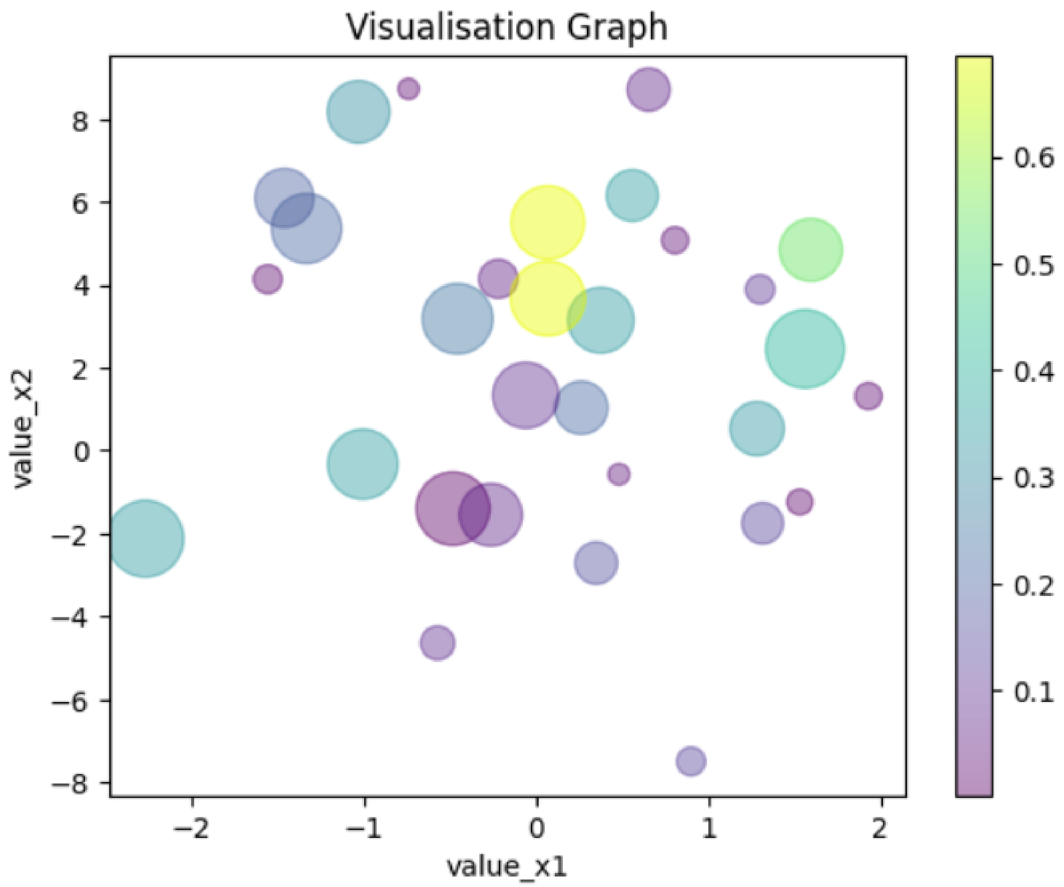
STATISTICS PROJECT 2

1.Using scipy.stats's rvs method, sample 30 tuples (x_1^i, x_2^i, x_3^i, x_4^i)$_{1≤i≤30}$ s.th. $x_1^i \sim$ Normal(0, 1) $x_2^i \sim$ Normal(2, 4) $x_3^i \sim$ Uniform(0, 1) $x_4^i = x_3^i \cdot z$ where $z \sim$ Uniform(0, 1) Using one of the visualisation techniques discussed in the lectures, plot this 4-D data. (Hint: you may find that you need to adjust some parameter(s) for your plot to be legible; if so please do it.). The four dimensions are not all independent of one another. How does this manifest itself on your plot?

```
#importing necessary Libraries
import numpy as np
import scipy.stats as stats
from scipy.stats import norm
from scipy.stats import uniform
from scipy.stats import cauchy
from scipy.stats import beta
from scipy.integrate import quad
from scipy.signal import convolve
import scipy.integrate as integrate
