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

Ans:-

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

# Function to perform a one-sample Z-test
def z_test(sample, population_mean, population_std, alpha=0.05):
    # Calculate sample mean and sample size
    sample_mean = np.mean(sample)
    n = len(sample)

    # Calculate the Z-score
    z_score = (sample_mean - population_mean) / (population_std /
np.sqrt(n))

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

    # Print results
    print(f"Sample Mean: {sample_mean:.2f}")
    print(f"Population Mean: {population_mean:.2f}")
    print(f"Z-Score: {z_score:.2f}")
    print(f"P-Value: {p_value:.4f}")

    # Interpret the results
    if p_value < alpha:
        print("Reject the null hypothesis: There is a significant
difference between the sample mean and the population mean.")
    else:
        print("Fail to reject the null hypothesis: There is no
significant difference between the sample mean and the population
mean.")

# Example usage
if __name__ == "__main__":
    # Sample data
    sample_data = [20, 22, 19, 24, 21, 23, 20, 22, 19, 21]

    # Known population parameters
    population_mean = 21  # Known population mean
    population_std = 2    # Known population standard deviation

    # Perform the Z-test
    z_test(sample_data, population_mean, population_std)
```

```
Sample Mean: 21.10
Population Mean: 21.00
```

```
Z-Score: 0.16
P-Value: 0.8744
Fail to reject the null hypothesis: There is no significant difference
between the sample mean and the population mean.
```

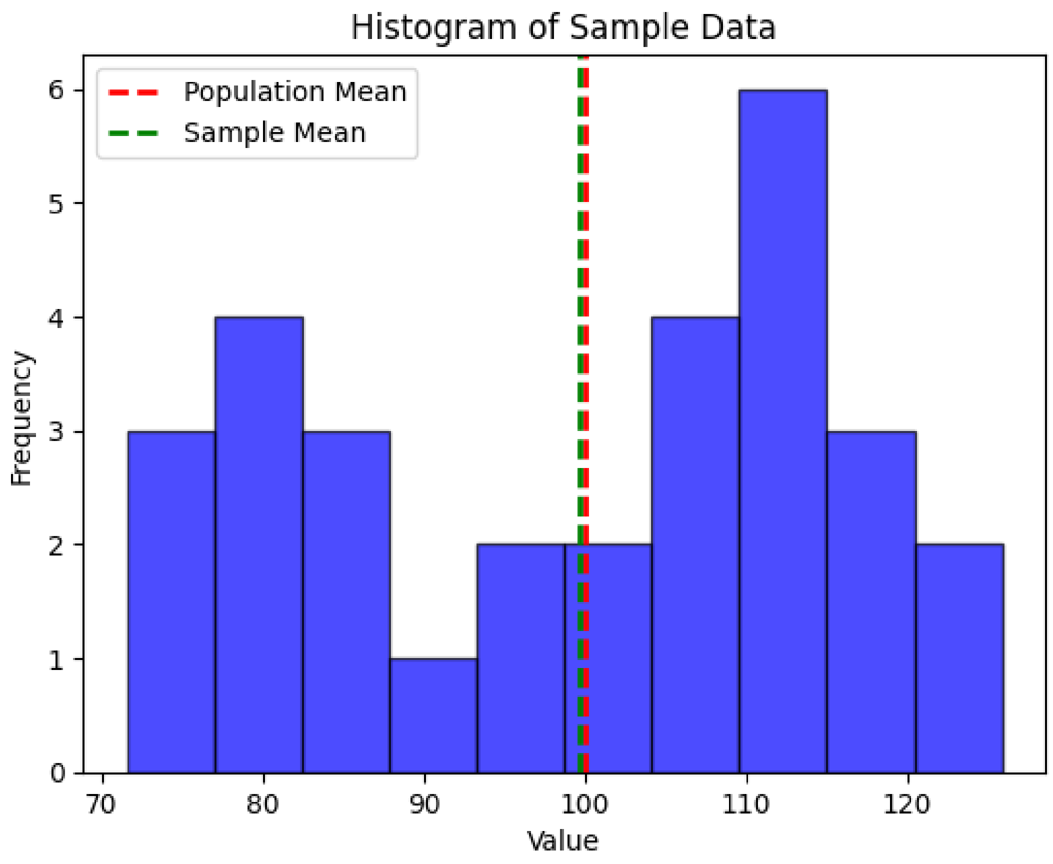
Q2.Simulate random data to perform hypothesis testing and calculate the corresponding P-value using Python?

Ans:-

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

# Function to perform a one-sample Z-test
def z_test(sample, population_mean, population_std, alpha=0.05):
    # Calculate sample mean and sample size
    sample_mean = np.mean(sample)
    n = len(sample)

    # Calculate the Z-score
    z_score = (sample_mean - population_mean) / (population_std /
np.sqrt(n))

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

    # Print results
    print(f"Sample Mean: {sample_mean:.2f}")
    print(f"Population Mean: {population_mean:.2f}")
    print(f"Z-Score: {z_score:.2f}")
    print(f"P-Value: {p_value:.4f}")

    # Interpret the results
    if p_value < alpha:
        print("Reject the null hypothesis: There is a significant
difference between the sample mean and the population mean.")
    else:
        print("Fail to reject the null hypothesis: There is no
significant difference between the sample mean and the population
mean.")

# Simulate random data
def simulate_data(sample_size, population_mean, population_std):
    # Generate random sample data from a normal distribution
    sample_data = np.random.normal(loc=population_mean,
scale=population_std, size=sample_size)
    return sample_data

# Example usage
```

```python
if __name__ == "__main__":
    # Parameters for simulation
    sample_size = 30  # Size of the sample
    population_mean = 100  # Known population mean
    population_std = 15  # Known population standard deviation

    # Simulate random data
    sample_data = simulate_data(sample_size, population_mean,
population_std)

    # Perform the Z-test
    z_test(sample_data, population_mean, population_std)

    # Optional: Visualize the sample data
    plt.hist(sample_data, bins=10, alpha=0.7, color='blue',
edgecolor='black')
    plt.axvline(population_mean, color='red', linestyle='dashed',
linewidth=2, label='Population Mean')
    plt.axvline(np.mean(sample_data), color='green',
linestyle='dashed', linewidth=2, label='Sample Mean')
    plt.title('Histogram of Sample Data')
    plt.xlabel('Value')
    plt.ylabel('Frequency')
    plt.legend()
    plt.show()
```

```
Sample Mean: 99.60
Population Mean: 100.00
Z-Score: -0.14
P-Value: 0.8849
Fail to reject the null hypothesis: There is no significant difference
between the sample mean and the population mean.
```



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

Ans:-

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

# Function to perform a one-sample Z-test
def one_sample_z_test(sample, population_mean, population_std,
alpha=0.05):
    # Calculate sample mean and sample size
    sample_mean = np.mean(sample)
    n = len(sample)

    # Calculate the Z-score
    z_score = (sample_mean - population_mean) / (population_std /
np.sqrt(n))

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

    # Print results
    print(f"Sample Mean: {sample_mean:.2f}")
    print(f"Population Mean: {population_mean:.2f}")
    print(f"Z-Score: {z_score:.2f}")
    print(f"P-Value: {p_value:.4f}")

    # Interpret the results
    if p_value < alpha:
        print("Reject the null hypothesis: There is a significant 
difference between the sample mean and the population mean.")
    else:
        print("Fail to reject the null hypothesis: There is no 
significant difference between the sample mean and the population 
mean.")

# Example usage
if __name__ == "__main__":
    # Sample data (you can modify this data)
    sample_data = [102, 98, 101, 105, 99, 100, 97, 104, 103, 101]

    # Known population parameters
    population_mean = 100  # Known population mean
    population_std = 10    # Known population standard deviation

    # Perform the one-sample Z-test
    one_sample_z_test(sample_data, population_mean, population_std)

Sample Mean: 101.00
Population Mean: 100.00
Z-Score: 0.32
P-Value: 0.7518
Fail to reject the null hypothesis: There is no significant difference 
between the sample mean and the population mean.
```

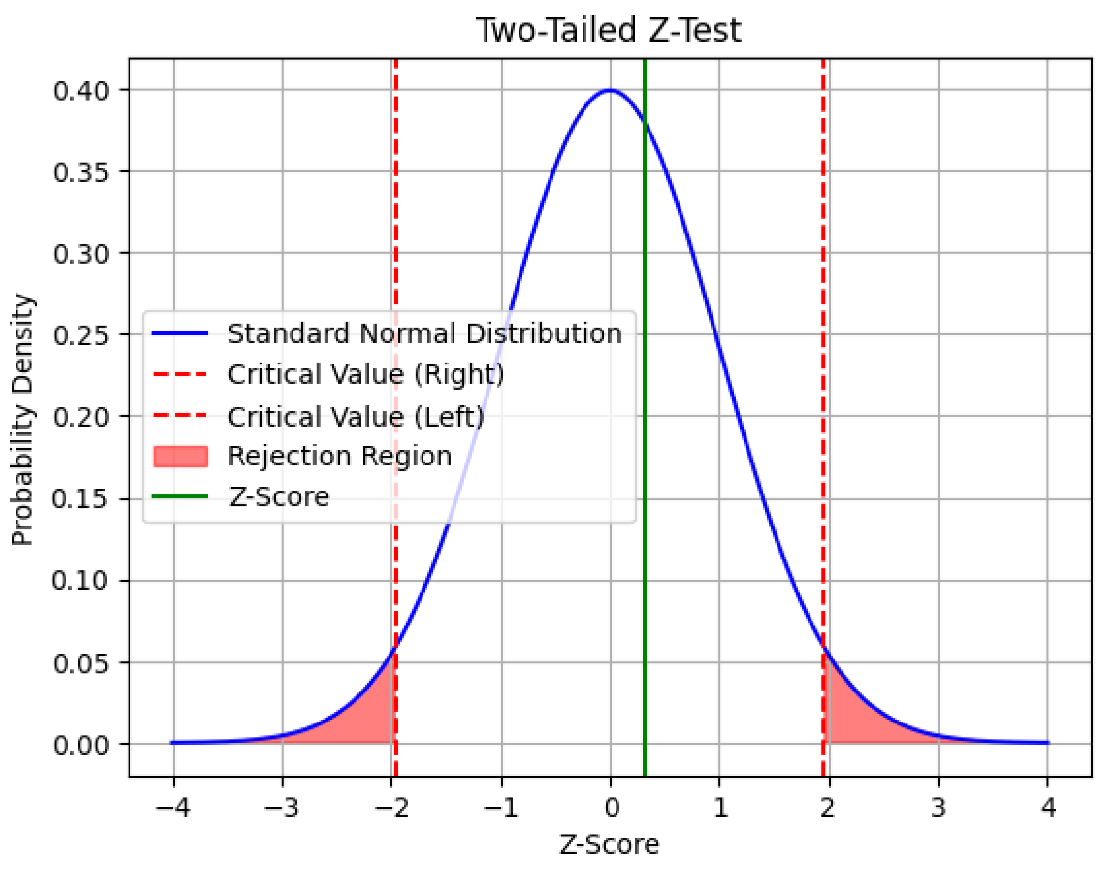
Q4.Perform a two-tailed Z-test using Python and visualize the decision region on a plot?

Ans:-

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

# Function to perform a two-tailed Z-test
def two_tailed_z_test(sample, population_mean, population_std, 
alpha=0.05):
    # Calculate sample mean and sample size
    sample_mean = np.mean(sample)
    n = len(sample)

    # Calculate the Z-score
    z_score = (sample_mean - population_mean) / (population_std / 
np.sqrt(n))
```

```
    # Calculate critical values
    critical_val1 = norm.ppf(1 - alpha / 2)
    critical_val2 = norm.ppf(alpha / 2)

    # Print results
    print(f"Sample Mean: {sample_mean:.2f}")
    print(f"Population Mean: {population_mean:.2f}")
    print(f"Z-Score: {z_score:.2f}")

    # Decision based on Z-score
    if abs(z_score) > critical_val1:
        print("Reject the null hypothesis: There is a significant
difference.")
    else:
        print("Fail to reject the null hypothesis: There is no
significant difference.")

    # Visualization
    x = np.linspace(-4, 4, 1000)
    y = norm.pdf(x, 0, 1)

    plt.plot(x, y, label='Standard Normal Distribution', color='blue')
    plt.axvline(x=critical_val1, color='red', linestyle='--',
label='Critical Value (Right)')
    plt.axvline(x=critical_val2, color='red', linestyle='--',
label='Critical Value (Left)')
    plt.fill_between(x, y, where=(x <= critical_val2) | (x >=
critical_val1), color='red', alpha=0.5, label='Rejection Region')
    plt.axvline(x=z_score, color='green', linestyle='-', label='Z-
Score')

    plt.title('Two-Tailed Z-Test')
    plt.xlabel('Z-Score')
    plt.ylabel('Probability Density')
    plt.legend()
    plt.grid()
    plt.show()

# Example usage
if __name__ == "__main__":
    # Sample data (you can modify this data)
    sample_data = [102, 98, 101, 105, 99, 100, 97, 104, 103, 101]

    # Known population parameters
    population_mean = 100  # Known population mean
    population_std = 10    # Known population standard deviation

    # Perform the two-tailed Z-test
    two_tailed_z_test(sample_data, population_mean, population_std)
```

```
Sample Mean: 101.00
Population Mean: 100.00
Z-Score: 0.32
Fail to reject the null hypothesis: There is no significant
difference.
```

