CreditWise : A Comprehensive Analysis of Credit Scores

By

Varad Hattekar
M.S Data Science
Department of Mathematical Sciences
Semester and year - Fall 2023
CWID - 20025448

Abstract : Analysing credit scores through advanced data analysis techniques.

This data analysis project dives deep into the intricate realm of credit scores, employing Bayesian theorem, joint probability distribution, and factor analysis to discover patterns and insights within real-world credit score data. By examining various customer attributes such as income, gender, number of children, and home ownership, I have tried to explain the factors contributing to both low and high credit scores.

The methodology involved the application of Bayesian theorem to assess the conditional probabilities of credit scores given specific customer features. Joint probability distributions were utilized to model the relationships between multiple variables, providing a comprehensive view of their interdependencies. Factor analysis was employed to identify latent variables influencing credit scores.

The analysis of both high and low credit scores show which factors and by how much affect the credit score of the individuals. Through analysis, I have identified correlations between income levels, demographic factors, and credit scores.

This data analysis project helps to have broader understanding of credit score dynamics and suggests actionable insights for financial institutions and individuals. Through this analysis of multiple factors affecting credit scores, institutions and individuals should aim to make right decisions in terms of lending and financial planning.

Chapter 1: Introduction

In modern times, the dynamics of money are changing every now and then. Earlier, the emphasis was lot on savings, but now more focus is on market investment and debt. The more you buy on your credit card and loan, the more economy progresses. In short, money is created by debt. Today, majority population of the world is under some kind of debt. To determine which individual is capable of handling how much debt, the system of credit scores is used. The better the credit score, the better your ability to repay your debt.

Markets and modern banking concepts use demographic data of customers to calculate credit scores. By leveraging advanced data analysis techniques, including Bayesian theorem, joint probability distribution, and factor analysis, this project aims to decipher the complex relationships between diverse customer attributes and credit scores.

This project gives proper reasoning to why some customers have low, average and high credit scores. Also explains how different attributes such as age, income, gender, house ownership and no. of children affect credit scores of customers. This project can become a good guide to individuals trying to understand the dynamics of credit scores and how to improve on them.

My aim through this project is simply to simplify the understanding of credit scores. The web of factors such as age, income, etc makes the understanding harder. But through graphs and diagrams, I have tried my best to explain the mystery of credit scores.

Chapter 2: Data Description

The project is divided into 4 parts i.e Simulation of Continous RVs, Simulating discrete RVs, Markov Chains and Real data analysis. The data generated for first 3 parts is random. This data is randomly generated to understand various theoretical concepts which proved useful for real data analysis.

For the last part i.e real data analysis, I did research and found out a suitable dataset on Kaggle. The link of the dataset is provided for reference: https://www.kaggle.com/datasets/sujithmandala/credit-score-classification-dataset/data (https://www.kaggle.com/datasets/sujithmandala/credit-score-classification-dataset/data)

The dataset had no errors or missing rows/columns/values. It has all the demographic details of people which are required to calculate and evaluate credit score.

The data includes following :

1. Age
2. Gender
3. Income
4. Education
5. Marital status
6. No. of children
7. Home ownership
8. Credit score

```
Chapter 3: Methodology

The methodology used in this project follows the instructions of Prof. Hadi Safari K. A systematic flowchart
and timeline was provided by the professor. The methodology follows 4 simple steps :

1. Simulation of continous random variables.
2. Simulation of discrete random variables.
3. Markov chains
4. Real data analysis

The order followed is the same as above.
```

Chapter 4: Analysis and results

4.1 SIMULATION DATA ANALYSIS

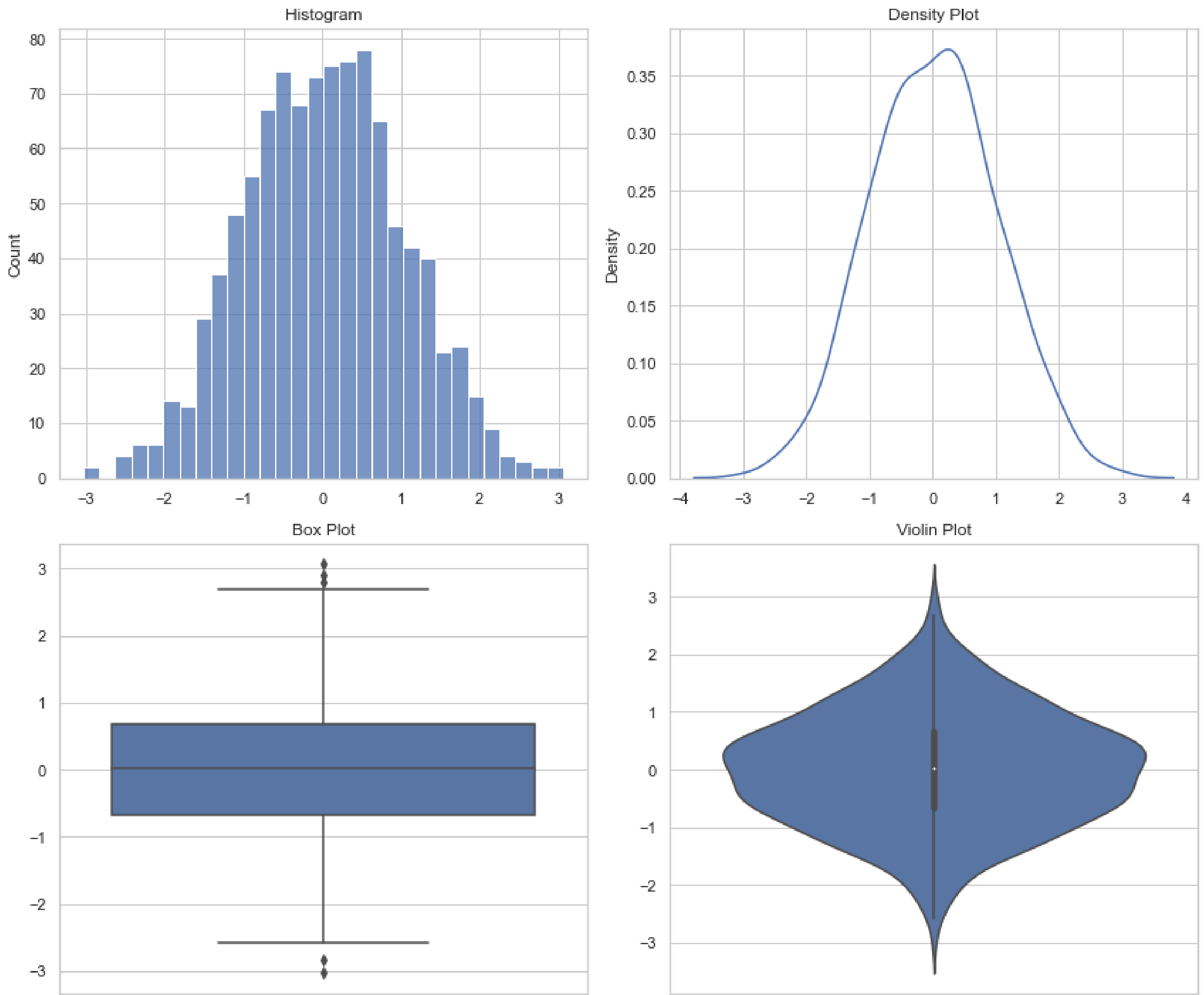
4.1.1 Simulating Continuous Random Variables

In [17]:
```python
# Generating random data

import numpy as np
mean = 0
std_dev = 1

random_values = np.random.normal(mean, std_dev, size=1000)
```

