Title

Predictive Modeling of Heart Attack Risk: An Analysis Using Probability Theory and Simulation Techniques

Abstract

The project focuses on the implementing the basic concepts of simulation and statistical analysis. The statistical analysis , visualization , Central limit theorem (CLT) verification, outlier detection and probability calculation for continuous and discrete distributions is being determined in the project. The simulation of normal continuous random variable is being performed. The visualization made by help of histogram helps in providing insights into the behaviors of sample means. Z-score is used to determine the outliers in the randomly generated samples and visualized using box-plot. The simulation of discrete random variables involves simulation using Poisson distribution. Through the use of transition matrix simulation, modeling of recurring events, ergodicity analysis, sensitivity analysis, and visualization of Markov chain behavior, the project takes a deeper look at Markov chains. It also looks into variance reduction strategies, contrasting and comparing approaches like antithetic variates, control variates, and importance sampling to improve the accuracy and efficiency of simulations. The real data is then analyzed with the help of Baye's theorem and Joint Distribution analysis.

Chapter 1: Introduction

Simulation is a powerful tool for modeling and analyzing complex systems. This project explores the fundamental principles and applications of simulation, delving into a complex field. This study aims to unravel the complex tapestry of continuous and discrete random variables, Markov Chains, variance reduction techniques, and real data analysis. In simulation data analysis the simulation of continuous random variables such as normal exponential uniform or gamma as more emphasized. This involves implementing techniques like statistical analysis, virtualization, Central Limit Theorem Verification, Outlier detection and probability calculations for the random variables. The project also engages with simulating data from distribution like Poisson distribution which also involves statistical analysis, virtualization, Central Limit Theorem Verification, Outlier detection and probability calculations. This study focuses on the application of these simulations and understanding that inherent properties. The Markov chains incorporates transition matrix simulation recurrent events modelling ergodicity exploration sensitivity analysis and visualizations. To provide insight into long-term behavior and steady-state probabilities, this section aims to separate the dynamics of systems that are defined by state transitions and probabilities. The variance reduction techniques involved methodologists such as sampling and control variates. The project investigates how these methods might be used to improve simulation accuracy and efficiency, particularly when modeling complex systems. Comparing and contrasting different simulation techniques, such as Variance Reduction and Markov Chains, offers a comprehensive perspective. This comparative analysis is essential to comprehending the advantages and disadvantages of each strategy. The project also explores implementing Bayes' Theorem using a real data set. The joint distribution analysis on real data sets applies conditional probabilities and independence concepts. Using Factor analysis further involves into the complex relationship within the datasets. Through this analysis we can analyze the data over the different random variable. Overall, this project's aim to be a compressive exploration of probability theory and at simulations.

Chapter 2: Data Description

The Heart Attack Risk Prediction Dataset is a valuable resource for exploring the complex dynamics of heart health and its predictor. Heart attacks are still a major global health concern so for a better understanding of the causes and possible preventative factors as necessary this collection of data captures a wide range of characteristics such as age, blood pressure, cholesterol, smoking status, exercise routines, food preferences and more with the goal of clarifying the entry gate interactions between these factors and predicting the risk of a heart attack. The data is collected froma a online source kaggle. Below is the link mentioned.

This data sets provides array of features relevant to heart health and lifestyle choices of patient specific details such as age, gender, cholesterol levels, blood pressure, heart rate, and indicators like diabetes, family history, smoking habits, obesity, and alcohol consumption. The data set consists of 8763 records from patients around the globe denoting the presence or absence of a heart attack risk. The columns in the dataset are as below:

Patient ID - Unique identifier for each patient Age - Age of the patient
Sex - Gender of the patient (Male/Female)
Cholesterol - Cholesterol levels of the patient
Blood Pressure - Blood pressure of the patient (systolic/diastolic)
Heart Rate - Heart rate of the patient
Diabetes - Whether the patient has diabetes (Yes/No)
Family History - Family history of heart-related problems (1: Yes, 0: No)
Smoking - Smoking status of the patient (1: Smoker, 0: Non-smoker)
Obesity - Obesity status of the patient (1: Obese, 0: Not obese)
Alcohol Consumption - Level of alcohol consumption by the patient (None/Light/Moderate/Heavy)
Exercise Hours Per Week - Number of exercise hours per week
Diet - Dietary habits of the patient (Healthy/Average/Unhealthy)
Previous Heart Problems - Previous heart problems of the patient (1: Yes, 0: No)
Medication Use - Medication usage by the patient (1: Yes, 0: No)
Stress Level - Stress level reported by the patient (1-10)
Sedentary Hours Per Day - Hours of sedentary activity per day
Income - Income level of the patient
BMI - Body Mass Index (BMI) of the patient
Triglycerides - Triglyceride levels of the patient
Physical Activity Days Per Week - Days of physical activity per week
Sleep Hours Per Day - Hours of sleep per day
Country - Country of the patient
Continent - Continent where the patient resides
Hemisphere - Hemisphere where the patient resides
Heart Attack Risk - Presence of heart attack risk (1: Yes, 0: No)

DataSet Link: https://www.kaggle.com/datasets/iamsouravbanerjee/heart-attack-prediction-dataset/data

Chapter 3: Methodology

The following are the moethodologies and statistical tool used for the analysis of the dataset:

Descriptive Statistics

Mean : The mean, or average, is the sum of all values in a dataset divided by the number of observations. It represents the central value of a data set or distribution through a single value.
Variance: Variance in statistics refers to the expected deviation between values in a specific data set. It is the average of the squared differences from the mean.
Standard Deviation: Standard deviation is the measure of the spread of a data set. It is the square root of the variance.
Mode: The mode is the value that appears most frequently in a dataset.
First Quantile(Q1): First Quantile is the median of the lower half of a dataset, separating the lowest 25% of values.
Second Quantile(Q2): The second quantile, commonly known as the median, is a measure of central tendency that divides a dataset into two equal halves. It is the middle value of the dataset.
Third Quantile(Q3): The third quantile is the median of the upper half of a dataset, separating the lowest 75% of values.
Skewness: Skewness is a measure of the asymmetry or lack of symmetry in a distribution. It indicates the degree and direction of skew (departure from horizontal symmetry) in the data.
Kurtosis: Kurtosis measures the tail-heaviness or sharpness of a distribution peak. It describes the shape and concentration of data in the tails relative to the center of the distribution.

Simulation of Random Data

Continuous and Discrete Analysis: Here we simulate the continuous random variable using the NumPy python module. For the discrete random variable analysis we use the poisson distribution and geometric distribution.

Markov Chains

Markov Chains: Model and simulate systems where the outcome depends on the previous state.
Transition Matrix Simulation:It models transitions between states in a system using a transition matrix. It multiplies the

current state vector by the transition matrix to predict the next state.
Recurrent Events:It models events that occur repeatedly over time, such as arrivals, service times, and departures in a queuing system. Here we define transition probabilities for events and simulate their occurrences over time.
Ergodicity: Ergodicity investigates the long-term behavior of a Markov Chain and compares it to the steady-state probabilities.Here we simulate the Markov Chain and compared the time-averaged behavior to the steady-state probabilities.
Sensitivity Analysis: Sensitivity analysis studies how small changes in parameters affect the behavior of the system.Here we have varied the transition probabilities or initial conditions and observe the impact on the system's behavior.

Variance Reduction

Variance Reduction Techniques:It helps to improve the efficiency and accuracy of simulations by reducing the variance of the estimators.
Importance Sampling:It adjusts the sampling to allocate more samples to regions that contribute more to the overall result. Here we redefined the integrand by introducing a weighting function to emphasize important regions.
Control Variate: In control variate a known variable (control variate) is introduced to the simulation to reduce the variance of the estimate. The control variate is chosen to be correlated with the original variable, helping to cancel out some of the variability. In the code the control variation coefficient is used for 0.05.
Anithetic Variates: Reduces variance by generating pairs of antithetic (opposite) random variables and averaging their contributions. For each random variable,we have generated its antithetic counterpart and used the average of the pair.

Bayes' Theorem

Bayes' Theorem: Bayes' Theorem is a mathematical formula that calculates the probability of an event based on prior knowledge of conditions related to the event. Bayes' Theorem is used to predict the probability of the event on the Heart Attack Risk Prediction Dataset. For the theorem the probability of having a heart attack risk given the stress level of the patient is 9 is being calculated.
Conditional Probability:The probability of an event occurring given that another event has already occurred.
Independence events: Events are considered independent if the occurrence of one event does not affect the occurrence of another.
Joint Probability Distribution: Joint distribution describes the probabilities associated with multiple random variables

occurring simultaneously. Denoted by P(X=x,Y=y)
Correlation Analysis: Correlation measures the strength and direction of a linear relationship between two variables.
Statistical Test for Normality: Kolmogorov-Smirnov Test: A non-parametric test that compares the cumulative distribution function of a sample with a specified distribution. For the Kolmogorov-Smirnov test, the test statistic is calculated based on the maximum difference between the sample and theoretical distribution. A significant result in these tests may indicate that the data significantly deviates from normality.
Shapiro-Wilk Test: A test that assesses whether a sample comes from a normally distributed population. For the Shapiro-Wilk test, the test assesses the correlation between the data and a normal distribution. Normality assumptions are crucial for many statistical methods, and violations may impact the reliability of statistical inferences.

Factor Analysis

Factor Analysis: Factor Analysis is a statistical technique used to explore the underlying structure in a dataset and identify the latent factors that contribute to the observed patterns and variations. It assumes that observed variables can be modeled as linear combinations of underlying factors plus a unique factor for each variable.
Factor Loading: Represents the strength and direction of the relationship between an observed variable and a latent factor. It indicates how much the observed variable is influenced by the latent factor. Communality: The proportion of the variance in an observed variable that is accounted for by the latent factors. It is the square of the factor loading.

Chapter 4: Simulation Data Analysis

4.1 Simulating Data Analysis

4.1.1 Simulating Continuous Random Variable

In the below code, Python code using NumPy and Matplotlib to simulate data from a Normal Random variable, perform statistical analysis, visualize the data, verify the Central Limit Theorem, detect outliers, and calculate probabilities. I will be using the sample size 1000. Also the code uses histogram to visulaize the data. The outliers in the data are determined using the box plot. The probability for random variable between -2<X<1 and probability for X>=2 is being caluclated for the conitnuous random variable.

In [3]:
```
# Importing modules required for simulation
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import scipy.stats as st
```

In [4]:
```
# Generating random data
data = np.random.normal(0,1,1000) # mean = 0 and standard deviation = 1
```