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
#generating 30 sample tuples with rvs method
value_x1 = norm.rvs( loc=0, scale = 1, size = 30 )
value_x2 = norm.rvs( loc=2, scale = 4, size = 30)
value_x3 = uniform.rvs( loc=0, scale = 1, size = 30)
value_z = uniform.rvs( loc=0, scale =1, size = 30)
value_x4 = value_x3 * value_z

#visualising the result
plt.scatter(value_x1, value_x2, c=value_x4, s=value_x3*800, alpha =
0.4)
plt.xlabel('value_x1')
plt.ylabel('value_x2')
plt.title('Visualisation Graph')
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7d4235088280>
```

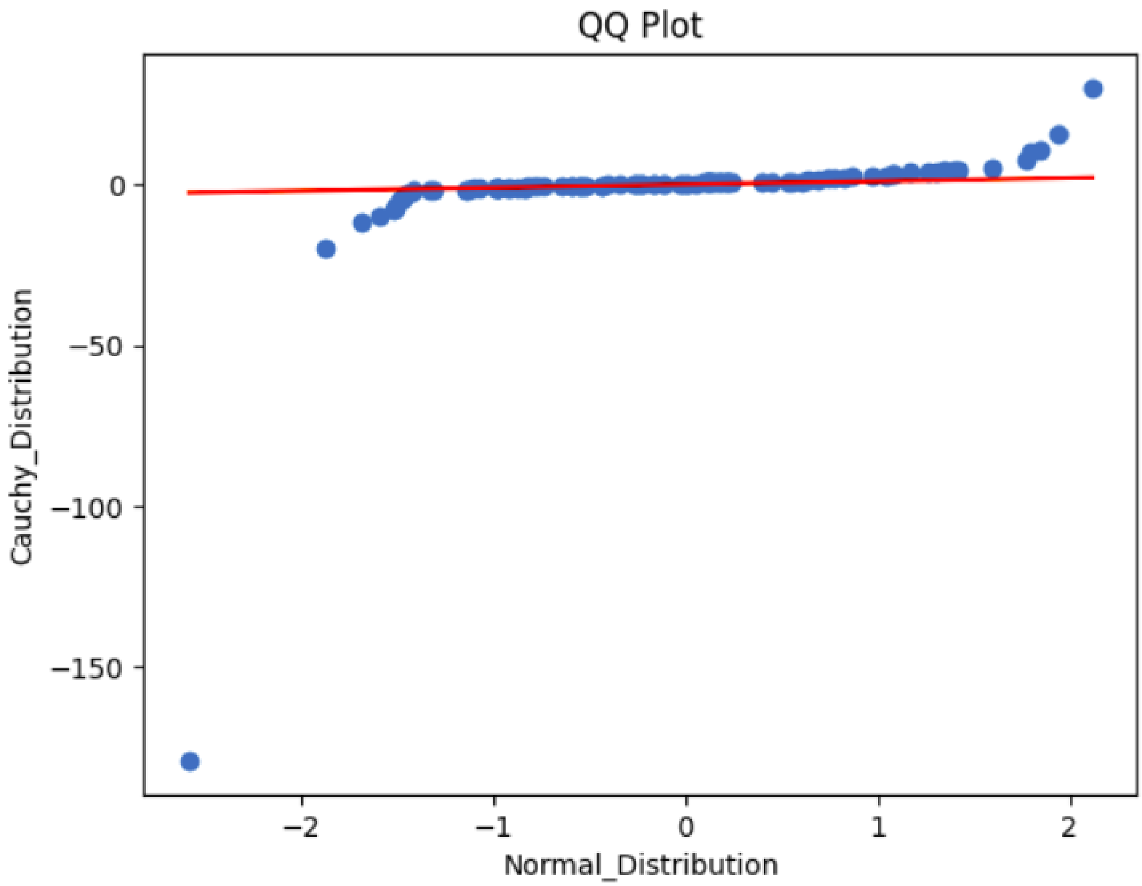


Justification: The value x1 and x2 are independent of each other, so those are plotted against x and y axis. As the x3 and x4 values depend each other, they are plotted together in the attribute space as size and color and at some points they overlap eachother.