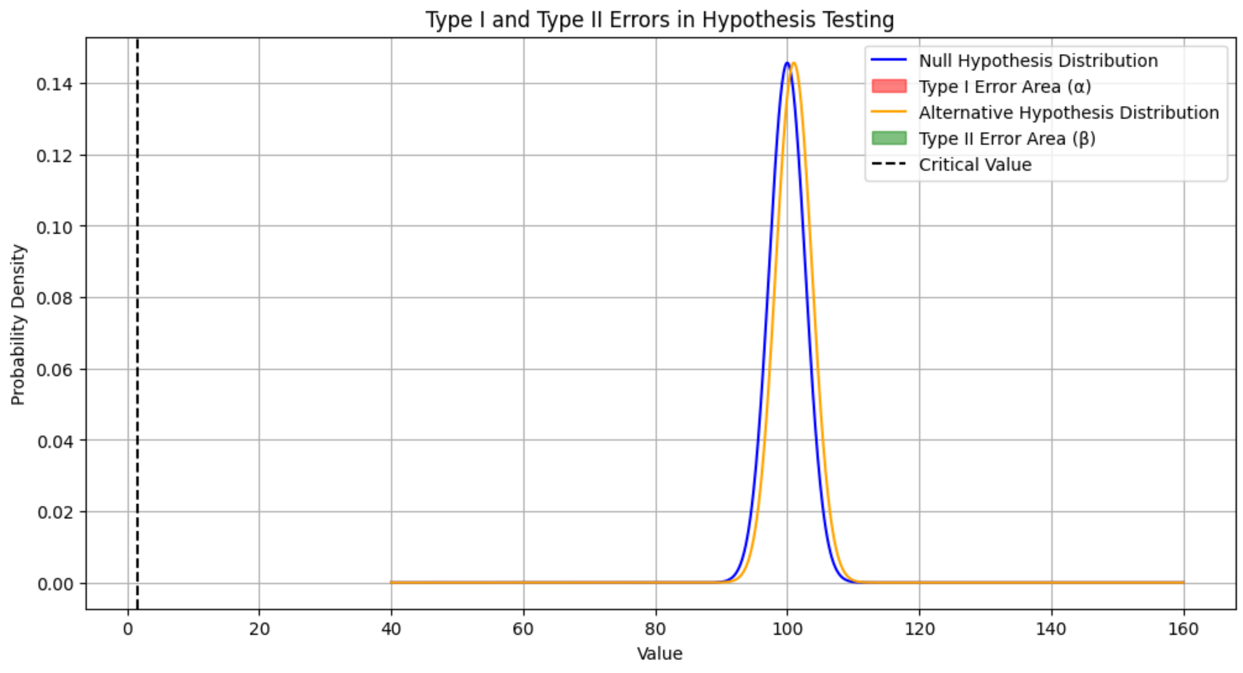


Q5.Create a Python function that calculates and visualizes Type 1 and Type 2 errors during hypothesis testing?

Ans:-

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

def visualize_errors(population_mean, population_std, sample_size,
alpha=0.05):
    # Define the null and alternative hypotheses
    null_hypothesis_mean = population_mean
    alternative_hypothesis_mean = population_mean + 1  # Shift the
mean for the alternative hypothesis

    # Calculate critical value for Type I error
    critical_value = norm.ppf(1 - alpha)

    # Generate x values for the normal distributions
    x = np.linspace(population_mean - 4 * population_std,
population_mean + 4 * population_std, 1000)

    # Calculate the probability density functions for the null and
alternative hypotheses
    null_pdf = norm.pdf(x, null_hypothesis_mean, population_std /
np.sqrt(sample_size))
    alternative_pdf = norm.pdf(x, alternative_hypothesis_mean,
population_std / np.sqrt(sample_size))

    # Calculate Type I error area (alpha)
    type_1_error_area = norm.cdf(critical_value)

    # Calculate Type II error area (beta)
    beta = norm.cdf(critical_value, loc=alternative_hypothesis_mean,
scale=population_std / np.sqrt(sample_size))

    # Plotting
    plt.figure(figsize=(12, 6))

    # Plot null hypothesis distribution
    plt.plot(x, null_pdf, label='Null Hypothesis Distribution',
color='blue')
    plt.fill_between(x, null_pdf, where=(x <= critical_value),
color='red', alpha=0.5, label='Type I Error Area (α)')

    # Plot alternative hypothesis distribution
    plt.plot(x, alternative_pdf, label='Alternative Hypothesis
Distribution', color='orange')
    plt.fill_between(x, alternative_pdf, where=(x <= critical_value),
color='green', alpha=0.5, label='Type II Error Area (β)')

    # Add critical value line
    plt.axvline(x=critical_value, color='black', linestyle='--',
label='Critical Value')

    # Add labels and legend
    plt.title('Type I and Type II Errors in Hypothesis Testing')
    plt.xlabel('Value')
    plt.ylabel('Probability Density')
    plt.legend()
    plt.grid()
    plt.show()

    # Print Type I and Type II error probabilities
    print(f"Type I Error (α): {type_1_error_area:.4f}")
    print(f"Type II Error (β): {beta:.4f}")
```

```python
# Example usage
if __name__ == "__main__":
    population_mean = 100  # Known population mean
    population_std = 15    # Known population standard deviation
    sample_size = 30       # Sample size
    alpha = 0.05           # Significance level

    visualize_errors(population_mean, population_std, sample_size, 
alpha)
```



```
Type I Error (α): 0.9500
Type II Error (β): 0.0000
```

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

Ans:-

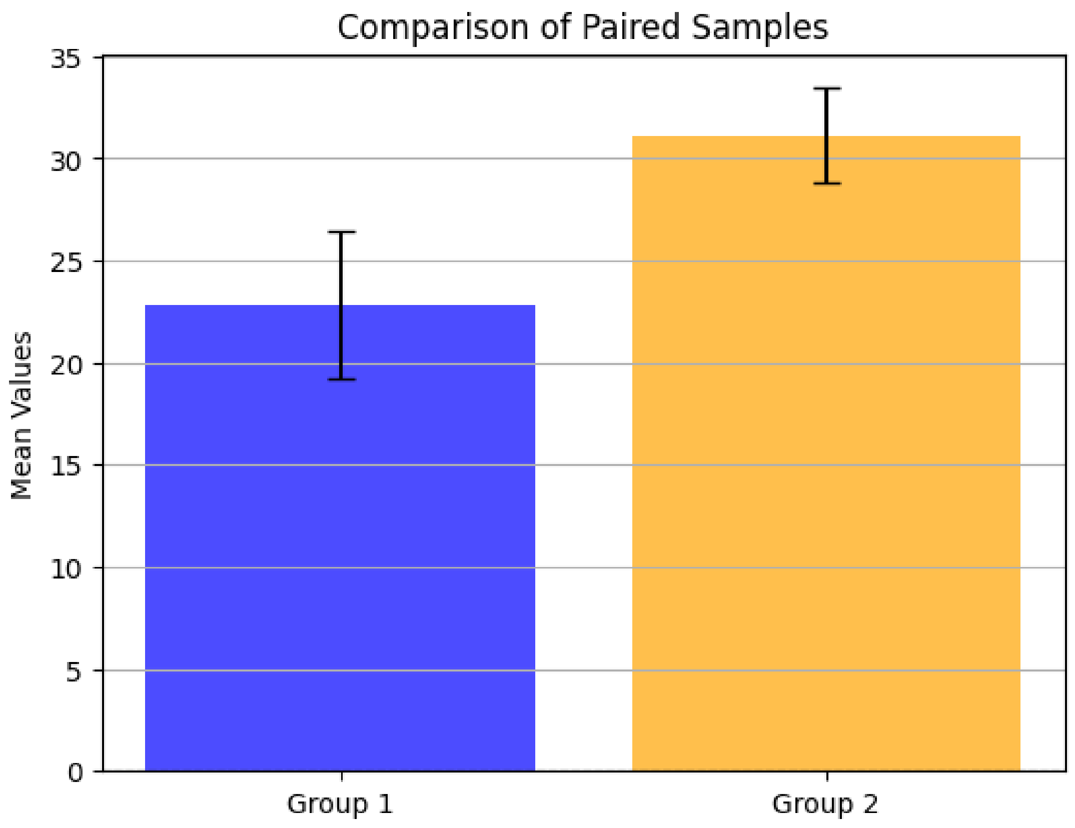
```python
import numpy as np
import scipy.stats as stats

# Function to perform an independent two-sample T-test
def independent_t_test(group1, group2, alpha=0.05):
    # Calculate means and standard deviations
    mean1 = np.mean(group1)
    mean2 = np.mean(group2)
    std1 = np.std(group1, ddof=1)  # Sample standard deviation
    std2 = np.std(group2, ddof=1)  # Sample standard deviation
    n1 = len(group1)
    n2 = len(group2)

    # Calculate the T-statistic and p-value
    t_statistic, p_value = stats.ttest_ind(group1, group2)

    # Print results
    print(f"Group 1 Mean: {mean1:.2f}, Standard Deviation: {std1:.2f}, 
Sample Size: {n1}")
    print(f"Group 2 Mean: {mean2:.2f}, Standard Deviation: {std2:.2f}, 
Sample Size: {n2}")
    print(f"T-Statistic: {t_statistic:.2f}")
    print(f"P-Value: {p_value:.4f}")

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

# Example usage
if __name__ == "__main__":
    # Sample data for two independent groups
    group1 = [23, 21, 18, 25, 30, 22, 19, 24, 20, 26]
    group2 = [30, 32, 29, 35, 31, 28, 33, 34, 30, 29]

    # Perform the independent T-test
    independent_t_test(group1, group2)

Group 1 Mean: 22.80, Standard Deviation: 3.61, Sample Size: 10
Group 2 Mean: 31.10, Standard Deviation: 2.33, Sample Size: 10
T-Statistic: -6.10
P-Value: 0.0000
Reject the null hypothesis: There is a significant difference between 
the two group means.
```

Q7.Perform a paired sample T-test using Python and visualize the comparison results?

Ans:-

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

# Function to perform a paired sample T-test
def paired_t_test(group1, group2, alpha=0.05):
    # Calculate the T-statistic and p-value
    t_statistic, p_value = stats.ttest_rel(group1, group2)
    # Print results
    print(f"T-Statistic: {t_statistic:.2f}")
    print(f"P-Value: {p_value:.4f}")

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

    return t_statistic, p_value

# Function to visualize the comparison results
def visualize_comparison(group1, group2):
    # Calculate means and standard deviations
    means = [np.mean(group1), np.mean(group2)]
    stds = [np.std(group1, ddof=1), np.std(group2, ddof=1)]
    labels = ['Group 1', 'Group 2']

    # Create a bar plot with error bars
    x = np.arange(len(labels))
    plt.bar(x, means, yerr=stds, capsize=5, color=['blue', 'orange'],
alpha=0.7)
    plt.xticks(x, labels)
    plt.ylabel('Mean Values')
    plt.title('Comparison of Paired Samples')
    plt.axhline(0, color='black', linewidth=0.8, linestyle='--')
    plt.grid(axis='y')
    plt.show()

# Example usage
if __name__ == "__main__":
    # Sample data for two paired groups
    group1 = [23, 21, 18, 25, 30, 22, 19, 24, 20, 26]
    group2 = [30, 32, 29, 35, 31, 28, 33, 34, 30, 29]

    # Perform the paired sample T-test
    t_statistic, p_value = paired_t_test(group1, group2)

    # Visualize the comparison results
    visualize_comparison(group1, group2)
```

```
T-Statistic: -6.56
P-Value: 0.0001
Reject the null hypothesis: There is a significant difference between
the two paired samples.
```



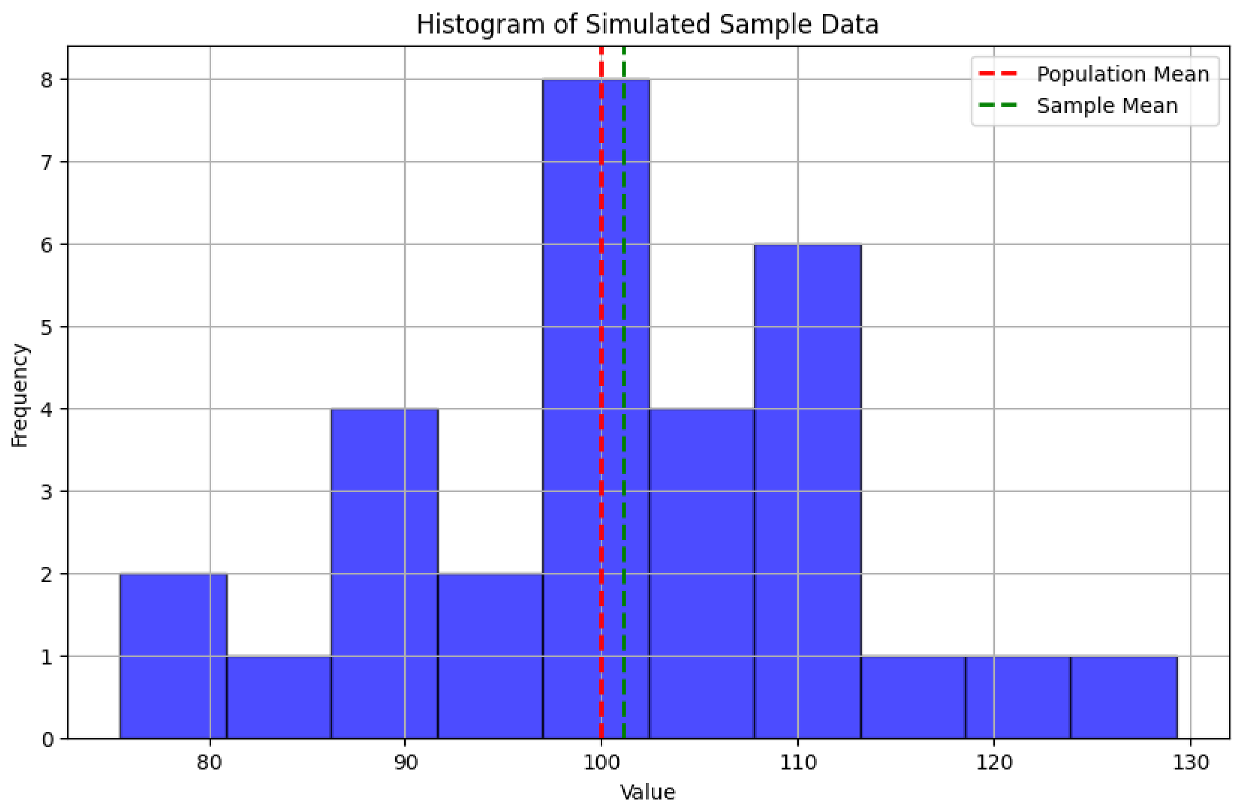
Q8.Simulate data and perform both Z-test and T-test, then compare the results using Python?

