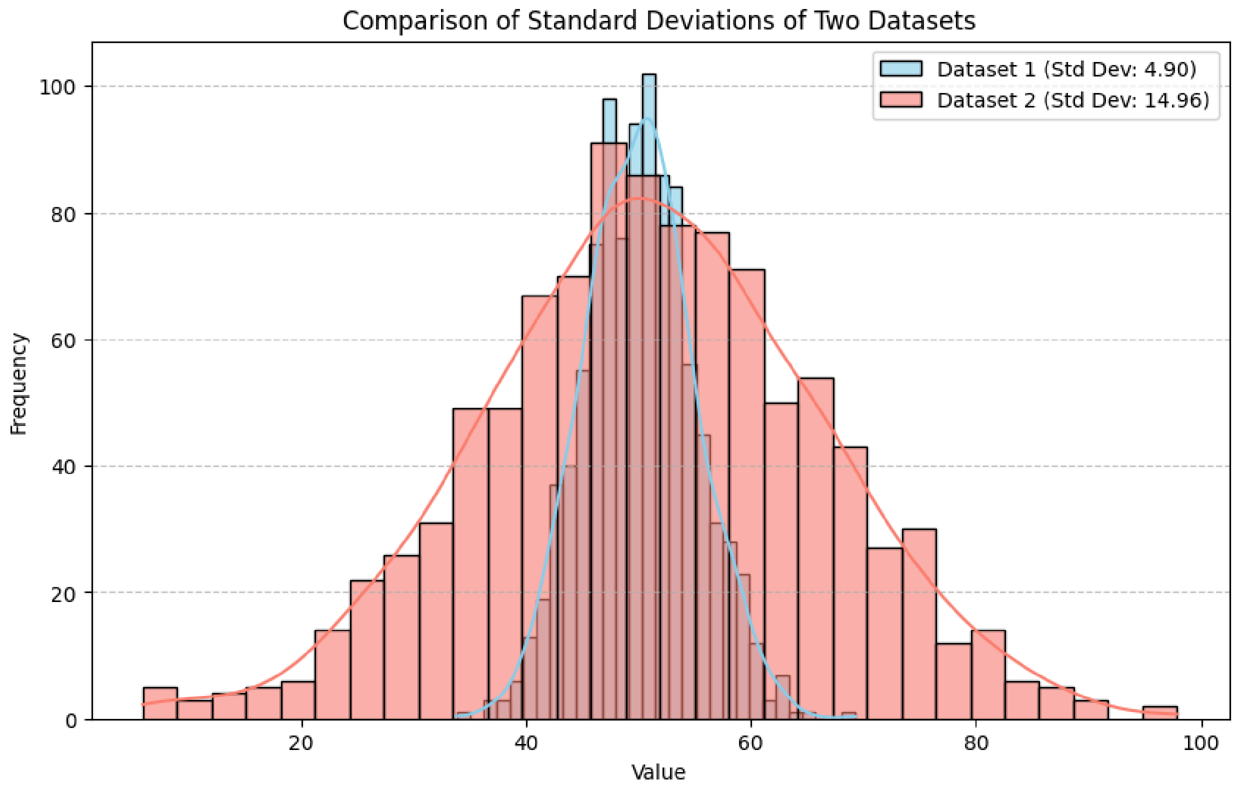
21. Write a Python program to compare the standard deviations of two datasets and visualize the difference.

In [26]:

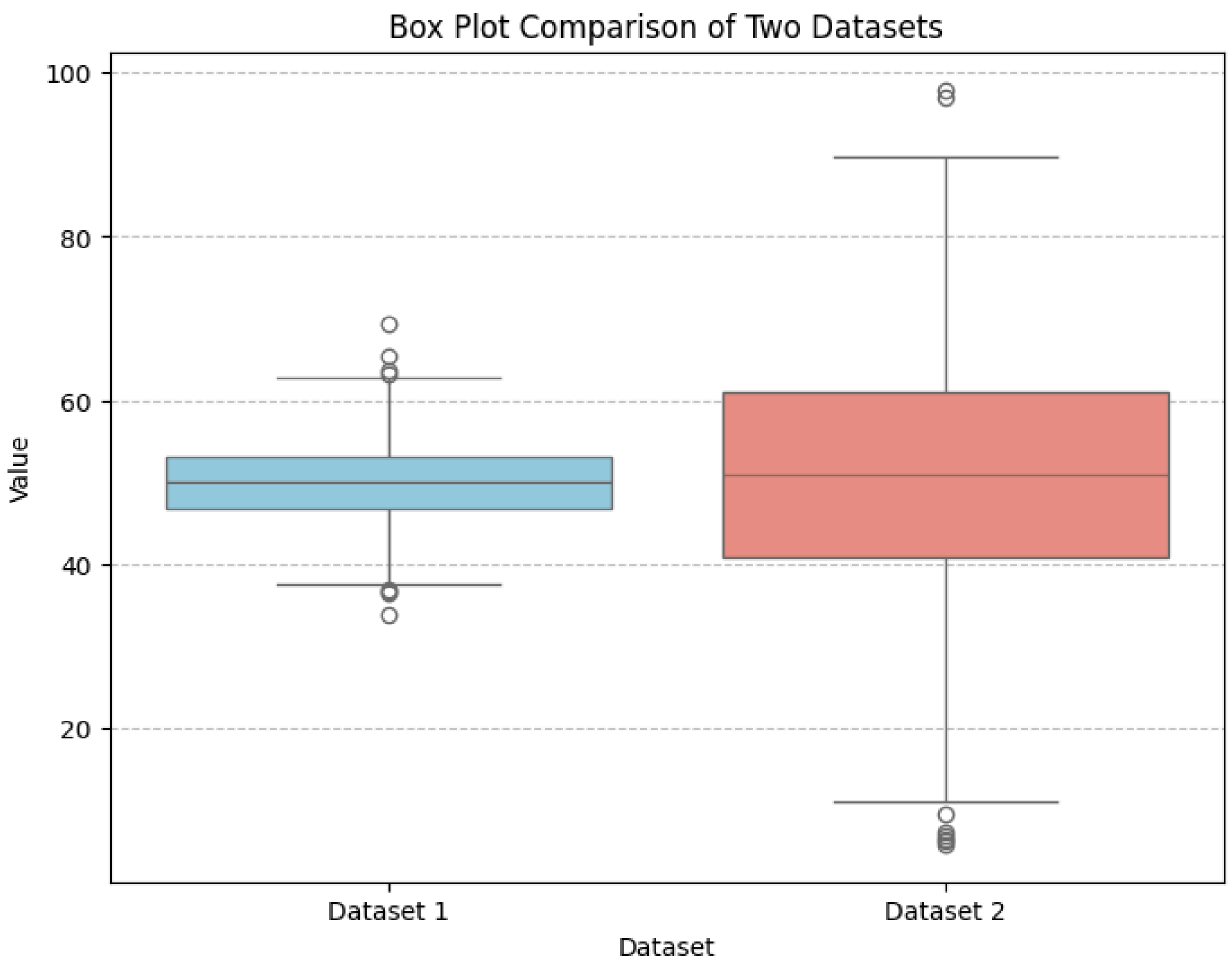
```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Generate two datasets with different standard deviations
np.random.seed(42) # for reproducibility

# Dataset 1: Lower variability
data1_mean = 50
data1_std = 5
dataset1 = np.random.normal(loc=data1_mean, scale=data1_std, size=1000)
```

```python
# Dataset 2: Higher variability
data2_mean = 50 # Same mean for easier comparison of spread
data2_std = 15
dataset2 = np.random.normal(loc=data2_mean, scale=data2_std, size=1000)

# Calculate standard deviations
std_dev_data1 = np.std(dataset1, ddof=1) # Use ddof=1 for sample std dev
std_dev_data2 = np.std(dataset2, ddof=1) #

print(f"Dataset 1 (Mean={data1_mean}, Std Dev={data1_std}) - Calculated Std
Dev: {std_dev_data1:.2f}") #
print(f"Dataset 2 (Mean={data2_mean}, Std Dev={data2_std}) - Calculated Std
Dev: {std_dev_data2:.2f}") #

# Visualize the difference using histograms
plt.figure(figsize=(10, 6))
sns.histplot(dataset1, kde=True, color='skyblue', label=f'Dataset 1 (Std Dev:
{std_dev_data1:.2f})', alpha=0.6, bins=30) #
sns.histplot(dataset2, kde=True, color='salmon', label=f'Dataset 2 (Std Dev:
{std_dev_data2:.2f})', alpha=0.6, bins=30) #
plt.title('Comparison of Standard Deviations of Two Datasets') #
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

# Visualize the difference using box plots (good for showing spread and
outliers)
combined_data = pd.DataFrame({
    'Value': np.concatenate([dataset1, dataset2]),
    'Dataset': ['Dataset 1'] * len(dataset1) + ['Dataset 2'] * len(dataset2)
})

plt.figure(figsize=(8, 6))
sns.boxplot(x='Dataset', y='Value', data=combined_data, palette={'Dataset 1':
'skyblue', 'Dataset 2': 'salmon'}) #
plt.title('Box Plot Comparison of Two Datasets') #
plt.ylabel('Value')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

print("\nInterpretation:")
print("- Dataset with a smaller standard deviation (Dataset 1) has values that
are more clustered around its mean.")
print("- Dataset with a larger standard deviation (Dataset 2) has values that
are more spread out from its mean.")
print("- The histograms show the narrower/wider spread visually, and the box
plots confirm this with smaller/larger boxes.")
```

```
Dataset 1 (Mean=50, Std Dev=5) - Calculated Std Dev: 4.90
Dataset 2 (Mean=50, Std Dev=15) - Calculated Std Dev: 14.96
```



```
<ipython-input-26-feda95e2c040>:44: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(x='Dataset', y='Value', data=combined_data, palette={'Dataset 1':
'skyblue', 'Dataset 2': 'salmon'}) #
```



```
Interpretation:
- Dataset with a smaller standard deviation (Dataset 1) has values that are
more clustered around its mean.
- Dataset with a larger standard deviation (Dataset 2) has values that are more
spread out from its mean.
- The histograms show the narrower/wider spread visually, and the box plots
confirm this with smaller/larger boxes.
```

22. Calculate the range and interquartile range (IQR) of a dataset generated from a normal distribution.

In [27]:

```python
import numpy as np

# Generate a dataset from a normal distribution
np.random.seed(42) # for reproducibility
mean_val = 100
std_dev_val = 10
num_points = 50
```

```
data = np.random.normal(loc=mean_val, scale=std_dev_val, size=num_points) #

print(f"Generated Dataset (first 10): {data[:10].round(2)}")
print(f"Mean of Dataset: {np.mean(data):.2f}")
print(f"Standard Deviation of Dataset: {np.std(data):.2f}")

# Calculate the Range
data_range = np.max(data) - np.min(data) #
print(f"\nRange: {data_range:.2f}") #

# Calculate the Interquartile Range (IQR)
Q1 = np.percentile(data, 25) # 25th percentile (First Quartile)
Q3 = np.percentile(data, 75) # 75th percentile (Third Quartile)
IQR = Q3 - Q1 #
print(f"Q1 (25th percentile): {Q1:.2f}") #
print(f"Q3 (75th percentile): {Q3:.2f}") #
print(f"Interquartile Range (IQR): {IQR:.2f}") #

# Interpretation
print("\nInterpretation:")
print("- **Range:** The difference between the highest and lowest values. It's
a simple measure of total spread but is highly sensitive to outliers.")
print("- **IQR:** The range of the middle 50% of the data. It's a robust
measure of spread, as it's not affected by extreme outliers.")
print(f"  The middle 50% of your data spans {IQR:.2f} units.")
```

```
Generated Dataset (first 10): [104.97  98.62 106.48 115.23  97.66  97.66 115.79
107.67  95.31 105.43]
Mean of Dataset: 97.75
Standard Deviation of Dataset: 9.24

Range: 38.12
Q1 (25th percentile): 91.39
Q3 (75th percentile): 103.36
Interquartile Range (IQR): 11.97

Interpretation:
- **Range:** The difference between the highest and lowest values. It's a
simple measure of total spread but is highly sensitive to outliers.
- **IQR:** The range of the middle 50% of the data. It's a robust measure of
spread, as it's not affected by extreme outliers.
  The middle 50% of your data spans 11.97 units.
```

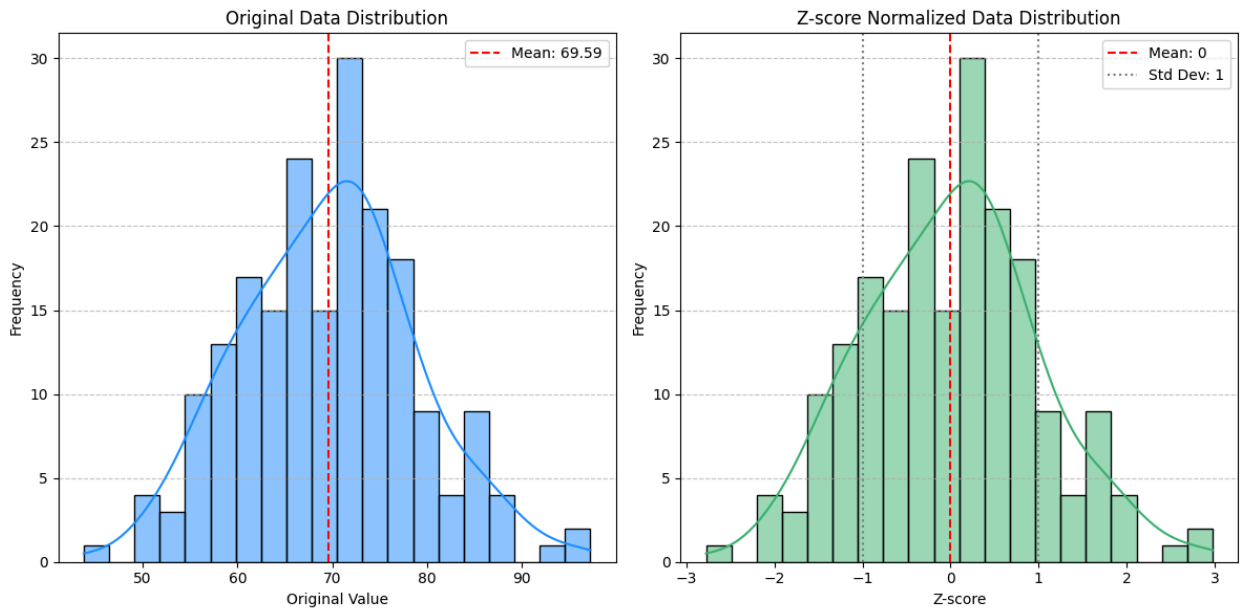
23. Implement Z-score normalization on a dataset and visualize its transformation.

In [28]:

```
import numpy as np
```

```python
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm

# Generate an original dataset (e.g., exam scores with a specific mean and std
dev)
np.random.seed(42) # for reproducibility
original_data = np.random.normal(loc=70, scale=10, size=200) # Mean 70, Std Dev
10

# Calculate Z-scores (Z-score normalization)
mean_original = np.mean(original_data)
std_dev_original = np.std(original_data)
if std_dev_original == 0:
    normalized_data = np.zeros_like(original_data)
else:
    normalized_data = (original_data - mean_original) / std_dev_original #

print(f"Original Data Mean: {mean_original:.2f}")
print(f"Original Data Std Dev: {std_dev_original:.2f}")
print(f"Normalized Data Mean (should be near 0):
{np.mean(normalized_data):.2f}")
print(f"Normalized Data Std Dev (should be near 1):
{np.std(normalized_data):.2f}")

# Visualize the transformation
plt.figure(figsize=(12, 6))

# Original Data Histogram
plt.subplot(1, 2, 1)
sns.histplot(original_data, kde=True, color='dodgerblue', edgecolor='black',
bins=20) #
plt.title('Original Data Distribution') #
plt.xlabel('Original Value')
plt.ylabel('Frequency')
plt.axvline(mean_original, color='red', linestyle='--', label=f'Mean:
{mean_original:.2f}')
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Normalized Data Histogram
plt.subplot(1, 2, 2)
sns.histplot(normalized_data, kde=True, color='mediumseagreen',
edgecolor='black', bins=20) #
plt.title('Z-score Normalized Data Distribution') #
plt.xlabel('Z-score')
plt.ylabel('Frequency')
plt.axvline(0, color='red', linestyle='--', label='Mean: 0')
plt.axvline(1, color='gray', linestyle=':', label='Std Dev: 1')
plt.axvline(-1, color='gray', linestyle=':')
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()

print("\nInterpretation of Z-score Normalization:")
print("- **Transformation:** Z-score normalization (standardization) transforms
data so that it has a mean of 0 and a standard deviation of 1.")
print("- **Comparability:** This makes data from different scales directly
comparable, as values are expressed in terms of standard deviations from their
respective means.")
print("- **Machine Learning:** It's a crucial preprocessing step for many
machine learning algorithms (e.g., K-Means, SVMs, Logistic Regression, Neural
Networks) that are sensitive to the scale of input features.")
```

```
Original Data Mean: 69.59
Original Data Std Dev: 9.29
Normalized Data Mean (should be near 0): 0.00
Normalized Data Std Dev (should be near 1): 1.00
```



```
Interpretation of Z-score Normalization:
- **Transformation:** Z-score normalization (standardization) transforms data
so that it has a mean of 0 and a standard deviation of 1.
- **Comparability:** This makes data from different scales directly comparable,
as values are expressed in terms of standard deviations from their respective
means.
```

```
- **Machine Learning:** It's a crucial preprocessing step for many machine
learning algorithms (e.g., K-Means, SVMs, Logistic Regression, Neural Networks)
that are sensitive to the scale of input features.
```

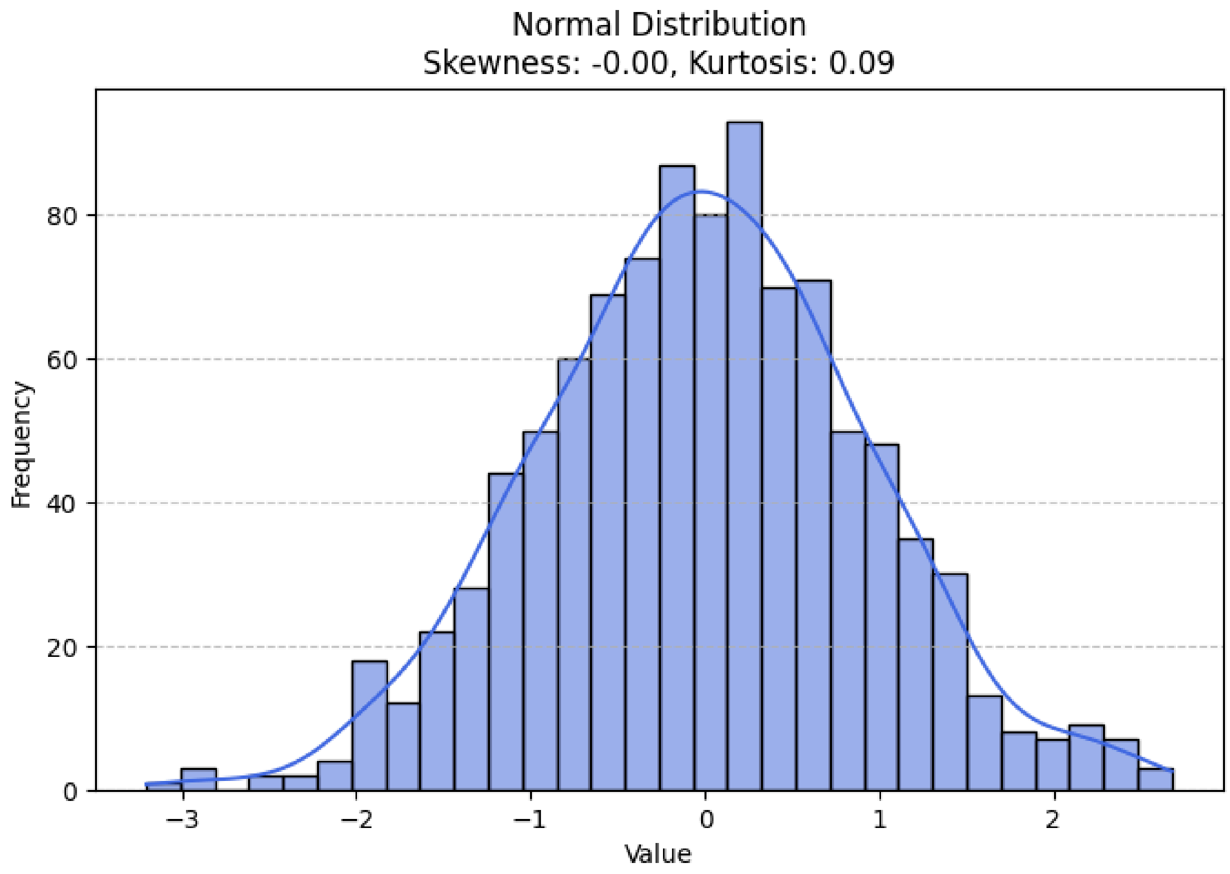
24. Write a Python function to calculate the skewness and kurtosis of a dataset generated from a normal distribution.

