modelo-de-clasificacion

November 5, 2024

1 Tarea 6: Ejercicio usando las técnicas de aprendizaje no supervisado de reducción de dimensionalidad y agrupación

Objetivo: Implementar un clasificador no supervisado que maximice el desempeño de clusterización sobre el espacio original de los datos y los espacios reducidos con técnicas no lineales (Isomap y MDS).

1.1 Cargar el Dataset

[ ]:
```
!wget -O dataset_wq.xlsx 'https://raw.githubusercontent.com/Negatix092/DM1/
↪bcd52726a66c65ab6d4ebeea6401517aa439ce73/UL_Clustering/dataset(wq).xlsx'
```

```
--2024-11-01 00:11:01--  https://raw.githubusercontent.com/Negatix092/DM1/bcd527
26a66c65ab6d4ebeea6401517aa439ce73/UL_Clustering/dataset(wq).xlsx
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.108.133, 185.199.109.133, 185.199.110.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 104921 (102K) [application/octet-stream]
Saving to: ‘dataset_wq.xlsx’

dataset_wq.xlsx     100%[===================>] 102.46K  --.-KB/s    in 0.004s

2024-11-01 00:11:01 (23.8 MB/s) - ‘dataset_wq.xlsx’ saved [104921/104921]
```

1.2 Crear Dataframe

[32]:
```
import pandas as pd

df = pd.read_excel('dataset_wq.xlsx')
df.head()
```

[32]:

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8

```
3   11.2  0.28  0.56  1.9  0.075  17.0  60.0  0.9980  3.16  0.58  9.8
4    7.4  0.70  0.00  1.9  0.076  11.0  34.0  0.9978  3.51  0.56  9.4
```

1.3 Reducción de Dimensionalidad con Isomap y MDS

- Implementar las dos técnicas de reducción de dimensionalidad (Isomap y MDS) y aplicarlas sobre al dataset de trabajo para obtener los espacios reducidos, dependiendo de la variación del número de componentes y otros parámetros internos de cada método.

  Se debe variar el parámetro de número de componentes al menos 3 veces para facilitar la búsqueda del mejor espacio reducido (ej: n_components = 3,5,7). Otros parámetros internos de cada método se pueden variar a su elección (ej: número de vecinos en el Isomap = 5,10). Mostrar los gráficos de los espacios reducidos para cada combinación de parámetros (Un subplot tipo grid).

[33]:
```
# Instala las librerías necesarias si aún no están instaladas
#!pip install matplotlib scikit-learn pandas openpyxl

# Importa las librerías
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.manifold import Isomap, MDS
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

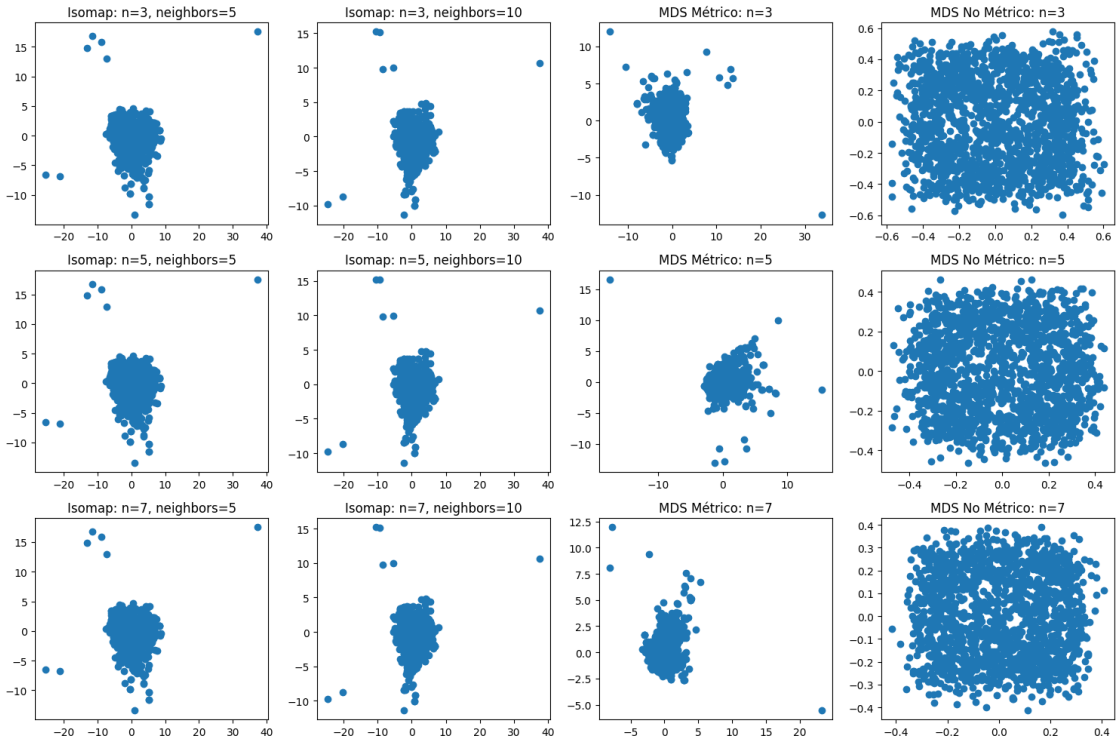
#Utilizando StandardScaler

[58]:
```
from scipy.spatial import distance_matrix

# Escala los datos
scaler = StandardScaler()
data_normalized = scaler.fit_transform(df)

# Parámetros
n_components = [3, 5, 7]  # Número de componentes para cada técnica
n_neighbors_isomap = [5, 10]  # Número de vecinos para Isomap

# Diccionarios para almacenar resultados de reducción y errores
results = {}
reduction_errors = {}

# Función para aplicar Isomap y MDS con ajuste de hiperparámetros
def apply_dimensionality_reduction(data, n_components, n_neighbors_isomap):

    for n in n_components:
        # Isomap con diferentes valores de n_neighbors y método del camino más␣
 ↪corto
        for neighbors in n_neighbors_isomap:
```

```
            isomap = Isomap(n_components=n, n_neighbors=neighbors,␣
 ↪path_method='auto', n_jobs=-1)
            data_isomap = isomap.fit_transform(data)
            # Calcular la distorsión de distancia para Isomap como error
            original_distances = distance_matrix(data, data)
            reduced_distances = distance_matrix(data_isomap, data_isomap)
            distortion_error = np.mean(np.abs(original_distances -␣
 ↪reduced_distances))

            results[f'Isomap (n={n}, neighbors={neighbors})'] = data_isomap
            reduction_errors[f'Isomap (n={n}, neighbors={neighbors})'] =␣
 ↪distortion_error


        # MDS métrico y no métrico
        for metric in [True, False]:
            mds = MDS(n_components=n, metric=metric, n_init=10, max_iter=500,␣
 ↪n_jobs=-1)
            data_mds = mds.fit_transform(data)
            # Guardar el estrés para MDS
            stress = mds.stress_
            metric_type = 'Métrico' if metric else 'No Métrico'
            results[f'MDS (n={n}, {metric_type})'] = data_mds
            reduction_errors[f'MDS (n={n}, {metric_type})'] = stress

# Aplicar reduccion de dimensionalidad
apply_dimensionality_reduction(data_normalized, n_components,␣
 ↪n_neighbors_isomap)
```

[59]:
```
import matplotlib.pyplot as plt

def visualize_dimensionality_reduction_results(results, n_components,␣
 ↪n_neighbors_isomap, figsize=(15, 10)):
    """
    Visualiza los resultados de reducción de dimensionalidad usando Isomap y MDS
    """
    fig, axs = plt.subplots(len(n_components), len(n_neighbors_isomap) + 2,␣
 ↪figsize=figsize)

    for i, n in enumerate(n_components):
        for j, neighbors in enumerate(n_neighbors_isomap):
            ax = axs[i, j]
            data_isomap = results[f'Isomap (n={n}, neighbors={neighbors})']
            ax.scatter(data_isomap[:, 0], data_isomap[:, 1])
            ax.set_title(f'Isomap: n={n}, neighbors={neighbors}')
```

```
        # Visualizacion del MDS Metrico
        ax = axs[i, -2]
        data_mds_metric = results[f'MDS (n={n}, Métrico)']
        ax.scatter(data_mds_metric[:, 0], data_mds_metric[:, 1])
        ax.set_title(f'MDS Métrico: n={n}')

        # Visualización de MDS No Métrico
        ax = axs[i, -1]
        data_mds_nonmetric = results[f'MDS (n={n}, No Métrico)']
        ax.scatter(data_mds_nonmetric[:, 0], data_mds_nonmetric[:, 1])
        ax.set_title(f'MDS No Métrico: n={n}')

    plt.tight_layout()
    plt.show()

visualize_dimensionality_reduction_results(results, n_components,␣
 ↪n_neighbors_isomap)
```



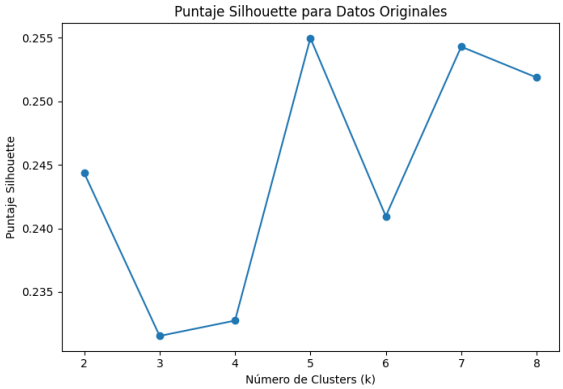
- Aplicar al menos un algoritmo de clusterización visto en clase (k-means, Hieralchical clustering, GMM) al espacio original de los datos y a los distintos espacios reducidos obtenidos por el punto anterior para validar el desempeño de clasificación de acuerdo a las métricas no supervisadas y a la variación de la cantidad de clusters posibles (k=2..8).

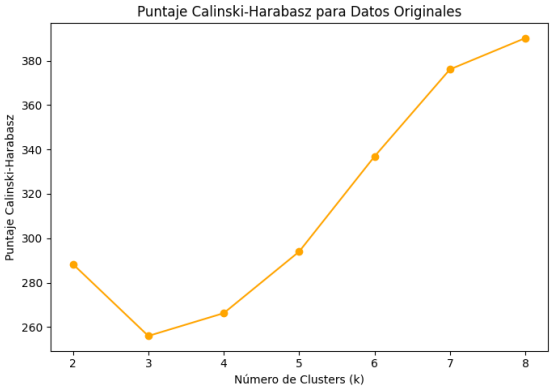
2 K-Means

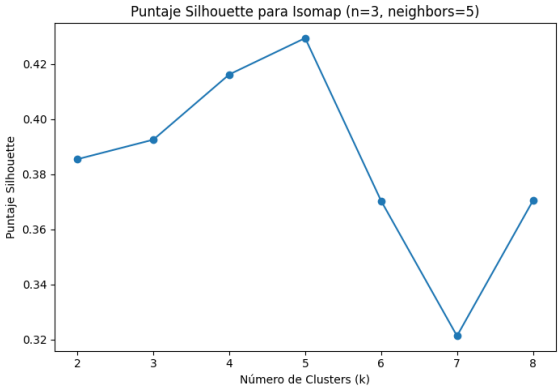
[60]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, calinski_harabasz_score
import warnings

warnings.filterwarnings("ignore")

#Diccionaro para almacenar métricas de clustering para cada configuración
clustering_metrics = {}

# Función para aplicar K-Means y calcular métricas
def apply_clustering(data, n_clusters, reduction_key, random_state=42):
    clustering_results = {}
    for k in n_clusters:
        model = KMeans(n_clusters=k, init='k-means++',␣
 ↪random_state=random_state)
        labels = model.fit_predict(data)
        silhouette_avg = silhouette_score(data, labels)
        calinski_harabasz = calinski_harabasz_score(data, labels)
        clustering_results[k] = (labels, silhouette_avg, calinski_harabasz)

        # Almacenar las métricas en el diccionario global
        if reduction_key not in clustering_metrics:
            clustering_metrics[reduction_key] = {}
        clustering_metrics[reduction_key][k] = (silhouette_avg,␣
 ↪calinski_harabasz)

    return clustering_results

# Función para graficar las métricas de clusterización
def plot_clustering_metrics(clustering_results, description):
    k_values = list(clustering_results.keys())
    silhouette_scores = [clustering_results[k][1] for k in k_values]
    calinski_harabasz_scores = [clustering_results[k][2] for k in k_values]

    # Crear gráficos para Silhouette y Calinski-Harabasz
    fig, axs = plt.subplots(1, 2, figsize=(14, 5))

    # Gráfico de Silhouette Score
    axs[0].plot(k_values, silhouette_scores, marker='o')
    axs[0].set_xlabel("Número de Clusters (k)")
    axs[0].set_ylabel("Puntaje Silhouette")
    axs[0].set_title(f"Puntaje Silhouette para {description}")
```

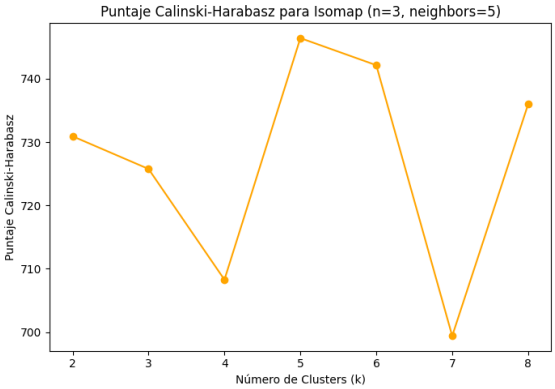
```
    # Gráfico de Calinski-Harabasz Score
    axs[1].plot(k_values, calinski_harabasz_scores, marker='o', color='orange')
    axs[1].set_xlabel("Número de Clusters (k)")
    axs[1].set_ylabel("Puntaje Calinski-Harabasz")
    axs[1].set_title(f"Puntaje Calinski-Harabasz para {description}")

    plt.tight_layout()
    plt.show()

# Función para graficar los resultados de la clusterización en el espacio␣
 ↪reducido
def plot_clustering_results(ax, data_reduced, labels, title):
    ax.scatter(data_reduced[:, 0], data_reduced[:, 1], c=labels,␣
 ↪cmap='viridis', s=10)
    ax.set_title(title)

# Función principal para aplicar reducción de dimensionalidad, clustering y␣
 ↪visualizar resultados
def dimensionality_reduction_clustering_with_visuals(results, n_components,␣
 ↪n_neighbors_isomap, n_clusters, data_normalized):
    # Aplicar clusterización a los datos originales
    original_clustering_results = apply_clustering(data_normalized, n_clusters,␣
 ↪"Datos Originales")
    plot_clustering_metrics(original_clustering_results, "Datos Originales")

    # Graficar los resultados para cada configuración de reducción de␣
 ↪dimensionalidad
    for n in n_components:
        for neighbors in n_neighbors_isomap:
            key = f'Isomap (n={n}, neighbors={neighbors})'
            data_reduced = results[key]
            reduced_clustering_results = apply_clustering(data_reduced,␣
 ↪n_clusters, key)

            # Gráficos de métricas para cada configuración
            plot_clustering_metrics(reduced_clustering_results, key)

            # Visualización de clustering en el espacio reducido para 3 valores␣
 ↪de k (por ejemplo, k=2, k=3, y k=4)
            fig, axs = plt.subplots(1, 3, figsize=(18, 5))
            k_values_for_visuals = [2, 3, 4]
            for i, k in enumerate(k_values_for_visuals):
                labels = reduced_clustering_results[k][0]
                plot_clustering_results(axs[i], data_reduced, labels, f'{key} -␣
 ↪K-Means (k={k})')
            plt.show()
```

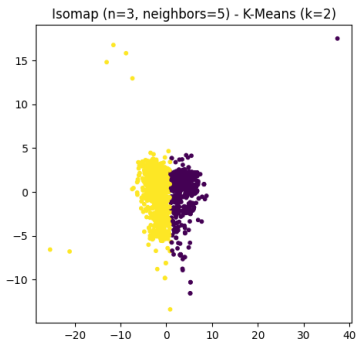
```
        # MDS Métrico
        key = f'MDS (n={n}, Métrico)'
        data_reduced = results[key]
        reduced_clustering_results = apply_clustering(data_reduced, n_clusters,␣
 ↪key)

        # Gráficos de métricas para MDS Métrico
        plot_clustering_metrics(reduced_clustering_results, key)

        # Visualización de clustering en el espacio reducido para 3 valores de␣
 ↪k (k=2, k=3, y k=4)
        fig, axs = plt.subplots(1, 3, figsize=(18, 5))
        for i, k in enumerate(k_values_for_visuals):
            labels = reduced_clustering_results[k][0]
            plot_clustering_results(axs[i], data_reduced, labels, f'{key} -␣
 ↪K-Means (k={k})')
        plt.show()

        # MDS No Métrico
        key = f'MDS (n={n}, No Métrico)'
        data_reduced = results[key]
        reduced_clustering_results = apply_clustering(data_reduced, n_clusters,␣
 ↪key)

        # Gráficos de métricas para MDS No Métrico
        plot_clustering_metrics(reduced_clustering_results, key)

        # Visualización de clustering en el espacio reducido para 3 valores de␣
 ↪k (k=2, k=3, y k=4)
        fig, axs = plt.subplots(1, 3, figsize=(18, 5))
        for i, k in enumerate(k_values_for_visuals):
            labels = reduced_clustering_results[k][0]
            plot_clustering_results(axs[i], data_reduced, labels, f'{key} -␣
 ↪K-Means (k={k})')
        plt.show()

# Parámetros de número de clusters
n_clusters = [2, 3, 4, 5, 6, 7, 8]  # Número de clusters para K-Means (puedes␣
 ↪ajustar estos valores)

# Ejecuta la función con los resultados de reducción de dimensionalidad y␣
 ↪clustering
dimensionality_reduction_clustering_with_visuals(results, n_components,␣
 ↪n_neighbors_isomap, n_clusters, data_normalized)
```

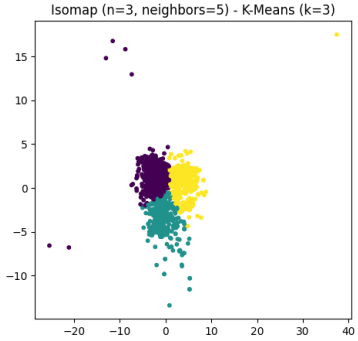

Puntaje Silhouette para Datos Originales
Puntaje Silhouette
Número de Clusters (k)

