Statistics Practice Questions

In [6]:
```
#Q1.Generate a list of 100 integers containing values between 90 to 130 and store it in the variable `int_list`
#A>ter generating the list, >find the >following:
import numpy as np

int_list=np.random.randint(90,131,size=100)
print("the int_list",int_list)
```

```
the int_list [107 104 115 114  94 123 130 114 105 127 117 103  90  95  90 115 117 130
  91 110 127  93 103  94 123  96  98 119 114 120 101 109 128 108  98  91
 103 113 104 100 112 107 111 112 100 122 112 122  92 119 106  95 115 129
 106 110 102  93 118  90  93 110  93 115 111 127 115 123 119  92 126 121
  99 102  94 105 116 115 102 122 104 118  91 112  90 113  92 127 126 117
 124 107  94  97 116 102  99 105 129 108]
```

In [31]:
```
#(i)Write a Python function to calculate the mean of a given list of numbers.Create a function to find the medi


int_list=np.random.randint(90,131,size=100)
print("the int_list",int_list)
def calculate_mean(numbers):
    return sum(numbers) / len(numbers)

mean_value = calculate_mean(int_list)
print("Mean:", mean_value)


def calculate_median(numbers):
    sorted_numbers = sorted(numbers)
    n = len(sorted_numbers)
    mid = n // 2
    if n % 2 == 0:
        return (sorted_numbers[mid - 1] + sorted_numbers[mid]) / 2
    else:
        return sorted_numbers[mid]

median_value = calculate_median(int_list)
print("Median:", median_value)
```

```
the int_list [110 107 127 115 111 126  94 124 101 124 123 109 101 123 115 123 121  96
 115 114 128 106 113 116  93 116 119  92  95  97 105 104 125 129  97 128
 119 101  93 103 114  94 112 119 128 101 104 108 115  93 127  97 119 104
 101  99 101 106 129  91 108 118  90 111 114  99  98 108 101 118 107 105
  93 114 111 118 111 104  91 102 101 115 129 101 116 124  91 106  91 106
 123 123 128 130  98 103 109 122 113  94]
Mean: 109.64
Median: 109.0
```

In [33]:
```
#(ii) Develop a program to compute the mode of a list of integers.

from collections import Counter

def calculate_mode(numbers):
    frequency = Counter(numbers)
    mode_data = frequency.most_common(1)
    return mode_data[0][0]

mode_value = calculate_mode(int_list)
print("Mode:", mode_value)
```

```
Mode: 101
```

In [34]:
```
#(iii)Implement a function to calculate the weighted mean of a list of values and their corresponding weights.

def weighted_mean(values, weights):
    return sum(value * weight for value, weight in zip(values, weights)) / sum(weights)

# Example usage
weights = np.random.randint(1, 5, size=100).tolist()  # Generate some random weights
weighted_mean_value = weighted_mean(int_list, weights)
print("Weighted Mean:", weighted_mean_value)
```

```
Weighted Mean: 108.90625
```

In [35]:
```
# (iv) Write a Python function to find the geometric mean of a list of positive numbers.

import math

def geometric_mean(numbers):
    product = 1
    for num in numbers:
        product *= num
    return product ** (1 / len(numbers))

geometric_mean_value = geometric_mean(int_list)
```

```
print("Geometric Mean:", geometric_mean_value)
```

```
Geometric Mean: 0.0
```

```
C:\Users\om\AppData\Local\Temp\ipykernel_5700\3147972371.py:8: RuntimeWarning: overflow encountered in scalar m
ultiply
  product *= num
```

In [36]:

```
#v) Create a program to calculate the harmonic mean of a list of values.

def harmonic_mean(numbers):
    return len(numbers) / sum(1 / num for num in numbers)

harmonic_mean_value = harmonic_mean(int_list)
print("Harmonic Mean:", harmonic_mean_value)
```

```
Harmonic Mean: 108.44289444796638
```

In [37]:

```
# (vi) Build a function to determine the midrange of a list of numbers (average of the minimum and maximum).

def midrange(numbers):
    return (max(numbers) + min(numbers)) / 2

midrange_value = midrange(int_list)
print("Midrange:", midrange_value)
```

```
Midrange: 110.0
```

In [38]:

```
# (vii) Implement a Python program to find the trimmed mean of a list, excluding a certain percentage of
#outliers.

from scipy.stats import trim_mean

def trimmed_mean(numbers, percentage):
    return trim_mean(numbers, proportiontocut=percentage)

trimmed_mean_value = trimmed_mean(int_list, 0.1)  # 10% trimmed mean
print("Trimmed Mean (10%):", trimmed_mean_value)
```

```
Trimmed Mean (10%): 109.5375
```

In [ ]:

