Universidad Simón Bolívar

IA 2 - CI-5438

Miguel Perez 15-11126

Gabriel Chaurio 17-10126

Repositorio de Github

Proyecto 3: Clustering

PROBLEMA

Implementar el algoritmo k-means para realizar categorizadores y segmentar imágenes mediante el uso de clustering.

IMPLEMENTACIÓN

La implementación del algoritmo de k-means se realiza en Python y es una ilustración clásica de cómo este algoritmo puede ser aplicado a problemas de clustering. Veamos cada componente del código en detalle:

Clase Kmeans

Constructor __init__:

Parámetros de Entrada:

- data: Un DataFrame de Pandas que contiene los datos a ser agrupados.
- k: El número de clusters a formar.
- normalize (opcional): Un booleano que indica si los datos deben ser normalizados.

Inicialización de Atributos:

- self.k, self.data, self.columns, self.X para almacenar el número de clusters, los datos, las columnas de los datos y una copia de los datos, respectivamente.

- Selección de Centroides Iniciales:

  Genera k centroides aleatorios dentro del rango de cada característica de los datos.

- Normalización:

  Opcionalmente normaliza los datos para que todas las características contribuyan equitativamente al proceso de clustering.

Método train:

- Iteración y Asignación de Clusters:

  Se itera un número definido de veces, asignando cada punto de datos al centroide más cercano basado en la distancia euclidiana al cuadrado.

- Actualización de Centroides:

  Después de cada asignación, recalcula los centroides como el promedio de todos los puntos en ese cluster.

- Convergencia:

  Si los centroides nuevos son iguales a los anteriores, el algoritmo ha convergido y se detiene.

Métodos Auxiliares:

- divide_by_number: Divide un vector por un número, utilizado en la actualización de los centroides.
- get_min_index: Encuentra el índice del valor mínimo en una lista, usado para asignar un punto al cluster más cercano.
- squared_euclidean_norm: Calcula la distancia euclidiana al cuadrado entre dos vectores, usada para medir la proximidad entre un punto y un centroide.
- sum_vectors: Suma dos vectores, útil en la actualización de los centroides.
- normalize: Normaliza los datos si se solicita.

Funcionalidad General:

El algoritmo de k-means implementado aquí sigue una lógica iterativa para agrupar los datos en k clusters basados en sus características. La inicialización de centroides de manera aleatoria puede afectar la convergencia y los resultados finales, lo cual es una característica inherente al algoritmo de k-means. La opción de normalización es importante, especialmente en casos donde las características tienen diferentes escalas.

img_util.py

Función segmentacion_imagen:

Parámetros de Entrada:

- input_file`: Ruta del archivo de imagen de entrada.
- output_file: Ruta del archivo de imagen de salida.
- colores: Número de colores (clusters) a utilizar en la segmentación.
- iteraciones: Número de iteraciones para el algoritmo K-means.

Funcionamiento:

- Lee la imagen de entrada y la convierte en un DataFrame.
- Inicializa y entrena el objeto Kmeans.
- Actualiza la imagen con los centroides resultantes.
- Escribe la imagen segmentada en el archivo de salida.

Función read_picture_rgb:

Parámetros de Entrada:

- picture_name: Ruta del archivo de imagen.

Retorno:

- DataFrame con los valores RGB de cada pixel, ancho y alto de la imagen.

Funcionamiento:

- Lee la imagen utilizando OpenCV.
- Convierte la imagen a formato RGB.

- Extrae los valores RGB de cada píxel y los almacena en un DataFrame.

Función update_picture_w_centroids:

Parámetros de Entrada:

- data: DataFrame con los datos de la imagen.
- centroides: Lista de centroides resultantes del algoritmo K-means.

Retorno:

- Lista de colores actualizados para cada píxel según el centroide más cercano.

Función write_picture_from_rgb:

Parámetros de Entrada:

- out_name: Nombre del archivo de salida.
- picture_array: Array de píxeles de la imagen.
- width: Ancho de la imagen.
- height: Altura de la imagen.

Funcionamiento:

- Convierte el array de píxeles a una imagen utilizando PIL.
- Guarda la imagen en el archivo de salida.

Funcionalidad General:

Este código realiza la segmentación de imágenes aplicando K-means para agrupar los colores en un número especificado de categorías. Trabaja directamente con los valores de los píxeles de la imagen, ajustando los colores en función de los centroides calculados. Esto permite simplificar la paleta de colores de la imagen original, lo que es útil en tareas de compresión de imágenes, análisis de imágenes y en la visualización de patrones de color.

DESCRIPCIÓN DE LOS DATOS

- **Iris Dataset:** El archivo “iris.csv”, contiene información de 3 clases. Cada clase es un tipo de planta del género Iris. Las columnas contienen las características de cada muestra y el nombre de la especie.

- **Imágenes**: En la carpeta input, se encuentran las imágenes que serán utilizadas para la segmentación. Cada una tiene tamaños distintos y la paleta de colores que presentan son distintas. Una imagen de un perro, el cual presenta diferentes tonos de negro, una imagene de Zelda, la cual presenta una paleta de colores pasteles y una imagen de un paisaje con una paleta de colores variada y vividos.