Ans:-

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

# Function to perform a one-sample Z-test
def z_test(sample, population_mean, population_std, alpha=0.05):
    sample_mean = np.mean(sample)
    n = len(sample)
    z_score = (sample_mean - population_mean) / (population_std /
np.sqrt(n))
    p_value = 2 * (1 - stats.norm.cdf(abs(z_score)))
    return z_score, p_value

# Function to perform a one-sample T-test
def t_test(sample, population_mean, alpha=0.05):
    t_statistic, p_value = stats.ttest_1samp(sample, population_mean)
    return t_statistic, p_value

# Function to simulate data
def simulate_data(sample_size, population_mean, population_std):
    return np.random.normal(loc=population_mean, scale=population_std,
```

```python
size=sample_size)

# Main function to run the tests and compare results
def main():
    # Parameters for simulation
    sample_size = 30
    population_mean = 100
    population_std = 15

    # Simulate data
    sample_data = simulate_data(sample_size, population_mean,
population_std)

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

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

    # Print results
    print(f"Z-Test: Z-Score = {z_score:.2f}, P-Value =
{z_p_value:.4f}")
    print(f"T-Test: T-Statistic = {t_statistic:.2f}, P-Value =
{t_p_value:.4f}")

    # Visualization
    plt.figure(figsize=(10, 6))
    plt.hist(sample_data, bins=10, alpha=0.7, color='blue',
edgecolor='black')
    plt.axvline(population_mean, color='red', linestyle='dashed',
linewidth=2, label='Population Mean')
    plt.axvline(np.mean(sample_data), color='green',
linestyle='dashed', linewidth=2, label='Sample Mean')
    plt.title('Histogram of Simulated Sample Data')
    plt.xlabel('Value')
    plt.ylabel('Frequency')
    plt.legend()
    plt.grid()
    plt.show()

# Run the main function
if __name__ == "__main__":
    main()
```

```
Z-Test: Z-Score = 0.42, P-Value = 0.6757
T-Test: T-Statistic = 0.54, P-Value = 0.5965
```



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

Ans:-

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

def confidence_interval(sample, confidence_level=0.95):
    """
    Calculate the confidence interval for the sample mean.

    Parameters:
    sample (array-like): The sample data.
    confidence_level (float): The confidence level (default is 0.95
for 95% confidence).

    Returns:
    tuple: The lower and upper bounds of the confidence interval.
    """
    # Calculate sample mean and standard error
    sample_mean = np.mean(sample)
    sample_std = np.std(sample, ddof=1)  # Sample standard deviation
    n = len(sample)  # Sample size
    standard_error = sample_std / np.sqrt(n)
    # Calculate the critical value for the t-distribution
    critical_value = stats.t.ppf((1 + confidence_level) / 2, df=n-1)

    # Calculate the margin of error
    margin_of_error = critical_value * standard_error

    # Calculate the confidence interval
    lower_bound = sample_mean - margin_of_error
    upper_bound = sample_mean + margin_of_error

    return lower_bound, upper_bound

# Example usage
if __name__ == "__main__":
    # Sample data
    sample_data = [23, 21, 18, 25, 30, 22, 19, 24, 20, 26]

    # Calculate the confidence interval
    ci = confidence_interval(sample_data, confidence_level=0.95)
    print(f"95% Confidence Interval for the Sample Mean: {ci[0]:.2f}
to {ci[1]:.2f}")
```

```
95% Confidence Interval for the Sample Mean: 20.21 to 25.39
```

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

Ans:-

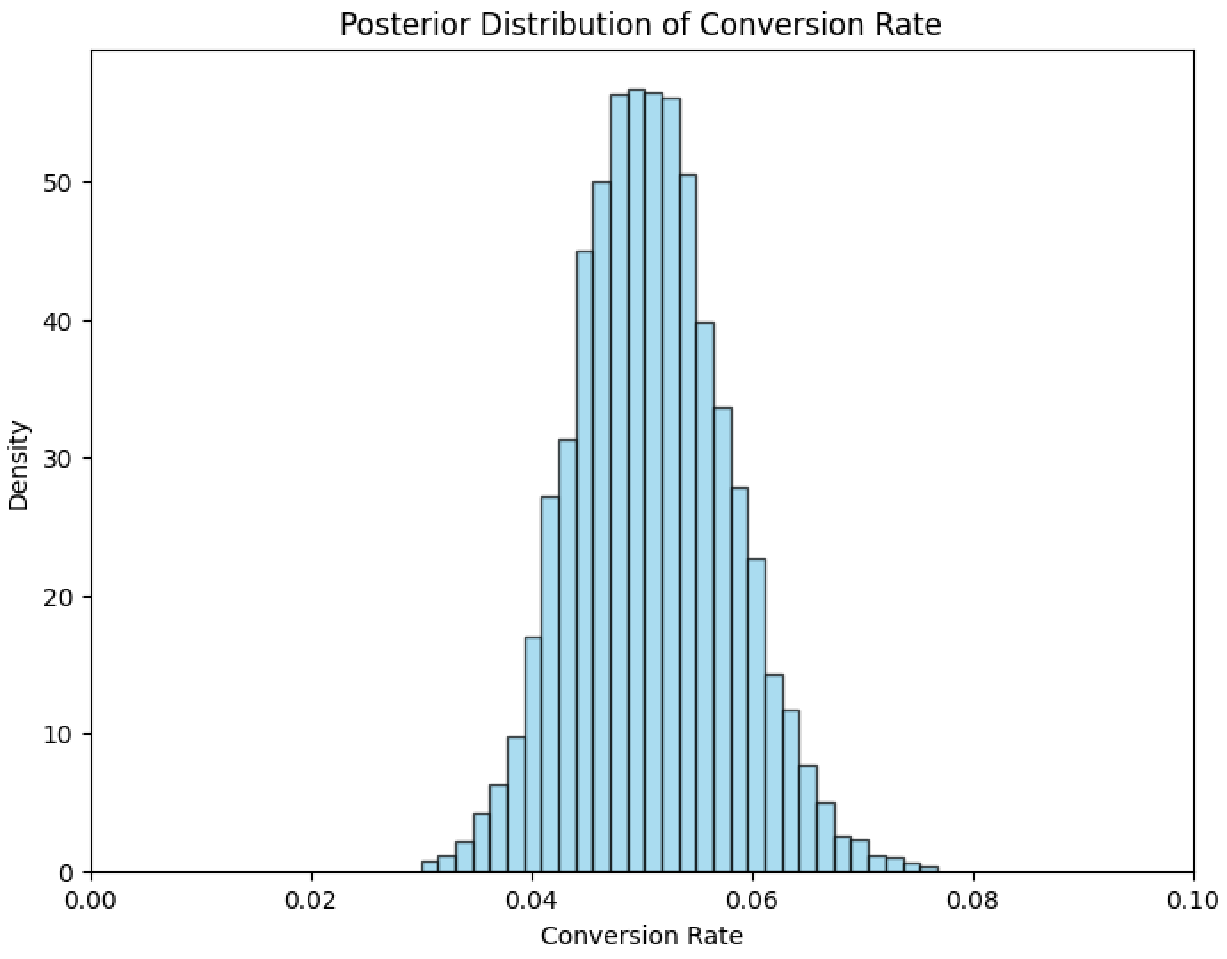
```
import numpy as np
import scipy.stats as stats

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

    Parameters:
    sample (array-like): The sample data.
    confidence_level (float): The confidence level (default is 0.95
for 95% confidence).

    Returns:
    float: The margin of error.
    """
    # Calculate sample standard deviation and sample size
    sample_std = np.std(sample, ddof=1)  # Sample standard deviation
    n = len(sample)  # Sample size

    # Calculate the standard error
    standard_error = sample_std / np.sqrt(n)

    # Calculate the critical value for the t-distribution
    critical_value = stats.t.ppf((1 + confidence_level) / 2, df=n-1)

    # Calculate the margin of error
    margin_of_error = critical_value * standard_error

    return margin_of_error

# Example usage
if __name__ == "__main__":
    # Sample data
    sample_data = [23, 21, 18, 25, 30, 22, 19, 24, 20, 26]

    # Calculate the margin of error
    me = margin_of_error(sample_data, confidence_level=0.95)
    print(f"Margin of Error at 95% Confidence Level: {me:.2f}")

Margin of Error at 95% Confidence Level: 2.59
```

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

Ans:-

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

# Observed data
num_visitors = 1000  # Total number of visitors to the website
num_conversions = 50  # Number of conversions (desired actions)

# Prior hyperparameters for the Beta distribution
prior_alpha = 1  # Shape parameter
prior_beta = 1   # Shape parameter

# Update the prior with the observed data to get the posterior
parameters
posterior_alpha = prior_alpha + num_conversions
posterior_beta = prior_beta + (num_visitors - num_conversions)

# Generate samples from the posterior Beta distribution
posterior_samples = np.random.beta(posterior_alpha, posterior_beta,
size=10000)

# Plot the posterior distribution
plt.figure(figsize=(8, 6))
plt.hist(posterior_samples, bins=30, density=True, color='skyblue',
```

```python
edgecolor='black', alpha=0.7)
plt.title('Posterior Distribution of Conversion Rate')
plt.xlabel('Conversion Rate')
plt.ylabel('Density')
plt.xlim(0, 0.1)  # Limiting x-axis to focus on conversion rates close
to zero
plt.show()

# Calculate summary statistics
mean_conversion_rate = posterior_alpha / (posterior_alpha +
posterior_beta)
mode_conversion_rate = (posterior_alpha - 1) / (posterior_alpha +
posterior_beta - 2)  # Mode of the Beta distribution

print("Mean conversion rate:", mean_conversion_rate)
print("Mode conversion rate:", mode_conversion_rate)
```



```
Mean conversion rate: 0.05089820359281437
Mode conversion rate: 0.05
```

Q12.Perform a Chi-square test for independence between two categorical variables in Python?

Ans:-

```python
import numpy as np
import pandas as pd
import scipy.stats as stats

# Sample data: A contingency table
data = {
    'Category A': [30, 10, 20],
    'Category B': [20, 25, 15]
}

# Create a DataFrame
df = pd.DataFrame(data, index=['Group 1', 'Group 2', 'Group 3'])

# Display the contingency table
print("Contingency Table:")
print(df)

# Perform the Chi-square test
chi2_statistic, p_value, dof, expected = stats.chi2_contingency(df)

# Print the results
print("\nChi-square Statistic:", chi2_statistic)
print("P-value:", p_value)
print("Degrees of Freedom:", dof)
print("Expected Frequencies:")
print(expected)

# Interpret the results
alpha = 0.05
if p_value < alpha:
    print("\nReject the null hypothesis: There is a significant 
association between the two categorical variables.")
else:
    print("\nFail to reject the null hypothesis: There is no 
significant association between the two categorical variables.")
```

```
Contingency Table:
         Category A  Category B
Group 1          30          20
Group 2          10          25
Group 3          20          15

Chi-square Statistic: 9.142857142857144
P-value: 0.010343173196618245
Degrees of Freedom: 2
Expected Frequencies:
[[25.  25. ]
```

```
 [17.5 17.5]
 [17.5 17.5]]

Reject the null hypothesis: There is a significant association between
the two categorical variables.
```

Q13.Write a Python program to calculate the expected frequencies for a Chi-square test based on observed data?

Ans:-

```
import numpy as np
import pandas as pd

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

    Parameters:
    observed (array-like): The observed data in a contingency table.

