**Module 5 - Probability**

**Basics of Probability**

1. Write a Python program to simulate the following scenarios:

  a. Tossing a coin 10,000 times and calculating the experimental probability of heads and tails.

  b. Rolling two dice and computing the probability of getting a sum of 7.

**Steps**

      a. Use Python's random module for simulations.

      b. Implement loops for repeated trials.

      c. Track outcomes and compute probabilities.

Answer:

1. Number of trials for the dice roll simulation NUM\_ROLLS = 10000
2. Initialize a counter for the sum of 7 count\_sum\_7 = 0

1. Simulate the dice rolls

for \_ in range(NUM\_ROLLS):

2. Roll the two dice

die1 = random.randint(1, 6)

die2 = random.randint(1, 6)

3.Check if the sum is 7

if die1 + die2 == 7:

count\_sum\_7 += 1

4.Calculate the experimental probability

prob\_sum\_7 = count\_sum\_7 / NUM\_ROLLS

print(f"--- Two Dice Roll Simulation ({NUM\_ROLLS} rolls) ---")

print(f"Count of sum being 7: {count\_sum\_7}")

print(f"Experimental probability of getting a sum of 7: {prob\_sum\_7:.4f}")

2. Write a function to estimate the probability of getting at least one "6" in 10 rolls of a fair die.

**Steps**

      a. Simulate rolling a die 10 times using a loop.

      b. Track trials where at least one "6" occurs.

      c. Calculate the proportion of successful trials.

Answer:

import random def estimate\_probability\_at\_least\_one\_six(num\_simulations):

Estimates the probability of getting at least one "6" in 10 rolls of a fair die.

Args:

num\_simulations (int): The number of times to run the simulation.

Returns: float: The estimated probability.

success\_count = 0

num\_rolls = 10

for \_ in range(num\_simulations):

# Simulate 10 rolls of a fair die

rolls = [random.randint(1, 6) for \_ in range(num\_rolls)]

# Check if at least one '6' is in the list of rolls

if 6 in rolls:

success\_count += 1

# Calculate the estimated probability

estimated\_probability = success\_count / num\_simulations

return estimated\_probability

# Run the simulation with a large number of trials

num\_trials = 100000

probability = estimate\_probability\_at\_least\_one\_six(num\_trials)

print(f"Number of simulations: {num\_trials}")

print(f"Estimated probability of at least one '6' in 10 rolls: {probability:.4f}")

**Conditional Probability and Bayes' Theorem**

3. A bag contains 5 red, 7 green, and 8 blue balls. A ball is drawn randomly, its color noted, and it is put back into the bag. If this process is repeated 1000 times, write a Python program to estimate:

  a. The probability of drawing a red ball given that the previous ball was blue.

  b. Verify Bayes' theorem with the simulation results.

**Steps**

    a. Use random sampling to simulate the process.

    b. Compute conditional probabilities directly from the data.

Answer:

1. Define the contents of the bag

bag = ['red'] \* 5 + ['green'] \* 7 + ['blue'] \* 8

total\_balls = len(bag)

1. Simulation parameters

num\_trials = 1000

1. Perform the simulation

draws = random.choices(bag, k=num\_trials)

count\_blue = 0

count\_blue\_and\_red = 0

1.Iterate from the second draw to compare with the previous draw for

i in range(1, len(draws)):

if draws[i-1] == 'blue':

count\_blue += 1 if draws[i] == 'red':

count\_blue\_and\_red += 1

2. Calculate the direct conditional probability

prob\_red\_given\_blue\_direct = count\_blue\_and\_red / count\_blue if count\_blue > 0 else 0

print(f"Number of times a blue ball was drawn: {count\_blue}")

print(f"Number of times a red ball was drawn after a blue ball: {count\_blue\_and\_red}")

print(f"Estimated P(Red | Previous was Blue) directly: {prob\_red\_given\_blue\_direct:.4f}")

Verify Bayes' Theorem ---

Bayes' Theorem: P(A|B) = [P(B|A) \* P(A)] / P(B)

In our case: P(Red | Blue) = [P(Blue | Red) \* P(Red)] / P(Blue)

