Advanced Statistics Assignment 2: Sampling Distribution & Estimation

Alli Ajagbe Olasunkanmi

Question 1

1. **Obtaining the sampling distribution of the three estimators for all three models for n = 20**

Defining the estimators:

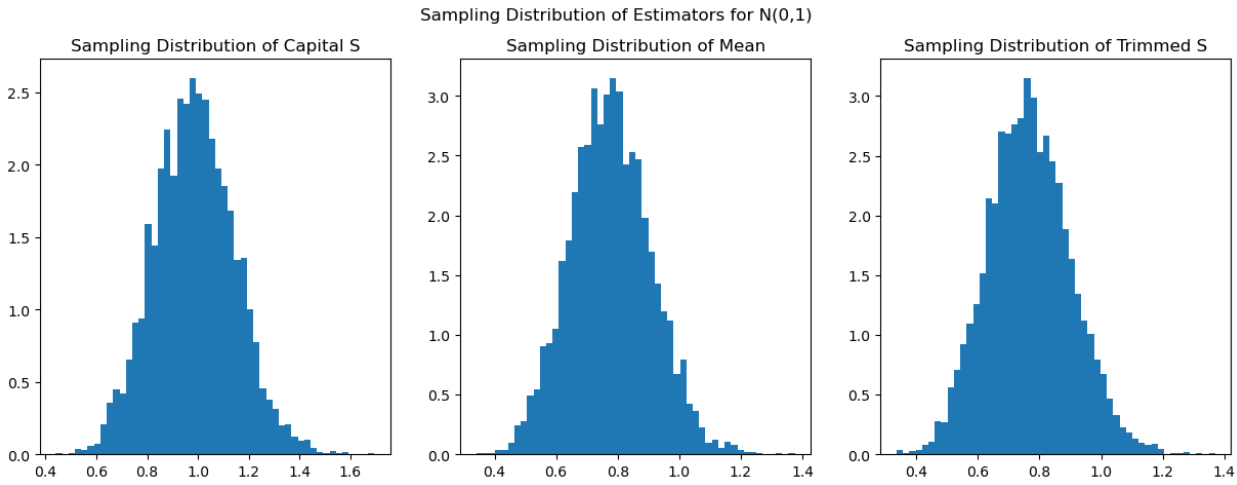
```python
def myEstimators(n,x):

    # sample standard deviation S
    divisor = 1/(n - 1)
    capital_S = np.sqrt(divisor * np.sum((x - np.mean(x))**2))

    # mean deviation
    mean = np.mean(abs(x - np.mean(x)))

    # trimmed standard deviation
    # discarding the smallest 0.1n/2 and largest 0.1n/2
    new_x = np.sort(x)
    new_x = new_x[int(0.1*n/2):int(n - 0.1*n/2)]
    trimmed_S = np.sqrt(divisor * np.sum((new_x - np.mean(new_x))**2))

    return capital_S, mean, trimmed_S      You, 2 hours ago • question1 …
```

Defining the function to compute the estimators for N(0,1):

```python
n = 20

def normal(n, iter=5000):

    # creating lists to store the estimators
    capital_Ss = []
    means = []
    trimmed_Ss = []

    for i in range(iter):

        # drawing a sample of size n from N(0,1)
        x = np.random.normal(0,1,n)

        # computing the estimators
        capital_S, mean, trimmed_S = myEstimators(n,x)

        capital_Ss.append(capital_S)
        means.append(mean)
        trimmed_Ss.append(trimmed_S)
    return capital_Ss, means, trimmed_Ss

capital_Ss, means, trimmed_Ss = normal(n)      You, 2 hours
```

Plotting the values:

```python
# plotting the sampling distribution of the estimators
fig, ax = plt.subplots(1,3,figsize=(15,5))
ax[0].hist(capital_Ss, bins=50, density=True)
ax[0].set_title('Sampling Distribution of Capital S')
ax[1].hist(means, bins=50, density=True)
ax[1].set_title('Sampling Distribution of Mean Deviations')
ax[2].hist(trimmed_Ss, bins=50, density=True)
ax[2].set_title('Sampling Distribution of Trimmed S')
plt.suptitle('Sampling Distribution of Estimators for N(0,1)')
plt.show()      You, 3 hours ago • question1
```