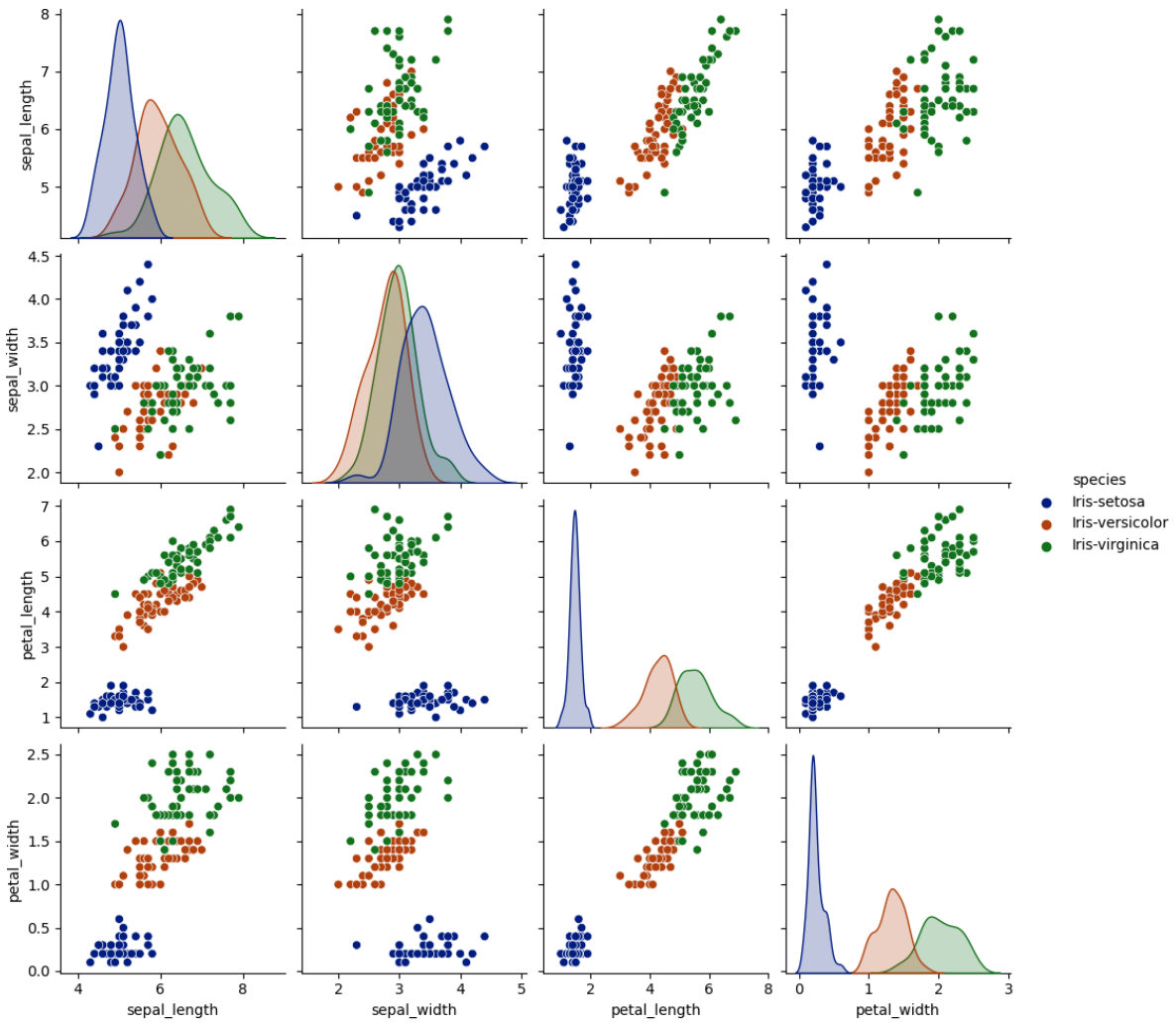
- RESULTADOS DE LOS EXPERIMENTOS

- Iris:

En el iris dataset, se aplicó el algoritmo con valores k en un rango de 2 a 5, como fue especificado en el enunciado. Buscamos ver cómo se comporta la generación de centroides para cada cluster. Se generarán k números de centroides.

Cuando se corre el algoritmo en el cual se buscan los centroides, al terminar su ejecución, se tiene entonces cada muestra asignada a un cluster, en base a cual centroide se encuentra más cercano.

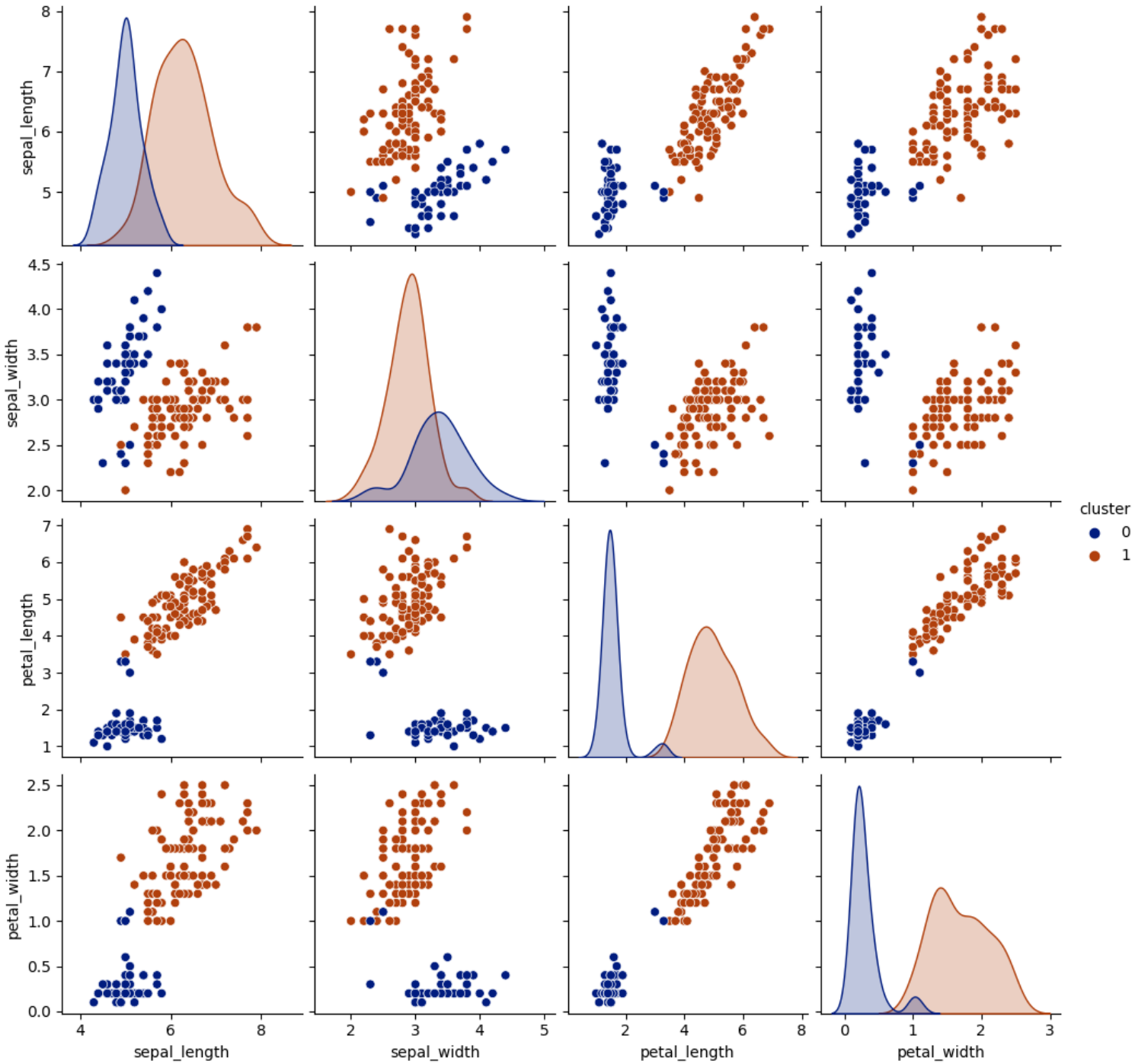
Tomemos como referencia la distribución inicial del dataset:



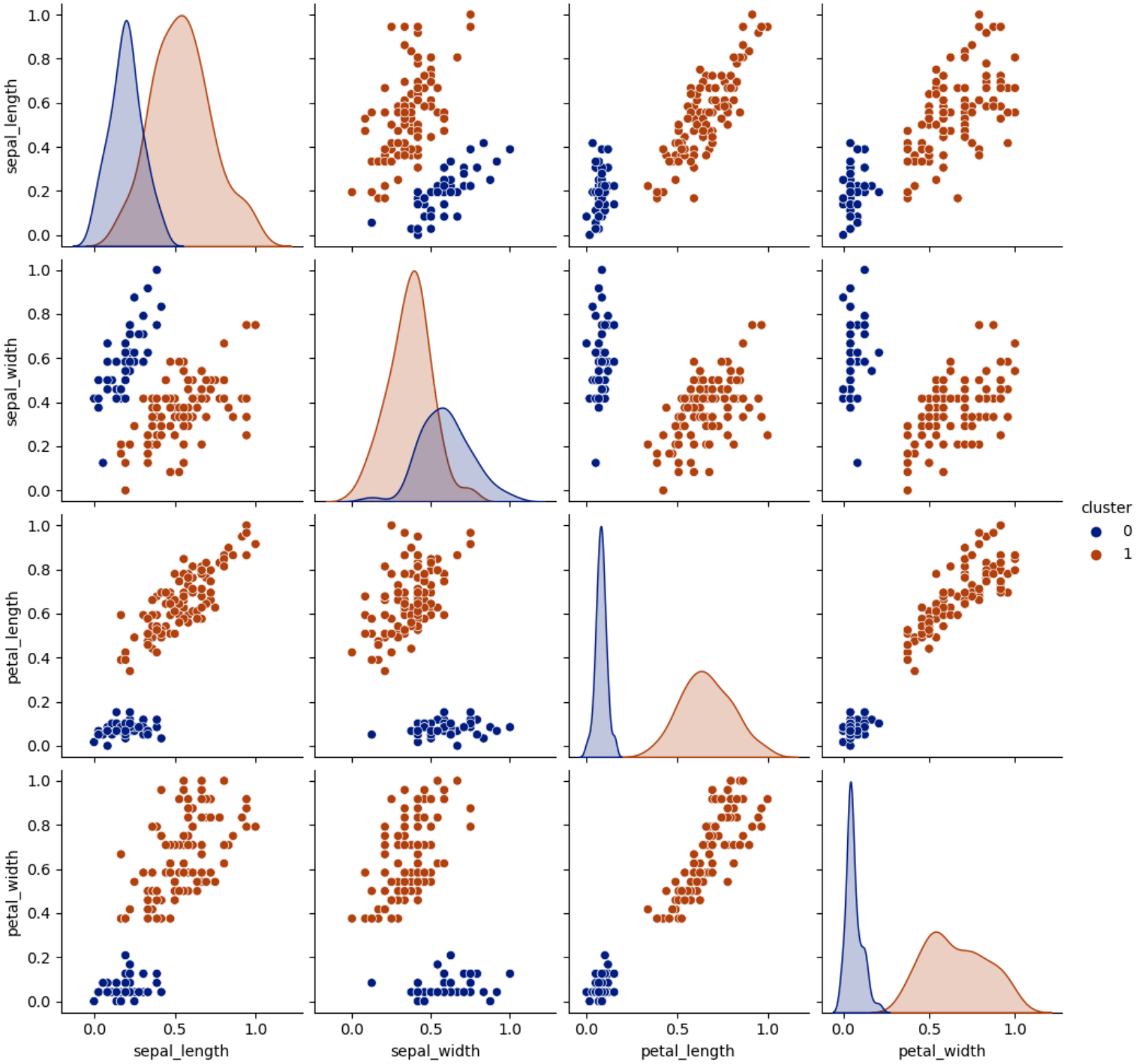
A continuación, los resultados obtenidos:

K = 2

- Sin Normalizar

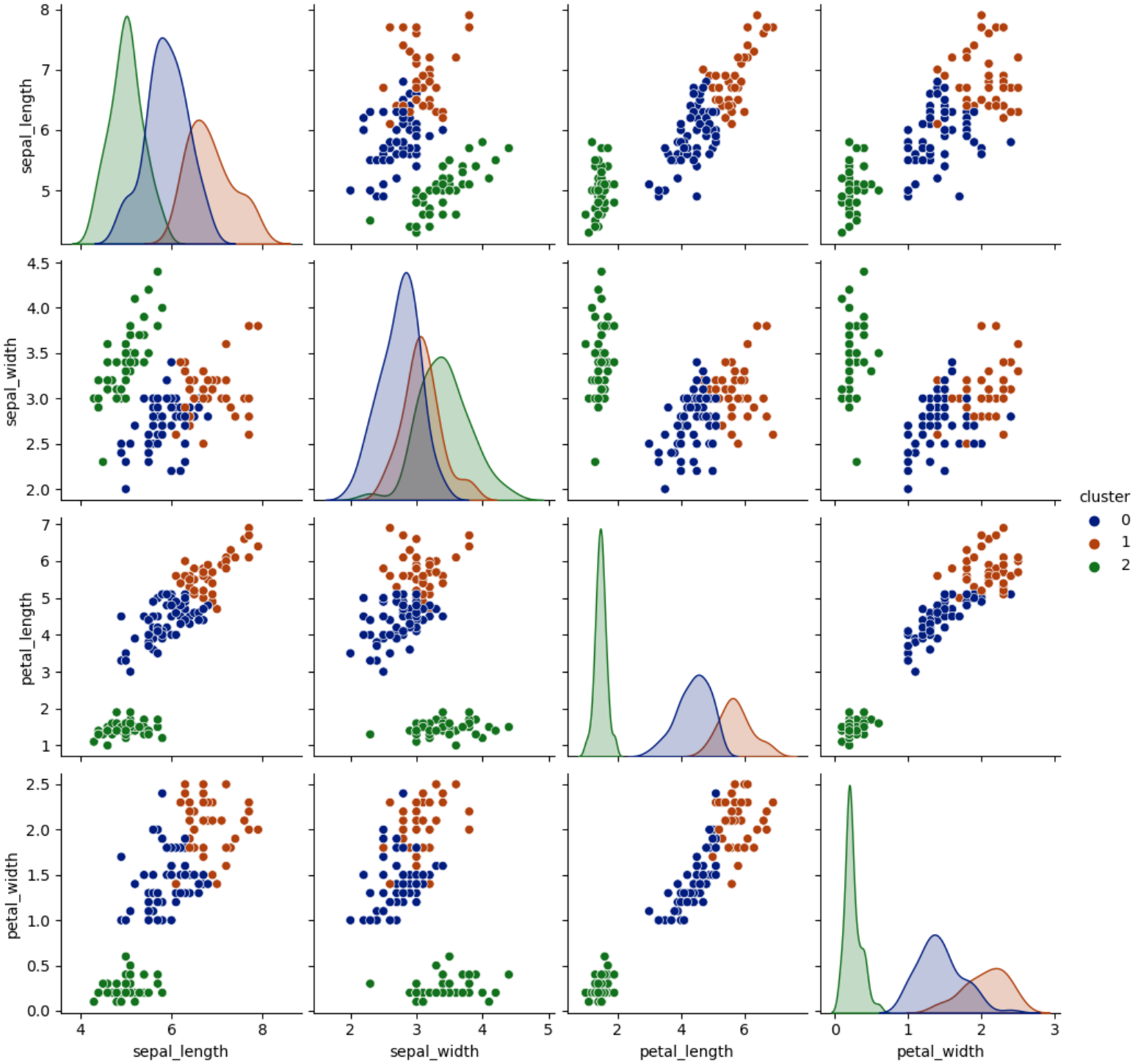


- Normalizado

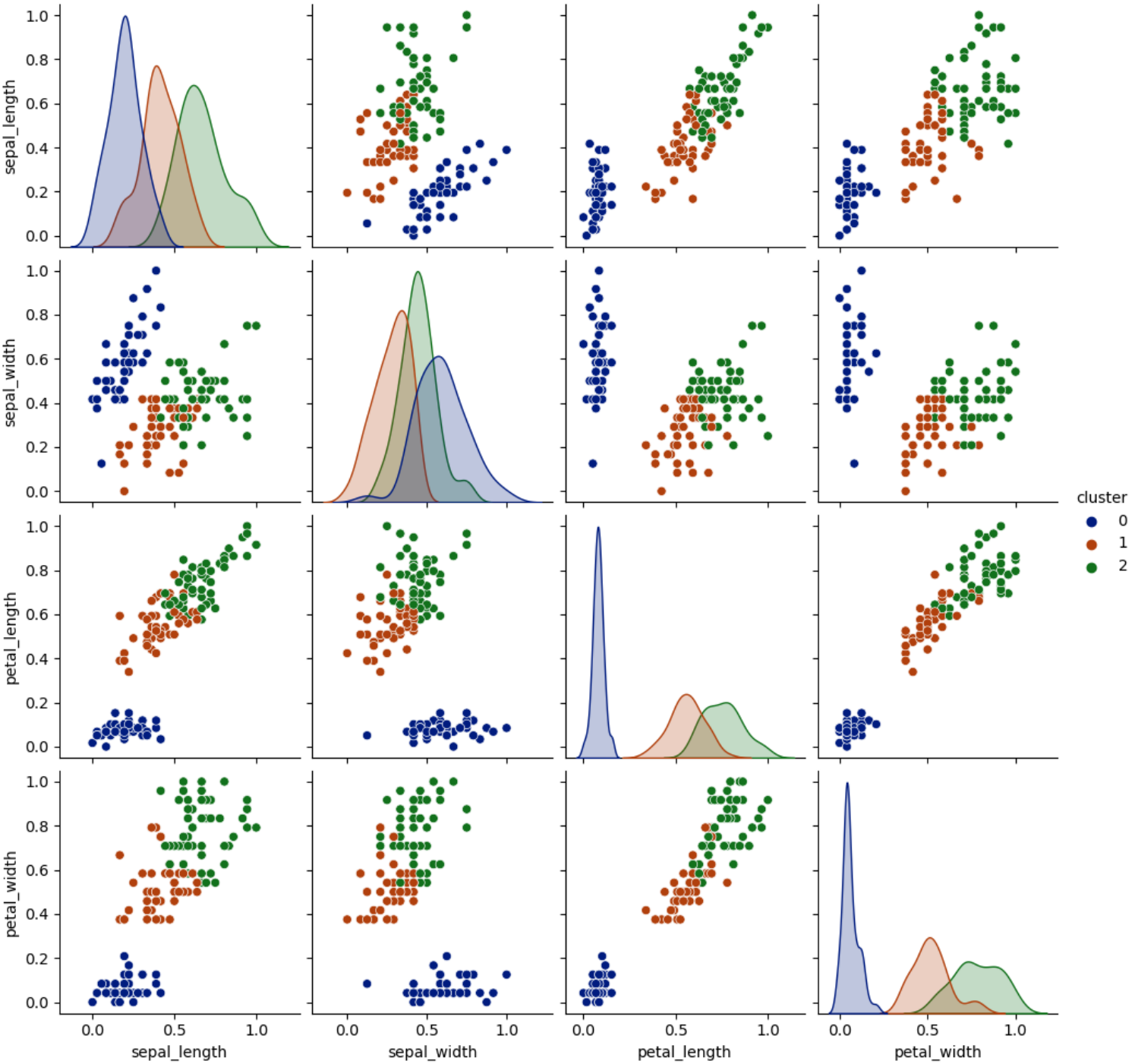


K = 3

- Sin Normalizar

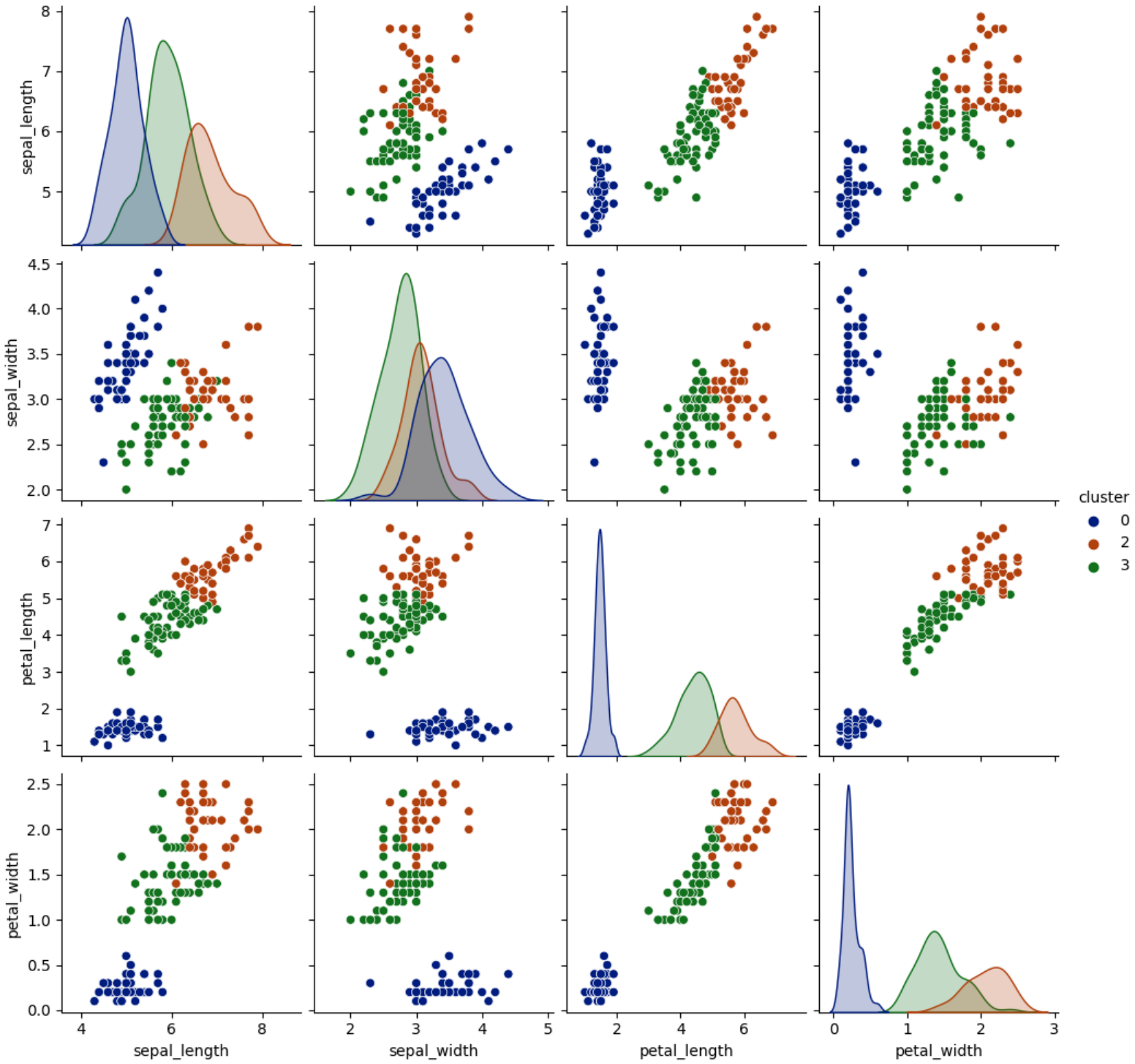


- Normalizado

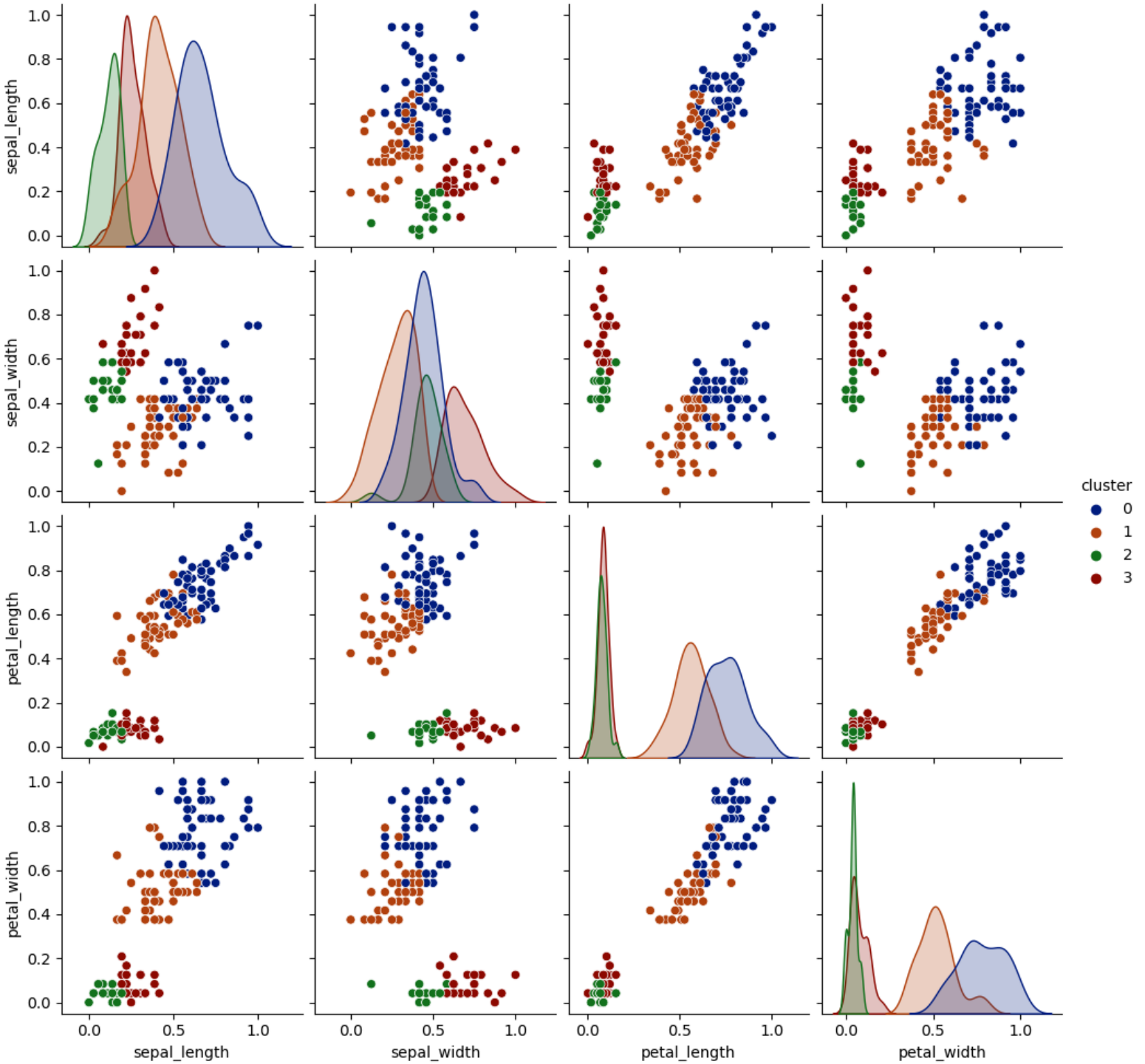


K = 4

- Sin Normalizar

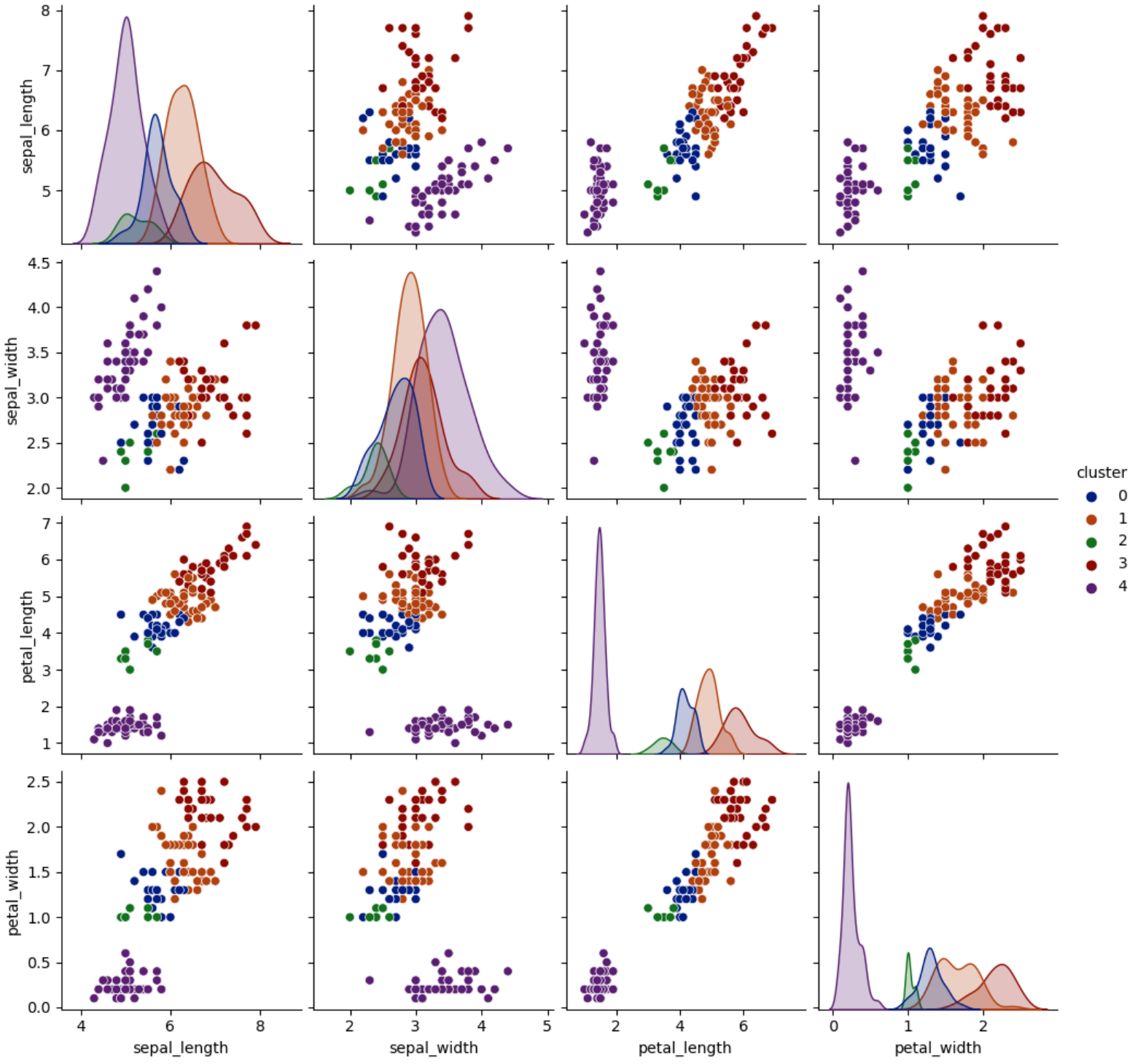


- Normalizado

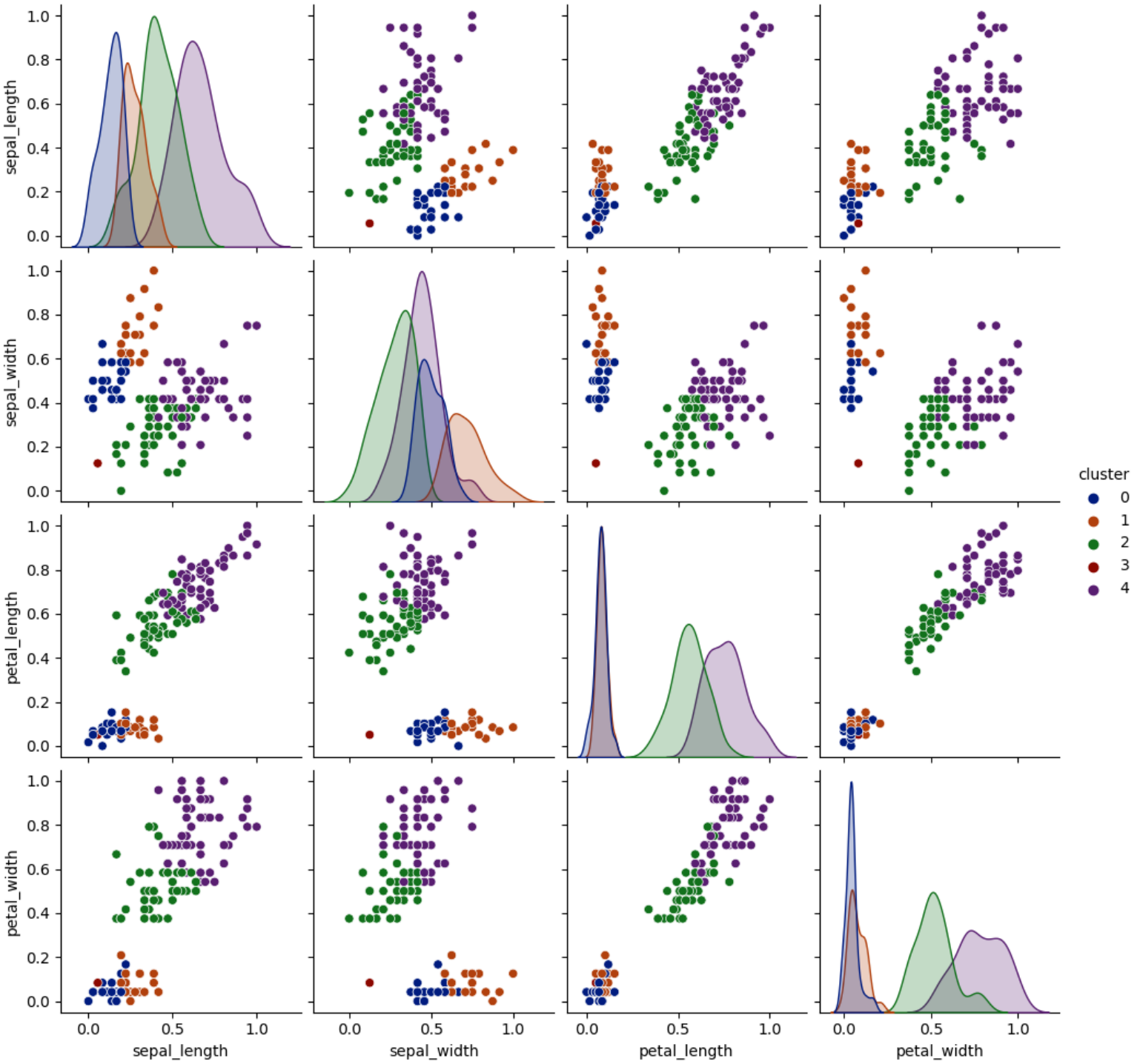


K = 5

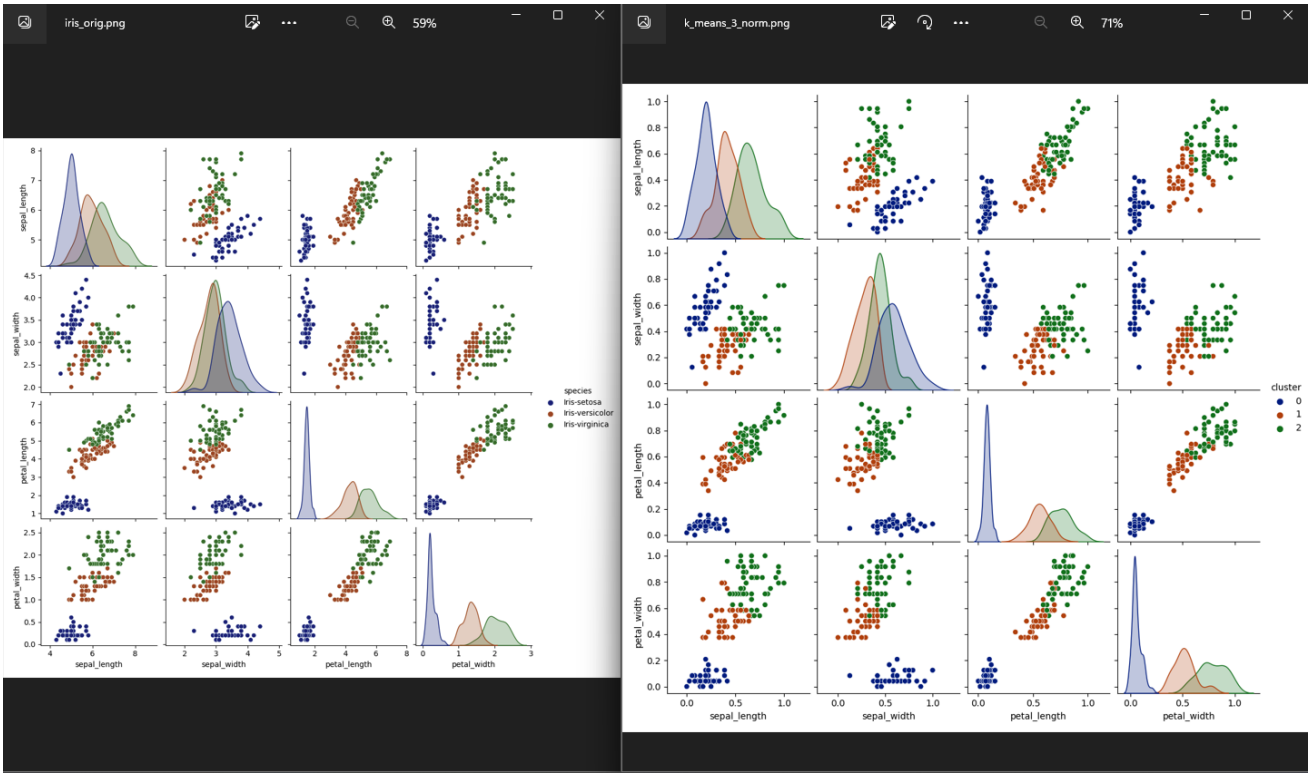
- Sin Normalizar



- Normalizado

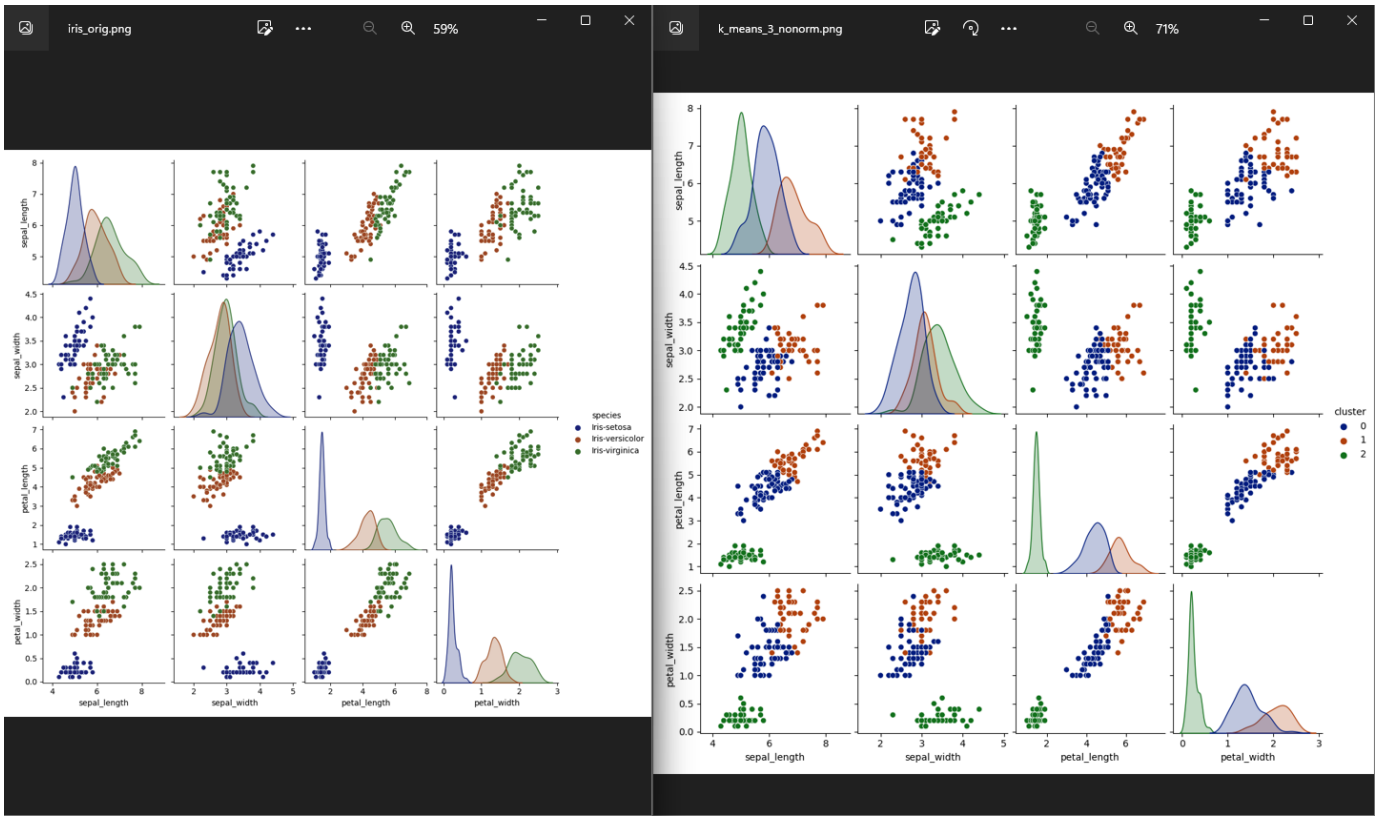


Podemos observar que la categorización hecha con k = 3 es la que más se asemeja a la categorización obtenida en el dataset original, ya que presenta una distribución muy parecida entre las especies originales y la asignación por centroides. La clasificacion