

▼ Normal Distribution

Prompt-1:

give me 100 random numbers using normal distribution in python

Note:

if don't mention the mean and variance, it will follow standard normal distribution.

```
import numpy as np

# Generate 100 random numbers using a normal distribution
# with a mean of 0 and a standard deviation of 1
random_numbers = np.random.randn(100)

print(random_numbers)

[-1.26607216  0.50460752  0.11115948 -0.25854711  0.26165614 -0.4428592
  0.3336817  0.59810849  0.16670752  0.53653646  1.04154293 -2.03979691
 -0.00305849 -1.75875505  1.21175479 -1.50478349 -1.10102675  0.39749525
  0.23250018 -0.5441935  0.68488804  0.79368008  0.38544147  0.75522192
 -1.05740659  0.96532718 -0.0819075 -0.77574759 -0.70070558 -0.14619132
  0.23732921 -0.07410313 -1.30874449  1.47483228  0.03700598 -1.31355497
 -0.19832612  0.97755705 -1.00337594 -0.51604858 -0.53075795  0.09760394
 -1.01785196  1.38018796  0.63783838  1.21117216  0.15551737  0.20207785
 -0.12837322 -0.91989279  0.27188376  0.29559453  2.1829401 -0.78455635
  0.42806623  0.46776497  0.57133327  0.60882826 -1.24941182 -0.70858088
  1.68370425  1.96426535 -1.0481233  0.74747313  0.53658817 -0.32204126
  0.42136839  0.43486997  1.28056212 -1.30503622 -1.54707743  1.02209226
  1.67421123 -1.88801717 -0.20555417 -1.38912428 -1.77335412 -1.07113026
  0.25894588  0.887845  1.53823592 -0.28684881  0.7890469  0.67465521
 -0.53540456 -2.24254491 -1.00029573  0.66223767  1.34330644 -0.92361691
  0.00429702 -1.71492933 -0.77272608 -0.67639056 -0.08982872 -1.06434135
 -0.69286211 -0.49652661 -1.35591004  2.32004263]
```

Prompt-2:

give me 100 random numbers using normal distribution with mean 5 and variance 0.1 in python

```
import numpy as np

mean = 5
std_dev = np.sqrt(0.1) # Standard deviation is the square root of variance

random_numbers = np.random.normal(mean, std_dev, 100)

print(random_numbers)

[4.86168254 5.07062102 4.77094181 4.80040915 4.67057783 5.3335688
 5.05656611 4.82444061 4.5795864 4.86174887 4.6560031 4.902903
 5.14782447 5.51601075 5.59816012 5.02685924 5.23368669 5.33423236
 5.15550337 5.00020047 4.6960955 5.09711998 4.92022046 4.73551948
 5.06433392 5.53059973 5.08377288 5.4035173 5.35325918 5.17181288
 5.43325238 5.31588858 5.4949218 4.91191273 5.18455955 5.01299081
 4.65796488 4.80112201 5.28714697 4.78560924 5.24093315 5.04436425
 4.43179767 5.58082159 4.80525248 5.46733204 5.14492732 5.21520662
 4.86529078 4.60516731 4.84180527 5.11388804 4.81718255 4.69980144
 5.16193856 4.37041194 4.84929012 4.99506047 4.87294338 5.30614682
 5.01233705 4.99749336 4.9382036 5.37970659 4.60401436 4.84059408
 4.93003771 4.87971897 4.67503347 4.98183928 5.25723978 5.03974372
 5.06265573 5.01242037 5.59530495 5.33681167 4.66783743 5.25717732
 4.93744276 5.21729207 4.66756691 5.41461727 4.41690636 5.04112152
 5.73602938 5.27034562 4.59927986 5.24687784 5.09449686 5.26753947
 5.08454743 5.42733796 5.4686879 5.08712418 4.94269271 4.97192238
 4.59764131 5.14985395 4.48851383 5.23462335]
```

Prompt-3:

now show a graph with 5 line graphs for 5 different mean and standard deviation using normal distribution