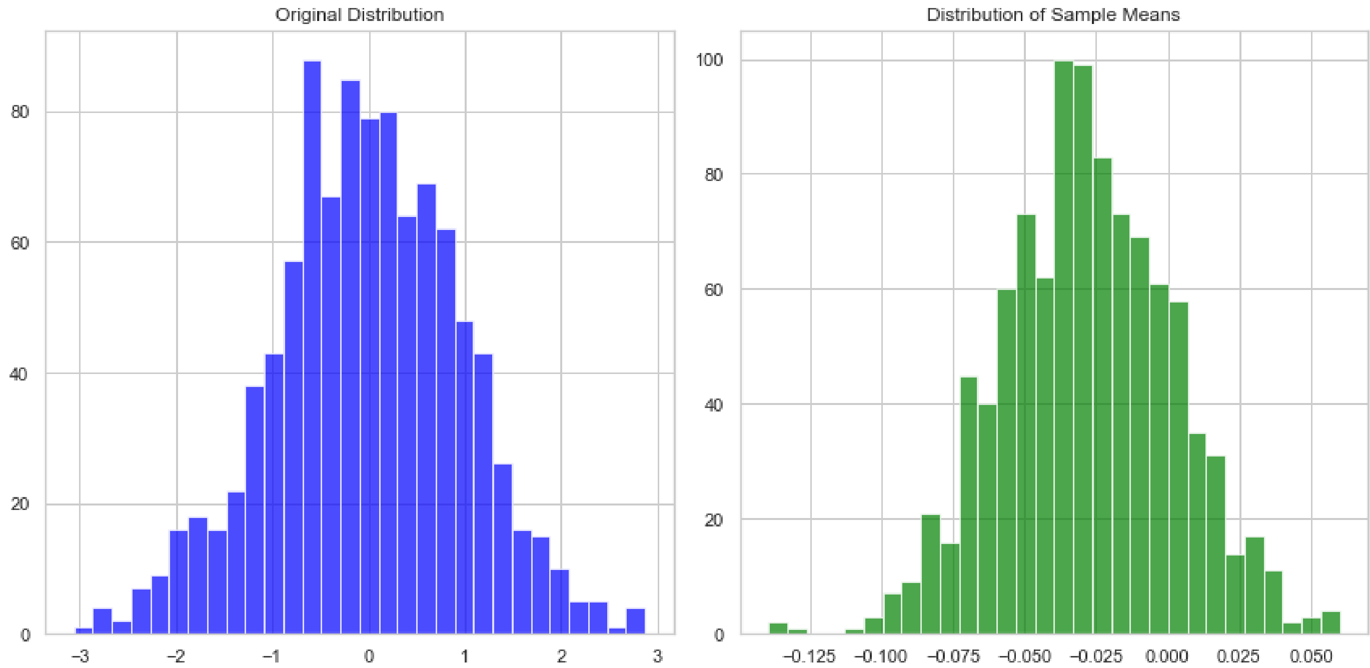
In [21]:
```python
# Statistical analysis

import numpy as np
from scipy.stats import mode

mean = 0
std_dev = 1
random_values = np.random.normal(mean, std_dev, size=1000)

mean_value = np.mean(random_values)
variance_value = np.var(random_values)
std_deviation_value = np.std(random_values)

first_quantile = np.percentile(random_values, 25)
third_quantile = np.percentile(random_values, 75)

mode_result = mode(random_values)
mode_value = mode_result.mode[0]

skewness_value = np.mean((random_values - mean_value) ** 3) / (std_deviation_value ** 3)
kurtosis_value = np.mean((random_values - mean_value) ** 4) / (std_deviation_value ** 4)

print("Mean:", mean_value)
print("Variance:", variance_value)
print("Standard Deviation:", std_deviation_value)
print("First Quantile (Q1):", first_quantile)
print("Third Quantile (Q3):", third_quantile)
print("Mode:", mode_value)
print("Skewness:", skewness_value)
print("Kurtosis:", kurtosis_value)
```

```
Mean: -0.06741167533937535
Variance: 1.0352276399372777
Standard Deviation: 1.017461370243253
First Quantile (Q1): -0.7200824455483543
Third Quantile (Q3): 0.6197754957330505
Mode: -3.7118548585026208
Skewness: -0.08423978572219087
Kurtosis: 3.128493621204394
```

In [22]:
```python
# VISUALISATION

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style="whitegrid")
mean = 0
std_dev = 1

random_values = np.random.normal(mean, std_dev, size=1000)
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

sns.histplot(random_values, bins=30, kde=False, ax=axes[0, 0])  # Plot histogram# Plot histogram
axes[0, 0].set_title('Histogram')

sns.kdeplot(random_values, ax=axes[0, 1])   # Plot density plot
axes[0, 1].set_title('Density Plot')

sns.boxplot(y=random_values, ax=axes[1, 0])  # Plot box plot
axes[1, 0].set_title('Box Plot')

sns.violinplot(y=random_values, ax=axes[1, 1])   # Plot violin plot
axes[1, 1].set_title('Violin Plot')

plt.tight_layout()
plt.show()
```



In [31]:
```
# CENTRAL LIMIT THEOREM

import numpy as np
import matplotlib.pyplot as plt

mean = 0
std_dev = 1
sample_size = 1000
num_samples = 1000

original_data = np.random.normal(mean, std_dev, size=1000)

def generate_sample_means(data, sample_size, num_samples):
    sample_means = [np.mean(np.random.choice(data, sample_size)) for _ in range(num_samples)]
    return sample_means

sample_means = generate_sample_means(original_data, sample_size, num_samples)

# Plot the original distribution and the distribution of sample means
plt.figure(figsize=(12, 6))

# Plot the original distribution
plt.subplot(1, 2, 1)
plt.hist(original_data, bins=30, color='blue', alpha=0.7)
plt.title('Original Distribution')

# Plot the distribution of sample means
plt.subplot(1, 2, 2)
plt.hist(sample_means, bins=30, color='green', alpha=0.7)
plt.title('Distribution of Sample Means')

plt.tight_layout()
plt.show()
```



In [36]:
```
# outlier detection using Z-Score method

import numpy as np
from scipy import stats

random_values = np.random.normal(0, 1, size=1000)
z_scores = np.abs(stats.zscore(random_values))
threshold = 3
outliers_zscore = np.where(z_scores > threshold)[0]
print("Indices of outliers (Z-score method):", outliers_zscore)
```

```
Indices of outliers (Z-score method): [203 228]
```

In [33]:
```python
#Outliers using IQR method

q1 = np.percentile(random_values, 25)
q3 = np.percentile(random_values, 75)
iqr = q3 - q1

lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr

outliers_iqr = np.where((random_values < lower_bound) | (random_values > upper_bound))[0]
print("Indices of outliers (IQR method):", outliers_iqr)
```

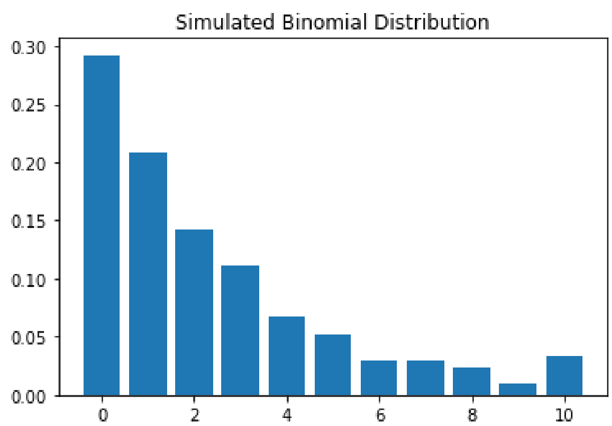
```
Indices of outliers (IQR method): [ 46 108 168 324 373 588 629 758 813 823]
```

In [34]:
```python
# Probability Calculations

import numpy as np
from scipy.stats import norm

mean = 0
std_dev = 1
lower_bound = -1
upper_bound = 1
threshold = 2

# Calculate probabilities using the cumulative distribution function (CDF)
prob_within_range = norm.cdf(upper_bound, loc=mean, scale=std_dev) - norm.cdf(lower_bound, loc=mean, scale=std
prob_above_threshold = 1 - norm.cdf(threshold, loc=mean, scale=std_dev)
prob_below_threshold = norm.cdf(threshold, loc=mean, scale=std_dev)

print(f"Probability that a value is within the range [{lower_bound}, {upper_bound}]: {prob_within_range:.4f}")
print(f"Probability that a value is above {threshold}: {prob_above_threshold:.4f}")
print(f"Probability that a value is below {threshold}: {prob_below_threshold:.4f}")
```

```
Probability that a value is within the range [-1, 1]: 0.6827
Probability that a value is above 2: 0.0228
Probability that a value is below 2: 0.9772
```

