Name : Ravi kumar yadav
E-mail : kumaryadavravi016@gmail.com
Assignment name : Data structure

Drive : drive

Github : Github

1 Statistics Advance part -1 assignment

2 Thoery questions

2.0.1 What is a random variable in probability theory?

2.0.2 In probability theory, a random variable is a numerical outcome of a random experiment or process. It is a function that assigns a real number to each possible outcome of a random event. Random variables are used to quantify uncertainty and to analyze probability distributions.

2.0.3 2) What are the types of random variable?

2.0.4 There are two main types of random variables:

(i) Discrete Random Variables: These take on a countable number of distinct values. For example, the number of heads in 10 coin tosses or the number of customers arriving at a store in an hour.

(ii) Continuous Random Variables: These can take any value within a given range and are usually associated with measurements. For example, the height of students in a class, or the time it takes for a car to travel from point A to point B.

2.0.5 3) What is the difference between discrete and continuous distributions?

2.0.6 (i) Discrete Distribution: Describes the probability distribution of discrete random variables. It is characterized by specific probabilities assigned to each possible value. Examples include the binomial distribution or Poisson distribution.

2.0.7 (ii) Continuous Distribution: Describes the probability distribution of continuous random variables. The probability that a continuous random variable equals a specific value is zero; instead, probabilities are described in terms of intervals. Examples include the normal distribution or uniform distribution.

2.0.8 4) What are probability distribution functions (PDF)?

2.0.9 A Probability Distribution Function (PDF) describes the probability distribution of a continuous random variable. The PDF is a function that defines the likelihood of a random variable taking a particular value. For continuous random variables, the probability that the variable takes any specific value is 0, and the probability of it falling within a range is given by the area under the curve of the PDF over that rang

2.0.10 5) How do cumulative distribution functions (CDF) differ from probability distribution functions (PDF)?

2.0.11 (i) CDF: The Cumulative Distribution Function (CDF) of a random variable gives the probability that the variable takes a value less than or equal to a certain value. It accumulates the probabilities from the PDF. For a continuous random variable, the CDF is the integral of the PDF.

2.0.12 (ii) PDF: The Probability Distribution Function (PDF) defines the probability of the random variable taking a specific value. The CDF is the integral of the PDF, which essentially accumulates the probabilities.

2.0.13 6) What is a discrete uniform distribution?

2.0.14 The Discrete Uniform Distribution is a distribution where each of a finite number of outcomes is equally likely. For example, when rolling a fair die, each of the outcomes 1, 2, 3, 4, 5, and 6 has an equal probability of 1/6.

2.0.15 7) What are the key properties of a Bernoulli distribution?

2.0.16 The Bernoulli Distribution represents a random experiment with exactly two outcomes, often termed as “success” (1) and “failure” (0).

2.0.17 (i) The probability of success is p, and the probability of failure is 1− .

2.0.18 (ii)The Bernoulli distribution is a special case of the binomial distribution when the number of trials is 1.

2.0.19 8) What is the binomial distribution, and how is it used in probability?