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

# Define x values
```

```

x = np.linspace(-5, 13, 1000) # This range should cover all the curves well

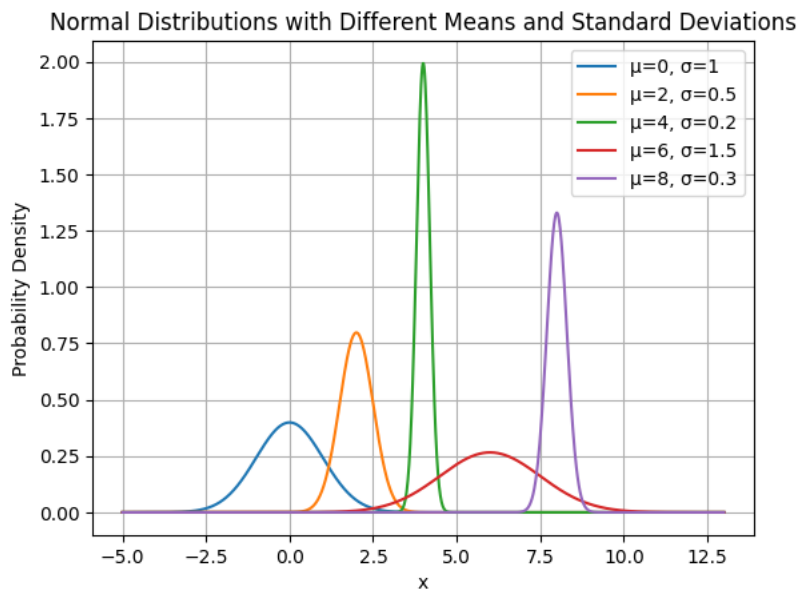
# Define the means and standard deviations
means = [0, 2, 4, 6, 8]
std_devs = [1, 0.5, 0.2, 1.5, 0.3]

# Plot each normal distribution curve
for mean, std_dev in zip(means, std_devs):
    y = (1 / (std_dev * np.sqrt(2 * np.pi))) * np.exp(-0.5 * ((x - mean) / std_dev)**2)
    plt.plot(x, y, label=f"μ={mean}, σ={std_dev}")

# Add legend, title, and labels
plt.legend()
plt.title("Normal Distributions with Different Means and Standard Deviations")
plt.xlabel("x")
plt.ylabel("Probability Density")

plt.grid(True)
plt.show()

```



▼ Binomial Distribution

Prompt-1:

generate a python script which will give me 100 random numbers from binomial distribution with probability 0.5 and no of trails 40. show it in a graph

```

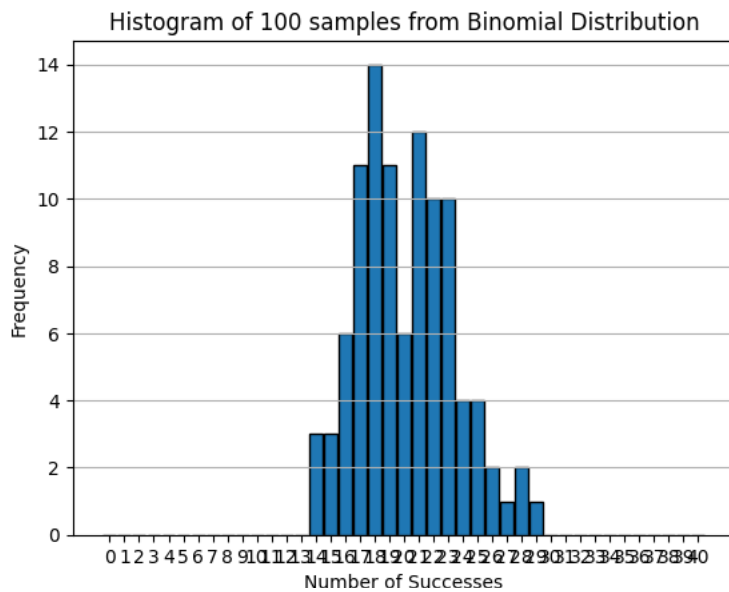
import numpy as np
import matplotlib.pyplot as plt

