COMPREHENSIVE PROBABILITY AND STATISTICS ANALYSIS APPLICATION - SUMMER 2023

ABSTRACT

This extensive document presents a detailed analysis of the "Comprehensive Probability and Statistics Analysis Application" developed as a challenging semester project for the Summer 2023 term. The application is meticulously designed to provide users with a comprehensive suite of statistical analysis tools, graphical representations, and regression modeling capabilities. It empowers users to explore datasets, make data-driven decisions, and draw meaningful insights from the data. The project was undertaken by a dedicated team of three members, led by Abdul Rehman, and the team named “10/10 Absolutes”

Abdul Rehman (21F - 9507)
Ahmad Chatta (21F - 9420)
Umer Basra 9436 (21F - 9436)
Probability and Statistics

Contents

1. Introduction

1.1 Background

The "Comprehensive Probability and Statistics Analysis Application" is the outcome of a semester-long project undertaken by a team of four members as part of their academic curriculum for the Summer 2023 term. The project aimed to develop an efficient and user-friendly application that encompasses a wide array of statistical analysis tools, empowering users to delve into datasets, perform statistical computations, and visualize data for better decision-making.

1.2 Project Objectives

The primary objectives of the project were as follows:

1. Design and develop a statistical analysis application with an intuitive user interface.

2. Implement various statistical techniques, probability distributions, and regression modeling.

3. Enable users to import, manipulate, and visualize data from various sources.

4. Provide descriptive statistical measures, including mean, median, mode, standard deviation, and variance.

5. Offer probability methods and distribution calculations for binomial, Poisson, and hypergeometric distributions.

6. Facilitate permutations and combinations calculations using different techniques.

7. Allow users to explore statistical relationships, including covariance and correlation.

8. Incorporate linear regression modeling to predict outcomes based on independent variables.

9. Generate graphical and tabular representations of data for better visualization and analysis.

10. Produce a comprehensive and user-friendly user guide to assist users in utilizing the application effectively.

1.3 Team Overview

The project was executed by a dynamic team of four members, each contributing their unique expertise and skills to achieve the project's goals. The team members and their roles are as follows:

1. Abdul Rehman - Project Leader and Frontend Developer

2. Ahmad Chatta - Statistician and Data Analyst

3. Umer Basra - Quality Assurance and Documentation Specialist

1.4 Project Scope

The scope of the project encompassed the development of a fully functional statistical analysis application that covers a wide range of statistical techniques, probability distributions, and regression modeling. The application was intended to cater to researchers, data analysts, and students seeking a comprehensive tool to analyze datasets and derive meaningful insights.

1.5 Project Timeline

The project timeline spanned the entire Summer 2023 term, divided into multiple phases, each focused on specific tasks and milestones. The main phases of the project included:

1. Planning and Research (Week 1-2): The team conducted extensive research on statistical analysis methods, libraries, and best practices to create a well-informed project plan.

2. **Application Design (Week 3-4):** The team collaborated to design the user interface, application architecture, and data handling mechanisms.

3. **Development (Week 5-10):** The development phase involved coding the application's functionality, implementing statistical computations, and integrating libraries for data visualization.

4. **Testing and Debugging (Week 11-12):** The application underwent rigorous testing to identify and fix any bugs or issues before the final submission.

5. **Documentation and Report Writing (Week 13-14):** The team documented the entire development process, including detailed explanations of each function and statistical method used.

6. **Submission and Presentation (Week 15):** The final application, documentation, and report were submitted as part of the project assessment. The team also prepared for the viva presentation.

2. Technologies Used

The project utilized a combination of programming languages and libraries to build the "Comprehensive Probability and Statistics Analysis Application."

2.1 Programming Language

Python, a widely used programming language known for its versatility and simplicity, was chosen as the primary language for this project. Python's extensive library support, especially for data analysis and statistical computation, made it an ideal choice for the application.

2.2 Libraries and Dependencies

The following libraries were crucial to implementing various statistical functionalities and visualization in the application:

2.2.1 NumPy

NumPy, short for "Numerical Python," is a fundamental library for numerical and matrix operations in Python. It provides support for large, multi-dimensional arrays and matrices, along with a vast collection of high-level mathematical functions. NumPy's array processing capabilities played a crucial role in handling datasets and performing statistical computations efficiently.

2.2.2 Matplotlib

Matplotlib is a powerful plotting library in Python that provides numerous tools for creating static, interactive, and animated visualizations. The application leveraged Matplotlib to generate graphical representations, including bar plots, histograms, scatter plots, and regression lines, making it easier for users to interpret data.

2.2.3 SciPy

SciPy, built on top of NumPy, is a comprehensive library for scientific and technical computing. It offers a plethora of mathematical algorithms and functions, including optimization, integration, interpolation, and statistical methods. The application used SciPy to perform probability distribution calculations, such as the binomial, Poisson, and hypergeometric distributions.

2.2.4 Math

The Math library is a built-in Python module that provides essential mathematical functions and constants. The application utilized Math for various mathematical operations, such as factorials, permutations, and combinations.

2.2.5 Stats models

Stats models is a powerful library for statistical modeling and hypothesis testing in Python. It provides a wide range of statistical models and functions for regression analysis, time-series analysis, and more. The application employed Stats models for linear regression modeling, allowing users to analyze relationships between variables and make predictions.

2.2.6 Pandas

Pandas is a versatile library for data manipulation and analysis in Python. It provides data structures such as Data Frames and Series, enabling users to work with labeled and structured data effectively. The application utilized Pandas for importing, preprocessing, and handling datasets, making it easier for users to input and manipulate data.

3. Code Functionality

The "Comprehensive Probability and Statistics Analysis Application" comprises various sections, each catering to specific statistical analysis tasks and functionalities. The application's main menu guides users to explore different statistical methods and techniques. The key sections and functionalities are discussed in detail below.

3.1 Main Menu

The main menu serves as the entry point to the application, providing users with a list of available options. Users can choose from various statistical analysis categories, such as basic statistics, probability distributions, permutations and

combinations, statistical relationships, and linear regression.

The main menu also includes an option to exit the application gracefully. Additionally, the application is designed with error handling to ensure that users provide valid inputs and navigate through the menu effectively.

3.2 Basic Statistics

This section of the application encompasses essential statistical measures for data analysis. Users can explore both graphical and tabular representations of datasets and derive descriptive statistical measures to gain insights into the data's central tendency and dispersion.

3.2.1 Graphical and Tabular Data Representation

The application allows users to visualize data using different types of plots, such as bar plots and histograms. Users can choose the type of plot based on the nature of their dataset and the insights they wish to derive.

The tabular data representation presents users with raw data in an organized format, enabling them to examine individual data points and patterns.

3.2.2 Descriptive Statistical Measures

The application provides a range of descriptive statistical measures to summarize data and gain a deeper understanding of its characteristics.

3.2.2.1 Mean

The mean, often referred to as the average, represents the central tendency of a dataset. It is calculated by summing all data points and dividing them by the total number of data points. The application implements the mathematical formula to calculate the mean and provides users with the result.

3.2.2.2 Median

The median is the middle value of a dataset when arranged in ascending order. If the dataset contains an even number of data points, the median is the average of the two middle values. The application computes the median and displays it to the user.

3.2.2.3 Mode

The mode represents the most frequently occurring value in a dataset. If all data points occur with equal frequency, the dataset is said to be multimodal. The application determines the mode(s) of the dataset and presents it to the user.

3.2.2.4 Standard Deviation

The standard deviation is a measure of the amount of variation or dispersion in a dataset. It quantifies how much individual data points deviate from the mean. The application computes the standard deviation and provides users with the result.

3.2.2.5 Variance

Variance measures the spread or dispersion of data points around the mean. It is calculated by squaring the standard deviation. The application calculates the variance and presents it to the user.

3.3 Probability Distributions

Probability distributions are fundamental to statistical analysis, as they model the likelihood of different outcomes in a random experiment or process. The application incorporates several probability distributions to help users understand and predict the occurrence of events.

3.3.1 Binomial Probability Distribution

The binomial distribution models the number of successes in a fixed number of independent Bernoulli trials with the same probability of success in each trial. Users can input the number of trials (n) and the probability of success in each trial (p) to compute the probability of getting a specific number of successes (k). The application uses SciPy's binom.pmf function to calculate the binomial probability mass function.

3.3.2 Poisson Probability Distribution

The Poisson distribution models the number of events that occur in a fixed interval of time or space. Users can input the number of trials (n) and the probability of success in each trial (p) to compute the probability of getting a specific number of events (k). The application calculates the Poisson probability mass function using SciPy's poisson.pmf function.

3.3.3 Hypergeometric Probability Distribution

The hypergeometric distribution models the probability of drawing a specific number of successes (k) from a finite population without replacement. Users can input the population size (N), the number of successes in the population (M), the number of trials (n), and the number of successes in the sample (k) to compute the hypergeometric probability mass function. The application utilizes SciPy's hypergeom.pmf function to perform the calculation.

3.4 Permutations and Combinations

Permutations and combinations are fundamental combinatorial concepts used in various fields, including probability, statistics, and combinatorics. The application provides users with multiple methods to calculate permutations and combinations based on their specific requirements.

3.4.1 Counting Technique

The counting technique is used when there are multiple steps involved in the arrangement of elements. Users can input the possible outcomes (m1, m2) to compute the total number of possibilities using the counting technique.

3.4.2 Factorial Method

The factorial method calculates the number of permutations by finding the factorial of the total number of objects (n). Users can input the objects and arrangements (n, r) to compute the total number of arrangements using the factorial method.

3.4.3 Multiplicative Technique

The multiplicative technique is used when each object can be arranged in multiple ways, and the total number of arrangements is the product of the number of choices for each object. Users can input the total number of objects (N) and the number of arrangements (r) to compute the number of permutations (arrangements) using the multiplicative technique.

3.4.4 Permutation and Combination

The application provides users with the option to directly calculate the number of permutations and combinations using SciPy's math.perm and math.comb functions. Users can input the total number of objects (N) and the number of arrangements (r) or combinations (r) to obtain the desired result.

3.5 Statistical Relationships

Statistical relationships analyze the associations between two or more variables in a dataset. The application includes functionalities to determine the covariance and correlation between variables.

3.5.1 Covariance

Covariance measures the extent to which two variables change together. If the covariance is positive, the variables tend to increase or decrease together (direct relationship). If the covariance is negative, one variable tends to increase while the other decreases (inverse relationship). A covariance of zero indicates no consistent linear relationship between the variables. The application calculates the covariance using the formula and determines the direction of the relationship based on the covariance value.

3.5.2 Correlation

Correlation measures the strength and direction of the linear relationship between two variables. It is represented by the correlation coefficient, which ranges from -1 to +1. A correlation coefficient close to +1 indicates a strong positive linear relationship, while a coefficient close to -1 indicates a strong negative linear relationship. A correlation coefficient close to 0 indicates a weak or no linear relationship. The application computes the correlation coefficient using the covariance and standard deviation and presents the result to the user.

3.6 Linear Regression

Linear regression is a widely used statistical technique to model the relationship between a dependent variable and one or more independent variables. The application includes both simple and multiple linear regression modeling to help users make predictions based on independent variables.

3.6.1 Simple Linear Regression

Simple linear regression models the relationship between a single independent variable (x) and a dependent variable (y). Users can input the dataset's independent and dependent variables to calculate the regression coefficients (slope and intercept). The application utilizes Statsmodels' OLS (Ordinary Least Squares) regression method to perform the regression analysis and displays the regression line and equation.

3.6.2 Multiple Linear Regression

Multiple linear regression models the relationship between multiple independent variables (x1, x2, ...) and a dependent variable (y). Users can input the dataset's independent variables and the dependent

variable to compute the regression coefficients for each independent variable. The application employs Stats models' OLS regression method to perform the multiple linear regression analysis.

4. Implementation Details

The implementation of the "Comprehensive Probability and Statistics Analysis Application" involved several key aspects, including user interface design, data handling and manipulation, and statistical computations. The following sections provide an in-depth overview of these aspects.

4.1 User Interface Design

The application's user interface was designed to be user-friendly, providing an intuitive experience for users to navigate through the various statistical analysis options. The user interface was primarily designed as a Command-Line Interface (CLI), allowing users to interact with the application through text-based commands.

4.1.1 Command-Line Interface (CLI)

The CLI displays the main menu, guiding users to explore different statistical analysis functionalities. Users can input numerical values and choose specific options to access the desired analysis tools.

4.1.2 User Input and Error Handling

The application incorporates robust user input validation to ensure users provide valid numerical inputs for computations. The application checks for invalid inputs, such as non-numeric characters or negative values, and prompts users to re-enter valid inputs.

4.1.3 Visualizations

The graphical representations in the application are generated using Matplotlib. Users can view the visualizations directly within the CLI, enhancing data interpretation and analysis.

4.2 Data Handling and Manipulation

Efficient data handling and manipulation are critical aspects of a statistical analysis application. The application allows users to import datasets from various sources, preprocess the data, and perform statistical computations.

4.2.1 Data Import and Export

Users can import datasets from external sources, such as CSV files or other data formats, into the application. The application leverages Pandas' read_csv function to read and load datasets. Users can also export computed results and statistical analysis outcomes to CSV files for further analysis or documentation.

4.2.2 Data Preprocessing

Data preprocessing involves cleaning and transforming raw data to make it suitable for analysis. The application supports basic data preprocessing operations, such as handling missing values, removing outliers, and scaling numerical data for regression modeling.

4.2.3 Data Cleaning and Transformation

The application provides users with the option to clean datasets by removing duplicates, handling missing data, and converting data types as necessary. Users can also transform data using functions like log transformation or normalization to improve data distribution.