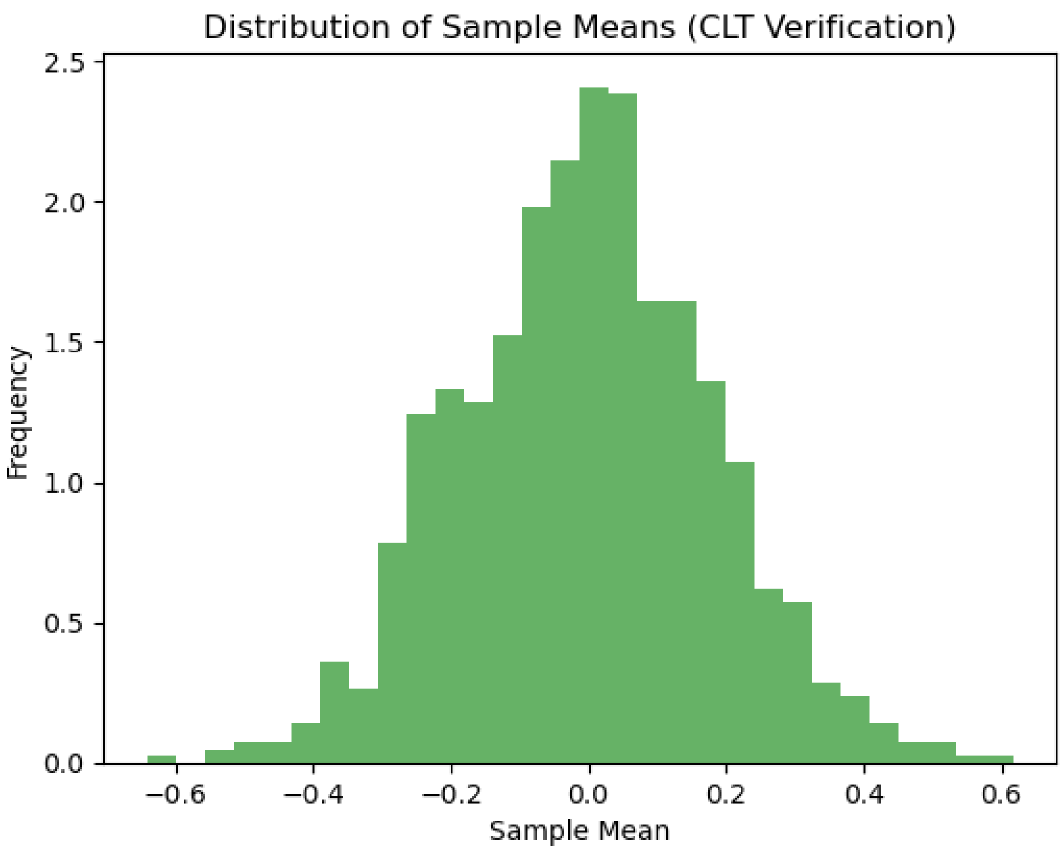
2.0.20 The Binomial Distribution describes the number of successes in a fixed number of independent Bernoulli trials. It is characterized by two parameters: the number of trials and the probability of success p. The probability mass function (PMF) for the binomial distribution is:

2.0.21 P(X=k)=(n/k)pk (1−p)n−k

2.0.22 9) What is the Poisson distribution and where is it applied?

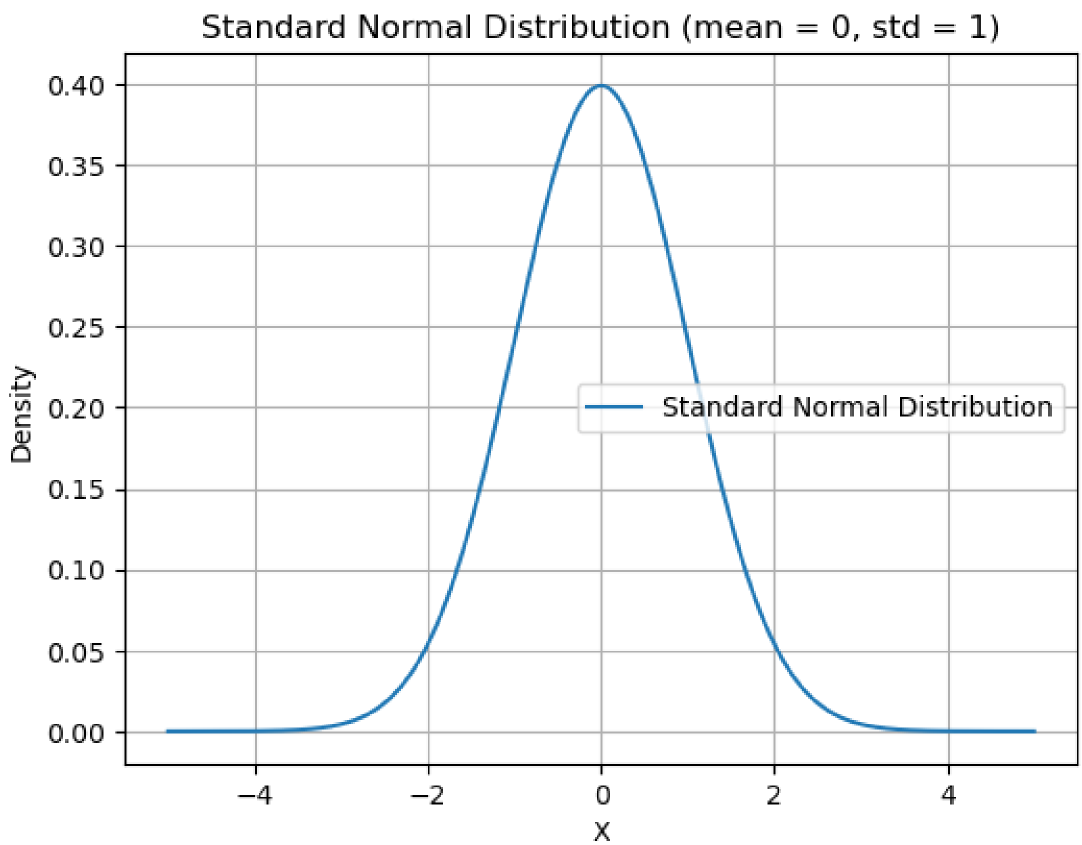
2.0.23 The Poisson Distribution models the number of events occurring within a fixed interval of time or space, given that these events happen with a known constant mean rate, and independently of each other. It is often used in cases