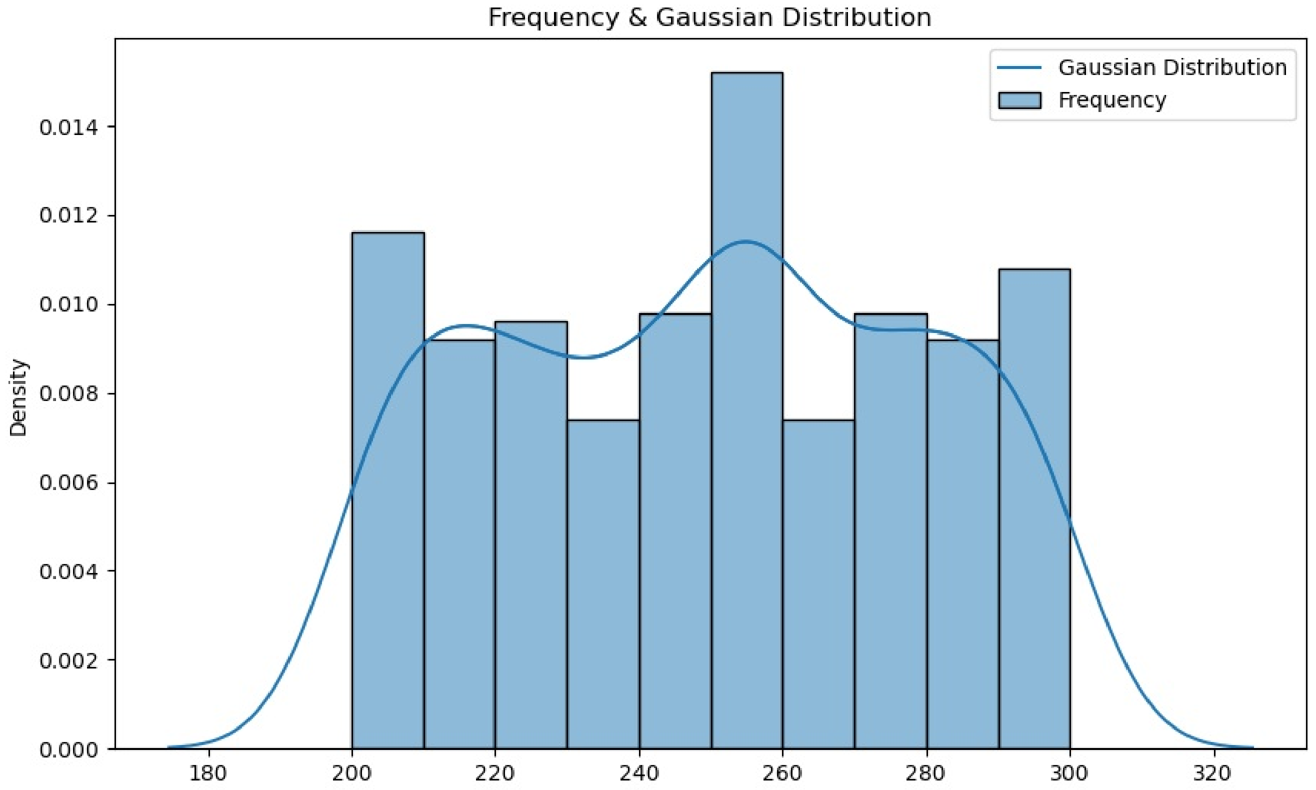
```

```

In [34]:

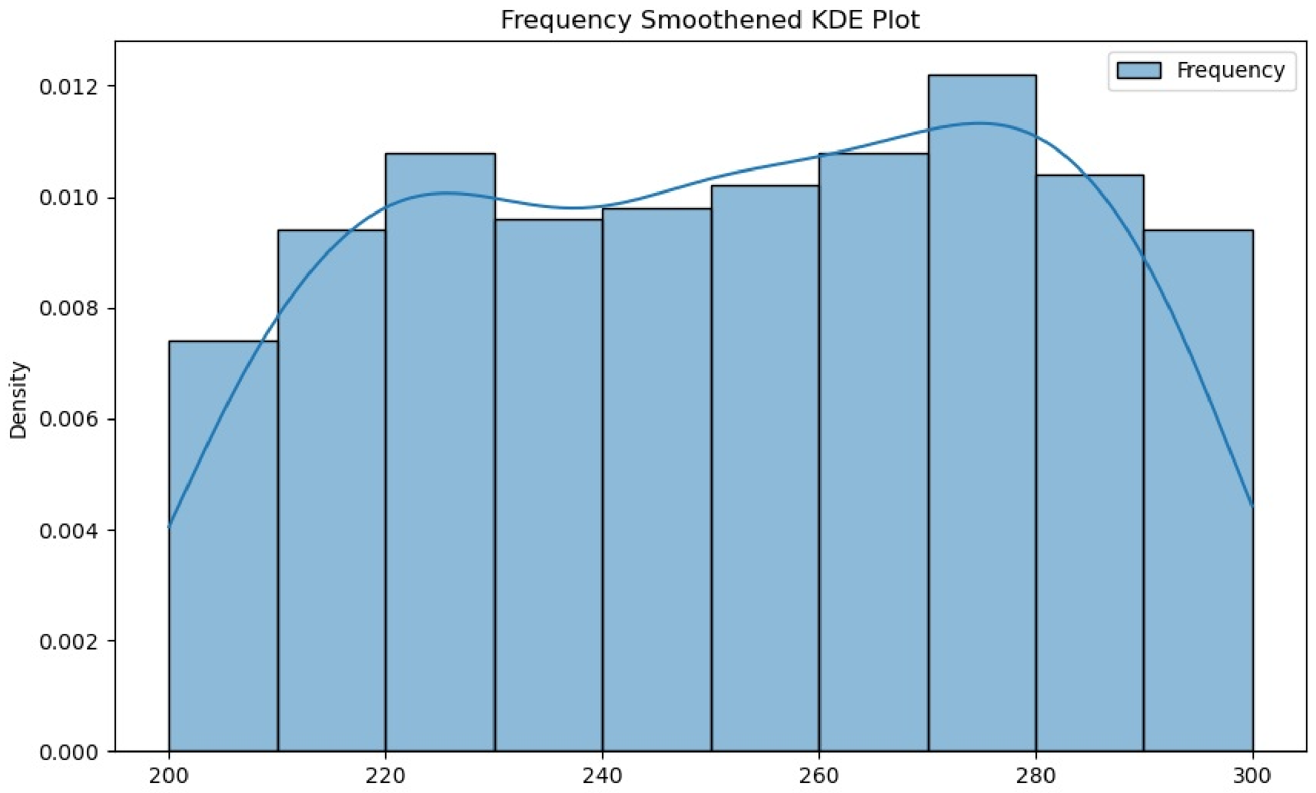
```
#Q 2. Generate a list of 500 integers containing values between 200 to 300 and store it in the variable `int_li
#After generating the list, find the following:

 #(i) Compare the given list of visualization for the given data:



   # 1. Frequency & Gaussian distribution

   # 2. Frequency smoothened KDE plot

   # 3. Gaussian distribution & smoothened KDE plot
```

In [35]:

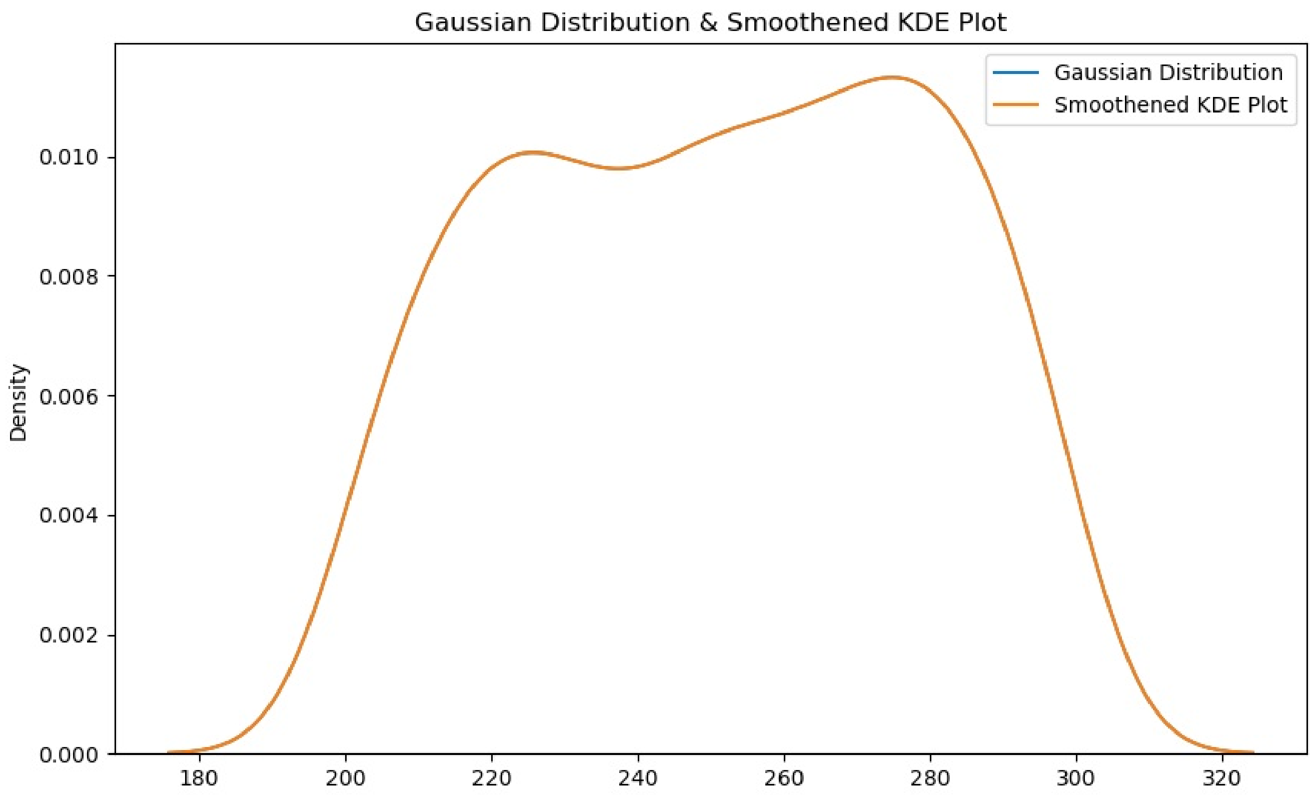
```
#(i)Frequency & Gaussian distribution

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

import matplotlib.pyplot as plt
import seaborn as sns

import random
int_list2 = [random.randint(200, 300) for _ in range(500)]

# Frequency & Gaussian distribution
plt.figure(figsize=(10, 6))
sns.histplot(int_list2, kde=True, stat="density", label="Frequency")
sns.kdeplot(int_list2, label="Gaussian Distribution")
plt.legend()
plt.title("Frequency & Gaussian Distribution")
plt.show()
```