    Returns:
    np.ndarray: The expected frequencies.
    """
    # Convert observed data to a DataFrame if it's not already
    if not isinstance(observed, pd.DataFrame):
        observed = pd.DataFrame(observed)

    # Calculate row totals, column totals, and grand total
    row_totals = observed.sum(axis=1).values.reshape(-1, 1)  # Reshape
for broadcasting
    column_totals = observed.sum(axis=0).values  # Column totals
    grand_total = observed.values.sum()  # Grand total

    # Calculate expected frequencies
    expected = (row_totals * column_totals) / grand_total

    return expected

# Example usage
if __name__ == "__main__":
    # Sample observed data (contingency table)
    observed_data = np.array([[30, 10, 20],
                              [20, 25, 15]])

    # Create a DataFrame for better visualization
    observed_df = pd.DataFrame(observed_data, columns=['Category A',
'Category B', 'Category C'], index=['Group 1', 'Group 2'])
```

```
    print("Observed Frequencies:")
    print(observed_df)

    # Calculate expected frequencies
    expected_frequencies =
calculate_expected_frequencies(observed_data)

    # Display expected frequencies
    expected_df = pd.DataFrame(expected_frequencies,
columns=['Category A', 'Category B', 'Category C'], index=['Group 1',
'Group 2'])
    print("\nExpected Frequencies:")
    print(expected_df)

Observed Frequencies:
         Category A  Category B  Category C
Group 1          30          10          20
Group 2          20          25          15

Expected Frequencies:
         Category A  Category B  Category C
Group 1        25.0        17.5        17.5
Group 2        25.0        17.5        17.5
```

Q14.Perform a goodness-of-fit test using Python to compare the observed data to an expected distribution.

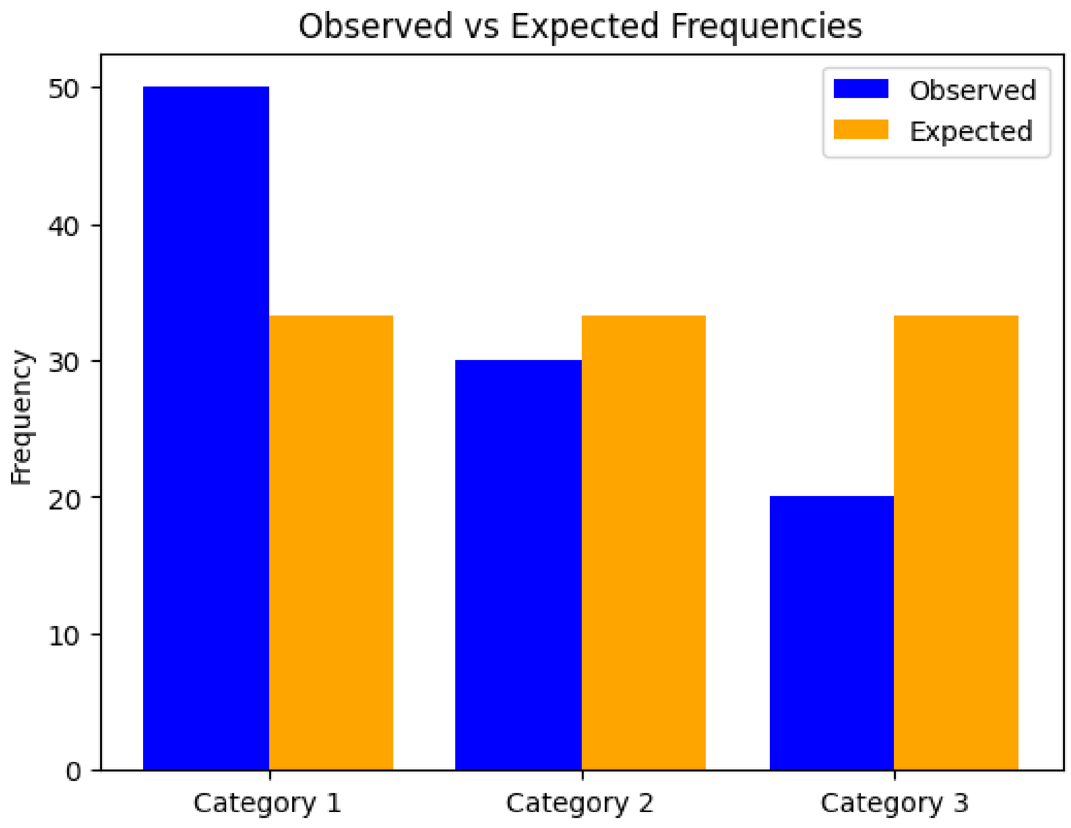
Ans:-

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

def chi_square_goodness_of_fit(observed, expected):
    """
    Perform a Chi-square goodness-of-fit test.

    Parameters:
    observed (array-like): The observed frequencies.
    expected (array-like): The expected frequencies.

    Returns:
    chi2_statistic (float): The Chi-square statistic.
    p_value (float): The p-value of the test.
    """
    # Calculate the Chi-square statistic and p-value
    chi2_statistic, p_value = stats.chisquare(observed, expected)
    return chi2_statistic, p_value
```

```python
# Example usage
if __name__ == "__main__":
    # Sample observed data (e.g., counts of outcomes)
    observed_data = np.array([50, 30, 20])  # Observed frequencies for
three categories

    # Expected frequencies for a uniform distribution
    total_observed = observed_data.sum()
    num_categories = len(observed_data)
    expected_data = np.full(num_categories, total_observed /
num_categories)  # Equal expected frequencies

    # Perform the Chi-square goodness-of-fit test
    chi2_statistic, p_value =
chi_square_goodness_of_fit(observed_data, expected_data)

    # Print the results
    print("Observed Frequencies:", observed_data)
    print("Expected Frequencies:", expected_data)
    print("Chi-square Statistic:", chi2_statistic)
    print("P-value:", p_value)

    # Interpret the results
    alpha = 0.05
    if p_value < alpha:
        print("\nReject the null hypothesis: The observed data does
not fit the expected distribution.")
    else:
        print("\nFail to reject the null hypothesis: The observed data
fits the expected distribution.")

    # Optional: Visualize the observed vs expected frequencies
    categories = ['Category 1', 'Category 2', 'Category 3']
    x = np.arange(len(categories))

    plt.bar(x - 0.2, observed_data, width=0.4, label='Observed',
color='blue')
    plt.bar(x + 0.2, expected_data, width=0.4, label='Expected',
color='orange')
    plt.xticks(x, categories)
    plt.ylabel('Frequency')
    plt.title('Observed vs Expected Frequencies')
    plt.legend()
    plt.show()
```

```
Observed Frequencies: [50 30 20]
Expected Frequencies: [33.33333333 33.33333333 33.33333333]
Chi-square Statistic: 14.0
P-value: 0.0009118819655545164
```

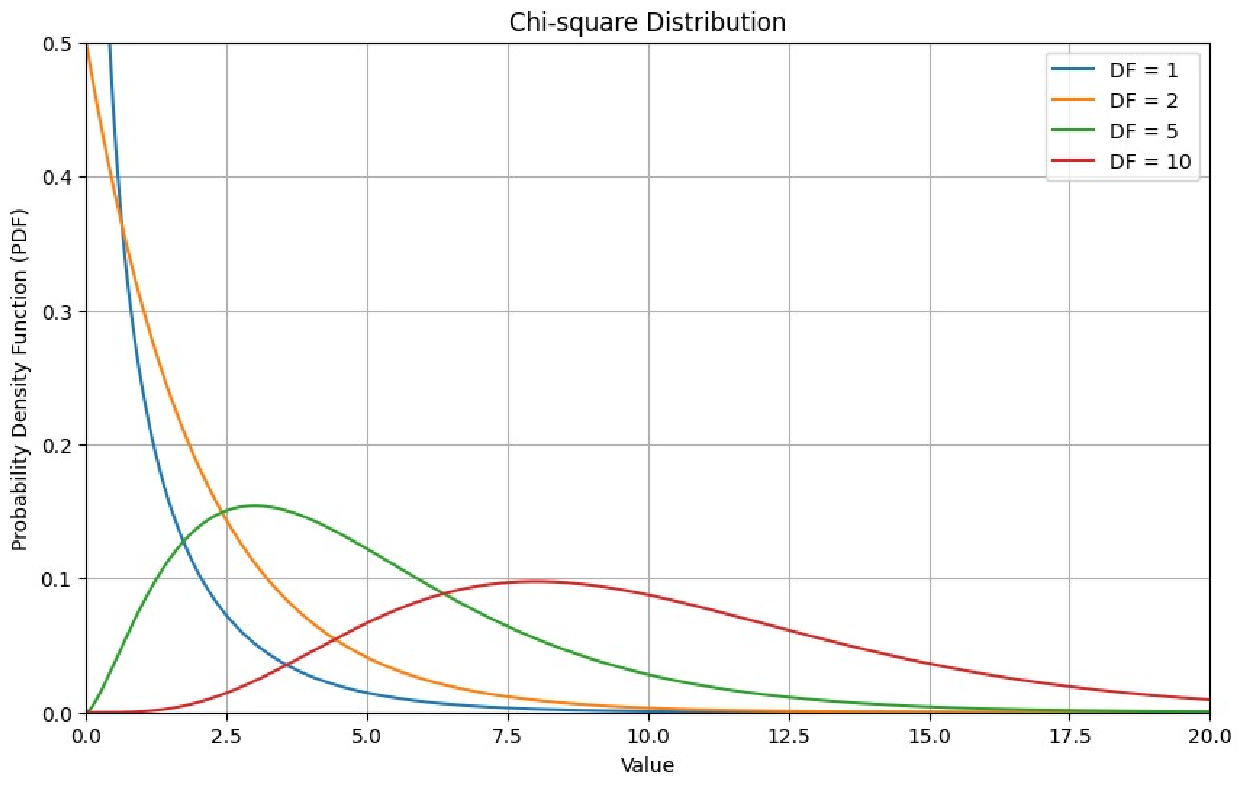
```
Reject the null hypothesis: The observed data does not fit the
expected distribution.
```



Q15.Create a Python script to simulate and visualize the Chi-square distribution and discuss its characteristics.

Ans:-

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

def plot_chi_square_distribution(degrees_of_freedom, x_range):
    """
    Plot the Chi-square distribution for given degrees of freedom.

    Parameters:
    degrees_of_freedom (list): List of degrees of freedom to plot.
    x_range (tuple): Range of x values for the plot.
    """
    x = np.linspace(x_range[0], x_range[1], 1000)

    plt.figure(figsize=(10, 6))
    for df in degrees_of_freedom:
        # Calculate the Chi-square PDF
        pdf = stats.chi2.pdf(x, df)
        plt.plot(x, pdf, label=f'DF = {df}')

    plt.title('Chi-square Distribution')
    plt.xlabel('Value')
    plt.ylabel('Probability Density Function (PDF)')
    plt.legend()
    plt.grid()
    plt.xlim(x_range)
    plt.ylim(0, 0.5)
    plt.show()

# Example usage
if __name__ == "__main__":
    degrees_of_freedom = [1, 2, 5, 10]  # Different degrees of freedom
to visualize
    x_range = (0, 20)  # Range of x values for the plot

    plot_chi_square_distribution(degrees_of_freedom, x_range)
```



Q16.Implement an F-test using Python to compare the variances of two random samples.

Ans:-

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

def f_test(sample1, sample2, alpha=0.05):
    """
    Perform an F-test to compare the variances of two samples.

    Parameters:
    sample1 (array-like): The first sample data.
    sample2 (array-like): The second sample data.
    alpha (float): The significance level (default is 0.05).

    Returns:
    f_statistic (float): The F-statistic.
    p_value (float): The p-value of the test.
    """
    # Calculate the variances of the samples
    var1 = np.var(sample1, ddof=1)  # Sample variance
    var2 = np.var(sample2, ddof=1)  # Sample variance

    # Calculate the F-statistic
    f_statistic = var1 / var2

    # Calculate the degrees of freedom
    df1 = len(sample1) - 1  # Degrees of freedom for sample1
    df2 = len(sample2) - 1  # Degrees of freedom for sample2

    # Calculate the p-value
    p_value = 1 - stats.f.cdf(f_statistic, df1, df2)

    return f_statistic, p_value

# Example usage
if __name__ == "__main__":
    # Sample data
    sample1 = np.array([20, 22, 19, 24, 21, 23, 20, 22, 19, 21])
    sample2 = np.array([30, 32, 29, 35, 31, 28, 33, 34, 30, 29])

    # Perform the F-test
    f_statistic, p_value = f_test(sample1, sample2)

    # Print the results
    print("F-Statistic:", f_statistic)
    print("P-Value:", p_value)

    # Interpret the results
    alpha = 0.05
    if p_value < alpha:
        print("\nReject the null hypothesis: The variances of the two 
samples are significantly different.")
    else:
        print("\nFail to reject the null hypothesis: The variances of the two samples are not significantly different.")
```

```
F-Statistic: 0.5092024539877301
P-Value: 0.8354277292129488

Fail to reject the null hypothesis: The variances of the two samples are not significantly different.
```

Q17.Write a Python program to perform an ANOVA test to compare means between multiple groups and interpret the results.

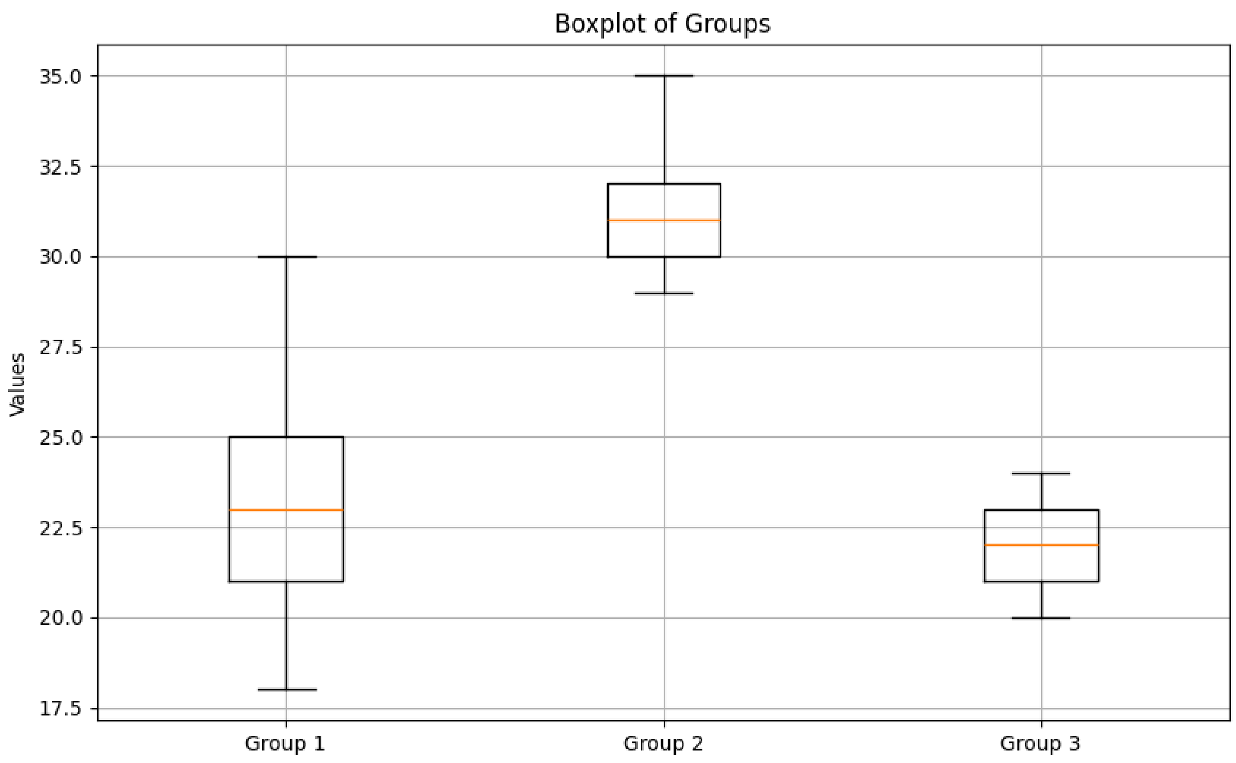
Ans:-

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

def perform_anova(*groups):
    """
    Perform a one-way ANOVA test to compare means between multiple groups.