[54]:
```
import numpy as np
import matplotlib.pyplot as plt

# Parameters
n_samples = 1000  # Number of samples
sample_size = 30  # Size of each sample
population_mean = 0
population_std = 1

# Simulate multiple samples
samples = [np.random.normal(population_mean, population_std, sample_size) for _
 ↪in range(n_samples)]
sample_means = [np.mean(sample) for sample in samples]

# Plotting the distribution of sample means
plt.hist(sample_means, bins=30, density=True, alpha=0.6, color='g')
plt.title('Distribution of Sample Means (CLT Verification)')
plt.xlabel('Sample Mean')
plt.ylabel('Frequency')
plt.show()
```



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

[56]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

def plot_standard_normal():
    x = np.linspace(-5, 5, 1000)
    y = norm.pdf(x, 0, 1)  # Standard normal distribution (mean=0, std=1)

    plt.plot(x, y, label='Standard Normal Distribution')
    plt.title('Standard Normal Distribution (mean = 0, std = 1)')
    plt.xlabel('X')
    plt.ylabel('Density')
    plt.grid(True)
    plt.legend()
    plt.show()

plot_standard_normal()
```



3.0.3 17) Generate random variables and calculate their corresponding probabilities using the binomial distribution.

[58]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import binom

# Parameters
n = 10  # Number of trials
p = 0.5  # Probability of success

# Generate binomial distribution
x = np.arange(0, n+1)
y = binom.pmf(x, n, p)

# Plotting
plt.stem(x, y, use_line_collection=True)
plt.title(f'Binomial Distribution (n={n}, p={p})')
plt.xlabel('Number of successes')
plt.ylabel('Probability')
plt.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[58], line 14
     11 y = binom.pmf(x, n, p)
     13 # Plotting
---> 14 plt.stem(x, y, use_line_collection=True)
     15 plt.title(f'Binomial Distribution (n={n}, p={p})')
     16 plt.xlabel('Number of successes')

TypeError: stem() got an unexpected keyword argument 'use_line_collection'
```

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

[60]:
```
import numpy as np
from scipy.stats import norm

def z_score(data_point, mean, std_dev):
    return (data_point - mean) / std_dev

# Example
data_point = 2
mean = 0
std_dev = 1
```

```
z = z_score(data_point, mean, std_dev)
print(f'Z-score for data point {data_point} is {z}')

# Compare Z-score to the standard normal distribution
probability = norm.cdf(z)  # CDF of standard normal distribution at Z
print(f'Probability of Z-score {z}: {probability}')
```

```
Z-score for data point 2 is 2.0
Probability of Z-score 2.0: 0.9772498680518208
```

3.0.5 19) Implement hypothesis testing using Z-statistics for a sample datase.

