Statistics Basics (Module 1)

Theory Questions

Q1. What is statistics, and why is it important?

Ans-> Statistics is a branch of mathematics that deals with the collection, analysis, interpretation, and presentation of data. It's important because it helps us make informed decisions, understand patterns and trends, and draw conclusions from data.

- Elaboration:
- Data Collection and Organization: Statistics provides methods for gathering, organizing, and summarizing data in a meaningful way, like tables, graphs, and charts.
- Data Analysis and Interpretation: It offers tools to analyze data, identify relationships, and draw conclusions based on patterns and trends.
- Inference and Prediction: Statistics allows us to make inferences about larger populations based on data from smaller samples and to make predictions about future events.
- Decision Making: By understanding the underlying patterns and trends in data, we can make more informed and data-driven decisions in various fields like business, science, and public policy.
- Real-world Applications: Statistics is used in diverse fields, including business (analyzing market trends), medicine (understanding the effectiveness of treatments), biology (studying populations), psychology (understanding human behavior), and social sciences (analyzing social phenomena).

Q2. What are two main types of statistics?

Ans-> The two main types of statistics are Descriptive Statistics and Inferential Statistics. Descriptive statistics focuses on summarizing and describing data using methods like charts and graphs, while inferential statistics uses data from samples to make inferences or generalizations about larger populations.

Q3. What are descriptive statistics?

Ans-> Descriptive Statistics:

- Purpose: To summarize and describe data, making it easier to understand and interpret.
- Methods: Includes measures like mean, median, mode, standard deviation, and creating graphs and charts.
- Example: Calculating the average age of students in a classroom (descriptive statistics).
- Inferential Statistics: Purpose: To draw conclusions or make inferences about a larger population based on data from a sample.

- Methods: Includes hypothesis testing, confidence intervals, and other statistical techniques.
- Example: Using data from a survey to estimate the proportion of voters who support a certain candidate in an entire city (inferential statistics).

Q4. What is inferential statistics?

Ans-> Inferential statistics is a branch of statistics that uses sample data to draw conclusions or make predictions about a larger population. It involves using probability theory and statistical models to estimate population parameters, test hypotheses, and make generalizations beyond the observed data.

- Here's a more detailed explanation:
- Key Concepts: Population: The entire group of individuals or items that are of interest in a study.
- Sample: A subset of the population that is selected for study. Parameter: A numerical value that describes a characteristic of the population (e.g., the average height of all students in a school).
- Statistic: A numerical value that describes a characteristic of a sample (e.g., the average height of a sample of students).
- Inference: The process of drawing conclusions or making predictions about a population based on sample data.

Q5. What is sampling in statistics?

Ans-> In statistics, sampling refers to the process of selecting a subset (a sample) from a larger group (a population) to make observations and inferences about that population. It's used when studying the entire population is impractical or impossible due to its size or cost.

Q6. What are the different types of sampling methods?

Ans-> In statistics, sampling methods are broadly categorized into probability and non-probability sampling. Probability sampling ensures each member of the population has a known chance of being selected, while non-probability sampling does not. Probability sampling includes methods like simple random, stratified, systematic, and cluster sampling, while non-probability sampling methods include convenience, quota, purposive, and snowball sampling.

- Probability Sampling Methods:
- Simple Random Sampling: Every member of the population has an equal chance of being selected.
- Stratified Sampling: The population is divided into subgroups (strata), and samples are randomly selected from each stratum.
- Systematic Sampling: Every kth member of the population is selected, where k is a fixed number.

- Cluster Sampling: The population is divided into clusters, and then entire clusters are randomly selected.
- Non-Probability Sampling Methods:
- Convenience Sampling: Individuals are selected based on accessibility or convenience.
- Quota Sampling: A sample is selected to match certain quotas within the population.
- Purposive Sampling: Individuals are selected based on a specific purpose or criterion.
- Snowball Sampling: Participants are asked to recruit other participants who are similar to them.

Q7. What is the difference between random and non-random sampling?

Ans-> Random sampling (also known as probability sampling) ensures every member of a population has an equal chance of being selected for the sample, leading to a more representative and unbiased sample. Non-random sampling (or non-probability sampling), on the other hand, uses criteria other than random selection, potentially introducing bias and making it harder to generalize findings to the broader population.

```
* Here's a more detailed breakdown:
```

- Random Sampling:
- Definition: Each member of the population has a known, non-zero probability of being selected.
- Methods: Examples include simple random sampling, stratified sampling, cluster sampling, and systematic sampling.
- Advantages: Reduces bias, allows for statistical inference to the broader population, and results in more representative samples.
- Disadvantages: Can be more time-consuming and costly, especially for large populations, and may not be feasible if a list of the population is unavailable.
- Non-Random Sampling:
- Definition: Selection is based on factors like convenience, judgment, or researcher preferences, rather than random chance.
- Methods: Examples include convenience sampling, purposive sampling, quota sampling, and snowball sampling.
- Advantages: Can be faster, cheaper, and easier to implement than random sampling, especially when studying specific subgroups or exploratory research.
- Disadvantages: Introduces bias, making it difficult to generalize findings to the broader population, and may not be suitable for statistical analysis.

Q8. Define and give examples of qualitative and quantitative data?

Ans-> Qualitative data describes characteristics or qualities that cannot be numerically measured, like colors or emotions, while quantitative data involves numerical measurements or counts, like height or weight.

- Qualitative Data:
- Definition: Data that describes qualities, characteristics, or concepts that cannot be expressed numerically.
- Examples: Colors: Red, blue, green.
- Feelings: Happy, sad, angry. Survey responses: Open-ended questions where respondents can write their thoughts or feelings.
- Interview transcripts: Words spoken during an interview. Ethnicity, gender, or marital status: Categorical data that describes characteristics.
- Quantitative Data:
- Definition: Data that can be measured or counted numerically.
- Examples:
- Height: Measured in centimeters or feet.
- Weight: Measured in kilograms or pounds.
- Test scores: Numerical grades or marks.
- Website traffic: Number of visits or page views.
- Age: Number of years a person has lived.

Q9. What are the different types of data in statistics?

Ans-> In statistics, data is broadly classified into qualitative (categorical) and quantitative (numerical) types, which are further subdivided into nominal, ordinal, discrete, and continuous. Qualitative data describes attributes or qualities, while quantitative data represents measurable quantities.

1. Qualitative (Categorical) Data:

- Nominal: Data that can be categorized but has no inherent order or ranking (e.g., colors, types of fruits).
    - Ordinal: Data that can be categorized and ranked, but the intervals between categories are not necessarily equal (e.g., levels of satisfaction, grades).

2. Quantitative (Numerical) Data:

    - Discrete: Data that can only take on a specific number of values, usually whole numbers, and cannot be subdivided (e.g., number of students in a class, number of cars in a parking lot).
    - Continuous: Data that can take on any value within a given range and can be subdivided into smaller increments (e.g., height, weight, temperature).

Q10. Explain nominal, ordinal, interval, and ratio levels of measurement?

Ans-> The four levels of measurement are nominal, ordinal, interval, and ratio. Nominal data can only be categorized, ordinal data can be categorized and ranked, interval data can be categorized, ranked, and have equal intervals between values, and ratio data can be categorized, ranked, have equal intervals, and have a meaningful zero point.

- Here's a more detailed explanation:
- Nominal: This is the most basic level. Data is simply placed into categories with no inherent order or ranking. Examples include gender (male, female), eye color, or types of fruits.
- Ordinal: Data can be categorized and ranked, but the difference between the ranks may not be equal. Think of survey responses like "strongly agree," "agree," "neutral," "disagree," "strongly disagree." The order is clear, but the difference between "strongly agree" and "agree" isn't necessarily the same as the difference between "agree" and "neutral".
- Interval: Data can be categorized, ranked, and the intervals between values are equal. However, there's no true zero point. A good example is temperature in Celsius or Fahrenheit. The difference between 20°C and 21°C is the same as the difference between 21°C and 22°C, but 0°C doesn't mean the absence of temperature.
- Ratio: This is the highest level. Data can be categorized, ranked, have equal intervals, and a meaningful zero point exists. Examples include height, weight, or income. A height of 0 means no height, and a weight of 0 means no weight.

Q11. What is the measure of central tendancy?

Ans-> A measure of central tendency (also referred to as measures of centre or central location) is a summary measure that attempts to describe a whole set of data with a single value that represents the middle or centre of its distribution.

Q12. Define mean, median and mode?

Ans-> In statistics, mean is the average of a dataset, found by summing all values and dividing by the number of values. Median is the middle value in an ordered dataset. Mode is the value that appears most frequently in a dataset.

- Mean: The mean, also known as the arithmetic average, is calculated by summing all the values in a dataset and then dividing by the total number of values. For example, if the dataset is {2, 4, 6, 8, 10}, the mean is (2+4+6+8+10)/5 = 6.
- Median: The median is the middle value in a dataset when the data is arranged in numerical order (ascending or descending). If there is an odd number of values, the median is the single middle value. If there is an even number of values, the median is the

average of the two middle values. For example, if the dataset is {2, 4, 6, 8, 10}, the median is 6.

- Mode: The mode is the value that appears most frequently in a dataset. A dataset can have one mode, multiple modes, or no mode at all. For example, if the dataset is {1, 2, 2, 3, 4, 4, 4, 5}, the mode is 4.

Q13. What is the significance of the measure of central tendancy?

Ans-> Measures of central tendency, like mean, median, and mode, are crucial in statistics as they provide a single value that represents the center or typical value of a dataset, offering a concise summary of the data's characteristics. They help in understanding the overall nature of a dataset and facilitate comparisons between different datasets. Here's a more detailed explanation of their significance:

1. Summarizing Data: Central tendency measures condense a large dataset into a single, representative value, making it easier to understand and interpret the data's central point. This summary provides a snapshot of the data's typical or average value, which can be useful for various applications.
2. Facilitating Comparison: Central tendency measures allow for easy comparison of different datasets or subgroups within a dataset. For instance, comparing the average income of two different cities or the median age of two different groups.
3. Guiding Decision-Making: In business, central tendency measures can be used to make informed decisions, such as setting pricing strategies, determining employee compensation, or forecasting future earnings. In research, they help in drawing conclusions and making generalizations about the data.
4. Understanding Data Distribution: While central tendency measures focus on the central value, understanding the context of the distribution is also important. For example, the median is often preferred over the mean when dealing with datasets that have outliers, as it is less sensitive to extreme values. Similarly, the mode is useful for identifying the most frequent value in a dataset.
5. Foundation for Further Analysis: Measures of central tendency serve as a foundational step in statistical analysis. They provide a starting point for exploring other aspects of the data, such as dispersion or variability.