In [29]:

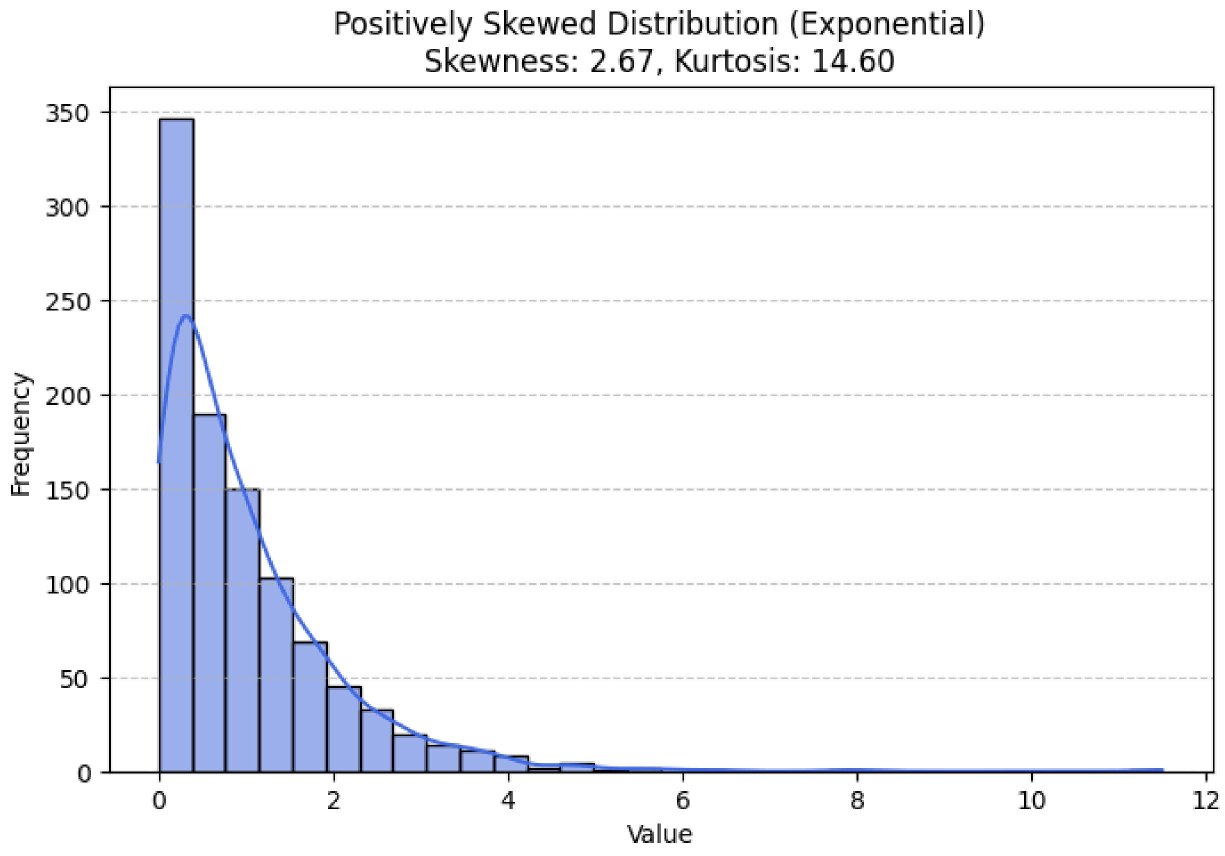
```python
import numpy as np
from scipy.stats import skew, kurtosis
import matplotlib.pyplot as plt
import seaborn as sns

def calculate_and_plot_shape_metrics(data, title="Data Distribution"):
    """
    Calculates skewness and kurtosis of a dataset and plots its histogram.
    """
    data_skewness = skew(data) #
    data_kurtosis = kurtosis(data) # Fisher's kurtosis (normal distribution
has 0 kurtosis)

    print(f"\n--- {title} ---")
    print(f"Skewness: {data_skewness:.4f}") #
    print(f"Kurtosis (Fisher): {data_kurtosis:.4f}") #

    plt.figure(figsize=(8, 5))
    sns.histplot(data, kde=True, bins=30, color='royalblue', edgecolor='black')
#
    plt.title(f'{title}\nSkewness: {data_skewness:.2f}, Kurtosis:
{data_kurtosis:.2f}') #
    plt.xlabel('Value')
    plt.ylabel('Frequency')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()

# Generate a dataset from a normal distribution
np.random.seed(10) # for reproducibility
normal_data = np.random.normal(loc=0, scale=1, size=1000) #

# Calculate and plot for normal data
calculate_and_plot_shape_metrics(normal_data, title="Normal Distribution")

# Example with positively skewed data (for comparison)
positive_skew_data = np.random.exponential(scale=1, size=1000)
calculate_and_plot_shape_metrics(positive_skew_data, title="Positively Skewed
Distribution (Exponential)")

# Example with platykurtic data (flatter than normal, e.g., uniform)
```

```
platykurtic_data = np.random.uniform(low=-3, high=3, size=1000)
calculate_and_plot_shape_metrics(platykurtic_data, title="Platykurtic
Distribution (Uniform)")

print("\nInterpretation of Skewness and Kurtosis:")
print("- **Skewness:** Measures the asymmetry of the distribution.")
print("  - A value close to 0 indicates a symmetric distribution (like
normal).")
print("  - Positive skewness (>0) means the tail is on the right (more values
on the left, a few large values pulling the mean).")
print("  - Negative skewness (<0) means the tail is on the left (more values on
the right, a few small values pulling the mean).")
print("- **Kurtosis (Fisher):** Measures the 'tailedness' or 'peakedness' of
the distribution compared to a normal distribution.")
print("  - A value close to 0 (for Fisher's kurtosis) indicates mesokurtic
(similar to normal).")
print("  - Positive kurtosis (>0, Leptokurtic): Sharper peak and fatter tails
(more outliers) than normal.")
print("  - Negative kurtosis (<0, Platykurtic): Flatter peak and thinner tails
(fewer outliers) than normal.")
```

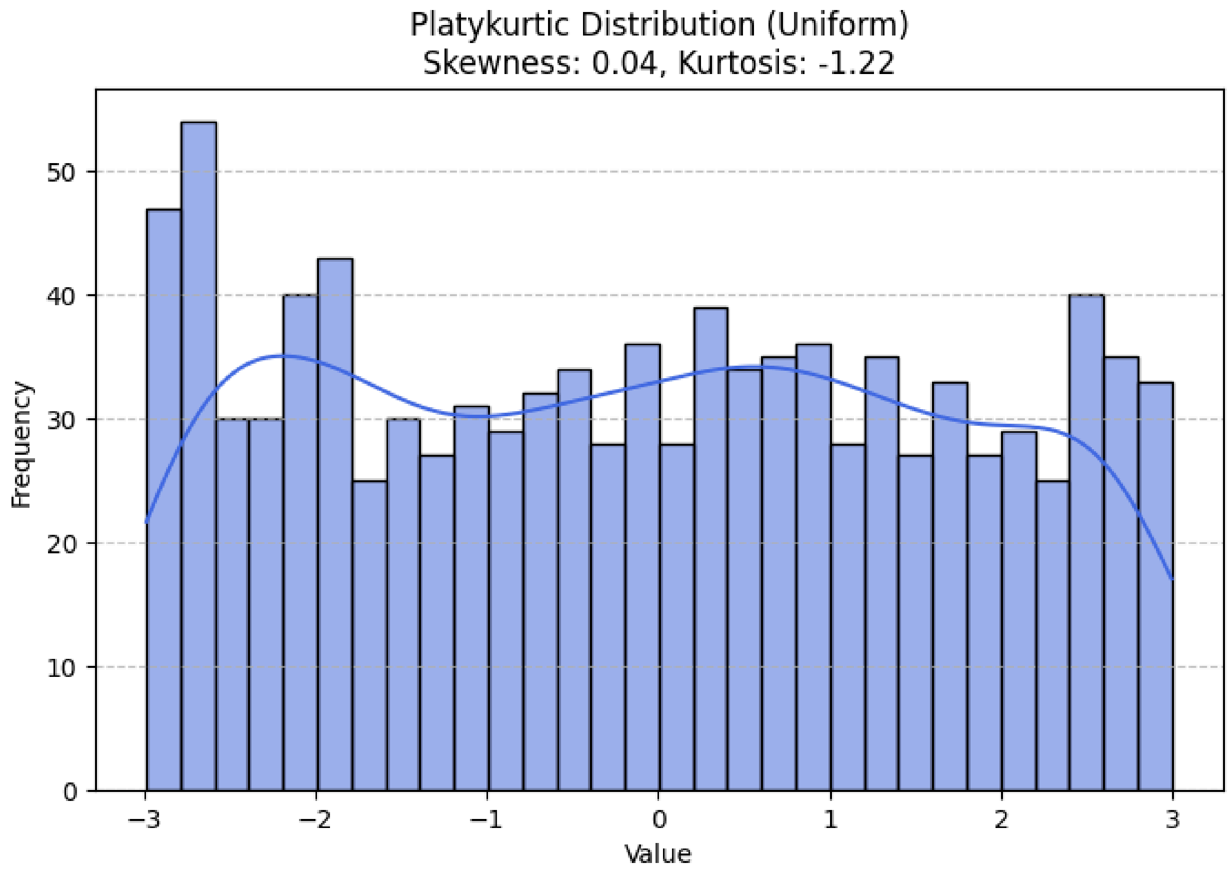
```
--- Normal Distribution ---
Skewness: -0.0031
Kurtosis (Fisher): 0.0923
```



```
--- Positively Skewed Distribution (Exponential) ---
Skewness: 2.6680
Kurtosis (Fisher): 14.6002
```



```
--- Platykurtic Distribution (Uniform) ---
Skewness: 0.0385
Kurtosis (Fisher): -1.2186
```



```
Interpretation of Skewness and Kurtosis:
- **Skewness:** Measures the asymmetry of the distribution.
  - A value close to 0 indicates a symmetric distribution (like normal).
  - Positive skewness (>0) means the tail is on the right (more values on the
left, a few large values pulling the mean).
  - Negative skewness (<0) means the tail is on the left (more values on the
right, a few small values pulling the mean).
- **Kurtosis (Fisher):** Measures the 'tailedness' or 'peakedness' of the
distribution compared to a normal distribution.
  - A value close to 0 (for Fisher's kurtosis) indicates mesokurtic (similar to
normal).
  - Positive kurtosis (>0, Leptokurtic): Sharper peak and fatter tails (more
outliers) than normal.
  - Negative kurtosis (<0, Platykurtic): Flatter peak and thinner tails (fewer
outliers) than normal.
```

Practical Questions Part - 2

1. Write a Python program to perform a Z-test for comparing a sample mean to a known population mean and interpret the results.

In [31]:

```
import numpy as np
from scipy import stats

def z_test_single_sample(sample_mean, population_mean, population_std,
sample_size, alpha=0.05):
    """
    Performs a one-sample Z-test and interprets the results.

    Args:
        sample_mean (float): The mean of the sample.
        population_mean (float): The known mean of the population.
        population_std (float): The known standard deviation of the population.
        sample_size (int): The number of observations in the sample.
        alpha (float): The significance level (default is 0.05).

    Returns:
        tuple: A tuple containing the Z-statistic, P-value, and a string
interpretation.
    """
    # Calculate the Z-statistic
    standard_error = population_std / np.sqrt(sample_size)
    z_statistic = (sample_mean - population_mean) / standard_error

    # Calculate the P-value (two-tailed)
    p_value = 2 * (1 - stats.norm.cdf(abs(z_statistic)))

    # Interpret the results
    interpretation = ""
    if p_value < alpha:
        interpretation = (
            f"The P-value ({p_value:.4f}) is less than the significance level
({alpha}).\n"
            "We reject the null hypothesis. There is statistically significant
evidence "
            "that the sample mean is different from the population mean."
        )
    else:
        interpretation = (
            f"The P-value ({p_value:.4f}) is greater than or equal to the
significance level ({alpha}).\n"
            "We fail to reject the null hypothesis. There is no statistically
significant evidence "
            "that the sample mean is different from the population mean."
        )
```

```python
    return z_statistic, p_value, interpretation

# Example Usage:
# A researcher wants to know if the average IQ of a group of students is
different from the national average.
# National average IQ (population mean) = 100, Population standard deviation =
15.
# Sample of 30 students has an average IQ of 105.

population_mean = 100
population_std = 15
sample_mean = 105
sample_size = 30
alpha = 0.05

z_stat, p_val, interpretation = z_test_single_sample(
    sample_mean, population_mean, population_std, sample_size, alpha
)

print(f"Z-statistic: {z_stat:.4f}")
print(f"P-value: {p_val:.4f}")
print("\nInterpretation:\n", interpretation)

# Another example: Sample mean is very close to population mean
sample_mean_2 = 101
sample_size_2 = 50
z_stat_2, p_val_2, interpretation_2 = z_test_single_sample(
    sample_mean_2, population_mean, population_std, sample_size_2, alpha
)
print("\n--- Another Example ---")
print(f"Z-statistic: {z_stat_2:.4f}")
print(f"P-value: {p_val_2:.4f}")
print("\nInterpretation:\n", interpretation_2)
```

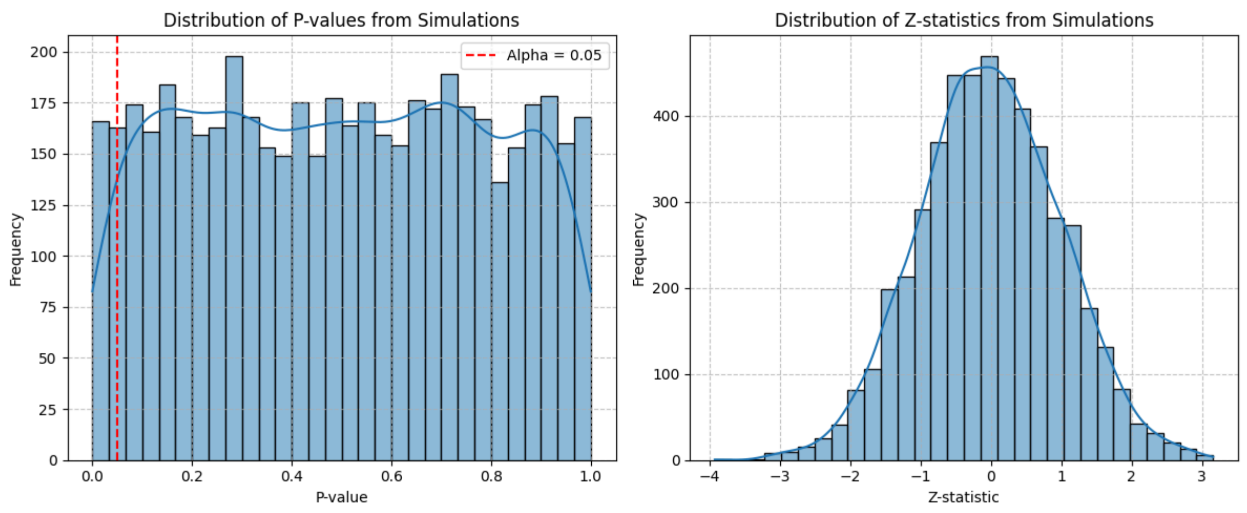
```
Z-statistic: 1.8257
P-value: 0.0679

Interpretation:
 The P-value (0.0679) is greater than or equal to the significance level
(0.05).
We fail to reject the null hypothesis. There is no statistically significant
evidence that the sample mean is different from the population mean.

--- Another Example ---
Z-statistic: 0.4714
P-value: 0.6374

Interpretation:
 The P-value (0.6374) is greater than or equal to the significance level
(0.05).
We fail to reject the null hypothesis. There is no statistically significant
evidence that the sample mean is different from the population mean.
```

2. Simulate random data to perform hypothesis testing and calculate the corresponding P-value using Python.

In [32]:

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns

def simulate_and_test(population_mean, population_std, sample_size,
num_simulations=1000, alpha=0.05):
    """
    Simulates random data, performs a one-sample Z-test, and calculates
P-values.

    Args:
        population_mean (float): The true population mean.
        population_std (float): The true population standard deviation.
        sample_size (int): The size of each simulated sample.
        num_simulations (int): The number of times to simulate and test.
        alpha (float): The significance level.