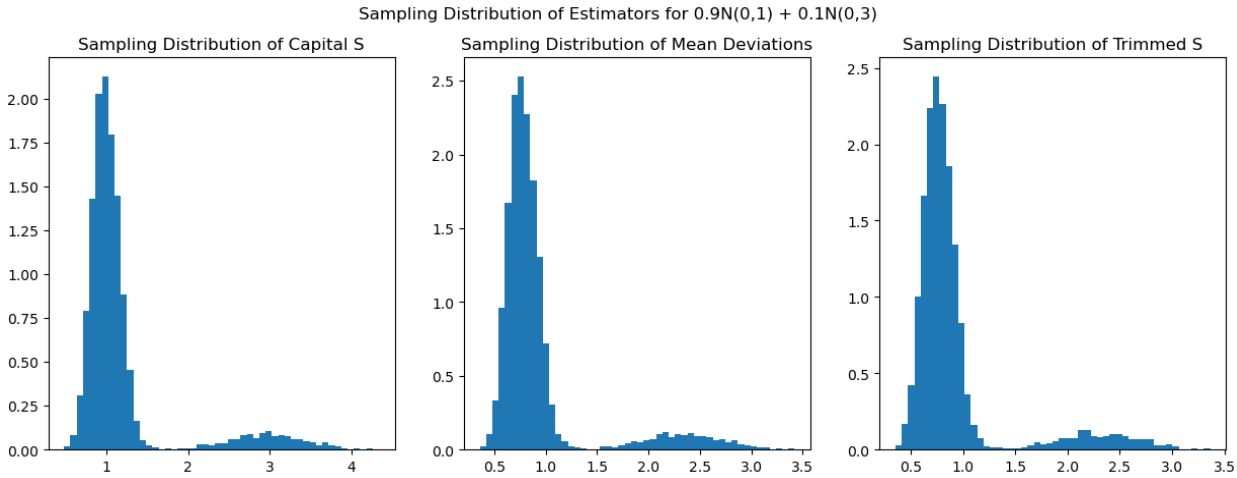


Defining the function for Mixture of Two Normals: 0.9N(0,1) + 0.1N(0,1)

```python
def mixture_of_normals(var1, var2, n, iter=5000):
    capital_Ss = []
    means = []
    trimmed_Ss = []
    for i in range(iter):

        # selecting N(0,1) with probability 0.9 and N(0,3) with probability 0.1
        epsilon = 0.1
        if np.random.uniform() < epsilon:
            x = np.random.normal(0, var2, n)
        else:
            x = np.random.normal(0, var1, n)

        capital_S, mean, trimmed_S = myEstimators(n,x)
        capital_Ss.append(capital_S)
        means.append(mean)
        trimmed_Ss.append(trimmed_S)
    return capital_Ss, means, trimmed_Ss
```

Plotting the values:

```
capital_Ss, means, trimmed_Ss = mixture_of_normals(1, 3, n)
fig, ax = plt.subplots(1,3,figsize=(15,5))
ax[0].hist(capital_Ss, bins=50, density=True)
ax[0].set_title('Sampling Distribution of Capital S')
ax[1].hist(means, bins=50, density=True)
ax[1].set_title('Sampling Distribution of Mean Deviations')
ax[2].hist(trimmed_Ss, bins=50, density=True)
ax[2].set_title('Sampling Distribution of Trimmed S')
plt.suptitle('Sampling Distribution of Estimators for 0.9N(0,1) + 0.1N(0,3)')
plt.show()
```