[62]:
```
import numpy as np
from scipy.stats import norm

# Sample data
sample_data = np.array([45, 48, 52, 50, 47, 53, 49])
sample_mean = np.mean(sample_data)
sample_size = len(sample_data)
population_mean = 50
population_std = 5  # Assumed standard deviation

# Z-test calculation
z_stat = (sample_mean - population_mean) / (population_std / np.
 ↪sqrt(sample_size))

# P-value calculation for two-tailed test
p_value = 2 * (1 - norm.cdf(abs(z_stat)))

print(f'Z-statistic: {z_stat}')
print(f'P-value: {p_value}')
```

```
Z-statistic: -0.45355736761107107
P-value: 0.6501474440948556
```

3.0.6 20) Create a confidence interval for a dataset using Python and interpret the resul

[64]:
```
import numpy as np
from scipy import stats

# Sample data
data = np.random.normal(50, 5, 100)  # Mean = 50, Std dev = 5, 100 samples
confidence_level = 0.95

# Calculate confidence interval
```

```
mean = np.mean(data)
std_error = stats.sem(data)
margin_of_error = std_error * stats.t.ppf((1 + confidence_level) / 2.,␣
 ↪len(data) - 1)

lower_bound = mean - margin_of_error
upper_bound = mean + margin_of_error

print(f'Confidence Interval: ({lower_bound}, {upper_bound})')
```

```
Confidence Interval: (49.207346424107676, 51.128524384215744)
```

3.0.7 21) Generate data from a normal distribution, then calculate and interpret the confidence interval for its mea.

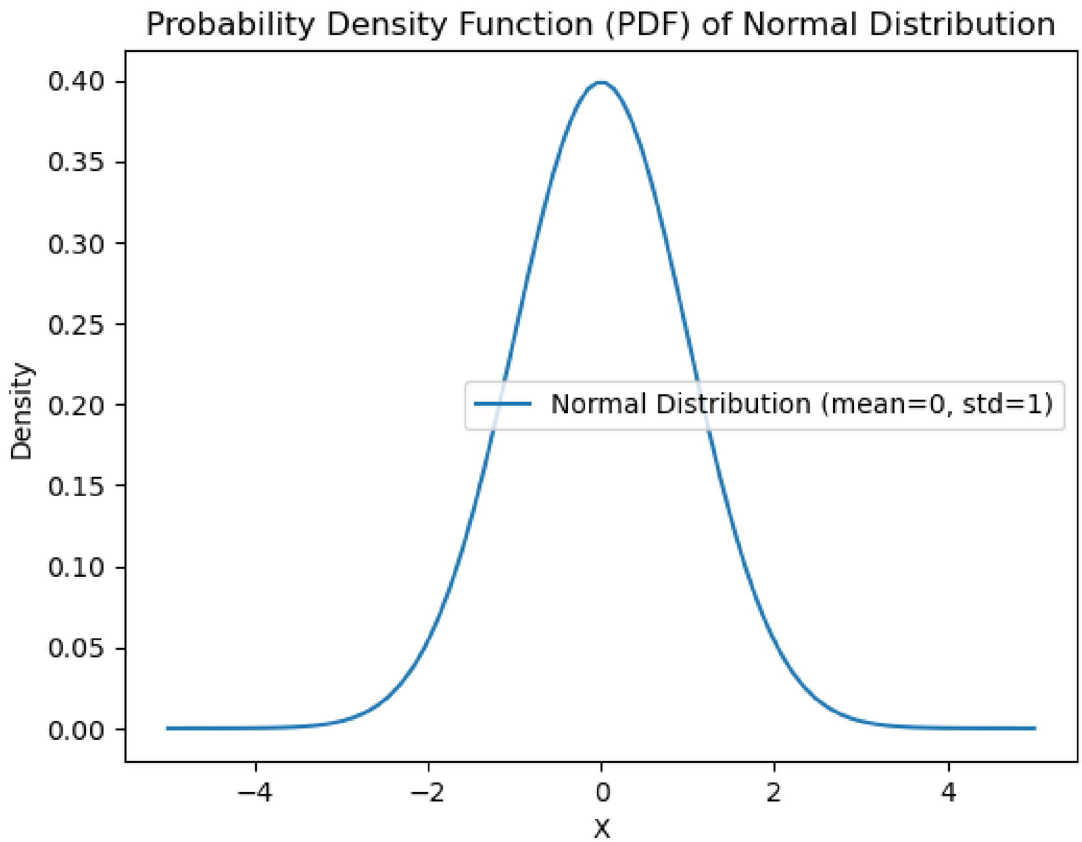
[66]:
```
import numpy as np
from scipy import stats

# Generate data
data = np.random.normal(10, 2, 100)  # Mean = 10, Std dev = 2, 100 samples

# Confidence interval for mean
mean = np.mean(data)
std_error = stats.sem(data)
confidence_interval = stats.t.interval(0.95, len(data)-1, loc=mean,␣
 ↪scale=std_error)

print(f'95% Confidence Interval for the mean: {confidence_interval}')
```

```
95% Confidence Interval for the mean: (9.621109196519436, 10.432356714072503)
```

3.0.8 22) Write a Python script to calculate and visualize the probability density function (PDF) of a normal distributio.

