Machine Learning 2023/2024 (2nd semester)

Master in Electrical and Computer Engineering

Department of Electrical and Computer Engineering

A. Pedro Aguiar (pedro.aguiar@fe.up.pt), **Aníbal Matos** (anibal@fe.up.pt), **Andry Pinto** (amgp@fe.up.pt), **Daniel Campos** (dfcampos@fe.up.pt), **Maria Inês Pereira** (up201505461@edu.fe.up.pt)

FEUP, Feb. 2024

Notebook #01: Introduction to the course

1- What is the Colaboratory?

This section is a partially copy of the "Colab getting started" file. See the original at https://colab.research.google.com/

[ ] ↳ *5 cells hidden*

2- Resources (Python)

Here goes a list of references where you can find more information about jupyter notebooks and Python.

↳ *1 cell hidden*

3- Python Essentials: a quickstart

This section provides a quick tour to some basic and useful functions in Python.

[ ] ↳ *197 cells hidden*

4- Machine Learning in Python

Here goes a very limited list of nice references/libraries.

↳ *2 cells hidden*

5- Exercises

Activity 1

Typical distributions for continuous random variables:

Gaussian distribution, also known as the normal distribution, is given by:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where x is the random variable, μ is the mean of the distribution, and σ is the standard deviation.

Exponential distribution with parameter λ is given by:

$$f(x) = \lambda e^{-\lambda x}$$

where x is the random variable. The exponential distribution models the time between events in a Poisson process, where events occur continuously and independently at a constant average rate λ.

Uniform distribution over the interval $[a, b]$ is given by:

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

where x is the random variable. The uniform distribution models a scenario where all values in the interval $[a, b]$ are equally likely.

Note that the probability of an event x to be between x_1 and x_2 is given by

$$P(x_1 < x \leq x_2) = \int_{x_1}^{x_2} f(x)\, dx$$

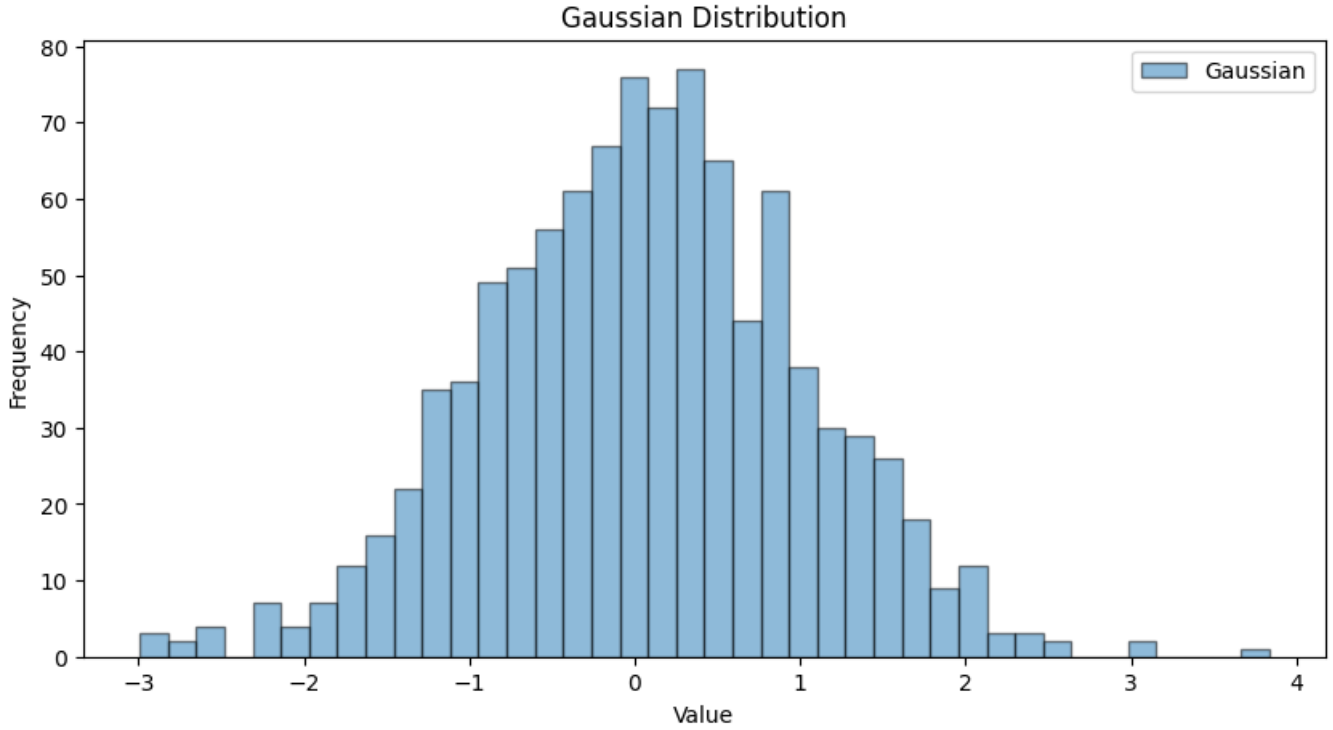
The next code generates random samples for each distribution, and plots the resulting histograms.