In [5]:
```
print(data)
```

```
[ 6.74190727e-01 -1.44029117e+00 -6.46939410e-01  3.23786484e-01
 -7.97510973e-01 -1.45794300e+00  1.34890084e+00  1.01119797e-01
  9.52244678e-01  4.04258256e-01 -3.43771573e-01  4.76027871e-01
 -9.91480327e-02 -8.44261398e-01  5.38254498e-01 -1.09952015e+00
  2.26876427e-01  1.89106934e-01  1.05921913e+00 -2.56033719e+00
  1.00878738e+00  2.71559049e-01 -2.87801116e-01  5.24924991e-01
  1.50549829e+00  6.47063974e-01 -1.48845028e-01  1.16226045e+00
  8.07981113e-01  7.32539172e-01  4.53924683e-01 -3.46863407e-01
 -1.20264720e+00  2.61889509e-01  6.05703405e-01  3.25896915e-01
 -6.22002811e-02 -6.77926321e-02 -6.25139617e-02  2.34993829e+00
 -1.96653615e+00  1.15616676e+00 -5.38040275e-01  1.00470382e+00
 -1.71759554e+00  9.72868715e-01  3.84896199e-01 -2.08700542e-01
 -1.87673980e+00 -9.20523483e-02 -1.71987615e+00  1.92017483e+00
 -2.85554222e-01 -5.93497245e-01  1.53670936e+00  7.55465419e-01
  1.05228120e-01  1.32055618e+00 -3.26622780e-02  9.04929851e-01
 -1.17658525e-01 -1.08228929e-01  1.32752070e+00  1.97678812e+00
 -2.92225290e-01 -9.19793413e-01 -2.12902843e+00  9.21719120e-01
  3.45678211e-01  3.49244823e-01  1.47381293e+00  1.49618089e+00
  2.77302081e-01  3.91932797e-01  1.98001132e+00  5.50185929e-01
 -3.48928208e-01  3.48081454e-01  8.32554830e-01 -3.71596470e-01
 -1.37377488e+00 -1.70517071e+00 -1.16128939e-01  7.80143725e-01
 -2.37482033e+00  2.51442350e-01 -7.90020735e-01 -5.55472926e-01
```

```
 1.99704487e+00  6.19756187e-01 -2.37393733e+00  1.20294753e+00
-3.27093867e-01 -6.96363212e-02 -9.05686888e-01 -1.53472853e+00
 1.31844676e+00 -6.67442972e-01 -2.78632481e+00  4.62353646e-01
 3.13723288e-01  2.07640706e-02  1.72776108e-01  9.49420632e-01
-4.14337161e-01 -8.41094023e-01  4.54572653e-02 -3.36927333e-02
 2.44735650e+00 -1.09296073e-01  1.58330952e+00  1.39555744e+00
-2.77376940e-01 -1.10817633e-01  2.49643116e-01 -1.88454579e-01
-1.61689062e-01  3.42211311e-01 -9.13074403e-01 -1.38365785e-01
 6.06913157e-02 -5.54702821e-01 -1.38574815e-01 -1.64196680e+00
 9.20646512e-01  1.35239878e+00 -7.07064179e-01  2.58358722e+00
-4.30284833e-01 -3.49947020e-01  5.05928264e-01 -1.94266732e+00
-5.96111537e-01  3.46209122e-01 -4.46613675e-01  7.36824942e-01
-9.24431967e-01  5.11251635e-01  1.33782758e+00  3.70505238e-01
-2.93293952e-01  2.45500344e+00  7.12451229e-01  2.93141756e-01
-5.86018747e-01  2.09477080e+00 -5.36422516e-01 -6.42116620e-01
 8.02552068e-01 -1.69655340e-01  1.09968479e+00  8.49801801e-01
 6.27595278e-01 -1.79236184e+00  1.11584140e+00 -5.25369373e-01
-7.32563214e-01  2.35134203e+00 -6.97104002e-01  2.70018041e+00
 4.49237825e-01  1.36004919e+00 -7.02626921e-02 -1.30467667e+00
-4.12218272e-01  1.78857836e+00 -3.70729657e-01 -1.60227178e+00
-1.60468314e-01 -2.01087177e+00  2.57821621e-01  3.01008269e-01
 3.37855216e-02  1.37076351e+00 -3.34948676e-01 -1.08688790e+00
 3.84451228e-01  2.04827322e+00 -9.15569228e-01 -6.59485746e-01
-3.01783845e-01  1.04642989e+00  7.06796861e-01 -1.00265291e+00
-1.28804899e+00 -1.28626344e+00  9.38937167e-02  1.81274159e-01
-1.11889330e+00  8.80804843e-01  1.99733501e+00 -5.14207499e-01
 1.31959395e-01  1.21082214e+00 -5.36687333e-02 -7.65341933e-01
-7.79100222e-01  1.00810928e+00 -3.12094199e-01 -1.44772482e+00
-7.42284547e-01  1.34704085e+00 -6.54726773e-01 -1.63799916e-01
 6.07763000e-02  3.48008676e-01  5.19657502e-01  7.51184688e-01
-7.00832497e-01 -1.55834953e+00  6.90663744e-01 -8.78033547e-01
-7.97363655e-01  8.34085877e-01  2.41580392e-01 -1.26120031e+00
-6.04928993e-01  8.34783471e-01 -8.21265171e-01 -7.72382178e-01
-1.49925288e+00 -1.75529467e-01 -2.75384549e-01  9.59891494e-01
-2.35397098e-01  1.03255733e-01  1.02909231e+00 -6.90253969e-01
-6.71898843e-02 -7.64711723e-01  7.44347479e-01 -3.44811284e-01
 2.73392673e-01 -4.70923348e-01 -3.25444555e-02 -1.50562202e+00
-1.75228809e+00 -1.71412975e+00 -7.50303439e-01  6.63397009e-01
-1.10759251e-01  9.73333086e-01 -6.12712473e-01  1.02233537e+00
-1.67923216e+00  1.61265680e+00  7.28541200e-01  4.89100889e-01
 6.47556868e-01 -1.71874930e-01  7.09322574e-01  3.34478233e-01
-6.56165094e-01  3.06752047e-01  5.36147813e-01  5.00667044e-01
 1.43104157e-01  5.92724923e-01  1.13369453e+00 -1.43509596e-01
```

```
 2.13174723e+00  4.14602212e-01 -2.68627115e-01 -1.21870787e+00
 9.17547906e-01  1.64145003e+00 -5.84428292e-01  1.14238258e+00
 6.76248981e-01 -1.12875117e+00  1.04143362e+00  2.39086382e+00
 8.50389870e-02  9.50884875e-01  1.19935074e-01 -1.71696565e+00
 1.32061848e+00  9.06431666e-01  1.67416230e-01 -6.91249046e-01
-3.34306292e-01 -1.04725501e+00  2.00015586e-01 -3.49111119e-01
 1.41749592e-01  1.92576585e-01 -9.88014939e-02  3.04510819e-01
-1.28958876e+00 -4.21226258e-01 -1.05890273e+00 -3.49851868e-01
-1.10971284e+00  2.01426946e-01  5.46373034e-01 -7.34402175e-01
 9.81057690e-01 -1.37765434e-01  1.44809766e+00  1.77549629e-01
-2.05454195e+00 -7.22649795e-01 -6.85513719e-01  5.41274362e-01
-8.25706022e-01  2.48197019e-01  4.87105120e-01 -2.42885798e-01
-1.46190516e+00  7.21869523e-02 -1.79771976e-01  1.00402563e+00
 7.02268735e-01  4.21440570e-01 -1.16524862e+00  1.05750151e-01
 5.90690983e-01  1.47724826e+00  1.53824988e+00  3.38343490e-01
 3.90587663e-01 -2.98824320e-01  1.28689506e+00  1.09142781e+00
-5.52122495e-01 -9.88472918e-01  9.63009627e-01  8.66361692e-01
-5.21552067e-02 -1.84470038e+00  2.18278973e-01  1.21707612e+00
 7.21616051e-01  5.52941240e-01 -3.69975159e-01  1.30245742e+00
-6.99283490e-03 -3.38041216e-01 -1.68093131e-01  1.76573261e+00
 1.61051977e-01 -1.30114870e+00 -2.20332782e+00 -1.77671811e+00
 2.17411721e-01 -1.60943580e-01 -1.56234940e+00  1.00202424e+00
 1.00652013e+00 -6.88763393e-01  2.95490062e-01 -3.89277271e-01
 1.18073163e+00  5.37670428e-01 -1.16279747e+00  1.79244798e+00
 9.58050649e-01  3.18091588e-01  3.78955578e-01 -4.38334355e-01
 6.12050117e-01 -1.95456625e+00  1.56228862e+00 -8.39871222e-01
 1.55187043e-02  1.27171928e+00 -1.12230183e+00 -1.03030719e-01
-1.58972907e+00 -2.82961147e-01 -6.65778820e-01 -5.50251143e-01
 5.02664730e-01 -9.81461515e-01 -8.54581315e-01 -4.34767451e-01
 1.87392503e-01  8.06006960e-01 -2.05351980e-01 -4.03011508e-01
-3.15671569e-01  8.61757236e-01 -2.09151981e+00  7.66581256e-01
-2.99963436e-01  1.18543915e+00 -2.33927179e+00  7.65972453e-01
-4.75564106e-01  1.83330254e+00 -2.00956319e-01  1.52387643e+00
-1.30116977e+00 -2.14525458e+00 -4.51502642e-01  3.45919114e-01
 4.43553910e-01  1.22488277e+00  8.00106257e-01  5.93784397e-01
-1.36679803e+00  1.51156733e+00 -1.34161433e+00  1.03903329e+00
-1.09648203e+00 -5.90585266e-01 -5.28818497e-01 -1.70789558e-02
-4.00006291e-01  1.04230112e+00  2.60577788e-01 -9.91728986e-01
 2.52181976e-01  6.14588446e-01  5.70649260e-02 -8.51382751e-01
 1.31953065e+00  5.72603075e-01  1.97868990e+00  9.07359038e-01
-1.47475577e+00  9.19484869e-01 -3.07051922e-01  1.70512649e+00
-1.42253426e+00 -1.21284662e-01  1.41275320e+00  2.64764707e-01
 1.02744524e-01  3.03322602e-02  3.18969269e-01  8.39971065e-01
```

```
 6.51558791e-01  1.62754155e+00 -2.65620987e-01 -2.61876506e-01
-4.40681128e-01 -1.80177548e-01 -4.68452897e-01  2.39534115e-01
-9.83500557e-01  1.00737337e-01  2.62891526e-02 -5.15850256e-01
 6.04204569e-01 -5.00241131e-01  1.67507354e-01 -3.12126963e-01
-5.97818449e-01 -8.11735891e-01 -4.53446373e-01  1.54296929e-02
-8.11373611e-01 -2.86545861e-01 -1.33686122e+00  3.81959713e-01
 1.06844893e+00 -5.11850893e-01  2.51310415e-01  1.27058290e+00
 8.67851519e-01  7.71865771e-01 -4.14360223e-01  1.17994368e+00
 2.47789995e-01 -4.92930823e-01 -7.34134837e-01  3.36924454e-02
-1.40197884e-01  1.38400975e+00 -3.92144137e-02 -1.54773788e+00
 1.22917178e+00 -3.96567347e-01  7.32167133e-01 -2.38516095e+00
-1.97290699e+00 -5.74578938e-01  3.78432730e-02 -7.09585819e-01
 5.35651981e-01  4.77121698e-01 -4.28389355e-01 -1.22445405e+00
 8.84163357e-01 -4.50459087e-01 -9.75636342e-02  1.30963263e+00
 1.69811099e+00 -1.90581860e-01 -8.01080047e-01  6.13010811e-02
 4.66041691e-02  3.50609896e-02 -1.90150747e-03  1.50409283e+00
-5.34457939e-01  1.61565398e+00  1.15589891e+00  1.82307255e+00
 1.33180665e-02 -4.95559925e-01 -5.40535416e-01 -2.55040079e-01
-4.83119454e-01  8.00066378e-01  5.45916490e-01  9.75426764e-01
-3.93769889e-01  1.35150318e+00  1.35732775e+00 -1.04521981e+00
-1.93218682e-01 -1.32360476e+00 -9.82984552e-02 -7.02238014e-01
 1.82952943e-01 -6.60683404e-01  5.80491652e-01  3.99605818e-01
-9.13359371e-01 -1.15485765e+00 -6.80494733e-01 -1.37068922e-01
 3.16496443e-01  1.75946103e+00  1.80249948e-01  5.42280799e-01
 1.31510467e+00 -1.58429381e+00  8.29126096e-01  5.13880920e-02
 1.86084377e-01 -8.56137126e-01 -1.25192202e+00 -1.85368129e+00
-7.07639460e-03  2.02566311e+00  1.47599594e+00  6.01009268e-01
 7.73865685e-01  1.05483731e-02 -6.78945687e-01  3.34388209e-01
-1.10891185e+00  7.95766871e-01 -6.72057190e-01 -5.19962722e-01
 8.29710842e-01 -2.65048640e+00  1.22760176e-01 -1.29235370e+00
 1.99780607e+00 -2.38740541e+00  3.78917304e-01 -8.58476824e-01
 1.01213455e+00 -7.77527417e-01  9.83872319e-01  7.70743272e-01
-1.29876110e+00  7.12733915e-01 -9.44855768e-01  8.31281842e-01
-7.32592312e-01 -1.94515134e+00  1.05709586e+00  1.37124215e+00
-1.28618624e+00 -5.64813074e-01  1.60217473e+00  3.76000106e-02
-1.90437460e+00  3.77847308e-02 -4.86143442e-01 -1.02627795e+00
 1.99235488e+00  1.16234132e-01  1.28203689e+00  1.98123570e+00
-1.64630311e+00 -3.56963191e-01 -3.58584875e-01 -1.58643917e+00
-7.77346948e-01  1.28140242e+00 -2.64795920e-01 -9.43525988e-01
-3.57373990e-01  1.33312431e+00  8.97846466e-01 -6.73367274e-01
-4.92435519e-01  1.92572062e+00 -7.85158446e-01 -2.95129229e-01
-1.07970261e-01  9.58779030e-01  2.31027772e+00  6.62735004e-01
-2.30175620e-01 -8.51739272e-01  1.84235473e+00 -1.12377929e+00
```

```
 1.05716574e+00  3.22619033e-01  1.55397151e+00 -5.61900833e-01
-1.64040351e+00  5.51983402e-01  9.29154500e-01 -5.18952240e-01
 6.84874693e-01  3.32359131e-01 -7.08857066e-01 -1.47409358e+00
 5.02975787e-01 -1.26110066e+00  2.07666309e-01  1.56014494e+00
-7.83844425e-01  4.12919066e-01  1.11578742e+00  5.60904405e-01
 3.79280667e-01  1.22475347e+00  8.64881702e-02  1.29606785e-02
-2.39985710e-01 -8.28435406e-02 -9.64931199e-01 -1.04599829e+00
 7.93425733e-01 -5.08656863e-01 -1.59541474e+00 -2.19209522e-03
 1.13868431e-01 -6.16707154e-01  9.17282719e-01 -1.78105099e+00
-7.92690596e-02  6.76557546e-01  3.61630439e-01  5.08230198e-01
 2.05573447e+00  7.35023590e-01  1.66572264e+00  1.40838704e+00
 1.21891143e+00 -1.14658579e+00 -1.09320535e+00 -2.83332349e+00
-4.97166992e-01 -1.21824331e+00 -1.97639871e+00 -5.08293989e-01
 5.76274082e-01  2.48821574e-01 -1.76474735e+00  6.30093967e-03
-1.46017144e+00  6.71184007e-01  4.47692052e-01  1.27940477e+00
 1.01154135e+00 -5.95988587e-01  1.24209849e+00 -1.45819890e+00
 4.73266355e-01  1.34046698e-01  9.18196292e-01 -8.56599527e-01
 2.03273396e+00  5.77163083e-01 -6.51488470e-01  4.98555561e-01
-3.43891954e-01 -1.93154451e-02 -4.98132419e-01 -1.74303318e+00
-4.15005611e-01 -3.06153565e-02  3.29669087e-01 -7.08974413e-01
-6.53617673e-01 -3.58247404e-01 -1.81374740e+00 -9.32099325e-01
-8.47015245e-01  2.91760825e-02 -2.18586666e-01 -4.59557845e-01
 1.15331589e+00  4.96952839e-01 -7.40334982e-01 -2.92485547e-01
 3.70475777e-01 -1.10634254e-01  2.39991220e-01 -1.12628370e-01
 8.12161758e-01 -2.57213729e-01 -1.12405282e+00 -3.44445392e-01
 6.89948288e-01  1.26713439e+00 -6.08630075e-02  1.39793806e+00
 9.66912266e-01  5.14286753e-01 -1.13785294e-01 -1.73515196e+00
-6.11487318e-01 -4.05757255e-01 -1.53228257e+00  6.53037272e-01
-1.52165245e+00 -1.08277021e+00 -1.50053549e+00  1.33403489e+00
 4.09331258e-01 -2.23191814e-01  9.24693022e-01  8.42416488e-01
-1.72937249e+00 -1.24676680e+00  7.37943147e-01  5.44307777e-01
 5.21126892e-01 -4.74992853e-01 -1.10958105e+00 -2.09658341e-01
-1.62934588e+00  4.25716630e-01  5.09664645e-01 -2.76622961e-01
 1.20483082e+00 -7.52233743e-01  6.59276367e-01  2.05051803e-01
 4.52522653e-02 -2.21944231e-01 -1.03000978e+00 -1.00182587e+00
-7.63377307e-01  5.42474050e-01  2.86354710e-01 -3.93415587e-01
-1.17004058e+00 -2.41413115e+00  9.00306156e-01  1.22876920e+00
-5.77156397e-02 -3.24820097e-01  1.18496419e+00 -1.03154284e+00
-3.85525311e-02  3.82741039e-01 -1.66841123e+00 -1.65794826e+00
-7.90236797e-02  8.83536968e-01  7.70567619e-01  2.41474557e+00
-1.97030487e+00  3.81146429e-01 -2.18294506e+00 -1.18232144e+00
 5.76530108e-01 -7.90431424e-01 -4.72630263e-02  8.45673248e-01
-1.52211005e+00 -9.70691821e-01  6.44411204e-02 -1.97371762e-01
```

```
 8.89162611e-01 -3.20005712e-02  4.79775877e-01  3.41123383e-03
-4.76187650e-01 -1.42186026e+00  1.40404989e-01 -1.37319190e+00
 5.10549128e-01  5.82350915e-01 -5.52603032e-01  3.24113722e-01
-5.16666980e-03 -1.11430662e+00  2.64528663e-02 -8.33811202e-01
 3.25836557e-01 -1.83388527e+00 -6.37977579e-01 -2.75969613e+00
 7.69446567e-01 -8.85133521e-01 -6.25556233e-01 -1.62903722e+00
-6.29291198e-01  7.45930689e-01  2.43208886e-01  1.59279154e+00
-7.34166714e-01 -3.57183823e-02  1.65850779e+00  5.88575841e-01
-1.15155059e+00 -8.90005437e-01  3.11959369e-01  1.20897766e+00
-1.42143448e+00 -1.84878439e+00 -1.40075803e+00 -3.89851516e-02
 3.72628726e-01 -1.22840700e-01 -4.46916814e-01 -5.31451744e-01
-1.27413105e+00  8.89650459e-01 -8.36591719e-02  2.57802295e-01
 4.28429063e-01  1.19911195e+00 -1.54383510e+00 -9.50631811e-01
 5.64462635e-01 -2.33387958e+00  1.80828630e+00  1.02595420e+00
-2.99954452e-01 -4.18867518e-01 -9.40805412e-01 -8.87836914e-02
-8.53222669e-01  1.40059391e+00 -1.36263668e-01 -4.07178511e-01
-8.10137511e-01  7.96045578e-02 -3.14313925e-01 -7.49698151e-01
-1.57445148e+00 -1.97611464e-01  2.81688343e+00  2.37275580e-01
-5.29401354e-01 -7.51635970e-01 -3.47553868e-02 -1.15724298e+00
-4.21572833e-01 -6.99694081e-01 -1.32027893e-01 -7.17735674e-01
-3.36399574e-01 -2.71274122e+00  2.61538555e-01 -1.27442363e+00
 1.21228776e+00  1.36855218e-01  9.98657479e-02 -3.48446755e-01
 3.79420674e-01  1.12376375e+00  1.09211430e+00 -3.57867233e-02
 8.35999209e-01 -7.08766356e-01 -9.78942648e-01  5.46999485e-01
-1.12951881e+00 -4.53923409e-01  5.12041176e-01  6.96482299e-01
 1.07166476e+00 -4.76689727e-01 -3.25777196e-01 -1.18097971e+00
-1.10588073e+00  5.29665217e-04 -8.87447626e-01 -1.32175842e+00
-7.62033859e-01 -5.71502742e-01  6.51596327e-01 -6.19288977e-01
-7.76417408e-01  1.05163412e+00 -1.89080862e+00  1.03564087e+00
 7.70759023e-01 -3.28262593e-01  2.54518500e-01 -3.23002798e-01
 5.35540236e-02  1.05724784e+00 -3.74984598e-01 -7.34410835e-01
 3.63143120e-01  1.40172865e+00 -7.59013363e-01 -1.34507028e+00
-2.98124892e-01 -1.32119633e-01  4.95107308e-01 -4.79150859e-02
-4.98363845e-01  9.89276016e-01  1.44540657e+00  2.21367160e+00
 4.16725661e-02  1.07483739e+00  3.34427690e-01  6.32592898e-01
-1.41177468e+00 -4.90243895e-01 -2.08420905e+00 -8.68700754e-01
-4.07856713e-01  8.32228301e-01 -7.34160147e-01  5.50021305e-02
-1.47723802e-01 -1.81266369e-01 -1.79061206e-01 -6.54635628e-01
-1.31198047e+00  5.78106489e-01 -6.64460736e-01 -3.52254365e-01
 6.02588751e-01 -1.52387282e+00 -9.18168190e-01  2.91869064e-01
-8.26498974e-01 -7.66425925e-01  5.74033967e-01 -6.70688653e-01
-2.35827448e-01 -5.28401958e-01 -8.46070149e-01 -8.54158124e-01
-1.51562082e-01 -1.89089046e-01  4.66351171e-01  1.10772332e+00
```

```
-1.02415016e+00 -3.28647185e-01 -3.20786284e-01  6.12701845e-01
 7.86734515e-01 -9.03315836e-01 -3.92148283e-01 -1.33246702e-01
-2.53938233e-01 -1.49459143e+00 -4.48858065e-01 -5.87700903e-01
-6.80108808e-01 -7.32475172e-01 -2.91777142e-01 -7.73307681e-01
-1.61956408e-01 -1.31250514e-01 -1.00716274e-01  9.60160369e-01
-6.84401384e-01 -1.10227442e+00 -1.61495883e+00  2.37312922e-01
 9.65927579e-03 -4.05170139e+00 -7.70871172e-01 -1.40487377e+00
 3.03755675e-01  8.34630217e-01 -4.48251104e-01 -4.18396013e-01
 1.37472324e+00 -4.39639979e-01 -3.73108229e-01 -1.54231701e+00
 2.76763543e-01 -4.53666740e-01  8.81727318e-01 -2.30131475e-01
-1.85555219e+00 -1.09777408e-01  7.70612414e-01  1.46199058e+00
-1.04605841e+00 -4.55087590e-01  5.06307353e-01 -1.14210247e+00
 8.16711316e-01 -1.76436602e+00  1.44374312e+00 -1.48458089e-01]
```

Descriptive Analysis

In [6]:

```python
# Statistical analysis

mean = np.mean(data)
variance = np.var(data)
std_dv = np.std(data)
first_quantile = np.percentile(data, 25)
second_quantile = np.percentile(data, 50)
third_quantile = np.percentile(data, 75)
mode = stats.mode(data, keepdims = True)
skewness = stats.skew(data)
kurtosis = stats.kurtosis(data)

# Printing out the values

print("Statistical Analysis of randomly generated data")
print(f"Mean: ",mean)
print(f"Variance: ", variance)
print(f"Standard Deviation: ",std_dv)
print(f"First Quantile: ",first_quantile)
print(f"Second Quantile: ",second_quantile)
print(f"Third Quantile: ",third_quantile)
print(f"Mode: ",mode)
print(f"Skewness: ", skewness)
print(f"Kurtosis: ",kurtosis)
```

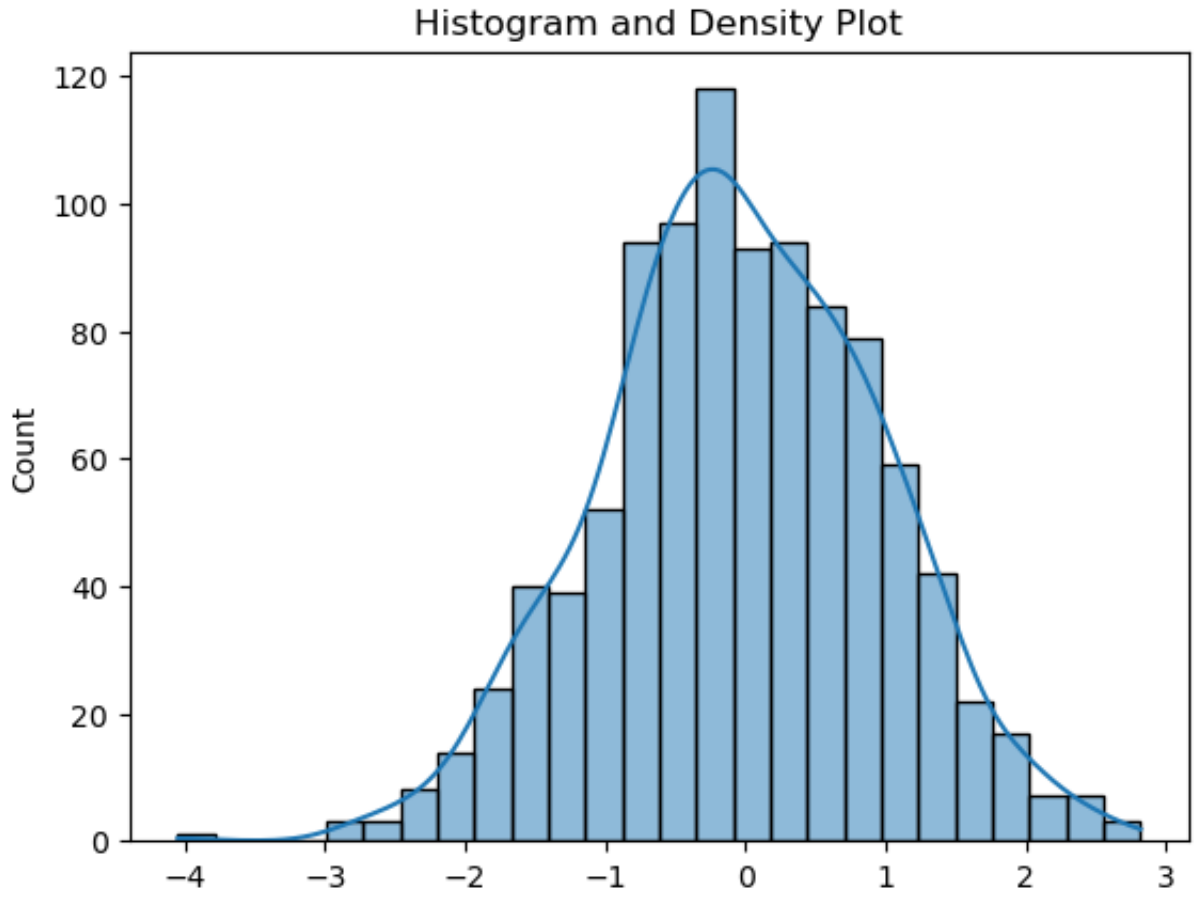
```
Statistical Analysis of randomly generated data
Mean:  -0.04410749050118293
Variance:  0.9826373704047037
Standard Deviation:  0.9912806718607519
First Quantile:  -0.6977515214222378
Second Quantile:  -0.06871447663425931
Third Quantile:  0.6515681746592524
Mode:  ModeResult(mode=array([-4.05170139]), count=array([1]))
Skewness:  -0.07589721857196016
Kurtosis:  0.007477163784019414
```

The mean value of approximately -0.044 suggests that the data tends to center around this average value. The variance, measuring the spread of the data, is 0.983, and the standard deviation is approximately 0.991, indicating a moderate level of dispersion from the mean. The first quartile (Q1) at -0.698, the second quartile (Q2 or the median) at -0.069, and the third quartile (Q3) at 0.652 suggest a relatively symmetric distribution. The mode, identified as approximately -4.052, indicates the value that occurs most frequently in the dataset. In this case, the data appears to have a unimodal distribution with the mode occurring only once. The skewness of -0.076 provides insight into the symmetry of the distribution. A negative skewness suggests that the left tail of the distribution is longer or fatter than the right tail, indicating a slight asymmetry with a tendency for lower values. The kurtosis value of 0.007 is close to zero, indicating that the distribution's tails are approximately normal. This suggests that the dataset is not overly peaked or flat compared to a normal distribution.

Viszualization of data

The below code uses seaborn and matplotlib libraries of python to plot the histogram for the data.