3.Estimate P(Red) and P(Blue) from the simulation

count\_red = draws.count('red')

count\_blue\_total = draws.count('blue')

prob\_red = count\_red / num\_trials

prob\_blue = count\_blue\_total / num\_trials

4.Estimate P(Blue | Red) from the simulation

count\_red\_and\_blue = 0

for i in range(1, len(draws)):

if draws[i-1] == 'red':

if draws[i] == 'blue':

count\_red\_and\_blue += 1

count\_red\_prev = count\_red # Same as draws.count('red') but just for clarity

prob\_blue\_given\_red = count\_red\_and\_blue / count\_red\_prev if count\_red\_prev > 0 else 0

print("-" \* 50)

print("Verifying Bayes' Theorem:")

print(f"Estimated P(Red) from simulation: {prob\_red:.4f}")

print(f"Estimated P(Blue) from simulation: {prob\_blue:.4f}")

print(f"Estimated P(Blue | Previous was Red) directly: {prob\_blue\_given\_red:.4f}")

5.Calculate P(Red | Blue) using Bayes' Theorem

prob\_red\_given\_blue\_bayes = (prob\_blue\_given\_red \* prob\_red) / prob\_blue if prob\_blue > 0 else 0

print(f"Calculated P(Red | Blue) using Bayes' theorem: {prob\_red\_given\_blue\_bayes:.4f}")

6. Compare results with theoretical probabilities ---

Since the balls are replaced, the draws are independent events.

Theoretical P(Red) = 5/20 = 0.25

Theoretical P(Blue) = 8/20 = 0.4

Theoretical P(Red | Blue) = P(Red) = 0.25

Theoretical P(Blue | Red) = P(Blue) = 0.4

print("-" \* 50)

print("Comparison with Theoretical Probabilities:")

print(f"Theoretical P(Red | Blue) (Independent events): {5/20:.4f}")

7. Compare the two simulation-based calculations difference = abs(prob\_red\_given\_blue\_direct - prob\_red\_given\_blue\_bayes)

print(f"Difference between direct and Bayes' theorem calculations: {difference:.4f}")

8.Check if the Bayes' theorem calculation is close to the direct calculation

print("The simulation results for the conditional probability are consistent with Bayes' theorem" if difference < 0.05 else "The simulation results are not consistent with Bayes' theorem.")

Code output

Number of times a blue ball was drawn: 404

Number of times a red ball was drawn after a blue ball: 102

Estimated P(Red | Previous was Blue) directly: 0.2525

--------------------------------------------------

Verifying Bayes' Theorem:

Estimated P(Red) from simulation: 0.2410

Estimated P(Blue) from simulation: 0.4040

Estimated P(Blue | Previous was Red) directly: 0.3776

Calculated P(Red | Blue) using Bayes' theorem: 0.2252

--------------------------------------------------

Comparison with Theoretical Probabilities:

Theoretical P(Red | Blue) (Independent events): 0.2500

Difference between direct and Bayes' theorem calculations: 0.0272

The simulation results for the conditional probability are consistent with Bayes' theorem

**Random Variables and Discrete Probability**

4. Generate a sample of size 1000 from a discrete random variable with the following distribution:

  - P(X=1) = 0.25

  - P(X=2) = 0.35

  - P(X=3) = 0.4

  Compute the empirical mean, variance, and standard deviation of the sample.

  Steps

      a. Use numpy.random.choice() to generate the sample.

      b. Use numpy methods to calculate mean, variance, and standard deviation.