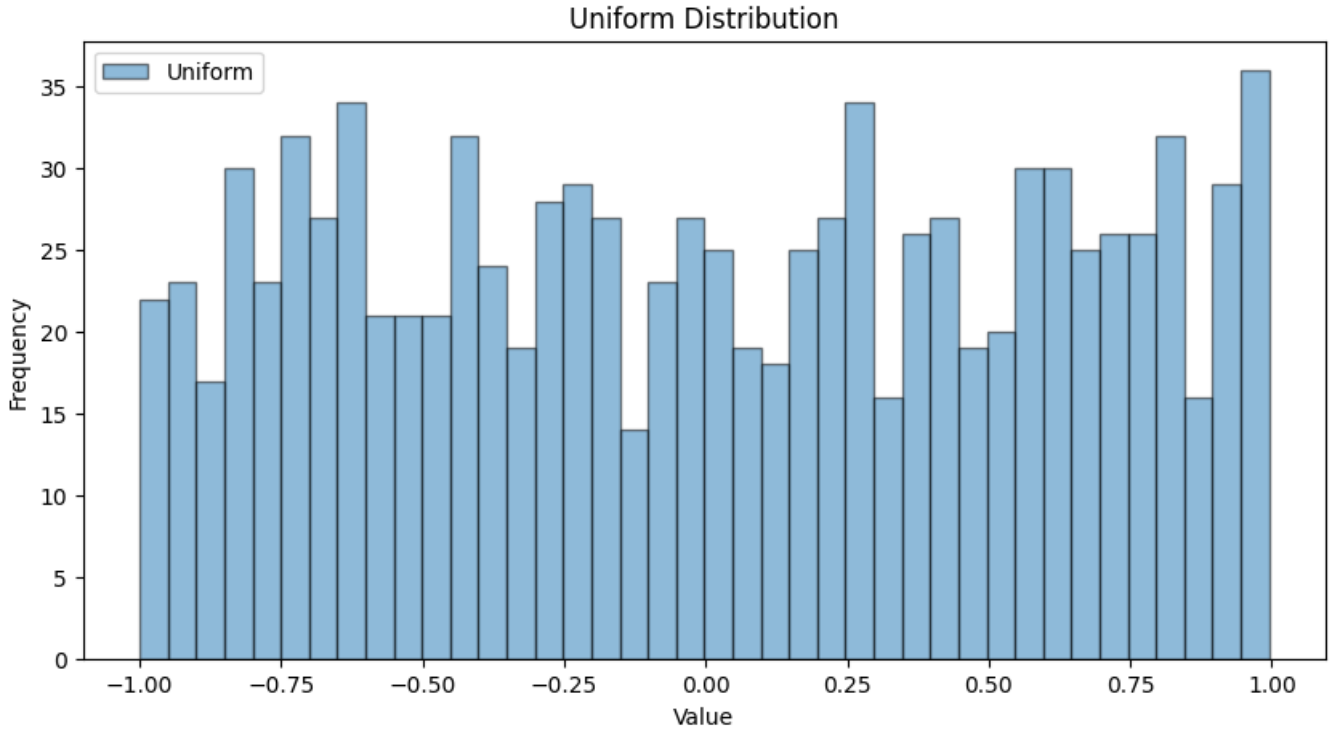
```
import numpy as np
import matplotlib.pyplot as plt

num_samples = 1000

# Generate Gaussian Distribution
mean = 0
std = 1
gaussian = np.random.normal(mean, std, num_samples)
```

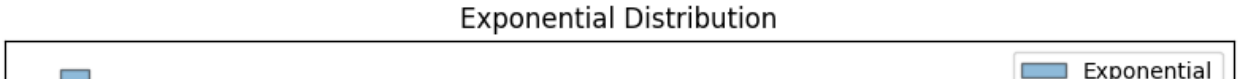
```
# Generate Uniform Distribution
low = -1
high = 1
#uniform = ... #To complete!

# Generate Exponential Distribution
lambd = 1
#exponential = ... #To complete!

# Plot Histograms
# Plot Gaussian Histogram
plt.figure(figsize=(10, 5))
plt.hist(gaussian, bins=40, edgecolor='black', alpha=0.5, label='Gaussian')
plt.title('Gaussian Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
plt.show()

# Plot Uniform Histogram
plt.figure(figsize=(10, 5))
plt.hist(uniform, bins=40, edgecolor='black', alpha=0.5, label='Uniform')
plt.title('Uniform Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
plt.show()

# Plot Exponential Histogram
plt.figure(figsize=(10, 5))
plt.hist(exponential, bins=40, edgecolor='black', alpha=0.5, label='Exponential
plt.title('Exponential Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
plt.show()

#Plot all in the same figure
plt.figure(figsize=(10, 5))
plt.hist(gaussian, bins=40, edgecolor='black', alpha=0.5, label='Gaussian')
plt.hist(uniform, bins=40, edgecolor='black', alpha=0.5, label='Uniform')
plt.hist(exponential, bins=40, edgecolor='black', alpha=0.5, label='Exponential
plt.legend()
plt.show()



#Solution
```