In [24]:
```python
# Frequency smoothened KDE plot
plt.figure(figsize=(10, 6))
sns.histplot(int_list2, kde=True, stat="density", label="Frequency")
plt.legend()
plt.title("Frequency Smoothened KDE Plot")
plt.show()
```



In [25]:
```python
# Gaussian distribution & smoothened KDE plot
plt.figure(figsize=(10, 6))
sns.kdeplot(int_list2, label="Gaussian Distribution")
sns.kdeplot(int_list2, label="Smoothened KDE Plot")
plt.legend()
plt.title("Gaussian Distribution & Smoothened KDE Plot")
plt.show()
```



In [36]:
```
# Q.3)Write a Python class representing a discrete random variable with methods to calculate its expected
#value and variance.

class DiscreteRandomVariable:
    def __init__(self, values, probabilities):
        """
        Initialize the discrete random variable.

        :param values: List of possible values of the random variable.
        :param probabilities: List of probabilities corresponding to each value.
        """
        if len(values) != len(probabilities):
            raise ValueError("The length of values and probabilities must be the same.")
        if not all(0 <= p <= 1 for p in probabilities):
            raise ValueError("Probabilities must be between 0 and 1.")
        if not abs(sum(probabilities) - 1) < 1e-6:
            raise ValueError("The sum of probabilities must be 1.")

        self.values = values
        self.probabilities = probabilities

    def expected_value(self):
        """
        Calculate the expected value (mean) of the random variable.

        :return: Expected value.
        """
        return sum(value * prob for value, prob in zip(self.values, self.probabilities))

    def variance(self):
        """
        Calculate the variance of the random variable.

        :return: Variance.
        """
        mean = self.expected_value()
        return sum(prob * (value - mean) ** 2 for value, prob in zip(self.values, self.probabilities))

# Example usage:
values = [1, 2, 3, 4, 5]
probabilities = [0.1, 0.2, 0.3, 0.2, 0.2]

drv = DiscreteRandomVariable(values, probabilities)
print("Expected Value:", drv.expected_value())
print("Variance:", drv.variance())
```

```
Expected Value: 3.2
Variance: 1.56
```

In [39]:
```
#Q.4) Implement a program to simulate the rolling of a fair six-sided die and calculate the expected value and
#variance of the outcomes.

import random
import numpy as np
```

```
class DieSimulator:
    def __init__(self, num_rolls):
        self.num_rolls = num_rolls
        self.outcomes = []

    def roll_die(self):
        self.outcomes = [random.randint(1, 6) for _ in range(self.num_rolls)]

    def expected_value(self):
        return np.mean(self.outcomes)

    def variance(self):
        return np.var(self.outcomes)

# Simulate rolling the die 1000 times
num_rolls = 1000
simulator = DieSimulator(num_rolls)
simulator.roll_die()

# Calculate expected value and variance
expected_value = simulator.expected_value()
variance = simulator.variance()

print(f"Expected Value: {expected_value}")
print(f"Variance: {variance}")
```

```
Expected Value: 3.531
Variance: 2.9070389999999997
```

In [39]:

```
#Q.5)Create a Python function to generate random samples from a given probability distribution (e.g.,
#binomial, Poisson) and calculate their mean and variance.

import numpy as np
from scipy.stats import binom, poisson

def generate_samples_and_calculate_stats(distribution, params, sample_size):
    """
    Generate random samples from a given probability distribution and calculate their mean and variance.

    :param distribution: str, type of distribution ('binomial' or 'poisson')
    :param params: dict, parameters for the distribution
                   For 'binomial': {'n': number of trials, 'p': probability of success}
                   For 'poisson': {'mu': mean number of events}
    :param sample_size: int, number of samples to generate
    :return: tuple, (mean, variance) of the generated samples
    """
    if distribution == 'binomial':
        n = params.get('n')
        p = params.get('p')
        samples = binom.rvs(n, p, size=sample_size)
    elif distribution == 'poisson':
        mu = params.get('mu')
        samples = poisson.rvs(mu, size=sample_size)
    else:
        raise ValueError("Unsupported distribution type. Use 'binomial' or 'poisson'.")

    mean = np.mean(samples)
    variance = np.var(samples)

    return mean, variance

# Example usage:
binomial_params = {'n': 10, 'p': 0.5}
poisson_params = {'mu': 3}

binomial_mean, binomial_variance = generate_samples_and_calculate_stats('binomial', binomial_params, 1000)
poisson_mean, poisson_variance = generate_samples_and_calculate_stats('poisson', poisson_params, 1000)

print(f"Binomial Distribution - Mean: {binomial_mean}, Variance: {binomial_variance}")
print(f"Poisson Distribution - Mean: {poisson_mean}, Variance: {poisson_variance}")
```

```
Binomial Distribution - Mean: 4.955, Variance: 2.566975
Poisson Distribution - Mean: 3.136, Variance: 3.075504
```