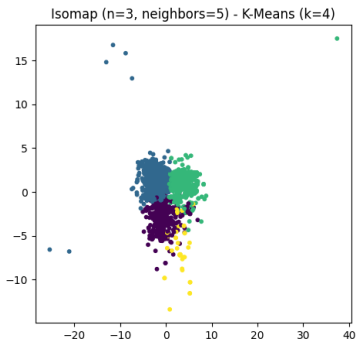

Puntaje Calinski-Harabasz para Datos Originales
Puntaje Calinski-Harabasz
Número de Clusters (k)

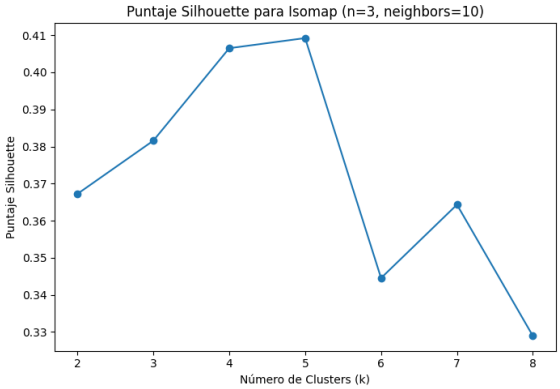

Puntaje Silhouette para Isomap (n=3, neighbors=5)
Puntaje Silhouette
Número de Clusters (k)

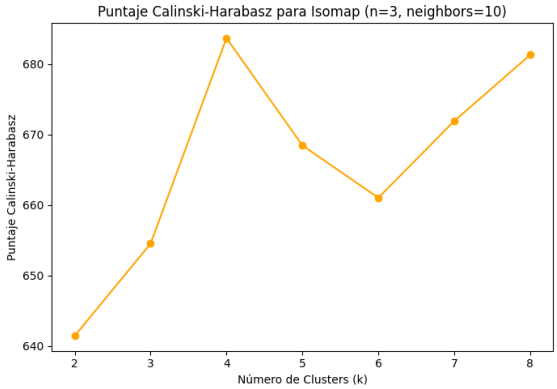

Puntaje Calinski-Harabasz para Isomap (n=3, neighbors=5)
Puntaje Calinski-Harabasz
Número de Clusters (k)

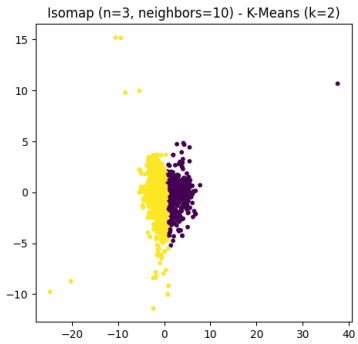

Isomap (n=3, neighbors=5) - K-Means (k=2)

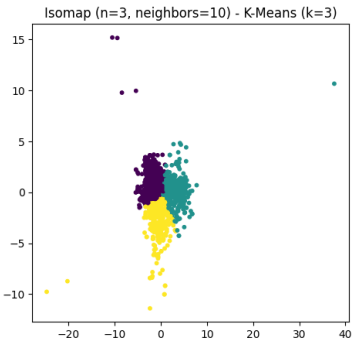

Isomap (n=3, neighbors=5) - K-Means (k=3)

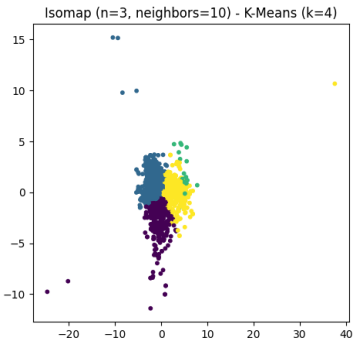

Isomap (n=3, neighbors=5) - K-Means (k=4)

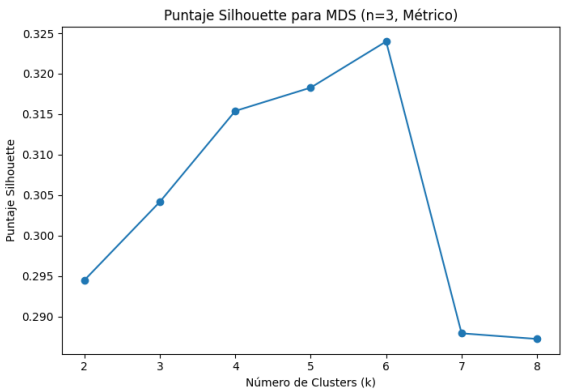

Puntaje Silhouette para Isomap (n=3, neighbors=10)
Puntaje Silhouette
Número de Clusters (k)

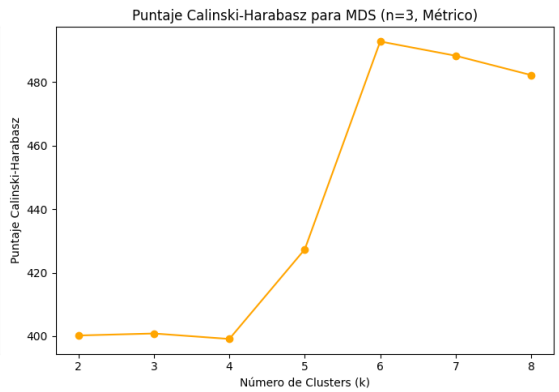

Puntaje Calinski-Harabasz para Isomap (n=3, neighbors=10)
Puntaje Calinski-Harabasz
Número de Clusters (k)

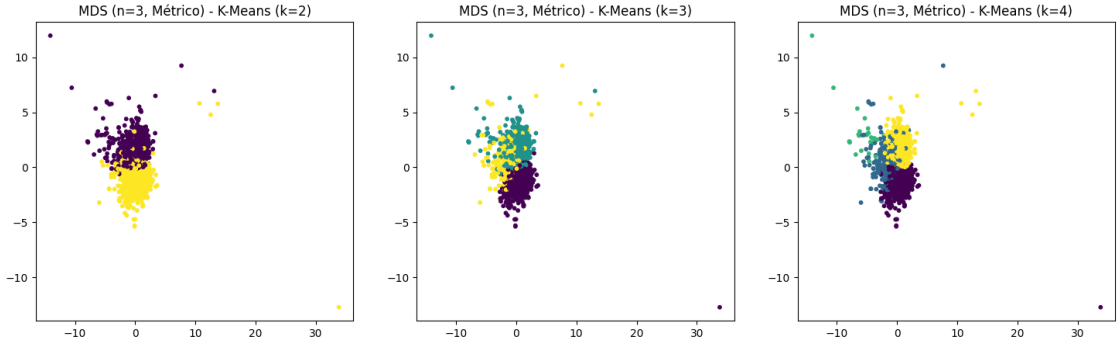

Isomap (n=3, neighbors=10) - K-Means (k=2)

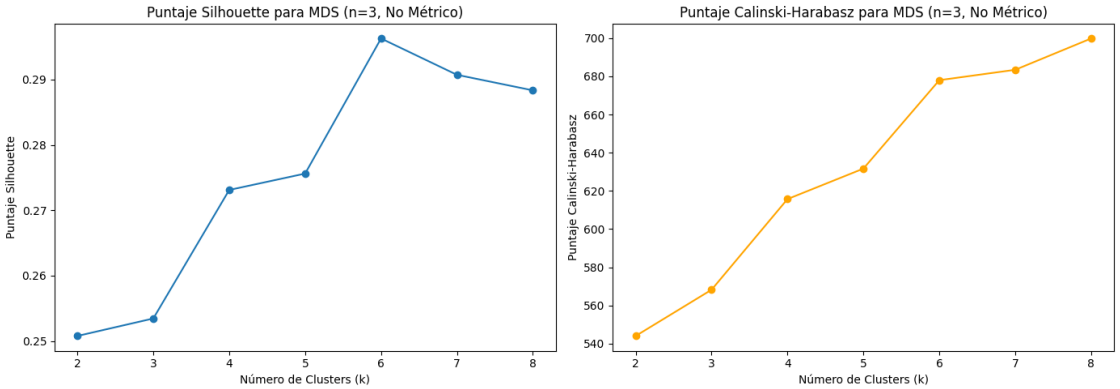

Isomap (n=3, neighbors=10) - K-Means (k=3)

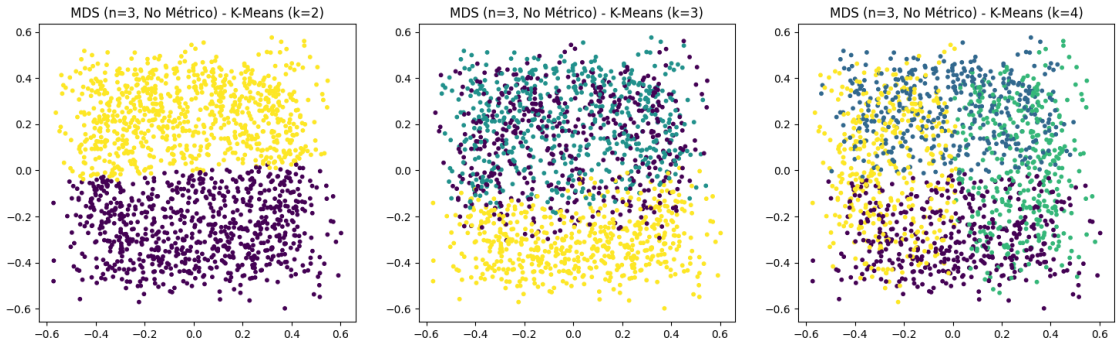

Isomap (n=3, neighbors=10) - K-Means (k=4)

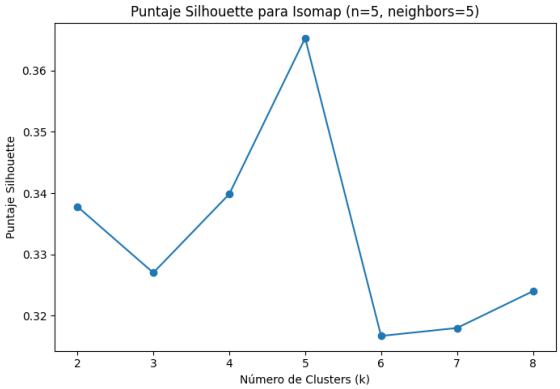

Puntaje Silhouette para MDS (n=3, Métrico)
Puntaje Silhouette
Número de Clusters (k)

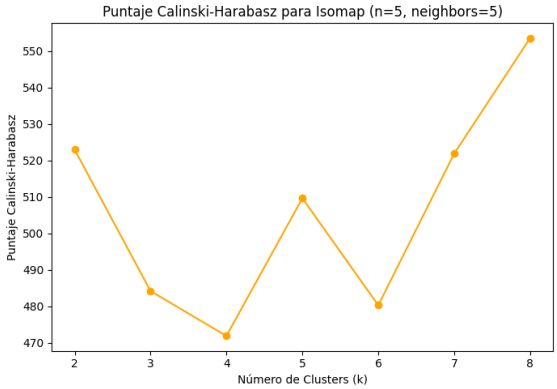

Puntaje Calinski-Harabasz para MDS (n=3, Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

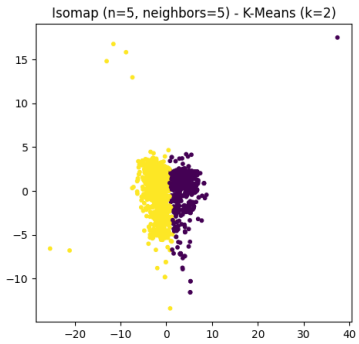

MDS (n=3, Métrico) - K-Means (k=2)
MDS (n=3, Métrico) - K-Means (k=3)
MDS (n=3, Métrico) - K-Means (k=4)

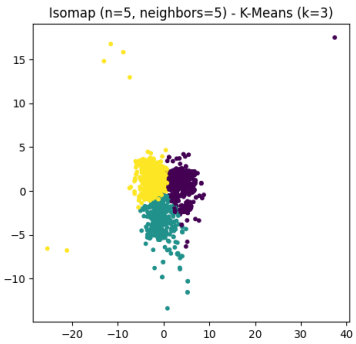

Puntaje Silhouette para MDS (n=3, No Métrico)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para MDS (n=3, No Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

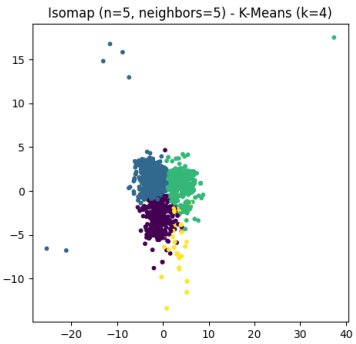

MDS (n=3, No Métrico) - K-Means (k=2)
MDS (n=3, No Métrico) - K-Means (k=3)
MDS (n=3, No Métrico) - K-Means (k=4)

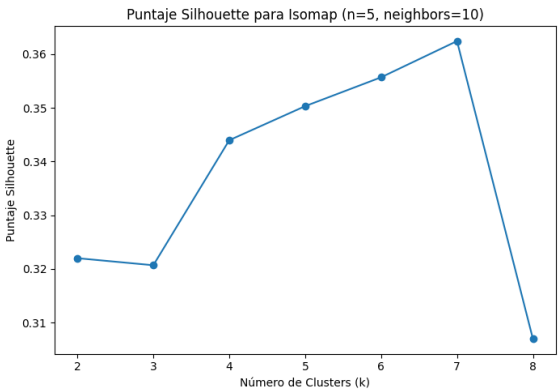

Puntaje Silhouette para Isomap (n=5, neighbors=5)
Puntaje Silhouette
Número de Clusters (k)

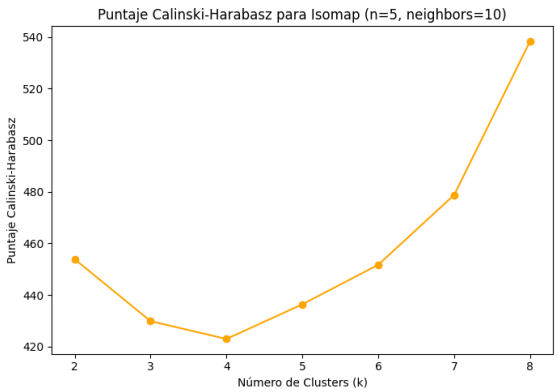

Puntaje Calinski-Harabasz para Isomap (n=5, neighbors=5)
Puntaje Calinski-Harabasz
Número de Clusters (k)

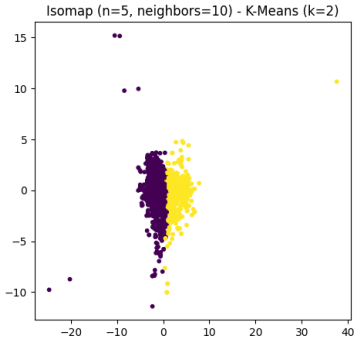

Isomap (n=5, neighbors=5) - K-Means (k=2)

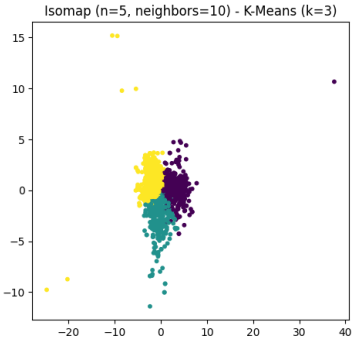

Isomap (n=5, neighbors=5) - K-Means (k=3)

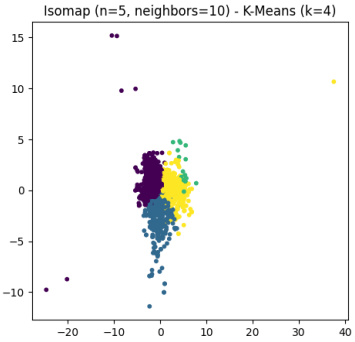

Isomap (n=5, neighbors=5) - K-Means (k=4)

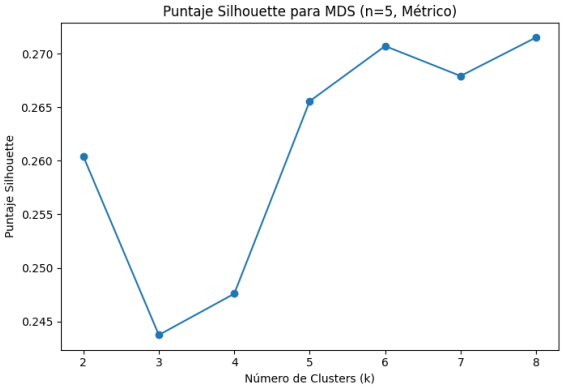

Puntaje Silhouette para Isomap (n=5, neighbors=10)
Puntaje Silhouette
Número de Clusters (k)

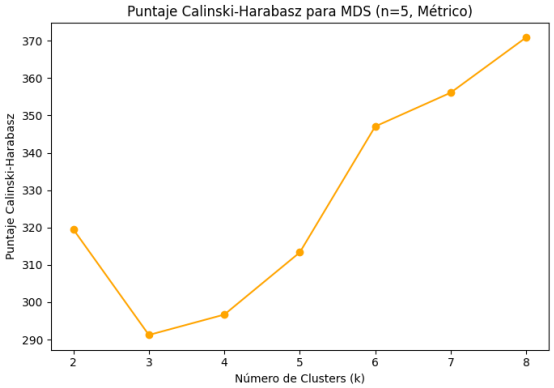

Puntaje Calinski-Harabasz para Isomap (n=5, neighbors=10)
Puntaje Calinski-Harabasz
Número de Clusters (k)

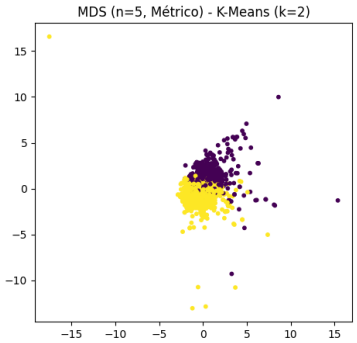

Isomap (n=5, neighbors=10) - K-Means (k=2)

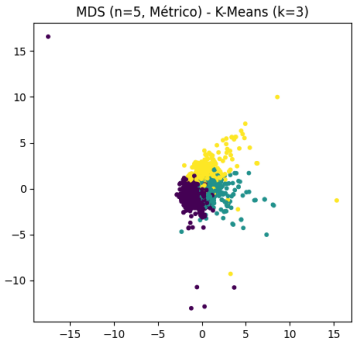

Isomap (n=5, neighbors=10) - K-Means (k=3)

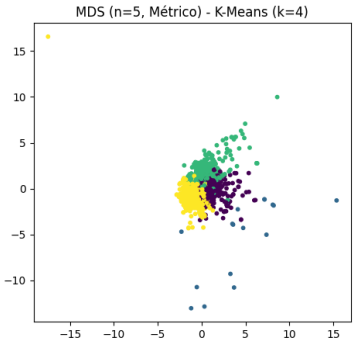

Isomap (n=5, neighbors=10) - K-Means (k=4)

