Q= Generate a list of 100 integers containing values between 90 to R30 and store it in the variable `int_list`. After generating the list, find the following:

(i) Write a Python function to calculate the mean of a given list of numbers.

Create a function to find the median of a list of numbers.

(ii) Develop a program to compute the mode of a list of integers.

(iii) Implement a function to calculate the weighted mean of a list of values and their corresponding weights.

(iv) Write a Python function to find the geometric mean of a list of positive numbers.

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

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

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

```python
import random
from collections import Counter

# Generate a list of 100 integers containing values between 90 to 130
int_list = [random.randint(90, 130) for i in range(100)]

# Function to compute the mode
def compute_mode(numbers):
    count = Counter(numbers)
    max_frequency = max(count.values())
    modes = [number for number, frequency in count.items() if 
frequency == max_frequency]
    return modes

# Function to calculate the mean
def calculate_mean(numbers):
    return sum(numbers) / len(numbers)

# Function to calculate the median
def calculate_median(numbers):
    sorted_numbers = sorted(numbers)
    n = len(sorted_numbers)
    middle = n // 2

    if n % 2 == 0:
        median = (sorted_numbers[middle - 1] + sorted_numbers[middle]) 
/ 2
    else:
        median = sorted_numbers[middle]
    return median

# Compute the mode, mean, and median of the generated list
modes = compute_mode(int_list)
mean = calculate_mean(int_list)
median = calculate_median(int_list)

# Output the generated list and its mode(s), mean, and median
print(f"Generated list: {int_list}")
print(f"The mode(s) of the list is/are: {modes}")
print(f"The mean of the list is: {mean}")
print(f"The median of the list is: {median}")
```

```
Generated list: [120, 118, 108, 90, 120, 95, 98, 101, 106, 94, 92,
114, 118, 111, 92, 106, 115, 97, 90, 101, 116, 118, 121, 90, 113, 110,
124, 106, 106, 106, 121, 91, 120, 108, 103, 128, 122, 107, 111, 118,
90, 93, 101, 120, 105, 97, 108, 118, 126, 124, 122, 99, 126, 94, 124,
123, 118, 103, 109, 102, 102, 117, 92, 104, 94, 106, 115, 95, 116,
129, 108, 95, 91, 111, 130, 125, 103, 128, 103, 102, 95, 122, 118, 93,
125, 120, 90, 97, 124, 105, 128, 115, 129, 101, 95, 108, 96, 102, 112,
117]
The mode(s) of the list is/are: [118]
The mean of the list is: 108.85
The median of the list is: 108.0
```

```
def calculate_weighted_mean(values, weights):
    if len(values) != len(weights):
        raise ValueError("The length of values and weights must be the
same.")

    total_weighted_value = sum(value * weight for value, weight in
zip(values, weights))
    total_weight = sum(weights)

    return total_weighted_value / total_weight

# Example usage
values = int_list
weights = int_list
weighted_mean = calculate_weighted_mean(values, weights)
print(f"The weighted mean is: {weighted_mean}")
```

```
The weighted mean is: 110.1266881028939
```

```
def calculate_harmonic_mean(numbers):
    if len(numbers) == 0:
        raise ValueError("The list of numbers cannot be empty.")

    reciprocal_sum = sum(1 / number for number in numbers if number !=
0)

    if reciprocal_sum == 0:
        raise ValueError("The list contains zero which would cause a 
division by zero.")

    harmonic_mean = len(numbers) / reciprocal_sum
    return harmonic_mean

# Example usage
numbers = int_list
harmonic_mean = calculate_harmonic_mean(numbers)
print(f"The harmonic mean is: {harmonic_mean}")
```

```
The harmonic mean is: 107.56453198693909
```

```
def calculate_midrange(numbers):
    if len(numbers) == 0:
        raise ValueError("The list of numbers cannot be empty.")

    min_value = min(numbers)
    max_value = max(numbers)
    midrange = (min_value + max_value) / 2

    return midrange


numbers = int_list
midrange = calculate_midrange(numbers)
print(f"The midrange is: {midrange}")
```

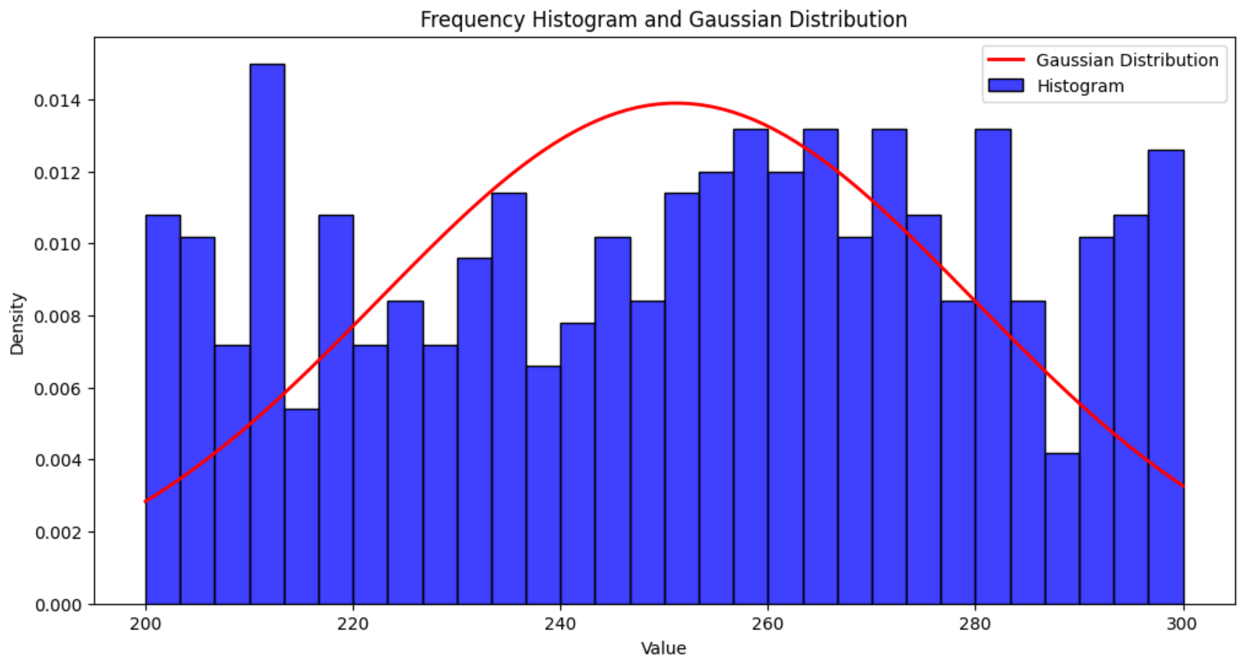
```
The midrange is: 110.0
```

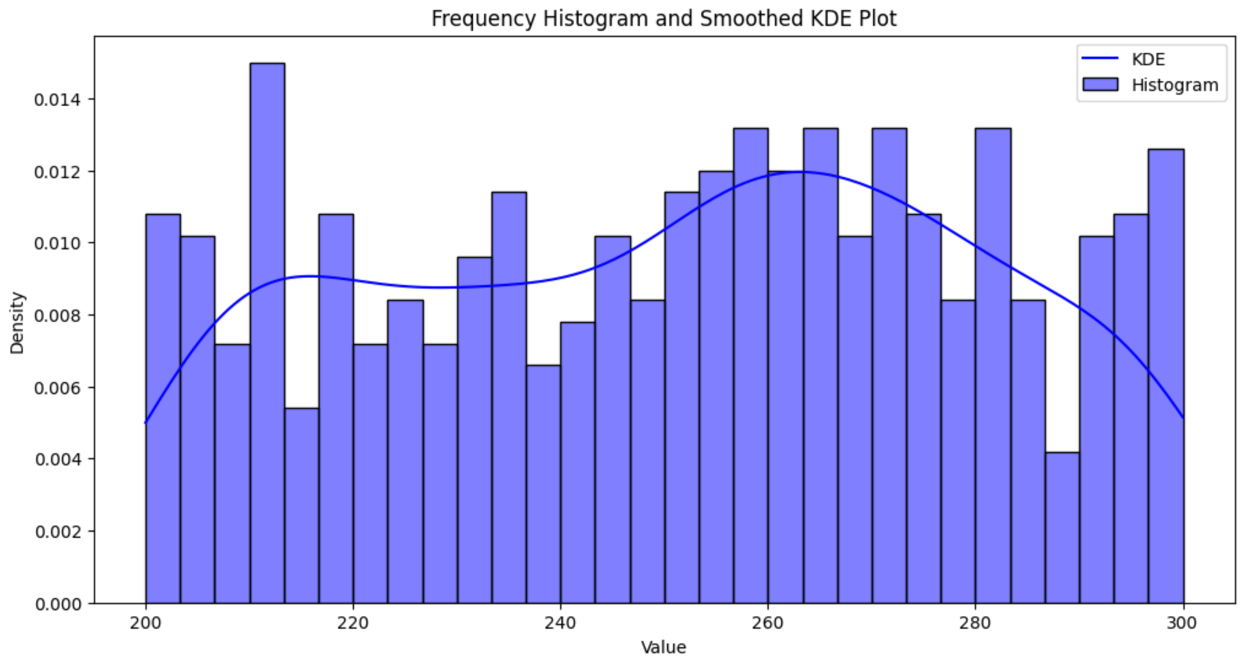
```
def calculate_trimmed_mean(numbers, trim_percentage):
    if not 0 <= trim_percentage < 50:
        raise ValueError("Trim percentage must be between 0 and 50.")

    n = len(numbers)
    if n == 0:
        raise ValueError("The list of numbers cannot be empty.")

    # Sort the numbers
    sorted_numbers = sorted(numbers)

    # Calculate the number of elements to trim from each end
    trim_count = int(n * (trim_percentage / 100))

    # Trim the numbers
    trimmed_numbers = sorted_numbers[trim_count:n - trim_count]
    # Calculate the mean of the trimmed list
    trimmed_mean = sum(trimmed_numbers) / len(trimmed_numbers)

    return trimmed_mean

# Example usage
numbers = int_list
trim_percentage = 10  # Exclude 10% of the smallest and largest values
trimmed_mean = calculate_trimmed_mean(numbers, trim_percentage)
print(f"The trimmed mean is: {trimmed_mean}")

The trimmed mean is: 108.7875
```