    Returns:
        list: A list of P-values from the simulations.
    """
    p_values = []
    z_statistics = []

    for _ in range(num_simulations):
        # Simulate a sample from a normal distribution
        sample_data = np.random.normal(loc=population_mean,
scale=population_std, size=sample_size)
        sample_mean = np.mean(sample_data)

        # Calculate Z-statistic
        standard_error = population_std / np.sqrt(sample_size)
        z_statistic = (sample_mean - population_mean) / standard_error

        # Calculate P-value (two-tailed)
        p_value = 2 * (1 - stats.norm.cdf(abs(z_statistic)))

        p_values.append(p_value)
        z_statistics.append(z_statistic)
    return p_values, z_statistics

# Simulation parameters
population_mean_true = 70
population_std_true = 10
sample_size_sim = 50
num_simulations = 5000
alpha_sim = 0.05

print(f"Simulating {num_simulations} Z-tests...")
simulated_p_values, simulated_z_statistics = simulate_and_test(
    population_mean_true, population_std_true, sample_size_sim,
num_simulations, alpha_sim
)

# Visualize the distribution of P-values
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.histplot(simulated_p_values, bins=30, kde=True)
plt.axvline(x=alpha_sim, color='red', linestyle='--', label=f'Alpha =
{alpha_sim}')
plt.title('Distribution of P-values from Simulations')
plt.xlabel('P-value')
plt.ylabel('Frequency')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)

# Visualize the distribution of Z-statistics
plt.subplot(1, 2, 2)
sns.histplot(simulated_z_statistics, bins=30, kde=True)
plt.title('Distribution of Z-statistics from Simulations')
plt.xlabel('Z-statistic')
plt.ylabel('Frequency')
plt.grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()

# Calculate the proportion of rejected null hypotheses (Type I error rate if
H0 is true)
rejected_count = np.sum(np.array(simulated_p_values) < alpha_sim)
rejection_rate = rejected_count / num_simulations

print(f"\nProportion of times the null hypothesis was rejected (P-value <
{alpha_sim}): {rejection_rate:.4f}")
print(f"This rate should be close to alpha ({alpha_sim}) if the null hypothesis
is true (as simulated here).")
```

```
Simulating 5000 Z-tests...
```



```
Proportion of times the null hypothesis was rejected (P-value < 0.05): 0.0490
This rate should be close to alpha (0.05) if the null hypothesis is true (as
simulated here).
```

Explanation:

Data Simulation: We use np.random.normal() to generate sample data points from a normal distribution with the specified population_mean_true and population_std_true. This means we are simulating a scenario where the null hypothesis is actually true.

Hypothesis Testing: For each simulated sample, we calculate the sample mean and then perform a one-sample Z-test as described in the previous question, obtaining a Z-statistic and P-value.

P-value Distribution:

When the null hypothesis is true (as in this simulation where sample_data is drawn from a distribution with population_mean_true), the P-values are expected to be uniformly distributed between 0 and 1.

The histogram of P-values visually confirms this.

The proportion of times we reject the null hypothesis (i.e., P-value < α) should be approximately equal to α. This demonstrates the Type I error rate.

Z-statistic Distribution: The Z-statistics, when the null hypothesis is true, should follow a standard normal distribution (mean 0, standard deviation 1). The histogram of Z-statistics should reflect this.

3. Implement a one-sample Z-test using Python to compare the sample mean with the population mean.

In [33]:

```
import numpy as np
from scipy import stats

def one_sample_z_test(sample_data, population_mean, population_std, alpha=0.05,
alternative='two-sided'):
    """
    Performs a one-sample Z-test.

    Args:
        sample_data (array-like): The observed sample data.
        population_mean (float): The known mean of the population.
        population_std (float): The known standard deviation of the population.
        alpha (float): The significance level (default is 0.05).
        alternative (str): The alternative hypothesis. Options:
                           'two-sided' (default): sample mean is not equal to
population mean.
                           'less': sample mean is less than population mean.
                           'greater': sample mean is greater than population
mean.

    Returns:
        dict: A dictionary containing the Z-statistic, P-value, and decision.
    """
    sample_mean = np.mean(sample_data)
    sample_size = len(sample_data)

    if sample_size == 0:
        raise ValueError("Sample data cannot be empty.")

    standard_error = population_std / np.sqrt(sample_size)
    z_statistic = (sample_mean - population_mean) / standard_error

    if alternative == 'two-sided':
        p_value = 2 * (1 - stats.norm.cdf(abs(z_statistic)))
        decision = "Reject H0" if p_value < alpha else "Fail to Reject H0"
        conclusion = f"Sample mean is significantly different from population
mean." if p_value < alpha else \
                     f"Sample mean is not significantly different from
population mean."
    elif alternative == 'less':
        p_value = stats.norm.cdf(z_statistic)
        decision = "Reject H0" if p_value < alpha else "Fail to Reject H0"
        conclusion = f"Sample mean is significantly less than population mean."
if p_value < alpha else \
                     f"Sample mean is not significantly less than population
mean."
    elif alternative == 'greater':
        p_value = 1 - stats.norm.cdf(z_statistic)
        decision = "Reject H0" if p_value < alpha else "Fail to Reject H0"
        conclusion = f"Sample mean is significantly greater than population
mean." if p_value < alpha else \
                     f"Sample mean is not significantly greater than population
mean."
    else:
        raise ValueError("Invalid 'alternative' argument. Must be 'two-sided',
'less', or 'greater'.")

    return {
        "Z-statistic": z_statistic,
        "P-value": p_value,
        "Alpha": alpha,
        "Decision": decision,
        "Conclusion": conclusion
    }

# Example Usage:
```

```python
# Scenario: A machine is supposed to fill bottles with 500ml of liquid.
# Known population standard deviation (from historical data) = 10ml.
# A sample of 40 bottles is taken, and the average fill is 495ml.

population_mean_fill = 500
population_std_fill = 10
sample_fill_data = np.random.normal(loc=495, scale=10, size=40) # Simulate a
sample with a mean of 495

print("--- Two-sided test ---")
results_two_sided = one_sample_z_test(sample_fill_data, population_mean_fill,
population_std_fill, alternative='two-sided')
for key, value in results_two_sided.items():
    print(f"{key}: {value}")

print("\n--- One-sided test (less) ---")
results_less = one_sample_z_test(sample_fill_data, population_mean_fill,
population_std_fill, alternative='less')
for key, value in results_less.items():
    print(f"{key}: {value}")

print("\n--- One-sided test (greater) ---")
# Let's simulate a sample where the mean is likely greater
sample_fill_data_greater = np.random.normal(loc=508, scale=10, size=40)
results_greater = one_sample_z_test(sample_fill_data_greater,
population_mean_fill, population_std_fill, alternative='greater')
for key, value in results_greater.items():
    print(f"{key}: {value}")
```

```
--- Two-sided test ---
Z-statistic: -3.475124163141082
P-value: 0.0005106169494573098
Alpha: 0.05
Decision: Reject H0
Conclusion: Sample mean is significantly different from population mean.

--- One-sided test (less) ---
Z-statistic: -3.475124163141082
P-value: 0.0002553084747286446
Alpha: 0.05
Decision: Reject H0
Conclusion: Sample mean is significantly less than population mean.

--- One-sided test (greater) ---
Z-statistic: 7.612242717306126
P-value: 1.3433698597964394e-14
Alpha: 0.05
Decision: Reject H0
Conclusion: Sample mean is significantly greater than population mean.
```

Key Enhancements:

Direct sample_data input: The function now takes raw sample_data and calculates the sample mean and size internally.

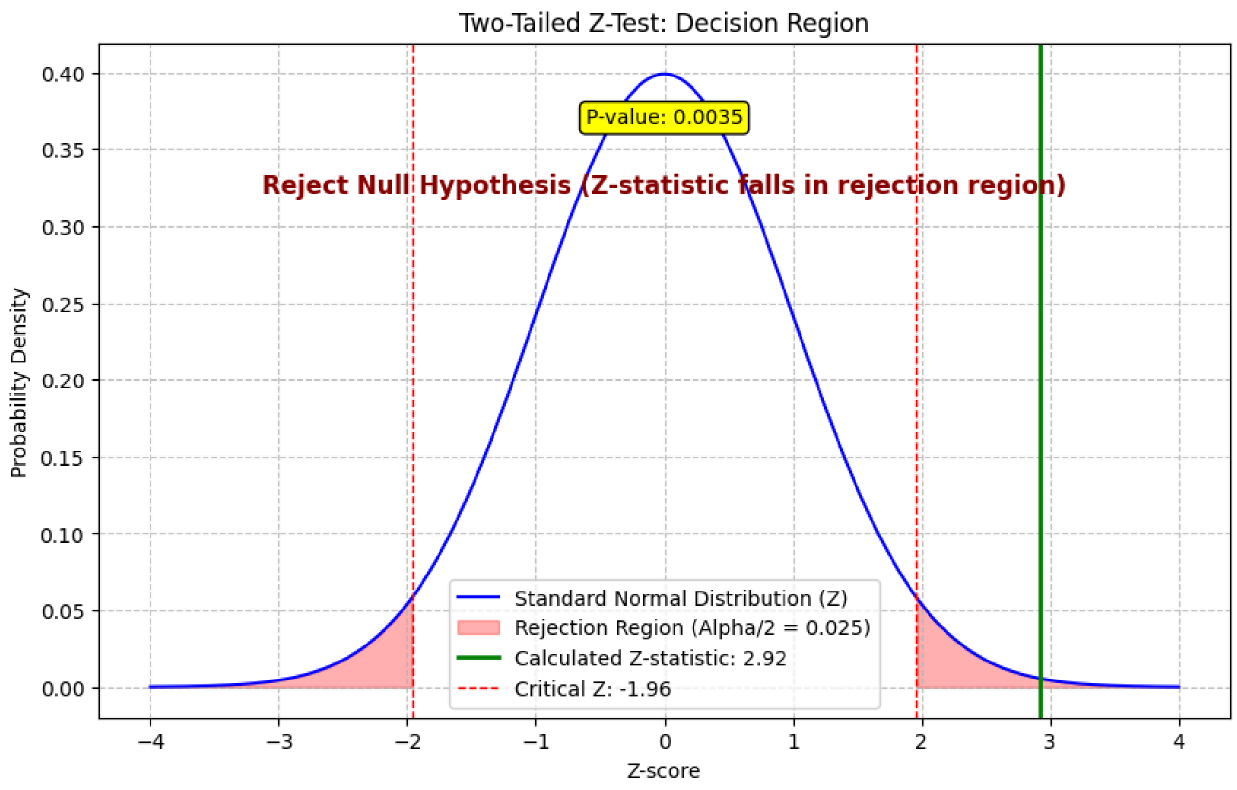
alternative argument: Allows specifying one-sided (less, greater) or two-sided tests, which changes how the P-value is calculated and the conclusion is framed.

4. Perform a two-tailed Z-test using Python and visualize the decision region on a plot.

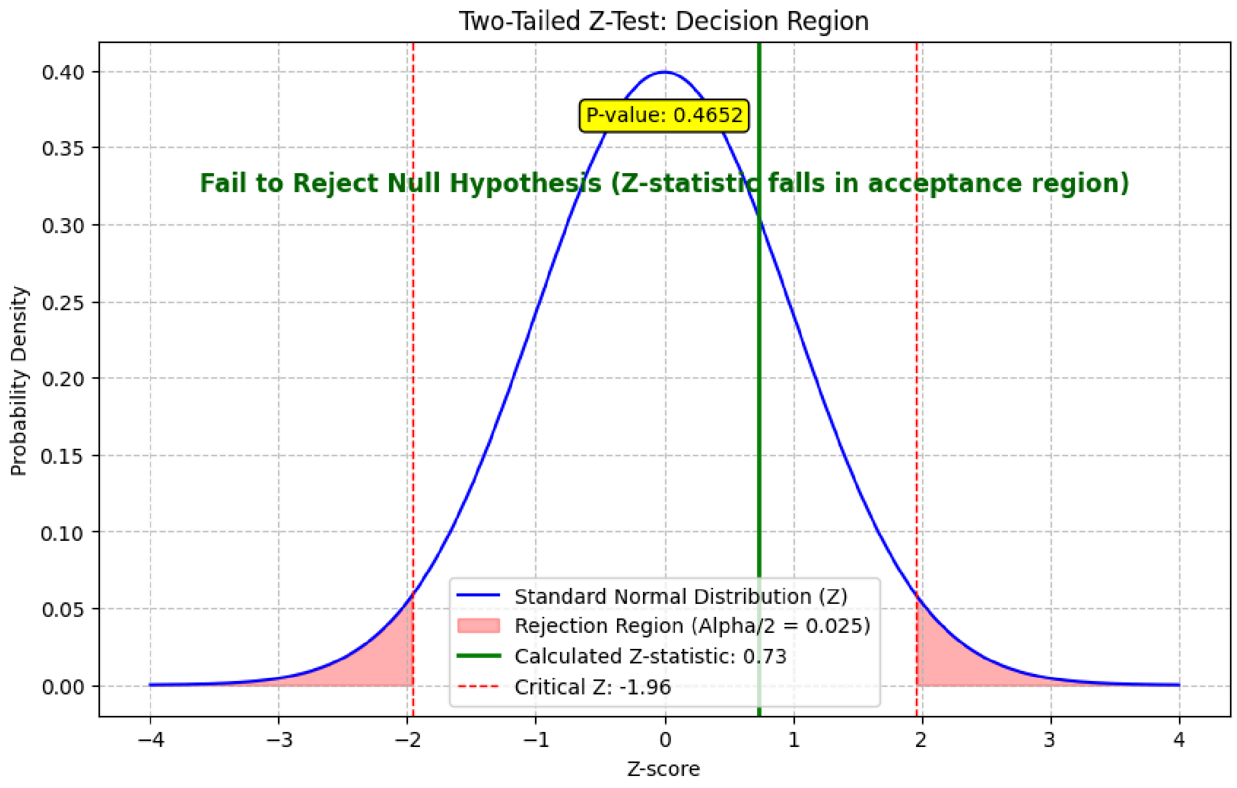
In [39]:

```python
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns

def plot_z_test_decision_region(sample_mean, population_mean, population_std,
sample_size, alpha=0.05):
    """
    Performs a two-tailed Z-test and visualizes the decision regions.

    Args:
        sample_mean (float): The mean of the sample.
        population_mean (float): The known mean of the population.
        population_std (float): The known standard deviation of the population.
        sample_size (int): The number of observations in the sample.
        alpha (float): The significance level (default is 0.05).
    """
    standard_error = population_std / np.sqrt(sample_size)
    z_statistic = (sample_mean - population_mean) / standard_error
    p_value = 2 * (1 - stats.norm.cdf(abs(z_statistic)))

    # Calculate critical Z-values for a two-tailed test
    critical_z_lower = stats.norm.ppf(alpha / 2)
    critical_z_upper = stats.norm.ppf(1 - alpha / 2)

    # Generate x values for the standard normal distribution
    x = np.linspace(-4, 4, 500)
    # Calculate y values (PDF)
    y = stats.norm.pdf(x, 0, 1)
    plt.figure(figsize=(10, 6))
    sns.lineplot(x=x, y=y, color='blue', label='Standard Normal Distribution
(Z)')
    plt.title('Two-Tailed Z-Test: Decision Region')
    plt.xlabel('Z-score')
    plt.ylabel('Probability Density')
    plt.grid(True, linestyle='--', alpha=0.7)

    # Shade the rejection regions
    x_reject_lower = x[x < critical_z_lower]
    y_reject_lower = y[x < critical_z_lower]
    plt.fill_between(x_reject_lower, 0, y_reject_lower, color='red', alpha=0.3,
label=f'Rejection Region (Alpha/2 = {alpha/2:.3f})')

    x_reject_upper = x[x > critical_z_upper]
    y_reject_upper = y[x > critical_z_upper]
    plt.fill_between(x_reject_upper, 0, y_reject_upper, color='red', alpha=0.3)

    # Plot the calculated Z-statistic
    plt.axvline(z_statistic, color='green', linestyle='-', linewidth=2,
label=f'Calculated Z-statistic: {z_statistic:.2f}')

    # Plot the critical Z-values
    plt.axvline(critical_z_lower, color='red', linestyle='--', linewidth=1,
label=f'Critical Z: {critical_z_lower:.2f}')
    plt.axvline(critical_z_upper, color='red', linestyle='--', linewidth=1)

    # Add text interpretation
    decision = ""
    if abs(z_statistic) > critical_z_upper: # Equivalently, if p_value < alpha
        decision = "Reject Null Hypothesis (Z-statistic falls in rejection
region)"
        decision_color = 'darkred'
    else:
        decision = "Fail to Reject Null Hypothesis (Z-statistic falls in
acceptance region)"
        decision_color = 'darkgreen'

    plt.text(0, plt.ylim()[1] * 0.9, f"P-value: {p_value:.4f}", ha='center',
va='top', bbox=dict(boxstyle="round,pad=0.3", fc="yellow", ec="black", lw=1))
    plt.text(0, plt.ylim()[1] * 0.8, decision, ha='center', va='top',
color=decision_color, fontsize=12, fontweight='bold')

    plt.legend()
    plt.show()

    print(f"Z-statistic: {z_statistic:.4f}")
    print(f"P-value: {p_value:.4f}")
    print(f"Critical Z-values: ({critical_z_lower:.4f},
{critical_z_upper:.4f})")
    print("Decision:", decision)

# Example Usage:
# Scenario 1: Reject H0 (sample mean significantly different)
print("--- Scenario 1: Reject H0 ---")
plot_z_test_decision_region(
    sample_mean=108,
    population_mean=100,
    population_std=15,
    sample_size=30,
    alpha=0.05
)

# Scenario 2: Fail to Reject H0 (sample mean not significantly different)
print("\n--- Scenario 2: Fail to Reject H0 ---")
plot_z_test_decision_region(
    sample_mean=102,
    population_mean=100,
    population_std=15,
    sample_size=30,
    alpha=0.05
)
```

```
--- Scenario 1: Reject H0 ---
```



```
Z-statistic: 2.9212
P-value: 0.0035
Critical Z-values: (-1.9600, 1.9600)
Decision: Reject Null Hypothesis (Z-statistic falls in rejection region)

