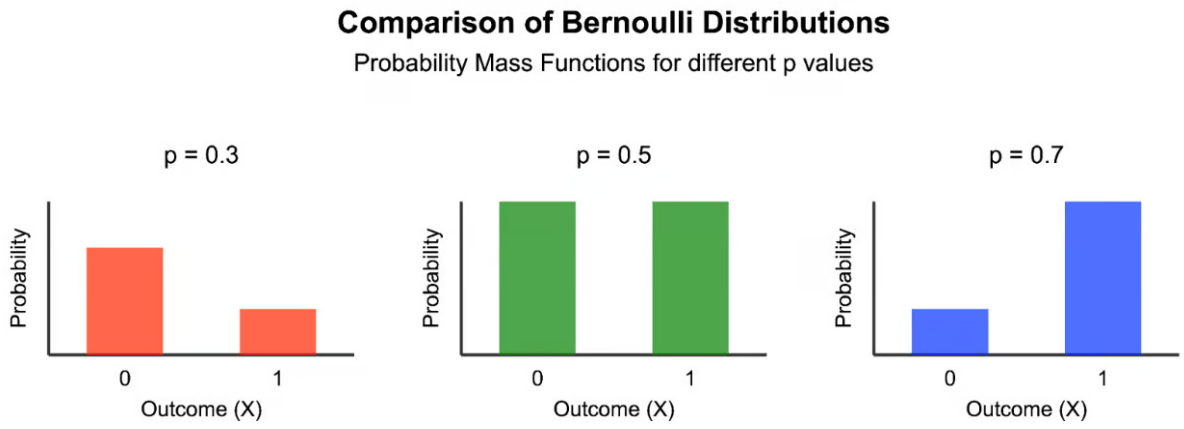
# Bernoulli & Binomial Distribution

## Bernoulli Distribution

- Common in machine learning

The **Bernoulli distribution** is a discrete probability distribution that represents a **single trial** with two possible outcomes:

1. **Success (1)** with probability : $p$
2. **Failure (0)** with probability : $1 - p$




### 1. Definition & Formula

A **random variable X** follows a **Bernoulli distribution** if it takes values:

$$P(X = 1) = p, \quad P(X = 0) = 1 - p$$

where $p$ is the probability of success ( $0 \leq p \leq 1$ ).

**Probability Mass Function (PMF):**

$$P(X = x) = p^x (1 - p)^{1-x}, \quad x \in \{0, 1\}$$

## Relation to Binomial Distribution:

The Bernoulli distribution is a special case of the Binomial distribution where only one trial is performed ( $n$ = 1):

$$\text{Binomial}(n = 1, p) = \text{Bernoulli}(p)$$

**Python Code:**

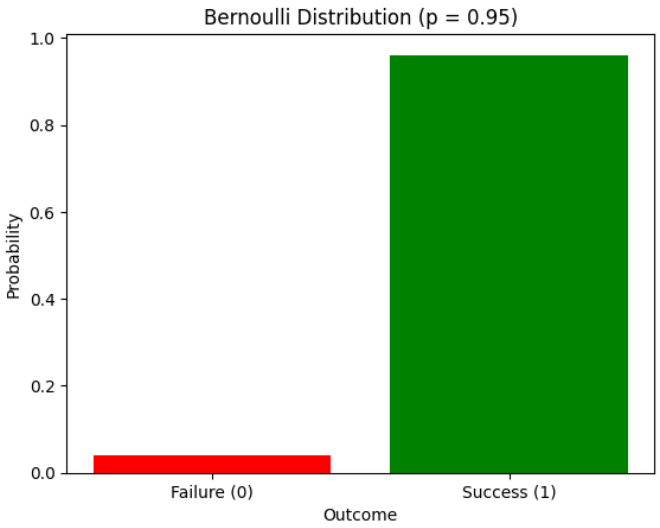
```
import numpy as np
import matplotlib.pyplot as plt

# Parameters
p = 0.95  # 95% chance of success
size = 1000  # 1000 trials

# Generate Bernoulli trials (0 = fail, 1 = success)
data = np.random.binomial(n=1, p=p, size=size)

# Count successes (1s) and failures (0s)
successes = sum(data)
failures = size - successes

print(f"Successes: {successes}, Failures: {failures}")
```