# Parameters
n_trials = 40
p_success = 0.5
n_samples = 100

# Generate random numbers from a binomial distribution
samples = np.random.binomial(n_trials, p_success, n_samples)

# Plotting
plt.hist(samples, bins=np.arange(n_trials+2)-0.5, edgecolor="k", align='mid', rwidth=0.9)
plt.xlabel('Number of Successes')
plt.ylabel('Frequency')
plt.title('Histogram of 100 samples from Binomial Distribution')
plt.xticks(np.arange(n_trials+1))
plt.grid(axis='y')
plt.show()

```



Prompt-2:

increase the graph size

```
# Set the figure size
plt.figure(figsize=(12, 8))
```

Prompt-3:

give me 5 different lines for 5 different probability in a same graph in python using binomial distribution

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

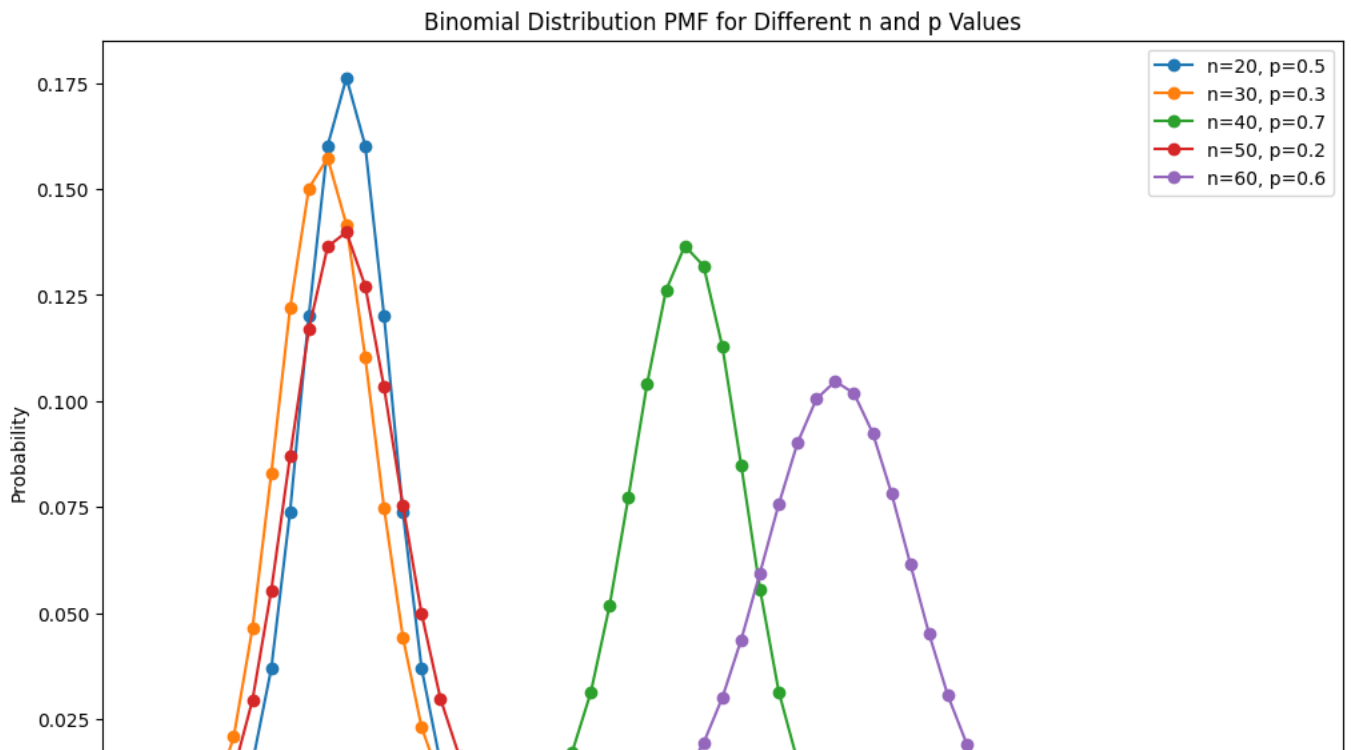
# Define n and p parameter combinations
params = [
    (20, 0.5),
    (30, 0.3),
    (40, 0.7),
    (50, 0.2),
    (60, 0.6)
]

# Set the figure size
plt.figure(figsize=(12, 8))

# Plot binomial PMF for each n and p combination
for n, p in params:
    x = np.arange(0, n+1)
    y = binom.pmf(x, n, p)
    plt.plot(x, y, '-o', label=f'n={n}, p={p}')

# Set labels, title, and legend
plt.xlabel('Number of Successes')
plt.ylabel('Probability')
plt.title('Binomial Distribution PMF for Different n and p Values')
plt.legend()

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



▼ Poisson Distribution

Prompt-1:

generate 100 random numbers using poisson ditribution in python