--- Scenario 2: Fail to Reject H0 ---
```



```
Z-statistic: 0.7303
P-value: 0.4652
Critical Z-values: (-1.9600, 1.9600)
Decision: Fail to Reject Null Hypothesis (Z-statistic falls in acceptance
region)
```

Explanation of Visualization:

Standard Normal Distribution: The blue curve represents the probability density function (PDF) of the standard normal distribution (Z-distribution).

Critical Z-values: For a two-tailed test with significance level α, we find two critical Z-values: critical_z_lower = stats.norm.ppf(alpha / 2): The Z-score below which α/2 of the distribution lies. critical_z_upper = stats.norm.ppf(1 - alpha / 2): The Z-score below which 1−α/2 of the distribution lies (or above which α/2 lies).

Rejection Regions (Red Shaded): These are the areas in the tails of the distribution where, if the calculated Z-statistic falls, we reject the null hypothesis. The total area of these regions is α.

Calculated Z-statistic (Green Line): This vertical line shows where our sample's Z-statistic falls on the distribution.

Decision:

> If the green line falls within the red shaded regions, we reject the null hypothesis.
>
> If the green line falls between the critical Z-values (in the white area), we fail to reject the null hypothesis.

P-value text: The P-value is also displayed, which is another way to make the decision: if P-value < α, reject H0.

5. Create a Python function that calculates and visualizes Type 1 and Type 2 errors during hypothesis testing.

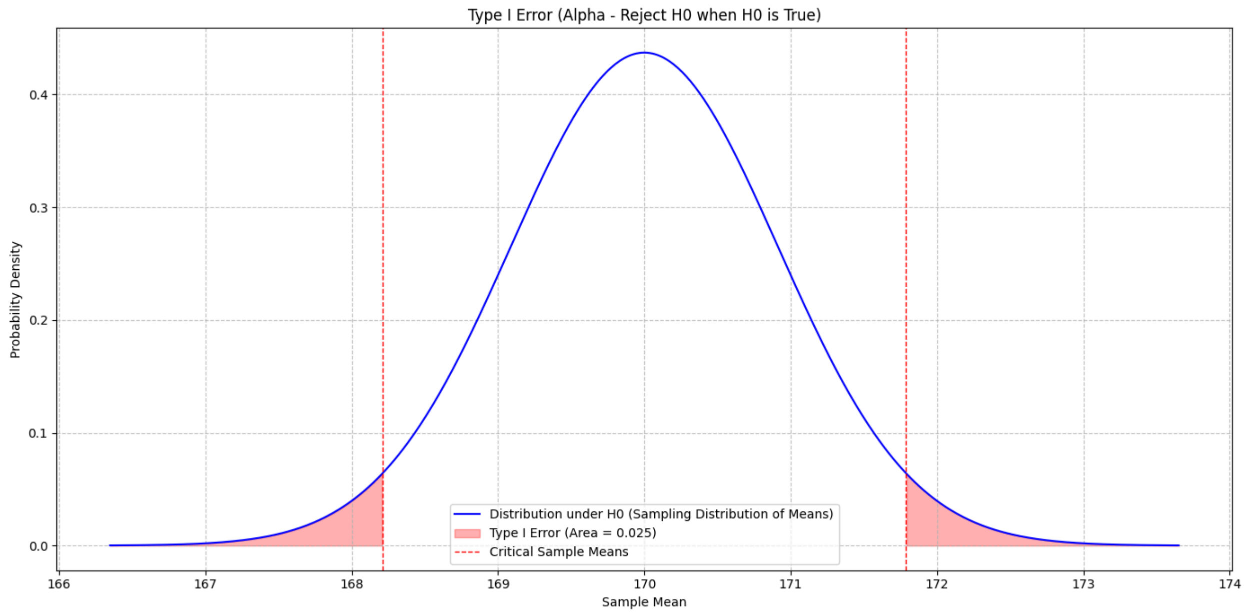
In [40]:

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns

def visualize_type_errors(population_mean_H0, population_std, sample_size, 
alpha=0.05,
                          true_population_mean_H1=None):
    """
    Calculates and visualizes Type I and Type II errors for a one-sample
Z-test.

    Args:
        population_mean_H0 (float): The mean specified in the null hypothesis
(e.g., 100).
        population_std (float): The known standard deviation of the population.
        sample_size (int): The number of observations in the sample.
        alpha (float): The significance level (Type I error rate, default
0.05).
        true_population_mean_H1 (float, optional): The true mean of the
population under
                                                  the alternative hypothesis. If
None,
                                                  only Type I error is
visualized.
    """
    standard_error = population_std / np.sqrt(sample_size)

    # 1. Visualize Type I Error (Null Hypothesis is TRUE)
    # We assume the true population mean is population_mean_H0
    critical_z_lower = stats.norm.ppf(alpha / 2)
    critical_z_upper = stats.norm.ppf(1 - alpha / 2)

    plt.figure(figsize=(14, 7))

    # Plot for Type I Error
    plt.subplot(1, 2 if true_population_mean_H1 is not None else 1, 1)
    x = np.linspace(population_mean_H0 - 4 * standard_error, population_mean_H0
+ 4 * standard_error, 500)
    pdf_H0 = stats.norm.pdf(x, population_mean_H0, standard_error)
    plt.plot(x, pdf_H0, color='blue', label='Distribution under H0 (Sampling
Distribution of Means)')
    plt.title('Type I Error (Alpha - Reject H0 when H0 is True)')
    plt.xlabel('Sample Mean')
    plt.ylabel('Probability Density')
    plt.grid(True, linestyle='--', alpha=0.7)

    # Convert critical Z-values to sample mean values
    critical_sample_mean_lower = population_mean_H0 + critical_z_lower *
standard_error
    critical_sample_mean_upper = population_mean_H0 + critical_z_upper *
standard_error

    x_reject_lower_H0 = x[x < critical_sample_mean_lower]
    y_reject_lower_H0 = pdf_H0[x < critical_sample_mean_lower]
    plt.fill_between(x_reject_lower_H0, 0, y_reject_lower_H0, color='red',
alpha=0.3, label=f'Type I Error (Area = {alpha/2:.3f})')
```

```
    x_reject_upper_H0 = x[x > critical_sample_mean_upper]
    y_reject_upper_H0 = pdf_H0[x > critical_sample_mean_upper]
    plt.fill_between(x_reject_upper_H0, 0, y_reject_upper_H0, color='red',
alpha=0.3)

    plt.axvline(critical_sample_mean_lower, color='red', linestyle='--',
linewidth=1)
    plt.axvline(critical_sample_mean_upper, color='red', linestyle='--',
linewidth=1, label=f'Critical Sample Means')
    plt.legend()


    # 2. Visualize Type II Error and Power (Null Hypothesis is FALSE)
    if true_population_mean_H1 is not None:
        plt.subplot(1, 2, 2)
        pdf_H1 = stats.norm.pdf(x, true_population_mean_H1, standard_error)
        plt.plot(x, pdf_H0, color='blue', label='Distribution under H0')
        plt.plot(x, pdf_H1, color='green', label='Distribution under H1 (True
Mean)')
        plt.title('Type II Error (Beta - Fail to Reject H0 when H0 is False) &
Power')
        plt.xlabel('Sample Mean')
        plt.ylabel('Probability Density')
        plt.grid(True, linestyle='--', alpha=0.7)

        # Rejection regions are still based on H0
        plt.fill_between(x_reject_lower_H0, 0, y_reject_lower_H0, color='red',
alpha=0.3, label='Type I Error Region')
        plt.fill_between(x_reject_upper_H0, 0, y_reject_upper_H0, color='red',
alpha=0.3)

        # Type II Error region (where H1 is true, but we fail to reject H0)
        x_beta_region = x[(x >= critical_sample_mean_lower) & (x <=
critical_sample_mean_upper)]
        y_beta_region = stats.norm.pdf(x_beta_region, true_population_mean_H1,
standard_error)
        plt.fill_between(x_beta_region, 0, y_beta_region, color='orange',
alpha=0.5, label='Type II Error (Beta)')

        # Power region (where H1 is true, and we correctly reject H0)
        x_power_lower = x[x < critical_sample_mean_lower]
        y_power_lower = stats.norm.pdf(x_power_lower, true_population_mean_H1,
standard_error)
        plt.fill_between(x_power_lower, 0, y_power_lower, color='purple',
alpha=0.3, label='Power (1-Beta)')

        x_power_upper = x[x > critical_sample_mean_upper]
        y_power_upper = stats.norm.pdf(x_power_upper, true_population_mean_H1,
standard_error)
```

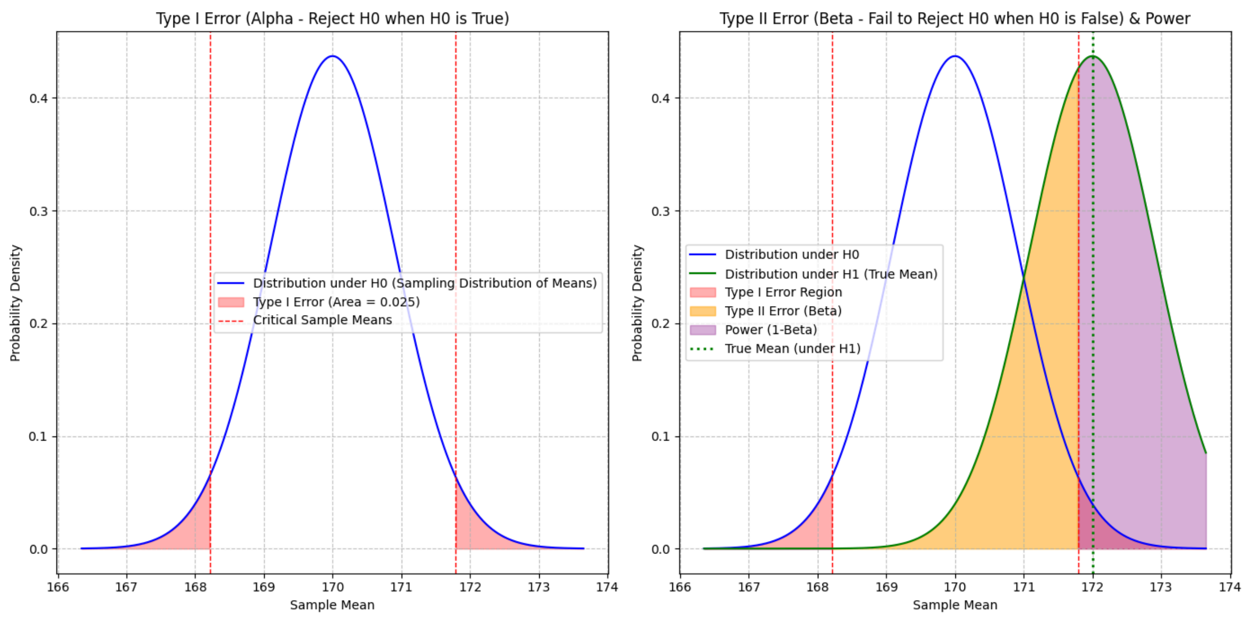
```
        plt.fill_between(x_power_upper, 0, y_power_upper, color='purple',
alpha=0.3)

        plt.axvline(critical_sample_mean_lower, color='red', linestyle='--',
linewidth=1)
        plt.axvline(critical_sample_mean_upper, color='red', linestyle='--',
linewidth=1)
        plt.axvline(true_population_mean_H1, color='green', linestyle=':',
linewidth=2, label='True Mean (under H1)')
        plt.legend()

        # Calculate Beta and Power
        # Probability of failing to reject H0 when H1 is true
        beta = stats.norm.cdf(critical_sample_mean_upper,
loc=true_population_mean_H1, scale=standard_error) - \
               stats.norm.cdf(critical_sample_mean_lower,
loc=true_population_mean_H1, scale=standard_error)
        power = 1 - beta

        print(f"\n--- Scenario with True H1 Mean ({true_population_mean_H1})
---")
        print(f"Type I Error (Alpha): {alpha:.4f}")
        print(f"Type II Error (Beta): {beta:.4f}")
        print(f"Power (1 - Beta): {power:.4f}")

    plt.tight_layout()
    plt.show()

# Example Usage:
# Scenario: Null hypothesis is that the average height is 170cm.
# Population standard deviation = 5cm. Sample size = 30.
# Alpha = 0.05.

print("--- Visualizing Type I Error (H0 is True) ---")
visualize_type_errors(
    population_mean_H0=170,
    population_std=5,
    sample_size=30,
    alpha=0.05
)

# Scenario: Also visualize Type II Error (True mean is actually 172cm)
print("\n--- Visualizing Type I and Type II Errors (H0 is False, H1 is True)
---")
visualize_type_errors(
    population_mean_H0=170,
    population_std=5,
    sample_size=30,
    alpha=0.05,
    true_population_mean_H1=172 # The true mean when H0 is false
)

# Scenario: Increase sample size to see effect on power
print("\n--- Effect of Increased Sample Size on Power ---")
visualize_type_errors(
    population_mean_H0=170,
    population_std=5,
    sample_size=100, # Increased sample size
    alpha=0.05,
    true_population_mean_H1=172
)

# Scenario: Increase difference between H0 and H1 to see effect on power
print("\n--- Effect of Larger Effect Size on Power ---")
visualize_type_errors(
    population_mean_H0=170,
    population_std=5,
    sample_size=30,
    alpha=0.05,
    true_population_mean_H1=175 # Larger difference
)
```

```
--- Visualizing Type I Error (H0 is True) ---
```

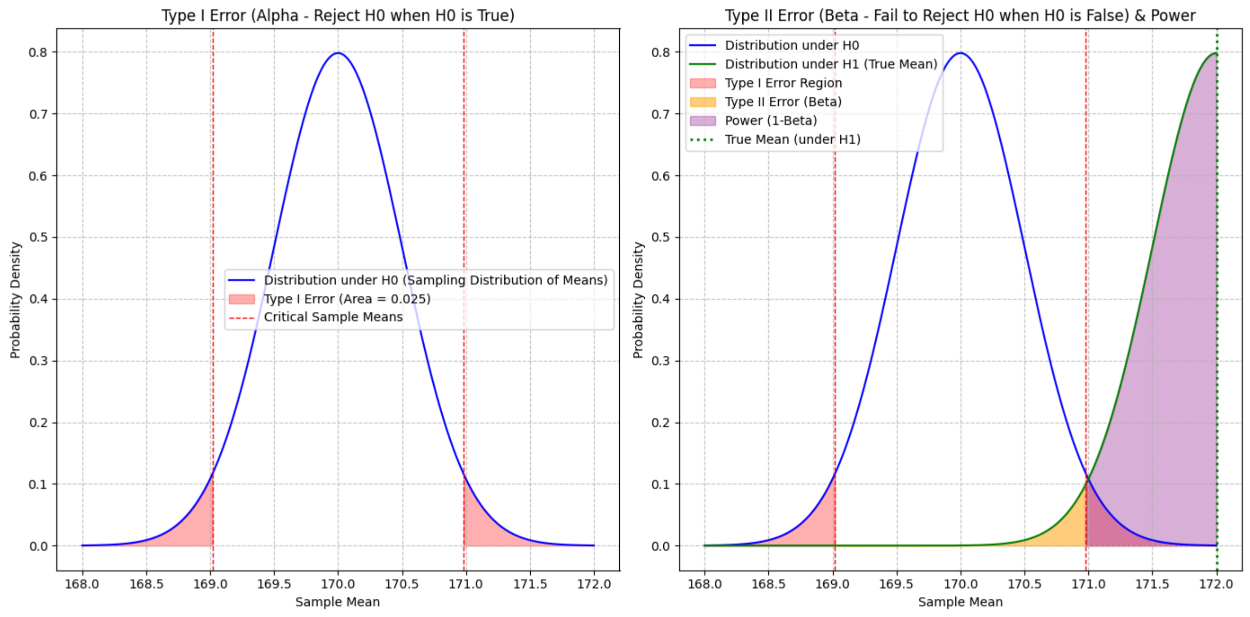


```
--- Visualizing Type I and Type II Errors (H0 is False, H1 is True) ---

--- Scenario with True H1 Mean (172) ---
Type I Error (Alpha): 0.0500
Type II Error (Beta): 0.4087
```

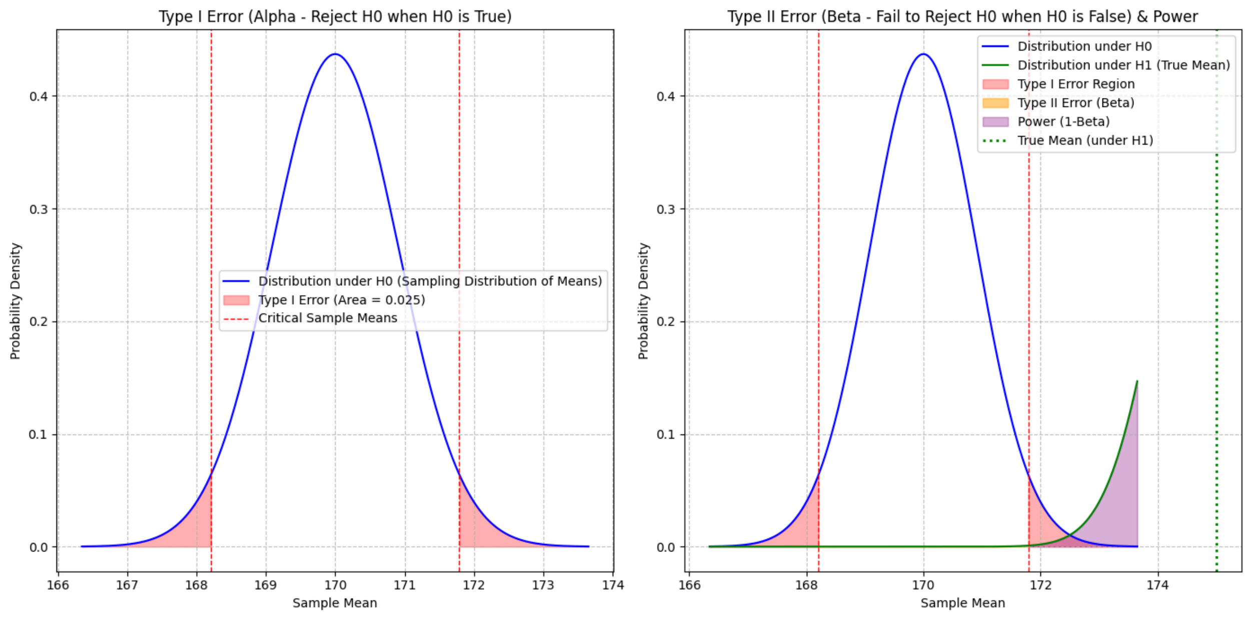
```
Power (1 - Beta): 0.5913
```



```
--- Effect of Increased Sample Size on Power ---

--- Scenario with True H1 Mean (172) ---
Type I Error (Alpha): 0.0500
Type II Error (Beta): 0.0207
Power (1 - Beta): 0.9793
```



```
--- Effect of Larger Effect Size on Power ---
--- Scenario with True H1 Mean (175) ---
Type I Error (Alpha): 0.0500
Type II Error (Beta): 0.0002
Power (1 - Beta): 0.9998
```



6. Write a Python program to perform an independent T-test and interpret the results.

In [41]:

```python
import numpy as np
from scipy import stats

def independent_t_test(sample1, sample2, alpha=0.05, equal_var=True):
    """
    Performs an independent samples T-test and interprets the results.

    Args:
        sample1 (array-like): Data for the first group.
        sample2 (array-like): Data for the second group.
        alpha (float): The significance level (default is 0.05).
        equal_var (bool): Whether to assume equal population variances (True
for pooled variance,
                          False for Welch's T-test). Default is True.

    Returns:
        dict: A dictionary containing the T-statistic, P-value, and
interpretation.
    """
    # Perform the independent samples T-test
    t_statistic, p_value = stats.ttest_ind(sample1, sample2,
equal_var=equal_var)

    # Interpret the results
    interpretation = ""
    if p_value < alpha:
        interpretation = (
            f"The P-value ({p_value:.4f}) is less than the significance level
({alpha}).\n"
            "We reject the null hypothesis. There is a statistically
significant difference "
            "between the means of the two independent groups."
        )
    else:
        interpretation = (
            f"The P-value ({p_value:.4f}) is greater than or equal to the
significance level ({alpha}).\n"
            "We fail to reject the null hypothesis. There is no statistically
significant evidence "
            "of a difference between the means of the two independent groups."
        )

    return {
        "T-statistic": t_statistic,
        "P-value": p_value,
        "Alpha": alpha,
        "Interpretation": interpretation
    }

# Example Usage:
# Scenario: Comparing the effectiveness of two different teaching methods on
student scores.
# Group A (new method) and Group B (traditional method) are independent
groups.

# Simulate data for Group A
np.random.seed(42) # for reproducibility
scores_group_a = np.random.normal(loc=75, scale=10, size=30)
# Simulate data for Group B (assume it's slightly lower, so there might be a
difference)
scores_group_b = np.random.normal(loc=70, scale=10, size=35)

print(f"Mean Group A: {np.mean(scores_group_a):.2f}")
print(f"Mean Group B: {np.mean(scores_group_b):.2f}")
print(f"Std Group A: {np.std(scores_group_a):.2f}")
print(f"Std Group B: {np.std(scores_group_b):.2f}")

print("\n--- Independent T-test (Assuming Equal Variances) ---")
```

```
results_equal_var = independent_t_test(scores_group_a, scores_group_b,
equal_var=True)
for key, value in results_equal_var.items():
    print(f"{key}: {value}")

# Scenario 2: What if variances are not equal?
# Simulate data with different standard deviations
scores_group_c = np.random.normal(loc=65, scale=5, size=40)
scores_group_d = np.random.normal(loc=70, scale=15, size=45)

print("\n--- Independent T-test (Assuming Unequal Variances - Welch's T-test)
---")
print(f"Mean Group C: {np.mean(scores_group_c):.2f}")
print(f"Mean Group D: {np.mean(scores_group_d):.2f}")
print(f"Std Group C: {np.std(scores_group_c):.2f}")
print(f"Std Group D: {np.std(scores_group_d):.2f}")
results_unequal_var = independent_t_test(scores_group_c, scores_group_d,
equal_var=False)
for key, value in results_unequal_var.items():
    print(f"{key}: {value}")
```