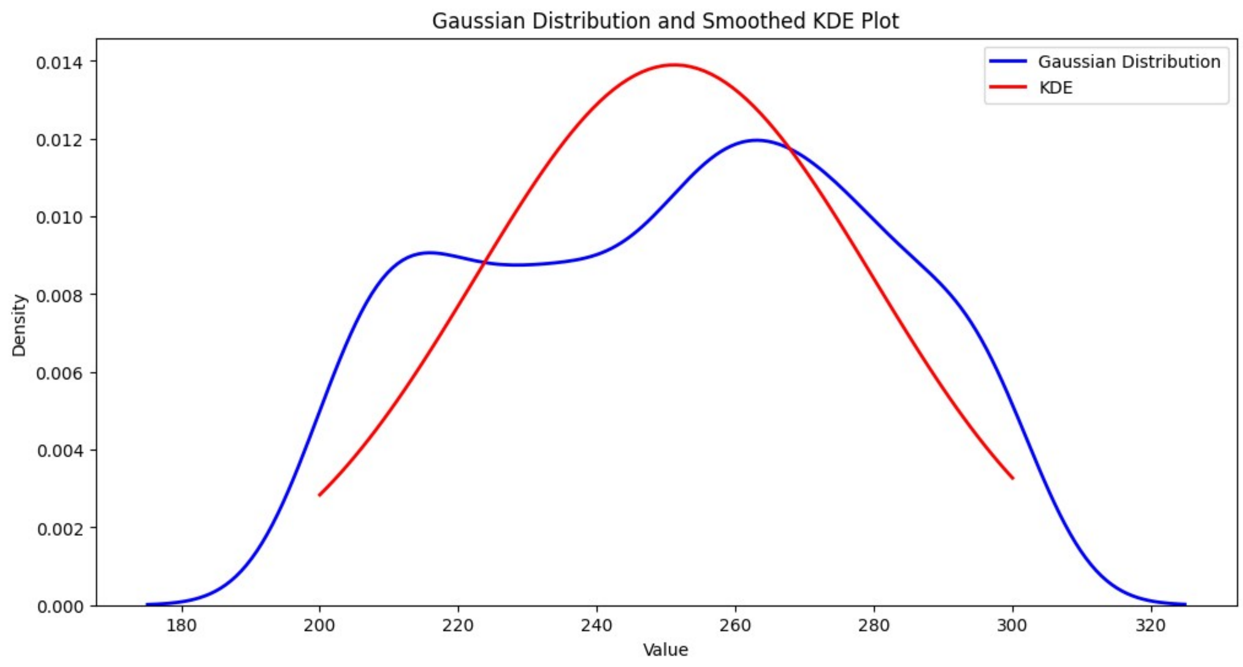
1. Generate a list o> 500 integers containing values between 200 to 300 and store it in the variable `int_list2`. A>ter generating the list, Find the Following:

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

# Generate a list of 500 integers containing values between 200 to 300
int_list2 = [random.randint(200, 300) for _ in range(500)]

# Frequency Histogram and Gaussian Distribution
plt.figure(figsize=(12, 6))
sns.histplot(int_list2, bins=30, kde=False, color='blue',
stat='density')
mean = np.mean(int_list2)
std_dev = np.std(int_list2)
x = np.linspace(min(int_list2), max(int_list2), 100)
plt.plot(x, norm.pdf(x, mean, std_dev), color='red', linewidth=2)
plt.title('Frequency Histogram and Gaussian Distribution')
plt.xlabel('Value')
plt.ylabel('Density')
plt.legend(['Gaussian Distribution', 'Histogram'])
plt.show()

# Frequency Histogram and Smoothed KDE Plot
plt.figure(figsize=(12, 6))
sns.histplot(int_list2, bins=30, kde=True, color='blue',
stat='density')
plt.title('Frequency Histogram and Smoothed KDE Plot')
plt.xlabel('Value')
plt.ylabel('Density')
plt.legend(['KDE', 'Histogram'])
plt.show()

# Gaussian Distribution and Smoothed KDE Plot
plt.figure(figsize=(12, 6))
```

```
sns.kdeplot(int_list2, color='blue', linewidth=2)
plt.plot(x, norm.pdf(x, mean, std_dev), color='red', linewidth=2)
plt.title('Gaussian Distribution and Smoothed KDE Plot')
plt.xlabel('Value')
plt.ylabel('Density')
plt.legend(['Gaussian Distribution', 'KDE'])
plt.show()
```







```python
def calculate_range(numbers):
    if len(numbers) == 0:
        raise ValueError("The list of numbers cannot be empty.")

    min_value = min(numbers)
    max_value = max(numbers)
    range_value = max_value - min_value

    return range_value

# Example usage
numbers = int_list2
range_value = calculate_range(numbers)
print(f"The range of the list is: {range_value}")
```

```
The range of the list is: 100
```

```python
#Create a program to find the variance and standard deviation of a
list of numbers.
import math

def calculate_mean(numbers):
    return sum(numbers) / len(numbers)

def calculate_variance(numbers):
    mean = calculate_mean(numbers)
    variance = sum((x - mean) ** 2 for x in numbers) / len(numbers)
    return variance

def calculate_standard_deviation(numbers):
    variance = calculate_variance(numbers)
    standard_deviation = math.sqrt(variance)
    return standard_deviation

# Example usage
numbers = int_list2
variance = calculate_variance(numbers)
standard_deviation = calculate_standard_deviation(numbers)
print(f"The variance of the list is: {variance}")
print(f"The standard deviation of the list is: {standard_deviation}")
```

```
The variance of the list is: 823.9584159999999
The standard deviation of the list is: 28.704675856034324
```

```
# Implement a function to compute the interquartile range (IQR) of a
list of values.
def calculate_iqr(numbers):
    if len(numbers) == 0:
        raise ValueError("The list of numbers cannot be empty.")

    # Sort the numbers
    sorted_numbers = sorted(numbers)
    n = len(sorted_numbers)

    # Calculate Q1 (first quartile)
    q1_index = n // 4
    if n % 4 == 0:
        q1 = (sorted_numbers[q1_index - 1] + sorted_numbers[q1_index])
/ 2
    else:
        q1 = sorted_numbers[q1_index]

    # Calculate Q3 (third quartile)
    q3_index = (3 * n) // 4
    if (3 * n) % 4 == 0:
        q3 = (sorted_numbers[q3_index - 1] + sorted_numbers[q3_index])
/ 2
    else:
        q3 = sorted_numbers[q3_index]

    # Calculate IQR
    iqr = q3 - q1

    return iqr

# Example usage
numbers = int_list2
iqr = calculate_iqr(numbers)
print(f"The interquartile range (IQR) is: {iqr}")
```

```
The interquartile range (IQR) is: 47.5
```

```
#Write a Python function to find the mean absolute deviation (MAD) of
a list of numbers.
def calculate_mean(numbers):
    if len(numbers) == 0:
        raise ValueError("The list of numbers cannot be empty.")
    return sum(numbers) / len(numbers)

def calculate_mad(numbers):
    if len(numbers) == 0:
        raise ValueError("The list of numbers cannot be empty.")

    mean = calculate_mean(numbers)
    mad = sum(abs(x - mean) for x in numbers) / len(numbers)

    return mad

# Example usage
numbers = int_list2
mad = calculate_mad(numbers)
print(f"The Mean Absolute Deviation (MAD) is: {mad:.2f}")
```

```
The Mean Absolute Deviation (MAD) is: 24.62
```

```
#Create a program to calculate the quartile deviation of a list of
values.
def calculate_quartiles(numbers):
    if len(numbers) == 0:
        raise ValueError("The list of numbers cannot be empty.")

    sorted_numbers = sorted(numbers)
    n = len(sorted_numbers)

    q1_index = n // 4
    q3_index = (3 * n) // 4

    if n % 4 == 0:
        q1 = (sorted_numbers[q1_index - 1] + sorted_numbers[q1_index])
/ 2
        q3 = (sorted_numbers[q3_index - 1] + sorted_numbers[q3_index])
/ 2
    else:
        q1 = sorted_numbers[q1_index]
        q3 = sorted_numbers[q3_index]

    return q1, q3

def calculate_quartile_deviation(numbers):
    q1, q3 = calculate_quartiles(numbers)
    quartile_deviation = (q3 - q1) / 2
    return quartile_deviation

numbers = int_list2
quartile_deviation = calculate_quartile_deviation(numbers)
print(f"The Quartile Deviation (QD) is: {quartile_deviation:.2f}")
```

```
The Quartile Deviation (QD) is: 23.75
```

```
# Implement a function to find the range-based coefficient of
dispersion for a dataset.
def calculate_mean(numbers):
    if len(numbers) == 0:
        raise ValueError("The list of numbers cannot be empty.")
    return sum(numbers) / len(numbers)

def calculate_range(numbers):
    if len(numbers) == 0:
        raise ValueError("The list of numbers cannot be empty.")
    return max(numbers) - min(numbers)

def calculate_coefficient_of_dispersion(numbers):
    mean = calculate_mean(numbers)
    if mean == 0:
        raise ValueError("The mean of the numbers cannot be zero for
calculating coefficient of dispersion.")
    range_value = calculate_range(numbers)
    coefficient_of_dispersion = range_value / mean
    return coefficient_of_dispersion

# Example usage
numbers = int_list2
coefficient_of_dispersion =
calculate_coefficient_of_dispersion(numbers)
print(f"The range-based coefficient of dispersion is:
{coefficient_of_dispersion:.2f}")
```

```
The range-based coefficient of dispersion is: 1.33
```

```
"""Write a Python class representing a discrete random variable with
methods to calculate its expected
value and variance."""

class Discrete_random_variable:
    def __init__(self,outcomes,probabilities):
        if len(outcomes) != len(probabilities):
            raise ValueError("Number of outcomes and probabilities
must be equal.")
        self.outcomes = outcomes
        self.probabilities = probabilities
    def expected_value(self):
        return sum(outcome * probability for outcome, probability in 
zip(self.outcomes, self.probabilities))

    def variance(self):
        mean = self.expected_value()
        return sum(probability * (outcome - mean) ** 2 for outcome, 
probability in zip(self.outcomes, self.probabilities))
outcomes = [1, 2, 3, 4, 5]
probabilities = [0.1, 0.2, 0.3, 0.2, 0.2]