[68]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Parameters for the normal distribution
mu = 0
sigma = 1

# Generate values for X and calculate the PDF
x = np.linspace(-5, 5, 100)
pdf_values = norm.pdf(x, mu, sigma)

# Plot the PDF
```

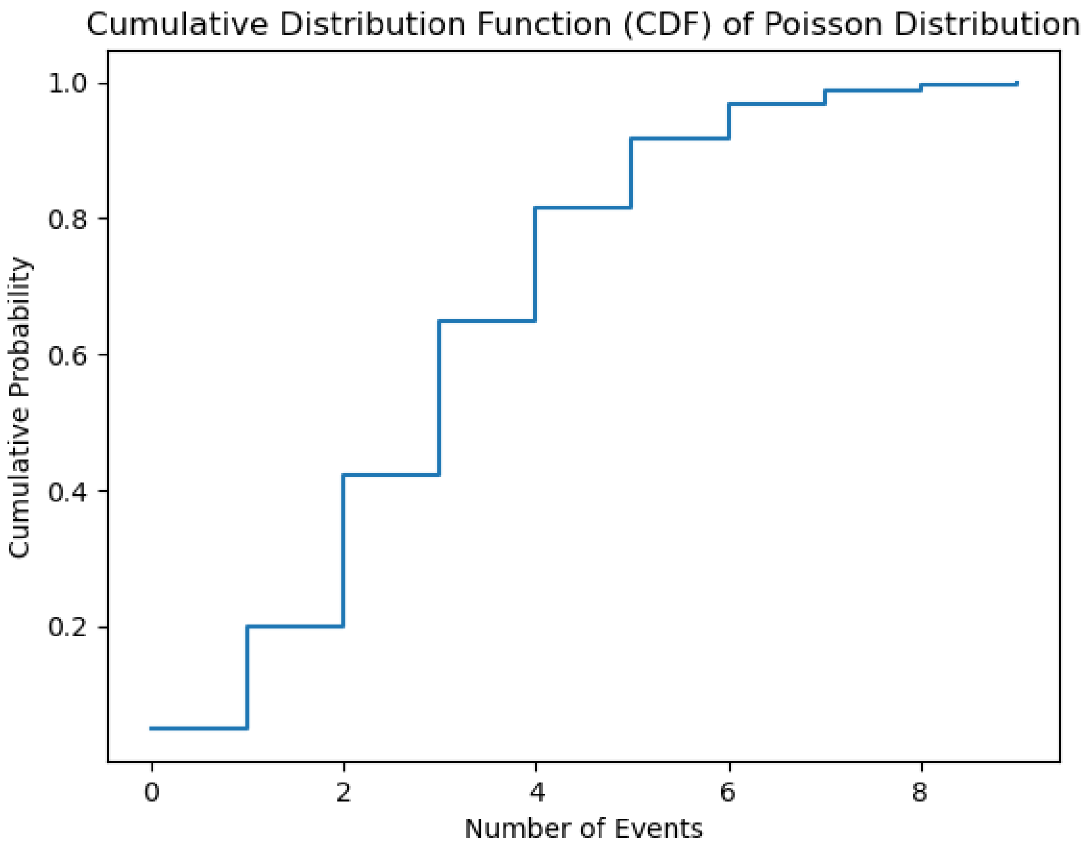
```
plt.plot(x, pdf_values, label='Normal Distribution (mean=0, std=1)')
plt.title('Probability Density Function (PDF) of Normal Distribution')
plt.xlabel('X')
plt.ylabel('Density')
plt.legend()
plt.show()
```



3.0.9 23) Use Python to calculate and interpret the cumulative distribution function (CDF) of a Poisson distributio

[70]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import poisson

# Parameters for the Poisson distribution
lambda_param = 3  # Average rate of occurrence

# Generate values and calculate the CDF
x = np.arange(0, 10)
```

```
cdf_values = poisson.cdf(x, lambda_param)

# Plot CDF
plt.step(x, cdf_values, where='post')
plt.title('Cumulative Distribution Function (CDF) of Poisson Distribution')
plt.xlabel('Number of Events')
plt.ylabel('Cumulative Probability')
plt.show()
```