4.1.2 Simulating from Discrete Distributions

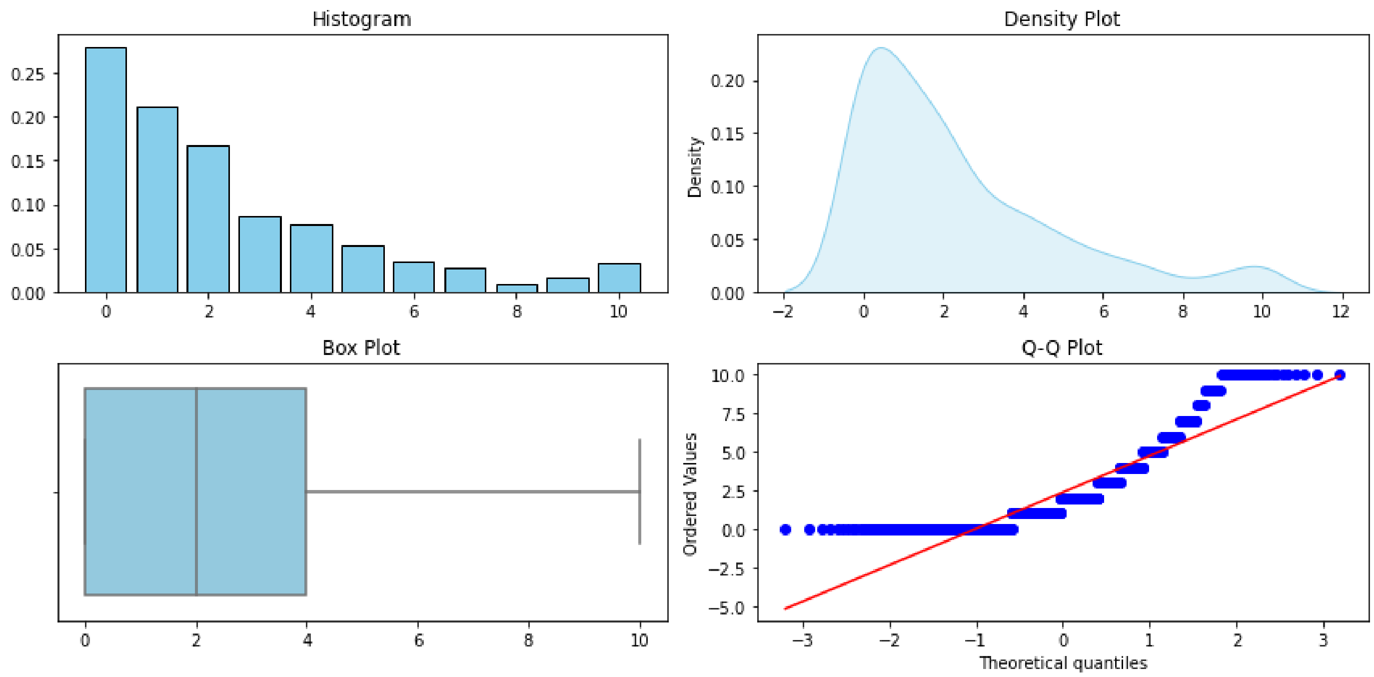
In [1]:
```python
# Simulating from discrete distributions

import numpy as np
import matplotlib.pyplot as plt

def inverse_transform_binomial(n, p, size):
    u = np.random.uniform(0, 1, size)
    x = np.floor(np.log(u) / np.log(1 - p))
    return np.minimum(x, n)

n = 10           ## Simulate data from a Binomial distribution
p = 0.3
size = 1000
simulated_data = inverse_transform_binomial(n, p, size)

plt.hist(simulated_data, bins=np.arange(0, n+2)-0.5, density=True, rwidth=0.8)
plt.title('Simulated Binomial Distribution')
plt.show()
```



In [2]:
```python
import numpy as np
from scipy.stats import moment, skew, kurtosis
from statistics import mode

def calculate_statistics(data):
    mean_value = np.mean(data)
    variance_value = np.var(data)
    std_deviation = np.std(data)
    first_quantile = np.percentile(data, 25)
    third_quantile = np.percentile(data, 75)
    mode_value = mode(data)
    skewness_value = skew(data)
    kurtosis_value = kurtosis(data)

    return {
        "Mean": mean_value,
        "Variance": variance_value,
        "Standard Deviation": std_deviation,
        "First Quantile (Q1)": first_quantile,
        "Third Quantile (Q3)": third_quantile,
        "Mode": mode_value,
        "Skewness": skewness_value,
        "Kurtosis": kurtosis_value
    }

n = 10
p = 0.3
size = 1000
simulated_data = inverse_transform_binomial(n, p, size)

# Perform statistical analysis
statistics = calculate_statistics(simulated_data)

for measure, value in statistics.items():
    print(f"{measure}: {value}")
```

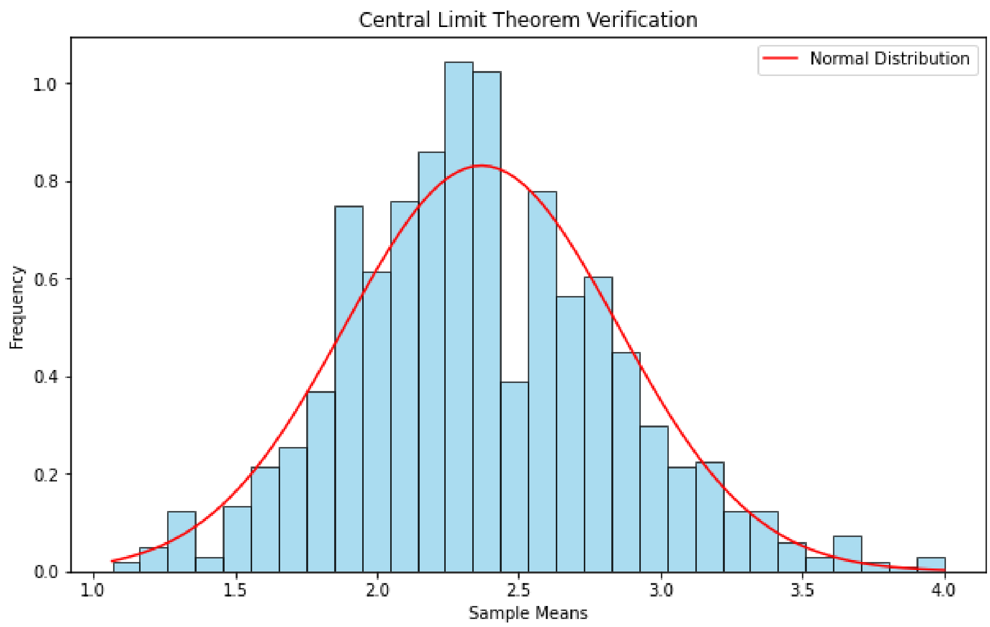
```
Mean: 2.293
Variance: 6.559151
Standard Deviation: 2.561083950205459
First Quantile (Q1): 0.0
Third Quantile (Q3): 3.0
Mode: 0.0
Skewness: 1.3618062927139298
Kurtosis: 1.318952415256243
```

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats


def visualize_distribution(data):

    plt.figure(figsize=(12, 6))             # Histogram
    plt.subplot(2, 2, 1)
    plt.hist(data, bins=np.arange(0, max(data)+2)-0.5, density=True, rwidth=0.8, color='skyblue', edgecolor='b
    plt.title('Histogram')

    plt.subplot(2, 2, 2)              # Density Plot (Kernel Density Estimate)
    sns.kdeplot(data, shade=True, color='skyblue')
    plt.title('Density Plot')

    plt.subplot(2, 2, 3)                # Box Plot
    sns.boxplot(x=data, color='skyblue')
    plt.title('Box Plot')

    plt.subplot(2, 2, 4)               # Quantile-Quantile (Q-Q) Plot
    stats.probplot(data, dist="norm", plot=plt)
    plt.title('Q-Q Plot')

    plt.tight_layout()
    plt.show()

