

## ✓ Assignment Statistics Advance Module 3 Part 2

### Theory Questions

#### Q1. What is hypothesis testing in statistics?

Ans-> Hypothesis testing is a form of statistical inference that uses data from a sample to draw conclusions about a population parameter or a population probability distribution. First, a tentative assumption is made about the parameter or distribution. This assumption is called the null hypothesis and is denoted by  $H_0$ .

#### Q2. What is the null hypothesis, and how does it differ from the alternative hypothesis?

Ans-> The null hypothesis and alternative hypothesis are two statements used in statistical testing to explore a research question. The null hypothesis ( $H_0$ ) proposes that there is no significant effect or relationship between variables being studied, while the alternative hypothesis ( $H_1$  or  $H_a$ ) suggests that a significant effect or relationship does exist. The goal of hypothesis testing is to determine whether there is enough evidence to reject the null hypothesis in favor of the alternative hypothesis.

- Here's a more detailed breakdown:
- Null Hypothesis ( $H_0$ ):
  - Assumes no significant difference or relationship exists between variables.
  - Often represents the status quo or a default position.
  - Researchers aim to reject or fail to reject this hypothesis.
- Alternative Hypothesis ( $H_1$  or  $H_a$ ):
  - States that a significant difference or relationship exists between variables.
  - Represents the researcher's prediction or what they are trying to prove.
  - Testing aims to support or accept this hypothesis if the null hypothesis is rejected.

#### Q3. What is the significance level in hypothesis testing, and why is it important?

Ans-> In hypothesis testing, the significance level (alpha, denoted by  $\alpha$ ) is the probability of incorrectly rejecting the null hypothesis when it is actually true. It essentially sets the threshold for determining statistical significance, influencing whether you conclude the observed results are due to a real effect or just chance.

- Why is it important?
- Controlling Type I Errors: The significance level helps manage the risk of making a Type I error (false positive), which occurs when you reject the null hypothesis when it's actually true. Setting a lower significance level (e.g., 0.01) reduces the risk of false positives, but it may also increase the risk of missing true effects (Type II error), according to the Statsig website.
- Interpreting Results: The significance level helps determine the strength of evidence against the null hypothesis. A lower p-value (the probability of observing the data if the null hypothesis is true) compared to the significance level leads to rejecting the null hypothesis.
- Decision Making: In many situations, especially in research and scientific studies, the significance level helps researchers make informed decisions about whether to accept or reject a hypothesis based on statistical evidence, according to the National Institutes of Health (NIH).
- In essence, the significance level acts as a gatekeeper in hypothesis testing, ensuring that conclusions are based on statistically strong evidence and minimizing the risk of drawing incorrect conclusions.

#### Q4. What does a P-value in hypothesis testing?

Ans-> In hypothesis testing, a p-value represents the probability of observing data at least as extreme as the data actually observed, assuming the null hypothesis is true. It essentially measures the strength of evidence against the null hypothesis.

- Here's a more detailed explanation:
- Null Hypothesis: This is a statement about the population that you're trying to disprove. For example, it might state that there is no difference between two groups.
- P-value: This value tells you how likely the observed data is if the null hypothesis were actually true.
- Interpreting the P-value:
  - Small p-value (e.g.,  $p < 0.05$ ): This suggests that the observed data is unlikely to have occurred if the null hypothesis is true. This provides evidence to reject the null hypothesis.
  - Large p-value (e.g.,  $p > 0.05$ ): This suggests that the observed data is likely to have occurred even if the null hypothesis is true. This provides less evidence to reject the null hypothesis.

#### Q5. How do you interpret the P-value in hypothesis testing?

Ans-> In hypothesis testing, the p-value represents the probability of observing the test statistic (or a more extreme value) if the null hypothesis were true. It essentially quantifies the strength of the evidence against the null hypothesis. A smaller p-value indicates stronger evidence against the null, suggesting the observed effect is unlikely due to chance.

- Elaboration:
- Null Hypothesis: The null hypothesis ( $H_0$ ) is a statement of no effect or no difference between groups. For example, in a drug trial, the null hypothesis might be that there's no difference in treatment outcomes between two drugs.
- P-value and Statistical Significance: The p-value is used to determine if the observed results are statistically significant. If the p-value is less than or equal to the significance level (often denoted as  $\alpha$ , usually 0.05), the null hypothesis is rejected in favor of the alternative hypothesis. This means there's enough evidence to conclude that the observed effect is not likely due to chance.
- Interpreting the P-value:
- Low P-value ( $p \leq \alpha$ ): Suggests the observed effect is unlikely to have occurred by chance alone, leading to rejection of the null hypothesis.
- High P-value ( $p > \alpha$ ): Indicates the observed effect is more likely due to chance, and the null hypothesis is not rejected (also called failing to reject the null hypothesis). This doesn't necessarily mean the null hypothesis is true, but rather that the data provides insufficient evidence to reject it.

## Q6. What are Type 1 and Type 2 errors in hypothesis testing?

Ans-> In hypothesis testing, a Type I error (false positive) occurs when you reject a true null hypothesis, and a Type II error (false negative) occurs when you fail to reject a false null hypothesis.

- Type I Error (False Positive):
- Definition: This error happens when you reject the null hypothesis ( $H_0$ ) when it is actually true. In other words, you conclude there's an effect or relationship when there isn't one.
- Example: Imagine you are testing a new drug for a disease. You find statistically significant evidence that the drug is effective, but in reality, the drug has no effect. This would be a Type I error.
- Symbol: The probability of a Type I error is often denoted by  $\alpha$  (alpha). Also known as: False alarm, false positive.
- Type II Error (False Negative):
- Definition: This error occurs when you fail to reject the null hypothesis when it is actually false. You conclude that there's no effect or relationship when one actually exists.
- Example: Using the same drug example, you might not find statistically significant evidence of its effectiveness, even though it actually is. This would be a Type II error.
- Symbol: The probability of a Type II error is often denoted by  $\beta$  (beta). Also known as: Missed detection, false negative.
- Power: The probability of correctly rejecting a false null hypothesis is called the power of the test. It's related to Type II error as power =  $1 - \beta$ .

## Q7. What is the difference between a one-tailed and a two-tailed test in hypothesis testing?

Ans-> In hypothesis testing, a one-tailed test looks for a difference in one specific direction (either greater than or less than), while a two-tailed test looks for a difference in either direction (greater than or less than). The key difference lies in the alternative hypothesis: one-tailed tests have a directional alternative hypothesis, while two-tailed tests have a non-directional one.

- One-tailed test:
- Directional: It specifies that the parameter of interest is either greater than or less than a particular value.
- Focus: Concentrates on detecting an effect in one specific direction (e.g., only if the new drug is more effective than the old one, or only if the new method is less expensive than the old one).
- Example: Testing if a new fertilizer increases crop yield (directional hypothesis).
- Two-tailed test:
- Non-directional: It tests if the parameter is different from a particular value, without specifying the direction of the difference.
- Focus: Detects any deviation from the null hypothesis, regardless of the direction (e.g., if the new drug is different from the old one, meaning either more or less effective).
- Example: Testing if a new teaching method is different from the current method (non-directional hypothesis).

## Q8. What is the Z-test, and when is it used in hypothesis testing?

Ans-> A Z-test is a statistical hypothesis test used to determine if there's a significant difference between a sample mean and a population mean, or between the means of two populations, when the population standard deviation is known, and the sample size is large. It's primarily used when the sample size is 30 or more, allowing the use of the normal distribution to approximate the test statistic, according to DataCamp.

- Here's a more detailed breakdown:
- 1. When to use a Z-test:
- Large sample size: A Z-test is generally preferred when you have a large sample size (usually  $n \geq 30$ ).
- Known population standard deviation: You need to know the population standard deviation ( $\sigma$ ) for a Z-test to be appropriate.

- Normal distribution (or Central Limit Theorem): The test assumes that the data is normally distributed, or that the sample mean is approximately normally distributed due to the Central Limit Theorem.
  - Comparing sample mean to population mean or comparing two sample means: Z-tests can be used to compare a sample mean to a known population mean (one-sample Z-test) or to compare the means of two independent samples (two-sample Z-test).
2. Steps in conducting a Z-test:
  3. State the null hypothesis ( $H_0$ ) and the alternative hypothesis ( $H_a$ ): The null hypothesis assumes no significant difference between the sample and the population mean (or between the two population means). The alternative hypothesis suggests there is a significant difference.
  4. Set the significance level ( $\alpha$ ): This is the probability of rejecting the null hypothesis when it's actually true (usually 0.05 or 0.01).
  5. Calculate the Z-score: This measures how many standard deviations the sample mean is away from the hypothesized population mean (or the difference between the two sample means).
  6. Determine the critical value(s): Based on the significance level and whether the test is one-tailed or two-tailed, you determine the critical Z-score(s) that define the rejection region.
  7. Compare the Z-score to the critical value(s): If the calculated Z-score falls within the rejection region, you reject the null hypothesis, indicating a statistically significant difference.
  8. Types of Z-tests: One-sample Z-test: Compares a sample mean to a known population mean. Two-sample Z-test: Compares the means of two independent samples.

### Q9. How do you calculate the Z-score, and what does it represent in hypothesis testing?

Ans-> The z-score, also known as a standard score, is calculated by determining how many standard deviations a particular data point is away from the mean of a population. In hypothesis testing, the z-score helps to evaluate the strength of evidence against a null hypothesis.

- Calculating the Z-score: The z-score is calculated using the following formula:  $z = (x - \mu) / \sigma$  Where:  $x$ : is the individual data point or raw score.  $\mu$ : is the population mean.  $\sigma$ : is the population standard deviation. If the population parameters (mean and standard deviation) are unknown, you can approximate the z-score using the sample mean and standard deviation:  $z = (x - \bar{x}) / s$  Where:  $\bar{x}$ : is the sample mean.  $s$ : is the sample standard deviation. Z-score in Hypothesis Testing: In hypothesis testing, the z-score is used as a test statistic to evaluate the strength of evidence against the null hypothesis.
1. State the Null Hypothesis: The null hypothesis ( $H_0$ ) typically assumes that there is no effect or difference in the population.
  2. State the Alternative Hypothesis: The alternative hypothesis ( $H_1$ ) states what you believe to be true if the null hypothesis is false.
  3. Calculate the Z-score: Calculate the z-score using the sample data.
  4. Determine the p-value: The p-value represents the probability of observing the test statistic (or a more extreme value) if the null hypothesis is true.
  5. Make a decision: Compare the p-value to the significance level ( $\alpha$ ), which is typically set at 0.05. If the p-value is less than or equal to  $\alpha$ , you reject the null hypothesis.

### Q10. What is the T-distribution, and when should it be used instead of the normal distribution?

Ans-> The T-distribution is a probability distribution similar to the normal distribution, but with heavier tails, meaning it has a higher probability of extreme values. It's used when dealing with small sample sizes or when the population standard deviation is unknown, offering a more accurate representation of data variability in such scenarios.

- Here's a more detailed explanation:
- Similarities to the Normal Distribution: The T-distribution is bell-shaped and symmetrical, just like the normal distribution, with the mean at zero.
- Difference in Tail Behavior: The key difference lies in the tails. The T-distribution has "heavier" tails, meaning it assigns more probability to extreme values compared to the normal distribution.
- When to Use T-Distribution: Small Sample Sizes: When the sample size is small (generally considered less than 30), the T-distribution provides a more accurate representation of the data compared to the normal distribution.
- Unknown Population Standard Deviation: When the population standard deviation is unknown and needs to be estimated from the sample, the T-distribution is more appropriate.
- Relationship to Normal Distribution: As the sample size increases, the T-distribution approaches the normal distribution, making them virtually indistinguishable for large sample sizes.
- Degrees of Freedom: The shape of the T-distribution is influenced by the "degrees of freedom," which are related to the sample size (typically calculated as  $n-1$ ).

### Q11. What is the difference between a Z-test and a T-test?

Ans-> The key difference between a Z-test and a T-test lies in the sample size and the availability of population standard deviation:

- Z-test: Used for large sample sizes (generally greater than 30) and when the population standard deviation is known. It relies on the Z-distribution (a standard normal distribution).
- T-test: Used for small sample sizes (generally less than 30) or when the population standard deviation is unknown. It relies on the T-distribution, which is similar to the normal distribution but accounts for the uncertainty in estimating the population standard deviation.

## Q12. What is the T-test, and how is it used in hypothesis testing?

Ans-> A t-test is a statistical hypothesis test used to determine if there's a significant difference between the means of two groups. It's commonly used in hypothesis testing to compare sample means to a known value, or to compare the means of two groups.