2.Display a QQ plot for the following probability measures: the standard normal Normal(0, 1) on the x-axis and the standard Cauchy distribution Cauchy(0, 1) on the y-axis. What does the QQ plot tell us about the tails of these distributions?

```
normal_distribution = norm.rvs(0,1,100)
cauchy_distribution = cauchy.rvs(0,1,100)

percentile = np.linspace(0,100,100)
normal_percentile = np.percentile(normal_distribution, percentile)
cauchy_percentile = np.percentile(cauchy_distribution, percentile)

plt.plot(normal_percentile,cauchy_percentile, 'o')
plt.plot(normal_percentile,normal_percentile, color='red')
plt.xlabel('Normal_Distribution')
plt.ylabel('Cauchy_Distribution')
plt.title('QQ Plot')
```

```
Text(0.5, 1.0, 'QQ Plot')
```



Justification: Tails seems to be fat in the plot, bacause most of the points together peaked at the centre, and it makes the distribution narrow and not normal.

3.Recall from the lectures that if we have two probability measures P1 and P2 with respective densities f1 and f2, then the density of the sum1 P1 + P2 is given by the convolution of the two densities, viz. f1+2(t) = Z ∞ −∞ f1(x)f2(t − x) dx. In this question we consider the sum of Beta (2, 8) + Beta (8, 2). What is the support of Beta (2, 8)? What is the support of Beta (8, 2)? Therefore, what is the support of Beta (2, 8) + Beta (8, 2)?

Answer: The support of beta(2,8) is in the closed interval [0,1] and the support of beta(8,2) is also [0,1], the support of those two beta is fixed and do not change according to the parameters.

The support of the sum cannot be determined directly, if we need to know the support of their sum, then we are supposed to take the convolution of those two beta distributions.

We can use python code to find out their convolution.

```
#generating values
values = np.linspace(-0.5, 3.5, 1000)

#finding pdf's of beta distribution
pdf_2_8 = beta.pdf(values, 2, 8)
pdf_8_2 = beta.pdf(values, 8, 2)
#finding convolution of those two beta distribution
convolution = convolve(pdf_2_8, pdf_8_2, mode='full')

#checking non-zero values where convolution is grater than 0
non_zero = np.where(convolution > 0)

#creating an empty array to store values and the convolution result
values_1 = []
convolution_final = []
for i,j in enumerate(non_zero[0]):
    if j < 1000:
        values_1.append(values[j])

for i in range(1000):
    c = convolution[i]
    convolution_final.append(c)

print("Support of (2,8)+(8,2) lies in the Interval
:[",min(values_1),",",max(values_1),"]")
```