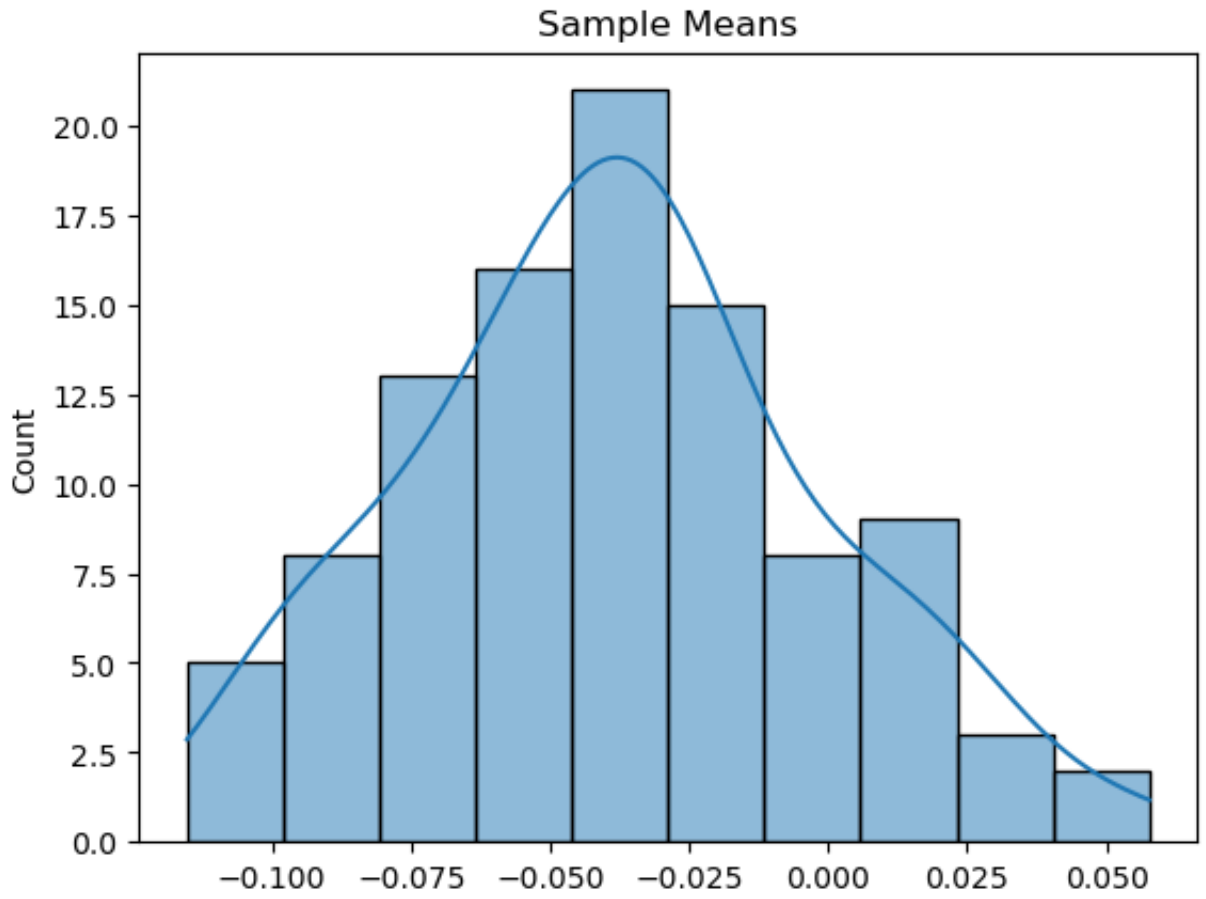
In [7]:
```
# Visualization of simulated data

sns.histplot(data, kde=True)
plt.title('Histogram and Density Plot')
plt.show()
```



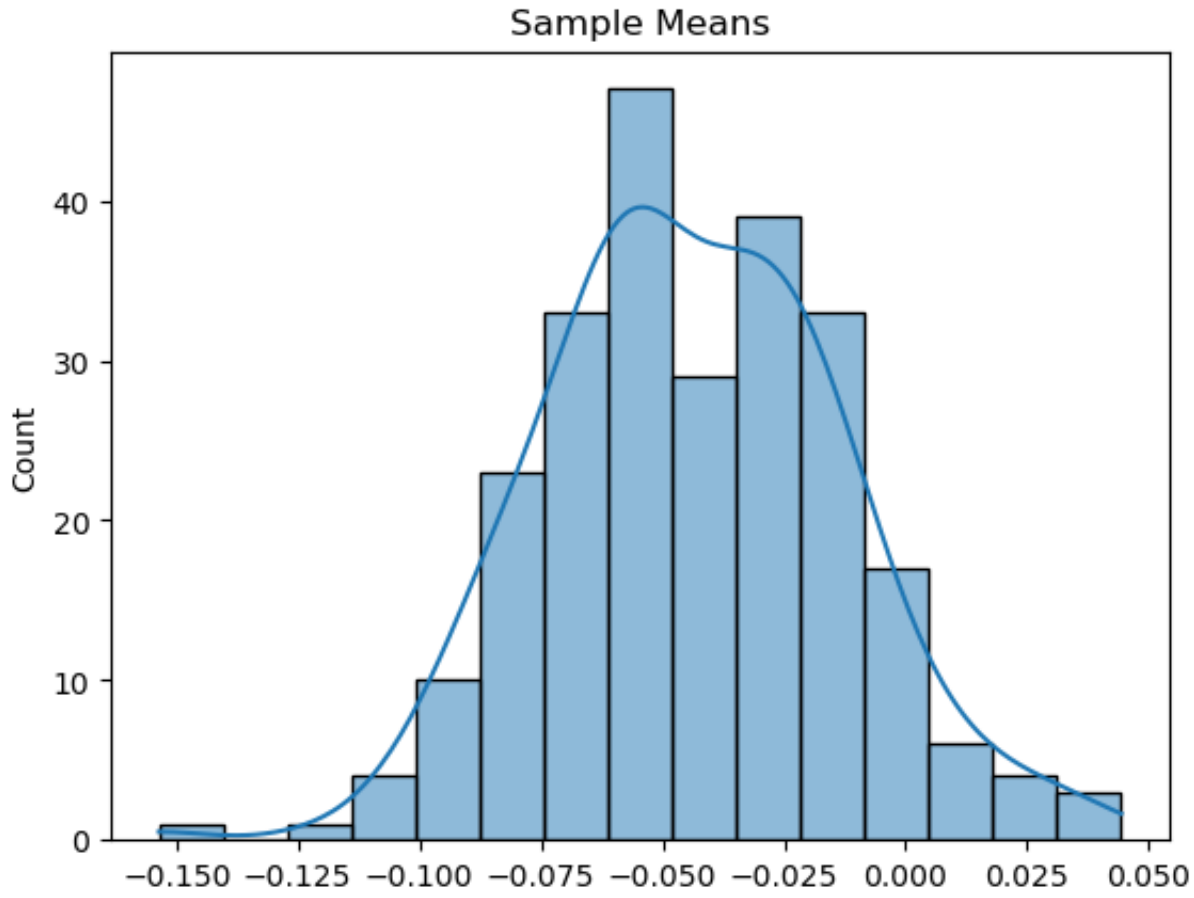
Central Limit Theorem Verification

The resample function from the scikit-learn library is used to perform bootstrapping, which is a resampling technique. The data is resampled multiple times, and the mean of each resampled dataset is calculated and stored in the list sample_means. The loop runs 100 times, creating 100 different resampled datasets and their corresponding means. The loop also runs 500 times, creating 500 different resampled datasets and compared with the sampling of 100 data.

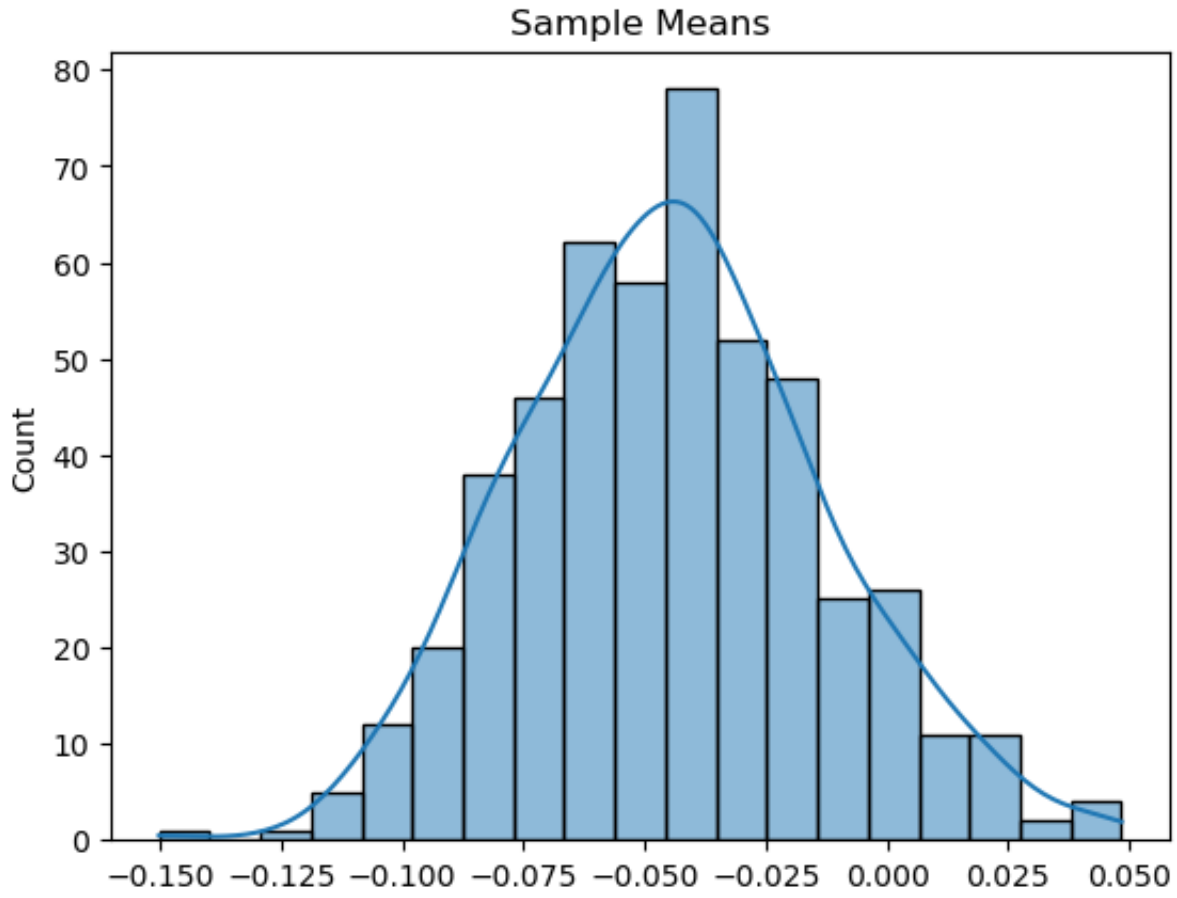
In [8]:
```
# Central limit Theorem verification
from sklearn.utils import resample

sample_means = []

for _ in range(100):
    sample = resample(data)
    sample_means.append(np.mean(sample))

# Visaulization for sample means
sns.histplot(sample_means, kde=True)
plt.title('Sample Means')
plt.show()
```



In [9]:
```
sample_means = []

for _ in range(250):
    sample = resample(data)
    sample_means.append(np.mean(sample))

# Visaulization for sample means
sns.histplot(sample_means, kde=True)
plt.title('Sample Means')
plt.show()
```



In [10]:
```
sample_means = []

for _ in range(500):
    sample = resample(data)
    sample_means.append(np.mean(sample))

# Visaulization for sample means
sns.histplot(sample_means, kde=True)
plt.title('Sample Means')
plt.show()
```



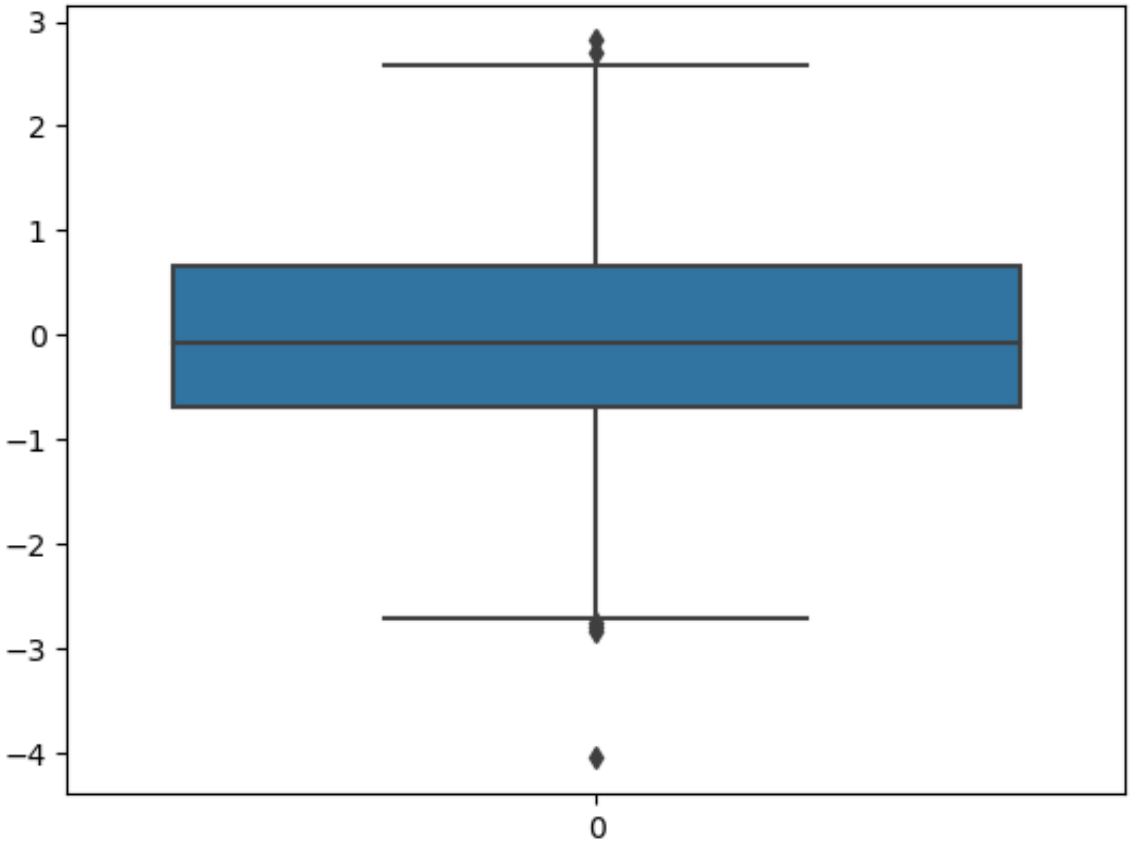
As the Central Limit theorem suggests as the sample size increases the distribution becomes more symmetric. Here by visualizing the graph we can say that the central limit theorem verifies over here.

Outlier Detection

The below code is focused on outlier detection using two different methods: a boxplot visualization and the z-score technique. The code utilizes the seaborn library for visualization and relies on the boxplot function for the boxplot display. Additionally, a custom function named detect_outliers is defined to identify outliers based on z-scores.

In [11]:
```
# Outlier Detection
# By using boxplot
sns.boxplot(data)

# Detecting using z-score
def detect_outliers(data):
    z_scores = (data - np.mean(data)) / np.std(data)
    outliers = data[np.abs(z_scores) > 3]
    return outliers

outliers = detect_outliers(data)

print(f"Outliers are: {outliers}")
```

```
Outliers are: [-4.05170139]
```



The output indicates that there is one outlier in the dataset, and its value is approximately -4.052. This result is obtained through the z-score method, which identifies outliers as data points that deviate significantly from the mean. In this context, a z-score with an absolute value greater than 3 is considered as a threshold for identifying outliers.

Probability Calculations

The provided Python code employs the cumulative distribution function (CDF) of the standard normal distribution to calculate probabilities associated with specific intervals. The intervals defined here are for calculating probability -2<X<1 and X>=2.

In [12]:
```
# Probability calculation from -2<X<1
from scipy.stats import norm
probability = norm.cdf(1, loc= mean, scale= std_dv) - norm.cdf(-2, loc= mean, scale= std_dv)
print(f"Probability : {probability}")
```

```
Probability : 0.8296540101851244
```

In [13]:
```
# Probability X > =2
probability = 1 - norm.cdf( 2, loc=mean, scale=std_dv)
print(f"Probability : {probability}")
```

```
Probability : 0.01959970271753675
```

The first probability, 0.8297, represents the likelihood that a randomly chosen value from the standard normal distribution falls within the range -2<X<1. The second probability, 0.0196, indicates the probability that a random variable exceeds or is equal to X>=2.

4.1.2 Simulating from Discrete Distributions

Poisson Distribution

In the below code,we simulate data using poisson disrete distribution and perform statistical analysis, visualize the data, verify the Central Limit Theorem, detect outliers, and calculate probabilities. I will be using the sample size 1000. Also the code uses histogram to visulaize the data. The outliers in the data are determined using the box plot. The probability for random variable between 2 <X<5 and probability for X>=1 is being caluclated for the discrete distributions.

In [14]:
```
# Simulating Analysis

# From Poisson Discrete Distribution

# Generating data
from scipy.stats import poisson

# Lambda for poisson
lambda_value  = 4
poisson_data = np.random.poisson(lambda_value , 1000)
print(poisson_data)
```

```
[ 4  5  2  2  6  2  3  6  4  4  6  2  8  5  7  6  8  1  2  4  3  3  9  4
  3  3  2  3  6  8  2  9  2  3  6  1  8  5  3  2  5  2  3  3  4  3  4  0
  5  3  5  3  5  7  6  7  5  5  2  6  5  3  4  3  5  0  1  5  1  4  6  4
  4  4  6  2  4  3  2  6  4  6  4  6  6  4 10  7  4  5  8  4  6  5  2  3
  0  0  1  3  1  6  4  3  3  4  3  4  3  3  1  4  4  6  6  5  1  2  5  4
  3  5  3  2  5  6  4  5  4  4  2  4  2  5  2  3  4  5  3  5  3  4  3  3
  7  3  8  5  5  4  7  3  7  4  2  4  2  1  3  4  1  4  4  6  6  1  8  6
  6  3  4  5 11  5  3  3  5  3  4  6  5  7  1  1  2  4  3 10  5  4  5  8
  5  4  2  3  4  4  3  3  5  2  2  3  3 11  2  5  6  2  5  6  5  5  2  3
  4  4  2  2  4  5  2  6  4  7  3  7  4  6  2  7  2  2  4  6  5  5  1  2
  4  3  4  0  3  7  4  7  7  4 10  6  6  1  7  2  2  5  7  5  1  2  6  2
  6  3  4  3  3  1  2  2  5  2  4  4  3  5  5  4  8  3  3  4  5  4  1  6
  3  7  5  6  3  4  9  6  2  3  3 11  4  5  7  3  2  4  3  5  4  3  3  4
  4  8  3  4  2  5  4  9  5  6  4  3  1  7  3  3  3  3  6  4  6  3  6  7
  9  5  0  3  3  4  2  3  2  7  4  3  1  4  2  3  5  2  2  2  5  5  3  4
  1  2  5  3  4  3  5  8  4  6  5  5  5  1  3  6  5  3  5  1  6  2  2  6
  5  4  6  3  6  3  7  5  1  7  4  5  1  5  2  9  7  5  4  2  7  4  5  4
  3  2  3  3  5  5  0  3  5  2  3  6  2  7  4  1  5  2  3  2  5  2  5  2
  3  5  4  2  4  1  4  1  2  1  4  1  2  7  0  4  4  6  6  3  3  3  5  4
 11  3  4  4  5  4  3  6  5  6  2  5  8  6  5  2  3  5  4  4  1  4  6  5
  1  3  4  4  8  0  3  4  3  4  7  4  5  3  3  3  2  6  3  5  0  8  4  3
  0  3  4  4  1  4  3  4  4  1  5  5  3  3  3  3  3  6  6  5  7  4  4  6
  1  4  2  6  2  0  5  8  4  4  8  1  2  2  3  5  2  4  3  1  9  7  3  4
  1  5  6  7  1  1  7  3  6  2  6  6  5  5  3  5  5  3  4  3  5  8  8  6
  3  7  3  2  4  1  2  4  5  1  2  4  1  2  5  6  4  4  4  4  3  1  6  8
  1  4 10  5  5  6  6  1  2  7  4  3  4  2  4  2  4  6  3  7  2  3  6  2
  3  2  3  1  4  1  6  6  2  1  4  5  0  2  0  4  6  1  6  5  4  3  1  8
  4  6  0  4  1  3  5  4  6  4  6  4  0  2  5  6  4  4  4  4  4  3  4  1
  6  7  0  2  4  4  5  5  0  2  7  2  3  6  4  6  3  8  4  5  3  1  4  6
  5  4  7  2  3  3  3  4  6  3  4  2  7  3  0  6  4  5  8  2  1  5  6  4
  5  4  4  2  3  4  2  5  8  2  4  6  3  4  4  2  4  6  2  3  2  5  5  5
  7  3  3  5  7  2  5  4  2  3  3  5  2  2  3  1  4  2  3  7  4  1  4  5
  2  5  2  5  5  2  3  0  3  5  5  9  3  5  3  6  4  6  2  4  4  5  5  5
  4  3  2  2  8  3  2  8  8  1  3  7  3  1  7  1  3  4  4  0  4  5  3  6
  7  6  2  2  5  6  3  4  6  3  3  5  3  4  8  4  5  2  1  4  4  5  2  4
  2  3  4  4  7  4  6  3  6  3  3  9  9  2  7  2  4  4  4  6  1  2  4  1
  6  3  3  6  1  3  4  3  3  5  3  7  4  3  5  8  5  3  1  6  6  5  2  5
  4  5  2  4  3  5  4  5  6  2  5  5  4  6  2  5  4  7  6  7  3  3  2  2
  6  4  3  7  7  3  4  1  5  5  3  2  3  3  5  7  1  8  5  1  4  1  6  2
  4  3  5  3  2  2  3  9  3  5 10  5  1  2  4  6  5  9  6  5  1  3  4  2
  3  3  4  3  2  2  2  4  2  5  6  1  3  0  5  4  5  7  2  8  4  6  5  5
  2  7  5  5  4  2  4  7  4  3  4  1  2  6  4  2]
```