```
import numpy as np

# Setting the random seed for reproducibility
np.random.seed(0)

# Lambda value
lmbda = 5

# Generate 100 random numbers from a Poisson distribution with lambda=5
random_numbers = np.random.poisson(lmbda, 100)

print(random_numbers)
```

[9 5 6 5 5 8 4 5 4 3 3 7 3 3 4 5 2 1 7 7 10 5 8 8
6 4 4 3 1 5 6 5 5 4 1 7 2 7 6 2 6 5 7 5 4 5 10 6
4 7 3 2 2 6 2 5 3 6 2 4 4 4 8 7 11 3 4 1 4 5 1 2
2 5 5 5 7 8 4 6 4 10 3 8 5 7 3 8 2 1 4 4 4 4 6 4
12 1 6 3]

Prompt-2:

show me a bar graph for 1000 random numbers getting from poisson distribution

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

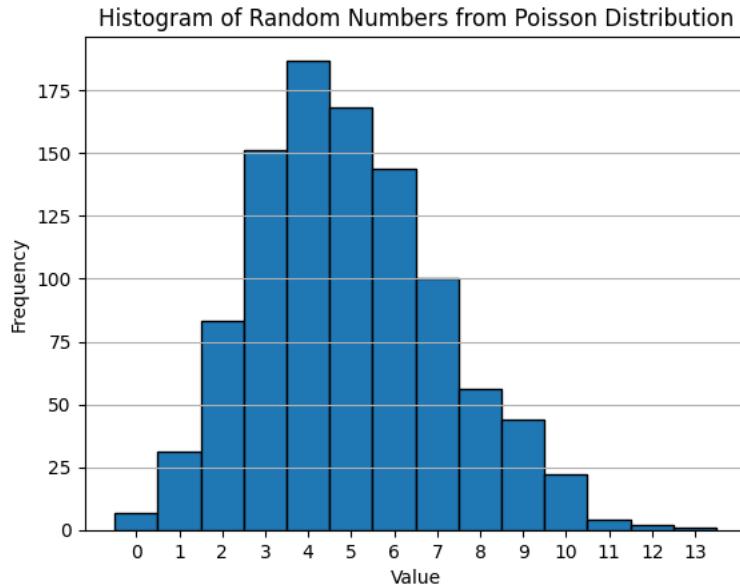
# Setting the random seed for reproducibility
np.random.seed(0)

# Lambda value
lmbda = 5

# Generate 1000 random numbers from a Poisson distribution with lambda=5
random_numbers = np.random.poisson(lmbda, 1000)

# Plot histogram
plt.hist(random_numbers, bins=np.arange(random_numbers.min(), random_numbers.max()+2) - 0.5, align='mid', edgecolor='black')
plt.xlabel('Value')
```

```
plt.ylabel('Frequency')
plt.title('Histogram of Random Numbers from Poisson Distribution')
plt.xticks(np.arange(random_numbers.min(), random_numbers.max()+1))
plt.grid(axis='y')
plt.show()
```