por categorias varia levemente de la clasificacion original, y esto puede ser apreciado tanto en la asignacion de clusters a cada muestra, como a los picos de densidad por cluster encontrado en cada variable.



Iris vs K = 3 Normalizado

Al tener los mismo colores, podemos asignar un cluster a cada variable, donde ambas presentan el mismo color. El unico punto donde hay una discrepancia de datos a considerar, el cual no es tan signiificativa es en la variable sepal_width, y esto probablemente se debe al ruido que presenta el dataframe, puesto que se obtiene mejor resultado que si se usa el dataset sin normalizar.



Iris vs K = 3 Sin Normalizar

De hecho, por el comportamiento que tuvo el algoritmo con distintos números de k, podemos afirmar que si se tienen una cantidad de centroides igual a la cantidad de posibles resultados a obtener, se puede obtener un clasificador bastante eficaz. Cabe destacar que el mismo algoritmo tiende a reducir el número de centroides si la cantidad de cluster son mayores a la cantidad de resultados de categorización posible en el dataset original, como podemos apreciar en el experimento de K = 4 sin normalizar. El algoritmo simplemente no asigna a uno de los centroides pues resulta irrelevante ante los otros que tienen menos distancia entre el conjunto de datos.

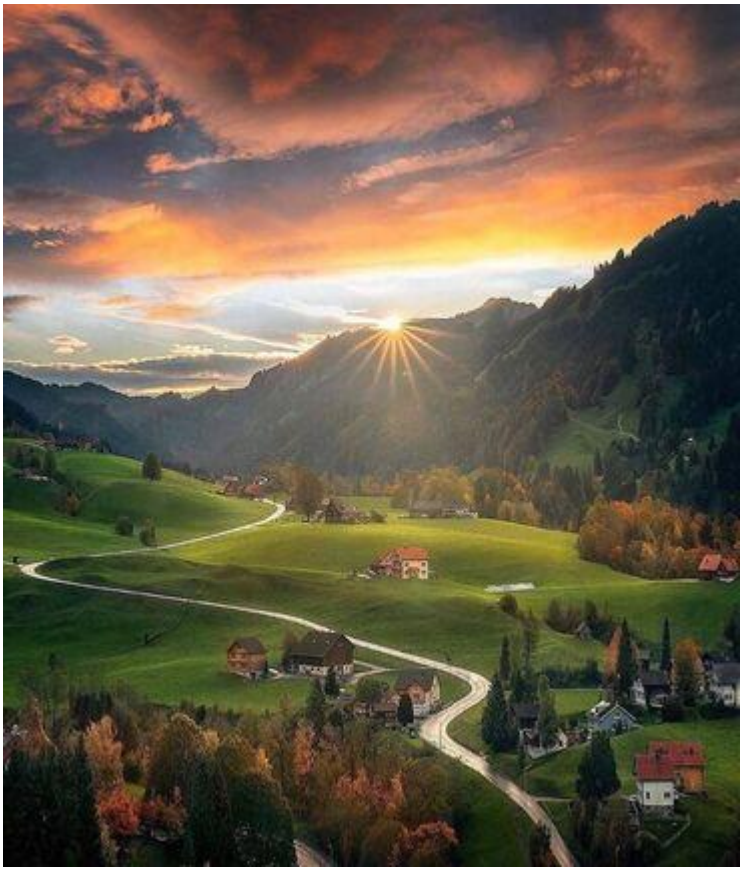
Podemos comparar este algoritmo al de redes neuronales, puesto que existe una analogía entre tener neuronas en la redes de salida para cada posible resultado de clasificación y tener un centroide para cada posible resultado de clasificación. Este algoritmo resulta mucho más sencillo tanto de implementar como de usar que el de redes neuronales, y presenta un resultado bastante

acertado, por lo que resulta muy eficiente para la tarea de clasificación en el iris dataset.

- **Segmentacion de imagenes:**

  Aplicando el k-means en imágenes, logramos crear k centroides los cuales corresponden a k colores que tendrá la imagen. Esto mediante las funciones implementadas en img_utils.py las cuales fueron explicadas anteriormente en el apartado de implementación. Cada pixel de la imagen corresponde a un color el cual sera categorizado segun los centroides calculados en k-means, para luego reconstruir la imagen con k colores.