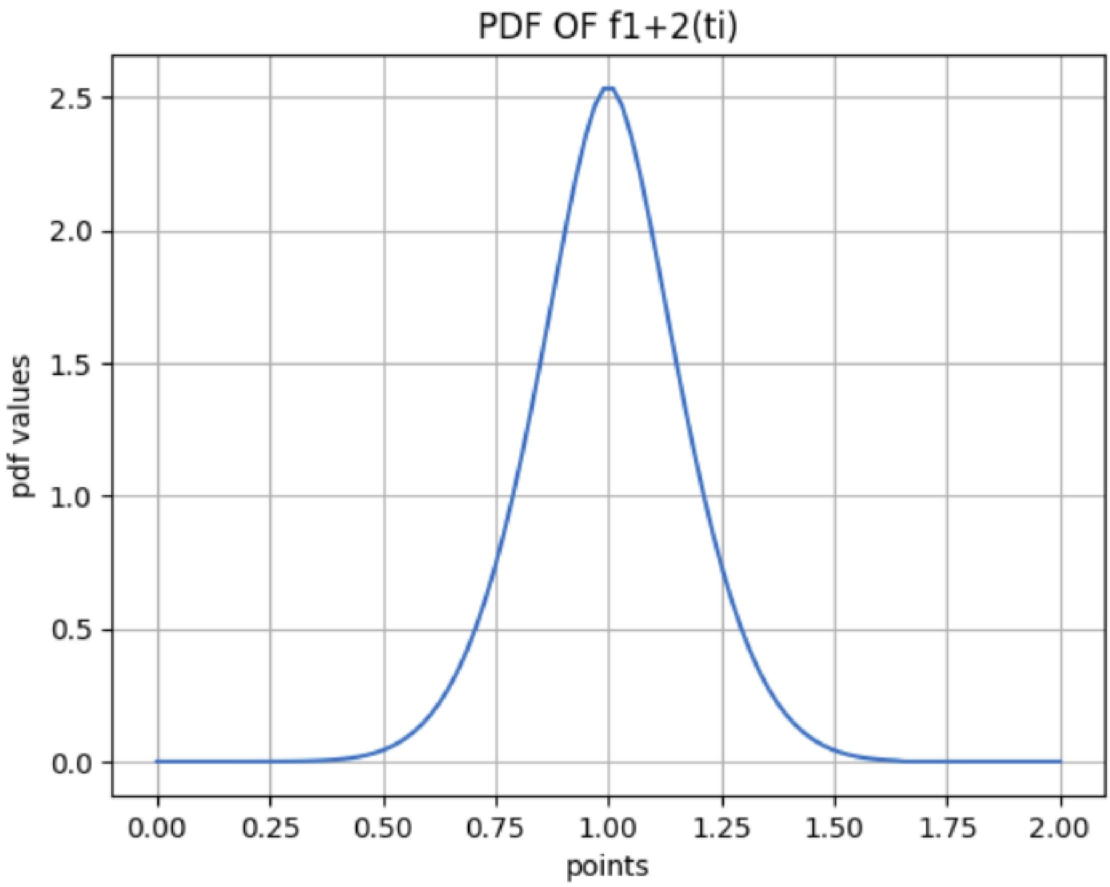
```
Support of (2,8)+(8,2) lies in the Interval :[ 0.5010010010010011 , 2.494994994994995 ]
```

Write a function which implements the integrand of the integral above, that is to say that implements f1(x)f2(t−x), where f1 is the density of Beta (2, 8) and f2 is the density of Beta (8, 2). (Hint: this function will need two arguments.)

```
#using lambda function to compute the product densities
a1,b1 = 2,8
a2,b2 = 8,2
integrand = lambda x,t : beta.pdf(x,a1,b1) * beta.pdf(t-x,a2,b2)
```

