

Machine Learning

# TAREA 2

**K-MEANS  
CLUSTERING JERARQUICO  
GAUSSIAN MIXTURES MODEL**

Catalina Alvarez  
Bruno Fonseca  
Ricardo Romero



Universidad del Desarrollo

# K MEANS

```
import numpy as np
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import random
import cv2
from PIL import Image
```

**1.-Generamos centroides iniciales:** Definimos la función para obtener los centroides iniciales únicos, donde X corresponde a la matriz de datos de entrada, y k son el número de centroides iniciales

```
def obtener_centroides_iniciales(X, k):
    numero_de_muestras = X.shape[0]
    ids_puntos_de_muestra = random.sample(range(0, numero_de_muestras), k)

    centroides = [tuple(X[id]) for id in ids_puntos_de_muestra]
    centroides_unicos = list(set(centroides))

    numero_de_centroides_unicos = len(centroides_unicos)

    while numero_de_centroides_unicos < k:
        nuevos_ids_puntos_de_muestra = random.sample(range(0, numero_de_muestras), k - numero_de_centroides_unicos)
        nuevos_centroides = [tuple(X[id]) for id in nuevos_ids_puntos_de_muestra]
        centroides_unicos = list(set(centroides_unicos + nuevos_centroides))

        numero_de_centroides_unicos = len(centroides_unicos)

    return np.array(centroides_unicos)
```

**2.-Calculamos distancia euclíadiana:** Esta función calcula la distancia euclíadiana entre las matrices A y B, y devuelve la matriz de distancias resultante, esto mide la distancia de cada dato y el centroide más cercano

```
def obtener_distancia_euclidiana(matriz_A, matriz_B):
    A_cuadrado = np.reshape(np.sum(matriz_A * matriz_A, axis=1), (matriz_A.shape[0], 1))
    B_cuadrado = np.reshape(np.sum(matriz_B * matriz_B, axis=1), (1, matriz_B.shape[0]))
    AB = matriz_A @ matriz_B.T

    C = -2 * AB + B_cuadrado + A_cuadrado
```

**3.-Asignamos puntos de datos a clusteres mas cercanos:** La función obtener\_clusters se utiliza para asignar punto de datos a los clústeres basándose en la distancia entre los puntos de datos y los centroides de los clústeres.

```
def obtener_clusters(X, centroides, metodo_medicion_distancia):
    k = centroides.shape[0]
    clusters = {}
    matriz_distancias = metodo_medicion_distancia(X, centroides)
    ids_cluster_mas_cercanos = np.argmin(matriz_distancias, axis=1)

    for i in range(k):
        clusters[i] = []

    for i, id_cluster in enumerate(ids_cluster_mas_cercanos):
        clusters[id_cluster].append(X[i])

    return clusters
```

**4.-Verificamos si los centroides convergen:** Esta función se encarga de verificar si los centroides en una iteración han dejado de moverse significativamente en comparación con los centroides en la iteración actual

```
def centroides_convergen(centroides_anteriores, centroides_nuevos, metodo_medicion_distancia, umbral_de_movimiento):
    distancias_entre_centroides_anteriores_y_nuevos = metodo_medicion_distancia(centroides_anteriores, centroides_nuevos)
    centroides_cubiertos = np.max(distancias_entre_centroides_anteriores_y_nuevos.diagonal()) <= umbral_de_movimiento

    return centroides_cubiertos
```

# K MEANS

**Juntamos pasos anteriores y definimos K-Means**. Se aplica un bucle que se ejecuta hasta que centroides\_convergen sea True lo que significa que los centroides se han estabilizado. En cada iteración se guardan los centroides anteriores, se asignan los puntos de datos a los clúster más cercanos , se calculan nuevos clústeres y se verifica si han convergido con la función centroides\_convergen.

```
def k_means(X, k, metodo_medicion_distancia, umbral_de_movimiento=0):

    nuevos_centroides = obtener_centroides_iniciales(X=X, k=k)

    centroides_cubiertos = False

    while not centroides_cubiertos:
        centroides_anteriores = nuevos_centroides
        clusters = obtener_clusters(X, centroides_anteriores, metodo_medicion_distancia)

        nuevos_centroides = np.array([np.mean(clusters[key], axis=0, dtype=X.dtype) for key in sorted(clusters.keys())])

        centroides_cubiertos = centroides_convergen(centroides_anteriores, nuevos_centroides, metodo_medicion_distancia, umbral_de_movimiento)

    return nuevos_centroides
```