Q14. What is variance, and how is it calculated?

Ans-> Variance is a statistical measure that quantifies the spread or dispersion of a set of data. It's essentially the average of the squared differences of each data point from the mean of the dataset. A higher variance indicates greater variability, meaning the data points are more spread out from the mean, while a lower variance suggests the data points are clustered closer to the mean.

- Calculation: Calculate the mean (average) of the dataset: Sum all the data points and divide by the number of data points. Find the difference between each data point and the mean: Subtract the mean from each individual data point in the dataset.
- Square each of the differences: Square the results obtained in step 2. Calculate the average of the squared differences: Sum all the squared differences and divide by the number of data points (for population variance) or the number of data points minus one (for sample variance). This final step gives you the variance. In summary, variance tells you how much the data points in a set are spread out from their average value.

Q15. What is standard deviation, and why is it important?

Ans-> Standard deviation is a measure of how spread out data points are from the average (mean) value in a dataset. A high standard deviation indicates that the data points are widely dispersed, while a low standard deviation suggests the data points are clustered closely around the mean. It's a key concept in statistics for understanding data variability and making informed decisions. Why is standard deviation important?

- Understanding Data Variability: Standard deviation helps quantify the amount of variation within a dataset, providing insights into how much individual data points deviate from the average.
- Data Interpretation: It helps interpret the mean in a more meaningful way by revealing the degree of spread or clustering around the average.
- Comparing Datasets: Standard deviation allows for comparing the variability of different datasets, even if they have the same mean.
- Risk Assessment: In fields like finance, standard deviation is used to measure the volatility of assets and assess investment risk.
- Quality Control: In manufacturing and other industries, standard deviation helps monitor the consistency of products and identify potential quality issues.
- Statistical Significance: Standard deviation is crucial in statistical analysis, helping determine if observed differences between groups are statistically significant or due to random variation.
- Normal Distribution: In normal distributions, standard deviation helps determine the proportion of data within certain intervals around the mean (e.g., 68% within one standard deviation, 95% within two standard deviations).
- Outlier Detection: It can help identify outliers or unusual data points that deviate significantly from the average.

Q16. Define and explain the term range in statistics?

Ans-> In statistics, the range is a measure of variability that represents the spread of a dataset. It's calculated by finding the difference between the highest and lowest values in the data set.

- Here's a more detailed explanation:

- Definition: The range is the difference between the maximum and minimum values in a dataset.
- Calculation: You find the range by subtracting the smallest value in the dataset from the largest value.
- Example: If you have a dataset {2, 5, 8, 10, 3}, the highest value is 10 and the lowest value is 2. The range is 10 - 2 = 8.
- Interpretation: A larger range indicates that the data points are more spread out, while a smaller range suggests that the data points are clustered closer together.

Q17. What is the difference between variance and standard deviation?

Ans-> Variance and standard deviation are both measures of data spread, but they differ in how they are calculated and what information they provide. Variance is the average squared difference of each data point from the mean, while standard deviation is the square root of the variance. Standard deviation is expressed in the same units as the original data, making it easier to interpret, while variance is expressed in squared units.

- Here's a more detailed breakdown:
- Variance:
- Calculates the average squared difference between each data point and the mean.
- Measures the spread or dispersion of the data. Units are squared, making it less intuitive to understand. Often used in statistical calculations and more advanced analyses. Formula: $\sigma^2 = \Sigma(x_i - \mu)^2 / N$ (population variance) or $s^2 = \Sigma(x_i - \bar{x})^2 / (n-1)$ (sample variance).
- Standard Deviation:
- The square root of the variance. Provides a measure of how much data points typically deviate from the mean. Expressed in the same units as the original data, making it more easily understood and interpreted. Widely used as a measure of spread in various applications. Formula: $\sigma = \sqrt{\sigma^2}$ (population standard deviation) or $s = \sqrt{s^2}$ (sample standard deviation).
- In simpler terms: Imagine you have a set of student scores. Variance would tell you how much the scores deviate from the average score, while standard deviation would tell you how much the scores typically deviate from the average in the same units as the scores themselves.

Q18. What is skewness in a dataset?

Ans-> Skewness in data refers to the degree of asymmetry in a probability distribution. It indicates whether the data is more concentrated on one side of the mean compared to the other. A positively skewed distribution has a longer tail on the right side, while a negatively skewed distribution has a longer tail on the left side.

Q19. What does it mean if a dataset is positively or negatively skewed?

Ans-> A positively skewed dataset has more data points on the lower end of the distribution, with a longer tail extending towards the higher values. In contrast, a negatively skewed dataset has more data points on the higher end, with a longer tail extending towards the lower values.

- Positively Skewed:
- Description: Most data points cluster around the lower values, with a few larger outliers pulling the tail to the right.
- Mean vs. Median: The mean (average) is typically greater than the median in a positively skewed distribution.
- Example: Income distributions often show a positive skew, with a small percentage of individuals earning significantly higher incomes than the majority.
- Negatively Skewed:
- Description: Most data points cluster around the higher values, with a few smaller outliers pulling the tail to the left.
- Mean vs. Median: The mean is typically less than the median in a negatively skewed distribution.
- Example: Test scores in an easy exam might be negatively skewed, with most students scoring high and a few lower scores pulling the tail to the left.

Q20. Define and explain kurtosis?

Ans-> Kurtosis is a statistical measure that describes the shape of a probability distribution's tails compared to its peak. It essentially quantifies the "tailedness" of a distribution, indicating whether it has heavy tails (more extreme values) or light tails (fewer extreme values) relative to a normal distribution. In simpler terms, it measures how "peaked" or "flat" the distribution is.

- Here's a more detailed explanation:
- Tailedness: Kurtosis helps determine if a distribution has many outliers (extreme values) in its tails (the ends of the distribution).
- Peakedness/Flatness: A distribution with high kurtosis (leptokurtic) is more peaked and has heavier tails than a normal distribution, while a distribution with low kurtosis (platykurtic) is flatter and has lighter tails.
- Types of Kurtosis: Leptokurtic: Higher kurtosis than a normal distribution (more peaked, heavy tails).
- Mesokurtic: Kurtosis is similar to a normal distribution (standard kurtosis is 3, often referred to as excess kurtosis of 0).
- Platykurtic: Lower kurtosis than a normal distribution (flatter, light tails).
- Applications: Kurtosis is useful in various fields like finance (assessing extreme price movements), quality control (consistency of measurements), and more. It's often used in

conjunction with other measures like skewness and standard deviation to get a complete picture of a dataset.

Q21. What is the purpose of covariance?

Ans-> Covariance in statistics quantifies the relationship between two random variables, specifically how much they change together. It indicates the direction of their relationship (positive, negative, or no linear relationship). In essence, covariance helps understand whether two variables tend to move in the same direction (positive covariance) or opposite directions (negative covariance).

Q22. What does correlation measure in statistics?

Ans-> In statistics, correlation measures the strength and direction of the relationship between two variables. It's a measure of how much two variables change together. Correlation doesn't imply causation, it only describes the association between variables.

- Here's a more detailed explanation:
- Strength of the relationship: The correlation coefficient, often denoted by 'r', ranges from -1 to +1. A correlation of +1 indicates a perfect positive linear relationship (as one variable increases, the other increases proportionally), while -1 indicates a perfect negative linear relationship (as one variable increases, the other decreases proportionally). A correlation of 0 indicates no linear relationship.
- Direction of the relationship: A positive correlation means the variables tend to move in the same direction, while a negative correlation means they tend to move in opposite directions.
- Linearity: Correlation measures the extent to which the relationship between two variables can be described by a straight line. It doesn't capture non-linear relationships.
- Correlation vs. Causation: It's crucial to remember that correlation does not imply causation. Just because two variables are correlated doesn't mean one causes the other. There could be other factors influencing both variables.

Q23. What is the difference between covariance and correlation?

Ans-> Covariance and correlation are both statistical measures used to assess the relationship between two variables, but they differ in their interpretation and application. Covariance measures the direction and magnitude of the relationship, while correlation standardizes this relationship, making it easier to compare across different datasets and variables.

- Here's a breakdown of the key differences:
- Covariance
- Measurement: Measures how two variables change together, indicating the direction and magnitude of their relationship.

- Interpretation: Covariance values can be positive (variables move in the same direction), negative (variables move in opposite directions), or zero (no linear relationship). The magnitude of covariance is not standardized and depends on the scales of the variables.
- Standardization: Not standardized, meaning the magnitude can vary greatly depending on the units of the variables.
- Purpose: Useful for understanding the direction and magnitude of a relationship, but not for comparing the strength of relationships across different variables or datasets.
- Correlation
- Measurement: Measures the strength and direction of a linear relationship, standardized to a range of -1 to 1.
- Interpretation: Correlation values range from -1 to 1, where -1 indicates a perfect negative correlation, 0 indicates no linear correlation, and 1 indicates a perfect positive correlation.
- Standardization: Standardized, meaning the magnitude is always between -1 and 1, making it easier to compare across different variables and datasets.
- Purpose: Useful for comparing the strength and direction of linear relationships across different variables and datasets, as the standardized values provide a consistent benchmark.

Q24. What are some real-world application of statistics?

Ans-> Statistics is widely used in various real-world applications, including business decision-making, scientific research, healthcare, and data analysis. It helps in understanding trends, making predictions, and drawing conclusions from data.

- Here's a more detailed look at some applications:
    1. Business and Finance: Market Research: Statistics helps businesses understand consumer behavior, identify target markets, and make informed decisions about product development and marketing strategies. Financial Analysis: Statistical methods are used to assess investment risks, analyze financial data, and predict market trends. Quality Control: In manufacturing, statistics helps ensure product quality by monitoring production processes and identifying potential defects. Business Decision Making: Statistics provides insights into market demand, customer preferences, and operational efficiency, enabling businesses to make better decisions.
    2. Science and Research: Scientific Experiments: Statistics is used to analyze experimental data, test hypotheses, and draw conclusions about the results of scientific research. Epidemiological Studies: Statistics helps researchers investigate the spread and causes of diseases, develop public health strategies, and assess the effectiveness of interventions. Predictive Modeling: Statistical models can be used

to predict future outcomes, such as weather patterns, disease outbreaks, or economic trends.