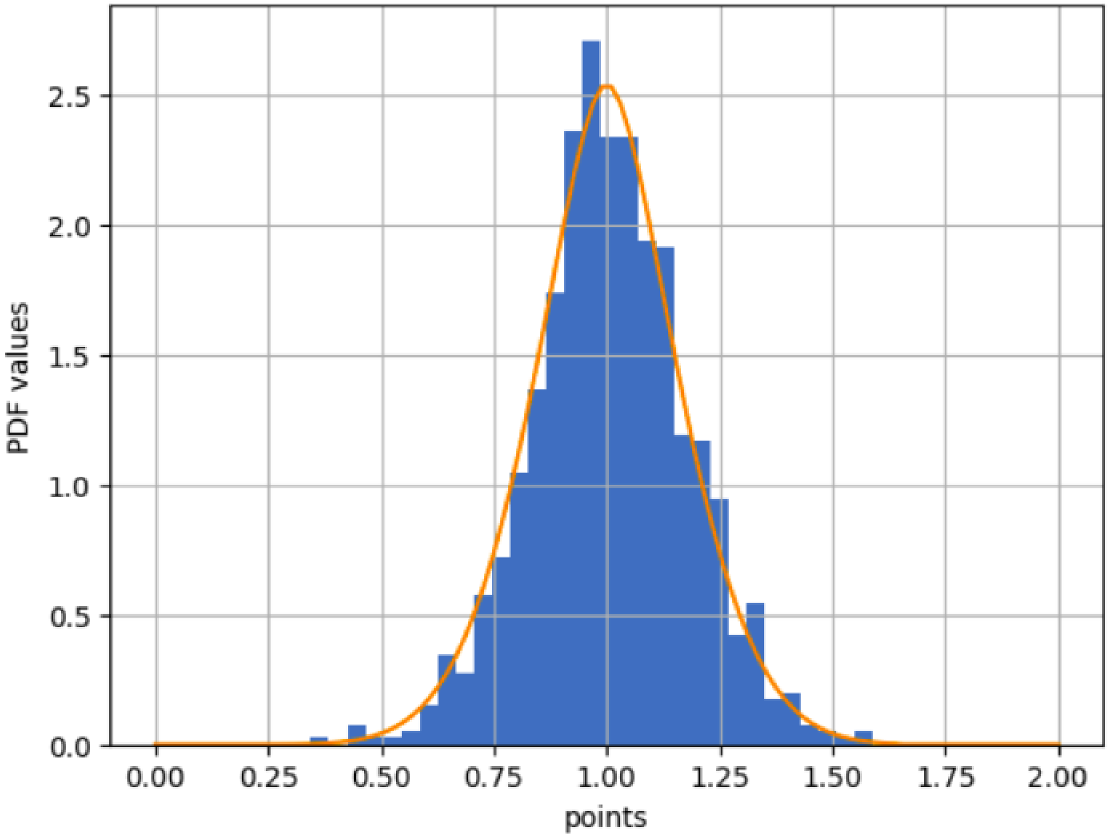
Next, generate 100 points (t1, . . . , t100) along the support of Beta (2, 8) + Beta (8, 2) (using numpy's linspace function), and using a for loop, compute the pdf f1+2(ti) at these 100 points using quad. (Hint: the documentation of quad has an example showing how to integrate a function with two arguments along its first argument.) Plot your result.

```
#generating sample points
points = np.linspace(0, 2, 100)
solution = []

#computing the pdf of f1+2(ti) using quad function
for i,j in enumerate(points):
    result, err = quad(integrand, float('-inf'), float('inf'),
args=(j,))
    solution.append(result)

#visualisation of the result
plt.plot(points,solution)
plt.title('PDF OF f1+2(ti)')
plt.xlabel('points')
plt.ylabel('pdf values')
plt.grid(True)
plt.show()
```



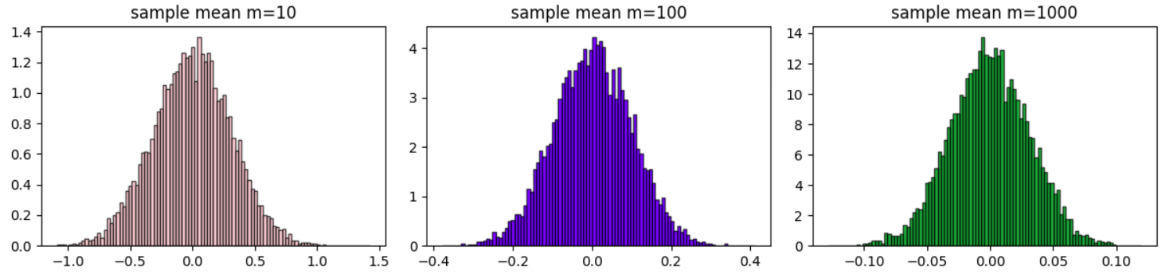
Finally, generate 10000 samples from Beta (2, 8), 10000 samples from Beta (8, 2), add them, and plot the histogram of these sums along with the pdf computed in the previous step. What do you observe?

```
#generating 10000 samples using rvs method
beta_1 = beta.rvs(2,8, size=1000)
beta_2 = beta.rvs(8,2, size=1000)

#adding the both beta values
sum_of_beta = beta_1 + beta_2

#visualisation result
plt.hist(sum_of_beta, int(np.sqrt(1000)), density = True)
plt.plot(points,solution)
plt.xlabel('points')
plt.ylabel('PDF values')
plt.grid(True)
plt.show()
```