n = 10
p = 0.3
size = 1000
simulated_data = inverse_transform_binomial(n, p, size)
visualize_distribution(simulated_data)
```

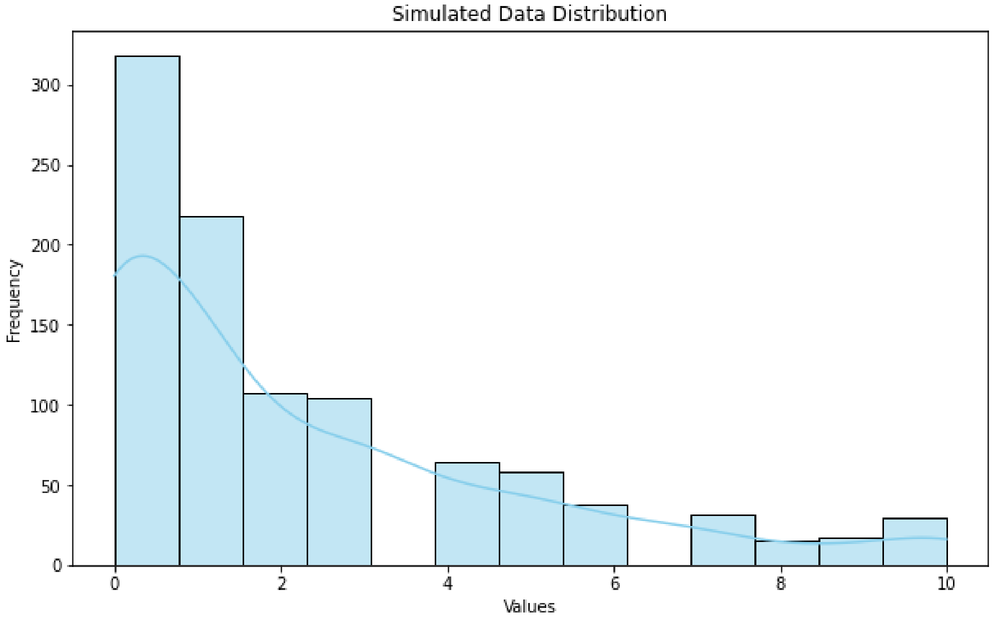


In [4]:

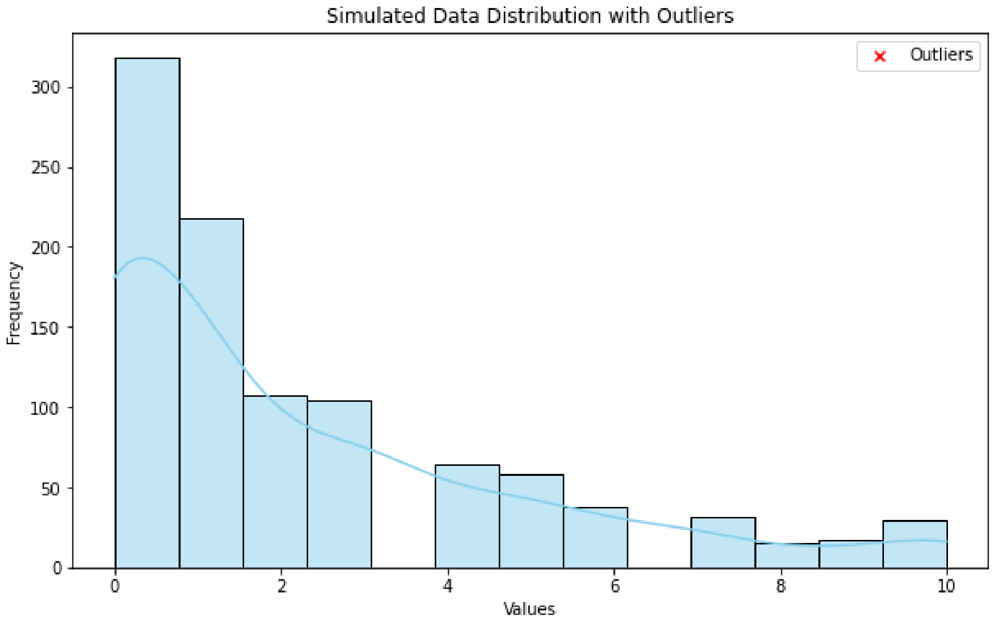
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Function to verify Central Limit Theorem
def clt_verification(data, sample_size, num_samples):
    sample_means = []

    for _ in range(num_samples):
        sample = np.random.choice(data, size=sample_size, replace=True)
        sample_mean = np.mean(sample)
        sample_means.append(sample_mean)

    plt.figure(figsize=(10, 6))                # Plot the histogram of sample means
    plt.hist(sample_means, bins=30, density=True, color='skyblue', edgecolor='black', alpha=0.7)

    mean, std_dev = np.mean(data), np.std(data)              # Plot the theoretical normal distribution
    xmin, xmax = min(sample_means), max(sample_means)
    x = np.linspace(xmin, xmax, 100)
    y = norm.pdf(x, mean, std_dev / np.sqrt(sample_size))
    plt.plot(x, y, 'r', label='Normal Distribution')

    plt.title('Central Limit Theorem Verification')
    plt.xlabel('Sample Means')
    plt.ylabel('Frequency')
    plt.legend()
    plt.show()

n = 10
p = 0.3
size = 1000
simulated_data = inverse_transform_binomial(n, p, size)

sample_size = 30
num_samples = 1000

clt_verification(simulated_data, sample_size, num_samples)
```



In [5]:
```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

n = 10
p = 0.3
size = 1000
simulated_data = inverse_transform_binomial(n, p, size)


plt.figure(figsize=(10, 6))
sns.histplot(simulated_data, kde=True, color='skyblue')
plt.title('Simulated Data Distribution')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.show()


def detect_outliers_iqr(data):

    Q1 = np.percentile(data, 25)
    Q3 = np.percentile(data, 75)
    IQR = Q3 - Q1


    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR


    outliers = (data < lower_bound) | (data > upper_bound)

    return outliers

outliers = detect_outliers_iqr(simulated_data)

print("Outliers:")
print(simulated_data[outliers])

plt.figure(figsize=(10, 6))
sns.histplot(simulated_data, kde=True, color='skyblue')
plt.scatter(simulated_data[outliers], np.zeros_like(simulated_data[outliers]), color='red', marker='x', label=
plt.title('Simulated Data Distribution with Outliers')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```




```
Outliers:
[]
```



In [6]:
```
import numpy as np
from scipy.stats import binom

n = 10
p = 0.3
size = 1000
simulated_data = inverse_transform_binomial(n, p, size)

def calculate_binomial_probabilities(data, lower_bound=None, upper_bound=None, threshold=None):
    pmf_values = binom.pmf(np.arange(0, max(data)+1), n, p)
    cdf_values = binom.cdf(np.arange(0, max(data)+1), n, p)

    if lower_bound is not None and upper_bound is not None:
        probability_range = cdf_values[upper_bound] - cdf_values[lower_bound - 1]
        return probability_range

    if threshold is not None:
        probability_above_threshold = 1 - cdf_values[threshold - 1]
        return probability_above_threshold

lower_bound = 3
upper_bound = 7
threshold = 5

probability_range = calculate_binomial_probabilities(simulated_data, lower_bound=lower_bound, upper_bound=uppe
probability_above_threshold = calculate_binomial_probabilities(simulated_data, threshold=threshold)

print(f"Probability that a randomly selected value is between {lower_bound} and {upper_bound}: {probability_ra
print(f"Probability that a randomly selected value is above {threshold}: {probability_above_threshold:.4f}")
```

```
Probability that a randomly selected value is between 3 and 7: 0.6156
Probability that a randomly selected value is above 5: 0.1503
```

4.1.3 Markov Chains

In [7]:
```
import numpy as np

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

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

        current_state = next_state

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

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

initial_state = 0
num_steps = 10000
final_state_probabilities = simulate_markov_chain(transition_matrix, initial_state, num_steps)

print("Transition Matrix:")
print(transition_matrix)
print("\nInitial State:", initial_state)
print("Number of Steps:", num_steps)
print("\nFinal State Probabilities:")
for state, probability in enumerate(final_state_probabilities):
    print(f"State {state}: {probability:.4f}")
```

```
Transition Matrix:
[[0.7 0.2 0.1]
 [0.3 0.5 0.2]
 [0.1 0.4 0.5]]

Initial State: 0
Number of Steps: 10000

Final State Probabilities:
State 0: 0.4173
State 1: 0.3563
State 2: 0.2264
```

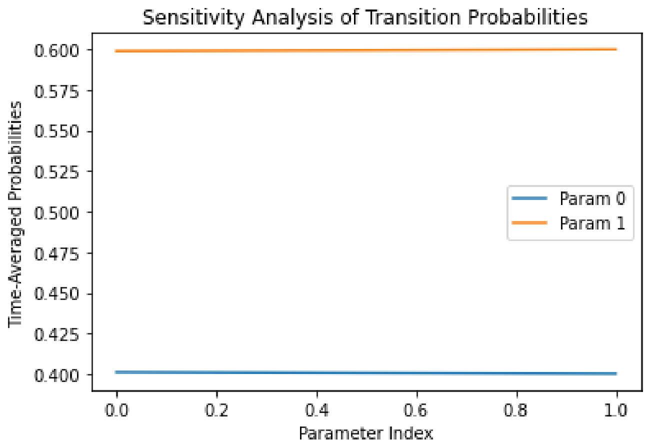
In [8]:
```python
import numpy as np