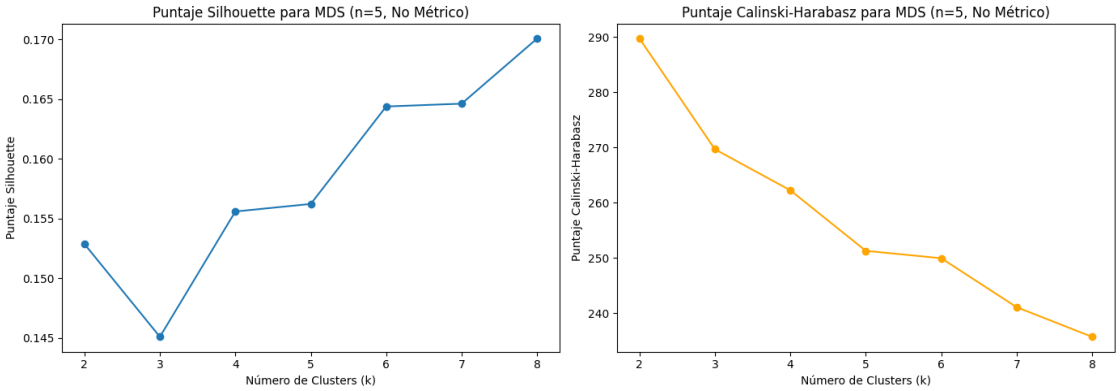

Puntaje Silhouette para MDS (n=5, Métrico)
Puntaje Silhouette
Número de Clusters (k)

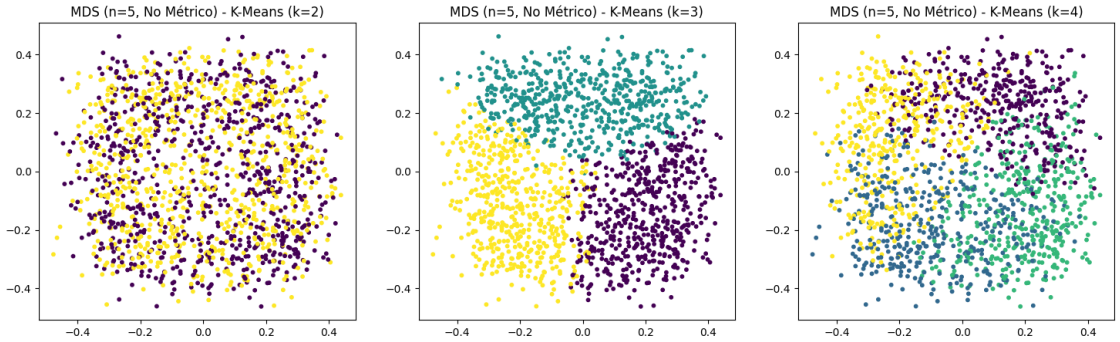

Puntaje Calinski-Harabasz para MDS (n=5, Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

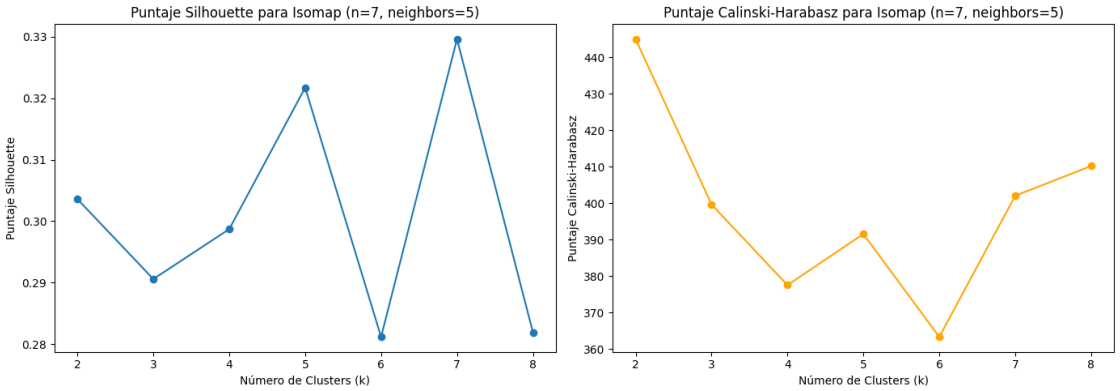

MDS (n=5, Métrico) - K-Means (k=2)

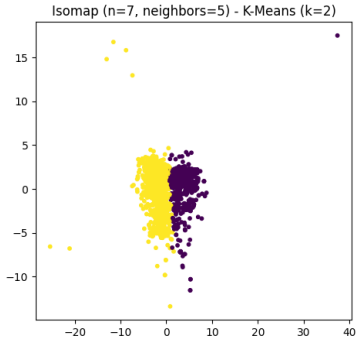

MDS (n=5, Métrico) - K-Means (k=3)

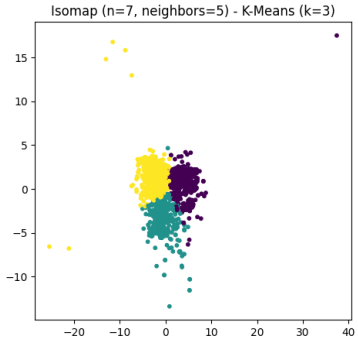

MDS (n=5, Métrico) - K-Means (k=4)

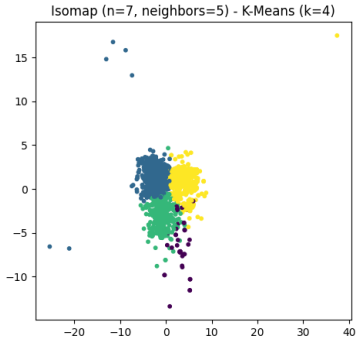

Puntaje Silhouette para MDS (n=5, No Métrico)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para MDS (n=5, No Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)


MDS (n=5, No Métrico) - K-Means (k=2)
MDS (n=5, No Métrico) - K-Means (k=3)
MDS (n=5, No Métrico) - K-Means (k=4)

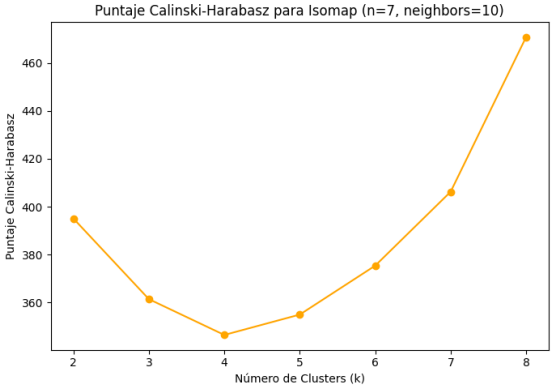

Puntaje Silhouette para Isomap (n=7, neighbors=5)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para Isomap (n=7, neighbors=5)
Puntaje Calinski-Harabasz
Número de Clusters (k)

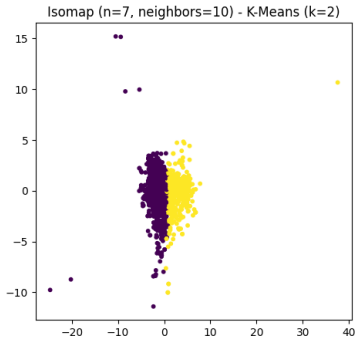

Isomap (n=7, neighbors=5) - K-Means (k=2)

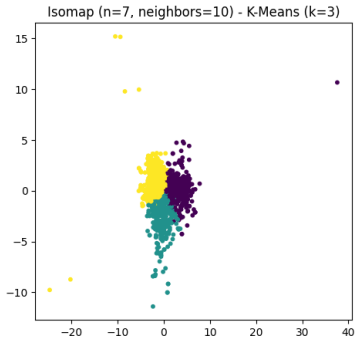

Isomap (n=7, neighbors=5) - K-Means (k=3)

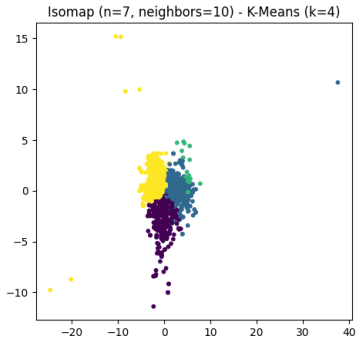

Isomap (n=7, neighbors=5) - K-Means (k=4)

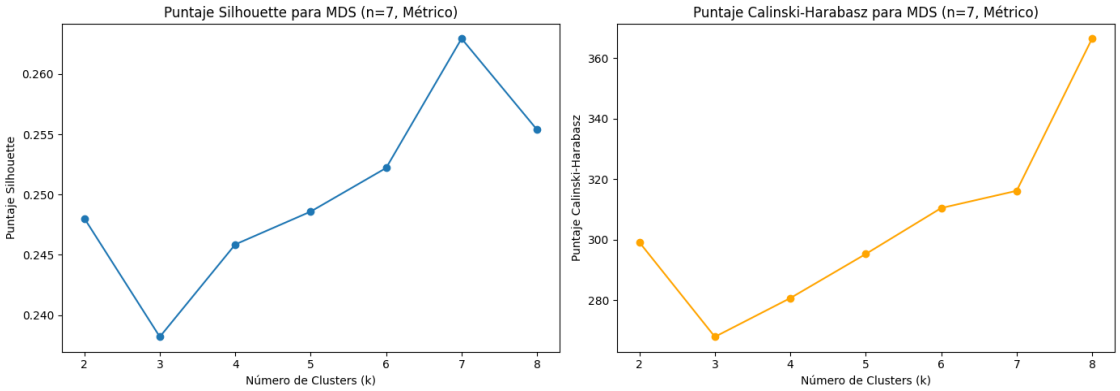

Puntaje Silhouette para Isomap (n=7, neighbors=10)
Puntaje Silhouette
Número de Clusters (k)

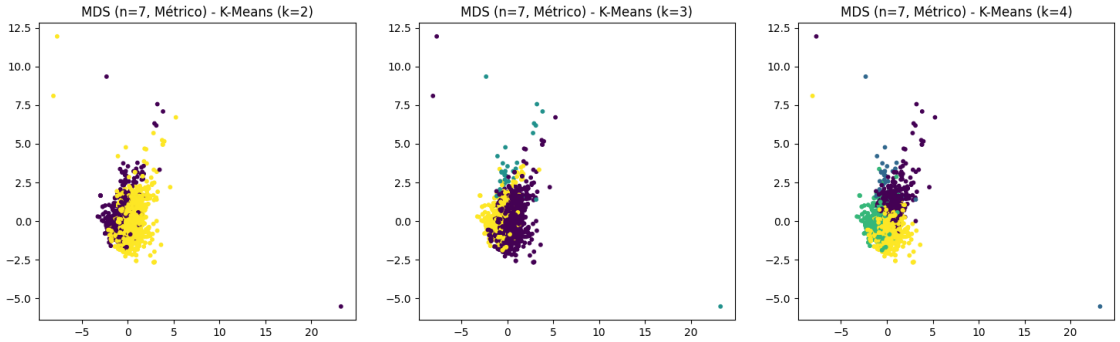

Puntaje Calinski-Harabasz para Isomap (n=7, neighbors=10)
Puntaje Calinski-Harabasz
Número de Clusters (k)

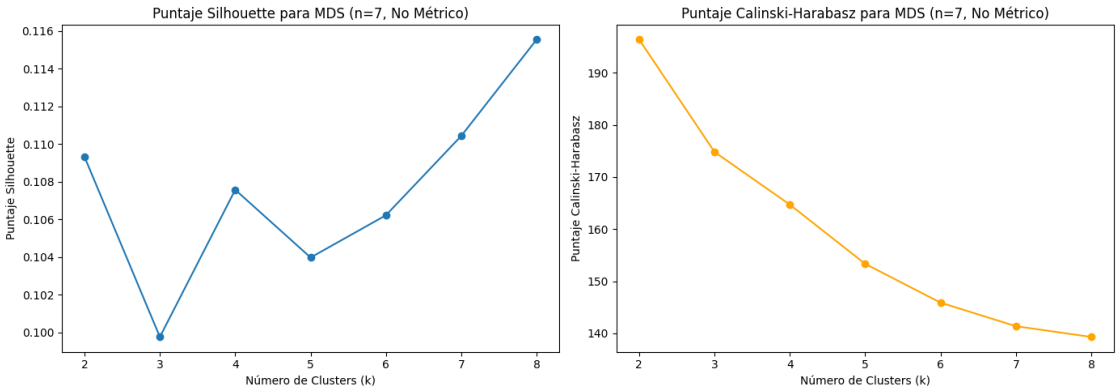

Isomap (n=7, neighbors=10) - K-Means (k=2)

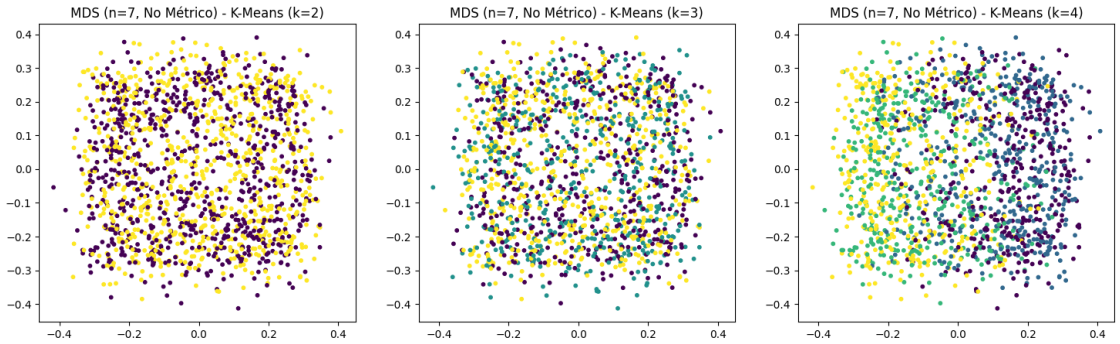

Isomap (n=7, neighbors=10) - K-Means (k=3)


Isomap (n=7, neighbors=10) - K-Means (k=4)


Puntaje Silhouette para MDS (n=7, Métrico)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para MDS (n=7, Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)


MDS (n=7, Métrico) - K-Means (k=2)
MDS (n=7, Métrico) - K-Means (k=3)
MDS (n=7, Métrico) - K-Means (k=4)


Puntaje Silhouette para MDS (n=7, No Métrico)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para MDS (n=7, No Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)



#Mejor Esquema de Clasificacion (KMeans)

[61]:
```
# Función para encontrar el mejor esquema de clustering
def find_best_scheme_kmeans(clustering_metrics, reduction_errors):
    best_scheme = None
    best_score = -np.inf

    for reduction_key, metrics in clustering_metrics.items():
        # Obtiene el error de reducción de dimensionalidad
        reduction_error = reduction_errors.get(reduction_key, 0)

        for k, (silhouette_avg, calinski_harabasz) in metrics.items():
            # Calcular un puntaje combinado para cada esquema (mayor es mejor)
            score = (silhouette_avg + calinski_harabasz) / 2 - reduction_error

            # Actualiza el mejor esquema si encuentra un puntaje más alto
            if score > best_score:
                best_score = score
                best_scheme = (reduction_key, k, silhouette_avg,␣
 ↪calinski_harabasz, reduction_error)

    return best_scheme

# Encuentra el mejor esquema de clasificación
best_scheme_kmeans = find_best_scheme_kmeans(clustering_metrics,␣
 ↪reduction_errors)

# Muestra el mejor esquema encontrado
print("Mejor esquema de clasificación (KMeans):")
print(f"Reducción de Dimensionalidad: {best_scheme_kmeans[0]}")
print(f"Número de Clusters (k): {best_scheme_kmeans[1]}")
print(f"Puntaje Silhouette Promedio: {best_scheme_kmeans[2]:.2f}")
print(f"Puntaje Calinski-Harabasz Promedio: {best_scheme_kmeans[3]:.2f}")
print(f"Error de Reducción: {best_scheme_kmeans[4]:.2f}")
```

```
Mejor esquema de clasificación (KMeans):
Reducción de Dimensionalidad: Isomap (n=3, neighbors=5)
Número de Clusters (k): 5
Puntaje Silhouette Promedio: 0.43
Puntaje Calinski-Harabasz Promedio: 746.43
Error de Reducción: 2.03
```

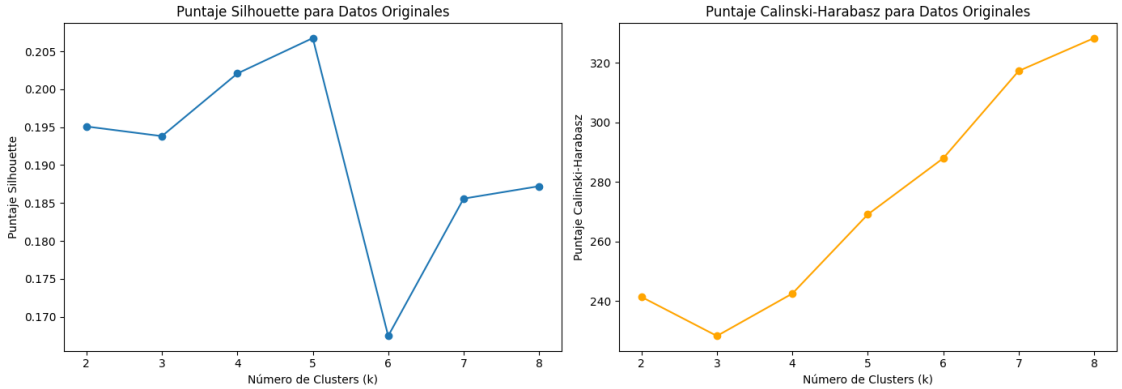
3 Hieralchical Clustering

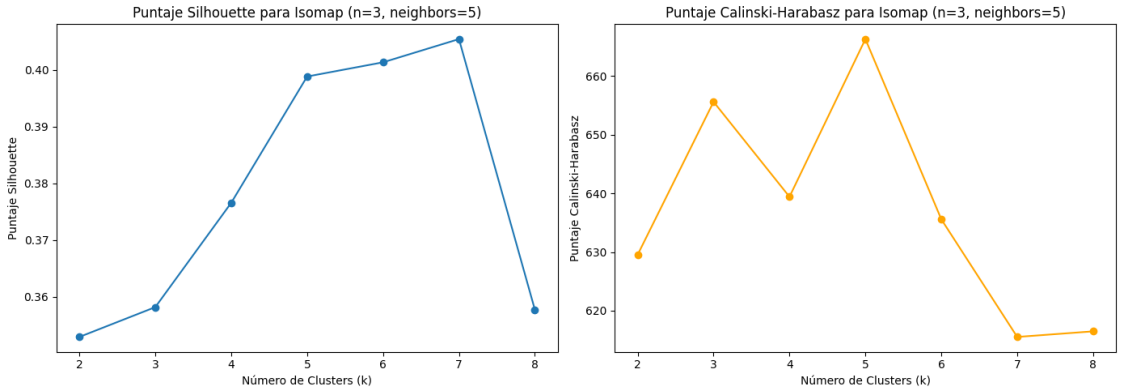
[62]:
```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score, calinski_harabasz_score
import warnings

warnings.filterwarnings("ignore")

# Diccionario para almacenar métricas de clustering para cada configuración
clustering_metrics_hierarchical = {}

# Función para aplicar Hierarchical Clustering y calcular métricas
def apply_hierarchical_clustering(data, n_clusters, reduction_key):
    clustering_results = {}
    for k in n_clusters:
        model = AgglomerativeClustering(n_clusters=k)
        labels = model.fit_predict(data)
        silhouette_avg = silhouette_score(data, labels)
        calinski_harabasz = calinski_harabasz_score(data, labels)
        clustering_results[k] = (labels, silhouette_avg, calinski_harabasz)

        # Almacenar las métricas en el diccionario global
        if reduction_key not in clustering_metrics_hierarchical:
            clustering_metrics_hierarchical[reduction_key] = {}
        clustering_metrics_hierarchical[reduction_key][k] = (silhouette_avg,␣
 ↪calinski_harabasz)

    return clustering_results

# Función para graficar las métricas de clusterización
def plot_clustering_metrics(clustering_results, description):
    k_values = list(clustering_results.keys())
    silhouette_scores = [clustering_results[k][1] for k in k_values]
    calinski_harabasz_scores = [clustering_results[k][2] for k in k_values]

    # Crear gráficos para Silhouette y Calinski-Harabasz
    fig, axs = plt.subplots(1, 2, figsize=(14, 5))
```