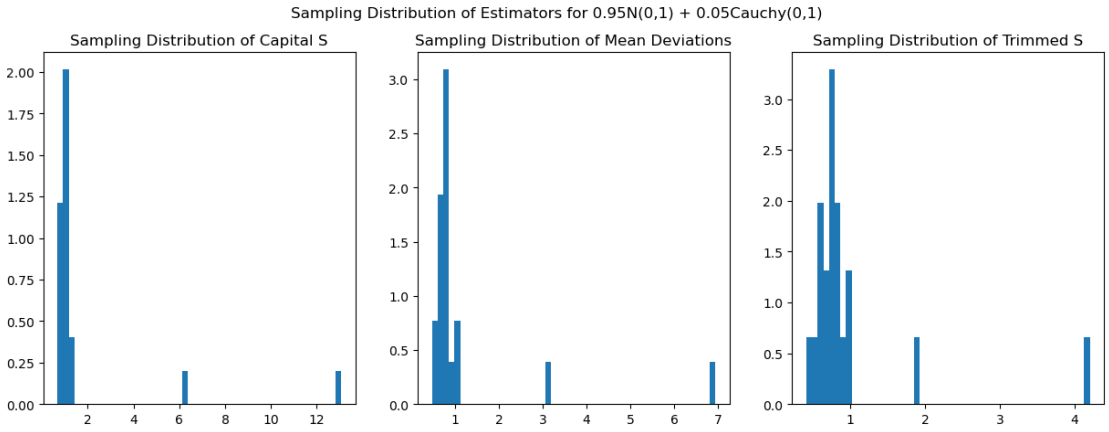


Defining the function for the Mixture of a Normal and a Cauchy: 0.95N(0,1) + 0.05C(0,1)

```
def normal_plus_cauchy(var1, iter=5000):
    capital_Ss = []
    means = []
    trimmed_Ss = []
    for i in range(iter):

        epsilon = 0.05
        if np.random.uniform() < epsilon:
            x = np.random.standard_cauchy(n)
        else:
            x = np.random.normal(0, var1, n)

        capital_S, mean, trimmed_S = myEstimators(n,x)
        capital_Ss.append(capital_S)
        means.append(mean)
        trimmed_Ss.append(trimmed_S)

    return capital_Ss, means, trimmed_Ss
```

Plotting the values:

```
capital_Ss, means, trimmed_Ss = normal_plus_cauchy(1, n)
fig, ax = plt.subplots(1,3,figsize=(15,5))
ax[0].hist(capital_Ss, bins=50, density=True)
ax[0].set_title('Sampling Distribution of Capital S')
ax[1].hist(means, bins=50, density=True)
ax[1].set_title('Sampling Distribution of Mean Deviations')
ax[2].hist(trimmed_Ss, bins=50, density=True)
ax[2].set_title('Sampling Distribution of Trimmed S')
plt.suptitle('Sampling Distribution of Estimators for 0.95N(0,1) + 0.05Cauchy(0,1)')
plt.show()
```



2. Computing the Bias and MSE for all three Estimators

```python
# computing the bias and mse of the estimators
def bias(estimator, true_value):
    return np.mean(estimator) - true_value

def mse(estimator, true_value):
    return np.var(estimator) + (np.mean(estimator) - true_value)**2
```

Normal Distribution:

	Sample Standard Deviation	**Mean Deviation**	**Trimmed Standard Deviation**
Bias	-0.0136797945743 73627	-0.2221524728723 8782	-0.2325980852717 8905
Mean Squared Error	0.0262095008122 2507	0.0673981380828 61	0.07318563277916 038

Mixture of Two Normals:

	Sample Standard Deviation	**Mean Deviation**	**Trimmed Standard Deviation**
Bias	0.1907542327956 3175	-0.0618431333824 12985	-0.0741528017865 1373
Mean Squared Error	0.4347608837805 74	0.2568051550083 415	0.2545556626254 3544

Normal + Cauchy:

	Sample Standard Deviation	**Mean Deviation**	**Trimmed Standard Deviation**
Bias	-0.0489511443647 34134	-0.245690389485 47195	-0.252542792560 95613
Mean Squared Error	0.0270675561477 09962	0.0790549808160 297	0.0841429216462 0482