# Create a DiscreteRandomVariable object
rv = Discrete_random_variable(outcomes, probabilities)

# Calculate and print the expected value and variance
print("Expected Value:", rv.expected_value())
print("Variance:", rv.variance())
```

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

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

def generate_samples(distribution, params, sample_size):
    if distribution == 'binomial':
        n, p = params
        samples = np.random.binomial(n, p, sample_size)
    elif distribution == 'poisson':
        lam = params
        samples = np.random.poisson(lam, sample_size)
    else:
        raise ValueError("Unsupported distribution. Supported 
distributions are 'binomial' and 'poisson'.")

    return samples

def calculate_mean_and_variance(samples):
    mean = np.mean(samples)
    variance = np.var(samples)
    return mean, variance

# Example usage
sample_size = 1000

# Generate samples from binomial distribution
binomial_samples = generate_samples('binomial', (10, 0.5), 
sample_size)
binomial_mean, binomial_variance =
calculate_mean_and_variance(binomial_samples)
print("Binomial Distribution:")
print("Mean:", binomial_mean)
print("Variance:", binomial_variance)

# Generate samples from Poisson distribution
poisson_samples = generate_samples('poisson', 5, sample_size)
poisson_mean, poisson_variance =
calculate_mean_and_variance(poisson_samples)
print("\nPoisson Distribution:")
print("Mean:", poisson_mean)
print("Variance:", poisson_variance)
```

```
Binomial Distribution:
Mean: 5.012
Variance: 2.6458559999999998

Poisson Distribution:
Mean: 4.938
Variance: 4.7821560000000005
```

```
"""Write a Python script to generate random numbers from a Gaussian
(normal) distribution and compute
the mean, variance, and standard deviation o6 the samples."""

import numpy as np

# Parameters
sample_size = 1000
mean = 0
std_dev = 1

# Generate random samples from a Gaussian distribution
samples = np.random.normal(mean, std_dev, sample_size)

# Calculate mean, variance, and standard deviation
sample_mean = np.mean(samples)
sample_variance = np.var(samples)
sample_std_dev = np.std(samples)

# Print results
print("Generated Samples:")
print(samples)
print("\nMean:", sample_mean)
print("Variance:", sample_variance)
print("Standard Deviation:", sample_std_dev)
```

```
Generated Samples:
[ 1.38522588e+00 -3.73625628e-01  1.33150172e+00 -1.42658335e-03
```

```
-8.90058366e-01  1.64964810e+00 -5.11551567e-01 -1.26086905e+00
-9.60903266e-01  1.93062892e+00 -5.11502040e-01  1.43166910e-01
 4.33989061e-01 -1.49009105e+00 -2.91088672e-01  1.93887904e-02
-3.68295654e-01  2.99899561e-01  1.39161064e+00  6.70724884e-01
-1.06970441e+00 -9.90798434e-01  2.39957893e-01  1.09080182e+00
-1.63723459e+00  1.55155839e+00 -9.19354999e-02 -1.29907752e+00
 7.65093809e-01  1.92127412e-01 -6.74569977e-01 -2.70183846e-02
-1.93062804e+00 -6.44114846e-01  2.40149690e-01  5.60184194e-02
-1.45719325e+00  4.99903402e-01 -1.26690047e-01 -2.22459065e+00
 6.23237594e-01  1.80235266e+00 -2.02343478e+00 -1.35964056e+00
-6.08968294e-01  8.49855033e-01 -5.05166370e-01  1.14290425e+00
-4.29399321e-01 -1.05048099e+00  2.23714606e+00  1.15002662e+00
-4.08577764e-01  1.64661727e+00 -1.29106930e-01 -2.90038349e+00
-9.07962450e-01 -4.35144910e-01 -3.59957245e-01  3.41473253e-01
-3.91272813e-01 -2.56850933e-01 -7.64708772e-01 -1.42308549e-01
-1.18171541e+00  2.07401880e+00 -2.31887756e+00 -4.94875009e-01
 1.06788938e+00 -6.77200369e-01  8.72413604e-01  5.46876678e-01
 1.69811851e-01  6.60764767e-01 -1.91264870e-02 -4.00284275e-02
-1.59775789e+00  2.10340618e-01  1.08491772e-01 -3.80107172e-01
-4.42013832e-01 -1.95443908e+00 -7.34319552e-01  7.20691135e-02
 2.03263424e+00  8.12306017e-02  1.82140275e+00 -3.26994792e-01
-4.24179835e-01 -1.38212560e-01 -8.53770646e-01 -5.58919537e-01
 1.21125499e+00 -3.48628155e-01 -6.78656856e-01  5.28260161e-01
-9.82510188e-02  4.67517555e-02  2.53574476e-01 -1.82615102e+00
 1.95292364e-01 -3.08237377e-01 -1.65529825e+00  2.72553983e+00
 9.83986724e-01  5.40791585e-01  1.22608283e+00  9.14852258e-01
 1.97673552e+00 -3.35744529e-01  6.49393378e-01  1.85782000e-01
-1.39576713e+00  4.24855292e-01  1.25216610e+00 -1.89587906e-02
-8.52026887e-01 -9.04490494e-01 -7.22411129e-01 -1.13371055e+00
 3.21036253e-01  1.80176288e-01  1.56754009e+00 -6.77279142e-01
-2.84244576e-01 -1.12325753e+00  2.47841349e-01  2.35957980e+00
 3.09307549e-01  5.30735295e-01 -4.14459340e-01  9.88730102e-01
 8.79363108e-01 -1.95379527e+00 -3.27574812e-01  2.77365109e-01
 2.51322203e-01  1.24523542e+00  7.63820527e-02 -9.20518189e-01
-1.86092760e-01 -7.08418671e-01  2.12862201e-01  3.69277613e-01
 1.26664601e-01 -9.46625943e-01 -6.84724922e-01  2.30139088e+00
-2.92743917e-01 -1.87460573e-01 -2.80769069e-01  2.09011957e+00
 1.48645658e+00 -6.69828279e-01  1.49160300e+00  1.01594587e+00
-7.99317922e-01  6.54863685e-02  1.96663589e+00  5.55653787e-01
 3.16738274e-01  7.35360880e-01  1.03198323e+00  2.52558381e-01
 1.14304160e+00  1.09348221e+00 -5.73963978e-01  8.63684380e-01
-5.35941930e-01 -2.18157323e-02 -9.98843118e-01  5.31677901e-01
-9.89357252e-01  7.02250770e-01  5.78794426e-01  5.93747357e-01
 6.37160673e-02 -1.09073605e+00  1.08970876e-02 -7.38976583e-01
-8.94652239e-01 -5.40842931e-01 -6.12317511e-01  1.53195612e-01
 2.33555887e-01 -2.12206595e-01 -7.19657840e-01  3.97687475e-02
-9.10740539e-01  2.54009928e-01 -6.05924643e-01  2.88024327e-01
 2.46406669e-01  1.05831268e+00 -2.49324922e-01 -2.46486506e-01
-1.29259451e-01  6.43508862e-02  4.07870641e-01 -5.53778264e-02
```

```
  7.45672543e-02  1.49904658e+00 -1.42849114e+00 -1.32326402e+00
 -2.51047670e-01  2.23599051e+00 -1.89408172e+00 -5.22693754e-01
  1.44290611e+00 -7.13003720e-01  1.94777825e+00  7.88992017e-01
  1.55856028e+00  9.19845828e-01 -1.48691059e+00 -1.07297465e+00
  1.42377728e+00  1.88915138e+00  1.53528206e-02 -1.17989359e+00
  1.40568892e+00  5.61145784e-01  4.66014311e-01 -4.49886216e-04
 -5.95203783e-01  7.35986228e-01  1.86606531e-01 -2.38543622e-01
 -2.46668156e-01 -3.38594482e-01 -2.47020146e-01 -1.03897448e+00
 -1.36090062e+00 -7.21646218e-01 -1.23473673e+00 -3.41343166e-01
  6.01236142e-01 -9.85319549e-01 -3.78325773e-01 -2.80991041e-01
  1.30933145e+00 -1.52268499e+00  2.09615814e-02 -2.30243207e+00
 -1.88049565e-01  5.44265230e-01 -2.78683822e-01 -5.53400195e-01
 -2.53751338e-01  4.94426618e-01  6.18753572e-02  7.60530858e-02
 -1.45606068e+00  1.84065033e+00  2.69658651e-01 -1.07194682e-01
  1.32294990e+00 -1.07804160e+00 -1.06447570e-01  1.06366431e+00
 -4.91866899e-01  1.09377046e+00  2.95770356e+00 -7.95319830e-01
  2.38135147e-01  8.74003024e-01  7.33394109e-01  6.05127893e-01
 -1.93854465e-01 -4.35621732e-01  1.18420875e-01 -1.68681603e+00
 -5.85134657e-01  1.08940437e+00  7.03291067e-01  1.83124155e-01
 -1.15011538e-01  3.36904490e-01 -2.39745628e-01  1.22808594e+00
 -1.07621969e+00 -4.82165371e-01  6.95305221e-01 -9.20903224e-01
  1.12438934e-01 -1.20809032e+00 -5.25116172e-01  3.91437897e-01
 -8.15387271e-02  4.02520519e-01  1.63768027e+00  6.24608276e-01
  9.68690568e-01  6.66077771e-01  5.30801982e-01  1.16924984e+00
  3.49583309e-02  4.52614785e-01  2.81839201e-01  5.63023029e-01
  8.03512565e-01  5.92308855e-01 -1.72552121e+00 -1.31195785e+00
  8.01617445e-01  4.22845016e-01 -1.81845131e+00 -1.63139570e-01
 -1.81473155e+00 -1.26693646e+00  1.07109327e-02  1.46361785e-01
 -3.39911257e-01  1.08979295e+00  1.09218460e+00  1.49491996e+00
 -5.58074944e-01  3.60520581e-01 -3.90486556e-01 -6.37392155e-01
 -1.03328704e+00 -3.08010324e-01 -3.48712225e-01 -5.79688451e-01
 -5.29670472e-01 -3.17899969e-01  3.57886188e-02 -5.45517566e-02
 -2.92422198e-01 -1.65587426e+00  2.58966250e-01  8.56008227e-01
 -1.46808219e+00  3.65120681e-01  2.07511520e-01  6.92776842e-01
 -5.91967858e-01 -1.25459058e+00 -6.27281667e-01  4.43459676e-02
  2.43154514e-01 -5.91722195e-01 -1.58313234e+00  1.28405111e+00
  5.76225626e-01 -8.84046442e-01  1.23255011e-01 -5.66692651e-02
  1.28644148e+00  1.47416993e+00  1.97908433e+00  2.27399898e-01
  5.34991950e-01  5.73820809e-01 -9.70448437e-01  3.02010239e-01
  1.35630072e+00 -1.07166601e+00  7.91255340e-01  9.39799476e-01
  1.77098353e-01 -6.84225910e-01 -2.16178079e+00 -8.18378407e-01
  1.95151623e+00 -6.63805657e-01  1.34832629e+00 -7.41571443e-01
  2.14190085e+00 -4.09773388e-01 -1.10296080e+00 -4.02110916e-01
  3.83732989e-01 -1.43294937e-01 -1.04160451e+00  6.74637679e-01
  1.91344456e+00  8.06335677e-01 -7.63169509e-01  1.78025121e+00
 -2.57378098e-01  2.45692976e-01 -5.43482516e-02  2.41501837e-01
 -9.68245792e-01  3.19568416e-01 -9.24388936e-01 -2.24889930e+00
  4.70341983e-01  2.80415634e-01  1.36691113e+00 -6.29314440e-01
  3.27166912e-01 -1.07703240e+00 -1.59636962e+00 -6.27107986e-01
  7.68373118e-01  4.91056610e-01 -9.42865001e-01 -3.78522995e-01
 -2.19491566e-01 -1.01348975e+00  3.98912284e-01  1.93257507e+00
 -5.98567216e-01 -1.25786027e+00  8.62627776e-01 -1.62501429e-01
 -1.16639508e+00  2.01465945e-02 -3.03897050e-01  2.20364102e+00
  5.47362328e-02 -3.21015368e-01  3.90743485e-01  1.64251571e-01
  1.97822860e+00  1.04312984e+00  3.25818666e-02  5.63560271e-01
 -3.85329860e-01  4.04779266e-01  1.18044604e+00  7.31338701e-01
 -1.53697728e-01 -1.21776487e+00 -1.46334991e-01  2.22434625e-01
  6.51632921e-01  1.64381475e+00 -9.63630422e-02  6.15308068e-01