# K MEANS

**Utilizamos k-means para la reducción de la cantidad de colores:** Esta función devuelve una imagen con el número especificado de colores reducidos

```
def imagen_con_colores_reducidos(imagen, numero_de_colores):  
    h, w, d = imagen.shape  
  
    X = np.reshape(imagen, (h * w, d))  
    X = np.array(X, dtype=np.int32)  
  
    centroides = k_means(X, k=numero_de_colores, metodo_medicion_distancia=obtener_distancia_euclidiana)  
    matriz_distancias = obtener_distancia_euclidiana(X, centroides)  
    ids_cluster_mas_cercanos = np.argmin(matriz_distancias, axis=1)  
  
    X_reconstruida = centroides[ids_cluster_mas_cercanos]  
    X_reconstruida = np.array(X_reconstruida, dtype=np.uint8)  
    imagen_reducida = np.reshape(X_reconstruida, (h, w, d))  
  
    return imagen_reducida
```

# K MEANS

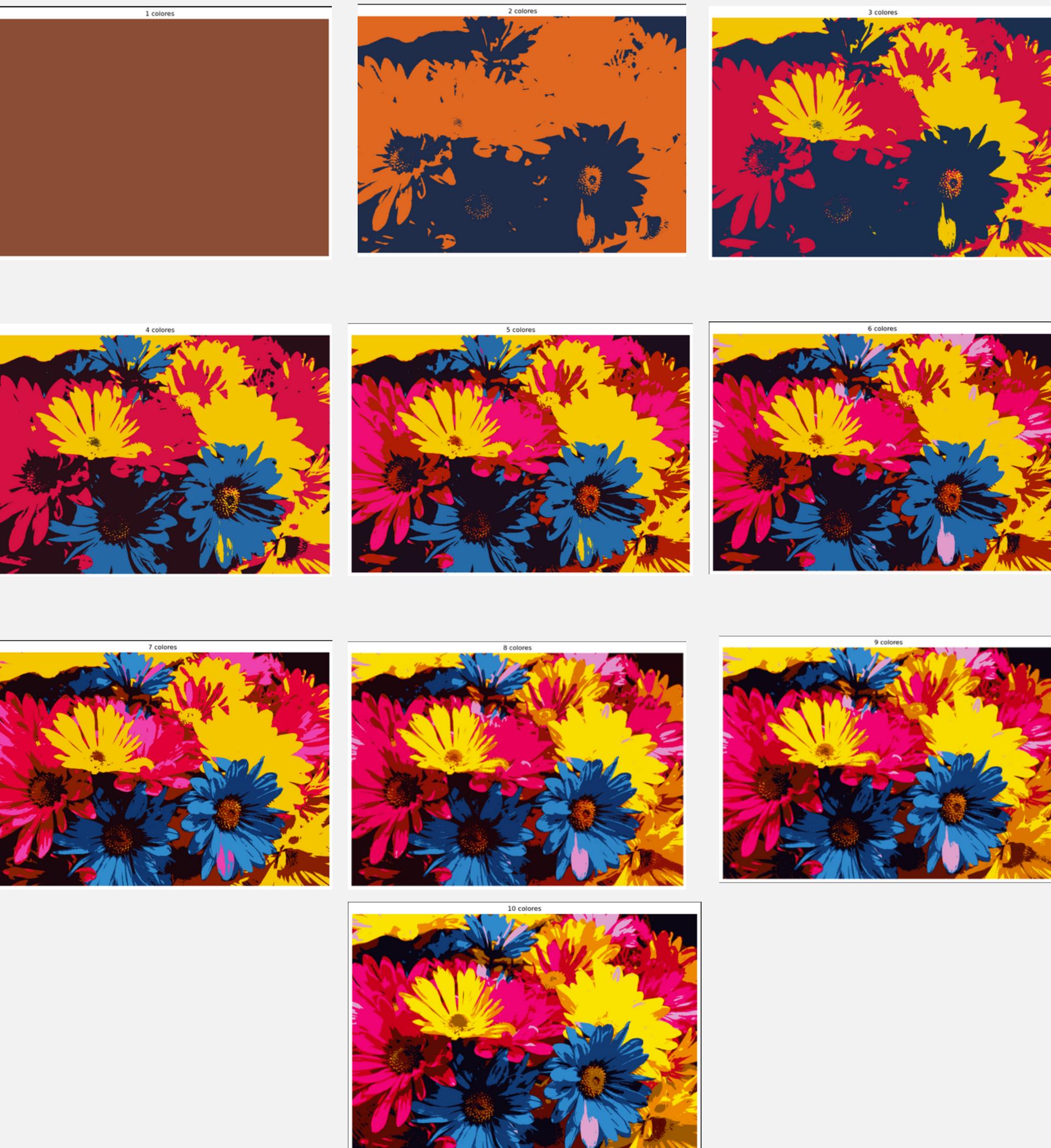
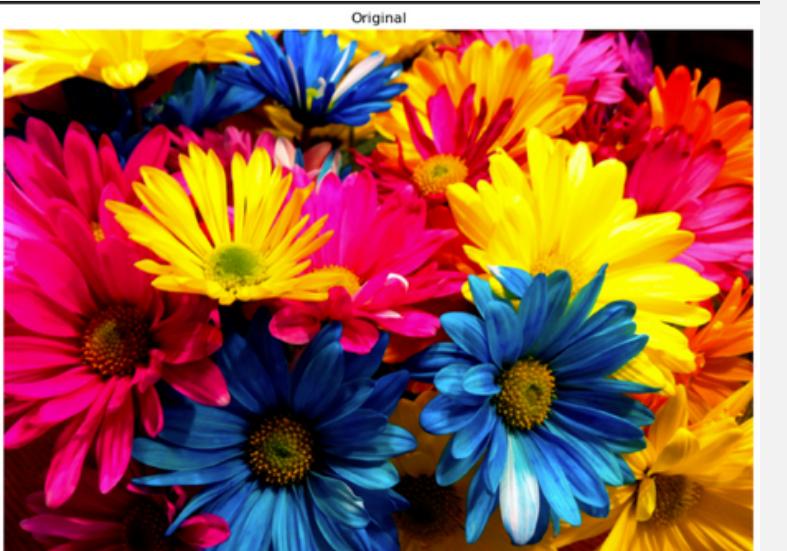
**Visualizamos la imagen desde 1 a 10 colores:** Cargamos nuestra imagen original y creamos un bucle donde se itera en cada uno de los valores de la lista `valores_k`, donde en cada iteración se llama a la función `imagen_con_colores_reducidos` y el valor `k` para obtener la imagen con colores reducidos.

```
valores_k = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

reconstrucciones = []

imagen = cv2.imread("flores.png")
plt.rcParams['figure.figsize'] = [15, 8]
plt.imshow(cv2.cvtColor(imagen, cv2.COLOR_BGR2RGB))
plt.title("Original")
plt.axis('off')
plt.show()

for k in valores_k:
    imagen_colores_reducidos = imagen_con_colores_reducidos(imagen, k)
    plt.title(f"{k} colores")
    plt.axis('off')
    plt.imshow(cv2.cvtColor(imagen_colores_reducidos, cv2.COLOR_BGR2RGB))
    plt.show()
    reconstrucciones.append(imagen_colores_reducidos)
```