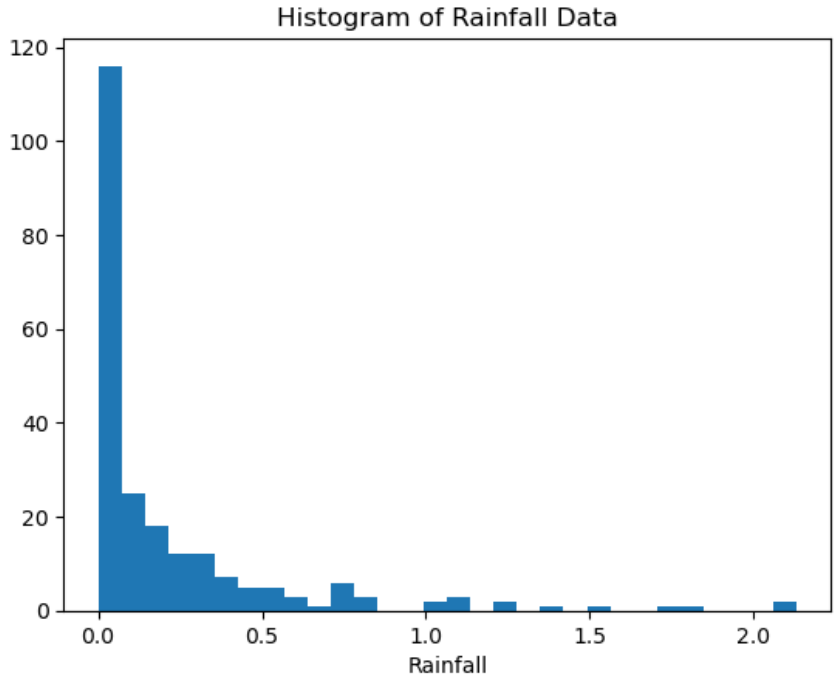
3. Observations

Among the three estimators, the Sample Standard Deviation (S) tends to exhibit bias, especially in the presence of outliers and deviations from normality, resulting in higher variance and mean squared error (MSE). Mean Deviation (M) generally demonstrates lower bias compared to S across various data distributions and offers moderate resilience against outliers. Trimmed Standard Deviation typically features even lower bias than both S and M and is highly robust against outliers and non-normal data, making it a precise choice, often with the lowest variance and MSE among the three estimators. Hence, the choice and properties of estimators should be considered in relation to the data distribution available for inspection and the required estimation goals.

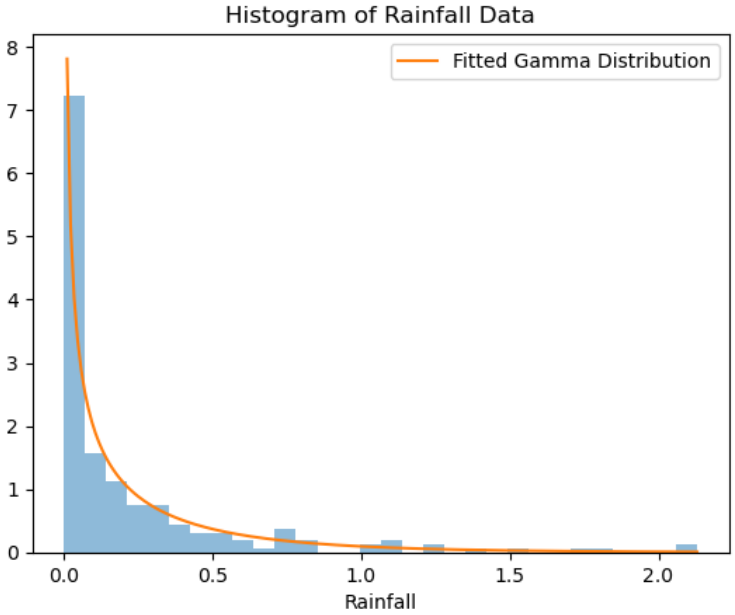
Question 2

1. Fitting a Gamma distribution to the data:

The Data:



With fitted gamma plot:



a. **Estimating the parameters using the Method of Moments:**

```
# estimating the parameters using the method of moments
mean = np.mean(rainfall_data_flattened)
var = np.var(rainfall_data_flattened)
shape = mean**2/var
scale = var/mean
print("Shape:",shape)
print("Scale:",scale)      You, 3 hours ago • updated f
```

With the above, we get ***0.3758789321633129*** for *alpha* and ***0.5966781904246276*** for *beta*.

Estimating the parameters using the Method of Maximum Likelihood with SciPy because the inbuilt function uses MLE to fit the curve to the data:

```
# estimating the parameters using the method of maximum likelihood using scipy
alpha, loc, beta = stats.gamma.fit(rainfall_data_flattened, floc=0)
print("Alpha:",alpha)
print("Beta:",beta)      You, 5 hours ago • plots of alpha and beta …
```

With the above, we get ***0.43912250445350176*** for *alpha* and ***0.5107430359121928*** for *beta*.

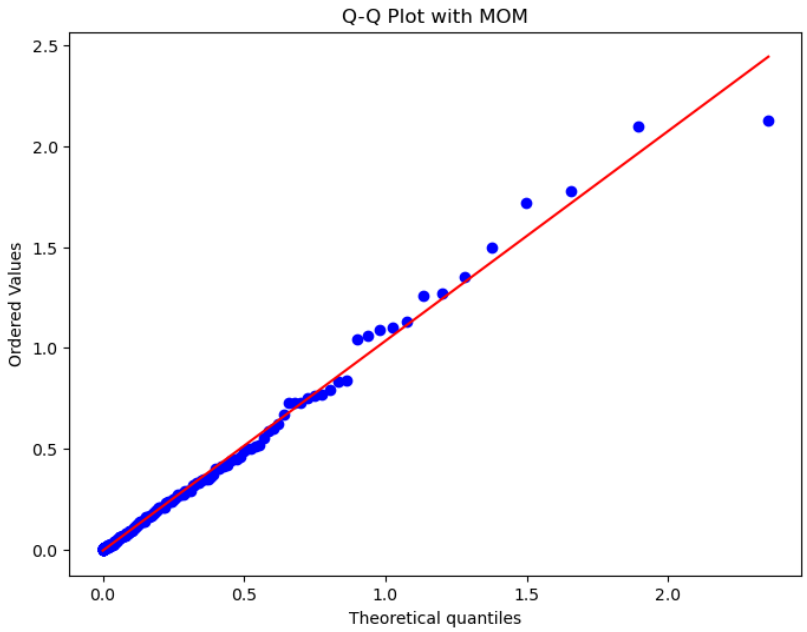
Assessing the Goodness of Fit of the Gamma Distribution to the Data:

This can be inspected visually by looking at the overlay of the curve on the actual data as plotted in the initial part of this question. We can proceed to visualise how well the gamma distribution fits the data by using a QQ Plot.

Using the estimates from the Method of Moments:

```
shape_value, scale_value = shape, scale
print(shape_value, scale_value)

# turning rainfall data to numpy array
rainfall_data = np.array(rainfall_data)

rainfall_data_flattened = rainfall_data.flatten()

# Create the Q-Q plot using stats.probplot
plt.figure(figsize=(8, 6))
res = stats.probplot(rainfall_data_flattened, dist=stats.gamma, sparams=(shape_value, 0, scale_value), plot=plt)
plt.title('Q-Q Plot with MOM')
plt.show()
```