***Outcome:***

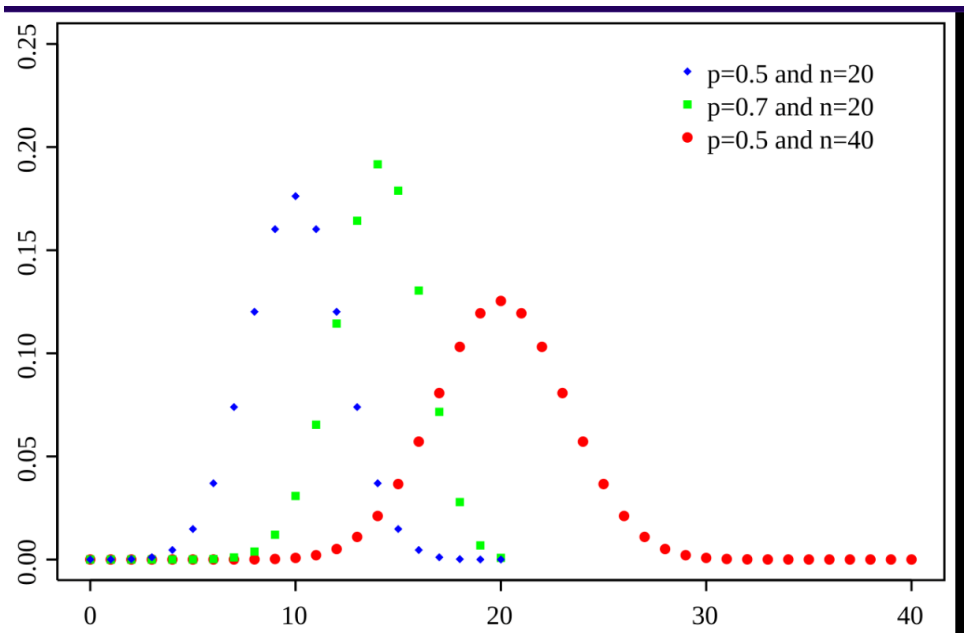
Successes: 961, Failures: 39

- Let's plot the probabilities:

```
# Plotting
plt.bar(['Failure (0)', 'Success (1)'], [failures/size, succe
sses/size], color=['red', 'green'])
plt.xlabel('Outcome')
plt.ylabel('Probability')
plt.title('Bernoulli Distribution (p = 0.95)')
plt.show()
```




# Binomial Distribution




- The **binomial distribution** models the number of **successes** in a fixed number of **independent Bernoulli trials**.
- Each trial has:
  - **Success probability = p**
  - **Failure probability = (1 - p)**
- It answers questions like:
  - "If I flip a coin 10 times, how many heads will I get?"
  - "If a medicine works 70% of the time, how many patients recover in 100 trials?"

## The probability mass function (PMF) of the binomial distribution:

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

- n = total number of trials
- k = number of successes
- p = probability of success
  - $p^k$ → Probability of k successes.
- (1–p) = probability of failure
- $\binom{n}{k}$ = **binomial coefficient** (number of ways to choose k successes in n trials)

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

- $n!$ = Factorial of n (i.e., $n \times (n-1) \times (n-2) \times \ldots \times 1$)
- $k!$ = Factorial of k
- $(n-k)!$ = Factorial of (n - k)

2. **Mean (Expected Value)**:

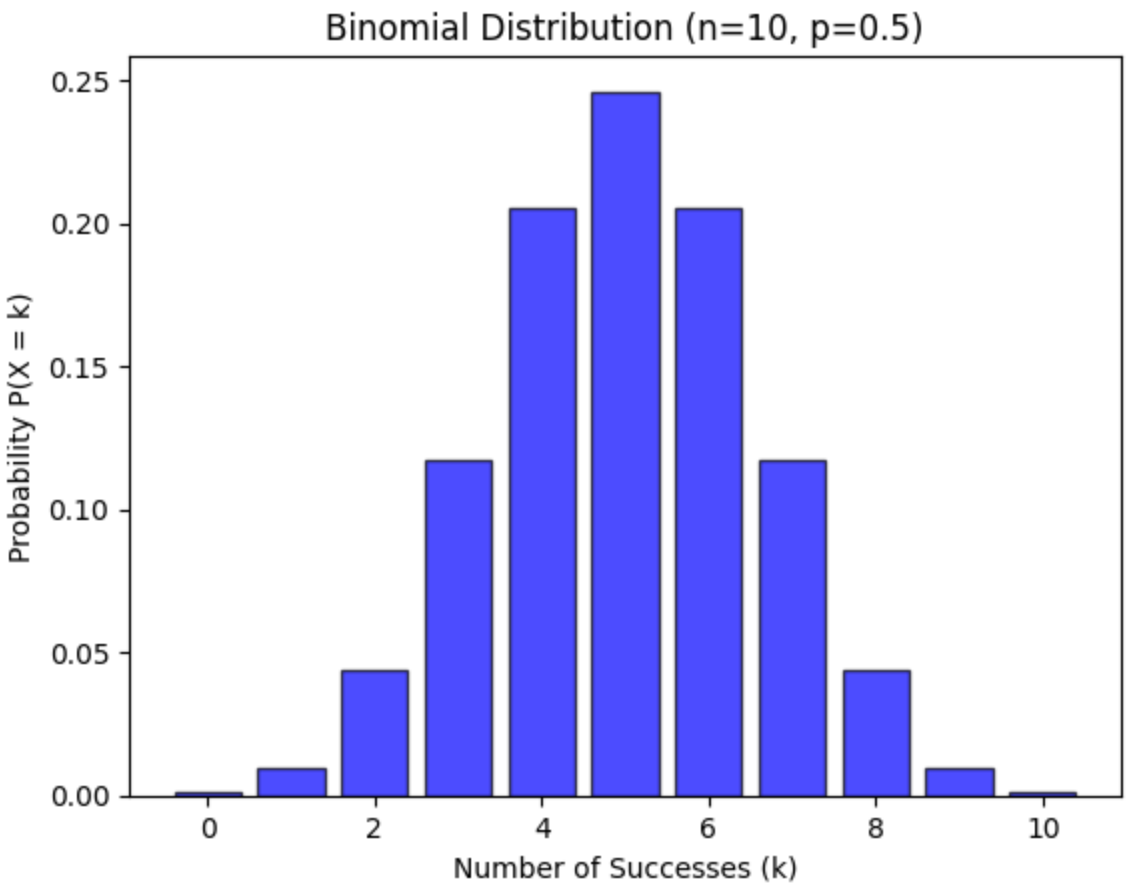
$$\text{Mean} = n \cdot p$$

3. **Variance**:

$$\text{Variance} = n \cdot p \cdot (1-p)$$

```
import numpy as np
import scipy.stats as stats
```

```
import matplotlib.pyplot as plt

# Define binomial parameters
n = 10  # Number of trials
p = 0.5  # Probability of success

# Generate possible values of k (0 to n)
k_values = np.arange(0, n + 1)

# Calculate PMF (probabilities for each k)
pmf_values = stats.binom.pmf(k_values, n, p)

# Plot the PMF as a bar chart
plt.bar(k_values, pmf_values, color='blue', alpha=0.7, edgeco
lor='black')

# Labels and title
plt.xlabel('Number of Successes (k)')
plt.ylabel('Probability P(X = k)')
plt.title(f'Binomial Distribution (n={n}, p={p})')

# Show the plot
plt.show()
```