3. Healthcare: Clinical Trials: Statistics plays a crucial role in designing, conducting, and analyzing clinical trials to evaluate the efficacy and safety of new drugs and treatments. Patient Care: Statistical data can help healthcare professionals diagnose diseases, personalize treatment plans, and monitor patient outcomes.
4. Daily Life: Personal Finance: Statistics helps individuals manage their budgets, track expenses, and make informed decisions about investments. Time Management: Statistical analysis can help individuals optimize their schedules and prioritize tasks. Health and Wellness: Statistics can be used to track fitness progress, analyze dietary habits, and make informed choices about health and wellness. Weather Forecasting: Statistical models are used to predict future weather patterns and conditions. Political Campaigns: Statistics is used to analyze voter data, predict election outcomes, and tailor campaign messages.

Practical Questions

Q1. How do you calculate the mean, median and mode of a dataset?

```
# prompt: How do you calculate the mean, median and mode of a dataset

import statistics

def calculate_stats(data):
    """Calculates the mean, median, and mode of a dataset.

    Args:
    data: A list of numerical values.

    Returns:
    A dictionary containing the mean, median, and mode of the dataset.
    Returns an error message if the dataset is empty or invalid.
    """
    try:
        if not data:
            return "Error: Dataset is empty."

        if not all(isinstance(x, (int, float)) for x in data):
            return "Error: Invalid dataset. Please provide numerical values."

        mean = statistics.mean(data)
        median = statistics.median(data)

        # Handle cases where there might be multiple modes or no mode
        try:
            mode = statistics.mode(data)
        except statistics.StatisticsError:
```

```
            mode = "No unique mode found."

        return {"mean": mean, "median": median, "mode": mode}

    except Exception as e:  # Catch any other potential errors
        return f"An error occurred: {e}"

# Example usage:
data1 = [2, 4, 6, 8, 10]
data2 = [1, 2, 2, 3, 4, 4, 4, 5]
data3 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] # Even number of data points
data4 = [] # Empty data set
data5 = [1, 2, 'a', 4] # Invalid data set

print(calculate_stats(data1))
print(calculate_stats(data2))
print(calculate_stats(data3))
print(calculate_stats(data4))
print(calculate_stats(data5))
```

```
{'mean': 6, 'median': 6, 'mode': 2}
{'mean': 3.125, 'median': 3.5, 'mode': 4}
{'mean': 5.5, 'median': 5.5, 'mode': 1}
Error: Dataset is empty.
Error: Invalid dataset. Please provide numerical values.
```

Q2. Write a Python program to compute the variance and standard deviation of a dataset.

```
# prompt: Write a Python program to compute the variance and standard deviation of a data

import math

def calculate_variance_std(data):
    """Calculates the variance and standard deviation of a dataset.

    Args:
        data: A list of numerical values.

    Returns:
        A dictionary containing the variance and standard deviation.
        Returns an error message if the input is invalid or empty.
    """
    try:
        if not data:
            return "Error: Dataset is empty."

        if not all(isinstance(x, (int, float)) for x in data):
            return "Error: Invalid dataset. Please provide numerical values."

        n = len(data)
```

```
            mean = sum(data) / n
            variance = sum((x - mean) ** 2 for x in data) / n
            std_dev = math.sqrt(variance)
            return {"variance": variance, "standard deviation": std_dev}

        except Exception as e:
            return f"An error occurred: {e}"

    # Example usage:
    data = [2, 4, 4, 4, 5, 5, 7, 9]
    result = calculate_variance_std(data)
    print(result)

    data2 = []
    result2 = calculate_variance_std(data2)
    print(result2)

    data3 = [1,2,'a',4]
    result3 = calculate_variance_std(data3)
    result3
```

```
{'variance': 4.0, 'standard deviation': 2.0}
Error: Dataset is empty.
'Error: Invalid dataset. Please provide numerical values.'
```

Q3. Create a dataset and classify it into nominal, ordinal, interval, and ratio types.

```
# prompt: Create a dataset and classify it into nominal, ordinal, interval, and ratio typ

import pandas as pd

def create_and_classify_dataset():
    """Creates a sample dataset and classifies its variables by data type."""
    data = {
        'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
        'Education Level': ['High School', 'Bachelor', 'Master', 'PhD', 'Bachelor'],
        'Temperature Preference': [20, 25, 18, 22, 24],
        'Happiness Score': [7, 5, 9, 6, 8],
        'Income': [50000, 70000, 90000, 120000, 65000],
        'Favorite Color': ['Blue', 'Red', 'Green', 'Blue', 'Yellow']
    }

    df = pd.DataFrame(data)

    # Classify data types
    data_types = {
        'Name': 'Nominal',
        'Education Level': 'Ordinal',  # Order exists (High School < Bachelor < Master <
        'Temperature Preference': 'Interval',  # Equal intervals, but no true zero
        'Happiness Score': 'Ordinal',  # Rankable, intervals not necessarily equal
```

```
        'Income': 'Ratio',  # Meaningful zero (0 income)
        'Favorite Color': 'Nominal'
    }

    # Return the DataFrame and data type classification
    return df, data_types


# Example usage:
df, data_types = create_and_classify_dataset()
print("Dataset:")
print(df)
print("\nData Type Classification:")
for column, data_type in data_types.items():
    print(f"{column}: {data_type}")
```

```
Dataset:
      Name Education Level  Temperature Preference  Happiness Score  Income  \
0    Alice     High School                      20                7   50000
1      Bob        Bachelor                      25                5   70000
2  Charlie          Master                      18                9   90000
3    David             PhD                      22                6  120000
4      Eve        Bachelor                      24                8   65000

  Favorite Color
0           Blue
1            Red
2          Green
3           Blue
4         Yellow

Data Type Classification:
Name: Nominal
Education Level: Ordinal
Temperature Preference: Interval
Happiness Score: Ordinal
Income: Ratio
Favorite Color: Nominal
```

Q4. Implement sampling techniques like random sampling and stratified sampling.

```
# prompt: Implement sampling techniques like random sampling and stratified sampling.

import random
import pandas as pd

def random_sampling(data, sample_size):
    """Performs random sampling on a dataset.

    Args:
        data: A list or pandas DataFrame representing the dataset.
```

```
            sample_size: The desired size of the sample.

        Returns:
            A new list or DataFrame containing the random sample.
            Returns an error message if the sample size is invalid.
        """
        n = len(data)
        if not 0 < sample_size <= n:
            return "Error: Invalid sample size."

        if isinstance(data, pd.DataFrame):
            return data.sample(n=sample_size, random_state=42) # random_state for reproducibi
        elif isinstance(data, list):
            return random.sample(data, sample_size)
        else:
            return "Error: Unsupported data type."


    def stratified_sampling(data, strata_column, sample_size_per_strata):
        """Performs stratified sampling on a dataset.

        Args:
          data: A pandas DataFrame representing the dataset.
          strata_column: The name of the column to stratify by.
          sample_size_per_strata: The desired sample size for each stratum.  Can be an intege

        Returns:
          A new DataFrame containing the stratified sample.
          Returns an error if input is invalid.
        """
        if not isinstance(data, pd.DataFrame):
            return "Error: Input data must be a pandas DataFrame."

        if not strata_column in data.columns:
            return f"Error: Strata column '{strata_column}' not found in DataFrame."

        strata = data[strata_column].unique()

        if isinstance(sample_size_per_strata, int):
          sample_size_per_strata = {s: sample_size_per_strata for s in strata} # Make it a di
        elif not isinstance(sample_size_per_strata, dict):
          return "Error: sample_size_per_strata must be an integer or a dictionary."

        samples = []
        for stratum in strata:
            stratum_data = data[data[strata_column] == stratum]
            n_stratum = len(stratum_data)

            if stratum not in sample_size_per_strata:
                return f"Error: Sample size not specified for stratum '{stratum}'."

            sample_size = min(sample_size_per_strata[stratum], n_stratum) # Handle strata sma

            if sample_size <= 0 :
                return f"Error: Sample size must be positive for stratum '{stratum}'."
```

```
            samples.append(stratum_data.sample(n=sample_size, random_state=42))

        return pd.concat(samples)

# Example usage (assuming df is created from Q3)
df, _ = create_and_classify_dataset()

# Random sampling
random_sample = random_sampling(df, 3)
print("\nRandom Sample:")
print(random_sample)

# Stratified sampling by education level
stratified_sample = stratified_sampling(df, 'Education Level', 1)  # Sample 1 from each e
print("\nStratified Sample by Education Level:")
print(stratified_sample)

stratified_sample2 = stratified_sampling(df, 'Education Level', {'High School': 1, 'Bache
print("\nStratified Sample by Education Level (uneven sample sizes):")
stratified_sample2
```

```
Random Sample:
      Name Education Level  Temperature Preference  Happiness Score  Income  \
1      Bob        Bachelor                      25                5   70000
4      Eve        Bachelor                      24                8   65000
2  Charlie          Master                      18                9   90000

  Favorite Color
1            Red
4         Yellow
2          Green

Stratified Sample by Education Level:
      Name Education Level  Temperature Preference  Happiness Score  Income  \
0    Alice     High School                      20                7   50000
4      Eve        Bachelor                      24                8   65000
2  Charlie          Master                      18                9   90000
3    David             PhD                      22                6  120000

  Favorite Color
0           Blue
4         Yellow
2          Green
3           Blue

Stratified Sample by Education Level (uneven sample sizes):
```

	Name	Education Level	Temperature Preference	Happiness Score	Income	Favorite Color
0	Alice	High School	20	7	50000	Blue
4	Eve	Bachelor	24	8	65000	Yellow
1	Bob	Bachelor	25	5	70000	Red
2	Charlie	Master	18	9	90000	Green
3	David	PhD	22	6	120000	Blue