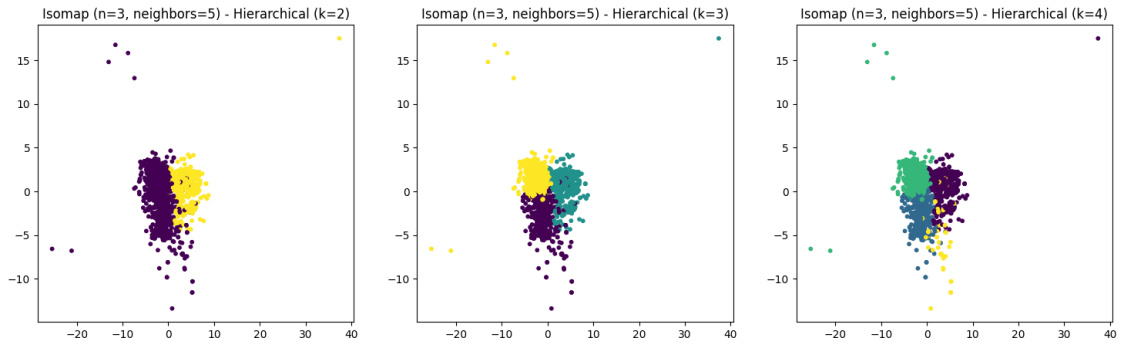
```
    # Gráfico de Silhouette Score
    axs[0].plot(k_values, silhouette_scores, marker='o')
    axs[0].set_xlabel("Número de Clusters (k)")
    axs[0].set_ylabel("Puntaje Silhouette")
    axs[0].set_title(f"Puntaje Silhouette para {description}")

    # Gráfico de Calinski-Harabasz Score
    axs[1].plot(k_values, calinski_harabasz_scores, marker='o', color='orange')
    axs[1].set_xlabel("Número de Clusters (k)")
    axs[1].set_ylabel("Puntaje Calinski-Harabasz")
    axs[1].set_title(f"Puntaje Calinski-Harabasz para {description}")

    plt.tight_layout()
    plt.show()

# Función para graficar los resultados de la clusterización en el espacio␣
 ↪reducido
def plot_clustering_results(ax, data_reduced, labels, title):
    ax.scatter(data_reduced[:, 0], data_reduced[:, 1], c=labels,␣
 ↪cmap='viridis', s=10)
    ax.set_title(title)

# Función principal para aplicar reducción de dimensionalidad, clustering y␣
 ↪visualizar resultados
def dimensionality_reduction_clustering_with_visuals(results, n_components,␣
 ↪n_neighbors_isomap, n_clusters, data_normalized):
    # Aplicar clusterización a los datos originales
    original_clustering_results =␣
 ↪apply_hierarchical_clustering(data_normalized, n_clusters, "Datos␣
 ↪Originales")
    plot_clustering_metrics(original_clustering_results, "Datos Originales")

    # Graficar los resultados para cada configuración de reducción de␣
 ↪dimensionalidad
    for n in n_components:
        for neighbors in n_neighbors_isomap:
            key = f'Isomap (n={n}, neighbors={neighbors})'
            data_reduced = results[key]
            reduced_clustering_results =␣
 ↪apply_hierarchical_clustering(data_reduced, n_clusters, key)

            # Gráficos de métricas para cada configuración
            plot_clustering_metrics(reduced_clustering_results, key)

            # Visualización de clustering en el espacio reducido para 3 valores␣
 ↪de k (por ejemplo, k=2, k=3, y k=4)
```

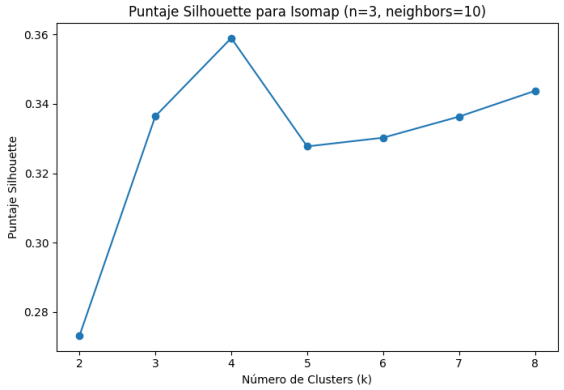
```
            fig, axs = plt.subplots(1, 3, figsize=(18, 5))
            k_values_for_visuals = [2, 3, 4]
            for i, k in enumerate(k_values_for_visuals):
                labels = reduced_clustering_results[k][0]
                plot_clustering_results(axs[i], data_reduced, labels, f'{key} -␣
↪Hierarchical (k={k})')
            plt.show()

        # MDS Métrico
        key = f'MDS (n={n}, Métrico)'
        data_reduced = results[key]
        reduced_clustering_results =␣
↪apply_hierarchical_clustering(data_reduced, n_clusters, key)

        # Gráficos de métricas para MDS Métrico
        plot_clustering_metrics(reduced_clustering_results, key)

        # Visualización de clustering en el espacio reducido para 3 valores de␣
↪k (k=2, k=3, y k=4)
        fig, axs = plt.subplots(1, 3, figsize=(18, 5))
        for i, k in enumerate(k_values_for_visuals):
            labels = reduced_clustering_results[k][0]
            plot_clustering_results(axs[i], data_reduced, labels, f'{key} -␣
↪Hierarchical (k={k})')
        plt.show()

        # MDS No Métrico
        key = f'MDS (n={n}, No Métrico)'
        data_reduced = results[key]
        reduced_clustering_results =␣
↪apply_hierarchical_clustering(data_reduced, n_clusters, key)

        # Gráficos de métricas para MDS No Métrico
        plot_clustering_metrics(reduced_clustering_results, key)

        # Visualización de clustering en el espacio reducido para 3 valores de␣
↪k (k=2, k=3, y k=4)
        fig, axs = plt.subplots(1, 3, figsize=(18, 5))
        for i, k in enumerate(k_values_for_visuals):
            labels = reduced_clustering_results[k][0]
            plot_clustering_results(axs[i], data_reduced, labels, f'{key} -␣
↪Hierarchical (k={k})')
        plt.show()

# Parámetros de número de clusters
```

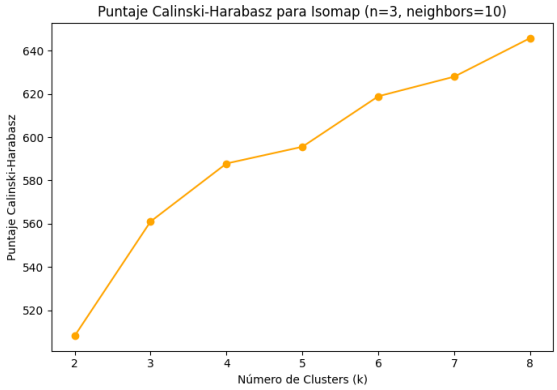
```
n_clusters = [2, 3, 4, 5, 6, 7, 8]  # Número de clusters para Agglomerative␣
↪Clustering

# Ejecuta la función con los resultados de reducción de dimensionalidad y␣
↪clustering
dimensionality_reduction_clustering_with_visuals(results, n_components,␣
↪n_neighbors_isomap, n_clusters, data_normalized)
```

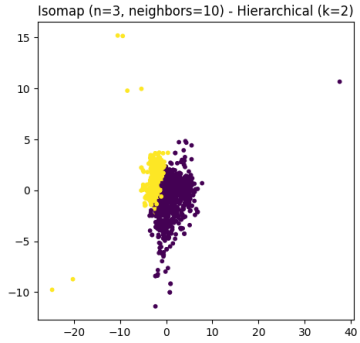


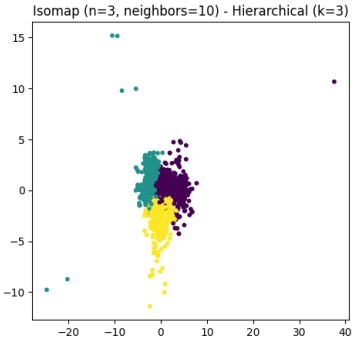


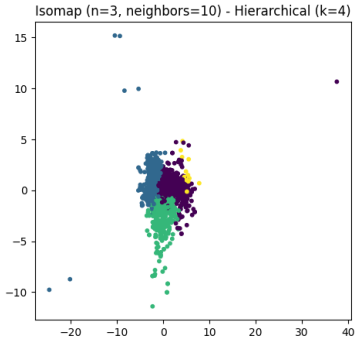


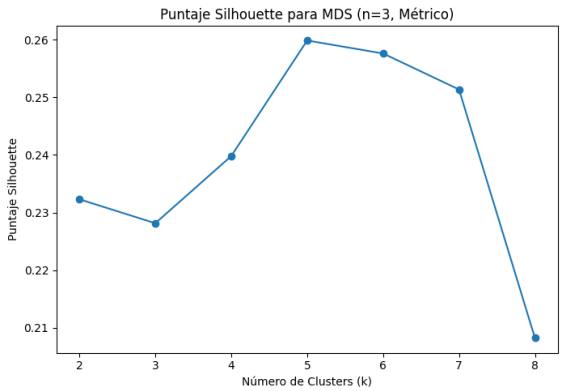

Puntaje Silhouette para Isomap (n=3, neighbors=10)
Puntaje Silhouette
Número de Clusters (k)

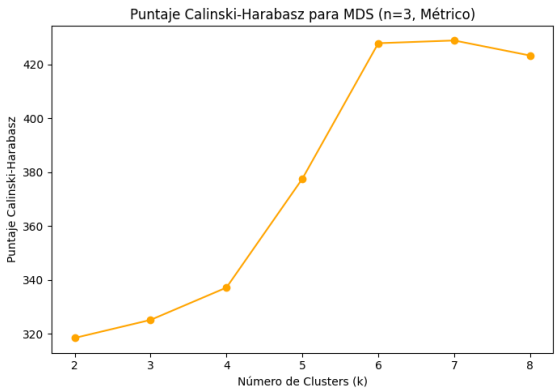

Puntaje Calinski-Harabasz para Isomap (n=3, neighbors=10)
Puntaje Calinski-Harabasz
Número de Clusters (k)

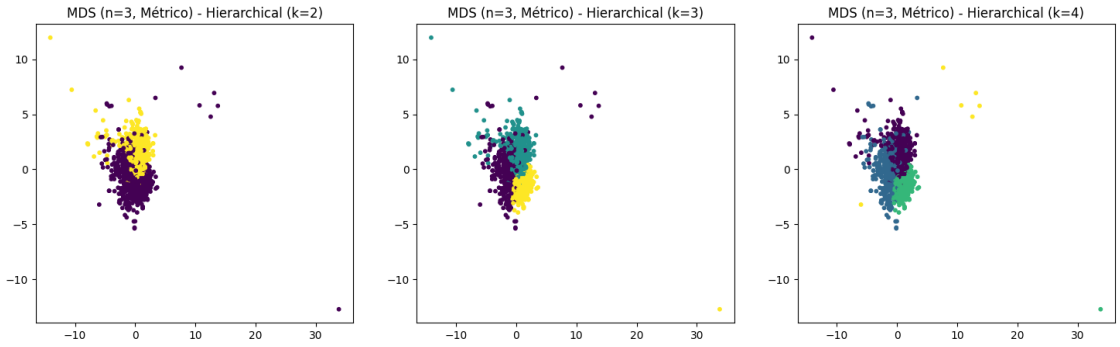

Isomap (n=3, neighbors=10) - Hierarchical (k=2)

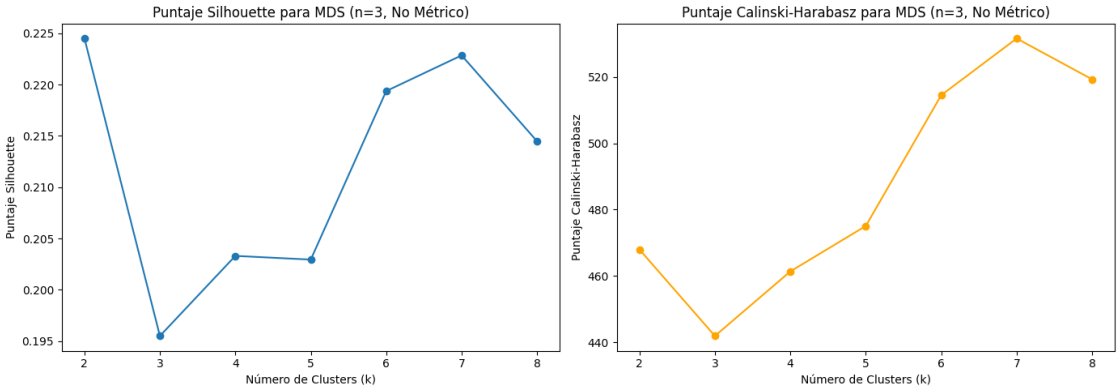

Isomap (n=3, neighbors=10) - Hierarchical (k=3)

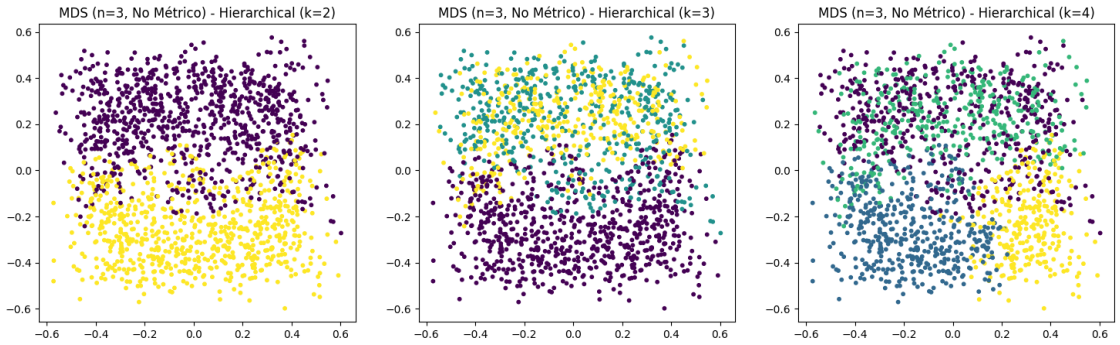

Isomap (n=3, neighbors=10) - Hierarchical (k=4)

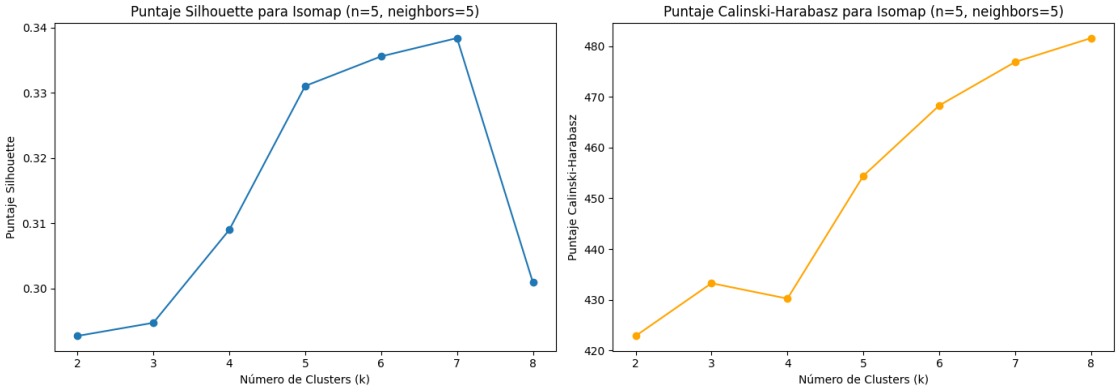

Puntaje Silhouette para MDS (n=3, Métrico)
Puntaje Silhouette
Número de Clusters (k)

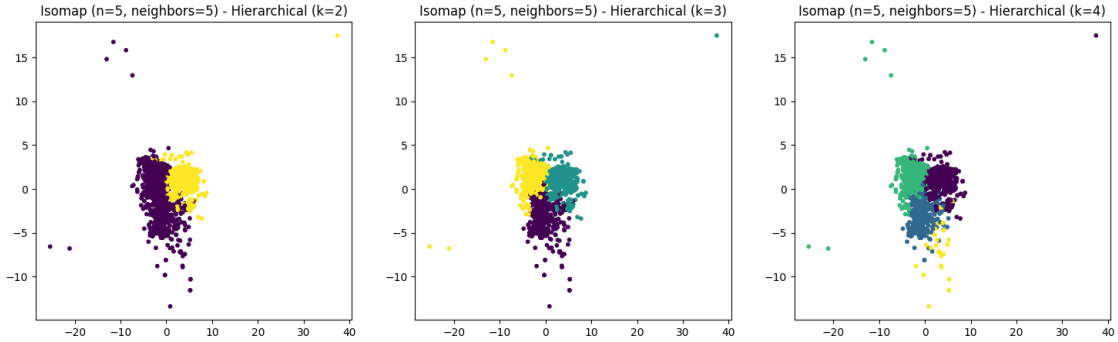

Puntaje Calinski-Harabasz para MDS (n=3, Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