Prompt-3:

show me 5 different line graphs for 5 lamdas in the same graph using poisson distribution

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

# Setting the random seed for reproducibility
np.random.seed(0)

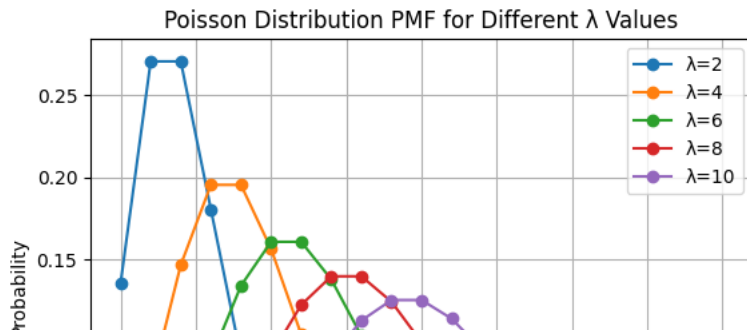
# Define the PMF of Poisson distribution
def poisson_pmf(k, lmbda):
    return (np.exp(-lmbda) * lmbda**k) / np.math.factorial(k)

# Lambda values
lambdas = [2, 4, 6, 8, 10]

# Generate k values (e.g., from 0 to 20 for clear visualization)
k_values = np.arange(0, 21)

# Plot the PMF for each lambda value
for lmbda in lambdas:
    probabilities = [poisson_pmf(k, lmbda) for k in k_values]
    plt.plot(k_values, probabilities, label=f'λ={lmbda}', marker='o')

plt.title('Poisson Distribution PMF for Different λ Values')
plt.xlabel('k')
plt.ylabel('Probability')
plt.legend()
plt.grid(True)
plt.show()
```



▼ Beta Distribution

Prompt-1:

generate a python script which will give me 100 random numbers from beta distribution with alpha 2 and beta 5. show it in a graph

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

# Parameters
alpha = 2
beta = 5
n_samples = 100

# Generate random numbers from a beta distribution
samples = np.random.beta(alpha, beta, n_samples)

# Set the figure size
plt.figure(figsize=(12, 8))

# Plotting
plt.hist(samples, bins=50, edgecolor="k", alpha=0.7)
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram of 100 samples from Beta Distribution (α=2, β=5)')
plt.grid(axis='y')
plt.show()
```

Prompt-2:

give me 5 line graphs in the same graph for different alpha and beta using beta distribution

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

# Define alpha and beta parameter combinations
params = [
    (2, 5),
    (5, 2),
    (3, 3),
    (0.5, 0.5),
    (2, 8)
]

# Prepare x values for plotting
x = np.linspace(0, 1, 1000)

# Set the figure size
plt.figure(figsize=(12, 8))

# Plot beta PDF for each alpha and beta combination
for a, b in params:
    y = beta.pdf(x, a, b)
    plt.plot(x, y, label=f'α={a}, β={b}')

# Set labels, title, and legend
plt.xlabel('Value')
plt.ylabel('Density')
plt.title('Beta Distribution PDF for Different α and β Values')
```

```
plt.legend()

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

Prompt-4:

give me the value of alpha and beta of beta distribution if mean 0.4, variance 0.01

Prompt-5:

using these alpha beta give me 1000 random numbers of beta distribution

▼ Confidence Intervals

Prompt-1:

why do we use confidence interval? explain me with an easy example

Prompt-2:

how to calculate ci in python

```
import numpy as np
from scipy import stats

# Sample data
sample_mean = 99
sample_std_dev = 3
n = 30
confidence_level = 0.95

# Calculate the standard error
standard_error = sample_std_dev / np.sqrt(n)

# Calculate the confidence interval
ci = stats.t.interval(
    confidence_level,      # Confidence level
    df=n-1,               # Degrees of freedom
    loc=sample_mean,      # Sample mean
    scale=standard_error  # Standard error
)

print("95% Confidence Interval:", ci)

95% Confidence Interval: (97.8797815897257, 100.1202184102743)
```