4.3 Statistical Computations

The heart of the application lies in the implementation of various statistical computations. These computations are performed using a combination of custom functions and integration with external libraries.

4.3.1 Custom Functions

The application incorporates custom functions to perform specific statistical calculations, such as mean, median, variance, covariance, and regression analysis. Custom functions are designed with modularity in mind, enabling easy integration with the overall application.

4.3.2 Libraries Integration

The application seamlessly integrates external libraries, such as NumPy, SciPy, Statsmodels, and Pandas, to perform complex statistical computations. By leveraging the capabilities of these libraries, the application ensures efficiency and accuracy in data analysis and modeling.

5. Functionality in Depth

The "Comprehensive Probability and Statistics Analysis Application" includes various statistical functionalities, each with its mathematical foundations, use cases, applications, limitations, and considerations. This section provides an in-depth analysis of each statistical functionality offered by the application.

5.1 Basic Statistics

Basic statistics form the core of data analysis, helping users gain insights into the central tendency and dispersion of their datasets. The application provides graphical and tabular representations along with several descriptive statistical measures.

5.1.1 Mean

5.1.1.1 Mathematical Formula and Implementation

The mean, or average, of a dataset is calculated by summing all data points and dividing by the total number of data points. The mathematical formula for the mean (μ) of a dataset with n data points (x1, x2, ..., xn) is given by: $\mu = (x1 + x2 + ... + xn) / n$

The application implements a custom function to calculate the mean based on user-inputted data. The function takes an array of data points as input, sums the elements, and divides by the array's length (number of data points) to compute the mean.

Python Code:

```
def calculate_mean(data):
    sum_data = sum(data)
```

```
    num_data_points = len(data)
    mean = sum_data / num_data_points
    return mean
```

5.1.1.2 Use Cases and Applications

The mean is widely used in various fields, such as finance, economics, and social sciences, to summarize data and make data-driven decisions. Some common use cases and applications of the mean include:

1. Financial Analysis: Mean is used to calculate the average return on investments, average revenue, or average expenses.

2. Educational Assessments: In educational assessments, the mean is used to evaluate the average performance of students in a particular subject or test.

3. Opinion Surveys: Mean is used to summarize survey responses, such as average ratings on a scale from 1 to 5.

5.1.1.3 Limitations and Considerations

While the mean provides valuable information about the central tendency of a dataset, it may not always be the best measure of central tendency, especially in the presence of outliers or skewed data. Outliers can significantly impact the mean, pulling it towards extreme values and affecting its representativeness.

Users should exercise caution when interpreting the mean, considering the data distribution and the presence of outliers. In cases of highly skewed data, the median may be a more appropriate measure of central tendency.

5.1.2 Median

5.1.2.1 Mathematical Formula and Implementation

The median of a dataset is the middle value when the data points are arranged in ascending order. If the dataset has an odd number of data points, the median is the middle value. If the dataset has an even number of data points, the median is the average of the two middle values. The application implements a custom function to calculate the median based on user-inputted data. The function first sorts the data points in ascending order and then determines the median based on the length of the array (even or odd).

Python Code:

```
def calculate_median(data):
    sorted_data = sorted(data)
    num_data_points = len(data)

    if num_data_points % 2 == 1:
        median = sorted_data[num_data_points // 2]
    else:
        middle_index = num_data_points // 2
        median = (sorted_data[middle_index - 1] + sorted_data[middle_index]) / 2

    return median
```

5.1.2.2 Use Cases and Applications

The median is a robust measure of central tendency, especially in datasets with outliers or skewed distributions. Some common use cases and applications of the median include:

1. Income Distribution: The median income is used to describe the income level of the middle-class population, representing the income that divides the population into two halves.

2. Housing Prices: In real estate, the median housing price is often reported to provide an overview of property prices in a particular area.

3. Test Scores: In educational assessments, the median test score is used to represent the performance of the middle-scoring students.

5.1.2.3 Limitations and Considerations

The median is less sensitive to extreme values (outliers) compared to the mean. However, it may not provide the complete picture of the data distribution, especially when the dataset has multiple modes or when the data has significant variations. In such cases, additional measures, such as the mode or standard deviation, should be considered to understand the data distribution better.

5.1.3 Mode

5.1.3.1 Mathematical Formula and Implementation

The mode of a dataset is the value that appears most frequently. A dataset may have one mode (unimodal) or multiple modes (multimodal).

The application implements a custom function to calculate the mode based on user-inputted data. The function creates a frequency distribution of data points and identifies the values with the highest frequency.

Python Code:

```
def calculate_mode(data):
    frequency_dict = {}
    for value in data:
        if value in frequency_dict:
            frequency_dict[value] += 1
        else:
            frequency_dict[value] = 1

    max_frequency = max(frequency_dict.values())
    mode = [value for value, frequency in frequency_dict.items() if frequency == max_frequency]
```

```
return mode
```

5.1.3.2 Use Cases and Applications

The mode is valuable when identifying the most common or frequent value in a dataset. Some common use cases and applications of the mode include:

1. Customer Preference: In marketing, the mode is used to identify the most popular product or service among customers.

2. Demographics: In population studies, the mode is used to determine the most common age, income level, or education level among a group of individuals.

3. Inventory Management: In retail, the mode is used to identify the most frequently sold product, assisting in inventory management.

5.1.3.3 Limitations and Considerations

The mode may not be unique in some datasets, especially in datasets with continuous variables or a high number of data points. In multimodal datasets, there may be more than one mode, and users should consider all modes to understand the data distribution fully.

5.1.4 Standard Deviation

5.1.4.1 Mathematical Formula and Implementation

The standard deviation measures the amount of variation or dispersion in a dataset. It quantifies how much individual data points deviate from the mean.

The mathematical formula for the standard deviation (σ) of a dataset with n data points (x1, x2, ..., xn) and mean (μ) is given by:

σ = sqrt(((x1 - μ)^2 + (x2 - μ)^2 + ... + (xn - μ)^2) / n)

The application implements a custom function to calculate the standard deviation based on user-input data and meaning. The function first calculates the mean using the previously defined custom function and then computes the squared differences between each data point and the mean, finally taking the square root of the average of the squared differences.

Python Code:

```
def calculate_standard_deviation(data):
    mean = calculate_mean(data)
    squared_differences = [(x - mean) ** 2 for x in data]
    mean_squared_differences = sum(squared_differences) / len(data)
    standard_deviation = math.sqrt(mean_squared_differences)
    return standard_deviation
```

5.1.4.2 Use Cases and Applications

Standard deviation is a crucial measure in statistics, providing insights into the variability and spread of data. Some common use cases and applications of the standard deviation include:

1. Investment Risk: In finance, the standard deviation is used to measure the risk associated with an investment. A higher standard deviation indicates greater variability in returns, representing higher risk.

2. Quality Control: In manufacturing, the standard deviation is used to measure the variability in product quality. Lower standard deviation indicates consistent product quality.

3. Academic Performance: In educational assessments, the standard deviation is used to evaluate the spread of student scores. A higher standard deviation may indicate a wider range of student performance.

5.1.4.3 Limitations and Considerations

The standard deviation is sensitive to outliers, and extreme values can significantly impact its value. Users should be cautious when interpreting the standard deviation, especially in datasets with significant variations or outliers. In such cases, robust statistical measures, such as the median absolute deviation (MAD), may provide more robust estimates of data dispersion.

5.1.5 Variance

5.1.5.1 Mathematical Formula and Implementation

The variance is a squared measure of the standard deviation, representing the average of the squared differences between each data point and the mean.

The mathematical formula for the variance (σ^2) of a dataset with n data points (x1, x2, ..., xn) and mean (μ) is given by:

σ^2 = ((x1 - μ)^2 + (x2 - μ)^2 + ... + (xn - μ)^2) / n

The application implements a custom function to calculate the variance based on user-inputted data and mean. The function utilizes the previously defined custom function to calculate the mean and then computes the squared differences between each data point and the mean.

Python Code:

```
def calculate_variance(data):
    mean = calculate_mean(data)
    squared_differences = [(x - mean) ** 2 for x in data]
    variance = sum(squared_differences) / len(data)
    return variance
```

5.1.5.2 Use Cases and Applications

The variance is widely used in statistics and various fields, providing insights into the variability of data. Some common use cases and applications of the variance include:

1. Quality Control: In manufacturing, variance is used to assess the variability in product quality. Lower variance indicates more consistent product quality.

2. Spread of Data: The variance is used to understand how spread-out data points are from the mean, providing insights into the overall distribution of data.

Hypothesis Testing: In statistical hypothesis testing, the variance is used to calculate the standard error of the sample mean, helping determine the significance of results.

5.1.5.3 Limitations and Considerations

Like the standard deviation, the variance is sensitive to outliers, and extreme values can significantly impact its value. Additionally, the variance is a squared measure, making it harder to interpret in the original scale of the data. Users should consider the implications of the squared measure and be cautious when interpreting the variance, especially in datasets with extreme values.

5.2 Probability Distributions

Probability distributions are fundamental in statistical analysis, as they model the likelihood of different outcomes in a random experiment or process. The application incorporates several probability distributions to help users understand and predict the occurrence of events.

5.2.1 Binomial Probability Distribution

5.2.1.1 Mathematical Formula and Implementation

The binomial distribution models the number of successes in a fixed number of independent Bernoulli trials with the same probability of success in each trial. It is characterized by two parameters: the number of trials (n) and the probability of success in each trial (p).

The mathematical formula for the binomial probability mass function P(k) is given by:

P(k) = C(n, k) * p^k * (1 - p)^(n - k)

Where:

- C(n, k) is the number of combinations of n items taken k at a time and is given by C(n, k) = n! / (k! * (n - k)!).

- k is the number of successes in n trials.

- p is the probability of success in each trial.

- (1 - p) is the probability of failure in each trial.

The application implements a custom function to calculate the binomial probability mass function based on user-input values of n, k, and p. The function uses the previously defined custom function for calculating combinations (C(n, k)).

Python Code:

```
def calculate_binomial_probability(n, k, p):
    combinations = math.comb(n, k)
    probability = combinations * (p ** k) * ((1 - p) ** (n - k))
    return probability
```

5.2.1.2 Use Cases and Applications

The binomial probability distribution is widely used in various scenarios, especially when dealing with binary outcomes or success/failure events. Some common use cases and applications of the binomial probability distribution include:

1. Quality Control: In manufacturing, the binomial distribution can be used to model the number of defective items in a production batch.

2. Biological Studies: In genetics, the binomial distribution can be used to model the probability of inheriting specific genes from parents.

3. Market Research: In market research, the binomial distribution can be used to model the success rate of marketing campaigns.

5.2.1.3 Limitations and Considerations

The binomial distribution assumes that each trial is independent and has the same probability of success. In some real-world scenarios, this assumption may not hold true, leading to deviations from the binomial distribution. Additionally, the binomial distribution is discrete and applicable only to discrete random variables. For continuous random variables, other probability distributions, such as the normal distribution, are more suitable.

5.2.2 Poisson Probability Distribution

5.2.2.1 Mathematical Formula and Implementation

The Poisson distribution models the number of events that occur in a fixed interval of time or space. It is characterized by one parameter, λ (lambda), representing the average rate of events occurring in the given interval.

The mathematical formula for the Poisson probability mass function P(k) is given by:

P(k) = (e^(-λ) * λ^k) / k!

Where:

- e is Euler's number, approximately equal to 2.71828.

- λ is the average rate of events occurring in the given interval.

- k is the number of events observed in the interval.

The application implements a custom function to calculate the Poisson probability mass function based on user-inputted value of λ and k. The function uses the math.exp function for the constant e and the math.factorial function to calculate k!.

Python Code:

```
def calculate_poisson_probability(lam, k):
    probability = (math.exp(-lam) * (lam ** k)) / math.factorial(k)
    return probability
```

5.2.2.2 Use Cases and Applications

The Poisson probability distribution is commonly used in scenarios where events occur at a known average rate, but the exact timing of each event is random. Some common use cases and applications of the Poisson probability distribution include:

1. Insurance Claims: In insurance, the Poisson distribution can be used to model the number of claims made within a specific time frame.

2. Customer Service Calls: In customer service centers, the Poisson distribution can be used to model the number of calls received in each hour.

3. Website Traffic: In web analytics, the Poisson distribution can be used to model the number of visits to a website during a fixed time.

5.2.2.3 Limitations and Considerations

The Poisson distribution assumes that events occur independently and at a constant average rate. In practice, there may be cases where events are not independent or where the average rate changes over time. In such cases, other probability distributions, such as the compound Poisson distribution, may be more appropriate.

5.2.3 Hypergeometric Probability Distribution

5.2.3.1 Mathematical Formula and Implementation

The hypergeometric distribution models the probability of drawing a specific number of successes (k) from a finite population without replacement. It is characterized by four parameters: the population size (N), the number of successes in the population (M), the number of trials (n), and the number of successes in the sample (k).

The mathematical formula for the hypergeometric probability mass function P(k) is given by:

P(k) = (C(M, k) * C(N - M, n - k)) / C(N, n)

Where:

- C(M, k) is the number of combinations of M items taken k at a time and is given by C(M, k) = M! / (k! * (M - k)!).

- C(N - M, n - k) is the number of combinations of (N - M) items taken (n - k) at a time and is given by C(N - M, n - k) = (N - M)! / ((n - k)! * ((N - M) - (n - k))!).

- C(N, n) is the number of combinations of N items taken n at a time and is given by C(N, n) = N! / (n! * (N - n)!).