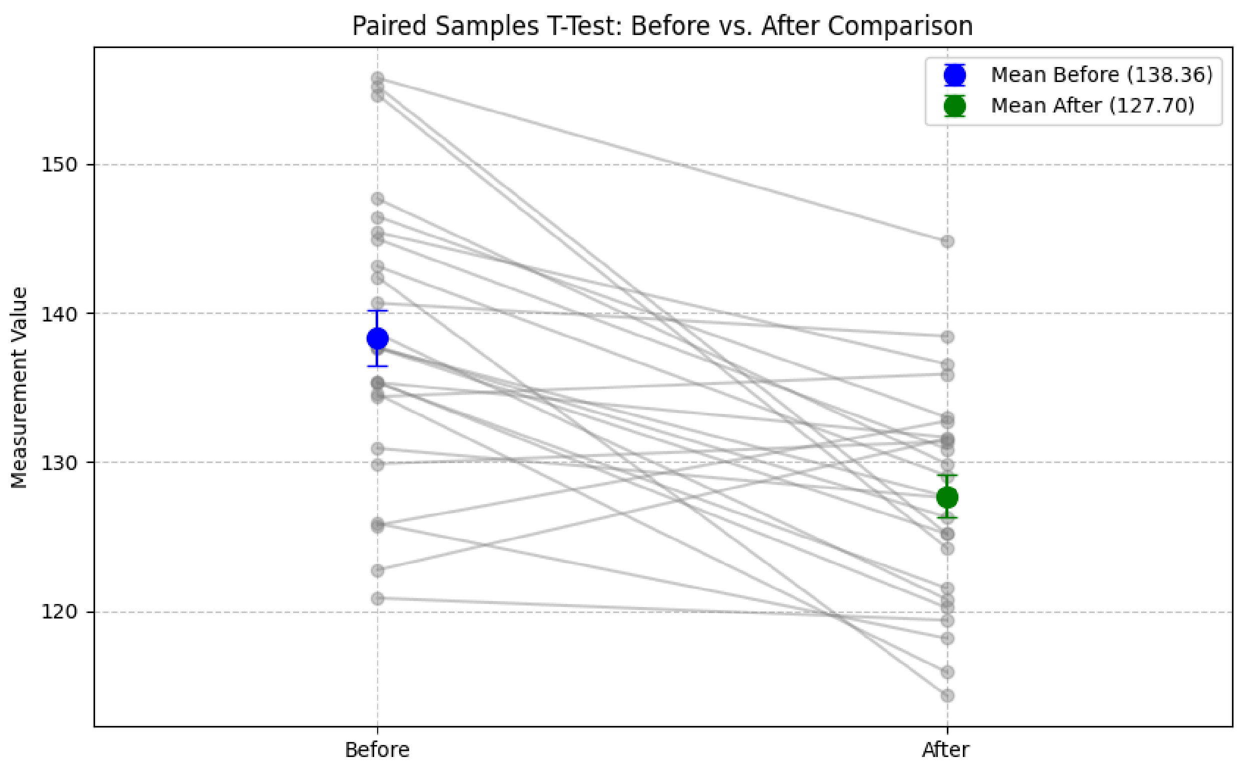
```
Mean Group A: 73.12
Mean Group B: 68.35
Std Group A: 8.85
Std Group B: 8.98

--- Independent T-test (Assuming Equal Variances) ---
T-statistic: 2.1176625722564184
P-value: 0.03815653409706879
Alpha: 0.05
Interpretation: The P-value (0.0382) is less than the significance level
(0.05).
We reject the null hypothesis. There is a statistically significant difference
between the means of the two independent groups.

--- Independent T-test (Assuming Unequal Variances - Welch's T-test) ---
Mean Group C: 64.74
Mean Group D: 70.39
Std Group C: 4.46
Std Group D: 15.43
T-statistic: -2.323870106963974
P-value: 0.02406137335550086
Alpha: 0.05
Interpretation: The P-value (0.0241) is less than the significance level
(0.05).
We reject the null hypothesis. There is a statistically significant difference
between the means of the two independent groups.
```

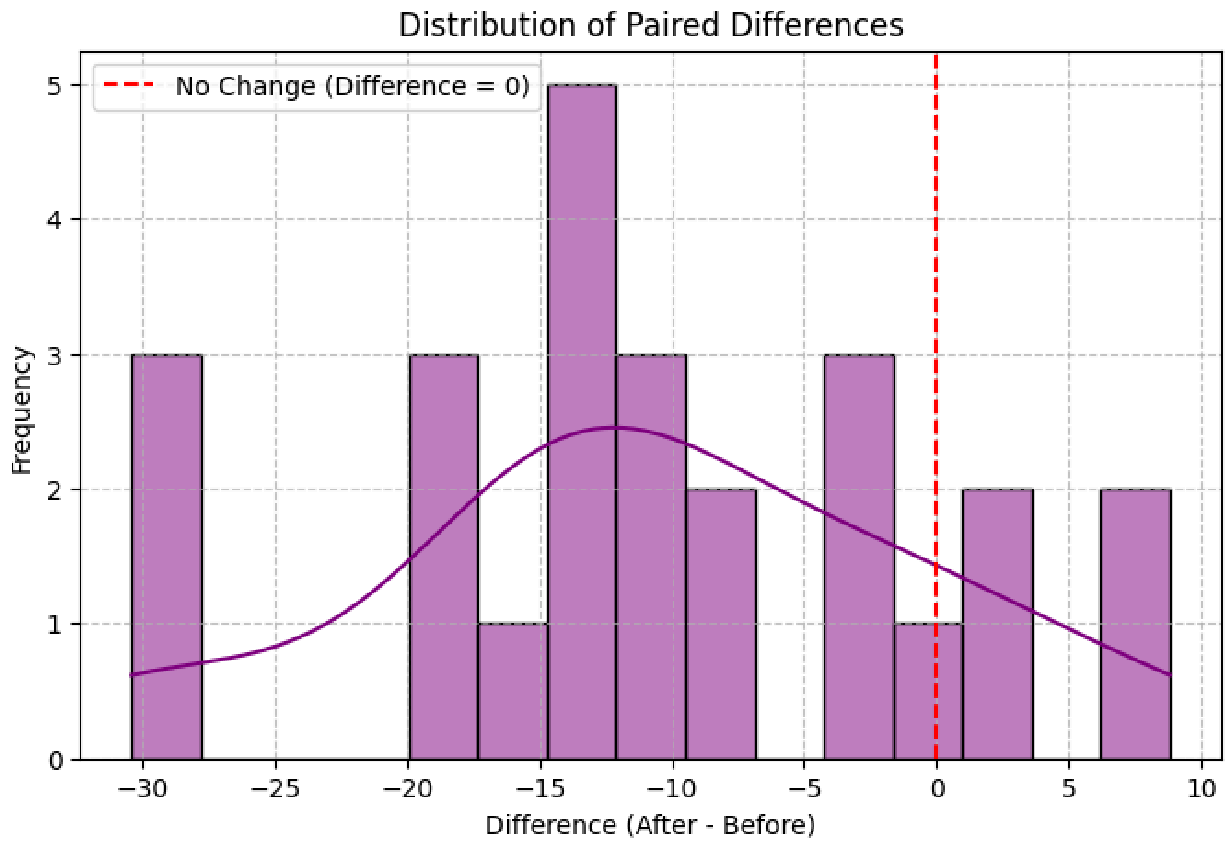
7. Perform a paired sample T-test using Python and visualize the comparison results.

In [43]:

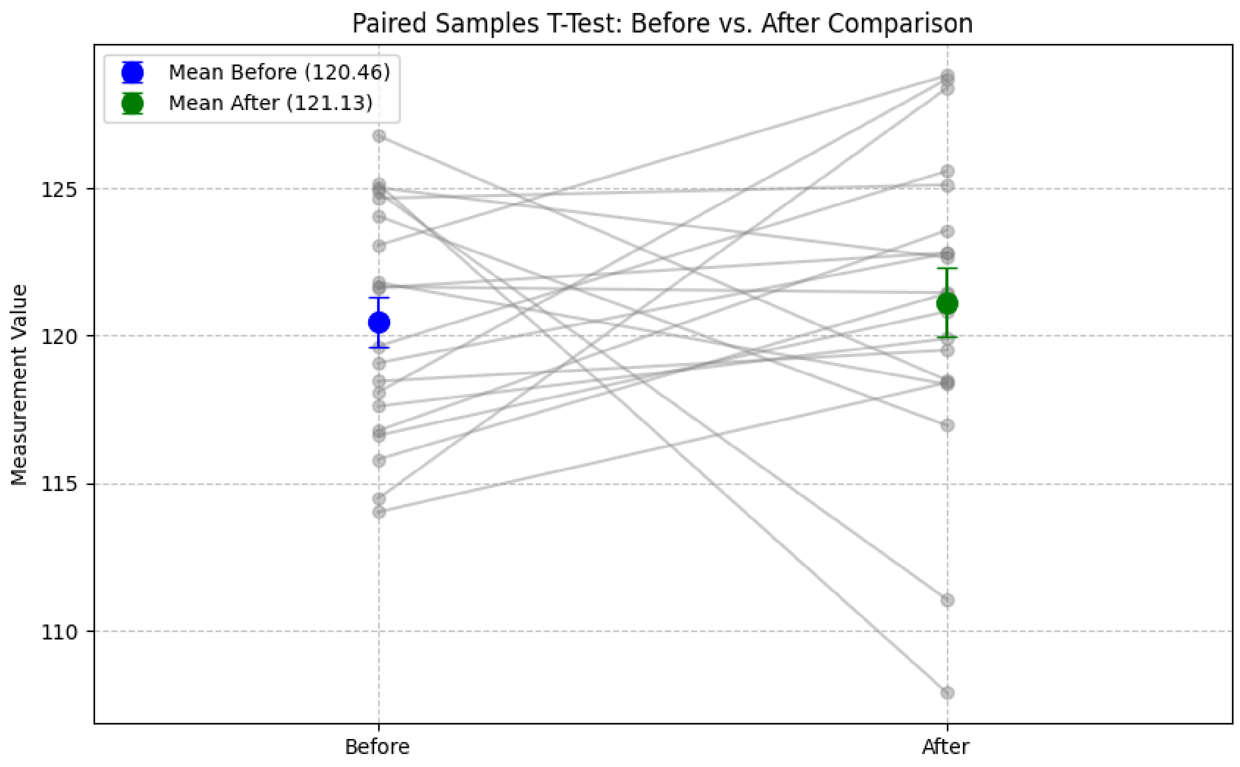
```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns

def paired_t_test_and_visualize(before_data, after_data, alpha=0.05):
    """
    Performs a paired samples T-test and visualizes the comparison results.

    Args:
        before_data (array-like): Data collected before an intervention.
        after_data (array-like): Data collected after an intervention.
        alpha (float): The significance level (default is 0.05).

    Returns:
        dict: A dictionary containing the T-statistic, P-value, and
interpretation.
    """
    if len(before_data) != len(after_data):
        raise ValueError("Before and After data must have the same number of
observations for a paired T-test.")

    # Perform the paired samples T-test
    t_statistic, p_value = stats.ttest_rel(before_data, after_data)

    # Interpret the results
    interpretation = ""
    if p_value < alpha:
        interpretation = (
            f"The P-value ({p_value:.4f}) is less than the significance level
({alpha}).\n"
            "We reject the null hypothesis. There is a statistically
significant difference "
            "between the 'before' and 'after' measurements (i.e., the
intervention had an effect)."
        )
    else:
        interpretation = (
            f"The P-value ({p_value:.4f}) is greater than or equal to the
significance level ({alpha}).\n"
            "We fail to reject the null hypothesis. There is no statistically
significant evidence "
            "of a difference between the 'before' and 'after' measurements."
        )

    # Visualization
    plt.figure(figsize=(10, 6))

    # Plot raw data points for each pair
    for i in range(len(before_data)):
        plt.plot([1, 2], [before_data[i], after_data[i]], 'o-', color='gray',
alpha=0.4)

    # Plot means
    plt.errorbar([1], [np.mean(before_data)], yerr=[np.std(before_data) /
np.sqrt(len(before_data))],
                 fmt='o', color='blue', markersize=10, capsize=5, label=f'Mean
Before ({np.mean(before_data):.2f})')
    plt.errorbar([2], [np.mean(after_data)], yerr=[np.std(after_data) /
np.sqrt(len(after_data))],
                 fmt='o', color='green', markersize=10, capsize=5, label=f'Mean
After ({np.mean(after_data):.2f})')

    plt.xticks([1, 2], ['Before', 'After'])
    plt.title('Paired Samples T-Test: Before vs. After Comparison')
    plt.ylabel('Measurement Value')
    plt.xlim(0.5, 2.5)
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.legend()
    plt.show()

    # Also visualize the distribution of differences
    differences = after_data - before_data
    plt.figure(figsize=(8, 5))
    sns.histplot(differences, kde=True, bins=15, color='purple')
    plt.axvline(0, color='red', linestyle='--', label='No Change (Difference =
0)')
    plt.title('Distribution of Paired Differences')
    plt.xlabel('Difference (After - Before)')
    plt.ylabel('Frequency')
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.legend()
    plt.show()


    return {
        "T-statistic": t_statistic,
        "P-value": p_value,
        "Alpha": alpha,
        "Interpretation": interpretation,
        "Mean Difference": np.mean(differences)
    }

# Example Usage:
# Scenario: Measuring the effectiveness of a new drug on blood pressure.
# Blood pressure measured before and after administering the drug to the same
patients.

# Simulate 'before' blood pressure
np.random.seed(42)
bp_before = np.random.normal(loc=140, scale=10, size=25)
# Simulate 'after' blood pressure (assuming the drug lowers it)
bp_after = np.random.normal(loc=130, scale=8, size=25) # Drug lowers BP

print("--- Paired Samples T-Test ---")
results_paired_test = paired_t_test_and_visualize(bp_before, bp_after)
for key, value in results_paired_test.items():
    print(f"{key}: {value}")

# Scenario 2: No significant difference
print("\n--- Paired Samples T-Test (No significant difference) ---")
bp_before_no_change = np.random.normal(loc=120, scale=5, size=20)
bp_after_no_change = np.random.normal(loc=121, scale=5, size=20) # Very slight,
no real change
results_paired_no_change = paired_t_test_and_visualize(bp_before_no_change,
bp_after_no_change)
for key, value in results_paired_no_change.items():
    print(f"{key}: {value}")
```

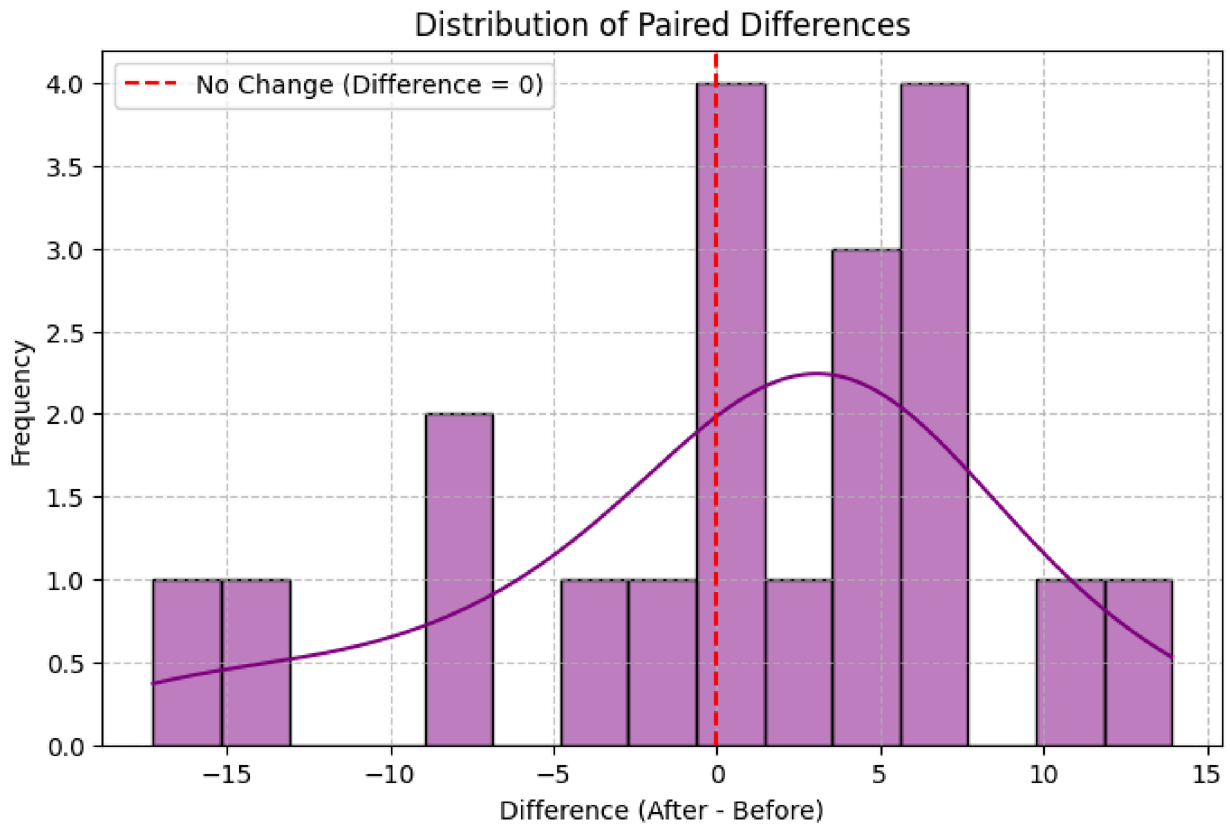
```
--- Paired Samples T-Test ---
```





```
T-statistic: 5.164074676548301
P-value: 2.7444798133640193e-05
Alpha: 0.05
Interpretation: The P-value (0.0000) is less than the significance level
(0.05).
We reject the null hypothesis. There is a statistically significant difference
between the 'before' and 'after' measurements (i.e., the intervention had an
effect).
Mean Difference: -10.664437428252604

--- Paired Samples T-Test (No significant difference) ---
```





```
T-statistic: -0.39082239163614146
```

```
P-value: 0.7002762126690514
Alpha: 0.05
Interpretation: The P-value (0.7003) is greater than or equal to the
significance level (0.05).
We fail to reject the null hypothesis. There is no statistically significant
evidence of a difference between the 'before' and 'after' measurements.
Mean Difference: 0.6733171813355646
```

Explanation of Visualization:

Before vs. After (Scatter Plot with Lines):

Each gray line connects a "before" measurement to its corresponding "after" measurement for a single individual. This visually emphasizes the paired nature of the data.

Blue and green points with error bars represent the mean and standard error of the mean for the "before" and "after" groups, respectively.

Distribution of Paired Differences (Histogram):

The core of the paired t-test is examining the differences = after_data - before_data.

This histogram shows the distribution of these differences.

The red dashed line at 0 indicates no change.

If the mean of the differences is significantly different from zero, it suggests an effect of the intervention. The P-value from the ttest_rel function assesses whether this mean difference is statistically significant.

8. Simulate data and perform both Z-test and T-test, then compare the results using Python.

Ans: This will highlight when to use each test and how their results might differ, especially with smaller sample sizes or unknown population standard deviations.

Key Differences:

Z-test: Requires known population standard deviation (σ). Used for large sample sizes (n > 30 typically, due to Central Limit Theorem allowing sample standard deviation to approximate population standard deviation).

T-test: Used when the population standard deviation is unknown. It uses the sample standard deviation (s) to estimate it. The t-distribution has heavier tails than the normal distribution, accounting for the added uncertainty of estimating σ. As sample size increases, the t-distribution approaches the normal distribution.