Prompt-3:

Relative Half-Width of 95% CI

```
import numpy as np
from scipy import stats

# Sample data
sample_mean = 99
sample_std_dev = 3
n = 30

# Calculate the standard error
standard_error = sample_std_dev / np.sqrt(n)

# Calculate the half-width of the 95% CI using the t-distribution
alpha = 0.05 # for a 95% CI
t_value = stats.t.ppf(1 - alpha/2, df=n-1)
half_width = t_value * standard_error

# Calculate the relative half-width
relative_half_width = half_width / sample_mean
```

```
print("Relative Half-Width:", relative_half_width)
```

Prompt-4:

take no of cases 2500, 3 probabilities such as 0.1, 0.25, 0.5 and 5 iterations with no of simulations (200, 500, 1000, 2000, 5000, 10000). for each probability and iteration generate random numbers using binomial distribution individually for size =no of simulations. then calculate the confidence interval and Relative Half-Width of 95% CI for each probability and no of simulations

```
import numpy as np
from scipy import stats

# Define the probabilities and sizes
probabilities = [0.1, 0.25, 0.5]
sizes = [200, 500, 1000, 2000, 5000, 10000]
iterations = 5

# Function to calculate the confidence interval
def calculate_CI(data, alpha=0.05):
    sample_mean = np.mean(data)
    standard_error = np.std(data, ddof=1) / np.sqrt(len(data))
    t_value = stats.t.ppf(1 - alpha/2, df=len(data)-1)
    half_width = t_value * standard_error
    return (sample_mean - half_width, sample_mean + half_width), half_width / sample_mean

for p in probabilities:
    print(f"Probability: {p}")
    for n in sizes:
        print(f"\tSize: {n}")
        for i in range(iterations):
            # Generate random numbers using binomial distribution
            data = np.random.binomial(2500, p, n)

            # Calculate the confidence interval and relative half-width
            CI, RHW = calculate_CI(data)
            print(f"\t\tIteration {i + 1}: CI = {CI}, RHW = {RHW}")
```



```

Iteration 1: CI = (1250.0014291/14103, 1252.1915/08285899), RHW = 0.0008/52888594842519
Iteration 2: CI = (1248.8012544109736, 1250.9477455890262), RHW = 0.0008586826829623743
Iteration 3: CI = (1248.4586142517992, 1250.6103857482008), RHW = 0.000861029245851819
Iteration 4: CI = (1249.0404648559593, 1251.221535144041), RHW = 0.000872336694347174
Iteration 5: CI = (1248.982047531685, 1251.151952468315), RHW = 0.0008679154543836362
Size: 5000
Iteration 1: CI = (1249.681642190184, 1251.071957809816), RHW = 0.0005559586596744522
Iteration 2: CI = (1249.0184170113534, 1250.4155829886467), RHW = 0.000558992946920564
Iteration 3: CI = (1249.155292697126, 1250.5251073028742), RHW = 0.0005479958980949046
Iteration 4: CI = (1249.981412471234, 1251.376587528766), RHW = 0.0005577670439545871
Iteration 5: CI = (1249.5115775086308, 1250.920422491369), RHW = 0.0005634406305542729
Size: 10000
Iteration 1: CI = (1249.6598216075547, 1250.630778392445), RHW = 0.00038833757359664375
Iteration 2: CI = (1249.697475701708, 1250.677524298292), RHW = 0.0003919606445368934
Iteration 3: CI = (1249.772008919668, 1250.759191080332), RHW = 0.0003947889795032299
Iteration 4: CI = (1249.5682259535438, 1250.5451740464564), RHW = 0.0003907615122228748
Iteration 5: CI = (1249.6592514142872, 1250.641148585713), RHW = 0.0003927116803347897