 -1.34186210e+00  8.75310736e-01  2.63192299e+00 -6.21790466e-01
 -6.52106661e-02  3.47958525e-02  4.75892064e-01 -1.14892845e+00
  9.47887800e-01 -1.72891200e-01 -3.27195241e-01 -8.41998453e-01
  5.92567912e-02 -1.07815837e-01  1.05464741e+00 -9.96473418e-01
 -4.40203640e-01  7.29271163e-02  1.14195078e+00  4.59987275e-02
 -8.42384226e-01  1.54619196e+00 -3.30838145e-01  1.19095545e-01
  1.34878992e+00 -4.76164257e-01 -1.22865768e-01  9.00291136e-01
  2.26326299e+00 -1.02056482e+00 -2.16726338e-01  1.73100287e-01
 -3.85271219e-01 -7.23058441e-01 -3.55045453e-01  1.16577491e+00
 -1.20978852e+00 -1.50484612e+00 -5.42736408e-01 -1.75885670e-01
  1.23734272e+00 -5.46041995e-01 -1.28833182e+00 -4.18439522e-01
  1.06071034e+00 -1.17836591e+00 -2.25702046e+00 -1.07083815e+00
  3.73515347e-01  3.73540778e-01 -1.00082372e-01  1.01826288e+00
  2.92282892e-01  1.44081888e-01  1.41846562e+00  1.01389828e+00
  2.35267025e-01 -6.09709852e-01 -7.04772561e-01  2.63184117e-01
 -5.92841219e-02  6.19109472e-01 -1.35776274e-01 -2.46062549e-01
  1.13386670e+00 -3.01829818e-01 -7.49898802e-01 -5.92998965e-01
  1.67790890e-01  2.15187681e-01 -5.83552632e-01 -9.72226791e-01
  9.19630194e-01 -8.62159103e-01  1.00173716e+00 -6.49802790e-01
  6.55881131e-01 -2.35874660e+00 -5.45574641e-01  1.21293007e+00
 -1.03481654e+00  9.40567042e-01  8.88395703e-01 -1.35510306e+00
 -1.44674457e-01  1.04074657e-01 -1.53205218e+00 -1.27323543e+00
  1.01817053e+00  2.08956633e-01  7.16347363e-02  1.16449097e+00
  5.71734691e-01  3.79855842e-01  1.39680572e+00 -9.69077280e-01
 -1.66008212e+00 -2.26963426e+00  9.45799411e-01 -7.77776863e-01
  1.87683666e-02  9.75721669e-01 -4.06836218e-01 -1.39008310e-02
 -5.90506578e-01  1.26265974e-01  1.00778197e+00 -5.58046409e-01
  4.76753461e-01  7.76597608e-01  3.61549124e-01  1.05729616e-01
  3.76824166e-01 -4.49995045e-01 -1.92525131e-01 -2.51022944e-01
 -1.07327246e+00  2.39407247e-01  1.62989724e-01  1.83943939e+00
  1.08036896e+00  3.71259231e-01  2.39283219e-01 -2.84259639e-01
 -1.16408772e+00 -7.63622533e-02 -2.20167174e-01  2.39642502e-01
 -6.84846613e-02 -2.06896156e-01  1.28121047e+00  1.32536145e-02
 -2.26160443e+00 -6.64849483e-01 -2.66616436e-01 -2.73139477e-01
  5.72912343e-01  1.29073574e+00  5.65690268e-01 -1.60182811e+00
  9.62477665e-01  7.45656933e-01  1.64184257e-01 -4.20315296e-01
  1.37282520e+00  9.49176939e-02 -1.16605320e+00  5.85369148e-01
 -2.75766369e-01  6.74811883e-01  1.10038203e+00  2.06376884e+00
  1.74621370e-01  7.66447110e-02 -5.42693156e-02 -4.39255758e-01
  6.19504179e-01 -5.63622925e-01 -1.64167196e+00 -1.37383703e+00
 -7.68125836e-01 -7.42095764e-01 -6.26992070e-02  2.78794787e-01
  5.27212866e-01  4.60184065e-01 -5.47330839e-01  1.52615013e-01
 -5.48415346e-01 -4.97946914e-01 -8.86449322e-01  4.86702706e-01
 -1.25128123e+00 -2.10010893e+00  1.89180725e+00  5.73036846e-01
 -3.47394324e-01 -2.30322698e+00 -1.17395719e+00 -2.05525106e+00
 -6.89357368e-01 -1.27330927e+00  1.46387178e+00  1.97990826e+00
 -3.48882779e-01 -2.07347183e-01 -5.02681478e-01  2.49311693e-01
  1.66337656e+00 -3.41155482e-02 -9.37353059e-01 -4.95593675e-01
 -1.61161966e+00 -1.00337273e-01 -3.38388034e-01 -6.48146103e-01
 -1.38766515e+00 -2.38272050e+00  6.04546893e-01  1.54513426e+00
 -1.11125542e+00  5.69509691e-01  1.32072545e+00 -7.71305056e-01
  3.10523155e-01  1.30561334e+00 -1.21496249e-01  1.03137108e+00
  2.53877464e-01 -1.19445767e+00 -1.99238559e-01 -1.12795433e+00
 -2.85426981e-01 -8.43591660e-01  7.20036912e-02  4.66833995e-01
 -1.35980422e+00  8.70704310e-01 -9.20939298e-01  6.88815039e-01
 -2.54000572e-01  3.09088979e-01  9.40706884e-01 -2.10726129e-01
  5.36540111e-01  5.48791778e-01 -7.12939643e-01 -9.76693528e-01
  7.90881587e-01 -1.20823573e+00 -6.35846268e-01 -1.54505646e+00
  1.30534740e+00 -7.24758186e-01 -1.07738883e+00 -2.50673361e-01
  8.37539209e-01  5.64056452e-01 -3.85739791e-01  6.31535889e-01
  5.15798979e-01 -7.48737592e-01  2.11324555e+00 -1.00068992e+00
  1.69940022e+00  4.60121416e-01 -1.19717313e+00  1.48345767e+00
  5.17192622e-01  1.40501535e-01  1.94242758e-01 -5.03160785e-01
 -2.04501253e-02  6.82519461e-01 -2.00200376e-02  1.36855747e-01
 -2.61867689e+00 -6.12636372e-01  4.67189368e-01 -2.82122957e-01
  2.78261022e-01  6.25396192e-01  1.51900549e-01 -1.92003095e-01
 -3.44709750e-01 -2.69427788e-02  4.56374488e-01 -1.80094609e+00
 -1.36106892e+00 -1.08831733e+00 -4.43833906e-01 -1.27068103e+00
  4.30850211e-01  9.87254795e-01  1.87492804e-01 -3.69071924e-01
  3.57587561e-01 -1.26167618e-01  3.39348116e-01  7.15937023e-01
  5.80249604e-01  2.07839935e+00  4.41466119e-01 -7.37114830e-01
  6.37232493e-01  1.62555483e+00  8.68112993e-01 -1.96173756e+00
  1.67878600e+00  1.74877858e+00  5.72532975e-01  1.71429161e+00
 -5.92862000e-03 -1.20243801e+00  1.03826522e+00 -1.41245509e+00
 -1.35777038e+00  6.58291896e-01  3.68658153e-01 -9.26101538e-01
 -1.22539359e+00  5.66299161e-01  1.93759089e+00 -1.41480324e-01
  1.18786487e-01  1.07788749e+00  6.96867769e-01 -6.51172433e-01
  2.38140915e-02  7.72217897e-01 -2.44112295e+00  6.84947714e-02
 -2.40434805e-01  7.60946028e-02  1.05556740e+00 -1.41041306e-01
  5.04288468e-01 -5.43404778e-01 -1.06792009e+00  1.20124470e+00
 -3.02680755e-01  1.44453341e-01 -7.66117074e-01  9.14103644e-01
  2.38422614e-01  3.24352717e-01  1.32954362e+00 -6.29871467e-02
  1.48005978e+00  3.87654919e-01 -1.13724215e+00  1.92602774e+00
  1.47725179e+00 -5.58654072e-01  9.98929783e-01  7.14603774e-01
  1.12153323e+00  6.42001886e-03 -5.60759437e-01 -3.72250544e-01
  1.03002173e+00 -9.80972015e-01  1.59267853e+00  1.70074909e+00
 -4.62820838e-01  2.63817534e-01  7.34071930e-01 -7.10412325e-01
  7.87382686e-01 -1.31686937e+00  1.08375841e+00  7.30332965e-01
 -2.90242201e-01 -1.45568644e+00 -3.08089685e-01  1.29317680e+00
```

```
  1.56870077e+00  1.23495789e+00  1.22350083e-01 -5.23775561e-01
  9.96760704e-01  2.08201212e+00  1.12389168e+00 -1.05001348e+00
 -1.58306659e+00 -7.90558230e-01  5.14222694e-01  2.86927447e-01
 -2.16287677e-01 -1.77371562e+00 -9.88949094e-01  7.14949050e-01
 -5.91252585e-01  5.99817487e-01  1.12098344e-01  4.92082100e-01
 -8.95045415e-02 -5.73821257e-01 -2.79982131e-02  4.22670922e-01
 -1.44564919e+00  1.12540819e+00  2.40030200e-01  1.21804538e-01
 -4.81496290e-01 -1.51799632e+00  1.52943104e+00 -1.00837389e+00
  1.78389211e-01 -9.31819466e-01 -3.46768748e-01 -9.47295561e-01
 -1.57613866e+00 -4.64722702e-01 -2.15352131e+00 -1.25792041e+00
 -4.15320365e-01  1.30534258e+00  8.48878210e-01  1.04143489e+00
  1.50783502e+00 -4.52823022e-01 -5.64895202e-01 -2.05476643e+00
  1.76854580e+00 -3.09404437e+00 -1.11301670e+00 -1.40086403e+00
  8.80899594e-01  8.12948217e-02 -8.28616539e-01  1.25861358e+00
  4.34949997e-01 -2.51085249e-01 -2.10053649e-01 -6.00222680e-01
 -8.30097423e-01 -1.29419294e+00  2.26445472e+00  6.75681319e-01
 -1.12177333e+00  1.52431110e+00  8.27365965e-01  2.01112859e-01
 -1.37621052e+00  8.83741753e-01 -3.32626787e-01  7.01420261e-02
  4.25445722e-01 -1.01320991e-01  1.93704785e+00  7.51853171e-01
  1.69796482e+00  4.46104655e-01  1.89591789e+00  8.67730764e-01
  2.81331636e-01 -8.58800148e-02 -4.60636680e-01 -2.33261728e-01
  4.53100286e-01  1.57631713e+00 -6.15731142e-01  7.64694719e-02
 -8.11331889e-01  1.61794738e-01  1.05933459e+00  7.69143968e-01
  1.14186899e+00  7.99089421e-01 -4.58179873e-02  1.05661035e-01
  6.36263038e-01  1.07202880e+00  1.06282291e+00  3.32179012e-01
 -1.03589575e-01  8.17307523e-01 -1.32024116e+00 -2.29765354e+00
  1.37010107e+00  2.32375490e-01 -1.69023664e-01  2.79401731e-01
 -1.28279393e+00  8.03846535e-01 -1.29037834e+00  6.70950172e-03
  1.01294683e+00 -7.76047815e-02  8.82111755e-01  6.45287755e-02
 -1.80861447e+00  1.95319924e+00 -2.01985554e+00  1.36344165e+00
 -5.21993658e-02 -4.62921701e-01 -4.60662591e-01  6.11904858e-01
  3.57624325e-01  9.40640078e-01  1.27283950e+00 -7.36394936e-01
 -2.18199281e+00  9.71177182e-01 -5.91514568e-01  5.02174810e-01
 -4.34978404e-01 -1.15289068e+00 -7.73594820e-01 -2.35015966e-01
 -2.06300620e-01 -5.96835991e-01  1.30208056e+00 -1.43310425e-01
  8.55331844e-01  3.31789429e-01 -6.11366222e-01 -6.47824521e-01
 -6.24623306e-01 -2.65432239e+00 -1.77249941e+00 -1.31648656e+00
 -1.00087195e+00 -2.97372590e-01 -6.46576826e-01 -9.57443051e-01
 -8.72077219e-02  5.26146108e-01  5.69014951e-01  9.78337152e-02
 -1.46451778e-01  6.73280332e-01  2.59740334e-03 -1.21123835e+00
 -5.85392748e-01 -1.71424159e+00 -3.16238714e-01  1.02184192e+00
 -2.98550791e-01  4.70425782e-01  1.45852144e+00  1.56557318e+00
 -5.70351201e-01 -5.89366421e-01  5.34495719e-01 -1.20295159e+00
 -1.99880333e+00  1.06716843e+00  6.66780797e-01 -4.94931738e-01
 -6.52416896e-01  1.30506531e+00 -5.62122018e-01  1.00473327e+00
 -8.92036908e-01  1.10734646e-01  3.19564208e-01 -3.05033279e-01
  2.68611949e-01  2.05400012e+00  5.77392957e-01 -8.25961628e-02
 -1.97542450e+00  5.39109635e-01  1.85890227e-01 -1.09793698e+00
 -6.41085328e-01 -3.63289233e-01 -1.84631144e+00 -4.13746989e-01
```

```
  1.34624888e+00 -6.83532236e-01 -3.77869312e-01 -5.16215049e-01
 -2.41838021e+00 -7.90187285e-01  1.62047624e-01  1.10154713e-02
 -3.46703458e-01 -1.03586736e+00 -4.56022768e-01 -9.30586831e-01
  1.18485263e+00  1.26081279e+00  1.15707243e+00  2.25870321e+00]

Mean: 0.007968223257593556
Variance: 0.967194252841536
Standard Deviation: 0.9834603463493259
```