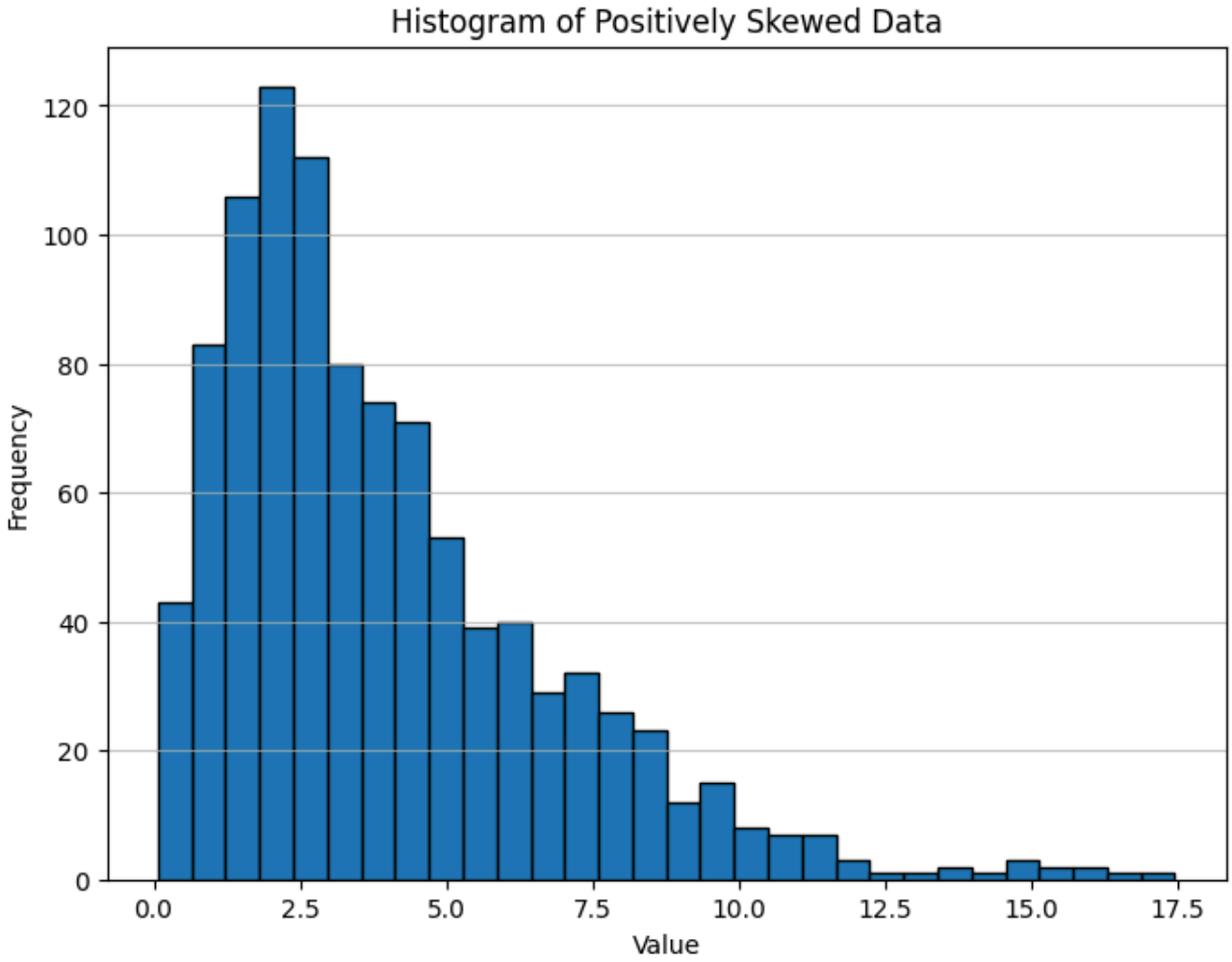
Q5. Write a Python function to calculate the range of a dataset.

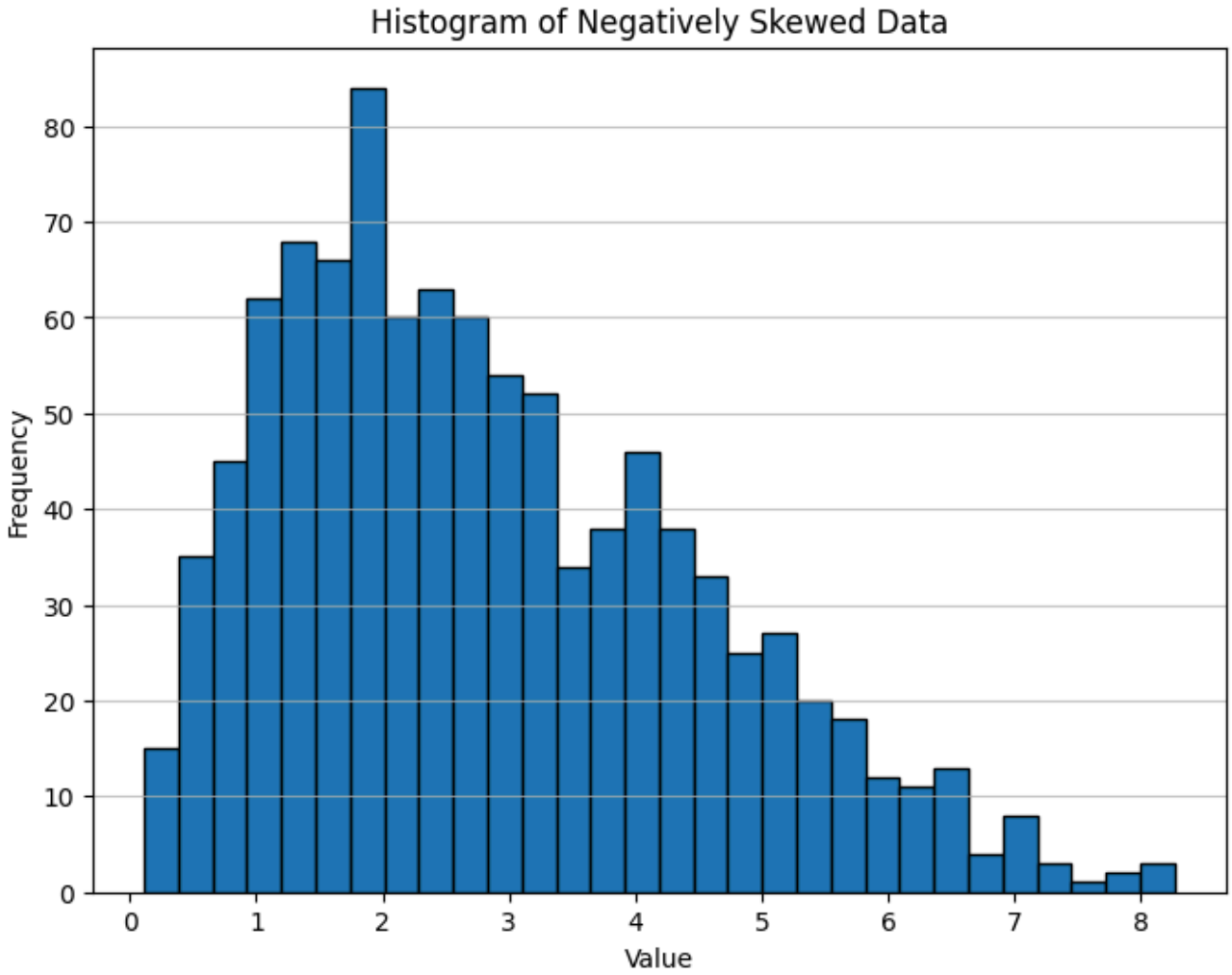
```
# prompt: Write a Python function to calculate the range of a dataset.

def calculate_range(data):
    """Calculates the range of a dataset.

    Args:
        data: A list of numerical values.

    Returns:
        The range of the dataset (difference between max and min).
        Returns an error message if the input is invalid or empty.
    """
    try:
        if not data:
            return "Error: Dataset is empty."
```

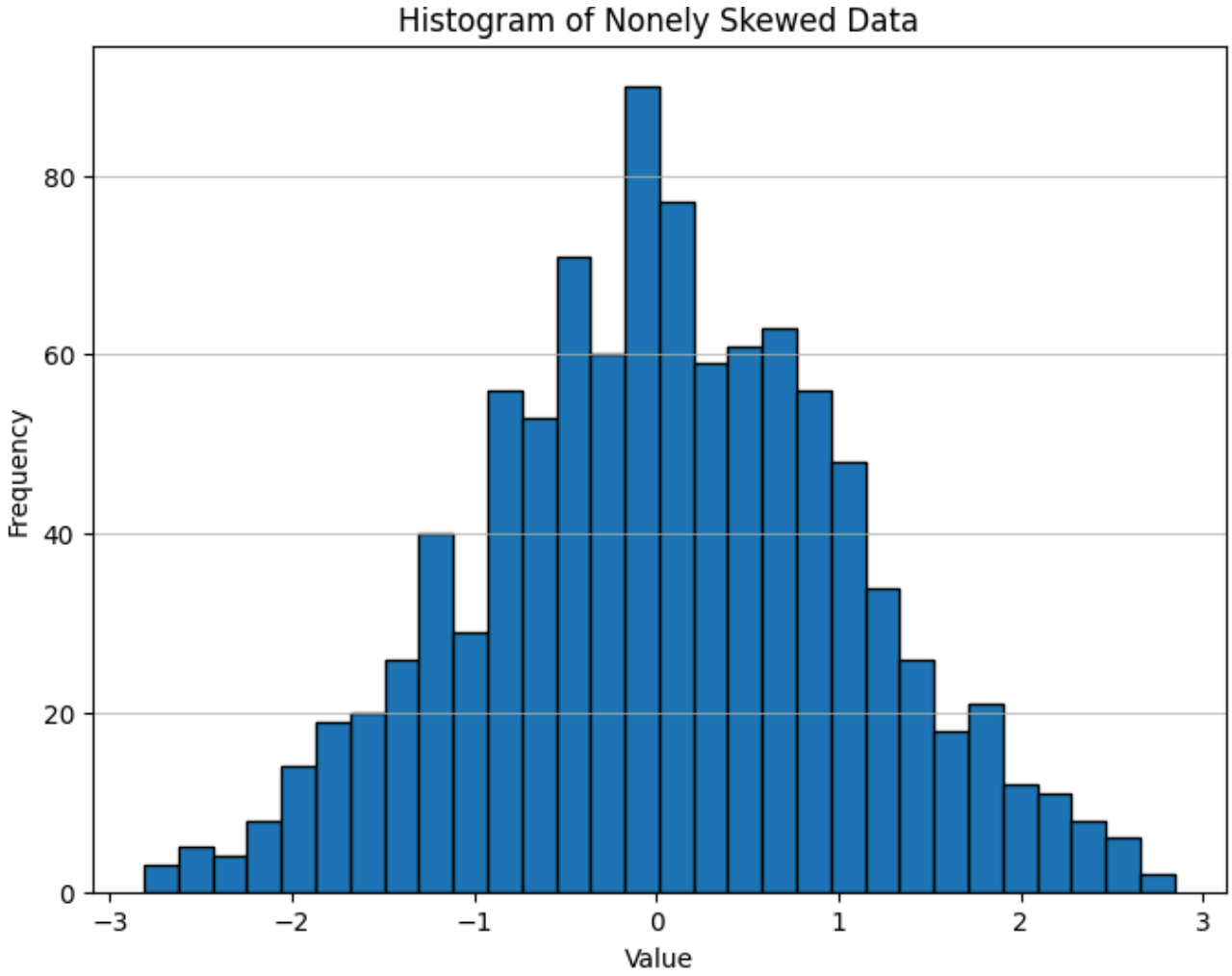
```
        if not all(isinstance(x, (int, float)) for x in data):
            return "Error: Invalid dataset. Please provide numerical values."

        data_range = max(data) - min(data)
        return data_range

    except Exception as e:
        return f"An error occurred: {e}"
```