```

Prompt-5:

now show me 3 line plots for each probability with x axis = no of simulations and y axis = relative half width 95% ci

```

import numpy as np
from scipy import stats
import matplotlib.pyplot as plt

# Define the probabilities and sizes
probabilities = [0.1, 0.25, 0.5]
sim_sizes = [200, 500, 1000, 2000, 5000, 10000]
iterations = 5
n = 2500 # No of cases

# Function to calculate the confidence interval
def calculate_CI(data, alpha=0.05):
    sample_mean = np.mean(data)
    standard_error = np.std(data, ddof=1) / np.sqrt(len(data))
    t_value = stats.t.ppf(1 - alpha/2, df=len(data)-1)
    half_width = t_value * standard_error
    return half_width / sample_mean

results = {}

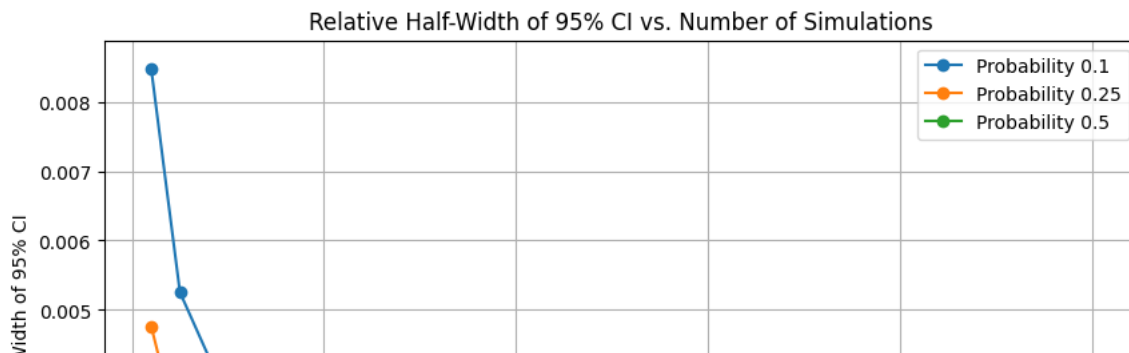
for p in probabilities:
    rhw_values = []
    for size in sim_sizes:
        avg_rhw = np.mean([calculate_CI(np.random.binomial(n, p, size)) for _ in range(iterations)])
        rhw_values.append(avg_rhw)
    results[p] = rhw_values

# Plotting the results
plt.figure(figsize=(10,6))

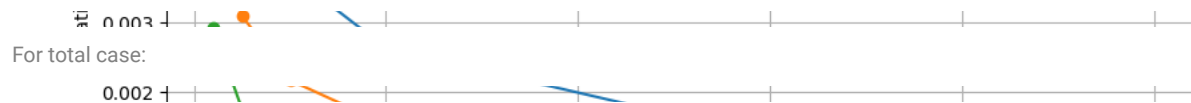
for p, rhw_values in results.items():
    plt.plot(sim_sizes, rhw_values, label=f'Probability {p}', marker='o')

plt.xlabel('Number of Simulations')
plt.ylabel('Relative Half-Width of 95% CI')
plt.title('Relative Half-Width of 95% CI vs. Number of Simulations')
plt.legend()
plt.grid(True)
plt.show()

```



Simulation: Using Total Cases



For total case:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
import matplotlib.pyplot as plt
#Initialization of fixed parameters
cases = 201
a1 = 0.59
a2 = 0.723
a3 = 0.128
population = 1019847
n_iter = 10

#Define alpha and beta values
def getAlphaBeta(mu, sigma):
    alpha=((1-mu)/sigma**2)-(1/mu))*mu**2
    beta=((1/mu)-1)*alpha

    return alpha, beta

true_incidence = cases / population
cases_dist = np.random.binomial(n = population, p=true_incidence, size=n_iter)
cases_dist

array([175, 203, 211, 217, 194, 196, 192, 218, 239, 169])