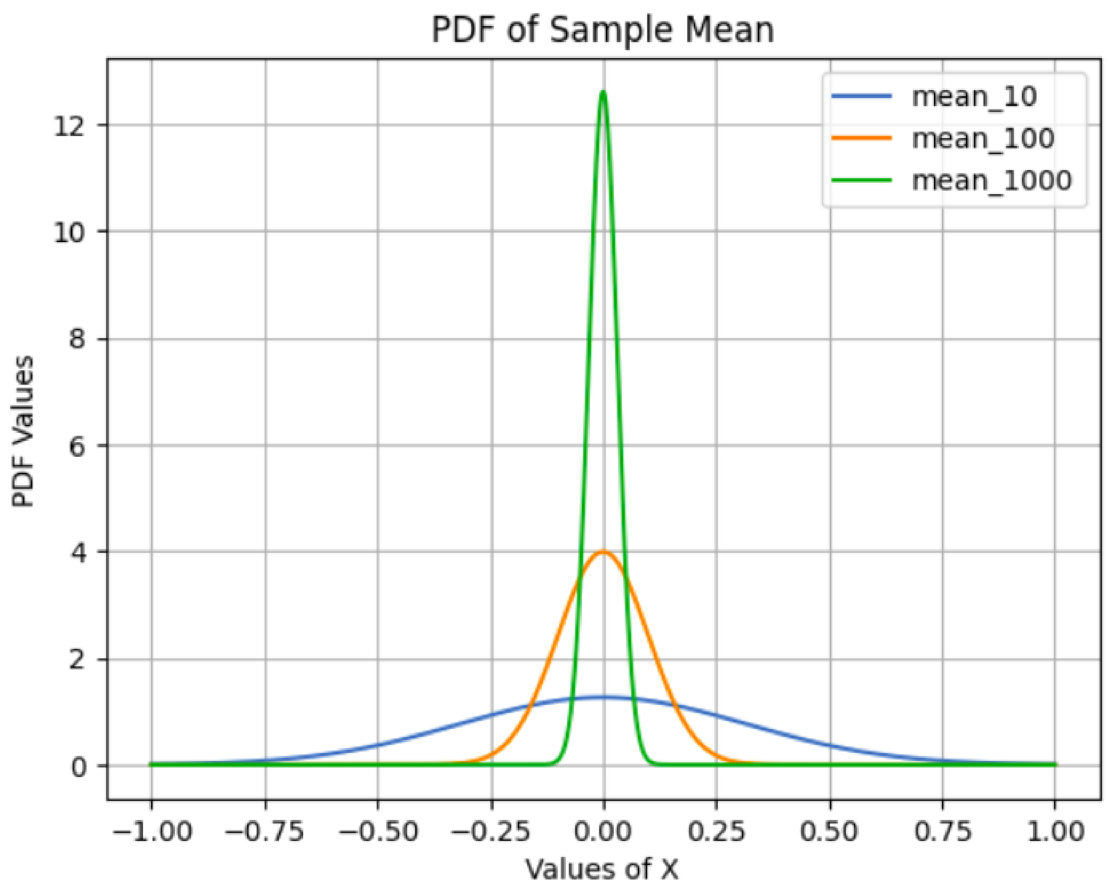
4.Write a function called sample_mean taking as inputs two integers m and n. The function should return an array of length n containing samples each obtained by taking m samples from the standard normal distribution and computing their sample mean. Call sample_mean(m=10, n=10000), sample_mean(m=100, n=10000) , and sample_mean(m=1000, n=10000) and plot a histogram for each of these outputs.

```
#creating a function sample_mwan that returns an array of length n
containing samples
def sample_mean(m,n):
  mean = np.zeros(n)

  for i in range(n):
    sample = np.random.normal(0, 1, m)
    mean[i] = np.mean(sample)
  return mean

#defining the length of n
N = 10000
mean_10 = sample_mean(m=10, n=N)
mean_100 = sample_mean(m=100, n=N)
mean_1000 = sample_mean(m=1000, n=N)

#visualisation graph
plt.figure(figsize=(12,3))

plt.subplot(1,3,1)
plt.hist(mean_10, density= True, alpha=0.7, bins=100, color='pink',
edgecolor='black')
plt.title("sample mean m=10")

plt.subplot(1,3,2)
plt.hist(mean_100, density= True, alpha=0.7, bins=100, color='blue',
edgecolor='black')
plt.title("sample mean m=100")

plt.subplot(1,3,3)
plt.hist(mean_1000, density= True, alpha= 0.7, bins=100, color='green',
edgecolor='black')
plt.title("sample mean m=1000")

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



By solving the first question of the Theory part, write a class called sample_mean_distribution whose constructor takes an integer m as input and implements the probability measure Normal(0, 1)m ≜ 1 m Xm i=1 Normal(0, 1) in other words, the distribution of the length-m estimator of the mean. Instantiate the objects sample_mean_distribution(10), sample_mean_distribution(100), sample_mean_distribution(1000) and plot their PDFs.