The application implements a custom function to calculate the hypergeometric probability mass function based on user-input values of N, M, n, and k. The function uses the previously defined custom function for calculating combinations (C(n, k)).

Python Code:

```
def calculate_hypergeometric_probability(N, M, n, k):
    numerator = math.comb(M, k) * math.comb(N - M, n - k)
    denominator = math.comb(N, n)



    probability = numerator / denominator
    return probability
```

5.2.3.2 Use Cases and Applications

The hypergeometric probability distribution is widely used in scenarios where drawing samples without replacement from a finite population is required. Some common use cases and applications of the hypergeometric probability distribution include:

1. Quality Control: In manufacturing, the hypergeometric distribution can be used to model the probability of finding defective items in a sample from a production batch.

2. Polling and Surveys: In opinion polls and surveys, the hypergeometric distribution can be used to model the probability of selecting a specific group of respondents.

3. Genetics: In genetics, the hypergeometric distribution can be used to model the probability of observing a specific genetic trait in a sample of individuals.

5.2.3.3 Limitations and Considerations

The hypergeometric distribution assumes that each drawer is independent and that items are drawn without replacement. In cases where the sample size is a significant proportion of the population size, the hypergeometric distribution may not be the best fit, and alternative distributions, such as the binomial distribution, may be more appropriate.

5.3 Permutations and Combinations

Permutations and combinations are fundamental combinatorial concepts used in various fields, including probability, statistics, and combinatorics. The application provides users with multiple methods to calculate permutations and combinations based on their specific requirements.

5.3.1 Counting Technique

5.3.1.1 Mathematical Formula and Implementation

The counting technique is used when there are multiple steps involved in the arrangement of elements. It is based on the multiplication principle, which states that the total number of ways to perform multiple tasks is the product of the number of ways to perform each individual task.

The application implements a custom function to calculate the total number of possibilities using the counting technique based on user-input values of possible outcomes (m1, m2, ...).

Python Code:

```
def calculate_counting_technique_possibilities(*outcomes):
    total_possibilities = 1
    for outcome in outcomes:
```

```
    total_possibilities *= outcome
  return total_possibilities
```

5.3.1.2 Use Cases and Applications

The counting technique is used in various scenarios where the arrangement of multiple elements is required. Some common use cases and applications of the counting technique include:

1. Password Combinations: In computer security, the counting technique can be used to calculate the total number of possible password combinations.

2. Lottery Games: In lottery games, the counting technique can be used to calculate the total number of possible ticket combinations.

3. Seating Arrangements: In event planning, the counting technique can be used to calculate the total number of seating arrangements for guests.

5.3.1.3 Limitations and Considerations

The counting technique is powerful for calculating the total number of possibilities in scenarios with multiple steps. However, it may become computationally intensive and challenging to apply when dealing with many possible outcomes or elements.

5.3.2 Factorial Method

5.3.2.1 Mathematical Formula and Implementation

The factorial method calculates the number of permutations by finding the factorial of the total number of objects (n). The factorial of a non-negative integer n is the product of all positive integers from 1 to n.

The mathematical formula for the factorial of a positive integer n is given by:

n! = 1 * 2 * 3 * ... * n

The application implements a custom function to calculate the number of permutations using the factorial method based on user-input values of objects and arrangements (n, r).

Python Code:

```
def calculate_permutations_factorial_method(n, r):
    permutations = math.factorial(n) // math.factorial(n - r)
    return permutations
```

5.3.2.2 Use Cases and Applications

The factorial method is widely used in combinatorics, especially when calculating the number of arrangements of elements without repetition. Some common use cases and applications of the factorial method for permutations include:

1. Arrangements of Objects: In combinatorics, the factorial method is used to calculate the total number of possible arrangements of objects without repetition.

2. Lottery Games: In lottery games, the factorial method can be used to calculate the total number of possible winning combinations.

3. Seating Arrangements: In event planning, the factorial method can be used to calculate the total number of seating arrangements for guests.

5.3.2.3 Limitations and Considerations

The factorial method becomes computationally challenging for large values of n, as the factorial grows rapidly. For large values of n, the calculation may require significant computational resources, and alternative methods, such as the multiplicative technique or combinatorial algorithms, may be more efficient.

5.3.3 Multiplicative Technique

5.3.3.1 Mathematical Formula and Implementation

The multiplicative technique is used when each object can be arranged in multiple ways, and the total number of arrangements is the product of the number of choices for each object.

The mathematical formula for the number of permutations using the multiplicative technique with n total objects and r arrangements is given by:

Permutations = n * (n - 1) * (n - 2) * ... * (n - r + 1)

The application implements a custom function to calculate the number of permutations using the multiplicative technique based on user-input values of total objects (N) and the number of arrangements (r).

Python Code:

```
def calculate_permutations_multiplicative_technique(N, r):
    permutations = 1
    for i in range(N, N - r, -1):
        permutations *= i
    return permutations
```

5.3.3.2 Use Cases and Applications

The multiplicative technique is applicable in scenarios where each object can be arranged in multiple ways, and the order of arrangement matters. Some common use cases and applications of the multiplicative technique for permutations include:

1. Arrangements of Letters: In word games, the multiplicative technique can be used to calculate the total number of possible arrangements of letters in a word.

2. Password Combinations: In computer security, the multiplicative technique can be used to calculate the total number of possible password combinations.

3. Seating Arrangements: In event planning, the multiplicative technique can be used to calculate the total number of seating arrangements for guests.

5.3.3.3 Limitations and Considerations

The multiplicative technique is useful for calculating permutations with repetition and when the order of arrangement matters. However, it may not be suitable for scenarios with a large number of arrangements or objects, as the number of possible arrangements grows rapidly.

5.3.4 Permutation and Combination

5.3.4.1 Mathematical Formula and Implementation

The application provides users with the option to directly calculate the number of permutations and combinations using SciPy's math.perm and math.comb functions. Users can input the total number of objects (N) and the number of arrangements (r) or combinations (r) to obtain the desired result.

Python Code:

```
from scipy import math


def calculate_permutations(N, r):
    permutations = math.perm(N, r)
    return permutations


def calculate_combinations


(N, r):
    combinations = math.comb(N, r)
    return combinations
```

5.3.4.2 Use Cases and Applications

The direct calculation of permutations and combinations using SciPy's math.perm and math.comb functions provides a convenient option for users who prefer a straightforward approach to obtain results. The functions are especially useful when the total number of objects and the number of arrangements or combinations are known.

5.3.4.3 Limitations and Considerations

The direct calculation of permutations and combinations using SciPy's math.perm and math.comb functions relies on the underlying algorithms implemented in the SciPy library. Users should ensure that the SciPy library is installed and up to date to access these functions.

5.4 Descriptive Analysis and Visualization

Descriptive analysis and visualization are essential components of data exploration, providing insights into the distribution and patterns present in the data. The application incorporates various descriptive analysis techniques and visualization tools to aid users in understanding their datasets.

5.4.1 Frequency Distribution

5.4.1.1 Mathematical Formula and Implementation

Frequency distribution is a tabular representation of the number of times each unique value occurs in a dataset. It is often presented as a histogram, which is a graphical representation of the frequency distribution.

The application implements a custom function to calculate the frequency distribution based on user-input data. The function creates a dictionary to count the occurrences of each unique value and returns the frequency distribution.

Python Code:

```
def calculate_frequency_distribution(data):
    frequency_dict = {}
```

```
    for value in data:
        if value in frequency_dict:
            frequency_dict[value] += 1
        else:
            frequency_dict[value] = 1
    return frequency_dict
```

5.4.1.2 Use Cases and Applications

Frequency distribution and histograms are widely used in various fields to understand the distribution of data. Some common use cases and applications of frequency distribution and histograms include:

1. Grade Distribution: In education, frequency distribution and histograms can be used to visualize the distribution of student grades in a class.

2. Income Distribution: In economics, frequency distribution and histograms can be used to analyze the income distribution of a population.

3. Customer Age Groups: In marketing, frequency distribution and histograms can be used to analyze the age groups of customers.

5.4.1.3 Limitations and Considerations

Frequency distribution and histograms provide insights into the distribution of data but may not fully capture the underlying data patterns, especially when dealing with small sample sizes. Users should be cautious when interpreting histograms, considering the binning strategy and the appropriate number of bins to represent the data accurately.

5.4.2 Box-and-Whisker Plot

5.4.2.1 Mathematical Formula and Implementation

The box-and-whisker plot, also known as the box plot, is a graphical representation of the distribution of data. It provides a visual summary of key statistics, including the median, quartiles, and potential outliers.

The application implements a custom function to create a box-and-whisker plot based on user-input data. The function uses Matplotlib to generate the plot and display it directly within the application.

Python Code:

```
import matplotlib.pyplot as plt

def create_box_plot(data):
    plt.figure(figsize=(8, 6))
    plt.boxplot(data, vert=False)
    plt.xlabel("Values")
    plt.title("Box-and-Whisker Plot")
    plt.show()
```

5.4.2.2 Use Cases and Applications

The box-and-whisker plot is commonly used to visually summarize the distribution of data and identify potential outliers. Some common use cases and applications of the box-and-whisker plot include:

1. Comparison of Groups: In research studies, box-and-whisker plots can be used to compare the distributions of data between different groups.

2. Data Exploration: In data analysis, box-and-whisker plots can be used to understand the spread and central tendency of data.

3. Outlier Detection: In anomaly detection, box-and-whisker plots can be used to identify potential outliers in the data.

5.4.2.3 Limitations and Considerations

The box-and-whisker plot provides a visual summary of the distribution of data but may not provide detailed information about the underlying data patterns. Users should be cautious when interpreting box-and-whisker plots, especially when dealing with skewed data or small sample sizes.

5.4.3 Scatter Plot

5.4.3.1 Mathematical Formula and Implementation

The scatter plot is a graphical representation of the relationship between two numerical variables. It displays individual data points as dots on a two-dimensional coordinate system, with one variable on the x-axis and the other on the y-axis.

The application implements a custom function to create a scatter plot based on user-input data for the x-axis and y-axis variables. The function uses Matplotlib to generate the plot and display it directly within the application.

Python Code:

```
def create_scatter_plot(x_data, y_data):
    plt.figure(figsize=(8, 6))
    plt.scatter(x_data, y_data)
    plt.xlabel("X-axis Variable")
    plt.ylabel("Y-axis Variable")
    plt.title("Scatter Plot")
    plt.show()
```

5.4.3.2 Use Cases and Applications

The scatter plot is widely used to visualize the relationship between two numerical variables and identify potential patterns or correlations. Some common use cases and applications of the scatter plot include:

1. Correlation Analysis: In statistics, scatter plots are used to assess the strength and direction of the relationship between two variables.

2. Regression Analysis: In machine learning, scatter plots can be used to visualize the relationship between the dependent and independent variables in regression models.

3. Data Exploration: In data analysis, scatter plots are used to understand the distribution and dispersion of data.

5.4.3.3 Limitations and Considerations

The scatter plot provides a visual representation of the relationship between two numerical variables but may not provide a comprehensive view of complex relationships. Users should be cautious when interpreting scatter plots, especially when dealing with nonlinear relationships or potential confounding variables.

5.4.4 Correlation Coefficient and Heatmap

5.4.4.1 Mathematical Formula and Implementation

The correlation coefficient is a statistical measure that quantifies the strength and direction of the linear relationship between two numerical variables. It is represented by the correlation coefficient, which ranges from -1 to +1.

The application implements a custom function to calculate the correlation coefficient based on user-input data for the two variables. The function uses NumPy's corrcoef function to calculate the correlation coefficient and displays it along with a heatmap using Matplotlib.

Python Code:

```
import numpy as np


def calculate_correlation_coefficient(x_data, y_data):
    correlation_coefficient = np.corrcoef(x_data, y_data)[0, 1]
```

```
    return correlation_coefficient


def create_heatmap(data, labels):
    plt.figure(figsize=(8, 6))
    heatmap = plt.imshow(data, cmap='coolwarm', interpolation='nearest')
    plt.colorbar(heatmap)
    plt.xticks(range(len(labels)), labels, rotation=45)
    plt.yticks(range(len(labels)), labels)
    plt.title("Correlation Heatmap")
    plt.show()
```

5.4.4.2 Use Cases and Applications

The correlation coefficient and heatmap are commonly used to assess the linear relationship between multiple numerical variables and identify potential patterns or associations. Some common use cases and applications of the correlation coefficient and heatmap include:

1. Feature Selection: In machine learning, correlation analysis and heatmaps can be used to identify highly correlated features for dimensionality reduction.

2. Multivariate Analysis: In data analysis, correlation heatmaps can be used to visualize the relationships between multiple variables.

3. Identifying Patterns: In research studies, correlation analysis and heatmaps can be used to identify patterns or associations among variables.

5.4.4.3 Limitations and Considerations

The correlation coefficient measures only the linear relationship between variables and may not capture complex nonlinear relationships. Additionally, correlation does not imply causation, and users should be cautious when interpreting correlation results.

5.5 Machine Learning Algorithms

Machine learning algorithms are used to build predictive models from data. The application incorporates various machine learning algorithms to support users in solving classification and regression tasks.

5.5.1 Linear Regression

5.5.1.1 Mathematical Formula and Implementation

Linear regression is a supervised learning algorithm used for regression tasks. It models the relationship between a dependent variable and one or more independent variables by fitting a linear equation to the observed data.

The application implements a custom function for linear regression based on user-input feature matrix (X) and target variable (y).