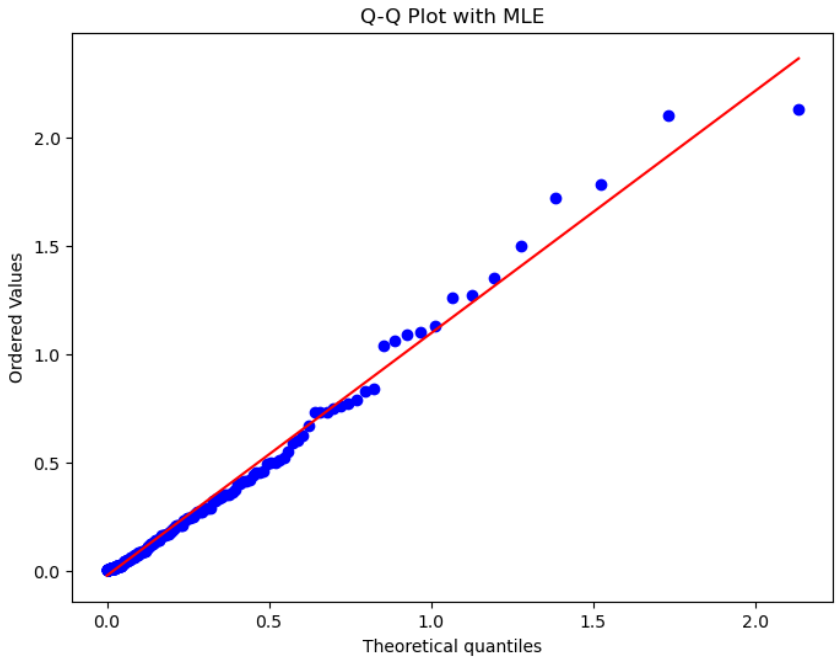
Resulting QQ Plot:



Using estimates from the Method of Maximum Likelihood:

```
shape_value, scale_value = alpha, beta
plt.figure(figsize=(8, 6))
res = stats.probplot(rainfall_data_flattened, dist=stats.gamma, sparams=(shape_value, 0, scale_value), plot=plt)
plt.title('Q-Q Plot with MLE')
plt.show()
```

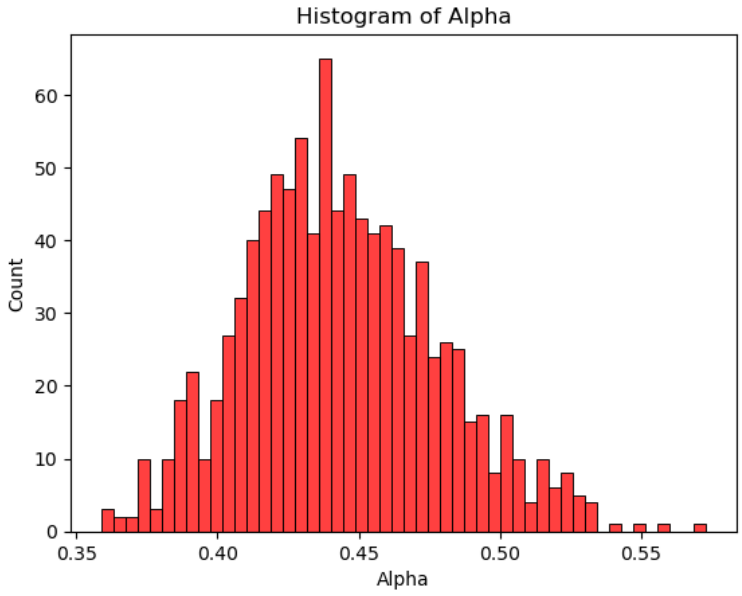
Resulting QQ Plot:

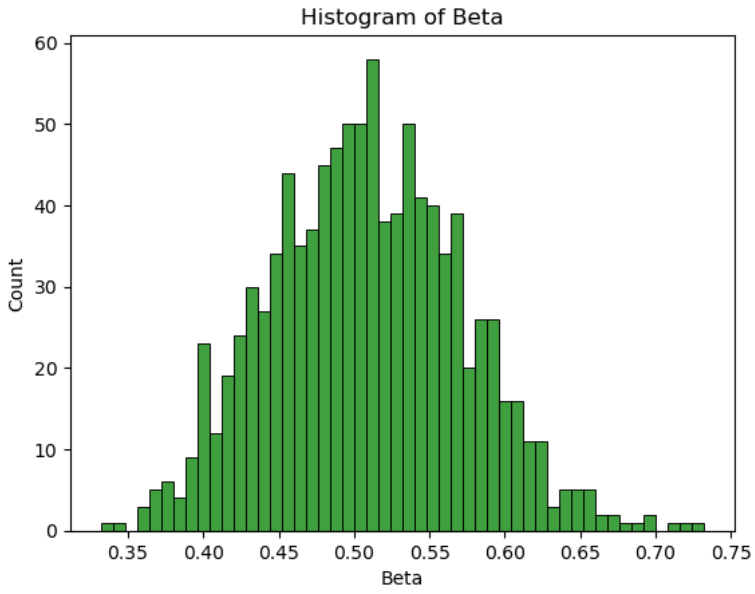


4. Parametric Bootstrap Method

a. Using Maximum Likelihood Estimators

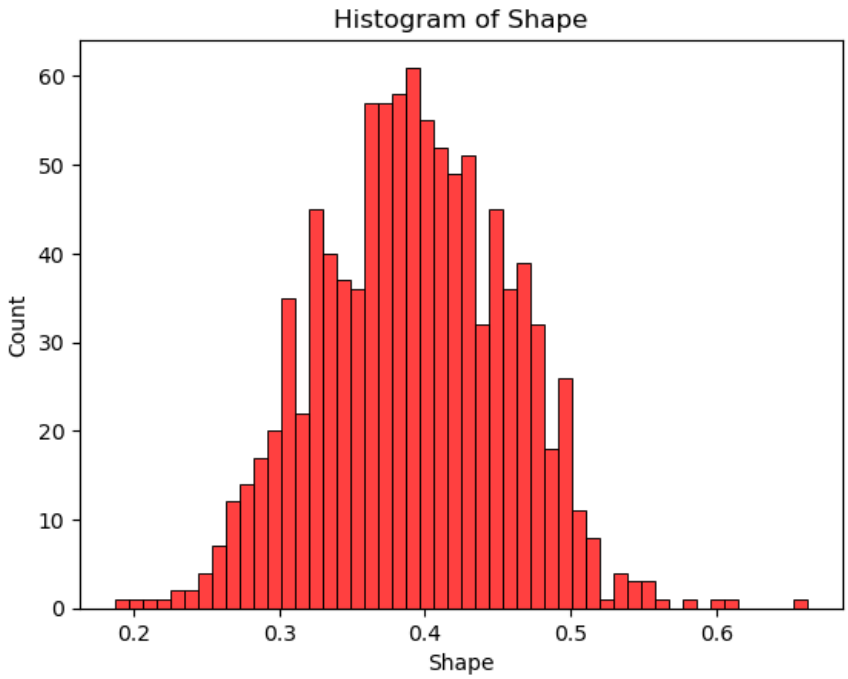
```python
# finding the mle of the sample using scipy
alphas = []
betas = []
for i in range(1000):
    sample = stats.gamma.rvs(alpha, loc, beta, size=226)
    alpha_sample, loc_sample, beta_sample = stats.gamma.fit(sample, floc=0)
    alphas.append(alpha_sample)
    betas.append(beta_sample)


# plotting the histogram of the mle of the sample
sns.histplot(alphas, bins=50, color='red')
plt.title('Histogram of Alpha')
plt.xlabel('Alpha')
plt.show()

sns.histplot(betas, bins=50, color='green')
plt.title('Histogram of Beta')
plt.xlabel('Beta')
plt.show()
```