`k_values = np.arange(0, n + 1):`

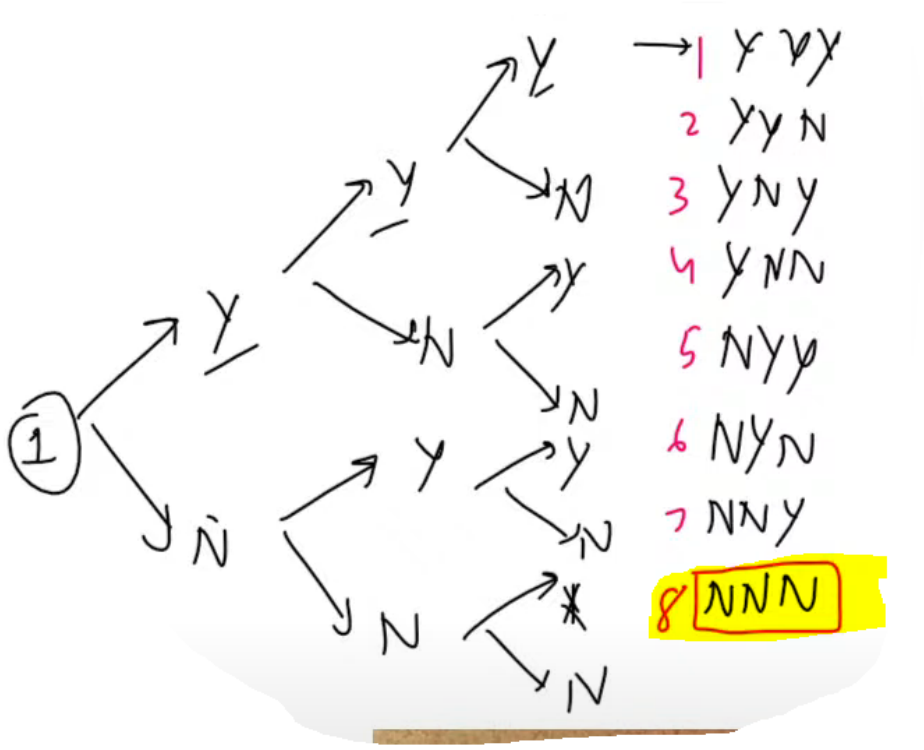
- Creates an array of values from **0 to n** (0 to 10).
- Each value represents **possible success counts** (0 to 10 successes).

`stats.binom.pmf(k_values, n, p)` calculates the probability of getting k successes for each $k$.

- Example values for `pmf_values`: *[0.00097656, 0.00976562, 0.04394531, 0.1171875, ..., 0.00097656]*

***Q. The Probability of anyone watching this lecture in the future and then liking it is 0.5. What is the probability that: No one out of 3 people will like it.***

- There are 8 probabilities → $2^3$
  - 2 p & 3 k




- ***No one out of 3 people will like it → NNN***
- There's only 1 NNN
- Therefore probability= 1/8

## Applications in Machine Learning

- Binary classification problems
- Hypothesis testing
- Logistic regression

- A/B testing

# Sampling Distribution

- **It is a probability distribution of a statistic** (like mean, variance, or proportion) calculated from **multiple samples** drawn from the same population.
- If we take **many random samples** from a population and compute a statistic (e.g., mean), the values of this statistic will form a **distribution**.
- This distribution is called the **sampling distribution of the statistic**.

### Example

Imagine a population of **10 million people** with an **average height of 170 cm**.

- If we take **one sample of 100 people**, the mean height might be **168 cm**.
- Another sample might have a mean of **172 cm**.
- If we repeat this **thousands of times**, we get a **distribution of sample means**.

**Purpose**:

- Quantifies how much a statistic varies from sample to sample.
- Allows estimation of the probability that a sample statistic is close to the population parameter (e.g., confidence intervals, hypothesis testing).

## Key Properties of Sampling Distributions

1. **The Mean of the Sampling Distribution (Expected Value)**
    - The mean of the sampling distribution is equal to the **true population mean**.
    - If μ is the population mean, then:

$$E(\bar{X}) = \mu$$

2. **The Standard Error (SE)**

- The **spread (standard deviation) of the sampling distribution** is called the **Standard Error (SE)**.
- It is given by