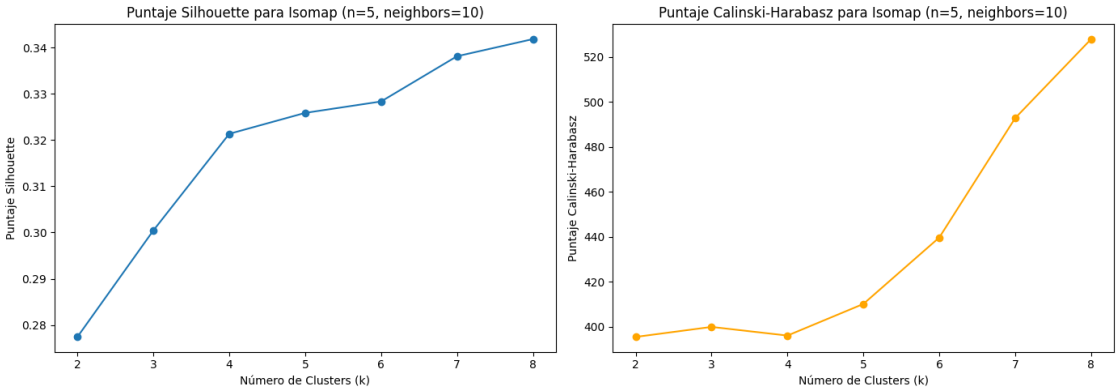

MDS (n=3, Métrico) - Hierarchical (k=2)
MDS (n=3, Métrico) - Hierarchical (k=3)
MDS (n=3, Métrico) - Hierarchical (k=4)

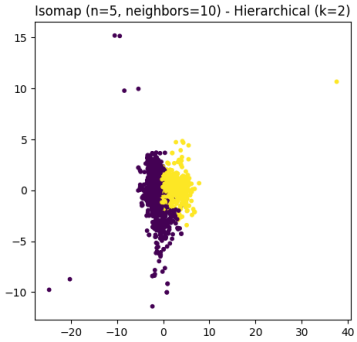

Puntaje Silhouette para MDS (n=3, No Métrico)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para MDS (n=3, No Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

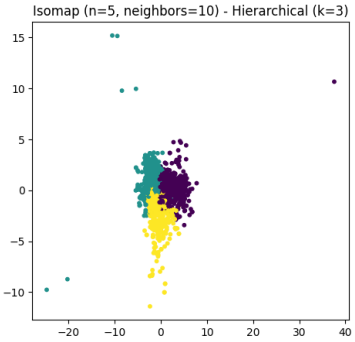

MDS (n=3, No Métrico) - Hierarchical (k=2)
MDS (n=3, No Métrico) - Hierarchical (k=3)
MDS (n=3, No Métrico) - Hierarchical (k=4)

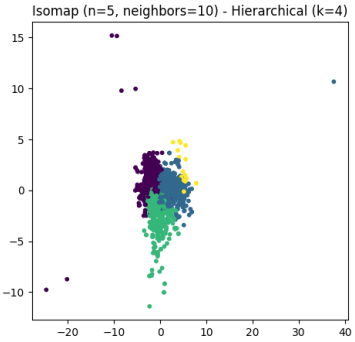

Puntaje Silhouette para Isomap (n=5, neighbors=5)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para Isomap (n=5, neighbors=5)
Puntaje Calinski-Harabasz
Número de Clusters (k)

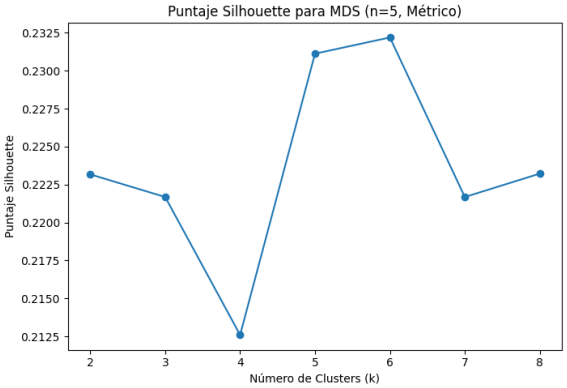

Isomap (n=5, neighbors=5) - Hierarchical (k=2)
Isomap (n=5, neighbors=5) - Hierarchical (k=3)
Isomap (n=5, neighbors=5) - Hierarchical (k=4)

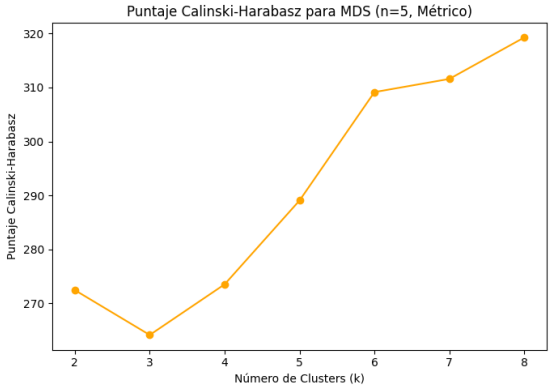

Puntaje Silhouette para Isomap (n=5, neighbors=10)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para Isomap (n=5, neighbors=10)
Puntaje Calinski-Harabasz
Número de Clusters (k)

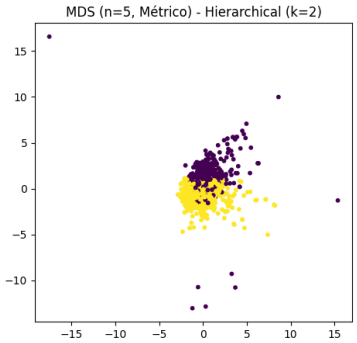

Isomap (n=5, neighbors=10) - Hierarchical (k=2)

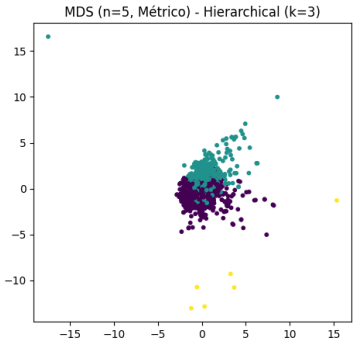

Isomap (n=5, neighbors=10) - Hierarchical (k=3)

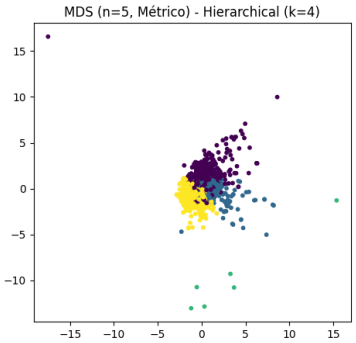

Isomap (n=5, neighbors=10) - Hierarchical (k=4)

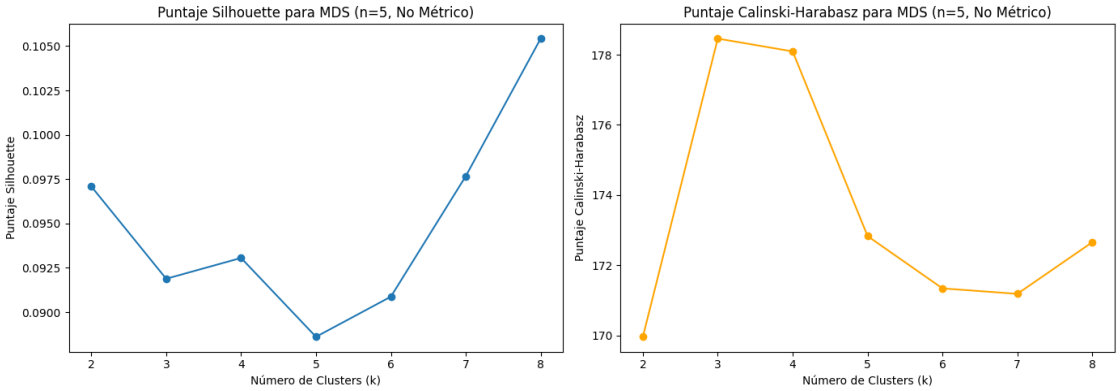

Puntaje Silhouette para MDS (n=5, Métrico)
Puntaje Silhouette
Número de Clusters (k)

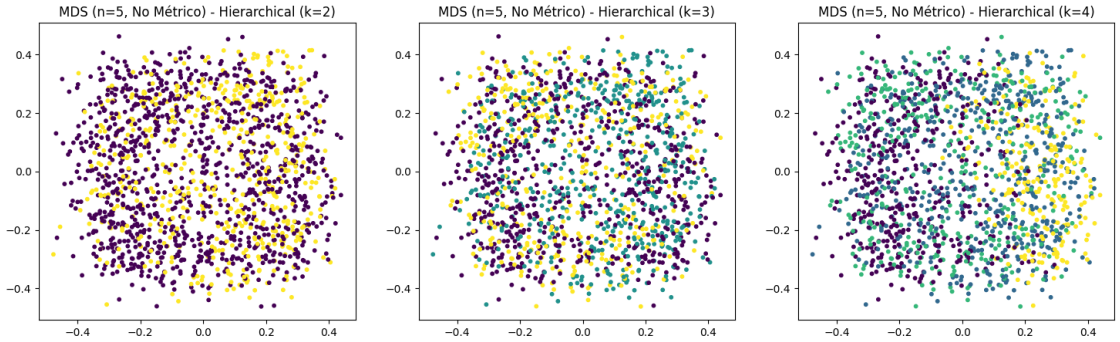

Puntaje Calinski-Harabasz para MDS (n=5, Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

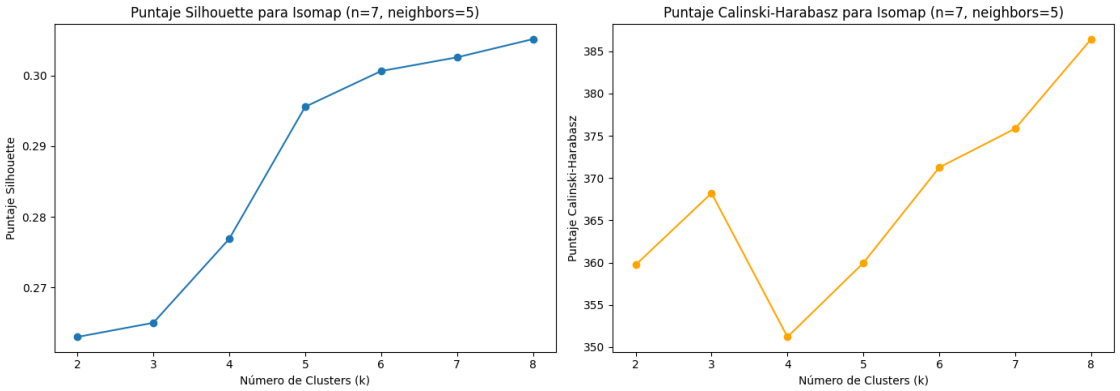

MDS (n=5, Métrico) - Hierarchical (k=2)

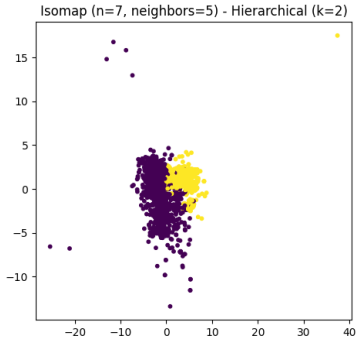

MDS (n=5, Métrico) - Hierarchical (k=3)

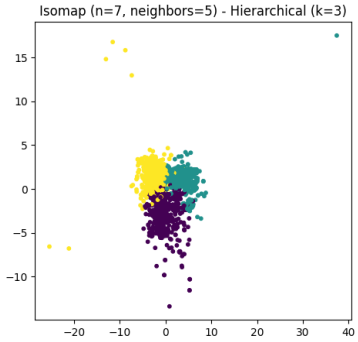

MDS (n=5, Métrico) - Hierarchical (k=4)

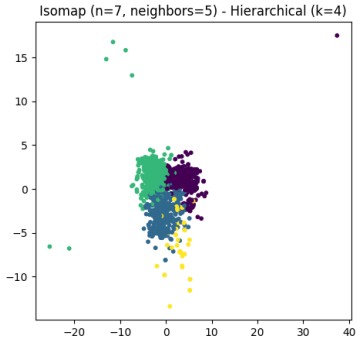

Puntaje Silhouette para MDS (n=5, No Métrico)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para MDS (n=5, No Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

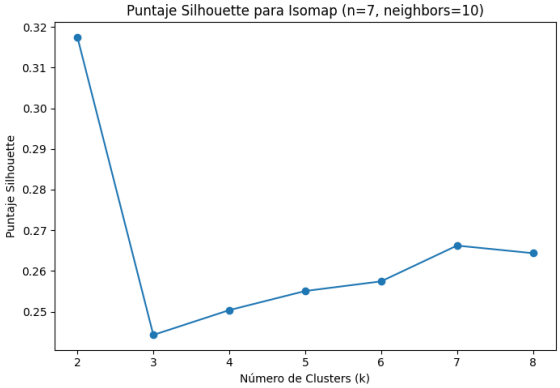

MDS (n=5, No Métrico) - Hierarchical (k=2)
MDS (n=5, No Métrico) - Hierarchical (k=3)
MDS (n=5, No Métrico) - Hierarchical (k=4)

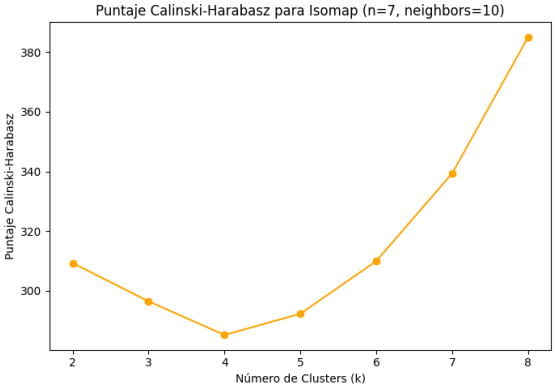

Puntaje Silhouette para Isomap (n=7, neighbors=5)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para Isomap (n=7, neighbors=5)
Puntaje Calinski-Harabasz
Número de Clusters (k)

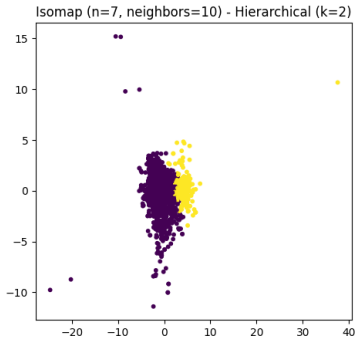

Isomap (n=7, neighbors=5) - Hierarchical (k=2)

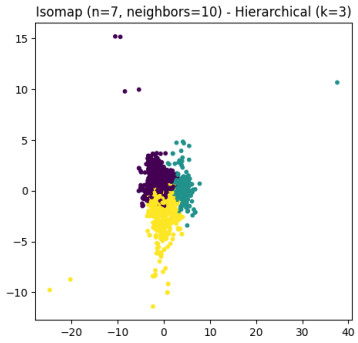

Isomap (n=7, neighbors=5) - Hierarchical (k=3)

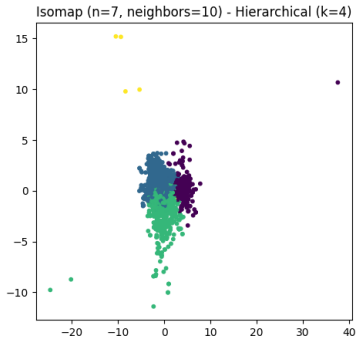

Isomap (n=7, neighbors=5) - Hierarchical (k=4)

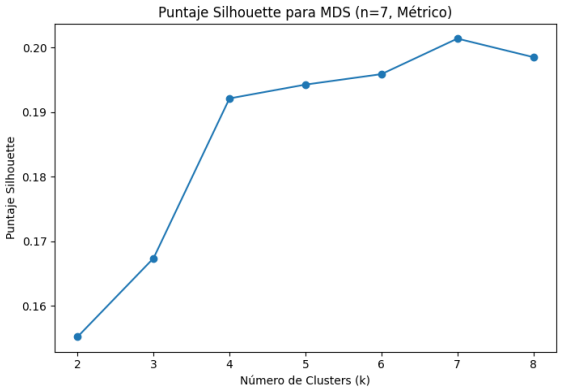

Puntaje Silhouette para Isomap (n=7, neighbors=10)
Puntaje Silhouette
Número de Clusters (k)

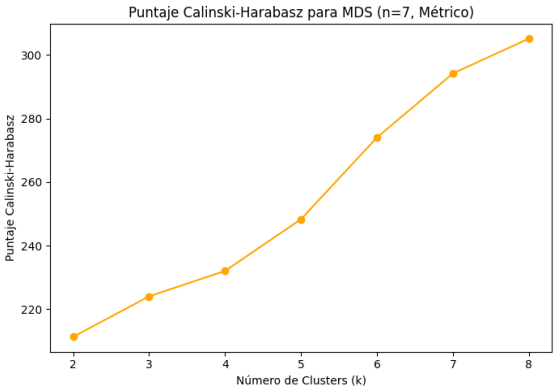

Puntaje Calinski-Harabasz para Isomap (n=7, neighbors=10)
Puntaje Calinski-Harabasz
Número de Clusters (k)

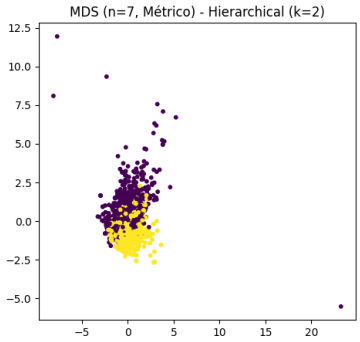

Isomap (n=7, neighbors=10) - Hierarchical (k=2)

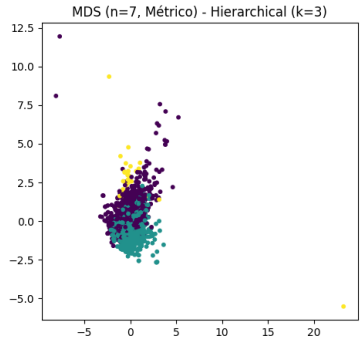

Isomap (n=7, neighbors=10) - Hierarchical (k=3)