(alpha1,beta1) = getAlphaBeta(a1,0.05) #Variance = 0.5 will show an error
a1_dist = np.random.beta(alpha1, beta1, size=n_iter)

(alpha2,beta2) = getAlphaBeta(a2,0.05)
a2_dist = np.random.beta(alpha2, beta2, size=n_iter)

(alpha3,beta3) = getAlphaBeta(a3,0.05)
a3_dist = np.random.beta(alpha3, beta3, size=n_iter)

crude_rate_dist = cases_dist / population
crude_rate_dist

array([0.00017159, 0.00019905, 0.00020689, 0.00021278, 0.00019022,
       0.00019219, 0.00018826, 0.00021376, 0.00023435, 0.00016571])

adjusted_rate_dist = crude_rate_dist / (a1_dist * a2_dist * a3_dist)
adjusted_rate_dist

array([0.0005358 , 0.00059831, 0.00068058, 0.00068735, 0.00046798,
       0.00052041, 0.00066199, 0.00060578, 0.00064037, 0.00052826])
```

Simulation: Bangladesh inpatient (DSH) for typhi

The Supplementary link is [here](#)

For Bangladesh inpatient (DSH) the table is given below:

Age Group	A1	A2	A3	Cases	Population
< 2 years	0.59	0.989	0.613	35	77,958
2 - 4 years	0.59	0.988	0.613	80	1,73,878
5 - 15 years	0.59	0.991	0.613	85	5,88,070

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
import matplotlib.pyplot as plt
#Initialization of fixed parameters
numbers = (35, 80, 85)
a1 = (0.59, 0.59, 0.59)
a2 = (0.989, 0.988, 0.991) #DSH
a3 = (0.613, 0.613, 0.613)
pop_changing = (77958, 173878, 588070)
age_group = ('<2', '2-4', '5-15')
n_iter = 100000

incidence = []
lower_ci = []
upper_ci = []

#Define alpha and beta values
def getAlphaBeta(mu, sigma):
    alpha=((1-mu)/sigma**2)-(1/mu))*mu**2
    beta=((1/mu)-1)*alpha

    return alpha, beta
#Calculate adjusted incidence for each year

for x in range(len(age_group)):
    # Define the distribution of cases
    true_incidence = numbers[x]/pop_changing[x]
    cases_dist = np.random.binomial(n=pop_changing[x], p=true_incidence, size=n_iter)

    # Define the distributions of the adjustment factors
    (alpha1,beta1) = getAlphaBeta(a1[x],0.05)
    (alpha2,beta2) = getAlphaBeta(a2[x],0.05)
    (alpha3,beta3) = getAlphaBeta(a3[x],0.05)
    a1_dist = np.random.beta(alpha1, beta1, size=n_iter)
    a2_dist = np.random.beta(alpha2, beta2, size=n_iter)
    a3_dist = np.random.beta(alpha3, beta3, size=n_iter)

    # Calculate the crude incidence rate
    crude_rate_dist = cases_dist / pop_changing[x]

    # Calculate the adjusted incidence rate
    adjusted_rate_dist = crude_rate_dist / (a1_dist * a2_dist * a3_dist)
    incidence.append(np.mean(adjusted_rate_dist*100000))

    # Calculate the uncertainty intervals (e.g. 95% confidence intervals) for the adjusted incidence rate
    lower_ci.append(np.percentile(adjusted_rate_dist*100000,2.5))
    upper_ci.append(np.percentile(adjusted_rate_dist*100000,97.5))

incidence
#Plotting the values

plt.plot(age_group, incidence, '#8F262A', linewidth=2, markersize=12)
#plt.plot(year, lower_ci,'#B298CB', linewidth=2, markersize=12)
#plt.plot(year, upper_ci,'#98CB98', linewidth=2, markersize=12)
plt.fill_between(age_group, lower_ci, upper_ci, color='#8F262A', alpha=.15)
plt.legend(['Adjusted incidence'])
plt.title("simulation of typhi incidence")
```



Text(0.5, 1.0, 'simulation of typhi incidence')