- Here's a more detailed explanation:
- What is a t-test? It's a statistical method that compares the means of two groups to see if the difference is statistically significant or just due to random chance. It's particularly useful when dealing with small sample sizes or when the population standard deviation is unknown. The t-test calculates a t-value, which quantifies the difference between the group means, taking into account the variability within each group.
- How is it used in hypothesis testing?
  1. Formulating Hypotheses: You start with a null hypothesis, which assumes no difference between the means of the groups being compared. You then propose an alternative hypothesis, which suggests that there is a difference.
  2. Calculating the t-value: The t-test formula is used to calculate the t-value based on the sample data, group means, and variability.
  3. Interpreting the t-value: The calculated t-value is compared to a critical value (obtained from a t-distribution table or using statistical software).
    - If the calculated t-value exceeds the critical value, the null hypothesis is rejected, suggesting a statistically significant difference.
    - If the calculated t-value is less than the critical value, the null hypothesis is not rejected, suggesting no significant difference.

## Q13. What is the relationship between Z-test and T-test in hypothesis testing?

Ans-> Z-tests and T-tests are both used in hypothesis testing to compare means, but they differ in their assumptions and when they're most appropriate. Z-tests are used when the population standard deviation is known, or when the sample size is large (typically  $n > 30$ ), and the sample data follows a normal distribution. T-tests are used when the population standard deviation is unknown, or when the sample size is small (typically  $n \leq 30$ ), and the sample data is assumed to be approximately normally distributed.

- Here's a more detailed breakdown:
- Z-test:
  - Assumptions: Population standard deviation is known, sample data is normally distributed, or the sample size is large ( $n > 30$ ). Test Statistic: Uses the standard normal distribution (Z-distribution).
  - When to use: Comparing a sample mean to a known population mean when the population standard deviation is known or the sample size is large.
- T-test:
  - Assumptions: Population standard deviation is unknown, sample data is approximately normally distributed.
  - Test Statistic: Uses the Student's t-distribution.
  - When to use: Comparing sample means when the population standard deviation is unknown, especially with smaller sample sizes.
- Key Differences:
  - Sample Size: T-tests are preferred for smaller samples ( $n \leq 30$ ), while Z-tests are more appropriate for larger samples ( $n > 30$ ).
  - Population Standard Deviation: Z-tests assume the population standard deviation is known, while T-tests deal with situations where it's unknown.
  - Test Statistic: Z-tests use the standard normal distribution, while T-tests use the Student's t-distribution.

## Q14. What is the confidence interval, and how is it used to interpret statistical results?

Ans-> A confidence interval is a range of values used to estimate an unknown population parameter, with a specified level of confidence. It provides a range within which the true population parameter is likely to fall based on sample data. A confidence level (e.g., 95%) indicates the probability that the interval will contain the true population value.

- Here's how it's used to interpret statistical results:
  1. Estimating Population Parameters: Confidence intervals help estimate unknown parameters like the population mean or proportion.
  2. Quantifying Uncertainty: They provide a measure of how precisely the sample data has estimated the population parameter. A narrower interval suggests more precision, while a wider interval indicates less certainty.
  3. Hypothesis Testing: Confidence intervals can be used to support decisions in hypothesis testing. For example, if a two-tailed test is performed and the hypothesized value falls within the confidence interval, the null hypothesis is not rejected.
  4. Statistical Significance: If a confidence interval for the difference between two means does not include zero, it suggests a statistically significant difference between those means.
  5. Interpreting Confidence Levels: A 95% confidence interval means that if you repeated the study many times, you would expect 95% of the constructed intervals to contain the true population parameter.

## Q15. What is the margin of error, and how does it affect the confidence interval?

Ans-> The margin of error quantifies the potential inaccuracy of a sample statistic in estimating a population parameter. It's essentially half the width of a confidence interval, which is a range of values within which the true population value is likely to fall. A smaller margin of error



indicates a more precise estimate. The margin of error directly impacts the confidence interval by defining its width: a wider confidence interval means a larger margin of error, and vice versa.

- Here's a more detailed explanation:
- Margin of Error:
  - Definition: The margin of error is the amount of random sampling error in a survey's results. It represents the potential difference between the sample result and the true population value.
  - Calculation: The margin of error is calculated based on the sample size, the variability of the data, and the desired level of confidence.
  - Interpretation: It's often expressed as a plus or minus ( $\pm$ ) value, indicating the range within which the true population value is likely to fall.
- Factors Affecting Margin of Error:
  - Sample Size: Larger sample sizes generally lead to smaller margins of error, as they provide more reliable estimates of the population.
  - Data Variability: Higher variability in the data (i.e., greater spread in values) tends to increase the margin of error.
- Confidence Level: A higher confidence level (e.g., 99% vs. 95%) will result in a wider confidence interval and thus a larger margin of error.
- Confidence Interval:
  - Definition: A confidence interval is a range of values that is likely to contain the true population parameter with a certain level of confidence.
  - Relationship with Margin of Error: The confidence interval is formed by adding and subtracting the margin of error from the sample statistic.
  - Interpretation: The confidence interval provides a range within which the true population value is likely to lie, based on the sample data and the chosen confidence level.
- Impact of Margin of Error: A smaller margin of error results in a narrower confidence interval, indicating a more precise estimate of the population parameter. Conversely, a larger margin of error leads to a wider confidence interval, reflecting greater uncertainty about the true population value.

## Q16. How is Bayes' Theorem used in statistics, and what is its significance?

Ans-> Bayes' Theorem is a fundamental concept in statistics that allows for updating probability estimates based on new evidence. It's used to calculate the conditional probability of an event, meaning the probability of an event occurring given that another event has already occurred. Its significance lies in its ability to refine our understanding of uncertain events by incorporating prior knowledge with new information.

- Here's a more detailed explanation:
  1. What is Bayes' Theorem? It's a mathematical formula that calculates conditional probability. It's expressed as:  $P(A|B) = [P(B|A) * P(A)] / P(B)$ . Where:  $P(A|B)$  is the posterior probability (probability of event A given event B).  $P(B|A)$  is the likelihood (probability of event B given event A).  $P(A)$  is the prior probability (initial probability of event A).  $P(B)$  is the probability of event B.
  2. How is it used in statistics? Updating Beliefs: Bayes' Theorem allows us to update our beliefs about an event (or hypothesis) as we receive new information. Conditional Probability: It helps calculate the probability of an event A given that event B has already happened.
- Bayesian Inference: It's a core concept in Bayesian statistics, a field that uses probabilities to express degrees of belief and update these beliefs based on new data. Various Applications: It's used in diverse fields like medical diagnosis, machine learning, and decision-making.
  3. What is its significance? Incorporating Prior Knowledge: It allows us to use our existing knowledge (prior probability) and new evidence (likelihood) to refine our understanding of an event. Decision Making: By updating probabilities, it aids in making better decisions in uncertain situations.
    - Handling Uncertainty: It provides a framework for dealing with uncertainty by allowing us to quantify our beliefs and update them as new information becomes available.
- Foundation of Bayesian Statistics: It's a foundational tool in Bayesian statistics, which is increasingly important in fields like machine learning and artificial intelligence.

## Q17. What is the Chi-square distribution, and when is it used?

Ans-> The Chi-square distribution is a continuous probability distribution used in statistical hypothesis testing, particularly for analyzing categorical data. It's used in tests like the goodness-of-fit test and the test of independence to determine if there's a significant difference between observed and expected frequencies or if two variables are related. Chi Square Distribution - Lean Six Sigma Glossary Term

- Here's a more detailed explanation:
- What is the Chi-square distribution? It's a family of continuous probability distributions, meaning the shape of the distribution changes based on a parameter called "degrees of freedom" (df), denoted by 'k'.
- The shape of a chi-square distribution is determined by the degrees of freedom (k).
- It's commonly used in hypothesis testing, not for describing real-world distributions themselves. It's derived from the sum of the squares of standard normal random variables.

- When is the Chi-square distribution used?
- Goodness-of-fit test: This test determines how well observed data fits a theoretical distribution. For example, it can be used to test if a coin is fair or if a population distribution matches a normal distribution.
- Test of independence: This test examines if there's a relationship between two categorical variables. For instance, it can be used to see if there's a relationship between gender and course choice.
- Other applications: It's also used in inferential problems dealing with variance, confidence intervals for variance, and even in cryptanalysis and bioinformatics.

## Q18. What is the Chi-square goodness of fit test, and how is it applied?

Ans-> The Chi-square goodness-of-fit test is a statistical hypothesis test used to determine if a sample's observed distribution of categorical data matches a theoretical or hypothesized distribution. It essentially checks how well the observed frequencies in a dataset align with the expected frequencies based on a particular model or distribution. Here's how it's applied:

1. Hypothesis Formulation: Null Hypothesis: The observed distribution is not significantly different from the theoretical distribution.  
Alternative Hypothesis: The observed distribution is significantly different from the theoretical distribution.
2. Data Collection and Preparation: Collect data for a categorical variable. Determine the expected frequencies for each category based on the hypothesized distribution. For continuous data, it may be necessary to group data into intervals to create categorical data.
3. Calculate the Test Statistic: The Chi-square test statistic is calculated as the sum of squared differences between the observed and expected frequencies, divided by the expected frequencies. Formula:  $\chi^2 = \sum [(O_i - E_i)^2 / E_i]$ .
4. Determine Degrees of Freedom: Degrees of freedom (df) are calculated as the number of categories minus one ( $df = k - 1$ ), where k is the number of categories.
5. Determine the p-value: Use the Chi-square distribution table or statistical software to find the p-value corresponding to the calculated test statistic and degrees of freedom.
6. Decision and Conclusion: Compare the p-value to the significance level (alpha, typically 0.05). If  $p\text{-value} \leq \alpha$ , reject the null hypothesis; otherwise, fail to reject the null hypothesis. A small p-value suggests the observed distribution significantly differs from the expected distribution.

## Q19. What is the F-distribution, and when is it used in hypothesis testing?

Ans-> The F-distribution is a probability distribution used in hypothesis testing, particularly when comparing variances between two or more groups. It's often used in conjunction with the F-test, a statistical method for comparing variances or mean comparisons in multiple groups.

- Here's a more detailed explanation:
  1. What is the F-distribution? The F-distribution is a continuous probability distribution that arises from the ratio of two independent chi-squared random variables, each divided by its degrees of freedom. It's characterized by two parameters: the degrees of freedom of the numerator and the degrees of freedom of the denominator. It's skewed to the right and has a positive value, meaning it can only take values greater than zero.
  2. When is the F-distribution used in hypothesis testing?
    - Comparing variances: The F-distribution is primarily used to test the null hypothesis that two or more populations have equal variances. The F-test utilizes the F-statistic, which is calculated as the ratio of two variances. Analysis of Variance (ANOVA): ANOVA, which stands for analysis of variance, is a statistical test used to determine if there is a significant difference between the means of two or more groups. The F-test is the core statistical test used in ANOVA. Comparing multiple means: When you want to compare the means of more than two groups, the F-distribution is often used in conjunction with the F-test to determine if there is a significant difference between any of the group means.
  3. How does the F-distribution work in hypothesis testing? Null hypothesis: The null hypothesis typically states that there is no significant difference between the variances or means of the groups being compared. Alternative hypothesis: The alternative hypothesis states that there is a significant difference between the variances or means.
- F-statistic: The F-statistic is calculated from the sample data and represents the ratio of the variance between groups to the variance within groups.
- Critical value: A critical value is determined from the F-distribution based on the chosen significance level and the degrees of freedom.
- Decision: If the calculated F-statistic is greater than the critical value, the null hypothesis is rejected, and the conclusion is that there is a significant difference.

## Q20. What is an ANOVA test, and what are its assumptions?

Ans-> ANOVA (Analysis of Variance) is a statistical test used to compare the means of two or more groups. It helps determine if there's a significant difference between the group means, considering that the groups are distinct and not influenced by other factors. Key assumptions of ANOVA include normality of data within each group, homogeneity of variance (equal variance across groups), and independent observations.

- Assumptions of ANOVA:

1. Normality: The data within each group should be approximately normally distributed. This means the data should follow a bell-shaped curve, without significant skewness or outliers.
2. Homogeneity of Variance: The variances within each group should be approximately equal. This means the spread or variability of the data should be similar across all groups.
3. Independence of Observations: Observations within each group should be independent of each other. This means that one observation does not influence or depend on another observation within the same group.
4. Random Sampling: The data should be collected through random sampling, ensuring that each individual has an equal chance of being selected. In simpler terms: Normal Distribution: The data within each group should look like a typical bell curve, not skewed or with extreme values.

## Q21. What are the different types of ANOVA tests?

Ans-> The main types of ANOVA tests are one-way ANOVA and two-way ANOVA. One-way ANOVA compares means of a single independent variable with multiple levels, while two-way ANOVA examines the effects of two or more independent variables and their interactions.

- Elaboration:
- One-way ANOVA: This test is used when you want to compare the means of three or more groups based on a single independent variable. For example, you could use it to compare the average test scores of students using different teaching methods.
- Two-way ANOVA: This test is used when you want to compare the means of two or more groups based on two or more independent variables. It also allows you to examine the interaction effects between these variables. For example, you could use it to compare the average productivity of employees based on both their training level and their work experience.