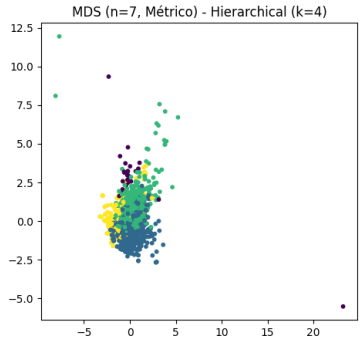

Isomap (n=7, neighbors=10) - Hierarchical (k=4)

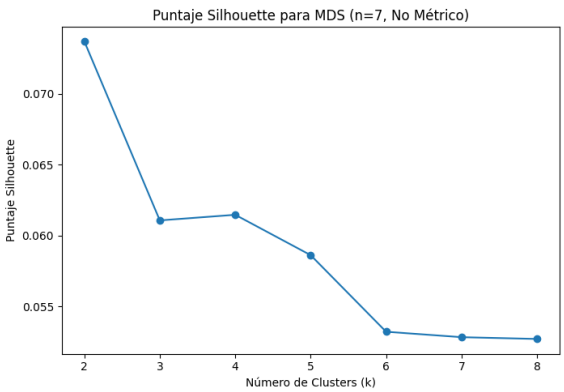

Puntaje Silhouette para MDS (n=7, Métrico)
Puntaje Silhouette
Número de Clusters (k)

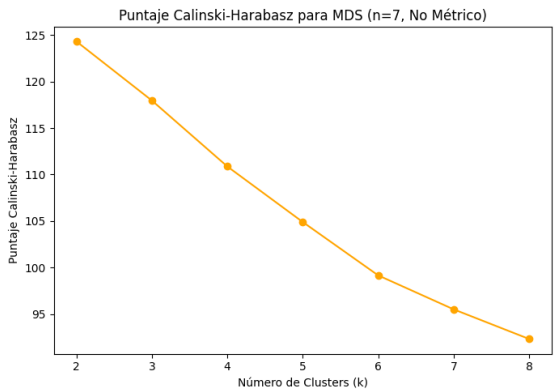

Puntaje Calinski-Harabasz para MDS (n=7, Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

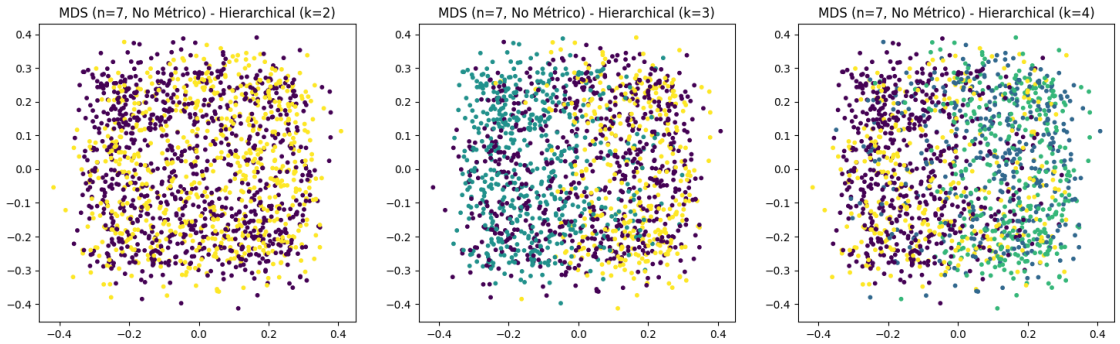

MDS (n=7, Métrico) - Hierarchical (k=2)


MDS (n=7, Métrico) - Hierarchical (k=3)


MDS (n=7, Métrico) - Hierarchical (k=4)


Puntaje Silhouette para MDS (n=7, No Métrico)
Puntaje Silhouette
Número de Clusters (k)


Puntaje Calinski-Harabasz para MDS (n=7, No Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)



#Mejor Esquema de Clasificacion (Hierarchical)

[63]:
```
# Función para encontrar el mejor esquema de clustering
def find_best_scheme_hierarchical(clustering_metrics_hierarchical,␣
 ↪reduction_errors):
    best_scheme = None
    best_score = -np.inf

    for reduction_key, metrics in clustering_metrics_hierarchical.items():
        # Obtiene el error de reducción de dimensionalidad
        reduction_error = reduction_errors.get(reduction_key, 0)

        for k, (silhouette_avg, calinski_harabasz) in metrics.items():
            # Calcular un puntaje combinado para cada esquema (mayor es mejor)
            score = (silhouette_avg + calinski_harabasz) / 2 - reduction_error

            # Actualiza el mejor esquema si encuentra un puntaje más alto
            if score > best_score:
                best_score = score
                best_scheme = (reduction_key, k, silhouette_avg,␣
 ↪calinski_harabasz, reduction_error)

    return best_scheme

# Encuentra el mejor esquema de clasificación
best_scheme_hierarchical =␣
 ↪find_best_scheme_hierarchical(clustering_metrics_hierarchical,␣
 ↪reduction_errors)

# Muestra el mejor esquema encontrado
print("Mejor esquema de clasificación (Hierarchical):")
print(f"Reducción de Dimensionalidad: {best_scheme_hierarchical[0]}")
print(f"Número de Clusters (k): {best_scheme_hierarchical[1]}")
print(f"Puntaje Silhouette Promedio: {best_scheme_hierarchical[2]:.2f}")
```

```
print(f"Puntaje Calinski-Harabasz Promedio: {best_scheme_hierarchical[3]:.2f}")
print(f"Error de Reducción: {best_scheme_hierarchical[4]:.2f}")
```

```
Mejor esquema de clasificación (Hierarchical):
Reducción de Dimensionalidad: Isomap (n=3, neighbors=5)
Número de Clusters (k): 5
Puntaje Silhouette Promedio: 0.40
Puntaje Calinski-Harabasz Promedio: 666.29
Error de Reducción: 2.03
```

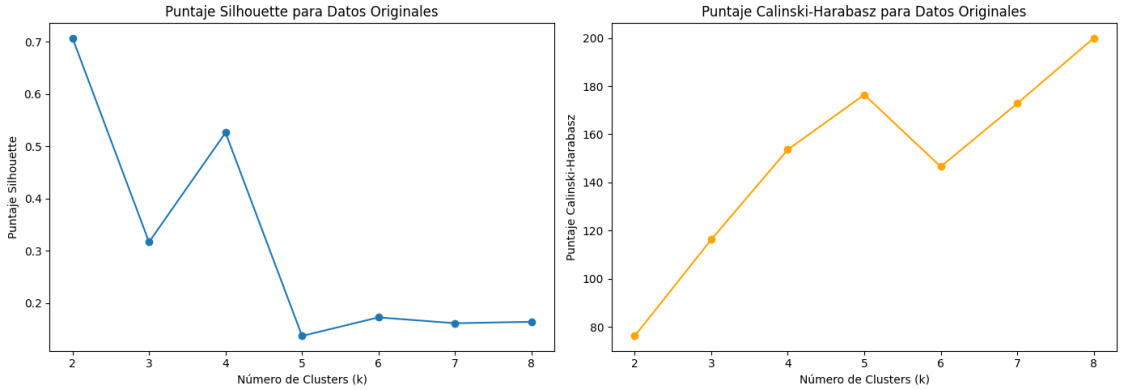
4 GMM

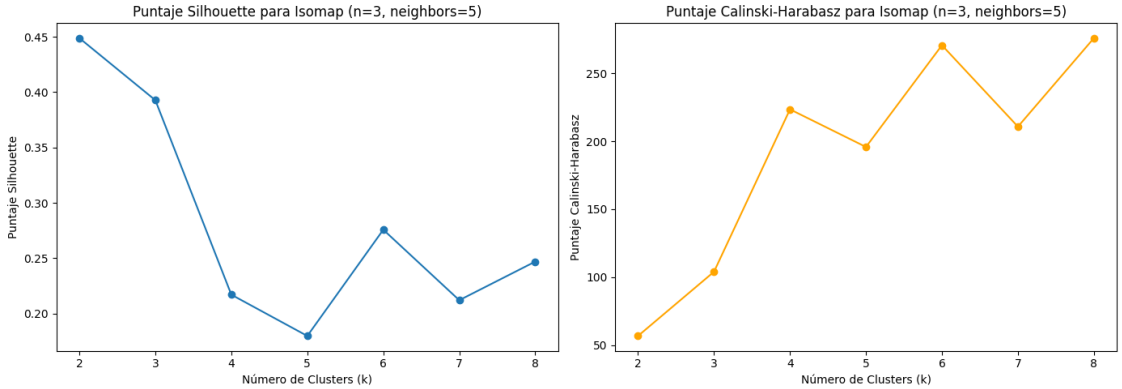
[64]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score, calinski_harabasz_score
import warnings

warnings.filterwarnings("ignore")

# Diccionario para almacenar métricas de clustering para cada configuración
clustering_metrics_gmm = {}

# Función para aplicar Gaussian Mixture Models (GMM) y calcular métricas
def apply_gmm_clustering(data, n_clusters, reduction_key, random_state=42):
    clustering_results = {}
    for k in n_clusters:
        model = GaussianMixture(n_components=k, random_state=random_state)
        labels = model.fit_predict(data)
        silhouette_avg = silhouette_score(data, labels)
        calinski_harabasz = calinski_harabasz_score(data, labels)
        clustering_results[k] = (labels, silhouette_avg, calinski_harabasz)

        # Almacenar las métricas en el diccionario global
        if reduction_key not in clustering_metrics_gmm:
            clustering_metrics_gmm[reduction_key] = {}
        clustering_metrics_gmm[reduction_key][k] = (silhouette_avg,␣
 ↪calinski_harabasz)

    return clustering_results

# Función para graficar las métricas de clusterización
def plot_clustering_metrics(clustering_results, description):
    k_values = list(clustering_results.keys())
    silhouette_scores = [clustering_results[k][1] for k in k_values]
    calinski_harabasz_scores = [clustering_results[k][2] for k in k_values]
```

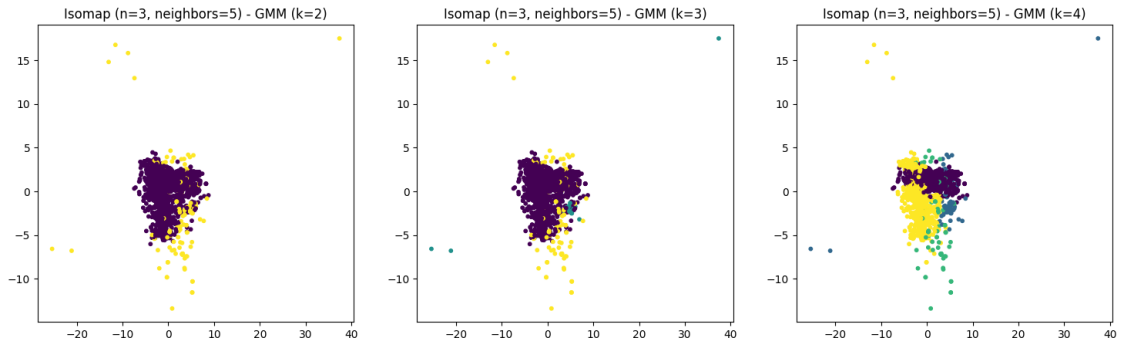
```
    # Crear gráficos para Silhouette y Calinski-Harabasz
    fig, axs = plt.subplots(1, 2, figsize=(14, 5))

    # Gráfico de Silhouette Score
    axs[0].plot(k_values, silhouette_scores, marker='o')
    axs[0].set_xlabel("Número de Clusters (k)")
    axs[0].set_ylabel("Puntaje Silhouette")
    axs[0].set_title(f"Puntaje Silhouette para {description}")

    # Gráfico de Calinski-Harabasz Score
    axs[1].plot(k_values, calinski_harabasz_scores, marker='o', color='orange')
    axs[1].set_xlabel("Número de Clusters (k)")
    axs[1].set_ylabel("Puntaje Calinski-Harabasz")
    axs[1].set_title(f"Puntaje Calinski-Harabasz para {description}")

    plt.tight_layout()
    plt.show()

# Función para graficar los resultados de la clusterización en el espacio␣
 ↪reducido
def plot_clustering_results(ax, data_reduced, labels, title):
    ax.scatter(data_reduced[:, 0], data_reduced[:, 1], c=labels,␣
 ↪cmap='viridis', s=10)
    ax.set_title(title)

# Función principal para aplicar reducción de dimensionalidad, clustering y␣
 ↪visualizar resultados
def dimensionality_reduction_clustering_with_visuals(results, n_components,␣
 ↪n_neighbors_isomap, n_clusters, data_normalized):
    # Aplicar clusterización a los datos originales
    original_clustering_results = apply_gmm_clustering(data_normalized,␣
 ↪n_clusters, "Datos Originales")
    plot_clustering_metrics(original_clustering_results, "Datos Originales")

    # Graficar los resultados para cada configuración de reducción de␣
 ↪dimensionalidad
    for n in n_components:
        for neighbors in n_neighbors_isomap:
            key = f'Isomap (n={n}, neighbors={neighbors})'
            data_reduced = results[key]
            reduced_clustering_results = apply_gmm_clustering(data_reduced,␣
 ↪n_clusters, key)

            # Gráficos de métricas para cada configuración
            plot_clustering_metrics(reduced_clustering_results, key)
```

```
            # Visualización de clustering en el espacio reducido para 3 valores␣
↪de k (por ejemplo, k=2, k=3, y k=4)
            fig, axs = plt.subplots(1, 3, figsize=(18, 5))
            k_values_for_visuals = [2, 3, 4]
            for i, k in enumerate(k_values_for_visuals):
                labels = reduced_clustering_results[k][0]
                plot_clustering_results(axs[i], data_reduced, labels, f'{key} -␣
↪GMM (k={k})')
            plt.show()

        # MDS Métrico
        key = f'MDS (n={n}, Métrico)'
        data_reduced = results[key]
        reduced_clustering_results = apply_gmm_clustering(data_reduced,␣
↪n_clusters, key)

        # Gráficos de métricas para MDS Métrico
        plot_clustering_metrics(reduced_clustering_results, key)

        # Visualización de clustering en el espacio reducido para 3 valores de␣
↪k (k=2, k=3, y k=4)
        fig, axs = plt.subplots(1, 3, figsize=(18, 5))
        for i, k in enumerate(k_values_for_visuals):
            labels = reduced_clustering_results[k][0]
            plot_clustering_results(axs[i], data_reduced, labels, f'{key} - GMM␣
↪(k={k})')
        plt.show()

        # MDS No Métrico
        key = f'MDS (n={n}, No Métrico)'
        data_reduced = results[key]
        reduced_clustering_results = apply_gmm_clustering(data_reduced,␣
↪n_clusters, key)

        # Gráficos de métricas para MDS No Métrico
        plot_clustering_metrics(reduced_clustering_results, key)

        # Visualización de clustering en el espacio reducido para 3 valores de␣
↪k (k=2, k=3, y k=4)
        fig, axs = plt.subplots(1, 3, figsize=(18, 5))
        for i, k in enumerate(k_values_for_visuals):
            labels = reduced_clustering_results[k][0]
            plot_clustering_results(axs[i], data_reduced, labels, f'{key} - GMM␣
↪(k={k})')
        plt.show()
```

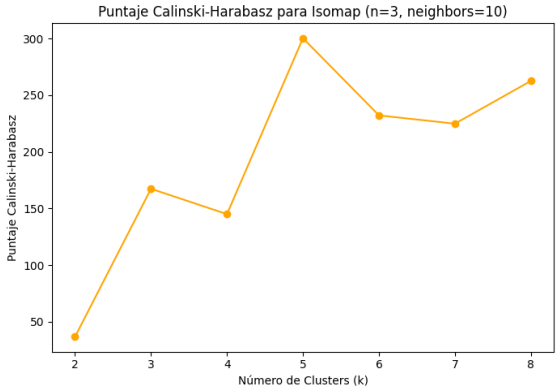
```
# Parámetros de número de clusters
n_clusters = [2, 3, 4, 5, 6, 7, 8]  # Número de clusters para GMM