Q6. Create a dataset and plot its histogram to visualize skewness.

```
# prompt: Create a dataset and plot its histogram to visualize skewness.

import matplotlib.pyplot as plt
import numpy as np

def create_and_plot_skewed_dataset(skew_type="positive", num_points=1000):
    """
    Creates a dataset with specified skewness and plots its histogram.

    Args:
        skew_type: Type of skewness ("positive", "negative", or "none"). Defaults to "pos
        num_points: Number of data points in the dataset. Defaults to 1000.
    """

    if skew_type == "positive":
        data = np.random.gamma(2, 2, num_points)  # Example: Gamma distribution
    elif skew_type == "negative":
        data = np.random.beta(2, 5, num_points) * 10 # Example: Beta distribution
    elif skew_type == "none":  # Approximately normal
        data = np.random.normal(0, 1, num_points)
    else:
        raise ValueError("Invalid skew_type. Choose 'positive', 'negative', or 'none'.")

    plt.figure(figsize=(8, 6))
    plt.hist(data, bins=30, edgecolor='black')  # Adjust the number of bins as needed
    plt.xlabel("Value")
    plt.ylabel("Frequency")
    plt.title(f"Histogram of {skew_type.title()}ly Skewed Data")
    plt.grid(axis='y', alpha=0.75)
    plt.show()

# Example usage:
create_and_plot_skewed_dataset(skew_type="positive")
create_and_plot_skewed_dataset(skew_type="negative")
create_and_plot_skewed_dataset(skew_type="none")
```


Histogram of Positively Skewed Data
Frequency
Value
0
20
40
60
80
100
120
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Histogram of Negatively Skewed Data
Frequency
Value
0
10
20
30
40
50
60
70
80
0
1
2
3
4
5
6
7
8


Histogram of Nonely Skewed Data
Frequency
Value
0
20
40
60
80
−3
−2
−1
0
1
2
3

Q7. Calculate skewness and kurtosis of a dataset using Python libraries.

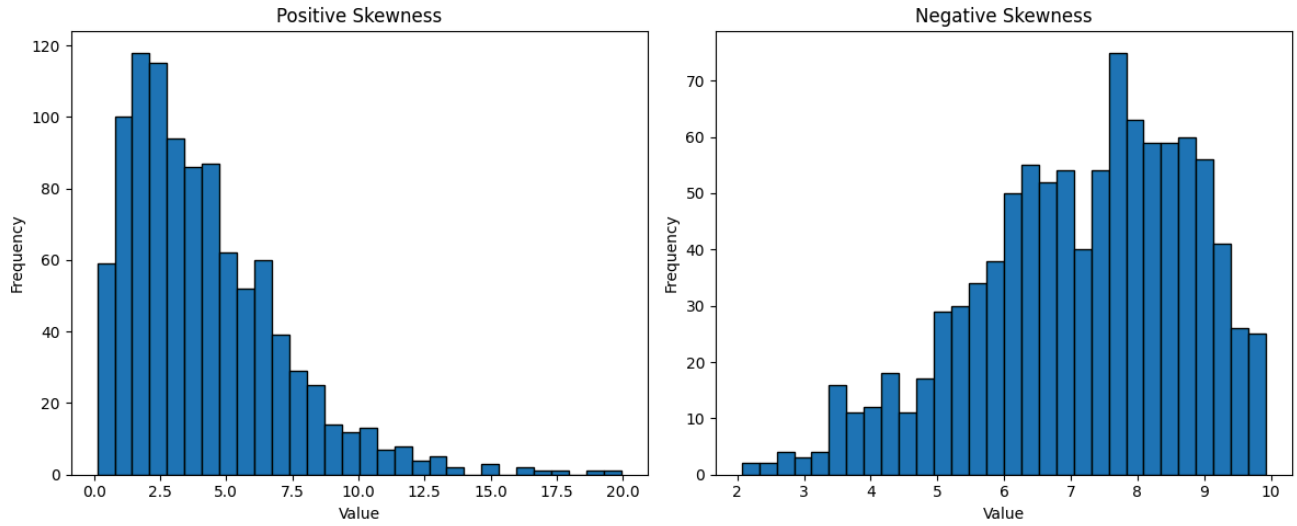
```
# prompt: Calculate skewness and kurtosis of a dataset using Python libraries.

import scipy.stats as stats

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

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

    Returns:
        A dictionary containing the skewness and kurtosis, or an error message
        if the input is invalid.
    """
    try:
        if not data:
            return "Error: Dataset is empty."

        if not all(isinstance(x, (int, float)) for x in data):
            return "Error: Invalid dataset. Please provide numerical values."

        skewness = stats.skew(data)
        kurtosis = stats.kurtosis(data)  # Fisher's definition (excess kurtosis)

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

    except Exception as e:
        return f"An error occurred: {e}"

# Example usage:
data = [1, 2, 2, 2, 3, 3, 4, 4, 5, 5, 5, 5, 6, 7, 8]
result = calculate_skewness_kurtosis(data)
result
```

```
{'skewness': np.float64(0.25757262272907716),
 'kurtosis': np.float64(-0.7434239142876775)}
```

Q8. Generate a dataset and demonstrate positive and negative skewness.

```
# prompt: Generate a dataset and demonstrate positive and negative skewness.

def generate_and_demonstrate_skewness(num_points=1000):
    """Generates datasets and demonstrates positive and negative skewness."""
```

```
    # Positive Skewness
    positive_skew_data = np.random.gamma(2, 2, num_points)

    # Negative Skewness
    negative_skew_data = np.random.beta(5, 2, num_points) * 10  # Adjust scaling as neede


    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.hist(positive_skew_data, bins=30, edgecolor='black')
    plt.title('Positive Skewness')
    plt.xlabel('Value')
    plt.ylabel('Frequency')


    plt.subplot(1, 2, 2)
    plt.hist(negative_skew_data, bins=30, edgecolor='black')
    plt.title('Negative Skewness')
    plt.xlabel('Value')
    plt.ylabel('Frequency')

    plt.tight_layout()
    plt.show()

generate_and_demonstrate_skewness()
```



Q9. Write a Python script to calculate covariance between two datasets.

```
# prompt: Write a Python script to calculate covariance between two datasets.

def calculate_covariance(dataset1, dataset2):
    """Calculates the covariance between two datasets.

    Args:
        dataset1: A list or numpy array of numerical values.
        dataset2: A list or numpy array of numerical values.

    Returns:
        The covariance between the two datasets. Returns an error message if the input is
    """
    try:
        if len(dataset1) != len(dataset2):
            return "Error: Datasets must have the same length."

        if not all(isinstance(x, (int, float)) for x in dataset1) or not all(isinstance(x
          return "Error: Invalid dataset. Please provide numerical values."

        n = len(dataset1)
        mean1 = sum(dataset1) / n
```

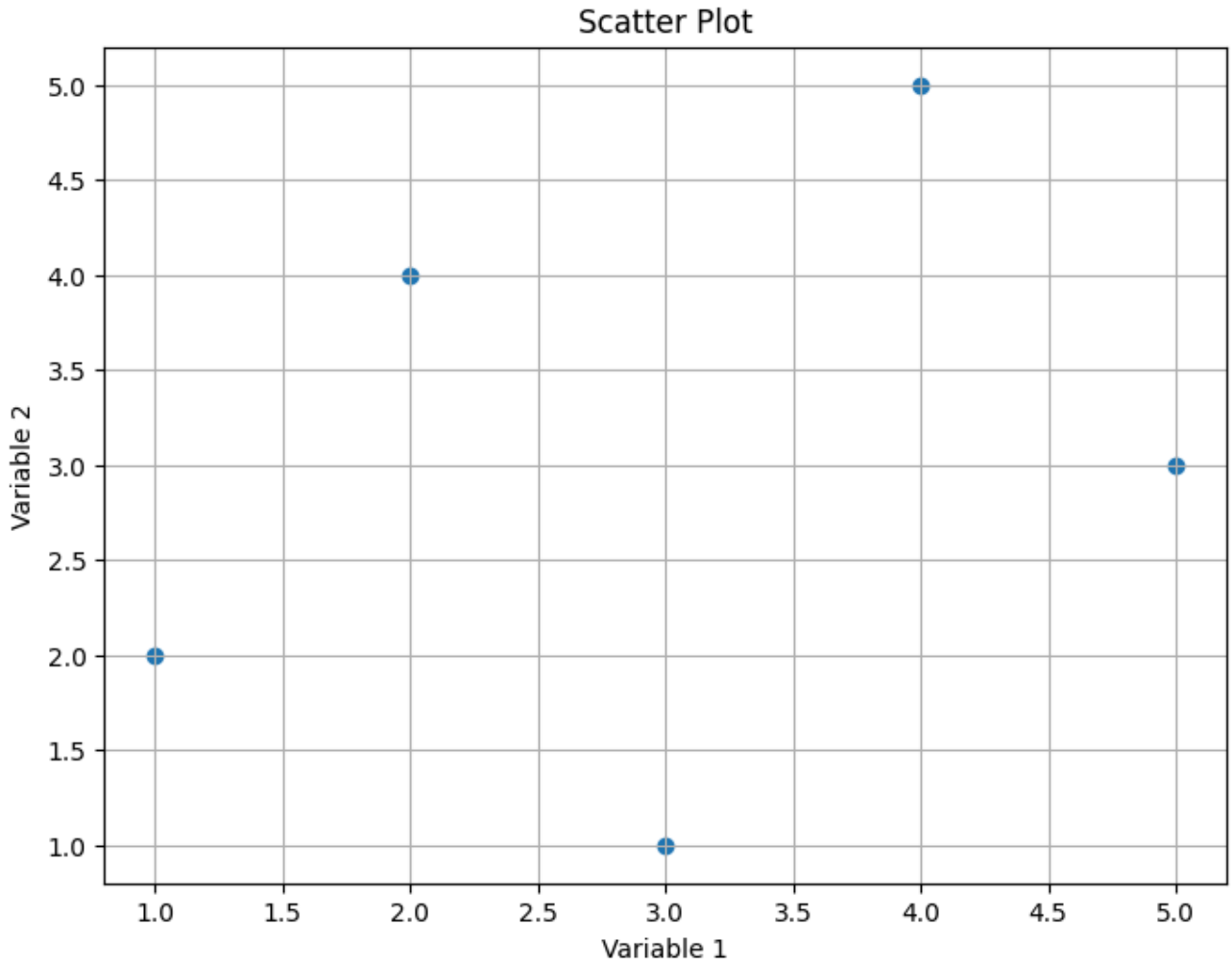
```
        mean2 = sum(dataset2) / n

        covariance = sum((dataset1[i] - mean1) * (dataset2[i] - mean2) for i in range(n))
        return covariance
    except Exception as e:
        return f"An error occurred: {e}"
```