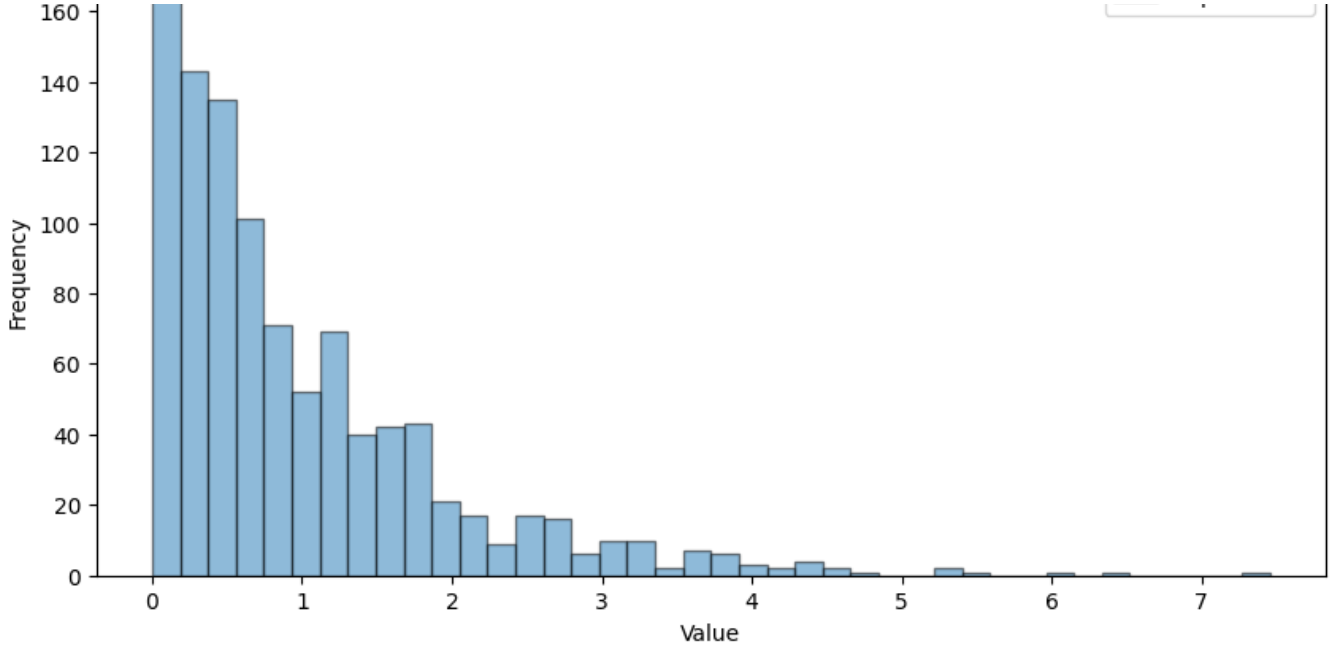
```
import numpy as np
import matplotlib.pyplot as plt

num_samples = 1000

# Generate Gaussian Distribution
mean = 0
std = 1
gaussian = np.random.normal(mean, std, num_samples)

# Generate Uniform Distribution
low = -1
high = 1
uniform = np.random.uniform(low, high, num_samples)

# Generate Exponential Distribution
lambd = 1
exponential = np.random.exponential(lambd, num_samples)

# Plot Histograms
# Plot Gaussian Histogram
plt.figure(figsize=(10, 5))
plt.hist(gaussian, bins=40, edgecolor='black', alpha=0.5, label='Gaussian')
plt.title('Gaussian Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
plt.show()

# Plot Uniform Histogram
plt.figure(figsize=(10, 5))
plt.hist(uniform, bins=40, edgecolor='black', alpha=0.5, label='Uniform')
plt.title('Uniform Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
plt.show()

# Plot Exponential Histogram
plt.figure(figsize=(10, 5))
plt.hist(exponential, bins=40, edgecolor='black', alpha=0.5, label='Exponential
plt.title('Exponential Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

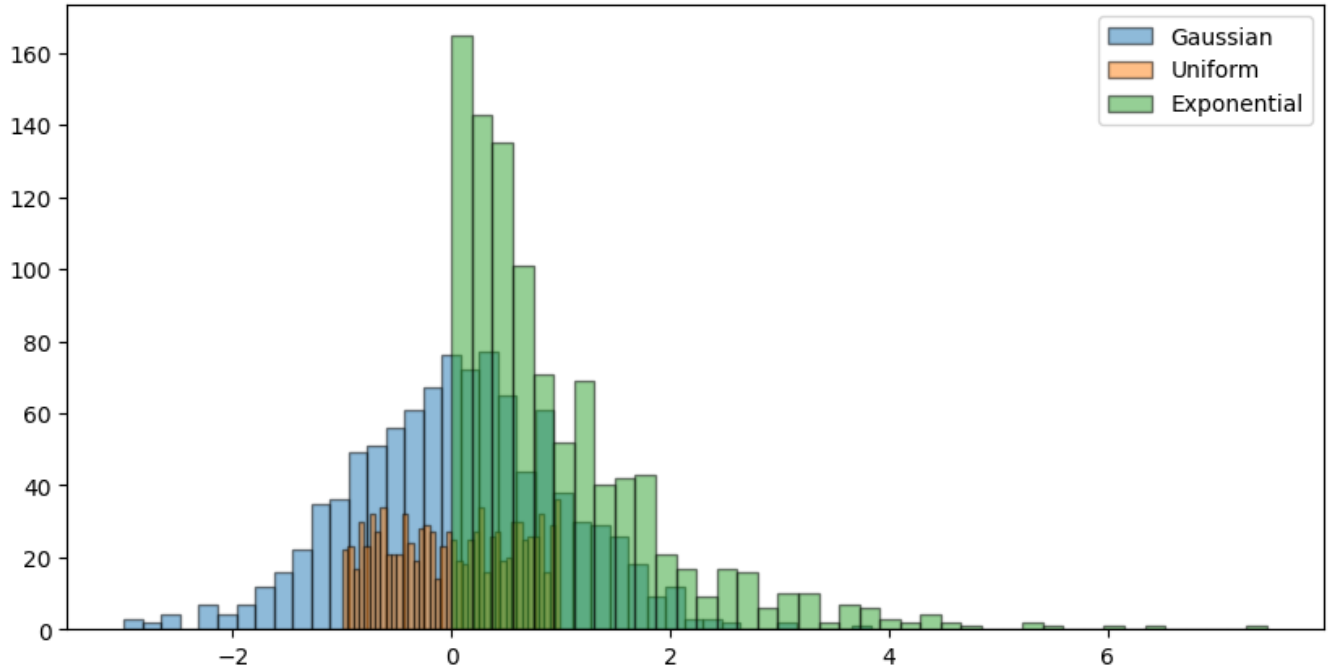
```
#Plot all in the same figure
plt.figure(figsize=(10, 5))
plt.hist(gaussian, bins=40, edgecolor='black', alpha=0.5, label='Gaussian')
plt.hist(uniform, bins=40, edgecolor='black', alpha=0.5, label='Uniform')
plt.hist(exponential, bins=40, edgecolor='black', alpha=0.5, label='Exponential
plt.legend()
plt.show()
```








Frequency
Value
0
20
40
60
80
100
120
140
160
0
1
2
3
4
5
6
7


Gaussian
Uniform
Exponential
0
20
40
60
80
100
120
140
160
−2
0
2
4
6

1.1. Try different parameter values and check the results.

Activity 2

Standardization

In machine learning applications, many datasets have input features with different scales. In such cases, particularly for Gaussian features, it is common to **standardize** the data, to ensure that each feature has mean 0 and variance 1. This is done by subtracting the **mean** μ and dividing by the **standard deviation** σ of each feature.

For the case that we do not known the mean and standard deviation, we have first to estimate them. More specifically, let $x = (x_1, x_2, \dots, x_n)^\top \in \mathbb{R}^n$ be a vector of samples of a single feature. Then an estimator for the mean is the **sample mean** given by

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{1}$$

and for the standard deviation is the **sample standard deviation**

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2} \tag{2}$$

In this case, the standardized vector $z = (z_1, z_2, \dots, z_n)^\top \in \mathbb{R}^n$ is just

$$z_i = \frac{x_i - \bar{x}}{s}, \quad i = 1, \dots, n$$

2.1. Consider now a data set $X \in \mathbb{R}^{n \times d}$, where d is the number of features, and each column of X corresponds to a feature with n samples. In particular, if $d = 1$ we have the previous case of just one vector of n samples.

For the data set X given below, compute for each feature the resulting

- sample mean
- sample standard deviation
- standardized X

Do this by implementing directly the formulas (do not use *np.mean* and *np.std*) and then compare the results with the solution provided by *sklearn* (in a cell below). Note that one may have a small diference because it is common to use the estimator for the standard deviation as $s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$ which is what is implemented in *np.std*. This is however not an unbiased estimator. The bias is not significant if n is not very small (the difference is between $\frac{1}{n}$ and $\frac{1}{n-1}$).