# for this simulation, I took the case of real time shopping.
class MM1QueueSimulation:
    def __init__(self, arrival_rate, service_rate, simulation_time):
        self.arrival_rate = arrival_rate
        self.service_rate = service_rate
        self.simulation_time = simulation_time
        self.state = 0

    def run_simulation(self):
        time = 0
        while time < self.simulation_time:

            arrival_time = np.random.exponential(1 / self.arrival_rate)
            departure_time = np.random.exponential(1 / self.service_rate)

            if arrival_time < departure_time:

                self.state += 1
                time += arrival_time
            else:

                if self.state > 0:
                    self.state -= 1
                time += departure_time

        average_customers = np.mean(self.state)
        return average_customers

arrival_rate = 0.5
service_rate = 1.0
simulation_time = 10000

mm1_queue_simulation = MM1QueueSimulation(arrival_rate, service_rate, simulation_time)
average_customers = mm1_queue_simulation.run_simulation()

print("Simulation Results:")
print(f"Arrival Rate: {arrival_rate}")
print(f"Service Rate: {service_rate}")
print(f"Simulation Time: {simulation_time}")
print(f"Average Number of Customers in the System: {average_customers:.2f}")
```

```
Simulation Results:
Arrival Rate: 0.5
Service Rate: 1.0
Simulation Time: 10000
Average Number of Customers in the System: 0.00
```

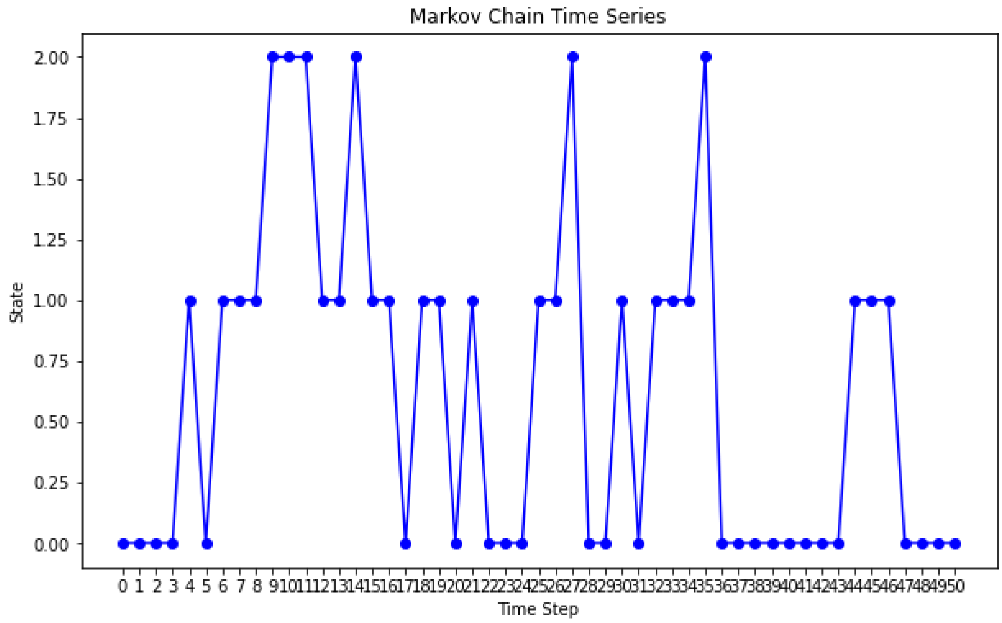
In [9]:
```python
import numpy as np

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

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

        current_state = next_state

    time_averaged_probabilities = np.array(state_counts) / num_steps
    return time_averaged_probabilities


transition_matrix = np.array([
    [0.7, 0.3],
    [0.2, 0.8]
])

initial_state = 0
num_steps = 10000

time_averaged_probabilities = simulate_markov_chain(transition_matrix, initial_state, num_steps)

print("Transition Matrix:")
print(transition_matrix)
print("\nInitial State:", initial_state)
print("Number of Steps:", num_steps)
print("\nTime-Averaged Probabilities:")
for state, probability in enumerate(time_averaged_probabilities):
    print(f"State {state}: {probability:.4f}")

# Steady-state probabilities (eigenvector of the transition matrix)
_, eigenvectors = np.linalg.eig(transition_matrix.T)
steady_state_probabilities = np.abs(eigenvectors[:, 0] / np.sum(np.abs(eigenvectors[:, 0])))
print("\nSteady-State Probabilities:")
for state, probability in enumerate(steady_state_probabilities):
    print(f"State {state}: {probability:.4f}")
```

```
Transition Matrix:
[[0.7 0.3]
 [0.2 0.8]]

Initial State: 0
Number of Steps: 10000

Time-Averaged Probabilities:
State 0: 0.3996
State 1: 0.6004

Steady-State Probabilities:
State 0: 0.5000
State 1: 0.5000
```

In [10]:
```python
import numpy as np
import matplotlib.pyplot as plt

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

    for step in range(num_steps):

        state_counts[current_state] += 1
        next_state = np.random.choice(len(transition_matrix), p=transition_matrix[current_state])

        current_state = next_state

    time_averaged_probabilities = np.array(state_counts) / num_steps
    return time_averaged_probabilities

def sensitivity_analysis(transition_matrix_base, initial_state, num_steps, perturbation_factor):
    num_parameters = len(transition_matrix_base)
    num_simulations = 100

    sensitivity_results = []

    for param_index in range(num_parameters):

        transition_matrix_perturbed = np.copy(transition_matrix_base)
        transition_matrix_perturbed[param_index] *= (1 + perturbation_factor)

        transition_matrix_perturbed /= transition_matrix_perturbed.sum(axis=1, keepdims=True)

        average_probabilities = np.zeros(num_parameters)
        for _ in range(num_simulations):
            time_averaged_probabilities = simulate_markov_chain(transition_matrix_perturbed, initial_state, nu
            average_probabilities += time_averaged_probabilities / num_simulations

        sensitivity_results.append(average_probabilities)

    return np.array(sensitivity_results)

transition_matrix_base = np.array([
    [0.7, 0.3],
    [0.2, 0.8]
])

initial_state = 0
num_steps = 10000

perturbation_factor = 0.1

sensitivity_results = sensitivity_analysis(transition_matrix_base, initial_state, num_steps, perturbation_fact

labels = [f"Param {i}" for i in range(len(transition_matrix_base))]
for param_index in range(len(transition_matrix_base)):
    plt.plot(sensitivity_results[:, param_index], label=f"Param {param_index}")

plt.xlabel("Parameter Index")
plt.ylabel("Time-Averaged Probabilities")
plt.legend()
plt.title("Sensitivity Analysis of Transition Probabilities")
plt.show()
```



In [11]:
```python
import numpy as np
import matplotlib.pyplot as plt

def simulate_markov_chain(transition_matrix, initial_state, num_steps):
    current_state = initial_state
    state_sequence = [current_state]

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

    return state_sequence


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

initial_state = 0
num_steps = 50

state_sequence = simulate_markov_chain(transition_matrix, initial_state, num_steps)

plt.figure(figsize=(10, 6))
plt.plot(range(num_steps + 1), state_sequence, marker='o', linestyle='-', color='b')
plt.title('Markov Chain Time Series')
plt.xlabel('Time Step')
plt.ylabel('State')
plt.xticks(range(num_steps + 1))
plt.show()
```



In [12]:

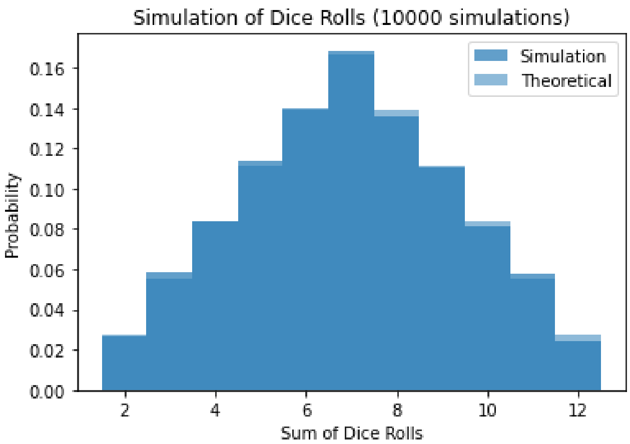
```
import numpy as np