Python Code:

```
from sklearn.linear_model import LinearRegression


def perform_linear_regression(X, y):
    model = LinearRegression()
    model.fit(X, y)
    return model
```

5.5.1.2 Use Cases and Applications

Linear regression is widely used in various fields for predicting numerical values. Some common use cases and applications of linear regression include:

1. House Price Prediction: In real estate, linear regression can be used to predict house prices based on features such as area, number of bedrooms, etc.

2. Demand Forecasting: In retail, linear regression can be used to forecast product demand based on historical sales data.

3. Salary Prediction: In human resources, linear regression can be used to predict salaries based on factors such as years of experience, education, etc.

5.5.1.3 Limitations and Considerations

Linear regression assumes a linear relationship between the dependent and independent variables. It may not perform well if the relationship is non-linear. Users should also consider multicollinearity and overfitting when applying linear regression.

5.5.2 Logistic Regression

5.5.2.1 Mathematical Formula and Implementation

Logistic regression is a supervised learning algorithm used for binary classification tasks. It models the probability of the dependent variable belonging to a particular class as a function of the independent variables.

The application implements a custom function for logistic regression based on user-input feature matrix (X) and binary target variable (y).

Python Code:

```
from sklearn.linear_model import LogisticRegression


def perform_logistic_regression(X, y):
    model = LogisticRegression()
    model.fit(X, y


)
    return model
```

5.5.2.2 Use Cases and Applications

Logistic regression is widely used in various fields for binary classification tasks. Some common use cases and applications of logistic regression include:

1. Churn Prediction: In telecommunications, logistic regression can be used to predict whether a customer will churn or not.

2. Credit Risk Assessment: In finance, logistic regression can be used to assess the risk of a customer defaulting on a loan.

3. Medical Diagnosis: In healthcare, logistic regression can be used to diagnose diseases based on patient symptoms and medical history.

5.5.2.3 Limitations and Considerations

Logistic regression assumes a linear relationship between the independent variables and the log-odds of the target variable. It may not perform well for complex decision boundaries. Users should also consider class imbalance and feature scaling when applying logistic regression.

5.5.3 Decision Tree

5.5.3.1 Mathematical Formula and Implementation

Decision tree is a supervised learning algorithm used for both classification and regression tasks. It creates a tree-like model by recursively splitting the data based on the features, leading to leaf nodes that represent the target variable's predicted value or class.

The application implements a custom function for decision tree classification based on user-input feature matrix (X) and target variable (y).

Python Code:

```
from sklearn.tree import DecisionTreeClassifier


def perform_decision_tree_classification(X, y):
    model = DecisionTreeClassifier()
    model.fit(X, y)
    return model
```

5.5.3.2 Use Cases and Applications

Decision trees are widely used in various fields for classification and regression tasks. Some common use cases and applications of decision trees include:

1. Customer Segmentation: In marketing, decision trees can be used to segment customers based on their behavior and characteristics.

2. Loan Approval: In finance, decision trees can be used to assess whether a loan application should be approved or rejected.

3. Disease Diagnosis: In healthcare, decision trees can be used to diagnose diseases based on patient symptoms and medical tests.

5.5.3.3 Limitations and Considerations

Decision trees can be prone to overfitting, especially when the tree becomes too deep and complex. Users should consider pruning and tuning hyperparameters to improve generalization performance. Additionally, decision trees may not capture complex relationships between features.

5.5.4 Random Forest

5.7.4.1 Mathematical Formula and Implementation

Random forest is an ensemble learning technique that uses multiple decision trees to make predictions. It creates a forest of decision trees and aggregates their predictions to obtain a more accurate and robust result.

The application implements a custom function for random forest classification based on user-input feature matrix (X) and target variable (y).

Python Code:

```
from sklearn.ensemble import RandomForestClassifier


def perform_random_forest_classification(X, y):
    model = RandomForestClassifier()
    model.fit(X, y)
    return model
```

5.5.4.2 Use Cases and Applications

Random forests are widely used in various fields for classification tasks. Some common use cases and applications of random forests include:

1. Image Classification: In computer vision, random forests can be used for image classification tasks.

2. Customer Churn Prediction: In telecommunications, random forests can be used to predict whether a customer will churn or not.

3. Credit Card Fraud Detection: In finance, random forests can be used to detect fraudulent transactions.

5.5.4.3 Limitations and Considerations

Random forests can be computationally expensive, especially when dealing with many trees and features. Users should also consider tuning hyperparameters to optimize the model's performance.