# K MEANS

Método del codo: Identificar el punto donde existe un equilibrio entre la reducción de la distorsión y la complejidad del modelo. Compara cada imagen reconstruida con la original. Ellegimos 6 colores.

```
distorsiones = []
for i, k in enumerate(valores_k):
    distorsion = np.sum(np.square(imagen - reconstrucciones[i]))
    distorsiones.append(distorsion)

plt.grid()
plt.title("Gráfico de distorsión en función del número de colores", fontsize=15)
plt.xlabel('Número de colores', fontsize=18)
plt.ylabel('Distorsión', fontsize=18)
plt.plot(valores_k, distorsiones, linestyle='--', marker='o')
plt.show()
```



# K MEANS

Recomponemos la imagen

```
foto = reconstrucciones[5]
foto_bgr = foto[:, :, [2, 1, 0]]
#CAMBIO RGB POR BGR PARA PODER SER TRABAJADO
plt.imshow(foto_bgr)
plt.axis('off')
```



Detallamos la paleta de colores

```
colores_unicos = np.unique(foto_bgr.reshape(-1, foto_bgr.shape[2]), axis=0)
paleta = Image.new('RGB', (len(colores_unicos), 1))

for i, color in enumerate(colores_unicos):
    paleta.putpixel((i, 0), tuple(color))

paleta.save('paleta_de_colores.png')
plt.imshow(paleta)
plt.axis('off')
plt.show()
```



# GAUSSIAN MIXTURES MODEL

Importamos las librerías necesarias

```
In [107]: import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy import stats
from sklearn.mixture import GaussianMixture
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score
import seaborn as sns
from pandas.plotting import scatter_matrix
from sklearn import preprocessing
```

Cargamos y visualizamos el dataset de iris

```
In [93]: from sklearn.datasets import load_iris
iris = load_iris()
data = pd.DataFrame(iris.data)
data.columns = ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width']

# Denotamos el output que corresponde al tipo de flor
y = pd.DataFrame(iris.target)
y.columns = ['Targets']
```