    Parameters:
    *groups: Variable number of group samples (arrays).

    Returns:
    f_statistic (float): The F-statistic.
    p_value (float): The p-value of the test.
    """
    # Perform the ANOVA test
    f_statistic, p_value = stats.f_oneway(*groups)
    return f_statistic, p_value

def plot_groups(groups):
    """
    Plot the boxplot of the groups for visual comparison.

    Parameters:
    groups: List of group samples (arrays).
    """
    plt.figure(figsize=(10, 6))
    plt.boxplot(groups, labels=[f'Group {i+1}' for i in range(len(groups))])
    plt.title('Boxplot of Groups')
    plt.ylabel('Values')
    plt.grid()
    plt.show()
```

```python
# Example usage
if __name__ == "__main__":
    # Sample data for three groups
    group1 = np.array([23, 21, 18, 25, 30])
    group2 = np.array([30, 32, 29, 35, 31])
    group3 = np.array([22, 24, 20, 23, 21])

    # Perform the ANOVA test
    f_statistic, p_value = perform_anova(group1, group2, group3)

    # Print the results
    print("F-Statistic:", f_statistic)
    print("P-Value:", p_value)

    # Interpret the results
    alpha = 0.05
    if p_value < alpha:
        print("\nReject the null hypothesis: At least one group mean
is significantly different.")
    else:
        print("\nFail to reject the null hypothesis: There is no
significant difference between group means.")

    # Plot the groups for visual comparison
    plot_groups([group1, group2, group3])
```

```
F-Statistic: 13.729537366548039
P-Value: 0.0007910539293303842

Reject the null hypothesis: At least one group mean is significantly
different.
```

```
<ipython-input-17-343a11efb66b>:29: MatplotlibDeprecationWarning: The
'labels' parameter of boxplot() has been renamed 'tick_labels' since
Matplotlib 3.9; support for the old name will be dropped in 3.11.
  plt.boxplot(groups, labels=[f'Group {i+1}' for i in
range(len(groups))])
```



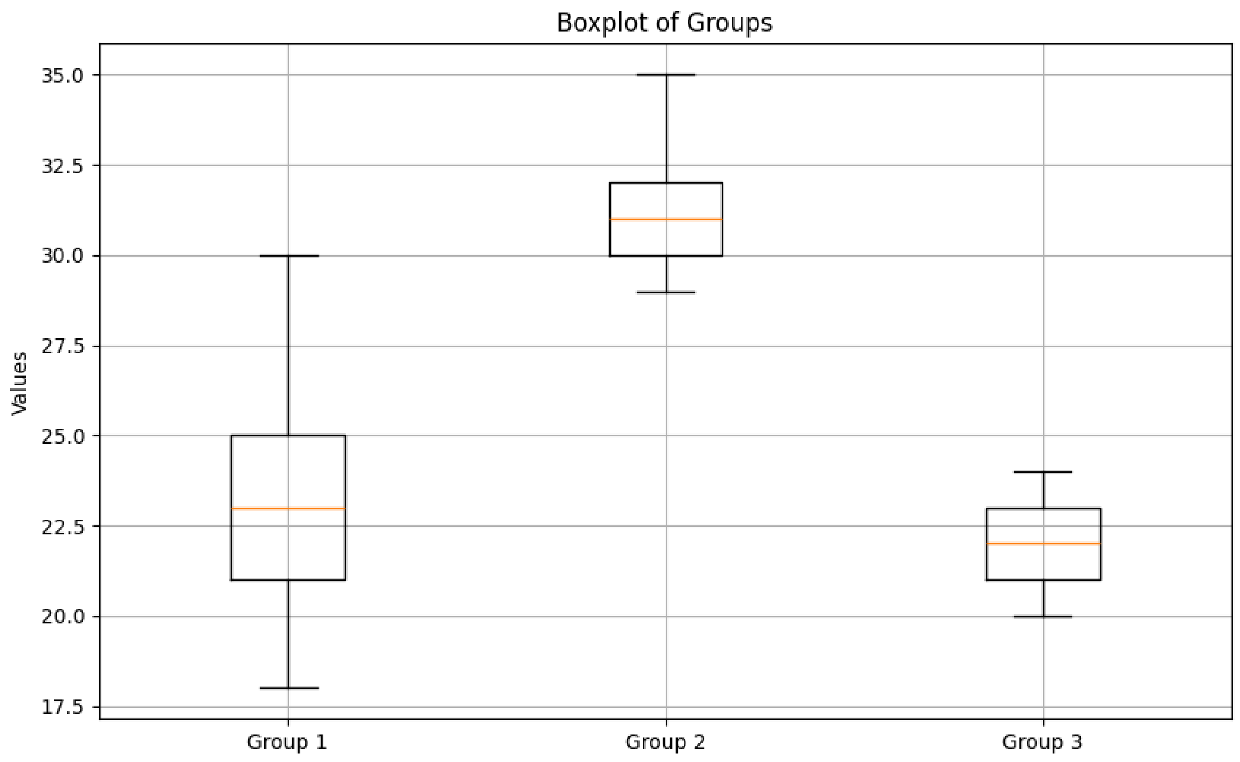
Q18.Perform a one-way ANOVA test using Python to compare the means of different groups and plot the results.

Ans:-

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

def perform_anova(*groups):
    """
    Perform a one-way ANOVA test to compare means between multiple
groups.

    Parameters:
    *groups: Variable number of group samples (arrays).

    Returns:
    f_statistic (float): The F-statistic.
    p_value (float): The p-value of the test.
    """
    # Perform the ANOVA test
    f_statistic, p_value = stats.f_oneway(*groups)
    return f_statistic, p_value

def plot_groups(groups, group_labels):
    """
    Plot the boxplot of the groups for visual comparison.

    Parameters:
    groups: List of group samples (arrays).
    group_labels: List of labels for each group.
    """
    plt.figure(figsize=(10, 6))
    plt.boxplot(groups, labels=group_labels)
    plt.title('Boxplot of Groups')
    plt.ylabel('Values')
    plt.grid()
    plt.show()

# Example usage
if __name__ == "__main__":
    # Sample data for three groups
    group1 = np.array([23, 21, 18, 25, 30])
    group2 = np.array([30, 32, 29, 35, 31])
    group3 = np.array([22, 24, 20, 23, 21])

    # Perform the ANOVA test
    f_statistic, p_value = perform_anova(group1, group2, group3)

    # Print the results
    print("F-Statistic:", f_statistic)
    print("P-Value:", p_value)

    # Interpret the results
    alpha = 0.05
    if p_value < alpha:
        print("\nReject the null hypothesis: At least one group mean
is significantly different.")
    else:
        print("\nFail to reject the null hypothesis: There is no
significant difference between group means.")

    # Plot the groups for visual comparison
    plot_groups([group1, group2, group3], ['Group 1', 'Group 2',
'Group 3'])
```

```
F-Statistic: 13.729537366548039
P-Value: 0.0007910539293303842

Reject the null hypothesis: At least one group mean is significantly
different.

<ipython-input-18-584463ee2571>:30: MatplotlibDeprecationWarning: The
'labels' parameter of boxplot() has been renamed 'tick_labels' since
```

```
Matplotlib 3.9; support for the old name will be dropped in 3.11.
  plt.boxplot(groups, labels=group_labels)
```

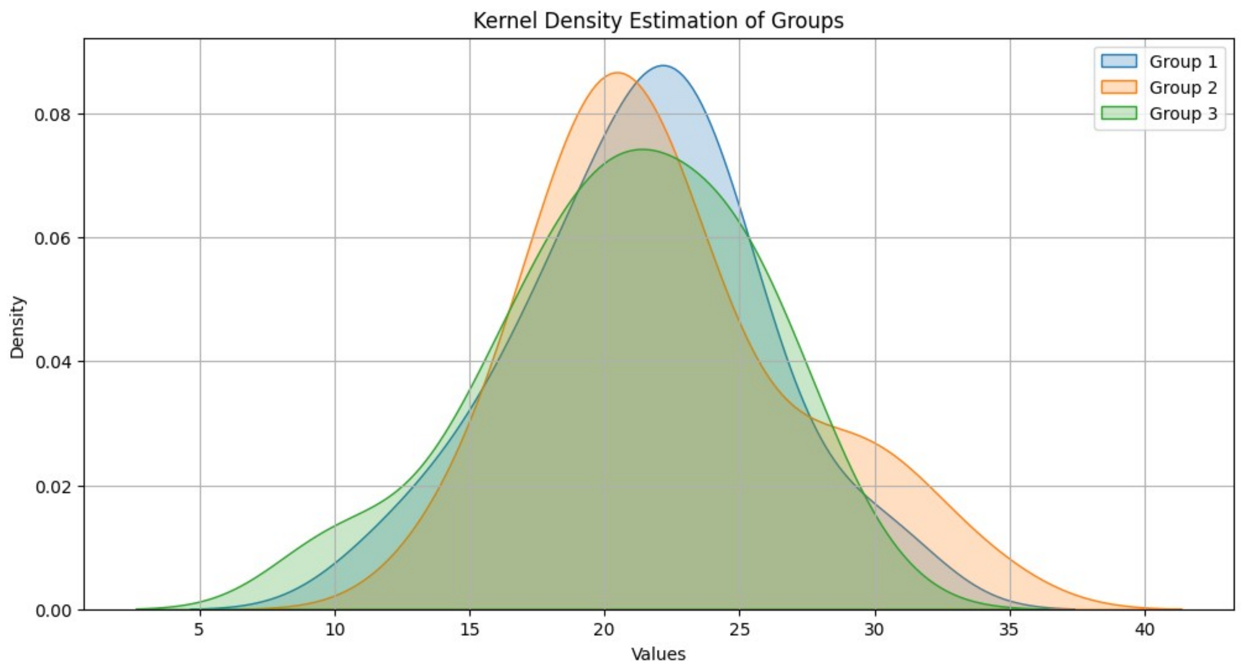


Q19.Write a Python function to check the assumptions (normality, independence, and equal variance) for ANOVA.

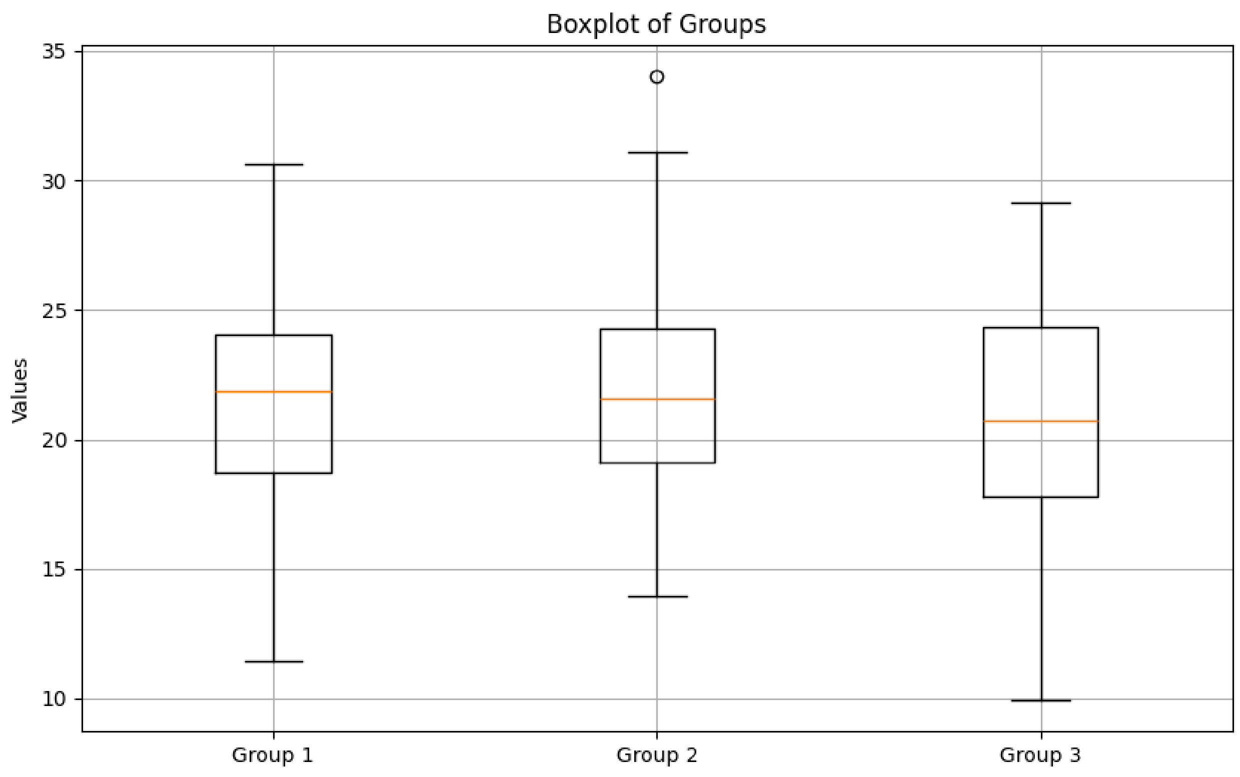
Ans:-

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

def check_anova_assumptions(*groups):
    """
    Check the assumptions for ANOVA: normality, independence, and
equal variance.