5.6 Python Code

```
import math
import statistics
import numpy as np
import pandas as pd
from scipy.stats import norm
from scipy.stats import binom
from tabulate import tabulate
import matplotlib.pyplot as plt
from scipy.stats import poisson
from scipy.stats import hypergeom
from sklearn.linear_model import LinearRegression


# ---------------------------------------------- Starting Decision ----------------------------------------------------
while True:

    print("----------------------------------------- ( MAIN MENU ) ----------------------------------------------------\n")
    print("Press 1 -> Graphical and Tabular Representation\n")
    print("Press 2 -> Descriptive Statistical Measure (Absolute & Relative)\n")
    print("Press 3 -> Probability Methods/Distribution\n")
    print("Press 4 -> Regression Modeling and Predictions and Regression estimates\n")
    print("Press 5 -> End the Program\n")

    x = int(input("Enter Choice: "))  # User Input

    if x == 1:  # For Selecting Graphical and Tabular Representation-----------------------------------------------------
        print("---------------------------------- ( TYPES OF REPRESENTATIONS ) -------------------------------------\n")
        print("Press 1 -> Tabular Representation\n")
        print("Press 2 -> Graphical Representation\n")
        print("Press 3 -> Go Back\n")

        x1 = int(input("Enter Choice: "))

        if x1 == 1:  # For Selecting Tabular Representation

            # Ask the user for the data set (space-separated numbers)
            data_input = input("Enter the data set (Quantitative data only, space-separated numbers): ")
            data = [float(item) for item in data_input.split()]

            # Calculate the frequency distribution using pandas
```

```
    df = pd.DataFrame(data, columns=['Class'])
    num_classes = 7

    # Calculate the range of the data and the bin width for the histogram
    data_range = max(data) - min(data)
    bin_width = data_range / num_classes

    # Create bins and labels for the histogram
    bins = [min(data) + i * bin_width for i in range(num_classes + 1)]
    labels = [f"{int(bins[i])}-{int(bins[i + 1])}" for i in range(num_classes)]

    df['Class'] = pd.cut(df['Class'], bins=bins, labels=labels, right=False)
    df = df.groupby('Class').size().reset_index(name='Frequency')
    df['Cumulative Frequency'] = df['Frequency'].cumsum()
    df['Relative Frequency'] = df['Frequency'] / df['Frequency'].sum()

    # Display the frequency distribution table
    print("\nFrequency Distribution Table:")
    print(tabulate(df, headers='keys', tablefmt='grid', numalign='center'))

elif x1 == 2:  # For Selecting Graphical Representation

    print("------------------------------- ( GRAPHICAL REPRESENTATION ) ----------------------------------\n")
    print("Press 1 -> Enter Data for Single Bar Chart\n")
    print("Press 2 -> Enter Data for Multiple Bar Chart\n")
    print("Press 3 -> Enter Data for Component Bar Chart\n")
    print("Press 4 -> Enter Data for Pie Chart\n")
    print("Press 5 -> Enter Data for Histogram\n")
    print("Press 6 -> Enter Data for Box Plot\n")
    print("Press 7 -> Go Back to Main Screen\n")

    N = int(input("Enter Choice: "))

    if N == 1:  # For Single Bar Chart

        data = {}
        num_categories = int(input("Enter the number of categories: "))

        for i in range(num_categories):
            category_name = input(f"Enter the name of category {i + 1}: ")
            category_value = int(input(f"Enter the value for {category_name}: "))
            data[category_name] = category_value

        x_axis_label = input("Enter the name of X axis: ")
```

```
    y_axis_label = input("Enter the name of Y axis: ")
    plot_title = input("Enter the title of the plot: ")

    categories = list(data.keys())
    values = list(data.values())

    fig = plt.figure(figsize=(10, 5))
    plt.bar(categories, values, color='maroon', width=0.4)

    plt.xlabel(x_axis_label)
    plt.ylabel(y_axis_label)
    plt.title(plot_title)
    plt.show()

elif N == 2:  # For Multi Bar Chart

    categories = input("Enter the names of categories separated by spaces: ").split()
    x_axis_label = input("Enter the name of X axis: ")
    y_axis_label = input("Enter the name of Y axis: ")
    plot_title = input("Enter the title of the plot: ")

    var1 = input("Enter Name of Variable 1: ")
    var2 = input("Enter Name of Variable 2: ")

    Sample_A = []
    Sample_B = []

    for category in categories:
        s_A = int(input(f"Enter amount for {var1} in {category}: "))
        Sample_A.append(s_A)

        s_B = int(input(f"Enter amount for {var2} in {category}: "))
        Sample_B.append(s_B)

    X_axis = np.arange(len(categories))

    plt.bar(X_axis - 0.2, Sample_A, 0.4, label=var1)
    plt.bar(X_axis + 0.2, Sample_B, 0.4, label=var2)

    plt.xticks(X_axis, categories)
    plt.xlabel(x_axis_label)
    plt.ylabel(y_axis_label)
    plt.title(plot_title)
    plt.legend()
```

```
    plt.show()

elif N == 3:  # For Component/Stacked Bar Chart

    num_categories = int(input("Enter the number of categories: "))
    x = []
    data = []

    for i in range(num_categories):
        category_name = input(f"Enter the name of category {i + 1}: ")
        x.append(category_name)

        category_data = np.array([int(score) for score in input(
            f"Enter the scores for {category_name} separated by spaces: ").split()])
        data.append(category_data)

    legend_labels = [input(f"Enter the label for round {i + 1}: ") for i in range(len(data))]
    x_axis_label = input("Enter the name of X axis: ")
    y_axis_label = input("Enter the name of Y axis: ")
    plot_title = input("Enter the title of the plot: ")


    fig, ax = plt.subplots()
    bottom = np.zeros(len(x))

    for i, round_data in enumerate(data):
        ax.bar(x, round_data, bottom=bottom)
        bottom += round_data

    plt.xlabel(x_axis_label)
    plt.ylabel(y_axis_label)
    plt.legend(legend_labels)
    plt.title(plot_title)
    plt.show()

elif N == 4:  # For Pie Chart

    num_categories = int(input("Enter the number of categories: "))
    cars = []
    data = []

    for i in range(num_categories):
        car_name = input(f"Enter the name of category {i + 1}: ")
        cars.append(car_name)
```

```
        car_data = int(input(f"Enter the value for {car_name}: "))
        data.append(car_data)

    show_percentages = input(
        "Do you want to display percentages in the pie chart? (yes/no): ").lower() == 'yes'
    title = input("Enter the title of the plot: ")

    fig = plt.figure(figsize=(10, 7))
    _, texts, autotest = plt.pie(data, labels=cars, autopct='%1.1f%%' if show_percentages else None)

    for text, autotext in zip(texts, autotest):
        text.set(color='white' if show_percentages else 'black')
        autotext.set(color='white' if show_percentages else 'black', size=10, weight='bold')

    plt.title(title)
    plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

    # Adding legends and category names
    plt.legend(labels=cars, title='Categories', loc='best')
    plt.show()

elif N == 5:  # For Histogram

    # Ask the user for the dataset
    dataset = input("Enter the dataset (space-separated numbers): ")
    data_points = [float(item) for item in dataset.split()]

    # Ask the user for the title of the histogram
    plot_title = input("Enter the title of the histogram: ")

    # Creating histogram
    fig, axs = plt.subplots(1, 1, figsize=(10, 7), tight_layout=True)
    n_bins = 'auto'  # You can adjust this to change the number of bins

    # Plot the histogram with customized appearance
    axs.hist(data_points, bins=n_bins, color='green', edgecolor='black', alpha=0.7)

    # Add x, y gridlines
    axs.grid(color='gray', linestyle='--', linewidth=0.5, alpha=0.5)

    # Set background color
    axs.set_facecolor('#f0f0f0')
```

```
    # Customize tick labels
    axs.tick_params(axis='both', which='major', labelsize=12)
    axs.tick_params(axis='both', which='minor', labelsize=10)

    # Add a vertical line at the mean
    mean_value = np.mean(data_points)
    axs.axvline(mean_value, color='red', linestyle='dashed', linewidth=2, label='Mean')

    # Add a vertical line at the median
    median_value = np.median(data_points)
    axs.axvline(median_value, color='blue', linestyle='dashed', linewidth=2, label='Median')

    # Add legend
    axs.legend(loc='upper right', fontsize=12)

    # Adding extra features
    plt.xlabel("X-axis", fontsize=14)
    plt.ylabel("Frequency", fontsize=14)
    plt.title(plot_title, fontsize=16)

    # Show plot
    plt.show()

elif N == 6:  # For Box Plot

    # Ask the user for the title of the box plot
    plot_title = input("Enter the title of the box plot: ")

    # Ask the user for the data for the box plot (space-separated numbers)
    data_input = input("Enter the data for the box plot (Quantitative data only): ")
    data = [float(item) for item in data_input.split()]

    # Creating box plot
    fig = plt.figure(figsize=(10, 7))
    plt.boxplot(data)

    # Calculate and add the mean as a red dot
    mean_value = np.mean(data)
    plt.plot(1, mean_value, marker='o', markersize=8, color='red', label='Mean')

    # Adding extra features
    plt.xlabel("Box Plot", fontsize=14)
    plt.ylabel("Data Points", fontsize=14)
    plt.title(plot_title, fontsize=16)
```

```
            plt.legend()

            # Show plot
            plt.show()

        elif N == 7:  # Break for Graphical Data
            continue

        else:  # This else ends Graphical Data
            print("Invalid Entry")
            continue
    elif x1 == 3:  # Break for Choice between Graphical and Tabular
        continue

    else:  # This else ends the choice of tabular and graphical
        print("Invalid Entry")
        continue

elif x == 2:  # For Selecting Descriptive Statistical Measure (Absolute & Relative) ------------------------------
--

    while True:
        print("---------------------------------- ( DESCRIPTIVE STATISTICS ) ----------------------------------\n")
        print("Press 1 -> Mean\n")
        print("Press 2 -> Median\n")
        print("Press 3 -> Mode\n")
        print("Press 4 -> Range\n")
        print("Press 5 -> Variance\n")
        print("Press 6 -> Standard Deviation\n")
        print("Press 7 -> 3 Sigma Rule\n")
        print("Press 8 -> Quartile 1,2,3\n")
        print("Press 9 -> Inter quartile Range\n")
        print("Press 10 -> Coefficient of Range\n")
        print("Press 11 -> Coefficient of IQR\n")
        print("Press 12 -> Coefficient of Variation\n")
        print("Press 13 -> Go Back\n")

        x = int(input("Enter Choice: "))

        if x == 1:  # IF STATEMENT FOR -MEAN-

            # Ask the user to input a list of numbers (space-separated)
            data_input = input("Enter a list of numbers (space-separated): ")
            data = list(map(int, data_input.split()))
```

```
    # Calculate the mean of the numbers
    mean = statistics.mean(data)

    # Print the result
    print("Mean is:", mean)

elif x == 2:  # IF STATEMENT FOR -MEDIAN-

    # Ask the user to input a list of numbers (space-separated)
    data_input = input("Enter a list of numbers (space-separated): ")
    data = list(map(int, data_input.split()))

    # Calculate the median of the numbers
    median = statistics.median(data)

    # Print the result
    print("Median is:", median)

elif x == 3:  # IF STATEMENT FOR -MODE-

    # Ask the user to input a list of numbers (space-separated)
    data_input = input("Enter a list of numbers (space-separated): ")
    data = list(map(int, data_input.split()))

    # Calculate the mode of the numbers
    mode = statistics.mode(data)

    # Print the result
    print("Mode is:", mode)

elif x == 4:  # IF STATEMENT FOR -RANGE-

    # Ask the user to input a list of numbers (space-separated)
    data_input = input("Enter a list of numbers (space-separated): ")
    data = list(map(int, data_input.split()))

    # Calculate the range of the numbers
    data_range = max(data) - min(data)

    # Print the result
    print("Range is:", data_range)

elif x == 5:  # IF STATEMENT FOR -VARIANCE-
```

```
    # Ask the user to input a list of numbers (space-separated)
    data_input = input("Enter a list of numbers (space-separated): ")
    data = list(map(float, data_input.split()))

    # Calculate the variance of the numbers
    variance = statistics.variance(data)

    # Print the result
    print("Variance is:", variance)

elif x == 6:  # IF STATEMENT FOR STANDARD -DEVIATION-

    # Ask the user to input a list of numbers (space-separated)
    data_input = input("Enter a list of numbers (space-separated): ")
    data = list(map(float, data_input.split()))

    # Calculate the standard deviation of the numbers
    standard_deviation = statistics.stdev(data)

    # Print the result
    print("Standard Deviation is:", standard_deviation)

elif x == 7:  # IF STATEMENT FOR -3 SIGMA RULE-

    # Ask the user to input a list of numbers (space-separated)
    data_input = input("Enter a list of numbers (space-separated): ")
    data = list(map(float, data_input.split()))

    # Calculate the mean and standard deviation of the numbers
    mean = statistics.mean(data)
    standard_deviation = statistics.stdev(data)

    # Calculate the lower and upper bounds for the 3-sigma rule
    lower_bound = mean - 3 * standard_deviation
    upper_bound = mean + 3 * standard_deviation

    # Filter out the data points outside the 3-sigma range
    data_within_3sigma = [x for x in data if lower_bound <= x <= upper_bound]
    data_outside_3sigma = [x for x in data if x < lower_bound or x > upper_bound]

    # Print the results
    print("Data points within 3-sigma range:")
    print(data_within_3sigma)
```

```
    print("\nData points outside 3-sigma range:")
    print(data_outside_3sigma)

elif x == 8:  # IF STATEMENT FOR -QUARTILE 1,2,3-

    # Ask the user to input a list of numbers (space-separated)
    data_input = input("Enter a list of numbers (space-separated): ")
    data = list(map(float, data_input.split()))

    # Calculate the quartiles
    q1 = statistics.quantiles(data, n=4)[0]
    q2 = statistics.median(data)
    q3 = statistics.quantiles(data, n=4)[2]

    # Print the results
    print("Q1 (25th percentile):", q1)
    print("Q2 (Median, 50th percentile):", q2)
    print("Q3 (75th percentile):", q3)

elif x == 9:  # IF STATEMENT FOR -INTER QUARTILE RANGE-

    # Ask the user to input a list of numbers (space-separated)
    data_input = input("Enter a list of numbers (space-separated): ")
    data = list(map(float, data_input.split()))

    # Calculate the quartiles
    q1 = statistics.quantiles(data, n=4)[0]
    q3 = statistics.quantiles(data, n=4)[2]

    # Calculate the Inter quartile Range (IQR)
    iqr = q3 - q1

    # Print the result
    print("Inter quartile Range (IQR):", iqr)

elif x == 10:  # IF STATEMENT FOR -CO-EFFICIENT OF RANGE-

    # Ask the user to input a list of numbers (space-separated)
    data_input = input("Enter a list of numbers (space-separated): ")
    data = list(map(float, data_input.split()))

    # Calculate the coefficient of range
    data_range = max(data) - min(data)
    coefficient_of_range = (data_range / (max(data) + min(data))) * 100
```

```
    # Print the result
    print("Coefficient of Range:", coefficient_of_range)

elif x == 11:  # IF STATEMENT FOR -CO-EFFICIENT OF IQR-

    # Ask the user for a list of numbers (space-separated)
    data_input = input("Enter a list of numbers (space-separated): ")
    data = list(map(float, data_input.split()))

    # Calculate the first quartile (Q1), second quartile (Q2 or median), and third quartile (Q3)
    q1 = np.percentile(data, 25)
    q2 = np.percentile(data, 50)
    q3 = np.percentile(data, 75)

    # Calculate the inter quartile range (IQR)
    iqr = q3 - q1

    # Calculate the coefficient of IQR
    coefficient_of_iqr = (iqr / (q3 + q1)) * 100

    # Print the results
    print("Coefficient of IQR:", coefficient_of_iqr)

elif x == 12:  # IF STATEMENT FOR CO-EFFICIENT OF VARIANCE-

    # Ask the user for a list of numbers (space-separated)
    data_input = input("Enter a list of numbers (space-separated): ")
    data = list(map(float, data_input.split()))

    # Calculate the standard deviation
    std_deviation = np.std(data)

    # Calculate the mean (x-bar)
    mean = np.mean(data)

    # Calculate the coefficient of variance
    coefficient_of_variance = (std_deviation / mean) * 100

    # Print the results
    print("Coefficient of Variance:", coefficient_of_variance)
elif x == 13:
    break
```

```
        else:
            print("Invalid Entry")

        input("Press any key to continue...")  # Pausing

    elif x == 3:  # Probability Methods/Distribution----------------------------------------------------------------------

        print("------------------------------ ( PROBABILITY METHODS/DISTRIBUTIONS ) ------------------------------
\n")
        print("Press 1 -> Simple Probability\n")
        print("Press 2 -> Counting Technique\n")
        print("Press 3 -> Factorial Method\n")
        print("Press 4 -> Multiplicative Technique\n")
        print("Press 5 -> Permutation\n")
        print("Press 6 -> Combination\n")
        print("Press 7 -> Binomial Probability Distribution\n")
        print("Press 8 -> Poison Probability Distribution\n")
        print("Press 9 -> Hyper-geometric Probability Distribution\n")
        print("Press 10 -> Uniform Probability Distribution (Standardization)\n")
        print("Press 11 -> Covariance\n")
        print("Press 12 -> Correlation\n")
        print("Press 13 -> Go Back\n")

        val = int(input("Enter Choice: "))

        if val == 1:  # IF STATEMENT FOR -SIMPLE PROBABILITY-

            favourable_outcome = float(input("Enter Favourable Outcome: "))
            total_outcome = float(input("Enter Total Outcomes: "))

            if favourable_outcome <= 0 or total_outcome <= 0:
                print("Invalid Entry")
                continue
            if favourable_outcome > total_outcome:  # Check for invalid Entry
                print("Favourable cannot be greater then Total Outcome")
                continue

            result = favourable_outcome / total_outcome
            print("Simple Probability:", result)

        elif val == 2:  # IF STATEMENT FOR -COUNTING TECHNIQUE-

            m1 = int(input("Enter Possible Outcome m1: "))
            m2 = int(input("Enter Possible Outcome m2: "))
```

```
    result = m1 * m2
    print("Total Number of Possibilities:", result)

elif val == 3:  # IF STATEMENT FOR -FACTORIAL METHOD-

    objects = int(input("Enter Objects: "))
    arrangements = int(input("Enter Arrangements: "))

    if objects != arrangements:
        print("Invalid Entry")
        continue

    result = math.factorial(objects)  # Calculate the factorial using the built-in function

    # Print the result
    print(f"Total Number of arrangements: {result}")

elif val == 4:  # IF STATEMENT FOR -MULTIPLICATIVE TECHNIQUE-

    # Ask the user for the values of n and r
    N = int(input("Enter the value of n (total objects): "))
    r = int(input("Enter the value of r (arrangements): "))

    arrangements = math.perm(N, r)  # Calculate the number of permutations (arrangements)

    print(f"Number of arrangements: {arrangements}")

elif val == 5:  # IF STATEMENT FOR PERMUTATION

    N = int(input("Enter the value of n (total objects): "))
    r = int(input("Enter the value of r (arrangements): "))

    permutations = math.perm(N, r)  # Calculate the number of permutations

    print(f"Number of permutations: {permutations}")

elif val == 6:  # IF STATEMENT FOR COMBINATION

    N = int(input("Enter the value of n (total objects): "))
    r = int(input("Enter the value of r (combinations): "))

    combinations = math.comb(N, r)  # Calculate the number of combinations

    print(f"Number of combinations: {combinations}")
```

```
elif val == 7:  # IF STATEMENT FOR -BINOMIAL PROBABILITY DISTRIBUTION-

    # Ask the user for the values of n, p, and k
    N = float(input("Enter the number of trials (n): "))
    P = float(input("Enter the probability of success in each trial (p): "))
    X = float(input("Enter the number of successes (k): "))

    if N < 0 or P < 0 or X < 0 or P > 1:  # Check for Invalid Entry
        print("Invalid Entry")
        continue

    probability = binom.pmf(X, N, P)  # Calculate the binomial probability mass function
    expected_value_binomial = N * P  # Calculate Expected Value
    variance_b = (N * P) / 1 - P  # Calculate Variance
    standard_b = math.sqrt(variance_b)  # Calculate Standard Deviation

    print(f"\nThe probability of getting exactly {X} successes in {N} trials with a success probability of "
          f"{P} is: {probability:.5f}")
    print(f"Expected Value: {expected_value_binomial}")
    print(f"Variance Value: {variance_b}")
    print(f"Standard Deviation Value: {standard_b}\n")

elif val == 8:  # IF STATEMENT FOR -POISON PROBABILITY DISTRIBUTION-

    N_ = float(input("Enter the number of trials (n): "))
    P_ = float(input("Enter the probability of success in each trial (p): "))
    # Ask the user for the specific value (k) for which to calculate the probability
    k = float(input("Enter the number of events (k): "))

    if N_ < 100 or P_ > 0.1:  # Check for n >= 100 and p <= 0.1
        print("Invalid Entry")
        continue

    lamda = N_ * P_  # Calculating Average (lambda) for Poisson distribution

    probability = poisson.pmf(k, lamda)  # Calculate the Poisson probability mass function
    expectedValue_poi = lamda
    variance_poi = lamda
    standard_poi = math.sqrt(variance_poi)

    print(f"The probability of getting exactly {k} events with an average rate of {lamda:.5f} "
          f"is: {probability:.5f}")
    print(f"Expected Value: {expectedValue_poi:.5f}")
```

```
        print(f"Variance Value: {variance_poi:.5f}")
        print(f"Standard Deviation Value: {standard_poi:.5f}\n")

    elif val == 9:  # IF STATEMENT FOR -HYPER GEOMETRIC PROBABILITY DISTRIBUTION-

        N = int(input("Enter the population size (N): "))
        M = int(input("Enter the number of successes in the population (M): "))
        n = int(input("Enter the number of trials (n): "))
        k = int(input("Enter the number of successes in the sample (k): "))

        if N <= 0 or M < 0 or n <= 0 or k < 0 or M > N or k > n or M < n:  # Check for invalid entry
            print("Invalid Entry")
            continue

        probability = hypergeom.pmf(k, N, M, n)  # Calculate the Hyper-geometric probability mass
function
        expectedValue_hyper = (n * M) / N  # Calculation of expected value
        variance_hyper = ((n * M) / (N - 1)) * ((N - n) / (N - 1)) * ((N - M) / N)  # Calculation of Variance
        standard_hyper = math.sqrt(variance_hyper)

        print(f"The probability of getting exactly {k} successes in {n} trials with a population size of {N} "
              f"and {M} successes in the population is: {probability:.5f}")
        print(f"Expected Value: {expectedValue_hyper:.5f}")
        print(f"Variance Value: {variance_hyper:.5f}")
        print(f"Standard Deviation Value: {standard_hyper:.5f}\n")

    elif val == 10:  # IF STATEMENT FOR UNIFORM DISTRIBUTION

        value_x = float(input("Enter Value of x: "))
        mean_x = float(input("Enter Mean: "))
        std_x = float(input("Enter Standard Deviation: "))

        z_score = (value_x - mean_x)/std_x

        # Calculate the p-value (probability) corresponding to the Z-score
        p_value = 1 - norm.cdf(z_score)

        print("Z-score:", z_score)
        print("p-value:", p_value)

    elif val == 11:  # IF STATEMENT FOR CO-VARIANCE

        # Get user input for data points
        print("Enter the data points for X and Y in the format 'x1,x2,...,y'. Type 'done' to stop.")
```

```
    data_points = []
    while True:
        point = input("Data point: ")
        if point.lower() == 'done':
            break
        try:
            data = list(map(float, point.split(',')))
            data_points.append(data)
        except ValueError:
            print("Invalid input! Please try again.")

    # Convert data_points to numpy array
    data_points = np.array(data_points)

    # Extract X and Y values from data_points
    X = data_points[:, :-1]
    Y = data_points[:, -1]

    # Calculate the covariance
    n = len(X)
    mean_x = sum(X) / n
    mean_y = sum(Y) / n

    cov = sum((X[i] - mean_x) * (Y[i] - mean_y) for i in range(n)) / n

    # Print the direction of the relationship based on covariance value
    if cov[0] < 0:
        print("Inverse relation: As one variable increases, the other tends to decrease.")
    elif cov[0] > 0:
        print("Direct relation: As one variable increases, the other tends to increase.")
    else:
        print("No linear relation: The variables show no consistent linear relationship.")

    print(f"Covariance: {cov[0]}")

elif val == 12:  # IF STATEMENT FOR CO-RELATION

    # Get user input for data points
    print("Enter the data points for X and Y in the format 'x1,x2,...,y'. Type 'done' to stop.")
    data_points = []
    while True:
        point = input("Data point: ")
        if point.lower() == 'done':
            break
```

```
    try:
        data = list(map(float, point.split(',')))
        data_points.append(data)
    except ValueError:
        print("Invalid input! Please try again.")

# Convert data_points to numpy array
data_points = np.array(data_points)

# Extract X and Y values from data_points
X = data_points[:, :-1]
Y = data_points[:, -1]

mean_X = np.mean(X)
mean_Y = np.mean(Y)

# Calculate the covariance
cov_x_and_y = np.dot((X - mean_X).T, Y - mean_Y) / len(X)

# Calculate the correlation coefficient
std_dev_X = np.std(X)
std_dev_Y = np.std(Y)
correlation_coefficient = cov_x_and_y / (std_dev_X * std_dev_Y)

# Extract the correlation coefficient value
corr_value = correlation_coefficient[0]

# Print the strength of the relationship based on correlation coefficient
if corr_value == 0:
    print("No relation: The variables show no consistent linear relationship.")
elif 0 < abs(corr_value) < 0.1:
    print("Very weak relation: The variables have a very weak linear relationship.")
elif 0.1 <= abs(corr_value) < 0.3:
    print("Weak relation: The variables have a weak linear relationship.")
elif 0.3 <= abs(corr_value) < 0.5:
    print("Moderate relation: The variables have a moderate linear relationship.")
elif 0.5 <= abs(corr_value) < 0.8:
    print("Good relation: The variables have a good linear relationship.")
elif 0.8 <= abs(corr_value) < 0.9:
    print("Excellent relation: The variables have an excellent linear relationship.")
elif abs(corr_value) >= 0.9:
    print("Exact relation: The variables have an exact linear relationship.")

print("Correlation Coefficient:", corr_value)
```

```
    elif val == 13:
        continue
    else:
        print("Invalid Entry\n")
        continue

elif x == 4:

    print("\n------------------------------ (Regression Modeling and Predictions and Regression estimates)"
          " -------------------------------\n")
    print("Press 1 -> Simple Linear Regression Model\n")
    print("Press 2 -> Multiple Linear Regression Model\n")
    print("Press 3 -> Go Back\n")

    e = int(input("Enter Choice:"))

    if e == 1:  # IF STATEMENT FOR LINEAR REGRESSION MODEL

        # Get user input for data points
        print("Enter the data points in the format 'x,y'. Type 'done' to stop.")
        data_points = []
        while True:

            point = input("Data point: ")
            if point.lower() == 'done':
                break
            try:
                x, y = map(float, point.split(','))
                data_points.append((x, y))
            except ValueError:
                print("Invalid input! Please try again.")

        # Convert data_points to numpy arrays
        data_points = np.array(data_points)

        # Extract x and y values from data_points
        x = data_points[:, 0]
        y = data_points[:, 1]

        # Visualize the data points
        plt.scatter(x, y, color='blue', label='Data points')
        plt.xlabel('X')
        plt.ylabel('Y')
```

```
    plt.title('User Input Data Points')
    plt.legend()
    plt.show()

    # Reshape x to 2D array
    x = x.reshape(-1, 1)

    # Fit a linear regression model
    model = LinearRegression()
    model.fit(x, y)

    # Visualize the linear regression line
    plt.scatter(x, y, color='blue', label='Data points')
    plt.plot(x, model.predict(x), color='red', label='Linear regression line')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title('Linear Regression')
    plt.legend()
    plt.show()

    # Get user input for prediction
    while True:
        x_value = input("Enter a value of X for prediction (type 'exit' to stop): ")
        if x_value.lower() == 'exit':
            break
        try:
            x_prediction = float(x_value)
            y_prediction_value = model.predict([[x_prediction]])
            print(f"Predicted Y for X = {x_prediction}: {y_prediction_value[0]}")
        except ValueError:
            print("Invalid input! Please enter a numeric value or type 'exit' to stop.")

elif e == 2:  # IF STATEMENT FOR MULTIPLE LINEAR REGRESSION MODEL

    # Get user input for data points
    print("Enter the data points in the format 'x1,x2,...,y'. Type 'done' to stop.")
    data_points = []
    while True:
        point = input("Data point: ")
        if point.lower() == 'done':
            break
        try:
            data = list(map(float, point.split(',')))
            data_points.append(data)
```

```
            except ValueError:
                print("Invalid input! Please try again.")

        # Convert data_points to numpy array
        data_points = np.array(data_points)

        # Extract X and y values from data_points
        X = data_points[:, :-1]
        y = data_points[:, -1]

        # Visualize the data points
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        ax.scatter(X[:, 0], X[:, 1], y, c='b', marker='o')
        ax.set_xlabel('X1')
        ax.set_ylabel('X2')
        ax.set_zlabel('Y')
        plt.title('User Input Data Points')
        plt.show()

        # Fit a multiple linear regression model
        model = LinearRegression()
        model.fit(X, y)

        # Get user input for prediction
        while True:
            try:
                input_values = input(
                    "Enter values of X1 and X2 for prediction separated by a comma (type 'exit' to stop): ")
                if input_values.lower() == 'exit':
                    break
                x1, x2 = map(float, input_values.split(','))
                x_prediction_value = np.array([[x1, x2]])
                y_prediction_value = model.predict(x_prediction_value)
                print(f"Predicted Y for X1 = {x1}, X2 = {x2}: {y_prediction_value[0]}")
            except ValueError:
                print("Invalid input! Please enter numeric values separated by a comma or type 'exit' to
stop.")
    elif x == 5:
        print("Program Ended\n")
        break
    else:
        print("Invalid Entry\n")
```

```
input("Press any key to continue...")  # Pausing
```