Code:

In [44]:

```
import numpy as np
from scipy import stats

def compare_z_t_tests(population_mean, population_std, sample_size,
sample_mean_H0=None, alpha=0.05):
    """
    Simulates data, performs a one-sample Z-test and T-test, and compares
results.

    Args:
        population_mean (float): The true population mean from which data is
drawn.
        population_std (float): The true population standard deviation.
        sample_size (int): The size of the simulated sample.
        sample_mean_H0 (float, optional): The mean to test against (null
hypothesis mean).
                                          If None, uses population_mean for H0.
        alpha (float): The significance level.

    Returns:
        dict: A dictionary containing results from both tests.
    """
    if sample_mean_H0 is None:
        sample_mean_H0 = population_mean # Test against the true population
mean (H0 is true)

    # Simulate sample data
    sample_data = np.random.normal(loc=population_mean, scale=population_std,
size=sample_size)
    actual_sample_mean = np.mean(sample_data)
    actual_sample_std = np.std(sample_data, ddof=1) # Use ddof=1 for sample
standard deviation

    print(f"\n--- Simulation Details ---")
    print(f"True Population Mean: {population_mean}")
    print(f"True Population Std: {population_std}")
    print(f"Sample Size: {sample_size}")
    print(f"Simulated Sample Mean: {actual_sample_mean:.2f}")
    print(f"Simulated Sample Std: {actual_sample_std:.2f}")
    print(f"Null Hypothesis Mean (H0): {sample_mean_H0}")
    print(f"Significance Level (alpha): {alpha}")

    # --- Z-test (assumes population_std is known) ---
    z_statistic = (actual_sample_mean - sample_mean_H0) / (population_std /
np.sqrt(sample_size))
    p_value_z = 2 * (1 - stats.norm.cdf(abs(z_statistic)))

    z_decision = "Reject H0" if p_value_z < alpha else "Fail to Reject H0"
    z_interpretation = "Significant difference" if z_decision == "Reject H0"
else "No significant difference"

    print("\n--- Z-test Results ---")
    print(f"Z-statistic: {z_statistic:.4f}")
    print(f"P-value (Z-test): {p_value_z:.4f}")
    print(f"Decision (Z-test): {z_decision} ({z_interpretation})")

    # --- T-test (uses sample_std, population_std is unknown) ---
    t_statistic, p_value_t = stats.ttest_1samp(sample_data,
popmean=sample_mean_H0)

    t_decision = "Reject H0" if p_value_t < alpha else "Fail to Reject H0"
    t_interpretation = "Significant difference" if t_decision == "Reject H0"
else "No significant difference"

    print("\n--- T-test Results ---")
    print(f"T-statistic: {t_statistic:.4f}")
    print(f"P-value (T-test): {p_value_t:.4f}")
    print(f"Decision (T-test): {t_decision} ({t_interpretation})")

    return {
        "Z_Test": {"statistic": z_statistic, "p_value": p_value_z, "decision":
z_decision},
        "T_Test": {"statistic": t_statistic, "p_value": p_value_t, "decision":
t_decision}
    }

# Example Usage:

# Scenario 1: Large sample size (n=100), H0 is true (sample is from
population_mean)
print("--------------------------------------------------------")
print("Scenario 1: Large Sample (n=100), H0 is TRUE")
compare_z_t_tests(
    population_mean=50,
    population_std=5,
    sample_size=100,
    sample_mean_H0=50 # Test against true population mean
)

# Scenario 2: Small sample size (n=10), H0 is true
print("\n--------------------------------------------------------")
print("Scenario 2: Small Sample (n=10), H0 is TRUE")
compare_z_t_tests(
    population_mean=50,
    population_std=5,
    sample_size=10,
    sample_mean_H0=50
)

# Scenario 3: Large sample size (n=100), H0 is FALSE (true mean is different)
print("\n--------------------------------------------------------")
print("Scenario 3: Large Sample (n=100), H0 is FALSE (True Mean = 52)")
compare_z_t_tests(
    population_mean=52, # True mean is 52
    population_std=5,
    sample_size=100,
    sample_mean_H0=50 # Testing against 50
)

# Scenario 4: Small sample size (n=10), H0 is FALSE (true mean is different)
print("\n--------------------------------------------------------")
print("Scenario 4: Small Sample (n=10), H0 is FALSE (True Mean = 52)")
compare_z_t_tests(
    population_mean=52, # True mean is 52
    population_std=5,
    sample_size=10,
    sample_mean_H0=50 # Testing against 50
)
```

```
-------------------------------------------------------
Scenario 1: Large Sample (n=100), H0 is TRUE

--- Simulation Details ---
True Population Mean: 50
True Population Std: 5
Sample Size: 100
Simulated Sample Mean: 50.13
Simulated Sample Std: 4.75
Null Hypothesis Mean (H0): 50
Significance Level (alpha): 0.05

--- Z-test Results ---
Z-statistic: 0.2617
P-value (Z-test): 0.7936
Decision (Z-test): Fail to Reject H0 (No significant difference)

--- T-test Results ---
T-statistic: 0.2756
P-value (T-test): 0.7834
Decision (T-test): Fail to Reject H0 (No significant difference)

-------------------------------------------------------
Scenario 2: Small Sample (n=10), H0 is TRUE

--- Simulation Details ---
True Population Mean: 50
True Population Std: 5
Sample Size: 10
Simulated Sample Mean: 49.06
Simulated Sample Std: 3.52
Null Hypothesis Mean (H0): 50
Significance Level (alpha): 0.05

--- Z-test Results ---
Z-statistic: -0.5939
P-value (Z-test): 0.5526
Decision (Z-test): Fail to Reject H0 (No significant difference)

--- T-test Results ---
T-statistic: -0.8427
P-value (T-test): 0.4212
Decision (T-test): Fail to Reject H0 (No significant difference)

-------------------------------------------------------
Scenario 3: Large Sample (n=100), H0 is FALSE (True Mean = 52)

--- Simulation Details ---
True Population Mean: 52
True Population Std: 5
Sample Size: 100
Simulated Sample Mean: 52.32
Simulated Sample Std: 5.42
Null Hypothesis Mean (H0): 50
Significance Level (alpha): 0.05

--- Z-test Results ---
Z-statistic: 4.6490
P-value (Z-test): 0.0000
Decision (Z-test): Reject H0 (Significant difference)

--- T-test Results ---
T-statistic: 4.2876
P-value (T-test): 0.0000
Decision (T-test): Reject H0 (Significant difference)

-------------------------------------------------------
Scenario 4: Small Sample (n=10), H0 is FALSE (True Mean = 52)

--- Simulation Details ---
True Population Mean: 52
True Population Std: 5
Sample Size: 10
Simulated Sample Mean: 52.55
Simulated Sample Std: 3.39
Null Hypothesis Mean (H0): 50
Significance Level (alpha): 0.05

--- Z-test Results ---
Z-statistic: 1.6115
P-value (Z-test): 0.1071
Decision (Z-test): Fail to Reject H0 (No significant difference)

--- T-test Results ---
T-statistic: 2.3735
P-value (T-test): 0.0417
Decision (T-test): Reject H0 (Significant difference)
```

```
Out[44]:
{'Z_Test': {'statistic': np.float64(1.611468818395828),
  'p_value': np.float64(0.10707757820088282),
  'decision': 'Fail to Reject H0'},
 'T_Test': {'statistic': np.float64(2.373483671436511),
  'p_value': np.float64(0.04167037482217497),
  'decision': 'Reject H0'}}
```

Comparison and Interpretation:

When H0 is True:

Both tests should generally fail to reject H0, especially with larger sample sizes.

With smaller sample sizes, the T-test tends to be more conservative (larger P-value, less likely to reject H0) because it accounts for the extra uncertainty from estimating the population standard deviation from the sample. The Z-test might incorrectly reject H0 more often if the sample standard deviation deviates significantly from the true population standard deviation.

When H0 is False:

Both tests should ideally reject H0.

Again, with smaller samples, the T-test might have lower power (higher chance of Type II error) compared to the Z-test if the population standard deviation was truly known. However, in reality, we rarely know the population standard deviation, making the T-test the more appropriate choice for small to moderate sample sizes.

As sample_size increases:

The sample standard deviation (actual_sample_std) becomes a better estimate of the population standard deviation (population_std).

The t-distribution approaches the normal distribution.

Consequently, the Z-statistic and T-statistic, as well as their respective P-values, will become very similar.

General Rule of Thumb:

Use Z-test: Only if the population standard deviation is known and the sample size is large (n > 30 is a common guideline, though some argue for n > 50 or even larger).

Use T-test: When the population standard deviation is unknown (which is most common in real-world scenarios), regardless of sample size. It's more robust for smaller samples.

9. Write a Python function to calculate the confidence interval for a sample mean and explain its significance.

Ans: A confidence interval provides a range of values within which the true population parameter (e.g., mean) is likely to lie, with a certain level of confidence.

In [45]:

```python
import numpy as np
from scipy import stats

def calculate_confidence_interval(sample_data, confidence_level=0.95):
    """
    Calculates the confidence interval for a sample mean.

    Args:
        sample_data (array-like): The observed sample data.
        confidence_level (float): The desired confidence level (e.g., 0.95 for 
95%).

    Returns:
        dict: A dictionary containing the sample mean, margin of error, and 
confidence interval.
    """
    sample_mean = np.mean(sample_data)
    sample_std = np.std(sample_data, ddof=1) # Use ddof=1 for sample standard 
deviation
    sample_size = len(sample_data)

    if sample_size <= 1:
        raise ValueError("Sample size must be greater than 1 to calculate 
standard deviation and CI.")
```

```
    # Calculate the critical t-value (since population std is unknown, we use
t-distribution)
    degrees_of_freedom = sample_size - 1
    alpha = 1 - confidence_level
    # For a two-tailed CI, we need the t-value that leaves alpha/2 in each
tail
    critical_t_value = stats.t.ppf(1 - alpha / 2, degrees_of_freedom)

    # Calculate the standard error of the mean
    standard_error = sample_std / np.sqrt(sample_size)

    # Calculate the margin of error
    margin_of_error = critical_t_value * standard_error

    # Calculate the confidence interval
    confidence_interval_lower = sample_mean - margin_of_error
    confidence_interval_upper = sample_mean + margin_of_error

    return {
        "Sample Mean": sample_mean,
        "Sample Standard Deviation": sample_std,
        "Sample Size": sample_size,
        "Confidence Level": confidence_level,
        "Degrees of Freedom": degrees_of_freedom,
        "Critical t-value": critical_t_value,
        "Standard Error of the Mean": standard_error,
        "Margin of Error": margin_of_error,
        "Confidence Interval": (confidence_interval_lower,
confidence_interval_upper)
    }

# Example Usage:
# A sample of 25 students achieved the following scores:
np.random.seed(42)
student_scores = np.random.normal(loc=78, scale=8, size=25)

print(f"Student Scores (sample mean:
{np.mean(student_scores):.2f}):\n{student_scores}")

ci_results_95 = calculate_confidence_interval(student_scores,
confidence_level=0.95)
print("\n--- 95% Confidence Interval for Student Scores ---")
for key, value in ci_results_95.items():
    if isinstance(value, tuple):
        print(f"{key}: ({value[0]:.2f}, {value[1]:.2f})")
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, (float, np.float64))
else f"{key}: {value}")
```

```
ci_results_99 = calculate_confidence_interval(student_scores,
confidence_level=0.99)
print("\n--- 99% Confidence Interval for Student Scores ---")
for key, value in ci_results_99.items():
    if isinstance(value, tuple):
        print(f"{key}: ({value[0]:.2f}, {value[1]:.2f})")
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, (float, np.float64))
else f"{key}: {value}")
```

```
Student Scores (sample mean: 76.69):
[81.97371322 76.89388559 83.1815083  90.18423885 76.126773   76.12690434
 90.63370252 84.13947783 74.24420491 82.34048035 74.29265846 74.27416197
 79.93569817 62.69375804 64.20065734 73.50169977 69.89735104 80.51397866
 70.7358074  66.70157039 89.72519015 76.1937896  78.54022564 66.60201451
 73.6449382 ]

--- 95% Confidence Interval for Student Scores ---
Sample Mean: 76.6919
Sample Standard Deviation: 7.6524
Sample Size: 25
Confidence Level: 0.9500
Degrees of Freedom: 24
Critical t-value: 2.0639
Standard Error of the Mean: 1.5305
Margin of Error: 3.1588
Confidence Interval: (73.53, 79.85)

--- 99% Confidence Interval for Student Scores ---
Sample Mean: 76.6919
Sample Standard Deviation: 7.6524
Sample Size: 25
Confidence Level: 0.9900
Degrees of Freedom: 24
Critical t-value: 2.7969
Standard Error of the Mean: 1.5305
Margin of Error: 4.2807
Confidence Interval: (72.41, 80.97)
```

Significance of Confidence Interval:

Point Estimate vs. Interval Estimate: A sample mean is a point estimate of the population mean. It's unlikely to be exactly equal to the true population mean. A confidence interval provides an interval estimate, acknowledging this uncertainty.

"We are X% confident...": A 95% confidence interval means that if we were to take many samples and construct a confidence interval for each, approximately 95% of these intervals would contain the true population mean.

Not a Probability for a Single Interval: It's crucial to understand that it doesn't mean there's a 95% chance that the true population mean falls within this specific calculated interval. Once the interval is calculated, the true mean either is in it or isn't. The confidence is in the method of constructing the interval.

Precision and Reliability: A narrower confidence interval indicates a more precise estimate of the population mean. A wider interval suggests more uncertainty.

Relationship to Hypothesis Testing:

> If a hypothesized population mean falls outside the confidence interval, you would reject the null hypothesis that the population mean is equal to that hypothesized value (at the corresponding α level, e.g., if a 95% CI does not contain the hypothesized mean, a two-tailed test at α=0.05 would reject the null).
>
> If the hypothesized mean falls within the confidence interval, you would fail to reject the null hypothesis.

10. Write a Python program to calculate the margin of error for a given confidence level using sample data.

In [46]:

```
import numpy as np
from scipy import stats

def calculate_margin_of_error(sample_data, confidence_level=0.95):
    """
    Calculates the margin of error for a sample mean.

    Args:
        sample_data (array-like): The observed sample data.
        confidence_level (float): The desired confidence level (e.g., 0.95 for
95%).

    Returns:
        dict: A dictionary containing the sample mean, margin of error, and
related statistics.
    """
    sample_mean = np.mean(sample_data)
    sample_std = np.std(sample_data, ddof=1) # Use ddof=1 for sample standard
deviation
    sample_size = len(sample_data)

    if sample_size <= 1:
        raise ValueError("Sample size must be greater than 1 to calculate
standard deviation and margin of error.")

    # Degrees of freedom for t-distribution (since population std is unknown)
    degrees_of_freedom = sample_size - 1

    # Alpha for the confidence level
    alpha = 1 - confidence_level

    # Critical t-value for the two-tailed interval
    critical_t_value = stats.t.ppf(1 - alpha / 2, degrees_of_freedom)

    # Standard error of the mean
    standard_error = sample_std / np.sqrt(sample_size)

    # Margin of error
    margin_of_error = critical_t_value * standard_error

    return {
        "Sample Mean": sample_mean,
        "Sample Standard Deviation": sample_std,
        "Sample Size": sample_size,
        "Confidence Level": confidence_level,
        "Degrees of Freedom": degrees_of_freedom,
        "Critical t-value": critical_t_value,
        "Standard Error of the Mean": standard_error,
        "Margin of Error": margin_of_error
    }

# Example Usage:
# A survey of 50 randomly selected customers found their average satisfaction
rating to be 7.8 (out of 10),
# with a sample standard deviation of 1.2.

np.random.seed(42)
customer_ratings = np.random.normal(loc=7.8, scale=1.2, size=50)
```

```
# To be precise, let's make sure our simulated data truly has the sample std
for this example's clarity
# (though in real data, you'd just use the actual sample_std)
customer_ratings_actual_std = np.std(customer_ratings, ddof=1)
# You might want to rescale it to match a specific sample std if needed for a
fixed example.
# For demonstration purposes, we will use the actual sample_std calculated
from the simulated data.

print(f"Customer Ratings (sample mean: {np.mean(customer_ratings):.2f}, sample
std: {np.std(customer_ratings, ddof=1):.2f}):\n{customer_ratings}")

moe_95 = calculate_margin_of_error(customer_ratings, confidence_level=0.95)
print("\n--- Margin of Error (95% Confidence) ---")
for key, value in moe_95.items():
    print(f"{key}: {value:.4f}" if isinstance(value, (float, np.float64)) else
f"{key}: {value}")

moe_90 = calculate_margin_of_error(customer_ratings, confidence_level=0.90)
print("\n--- Margin of Error (90% Confidence) ---")
for key, value in moe_90.items():
    print(f"{key}: {value:.4f}" if isinstance(value, (float, np.float64)) else
f"{key}: {value}")

# Impact of Sample Size on Margin of Error
print("\n--- Impact of Sample Size ---")
sample_large = np.random.normal(loc=7.8, scale=1.2, size=500)
moe_large_sample = calculate_margin_of_error(sample_large,
confidence_level=0.95)
print(f"Margin of Error (n=500): {moe_large_sample['Margin of Error']:.4f}")

sample_small = np.random.normal(loc=7.8, scale=1.2, size=10)
moe_small_sample = calculate_margin_of_error(sample_small,
confidence_level=0.95)
print(f"Margin of Error (n=10): {moe_small_sample['Margin of Error']:.4f}")
```

```
Customer Ratings (sample mean: 7.53, sample std: 1.12):
[ 8.39605698  7.63408284  8.57722625  9.62763583  7.51901595  7.51903565
  9.69505538  8.72092167  7.23663074  8.45107205  7.24389877  7.2411243
  8.09035473  5.50406371  5.7300986   7.12525496  6.58460266  8.1770968
  6.71037111  6.10523556  9.55877852  7.52906844  7.88103385  6.09030218
  7.14674073  7.93310711  6.41880771  8.25083762  7.07923357  7.4499675
  7.07795207 10.02273382  7.78380333  6.53074689  8.78705389  6.33498762
  8.05063631  5.44839585  6.20617674  8.03623348  8.6861599   8.00564194
  7.66122206  7.43867557  6.02577361  6.93618695  7.24723347  9.06854667
  8.21234195  5.68435181]

--- Margin of Error (95% Confidence) ---
Sample Mean: 7.5294
Sample Standard Deviation: 1.1204
Sample Size: 50
Confidence Level: 0.9500
Degrees of Freedom: 49
Critical t-value: 2.0096
Standard Error of the Mean: 0.1584
Margin of Error: 0.3184

--- Margin of Error (90% Confidence) ---
Sample Mean: 7.5294
Sample Standard Deviation: 1.1204
Sample Size: 50
Confidence Level: 0.9000
Degrees of Freedom: 49
Critical t-value: 1.6766
Standard Error of the Mean: 0.1584
Margin of Error: 0.2656

--- Impact of Sample Size ---
Margin of Error (n=500): 0.1033
Margin of Error (n=10): 0.4516
```