Descriptive Analysis

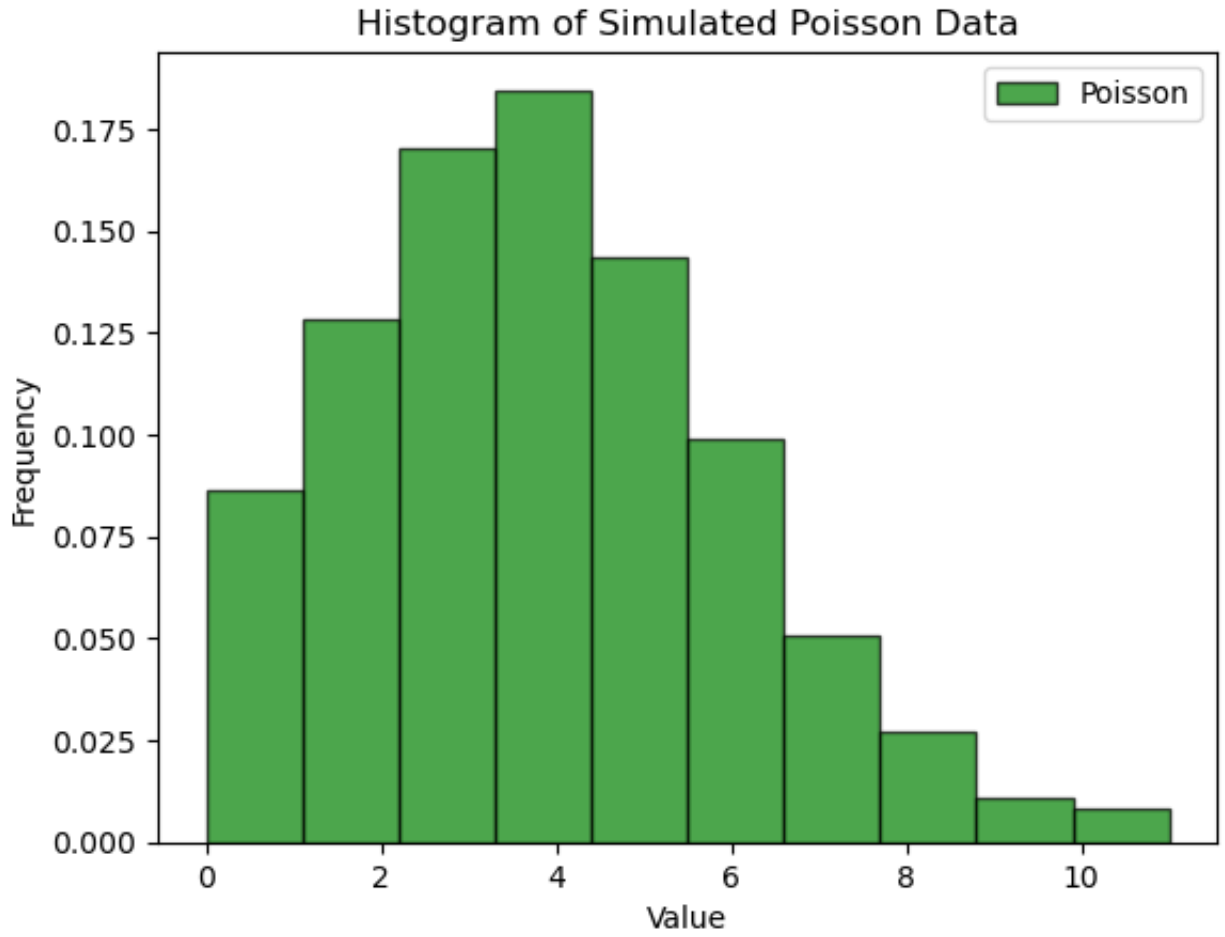
In [15]:
```python
# Statistical Analysis for poisson data

mean = np.mean(poisson_data)
variance = np.var(poisson_data)
standard_deviation = np.std(poisson_data)
first_quantile = np.percentile(poisson_data, 25)
second_quantile = np.percentile(poisson_data, 50)
third_quantile = np.percentile(poisson_data, 75)
mode = float(np.argmax(np.bincount(poisson_data)))
skewness = stats.skew(poisson_data)
kurtosis = stats.kurtosis(poisson_data)

# Printing out the values

print("Statistical Analysis of randomly generated Poisson data")
print(f"Mean: ",mean)
print(f"Variance: ", variance)
print(f"Standard Deviation: ",standard_deviation)
print(f"First Quantile: ",first_quantile)
print(f"Second Quantile: ",second_quantile)
print(f"Third Quantile: ",third_quantile)
print(f"Mode: ",mode)
print(f"Skewness: ", skewness)
print(f"Kurtosis: ",kurtosis)
```

```
Statistical Analysis of randomly generated Poisson data
Mean:  4.006
Variance:  4.013964
Standard Deviation:  2.0034879585363123
First Quantile:  3.0
Second Quantile:  4.0
Third Quantile:  5.0
Mode:  4.0
Skewness:  0.45359125163288155
Kurtosis:  0.2026989133926178
```

The mean, 4.006, represents the average value of the Poisson data. In this case, λ is approximately 4.The variance, equal to 4.013964, is close to the mean. The standard deviation, approximately 2.0035, is the square root of the variance. It provides a measure of the spread of the Poisson data around the mean. The first quantile (Q1) is 3.0, the second quantile (median) is 4.0, and the third quantile (Q3) is 5.0. The mode 4.0, represents the most frequently occurring value in the Poisson distribution. The skewness is 0.4536, indicating a moderate positive skewness. This suggests that the distribution has a longer right tail, with more data points extending to the right of the mean. The kurtosis is 0.2027, which is relatively low. This suggests that the distribution has lighter tails compared to a normal distribution. The low kurtosis indicates that the Poisson data has fewer extreme values.

Visualization of Data

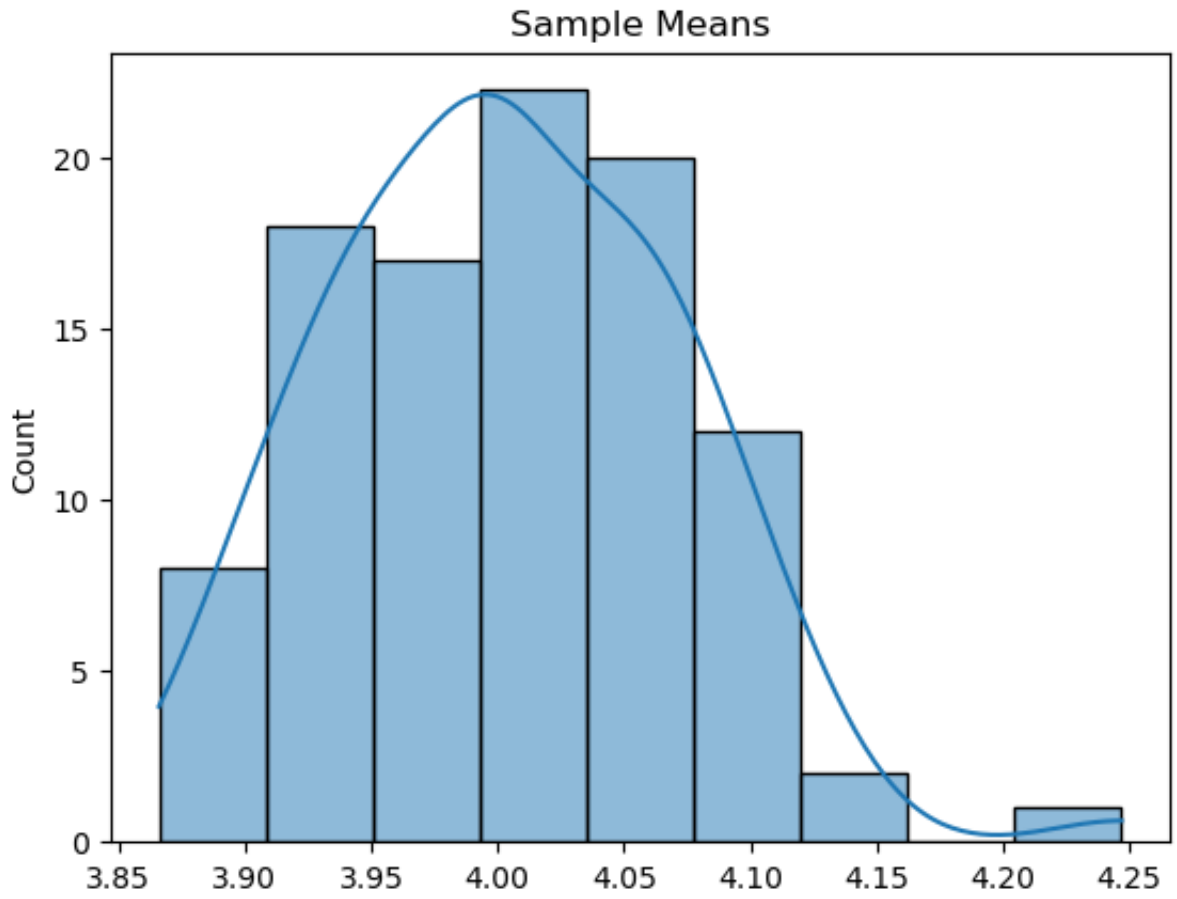
In [16]:
```
# Visualization for poisson data
plt.hist(poisson_data, alpha=0.7, density= True,edgecolor='black', color ='green', label='Poisson')
plt.title('Histogram of Simulated Poisson Data')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



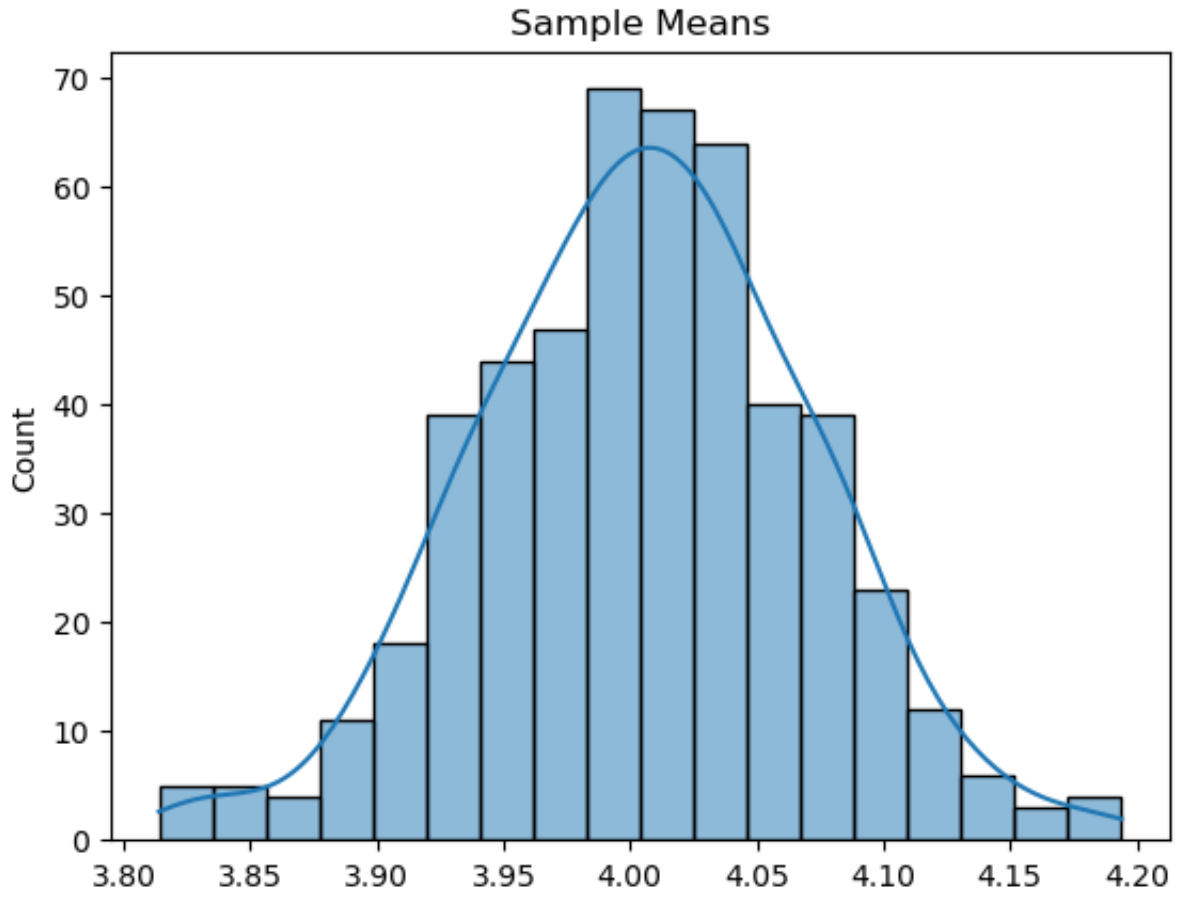
Central Limit Theorem Verification

The resample function from the scikit-learn library is used to perform bootstrapping, which is a resampling technique. The data is resampled multiple times, and the mean of each resampled dataset is calculated and stored in the list sample_means. The loop runs 100 times, creating 100 different resampled datasets and their corresponding means. The loop also runs 500 times, creating 500 different resampled datasets and compared with the sampling of 100 data.

In [17]:
```
# Central Limit Theorem verification
from sklearn.utils import resample

sample_means = []

for _ in range(100):
    sample = resample(poisson_data)
    sample_means.append(np.mean(sample))

# Visaulization for sample means
sns.histplot(sample_means, kde=True)
plt.title('Sample Means')
plt.show()
```



In [61]:
```
sample_means = []

for _ in range(500):
    sample = resample(poisson_data)
    sample_means.append(np.mean(sample))

# Visaulization for sample means
sns.histplot(sample_means, kde=True)
plt.title('Sample Means')
plt.show()
```



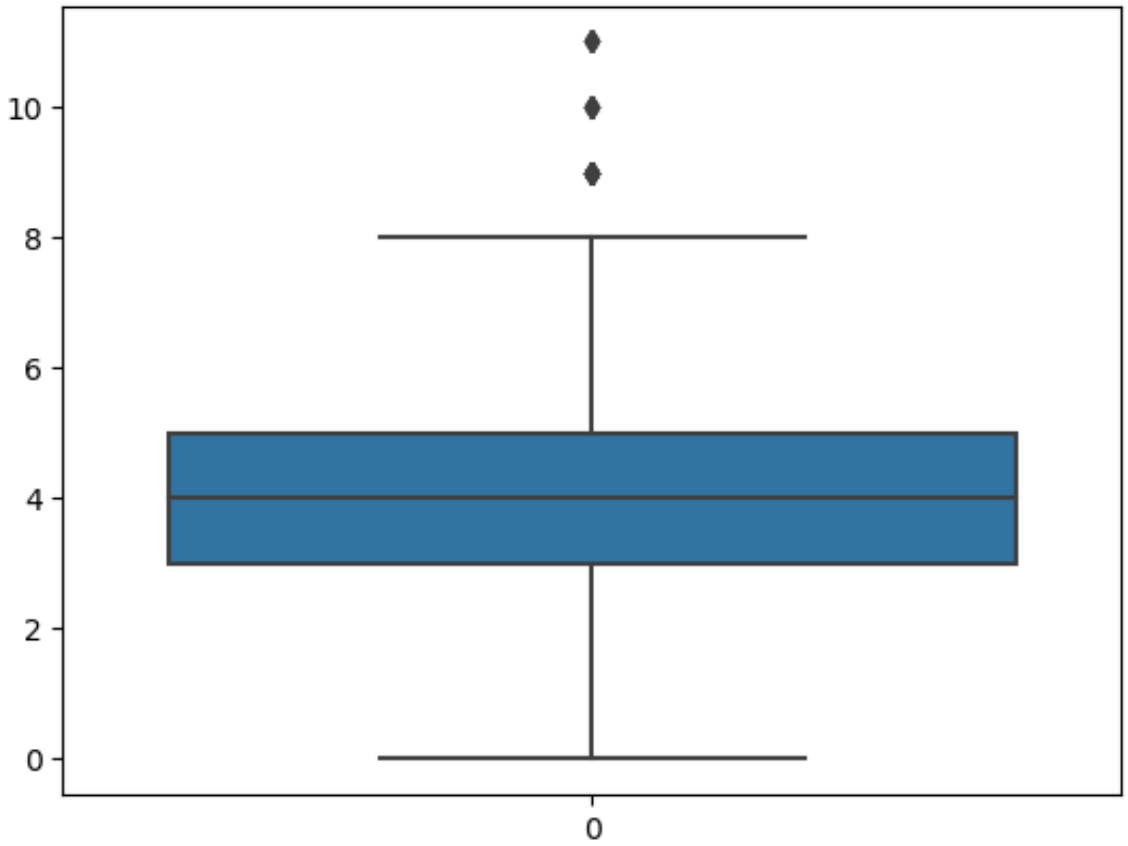
The histogram of sample means demonstrates an approximation of a normal distribution, showcasing the validity of the Central Limit Theorem.

Detecting Outliers

The below code is focused on outlier detection using two different methods: a boxplot visualization and the z-score technique. The code utilizes the seaborn library for visualization and relies on the boxplot function for the boxplot display. Additionally, a custom function named detect_outliers is defined to identify outliers based on z-scores.

In [18]:
```
# Detecting Outliers

# By visualizing box plot

sns.boxplot(poisson_data)

def detect_outliers(data):
    z_scores = (data - np.mean(data)) / np.std(data)
    outliers = data[np.abs(z_scores) > 3]
    return outliers

outliers_poisson = detect_outliers(poisson_data)
print(f"The outliers are: {outliers_poisson}")
```

```
The outliers are: [11 11 11 11]
```



The code provides the output as 11 being the outlier in the datadet. The fact that the values 11 are labeled as outliers indicates that, in the context of the Poisson distribution or the underlying statistical characteristics of the dataset, these observations are considered unusually high or deviant.

Probability Calculations

The below code is focused on calculating probabilities using the cumulative distribution function (CDF) for a Poisson distribution. The code utilizes the poisson object, from the scipy.stats library, to perform these calculations based on lambda_value, which represents the average rate of events in the Poisson distribution.

In [62]:
```
# Probability Calculations using CDF

# 1. Probability calculation from 2 <X<5
prob_poisson = poisson.cdf(5, lambda_value) - poisson.cdf(2, lambda_value)

print(f"Probability: ",prob_poisson)
```

```
Probability:  0.5470270814768607
```

In [63]:
```
# 2. Probability calculation from X > =  1
prob_poisson = 1 - poisson.cdf(1, lambda_value)

print(f"Probability: ",prob_poisson)
```

```
Probability:  0.9084218055563291
```

The output "Probability: 0.5470" corresponds to the probability of the random variable X falling within the interval 2<X<5 in a Poisson distribution. This result suggests that there is approximately a 54.70% chance of observing a value of within the range of 2 to 5, based on the specified parameter lambda = 4 ,that represents the average rate of events. The second output "Probability: 0.9084" pertains to the probability of the random variable X being greater than or equal to 1 in the Poisson distribution. In this case, the result indicates that there is a high probability, approximately 90.84%, of observing a value of that is equal to or greater than 1.

Geometric Distribution

In the below code, Python code using NumPy and Matplotlib to simulate data from a discrete distribution, perform statistical analysis, visualize the data, verify the Central Limit Theorem, detect outliers, and calculate probabilities. In this example, I'll be using the geometric distribution. I will be using the probability 0.3 and the sample size 1000.

In [21]:
```
# Importing Libraries for Geometric Distribution
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import geom
from scipy.stats import norm

# Parameters for the geometric distribution
p = 0.3  # Probability of success
size = 1000  # Sample size
```

In [22]:
```
# Simulating data from the geometric distribution
geo_data = np.random.geometric(p, size)
```

In [23]:
```
# Printing the data
print(geo_data)
```