```
import numpy as np

n, d = 10, 3
#n, d = 1000, 3
np.random.seed(1)
mu, sigma = 10, 0.1 # mean and standard deviation
x1 = np.random.normal(mu, sigma, size=(n, 1))
mu, sigma = 2, 10 # mean and standard deviation
x2 = np.random.normal(mu, sigma, size=(n, 1))
mu, sigma = -10, 100 # mean and standard deviation
x3 = np.random.normal(mu, sigma, size=(n, 1))

X = np.block([x1, x2, x3])
print(X)
```

```
[[  10.16243454   16.62107937 -120.06191772]
 [   9.93882436  -18.60140709  104.47237098]
 [   9.94718282   -1.22417204   80.15907206]
 [   9.89270314   -1.84054355   40.24943389]
 [  10.08654076   13.33769442   80.08559493]
 [   9.76984613   -8.99891267  -78.37278592]
 [  10.17448118    0.27571792  -22.28902255]
 [   9.92387931   -6.77858418 -103.57694343]
 [  10.03190391    2.42213747  -36.78880796]
 [   9.97506296    7.82815214   43.03554667]]
```

```
def my_sample_mean(x):
  ''' Implement equation (1)
      Include your code here...
  '''

  return my_mean

def my_sample_std(x, mean):
  ''' Implement equation (2)
      Include your code here...
  '''

  return my_std

def my_standardize(X):
  ''' Include your code here...
  '''

  return my_std_data

Z = my_standardize(X)
print(Z)
print("\n\n", np.mean(Z, axis=0), "\n\n", np.std(Z, axis=0))


# Solution

def my_sample_mean(x):
  ''' Implement equation (1)
      Include your code here...
  '''
  mean = 0
  for i in range(len(x)):
    mean += x[i]

  mean = mean/len(x)
  return mean



def my_sample_std(x, mean):
  ''' Implement equation (2)
      Include your code here...
  '''
  std = 0
  for i in range(len(x)):
    std += (x[i] - mean)**2

#  std = np.sqrt((1 / (len(x))) * std)
```

```
  std = np.sqrt((1 / (len(x) - 1)) * std)
  return std


def my_standardize(X):
  ''' Include your code here...
  '''
  std_data = np.zeros(X.shape)
  for j in range(len(X[0])):
    mean = my_sample_mean(X[:,j])
    std = my_sample_std(X[:,j], mean)
    for i in range(len(X[:,j])):
      std_data[i][j]= (X[i][j] - mean) / std

  return std_data


Z = my_standardize(X)
print(Z)
print("In this case, we should have something closer to mean=1 and sigma=1 \n\r
```

```
[[ 1.37135548  1.54929569 -1.45026995]
 [-0.40994856 -1.79507947  1.29184907]
 [-0.34336407 -0.14511097  0.99492356]
 [-0.77735544 -0.20363545  0.50752816]
 [ 0.76677707  1.23753828  0.99402622]
 [-1.7560483  -0.88332261 -0.94114254]
 [ 1.46732038 -0.00269641 -0.25622107]
 [-0.52900252 -0.67250241 -1.24894764]
 [ 0.3315337   0.20110613 -0.43329931]
 [-0.12126773  0.71440723  0.54155349]]
In this case, we should have something closer to mean=1 and sigma=1

 [-1.27328703e-14  0.00000000e+00  3.33066907e-17]

 [0.9486833 0.9486833 0.9486833]
```

```
# For comparison run this code:

from sklearn import preprocessing
import numpy as np

scaler = preprocessing.StandardScaler().fit(X)

print("Original data X\n","sample mean =", scaler.mean_, "\n sample standard de

X_scaled = scaler.transform(X)

print("\n Standardized data\n","sample mean =", X_scaled.mean(axis=0), "\n samp

print("\n", X_scaled)

# print("Diference\n", Z-X_scaled)
```

```
Original data X
 sample mean = [ 9.99028591  0.30411618 -1.3087459 ]
 sample standard deviation = [ 0.11908986  9.99139839 77.6815039 ]

 Standardized data
 sample mean = [-1.34253719e-14  1.11022302e-17  1.11022302e-17]
 sample standard deviation = [1. 1. 1.]

