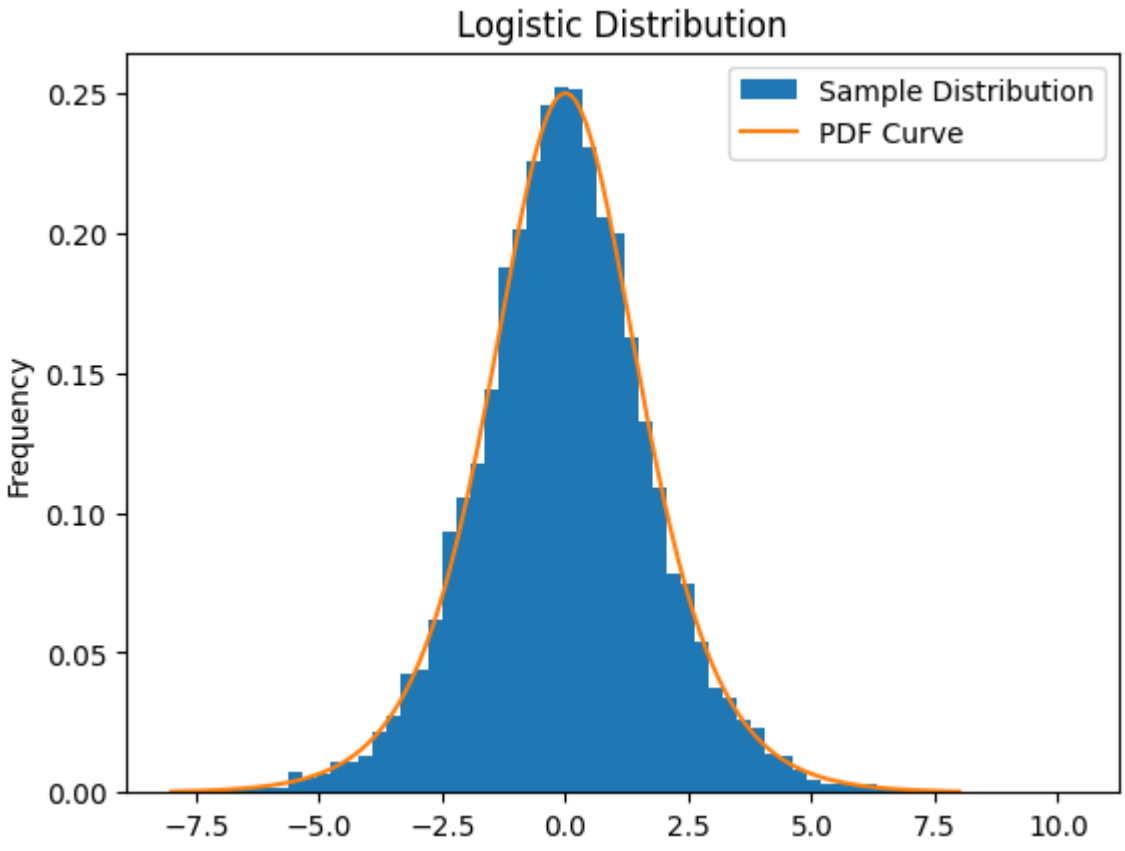
In [64]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
import random
```

Problem 1. Inverse Transform Sampling

(a)Logistic distribution

In [62]:
```
uniform_samples = np.random.uniform(0, 1, 10000)
transformed_samples = -np.log((1 / uniform_samples) - 1)
x_values = np.linspace(-8, 8, 1000)
pdf_values = np.exp(-x_values) / ((1 + np.exp(-x_values))**2)
plt.hist(transformed_samples, bins=64, density=True, label="Sample Distribution"