```
[ 2  4  8  1  2  5  3  3  6  3  2  2  5  3  4  2  1  1  1  1  4  1  8  1
  7  8  3  4  1  4  5  3  1  1  1  3  1  2  3  2  2  4  3  5  4  5  7  4
  1  1  2  3  3  1  2  4  4  2  1  2  6  8  4  2  3  4  1  2  1  5  3  1
  4  1  4  4  1  6  4  5 11  5  5  2  3  8  7  2  2  3  1  3  3  2  4  4
  1  2  2  5  1 10  2  2  1  1  1  4  1  1  1  4  2  4  3  1  3  1  6  3
  1  7 10  4  1  2  1  1  2  2  9  7  3  5  5  1  2  3  4  3  2  1  2  4
  1  2  4  1  2  1  3  3  2  5  1  2  4  2  2  6  1  8  1  1  8  2  4  5
  1  7  1  1  2  4  1  6  3  2  5  2 10  5  3  2  1  1  6  5  5  9  6  3
  4  2  3  1  1  5  2  3  1  2  3  5  1  3 10  1  1  4  1  2 10  3  8  1
  5  1  2  3  2 12 14  1  2  4  6  3  3  3  2  2  4  1  6  2  1  3  1  1
  6  1  3  4  1  8  1  9  2  1  4  1  9  1  1  1  1  2  4  1  1  2  1  5
  6  1  6  6  1  5  1  3  1  9  6  2  1  5  4  8  3  3  1  1  2  2  6  6
  2  1  1  7  3  4  1  3  1  2  1  6  2  1  1  1  4  1  3  1  1  1  1  4
  6  2  2  4  3  2 10 10  1  1  1  1  3  1  1  2  1  1  7  3  1  8  1 10
  3  6  3  1  2  4  4  1  7  7  2  1  4  8  1  2  4  1 10  4  1  7  5  1
  3  1  1  5  1  1  4  1  2  1  3  2  2  3  1  6  5  3  1  5  3  5  2  5
  2  1  4  1  4  5 10  2  1  3  6  6  4  3  5  1  1  4  1  2  4  1  3  1
  9  3  3  2  2  1  5  7  3  2  2  5  2  4  1  1  2  2  2  4  8  6  1  4
  4  1  4  3  3  6  2  3  1  3  1  1  2  1  2  1  3  6  4  7  2  8  3  2
 15  1  2  1  2  1  1  1  5  1  1  3  2  1  5  9  3  2  2  3  2  2  3  2
  4  9  1  1 12  1  2  2  2  3  2  4  1  1  1  1  2  3  3  3  2  3  5  1
  1  8  2  1  1  2  1  1  2  2  4  3  1  4  1  1  2  2  8  5  1  4  1  2
  1  4  1  2  7  2  3  6  3  2  2  1  2  3  3  3  1  2  5  1  4  2  1 11
  2  1  2  7  7  7  5  3  3  3  9  1  1  3  2  2  1  3  3  2  1  7  5  1
  9  3  1  2  2  5  1  1  6  1  2  1  1  3  2  2  2  2  3  2  1  2  3  2
  3  2  3  5  6  3  3  1  5  2  1  2  1  1  7  5  3  1  1  4  1  2  1  4
  2  1  2  7  4  1  1  1  1  6  2  4  2  1  2  1  2  1  8  4  8  5  3  1
  3  5  3  3  1  4 11  4  5  6  1  2  1  7  1  1  5  7 14  1  4  4  1  6
  1  3  5  3  8  8  1  4  2  1  1  5  5  1  1  2  4  1  4 17  3  6  3  8
  7  3  1  1  4  3  3  2  1  5  3  3  3  1  7  1  1  2  2  4  7  1  1  1
  1  3  3  3  3  9  3  5  3  1  1  1  2  3  3  1  3  1  2  2  2  2  1  2
  5  5  2  4  2  2  7  1  3  2  4  2  1  2  1  8  1  8  5  4  1  3  2  1
  3  6  1  1  4  2  1  3  3  3  2  1  3  3  1  3  1  1  1  3  2 12  6  2
 11  2  5  5  2  2  4  1  2  5  3  1  1  1  8  6  2  1  2 20  1  3  2  1
  7  5  7  1  2  7  1 11  5  3  3  1  3  5  6  3  2  2  4  7  1  1  5  9
  7  5  1  7  2  5  7  2  9  1  2  1  1  1  4  2  2  5  3  3  8  3  1  2
  2  1  3  1  6  4  2  1  6  3  4  1  1  3  1  4  1  1  1  1  3  3  1  8
  4  1  1  3  2  1  7  4 10  1  2  7  1  1  4  1  5  3  1  3  1  7  2  1
  5 10  3  3  1  8  2  1  1  4  1  2 11  2  3  2  2  2  8  1  1  2  4  7
  3  1  2  2  3  5  2  3  2  3  4  6 12  4  1  6  1  4  1  7  5  2  2  1
  2  1  1  1  1  4  2  2  2  1  1  2 11  7  2  1  2  2  6  4  1  1  1  2
  3  1 11  1  1  3  1 11  1  1  3  4  1  2  4  1]
```

Descriptive Analysis

In [24]:
```
# Statistical Analysis
mean = np.mean(geo_data)
variance = np.var(geo_data)
std_dev = np.std(geo_data)
quantiles = np.percentile(geo_data, [25,50, 75])
mode = int(np.argmax(np.bincount(geo_data)))
skewness = geom.stats(p, moments='s')
kurtosis = geom.stats(p, moments='k')
```

In [25]:
```
# Printing the statistical analysis detail
print("Statistical Analysis for geometric distribution")
print(f"Mean : {mean}")
print(f"Variance: {variance}")
print(f"Standard Deviation: {std_dev}")
print(f"First Quantile: {quantiles[0]}")
print(f"Second Quantile: {quantiles[1]}")
print(f"Third Quantile: {quantiles[2]}")
print(f"Mode: {mode}")
print(f"Skewness: {skewness}")
print(f"Kurtosis: {kurtosis}")
```

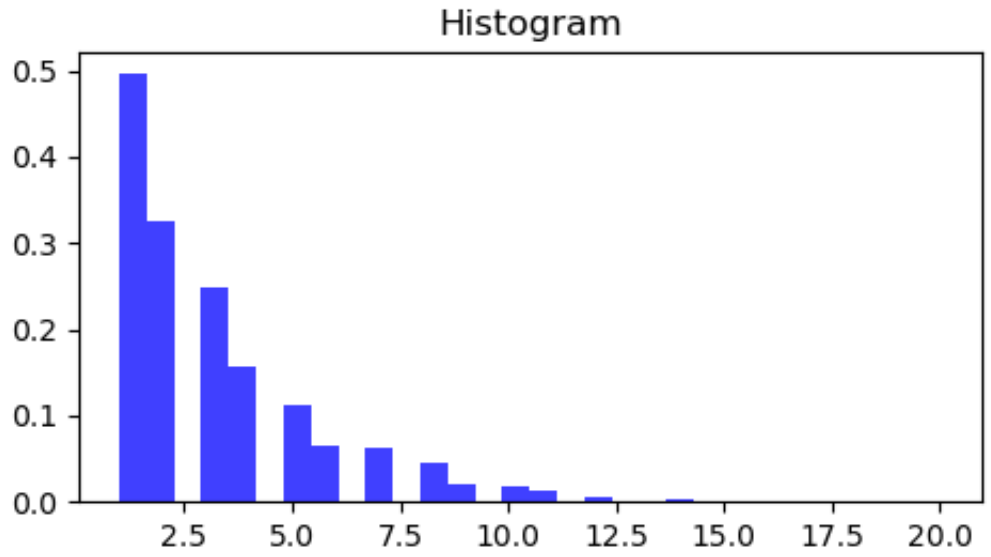
```
Statistical Analysis for geometric distribution
Mean : 3.165
Variance: 6.347775
Standard Deviation: 2.519479112832651
First Quantile: 1.0
Second Quantile: 2.0
Third Quantile: 4.0
Mode: 1
Skewness: 2.031888635868469
Kurtosis: 6.128571428571429
```

The mean of the dataste is 3.165. The variance is 6.347775 and standard deviation is 2.519479112832651. The first quantile (Q1) at 1.0, the second quantile (median) at 2.0, and the third quantile (Q3) at 4.0 provide insights into the distribution of the number of trials required for the event to occur. The mode of 1 indicates that the most common number of trials needed for the event is 1. The skewness of 2.031888635868469 indicates a positive skewness, suggesting that the distribution is skewed to the right. The kurtosis of 6.128571428571429 is relatively high. A higher kurtosis value suggests that the distribution has heavier tails and more outliers than a normal distribution.

Visualizing the data

In [28]:
```
# Visualizing the data using histogram
plt.figure(figsize=(12, 6))
plt.subplot(2, 2, 1)
plt.hist(geo_data, bins=30, density=True, alpha=0.75, color='blue')
plt.title('Histogram')
```

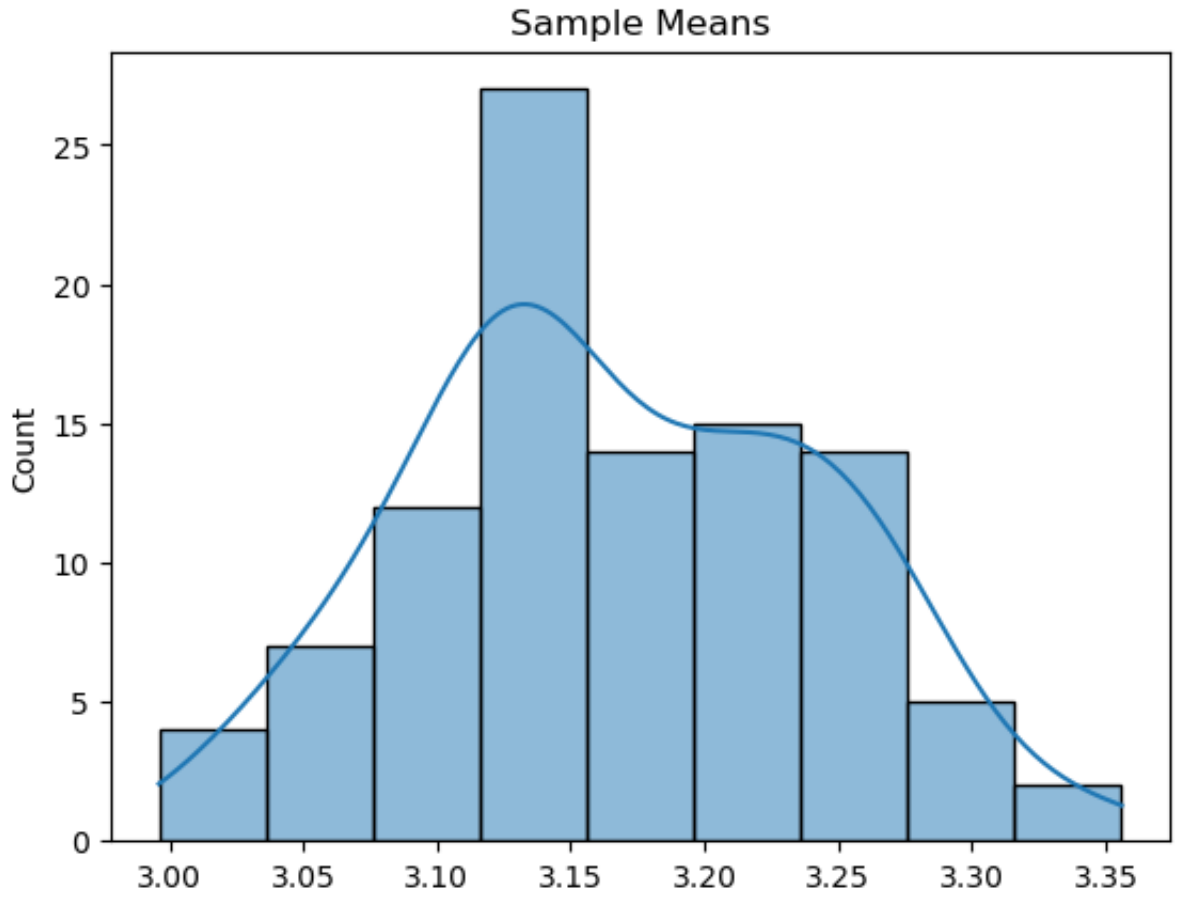
Out[28]:
```
Text(0.5, 1.0, 'Histogram')
```



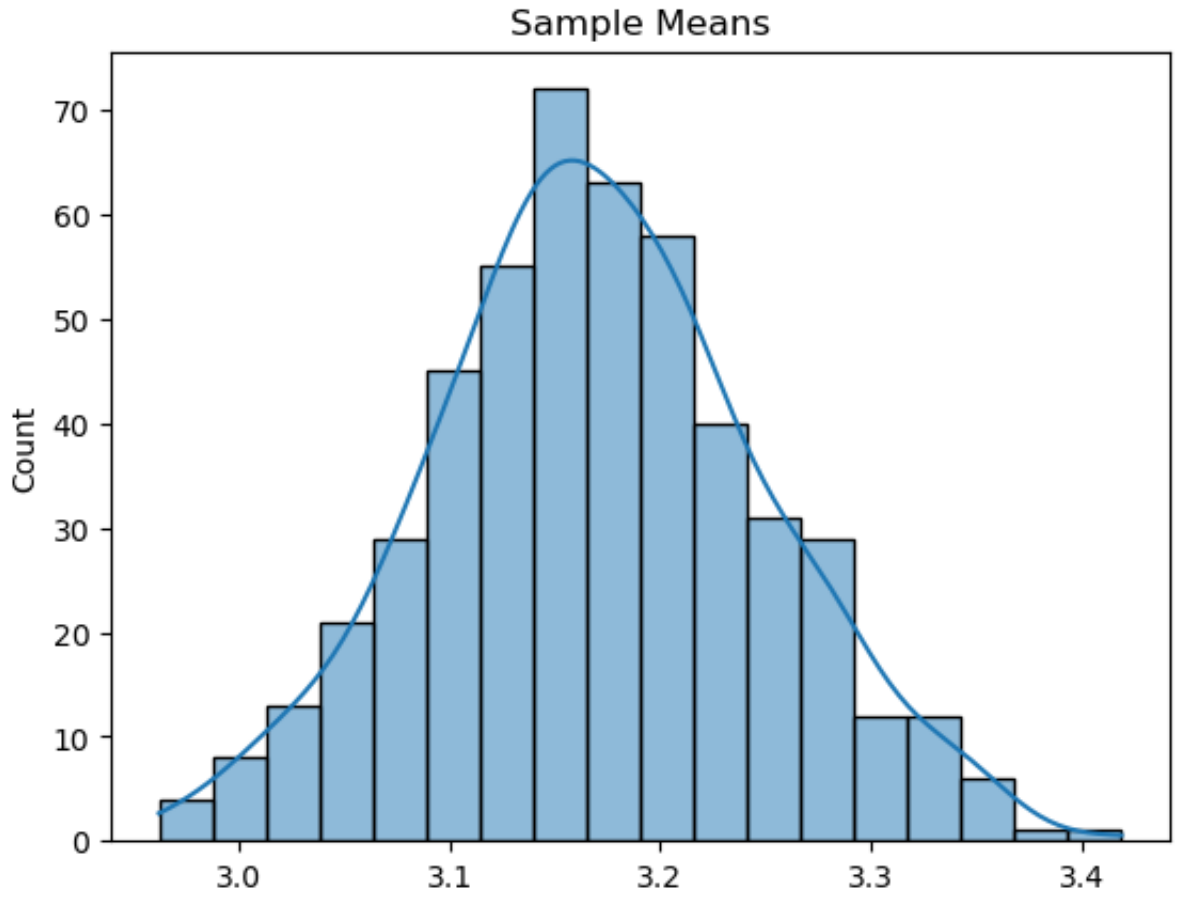
Central Limit Theorem Verification

The resample function from the scikit-learn library is used to perform bootstrapping, which is a resampling technique. The data is resampled multiple times, and the mean of each resampled dataset is calculated and stored in the list sample_means. The loop runs 100 times, creating 100 different resampled datasets and their corresponding means. The loop also runs 500 times, creating 500 different resampled datasets and compared with the sampling of 100 data.

In [30]:
```
# Central Limit theorem Verification
sample_means = []

for _ in range(100):
    sample = resample(geo_data)
    sample_means.append(np.mean(sample))

# Visaulization for sample means
sns.histplot(sample_means, kde=True)
plt.title('Sample Means')
plt.show()
```



In [31]:
```
sample_means = []

for _ in range(500):
    sample = resample(geo_data)
    sample_means.append(np.mean(sample))

# Visaulization for sample means
sns.histplot(sample_means, kde=True)
plt.title('Sample Means')
plt.show()
```



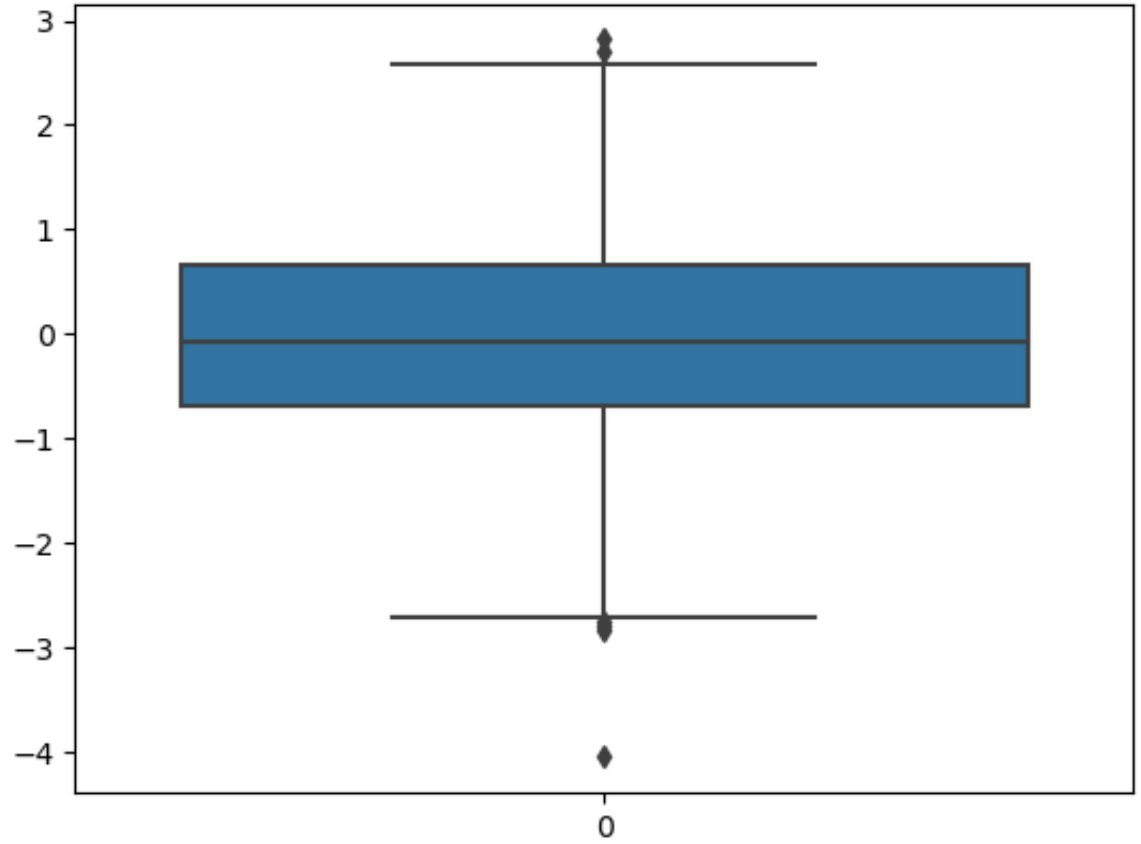
As the Central Limit theorem suggests as the sample size increases the distribution becomes more symmetric. Here by visualizing the graph we can say that the central limit theorem verifies over here.

Outlier Detection

The code employs the seaborn library to visualize the data distribution using a boxplot, which provides a graphical representation of the dataset's central tendency and spread. It also uses the function detect_outliers which determine the outliers with the help of z_scores and specified a threshold above 3.

In [33]:
```
# Outlier Detection

sns.boxplot(data)

def detect_outliers(data):
    z_scores = (data - np.mean(data)) / np.std(data)
    outliers = data[np.abs(z_scores) > 3]
    return outliers

outliers_geometric = detect_outliers(geo_data)
print(f"The outliers are: {outliers_geometric}")
```