## Q22. What is the F-test, and how does it relate to hypothesis testing?

Ans-> The F-test is a statistical test that compares variances between two or more groups, primarily used in hypothesis testing to determine if there's a significant difference in means. It's related to hypothesis testing because it helps us decide whether to reject the null hypothesis of equal variances or equal means (depending on the context).

- How the F-test Relates to Hypothesis Testing:
  1. Testing for Significant Differences: The F-test is a core tool in hypothesis testing, especially when comparing multiple groups or factors. It determines if the variation between groups is significantly larger than the variation within groups.
  2. Null Hypothesis: The null hypothesis in an F-test usually states that there is no difference in the variances (or means) of the groups being compared.
  3. Alternative Hypothesis: The alternative hypothesis suggests that there is a significant difference in the variances (or means).
  4. F-Statistic: The F-test calculates an F-statistic, which is the ratio of the variance between groups to the variance within groups.
  5. F-Distribution: This F-statistic is then compared to a critical value from the F-distribution, which helps determine the probability of observing such an F-statistic if the null hypothesis were true.
  6. Decision: If the calculated F-statistic is greater than the critical value, the null hypothesis is rejected, suggesting that there is a significant difference in the variances or means.

## Practical Part - 1

### Q1. Write a Python program to generate a random variable and display its value

# prompt: Write a Python program to generate a random variable and display its value

```
import random
```

```
# Generate a random floating-point number between 0 and 1
random_variable = random.random()
```

```
# Display the value of the random variable
print("Random Variable:", random_variable)
```

➞ Random Variable: 0.19153992765536176

### Q2. Generate a discrete uniform distribution using Python and plot the probability mass function (PMF)

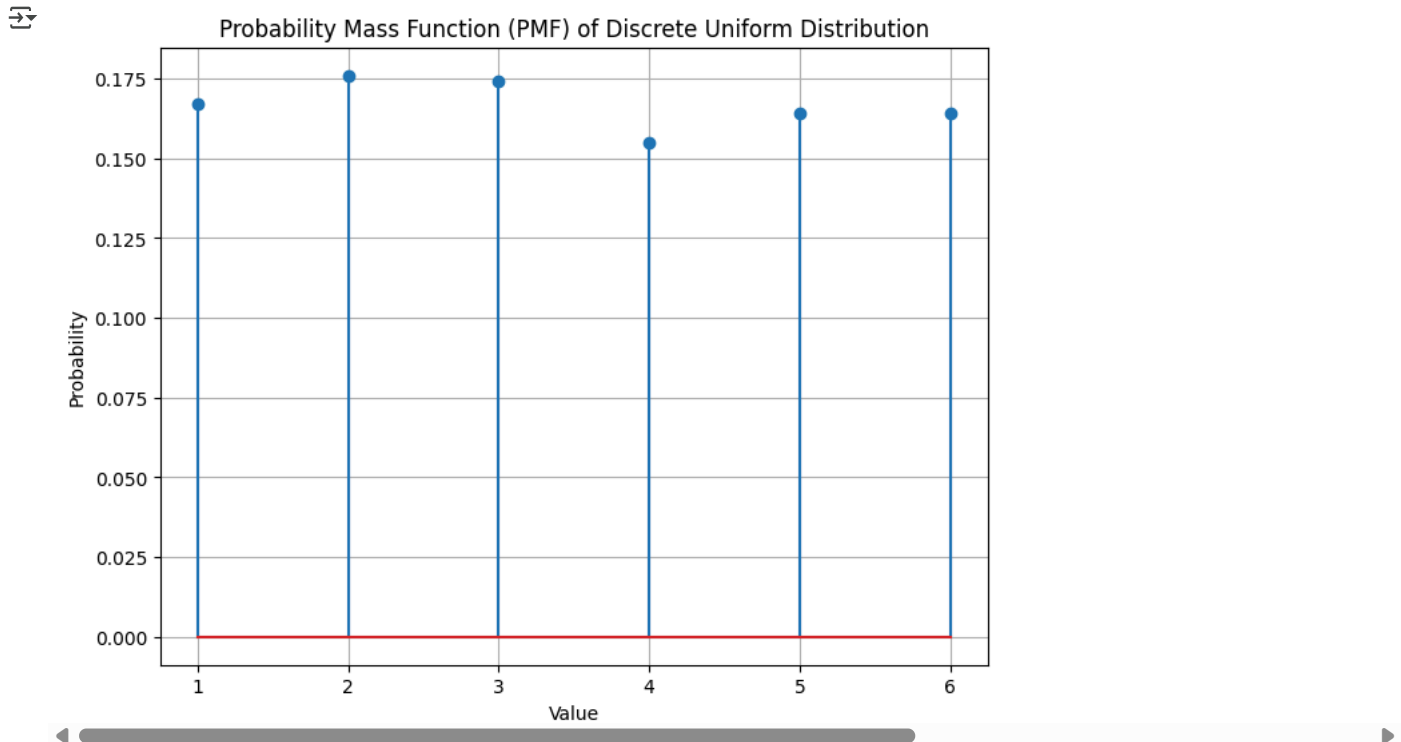
```
import matplotlib.pyplot as plt
import numpy as np
```

```
# Define the parameters of the discrete uniform distribution
low = 1 # Lower bound of the distribution
high = 6 # Upper bound of the distribution
size = 1000 # Number of random variables to generate
```

```
# Generate random variables from a discrete uniform distribution
random_variables = np.random.randint(low, high + 1, size=size)

# Calculate the probability mass function (PMF)
unique, counts = np.unique(random_variables, return_counts=True)
pmf = counts / size

# Plot the PMF
plt.figure(figsize=(8, 6)) # Adjust figure size if needed
# Remove use_line_collection or update matplotlib
# plt.stem(unique, pmf, use_line_collection=True)
plt.stem(unique, pmf)
plt.title('Probability Mass Function (PMF) of Discrete Uniform Distribution')
plt.xlabel('Value')
plt.ylabel('Probability')
plt.grid(True)
plt.show()
```



✓ Q3. Write a Python function to calculate the probability distribution (PDF) of a Bernoulli distribution

# prompt: Write a Python function to calculate the probability distribution (PDF) of a Bernoulli distribution

```
def bernoulli_pdf(p, k):
    """
    Calculates the probability distribution function (PDF) of a Bernoulli distribution.

    Args:
        p: The probability of success (a value between 0 and 1).
        k: The outcome (either 0 or 1).

    Returns:
        The probability of the given outcome, or an error message if input is invalid.
    """

    if not 0 <= p <= 1:
        return "Error: Probability 'p' must be between 0 and 1."
    if k not in (0, 1):
        return "Error: Outcome 'k' must be either 0 or 1."

    if k == 1:
        return p
    else: # k == 0
        return 1 - p
```



#### ✓ Q4. Write a Python script to simulate a binominal distribution with $n=10$ and $p=0.5$ , then plot its histogram

# prompt: Write a Python script to simulate a binominal distribution with  $n=10$  and  $p=0.5$ , then plot its histogram

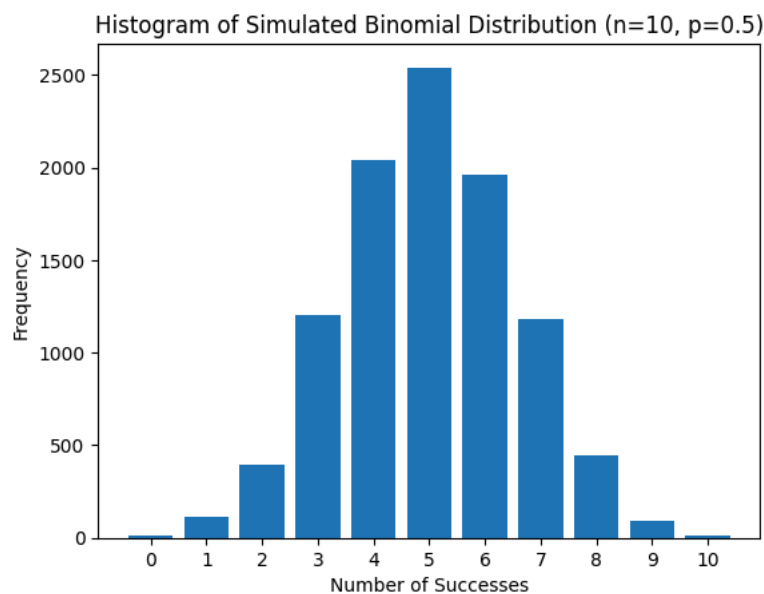
```
import random
import matplotlib.pyplot as plt
import numpy as np

def binomial_simulation(n, p, num_trials):
    results = []
    for _ in range(num_trials):
        successes = 0
        for _ in range(n):
            if random.random() < p:
                successes += 1
        results.append(successes)
    return results

# Set parameters
n = 10
p = 0.5
num_trials = 10000

# Simulate the binomial distribution
simulated_data = binomial_simulation(n, p, num_trials)

# Plot the histogram
plt.hist(simulated_data, bins=range(n + 2), align='left', rwidth=0.8)
plt.xlabel('Number of Successes')
plt.ylabel('Frequency')
plt.title('Histogram of Simulated Binomial Distribution (n=10, p=0.5)')
plt.xticks(range(n + 1))
plt.show()
```



#### ✓ Q5. Create a Poisson distribution and visualize it using Python

# prompt: Create a Poisson distribution and visualize it using Python

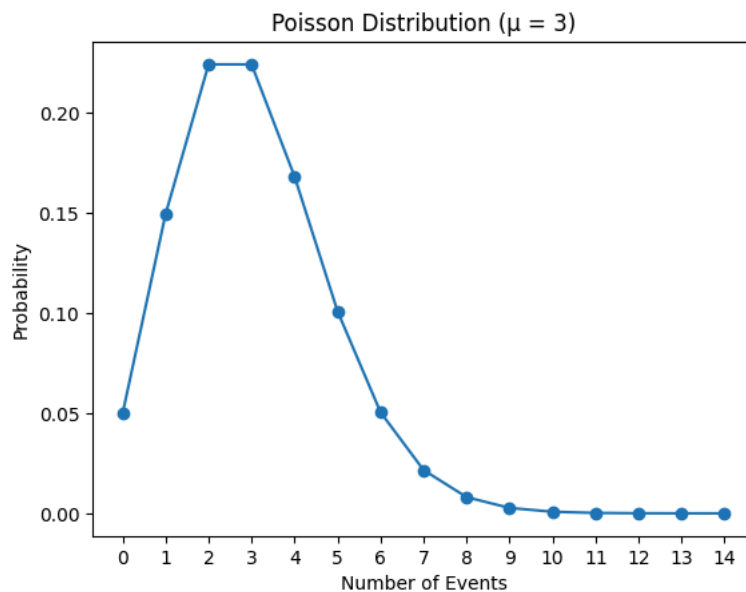
```
import matplotlib.pyplot as plt
from scipy.stats import poisson

# Define the parameters of the Poisson distribution
mu = 3 # Mean of the distribution

# Generate random variables from a Poisson distribution
x = np.arange(0, 15) # Values for the x-axis
y = poisson.pmf(x, mu)

# Plot the distribution
plt.plot(x, y, 'o-')
```

```
plt.title('Poisson Distribution ( $\mu = 3$ )')
plt.xlabel('Number of Events')
plt.ylabel('Probability')
plt.xticks(x) # Set x-axis ticks
plt.show()
```



✓ Q6. Write a Python program to calculate and plot the cumulative distribution function(CDF) of a discrete uniform distribution

# prompt: Write a Python program to calculate and plot the cumulative distribution function(CDF) of a discrete uniform distribution