  $SE = \frac{\sigma}{\sqrt{n}}$

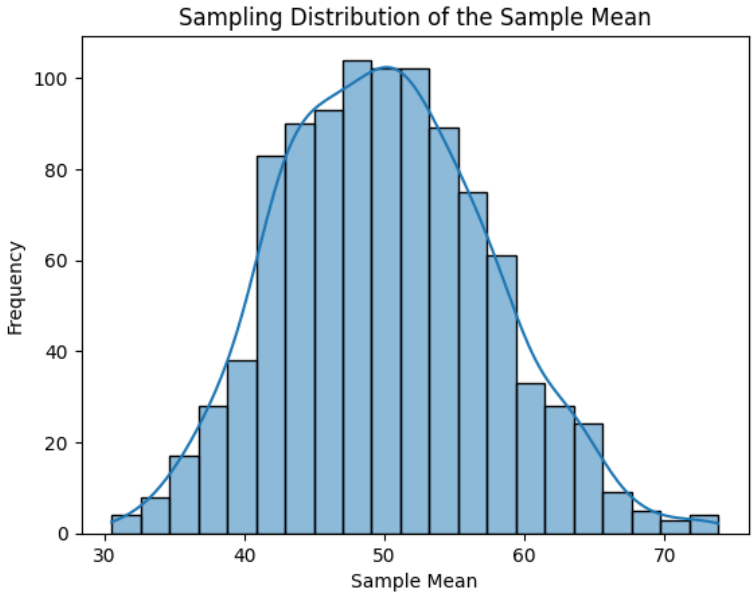
  Where:

  - σ= Population standard deviation
  - n = Sample size
- **Larger sample sizes reduce standard error**, making the sample mean more reliable.

3. **Central Limit Theorem (CLT)**
   - **If we take large enough random samples**, the sampling distribution of the mean will **approximate a normal distribution**, **regardless** of the population's original shape.
   - This is true even if the **population itself is skewed or non-normal**.

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

# Generate population data (non-normal distribution)
population = np.random.exponential(scale=50, size=10000)

# Take multiple samples of size 50 and compute means
sample_means = [np.mean(np.random.choice(population, 50)) for
_ in range(1000)]

# Plot the sampling distribution of the sample mean
sns.histplot(sample_means, kde=True)
plt.title("Sampling Distribution of the Sample Mean")
plt.xlabel("Sample Mean")
```

```
plt.ylabel("Frequency")
plt.show()
```

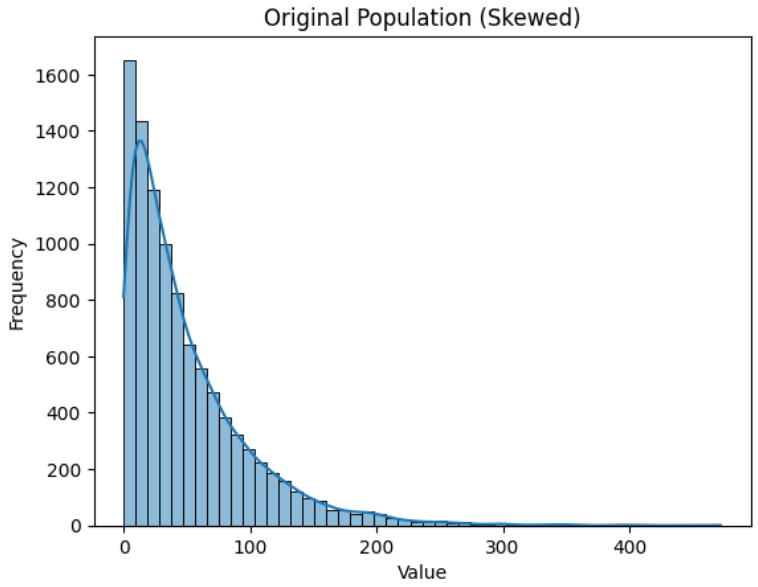



# Central Limit Theorem

- S**ampling distribution of the sample mean** becomes approximately **normal** (bell-shaped) as the sample size increases, even if the underlying population distribution is not normal.
- In above example, `population` was **exponential**
    - But the **distribution of the sample mean** was normally distributed.

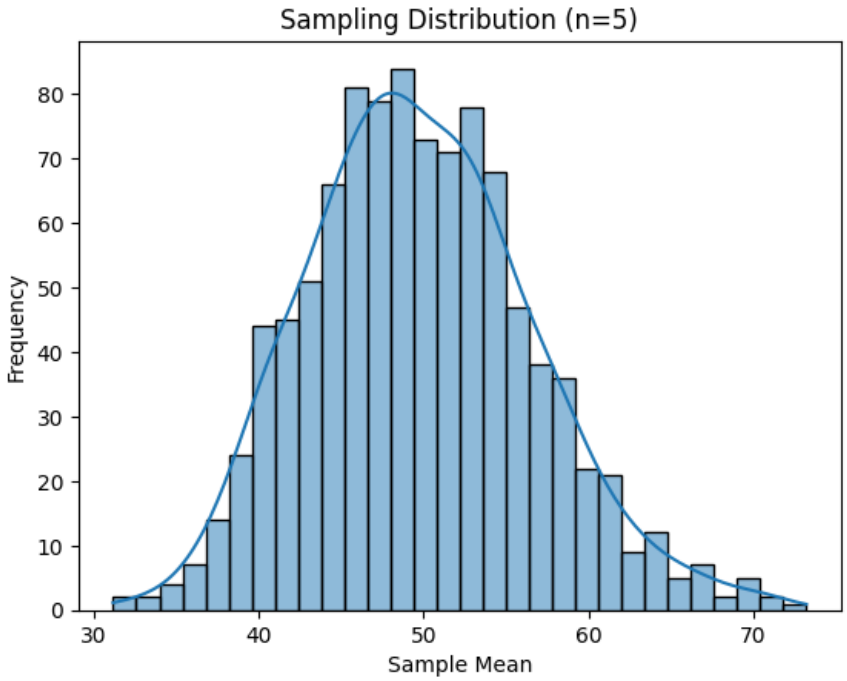
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Generate a non-normal population (exponential)
population = np.random.exponential(scale=50, size=10000)
```

```
# Plot the population distribution
sns.histplot(population, kde=True, bins=50)
plt.title("Original Population (Skewed)")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()
```