In [41]:

```
#Q.6)}) Write a Python script to generate random numbers 6rom a Gaussian (normal) distribution and compute
#the mean, variance, and standard deviation o6 the samples

import numpy as np

# Generate random numbers from a Gaussian distribution
mean = 0
std_dev = 1
num_samples = 1000
samples = np.random.normal(loc=mean, scale=std_dev, size=num_samples)

# Compute the mean, variance, and standard deviation
computed_mean = np.mean(samples)
computed_variance = np.var(samples)
computed_std_dev = np.std(samples)
```

```
print(f"Generated {num_samples} samples from a Gaussian distribution with mean {mean} and standard deviation {s
print(f"Computed Mean: {computed_mean}")
print(f"Computed Variance: {computed_variance}")
print(f"Computed Standard Deviation: {computed_std_dev}")
```

```
Generated 1000 samples from a Gaussian distribution with mean 0 and standard deviation 1
Computed Mean: 0.023919801044198642
Computed Variance: 1.0100433276234895
Computed Standard Deviation: 1.0050091181792777
```

In [42]:
```
#Q.7)Use seaborn library to load tips dataset. Find the following from the dataset for the columns total_bill
#and tip`:

  #(i) Write a Python function that calculates their skewness.

import seaborn as sns
import scipy.stats as stats

# Load the tips dataset
tips = sns.load_dataset("tips")

# Function to calculate skewness
def calculate_skewness(data, column):
    return stats.skew(data[column])

# Calculate skewness for total_bill and tip
total_bill_skewness = calculate_skewness(tips, 'total_bill')
tip_skewness = calculate_skewness(tips, 'tip')

print(f"Skewness of total_bill: {total_bill_skewness}")
print(f"Skewness of tip: {tip_skewness}")
```

```
Skewness of total_bill: 1.1262346334818638
Skewness of tip: 1.4564266884221506
```

In [43]:
```
#ii) Create a program that determines whether the columns exhibit posittive skewness, negative skewness, or is
#approximately symmetric.

import seaborn as sns
import scipy.stats as stats

# Load the tips dataset
tips = sns.load_dataset("tips")

# Function to calculate skewness and determine its type
def determine_skewness(data, column):
    skewness = stats.skew(data[column])
    if skewness > 0:
        skew_type = "positive skewness"
    elif skewness < 0:
        skew_type = "negative skewness"
    else:
        skew_type = "approximately symmetric"
    return skewness, skew_type

# Determine skewness for total_bill and tip
total_bill_skewness, total_bill_skew_type = determine_skewness(tips, 'total_bill')
tip_skewness, tip_skew_type = determine_skewness(tips, 'tip')

print(f"Total Bill - Skewness: {total_bill_skewness}, Type: {total_bill_skew_type}")
print(f"Tip - Skewness: {tip_skewness}, Type: {tip_skew_type}")
```

```
Total Bill - Skewness: 1.1262346334818638, Type: positive skewness
Tip - Skewness: 1.4564266884221506, Type: positive skewness
```

In [44]:
```
#iii) Write a function that calculates the covariance between two columns

import seaborn as sns
import numpy as np

# Load the tips dataset
tips = sns.load_dataset("tips")

# Function to calculate covariance between two columns
def calculate_covariance(data, column1, column2):
    covariance_matrix = np.cov(data[column1], data[column2])
    covariance = covariance_matrix[0, 1]
    return covariance

# Calculate covariance between total_bill and tip
covariance_total_bill_tip = calculate_covariance(tips, 'total_bill', 'tip')

print(f"Covariance between total_bill and tip: {covariance_total_bill_tip}")
```

```
Covariance between total_bill and tip: 8.323501629224854
```

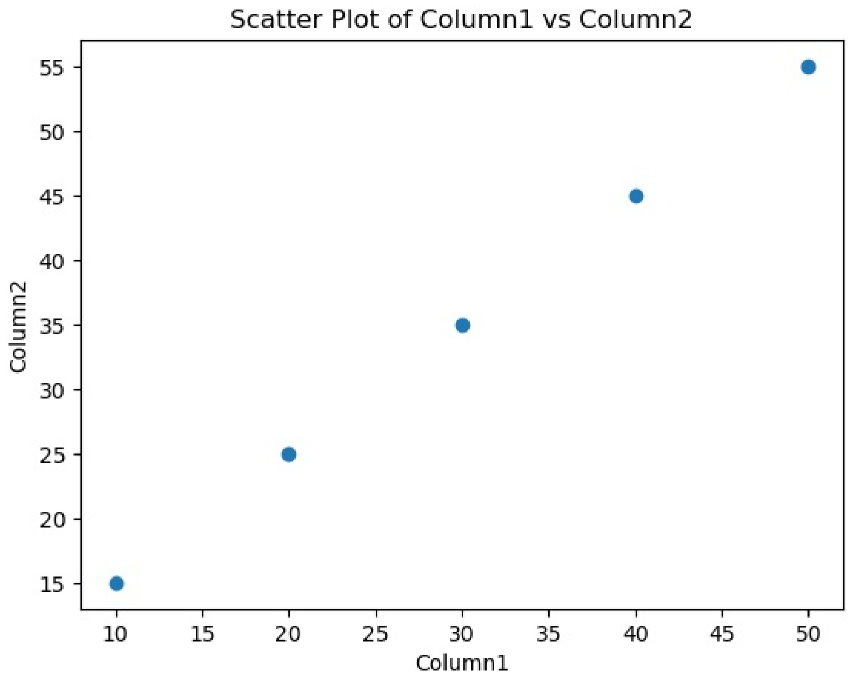
In [45]:
```
#iv)Implement a Python program that calculates the Pearson correlation coefficient between two columns.

import pandas as pd
```

```
from scipy.stats import pearsonr

# Sample data
data = {
    'Column1': [10, 20, 30, 40, 50],
    'Column2': [15, 25, 35, 45, 55]
}

# Create a DataFrame
df = pd.DataFrame(data)

# Calculate Pearson correlation coefficient
corr, _ = pearsonr(df['Column1'], df['Column2'])

print(f'Pearson correlation coefficient: {corr}')
```

```
Pearson correlation coefficient: 1.0
```

In [46]:

```
#(v) Write a script to visualize the correlation between two specific columns in a Pandas DataFrame using
#scatter plots.


import pandas as pd
import matplotlib.pyplot as plt

# Sample data
data = {
    'Column1': [10, 20, 30, 40, 50],
    'Column2': [15, 25, 35, 45, 55]
}

# Create a DataFrame
df = pd.DataFrame(data)

# Create a scatter plot
plt.scatter(df['Column1'], df['Column2'])

# Add title and labels
plt.title('Scatter Plot of Column1 vs Column2')
plt.xlabel('Column1')
plt.ylabel('Column2')

# Show plot
plt.show()
```



In [47]:

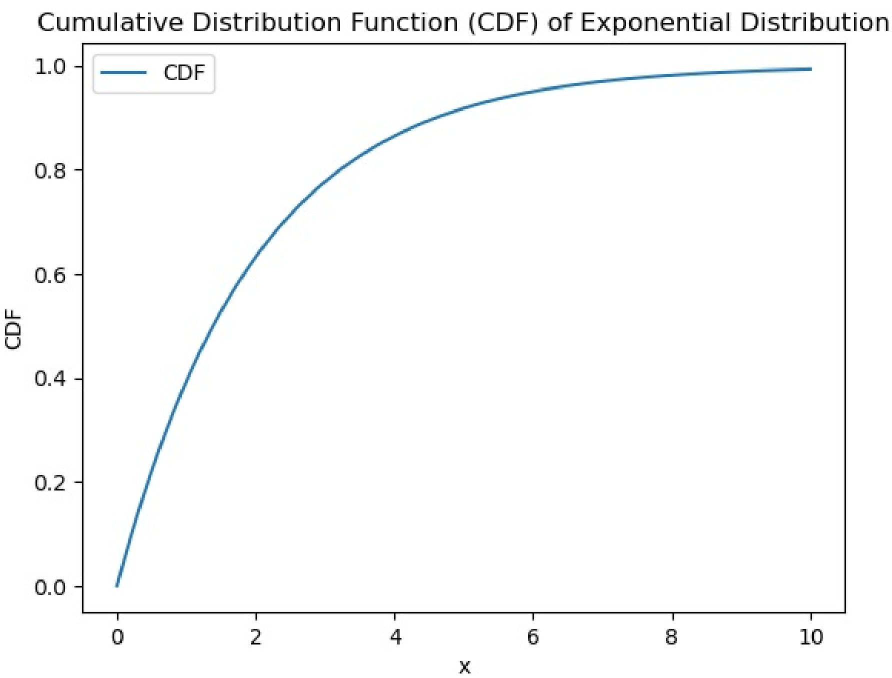
```
#Q.8) Write a Python 6unction to calculate the probability density function (PDF) o6 a continuous random
#variable for a given normal distribution.


import numpy as np
from scipy.stats import norm

def calculate_pdf(x, mean, std_dev):
    """
    Calculate the PDF of a normal distribution.

    Parameters:
    x (float or array-like): The value(s) at which to evaluate the PDF.
    mean (float): The mean of the normal distribution.
    std_dev (float): The standard deviation of the normal distribution.
```

```
    Returns:
    float or array-like: The PDF value(s) at x.
    """
    pdf = norm.pdf(x, loc=mean, scale=std_dev)
    return pdf

# Example usage
x_values = np.linspace(-10, 10, 100)
mean = 0
std_dev = 2
pdf_values = calculate_pdf(x_values, mean, std_dev)

print(pdf_values)
```

```
[7.43359757e-07 1.22553052e-06 1.99994519e-06 3.23058320e-06
 5.16550329e-06 8.17547945e-06 1.28080406e-05 1.98619112e-05
 3.04879520e-05 4.63238177e-05 6.96705616e-05 1.03720154e-04
 1.52843113e-04 2.22944862e-04 3.21897749e-04 4.60052385e-04
 6.50826921e-04 9.11365548e-04 1.26324789e-03 1.73321896e-03
 2.35389538e-03 3.16438821e-03 4.21076724e-03 5.54627742e-03
 7.23120740e-03 9.33230497e-03 1.19216373e-02 1.50748070e-02
 1.88684616e-02 2.33770712e-02 2.86690026e-02 3.48019792e-02
 4.18180886e-02 4.97385694e-02 5.85586798e-02 6.82430046e-02
 7.87215938e-02 8.98873326e-02 1.01594918e-01 1.13661753e-01
 1.25870973e-01 1.37976686e-01 1.49711342e-01 1.60795012e-01
 1.70946147e-01 1.79893279e-01 1.87386990e-01 1.93211427e-01
 1.97194617e-01 1.99216901e-01 1.99216901e-01 1.97194617e-01
 1.93211427e-01 1.87386990e-01 1.79893279e-01 1.70946147e-01
 1.60795012e-01 1.49711342e-01 1.37976686e-01 1.25870973e-01
 1.13661753e-01 1.01594918e-01 8.98873326e-02 7.87215938e-02
 6.82430046e-02 5.85586798e-02 4.97385694e-02 4.18180886e-02
 3.48019792e-02 2.86690026e-02 2.33770712e-02 1.88684616e-02
 1.50748070e-02 1.19216373e-02 9.33230497e-03 7.23120740e-03
 5.54627742e-03 4.21076724e-03 3.16438821e-03 2.35389538e-03
 1.73321896e-03 1.26324789e-03 9.11365548e-04 6.50826921e-04
 4.60052385e-04 3.21897749e-04 2.22944862e-04 1.52843113e-04
 1.03720154e-04 6.96705616e-05 4.63238177e-05 3.04879520e-05
 1.98619112e-05 1.28080406e-05 8.17547945e-06 5.16550329e-06
 3.23058320e-06 1.99994519e-06 1.22553052e-06 7.43359757e-07]
```

In [48]:

```
#Q¸9) Create a program to calculate the cumulative distribution function (CDF) of exponential distribution.

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

def calculate_cdf(lam, x_values):
    """
    Calculate the CDF of an exponential distribution.

    Parameters:
    lam (float): The rate parameter (lambda) of the exponential distribution.
    x_values (array-like): The values at which to evaluate the CDF.

    Returns:
    array-like: The CDF values at x_values.
    """
    cdf_values = expon.cdf(x_values, scale=1/lam)
    return cdf_values

# Example usage
lam = 0.5  # Rate parameter (lambda)
x_values = np.linspace(0, 10, 100)
cdf_values = calculate_cdf(lam, x_values)

# Plotting the CDF
plt.plot(x_values, cdf_values, label='CDF')
plt.title('Cumulative Distribution Function (CDF) of Exponential Distribution')
plt.xlabel('x')
plt.ylabel('CDF')
plt.legend()
plt.show()
```



In [49]:

```
#Q.10) Write a Python function to calculate the probability mass 6unction (PMF) of Poisson distribution.

import math

def poisson_pmf(k, mu):
    """
    Calculate the Probability Mass Function (PMF) of Poisson distribution.

    Parameters:
    k (int): The number of occurrences.
    mu (float): The average number of occurrences (rate parameter).

    Returns:
    float: The probability of observing k occurrences.
    """
    return (math.exp(-mu) * mu**k) / math.factorial(k)

# Example usage:
k = 3
mu = 2.5
print(f"The probability of observing {k} occurrences is {poisson_pmf(k, mu):.4f}")
```

```
The probability of observing 3 occurrences is 0.2138
```

In [51]:

```
#Q.11)A company wants to test in a new website layout leads to a higher conversion rate (percentage oN visitors
#who make a purchase). They collect data from the old and new layouts to compare.

#To ge<erate the data use the following command:



import numpy as np

# 50 purchases out of 1000 visitors

#old_layout = np.array([1] * 50 + [0] * 950)

# 70 purchases out of 1000 visitors

#New_layout = np.array([1] * 70 + [0] * 930)



#Apply z-test to fi<d which layout is successful



import numpy as np
from statsmodels.stats.proportion import proportions_ztest

# Generate the data
old_layout = np.array([1] * 50 + [0] * 950)
new_layout = np.array([1] * 70 + [0] * 930)

# Calculate the number of successes and the number of trials for each layout
successes = np.array([old_layout.sum(), new_layout.sum()])
trials = np.array([len(old_layout), len(new_layout)])

# Perform the two-sample z-test for proportions
```

```python
z_stat, p_value = proportions_ztest(successes, trials)

print(f"Z-statistic: {z_stat:.4f}")
print(f"P-value: {p_value:.4f}")

# Interpret the result
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant difference in conversion rates.")
    if successes[1] / trials[1] > successes[0] / trials[0]:
        print("The new layout has a higher conversion rate.")
    else:
        print("The old layout has a higher conversion rate.")
else:
    print("Fail to reject the null hypothesis: There is no significant difference in conversion rates.")
```

```
Z-statistic: -1.8831
P-value: 0.0597
Fail to reject the null hypothesis: There is no significant difference in conversion rates.
```