# Ejecuta la función con los resultados de reducción de dimensionalidad y␣
 ↪clustering
dimensionality_reduction_clustering_with_visuals(results, n_components,␣
 ↪n_neighbors_isomap, n_clusters, data_normalized)
```

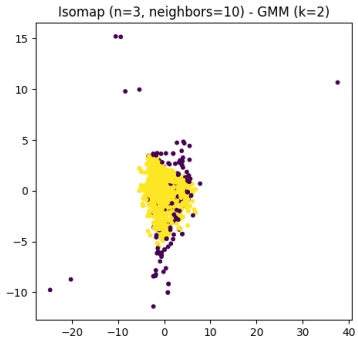


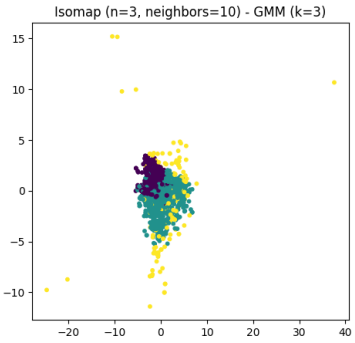


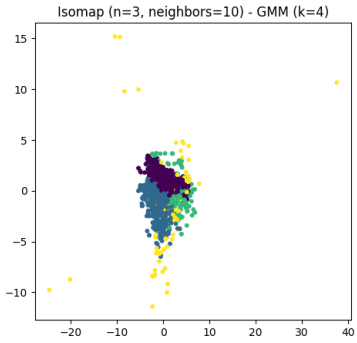


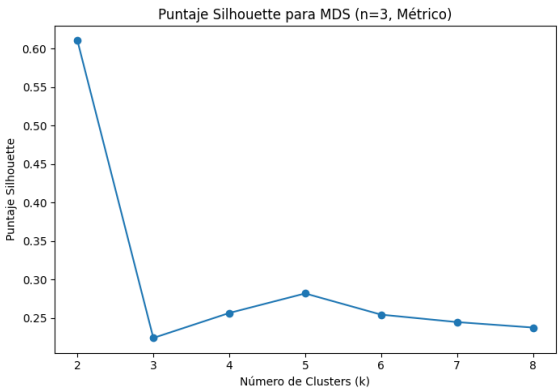

Puntaje Silhouette para Isomap (n=3, neighbors=10)
Puntaje Silhouette
Número de Clusters (k)

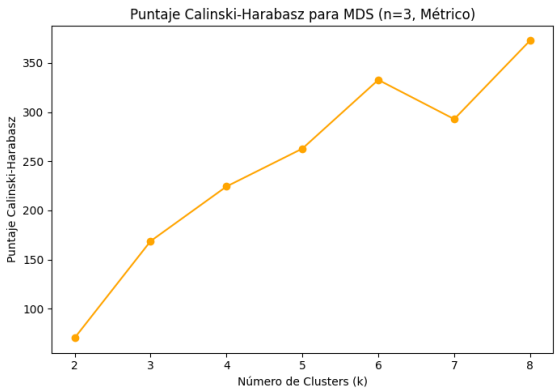

Puntaje Calinski-Harabasz para Isomap (n=3, neighbors=10)
Puntaje Calinski-Harabasz
Número de Clusters (k)

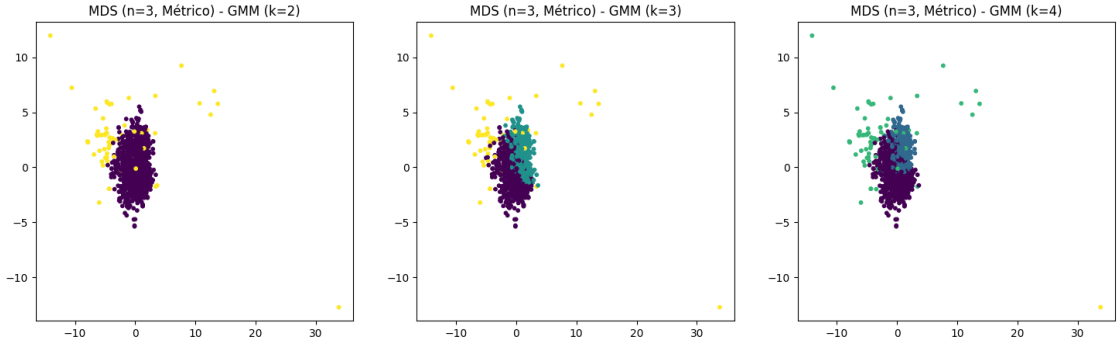

Isomap (n=3, neighbors=10) - GMM (k=2)

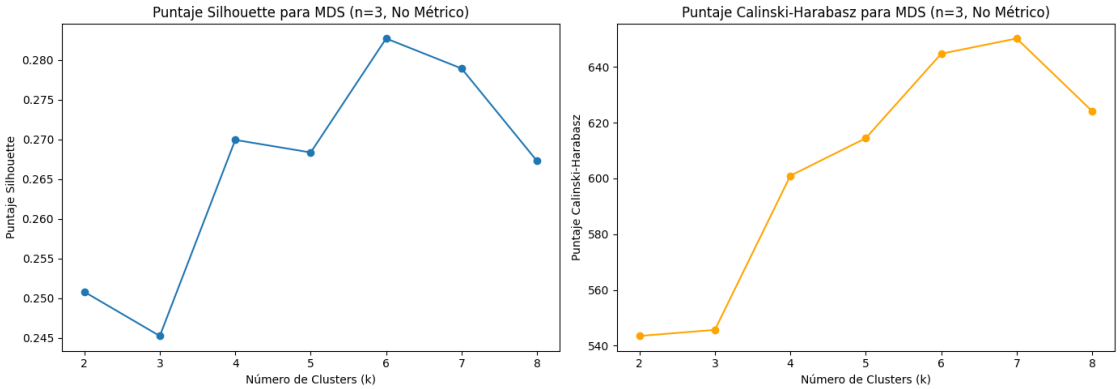

Isomap (n=3, neighbors=10) - GMM (k=3)

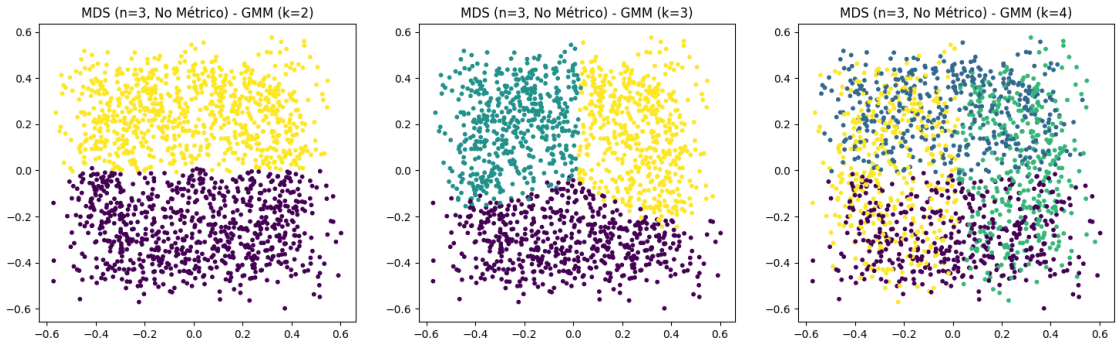

Isomap (n=3, neighbors=10) - GMM (k=4)

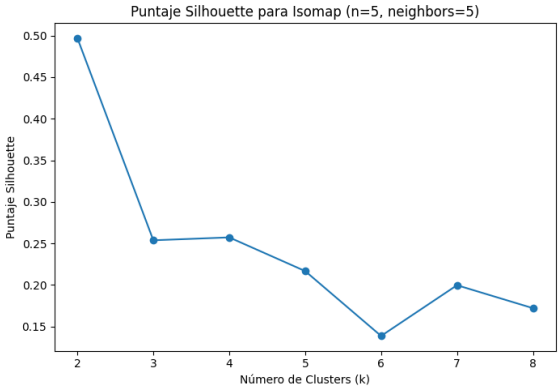

Puntaje Silhouette para MDS (n=3, Métrico)
Puntaje Silhouette
Número de Clusters (k)

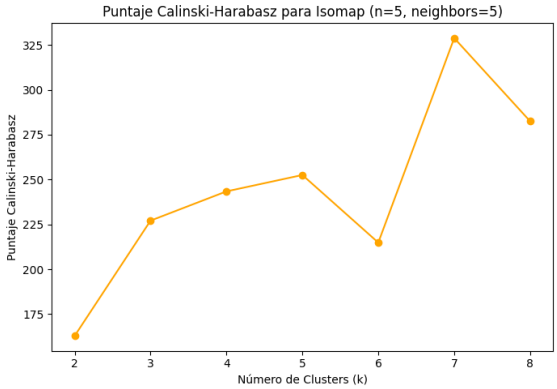

Puntaje Calinski-Harabasz para MDS (n=3, Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

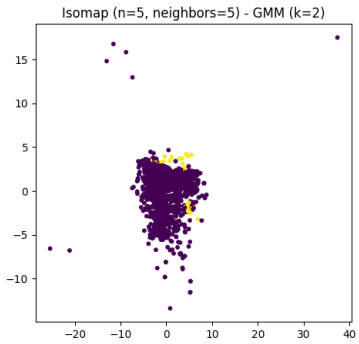

MDS (n=3, Métrico) - GMM (k=2)
MDS (n=3, Métrico) - GMM (k=3)
MDS (n=3, Métrico) - GMM (k=4)

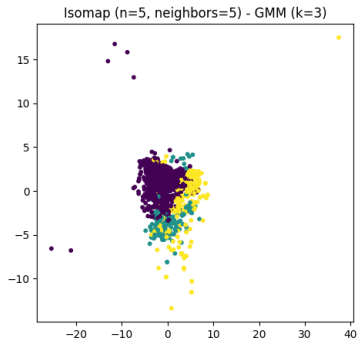

Puntaje Silhouette para MDS (n=3, No Métrico)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para MDS (n=3, No Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

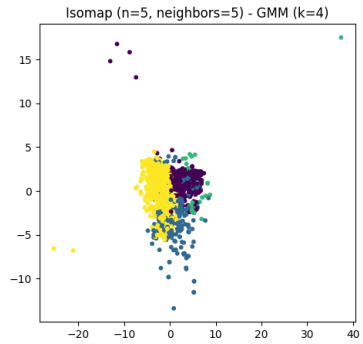

MDS (n=3, No Métrico) - GMM (k=2)
MDS (n=3, No Métrico) - GMM (k=3)
MDS (n=3, No Métrico) - GMM (k=4)

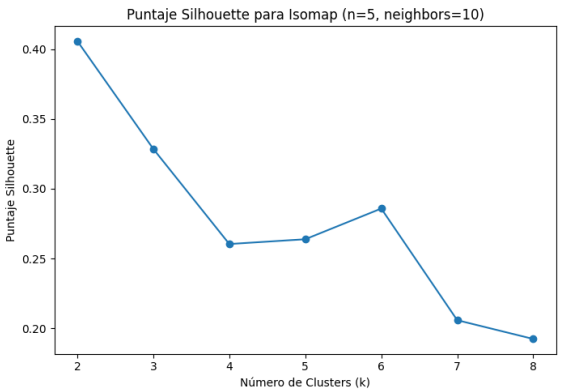

Puntaje Silhouette para Isomap (n=5, neighbors=5)
Puntaje Silhouette
Número de Clusters (k)

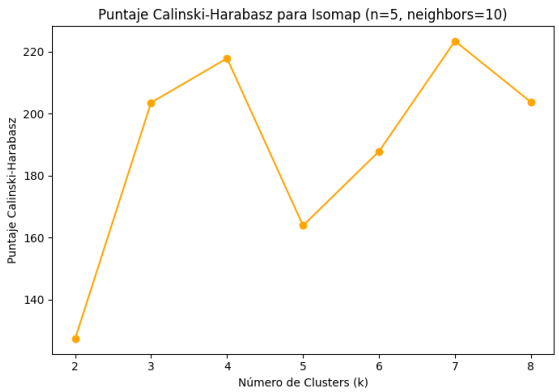

Puntaje Calinski-Harabasz para Isomap (n=5, neighbors=5)
Puntaje Calinski-Harabasz
Número de Clusters (k)

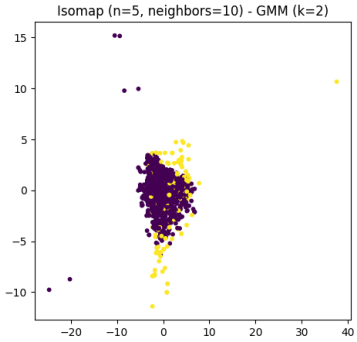

Isomap (n=5, neighbors=5) - GMM (k=2)

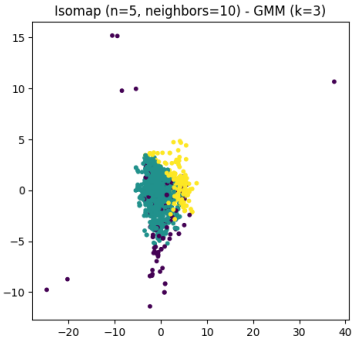

Isomap (n=5, neighbors=5) - GMM (k=3)

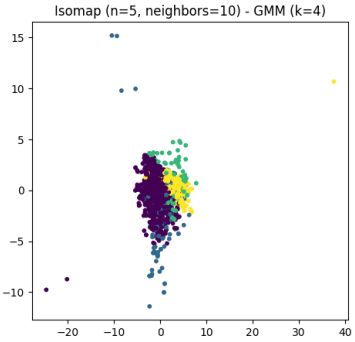

Isomap (n=5, neighbors=5) - GMM (k=4)

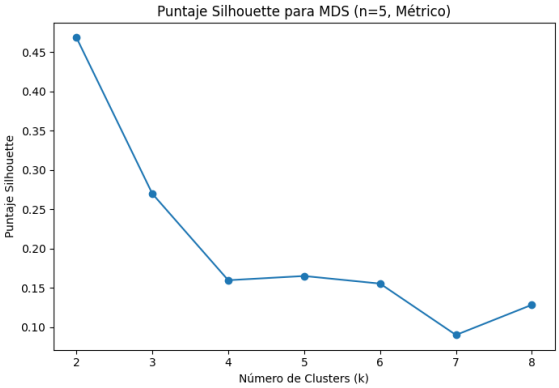

Puntaje Silhouette para Isomap (n=5, neighbors=10)
Puntaje Silhouette
Número de Clusters (k)

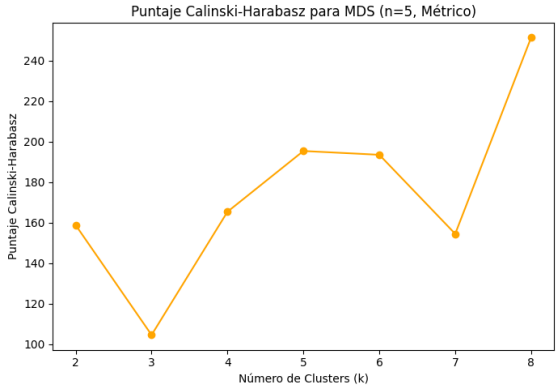

Puntaje Calinski-Harabasz para Isomap (n=5, neighbors=10)
Puntaje Calinski-Harabasz
Número de Clusters (k)

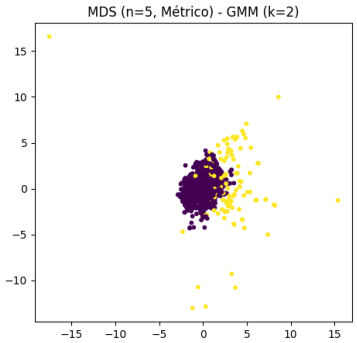

Isomap (n=5, neighbors=10) - GMM (k=2)

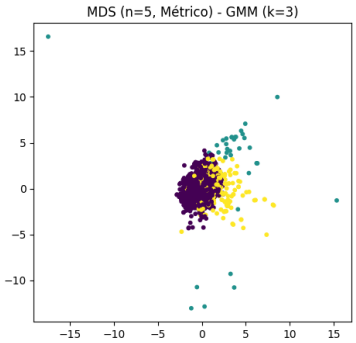

Isomap (n=5, neighbors=10) - GMM (k=3)

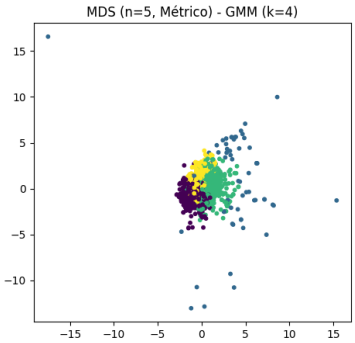

Isomap (n=5, neighbors=10) - GMM (k=4)

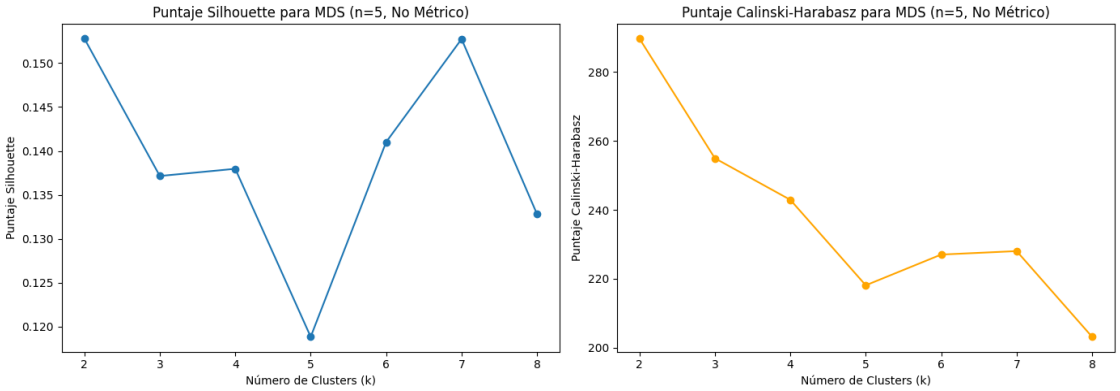

Puntaje Silhouette para MDS (n=5, Métrico)
Puntaje Silhouette
Número de Clusters (k)

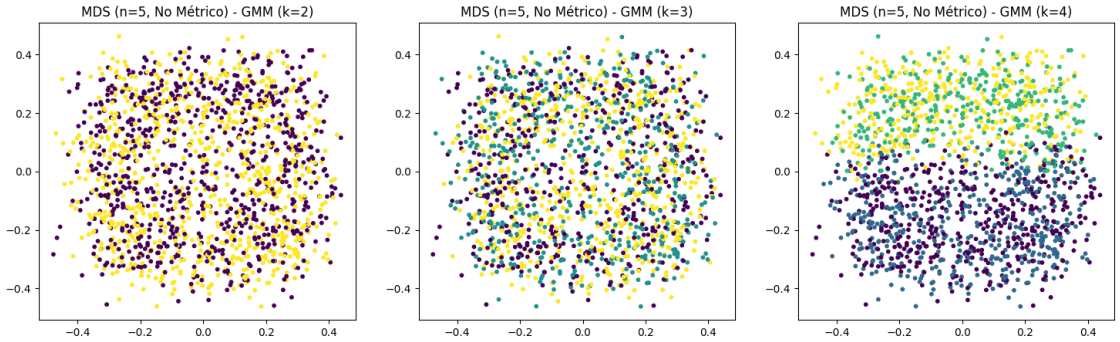

Puntaje Calinski-Harabasz para MDS (n=5, Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

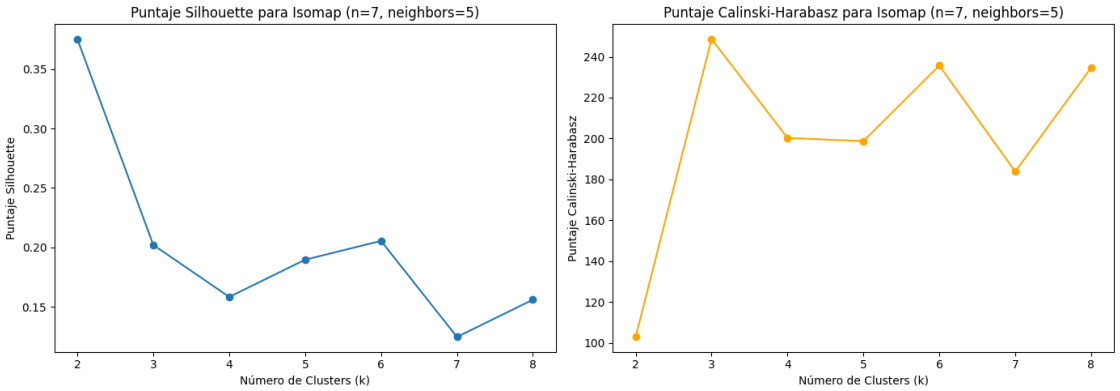

MDS (n=5, Métrico) - GMM (k=2)

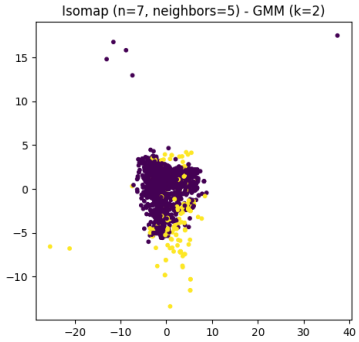

MDS (n=5, Métrico) - GMM (k=3)

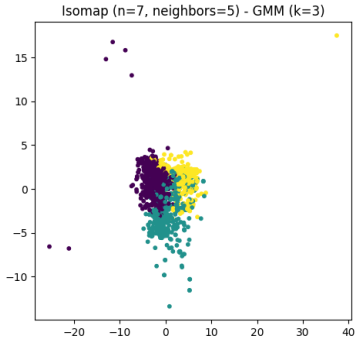

MDS (n=5, Métrico) - GMM (k=4)

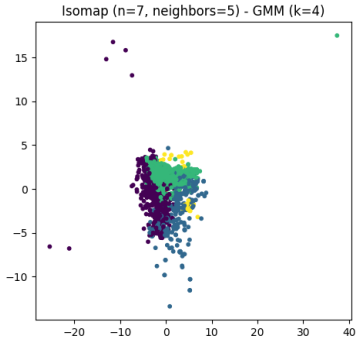

Puntaje Silhouette para MDS (n=5, No Métrico)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para MDS (n=5, No Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

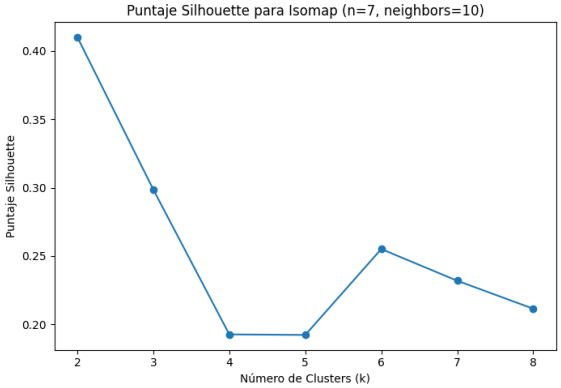

MDS (n=5, No Métrico) - GMM (k=2)
MDS (n=5, No Métrico) - GMM (k=3)
MDS (n=5, No Métrico) - GMM (k=4)

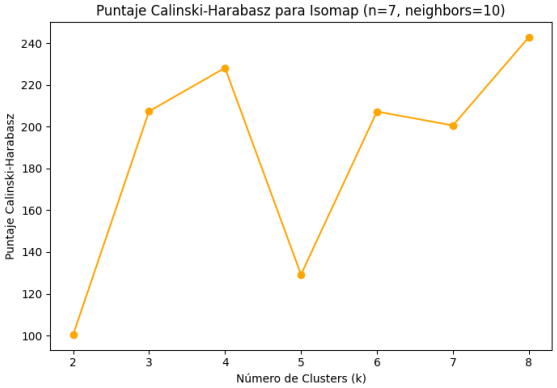

Puntaje Silhouette para Isomap (n=7, neighbors=5)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para Isomap (n=7, neighbors=5)
Puntaje Calinski-Harabasz
Número de Clusters (k)

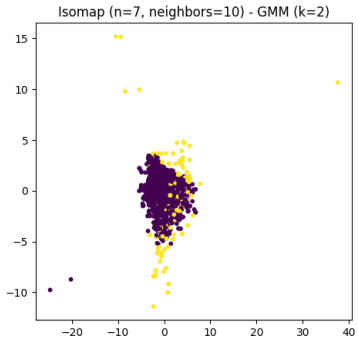

Isomap (n=7, neighbors=5) - GMM (k=2)

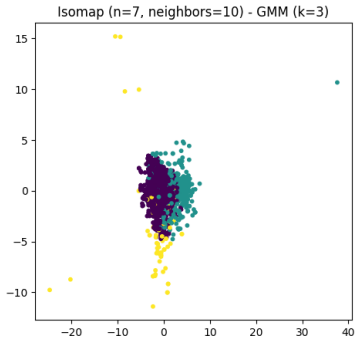

Isomap (n=7, neighbors=5) - GMM (k=3)

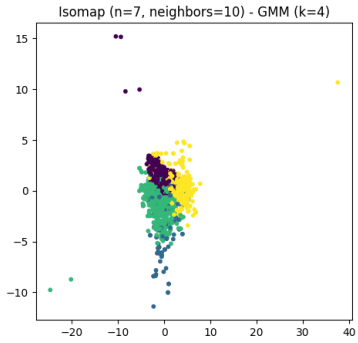

Isomap (n=7, neighbors=5) - GMM (k=4)

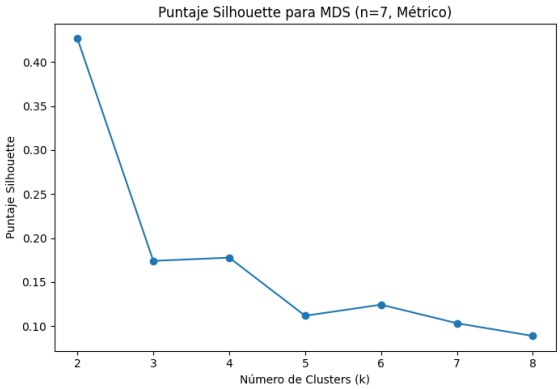

Puntaje Silhouette para Isomap (n=7, neighbors=10)
Puntaje Silhouette
Número de Clusters (k)

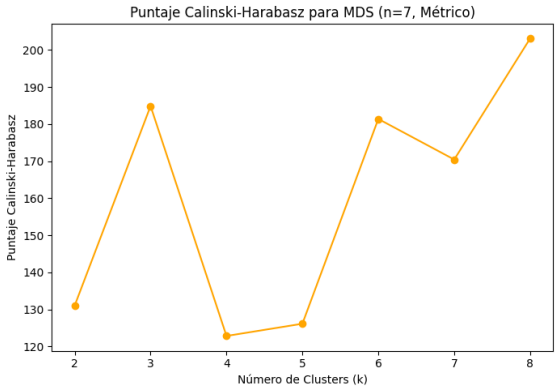

Puntaje Calinski-Harabasz para Isomap (n=7, neighbors=10)
Puntaje Calinski-Harabasz
Número de Clusters (k)

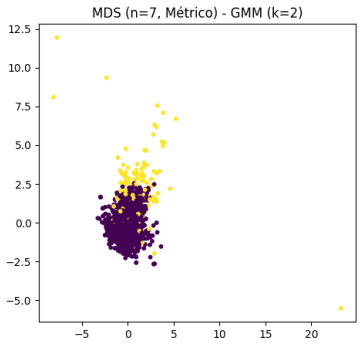

Isomap (n=7, neighbors=10) - GMM (k=2)

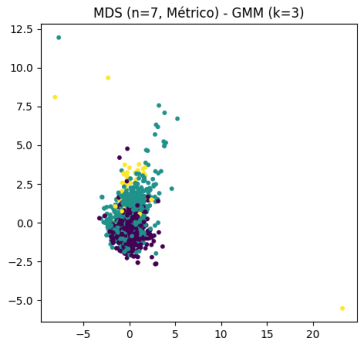

Isomap (n=7, neighbors=10) - GMM (k=3)

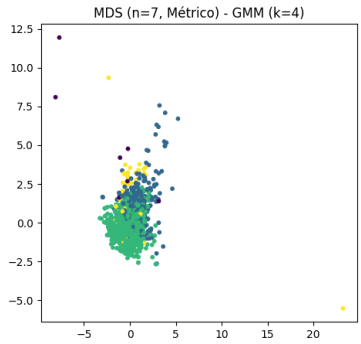

Isomap (n=7, neighbors=10) - GMM (k=4)

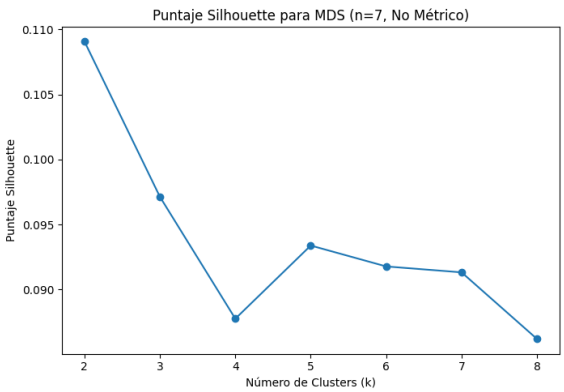

Puntaje Silhouette para MDS (n=7, Métrico)
Puntaje Silhouette
Número de Clusters (k)

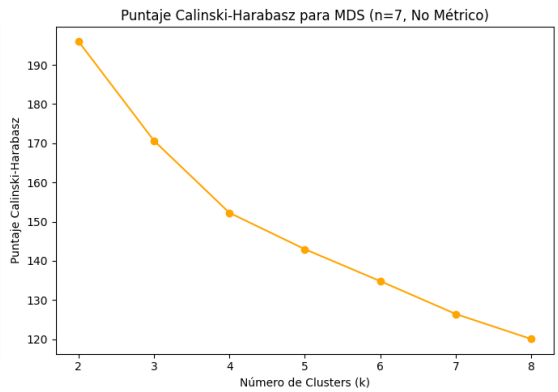

Puntaje Calinski-Harabasz para MDS (n=7, Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

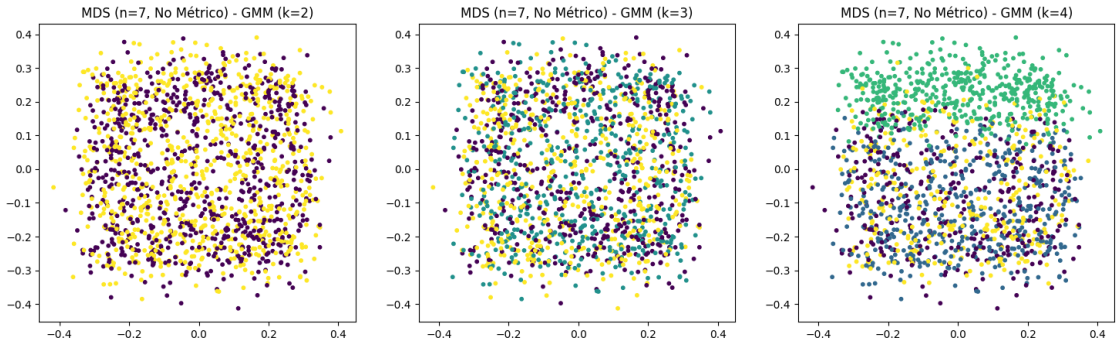

MDS (n=7, Métrico) - GMM (k=2)


MDS (n=7, Métrico) - GMM (k=3)


MDS (n=7, Métrico) - GMM (k=4)


Puntaje Silhouette para MDS (n=7, No Métrico)
Puntaje Silhouette
Número de Clusters (k)


Puntaje Calinski-Harabasz para MDS (n=7, No Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)



#Mejor Esquema de Clasificacion (GMM)

[65]:
```
# Función para encontrar el mejor esquema de clustering para GMM
def find_best_scheme_gmm(clustering_metrics_gmm, reduction_errors):
    best_scheme = None
    best_score = -np.inf