plt.plot(x_values, pdf_values,label="PDF Curve")
plt.ylabel('Frequency')
plt.title('Logistic Distribution')
plt.legend()
plt.show()
print("1")
```

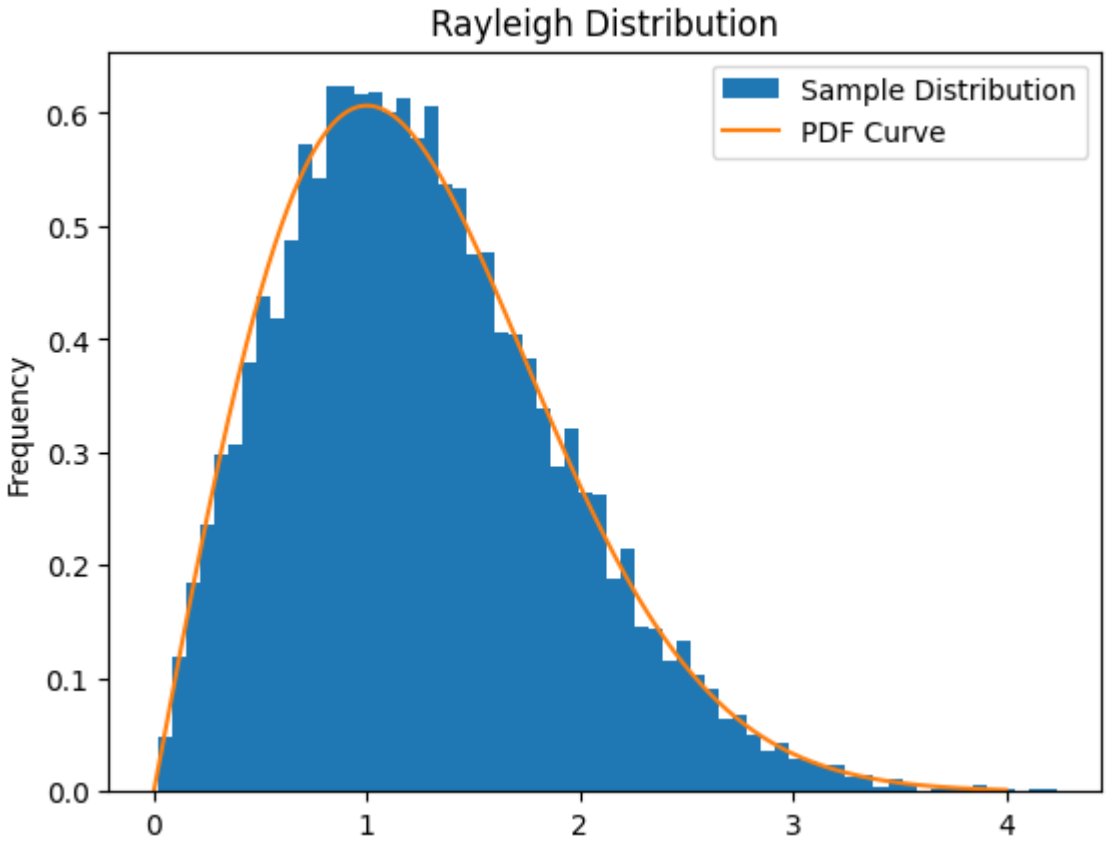


1

(b) Rayleigh Distribution

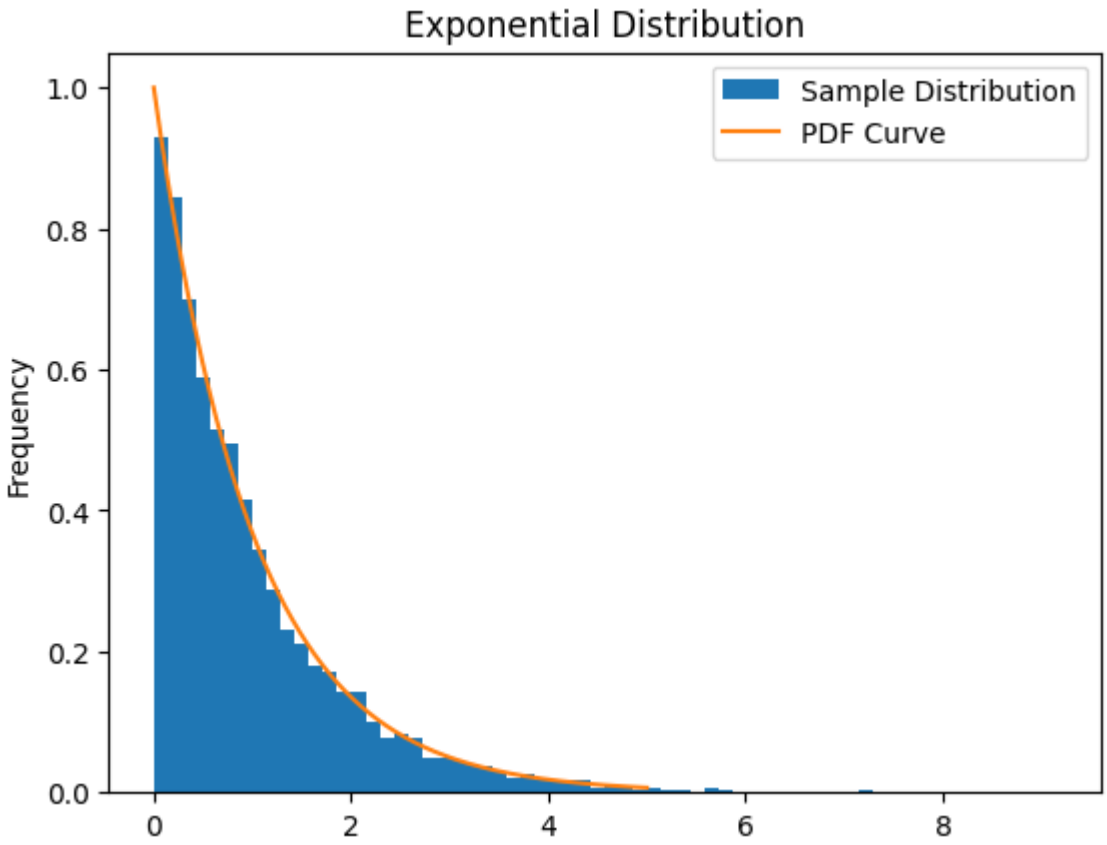
In [46]:
```
uniform_samples = np.random.uniform(0, 1, 10000)
transformed_samples = np.sqrt(-2 * np.log(1 - uniform_samples))
x_values = np.linspace(0, 4, 1000)
pdf_values = x_values * np.exp(-1/2 * (x_values**2))
```