3.0.10 24) Simulate a random variable using a continuous uniform distribution and calculate its expected value

[72]:
```
import numpy as np

# Parameters
a = 0   # Minimum value
b = 1   # Maximum value

# Simulate data from a continuous uniform distribution
data = np.random.uniform(a, b, 1000)
```

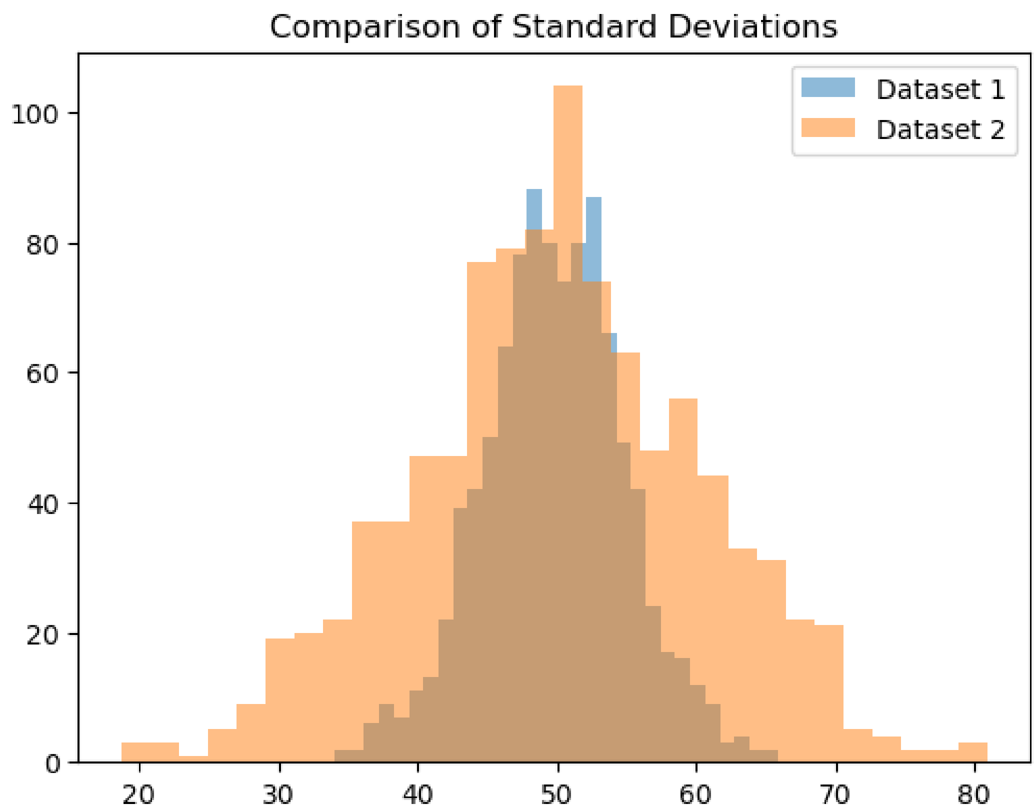
```
# Calculate expected value (mean) of the distribution
expected_value = (a + b) / 2
print(f'Expected Value (Mean) of Continuous Uniform Distribution:␣
 ↪{expected_value}')
```

```
Expected Value (Mean) of Continuous Uniform Distribution: 0.5
```

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

[74]:
```
import numpy as np
import matplotlib.pyplot as plt

# Simulate two datasets
data1 = np.random.normal(50, 5, 1000)
data2 = np.random.normal(50, 10, 1000)

# Calculate standard deviations
std1 = np.std(data1)
std2 = np.std(data2)

print(f'Standard Deviation of Dataset 1: {std1}')
print(f'Standard Deviation of Dataset 2: {std2}')

# Plot histograms to visualize the difference
plt.hist(data1, bins=30, alpha=0.5, label='Dataset 1')
plt.hist(data2, bins=30, alpha=0.5, label='Dataset 2')
plt.legend()
plt.title('Comparison of Standard Deviations')
plt.show()
```

```
Standard Deviation of Dataset 1: 5.116466386432097
Standard Deviation of Dataset 2: 10.338632346037777
```