Q10. Write a Python script to calculate the correlation coefficient between two datasets.

```
# prompt: Write a Python script to calculate the correlation coefficient between two data

def calculate_correlation(dataset1, dataset2):
    """Calculates the correlation coefficient between two datasets.

    Args:
        dataset1: A list or numpy array of numerical values.
        dataset2: A list or numpy array of numerical values.

    Returns:
        The correlation coefficient between the two datasets.
        Returns an error message if the input is invalid.
    """
    try:
        if len(dataset1) != len(dataset2):
            return "Error: Datasets must have the same length."

        if not all(isinstance(x, (int, float)) for x in dataset1) or not all(isinstance(x
          return "Error: Invalid dataset. Please provide numerical values."

        n = len(dataset1)
        mean1 = sum(dataset1) / n
        mean2 = sum(dataset2) / n

        covariance = sum((dataset1[i] - mean1) * (dataset2[i] - mean2) for i in range(n))

        std_dev1 = math.sqrt(sum((x - mean1) ** 2 for x in dataset1) / n)
        std_dev2 = math.sqrt(sum((x - mean2) ** 2 for x in dataset2) / n)

        if std_dev1 == 0 or std_dev2 == 0:
            return "Error: Cannot calculate correlation, one of the datasets has zero sta

        correlation = covariance / (std_dev1 * std_dev2)
        return correlation

    except Exception as e:
        return f"An error occurred: {e}"
```

Q11. Create a scatter plot to visualize the relationship between two variables.

```
# prompt: Create a scatter plot to visualize the relationship between two variables.

import matplotlib.pyplot as plt

def create_scatter_plot(dataset1, dataset2, xlabel="Variable 1", ylabel="Variable 2", tit
    """
    Creates a scatter plot to visualize the relationship between two variables.

    Args:
        dataset1: A list or numpy array representing the x-axis values.
        dataset2: A list or numpy array representing the y-axis values.
        xlabel: Label for the x-axis. Defaults to "Variable 1".
        ylabel: Label for the y-axis. Defaults to "Variable 2".
        title: Title of the plot. Defaults to "Scatter Plot".
    """
    try:
        if len(dataset1) != len(dataset2):
            return "Error: Datasets must have the same length."

        plt.figure(figsize=(8, 6))
        plt.scatter(dataset1, dataset2)
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)
        plt.title(title)
        plt.grid(True)
        plt.show()
    except Exception as e:
        return f"An error occurred: {e}"

# Example usage:
data1 = [1, 2, 3, 4, 5]
data2 = [2, 4, 1, 5, 3]

create_scatter_plot(data1, data2)
```



Q12. Implement and compare single random sampling and systematic sampling.

```
# prompt: Implement and compare single random sampling and systematic sampling.

import pandas as pd
import random

def systematic_sampling(data, k):
    """
    Performs systematic sampling on a dataset.

    Args:
        data: A pandas DataFrame or a list.
        k: The sampling interval.

    Returns:
        A new DataFrame or list containing the systematic sample.
        Returns an error message if the input is invalid.
    """
    if isinstance(data, pd.DataFrame):
```

```
            n = len(data)
            indices = random.sample(range(k), 1)  # Random start index
            systematic_sample = data.iloc[indices[0]::k]  # Select every kth element
            return systematic_sample
        elif isinstance(data, list):
            n = len(data)
            indices = random.sample(range(k), 1)
            return data[indices[0]::k]
        else:
            return "Error: Input data must be a pandas DataFrame or a list."


    def compare_sampling_techniques(data, sample_size):
        """
        Compares random sampling and systematic sampling.
        """

        # Random Sampling
        random_sample = random_sampling(data, sample_size)  # Assuming random_sampling is def


        # Systematic Sampling (choose k such that it approximates the sample size)
        k = len(data) // sample_size
        systematic_sample = systematic_sampling(data, k)

        print("Random Sample:")
        print(random_sample)

        print("\nSystematic Sample:")
        print(systematic_sample)


    # Example usage
    # Create a sample DataFrame (replace with your data)
    data = pd.DataFrame({'value': range(100)})

    # Set the desired sample size
    sample_size = 10

    # Compare the sampling techniques
    compare_sampling_techniques(data, sample_size)
```

```
Random Sample:
    value
83     83
53     53
70     70
45     45
44     44
39     39
22     22
80     80
10     10
0       0
```

```
Systematic Sample:
    value
2       2
12     12
22     22
32     32
42     42
52     52
62     62
72     72
82     82
92     92
```

Q13. Calculate the mean, median, and mode of grouped data.

```
# prompt: Calculate the mean, median, and mode of grouped data.

import pandas as pd

def calculate_grouped_stats(data, lower_bounds, upper_bounds, frequencies):
    """Calculates the mean, median, and mode of grouped data.

    Args:
        data: A pandas DataFrame or a list representing the dataset.
        lower_bounds: A list of the lower bounds of each group.
        upper_bounds: A list of the upper bounds of each group.
        frequencies: A list of the frequencies for each group.

    Returns:
        A dictionary containing the mean, median, and mode of the grouped data.
        Returns an error if input is invalid.
    """
    if not isinstance(lower_bounds, list) or not isinstance(upper_bounds, list) or not is
        return "Error: lower_bounds, upper_bounds, and frequencies must be lists."

    if len(lower_bounds) != len(upper_bounds) or len(lower_bounds) != len(frequencies):
        return "Error: Input lists must have the same length."

    if len(lower_bounds) == 0 :
        return "Error: Input list cannot be empty."

    try:
        midpoints = [(lower_bounds[i] + upper_bounds[i]) / 2 for i in range(len(lower_bou

        # Calculate the mean
        mean = sum(midpoints[i] * frequencies[i] for i in range(len(midpoints))) / sum(fr

        # Calculate the cumulative frequencies
        cumulative_frequencies = [sum(frequencies[:i+1]) for i in range(len(frequencies))
        N = cumulative_frequencies[-1]
        median_index = (N + 1) / 2

        # Find the group containing the median
```

```
            median_group_index = 0
            for i in range(len(cumulative_frequencies)):
                if cumulative_frequencies[i] >= median_index:
                    median_group_index = i
                    break

            # Calculate the median using linear interpolation
            median = lower_bounds[median_group_index] + ((median_index - cumulative_frequenci

            #Calculate the mode
            mode_group_index = frequencies.index(max(frequencies))
            mode = midpoints[mode_group_index]

            return {"mean": mean, "median": median, "mode": mode}
        except (IndexError, ValueError, TypeError) as e:
            return f"An error occurred during calculations: {e}"


    # Example usage:

    lower_bounds = [10, 20, 30, 40, 50]
    upper_bounds = [19, 29, 39, 49, 59]
    frequencies = [5, 10, 15, 12, 8]
    result = calculate_grouped_stats(None, lower_bounds, upper_bounds, frequencies)
    result
```

```
{'mean': 36.1, 'median': 36.3, 'mode': 34.5}
```

Q14. Simulate data using Python and calculate its central tendency and dispersion.

```
# prompt: Simulate data using Python and calculate its central tendency and dispersion.

import numpy as np

def simulate_and_analyze_data(distribution='normal', num_samples=1000):
    """Simulates data from a specified distribution and calculates central tendency and d

    Args:
        distribution: The type of distribution ('normal', 'uniform', 'exponential').
        num_samples: The number of data points to generate.

    Returns:
        A dictionary containing the simulated data, mean, median, mode (if applicable),
        variance, standard deviation, range, and skewness. Returns an error message if th
        distribution type is not supported.
    """
    try:
        if distribution == 'normal':
            data = np.random.normal(loc=0, scale=1, size=num_samples)
```

```
            elif distribution == 'uniform':
                data = np.random.uniform(low=0, high=1, size=num_samples)
            elif distribution == 'exponential':
                data = np.random.exponential(scale=1, size=num_samples)
            else:
                return "Error: Distribution not supported."

            mean = np.mean(data)
            median = np.median(data)
            # Mode calculation for continuous data (approximate) - find the value with max fr
            from scipy.stats import mode
            mode_val = mode(np.round(data))[0][0] #rounding might need to be adjusted
            variance = np.var(data)
            std_dev = np.std(data)
            data_range = np.max(data) - np.min(data)
            skewness = stats.skew(data)

            return {
                "data": data,
                "mean": mean,
                "median": median,
                "mode": mode_val,
                "variance": variance,
                "standard deviation": std_dev,
                "range": data_range,
                "skewness": skewness
            }

        except Exception as e:
            return f"An error occurred: {e}"


    # Example usage
    normal_stats = simulate_and_analyze_data(distribution='normal')
    uniform_stats = simulate_and_analyze_data(distribution='uniform')
    exponential_stats = simulate_and_analyze_data(distribution='exponential')

    print("Normal Distribution Stats:")
    print(normal_stats)

    print("\nUniform Distribution Stats:")
    print(uniform_stats)

    print("\nExponential Distribution Stats:")
    exponential_stats
```

```
Normal Distribution Stats:
An error occurred: invalid index to scalar variable.

Uniform Distribution Stats:
An error occurred: invalid index to scalar variable.

Exponential Distribution Stats:
'An error occurred: invalid index to scalar variable.'
```