```
plt.hist(transformed_samples, bins=64, density=True, label="Sample Distribution"
plt.plot(x_values, pdf_values, label="PDF Curve")
plt.ylabel('Frequency')
plt.title('Rayleigh Distribution')
plt.legend()
plt.show()
```



(c) Exponential Distribution

In [47]:
```
uniform_samples = np.random.uniform(0, 1, 10000)
transformed_samples = -np.log(1 - uniform_samples)
x_values = np.linspace(0, 5, 1000)
pdf_values = np.exp(-x_values)
plt.hist(transformed_samples, bins=64, density=True, label="Sample Distribution"
plt.plot(x_values, pdf_values, label="PDF Curve")
plt.ylabel('Frequency')
plt.title('Exponential Distribution')
plt.legend()
plt.show()
```



Problem 2. Samples Discrete Distributions

(a) Bern(0.5)

In [80]:
```
N = 10000
p = 0.5

# Generate Bernoulli distributed samples directly
bern_samples = np.random.choice([0, 1], size=N, p=[1-p, p])
```

(b) Bin(20, 0.5)

In [81]:
```
N = 10000
n = 20
p = 0.5

# Generate samples using Bernoulli trials
bernoulli_trials = np.random.uniform(0, 1, (n, N)) > p
binomial_samples = bernoulli_trials.sum(axis=0)
```

(c) Geom(0.5)

In [82]:
```
N, p = 100000, 0.5
geom = np.zeros(N)

for i in range(N):
    u = np.random.random()  # Generate a uniform random number
    k = 0
    while u > 1 - (1 - p) ** (k + 1):
```