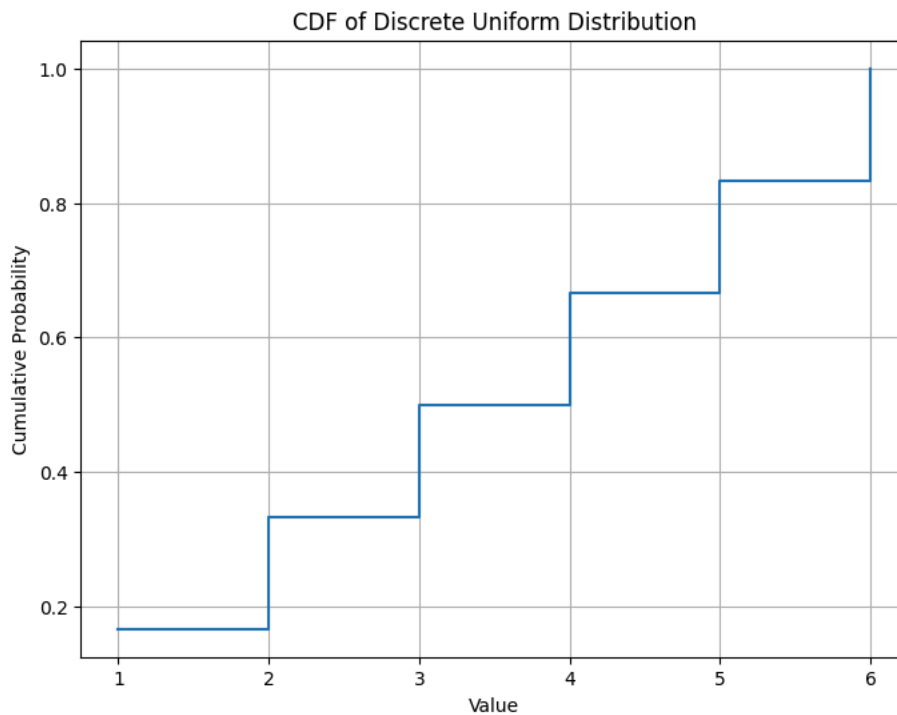
```
import matplotlib.pyplot as plt
import numpy as np

def plot_cdf_discrete_uniform(low, high):
    """
    Calculates and plots the cumulative distribution function (CDF) of a discrete uniform distribution.

    Args:
        low: The lower bound of the distribution.
        high: The upper bound of the distribution.
    """
    values = np.arange(low, high + 1)
    probabilities = np.ones(len(values)) / len(values)
    cdf = np.cumsum(probabilities)

    plt.figure(figsize=(8, 6))
    plt.step(values, cdf, where='post') # Use 'step' for discrete CDF
    plt.xlabel('Value')
    plt.ylabel('Cumulative Probability')
    plt.title('CDF of Discrete Uniform Distribution')
    plt.grid(True)
    plt.show()

# Example usage
low = 1
high = 6
plot_cdf_discrete_uniform(low, high)
```

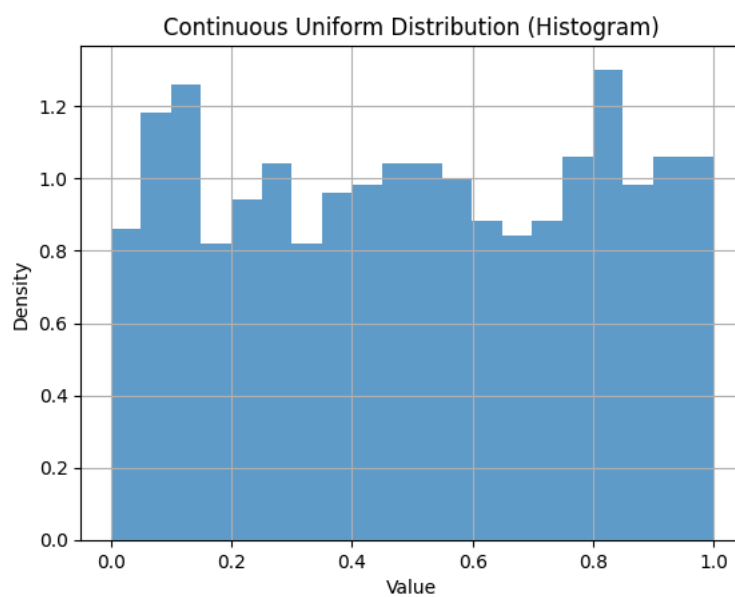


### Q7. Generate a continuous uniform distribution using Numpy and visualize it

# prompt: Generate a continuous uniform distribution using Numpy and visualize it

```
# Generate a continuous uniform distribution
low = 0 # Lower bound of the distribution
high = 1 # Upper bound of the distribution
size = 1000 # Number of random variables to generate
uniform_data = np.random.uniform(low, high, size)

# Visualize the distribution using a histogram
plt.hist(uniform_data, bins=20, density=True, alpha=0.7) # Adjust number of bins as needed
plt.xlabel('Value')
plt.ylabel('Density')
plt.title('Continuous Uniform Distribution (Histogram)')
plt.grid(True)
plt.show()
```

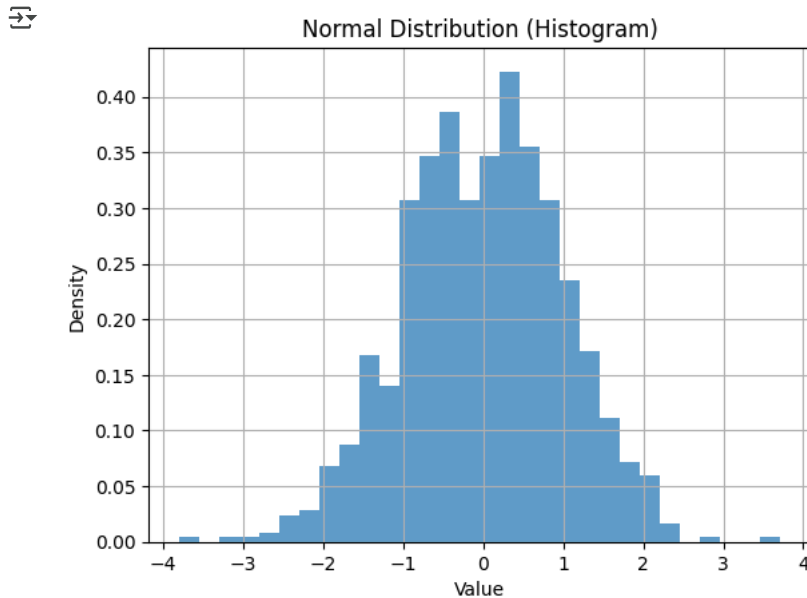


### Q8. Simulate data from a normal distribution and plot its histogram

```
# prompt: Simulate data from a normal distribution and plot its histogram
```

```
# Simulate data from a normal distribution
mu = 0 # Mean of the normal distribution
sigma = 1 # Standard deviation of the normal distribution
size = 1000 # Number of data points to simulate
normal_data = np.random.normal(mu, sigma, size)

# Plot the histogram
plt.hist(normal_data, bins=30, density=True, alpha=0.7)
plt.xlabel('Value')
plt.ylabel('Density')
plt.title('Normal Distribution (Histogram)')
plt.grid(True)
plt.show()
```



### ✓ Q9. Write a Python function to calculate Z-scores from a dataset and plot them

```
# prompt: Write a Python function to calculate Z-scores from a dataset and plot them
```

```
def calculate_and_plot_zscores(data):
    import numpy as np
    import matplotlib.pyplot as plt

    # Calculate the mean and standard deviation of the dataset
    data_mean = np.mean(data)
    data_std = np.std(data)

    # Calculate the Z-scores
    z_scores = (data - data_mean) / data_std

    # Plot the Z-scores
    plt.figure(figsize=(8, 6))
    plt.plot(z_scores)
    plt.xlabel("Data Point Index")
    plt.ylabel("Z-score")
    plt.title("Z-scores of the Dataset")
    plt.grid(True)
    plt.show()

    return z_scores
```

### ✓ Q10. Implement the Central Limit Theorem (CLT) using Python for a non-normal distribution

```
# prompt: Implement the Central Limit Theorem (CLT) using Python for a non-normal distribution
```

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

```
def clt_simulation(distribution, n_samples, n_simulations):
    """
    Simulates the Central Limit Theorem for a given distribution.

    Args:
        distribution: A function that generates random numbers from the desired distribution.
        n_samples: The number of samples to draw from the distribution in each simulation.
        n_simulations: The number of simulations to run.

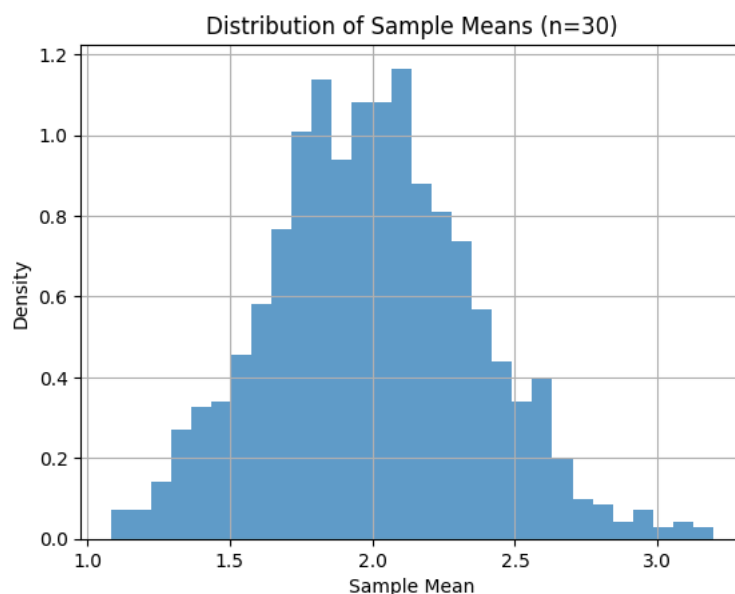
    Returns:
        A list of sample means from each simulation.
    """
    sample_means = []
    for _ in range(n_simulations):
        samples = [distribution() for _ in range(n_samples)]
        sample_mean = np.mean(samples)
        sample_means.append(sample_mean)
    return sample_means

# Example usage with an exponential distribution (non-normal)
def exponential_distribution():
    return random.expovariate(lambd=0.5) # Example lambda

n_samples = 30
n_simulations = 1000

sample_means = clt_simulation(exponential_distribution, n_samples, n_simulations)

# Plot the histogram of sample means
plt.hist(sample_means, bins=30, density=True, alpha=0.7)
plt.xlabel('Sample Mean')
plt.ylabel('Density')
plt.title(f'Distribution of Sample Means (n={n_samples})')
plt.grid(True)
plt.show()
```



## Q11. Simulates a multiple samples from a normal distribution and verify the Central Limit Theorem

# prompt: Simulates a multiple samples from a normal distribution and verify the Central Limit Theorem

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

def clt_normal_simulation(mu, sigma, n_samples, n_simulations):
    """
    Simulates the Central Limit Theorem for a normal distribution.

    Args:
        mu: Mean of the normal distribution.
        sigma: Standard deviation of the normal distribution.
        n_samples: Number of samples to draw in each simulation.
        n_simulations: Number of simulations to run.
```



```

Returns:
    A list of sample means from each simulation.
"""
sample_means = []
for _ in range(n_simulations):
    samples = np.random.normal(mu, sigma, n_samples)
    sample_mean = np.mean(samples)
    sample_means.append(sample_mean)
return sample_means

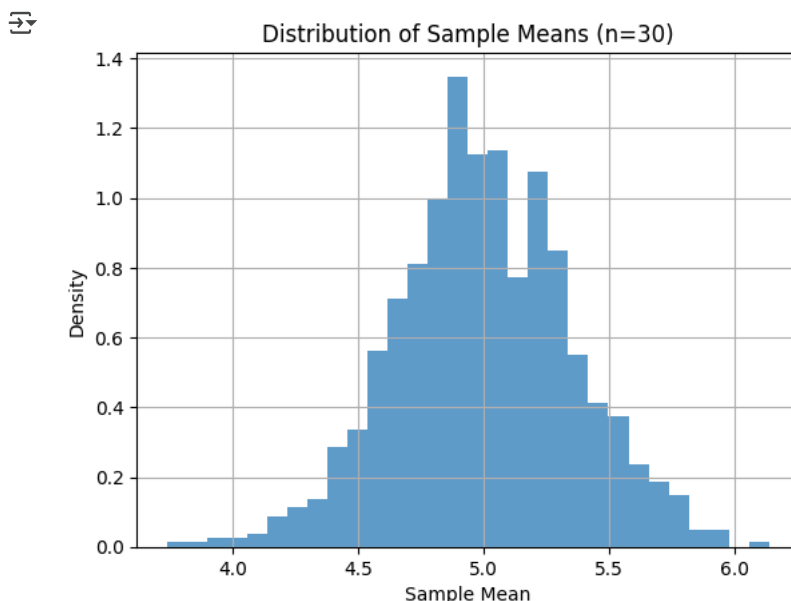
# Set parameters
mu = 5 # Population mean
sigma = 2 # Population standard deviation
n_samples = 30 # Sample size
n_simulations = 1000 # Number of simulations

# Simulate the CLT for the normal distribution
sample_means = clt_normal_simulation(mu, sigma, n_samples, n_simulations)

# Plot the histogram of sample means
plt.hist(sample_means, bins=30, density=True, alpha=0.7)
plt.xlabel('Sample Mean')
plt.ylabel('Density')
plt.title(f'Distribution of Sample Means (n={n_samples})')
plt.grid(True)
plt.show()

# Calculate and print the mean and standard deviation of the sample means
mean_of_sample_means = np.mean(sample_means)
std_of_sample_means = np.std(sample_means)
print(f"Mean of sample means: {mean_of_sample_means}")
print(f"Standard deviation of sample means: {std_of_sample_means}")

```