Use seaborn libraries to load tips dataset. Find the following from the dataset for the columns total_bill and tip`:

```
import seaborn as sns
df=sns.load_dataset("tips")

df

     total_bill   tip     sex smoker   day    time  size
0         16.99  1.01  Female     No   Sun  Dinner     2
1         10.34  1.66    Male     No   Sun  Dinner     3
2         21.01  3.50    Male     No   Sun  Dinner     3
3         23.68  3.31    Male     No   Sun  Dinner     2
4         24.59  3.61  Female     No   Sun  Dinner     4
..          ...   ...     ...    ...   ...     ...   ...
239       29.03  5.92    Male     No   Sat  Dinner     3
240       27.18  2.00  Female    Yes   Sat  Dinner     2
241       22.67  2.00    Male    Yes   Sat  Dinner     2
242       17.82  1.75    Male     No   Sat  Dinner     2
243       18.78  3.00  Female     No  Thur  Dinner     2

[244 rows x 7 columns]

tdf=df['total_bill'].describe()

tdf

count    244.000000
mean      19.785943
std        8.902412
min        3.070000
25%       13.347500
50%       17.795000
75%       24.127500
max       50.810000
Name: total_bill, dtype: float64

tdm=df['tip'].describe()

tdm
```

```
count    244.000000
mean       2.998279
std        1.383638
min        1.000000
25%        2.000000
50%        2.900000
75%        3.562500
max       10.000000
Name: tip, dtype: float64
```

Use seaborn library to load tips dataset. Find the 6ollowing 6rom the dataset 6or the columns total_bill and tip`:

Write a Python function that calculates their skewness.

Create a program that determnes whether the columns exhibit positive skewness, negative skewness, or is approximately symmetric.

Write a function that calculates the covariance between two columns.

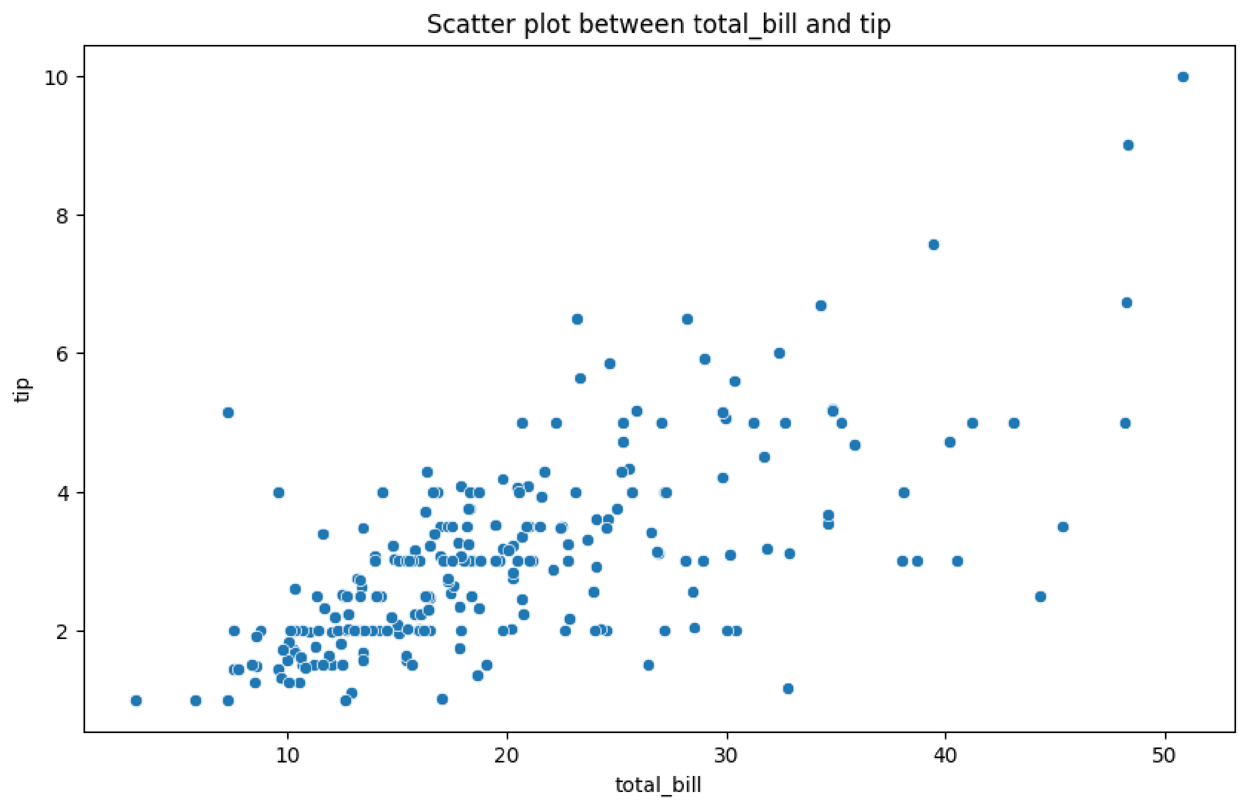
Implement a Python program that calculates the Pearson correlation coefficient between two columns.