```
        k += 1
    geom[i] = k
```

(d) Nbin(10, 0.5)

In [83]:
```
N, n, p = 10000, 10, 0.5
Ngeom = np.zeros(N)

for i in range(N):
    for _ in range(n):
        failures = 0
        while np.random.random() >= p:
            failures += 1
        Ngeom[i] += failures
```

(e)Pois(1)

In [84]:
```
N = 10000
lam = 1

poisson_samples = np.zeros(N)

for i in range(N):
    L = np.exp(-lam)
    k = 0
    p = 1
    while p > L:
        k += 1
        p *= np.random.uniform(0, 1)
    poisson_samples[i] = k - 1
```

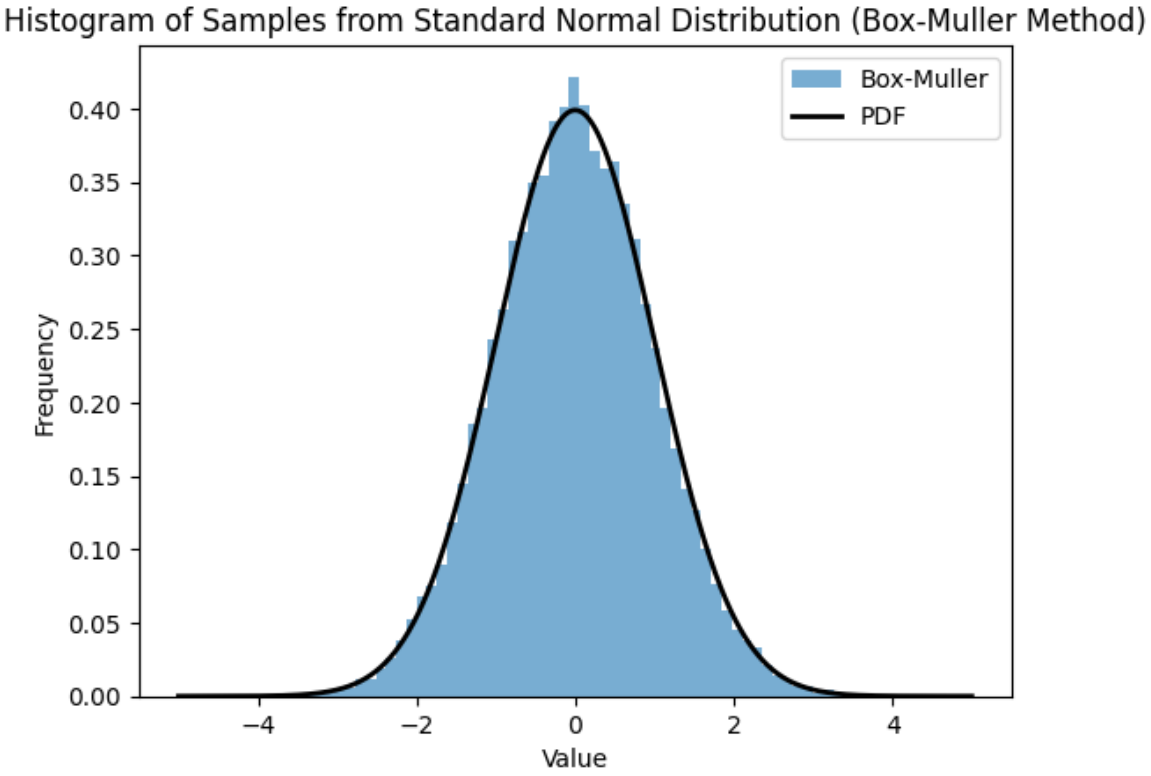
Problem 3

(a) Box-Muller

In [52]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
```