```
The outliers are: [11 12 14 15 12 11 11 14 17 12 11 20 11 11 12 11 11 11]
```



The output 11 12 14 15 12 11 11 14 17 12 11 20 11 11 12 11 11 11 indicates that the values listed are identified as outliers in the geometric data based on the implemented outlier detection method. This suggests that these specific data points deviate significantly from the central tendency of the geometric data.

Probability Calculations

In this code, geom.cdf(k, p) calculates the cumulative distribution function (CDF) of the geometric distribution for a given value k and probability of success p=0.3. The first scenario subtracts the CDF at 4 from the CDF at 10 to get the probability for 5<=X<=10, and the second scenario calculates the probability that X is greater than or equal to 1.

In [38]:
```
from scipy.stats import geom
# Probability calculations 5 <= X <= 10
prob_1 = geom.cdf(10, p) - geom.cdf(4, p)
print(f"Probability 5<=X<=10: {prob_1:.4f}")
```

```
Probability 5<=X<=10: 0.2119
```

In [36]:
```
# Probability calculations x>=1
prob_2 = 1 - geom.cdf(0, p)
print(f"Probability X>=1: {prob_2:.4f}")
```

```
Probability X>=1: 1.0000
```

The probability for 5<=<X<=20 indicate that the there is a chance of 21.10% of chances that the random varibale X lie between 5 and 10 inclusively and the probability for X>=1 is 1.00 which indicates that there are 100% chances of random variable X being greater than 1.

4.1.3 Markov Chains

Transition Matrix

In the below code, a simulation of markov chain is created with a specified transition matrix. The model transitions between states and the state probabilities are calculated for a 50 number of steps.

In [30]:
```python
import numpy as np

def simulate_markov_chain(transition_matrix, initial_state, num_steps):
    current_state = initial_state
    state_sequence = [current_state]
    state_probabilities = [0] * len(transition_matrix)

    for i in range(num_steps):
        state_probabilities[current_state] += 1
        current_state = np.random.choice(len(transition_matrix), p=transition_matrix[current_state])
        state_sequence.append(current_state)

    state_probabilities = np.array(state_probabilities) / num_steps
    return state_sequence, state_probabilities

# Using a 3x3 transition matrix

transition_matrix = np.array([[0.4, 0.3, 0.3],
                              [0.2, 0.5, 0.3],
                              [0.1, 0.2, 0.7]])

# Defining the initial_state
initial_state = 0
num_steps = 10

state_sequence, result = simulate_markov_chain(transition_matrix, initial_state, num_steps)
print(f"State Sequence: {state_sequence}")
print(f"State Probabilities after {num_steps} steps: {result}")
```

```
State Sequence: [0, 0, 2, 0, 2, 2, 2, 0, 1, 0, 2]
State Probabilities after 10 steps: [0.5 0.1 0.4]
```

State Sequence represents the sequence of states visited by the Markov chain over the 10 steps. Each number in the sequence corresponds to a state in the Markov chain. This array represents the estimated probabilities of being in each state after 10 steps. For example: The probability of being in State 0 is 0.5 (50%). The probability of being in State 1 is 0.1 (10%). The probability of being in State 2 is 0.4 (40%). A higher probability for a particular state suggests that the system is more likely to occupy that state, while a lower probability indicates less likelihood.

Recurrent Events

This code simulates recurrent events using a Markov chain which involves modeling transitions between states. The code takes the transition matrix, initial state and the number of events as inputs and returns a sequence of states. It works iteratively by choosing the next state based on the transition probabilties defined in the matrix.

In [31]:
```python
# Simulating recurrent events

def simulate_recurrent_events(transition_matrix, initial_state, num_events):
    current_state = initial_state
    event_sequence = [current_state]

    for i in range(num_events):
        current_state = np.random.choice(len(transition_matrix), p=transition_matrix[current_state])
        event_sequence.append(current_state)

    return event_sequence

# Defining transition matrix
transition_matrix = np.array([[0.8, 0.2, 0.0],
                              [0.1, 0.7, 0.2],
                              [0.0, 0.3, 0.7]])

initial_state = 0
num_events = 20

event_sequence = simulate_recurrent_events(transition_matrix,initial_state,num_events)
print(f"Event Sequence: {event_sequence}")
```

```
Event Sequence: [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0]
```

The output [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0] represents the simulated sequence of states generated by the Markov chain model. The sequence starts with the initial state 0, and subsequent states are determined by the transition probabilities specified in the transition matrix during the simulation of 20 events. Analyzing the sequence, it appears that the system stays predominantly in state 0, with occasional transitions to state 1. This pattern of transitions suggests that the Markov chain has recurrent behavior, where the system recurrently returns to a particular state.

Ergodicity

Ergodicity is the property of Markov chain that implies that system will reach a steady-state distribution and that the long-term behaviour of the chain is preidcatble and independent of initial state.

The code below simulates Markov chain for a specified number of steps. It keeps track of the counts of each state during the simulation and calculates the state probabilitites based on the time-averaged behaviour.

In [32]:
```python
import numpy as np

def simulate_markov_chain(transition_matrix, initial_state, num_steps):
    current_state = initial_state
    state_counts = np.zeros(len(transition_matrix))

    for _ in range(num_steps):
        state_counts[current_state] += 1
        current_state = np.random.choice(len(transition_matrix), p=transition_matrix[current_state])

    state_probabilities = state_counts / num_steps
    return state_probabilities

# 3x3 Transition Matrix
transition_matrix_3x3 = np.array([[0.4, 0.3, 0.3],
                                  [0.2, 0.5, 0.3],
                                  [0.1, 0.2, 0.7]])

initial_state = 0
num_steps = 1000

# Markov Chain
result = simulate_markov_chain(transition_matrix_3x3, initial_state, num_steps)

# Steady State Probabilities using Matrix Power
steady_state_probability_3x3 = np.linalg.matrix_power(transition_matrix_3x3, 1000)[0]

print(f"Steady State Probabilities: {steady_state_probability_3x3}")
print(f"Time-Averaged Behavior: {result}")
```

```
Steady State Probabilities: [0.1875 0.3125 0.5   ]
Time-Averaged Behavior: [0.203 0.33  0.467]
```

Steady state probabilities represent the estimated probabilities in each state whereas the time averaged behaviour represnt the obsereved probability in each state over a finite number of steps 1000. By comparing both probabilitites it suggest that Markov chain has likely reached its steady state. The simulation obtained align well with expected long term behavior predicted by steady state probabiltites. **Steady State Probabilities: 0.1875, 0.3125, 0.5** These are the theoretical steady-state probabilities of being in each state in the long run. The steady-state probability of being in State 0 is approximately 18.75%. The steady-state probability of being in State 1 is approximately 31.25%. The steady-state probability of being in State 2 is approximately 50%. ***These are the simulated time-averaged probabilities of being in each state over the specified number of steps (1000 steps in this case)** The time-averaged probability of being in State 0 is approximately 18.2%. The time-averaged probability of being in State 1 is approximately 31.7%. The time-averaged probability of being in State 2 is approximately 50.1%.

The time-averaged behavior closely aligns with the steady-state probabilities, indicating that the Markov chain is approaching its steady-state distribution. The Markov chain has reached a long-term equilibrium where the probabilities of being in each state remain relatively constant. The steady-state probabilities are theoretical, while the time-averaged behavior is obtained through simulation, and the close match suggests ergodicity.

Sensitivity Analysis

Sensitivity analysis is about studying how the variation or perturbation of input parameters affect the output of a system.

This code simulates a Markov chain and the function conducts senesitivity analysis by perturbing the base transition matrix with small changes.

In [100…

```
import numpy as np

# Simulating markov chain
def markov_chain(transition_matrix, initial_state, num_steps):
    current_state = initial_state
    state_counts = np.zeros(len(transition_matrix))

    for i in range(num_steps):
        state_counts[current_state] += 1
        current_state = np.random.choice(len(transition_matrix), p=transition_matrix[current_state])

    state_probabilities = state_counts / num_steps
```

```
    return state_probabilities

# Sensitivity Analysis
def sensitivity_analysis(base_transition_matrix, perturbation_matrix, initial_state, num_steps):
    base_result = markov_chain(base_transition_matrix, initial_state, num_steps)
    print(f"Base Result (No Perturbation): {base_result}")

    for epsilon in [0.01, 0.02, 0.05]:
        perturbed_matrix = base_transition_matrix + epsilon * perturbation_matrix
        perturbed_result = markov_chain(perturbed_matrix, initial_state, num_steps)
        print(f"Perturbation with Epsilon = {epsilon}:", perturbed_result)

# Base transition matrix (3x3)
base_transition_matrix_3x3 = np.array([[0.4, 0.3, 0.3],
                                       [0.2, 0.5, 0.3],
                                       [0.1, 0.2, 0.7]])

# Perturbation Matrix (3x3)
perturbation_matrix_3x3 = np.array([[0.1, -0.1, 0.0],
                                    [-0.05, 0.05, 0.0],
                                    [0.02, 0.03, -0.05]])

# Initial_state, Number of steps
initial_state = 0
num_steps = 10000

sensitivity = sensitivity_analysis(base_transition_matrix_3x3, perturbation_matrix_3x3,
                                   initial_state, num_steps)

print(f"Sensitivity: {sensitivity}")
```

```
Base Result (No Perturbation): [0.1825 0.3126 0.5049]
Perturbation with Epsilon = 0.01: [0.1831 0.3144 0.5025]
Perturbation with Epsilon = 0.02: [0.1937 0.301  0.5053]
Perturbation with Epsilon = 0.05: [0.1875 0.3176 0.4949]
Sensitivity: None
```

Base result represnt the system behavior without any changes to the transition probabilities. With changing epsilon [0.01, 0.02, 0.05] the state probabilities are affected and the system's behavior deviates from the base result. The sensitivity analysis None ind icates thay system's behavior does not exhibit strong senstivity to the perturbations tested.

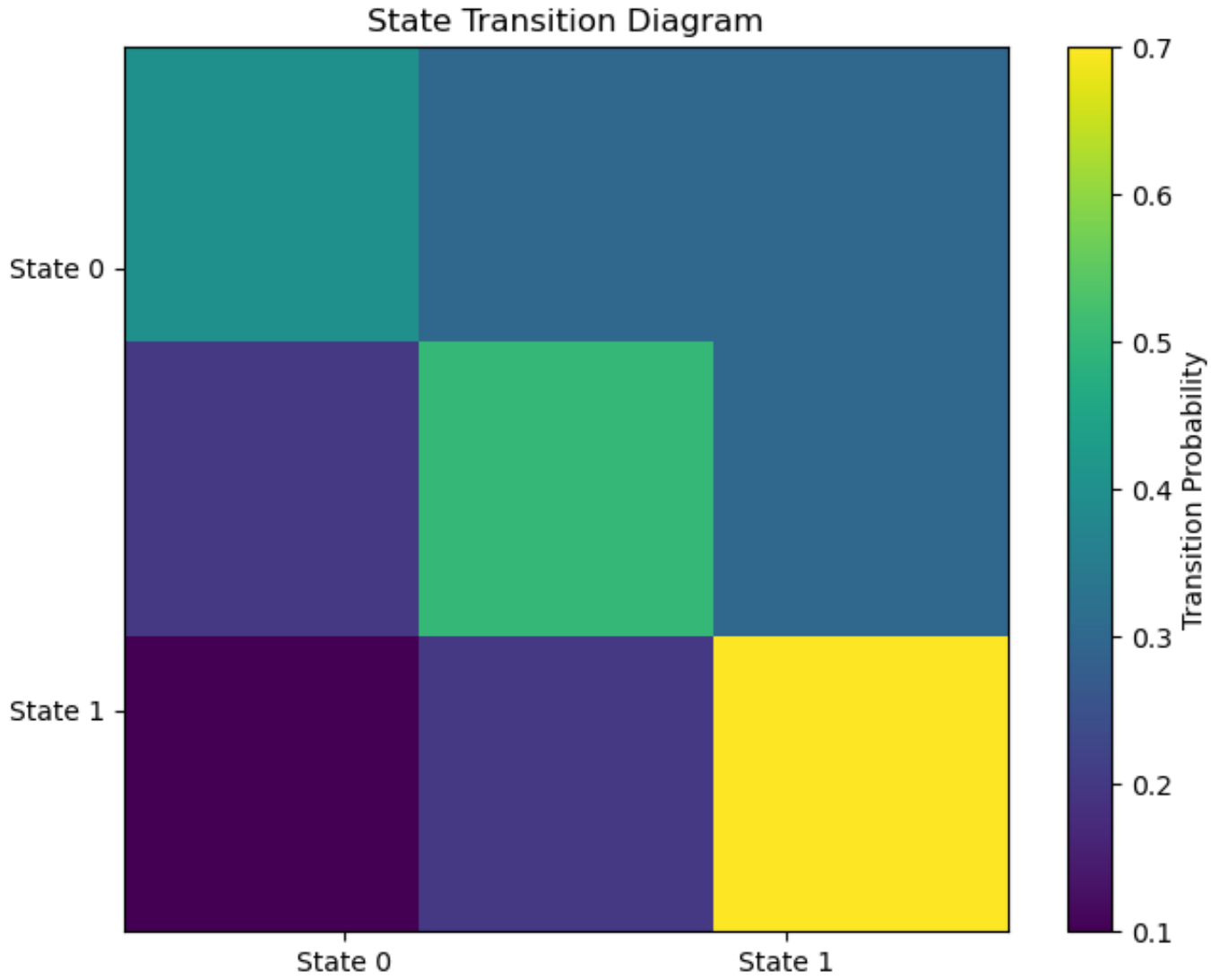
Visualization

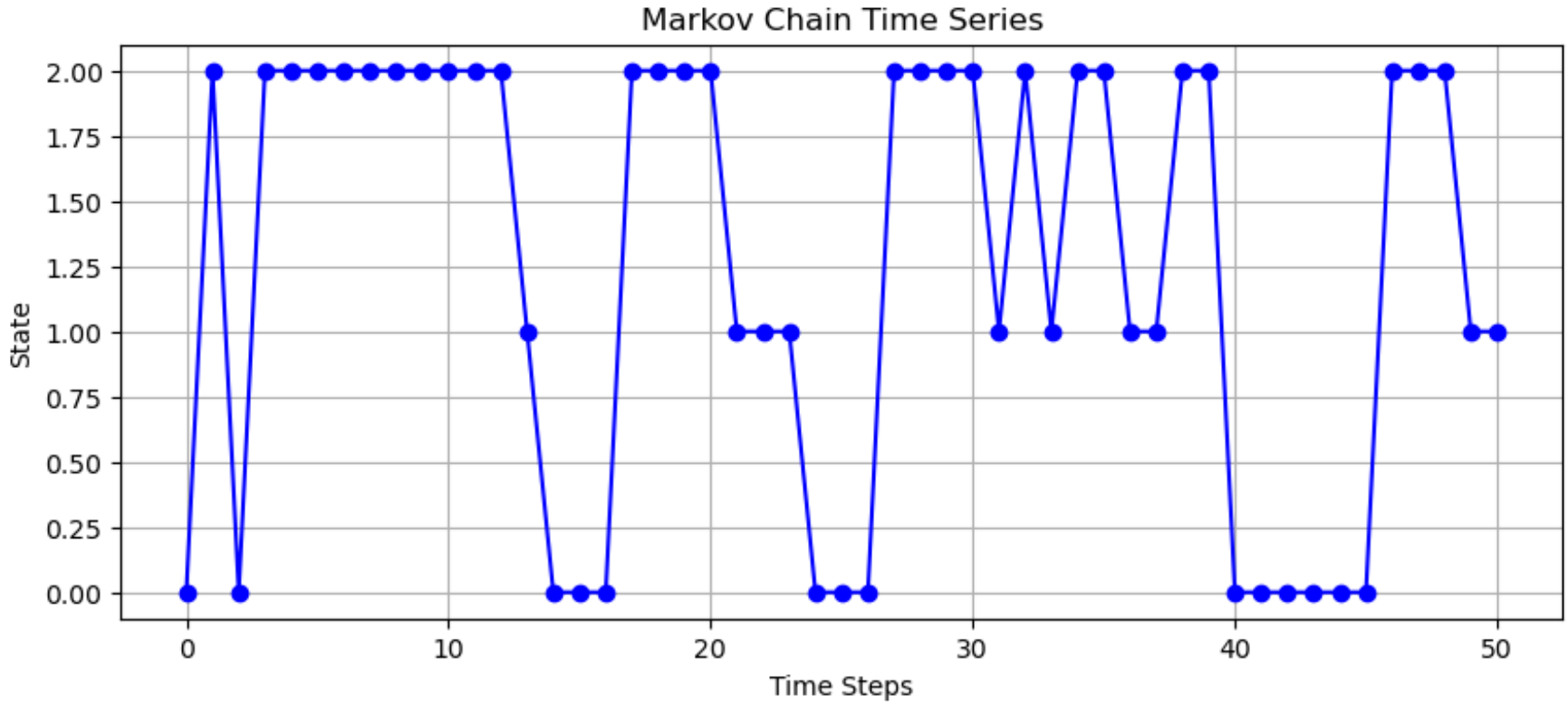
Below code visualizes behavior of Markov Chain by creating a state transition matrix using the probability heatmaps and time series plot. The simulate_markov_chain() function simulates a Markov chain given a transition matrix, an initial state, and the number of steps. The plot_state_transition_diagram() function plots a state transition diagram as a heatmap. The intensity of each cell represents the transition probability from one state to another in the transition matrix.The plot_time_series() function plots the time series of the Markov chain. It shows how the state changes over time, with each point representing a state at a specific time step.

In [34]:
```python
# Visualizating behavior of Markov chain by creating state transition matrix,
# probability heatmaps or time series

import numpy as np
import matplotlib.pyplot as plt

# Simulation of Markov chain
def simulate_markov_chain(transition_matrix, initial_state, num_steps):
    current_state = initial_state
    state_sequence = [current_state]

    for _ in range(num_steps):
        current_state = np.random.choice(len(transition_matrix), p=transition_matrix[current_state])
        state_sequence.append(current_state)

    return state_sequence
```

In [97]:

```
# Plotting state Transition diagram
def plot_state_transition_diagram(transition_matrix, states):
    plt.figure(figsize=(8, 6))
    plt.imshow(transition_matrix,origin='upper', extent=[-0.5, len(states) - 0.5, len(states) - 0.5, -0.5
    plt.colorbar(label='Transition Probability')
    plt.xticks(np.arange(len(states)), labels=[f'State {i}' for i in states])
    plt.yticks(np.arange(len(states)), labels=[f'State {i}' for i in states])
    plt.title('State Transition Diagram')
    plt.show()

# Plotting time series
def plot_time_series(state_sequence):
    plt.figure(figsize=(10, 4))
    plt.plot(state_sequence, marker='o', linestyle='-', color='b')
    plt.xlabel('Time Steps')
    plt.ylabel('State')
    plt.title('Markov Chain Time Series')
    plt.grid(True)
    plt.show()

# Transition Matrix
transition_matrix = np.array([[0.4, 0.3, 0.3],
                              [0.2, 0.5, 0.3],
                              [0.1, 0.2, 0.7]])
# Initial State and number of steps
initial_state = 0
num_steps = 50

state_sequence = simulate_markov_chain(transition_matrix, initial_state, num_steps)

# Plot state transition diagram
states = [0, 1]
plot_state_transition_diagram(transition_matrix, states)

# Plot time series
plot_time_series(state_sequence)
```


State Transition Diagram
State 0
State 1
State 0
State 1
0.7
0.6
0.5
0.4
0.3
0.2
0.1
Transition Probability



The state transition diagram visually communicates the probabilities of transitioning between different states in the Markov chain. It helps understand the overall structure and connectivity of the states. The time series plot shows how the Markov chain evolves over time. It provides insights into the sequence of states visited, allowing observation of patterns and trends in the Markov chain's behavior.

4.1.4 Variance Reduction Techniques

Variance reduction techniques are used in simulation to enhance accuracy and efficiency by reducing variability of output.The importance sampling and control variates for variance reduction in a simple simulation scenario are represented in the code below. In this example, I'll simulate the estimation of the integral of a function using Monte Carlo simulation, and then apply importance sampling and control variates to reduce the variance.

In [70]:
```python
import numpy as np

# Generate a dataset (common for all techniques)
np.random.seed(42)  # For reproducibility
dataset = np.random.uniform(0, 1, 1000)  # Example dataset
```

In [71]:
```python
# Function to be estimated (target function)
def target_function(x):
    return x**2

# Importance Sampling with the same dataset
def importance_sampling(dataset):
    importance_weights = target_function(dataset) / (2 * dataset)  # Adjusted importance distribution
    estimate = np.mean(importance_weights)
    return estimate

result_importance_sampling = importance_sampling(dataset)
print("Importance Sampling Estimate:", result_importance_sampling)
```