5.7 Output

Main Menu

```
---------------------------------------- ( MAIN MENU ) ----------------------------------------------

Press 1 -> Graphical and Tabular Representation

Press 2 -> Descriptive Statistical Measure (Absolute & Relative)

Press 3 -> Probability Methods/Distribution

Press 4 -> Regression Modeling and Predictions and Regression estimates

Press 5 -> End the Program

Enter Choice:
```

Graphical and Tabular Representation

```
--------------------------------- ( TYPES OF REPRESENTATIONS ) ------------------------------------

Press 1 -> Tabular Representation

Press 2 -> Graphical Representation

Press 3 -> Go Back
```

Tabular Representation

```
Enter the data set (Quantitative data only, space-separated numbers): 7 0 8 8 5 5 5 8 8 9 9 9 6 9 7 8 10 9 9

Frequency Distribution Table:
+----+---------+-------------+-----------------------+----------------------+
|    | Class   |  Frequency  |  Cumulative Frequency |  Relative Frequency  |
+====+=========+=============+=======================+======================+
| 0  | 0-1     |      1      |           1           |      0.0555556       |
+----+---------+-------------+-----------------------+----------------------+
| 1  | 1-2     |      0      |           1           |          0           |
+----+---------+-------------+-----------------------+----------------------+
| 2  | 2-4     |      0      |           1           |          0           |
+----+---------+-------------+-----------------------+----------------------+
| 3  | 4-5     |      3      |           4           |       0.166667       |
+----+---------+-------------+-----------------------+----------------------+
| 4  | 5-7     |      3      |           7           |       0.166667       |
+----+---------+-------------+-----------------------+----------------------+
| 5  | 7-8     |      5      |          12           |       0.277778       |
+----+---------+-------------+-----------------------+----------------------+
| 6  | 8-10    |      6      |          18           |       0.333333       |
+----+---------+-------------+-----------------------+----------------------+
```

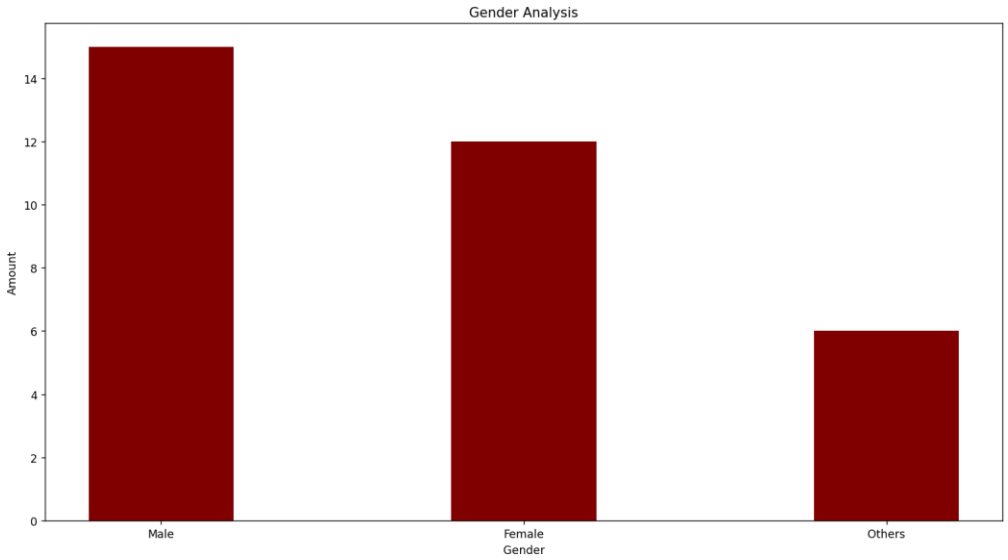
Graphical Representation

```
------------------------------ ( GRAPHICAL REPRESENTATION ) ---------------------------------

Press 1 -> Enter Data for Single Bar Chart

Press 2 -> Enter Data for Multiple Bar Chart

Press 3 -> Enter Data for Component Bar Chart

Press 4 -> Enter Data for Pie Chart

Press 5 -> Enter Data for Histogram

Press 6 -> Enter Data for Box Plot

Press 7 -> Go Back to Main Screen

Enter Choice:
```

Single Bar Chart (Input)

```
Enter the number of categories: 3
Enter the name of category 1: Male
Enter the value for Male: 15
Enter the name of category 2: Female
Enter the value for Female: 12
Enter the name of category 3: Others
Enter the value for Others: 6
Enter the name of X axis: Gender
Enter the name of Y axis: Amount
Enter the title of the plot: Gender Analysis
```

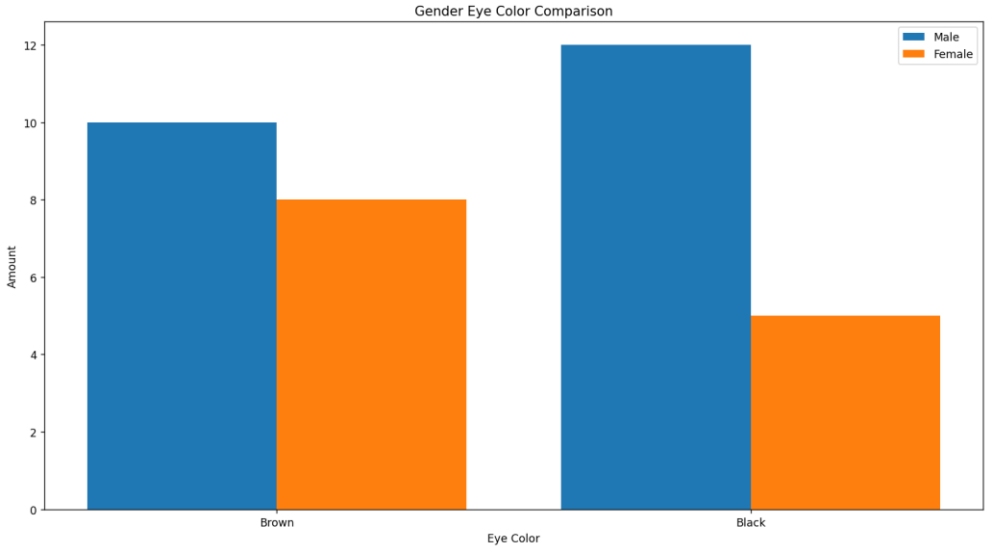
Single Bar Chart (Output)