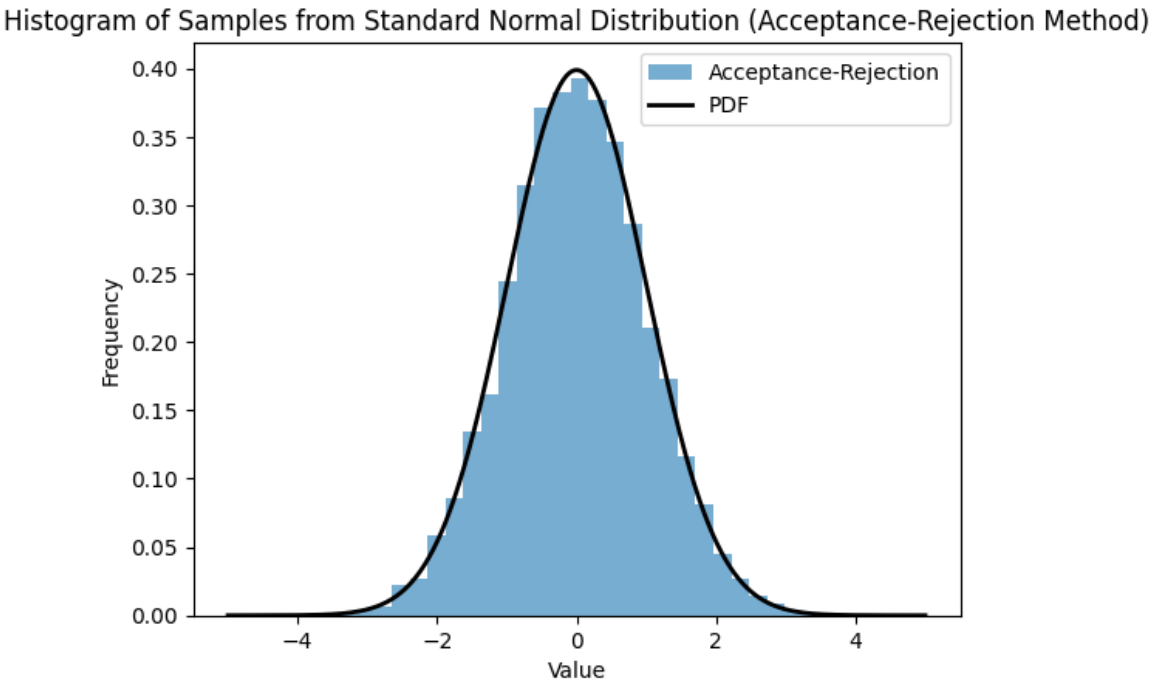
In [53]:
```
def box_muller(n):
    u1, u2 = np.random.uniform(size=n), np.random.uniform(size=n)
    z1 = np.sqrt(-2 * np.log(u1)) * np.cos(2 * np.pi * u2)
    z2 = np.sqrt(-2 * np.log(u1)) * np.sin(2 * np.pi * u2)
    return np.concatenate([z1, z2])

samples_box_muller = box_muller(10000)

plt.hist(samples_box_muller, bins=60, density=True, alpha=0.6, label="Box-Muller
x_values = np.linspace(-5, 5, 1000)
pdf_values = norm.pdf(x_values, 0, 1)
plt.plot(x_values, pdf_values, 'k', linewidth=2, label="PDF")
plt.title("Histogram of Samples from Standard Normal Distribution (Box-Muller Me
plt.xlabel("Value")
plt.ylabel("Frequency")
```

```
plt.legend()
plt.show()
```



(b) Acceptance-Rejection

In [54]:
```
def acceptance_rejection(n):
    samples = []
    while len(samples) < n:
        x = np.random.uniform(-4, 4)
        y = np.random.uniform(0, 0.4)
        pdf = (1/np.sqrt(2*np.pi)) * np.exp(-0.5*x**2)
        if y < pdf:
            samples.append(x)
    return np.array(samples)

samples_acc_rej = acceptance_rejection(10000)

plt.hist(samples_acc_rej, bins=30, density=True, alpha=0.6, label="Acceptance-Re
plt.plot(x_values, pdf_values, 'k', linewidth=2, label="PDF")
plt.title("Histogram of Samples from Standard Normal Distribution (Acceptance-Re
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.legend()
plt.show()
```