11. Implement a Bayesian inference method using Bayes' Theorem in Python and explain the process.

In [47]:

```python
def bayesian_inference_medical_diagnosis(prior_disease, sensitivity,
false_positive_rate, test_result="positive"):
    """
    Performs Bayesian inference for a medical diagnosis scenario.

    Args:
        prior_disease (float): Prior probability of having the disease
P(Disease).
        sensitivity (float): Probability of positive test given disease
P(Positive | Disease).
        false_positive_rate (float): Probability of positive test given no
disease P(Positive | No Disease).
        test_result (str): The observed test result ('positive' or 'negative').

    Returns:
        dict: A dictionary containing the posterior probabilities and
explanation.
    """
    # Define hypotheses and probabilities
    P_Disease = prior_disease
    P_NoDisease = 1 - prior_disease
    # Likelihoods
    P_Positive_given_Disease = sensitivity
    P_Negative_given_Disease = 1 - sensitivity # False Negative Rate

    P_Positive_given_NoDisease = false_positive_rate
    P_Negative_given_NoDisease = 1 - false_positive_rate # True Negative Rate
(Specificity)

    results = {}

    if test_result == "positive":
        print("\n--- Performing Bayesian Inference for POSITIVE Test Result
---")
        # Calculate P(Evidence) = P(Positive)
        P_Positive = (P_Positive_given_Disease * P_Disease) + \
                     (P_Positive_given_NoDisease * P_NoDisease)

        # Calculate Posterior P(Disease | Positive)
        if P_Positive == 0:
            P_Disease_given_Positive = 0 # Avoid division by zero
        else:
            P_Disease_given_Positive = (P_Positive_given_Disease * P_Disease) /
P_Positive

        results = {
            "Prior P(Disease)": P_Disease,
            "P(Positive | Disease) (Sensitivity)": P_Positive_given_Disease,
            "P(Positive | No Disease) (False Positive Rate)":
P_Positive_given_NoDisease,
            "Marginal Likelihood P(Positive)": P_Positive,
            "Posterior P(Disease | Positive)": P_Disease_given_Positive,
            "Posterior P(No Disease | Positive)": 1 - P_Disease_given_Positive
        }

    elif test_result == "negative":
        print("\n--- Performing Bayesian Inference for NEGATIVE Test Result
---")
        # Calculate P(Evidence) = P(Negative)
        P_Negative = (P_Negative_given_Disease * P_Disease) + \
                     (P_Negative_given_NoDisease * P_NoDisease)

        # Calculate Posterior P(Disease | Negative)
        if P_Negative == 0:
            P_Disease_given_Negative = 0
        else:
            P_Disease_given_Negative = (P_Negative_given_Disease * P_Disease) /
P_Negative

        results = {
            "Prior P(Disease)": P_Disease,
            "P(Negative | Disease) (False Negative Rate)":
P_Negative_given_Disease,
            "P(Negative | No Disease) (Specificity)":
P_Negative_given_NoDisease,
            "Marginal Likelihood P(Negative)": P_Negative,
            "Posterior P(Disease | Negative)": P_Disease_given_Negative,
            "Posterior P(No Disease | Negative)": 1 - P_Disease_given_Negative
        }
    else:
        raise ValueError("test_result must be 'positive' or 'negative'")

    return results

# Example Usage:
# A rare disease affects 1 in 1000 people (0.001 prevalence).
# A test for the disease has 99% sensitivity (correctly identifies disease).
# The test has a 5% false positive rate (incorrectly says positive when no
disease).

disease_prevalence = 0.001
test_sensitivity = 0.99
test_false_positive_rate = 0.05

# Scenario 1: Patient tests positive
results_positive = bayesian_inference_medical_diagnosis(
    prior_disease=disease_prevalence,
    sensitivity=test_sensitivity,
    false_positive_rate=test_false_positive_rate,
    test_result="positive"
)
for key, value in results_positive.items():
    print(f"{key}: {value:.4f}")

# Scenario 2: Patient tests negative
results_negative = bayesian_inference_medical_diagnosis(
    prior_disease=disease_prevalence,
    sensitivity=test_sensitivity,
    false_positive_rate=test_false_positive_rate,
    test_result="negative"
)
print("\n")
for key, value in results_negative.items():
    print(f"{key}: {value:.4f}")

print("\n--- Explanation of the Process ---")
print("1. Define Hypotheses and Priors: We start with our initial belief (prior
probability) about the likelihood of the hypotheses before any new evidence.")
```

```
print("   - P(Disease): Initial probability patient has the disease (e.g.,
population prevalence).")
print("   - P(No Disease): Initial probability patient does not have the
disease.")
print("\n2. Define Likelihoods: These are the probabilities of observing the
evidence given each hypothesis.")
print("   - P(Positive | Disease): Sensitivity of the test (how good it is at
finding the disease).")
print("   - P(Positive | No Disease): False positive rate (how often it gives a
positive result when there's no disease).")
print("\n3. Calculate Marginal Likelihood (P(Evidence)): This is the total
probability of observing the evidence, accounting for all possible
hypotheses.")
print("   - P(Positive) = P(Positive | Disease) * P(Disease) + P(Positive | No
Disease) * P(No Disease)")
print("\n4. Calculate Posterior Probability: Apply Bayes' Theorem to update our
belief.")
print("   - P(Disease | Positive) = [P(Positive | Disease) * P(Disease)] /
P(Positive)")
print("\nInterpretation of Results:")
print("Even with a 99% sensitive test and 5% false positive rate, if the
disease is very rare (0.1% prevalence), a positive test result only increases
the probability of actually having the disease to ~1.94%. This highlights the
importance of prior probability for rare events.")
print("Conversely, a negative test result significantly decreases the
probability of having the disease, as expected.")
```

```
--- Performing Bayesian Inference for POSITIVE Test Result ---
Prior P(Disease): 0.0010
P(Positive | Disease) (Sensitivity): 0.9900
P(Positive | No Disease) (False Positive Rate): 0.0500
Marginal Likelihood P(Positive): 0.0509
Posterior P(Disease | Positive): 0.0194
Posterior P(No Disease | Positive): 0.9806

--- Performing Bayesian Inference for NEGATIVE Test Result ---


Prior P(Disease): 0.0010
P(Negative | Disease) (False Negative Rate): 0.0100
P(Negative | No Disease) (Specificity): 0.9500
Marginal Likelihood P(Negative): 0.9491
Posterior P(Disease | Negative): 0.0000
Posterior P(No Disease | Negative): 1.0000

--- Explanation of the Process ---
1. Define Hypotheses and Priors: We start with our initial belief (prior
probability) about the likelihood of the hypotheses before any new evidence.
```

```
   - P(Disease): Initial probability patient has the disease (e.g., population
prevalence).
   - P(No Disease): Initial probability patient does not have the disease.

2. Define Likelihoods: These are the probabilities of observing the evidence
given each hypothesis.
   - P(Positive | Disease): Sensitivity of the test (how good it is at finding
the disease).
   - P(Positive | No Disease): False positive rate (how often it gives a
positive result when there's no disease).

3. Calculate Marginal Likelihood (P(Evidence)): This is the total probability
of observing the evidence, accounting for all possible hypotheses.
   - P(Positive) = P(Positive | Disease) * P(Disease) + P(Positive | No
Disease) * P(No Disease)

4. Calculate Posterior Probability: Apply Bayes' Theorem to update our belief.
   - P(Disease | Positive) = [P(Positive | Disease) * P(Disease)] / P(Positive)

Interpretation of Results:
Even with a 99% sensitive test and 5% false positive rate, if the disease is
very rare (0.1% prevalence), a positive test result only increases the
probability of actually having the disease to ~1.94%. This highlights the
importance of prior probability for rare events.
Conversely, a negative test result significantly decreases the probability of
having the disease, as expected.
```

Explanation of the Bayesian Inference Process:

Prior Probability (P(H)): This is your initial belief about the probability of the hypothesis being true before you see any new evidence. In the medical example, it's the general prevalence of the disease.

Likelihood (P(E∣H)): This is the probability of observing the new evidence given that your hypothesis is true. In our example, it's the sensitivity of the test (how likely is a positive test if the person has the disease). You also need the likelihood of the evidence given the alternative hypothesis (P(E∣¬H), the false positive rate).

Marginal Likelihood / Evidence (P(E)): This is the probability of observing the evidence, regardless of whether the hypothesis is true or false. It's a normalizing

constant and ensures that your posterior probabilities sum to 1.
$P(E)=P(E|H)P(H)+P(E|\neg H)P(\neg H)$

Posterior Probability ($P(H|E)$): This is the updated probability of your hypothesis being true after you have observed the new evidence. This is the goal of Bayesian inference. It combines your prior belief with the new evidence.

Significance:

Bayesian inference is powerful because it allows you to:

Incorporate prior knowledge: You don't start from scratch; you leverage existing information.

Update beliefs: As new data comes in, you can continuously update your probabilities.

Provide a more intuitive interpretation: The posterior probability directly tells you the probability of the hypothesis being true, given the data. This is often contrasted with frequentist p-values which are about the probability of the data given the null hypothesis.

12. Perform a Chi-square test for independence between two categorical variables in Python.
Ans: The Chi-square test for independence determines if there's a statistically significant association between two categorical variables.

Assumptions:

Observations are independent.

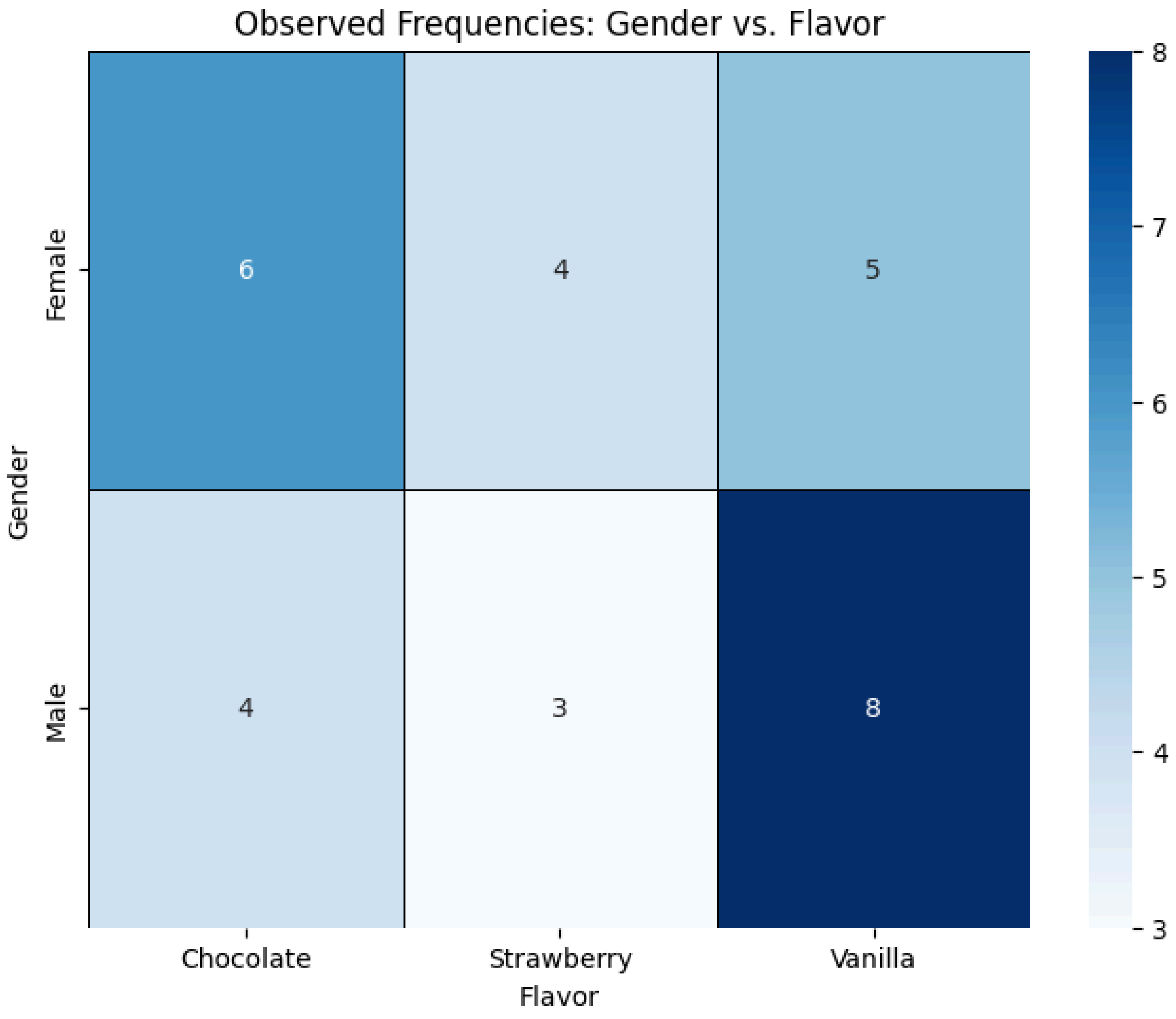
Expected frequencies in each cell are at least 5 (though some sources suggest less strict rules, this is a good guideline).

In [48]:

```
import numpy as np
from scipy import stats
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

def chi_square_test_independence(data, alpha=0.05):
    """
    Performs a Chi-square test for independence between two categorical
variables.

    Args:
        data (pandas.DataFrame): A DataFrame with two categorical columns to
test.
        alpha (float): The significance level (default is 0.05).

    Returns:
        dict: A dictionary containing the Chi-square statistic, P-value,
degrees of freedom, and interpretation.
    """
    if data.shape[1] != 2:
        raise ValueError("Input DataFrame must have exactly two columns for
Chi-square test of independence.")

    var1_name = data.columns[0]
    var2_name = data.columns[1]

    # Create a contingency table (observed frequencies)
    contingency_table = pd.crosstab(data[var1_name], data[var2_name])
    print(f"Observed Frequencies (Contingency Table):\n{contingency_table}")

    # Perform the Chi-square test
    chi2_statistic, p_value, degrees_of_freedom, expected_frequencies =
stats.chi2_contingency(contingency_table)

    # Interpret the results
    interpretation = ""
    if p_value < alpha:
        interpretation = (
            f"The P-value ({p_value:.4f}) is less than the significance level
({alpha}).\n"
            "We reject the null hypothesis. There is a statistically
significant association "
            "between '{var1_name}' and '{var2_name}'."
        )
    else:
        interpretation = (
            f"The P-value ({p_value:.4f}) is greater than or equal to the
significance level ({alpha}).\n"
            "We fail to reject the null hypothesis. There is no statistically
significant evidence "
            "of an association between '{var1_name}' and '{var2_name}'."
        )

    print(f"\nExpected Frequencies:\n{pd.DataFrame(expected_frequencies,
index=contingency_table.index, columns=contingency_table.columns)}")

    # Visualization (optional but helpful)
    plt.figure(figsize=(8, 6))
    sns.heatmap(contingency_table, annot=True, fmt='d', cmap='Blues',
linewidths=.5, linecolor='black')
    plt.title(f'Observed Frequencies: {var1_name} vs. {var2_name}')
    plt.xlabel(var2_name)
    plt.ylabel(var1_name)
    plt.show()


    return {
        "Chi2-statistic": chi2_statistic,
        "P-value": p_value,
        "Degrees of Freedom": degrees_of_freedom,
        "Alpha": alpha,
        "Interpretation": interpretation,
        "Observed Frequencies": contingency_table,
        "Expected Frequencies": pd.DataFrame(expected_frequencies,
index=contingency_table.index, columns=contingency_table.columns)
    }

# Example Usage:
# Scenario: Is there a relationship between gender and preferred ice cream
flavor?

data_independence = {
    'Gender': ['Male', 'Female', 'Male', 'Female', 'Male', 'Female', 'Male',
'Female', 'Male', 'Female',
               'Male', 'Female', 'Male', 'Female', 'Male', 'Female', 'Male',
'Female', 'Male', 'Female',
                'Male', 'Female', 'Male', 'Female', 'Male', 'Female', 'Male',
'Female', 'Male', 'Female'],
    'Flavor': ['Vanilla', 'Chocolate', 'Vanilla', 'Strawberry', 'Chocolate',
'Vanilla', 'Strawberry', 'Chocolate',
                'Vanilla', 'Chocolate', 'Vanilla', 'Chocolate', 'Strawberry',
'Vanilla', 'Chocolate', 'Vanilla',
                'Vanilla', 'Strawberry', 'Chocolate', 'Vanilla', 'Vanilla',
'Chocolate', 'Strawberry', 'Vanilla',
                'Chocolate', 'Strawberry', 'Vanilla', 'Chocolate', 'Vanilla',
'Strawberry']
}
df_independence = pd.DataFrame(data_independence)

print("--- Chi-square Test for Independence ---")
results_chi2_independence = chi_square_test_independence(df_independence)
for key, value in results_chi2_independence.items():
    if key not in ["Observed Frequencies", "Expected Frequencies"]: # Print
tables separately
        print(f"{key}: {value}")

# Scenario 2: Data where there might be no significant association
data_no_association = {
    'Education': ['High School', 'College', 'High School', 'College',
'University', 'High School', 'University', 'College', 'High School',
'University',
                   'College', 'High School', 'University', 'College', 'High
School', 'University', 'College', 'High School', 'University', 'College'],
    'Opinion': ['Agree', 'Agree', 'Disagree', 'Neutral', 'Agree', 'Disagree',
'Agree', 'Neutral', 'Agree', 'Disagree',
                 'Neutral', 'Agree', 'Disagree', 'Agree', 'Neutral', 'Agree',
'Disagree', 'Neutral', 'Agree', 'Disagree']
}
df_no_association = pd.DataFrame(data_no_association)

print("\n--- Chi-square Test (No Strong Association Expected) ---")
results_no_assoc = chi_square_test_independence(df_no_association)
for key, value in results_no_assoc.items():
    if key not in ["Observed Frequencies", "Expected Frequencies"]:
        print(f"{key}: {value}")
```

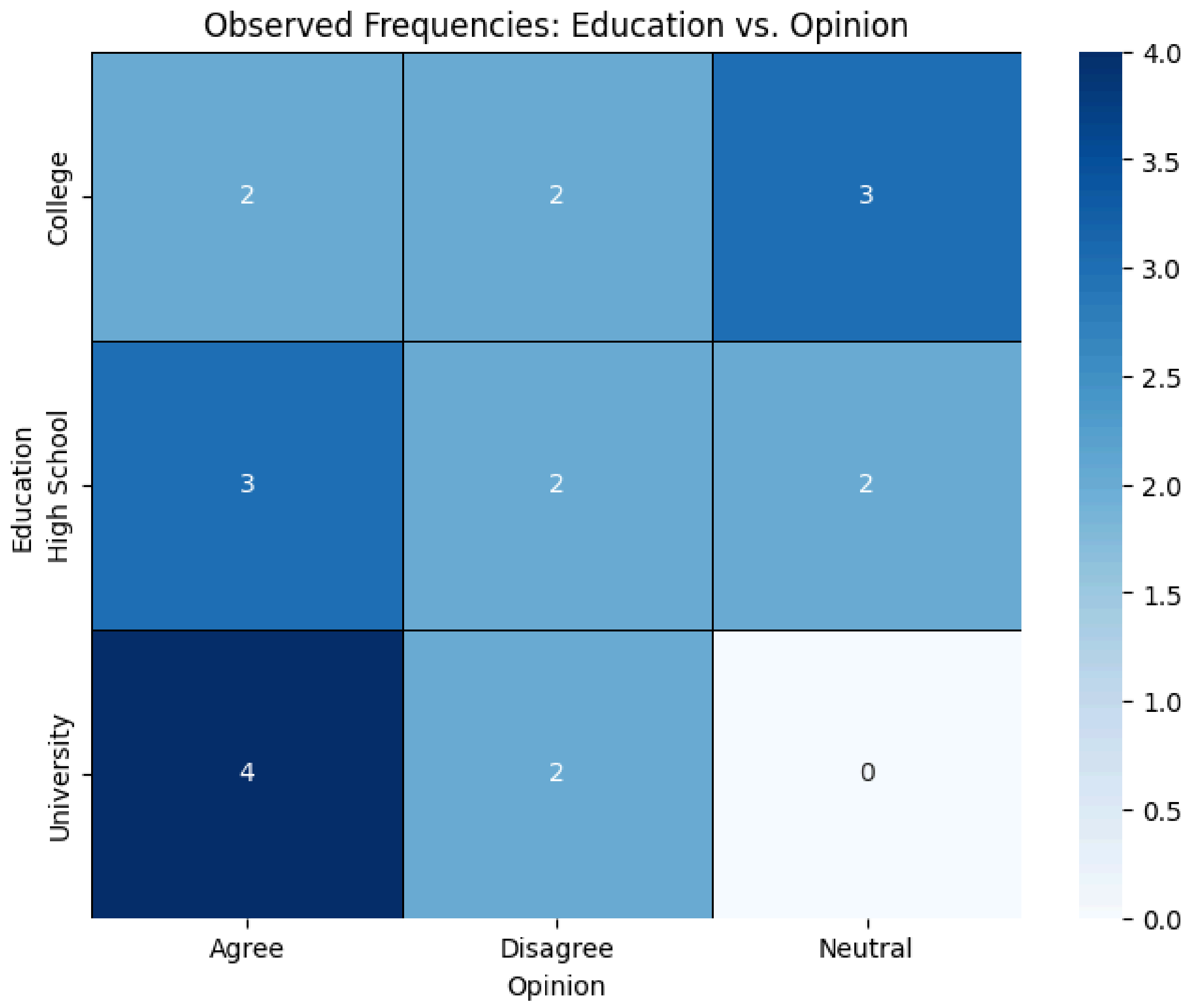
```
--- Chi-square Test for Independence ---
Observed Frequencies (Contingency Table):
Flavor  Chocolate  Strawberry  Vanilla
Gender
Female          6           4        5
Male            4           3        8

Expected Frequencies:
Flavor  Chocolate  Strawberry  Vanilla
```

```
Gender
Female         5.0          3.5      6.5
Male           5.0          3.5      6.5
```



```
Chi2-statistic: 1.2351648351648352
P-value: 0.5392465359394735
Degrees of Freedom: 2
Alpha: 0.05
Interpretation: The P-value (0.5392) is greater than or equal to the
significance level (0.05).
We fail to reject the null hypothesis. There is no statistically significant
evidence of an association between '{var1_name}' and '{var2_name}'.

--- Chi-square Test (No Strong Association Expected) ---
Observed Frequencies (Contingency Table):
Opinion      Agree  Disagree  Neutral
Education
```

```
College           2          2         3
High School       3          2         2
University        4          2         0

Expected Frequencies:
Opinion       Agree  Disagree  Neutral
Education
College        3.15       2.1     1.75
High School    3.15       2.1     1.75
University     2.70       1.8     1.50
```



```
Chi2-statistic: 3.513227513227513
P-value: 0.4758699245153669
Degrees of Freedom: 4
Alpha: 0.05
Interpretation: The P-value (0.4759) is greater than or equal to the
significance level (0.05).
```

```
We fail to reject the null hypothesis. There is no statistically significant
evidence of an association between '{var1_name}' and '{var2_name}'.
```