 [[ 1.4455356   1.63310105 -1.52871875]
 [-0.43212373 -1.89217991  1.36172849]
 [-0.36193751 -0.15296039  1.04874151]
 [-0.81940458 -0.21465061  0.53498166]
 [ 0.808254    1.30447989  1.04779564]
 [-1.85103744 -0.93110378 -0.99205134]
 [ 1.54669149 -0.00284227 -0.27008072]
 [-0.55761761 -0.70887979 -1.31650641]
 [ 0.3494672   0.21198447 -0.45673758]
 [-0.12782742  0.75305134  0.5708475 ]]
```

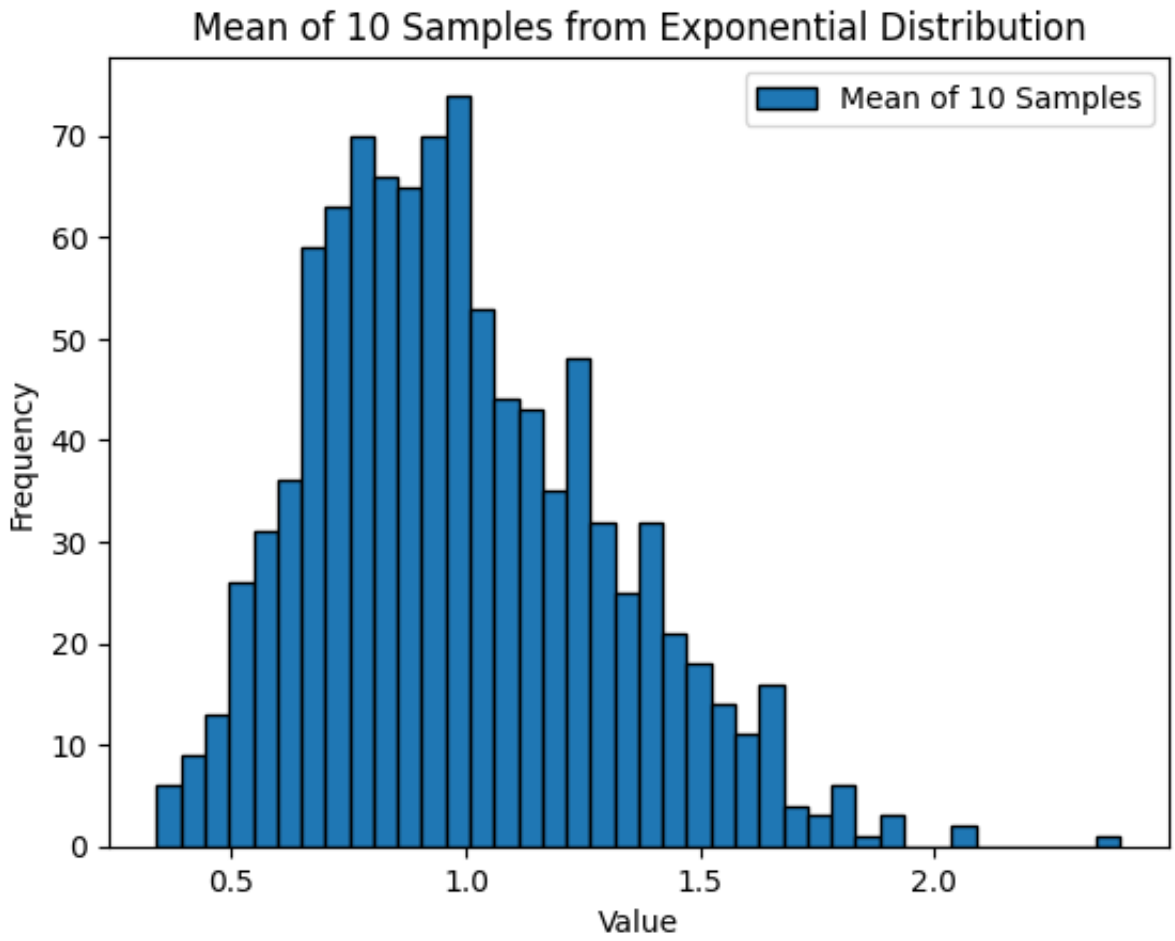
Activity 3

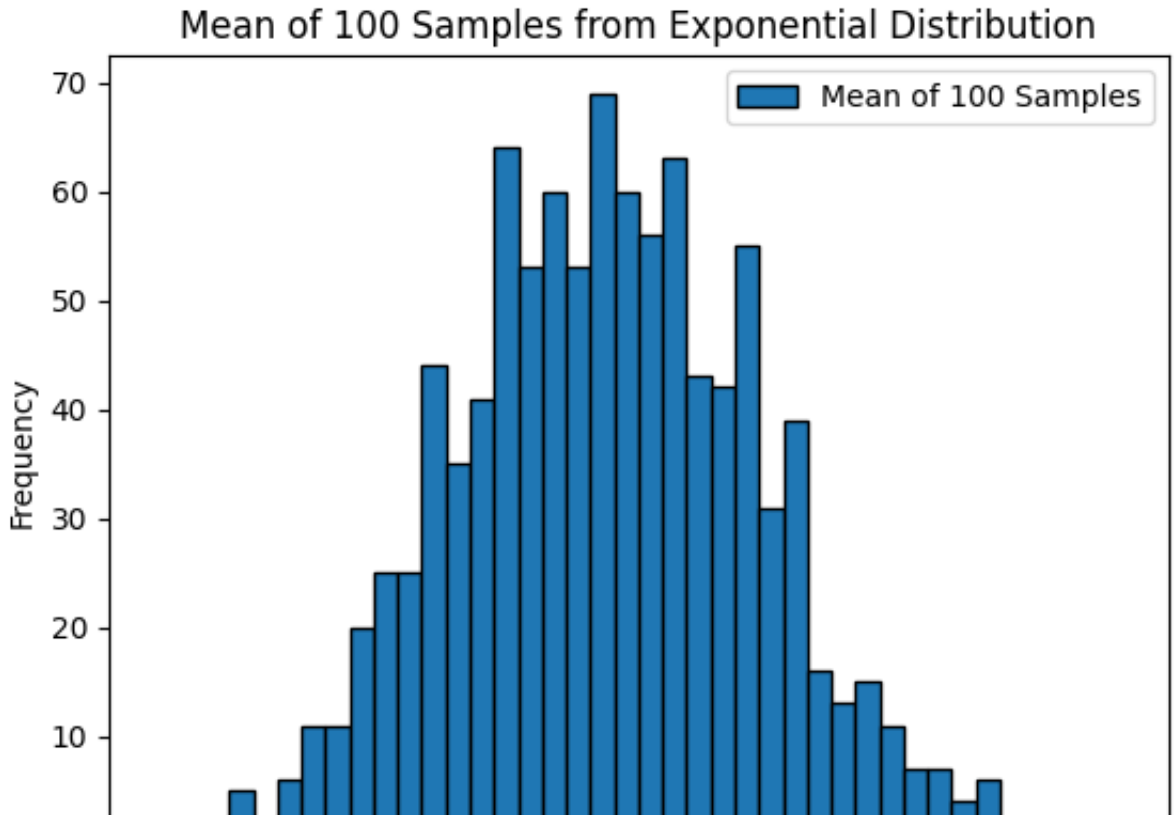
We will now see the **central limit theorem** in action!

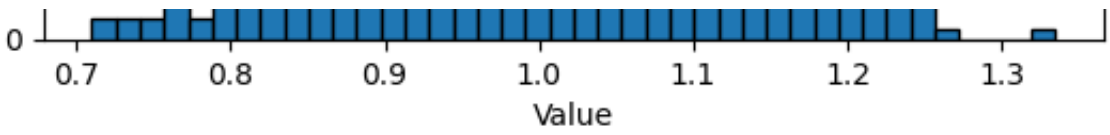
Basically we will see that regardless of the population distribution, as the sample size increases, the distributions of the sample means approach a normal distribution.

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


# Generate 1000 Means of 10 Samples from Exponential Distribution
n = 10
mean_samples = [np.mean(np.random.exponential(lambd, n)) for i in range(1000)]

# Plot Mean of n Samples Histogram
plt.hist(mean_samples, bins=40, edgecolor='black', label='Mean of 10 Samples')
plt.title(f'Mean of {n} Samples from Exponential Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
plt.show()

# Generate 1000 Mean of 100 Samples from Exponential Distribution
n = 100
#To complete...


#Solution

import numpy as np
import matplotlib.pyplot as plt


# Generate 1000 Means of 10 Samples from Exponential Distribution
n = 10
mean_samples = [np.mean(np.random.exponential(lambd, n)) for i in range(1000)]

# Plot Mean of n Samples Histogram
plt.hist(mean_samples, bins=40, edgecolor='black', label='Mean of 10 Samples')
plt.title(f'Mean of {n} Samples from Exponential Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
plt.show()

# Generate 1000 Mean of 100 Samples from Exponential Distribution
n = 100
mean_samples = [np.mean(np.random.exponential(lambd, n)) for i in range(1000)]

# Plot Mean of n Samples Histogram
plt.hist(mean_samples, bins=40, edgecolor='black', label='Mean of 100 Samples')
plt.title(f'Mean of {n} Samples from Exponential Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
```