```

Mean of sample means: 5.001945010222537
Standard deviation of sample means: 0.35760702060009412

```

✓ Q12. Write a Python function to calculate and plot the standard normal distribution (mean=0, std=1)

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

```

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

def plot_standard_normal_distribution():
    """Calculates and plots the standard normal distribution (mean=0, std=1)."""
    x = np.linspace(-4, 4, 100) # Values for the x-axis
    y = norm.pdf(x, loc=0, scale=1) # Calculate PDF using norm.pdf

    plt.plot(x, y)
    plt.title('Standard Normal Distribution (μ=0, σ=1)')
    plt.xlabel('x')

```

```
plt.ylabel('Probability Density')
plt.grid(True)
plt.show()
```

### Q13. Generate random variables and calculate their corresponding probabilities using the binomial distribution

# prompt: Generate random variables and calculate their corresponding probabilities using the binomial distribution

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

# Set parameters for the binomial distribution
n = 10 # Number of trials
p = 0.5 # Probability of success

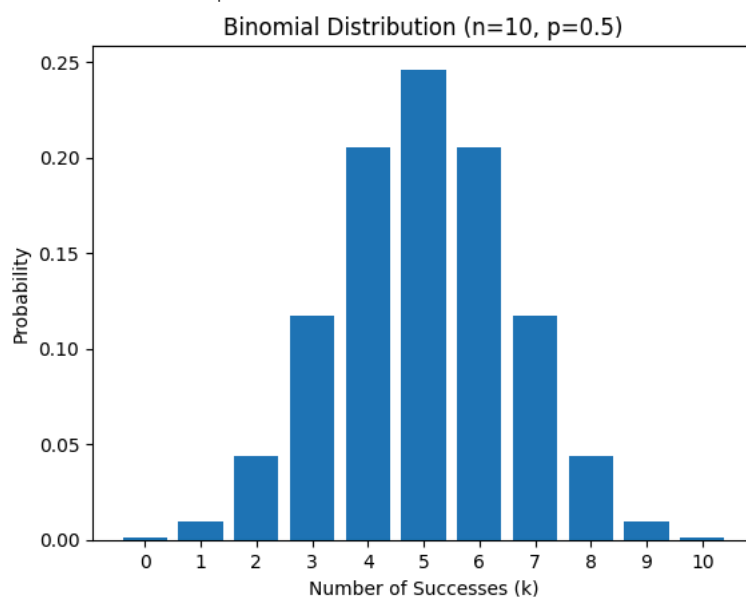
# Generate random variables (number of successes)
k_values = range(n + 1) # Possible number of successes (0 to n)

# Calculate probabilities using the binomial probability mass function (PMF)
probabilities = binom.pmf(k_values, n, p)

# Print the results
print("Number of Successes (k) | Probability")
for k, prob in zip(k_values, probabilities):
    print(f"{k:20} | {prob:.4f}") # Print with formatting for readability

# Plot the binomial distribution
plt.bar(k_values, probabilities)
plt.title(f'Binomial Distribution (n={n}, p={p})')
plt.xlabel('Number of Successes (k)')
plt.ylabel('Probability')
plt.xticks(k_values) # Ensure all k values are shown on x-axis
plt.show()
```

```
↔ Number of Successes (k) | Probability
0 | 0.0010
1 | 0.0098
2 | 0.0439
3 | 0.1172
4 | 0.2051
5 | 0.2461
6 | 0.2051
7 | 0.1172
8 | 0.0439
9 | 0.0098
10 | 0.0010
```



### Q14. Write a Python program to calculate the Z-score for a given data point and compare it to a standard normal distribution

```
# prompt: Write a Python program to calculate the Z-score for a given data point and compare it to a standard normal distribution

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

def calculate_zscore(data_point, data_mean, data_std):
    """Calculates the Z-score for a given data point."""
    z_score = (data_point - data_mean) / data_std
    return z_score

def compare_to_standard_normal(z_score):
    """Compares a Z-score to the standard normal distribution."""
    x = np.linspace(-4, 4, 100)
    y = norm.pdf(x, loc=0, scale=1)

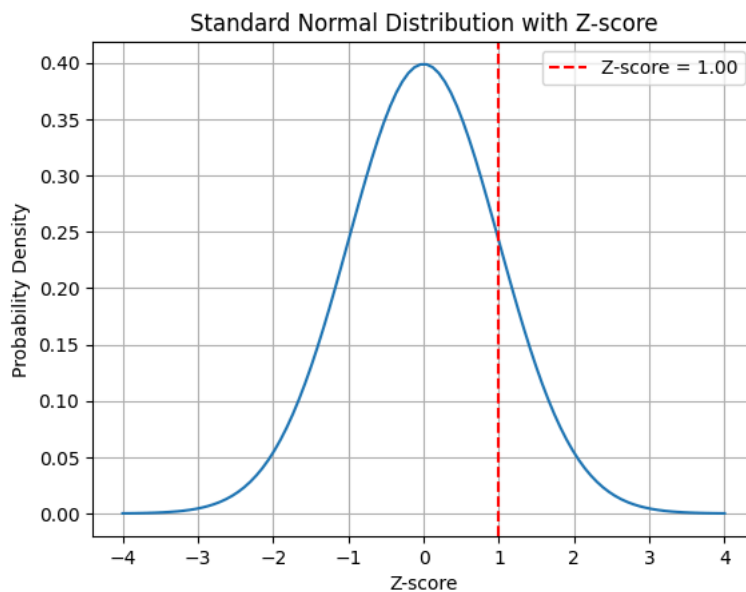
    plt.plot(x, y)
    plt.title('Standard Normal Distribution with Z-score')
    plt.xlabel('Z-score')
    plt.ylabel('Probability Density')
    plt.grid(True)
    plt.axvline(x=z_score, color='red', linestyle='--', label=f'Z-score = {z_score:.2f}')
    plt.legend()
    plt.show()

# Example Usage
data_point = 75 # Example data point
data_mean = 70 # Example mean of the data
data_std = 5 # Example standard deviation

# Calculate the Z-score
z_score = calculate_zscore(data_point, data_mean, data_std)
print(f"The Z-score for the data point {data_point} is: {z_score}")

# Compare the Z-score to the standard normal distribution
compare_to_standard_normal(z_score)
```

↗ The Z-score for the data point 75 is: 1.0



## ✓ Q15. Implement hypothesis testing using Z-statistics for a sample dataset

```
# prompt: Implement hypothesis testing using Z-statistics for a sample dataset

from scipy.stats import norm

def z_test(sample_mean, population_mean, population_std, sample_size, alpha=0.05):
    """
    Performs a one-sample Z-test for a population mean.

    Args:
        sample_mean: The mean of the sample.
        population_mean: The hypothesized population mean.
        population_std: The population standard deviation.
        sample_size: The size of the sample.
        alpha: The significance level (default is 0.05).
```

```

Returns:
    A tuple containing:
    - The Z-statistic.
    - The p-value.
    - A string indicating whether to reject or fail to reject the null hypothesis.
"""

# Calculate the Z-statistic
z_statistic = (sample_mean - population_mean) / (population_std / (sample_size**0.5))

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

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

return z_statistic, p_value, decision

# Example usage
sample_mean = 72 # Sample mean
population_mean = 70 # Hypothesized population mean
population_std = 5 # Population standard deviation
sample_size = 50 # Sample size
alpha = 0.05 # Significance level

z_statistic, p_value, decision = z_test(sample_mean, population_mean, population_std, sample_size, alpha)

print(f"Z-statistic: {z_statistic:.2f}")
print(f"P-value: {p_value:.3f}")
decision

```

```

↗ Z-statistic: 2.83
P-value: 0.005
'Reject the null hypothesis'

```

## ✓ Q16. Create a confidence interval for a dataset using Python and interpret the result

```

import numpy as np
import scipy.stats # Import scipy.stats
from scipy.stats import t

def confidence_interval(data, confidence=0.95):
    """
    Calculates the confidence interval for a dataset.

    Args:
        data: A list or numpy array of numerical data.
        confidence: The desired confidence level (e.g., 0.95 for a 95% confidence interval).

    Returns:
        A tuple containing the lower and upper bounds of the confidence interval.
    """
    a = 1.0 * np.array(data)
    n = len(a)
    m, se = np.mean(a), scipy.stats.sem(a) # Use imported scipy.stats.sem
    h = se * scipy.stats.t.ppf((1 + confidence) / 2., n-1) # Use imported scipy.stats.t.ppf
    return m-h, m+h

# Example usage:
data = [1,2,3,4,5,6,7,8,9,10] # Example dataset
lower_bound, upper_bound = confidence_interval(data)
print(f"95% Confidence Interval: ({lower_bound:.2f}, {upper_bound:.2f})")

# Interpretation:
# We are 95% confident that the true population mean lies between the calculated lower and upper bounds of the confidence interval.

↗ 95% Confidence Interval: (3.33, 7.67)

```

## ✓ Q17. Generate data from a normal distribution, then calculate and interpret the confidence interval for its mean

```
# prompt: Generate data from a normal distribution, then calculate and interpret the confidence interval for its mean

import numpy as np
from scipy.stats import t

def confidence_interval_normal(data, confidence=0.95):
    """
    Calculates the confidence interval for a dataset assuming a normal distribution.

    Args:
        data: A list or numpy array of numerical data.
        confidence: The desired confidence level (e.g., 0.95 for a 95% confidence interval).

    Returns:
        A tuple containing the lower and upper bounds of the confidence interval.
    """
    a = 1.0 * np.array(data)
    n = len(a)
    m, se = np.mean(a), np.std(a, ddof=1) / np.sqrt(n)
    h = se * t.ppf((1 + confidence) / 2., n - 1)
    return m - h, m + h

# Generate data from a normal distribution
np.random.seed(42) # for reproducibility
data = np.random.normal(loc=10, scale=2, size=50)

# Calculate the confidence interval
lower_bound, upper_bound = confidence_interval_normal(data)

print(f"95% Confidence Interval: ({lower_bound:.2f}, {upper_bound:.2f})")

# Interpretation:
# We are 95% confident that the true population mean lies between the calculated lower and upper bounds of the confidence interval.
```

→ 95% Confidence Interval: (9.02, 10.08)

## Q18. Write a Python script to calculate and visualize the probability density function (PDF) of a normal distribution

```
# prompt: Write a Python script to calculate and visualize the probability density function (PDF) of a normal distribution

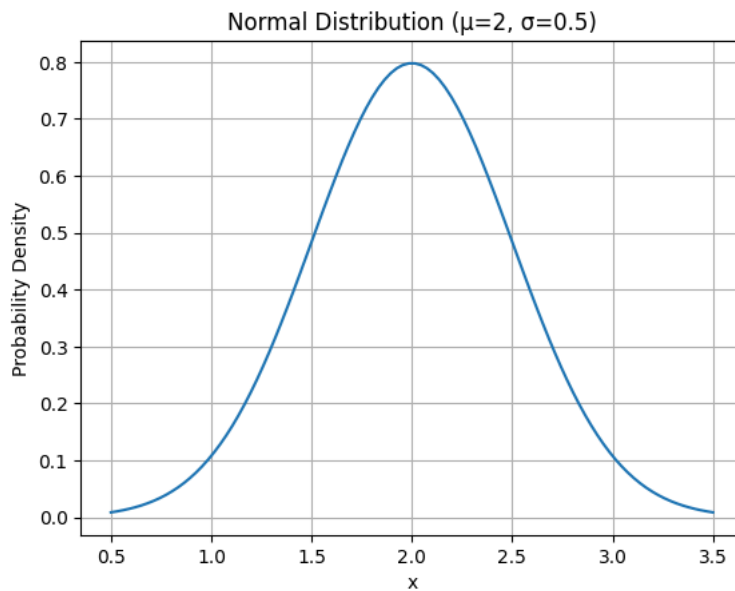
def plot_normal_pdf(mu, sigma):
    """
    Calculates and visualizes the probability density function (PDF) of a normal distribution.

    Args:
        mu: Mean of the normal distribution.
        sigma: Standard deviation of the normal distribution.
    """
    x = np.linspace(mu - 3 * sigma, mu + 3 * sigma, 100)
    y = norm.pdf(x, loc=mu, scale=sigma)

    plt.plot(x, y)
    plt.title(f'Normal Distribution ( $\mu={mu}$ ,  $\sigma={sigma}$ )')
    plt.xlabel('x')
    plt.ylabel('Probability Density')
    plt.grid(True)
    plt.show()

# Example usage
mu = 2
sigma = 0.5
plot_normal_pdf(mu, sigma)
```





✓ Q19. Use Python to calculate and interpret the cumulative distribution function(CDF) of a poisson distribution