In [52]:

```python
#Q.12) A tutoring service claims that its program improves students' exam scores. A sample oN students who
#participated in the program was taken, and their scores beNore and aNter the program were recorded.


#Use the below code to ge<erate samples of respective arrays of marks:



#before_program = <p.array([75, 80, 85, 70, 90, 78, 92, 88, 82, 87])

#after_program = <p.array([80, 85, 90, 80, 92, 80, 95, 90, 85, 88])


#Use z-test to fi<d if the claims made by tutor are true or false.




import numpy as np
from scipy import stats

# Generate the data
before_program = np.array([75, 80, 85, 70, 90, 78, 92, 88, 82, 87])
after_program = np.array([80, 85, 90, 80, 92, 80, 95, 90, 85, 88])

# Calculate the differences
differences = after_program - before_program

# Calculate the mean and standard deviation of the differences
mean_diff = np.mean(differences)
std_diff = np.std(differences, ddof=1)

# Calculate the z-score
n = len(differences)
z_score = mean_diff / (std_diff / np.sqrt(n))

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

print(f"Z-score: {z_score:.4f}")
print(f"P-value: {p_value:.4f}")

# Interpret the result
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: The tutoring program significantly improves exam scores.")
else:
    print("Fail to reject the null hypothesis: There is no significant improvement in exam scores.")
```

```
Z-score: 4.5932
P-value: 0.0000
Reject the null hypothesis: The tutoring program significantly improves exam scores.
```

In [54]:

```python
#Q.13)A pharmaceutical company wants to determine iN a new drug is eNNective in reducing blood pressure. They
#conduct a study and record blood pressure measurements beNore and aNter administering the drug.

#Use the below code to ge<erate samples of respective arrays of blood pressure:
#before_drug = <p.array([145, 150, 140, 135, 155, 160, 152, 148, 130, 138])

#after_drug = <p.array([130, 140, 132, 128, 145, 148, 138, 136, 125, 130])

#Impleme<t z-test to fi<d if the drug really works or <ot.

import numpy as np
from scipy import stats

# Blood pressure measurements before and after administering the drug
before_drug = np.array([145, 150, 140, 135, 155, 160, 152, 148, 130, 138])
after_drug = np.array([130, 140, 132, 128, 145, 148, 138, 136, 125, 130])
```

```python
# Calculate the mean and standard deviation of the differences
differences = before_drug - after_drug
mean_diff = np.mean(differences)
std_diff = np.std(differences, ddof=1)

# Number of samples
n = len(differences)

# Calculate the z-score
z_score = mean_diff / (std_diff / np.sqrt(n))

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

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

# Determine if the drug is effective
alpha = 0.05
if p_value < alpha:
    print("The drug is effective in reducing blood pressure.")
else:
    print("The drug is not effective in reducing blood pressure.")
```

```
Z-score: 10.049875621120888
P-value: 0.0
The drug is effective in reducing blood pressure.
```

In [55]:

```python
#Q.14)A customer service department claims that their average response time is less than 5. A sample
#of recent custo:er interactions was taken, and the response ti:es were recorded.

#Implement the below code to ge3erate the array of respo3se time:

#response_times = np.array([4.3, 3.8, 5.1, 4.9, 4.7, 4.2, 5.2, 4.5, 4.6, 4.4])

#Implement z-test to find the claims made by customer service departme3t are true or false


import numpy as np
from scipy import stats

# Response times recorded
response_times = np.array([4.3, 3.8, 5.1, 4.9, 4.7, 4.2, 5.2, 4.5, 4.6, 4.4])

# Hypothesized mean response time
mu = 5

# Sample mean and standard deviation
sample_mean = np.mean(response_times)
sample_std = np.std(response_times, ddof=1)

# Number of samples
n = len(response_times)

# Calculate the z-score
z_score = (sample_mean - mu) / (sample_std / np.sqrt(n))

# Calculate the p-value
p_value = stats.norm.cdf(z_score)

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

# Determine if the claim is true
alpha = 0.05
if p_value < alpha:
    print("The claim that the average response time is less than 5 minutes is true.")
else:
    print("The claim that the average response time is less than 5 minutes is false.")
```

```
Z-score: -3.184457226042963
P-value: 0.0007251287113068958
The claim that the average response time is less than 5 minutes is true.
```

In [57]:

```python
#Q.15)2£U A company is testing two different website layouts to see which one leads to higher click-through rat
#Write a Python function to perform an A/B test analysis, including calculating the t-statistic, degrees of
#freedom, and p-value.

#Use the followi3g data:


#layout_a_clicks = [28, 32, 33, 29, 31, 34, 30, 35, 36, 37]

#layout_b_clicks = [40, 41, 38, 42, 39, 44, 43, 41, 45, 47]



import numpy as np
```

```python
from scipy import stats

def ab_test_analysis(layout_a_clicks, layout_b_clicks):
    # Calculate the means
    mean_a = np.mean(layout_a_clicks)
    mean_b = np.mean(layout_b_clicks)

    # Calculate the standard deviations
    std_a = np.std(layout_a_clicks, ddof=1)
    std_b = np.std(layout_b_clicks, ddof=1)

    # Calculate the number of observations
    n_a = len(layout_a_clicks)
    n_b = len(layout_b_clicks)

    # Calculate the t-statistic
    t_statistic = (mean_a - mean_b) / np.sqrt((std_a**2 / n_a) + (std_b**2 / n_b))

    # Calculate the degrees of freedom
    df = ((std_a**2 / n_a) + (std_b**2 / n_b))**2 / (((std_a**2 / n_a)**2 / (n_a - 1)) + ((std_b**2 / n_b)**2 /

    # Calculate the p-value
    p_value = stats.t.sf(np.abs(t_statistic), df) * 2  # two-tailed test

    return t_statistic, df, p_value

# Example data
layout_a_clicks = [28, 32, 33, 29, 31, 34, 30, 35, 36, 37]
layout_b_clicks = [40, 41, 38, 42, 39, 44, 43, 41, 45, 47]

# Perform A/B test analysis
t_statistic, df, p_value = ab_test_analysis(layout_a_clicks, layout_b_clicks)

print(f"T-statistic: {t_statistic}")
print(f"Degrees of freedom: {df}")
print(f"P-value: {p_value}")
```

```
T-statistic: -7.298102156175071
Degrees of freedom: 17.879871863320876
P-value: 9.196596070789357e-07
```