13. Write a Python program to calculate the expected frequencies for a Chi-square test based on observed data.

In [49]:

```
import numpy as np
import pandas as pd

def calculate_expected_frequencies(observed_data_df):
    """
    Calculates the expected frequencies for a Chi-square test based on observed
categorical data.

    Args:
        observed_data_df (pandas.DataFrame): A DataFrame with two categorical
columns.

    Returns:
        tuple: A tuple containing the observed contingency table (DataFrame)
               and the expected frequencies table (DataFrame).
    """
    if observed_data_df.shape[1] != 2:
        raise ValueError("Input DataFrame must have exactly two columns.")

    var1_name = observed_data_df.columns[0]
    var2_name = observed_data_df.columns[1]

    # Create the observed contingency table
    observed_contingency_table = pd.crosstab(observed_data_df[var1_name],
observed_data_df[var2_name])

    row_totals = observed_contingency_table.sum(axis=1)
    col_totals = observed_contingency_table.sum(axis=0)
    grand_total = observed_contingency_table.sum().sum()

    expected_frequencies_array = np.outer(row_totals, col_totals) / grand_total
    expected_frequencies_df = pd.DataFrame(expected_frequencies_array,

index=observed_contingency_table.index,

columns=observed_contingency_table.columns)

    return observed_contingency_table, expected_frequencies_df

# Example Usage:
# Same example as before: Gender vs. Preferred Ice Cream Flavor
```

```
data_exp = {
    'Gender': ['Male', 'Female', 'Male', 'Female', 'Male', 'Female', 'Male',
'Female', 'Male', 'Female',
               'Male', 'Female', 'Male', 'Female', 'Male', 'Female', 'Male',
'Female', 'Male', 'Female',
               'Male', 'Female', 'Male', 'Female', 'Male', 'Female', 'Male',
'Female', 'Male', 'Female'],
    'Flavor': ['Vanilla', 'Chocolate', 'Vanilla', 'Strawberry', 'Chocolate',
'Vanilla', 'Strawberry', 'Chocolate',
               'Vanilla', 'Chocolate', 'Vanilla', 'Chocolate', 'Strawberry',
'Vanilla', 'Chocolate', 'Vanilla',
               'Vanilla', 'Strawberry', 'Chocolate', 'Vanilla', 'Vanilla',
'Chocolate', 'Strawberry', 'Vanilla',
               'Chocolate', 'Strawberry', 'Vanilla', 'Chocolate', 'Vanilla',
'Strawberry']
}
df_exp = pd.DataFrame(data_exp)

print("--- Calculating Observed and Expected Frequencies ---")
observed_table, expected_table = calculate_expected_frequencies(df_exp)

print("\nObserved Frequencies:\n", observed_table)
print("\nExpected Frequencies:\n", expected_table)

print("\n--- Verification with scipy.stats.chi2_contingency ---")
from scipy.stats import chi2_contingency
chi2_stat, p_val, dof, expected_scipy = chi2_contingency(observed_table)
print("\nExpected Frequencies (from scipy.stats.chi2_contingency):\n",
pd.DataFrame(expected_scipy, index=observed_table.index,
columns=observed_table.columns))
```

```
--- Calculating Observed and Expected Frequencies ---

Observed Frequencies:
 Flavor  Chocolate  Strawberry  Vanilla
Gender
Female          6           4        5
Male            4           3        8

Expected Frequencies:
 Flavor  Chocolate  Strawberry  Vanilla
Gender
Female        5.0         3.5      6.5
Male          5.0         3.5      6.5

--- Verification with scipy.stats.chi2_contingency ---

Expected Frequencies (from scipy.stats.chi2_contingency):
 Flavor  Chocolate  Strawberry  Vanilla
Gender
Female         5.0         3.5      6.5
Male           5.0         3.5      6.5
```

14. Perform a goodness-of-fit test using Python to compare the observed data to an expected distribution.

Ans: The Chi-square goodness-of-fit test determines if observed frequencies for a single categorical variable differ significantly from expected frequencies, based on a hypothesized distribution.

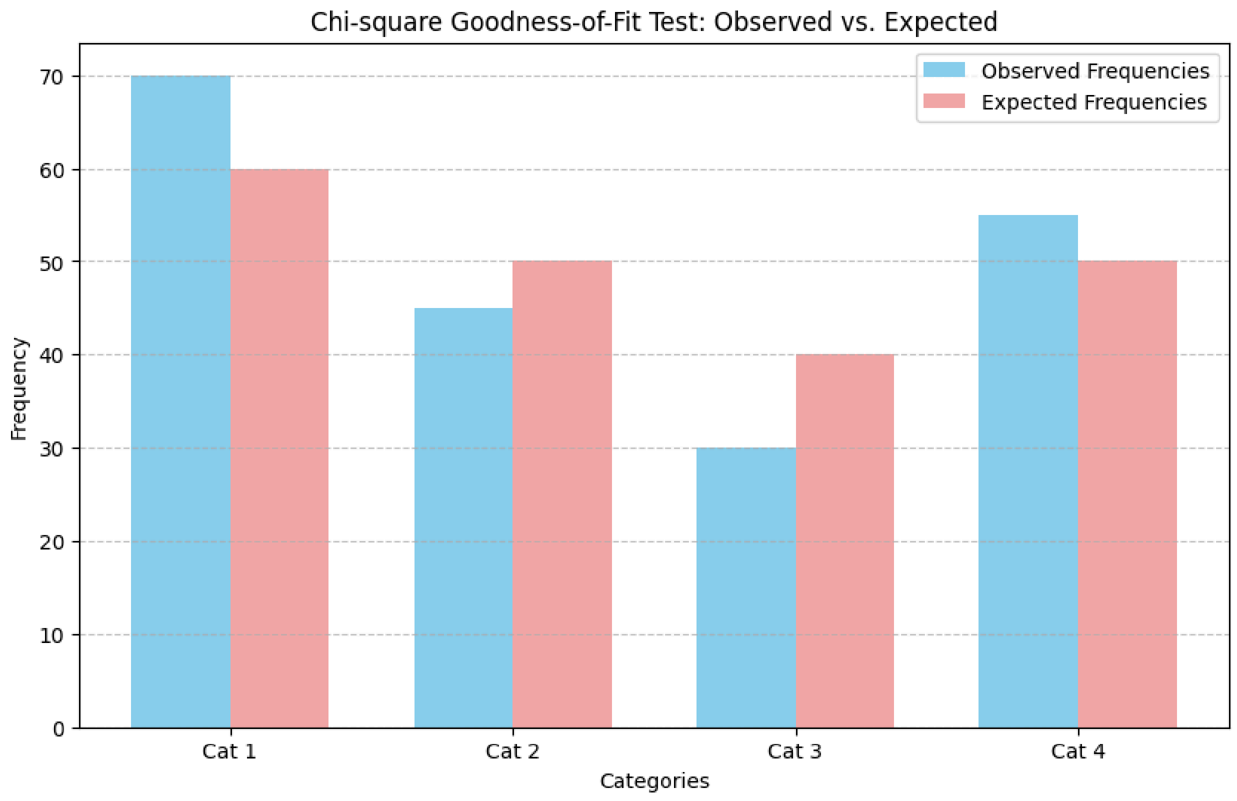
Assumptions:

Observations are independent. Expected frequencies in each category are at least 5.

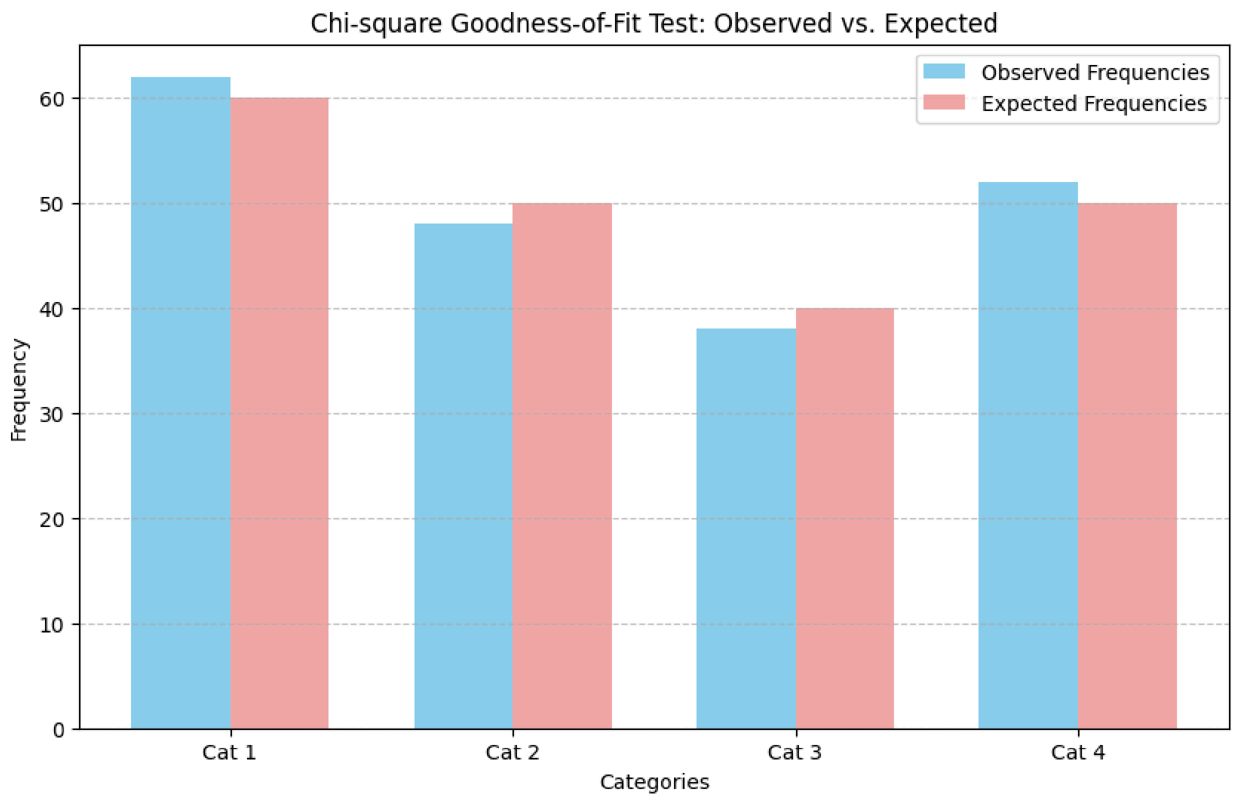
In [50]:

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns

def chi_square_goodness_of_fit(observed_frequencies, expected_frequencies,
alpha=0.05):
    """
    Performs a Chi-square goodness-of-fit test.

    Args:
        observed_frequencies (array-like): The observed counts for each
category.
        expected_frequencies (array-like): The hypothesized expected counts for
each category.
                                           Must have the same length as
observed_frequencies.
        alpha (float): The significance level (default is 0.05).

    Returns:
        dict: A dictionary containing the Chi-square statistic, P-value,
degrees of freedom, and interpretation.
    """
    if len(observed_frequencies) != len(expected_frequencies):
        raise ValueError("Observed and Expected frequencies must have the same
number of categories.")
    if any(e < 0 for e in expected_frequencies):
        raise ValueError("Expected frequencies cannot be negative.")
    if sum(observed_frequencies) != sum(expected_frequencies):
        print("Warning: Sum of observed frequencies does not equal sum of
expected frequencies. "
               "This might indicate an issue with expected distribution
definition.")


    # Perform the Chi-square goodness-of-fit test
    # stats.chisquare can take 'f_exp' (expected frequencies)
    chi2_statistic, p_value = stats.chisquare(f_obs=observed_frequencies,
f_exp=expected_frequencies)

    # Degrees of freedom for goodness-of-fit is (number of categories - 1)
    degrees_of_freedom = len(observed_frequencies) - 1

    # Interpret the results
    interpretation = ""
    if p_value < alpha:
        interpretation = (
            f"The P-value ({p_value:.4f}) is less than the significance level
({alpha}).\n"
            "We reject the null hypothesis. There is statistically significant
evidence "
            "that the observed distribution is different from the expected
distribution."
        )
    else:
        interpretation = (
            f"The P-value ({p_value:.4f}) is greater than or equal to the
significance level ({alpha}).\n"
            "We fail to reject the null hypothesis. There is no statistically
significant evidence "
            "that the observed distribution is different from the expected
distribution."
        )

    # Visualization
    categories = [f"Cat {i+1}" for i in range(len(observed_frequencies))]
    x = np.arange(len(categories))

    plt.figure(figsize=(10, 6))
    width = 0.35
    plt.bar(x - width/2, observed_frequencies, width, label='Observed
Frequencies', color='skyblue')
    plt.bar(x + width/2, expected_frequencies, width, label='Expected
Frequencies', color='lightcoral', alpha=0.7)

    plt.xlabel('Categories')
    plt.ylabel('Frequency')
    plt.title('Chi-square Goodness-of-Fit Test: Observed vs. Expected')
    plt.xticks(x, categories)
    plt.legend()
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()


    return {
        "Chi2-statistic": chi2_statistic,
        "P-value": p_value,
        "Degrees of Freedom": degrees_of_freedom,
        "Alpha": alpha,
        "Interpretation": interpretation
    }

# Example Usage:
# Scenario: A marketing company claims that 30% of people prefer Brand A, 25%
Brand B, 20% Brand C, and 25% Brand D.
# A survey of 200 people yields the following observed preferences.
# Null Hypothesis: The observed distribution of preferences fits the claimed
distribution.

total_people = 200
observed_prefs = np.array([70, 45, 30, 55]) # Observed counts for Brand A, B,
C, D
# Sum: 70+45+30+55 = 200

# Expected preferences based on the claim (total 200 people)
expected_prefs = np.array([total_people * 0.30, # Brand A
                           total_people * 0.25, # Brand B
                           total_people * 0.20, # Brand C
                           total_people * 0.25])# Brand D
# Sum: 60+50+40+50 = 200

print("--- Chi-square Goodness-of-Fit Test ---")
print(f"Observed Frequencies: {observed_prefs}")
print(f"Expected Frequencies: {expected_prefs}")

results_goodness_of_fit = chi_square_goodness_of_fit(observed_prefs,
expected_prefs)
for key, value in results_goodness_of_fit.items():
    print(f"{key}: {value}")

# Scenario 2: Data that is likely to fit the expected distribution
print("\n--- Chi-square Goodness-of-Fit Test (Likely Good Fit) ---")
observed_good_fit = np.array([62, 48, 38, 52]) # Close to expected
results_good_fit = chi_square_goodness_of_fit(observed_good_fit,
expected_prefs)
for key, value in results_good_fit.items():
    print(f"{key}: {value}")
```

```
--- Chi-square Goodness-of-Fit Test ---
Observed Frequencies: [70 45 30 55]
Expected Frequencies: [60. 50. 40. 50.]
```



```
Chi2-statistic: 5.166666666666667
P-value: 0.1599919632008065
Degrees of Freedom: 3
Alpha: 0.05
Interpretation: The P-value (0.1600) is greater than or equal to the
significance level (0.05).
We fail to reject the null hypothesis. There is no statistically significant
evidence that the observed distribution is different from the expected
distribution.

--- Chi-square Goodness-of-Fit Test (Likely Good Fit) ---
```



```
Chi2-statistic: 0.32666666666666666
P-value: 0.9549376042087493
Degrees of Freedom: 3
Alpha: 0.05
Interpretation: The P-value (0.9549) is greater than or equal to the
significance level (0.05).
We fail to reject the null hypothesis. There is no statistically significant
evidence that the observed distribution is different from the expected
distribution.
```

Explanation:

Hypotheses:

Null Hypothesis (H 0): The observed frequencies are consistent with the hypothesized expected frequencies (i.e., the data fits the specified distribution).

Alternative Hypothesis (H 1): The observed frequencies are significantly different from the hypothesized expected frequencies (i.e., the data does not fit the specified distribution).

observed_frequencies: These are the actual counts you collected for each category.

expected_frequencies: These are the counts you would expect to see in each category if the null hypothesis were true. You calculate these based on your hypothesized distribution and the total number of observations.

Important: The sum of your observed_frequencies must be equal to the sum of your expected_frequencies (or nearly equal if there are rounding errors).

Chi-square Statistic: Calculated similarly to the independence test, it quantifies the difference between observed and expected counts: $\chi^2 = \sum E_i$

$(O_i - E_i)^2$

Where:

O_i is the observed frequency for category i. E_i is the expected frequency for category i.

Degrees of Freedom (df): For a goodness-of-fit test, df=(number of categories−1).

P-value: The probability of observing a Chi-square statistic as extreme as, or more extreme than, the one calculated, assuming the null hypothesis of a good fit is true.

Interpretation:

If P-value $< \alpha$: Reject H_0. Conclude that the observed distribution is significantly different from the expected distribution.

If P-value $\geq \alpha$: Fail to reject H_0. Conclude that there's no significant evidence that the observed distribution differs from the expected distribution.

Visualization (Bar Chart): A bar chart comparing the observed and expected frequencies side-by-side makes it easy to visually assess how well the observed data matches the hypothesized distribution.