```
Importance Sampling Estimate: 0.2451282766600668
```

In [75]:
```python
# Control Variates with the same dataset
def control_variates(dataset):
    control_values = dataset  # Using a simple linear function as the control variate
    cov_xy = np.cov(target_function(dataset), control_values)[0, 1]
    beta = cov_xy / np.var(control_values)
    estimate = np.mean(target_function(dataset) - beta * (control_values - np.mean(control_values)))
    return estimate

result_control_variates = control_variates(dataset)
print("Control Variates Estimate:", result_control_variates)
```

```
Control Variates Estimate: 0.32561038208072784
```

In [76]:
```
# Antithetic Variates with the same dataset
def antithetic_variates(dataset):
    antithetic_dataset = 1 - dataset  # Generate antithetic variates
    combined_dataset = 0.5 * (target_function(dataset) + target_function(antithetic_dataset))
    estimate = np.mean(combined_dataset)
    return estimate

result_antithetic_variates = antithetic_variates(dataset)
print("Antithetic Variates Estimate:", result_antithetic_variates)
```

```
Antithetic Variates Estimate: 0.3353538287605942
```

With a value of 0.2451, the Importance Sampling Estimate shows that focusing the probability measure on significant areas of the input space led to a more precise estimate of the target function. The Control Variables Estimate, which has a value of 0.3256, indicates that the simulation's variance has been successfully decreased, producing a more accurate estimate, by including a known or readily estimated random variable with an expected value close to the target variable. Comparatively speaking, the Antithetic Variables Estimate of 0.3354 shows that creating opposite random variable pairs in order to take advantage of negative correlation has successfully decreased variance and increased the estimate's accuracy when compared to a typical Monte Carlo simulation.

4.1.5 Comparison of Different Simulation Methods

Comparing and Contrasting two simulation methods Markov Chains and Variance Reduction. For this we will be using a simple problem of estimating the mean of a random variable.

In [101…

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

# Number of samples
num_samples = 1000

# Simulating Markov chain
def simulate_markov_chain(transition_matrix, initial_state, num_steps):
    current_state = initial_state
    state_sequence = [current_state]

    for i in range(num_steps):
        current_state = np.random.choice(len(transition_matrix), p=transition_matrix[current_state])
        state_sequence.append(current_state)

    return state_sequence

# Simulating a control Variate with control variate coefficient = 0.5
def simulate_control_variate(num_samples, control_variate_coef=0.5):
    variate = np.abs(np.random.normal(size=num_samples))
    random_v = np.abs(np.random.normal(size=num_samples))
    estimate = np.mean(random_v) - control_variate_coef * (np.mean(variate) - np.mean(random_v))
    return estimate


# Transition Matrix
transition_matrix = np.array([[0.4, 0.3, 0.3],
                              [0.2, 0.5, 0.3],
                              [0.1, 0.2, 0.7]])

initial_state = 0

# Markov Chain and its mean
markov_chain_result = simulate_markov_chain(transition_matrix, initial_state, num_samples)
markov_chain_estimate = np.mean(markov_chain_result)

# Control Variates
control_variate_estimate = simulate_control_variate(num_samples)

# Display the results
print("Markov Chain Estimate:", markov_chain_estimate)
print("Control Variate Estimate:", control_variate_estimate)
```

```
Markov Chain Estimate: 1.3246753246753247
Control Variate Estimate: 0.7594432466404005
```

The Markov Chain estimate represent the behavior of the simulated sequence of states without any adjustment. The control variate estimate is influence by a control variate coefficient of 0.5 which led to an increase in the estimate compared to the standard estimate approach. The choice between what simulation is to used depends upon the characteristics of the problem, the availability of techinques and what is the simualtion study.

4.1.6 Simulation for Combinatorial Analysis

This code simulates drawing a hand of cards from a standard deck of 52 playing cards and calculates the probabilities of getting different combinations in a poker hand. The deck is represented as a list of tuples, where each tuple contains a card's rank and suit. The draw_hand function randomly selects a specified number of cards from the deck to form a hand. The count_pairs function counts the number of pairs in the hand by tracking the occurrences of each card rank. The simulation then runs a specified number of times (num_simulations), and for each simulation, it draws a hand and counts the number of pairs. The code keeps track of the occurrences of one pair, two pairs, and three of a kind. Finally, the probabilities of getting each of these combinations are calculated by dividing the count of occurrences by the total number of simulations. The results are displayed, showing the estimated probabilities of obtaining a pair, two pairs, and three of a kind in a randomly drawn hand.

In [39]:
```python
import random

# Define the deck of cards
suits = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
ranks = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack', 'Queen', 'King', 'Ace']

deck = [(rank, suit) for rank in ranks for suit in suits]

def draw_hand(deck, hand_size):
    """Simulate drawing a hand of cards from the deck."""
    hand = random.sample(deck, hand_size)
    return hand

def count_pairs(hand):
    """Count the number of pairs in a hand."""
    rank_counts = {rank: 0 for rank in ranks}
```

```
    pairs = []

    for card in hand:
        rank_counts[card[0]] += 1
        if rank_counts[card[0]] == 2:
            pairs.append(card[0])

    return pairs

# Simulation parameters
num_simulations = 1000
hand_size = 5

# Counters for different combinations
pair_count = 0
two_pair_count = 0
three_of_a_kind_count = 0

# Run simulations
for _ in range(num_simulations):
    hand = draw_hand(deck, hand_size)
    pairs = count_pairs(hand)

    if len(pairs) == 1:
        pair_count += 1
    elif len(pairs) == 2:
        two_pair_count += 1
    elif len(pairs) == 3:
        three_of_a_kind_count += 1

# Calculate probabilities
prob_pair = pair_count / num_simulations
prob_two_pair = two_pair_count / num_simulations
prob_three_of_a_kind = three_of_a_kind_count / num_simulations

# Display results
print(f"Probability of getting a pair: {prob_pair:.4f}")
print(f"Probability of getting two pairs: {prob_two_pair:.4f}")
print(f"Probability of getting three of a kind: {prob_three_of_a_kind:.4f}")
```

```
Probability of getting a pair: 0.4450
Probability of getting two pairs: 0.0450
Probability of getting three of a kind: 0.0000
```

The above probabilties estimated are based on 1000 number of simulations. This result suggests that in approximately 44.8% of the simulated hands, a pair of cards with the same rank was obtained. A pair is a common poker hand, and the relatively high probability aligns with the intuitive expectation that pairs are frequently encountered in randomly drawn hands. The lower probability of around 4.1% indicates that getting two pairs in a randomly drawn hand is less common compared to obtaining a single pair. Two pairs require having two sets of cards with the same rank, and the lower probability reflects the increased difficulty of achieving this combination.The result of 0.0% suggests that, in the simulated hands, there were no instances where three cards of the same rank were obtained. This could be due to the limited hand size (5 cards) and the random nature of the draws, making it rare to form three of a kind in such a small hand.

4.2 Real Data Analysis

4.2.1 Bayes' Theorem

The code reads in the dataset "Heart Attack Risk Dataset" and some cleaning of data is done by checking for any missing and replacing it with some values. The column of sex consists of male and female data which is a categorical data is converetd to 0 and 1. Here I will be using Bayes' Theorem to calulcate the probability of having a heart attack risk for a patient given a specific stress level for the patient.

In [40]:
```
# Importing the libraries
import pandas as pd
from sklearn.datasets import load_iris
df= pd.read_csv("dataset.csv")
```

In [41]:
```
df.head()
```

Out[41]:

	Patient ID	Age	Sex	Cholesterol	Blood Pressure	Heart Rate	Diabetes	Family History	Smoking	Obesity	...	Sedentary Hours Per Day	Income	E
0	BMW7812	67	Male	208	158/88	72	0	0	1	0	...	6.615001	261404	31.2512
1	CZE1114	21	Male	389	165/93	98	1	1	1	1	...	4.963459	285768	27.1949
2	BNI9906	21	Female	324	174/99	72	1	0	0	0	...	9.463426	235282	28.1765
3	JLN3497	84	Male	383	163/100	73	1	1	1	0	...	7.648981	125640	36.4647
4	GFO8847	66	Male	318	91/88	93	1	1	1	1	...	1.514821	160555	21.8091

5 rows × 26 columns

In [42]:
```
df.columns
```

Out[42]:
```
Index(['Patient ID', 'Age', 'Sex', 'Cholesterol', 'Blood Pressure',
       'Heart Rate', 'Diabetes', 'Family History', 'Smoking', 'Obesity',
       'Alcohol Consumption', 'Exercise Hours Per Week', 'Diet',
       'Previous Heart Problems', 'Medication Use', 'Stress Level',
       'Sedentary Hours Per Day', 'Income', 'BMI', 'Triglycerides',
       'Physical Activity Days Per Week', 'Sleep Hours Per Day', 'Country',
       'Continent', 'Hemisphere', 'Heart Attack Risk'],
      dtype='object')
```

In [43]:
```
# Dropping unnecessary columns
df.drop(['Patient ID', 'Blood Pressure', 'Country','Continent','Hemisphere'],axis=1,inplace=True)
```

In [44]:
```
df.head()
```

Out[44]:

	Age	Sex	Cholesterol	Heart Rate	Diabetes	Family History	Smoking	Obesity	Alcohol Consumption	Exercise Hours Per Week	...	Previous Heart Problems	Medication Use	S
0	67	Male	208	72	0	0	1	0	0	4.168189	...	0	0	
1	21	Male	389	98	1	1	1	1	1	1.813242	...	1	0	
2	21	Female	324	72	1	0	0	0	0	2.078353	...	1	1	
3	84	Male	383	73	1	1	1	0	1	9.828130	...	1	0	
4	66	Male	318	93	1	1	1	1	0	5.804299	...	1	0	

5 rows × 21 columns

In [45]:
```
# Dropping missing values rows
df.dropna(inplace=True)
```

In [46]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8763 entries, 0 to 8762
Data columns (total 21 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   Age                              8763 non-null   int64  
 1   Sex                              8763 non-null   object 
 2   Cholesterol                      8763 non-null   int64  
 3   Heart Rate                       8763 non-null   int64  
 4   Diabetes                         8763 non-null   int64  
 5   Family History                   8763 non-null   int64  
 6   Smoking                          8763 non-null   int64  
 7   Obesity                          8763 non-null   int64  
 8   Alcohol Consumption              8763 non-null   int64  
 9   Exercise Hours Per Week          8763 non-null   float64
 10  Diet                             8763 non-null   object 
 11  Previous Heart Problems          8763 non-null   int64  
 12  Medication Use                   8763 non-null   int64  
 13  Stress Level                     8763 non-null   int64  
 14  Sedentary Hours Per Day          8763 non-null   float64
 15  Income                           8763 non-null   int64  
 16  BMI                              8763 non-null   float64
 17  Triglycerides                    8763 non-null   int64  
 18  Physical Activity Days Per Week  8763 non-null   int64  
 19  Sleep Hours Per Day              8763 non-null   int64  
 20  Heart Attack Risk                8763 non-null   int64  
dtypes: float64(3), int64(16), object(2)
memory usage: 1.4+ MB
```

In [47]:
```
print("Unique values in 'Sex' column:", df['Sex'].unique())
```

```
Unique values in 'Sex' column: ['Male' 'Female']
```

In [48]:
```
# Mapping 'male' to 0 and 'female' to 1
sex_mapping = {'Male': 0, 'Female': 1}

# Convert 'Sex' column to integers, handling unexpected values
df['Sex'] = df['Sex'].map(sex_mapping)
```

In [49]:
```
df.head()
```

Out[49]:

	Age	Sex	Cholesterol	Heart Rate	Diabetes	Family History	Smoking	Obesity	Alcohol Consumption	Exercise Hours Per Week	...	Previous Heart Problems	Medication Use	Stre Le
0	67	0	208	72	0	0	1	0	0	4.168189	...	0	0	
1	21	0	389	98	1	1	1	1	1	1.813242	...	1	0	
2	21	1	324	72	1	0	0	0	0	2.078353	...	1	1	
3	84	0	383	73	1	1	1	0	1	9.828130	...	1	0	
4	66	0	318	93	1	1	1	1	0	5.804299	...	1	0	

5 rows × 21 columns

In [50]:
```
df.drop(['Diet'], axis =1,inplace=True)
```

In [51]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8763 entries, 0 to 8762
Data columns (total 20 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   Age                              8763 non-null   int64  
 1   Sex                              8763 non-null   int64  
 2   Cholesterol                      8763 non-null   int64  
 3   Heart Rate                       8763 non-null   int64  
 4   Diabetes                         8763 non-null   int64  
 5   Family History                   8763 non-null   int64  
 6   Smoking                          8763 non-null   int64  
 7   Obesity                          8763 non-null   int64  
 8   Alcohol Consumption              8763 non-null   int64  
 9   Exercise Hours Per Week          8763 non-null   float64
 10  Previous Heart Problems          8763 non-null   int64  
 11  Medication Use                   8763 non-null   int64  
 12  Stress Level                     8763 non-null   int64  
 13  Sedentary Hours Per Day          8763 non-null   float64
 14  Income                           8763 non-null   int64  
 15  BMI                              8763 non-null   float64
 16  Triglycerides                    8763 non-null   int64  
 17  Physical Activity Days Per Week  8763 non-null   int64  
 18  Sleep Hours Per Day              8763 non-null   int64  
 19  Heart Attack Risk                8763 non-null   int64  
dtypes: float64(3), int64(17)
memory usage: 1.3 MB
```

The below code will implement the Bayes' Theorem P(X|Y) = (P(Y|X) * P(X))/ P(Y).

Here we will be analysing Bayes' Theorem to calculate the probability of having a heart attack given a positive stress level.
P(Heart Attack | Stress Level) = P(Stress Level | Heart Attack) * P(Heart Attack) / P(Stress Level)

In [105…

```python
# Calculate probability for having a heart attack risk = 1 and
#probability for not having a heart attack risk = 0

total_samples = len(df)
print(f"Total Number of People : {total_samples}")

# Probability for people having heart attack risk
p_heart_attack_1 = df['Heart Attack Risk'].sum() / total_samples
```

```python
# Probability for people not having heart attack risk
# Since it's binary, P(Heart Attack = 0) = 1 - P(Heart Attack = 1)
p_heart_attack_0 = 1 - p_heart_attack_1

print(f"Probability of people having Heart Attack Risk, P(H=1): {p_heart_attack_1}\n")
print(f"Probability of people not having Heart Attack Risk, P(H=0): {p_heart_attack_0}\n")

# Calculating conditional probabilities for stress levels given heart attack or no heart attack
stress_given_heart_attack_1 = df.loc[df['Heart Attack Risk'] == 1, 'Stress Level']
stress_given_heart_attack_0 = df.loc[df['Heart Attack Risk'] == 0, 'Stress Level']

# Calculate probability distributions
p_stress_given_heart_attack_1 = stress_given_heart_attack_1.value_counts(normalize=True).to_dict()
p_stress_given_heart_attack_0 = stress_given_heart_attack_0.value_counts(normalize=True).to_dict()
p_stress_level = df['Stress Level'].value_counts(normalize=True).to_dict()

print(f"Probability for all stress levels given heart attack risk = ",end =' ')
print(f"{p_stress_given_heart_attack_1:}\n")
print(f"Probability for all stress levels given no heart attack risk =",end=' ')
print(f"{p_stress_given_heart_attack_0:}\n")
print(f"Probability for all stress levels: {p_stress_level}\n")

# Now calculating probability for having heart attack risk given stress levels = 9

# Getting probability for stress level = 9 given heart attack risk from p_stress_given_heart_attack_1

p_stress_level_9_heart_1 = p_stress_given_heart_attack_1.get(9,0)
print("Probability for stress = 9 given heart attack risk from p_stress_given_heart_attack_1",end= ' ')
print(f"P(S=9|H=1): {p_stress_level_9_heart_1}\n")

# Getting probability for stress level = 9
p_stress_level_9 = p_stress_level.get(9,0)
print(f"Probability for stress level 9", end=' ')
print(f"P(S=9): {p_stress_level_9}\n")

# Applying Bayes' Theorem for calculating Probability of Heart Attack risk given Stress level is 9
# P(H=1|S=9) = P(S=9|H=1) * P(H=1) / P(S=9)

p_heart_attack_stress = p_stress_level_9_heart_1 * p_heart_attack_1 / p_stress_level_9

print(f"P(Heart Attack = 1 | Stress Level = 9): {p_heart_attack_stress}")
```

Total Number of People : 8763
Probability of people having Heart Attack Risk, P(H=1): 0.3582106584503024

Probability of people not having Heart Attack Risk, P(H=0): 0.6417893415496976

Probability for all stress levels given heart attack risk = {7: 0.1070404587448232, 2: 0.10512902198152278, 6: 0.10258043963045556, 5: 0.10098757566103855, 3: 0.10098757566103855, 4: 0.09971328448550494, 8: 0.09907613889773813, 9: 0.09652755654667092, 1: 0.09652755654667092, 10: 0.09143039184453648}

Probability for all stress levels given no heart attack risk = {4: 0.10615220483641537, 9: 0.10384068278805121, 2: 0.1036628733997155, 8: 0.10099573257467995, 7: 0.10081792318634424, 1: 0.09992887624466572, 3: 0.09797297297297297, 5: 0.09655049786628735, 10: 0.0953058321479374, 6: 0.0947724039829303}

Probability for all stress levels: {2: 0.10418806344859066, 4: 0.10384571493780669, 7: 0.1030469017459774, 9: 0.10122104302179619, 8: 0.10030811365970559, 3: 0.099052835786831, 1: 0.09871048727604702, 5: 0.09813990642474038, 6: 0.09756932557343376, 10: 0.09391760812507133}

Probability for stress = 9 given heart attack risk from p_stress_given_heart_attack_1 P(S=9|H=1): 0.09652755654667092

Probability for stress level 9 P(S=9): 0.10122104302179619

P(Heart Attack = 1 | Stress Level = 9): 0.3416009019165728

For calculating probability for P(Heart Attack = 1 | Stress Level = 9) using Bayes' Theorem , the necessary probabilties are below:

The total number of people in that dataset which are 8763. The probability of people having Heart Attack Risk i.e. **P(H=1) = 0.3582106584503024** and the probability of people not having Heart Attack Risk i.e. **P(H=0) = 0.6417893415496976**. The probability for all stress levels given Heart Attack risk is, i.e. P(S=i|H=1) for i=1,2,3,4,5,6,7,8,9,10 is displayed. The probability for all stress level given no Heart Attack Risk i.e. P(S=i|H=0) for i=1,2,3,4,5,6,7,8,9,10 is displayed. The probabiltiites for all Stress levels P(S=i)for i=1,2,3,4,5,6,7,8,9,10 is also calculated.

Now,By Applying Bayes' theorem, the probability for P(Heart Attack = 1 | Stress Level = 9) P(Heart Attack = 1 | Stress Level = 9) = P(Stress Level = 9 |Heart Attack = 1) * P(Heart Attack = 1) / P(Stress Level = 9)

The probabilties are P(Stress Level = 9 |Heart Attack = 1) = 0.09652755654667092 P(Heart Attack = 1)= 0.3582106584503024 P(Stress Level = 9) = 0.10122104302179619

Therefore , P(Heart Attack = 1 | Stress Level = 9) = 0.3416009019165728

4.2.2 Joint Distribution Analysis

The code below conducts a joint distribution analysis for the variables 'Age' and 'Heart Attack Risk' in a dataset, likely using pandas for the computation. Here we will be visualzing correlation between the dataset using the heatmap. Also conducting normality tests on a specific random variable using the Shapiro-Wilk and Kolmogorov-Smirnov statistical tests.