# prompt: Use Python to calculate and interpret the cumulative distribution function(CDF) of a poisson distribution

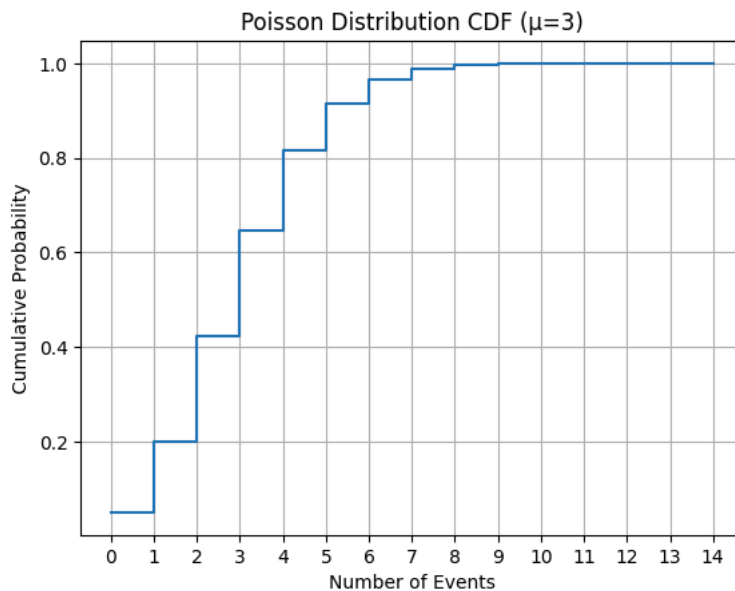
```
def plot_poisson_cdf(mu):
    """
    Calculates and visualizes the cumulative distribution function (CDF) of a Poisson distribution.

    Args:
        mu: Mean of the Poisson distribution.
    """
    x = np.arange(0, 15) # Values for the x-axis
    cdf = poisson.cdf(x, mu) # Calculate CDF using poisson.cdf

    plt.step(x, cdf, where='post') # Use 'step' for discrete CDF
    plt.title(f'Poisson Distribution CDF ( $\mu=\{mu\}$ )')
    plt.xlabel('Number of Events')
    plt.ylabel('Cumulative Probability')
    plt.xticks(x)
    plt.grid(True)
    plt.show()

# Example usage:
mu = 3
plot_poisson_cdf(mu)

# Interpretation
# The CDF shows the probability of observing up to a certain number of events in a given interval.
# For example, at x = 5, the CDF value indicates the probability of observing 5 or fewer events.
```



Q20. Simulate a random variable using a continuous uniform distribution and calculate its expected value

# prompt: Simulate a random variable using a continuous uniform distribution and calculate its expected value

```
import random
```

```
def simulate_uniform_and_calculate_expected_value(low, high, num_simulations):
```

```
    """
```

```
    Simulates a random variable using a continuous uniform distribution and calculates its expected value.
```

```
    Args:
```

```
        low: Lower bound of the uniform distribution.
```

```
        high: Upper bound of the uniform distribution.
```

```
        num_simulations: The number of times to simulate the random variable.
```

```
    Returns:
```

```
        The expected value of the simulated random variable.
```

```
    """
```

```
    simulated_values = [random.uniform(low, high) for _ in range(num_simulations)]
```

```
    expected_value = sum(simulated_values) / num_simulations
```

```
    return expected_value
```

```
# Example usage
```

```
low = 1
```

```
high = 5
```

```
num_simulations = 10000
```

```
expected_value = simulate_uniform_and_calculate_expected_value(low, high, num_simulations)
```

```
print(f"Expected value of the uniform distribution: {expected_value}")
```



```
Expected value of the uniform distribution: 3.008104213683438
```

Q21. Write a Python program to compare the standard deviations of two datasets and visualize the difference

# prompt: Write a Python program to compare the standard deviations of two datasets and visualize the difference

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
def compare_standard_deviations(data1, data2):
```

```
    """
```

```
    Compares the standard deviations of two datasets and visualizes the difference.
```

```
    Args:
```

```
        data1: The first dataset (list or numpy array).
```

```
        data2: The second dataset (list or numpy array).
```

```

"""
std_dev1 = np.std(data1)
std_dev2 = np.std(data2)

print(f"Standard Deviation of Dataset 1: {std_dev1:.2f}")
print(f"Standard Deviation of Dataset 2: {std_dev2:.2f}")

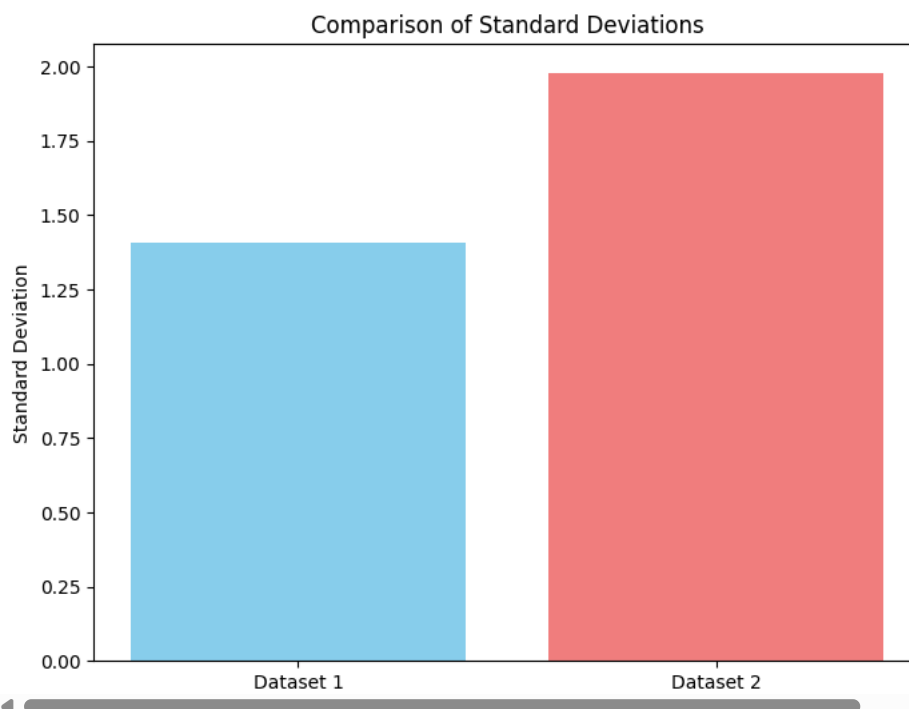
# Visualization
plt.figure(figsize=(8, 6))
plt.bar(['Dataset 1', 'Dataset 2'], [std_dev1, std_dev2], color=['skyblue', 'lightcoral'])
plt.ylabel('Standard Deviation')
plt.title('Comparison of Standard Deviations')
plt.show()

# Example usage
data1 = np.random.normal(loc=5, scale=1.5, size=100) # Example dataset 1
data2 = np.random.normal(loc=7, scale=2.0, size=100) # Example dataset 2

compare_standard_deviations(data1, data2)

```

→ Standard Deviation of Dataset 1: 1.41  
Standard Deviation of Dataset 2: 1.98



✓ Q22. Calculate the range and interquartile range (IQR) of a dataset generated from a normal distribution

# prompt: Calculate the range and interquartile range (IQR) of a dataset generated from a normal distribution

```

import numpy as np

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

    Args:
        data: A list or numpy array of numerical data.

    Returns:
        A dictionary containing the range and IQR.
    """
    data = np.array(data) # Convert to numpy array for easier calculations
    data_range = np.ptp(data) # Peak-to-peak (range)
    q1 = np.percentile(data, 25) # 25th percentile (first quartile)
    q3 = np.percentile(data, 75) # 75th percentile (third quartile)
    iqr = q3 - q1 # Interquartile range

    return {"range": data_range, "iqr": iqr}

# Example usage (generating data from a normal distribution):
np.random.seed(0) # for reproducibility

```

```
data = np.random.normal(loc=50, scale=10, size=100) # Example normal distribution data
range_iqr_result = calculate_range_iqr(data)
range_iqr_result
```

```
{'range': np.float64(48.22744439821686), 'iqr': np.float64(13.809340351246114)}
```

## ✓ Q23. Implement Z-score normalization on a dataset and visualize its transformation

# prompt: Implement Z-score normalization on a dataset and visualize its transformation

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

def zscore_normalize(data):
    """
    Performs Z-score normalization on a dataset.

    Args:
        data: A list or numpy array of numerical data.

    Returns:
        A numpy array containing the Z-score normalized data.
    """
    data = np.array(data)
    mean = np.mean(data)
    std = np.std(data)
    normalized_data = (data - mean) / std
    return normalized_data

# Example usage:
# Generate some sample data (replace with your actual dataset)
np.random.seed(42)
data = np.random.normal(loc=50, scale=10, size=100)

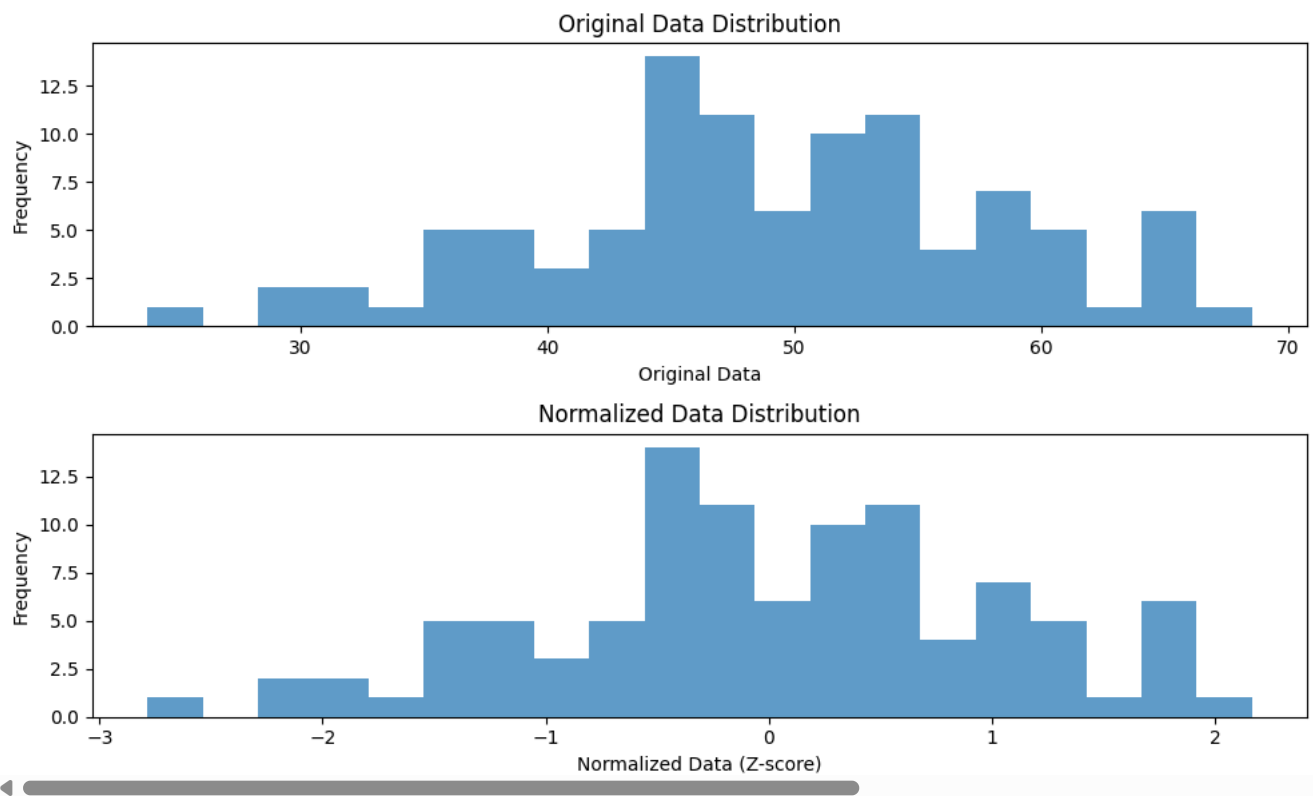
# Perform Z-score normalization
normalized_data = zscore_normalize(data)

# Visualize the transformation
plt.figure(figsize=(10, 6))

plt.subplot(2, 1, 1)
plt.hist(data, bins=20, alpha=0.7)
plt.xlabel("Original Data")
plt.ylabel("Frequency")
plt.title("Original Data Distribution")

plt.subplot(2, 1, 2)
plt.hist(normalized_data, bins=20, alpha=0.7)
plt.xlabel("Normalized Data (Z-score)")
plt.ylabel("Frequency")
plt.title("Normalized Data Distribution")

plt.tight_layout()
plt.show()
```



Q24. Write a Python function to calculate the skewness and kurtosis of a dataset generated from a normal distribution