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

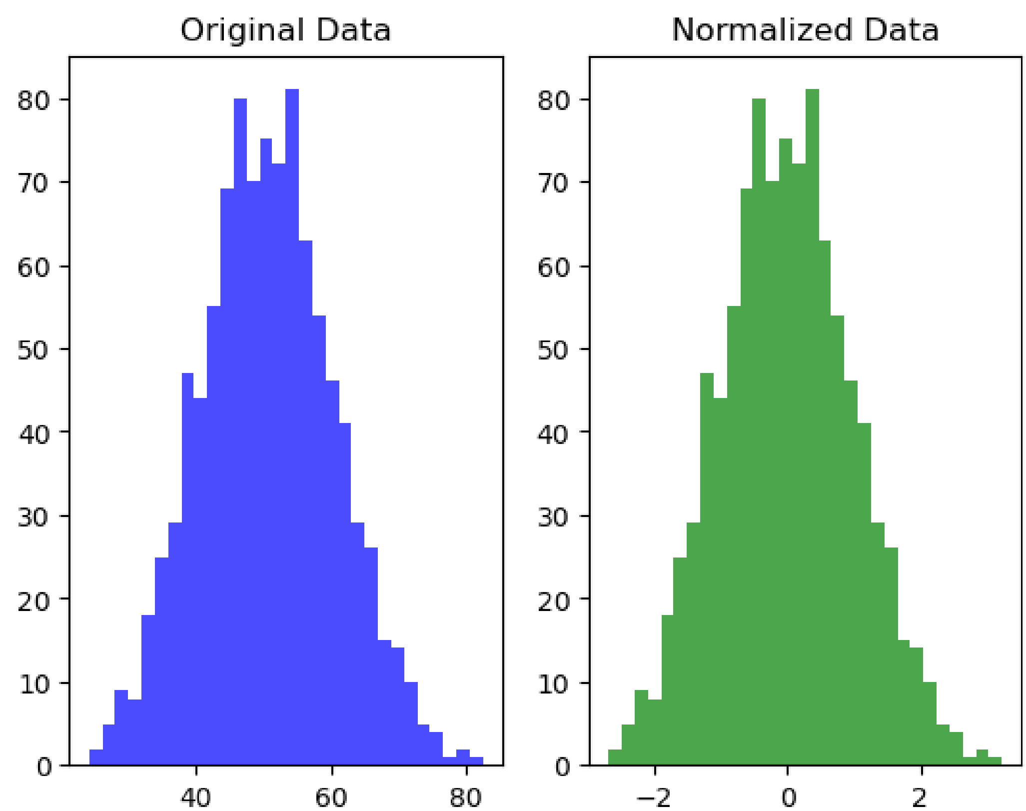
[76]:
```
import numpy as np

# Generate normal distribution data
data = np.random.normal(10, 2, 1000)

# Calculate range and IQR
data_range = np.max(data) - np.min(data)
iqr = np.percentile(data, 75) - np.percentile(data, 25)

print(f'Range of the dataset: {data_range}')
print(f'Interquartile Range (IQR) of the dataset: {iqr}')
```

```
Range of the dataset: 13.408509634190036
Interquartile Range (IQR) of the dataset: 2.502982510771261
```

3.0.13 27) Implement Z-score normalization on a dataset and visualize its transformation .

[78]:

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

# Generate data
data = np.random.normal(50, 10, 1000)

# Z-score normalization
normalized_data = (data - np.mean(data)) / np.std(data)

# Plot original vs normalized data
plt.subplot(1, 2, 1)
plt.hist(data, bins=30, alpha=0.7, color='blue')
plt.title('Original Data')

plt.subplot(1, 2, 2)
plt.hist(normalized_data, bins=30, alpha=0.7, color='green')
plt.title('Normalized Data')

plt.show()
```



3.0.14 28) Write a Python function to calculate the skewness and kurtosis of a dataset generated from anormal distributiion.n.

```
[ ]:
```