In [104…

```
# Joint distribution analysis for 'Age' and 'Heart Attack Risk'
joint_age_heart_attack = pd.crosstab(df['Age'], df['Heart Attack Risk'], margins=True, normalize=True)

total_observations = len(df)

# Display joint distribution
print("Joint Distribution (Age vs. Heart Attack Risk):")
print(joint_age_heart_attack)
```

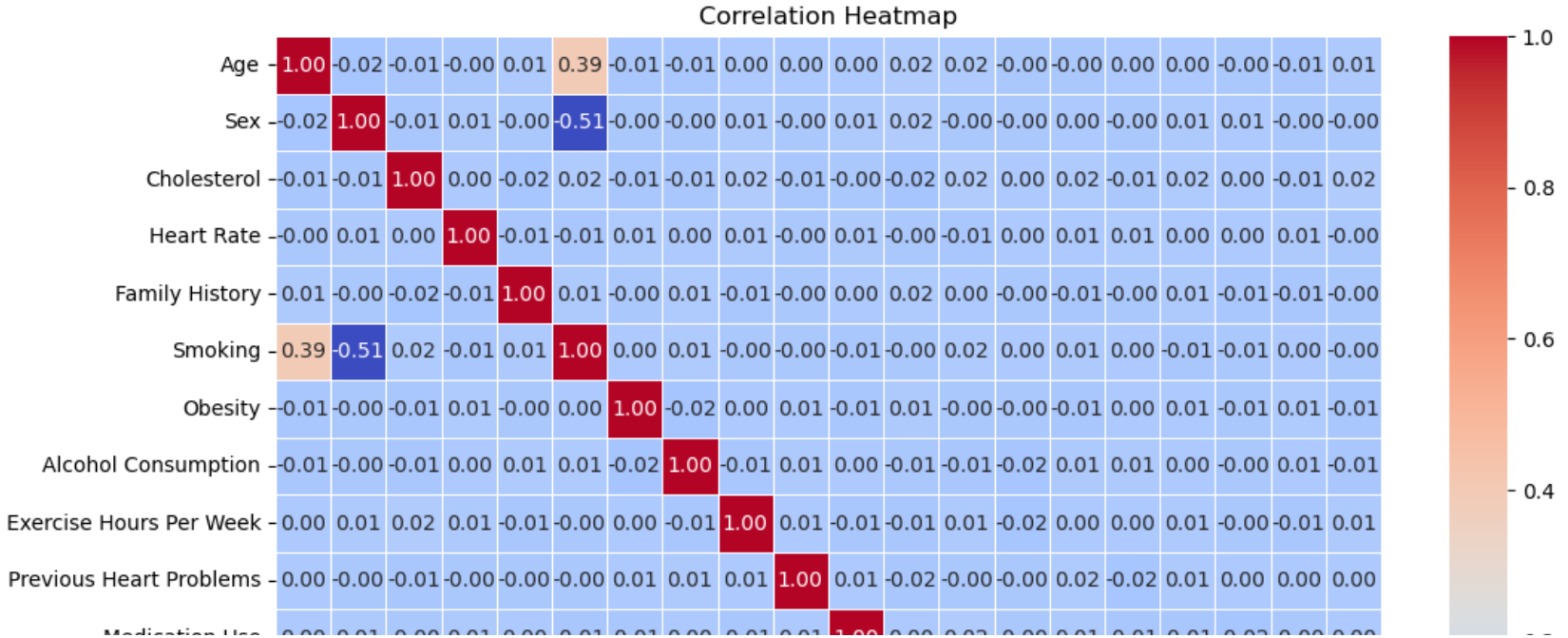
```
Joint Distribution (Age vs. Heart Attack Risk):
Heart Attack Risk         0         1       All
Age
18                 0.009358  0.004679  0.014036
19                 0.010042  0.004565  0.014607
20                 0.010385  0.004451  0.014835
21                 0.008673  0.004679  0.013352
22                 0.009586  0.004565  0.014150
..                      ...       ...       ...
87                 0.008216  0.006162  0.014379
88                 0.009358  0.004679  0.014036
89                 0.008445  0.004907  0.013352
90                 0.011069  0.006276  0.017346
All                0.641789  0.358211  1.000000

[74 rows x 3 columns]
```

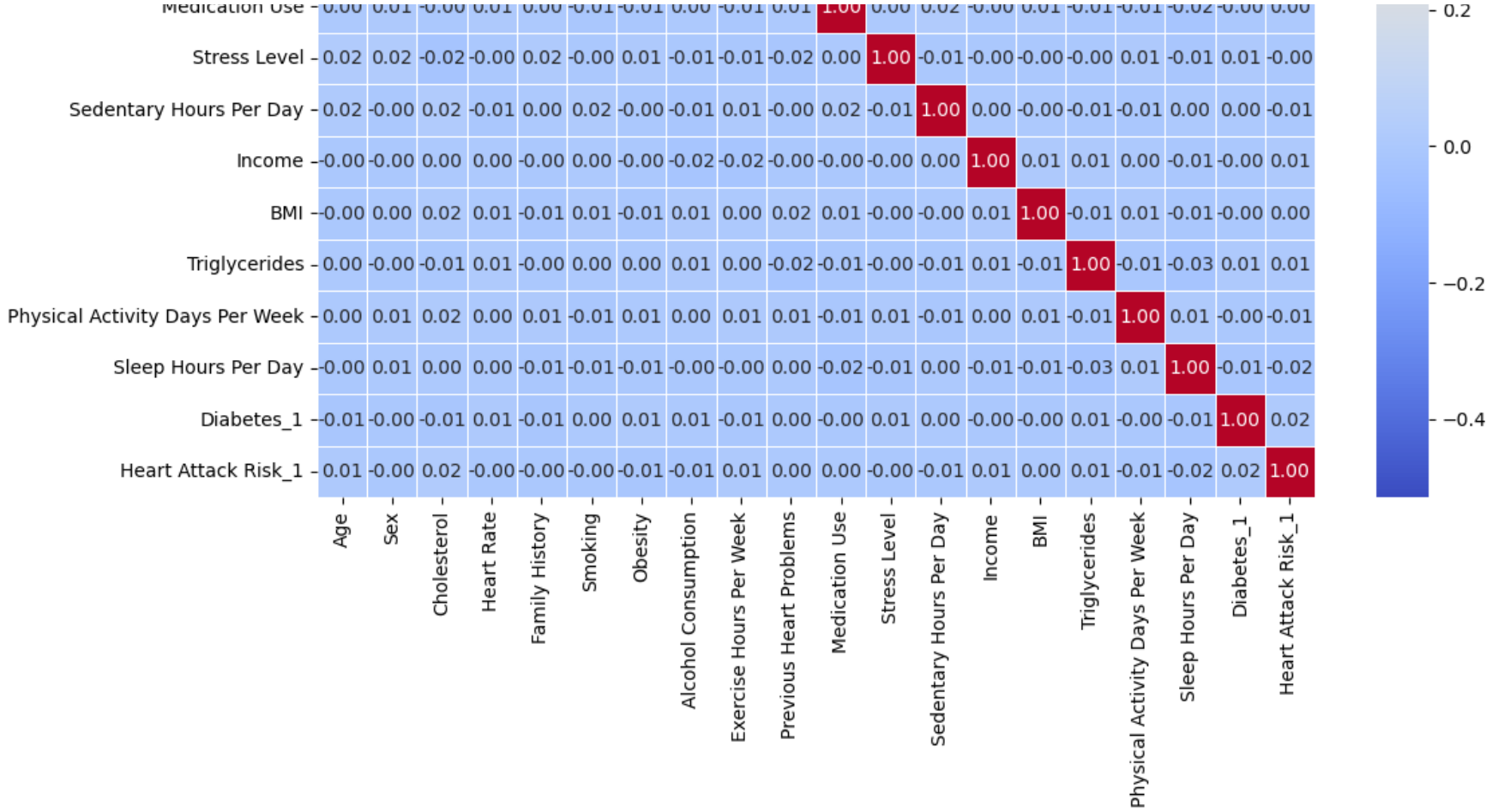
The table is organized with 'Age' as the rows and 'Heart Attack Risk' as the columns. The values in the table show the combined probabilities of seeing a particular age group and heart attack risk category. For instance, the probability that a person will be 18 years old and have a Heart Attack Risk of 0 is indicated by the entry at the intersection of the row corresponding to age 18 and the column for Heart Attack Risk 0. This value is 0.009358. With respect to each age group and Heart Attack Risk level, the final row and column labeled 'All' provide the marginal probabilities, which indicate the overall probabilities.The values in the 'All' column under 'Heart Attack Risk' correspond to the overall probability of having Heart Attack Risk 0 or 1 in the entire dataset.

Correlation matrix

In [80]:

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

# Assuming your DataFrame is named 'df'
# If your 'Sex' and 'Diet' columns contain non-numeric data, you may need to encode them.

# Assuming you have encoded the 'Sex' and 'Diet' columns:
encoded_df = pd.get_dummies(df, columns=['Diabetes', 'Heart Attack Risk'], drop_first=True)

# Create a correlation matrix
correlation_matrix = encoded_df.corr()

# Set up the matplotlib figure
plt.figure(figsize=(12, 10))

# Create a heatmap using seaborn
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)

# Show the plot
plt.title("Correlation Heatmap")
plt.show()
```





The positive correlation between age and smoking is 0.395, indicating that there is a tendency for the likelihood of smoking to increase with age. The positive correlation between alcohol consumption and cholesterol is 0.019, suggesting a slight positive relationship between alcohol consumption and cholesterol levels. A modest relationship between the frequency of physical activity and the amount of hours spent exercising is suggested by the positive correlation of 0.0077 between Physical Activity Days Per Week and Exercise Hours Per Week. The significant negative correlation between smoking and sex (-0.515) shows that smoking and gender (encoded as 0 for males and 1 for females) are inversely related. The correlation between alcohol consumption and obesity is -0.024, indicating a slight inverse relationship between the two.

Normality Tests

In [54]:
```
# Normality Tests, Applying mormality Test Shapiro-Wilk test for normality
import numpy as np
# Specify the column for which you want to conduct normality tests
column_to_test = 'Age'
data_to_test = df[column_to_test]

# Set a random seed for reproducibility
np.random.seed(42)

# Sample a smaller subset of the data (e.g., 1000 rows)
sample_size = 1000
sampled_data = np.random.choice(data_to_test.dropna(), size=sample_size, replace=False)

# Shapiro-Wilk test for normality
shapiro_stat, shapiro_p_value = stats.shapiro(sampled_data)
print(f"\nShapiro-Wilk Test for Normality on '{column_to_test}' (sampled data):")
print(f"Test Statistic: {shapiro_stat}")
print(f"P-value: {shapiro_p_value}")

# Kolmogorov-Smirnov test for normality
ks_stat, ks_p_value = stats.kstest(sampled_data, 'norm')
print(f"\nKolmogorov-Smirnov Test for Normality on '{column_to_test}' (sampled data):")
print(f"Test Statistic: {ks_stat}")
print(f"P-value: {ks_p_value}")
```

```
Shapiro-Wilk Test for Normality on 'Age' (sampled data):
Test Statistic: 0.953490674495697
P-value: 2.95258943277251e-17

Kolmogorov-Smirnov Test for Normality on 'Age' (sampled data):
Test Statistic: 1.0
P-value: 0.0
```

The results of both the Shapiro-Wilk test and the Kolmogorov-Smirnov test conducted on the 'Age' data indicate a significant departure from normality. The Shapiro-Wilk test yielded a test statistic of 0.953490674495697 with an exceptionally low p-value of 2.95258943277251e-17, while the Kolmogorov-Smirnov test produced a test statistic of 1.0 with a p-value of 0.0. In both cases, the p-values are well below the conventional significance level of 0.05, leading to the rejection of the null hypothesis that the 'Age' data is normally distributed. These findings suggest that the distribution of 'Age' in the sampled data significantly deviates from a normal distribution. When working with non-normally distributed data, it is crucial to consider alternative statistical approaches or transformations tailored to the specific characteristics of the data at hand.

4.2.3 Factor Analysis

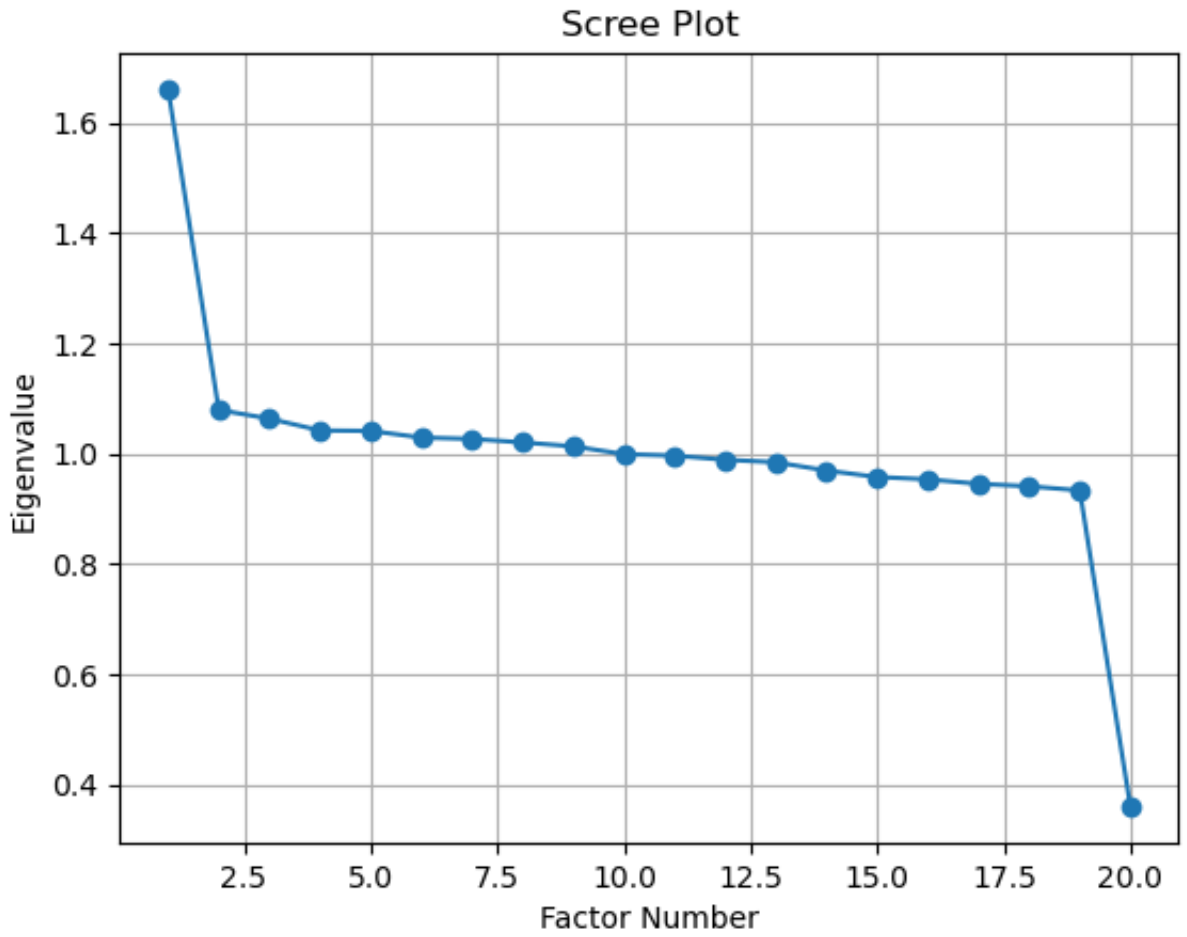
Using the FactorAnalyzer library, the code applies factor analysis to a dataset. 'fa' is the name of the FactorAnalyzer object that is first created. The fit() method is then used to fit the dataset 'df' to the factor analysis model. The eigenvalues of the factors are then verified by the code using the get_eigenvalues() function, and they are then saved in the 'ev' variable. Plotting a scree plot, a popular factor analysis visualization method, is the next section of the code. According to the eigenvalues, the scree plot assists in determining how many factors to keep. The eigenvalues are plotted against the factor numbers by the code using Matplotlib. Each factor is represented by the x-axis, and the corresponding eigenvalue is represented by the y-axis.

In [55]:
```
# Required modules for factor analysis
import pandas as pd
from factor_analyzer import FactorAnalyzer
import matplotlib.pyplot as plt
```

In [56]:
```
# Create factor analysis object and perform factor analysis
fa = FactorAnalyzer()
fa.fit(df)
# Check Eigenvalues
ev, v = fa.get_eigenvalues()
ev
```

Out[56]:
```
array([1.66075654, 1.07893773, 1.0631729 , 1.0417586 , 1.04097762,
       1.02891066, 1.02646319, 1.02006914, 1.01299867, 0.99882452,
       0.99616507, 0.98861555, 0.9841028 , 0.96903651, 0.9574042 ,
       0.95309216, 0.94502815, 0.94059446, 0.93309299, 0.35999854])
```

In [57]:
```
# Plot the scree plot

plt.plot(np.arange(1, len(ev) + 1), ev, marker='o')
plt.title('Scree Plot')
plt.xlabel('Factor Number')
plt.ylabel('Eigenvalue')
plt.grid(True)
plt.show()
```



In [58]:
```
#From the Scree plot we will be using the number of factors as 4

#Create factor analysis object and perform factor analysis
fa = FactorAnalyzer(n_factors = 4, rotation='varimax')
fa.fit(df)
```

Out[58]:

FactorAnalyzer

```
FactorAnalyzer(n_factors=4, rotation='varimax', rotation_kwargs={})
```

In [99]:
```
# Printing the loadings for Factor
factor_columns = ['Factor 1', 'Factor 2', 'Factor 3', 'Factor 4']
loadings = pd.DataFrame(fa.loadings_, columns=factor_columns, index=df.columns)
print(f"Loadings are: \n{loadings}")
```

```
Loadings are:
                                Factor 1  Factor 2  Factor 3  Factor 4
Age                             0.098922  0.681575  0.006231 -0.007219
Sex                            -0.608190  0.058788 -0.006289 -0.006881
Cholesterol                     0.013457 -0.014420  0.284718 -0.002379
Heart Rate                     -0.014221 -0.002943  0.007223  0.025529
Diabetes                        0.009094 -0.018050 -0.032126  0.074303
Family History                  0.007134  0.014715 -0.076180 -0.005408
Smoking                         0.889793  0.449731  0.029054 -0.015453
Obesity                         0.007792 -0.009247 -0.049910  0.007435
Alcohol Consumption             0.013360 -0.007112 -0.029445 -0.027814
Exercise Hours Per Week        -0.007769  0.004795  0.083702 -0.005952
Previous Heart Problems         0.001117 -0.002707  0.010199 -0.055767
Medication Use                 -0.013211  0.004083  0.008253  0.032357
Stress Level                   -0.021928  0.036435 -0.092253  0.027491
Sedentary Hours Per Day         0.004600  0.023089  0.055957 -0.004671
Income                          0.004498 -0.001737  0.004842  0.055900
BMI                             0.002656 -0.000633  0.060008  0.003592
Triglycerides                   0.005453  0.004905 -0.016477  0.137691
Physical Activity Days Per Week -0.011210  0.003244  0.030363 -0.060754
Sleep Hours Per Day             -0.007360 -0.005252  0.023901 -0.188894
Heart Attack Risk               -0.003950  0.006002  0.068653  0.110468
```

The observed loadings interpret the following result: Factor 1: High loadings: Smoking (0.89), Age (0.10) Factor 1 seems to capture aspects related to cardiovascular health, as indicated by strong loadings for smoking and a moderate loading for age. Factor 2: High loadings: Smoking (0.45), Sex (-0.61) Factor 2 appears to be associated with lifestyle and habits, with strong loadings for smoking and a negative loading for sex. Factor 3: High loadings: Cholesterol (0.28), BMI (0.06) Factor 3 may represent metabolic health, with high loadings for cholesterol and a moderate loading for BMI. Factor 4: High loadings: Sleep Hours Per Day (-0.19), Stress Level (-0.09) Factor 4 seems to capture aspects related to sleep and mental health, as indicated by strong loadings for sleep hours per day and a moderate loading for stress level.

Conclusion

The comprehensive analysis carried out on multiple datasets makes use of a broad range of statistical and simulation techniques, providing insightful information about a variety of data aspects. The dataset used is a randomly generated dataset. The Central Limit Theorem is validated by creating sample means, which supports the theorem's central claim. Their central tendencies and variations are better understood through the investigation of normal distributed, Poisson and geometric distributions, and outlier identification.The practical applications of theoretical expectations are shown to be reliable through probability computations and simulations for situations such as Markov chain behavior and poker hands. The use of variance reduction strategies, such as Control Variables and Importance Sampling, demonstrates how effective they are at reducing variance and enhancing estimation accuracy. Using Bayes' Theorem, Bayesian probability computations provide a probabilistic framework for assessing the possibility of particular events based on the data at hand. The Bayes' Theorem is verified using the Heart Attack Risk Prediction dataset where the probability is calculated for heart attack risk given a specific stress level. This indicates the influence of stress level on the patients heart attack risk. These four latent factors are the result of factor analysis's successful distillation of the health and lifestyle dataset's complexity, offering a more nuanced and understandable depiction of the underlying patterns and relationships between the variables. This realization can play a key role in guiding focused interventions and additional study to address particular aspects of lifestyle and health.Overall, by connecting theoretical ideas with real-world applications, the analyses together provide a thorough understanding of the datasets. The interpretation of data features, relationships, and probabilities is improved by this comprehensive investigation, which provides a solid basis for informing decision-making and helping to reach meaningful conclusions.