    Parameters:
    *groups: Variable number of group samples (arrays).
    """
    # Check Normality using Shapiro-Wilk test
    print("Normality Test Results:")
    for i, group in enumerate(groups):
        stat, p_value = stats.shapiro(group)
        print(f"Group {i+1}: W-statistic = {stat:.4f}, p-value =
{p_value:.4f}")
        if p_value < 0.05:
            print(f"Group {i+1} does not follow a normal distribution
(reject H0).\n")
        else:
            print(f"Group {i+1} follows a normal distribution (fail to
reject H0).\n")

    # Check Equal Variance using Levene's test
    stat, p_value = stats.levene(*groups)
    print("Equal Variance Test (Levene's Test):")
    print(f"Levene's W-statistic = {stat:.4f}, p-value =
{p_value:.4f}")
    if p_value < 0.05:
        print("The variances are significantly different (reject H0).\
n")
    else:
        print("The variances are equal (fail to reject H0).\n")

    # Visualize the data distribution
    plt.figure(figsize=(12, 6))
    for i, group in enumerate(groups):
        sns.kdeplot(group, label=f'Group {i+1}', fill=True)
    plt.title('Kernel Density Estimation of Groups')
    plt.xlabel('Values')
    plt.ylabel('Density')
    plt.legend()
    plt.grid()
    plt.show()

    # Boxplot for visual inspection of equal variance
    plt.figure(figsize=(10, 6))
    plt.boxplot(groups, labels=[f'Group {i+1}' for i in
range(len(groups))])
    plt.title('Boxplot of Groups')
    plt.ylabel('Values')
    plt.grid()
    plt.show()

# Example usage
if __name__ == "__main__":
    # Sample data for three groups
    group1 = np.random.normal(loc=20, scale=5, size=30)
    group2 = np.random.normal(loc=22, scale=5, size=30)
    group3 = np.random.normal(loc=21, scale=5, size=30)

    # Check ANOVA assumptions
    check_anova_assumptions(group1, group2, group3)
```

```
Normality Test Results:
Group 1: W-statistic = 0.9838, p-value = 0.9157
Group 1 follows a normal distribution (fail to reject H0).

Group 2: W-statistic = 0.9456, p-value = 0.1287
Group 2 follows a normal distribution (fail to reject H0).

Group 3: W-statistic = 0.9666, p-value = 0.4510
Group 3 follows a normal distribution (fail to reject H0).

Equal Variance Test (Levene's Test):
Levene's W-statistic = 0.1427, p-value = 0.8672
The variances are equal (fail to reject H0).
```



```
<ipython-input-19-4a75d65e6e31>:46: MatplotlibDeprecationWarning: The
'labels' parameter of boxplot() has been renamed 'tick_labels' since
Matplotlib 3.9; support for the old name will be dropped in 3.11.
  plt.boxplot(groups, labels=[f'Group {i+1}' for i in
range(len(groups))])
```



Q20.Perform a two-way ANOVA test using Python to study the interaction between two factors and visualize the results.

Ans:-Step 1: Import Required Libraries

python

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
import seaborn as sns
import matplotlib.pyplot as plt
```

Step 2: Create or Load Your Data

For demonstration, let's create a sample dataset. In practice, you would load your data from a CSV or other source.

```
# Create a sample dataset
np.random.seed(42)
data = {
    'Factor1': np.repeat(['A', 'B', 'C'], 20),
    'Factor2': np.tile(np.repeat(['X', 'Y'], 10), 3),
    'Response': np.random.normal(loc=0, scale=1, size=60)
}
df = pd.DataFrame(data)
```

Step 3: Perform Two-Way ANOVA

```python
# Fit the model
model = smf.ols('Response ~ C(Factor1) * C(Factor2)', data=df).fit()

# Perform ANOVA
anova_table = sm.stats.anova_lm(model, typ=2)
print(anova_table)
```

```
                          sum_sq    df         F    PR(>F)
C(Factor1)              0.580882   2.0  0.398223  0.673468
C(Factor2)              1.275222   1.0  1.748451  0.191645
C(Factor1):C(Factor2)   7.458570   2.0  5.113208  0.009255
Residual               39.384553  54.0       NaN       NaN
```

Step 4: Check for Significant Interactions

The ANOVA table will show you the p-values for the main effects and their interaction. If the p-value for the interaction term is less than 0.05, it indicates a significant interaction between the two factors.

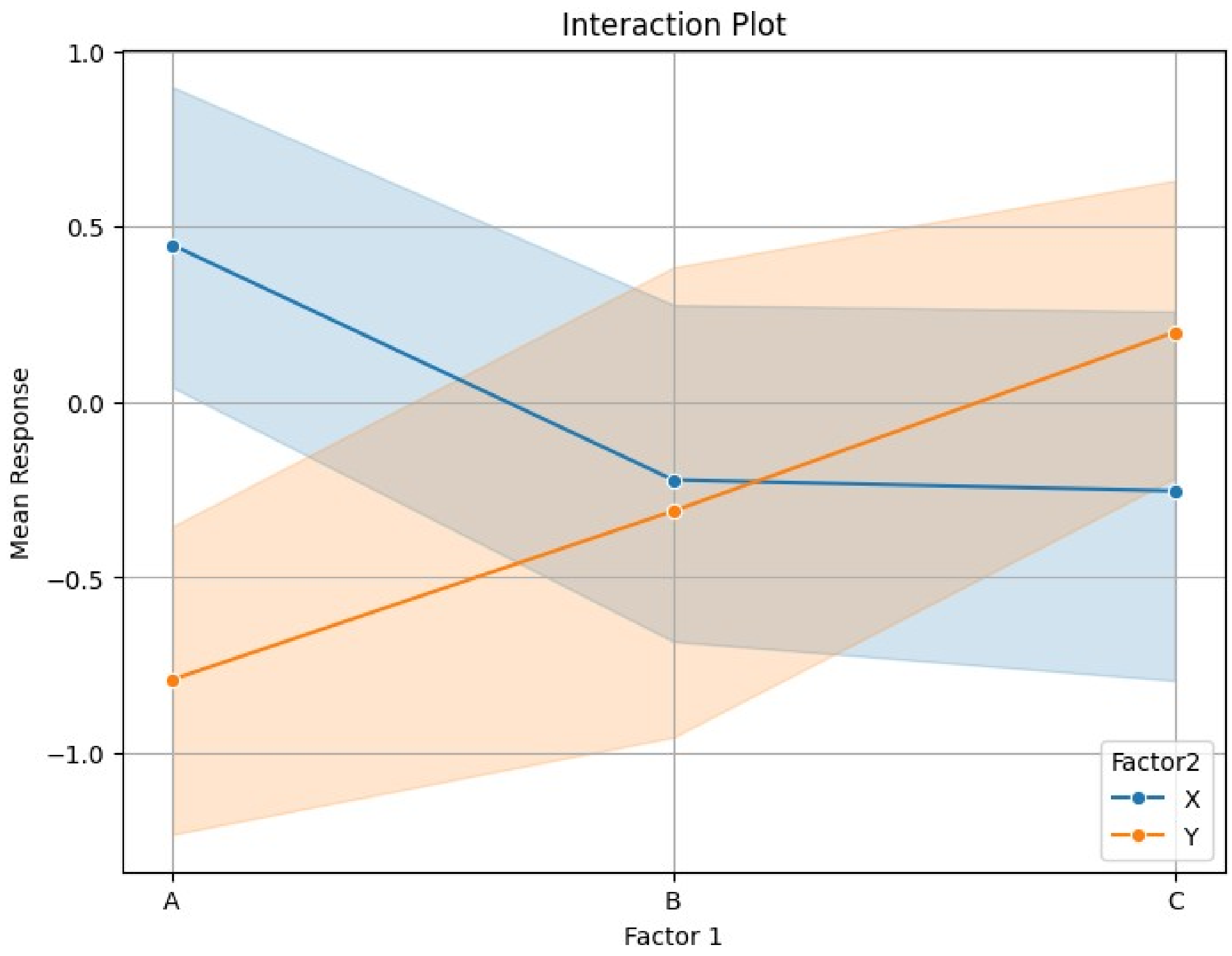
Step 5: Visualize the Results

You can visualize the interaction using an interaction plot.

```python
import pandas as pd
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
import seaborn as sns
import matplotlib.pyplot as plt

# Create a sample dataset
np.random.seed(42)
data = {
    'Factor1': np.repeat(['A', 'B', 'C'], 20),
    'Factor2': np.tile(np.repeat(['X', 'Y'], 10), 3),
    'Response': np.random.normal(loc=0, scale=1, size=60)
}
df = pd.DataFrame(data)

# Fit the model
model = smf.ols('Response ~ C(Factor1) * C(Factor2)', data=df).fit()

# Perform ANOVA
anova_table = sm.stats.anova_lm(model, typ=2)
print(anova_table)
```

```python
# Create interaction plot using lineplot
plt.figure(figsize=(8, 6))
sns.lineplot(data=df, x='Factor1', y='Response', hue='Factor2',
estimator='mean', marker='o')
plt.title('Interaction Plot')
plt.xlabel('Factor 1')
plt.ylabel('Mean Response')
plt.grid()
plt.show()
```

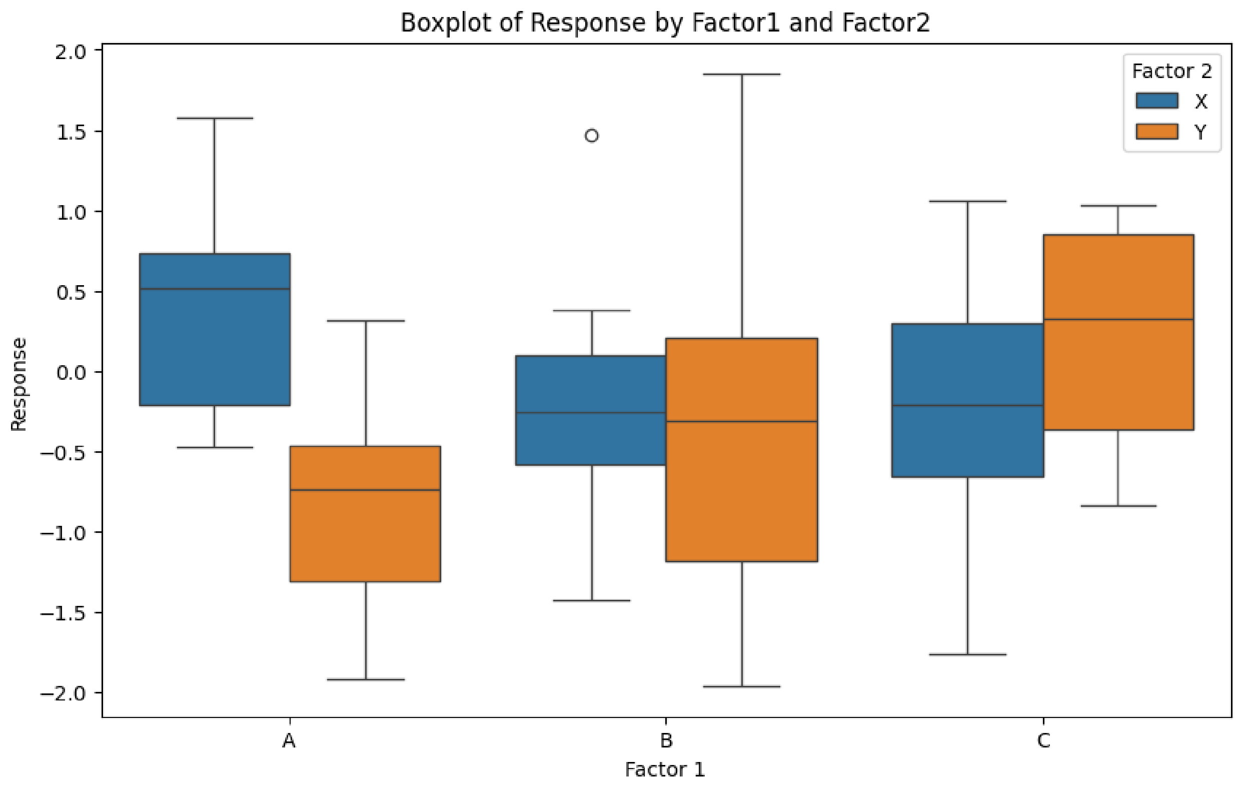
```
                          sum_sq    df         F    PR(>F)
C(Factor1)              0.580882   2.0  0.398223  0.673468
C(Factor2)              1.275222   1.0  1.748451  0.191645
C(Factor1):C(Factor2)   7.458570   2.0  5.113208  0.009255
Residual               39.384553  54.0       NaN       NaN
```



Step 6: Boxplot for Group Comparison

Additionally, you can create boxplots to visualize the distribution of the response variable across the different groups.