  Imágenes originales:



Resultados Obtenidos:

- *Imagen 1:*

- *K = 2*

- *K = 4*

- *K = 8*

- *K = 16*

- *K = 32*

- *Imagen 2:*
  - *K = 2*

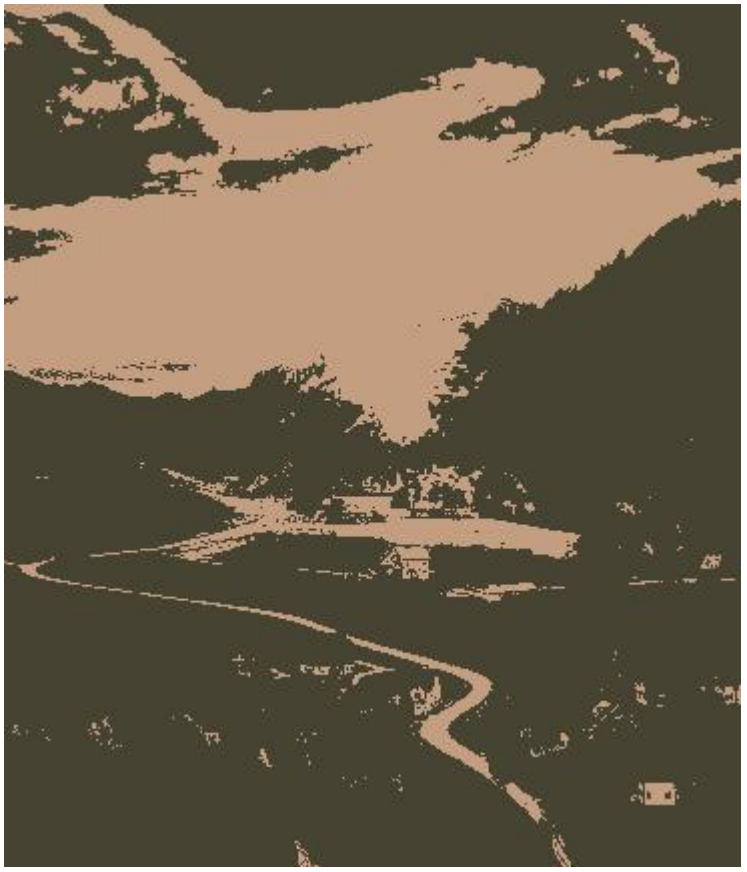
- *K = 4*

- *K = 8*

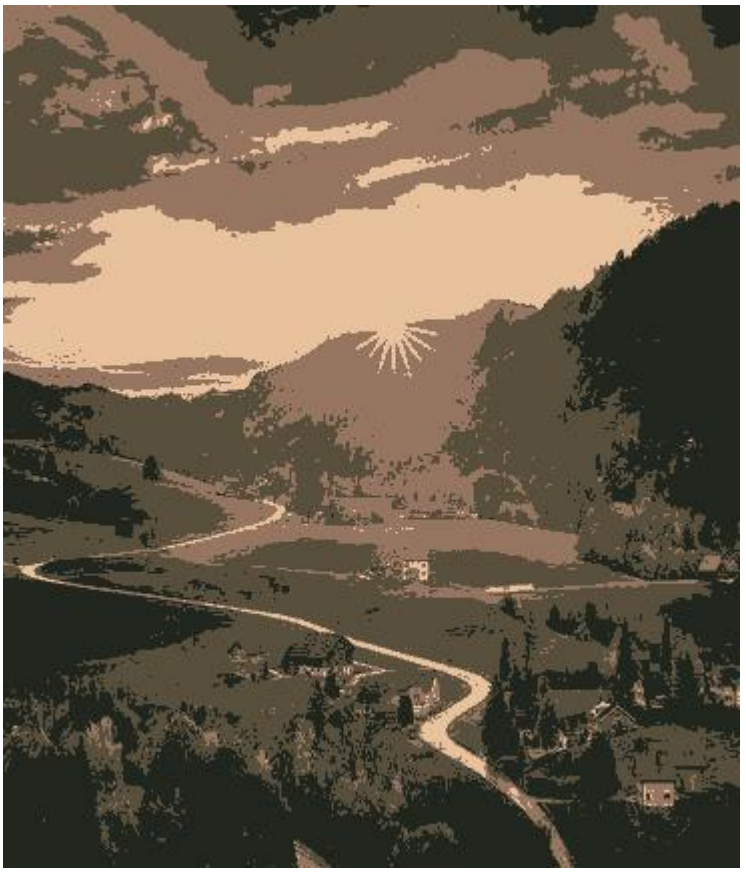
- *K = 16*

- *K = 32*

- *Imagen 3:*
    - *K = 2*



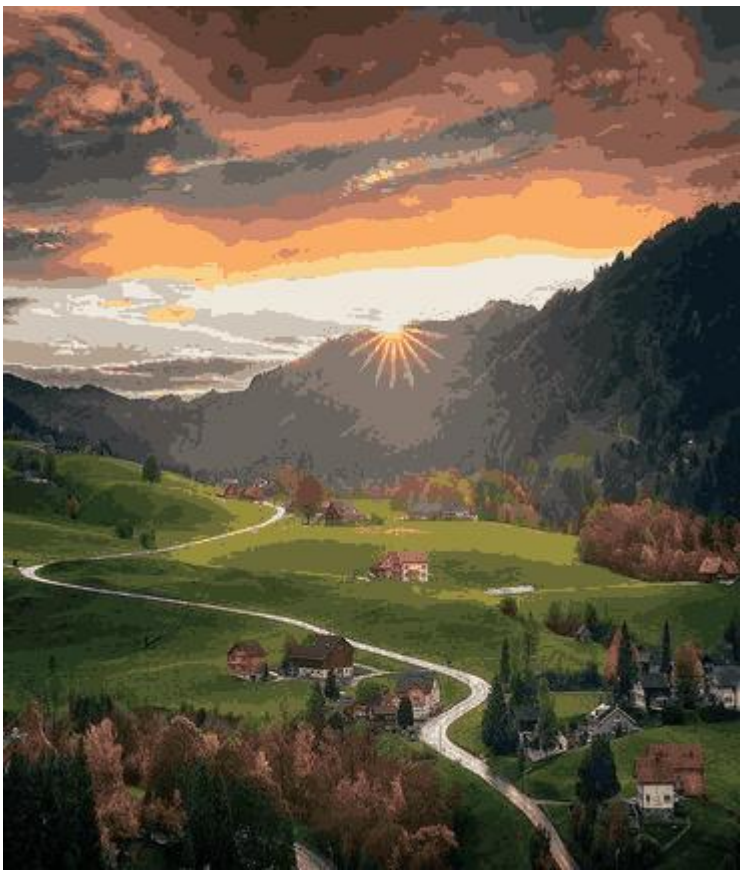
    - *K = 4*



- *K = 8*

- *K = 16*

- *K = 32*



Podemos observar que un número elevado de clusters conllevan a una mejor calidad de imagen, debido a que mientras más colores, más se pueden detallar los objetos tanto en su forma como en su naturaleza y dependiendo de los colores que tenga la imagen original, haran falta mas o menos cantidad de los mismos para conseguir una imagen que sea capaz de representar lo que representa la imagen original. En las imágenes 1 y 2, podemos apreciar como se puede notar un nivel acertado de similitud con la foto original con 8 y 4 clusters respectivamente, esto debido a las “pocas” tonalidades y pequeña paleta de colores, a diferencia de en la 3era imagen, donde es con k = 16 que se puede apreciar un parecido decente con la imagen original pero es en k = 32 donde podemos obtener una imagen satisfactoria en comparación con la original.

CONCLUSIÓN

Podemos concluir que el algoritmo k-means y su implementación en clusters es un buen algoritmo para clasificar, siempre y cuando, al igual que todos los algoritmos vistos, se utilice bajo casos donde sea competente. Para el caso de clasificación de iris dataset, el algoritmo tuvo un buen desempeño clasificando cuando el número de clusters era igual a la cantidad de posibles resultados diferentes. Este llegó a una solución mucho más rápido que otros algoritmos como el de redes neuronales, y con un desempeño bastante aceptable. Para el caso de la segmentación de imágenes, la clasificación errónea de píxeles no tiene tanto impacto en el resultado final, debido a la cantidad de píxeles que presentan las imágenes, por lo que pequeños errores no afectan en gran medida al producto final. Podemos considerar entonces a k-means como un algoritmo flexible, fácil de implementar y de buen rendimiento que es capaz de hacer clasificaciones buenas antes la parametrización correcta en datasets pequeños y capaz de tener un resultado excelente en las clasificaciones de datasets grandes donde pequeños errores no afectan al producto final.