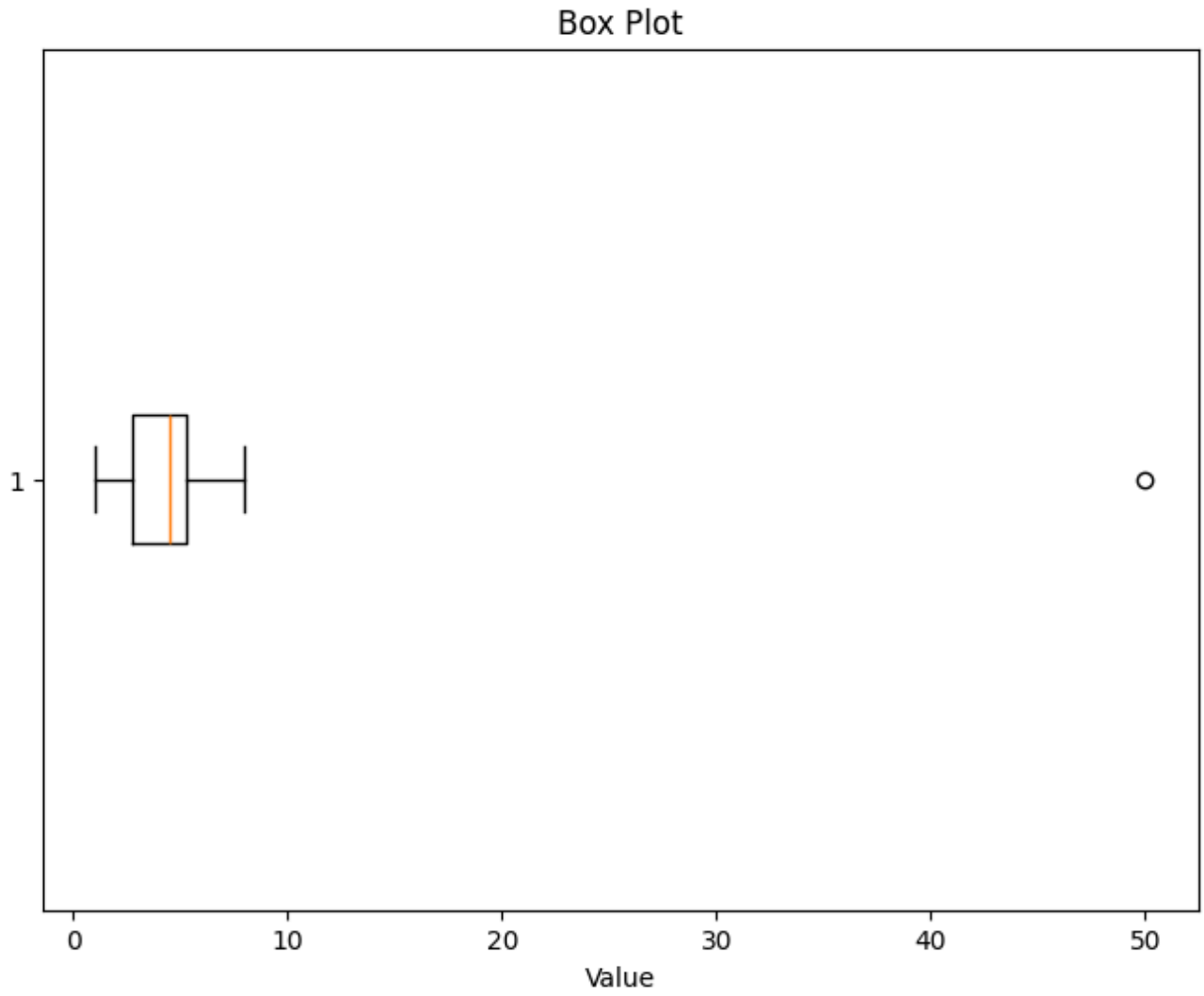
Q15. Use NumPy and pandas to summarize a dataset's descriptive statistics.

```
# prompt: Use NumPy and pandas to summarize a dataset's descriptive statistics.

import pandas as pd
import numpy as np

def summarize_dataset(data):
    """Summarizes a dataset's descriptive statistics using NumPy and pandas.

    Args:
        data: A list or numpy array of numerical values, or a pandas DataFrame.

    Returns:
        A pandas DataFrame containing descriptive statistics, or an error message if the
    """
    try:
        if isinstance(data, (list, np.ndarray)):
            df = pd.DataFrame({'value': data})  # Convert to DataFrame
        elif isinstance(data, pd.DataFrame):
            df = data  # Use DataFrame directly
        else:
            return "Error: Invalid data type. Input must be a list, numpy array, or panda

        if df.empty:
          return "Error: DataFrame is empty."

        # Numerical summaries using pandas describe()
        summary = df.describe()

        # Add other statistics (e.g., skewness, kurtosis)
        summary.loc['skewness'] = df['value'].skew()  # Assuming 'value' column for DataF
        summary.loc['kurtosis'] = df['value'].kurt()


        return summary

    except (KeyError, TypeError) as e:  # Handle specific potential errors
        return f"An error occurred during calculations: {e}"

# Example usage:
data = [1, 2, 2, 2, 3, 3, 4, 4, 5, 5, 5, 5, 6, 7, 8]
summary = summarize_dataset(data)
summary
```

	value
count	15.000000
mean	4.133333
std	1.995232
min	1.000000
25%	2.500000
50%	4.000000
75%	5.000000
max	8.000000
skewness	0.287122
kurtosis	-0.529019

Q16. Plot a boxplot to understand the spread and identify outliers.

```
# prompt: plot a boxplot to understand the spread and identify outliers

import matplotlib.pyplot as plt

def plot_boxplot(data, title="Box Plot"):
    """Plots a boxplot to visualize data distribution and identify outliers.

    Args:
        data: A list or numpy array of numerical values.
        title: Title of the plot.
    """
    try:
        plt.figure(figsize=(8, 6))
        plt.boxplot(data, vert=False) #vert=False for horizontal boxplot
        plt.title(title)
        plt.xlabel("Value")
        plt.show()
    except Exception as e:
        print(f"An error occurred: {e}")


# Example usage
data = [1, 2, 2, 2, 3, 3, 4, 4, 5, 5, 5, 5, 6, 7, 8, 50]  # Added an outlier for demonstr
plot_boxplot(data)
```



Q17. Calculate the interquartile range (IQR) of a dataset.

```
# prompt: Calculate the interquartile range (IQR) of a dataset.

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

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

    Returns:
        The IQR of the dataset.
        Returns an error message if the input is invalid.
    """
    try:
        if not data:
            return "Error: Dataset is empty."

        if not all(isinstance(x, (int, float)) for x in data):
            return "Error: Invalid dataset. Please provide numerical values."

        q1 = np.percentile(data, 25)
```

```
            q3 = np.percentile(data, 75)
            iqr = q3 - q1
            return iqr
        except Exception as e:
            return f"An error occurred: {e}"

    # Example usage:
    data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    iqr = calculate_iqr(data)
    print(f"The IQR of the dataset is: {iqr}")
```

```
The IQR of the dataset is: 4.5
```

Q18. Implement Z-score normalization and explain its significance.

```
# prompt: Implement Z-score normalization and explain its significance.

import numpy as np

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

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

    Returns:
        A numpy array containing the normalized data.
        Returns an error message if the input is invalid.
    """
    try:
        if not data:
            return "Error: Dataset is empty."

        if not all(isinstance(x, (int, float)) for x in data):
            return "Error: Invalid dataset. Please provide numerical values."

        data = np.array(data)  # Convert to numpy array for efficient calculations
        mean = np.mean(data)
        std_dev = np.std(data)

        if std_dev == 0:
            return "Error: Standard deviation is zero, cannot normalize."

        normalized_data = (data - mean) / std_dev
        return normalized_data

    except Exception as e:
        return f"An error occurred: {e}"
```

```
# Example usage
data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
normalized_data = zscore_normalize(data)
normalized_data


# Significance of Z-score normalization:
# 1. Feature Scaling: It transforms data to have a mean of 0 and a standard deviation of
# 2. Algorithm Performance: Many machine learning algorithms are sensitive to the scale o
# 3. Outlier Detection: Z-scores can highlight outliers as values that are significantly
# 4. Data Comparison: After normalization, data from different distributions or with diff
# 5. Statistical Analysis: In statistical analysis, Z-scores are used for hypothesis test
```

```
array([-1.5666989 , -1.21854359, -0.87038828, -0.52223297, -0.17407766,
        0.17407766,  0.52223297,  0.87038828,  1.21854359,  1.5666989 ])
```

Q19. Compare two datasets using their standard deviations.

```
# prompt: Compare two datasets using their standard deviations.

import numpy as np

def compare_datasets_by_std(dataset1, dataset2):
    """Compares two datasets based on their standard deviations.

    Args:
        dataset1: A list or numpy array of numerical values.
        dataset2: A list or numpy array of numerical values.

    Returns:
        A string describing the comparison, or an error message if inputs are invalid.
    """
    try:
        std_dev1 = np.std(dataset1)
        std_dev2 = np.std(dataset2)

        if std_dev1 > std_dev2:
            return "Dataset 1 has a larger standard deviation than Dataset 2."
        elif std_dev2 > std_dev1:
            return "Dataset 2 has a larger standard deviation than Dataset 1."
        else:
            return "Dataset 1 and Dataset 2 have equal standard deviations."

    except (TypeError, ValueError) as e:
        return f"An error occurred: {e}"


# Example usage
dataset1 = [1, 2, 3, 4, 5]
dataset2 = [1, 1, 1, 10, 10]  # Higher variability
```