Write a script to visualize the correlation between two specific columns in a Pandas DataFrame using scatter plots

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

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

# Function to calculate skewness
def calculate_skewness(column):
    return column.skew()

# Function to determine skewness type
def skewness_type(skewness):
    if skewness > 0:
        return 'Positive skewness'
    elif skewness < 0:
        return 'Negative skewness'
    else:
        return 'Approximately symmetric'

# Function to calculate covariance between two columns
def calculate_covariance(column1, column2):
    return column1.cov(column2)
```

```python
# Function to calculate Pearson correlation coefficient between two
columns
def calculate_pearson_correlation(column1, column2):
    return column1.corr(column2)

# Function to visualize the correlation between two columns using
scatter plot
def visualize_correlation(data, column1, column2):
    plt.figure(figsize=(10, 6))
    sns.scatterplot(x=column1, y=column2, data=data)
    plt.title(f'Scatter plot between {column1} and {column2}')
    plt.xlabel(column1)
    plt.ylabel(column2)
    plt.show()

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

# Determine skewness type for total_bill and tip
total_bill_skewness_type = skewness_type(total_bill_skewness)
tip_skewness_type = skewness_type(tip_skewness)

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

# Calculate Pearson correlation coefficient between total_bill and tip
pearson_correlation =
calculate_pearson_correlation(tips['total_bill'], tips['tip'])

# Print the results
print(f"Skewness of total_bill: {total_bill_skewness}
({total_bill_skewness_type})")
print(f"Skewness of tip: {tip_skewness} ({tip_skewness_type})")
print(f"Covariance between total_bill and tip: {covariance}")
print(f"Pearson correlation coefficient between total_bill and tip:
{pearson_correlation}")

# Visualize the correlation between total_bill and tip
visualize_correlation(tips, 'total_bill', 'tip')
```

```
Skewness of total_bill: 1.1332130376158205 (Positive skewness)
Skewness of tip: 1.4654510370979401 (Positive skewness)
Covariance between total_bill and tip: 8.323501629224854
Pearson correlation coefficient between total_bill and tip:
0.6757341092113645
```



Write a Python function to calculate the probabilites density function (PDF) of a continuous random variable for a given normal distribution.

```
import scipy.stats as stats

def calculate_pdf(mu, sigma, x):

    return stats.norm.pdf(x, loc=mu, scale=sigma)

mu = 0     # Mean
sigma = 1 # Standard deviation
x = 1      # Value at which to calculate the PDF

pdf_value = calculate_pdf(mu, sigma, x)
print(f"The PDF value at x={x} for a normal distribution with
mean={mu} and sigma={sigma} is {pdf_value}")

The PDF value at x=1 for a normal distribution with mean=0 and sigma=1
is 0.24197072451914337
```

Create a program to calculate the cumulative distribution 6unction (CDF) of exponential distribution.

```
import scipy.stats as stats
```

```python
def calculate_exponential_cdf(lambd, x):

    # The scale parameter for scipy's expon function is 1/lambda
    scale = 1 / lambd
    return stats.expon.cdf(x, scale=scale)


lambd = 1  # Rate parameter (lambda)
x = 5      # Value at which to calculate the CDF

cdf_value = calculate_exponential_cdf(lambd, x)
print(f"The CDF value at x={x} for an exponential distribution with 
lambda={lambd} is {cdf_value}")
```

```
The CDF value at x=5 for an exponential distribution with lambda=1 is 
0.9932620530009145
```

Write a Python function to calculate the probabilites mass function (PMF) of Poisson distribution.

```python
import scipy.stats as stats

def calculate_poisson_pmf(mu, k):

    return stats.poisson.pmf(k, mu)

mu = 3  # Rate parameter (lambda)
k = 8   # Value at which to calculate the PMF

pmf_value = calculate_poisson_pmf(mu, k)
print(f"The PMF value at k={k} for a Poisson distribution with 
lambda={mu} is {pmf_value}")
```

```
The PMF value at k=8 for a Poisson distribution with lambda=3 is 
0.008101511794681432
```

```
pip install statsmodels
```

```
Collecting statsmodelsNote: you may need to restart the kernel to use 
updated packages.

  Downloading statsmodels-0.14.2-cp312-cp312-win_amd64.whl.metadata 
(9.5 kB)
Requirement already satisfied: numpy>=1.22.3 in e:\dddowld\lib\site-
packages (from statsmodels) (1.26.4)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in e:\dddowld\lib\
site-packages (from statsmodels) (1.12.0)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in e:\dddowld\lib\
site-packages (from statsmodels) (2.2.1)
Collecting patsy>=0.5.6 (from statsmodels)
```

```
  Using cached patsy-0.5.6-py2.py3-none-any.whl.metadata (3.5 kB)
Requirement already satisfied: packaging>=21.3 in c:\users\dell\
appdata\roaming\python\python312\site-packages (from statsmodels)
(23.2)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\
dell\appdata\roaming\python\python312\site-packages (from pandas!
=2.1.0,>=1.4->statsmodels) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in e:\dddowld\lib\site-
packages (from pandas!=2.1.0,>=1.4->statsmodels) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in e:\dddowld\lib\site-
packages (from pandas!=2.1.0,>=1.4->statsmodels) (2024.1)
Requirement already satisfied: six in c:\users\dell\appdata\roaming\
python\python312\site-packages (from patsy>=0.5.6->statsmodels)
(1.16.0)
Downloading statsmodels-0.14.2-cp312-cp312-win_amd64.whl (9.8 MB)
   ---------------------------------------- 0.0/9.8 MB ? eta -:--:--
   ---------------------------------------- 0.0/9.8 MB 640.0 kB/s eta
0:00:16
   ---------------------------------------- 0.1/9.8 MB 650.2 kB/s eta
0:00:16
    --------------------------------------- 0.1/9.8 MB 1.0 MB/s eta
0:00:10
   - -------------------------------------- 0.2/9.8 MB 1.5 MB/s eta
0:00:07
   - -------------------------------------- 0.5/9.8 MB 2.3 MB/s eta
0:00:05
   --- ------------------------------------ 0.7/9.8 MB 2.9 MB/s eta
0:00:04
   --- ------------------------------------ 1.0/9.8 MB 3.4 MB/s eta
0:00:03
   ----- ---------------------------------- 1.3/9.8 MB 3.7 MB/s eta
0:00:03
   ----- ---------------------------------- 1.4/9.8 MB 3.8 MB/s eta
0:00:03
   ------ --------------------------------- 1.6/9.8 MB 3.8 MB/s eta
0:00:03
   ------- -------------------------------- 1.8/9.8 MB 3.8 MB/s eta
0:00:03
   ------- -------------------------------- 1.9/9.8 MB 3.8 MB/s eta
0:00:03
   ------- -------------------------------- 1.9/9.8 MB 3.8 MB/s eta
0:00:03
   ------- -------------------------------- 1.9/9.8 MB 3.8 MB/s eta
0:00:03
   -------- ------------------------------- 2.0/9.8 MB 3.0 MB/s eta
0:00:03
   --------- ------------------------------ 2.2/9.8 MB 3.1 MB/s eta
0:00:03
   ---------- ----------------------------- 2.9/9.8 MB 3.8 MB/s eta
```

```
0:00:02
   ------------ --------------------------- 3.1/9.8 MB 3.9 MB/s eta
0:00:02
   ------------- -------------------------- 3.3/9.8 MB 3.8 MB/s eta
0:00:02
   ------------- -------------------------- 3.3/9.8 MB 3.9 MB/s eta
0:00:02
   ------------- -------------------------- 3.3/9.8 MB 3.9 MB/s eta
0:00:02
   ------------- -------------------------- 3.3/9.8 MB 3.9 MB/s eta
0:00:02
   ------------- -------------------------- 3.4/9.8 MB 3.3 MB/s eta
0:00:02
   -------------- ------------------------- 3.6/9.8 MB 3.4 MB/s eta
0:00:02
   ----------------- ---------------------- 4.3/9.8 MB 3.9 MB/s eta
0:00:02
   ------------------ --------------------- 4.5/9.8 MB 3.8 MB/s eta
0:00:02
   ------------------ --------------------- 4.6/9.8 MB 3.9 MB/s eta
0:00:02
   ------------------- -------------------- 4.8/9.8 MB 3.9 MB/s eta
0:00:02
   -------------------- ------------------- 5.0/9.8 MB 3.9 MB/s eta
0:00:02
   --------------------- ------------------ 5.2/9.8 MB 3.8 MB/s eta
0:00:02
   --------------------- ------------------ 5.2/9.8 MB 3.9 MB/s eta
0:00:02
   --------------------- ------------------ 5.2/9.8 MB 3.9 MB/s eta
0:00:02
   --------------------- ------------------ 5.2/9.8 MB 3.9 MB/s eta
0:00:02
   ---------------------- ----------------- 5.4/9.8 MB 3.6 MB/s eta
0:00:02
   ----------------------- ---------------- 5.7/9.8 MB 3.7 MB/s eta
0:00:02
   ------------------------- -------------- 6.3/9.8 MB 3.9 MB/s eta
0:00:01
   ------------------------- -------------- 6.4/9.8 MB 3.9 MB/s eta
0:00:01
   ------------------------- -------------- 6.4/9.8 MB 3.9 MB/s eta
0:00:01
   ------------------------- -------------- 6.4/9.8 MB 3.9 MB/s eta
0:00:01
   -------------------------- ------------- 6.5/9.8 MB 3.6 MB/s eta
0:00:01
   --------------------------- ------------ 6.8/9.8 MB 3.7 MB/s eta
0:00:01
```