- This is a non-normal population.
- But if we draw samples from this and plot means of them, it will follow a normal distribution

```
sample_means_50 = [np.mean(np.random.choice(population, 50))
for _ in range(1000)]

# Plot the sampling distribution of mean (n=5)
sns.histplot(sample_means_50, kde=True, bins=30)
plt.title("Sampling Distribution (n=5)")
plt.xlabel("Sample Mean")
plt.ylabel("Frequency")
plt.show()
```




- Larger the sample, closer it it to normal distribution

## Practical Uses of CLT in Data Science & Machine Learning

### (A) Confidence Intervals

- If we know the **sampling distribution is normal**, we can compute **confidence intervals** for predictions.
- Example: Predicting **customer spending** from a sample.

### (B) Hypothesis Testing

- Many tests (e.g., **t-tests, Z-tests**) **assume normality**.
- Even if **data is skewed**, CLT allows testing under normal assumptions for large n.

  nn

### (C) A/B Testing

- Used in **website conversion rate analysis**.
- The mean difference in test/control groups follows a normal distribution.

### (D) Machine Learning Model Validation

- Helps in **resampling methods** (e.g., bootstrap) for **model evaluation**.

## Conditions Required for the Central Limit Theorem (CLT) to Hold

- The samples must be **randomly selected** from the population.
- The rule of thumb: $n$ ≥ 30 for most distributions.
  - If the population is highly skewed, a larger $n$ n (e.g., 50 or 100) may be needed.
  - If the population is already normal, even a small $n$ n is sufficient.
- Finite Variance ($sigma^2$)
  - If variance is infinite (e.g., **Cauchy distribution**), the sample mean does not converge to normality.

# Case Study - What is the average income of Indians

**Step-by-step process:**

1. Collect multiple random samples of salaries from a representative group of Indians. Each sample should be large enough (usually, n > 30) to ensure the CLT holds. Make sure the samples are representative and unbiased to avoid skewed results.
2. Calculate the sample mean (average salary) and sample standard deviation for each sample.
3. Calculate the average of the sample means. This value will be your best estimate of the population mean (average salary of all Indians).

4. Calculate the standard error of the sample means, which is the standard deviation of the sample means divided by the square root of the number of samples.
5. Calculate the confidence interval around the average of the sample means to get a range within which the true population mean likely falls. For a 95% confidence interval:

***lower_limit = average_sample_means - 1.96 * standard_error***
***upper_limit = average_sample_means + 1.96 * standard_error***