# prompt: Write a Python function to calculate the skewness and kurtosis of a dataset generated from a normal distribution

```
from scipy.stats import skew, kurtosis

def calculate_skewness_kurtosis(data):
    """
    Calculates the skewness and kurtosis of a dataset.

    Args:
        data: A list or numpy array of numerical data.

    Returns:
        A dictionary containing the skewness and kurtosis.
    """
    skewness = skew(data)
    kurt = kurtosis(data) # Fisher's definition (excess kurtosis)

    return {"skewness": skewness, "kurtosis": kurt}

# Example usage:
np.random.seed(0) # for reproducibility
data = np.random.normal(loc=0, scale=1, size=1000) # Example normal distribution data
results = calculate_skewness_kurtosis(data)
results
```

```
{'skewness': np.float64(0.03385895323565712),
 'kurtosis': np.float64(-0.0467663244783294)}
```

## Practical Part-2

Q1. Write a Python program to perform a Z-test for comparing a sample mean to a known population mean and interpret the results

# prompt: Write a Python program to perform a Z-test for comparing a sample mean to a known population mean and interpret the results

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



```
def z_test(sample_mean, population_mean, population_std, sample_size, alpha=0.05):
    """
    Performs a one-sample Z-test for a population mean.

    Args:
        sample_mean: The mean of the sample.
        population_mean: The hypothesized population mean.
        population_std: The population standard deviation.
        sample_size: The size of the sample.
        alpha: The significance level (default is 0.05).

    Returns:
        A tuple containing:
        - The Z-statistic.
        - The p-value.
        - A string indicating whether to reject or fail to reject the null hypothesis.
    """

    # Calculate the Z-statistic
    z_statistic = (sample_mean - population_mean) / (population_std / (sample_size**0.5))

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

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

    return z_statistic, p_value, decision

# Example usage
sample_mean = 72 # Sample mean
population_mean = 70 # Hypothesized population mean
population_std = 5 # Population standard deviation
sample_size = 50 # Sample size
alpha = 0.05 # Significance level

z_statistic, p_value, decision = z_test(sample_mean, population_mean, population_std, sample_size, alpha)

print(f"Z-statistic: {z_statistic:.2f}")
print(f"P-value: {p_value:.3f}")
decision
```

```
↗ Z-statistic: 2.83
P-value: 0.005
'Reject the null hypothesis'
```

## ✓ Q2. Simulate random data to perform hypothesis testing and calculate the corresponding P-value using Python

# prompt: Simulate random data to perform hypothesis testing and calculate the corresponding P-value using Python

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

# Generate two random samples
sample1 = np.random.normal(loc=10, scale=2, size=50)
sample2 = np.random.normal(loc=12, scale=2, size=50)

# Perform an independent samples t-test
t_statistic, p_value = ttest_ind(sample1, sample2)

print(f"T-statistic: {t_statistic:.3f}")
print(f"P-value: {p_value:.3f}")

# Interpret the results
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis")
else:
    print("Fail to reject the null hypothesis")
```

```
↗ T-statistic: -6.149
P-value: 0.000
Reject the null hypothesis
```

### Q3. Implement a one-sample Z-test using Python to compare the sample mean with the population mean

# prompt: Implement a one-sample Z-test using Python to compare the sample mean with the population mean

```
from scipy.stats import norm

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

    Args:
        sample_data: A list or numpy array of sample data.
        population_mean: The hypothesized population mean.
        population_std: The population standard deviation.
        alpha: The significance level (default is 0.05).

    Returns:
        A dictionary containing the test statistic, p-value, and conclusion.
    """
    import numpy as np
    sample_mean = np.mean(sample_data)
    sample_size = len(sample_data)
    z_score = (sample_mean - population_mean) / (population_std / np.sqrt(sample_size))
    p_value = 2 * (1 - norm.cdf(abs(z_score))) # Two-tailed test

    conclusion = "Fail to reject the null hypothesis"
    if p_value < alpha:
        conclusion = "Reject the null hypothesis"

    return {"z_score": z_score, "p_value": p_value, "conclusion": conclusion}

# Example usage
sample_data = [72, 75, 68, 70, 73, 71, 74, 69, 76, 72] # Example sample data
population_mean = 70 # Hypothesized population mean
population_std = 3 # Population standard deviation
alpha = 0.05 # Significance level

results = one_sample_z_test(sample_data, population_mean, population_std, alpha)
results
```

```
{'z_score': np.float64(2.1081851067789197),
 'p_value': np.float64(0.035014981019662494),
 'conclusion': 'Reject the null hypothesis'}
```

### Q4. Perform a two-tailed Z-test using Python and visualize the decision region on a plot

# prompt: Perform a two-tailed Z-test using Python and visualize the decision region on a plot

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

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

    Args:
        sample_mean: The sample mean.
        population_mean: The population mean.
        population_std: The population standard deviation.
        sample_size: The sample size.
        alpha: The significance level (default is 0.05).
    """

    z_score = (sample_mean - population_mean) / (population_std / np.sqrt(sample_size))
    p_value = 2 * (1 - norm.cdf(abs(z_score)))

    # Visualization
    x = np.linspace(-4, 4, 500)
    y = norm.pdf(x, 0, 1)
    plt.plot(x, y)

    z_critical = norm.ppf(1 - alpha / 2)
    plt.fill_between(x[x > z_critical], y[x > z_critical], color='red', alpha=0.3, label="Rejection Region")
    plt.fill_between(x[x < -z_critical], y[x < -z_critical], color='red', alpha=0.3)
```

```

plt.axvline(x=z_score, color='blue', linestyle='--', label=f"Z-score = {z_score:.2f}")
plt.axvline(x=z_critical, color='green', linestyle='--', label=f"Critical Value = {z_critical:.2f}")
plt.axvline(x=-z_critical, color='green', linestyle='--')

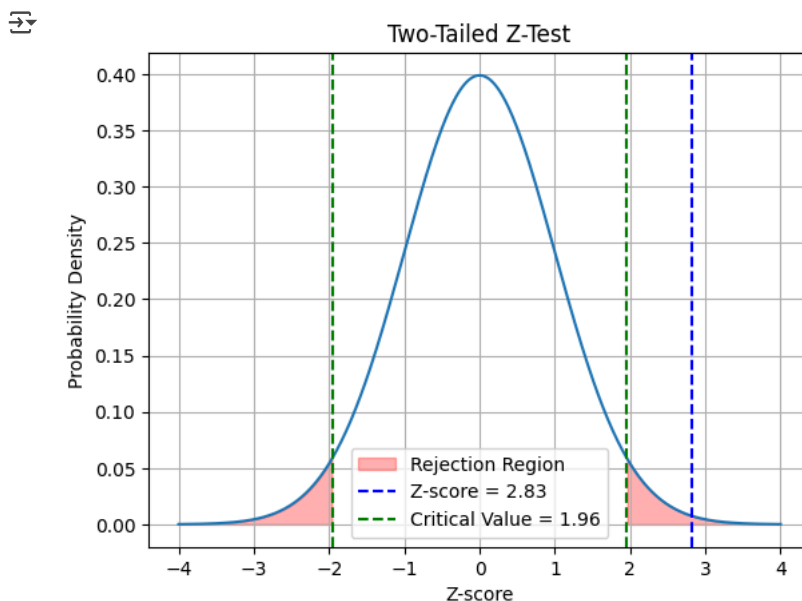
plt.xlabel("Z-score")
plt.ylabel("Probability Density")
plt.title("Two-Tailed Z-Test")
plt.legend()
plt.grid(True)
plt.show()

print(f"Z-score: {z_score:.2f}")
print(f"P-value: {p_value:.3f}")

if p_value < alpha:
    print("Reject the null hypothesis")
else:
    print("Fail to reject the null hypothesis")

# Example usage:
sample_mean = 72
population_mean = 70
population_std = 5
sample_size = 50
alpha = 0.05
two_tailed_z_test(sample_mean, population_mean, population_std, sample_size, alpha)

```



Z-score: 2.83  
P-value: 0.005  
Reject the null hypothesis

## Q5. Create a Python function that calculates and visualizes Type 1 and Type 2 errors during hypothesis testing

# prompt: Create a Python function that calculates and visualizes Type 1 and Type 2 errors during hypothesis testing

```

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

def visualize_errors(mu0, mu1, sigma, n, alpha):
    """
    Calculates and visualizes Type 1 and Type 2 errors.

    Args:
        mu0: True population mean under the null hypothesis.
        mu1: True population mean under the alternative hypothesis.
        sigma: Population standard deviation.
        n: Sample size.
        alpha: Significance level.
    """

```

```

# Calculate the critical z-value
z_critical = norm.ppf(1 - alpha / 2)

# Calculate the standard error
se = sigma / np.sqrt(n)

# Calculate the rejection region boundaries
lower_bound = mu0 - z_critical * se
upper_bound = mu0 + z_critical * se

# Generate x-values for plotting
x = np.linspace(mu0 - 4 * se, mu1 + 4 * se, 500)

# Plot the distributions
plt.plot(x, norm.pdf(x, loc=mu0, scale=se), label="Null Hypothesis (H0)")
plt.plot(x, norm.pdf(x, loc=mu1, scale=se), label="Alternative Hypothesis (H1)")

# Shade the rejection regions
plt.fill_between(x[x < lower_bound], norm.pdf(x[x < lower_bound], loc=mu0, scale=se), color='red', alpha=0.3, label='Type I Error (Fa
plt.fill_between(x[x > upper_bound], norm.pdf(x[x > upper_bound], loc=mu0, scale=se), color='red', alpha=0.3)

# Shade the Type II error region
plt.fill_between(x[(x > lower_bound) & (x < upper_bound)], norm.pdf(x[(x > lower_bound) & (x < upper_bound)], loc=mu1, scale=se), col

# Mark the critical values and means
plt.axvline(lower_bound, color='black', linestyle='--', label="Rejection Boundaries")
plt.axvline(upper_bound, color='black', linestyle='--')
plt.axvline(mu0, color='green', linestyle='--', label="True Mean (H0)")
plt.axvline(mu1, color='orange', linestyle='--', label="True Mean (H1)")

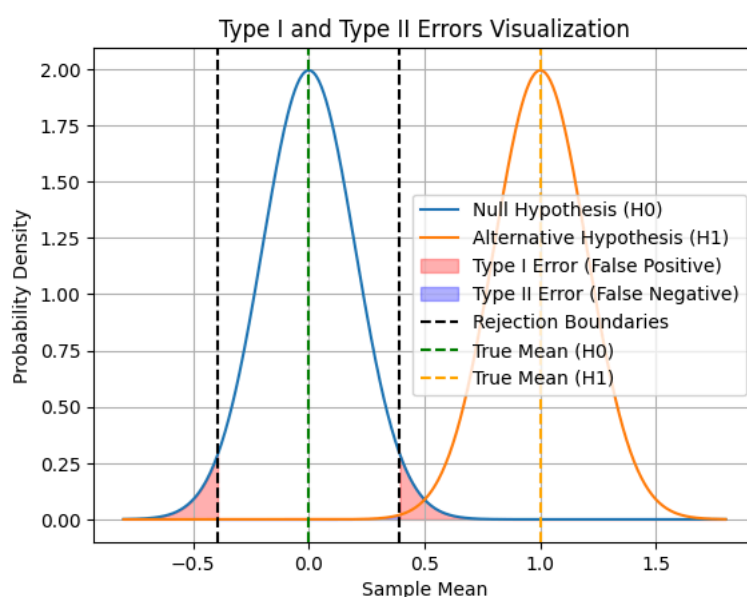
plt.xlabel("Sample Mean")
plt.ylabel("Probability Density")
plt.title("Type I and Type II Errors Visualization")
plt.legend()
plt.grid(True)
plt.show()

# Calculate Type II error (beta)
beta = norm.cdf(upper_bound, loc=mu1, scale=se) - norm.cdf(lower_bound, loc=mu1, scale=se)
print(f"Type II Error (Beta) = {beta:.3f}")

# Example
mu0 = 0
mu1 = 1
sigma = 2
n = 100
alpha = 0.05

visualize_errors(mu0, mu1, sigma, n, alpha)

```



Type II Error (Beta) = 0.001

✓ Q6. Write a Python program to perform an independent T-test and interpret the results

# prompt: Write a Python program to perform an independent T-test and interpret the results

```
from scipy.stats import ttest_ind

def perform_independent_t_test(data1, data2, alpha=0.05):
    """
    Performs an independent two-sample t-test.

    Args:
        data1: First dataset (list or numpy array).
        data2: Second dataset (list or numpy array).
        alpha: Significance level (default is 0.05).

    Returns:
        A dictionary containing the t-statistic, p-value, and interpretation.
    """

    t_statistic, p_value = ttest_ind(data1, data2)

    interpretation = ""
    if p_value < alpha:
        interpretation = "Reject the null hypothesis. There is a statistically significant difference between the means of the two groups."
    else:
        interpretation = "Fail to reject the null hypothesis. There is no statistically significant difference between the means of the two groups."

    return {
        "t_statistic": t_statistic,
        "p_value": p_value,
        "interpretation": interpretation
    }

# Example usage:
data1 = [10, 12, 15, 11, 13]
data2 = [18, 20, 19, 17, 22]
results = perform_independent_t_test(data1, data2)
results
```

```
{'t_statistic': np.float64(-5.753964555687505),
 'p_value': np.float64(0.0004270514763262176),
 'interpretation': 'Reject the null hypothesis. There is a statistically significant difference between the means of the two groups.'}
```

## ✓ Q7. Perform a paired sample t-test using Python and visualize the comparison results

# prompt: Perform a paired sample t-test using Python and visualize the comparison results

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

def paired_sample_t_test_with_visualization(data1, data2, alpha=0.05):
    """
    Performs a paired sample t-test and visualizes the results.