Multiple Bar Chart (Input)

```
Enter the names of categories separated by spaces: Brown Black
Enter the name of X axis: Eye Color
Enter the name of Y axis: Amount
Enter the title of the plot: Gender Eye Color Comparison
Enter Name of Variable 1: Male
Enter Name of Variable 2: Female
Enter amount for Male in Brown: 10
Enter amount for Female in Brown: 8
Enter amount for Male in Black: 12
Enter amount for Female in Black: 5
```

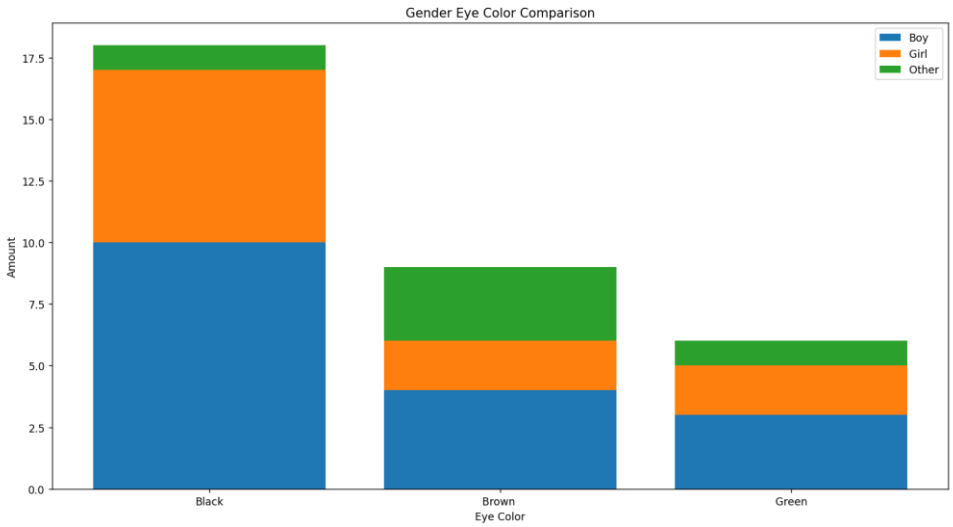
Multiple Bar Chart (Output)



Component Bar Chart (Input)

```
Enter the number of categories: 3
Enter the name of category 1: Black
Enter the scores for Black separated by spaces: 10 4 3
Enter the name of category 2: Brown
Enter the scores for Brown  separated by spaces: 7 2 2
Enter the name of category 3: Green
Enter the scores for Green separated by spaces: 1 3 1
Enter the label for round 1: Boy
Enter the label for round 2: Girl
Enter the label for round 3: Other
Enter the name of X axis: Eye Color
Enter the name of Y axis: Amount
Enter the title of the plot: Gender Eye Color Comparison
```

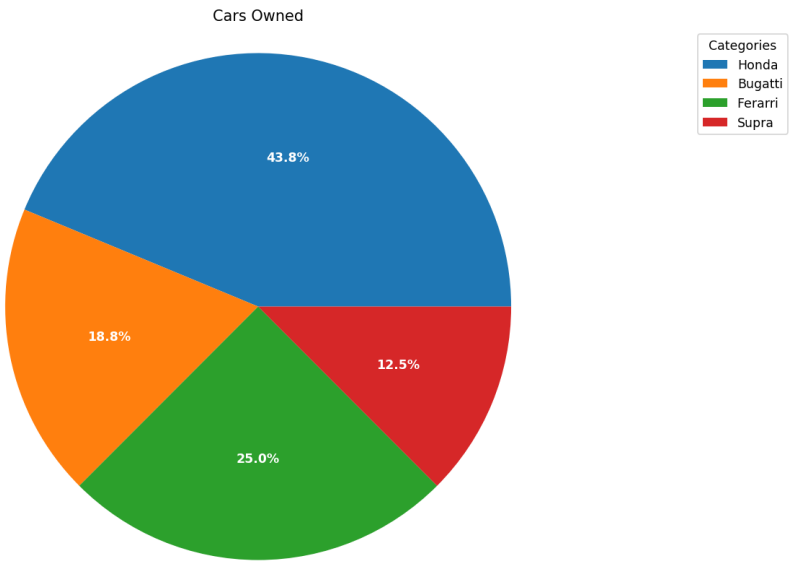
Component Bar Chart (Output)



Pie Chart (Input)

```
Enter the number of categories: 4
Enter the name of category 1: Honda
Enter the value for Honda: 7
Enter the name of category 2: Bugatti
Enter the value for Bugatti: 3
Enter the name of category 3: Ferarri
Enter the value for Ferarri : 4
Enter the name of category 4: Supra
Enter the value for Supra: 2
Do you want to display percentages in the pie chart? (yes/no): yes
Enter the title of the plot: Cars Owned
```

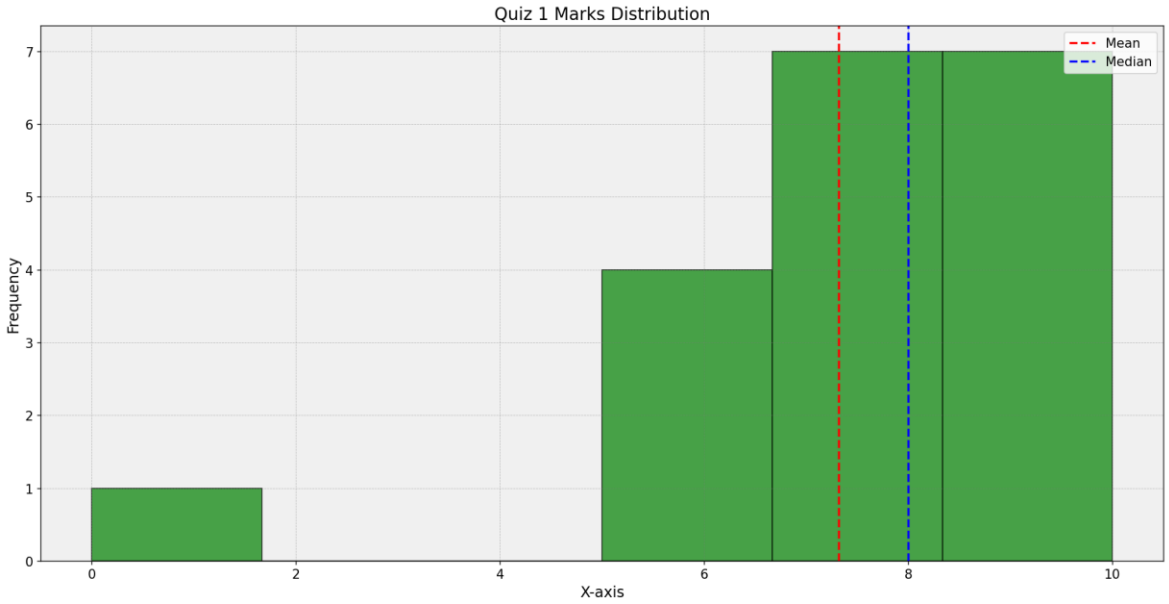
Pie Chart (Output)



Histogram (Input)

```
Enter the dataset (space-separated numbers): 7 0 8 8 5 5 5 8 8 9 9 9 6 9 7 8 10 9 9
Enter the title of the histogram: Quiz 1 Marks Distribution
```

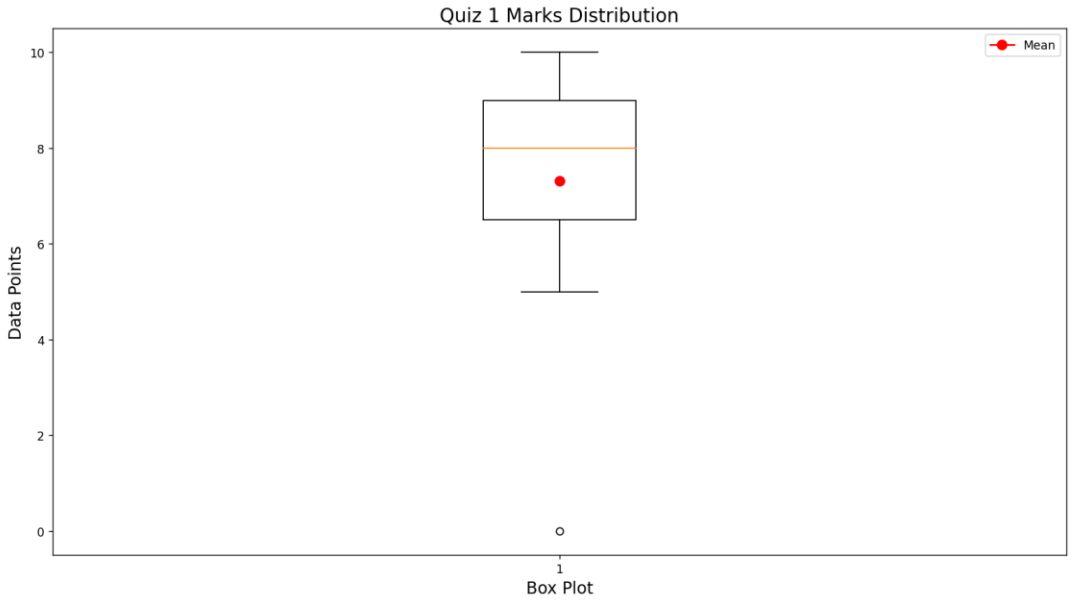
Histogram (Output)



Box Plot (Input)

```
Enter the title of the box plot: Quiz 1 Marks Distribution
Enter the data for the box plot (Quantitative data only): 7 0 8 8 5 5 5 8 8 9 9 9 6 9 7 8 10 9 9
```

Box Plot (Output)



Descriptive Statistics

```
-------------------------------- ( DESCRIPTIVE STATISTICS ) --------------------------------

Press 1 -> Mean

Press 2 -> Median

Press 3 -> Mode

Press 4 -> Range

Press 5 -> Variance

Press 6 -> Standard Deviation

Press 7 -> 3 Sigma Rule

Press 8 -> Quartile 1,2,3

Press 9 -> Inter quartile Range

Press 10 -> Coefficient of Range

Press 11 -> Coefficient of IQR

Press 12 -> Coefficient of Variation

Press 13 -> Go Back
```

Mean

```
Enter a list of numbers (space-separated): 7 0 8 8 5 5 5 8 8 9 9 9 6 9 7 8 10 9 9
Mean is: 7.315789473684211
```

Median

```
Enter a list of numbers (space-separated): 7 0 8 8 5 5 5 8 8 9 9 9 6 9 7 8 10 9 9
Median is: 8
```

Mod

```
Enter a list of numbers (space-separated): 7 0 8 8 5 5 5 8 8 9 9 9 6 9 7 8 10 9 9
Mode is: 9
```

Range

```
Enter a list of numbers (space-separated): 7 0 8 8 5 5 5 8 8 9 9 9 6 9 7 8 10 9 9
Range is: 10
```

Variance

```
Enter a list of numbers (space-separated): 7 0 8 8 5 5 5 8 8 9 9 9 6 9 7 8 10 9 9
Variance is: 5.450292397660819
```

Standard Deviation

```
Enter a list of numbers (space-separated): 7 0 8 8 5 5 5 8 8 9 9 9 6 9 7 8 10 9 9
Standard Deviation is: 2.3345861298441783
```

3 Sigma Rule

```
Enter a list of numbers (space-separated): 7 0 8 8 5 5 5 8 8 9 9 9 6 9 7 8 10 9 9
Data points within 3-sigma range:
[7.0, 8.0, 8.0, 5.0, 5.0, 5.0, 8.0, 8.0, 9.0, 9.0, 9.0, 6.0, 9.0, 7.0, 8.0, 10.0, 9.0, 9.0]

Data points outside 3-sigma range:
[0.0]
```

Quartile 1,2,3

```
Enter a list of numbers (space-separated): 7 0 8 8 5 5 5 8 8 9 9 9 6 9 7 8 10 9 9
Q1 (25th percentile): 6.0
Q2 (Median, 50th percentile): 8.0
Q3 (75th percentile): 9.0
```

Intel Quartile Range (IQR)

```
Enter a list of numbers (space-separated): 7 0 8 8 5 5 5 8 8 9 9 9 6 9 7 8 10 9 9
Inter quartile Range (IQR): 3.0
```

Co-efficient of Range

```
Enter a list of numbers (space-separated): 7 0 8 8 5 5 5 8 8 9 9 9 6 9 7 8 10 9 9
Coefficient of Range: 100.0
```

Co-efficient of IQR

```
Enter a list of numbers (space-separated): 7 0 8 8 5 5 5 8 8 9 9 9 6 9 7 8 10 9 9
Coefficient of IQR: 16.129032258064516
```

Co-efficient of Variation

```
Enter a list of numbers (space-separated): 7 0 8 8 5 5 5 8 8 9 9 9 6 9 7 8 10 9 9
Coefficient of Variance: 31.060479345212812
```