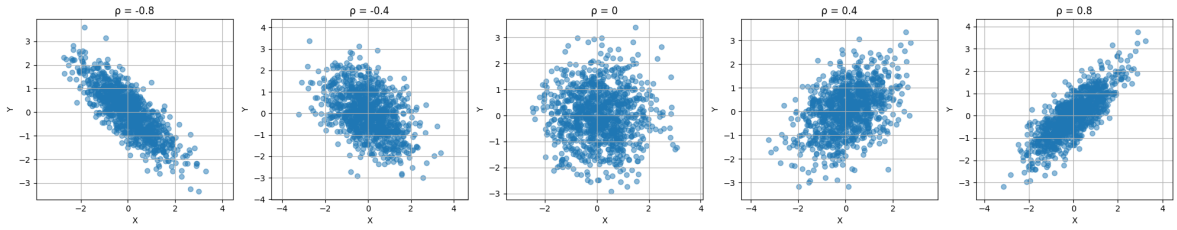
(c) Compare

The Box-Muller method is favored for its straightforward approach to creating standard normal random variables from a uniform distribution, which is especially useful when dealing with standard normal distributions. However, this feature is also a drawback because it means that Box-Muller can't provide the variety of distributions some more complex scenarios might call for. Howeverly, the Acceptance-Rejection method ifavoreded for its adaptability. It allows us to draw samples from a broarangeum of probability distributions, including those that might be challenging to sample from directly due to their complexity. The key to its success lies in the relationship between the proposal and target distributions. For the method to be efficient, the proposal distribution must be a good fit over the entire range of the target distribution. If the fit is too tight, with the optimal constant near one, sampling becomes a tough task, and if the fit is too loose, indicated by a constant near zero, the acceptance rate plummets, which means a lot of generated samples end up being discaed.h.

(d) bivariate

In [86]:
```
def generate_bivariate_normal_samples(rho, n_samples):
    Z = np.random.normal(0, 1, n_samples)
    W = np.random.normal(0, 1, n_samples)
    X = Z
    Y = rho * Z + np.sqrt(1 - rho**2) * W
    return X, Y
rhos = [-0.8, -0.4, 0, 0.4, 0.8]
n_samples = 1000
plt.figure(figsize=(20, 4))

for i, rho in enumerate(rhos, 1):
    plt.subplot(1, 5, i)
    X_samples, Y_samples = generate_bivariate_normal_samples(rho, n_samples)
```