# To apply variance reduction concept, I used a case study of portfolio management.

def simulate_portfolio_returns(num_simulations, asset_returns, correlations, weights):
    num_assets = len(weights)

    # Cholesky decomposition
    correlated_returns = np.random.multivariate_normal(mean=np.zeros(num_assets),
                                                       cov=np.dot(correlations, correlations.T),
                                                       size=num_simulations)

    portfolio_returns = np.dot(correlated_returns, np.array(weights))

    return portfolio_returns
num_simulations = 10000

asset_returns = np.array([0.02, 0.01, -0.01])
correlations = np.array([[1.0, 0.5, 0.3], [0.5, 1.0, 0.2], [0.3, 0.2, 1.0]])
weights = [0.4, 0.4, 0.2]

simulated_returns_no_variance_reduction = simulate_portfolio_returns(num_simulations, asset_returns, correlati

var_no_reduction = np.percentile(simulated_returns_no_variance_reduction, 5)
print(f"VaR without variance reduction: {var_no_reduction:.4f}")
```

```
VaR without variance reduction: -1.6399
```

In [13]:
```python
# BASIC MONTE CARLO SIMULATION

import numpy as np

mu = 5
sigma = 2
num_samples = 10000

samples = np.random.normal(mu, sigma, num_samples)

mean_estimate_basic = np.mean(samples)

print(f"Mean Estimate (Basic Monte Carlo): {mean_estimate_basic:.4f}")

# IMPORTANCE SAMPLING

mu_important = 8
sigma_important = 1

samples_importance = np.random.normal(mu_important, sigma_important, num_samples)

weights = np.exp(-(samples_importance - mu_important)**2 / (2 * sigma_important**2)) / \
          np.exp(-(samples - mu)**2 / (2 * sigma**2))

mean_estimate_importance = np.sum(samples_importance * weights) / np.sum(weights)

print(f"Mean Estimate (Importance Sampling): {mean_estimate_importance:.4f}")

# CONTROL VARIATES

correlated_variable = np.random.normal(mu + 0.5 * sigma, sigma, num_samples)
covariance = np.cov(samples, correlated_variable)[0, 1]

control_variate = correlated_variable

optimal_coefficient = -covariance / sigma**2

samples_control_variates = samples + optimal_coefficient * (control_variate - mu)

mean_estimate_control_variates = np.mean(samples_control_variates)

print(f"Mean Estimate (Control Variates): {mean_estimate_control_variates:.4f}")

# ANTITHETIC VARIATES

antithetic_samples = mu + sigma * np.random.normal(size=num_samples)
antithetic_samples = 2 * mu - antithetic_samples  # Reflect samples around the mean

combined_samples = 0.5 * (samples + antithetic_samples)

mean_estimate_antithetic = np.mean(combined_samples)

print(f"Mean Estimate (Antithetic Variates): {mean_estimate_antithetic:.4f}")

# RESULT COMPARISION

print("\nResults Comparison:")
print(f"True Mean: {mu}")
print(f"Basic Monte Carlo: {mean_estimate_basic:.4f}")
print(f"Importance Sampling: {mean_estimate_importance:.4f}")
print(f"Control Variates: {mean_estimate_control_variates:.4f}")
print(f"Antithetic Variates: {mean_estimate_antithetic:.4f}")
```

```
Mean Estimate (Basic Monte Carlo): 5.0111
Mean Estimate (Importance Sampling): 8.0694
Mean Estimate (Control Variates): 5.0294
Mean Estimate (Antithetic Variates): 5.0053

Results Comparison:
True Mean: 5
Basic Monte Carlo: 5.0111
Importance Sampling: 8.0694
Control Variates: 5.0294
Antithetic Variates: 5.0053
```

In [14]:
```python
import numpy as np
import matplotlib.pyplot as plt

# For this case , I have taken an experiment of dice rolls.

def simulate_dice_rolls(num_simulations):

    dice_rolls = np.random.randint(1, 7, size=(num_simulations, 2))
    sum_of_rolls = np.sum(dice_rolls, axis=1)

    return sum_of_rolls

def plot_simulation_results(simulation_results, num_simulations):

    plt.hist(simulation_results, bins=np.arange(1.5, 13.5, 1), density=True, alpha=0.7, label='Simulation')

    theoretical_probs = np.array([1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1]) / 36
    plt.bar(range(2, 13), theoretical_probs, width=1, alpha=0.5, label='Theoretical')

    plt.xlabel('Sum of Dice Rolls')
    plt.ylabel('Probability')
    plt.title(f'Simulation of Dice Rolls ({num_simulations} simulations)')
    plt.legend()
    plt.show()

num_simulations = 10000
simulation_results = simulate_dice_rolls(num_simulations)
plot_simulation_results(simulation_results, num_simulations)
```



4.2 REAL DATA ANALYSIS

4.2.1 Bayes' theorem

In [15]:
```python
# PROBLEM STATEMENT : Apply Bayes's theorem, joint probability concepts and factor analysis to determine to hoı
#                     personal details of customers(For eg. age, gender, education level, etc) help banks and j
#                     organizations to assess the credit scores of the customers.
#
# APPROACH          : Consider various attributes of the dataset to calculate conditional and joint probabilit
#                     different parameters individually and combined, provide valueable insights. Perform facto
#                     identify underlying factors or latent variables  that explain patterns of correlations ar
#                     variables.
```

In [1]:
```python
# Step 1 - Display the dataset.

import pandas as pd

data = pd.read_csv(r'C:\Users\lenovo\Downloads\Credit Score Classification Dataset.csv')
data
```

Out[1]:

	Age	Gender	Income	Education	Marital Status	Number of Children	Home Ownership	Credit Score
0	25	Female	50000	Bachelor's Degree	Single	0	Rented	High
1	30	Male	100000	Master's Degree	Married	2	Owned	High
2	35	Female	75000	Doctorate	Married	1	Owned	High
3	40	Male	125000	High School Diploma	Single	0	Owned	High
4	45	Female	100000	Bachelor's Degree	Married	3	Owned	High
...	...	...	...	...	...	...	...	...
159	29	Female	27500	High School Diploma	Single	0	Rented	Low
160	34	Male	47500	Associate's Degree	Single	0	Rented	Average
161	39	Female	62500	Bachelor's Degree	Married	2	Owned	High
162	44	Male	87500	Master's Degree	Single	0	Owned	High
163	49	Female	77500	Doctorate	Married	1	Owned	High

164 rows × 8 columns

In [2]:
```python
# Step 2 - Calculating probabilities of credit scores being low, average and low.

# The function .shape() is used to fetch the dimensions of numpy type objects in python.

low_credit_prob = data[data['Credit Score'] == 'Low'].shape[0] / data.shape[0]
Average_credit_prob = data[data['Credit Score'] == 'Average'].shape[0] / data.shape[0]
High_credit_prob = data[data['Credit Score'] == 'High'].shape[0] / data.shape[0]

print(f"Probability of Low Credit Score: {low_credit_prob:.2%}")
print(f"Probability of Average Credit Score: {Average_credit_prob:.2%}")
print(f"Probability of High Credit Score: {High_credit_prob:.2%}")
```

```
Probability of Low Credit Score: 9.15%
Probability of Average Credit Score: 21.95%
Probability of High Credit Score: 68.90%
```

In [3]:
```python
# Step 3 (a) - Bayes theorem application

female_given_low_credit = data[data['Credit Score'] == 'Low']['Gender'].value_counts(normalize=True)['Female']
female_prob = data['Gender'].value_counts(normalize=True)['Female']

posterior_low_credit_prob = (female_given_low_credit * low_credit_prob) / female_prob

print(f"Prior Probability of Low Credit Score: {low_credit_prob:.2%}")
print(f"Posterior Probability of Low Credit Score given Female: {posterior_low_credit_prob:.2%}")
```

```
Prior Probability of Low Credit Score: 9.15%
Posterior Probability of Low Credit Score given Female: 17.44%
```

In [4]:
```
# Step 3 (b) - Bayes theorem application

single_given_low_credit = data[data['Credit Score'] == 'Low']['Marital Status'].value_counts(normalize=True).g
married_given_low_credit = data[data['Credit Score'] == 'Low']['Marital Status'].value_counts(normalize=True).

single_prob = data['Marital Status'].value_counts(normalize=True).get('Single', 0)
married_prob = data['Marital Status'].value_counts(normalize=True).get('Married', 0)

# "Single" case with "low" credit score

if single_prob != 0:
    posterior_low_credit_prob_single = (single_given_low_credit * low_credit_prob) / single_prob

    print(f"Prior Probability of Low Credit Score: {low_credit_prob:.2%}")
    print(f"Posterior Probability of Low Credit Score given Single: {posterior_low_credit_prob_single:.2%}")

else:
    print("No singles in the dataset, cannot compute posterior probability.")