    Args:
        data1: First dataset (list or numpy array).
        data2: Second dataset (list or numpy array).
        alpha: Significance level (default is 0.05).
    """

    # Perform the paired t-test
    t_statistic, p_value = ttest_rel(data1, data2)

    # Visualization
    plt.figure(figsize=(8, 6))

    plt.scatter(data1, data2, label="Data Points", alpha=0.7)
    plt.xlabel("Dataset 1")
    plt.ylabel("Dataset 2")
    plt.title("Paired Sample T-Test Visualization")
    plt.plot([min(data1), max(data1)], [min(data1), max(data1)], color="red", linestyle="--", label="Equality Line") # Add equality line

    # Add text annotations for statistics
    plt.text(0.05, 0.95, f"t-statistic = {t_statistic:.2f}", transform=plt.gca().transAxes)
    plt.text(0.05, 0.90, f"p-value = {p_value:.3f}", transform=plt.gca().transAxes)

    if p_value < alpha:
        plt.text(0.05, 0.85, "Reject the null hypothesis", transform=plt.gca().transAxes)
```



```

else:
    plt.text(0.05, 0.85, "Fail to reject the null hypothesis", transform=plt.gca().transAxes)

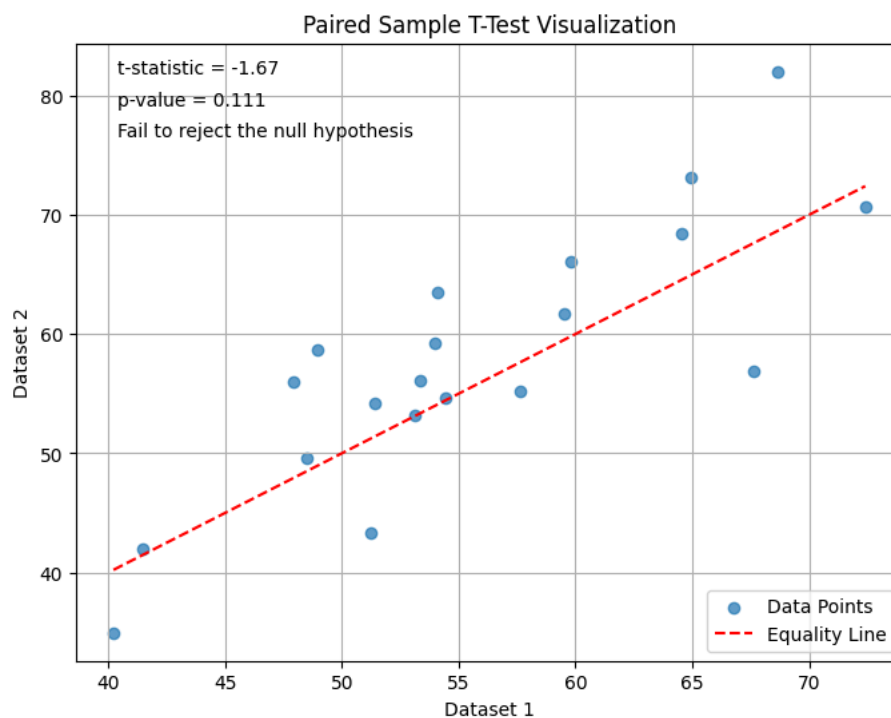
plt.legend()
plt.grid(True)
plt.show()

# Interpret the results
if p_value < alpha:
    print("Reject the null hypothesis. There is a statistically significant difference between the means of the paired samples.")
else:
    print("Fail to reject the null hypothesis. There is no statistically significant difference between the means of the paired samples.")

# Example usage
np.random.seed(0) # for reproducibility
data1 = np.random.normal(loc=50, scale=10, size=20)
data2 = data1 + np.random.normal(loc=2, scale=5, size=20)

paired_sample_t_test_with_visualization(data1, data2)

```



Fail to reject the null hypothesis. There is no statistically significant difference between the means of the paired samples.

## Q8. Simulate data and perform both Z-test and T-test, then compare the results using Python

# prompt: Simulate data and perform both Z-test and T-test, then compare the results using Python

```

import numpy as np
from scipy.stats import norm, ttest_1samp, t

# Simulate data from a normal distribution
np.random.seed(42) # for reproducibility
population_mean = 50
population_std = 10
sample_size = 100
sample_data = np.random.normal(loc=population_mean, scale=population_std, size=sample_size)
sample_mean = np.mean(sample_data)

# Z-test
def z_test(sample_mean, population_mean, population_std, sample_size, alpha=0.05):
    z = (sample_mean - population_mean) / (population_std / np.sqrt(sample_size))
    p_value = 2 * (1 - norm.cdf(abs(z))) # Two-tailed test
    return z, p_value

z_statistic, z_p_value = z_test(sample_mean, population_mean, population_std, sample_size)
print(f"Z-test Results: z = {z_statistic:.2f}, p = {z_p_value:.3f}")

# T-test
t_statistic, t_p_value = ttest_1samp(sample_data, population_mean)

```

```
print(f"T-test Results: t = {t_statistic:.2f}, p = {t_p_value:.3f}")
```

```
# Compare results
print("\nComparison:")
print(f"Z-test p-value: {z_p_value:.3f}")
print(f"T-test p-value: {t_p_value:.3f}")

if z_p_value < 0.05 and t_p_value < 0.05:
    print("Both tests reject the null hypothesis.")
elif z_p_value >= 0.05 and t_p_value >= 0.05:
    print("Both tests fail to reject the null hypothesis.")
else:
    print("Tests yield different conclusions.")
```

```
→ Z-test Results: z = -1.04, p = 0.299
   T-test Results: t = -1.14, p = 0.256

Comparison:
Z-test p-value: 0.299
T-test p-value: 0.256
Both tests fail to reject the null hypothesis.
```

## ✓ Q9. Write a Python function to calculate the confidence interval for a sample mean and explain its significance

# prompt: Write a Python function to calculate the confidence interval for a sample mean and explain its significance

```
from scipy.stats import t
```

```
def confidence_interval_sample_mean(data, confidence=0.95):
    """
    Calculates the confidence interval for a sample mean.

    Args:
        data: A list or numpy array of numerical data.
        confidence: The desired confidence level (e.g., 0.95 for a 95% confidence interval).

    Returns:
        A tuple containing the lower and upper bounds of the confidence interval,
        or None if the input data is invalid.
    """
    import numpy as np
    if not isinstance(data, (list, np.ndarray)):
        print("Error: Input data must be a list or numpy array.")
        return None

    if len(data) < 2:
        print("Error: Input data must contain at least two elements.")
        return None

    a = 1.0 * np.array(data)
    n = len(a)
    m, se = np.mean(a), np.std(a, ddof=1) / np.sqrt(n)
    h = se * t.ppf((1 + confidence) / 2., n - 1)
    return m - h, m + h
```

# Significance of Confidence Interval:  
 # The confidence interval provides a range of plausible values for the population mean.  
 # A 95% confidence interval, for instance, means that if we were to repeatedly sample from the population and construct confidence intervals, 95% of them would contain the true population mean.  
 # It quantifies the uncertainty associated with estimating the population mean from a sample. A narrower interval indicates higher precision.

```
# Example Usage
data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
lower, upper = confidence_interval_sample_mean(data)
if lower is not None and upper is not None:
    print(f"The 95% confidence interval is: ({lower:.2f}, {upper:.2f})")
```

```
→ The 95% confidence interval is: (3.33, 7.67)
```

## ✓ Q10. Write a Python program to calculate the margin of error for a given confidence level using sample data

# prompt: Write a Python program to calculate the margin of error for a given confidence level using sample data

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


def margin_of_error(data, confidence=0.95):
    """
    Calculates the margin of error for a given confidence level using sample data.

    Args:
        data: A list or numpy array of numerical data.
        confidence: The desired confidence level (e.g., 0.95 for a 95% confidence interval).

    Returns:
        The margin of error.
    """
    a = 1.0 * np.array(data)
    n = len(a)
    m, se = np.mean(a), np.std(a, ddof=1) / np.sqrt(n)
    h = se * t.ppf((1 + confidence) / 2., n - 1)
    return h

# Example usage:
data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] # Example dataset
confidence_level = 0.95
error = margin_of_error(data, confidence=confidence_level)

print(f"The margin of error for a {confidence_level*100:.0f}% confidence interval is: {error:.2f}")
```

 The margin of error for a 95% confidence interval is: 2.17

## Q11. Implement a Bayesian inference method using Bayes' Theorem in Python and explain the process

# prompt: Implement a Bayesian inference method using Bayes' Theorem in Python and explain the process

```
import numpy as np
import math

def bayesian_inference(prior, likelihood, evidence):
    """
    Performs Bayesian inference using Bayes' Theorem.

    Args:
        prior: A dictionary representing the prior probabilities of hypotheses.
        likelihood: A dictionary representing the likelihood of observing the evidence given each hypothesis.
        evidence: The observed evidence.

    Returns:
        A dictionary representing the posterior probabilities of hypotheses.
    """

    posterior = {}
    for hypothesis, prior_prob in prior.items():
        # Bayes' Theorem: P(H|E) = [P(E|H) * P(H)] / P(E)
        posterior[hypothesis] = (likelihood[hypothesis] * prior_prob)

    # Normalize the posterior probabilities so they sum to 1
    evidence_prob = sum(posterior.values())
    for hypothesis in posterior:
        posterior[hypothesis] /= evidence_prob

    return posterior

# Example usage
# Prior probabilities of hypotheses (e.g., a coin being fair or biased)
prior = {'Fair': 0.5, 'Biased': 0.5}

# Likelihood of observing 5 heads in 10 coin tosses given each hypothesis
# Use math.factorial instead of np.math.factorial
likelihood = {'Fair': math.factorial(10)/(math.factorial(5)*math.factorial(5)) * (0.5**10),
              'Biased': math.factorial(10)/(math.factorial(5)*math.factorial(5)) * (0.7**5) * (0.3**5)} # Assume biased coin has p=0.7

# Observed evidence (5 heads in 10 tosses)
evidence = '5 heads in 10 tosses'

posterior = bayesian_inference(prior, likelihood, evidence)
```

```
print("Posterior Probabilities:")
for hypothesis, prob in posterior.items():
    print(f"Hypothesis '{hypothesis}': {prob: .4f}")
```

➦ Posterior Probabilities:  
Hypothesis 'Fair': 0.7051  
Hypothesis 'Biased': 0.2949

## ✓ Q12. Perform a Chi-square test for independence between two categorical variable in Python

```
# prompt: Perform a Chi-square test for independence between two categorical variable in Python

from scipy.stats import chi2_contingency

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

    Args:
        data: A 2D list or numpy array representing the contingency table.

    Returns:
        A dictionary containing the chi-square statistic, p-value, degrees of freedom,
        and expected frequencies.
    """

    chi2, p, dof, expected = chi2_contingency(data)

    return {
        "chi2_statistic": chi2,
        "p_value": p,
        "degrees_of_freedom": dof,
        "expected_frequencies": expected
    }

# Example usage
# Create a contingency table (replace with your data)
contingency_table = [
    [20, 15], # Observed frequencies for the first category
    [10, 25] # Observed frequencies for the second category
]

# Perform Chi-square test
results = chi_square_test_independence(contingency_table)

# Print the results
print("Chi-Square Test Results:")
for key, value in results.items():
    print(f"{key.replace('_', ' ').title()}: {value}")

# Interpretation of results:
# If the p-value is less than a predetermined significance level (alpha, often 0.05),
# you reject the null hypothesis, concluding that the two variables are dependent.
# Otherwise, you fail to reject the null hypothesis, suggesting there is not enough evidence to claim a relationship.
```

➦ Chi-Square Test Results:  
Chi2 Statistic: 4.725  
P Value: 0.029727183306054526  
Degrees Of Freedom: 1  
Expected Frequencies: [[15. 20.]  
[15. 20.]]

## ✓ Q13. Write a Python program to calculate the expected frequencies of a Chi-square test based on observed data

```
# prompt: Write a Python program to calculate the expected frequencies of a Chi-square test based on observed data

import numpy as np
from scipy.stats import chi2_contingency
```