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







Activity 4

A Likelihood classification problem

Consider now a classification problem. Suppose that we have 2 models, each one characterized by a given **multivariate Gaussian distribution**:

$$p(x\ \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}}\ \Sigma\ ^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^{\top}\Sigma^{-1}(x-\mu)\right) \tag{3}$$

where $\mu \in \mathbb{R}^d$ is the mean and $\Sigma \in \mathbb{R}^{d\times d}$ is the covariance matrix. Thus, each model is described by the set of parameters $\boldsymbol{\theta_1} = (\mu_1, \Sigma_1)$ and $\boldsymbol{\theta_2} = (\mu_2, \Sigma_2)$.

Let $x_n \in \mathbb{R}^d, n = 1, \ldots, N$ be a given set of samples and suppose that we would like to assign each sample x_n to the model for which the likelihood is maximum, that is, we say that x_n is classified to belong to model 1 if

$$p(x_n\ \mu_1, \Sigma_1) > p(x_n\ \mu_2, \Sigma_2)$$

otherwise, it belongs to model 2.

4.1. Implement a function that receives as input a data set $X \in \mathbb{R}^{n\times d}$ and returns a list with elements "blue" or "green" according to the likelihood of belonging to model 1 or 2, respectively.

```
import numpy as np

N, d = 10, 2
mu1, sigma1 = np.array([[1], [1]]), np.diag([0.1, 0.1]) # mean and covariance c
mu2, sigma2 = np.array([[-1], [-1]]), np.diag([1, 2]) # mean and covariance of

X = np.random.rand(N,d)
print(X)
```