In [1]:

```python
#Q.16)A pharmaceutical company wants to determine if a new drug is more effective than an existing drug in
#reducing cholesterol levelsV Create a program to analyze the clinical trial data and calculate the tstatistic


#Use the followi3g data of cholestrol level:

#existing_drug_levels = [180, 182, 175, 185, 178, 176, 172, 184, 179, 183]

#new_drug_levels = [170, 172, 165, 168, 175, 173, 170, 178, 172, 176]


import scipy.stats as stats

# Data
existing_drug_levels = [180, 182, 175, 185, 178, 176, 172, 184, 179, 183]
new_drug_levels = [170, 172, 165, 168, 175, 173, 170, 178, 172, 176]

# Calculate the t-statistic and p-value
t_statistic, p_value = stats.ttest_ind(existing_drug_levels, new_drug_levels)

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

```
T-statistic: 4.14048098620866
P-value: 0.0006143398442372505
```

In [2]:

```python
#Q.17)A school district introduces an educational intervention program to improve :ath scoresV Write a Python
#function to analyze pre- and post-intervention test scores, calculating the t-statistic and p-value to
#determine if the intervention had a significant impact.

#Use the following data of test score:
#pre_i3terve3tio3_scores = [80, 85, 90, 75, 88, 82, 92, 78, 85, 87]
#post_i3terve3tio3_scores = [90, 92, 88, 92, 95, 91, 96, 93, 89, 93]


import scipy.stats as stats

def analyze_intervention(pre_scores, post_scores):
    # Calculate the t-statistic and p-value
    t_statistic, p_value = stats.ttest_rel(pre_scores, post_scores)

    return t_statistic, p_value

# Data
pre_intervention_scores = [80, 85, 90, 75, 88, 82, 92, 78, 85, 87]
post_intervention_scores = [90, 92, 88, 92, 95, 91, 96, 93, 89, 93]

# Analyze the intervention
```

```
t_stat, p_val = analyze_intervention(pre_intervention_scores, post_intervention_scores)

print(f"T-statistic: {t_stat}")
print(f"P-value: {p_val}")
```

```
T-statistic: -4.42840883965761
P-value: 0.0016509548165795493
```

In [8]:

```
#Q.18.) An HR department wants to investigate iF there's a gender-based salary gap within the company. Develop
#a program to analyze salary data, calculate the t-statistic, and determine if there's a statistically
#significant difference between the average salaries of male and female employees.

#Use the below code to generate synthetic data

# Generate synthetic salary data for male and female employees
import numpy as np

np.random.seed(0)  # For reproducibility

male_salaries = np.random.normal(loc=50000, scale=10000, size=20)

female_salaries = np.random.normal(loc=55000, scale=9000, size=20)


import numpy as np
import scipy.stats as stats

# Generate synthetic salary data for male and female employees
np.random.seed(0)  # For reproducibility

male_salaries = np.random.normal(loc=50000, scale=10000, size=20)
female_salaries = np.random.normal(loc=55000, scale=9000, size=20)

def analyze_salary_gap(male_salaries, female_salaries):
    # Calculate the t-statistic and p-value
    t_statistic, p_value = stats.ttest_ind(male_salaries, female_salaries)

    return t_statistic, p_value

# Analyze the salary gap
t_stat, p_val = analyze_salary_gap(male_salaries, female_salaries)

print(f"T-statistic: {t_stat}")
print(f"P-value: {p_val}")
```

```
T-statistic: 0.06114208969631383
P-value: 0.9515665020676465
```

In [10]:

```
#Q.19) A manufacturer produces two different versions of a product and wants to compare their quality scores.
#Create a Python function to analyze quality assessment data, calculate the t-statistic, and decide
#whether there's a significant difference in quality between the two versions.


#Use the following data

version1_scores = [85, 88, 82, 89, 87, 84, 90, 88, 85, 86, 91, 83, 87, 84, 89, 86, 84, 88, 85, 86, 89, 90, 87,

version2_scores = [80, 78, 83, 81, 79, 82, 76, 80, 78, 81, 77, 82, 80, 79, 82, 79, 80, 81, 79, 82, 79, 78, 80,


import scipy.stats as stats

def compare_quality_scores(version1_scores, version2_scores):
    # Calculate the t-statistic and p-value
    t_stat, p_value = stats.ttest_ind(version1_scores, version2_scores)

    # Print the results
    print(f"T-statistic: {t_stat}")
    print(f"P-value: {p_value}")

    # Determine if there's a significant difference
    alpha = 0.05  # significance level
    if p_value < alpha:
        print("There is a significant difference in quality between the two versions.")
    else:
        print("There is no significant difference in quality between the two versions.")

# Data
version1_scores = [85, 88, 82, 89, 87, 84, 90, 88, 85, 86, 91, 83, 87, 84, 89, 86, 84, 88, 85, 86, 89, 90, 87,
version2_scores = [80, 78, 83, 81, 79, 82, 76, 80, 78, 81, 77, 82, 80, 79, 82, 79, 80, 81, 79, 82, 79, 78, 80,

# Compare the quality scores
compare_quality_scores(version1_scores, version2_scores)
```

```
T-statistic: 11.325830417646698
P-value: 3.6824250702873965e-15
There is a significant difference in quality between the two versions.
```

In [11]:

```
#Q.20) A restaurant chain collects customer satisfaction scores for two different branches. Write a program to
```

```
#analyze the scores, calculate the t-statistic, and determine if there's a statistically significant difference
#customer satisfaction between the branches.

#Use the below data of scores:



#branch_a_scores = [4, 5, 3, 4, 5, 4, 5, 3, 4, 4, 5, 4, 4, 3, 4, 5, 5, 4, 3, 4, 5, 4, 3, 5, 4, 4, 5, 3, 4, 5, 4

#branch_b_scores = [3, 4, 2, 3, 4, 3, 4, 2, 3, 3, 4, 3, 3, 2, 3, 4, 4, 3, 2, 3, 4, 3, 2, 4, 3, 3, 4, 2, 3, 4, 3


import numpy as np
from scipy import stats

# Customer satisfaction scores for two branches
branch_a_scores = [4, 5, 3, 4, 5, 4, 5, 3, 4, 4, 5, 4, 4, 3, 4, 5, 5, 4, 3, 4, 5, 4, 3, 5, 4, 4, 5, 3, 4, 5, 4]
branch_b_scores = [3, 4, 2, 3, 4, 3, 4, 2, 3, 3, 4, 3, 3, 2, 3, 4, 4, 3, 2, 3, 4, 3, 2, 4, 3, 3, 4, 2, 3, 4, 3]

# Calculate the t-statistic and p-value
t_stat, p_value = stats.ttest_ind(branch_a_scores, branch_b_scores)

# Output the results
print(f"T-statistic: {t_stat}")
print(f"P-value: {p_value}")

# Determine if there's a statistically significant difference
alpha = 0.05
if p_value < alpha:
    print("There is a statistically significant difference in customer satisfaction between the two branches.")
else:
    print("There is no statistically significant difference in customer satisfaction between the two branches."
```

```
T-statistic: 5.480077554195743
P-value: 8.895290509945655e-07
There is a statistically significant difference in customer satisfaction between the two branches.
```

In [19]:

```
#Q.21)RV? A political analyst wants to determine if there is a significant association between age groups and v
#preferences Candidate A or Candidate B). They collect data from a sample of 500 voters and classify
#them into different age groups and candidate preferences. Perform a Chi-Square test to determine if
#there is a significant association between age groups and voter prefrences.