Answer:

1.Define the values and their corresponding probabilities for the discrete random variable values = [1, 2, 3]

probabilities = [0.25, 0.35, 0.4]

sample\_size = 1000

2. Generate a sample of size 1000 from the discrete random variable

sample = np.random.choice(values, size=sample\_size, p=probabilities)

3.Compute the empirical statistics of the sample

empirical\_mean = np.mean(sample)

empirical\_variance = np.var(sample)

empirical\_std\_dev = np.std(sample)

# Print the results:

print("Generated sample (first 10 elements):", sample[:10])

print("\n--- Empirical Statistics ---")

print(f"Empirical Mean: {empirical\_mean:.4f}")

print(f"Empirical Variance: {empirical\_variance:.4f}")

print(f"Empirical Standard Deviation: {empirical\_std\_dev:.4f}")

4.Optional: Calculate and print theoretical statistics for comparison ---

Theoretical Mean (E[X]) = (1 \* 0.25) + (2 \* 0.35) + (3 \* 0.4)

theoretical\_mean = (1 \* 0.25) + (2 \* 0.35) + (3 \* 0.4)

5Theoretical Variance (Var(X)) = E[X^2] - (E[X])^2

e\_x\_squared = (1\*\*2 \* 0.25) + (2\*\*2 \* 0.35) + (3\*\*2 \* 0.4)

theoretical\_variance = e\_x\_squared - (theoretical\_mean\*\*2)

theoretical\_std\_dev = np.sqrt(theoretical\_variance)

print("\n--- Theoretical Statistics for Comparison ---")

print(f"Theoretical Mean: {theoretical\_mean:.4f}")

print(f"Theoretical Variance: {theoretical\_variance:.4f}")

print(f"Theoretical Standard Deviation: {theoretical\_std\_dev:.4f}")

Code output

Generated sample (first 10 elements): [1 3 3 2 3 1 3 3 2 1]

--- Empirical Statistics ---

Empirical Mean: 2.1770

Empirical Variance: 0.6337

Empirical Standard Deviation: 0.7960

--- Theoretical Statistics for Comparison ---

Theoretical Mean: 2.1500

Theoretical Variance: 0.6275

Theoretical Standard Deviation: 0.7921

**Continuous Random Variables**

5. Simulate 2000 random samples from an exponential distribution with a mean of 5. Visualize the distribution using:

  a. A histogram.

  b. A probability density function (PDF) overlay.

  Steps

      a. Use numpy.random.exponential().

      b. Use matplotlib to create visualizations.

Answer:

import numpy as np

import matplotlib.pyplot as plt

# Simulation parameters

mean = 5

num\_samples = 2000

# 1. Simulate the data from an exponential distribution

# The scale parameter for numpy.random.exponential is the mean.

samples = np.random.exponential(scale=mean, size=num\_samples)

# 2. Visualize the distribution

# Create a figure and an axes object

fig, ax = plt.subplots(figsize=(10, 6))

#Plot a histogram of the simulated data

# density=True normalizes the histogram so that the area is 1,

# which allows for direct comparison with the PDF.

ax.hist(samples, bins=50, density=True, alpha=0.6, color='skyblue', label='Simulated Data (Histogram)')

# Overlay the theoretical Probability Density Function (PDF)

# The PDF for an exponential distribution is f(x; lambda) = lambda \* exp(-lambda \* x),

# where lambda = 1 / mean.

x = np.linspace(0, max(samples), 100)

pdf = (1 / mean) \* np.exp(-x / mean)

ax.plot(x, pdf, 'r-', lw=2, label=f'Theoretical PDF (mean={mean})')

# Set plot title and labels

ax.set\_title(f'Exponential Distribution Simulation (mean={mean}, N={num\_samples})') ax.set\_xlabel('Value')

ax.set\_ylabel('Probability Density')