6. Report the estimated average salary and the confidence interval.

# CLT Case Study

**The code:**

- Generates a large "population" of salaries (in thousands) drawn from a lognormal distribution.
- Takes many random samples (each of a fixed sample size) from this population.
- Computes the sample means and standard deviations.
- Uses these sample means to estimate the overall average salary, computes the standard error, and then calculates a 95% confidence interval for the estimated average.

```
import numpy as np

# Set the parameters
population_size = 100000
sample_size = 50
num_samples = 100

# Generate a random representative sample of salaries (in tho
usands)
# You should replace this with actual collected salary data
np.random.seed(42)  # Setting a seed for reproducibility
```

```
population_salaries = np.random.lognormal(mean=4.5, sigma=0.
8, size=population_size)

# Generate multiple samples and calculate the sample means an
d standard deviations
sample_means = []
sample_std_devs = []

for _ in range(num_samples):
  sample_salaries = np.random.choice(population_salaries, siz
e=sample_size)
  sample_means.append(np.mean(sample_salaries))
  sample_std_devs.append(np.std(sample_salaries))

# Calculate the average of the sample means and the standard
error
average_sample_means = np.mean(sample_means)
standard_error = np.std(sample_means) / np.sqrt(num_samples)

# Calculate the 95% confidence interval
margin_of_error = 1.96 * standard_error
lower_limit = average_sample_means - margin_of_error
upper_limit = average_sample_means + margin_of_error

# Report the results
print(f"Estimated average salary (in thousands): {average_sam
ple_means:.2f}")
print(f"95% confidence interval (in thousands): ({lower_limi
t:.2f}, {upper_limit:.2f})")
```

**Output:**

Estimated average salary (in thousands): **124.74**
95% confidence interval (in thousands): (
121.23, 128.26)

`np.random.seed(42)`

- **Purpose:** Ensures that the random numbers generated are reproducible (i.e., the same every time you run the code).
- **Detail:** The seed value `42` is arbitrary but fixed; without setting the seed, each run would produce different random values.

`np.random.lognormal(mean=4.5, sigma=0.8, size=population_size)`

- **Distribution:** **The lognormal distribution.**
  - **Why lognormal?** Salaries are positive and often skewed, making the lognormal a realistic choice.

  **Parameters:**

  - `mean=4.5`: This is the mean of the underlying normal distribution.
    - It is not the mean of the generated salaries but of the logarithms of those salaries.
  - `sigma=0.8`: This is the standard deviation of the underlying normal distribution.
  - `size=population_size`: Generates `100000` random values.

```
for _ in range(num_samples):
  sample_salaries = np.random.choice(population_salaries, siz
e=sample_size)
  sample_means.append(np.mean(sample_salaries))
  sample_std_devs.append(np.std(sample_salaries))
```

Loop Structure: The for loop runs num_samples (i.e., 100) times. The underscore _ is used as a throwaway variable since the index is not needed.

- `sample_salaries = np.random.choice(population_salaries, size=sample_size)`
  - **Purpose:** Randomly selects `sample_size` (50) values from `population_salaries`.
  - **Note:** `np.random.choice` randomly picks values (with replacement by default) from the population array.

- After the loop,
  - `sample_means` holds 100 values (one for each sample),
  - `sample_std_devs` holds 100 corresponding standard deviations.

- `average_sample_means = np.mean(sample_means)`
  - **Purpose:** Compute the mean of all the sample means.
  - **Interpretation:**This value is the overall estimated average salary (in thousands) of the population based on the 100 samples.

- `standard_error = np.std(sample_means) / np.sqrt(num_samples)`
  - **Purpose:** Calculate the **standard error of the mean**.
  - **Steps:**
    1. `np.std(sample_means)`: Compute the standard deviation of the sample means. This shows how much the sample means vary.
    2. **Division by** `np.sqrt(num_samples)`:
       - Adjusts for the number of samples to provide the standard error.
       - **Rationale:** Standard error (SE) is the standard deviation of the sampling distribution, and it decreases as the number of samples increases.
  - **Result:** The computed `standard_error` quantifies the uncertainty around the estimated average salary.

### Margin of Error Calculation:

- `margin_of_error = 1.96 * standard_error`
  - **Why 1.96?**For a normal distribution, approximately 95% of the data lies within 1.96 standard deviations of the mean.

### Confidence Interval:

- `lower_limit = average_sample_means - margin_of_error` The lower bound of the interval.
- `upper_limit = average_sample_means + margin_of_error` The upper bound of the interval.