```
#creating a clas mean_distribution with two functions
class mean_distribution:
    def __init__(self, n):
      self.n = n
    def pdf(self,i):
      return norm.pdf(i, loc=0, scale=1/np.sqrt(self.n))
sample_mean_10 = mean_distribution(10)
sample_mean_100 = mean_distribution(100)
sample_mean_1000 = mean_distribution(1000)

#creating three empty arrays to store pdf's of means
pdf_mean_10 = []
pdf_mean_100 = []
pdf_mean_1000 = []


x = np.linspace(-1,1,1000)

#for loop generating pdf for the means
for i,j in enumerate(x):
  pdf_mean_10.append(sample_mean_10.pdf(j))
  pdf_mean_100.append(sample_mean_100.pdf(j))
  pdf_mean_1000.append(sample_mean_1000.pdf(j))

#visualisation graph
plt.plot(x, pdf_mean_10, label='mean_10')
plt.plot(x, pdf_mean_100, label='mean_100')
plt.plot(x, pdf_mean_1000, label='mean_1000')
plt.title('PDF of Sample Mean')
plt.xlabel('Values of X')
plt.ylabel('PDF Values')
plt.grid(True)
plt.legend()
plt.show()
```



Compare (a) the 3 histograms, (b) the 3 PDFs and (c) the histograms with the PDF. What conclusions do you draw?

According to the Central Limit Theorem, I can Observe that in the three Histograms, the sample values are directly proportional to the Normal Distribuition. That is, if the sample values increases they are more likely to be Normally Distributed.

In the three PDF's , I can observe the increasing of random variables results in the distribution of sample means to proceed towards a normal distribution and also we can notice that the standard deviation decreases, thereby centering around with True mean of 0. Sample mean is distributed very closely towards the true mean, So we can conclude that law of large numbers and Central Limit Theorem is followed here.

While comparing histogram with the pdf, both are achieving their normal distribution curve (bell curve). The both Histogram and PDF follows the similar normal distribution and the histogram attained its normal distribution, when we add large number of samples together. The pdf plot is centered around the True mean. The both Histogram and PDF plot follows the Central Limit Theorem.