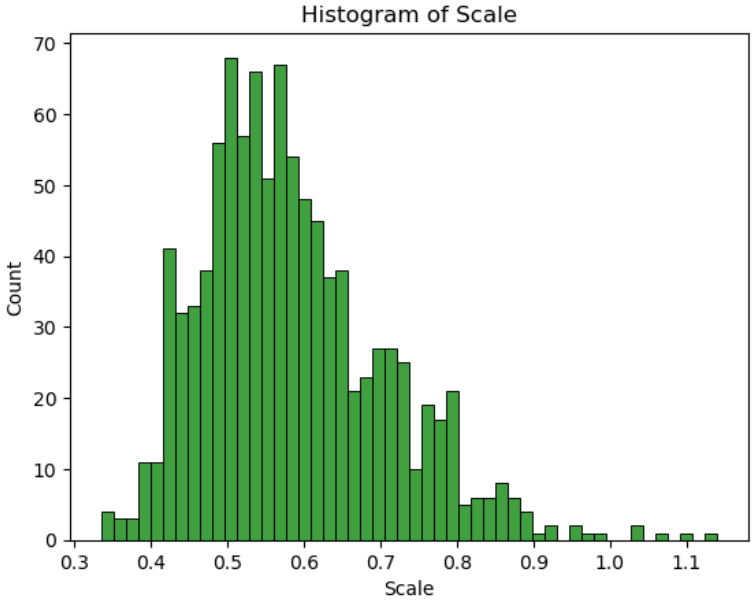




b. Using Methods of Moment Estimators

```python
# finding the mom estimators of the sample
shapes = []
scales = []
for i in range(1000):
    sample2 = stats.gamma.rvs(shape, loc, scale, size=226)
    mean_sample = np.mean(sample2)
    var_sample = np.var(sample2)
    shape_sample = mean_sample**2/var_sample
    scale_sample = var_sample/mean_sample
    shapes.append(shape_sample)
    scales.append(scale_sample)      You, 5 hours ago • plot for mom samp

# plotting the histogram of the mom estimators of the sample
sns.histplot(shapes, bins=50, color='red')
plt.title('Histogram of Shape')
plt.xlabel('Shape')
plt.show()

sns.histplot(scales, bins=50, color='green')
plt.title('Histogram of Scale')
plt.xlabel('Scale')
plt.show()
```


Histogram of Shape
Count
Shape
0
10
20
30
40
50
60
0.2
0.3
0.4
0.5
0.6


Histogram of Scale
Count
Scale
0
10
20
30
40
50
60
70
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
1.1

c. Comparing Bias and MSEs

```
# comparing the mle and mom estimators using the bias and mse
print("Bias of Alpha MLE:", np.mean(alphas) - shape)
print("Bias of Beta MLE:", np.mean(betas) - scale)
print("Bias of Alpha MOM:", np.mean(shapes) - shape)
print("Bias of Beta MOM:", np.mean(scales) - scale)


# using var plus bias squared to calculate mse
print("MSE of Alpha MLE:", np.var(alphas) + (np.mean(alphas) - shape)**2)
print("MSE of Beta MLE:", np.var(betas) + (np.mean(betas) - scale)**2)
print("MSE of Alpha MOM:", np.var(shapes) + (np.mean(shapes) - shape)**2)
print("MSE of Beta MOM:", np.var(scales) + (np.mean(scales) - scale)**2)
```

	MLEs	**MoMs**
Bias - Alpha	0.0038493895265862554	0.017758482736227654
Bias - Beta	-0.005295450033847349	-0.010900517943966737
MSE - Alpha	0.0010812793249852477	0.006262273554637538
MSE - Beta	0.003407729935146985 4	0.014519833341281338

Based on the information in the table above, it is evident that Maximum Likelihood Estimation (MLE) performs better than the Method of Moments (MoM) in estimating the parameters. This conclusion is supported by two key observations:

Bias Comparison: The bias values for both Alpha and Beta are lower for MLE compared to MoM. This indicates that MLE provides more accurate parameter estimates, as lower bias values suggest estimates closer to the true values.

Mean Squared Error (MSE) Comparison: The MSE values for both Alpha and Beta are also lower for MLE compared to MoM. This suggests that MLE is more precise in estimating the parameters, as lower MSE values indicate estimates with smaller overall errors.