```
   ---------------------------- ----------- 7.0/9.8 MB 3.7 MB/s eta
0:00:01
   ----------------------------- ---------- 7.2/9.8 MB 3.7 MB/s eta
0:00:01
   ------------------------------ --------- 7.5/9.8 MB 3.8 MB/s eta
0:00:01
   ------------------------------ --------- 7.5/9.8 MB 3.8 MB/s eta
0:00:01
   ------------------------------ --------- 7.5/9.8 MB 3.8 MB/s eta
0:00:01
   ------------------------------ --------- 7.5/9.8 MB 3.6 MB/s eta
0:00:01
   ------------------------------ --------- 7.5/9.8 MB 3.6 MB/s eta
0:00:01
   ------------------------------ --------- 7.5/9.8 MB 3.6 MB/s eta
0:00:01
   ----------------------------------- ---- 8.6/9.8 MB 3.8 MB/s eta
0:00:01
   ----------------------------------- ---- 8.6/9.8 MB 3.7 MB/s eta
0:00:01
   ------------------------------------ --- 8.9/9.8 MB 3.8 MB/s eta
0:00:01
   ------------------------------------ --- 9.0/9.8 MB 3.8 MB/s eta
0:00:01
   ------------------------------------- -- 9.2/9.8 MB 3.8 MB/s eta
0:00:01
   ------------------------------------- -- 9.3/9.8 MB 3.8 MB/s eta
0:00:01
   -------------------------------------- - 9.4/9.8 MB 3.8 MB/s eta
0:00:01
   -------------------------------------- - 9.6/9.8 MB 3.8 MB/s eta
0:00:01
   ---------------------------------------  9.8/9.8 MB 3.8 MB/s eta
0:00:01
   ---------------------------------------  9.8/9.8 MB 3.8 MB/s eta
0:00:01
   ---------------------------------------- 9.8/9.8 MB 3.7 MB/s eta
0:00:00
Using cached patsy-0.5.6-py2.py3-none-any.whl (233 kB)
Installing collected packages: patsy, statsmodels
Successfully installed patsy-0.5.6 statsmodels-0.14.2
```

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 Nrom the old and new layouts to compare.

To ge<erate the data use the following command:

```
import <umpy as <p
```

```
# 50 purchases out of 1000 visitors

old_layout = <p.array([1] * 50 + [0] * 950)

# 70 purchases out of 1000 visitors

<ew_layout = <p.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)  # 50 purchases out of
1000 visitors
new_layout = np.array([1] * 70 + [0] * 930)  # 70 purchases out of
1000 visitors

# Calculate the number of successes and the number of trials for both
layouts
successes = np.array([old_layout.sum(), new_layout.sum()])
n_trials = np.array([old_layout.size, new_layout.size])

# Apply the z-test
stat, p_value = proportions_ztest(successes, n_trials)

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

# Interpretation
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant
difference in conversion rates between the old and new layouts.")
else:
    print("Fail to reject the null hypothesis: There is no significant
difference in conversion rates between the old and new layouts.")

Z-statistic: -1.883108942886774
P-value: 0.05968560553242621
Fail to reject the null hypothesis: There is no significant difference
in conversion rates between the old and new layouts.
```

M 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 between the scores
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)  # Use ddof=1 for sample
standard deviation
n = len(differences)

# Calculate the z-statistic
z_stat = mean_diff / (std_diff / np.sqrt(n))

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

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

# Interpretation
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 due to the tutoring program.")
```

```
Z-statistic: 4.593190894944668
P-value: 4.365194105293568e-06
Reject the null hypothesis: The tutoring program significantly
improves exam scores.
```

A pharmaceutical company wants to determine in a new drug is effective in reducing blood pressure. They conduct a study and record blood pressure measurements beNore and after administering the drug.

Use the below code to generate samples of respective arrays of blood pressure:

```
before_drug = <p.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])
```

Implement z-test to find if the drug really works or not

```
import numpy as np
from scipy import stats

# Generate the data
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])

# Calculate the differences between the blood pressure measurements
differences = before_drug - after_drug

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

# Calculate the z-statistic
z_stat = mean_diff / (std_diff / np.sqrt(n))

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

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

# Interpretation
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: The drug significantly reduces
blood pressure.")
else:
    print("Fail to reject the null hypothesis: There is no significant reduction in blood pressure due to the drug.")
```

```
Z-statistic: 10.049875621120888
P-value: 0.0
Reject the null hypothesis: The drug significantly reduces blood pressure.
```

A customer service depart:ent clai:s that their average response ti:e is less than 5 :inutesV 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 response 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])
```

Impleme3t z-test to fi3d the claims made by customer service departme3t are tru or false.

```
import numpy as np
from scipy import stats

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

# Population mean (claimed average response time)
mu = 5

# Calculate the sample mean and standard deviation
sample_mean = np.mean(response_times)
sample_std = np.std(response_times, ddof=1)  # Use ddof=1 for sample standard deviation
n = len(response_times)

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

# Calculate the p-value (one-tailed test)
p_value = stats.norm.cdf(z_stat)

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

# Interpretation
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: The average response time is significantly less than 5 minutes.")
```

```python
else:
    print("Fail to reject the null hypothesis: There is no significant
evidence that the average response time is less than 5 minutes.")
```

```
Z-statistic: -3.184457226042963
P-value: 0.0007251287113068958
Reject the null hypothesis: The average response time is significantly
less than 5 minutes.
```

USA company is testing two different website layouts to see which one leads to higher click-through rates. 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:

~~~python

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]

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

def ab_test_analysis(layout_a_clicks, layout_b_clicks):
    """
    Perform an A/B test analysis.

    Parameters:
    layout_a_clicks (list): List of click-through rates for layout A.
    layout_b_clicks (list): List of click-through rates for layout B.

    Returns:
    tuple: (t_statistic, degrees_of_freedom, p_value)
    """
    # Convert lists to numpy arrays
    layout_a_clicks = np.array(layout_a_clicks)
    layout_b_clicks = np.array(layout_b_clicks)

    # Calculate sample statistics
    mean_a = np.mean(layout_a_clicks)
    mean_b = np.mean(layout_b_clicks)
    var_a = np.var(layout_a_clicks, ddof=1)
    var_b = np.var(layout_b_clicks, ddof=1)
    n_a = len(layout_a_clicks)
    n_b = len(layout_b_clicks)

    # Calculate pooled standard deviation and standard error
    pooled_var = ((n_a - 1) * var_a + (n_b - 1) * var_b) / (n_a + n_b
- 2)
    pooled_std = np.sqrt(pooled_var)
~~~

```
    standard_error = pooled_std * np.sqrt(1 / n_a + 1 / n_b)

    # Calculate t-statistic
    t_statistic = (mean_a - mean_b) / standard_error

    # Calculate degrees of freedom
    degrees_of_freedom = n_a + n_b - 2

    # Calculate p-value
    p_value = 2 * (1 - stats.t.cdf(abs(t_statistic),
degrees_of_freedom))

    return t_statistic, degrees_of_freedom, p_value

# Example usage
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]

t_statistic, degrees_of_freedom, p_value =
ab_test_analysis(layout_a_clicks, layout_b_clicks)
print(f"T-statistic: {t_statistic}")
print(f"Degrees of Freedom: {degrees_of_freedom}")
print(f"P-value: {p_value}")

# Interpretation
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant
difference between layout A and layout B.")
else:
    print("Fail to reject the null hypothesis: There is no significant
difference between layout A and layout B.")

T-statistic: -7.298102156175071
Degrees of Freedom: 18
P-value: 8.833437608046779e-07
Reject the null hypothesis: There is a significant difference between
layout A and layout B.
```

U A phar:aceutical co:pany wants to deter:ine if a new drug is :ore effective than an existing drug in reducing cholesterol levelsV Create a progra: to analyze the clinical trial data and calculate the tstatistic and p-value for the treat:ent effect.

Use the followi3g data of cholestrol level:

```
existi3g_drug_levels = [180, 182, 175, 185, 178, 176, 172, 184, 179,
183]

3ew_drug_levels = [170, 172, 165, 168, 175, 173, 170, 178, 172, 176]
```

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

def compare_drugs(existing_drug_levels, new_drug_levels):
    """
    Analyze clinical trial data to compare the effectiveness of
existing and new drugs in reducing cholesterol levels.

    Parameters:
    existing_drug_levels (list): Cholesterol levels for patients
treated with the existing drug.
    new_drug_levels (list): Cholesterol levels for patients treated
with the new drug.

    Returns:
    tuple: (t_statistic, p_value)
    """
    # Convert lists to numpy arrays
    existing_drug_levels = np.array(existing_drug_levels)
    new_drug_levels = np.array(new_drug_levels)

    # Calculate sample statistics
    mean_existing = np.mean(existing_drug_levels)
    mean_new = np.mean(new_drug_levels)
    var_existing = np.var(existing_drug_levels, ddof=1)
    var_new = np.var(new_drug_levels, ddof=1)
    n_existing = len(existing_drug_levels)
    n_new = len(new_drug_levels)

    # Calculate pooled standard deviation and standard error
    pooled_var = ((n_existing - 1) * var_existing + (n_new - 1) *
var_new) / (n_existing + n_new - 2)
    pooled_std = np.sqrt(pooled_var)
    standard_error = pooled_std * np.sqrt(1 / n_existing + 1 / n_new)

    # Calculate t-statistic
    t_statistic = (mean_existing - mean_new) / standard_error

    # Calculate degrees of freedom
    degrees_of_freedom = n_existing + n_new - 2

    # Calculate p-value
    p_value = 2 * (1 - stats.t.cdf(abs(t_statistic),
degrees_of_freedom))

    return t_statistic, p_value

# Example usage
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]
```

```
t_statistic, p_value = compare_drugs(existing_drug_levels,
new_drug_levels)
print(f"T-statistic: {t_statistic}")
print(f"P-value: {p_value}")

# Interpretation
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: The new drug is more effective
than the existing drug in reducing cholesterol levels.")
else:
    print("Fail to reject the null hypothesis: There is no significant
difference in effectiveness between the existing and new drugs.")

T-statistic: 4.14048098620866
P-value: 0.0006143398442373105
Reject the null hypothesis: The new drug is more effective than the
existing drug in reducing cholesterol levels.
```

A school district introduces an educational intervention progra: to i:prove :ath scoresV Write a Python function to analyze pre- and post-intervention test scores, calculating the t-statistic and p-value to deter:ine if the intervention had a significant i:pact.

Use the followi3g data of test score:

'''python

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 numpy as np
from scipy import stats

def intervention_effect(pre_intervention_scores,
post_intervention_scores):
    """
    Analyze the impact of an educational intervention program on math
scores using a paired t-test.

    Parameters:
    pre_intervention_scores (list): List of pre-intervention test
scores.
    post_intervention_scores (list): List of post-intervention test
scores.