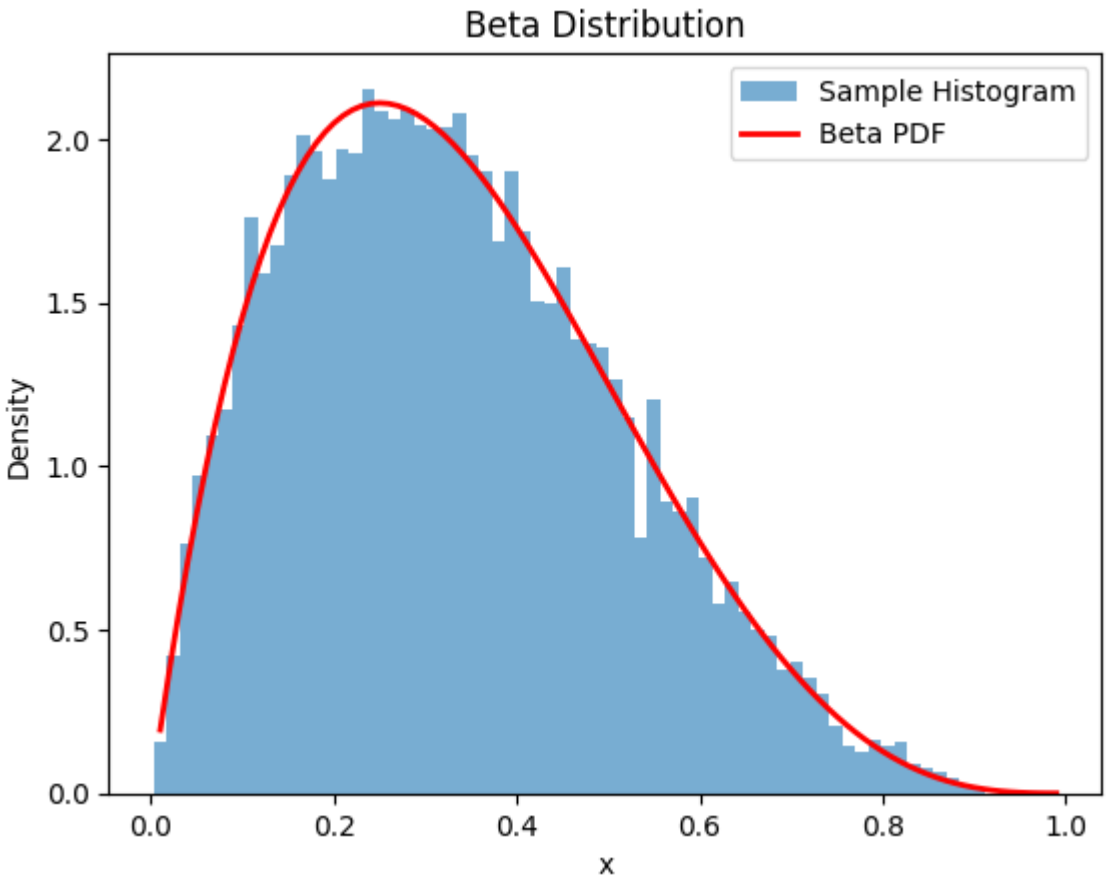
```
    plt.scatter(X_samples, Y_samples, alpha=0.5)
    plt.title("ρ = {}".format(rho))
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.grid(True)
    plt.axis('equal')

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



Problem 4

(a) Acceptance-Rejection

In [56]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import beta
import random
import math

def beta_acceptance_rejection(a, b, n):
    i = 0
    result = []
    while i < n:
        x = random.uniform(0, 1)
        u = random.uniform(0, 1)
        beta_value = math.factorial(a-1) * math.factorial(b-1) / math.factorial(
        c = 1 / (beta_value * pow(((a-1)/(a+b-2)), a-1) * pow(((b-1)/(a+b-2)), b
        if u <= pow(x, a-1) * pow(1-x, b-1) / (c * pow(((a-1)/(a+b-2)), a-1) * p
            result.append(x)
            i += 1
    return result

a = 2
b = 4
n_reduced = 10000

samples_reduced = beta_acceptance_rejection(a, b, n_reduced)

plt.hist(samples_reduced, bins=64, density=True, alpha=0.6, label="Sample Histog

x = np.arange(0.01, 1, 0.01)
y = beta.pdf(x, a, b)
plt.plot(x, y, 'r-', lw=2, label='Beta PDF')
plt.title('Beta Distribution')
plt.xlabel('x')
plt.ylabel('Density')
plt.legend()

plt.show()
```



(b) Monte Carlo

In [57]:
```
n_points = 1000000
x_random = np.random.uniform(0, 1, n_points)
integrand_values = 4 / (1 + x_random**2)
integral_estimate = np.mean(integrand_values)
integral_estimate
```

Out[57]: 3.1411597720834674

(c) Monte Carlo

In [58]:
```
def integrand(x):
    return np.sqrt(x + np.sqrt(x + np.sqrt(x + np.sqrt(x))))
n_points = 1000000
x_random = np.random.uniform(0, 4, n_points)
integrand_values = integrand(x_random)
integral_estimate = 4 * np.mean(integrand_values)
integral_estimate
```

Out[58]: 7.676783736266756

(d) Monte Carlo

In [59]:
```
n_points = 10000000
x_random = np.random.uniform(-1, 1, n_points)
y_random = np.random.uniform(-1, 1, n_points)
inside_circle = (x_random ** 2 + y_random ** 2) <= 1
```

```
pi_estimate = 4 * np.mean(inside_circle)
pi_estimate
```

Out[59]: 3.1417264

(e) importance sampling

In [60]:
```
import numpy as np
import math

def importance_sampling(n):
    sample = np.random.normal(8, 1, n)
    total_sum = np.sum(np.exp(8*(4-sample[sample > 8])))
    return total_sum / n

n = 10000000
prob_with_importance_sampling = importance_sampling(n)
print(f" importance sampling: {prob_with_importance_sampling}")
```

```
importance sampling: 6.217982714249921e-16
```