#Use the below code to generate data:

#np.random.seed(0)

#age_groups = np.random.choice([ 18 30 , 31 50 , '51+, '51+'], size=30)

#voter_preferences = np.random.choice(['Candidate A', 'Candidate B'], size=30)


import numpy as np
import pandas as pd
from scipy.stats import chi2_contingency

# Set the random seed for reproducibility
np.random.seed(0)

# Generate age groups and voter preferences
age_groups = np.random.choice(['18-30', '31-50', '51+'], size=500)
voter_preferences = np.random.choice(['Candidate A', 'Candidate B'], size=500)

# Create a DataFrame for better visualization
data = pd.DataFrame({'Age Group': age_groups, 'Voter Preference': voter_preferences})

# Create a contingency table
contingency_table = pd.crosstab(data['Age Group'], data['Voter Preference'])

# Perform the Chi-Square test
chi2, p, dof, expected = chi2_contingency(contingency_table)

# Print the results
print("Chi-Square Statistic:", chi2)
print("p-value:", p)
print("Degrees of Freedom:", dof)
print("Expected Frequencies Table:")
print(expected)

# Interpretation
alpha = 0.05
if p < alpha:
    print("There is a significant association between age groups and voter preferences (reject H0).")
else:
    print("There is no significant association between age groups and voter preferences (fail to reject H0).")
```

```
Chi-Square Statistic: 0.8779923945254768
p-value: 0.6446832311860852
Degrees of Freedom: 2
Expected Frequencies Table:
[[96.824 85.176]
 [89.908 79.092]
 [79.268 69.732]]
There is no significant association between age groups and voter preferences (fail to reject H0).
```

In [21]:
```
#Q.22) A company conducted a customer satisfaction survey to determine if there is a significant relationship
#between product satisfaction levels (Satisfied, Neutral, Dissatisfied) and the region where customers are
#located (East, West, North, South). The survey data is summarized in a contingency table. Conduct a ChiSquare
#customer regions.



#Sample data: Product satisfaction levels (rows) vs. Customer regions (columns)

data = np.array([[50, 30, 40, 20], [30, 40, 30, 50], [20, 30, 40, 30]])


import numpy as np
import pandas as pd
from scipy.stats import chi2_contingency

import numpy as np
from scipy.stats import chi2_contingency

# Sample data: Product satisfaction levels (rows) vs. Customer regions (columns)
data = np.array([[50, 30, 40, 20], [30, 40, 30, 50], [20, 30, 40, 30]])

# Perform the Chi-Square test
chi2, p, dof, expected = chi2_contingency(data)

# Print the results
print("Chi-Square Statistic:", chi2)
print("p-value:", p)
print("Degrees of Freedom:", dof)
print("Expected Frequencies Table:")
print(expected)

# Interpretation
alpha = 0.05
if p < alpha:
    print("There is a significant relationship between product satisfaction levels and customer regions (reject
else:
    print("There is no significant relationship between product satisfaction levels and customer regions (fail
```

```
Chi-Square Statistic: 27.777056277056275
p-value: 0.00010349448486004387
Degrees of Freedom: 6
Expected Frequencies Table:
[[34.14634146 34.14634146 37.56097561 34.14634146]
 [36.58536585 36.58536585 40.24390244 36.58536585]
 [29.26829268 29.26829268 32.19512195 29.26829268]]
There is a significant relationship between product satisfaction levels and customer regions (reject H0).
```

In [23]:
```
#Q.23) .A company implemented an employee training program to improve job performance (Effective, Neutral,
#Ineffective).
#After the training, they collected data from a sample of employees and classified them based
#on their job performance before and after the training. Perform a Chi-Square test to determine if there is a
#significant difference between job performance levels before and after the training.

# Sample data: Job performance levels before (rows) and after (columns) training

data = np.array([[50, 30, 20], [30, 40, 30], [20, 30, 40]])

import numpy as np
from scipy.stats import chi2_contingency

# Sample data: Job performance levels before (rows) vs. after (columns) training
data = np.array([[50, 30, 20], [30, 40, 30], [20, 30, 40]])

# Perform the Chi-Square test
chi2, p, dof, expected = chi2_contingency(data)

# Print the results
print("Chi-Square Statistic:", chi2)
print("p-value:", p)
print("Degrees of Freedom:", dof)
print("Expected Frequencies Table:")
print(expected)

# Interpretation
alpha = 0.05
if p < alpha:
    print("There is a significant difference between job performance levels before and after the training (reje
else:
    print("There is no significant difference between job performance levels before and after the training (fai
```

```
Chi-Square Statistic: 22.161728395061726
p-value: 0.00018609719479882557
Degrees of Freedom: 4
Expected Frequencies Table:
[[34.48275862 34.48275862 31.03448276]
 [34.48275862 34.48275862 31.03448276]
 [31.03448276 31.03448276 27.93103448]]
There is a significant difference between job performance levels before and after the training (reject H0).
```

In [26]:

```python
#Q. 24. A company produces three different versions of a product: Standard, Premium, and Deluxe. The
#company wants to determine if there is a significant difference in customer satisfaction scores among the
#three product versions. They conducted a survey and collected customer satisfaction scores for each
#version from a random sample of customers. Perform an ANOVA test to determine if there is a significant
#difference in customer satisfaction scores.

  # Sample data: Customer satisfaction scores for each product version

standard_scores = [80, 85, 90, 78, 88, 82, 92, 78, 85, 87]

premium_scores = [90, 92, 88, 92, 95, 91, 96, 93, 89, 93]

deluxe_scores = [95, 98, 92, 97, 96, 94, 98, 97, 92, 99]


import scipy.stats as stats

# Customer satisfaction scores for each product version
standard_scores = [80, 85, 90, 78, 88, 82, 92, 78, 85, 87]
premium_scores = [90, 92, 88, 92, 95, 91, 96, 93, 89, 93]
deluxe_scores = [95, 98, 92, 97, 96, 94, 98, 97, 92, 99]

# Perform one-way ANOVA
f_statistic, p_value = stats.f_oneway(standard_scores, premium_scores, deluxe_scores)

# Print the results
print("F-Statistic:", f_statistic)
print("p-value:", p_value)

# Interpretation
alpha = 0.05
if p_value < alpha:
    print("There is a significant difference in customer satisfaction scores among the three product versions (
else:
    print("There is no significant difference in customer satisfaction scores among the three product versions
```

```
F-Statistic: 27.03556231003039
p-value: 3.5786328857349003e-07
There is a significant difference in customer satisfaction scores among the three product versions (reject H0).
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js