    Returns:
    tuple: (t_statistic, p_value)
    """
    # Convert lists to numpy arrays
```

```python
    pre_intervention_scores = np.array(pre_intervention_scores)
    post_intervention_scores = np.array(post_intervention_scores)

    # Calculate differences between pre- and post-intervention scores
    differences = post_intervention_scores - pre_intervention_scores

    # Calculate mean and standard deviation of differences
    mean_diff = np.mean(differences)
    std_diff = np.std(differences, ddof=1)  # Use ddof=1 for sample
standard deviation
    n = len(differences)

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

    # Calculate t-statistic
    t_statistic = mean_diff / standard_error

    # Calculate degrees of freedom
    degrees_of_freedom = n - 1

    # Calculate p-value
    p_value = 2 * (1 - stats.t.cdf(abs(t_statistic),
degrees_of_freedom))

    return t_statistic, p_value

# Example usage
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]

t_statistic, p_value = intervention_effect(pre_intervention_scores,
post_intervention_scores)
print(f"T-statistic: {t_statistic}")
print(f"P-value: {p_value}")
# Interpretation
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: The intervention had a
significant impact on test scores.")
else:
    print("Fail to reject the null hypothesis: The intervention did
not have a significant impact on test scores.")
```

```
T-statistic: 4.428408839657611
P-value: 0.0016509548165795085
Reject the null hypothesis: The intervention had a significant impact
on test scores.
```

An HR department wants to investigate i@ there's a gender-based salary gap within the company. Develop a program to analyze salary data, calculate the t-statistic, and determine i@ there's a statistically signi@icant di@@erence between the average salaries o@ male and @emale employees.

Use the below code to generate synthetic dataY

```
# 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)

import numpy as np
from scipy import 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):
    """
    Analyze salary data to determine if there is a statistically
significant difference between the average salaries of male and female
employees.

    Parameters:
    male_salaries (numpy array): Salaries of male employees.
    female_salaries (numpy array): Salaries of female employees.

    Returns:
    tuple: (t_statistic, p_value)
    """
    # Calculate sample statistics
    mean_male = np.mean(male_salaries)
    mean_female = np.mean(female_salaries)
    var_male = np.var(male_salaries, ddof=1)
    var_female = np.var(female_salaries, ddof=1)
    n_male = len(male_salaries)
    n_female = len(female_salaries)

    # Calculate pooled standard deviation and standard error
    pooled_var = ((n_male - 1) * var_male + (n_female - 1) *
var_female) / (n_male + n_female - 2)
    pooled_std = np.sqrt(pooled_var)
    standard_error = pooled_std * np.sqrt(1 / n_male + 1 / n_female)
    # Calculate t-statistic
    t_statistic = (mean_male - mean_female) / standard_error

    # Calculate degrees of freedom
    degrees_of_freedom = n_male + n_female - 2

    # Calculate p-value
    p_value = 2 * (1 - stats.t.cdf(abs(t_statistic),
degrees_of_freedom))

    return t_statistic, p_value

# Analyze the salary data
t_statistic, p_value = analyze_salary_gap(male_salaries,
female_salaries)
print(f"T-statistic: {t_statistic}")
print(f"P-value: {p_value}")

# Interpretation
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant
difference in salaries between male and female employees.")
else:
    print("Fail to reject the null hypothesis: There is no significant
difference in salaries between male and female employees.")

T-statistic: 0.06114208969631383
P-value: 0.9515665020676465
Fail to reject the null hypothesis: There is no significant difference
in salaries between male and female employees.
```

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 dataY

```
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, 88, 85]

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, 81, 82]

import numpy as np
from scipy import stats

def analyze_quality(version1_scores, version2_scores):
    # Convert lists to numpy arrays
    version1_scores = np.array(version1_scores)
    version2_scores = np.array(version2_scores)

    # Calculate sample statistics
    mean_v1 = np.mean(version1_scores)
    mean_v2 = np.mean(version2_scores)
    var_v1 = np.var(version1_scores, ddof=1)
    var_v2 = np.var(version2_scores, ddof=1)
    n_v1 = len(version1_scores)
    n_v2 = len(version2_scores)

    # Calculate pooled standard deviation and standard error
    pooled_var = ((n_v1 - 1) * var_v1 + (n_v2 - 1) * var_v2) / (n_v1 +
n_v2 - 2)
    pooled_std = np.sqrt(pooled_var)
    standard_error = pooled_std * np.sqrt(1 / n_v1 + 1 / n_v2)

    # Calculate t-statistic
    t_statistic = (mean_v1 - mean_v2) / standard_error

    # Calculate degrees of freedom
    degrees_of_freedom = n_v1 + n_v2 - 2

    # Calculate p-value
    p_value = 2 * (1 - stats.t.cdf(abs(t_statistic),
degrees_of_freedom))

    return t_statistic, p_value


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, 88, 85]
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, 81, 82]

t_statistic, p_value = analyze_quality(version1_scores,
version2_scores)
print(f"T-statistic: {t_statistic}")
print(f"P-value: {p_value}")

# Interpretation
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant
difference in quality between the two product versions.")
else:
    print("Fail to reject the null hypothesis: There is no significant
difference in quality between the two product versions.")
```

```
T-statistic: 11.325830417646698
P-value: 3.552713678800501e-15
Reject the null hypothesis: There is a significant difference in
quality between the two product versions.
```

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 in customer satisfaction between the branches.

Use the below data of scores:

~~~python

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]

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

def analyze_satisfaction(branch_a_scores, branch_b_scores):

    # Convert lists to numpy arrays
    branch_a_scores = np.array(branch_a_scores)
    branch_b_scores = np.array(branch_b_scores)

    # Calculate sample statistics
    mean_a = np.mean(branch_a_scores)
    mean_b = np.mean(branch_b_scores)
    var_a = np.var(branch_a_scores, ddof=1)
    var_b = np.var(branch_b_scores, ddof=1)
    n_a = len(branch_a_scores)
    n_b = len(branch_b_scores)

    # Calculate pooled standard deviation and standard error
    pooled_var = ((n_a - 1) * var_a + (n_b - 1) * var_b) / (n_a + n_b
- 2)
    pooled_std = np.sqrt(pooled_var)
    standard_error = pooled_std * np.sqrt(1 / n_a + 1 / n_b)

    # Calculate t-statistic
    t_statistic = (mean_a - mean_b) / standard_error

    # Calculate degrees of freedom
    degrees_of_freedom = n_a + n_b - 2

    # Calculate p-value
    p_value = 2 * (1 - stats.t.cdf(abs(t_statistic),
degrees_of_freedom))

    return t_statistic, p_value
~~~

```
# Example usage
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]

t_statistic, p_value = analyze_satisfaction(branch_a_scores,
branch_b_scores)
print(f"T-statistic: {t_statistic}")
print(f"P-value: {p_value}")

# Interpretation
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant
difference in customer satisfaction between the two branches.")
else:
    print("Fail to reject the null hypothesis: There is no significant
difference in customer satisfaction between the two branches.")

T-statistic: 5.480077554195742
P-value: 8.895290508625919e-07
Reject the null hypothesis: There is a significant difference in
customer satisfaction between the two branches.
```

? A political analyst wants to determine i@ there is a signi@icant association between age groups and voter pre@erences FCandidate A or Candidate B). They collect data @rom a sample o@ 500 voters and classi@y them into di@@erent age groups and candidate pre@erences. Per@orm a Chi-Square test to determine i@ there is a signi@icant association between age groups and voter pre@erences.

Use the below code to generate data:

```python

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

# Generate data
np.random.seed(0)
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
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(f"Chi-Square statistic: {chi2}")
print(f"P-value: {p}")
print(f"Degrees of freedom: {dof}")
print("Expected frequencies:")
print(expected)

# Interpretation
alpha = 0.05
if p < alpha:
    print("Reject the null hypothesis: There is a significant
association between age groups and voter preferences.")
else:
    print("Fail to reject the null hypothesis: There is no significant
association between age groups and voter preferences.")

Chi-Square statistic: 0.8779923945254768
P-value: 0.6446832311860852
Degrees of freedom: 2
Expected frequencies:
[[96.824 85.176]
 [89.908 79.092]
 [79.268 69.732]]
Fail to reject the null hypothesis: There is no significant
association between age groups and voter preferences.
```

1. 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 test to determine if there is a significant relationship between product satisfaction levels and customer regions.

Sample data:

```
#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
from scipy.stats import chi2_contingency


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(f"Chi-Square statistic: {chi2}")
print(f"P-value: {p}")
print(f"Degrees of freedom: {dof}")
print("Expected frequencies:")
print(expected)

# Interpretation
alpha = 0.05
if p < alpha:
    print("Reject the null hypothesis: There is a significant
relationship between product satisfaction levels and customer
regions.")
else:
    print("Fail to reject the null hypothesis: There is no significant
relationship between product satisfaction levels and customer
regions.")

Chi-Square statistic: 27.777056277056275
P-value: 0.00010349448486004387
Degrees of freedom: 6
Expected frequencies:
[[34.14634146 34.14634146 37.56097561 34.14634146]
 [36.58536585 36.58536585 40.24390244 36.58536585]
 [29.26829268 29.26829268 32.19512195 29.26829268]]
Reject the null hypothesis: There is a significant relationship
between product satisfaction levels and customer regions.
```

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:

```
# 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) and 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(f"Chi-Square statistic: {chi2}")
print(f"P-value: {p}")
print(f"Degrees of freedom: {dof}")
print("Expected frequencies:")
print(expected)

# Interpretation
alpha = 0.05
if p < alpha:
    print("Reject the null hypothesis: There is a significant
relationship between product satisfaction levels and customer
regions.")
else:
    print("Fail to reject the null hypothesis: There is no significant
relationship between product satisfaction levels and customer
regions.")

Chi-Square statistic: 22.161728395061726
P-value: 0.00018609719479882554
Degrees of freedom: 4
Expected frequencies:
[[34.48275862 34.48275862 31.03448276]
 [34.48275862 34.48275862 31.03448276]
 [31.03448276 31.03448276 27.93103448]]
Reject the null hypothesis: There is a significant relationship
between product satisfaction levels and customer regions.
```

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.

Use the following data:

~~~python

# 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]

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


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 = f_oneway(standard_scores, premium_scores,
deluxe_scores)

print(f"F-statistic: {f_statistic}")
print(f"P-value: {p_value}")

# Interpretation
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant
difference in customer satisfaction scores among the three product
versions.")
else:
    print("Fail to reject the null hypothesis: There is no significant
difference in customer satisfaction scores among the three product
versions.")
~~~

```
F-statistic: 27.03556231003039
P-value: 3.578632885734896e-07
Reject the null hypothesis: There is a significant difference in
customer satisfaction scores among the three product versions.
```