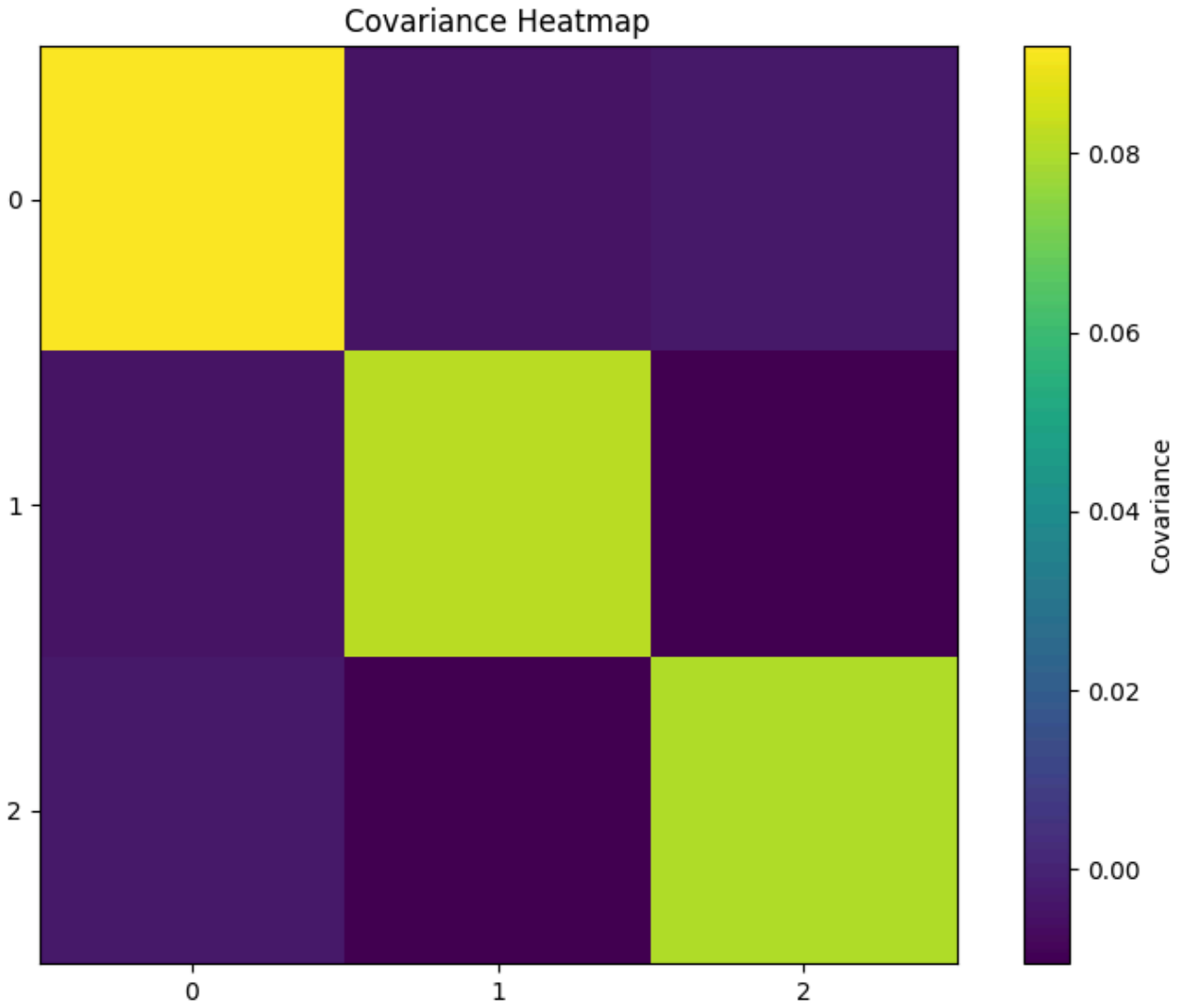
```
comparison_result = compare_datasets_by_std(dataset1, dataset2)
comparison_result
```

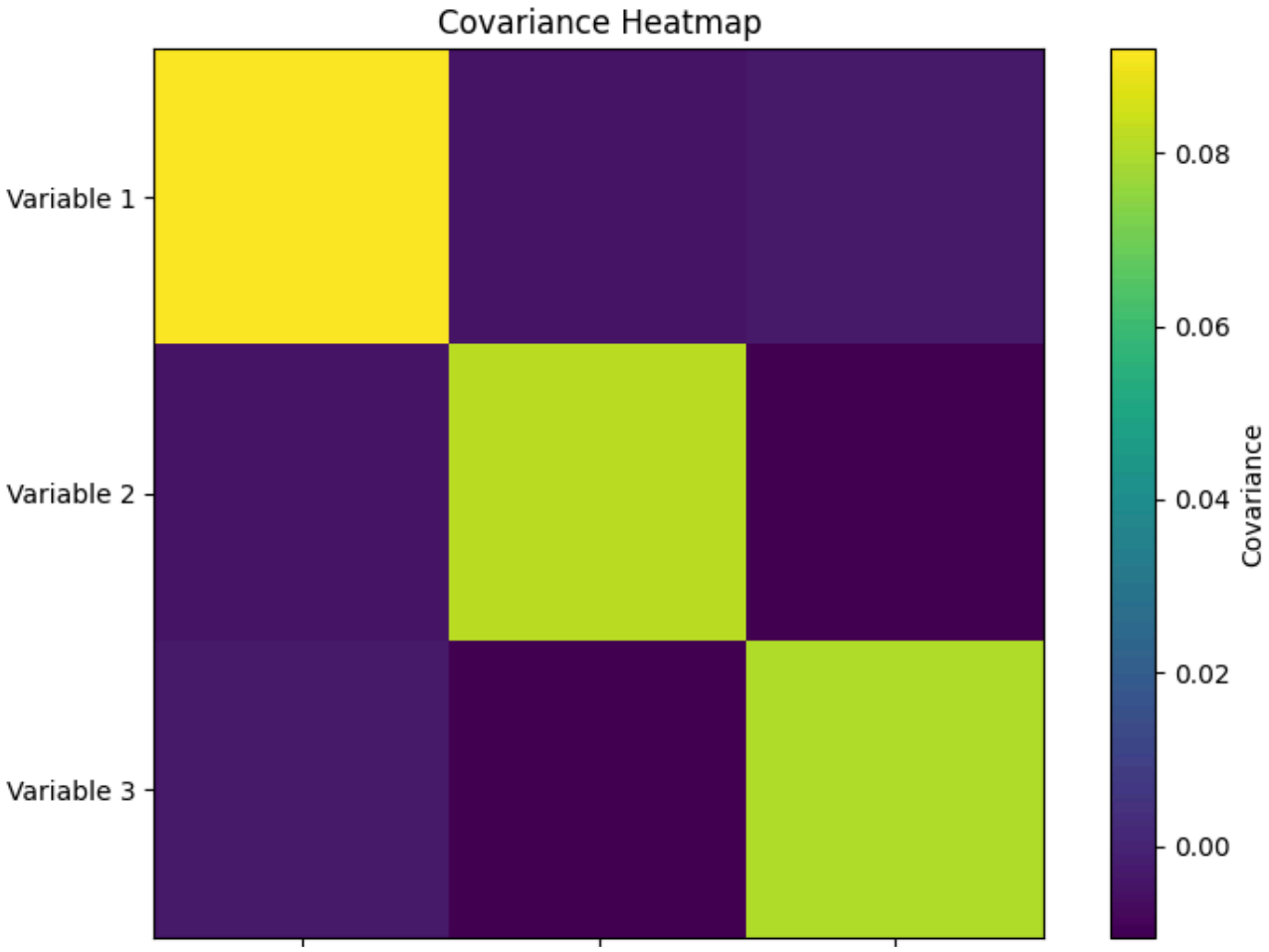
```
'Dataset 2 has a larger standard deviation than Dataset 1.'
```

Q20. Write a Python program to visualize covariance using a heatmap.

```
# prompt: Write a Python program to visualize covariance using a heatmap.

import matplotlib.pyplot as plt
import numpy as np

def visualize_covariance_heatmap(data):
    """Visualizes the covariance matrix of a dataset using a heatmap.

    Args:
      data: A NumPy array or pandas DataFrame where each column represents a variable.
    """
    try:
        # Calculate the covariance matrix
        covariance_matrix = np.cov(data, rowvar=False)  # rowvar=False for columns as var

        # Create the heatmap
        plt.figure(figsize=(8, 6))
        plt.imshow(covariance_matrix, cmap='viridis', interpolation='nearest')
        plt.colorbar(label='Covariance')
        plt.title('Covariance Heatmap')

        # Add variable labels (optional)
        if isinstance(data, pd.DataFrame):
            variable_names = data.columns
            plt.xticks(range(len(variable_names)), variable_names, rotation=45, ha='right
            plt.yticks(range(len(variable_names)), variable_names)
        else:
            num_variables = data.shape[1]
            plt.xticks(range(num_variables))
            plt.yticks(range(num_variables))

        plt.tight_layout() # Adjust layout for better label visibility
        plt.show()

    except Exception as e:
        print(f"An error occurred: {e}")

# Example usage (replace with your data)
# Sample data (replace with your actual dataset)
data = np.random.rand(100, 3)  # 100 samples, 3 variables
visualize_covariance_heatmap(data)
```

```
# Example with pandas DataFrame
import pandas as pd
data_df = pd.DataFrame(data, columns=['Variable 1', 'Variable 2', 'Variable 3'])
visualize_covariance_heatmap(data_df)
```


Covariance Heatmap
0
1
2
0
1
2
0.08
0.06
0.04
0.02
0.00
Covariance


Covariance Heatmap
Variable 1
Variable 2
Variable 3
0.08
0.06
0.04
0.02
0.00
Covariance

Q21. Use Seaborn to create a correlation matrix for a dataset.

```
# prompt: Use Seaborn to create a correlation matrix for a dataset.

import seaborn as sns
import matplotlib.pyplot as plt

def create_correlation_matrix(data):
    """
    Creates a correlation matrix heatmap using Seaborn.

    Args:
        data: A pandas DataFrame.
    """
    try:
        plt.figure(figsize=(10, 8))
        correlation_matrix = data.corr()  # Calculate the correlation matrix
        sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
        plt.title("Correlation Matrix")
        plt.show()
    except Exception as e:
        print(f"An error occurred: {e}")

# Example usage (replace with your actual DataFrame)
# Assuming 'df' is the DataFrame from the previous code blocks
# create_correlation_matrix(df)
```

Q22. Generate a dataset and implement both variance and standard deviation computations.

```
# prompt: Generate a dataset and implement both variance and standard deviation computati

import numpy as np

def calculate_variance_std(data):
    """Calculates the variance and standard deviation of a dataset."""
    variance = np.var(data)
    std_dev = np.std(data)
    return variance, std_dev
```

```
# Generate a sample dataset
data = np.random.rand(100)  # Example: 100 random numbers between 0 and 1

# Calculate variance and standard deviation
variance, std_dev = calculate_variance_std(data)

print("Variance:", variance)
print("Standard Deviation:", std_dev)
```

```
Variance: 0.07510221108737988
Standard Deviation: 0.27404782627742164
```

Q23. Visualize skewness and kurtosis using Python libraries like matplotlib or seaborn.

```
# prompt: Visualize skewness and kurtosis using Python libraries like matplotlib or seabo

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