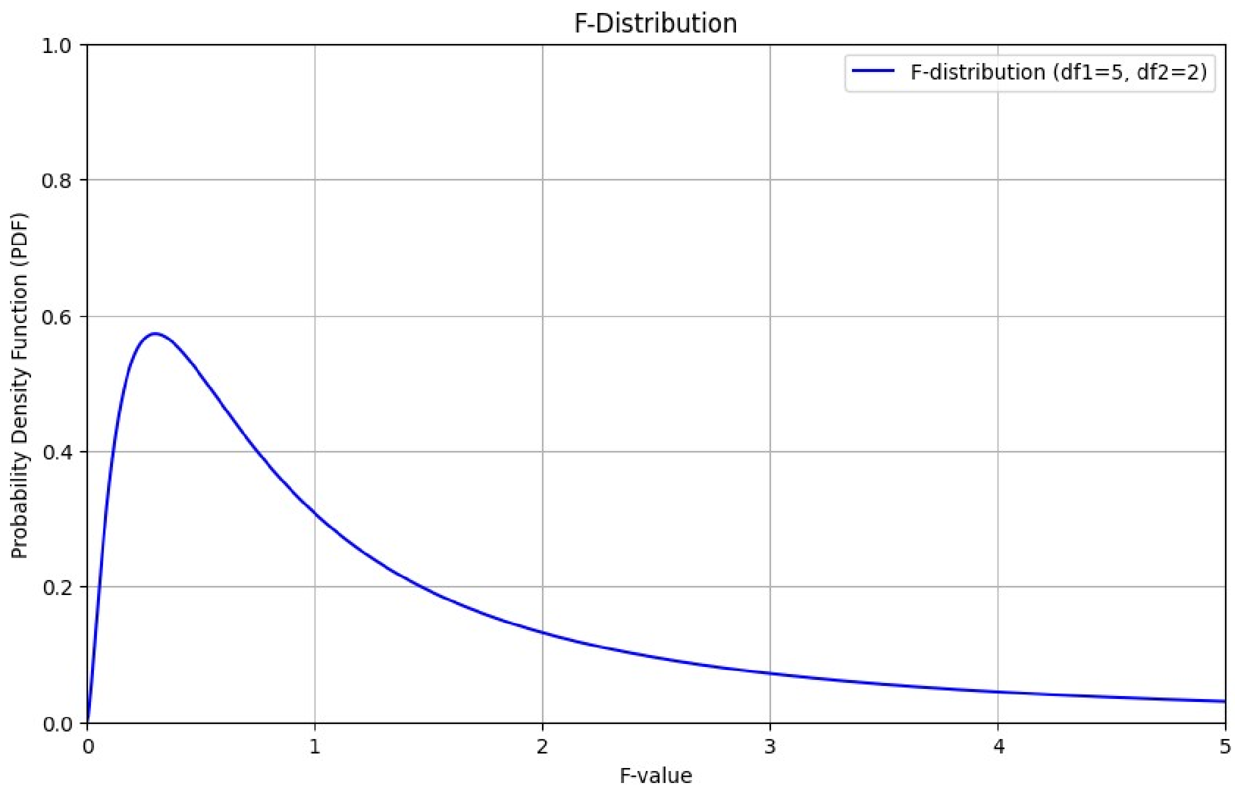
```
# Boxplot
plt.figure(figsize=(10, 6))
sns.boxplot(x='Factor1', y='Response', hue='Factor2', data=df)
plt.title('Boxplot of Response by Factor1 and Factor2')
plt.xlabel('Factor 1')
plt.ylabel('Response')
plt.legend(title='Factor 2')
plt.show()
```



Q21.Write a Python program to visualize the F-distribution and discuss its use in hypothesis testing.

Ans:-

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

def plot_f_distribution(degrees_of_freedom1, degrees_of_freedom2):
    """
    Plot the F-distribution for given degrees of freedom.

    Parameters:
    degrees_of_freedom1 (int): Degrees of freedom for the numerator.
    degrees_of_freedom2 (int): Degrees of freedom for the denominator.
    """
    x = np.linspace(0, 5, 1000)  # Range of x values
    pdf = stats.f.pdf(x, degrees_of_freedom1, degrees_of_freedom2)  #
F-distribution PDF

    plt.figure(figsize=(10, 6))
    plt.plot(x, pdf, label=f'F-distribution
(df1={degrees_of_freedom1}, df2={degrees_of_freedom2})', color='blue')
    plt.title('F-Distribution')
    plt.xlabel('F-value')
    plt.ylabel('Probability Density Function (PDF)')
    plt.grid()
    plt.legend()
    plt.xlim(0, 5)
    plt.ylim(0, 1)
    plt.show()

# Example usage
if __name__ == "__main__":
    # Define degrees of freedom
    df1 = 5  # Degrees of freedom for the numerator
    df2 = 2  # Degrees of freedom for the denominator

    # Plot the F-distribution
    plot_f_distribution(df1, df2)
```

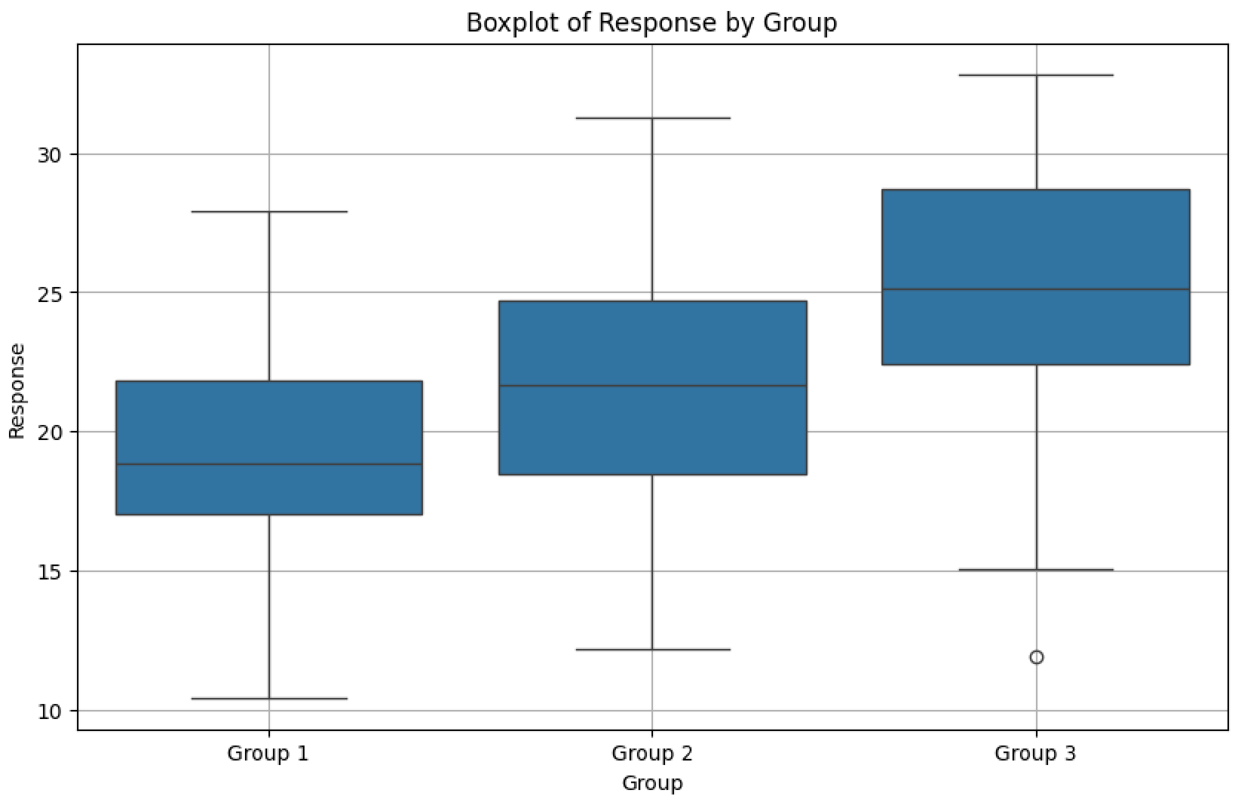


Q22.Perform a one-way ANOVA test in Python and visualize the results with boxplots to compare group means.

Ans:-

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

# Create a sample dataset
np.random.seed(42)
group1 = np.random.normal(loc=20, scale=5, size=30)  # Group 1
group2 = np.random.normal(loc=22, scale=5, size=30)  # Group 2
group3 = np.random.normal(loc=25, scale=5, size=30)  # Group 3

# Combine the groups into a DataFrame
data = {
    'Group': ['Group 1'] * 30 + ['Group 2'] * 30 + ['Group 3'] * 30,
    'Response': np.concatenate([group1, group2, group3])
}
df = pd.DataFrame(data)

# Perform the one-way ANOVA test
f_statistic, p_value = stats.f_oneway(df[df['Group'] == 'Group 1']
['Response'],
                                       df[df['Group'] == 'Group 2']
['Response'],
                                       df[df['Group'] == 'Group 3']
['Response'])

# Print the results
print("F-Statistic:", f_statistic)
print("P-Value:", p_value)

# Interpret the results
alpha = 0.05
if p_value < alpha:
    print("\nReject the null hypothesis: At least one group mean is
significantly different.")
else:
    print("\nFail to reject the null hypothesis: There is no
significant difference between group means.")

# Visualize the results with boxplots
plt.figure(figsize=(10, 6))
sns.boxplot(x='Group', y='Response', data=df)
plt.title('Boxplot of Response by Group')
plt.xlabel('Group')
plt.ylabel('Response')
plt.grid()
plt.show()
```

```
F-Statistic: 12.397871901825155
P-Value: 1.8305780660638664e-05

Reject the null hypothesis: At least one group mean is significantly
different.
```



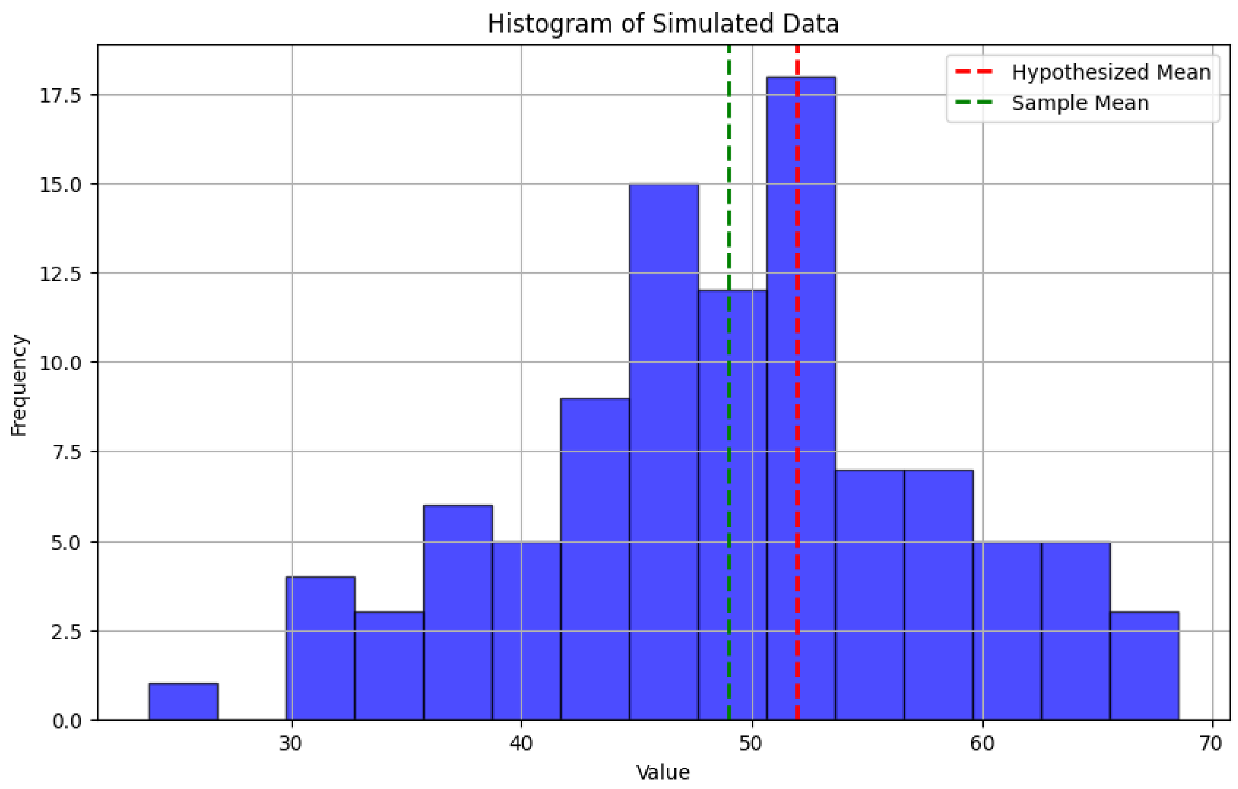
Q23.Simulate random data from a normal distribution, then perform hypothesis testing to evaluate the means.

Ans:-Python Code for Simulating Data and Performing Hypothesis Testing

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

# Set parameters for the normal distribution
np.random.seed(42)  # For reproducibility
mean = 50            # True mean of the population
std_dev = 10         # Standard deviation of the population
sample_size = 100    # Number of samples to simulate
```

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

# Perform a one-sample t-test
population_mean = 52  # Hypothesized population mean
t_statistic, p_value = stats.ttest_1samp(data, population_mean)

# Print the results
print("Sample Mean:", np.mean(data))
print("T-Statistic:", t_statistic)
print("P-Value:", p_value)

# Interpret the results
alpha = 0.05
if p_value < alpha:
    print("\nReject the null hypothesis: The sample mean is
significantly different from the population mean.")
else:
    print("\nFail to reject the null hypothesis: The sample mean is
not significantly different from the population mean.")

# Visualize the data
plt.figure(figsize=(10, 6))
plt.hist(data, bins=15, alpha=0.7, color='blue', edgecolor='black')
plt.axvline(population_mean, color='red', linestyle='dashed',
linewidth=2, label='Hypothesized Mean')
plt.axvline(np.mean(data), color='green', linestyle='dashed',
linewidth=2, label='Sample Mean')
plt.title('Histogram of Simulated Data')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
plt.grid()
plt.show()
```

```
Sample Mean: 48.96153482605907
T-Statistic: -3.3457066775666733
P-Value: 0.0011606436334217142

Reject the null hypothesis: The sample mean is significantly different
from the population mean.
```



Q24.Perform a hypothesis test for population variance using a Chi-square distribution and interpret the results.