# "Marreied" case with "low" credit score

if married_prob != 0:
    posterior_low_credit_prob = (married_given_low_credit * low_credit_prob) / married_prob

    print(f"Posterior Probability of Low Credit Score given Married: {posterior_low_credit_prob:.2%}")
else:
    print("No married individuals in the dataset, cannot compute posterior probability.")
```

```
Prior Probability of Low Credit Score: 9.15%
Posterior Probability of Low Credit Score given Single: 19.48%
Posterior Probability of Low Credit Score given Married: 0.00%
```

In [5]:
```
# Step 3 (c) - Bayes theorem application

single_given_avg_credit = data[data['Credit Score'] == 'Average']['Marital Status'].value_counts(normalize=Tru
married_given_average_credit = data[data['Credit Score'] == 'Average']['Marital Status'].value_counts(normaliz

single_prob = data['Marital Status'].value_counts(normalize=True).get('Single', 0)
married_prob = data['Marital Status'].value_counts(normalize=True).get('Married', 0)

# "Single" case with "average" credit score

if single_prob != 0:
    posterior_avg_credit_prob = (single_given_avg_credit * Average_credit_prob) / single_prob
    print(f"Prior Probability of Average Credit Score: {Average_credit_prob:.2%}")
    print(f"Posterior Probability of Average Credit Score given Single: {posterior_avg_credit_prob:.2%}")
else:
    print("No singles in the dataset, cannot compute posterior probability.")

# "Married" case with "average" credit score

if married_prob != 0:
    posterior_average_credit_prob = (married_given_average_credit * Average_credit_prob) / married_prob
    print(f"Posterior Probability of Average Credit Score given Married: {posterior_average_credit_prob:.2%}")
else:
    print("No married individuals in the dataset, cannot compute posterior probability.")
```

```
Prior Probability of Average Credit Score: 21.95%
Posterior Probability of Average Credit Score given Single: 44.16%
Posterior Probability of Average Credit Score given Married: 2.30%
```

In [6]:
```
# Step 3 (d) - Bayes theorem application

owned_home_given_high_credit = data[data['Credit Score'] == 'High']['Home Ownership'].value_counts(normalize=T
rented_given_high_credit = data[data['Credit Score'] == 'High']['Home Ownership'].value_counts(normalize=True)

owned_home_prob = data['Home Ownership'].value_counts(normalize=True).get("Owned", 0)
rented_prob = data['Home Ownership'].value_counts(normalize=True).get("Rented", 0)

# "owned home" case

if owned_home_prob != 0:
    posterior_high_credit_prob = (owned_home_given_high_credit * High_credit_prob) / owned_home_prob

    print(f"Prior Probability of High Credit Score: {High_credit_prob:.2%}")
    print(f"Posterior Probability of High Credit Score given Owned Home: {posterior_high_credit_prob:.2%}")
else:
    print("No individuals with Owned Home in the dataset, cannot compute posterior probability.")
```

```
Prior Probability of High Credit Score: 68.90%
Posterior Probability of High Credit Score given Owned Home: 98.20%
```

In [7]:
```
# Step 3 (e) - Bayes theorem application

data_case = data[data['Age'] <= 30]
age_below_30_prob = data_case.shape[0] / data.shape[0]
age_below_30_given_high_credit = data_case[data_case['Credit Score'] == 'High'].shape[0] / data[data['Credit S

# "Age less than 30" case

if age_below_30_prob != 0:
    posterior_high_credit_prob = (age_below_30_given_high_credit * High_credit_prob) / age_below_30_prob
    print(f"Prior Probability of High Credit Score: {High_credit_prob:.2%}")
    print(f"Posterior Probability of High Credit Score given Age below or equal to 30 : {posterior_high_credit_
else:
    print("No individuals with age below or equal to 30 in the dataset, cannot compute posterior probability."
```

```
Prior Probability of High Credit Score: 68.90%
Posterior Probability of High Credit Score given Age below or equal to 30 : 24.39%
```

4.2.2 Joint Distribution Analysis

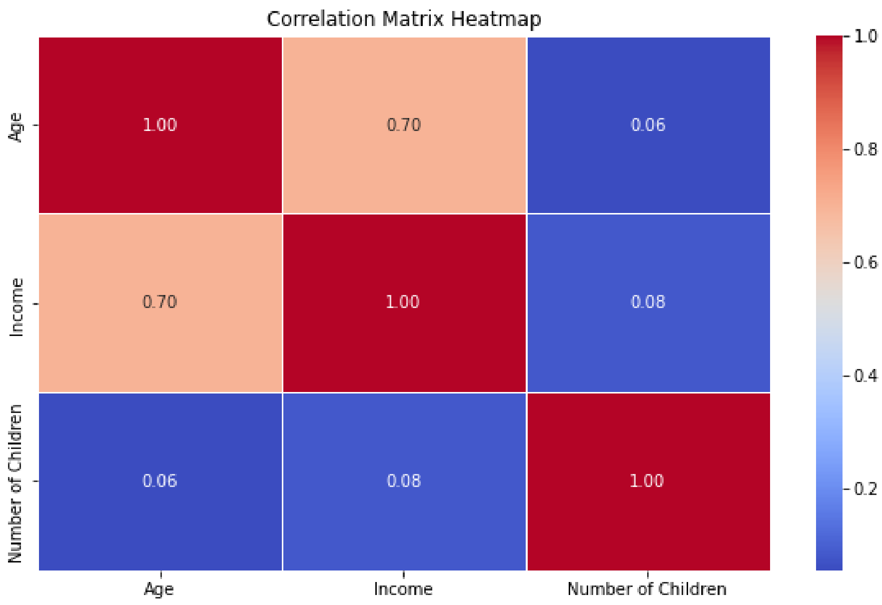
In [8]:
```
# Step 4 (a) - Joint probability

female_high_income_count = data[(data['Gender'] == 'Female') & (data['Income'] == 50000)].shape[0]
total_individuals = data.shape[0]

joint_probability_female_high_income = female_high_income_count / total_individuals

print("Joint Probability of being Female and having High Income:", joint_probability_female_high_income)
```

```
Joint Probability of being Female and having High Income: 0.012195121951219513
```

In [9]:
```
# Step 4 (b) - Joint probability

joint_probability_count = data[(data['Education'] == "High School Diploma") & (data['Credit Score'] == 'High')
total_individuals = data.shape[0]
joint_probability = joint_probability_count / total_individuals

print("Joint Probability of High School Diploma and High Credit Score:", joint_probability)
```

```
Joint Probability of High School Diploma and High Credit Score: 0.09146341463414634
```

```
In [10]: # Step 5 - Visualise the correlation between random variables.