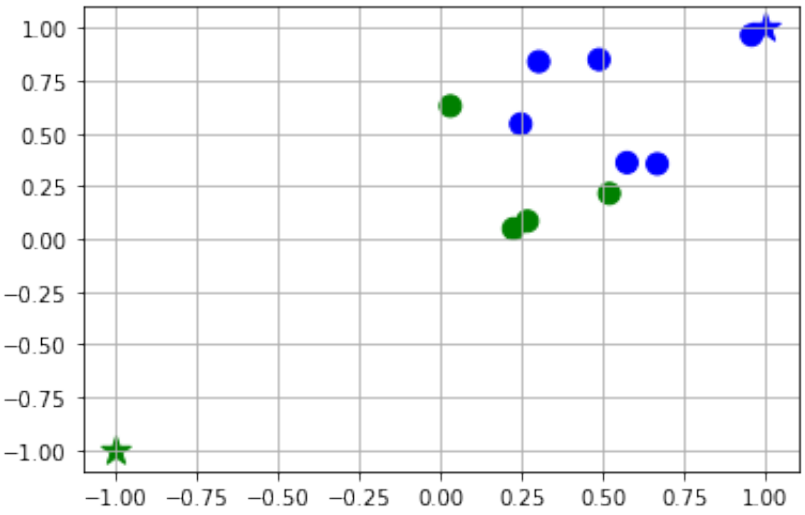
```
[[0.26692027 0.08459342]
 [0.30179736 0.83945177]
 [0.6661329  0.35546511]
 [0.51876381 0.21521455]
 [0.57317866 0.36138978]
 [0.24651206 0.54448884]
 [0.03020289 0.62935344]
 [0.4872843  0.84916245]
 [0.22393037 0.04836989]
 [0.95608589 0.96483611]]
```

```
def compute_p(X, mu, sigma):
  ''' Implement equation (3)
      Include your code here...
  '''

  return vector_p


v_dif = compute_p(X, mu1, sigma1)-compute_p(X, mu2, sigma2)

result = []
for n in v_dif:
  if n > 0:
    result.append('blue')
  else:
    result.append('green')


print(result)
```

```
# Solution

def compute_p(X, mu, sigma):
  ''' Implement equation (3)
      Include your code here...
  '''
  [N, d] = np.shape(X)
  vector_p = np.zeros([N,1])

  c = np.power(2*np.pi, d/2)*np.power(np.linalg.det(sigma), 0.5)
  inv_sigma = np.linalg.inv(sigma)
  for n in range(N):
    #x_mu = (X[n]-mu.T).T
    x_mu = np.atleast_2d(X[n]).T - mu
    vector_p[n] = 1/c * np.exp( - 0.5 * x_mu.T @ inv_sigma @ x_mu)
  return vector_p


v_dif = compute_p(X, mu1, sigma1)-compute_p(X, mu2, sigma2)

result = []
for n in v_dif:
  if n > 0:
    result.append('blue')
  else:
    result.append('green')


print(result)
```

```
['green', 'blue', 'blue', 'green', 'blue', 'blue', 'green', 'blue', 'green'
```

```
import matplotlib.pyplot as plt

plt.figure()
colors = np.array(result)
plt.scatter(X[:, 0], X[:, 1], c=colors, s=100)
plt.scatter(mu1[0], mu1[1], marker='*', s=200, c='b')#, mu2[:,0], marker='*')#,
plt.scatter(mu2[0], mu2[1], marker='*', s=200, c='g')
plt.grid()
plt.show()
```



4.2. Do the same, but now notice that (3) can be simplified by applying a log transform, where in this case we only get additions and multiplications, which are easier to handle, and does not impact the comparison, that is, if $p_1 > p_2$, then also $\log(p_1) > \log(p_2)$.

```
def compute_log_p(X, mu, sigma):
  ''' Implement equation (3)
      Include your code here...
  '''

  return vector_p


v_dif = compute_log_p(X, mu1, sigma1)-compute_log_p(X, mu2, sigma2)

result = []
for n in v_dif:
  if n > 0:
    result.append('blue')
  else:
    result.append('green')


print(result)
```

```
# Solution

def compute_log_p(X, mu, sigma):
  ''' Implement equation (3)
      Include your code here...
  '''
  [N, d] = np.shape(X)
  vector_p = np.zeros([N,1])

  c = -d/2*np.log(2*np.pi) - 0.5*np.log(np.linalg.det(sigma))
  inv_sigma = np.linalg.inv(sigma)
  for n in range(N):
    #x_mu = (X[n]-mu.T).T
    x_mu = np.atleast_2d(X[n]).T - mu
    vector_p[n] = c - 0.5 * x_mu.T @ inv_sigma @ x_mu

  return vector_p


v_dif = compute_log_p(X, mu1, sigma1)-compute_log_p(X, mu2, sigma2)

result = []
for n in v_dif:
  if n > 0:
    result.append('blue')
  else:
    result.append('green')


print(result)
```

```
['green', 'blue', 'blue', 'green', 'blue', 'blue', 'green', 'blue', 'green'
```

```
import matplotlib.pyplot as plt
%matplotlib inline

plt.figure()
colors = np.array(result)
plt.scatter(X[:, 0], X[:, 1], c=colors, s=100)
plt.scatter(mu1[0], mu1[1], marker='*', s=200, c='b')#, mu2[:,0], marker='*')#,
plt.scatter(mu2[0], mu2[1], marker='*', s=200, c='g')
plt.grid()
plt.show()
```