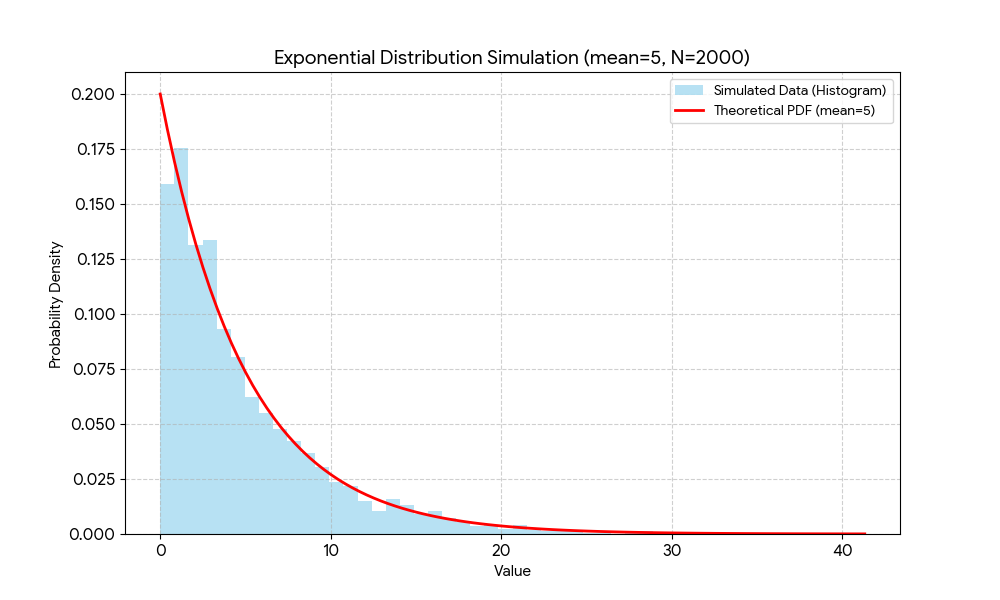
ax.legend()

ax.grid(True, linestyle='--', alpha=0.6)

# Save the plot to a file

plt.savefig('exponential\_distribution\_simulation.png')

plt.close()



**Central Limit Theorem**

6. Simulate the Central Limit Theorem by following these steps

  a. Generate 10,000 random numbers from a uniform distribution.

  b. Draw 1000 samples of size n = 30.

  c. Calculate and visualize the distribution of sample means.

  Steps

      a. Use numpy.random.uniform().

      b. Plot both the uniform distribution and the sample mean distribution for comparison.

Answer:

import numpy as np

import matplotlib.pyplot as plt

# Simulation parameters

num\_initial\_samples = 10000

num\_clt\_samples = 1000

sample\_size = 30

# 1. Generate 10,000 random numbers from a uniform distribution [0, 1)

uniform\_numbers = np.random.uniform(low=0, high=1, size=num\_initial\_samples)

# 2. Draw 1000 samples of size n=30 and calculate their means sample\_means = []

for \_ in range(num\_clt\_samples):

# Draw a random sample of size 30 from the uniform distribution sample = np.random.choice(uniform\_numbers, size=sample\_size)

# Calculate the mean of the sample and append it to the list sample\_means.append(np.mean(sample))

# 3. Visualize the distributions for comparison

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

fig.suptitle('Central Limit Theorem Simulation', fontsize=16)

# Plot the original uniform distribution ax1.hist(uniform\_numbers, bins=50, density=True, color='skyblue', edgecolor='black')

ax1.set\_title('Original Uniform Distribution')

ax1.set\_xlabel('Value')

ax1.set\_ylabel('Density')

ax1.grid(axis='y', alpha=0.75)

# Plot the distribution of the sample means

ax2.hist(sample\_means, bins=50, density=True, color='lightgreen', edgecolor='black') ax2.set\_title(f'Distribution of Sample Means (n={sample\_size})')

ax2.set\_xlabel('Sample Mean')

ax2.set\_ylabel('Density')

ax2.grid(axis='y', alpha=0.75)

# Show a plot with the mean and standard deviation of the sample means

mean\_of\_means = np.mean(sample\_means)

std\_dev\_of\_means = np.std(sample\_means)

ax2.axvline(mean\_of\_means, color='red', linestyle='dashed', linewidth=2, label=f'Mean: {mean\_of\_means:.4f}')

ax2.axvline(mean\_of\_means + std\_dev\_of\_means, color='purple', linestyle='dashed', linewidth=2, label=f'Std Dev: {std\_dev\_of\_means:.4f}')

ax2.axvline(mean\_of\_means - std\_dev\_of\_means, color='purple', linestyle='dashed', linewidth=2)

ax2.legend()

plt.tight\_layout(rect=[0, 0.03, 1, 0.95])

plt.savefig('central\_limit\_theorem\_simulation.png')

plt.close()