Visualizamos el dataset

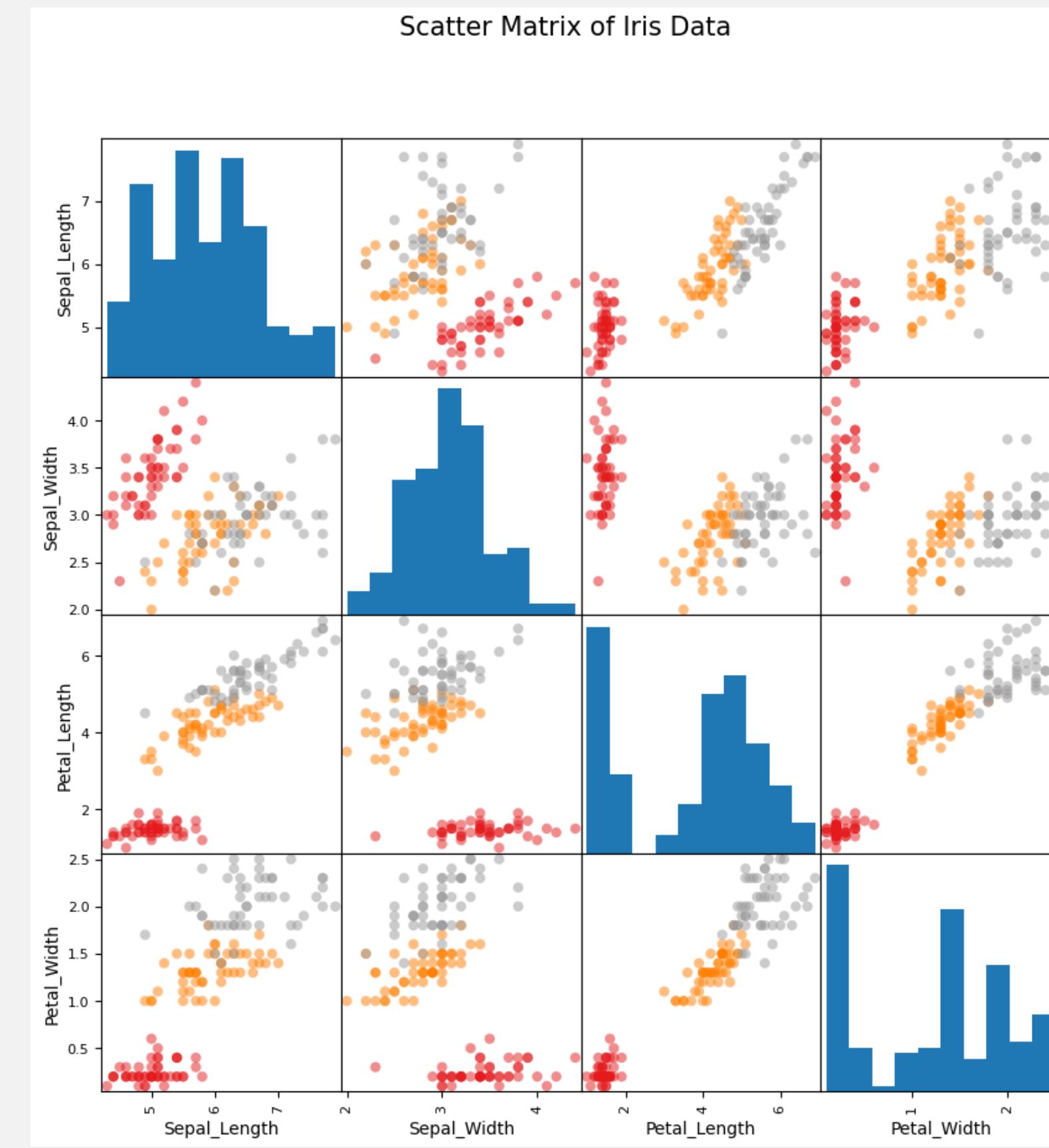
```
In [95]: data
```

```
Out[95]:
```

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

# GAUSSIAN MIXTURES MODEL



# GAUSSIAN MIXTURES MODEL

Se normaliza la data para que la media tenga valor 0 y la desviación estandar 1

```
In [132]: scaler = preprocessing.StandardScaler()  
scaler.fit(data)
```

```
Out[132]: StandardScaler  
StandardScaler()
```

```
In [133]: scaled_data = scaler.transform(data)  
DataSCALED = pd.DataFrame(scaled_data, columns= data.columns)
```

```
In [134]: GMM = GaussianMixture(n_components = 3 )  
GMM_y = gmm.fit_predict(DataSCALED)  
labels = np.zeros_like(clusters)  
  
for i in range(3):  
    categoria = (GMM_y == i)  
    labels[categoria] = mode(iris.target[categoria])[0]  
  
acc = accuracy_score(iris.target, labels)  
print("La exactitud del modeo está dada por un", acc)
```

La exactitud del modeo está dada por un 0.9666666666666667

```
: import matplotlib.pyplot as plt  
import pandas as pd  
from sklearn.datasets import load_iris  
from sklearn.mixture import GaussianMixture  
  
# Cargar el conjunto de datos Iris  
iris = load_iris()  
data = pd.DataFrame(iris.data, columns=['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width'])  
  
# Ajustar un modelo de mezcla gaussiana (Gaussian Mixture Model)  
gmm = GaussianMixture(n_components=3)  
gmm.fit(data)  
  
# Obtener las etiquetas de clusters del modelo GMM  
gmm_y = gmm.predict(data)  
  
# Definir los colores en orden: rojo, celeste y azul  
colors = ['red', 'lightblue', 'blue']  
  
# Ajustar el tamaño de la figura  
plt.figure(figsize=(10, 10))
```

# GAUSSIAN MIXTURES MODEL