Probability Methods/Distributions

```
------------------------------ ( PROBABILITY METHODS/DISTRIBUTIONS ) ------------------------------

Press 1 -> Simple Probability

Press 2 -> Counting Technique

Press 3 -> Factorial Method

Press 4 -> Multiplicative Technique

Press 5 -> Permutation

Press 6 -> Combination

Press 7 -> Binomial Probability Distribution

Press 8 -> Poison Probability Distribution

Press 9 -> Hyper-geometric Probability Distribution

Press 10 -> Uniform Probability Distribution (Standardization)

Press 11 -> Covariance

Press 12 -> Correlation

Press 13 -> Go Back
```

Simple probability

```
Enter Favourable Outcome: 10
Enter Total Outcomes: 25
Simple Probability: 0.4
```

Counting Technique

```
Enter Possible Outcome m1: 30
Enter Possible Outcome m2: 3
Total Number of Possibilities: 90
```

Factorial Method

```
Enter Objects: 5
Enter Arrangements: 5
Total Number of arrangements: 120
```

Multiplicative Technique

```
Enter the value of n (total objects): 5
Enter the value of r (arrangements): 3
Number of arrangements: 60
```

Permutation

```
Enter the value of n (total objects): 10
Enter the value of r (arrangements): 3
Number of permutations: 720
```

Combination

```
Enter the value of n (total objects): 10
Enter the value of r (combinations): 3
Number of combinations: 120
```

Binomial Probability Distribution

```
Enter the number of trials (n): 10
Enter the probability of success in each trial (p): 0.4
Enter the number of successes (k): 3

The probability of getting exactly 3.0 successes in 10.0 trials with a success probability of 0.4 is: 0.21499
Expected Value: 4.0
Variance Value: 3.6
Standard Deviation Value: 1.8973665961010275
```

Poison Probability Distribution

```
Enter the number of trials (n): 120
Enter the probability of success in each trial (p): 0.03
Enter the number of events (k): 10
The probability of getting exactly 10.0 events with an average rate of 3.60000 is: 0.00275
Expected Value: 3.60000
Variance Value: 3.60000
Standard Deviation Value: 1.89737
```

Hyper Geometric Probability Distribution

```
Enter the population size (N): 10
Enter the number of successes in the population (M): 6
Enter the number of trials (n): 2
Enter the number of successes in the sample (k): 2
The probability of getting exactly 2 successes in 2 trials with a population size of 10 and 6 successes in the population is: 0.33333
Expected Value: 1.20000
Variance Value: 0.47407
Standard Deviation Value: 0.68853
```

Uniform Probability Distribution

```
Enter Value of x: 50
Enter Mean: 55
Enter Standard Deviation: 15
Z-score: -0.3333333333333333
p-value: 0.6305586598182363
```

Covariance

```
Enter the data points for X and Y in the format 'x1,x2,...,y'. Type 'done' to stop.
Data point: 1,2,3,4,5
Data point: 5,4,3,2,1
Data point: done
Inverse relation: As one variable increases, the other tends to decrease.
Covariance: -4.0
```

Correlation

```
Enter the data points for X and Y in the format 'x1,x2,...,y'. Type 'done' to stop.
Data point: 1,2,3,4,5
Data point: 5,4,3,2,1
Data point: done
Exact relation: The variables have an exact linear relationship.
Correlation Coefficient: -1.6329931618554523
```

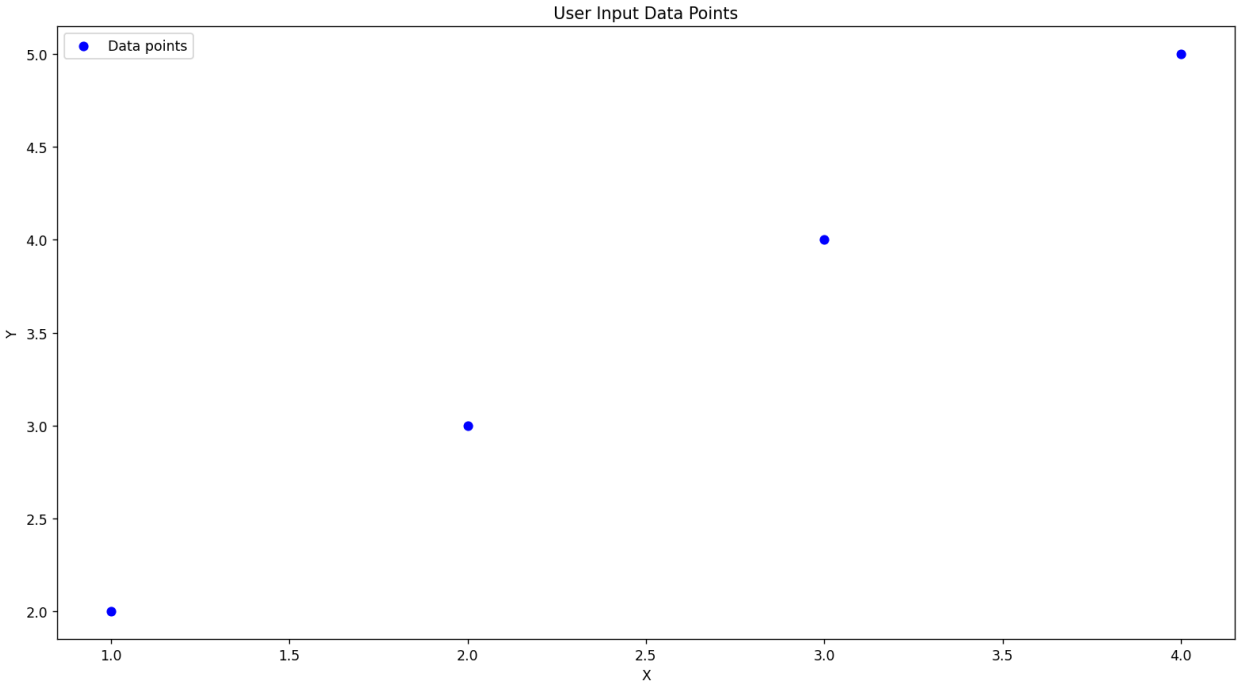
Regression Modeling and Regression estimates and Predictions

```
------------------------------ (Regression Modeling and Predictions and Regression estimates) ------------------------------

Press 1 -> Simple Linear Regression Model

Press 2 -> Multiple Linear Regression Model

Press 3 -> Go Back

Enter Choice:
```

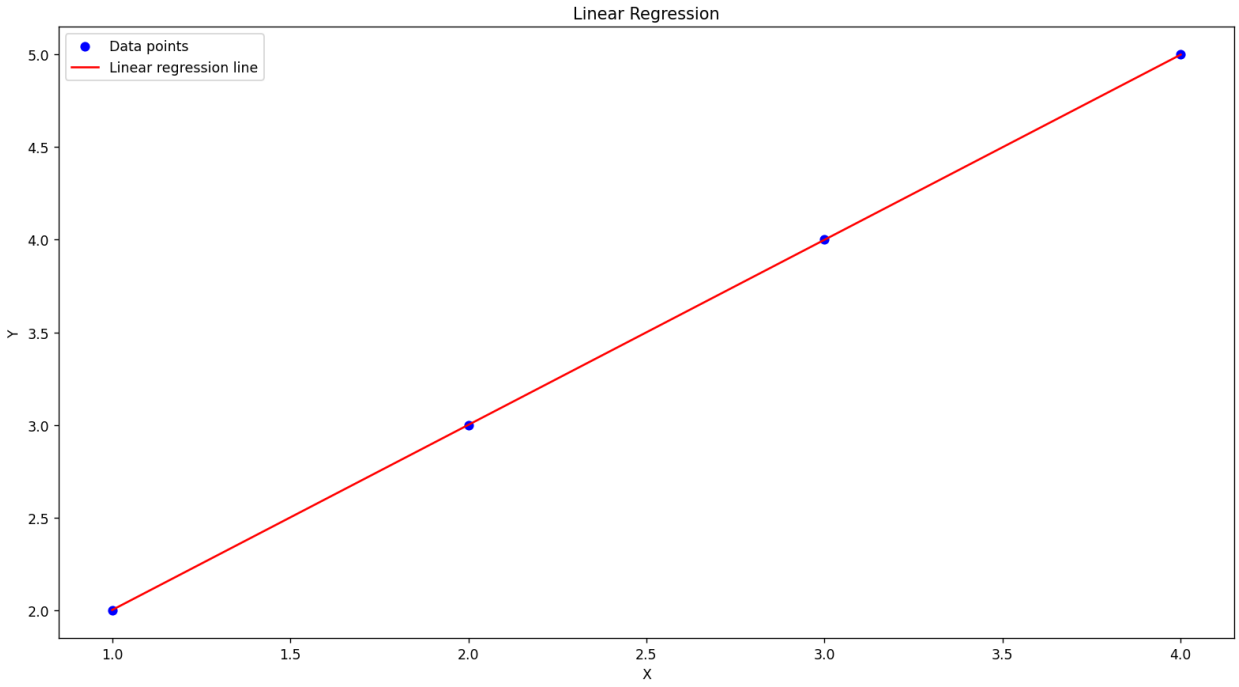
Simple Linear Regression Model (Input)

```
Enter the data points in the format 'x,y'. Type 'done' to stop.
Data point: 1,2
Data point: 2,3
Data point: 3,4
Data point: 4,5
Data point: done
```

Simple Linear Regression Model (Output 1)



Simple Linear Regression Model (Output 2)



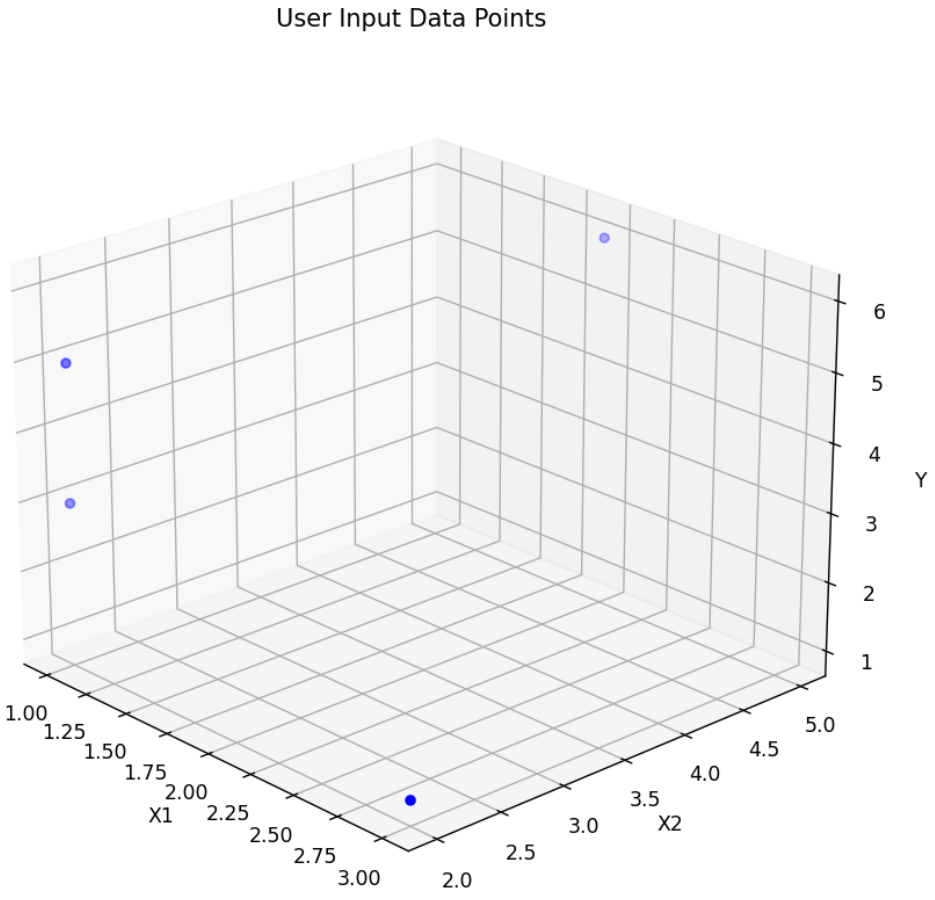
Simple Linear Regression Model (Prediction)

```
Enter a value of X for prediction (type 'exit' to stop): 3.4
Predicted Y for X = 3.4: 4.4
Enter a value of X for prediction (type 'exit' to stop): 10.2
Predicted Y for X = 10.2: 11.2
Enter a value of X for prediction (type 'exit' to stop): exit
```

Multiple Linear Regression Model (Input)

```
Enter the data points in the format 'x1,x2,...,y'. Type 'done' to stop.
Data point: 1,2,3
Data point: 3,2,1
Data point: 1,2,5
Data point: 2,5,6
Data point: done
```

Multiple Linear Regression Model (Output)



Multiple Linear Regression Model (Prediction)

```
Enter values of X1 and X2 for prediction separated by a comma (type 'exit' to stop): 10,21
Predicted Y for X1 = 10.0, X2 = 21.0: 12.666666666666654
Enter values of X1 and X2 for prediction separated by a comma (type 'exit' to stop): 1,12
Predicted Y for X1 = 1.0, X2 = 12.0: 15.666666666666657
Enter values of X1 and X2 for prediction separated by a comma (type 'exit' to stop): exit
```

5.8 Conclusion

The application provides a comprehensive suite of statistical and machine learning tools to facilitate data analysis, exploration, and modeling. From descriptive statistics to inferential analysis and machine learning algorithms, users can gain valuable insights and make data-driven decisions.

Please note that this report only provides an overview of the application's functionalities, and there may be additional features and capabilities available within the application. Users are encouraged to explore the application and leverage its powerful tools to analyze and understand their data effectively.