Ans:-

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

# Sample data
np.random.seed(42)  # For reproducibility
data = np.random.normal(loc=50, scale=10, size=30)  # Simulated data

# Hypothesized population variance
sigma_squared_0 = 100  # Hypothesized variance (10^2)

# Calculate sample variance and sample size
n = len(data)
sample_variance = np.var(data, ddof=1)  # Sample variance with
Bessel's correction

# Calculate the Chi-square statistic
chi_square_statistic = (n - 1) * sample_variance / sigma_squared_0

# Degrees of freedom
df = n - 1
```

```python
# Significance level
alpha = 0.05

# Critical values for two-tailed test
critical_value_lower = stats.chi2.ppf(alpha / 2, df)
critical_value_upper = stats.chi2.ppf(1 - alpha / 2, df)

# Print results
print("Sample Variance:", sample_variance)
print("Chi-square Statistic:", chi_square_statistic)
print("Critical Value Lower:", critical_value_lower)
print("Critical Value Upper:", critical_value_upper)

# Interpret the results
if chi_square_statistic < critical_value_lower or chi_square_statistic
> critical_value_upper:
    print("\nReject the null hypothesis: The population variance is
significantly different from the hypothesized variance.")
else:
    print("\nFail to reject the null hypothesis: The population
variance is not significantly different from the hypothesized
variance.")
```

```
Sample Variance: 81.00115693656518
Chi-square Statistic: 23.490335511603902
Critical Value Lower: 16.04707169536489
Critical Value Upper: 45.72228580417452

Fail to reject the null hypothesis: The population variance is not
significantly different from the hypothesized variance.
```

Q25.Write a Python script to perform a Z-test for comparing proportions between two datasets or groups.

Ans:-

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

def z_test_proportions(success_a, n_a, success_b, n_b):
    """
    Perform a Z-test for comparing proportions between two groups.

    Parameters:
    success_a (int): Number of successes in group A.
    n_a (int): Total number of observations in group A.
    success_b (int): Number of successes in group B.
    n_b (int): Total number of observations in group B.

    Returns:
```

```
    z_statistic (float): The Z-statistic.
    p_value (float): The p-value of the test.
    """
    # Calculate proportions
    p_a = success_a / n_a
    p_b = success_b / n_b

    # Calculate the pooled proportion
    p_pool = (success_a + success_b) / (n_a + n_b)

    # Calculate the standard error
    standard_error = np.sqrt(p_pool * (1 - p_pool) * (1/n_a + 1/n_b))

    # Calculate the Z-statistic
    z_statistic = (p_a - p_b) / standard_error

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

    return z_statistic, p_value

# Example usage
if __name__ == "__main__":
    # Data for group A
    success_a = 30  # Number of successes in group A
    n_a = 100       # Total observations in group A

    # Data for group B
    success_b = 20  # Number of successes in group B
    n_b = 80        # Total observations in group B

    # Perform the Z-test for proportions
    z_statistic, p_value = z_test_proportions(success_a, n_a,
success_b, n_b)

    # Print the results
    print("Z-Statistic:", z_statistic)
    print("P-Value:", p_value)

    # Interpret the results
    alpha = 0.05
    if p_value < alpha:
        print("\nReject the null hypothesis: There is a significant
difference between the proportions.")
    else:
        print("\nFail to reject the null hypothesis: There is no
significant difference between the proportions.")
```

```
Z-Statistic: 0.7442084075352506
P-Value: 0.45675040242316944
```

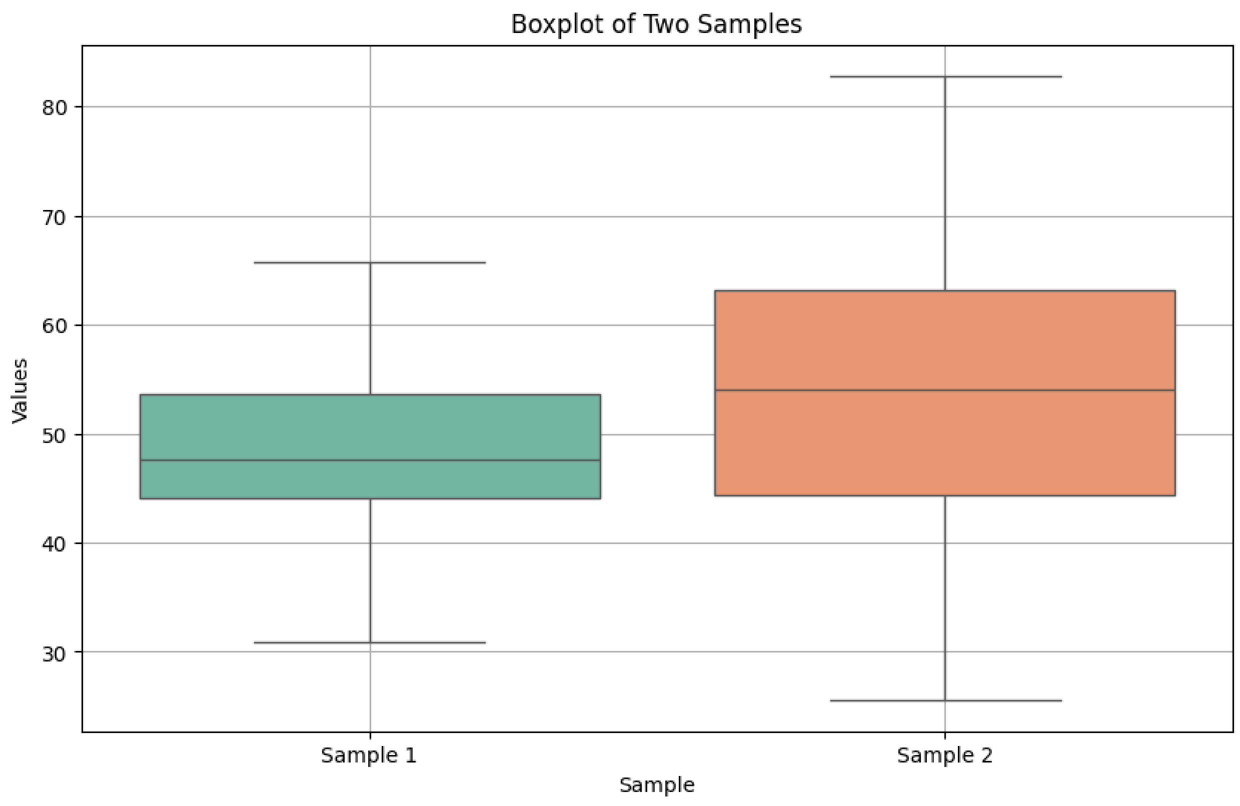
```
Fail to reject the null hypothesis: There is no significant difference
between the proportions.
```

Q26.Implement an F-test for comparing the variances of two datasets, then interpret and visualize the results.

Ans:-

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

def f_test(sample1, sample2):
    """
    Perform an F-test to compare the variances of two samples.

    Parameters:
    sample1 (array-like): The first sample data.
    sample2 (array-like): The second sample data.

    Returns:
    f_statistic (float): The F-statistic.
    p_value (float): The p-value of the test.
    """
    # Calculate the variances of the samples
    var1 = np.var(sample1, ddof=1)  # Sample variance
    var2 = np.var(sample2, ddof=1)  # Sample variance

    # Calculate the F-statistic
    f_statistic = var1 / var2

    # Calculate the degrees of freedom
    df1 = len(sample1) - 1  # Degrees of freedom for sample1
    df2 = len(sample2) - 1  # Degrees of freedom for sample2

    # Calculate the p-value
    p_value = 1 - stats.f.cdf(f_statistic, df1, df2)

    return f_statistic, p_value

# Example usage
if __name__ == "__main__":
    # Generate sample data
    np.random.seed(42)  # For reproducibility
    sample1 = np.random.normal(loc=50, scale=10, size=30)  # Sample 1
    sample2 = np.random.normal(loc=55, scale=15, size=30)  # Sample 2

    # Perform the F-test
    f_statistic, p_value = f_test(sample1, sample2)

    # Print the results
    print("F-Statistic:", f_statistic)
    print("P-Value:", p_value)

    # Interpret the results
    alpha = 0.05
    if p_value < alpha:
        print("\nReject the null hypothesis: The variances of the two
samples are significantly different.")
    else:
        print("\nFail to reject the null hypothesis: The variances of
the two samples are not significantly different.")

    # Visualize the results with boxplots
    plt.figure(figsize=(10, 6))
    sns.boxplot(data=[sample1, sample2], palette="Set2")
    plt.title('Boxplot of Two Samples')
    plt.xlabel('Sample')
    plt.ylabel('Values')
    plt.xticks([0, 1], ['Sample 1', 'Sample 2'])
    plt.grid()
    plt.show()
```

```
F-Statistic: 0.41525416799961246
P-Value: 0.9895622022746069

Fail to reject the null hypothesis: The variances of the two samples
are not significantly different.
```



Q27.Perform a Chi-square test for goodness of fit with simulated data and analyze the results.

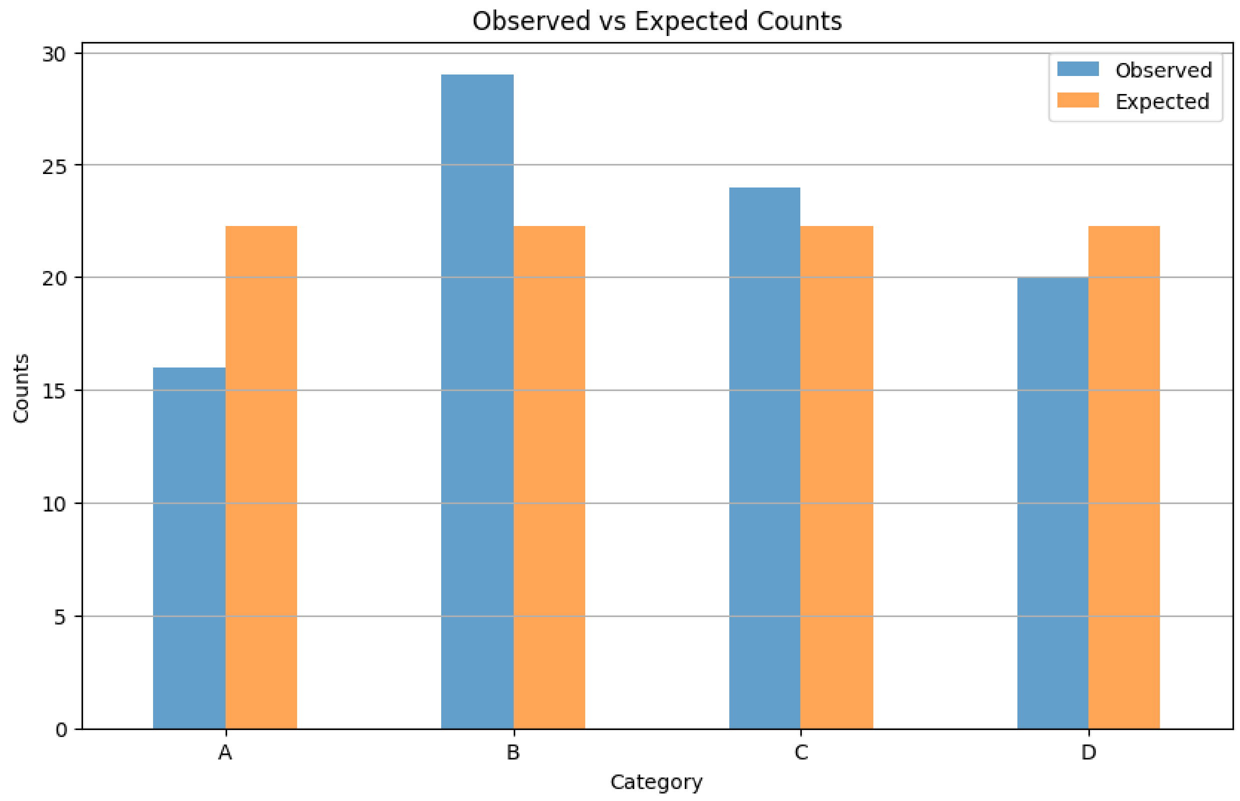
Ans:-Python Code for Chi-Square Goodness of Fit Test

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

def chi_square_goodness_of_fit(observed, expected):
    """
    Perform a Chi-square goodness of fit test.

    Parameters:
    observed (array-like): Observed frequencies.
    expected (array-like): Expected frequencies.

    Returns:
    chi2_statistic (float): The Chi-square statistic.
    p_value (float): The p-value of the test.
    """
    chi2_statistic, p_value = stats.chisquare(f_obs=observed,
f_exp=expected)
    return chi2_statistic, p_value

# Simulate categorical data
```

```
np.random.seed(42)  # For reproducibility
categories = ['A', 'B', 'C', 'D']
observed_counts = np.random.randint(10, 30, size=len(categories))  #
Simulated observed counts

# Define expected proportions (e.g., equal distribution)
expected_proportions = [0.25, 0.25, 0.25, 0.25]  # Equal proportions
for 4 categories
expected_counts = np.array(expected_proportions) *
sum(observed_counts)  # Calculate expected counts

# Perform the Chi-square goodness of fit test
chi2_statistic, p_value = chi_square_goodness_of_fit(observed_counts,
expected_counts)

# Print the results
print("Observed Counts:", observed_counts)
print("Expected Counts:", expected_counts)
print("Chi-square Statistic:", chi2_statistic)
print("P-Value:", p_value)

# Interpret the results
alpha = 0.05
if p_value < alpha:
    print("\nReject the null hypothesis: The observed frequencies
differ significantly from the expected frequencies.")
else:
    print("\nFail to reject the null hypothesis: The observed
frequencies do not differ significantly from the expected
frequencies.")

# Visualize the results
df = pd.DataFrame({'Category': categories, 'Observed':
observed_counts, 'Expected': expected_counts})
df.set_index('Category', inplace=True)

# Plotting
df.plot(kind='bar', figsize=(10, 6), alpha=0.7)
plt.title('Observed vs Expected Counts')
plt.ylabel('Counts')
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.show()
```

```
Observed Counts: [16 29 24 20]
Expected Counts: [22.25 22.25 22.25 22.25]
Chi-square Statistic: 4.168539325842697
P-Value: 0.24383063094585364
```

```
Fail to reject the null hypothesis: The observed frequencies do not
differ significantly from the expected frequencies.
```



Theory Question Answer Assignment :- https://drive.google.com/file/d/1RCj49Bgsz5JHPqj-0qmjIlasw18aj4r8/view?usp=drivesdk