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

correlation_matrix = data.corr()
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title("Correlation Matrix Heatmap")
plt.show()
```

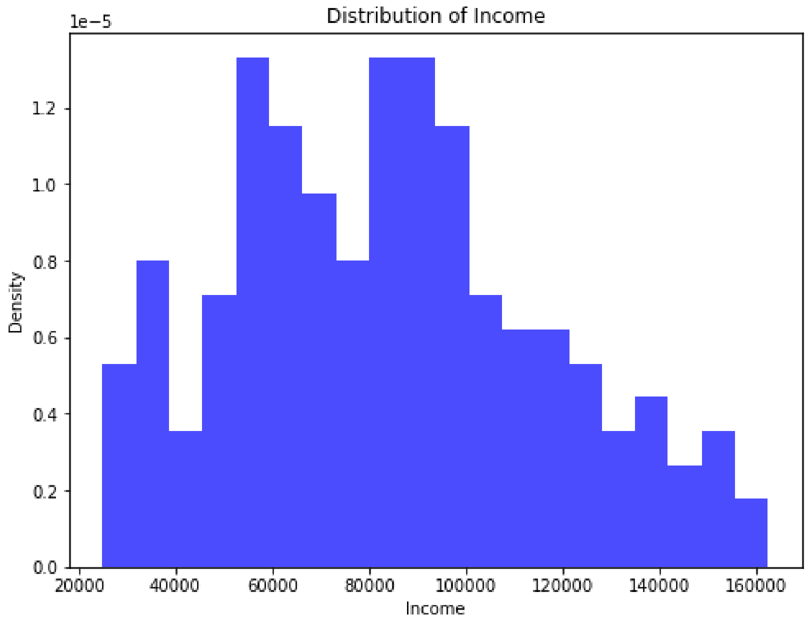



```
In [11]: import pandas as pd
from factor_analyzer.factor_analyzer import calculate_kmo


selected_columns = ['Age', 'Income', 'Number of Children']
selected_data = data[selected_columns]
kmo_all, kmo_model = calculate_kmo(selected_data)
print("KMO overall:", kmo_all)
print("KMO per variable:")
print(kmo_model)
```

```
KMO overall: [0.50234586 0.50232016 0.71123317]
KMO per variable:
0.5038464515370857
```

In [12]:
```python
import pandas as pd
from scipy.stats import shapiro, anderson
import matplotlib.pyplot as plt

variable_to_test = 'Income'
sample_data = data[variable_to_test]

shapiro_stat, shapiro_p_value = shapiro(sample_data)
print(f'Shapiro-Wilk Test - Statistic: {shapiro_stat}, p-value: {shapiro_p_value}')

anderson_stat, anderson_critical_values, anderson_significance_levels = anderson(sample_data)
print(f'Anderson-Darling Test - Statistic: {anderson_stat}')

is_normal = anderson_stat < anderson_critical_values[2]
print(f'The data is likely {"normal" if is_normal else "not normal"}.')

plt.figure(figsize=(8, 6))
plt.hist(sample_data, bins=20, density=True, alpha=0.7, color='blue')
plt.title(f'Distribution of {variable_to_test}')
plt.xlabel(variable_to_test)
plt.ylabel('Density')
plt.show()
```

```
Shapiro-Wilk Test - Statistic: 0.9798086285591125, p-value: 0.017001418396830 56
Anderson-Darling Test - Statistic: 0.6614447936732688
The data is likely normal.
```



4.2.3 Factor Analysis

In [13]:
```python
# Step 3______FACTOR ANALYSIS

import pandas as pd
from factor_analyzer import FactorAnalyzer


selected_columns = ['Age', 'Income', 'Number of Children']
selected_data = data[selected_columns]
fa = FactorAnalyzer(n_factors=2, rotation='varimax')
fa.fit(selected_data)

print("Factor Loadings:")
print(pd.DataFrame(fa.loadings_, index=selected_data.columns))

print("\nVariance Explained:")
print(fa.get_factor_variance())
```

```
Factor Loadings:
                           0         1
Age                 0.834374  0.115120
Income              0.798784  0.286478
Number of Children  0.041714  0.178813

Variance Explained:
(array([1.33597535, 0.1272962 ]), array([0.44532512, 0.04243207]), array([0.44532512, 0.48775719]))
```

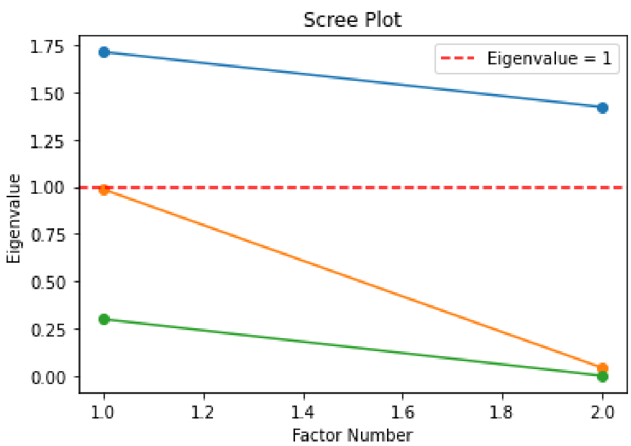
Above I have calculated Factor loadings and variance explained for my dataset.
As we can see above, for this dataset, 3 variables (Age, Income and No. of children) and 2 factors (0 and 1) have been found out. All variables have positive relationship with each factor.

Interpretation of Factor loadings :
The factor loadings of Age and Income on factor 0 are 0.83 and 0.79 respectively which indicates stronger relationship of variables Age and income with Factor 0.
The factor loadings of Number of children on factor 0 and factor 1 are 0.04 and 0.17 which indicate poor and moderate relationship of variable Number of children with factor 0 and factor 1 respectively.

Interpretation of variance explained :
In this case,
Eigenvalues : [1.33597535, 0.1272962]
Proportion of variance : [0.44532512, 0.04243207]
Cumulative proportion of variance : [0.44532512, 0.48775719]

Eigenvalues represent the amount of variance explained by each factor. In this case, factor 0 explains more variance than
factor 1.
Factor 0 is responsible for 44.5 % of total variance and factor 1 accounts for 4.2 % .
Both factors (factor 0 and factor 1) together explain 48.78 % of the total variance.

In [14]:
```
import numpy as np
import matplotlib.pyplot as plt

# Eigenvalues from factor analysis
eigenvalues = fa.get_eigenvalues()

# Plot scree plot
plt.plot(np.arange(1, len(eigenvalues) + 1), eigenvalues, marker='o')
plt.title('Scree Plot')
plt.xlabel('Factor Number')
plt.ylabel('Eigenvalue')
plt.axhline(1, color='red', linestyle='--', label='Eigenvalue = 1')
plt.legend()
plt.show()
```



Key takeaways from Simulations of continous, discrete RVs and Markov Chains:

1. Mean, mode, Median, Kurtosis, Skewness etc are some of basic things an analyst should calculate and learn about the type of
   data.
2. Plots like Histogram, Violin plot and Box plots are useful for statistical analysis.
3. Impact of outliers and their detection using 2 methods (Z-score method and IQR method).
4. Importance of normal distribution, central limit theorem and relation of the two.
5. Uses and applications of transition matrices, steady state and transition probabilities,sensitivity analysis.
6. Uses and applications of variance reduction techniques.

Key takeawys from real data analysis:

1. The cases of 'low credit scores' in females are comparitively more than men.
2. Married people do have a higher credit score than the unmarried people. Around 44% of 'single' population has average
   credit score.
3. Credit score of people who own a house is high.
4. If the age of person is below 30, the chances of him/her having high credit score is only 25%.
5. The joint probability that you have high credit score with high school degree is only 0.1%.
6. From corelation matrix heatmap, the combination of factors (age and income) plays a significant income.

7. Shapiro wilk test provides information about data being normal or not.

Conclusion:

For the first 3 parts (simulation of continous rv, discrete rv and Markov chains):

By generating random values that follow specified probability distributions, I gained a deeper understanding of the possible outcomes and the likelihood of different events. This approach has proven particularly effective in scenarios where analytical solutions were difficult to understand.

Simulation of continous and discrete random variables give a clarity of concepts and better understanding and difference between
the two. These simulations also give ideas about the values that continous and discrete random variables can take. Statistical
analysis gives a clear picture of data, providing all the basic attributes (mean,median,mode,etc.) of big data.

Markov Chains helped in capturing sequential dependencies and state transitions within dynamic systems.This technique has proven especially useful in predicting future states and understanding the long-term behavior of systems with inherent memory.

For the last part (real data analysis):

The findings focus on the factors influencing credit scores offer practical implications for decision-makers and consumers alike. Credit scores are impacted a lot by factors like income, no. of children, house ownership and marital status. From this dataset it can be concluded that credit scores are a lot about financial stability. For example, owning a house, having a high income, having less number of children and being aged above 30.

Education plays a huge role in good credit score, as good education lands you a good job and good pay. Married people are the ones with more responsibiliy of providing for the family and hence thy have high credit score. Combination of factors such as
age and income plays a significant role in credit score analysis. The balence of both (good career growth over the years) ensures high career growth.

This project marks a significant step towards demystifying the dynamics of credit scores. The findings call for a paradigm shift in how creditworthiness is perceived and evaluated, advocating for a more holistic and individualized approach.