    for reduction_key, metrics in clustering_metrics_gmm.items():
        # Obtiene el error de reducción de dimensionalidad
        reduction_error = reduction_errors.get(reduction_key, 0)

        for k, (silhouette_avg, calinski_harabasz) in metrics.items():
            # Calcular un puntaje combinado para cada esquema (mayor es mejor)
            score = (silhouette_avg + calinski_harabasz) / 2 - reduction_error

            # Actualiza el mejor esquema si encuentra un puntaje más alto
            if score > best_score:
                best_score = score
                best_scheme = (reduction_key, k, silhouette_avg,␣
 ↪calinski_harabasz, reduction_error)

    return best_scheme

# Encuentra el mejor esquema de clasificación en GMM
best_scheme_gmm = find_best_scheme_gmm(clustering_metrics_gmm, reduction_errors)

# Muestra el mejor esquema encontrado
print("Mejor esquema de clasificación (GMM):")
print(f"Reducción de Dimensionalidad: {best_scheme_gmm[0]}")
print(f"Número de Clusters (k): {best_scheme_gmm[1]}")
print(f"Puntaje Silhouette Promedio: {best_scheme_gmm[2]:.2f}")
print(f"Puntaje Calinski-Harabasz Promedio: {best_scheme_gmm[3]:.2f}")
print(f"Error de Reducción: {best_scheme_gmm[4]:.2f}")
```

```
Mejor esquema de clasificación (GMM):
```

```
Reducción de Dimensionalidad: Isomap (n=5, neighbors=5)
Número de Clusters (k): 7
Puntaje Silhouette Promedio: 0.20
Puntaje Calinski-Harabasz Promedio: 328.88
Error de Reducción: 2.54
```

- Determinar cuál es el mejor esquema de clasificación (modelo de reducción + método de clusterización) de los datos entre todos los evaluados.

#Mejor Esquema de Clasificacion (Overall)

[66]:
```python
# Función para encontrar el mejor esquema entre los tres algoritmos de clustering
def find_overall_best_scheme(best_scheme_kmeans, best_scheme_hierarchical, best_scheme_gmm):
    # Inicializa variables para el mejor esquema general
    best_overall_scheme = None
    best_score = -np.inf

    # Diccionario con los resultados de los mejores esquemas para cada algoritmo
    best_schemes = {
        "KMeans": best_scheme_kmeans,
        "Hierarchical": best_scheme_hierarchical,
        "GMM": best_scheme_gmm
    }

    # Itera sobre cada esquema y calcula el puntaje combinado para cada uno
    for algorithm, scheme in best_schemes.items():
        reduction_key, k, silhouette_avg, calinski_harabasz, reduction_error = scheme
        score = (silhouette_avg + calinski_harabasz) / 2 - reduction_error

        # Verifica si este esquema es mejor que el actual mejor esquema
        if score > best_score:
            best_score = score
            best_overall_scheme = (algorithm, reduction_key, k, silhouette_avg, calinski_harabasz, reduction_error)

    return best_overall_scheme

# Encuentra el mejor esquema entre los resultados de KMeans, Hierarchical y GMM
best_overall_scheme = find_overall_best_scheme(best_scheme_kmeans, best_scheme_hierarchical, best_scheme_gmm)

# Muestra el mejor esquema general encontrado
print("Mejor esquema de clasificación entre los tres algoritmos:")
print(f"Algoritmo: {best_overall_scheme[0]}")
print(f"Reducción de Dimensionalidad: {best_overall_scheme[1]}")
```

```
print(f"Número de Clusters (k): {best_overall_scheme[2]}")
print(f"Puntaje Silhouette Promedio: {best_overall_scheme[3]:.2f}")
print(f"Puntaje Calinski-Harabasz Promedio: {best_overall_scheme[4]:.2f}")
print(f"Error de Reducción: {best_overall_scheme[5]:.2f}")
```

```
Mejor esquema de clasificación entre los tres algoritmos:
Algoritmo: KMeans
Reducción de Dimensionalidad: Isomap (n=3, neighbors=5)
Número de Clusters (k): 5
Puntaje Silhouette Promedio: 0.43
Puntaje Calinski-Harabasz Promedio: 746.43
Error de Reducción: 2.03
```

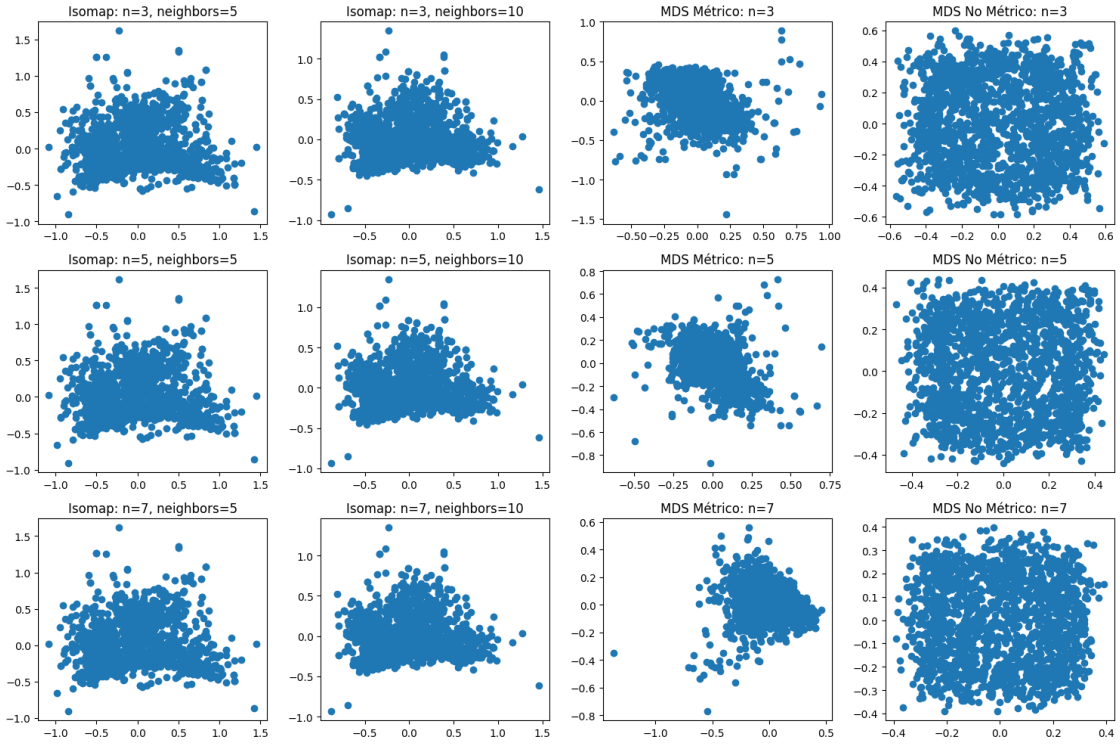
#Utilizando MinMax

[43]:
```
from scipy.spatial import distance_matrix

scaler2 = MinMaxScaler(feature_range=(0, 1))
data_normalized = scaler2.fit_transform(df)

# Parámetros
n_components = [3, 5, 7]  # Número de componentes para cada técnica
n_neighbors_isomap = [5, 10]  # Número de vecinos para Isomap

# Diccionarios para almacenar resultados de reducción y errores
results = {}
reduction_errors = {}

# Función para aplicar Isomap y MDS con ajuste de hiperparámetros
def apply_dimensionality_reduction(data, n_components, n_neighbors_isomap):

    for n in n_components:
        # Isomap con diferentes valores de n_neighbors y método del camino más␣
 ↪corto
        for neighbors in n_neighbors_isomap:
            isomap = Isomap(n_components=n, n_neighbors=neighbors,␣
 ↪path_method='auto', n_jobs=-1)
            data_isomap = isomap.fit_transform(data)
            # Calcular la distorsión de distancia para Isomap como error
            original_distances = distance_matrix(data, data)
            reduced_distances = distance_matrix(data_isomap, data_isomap)
            distortion_error = np.mean(np.abs(original_distances -␣
 ↪reduced_distances))

            results[f'Isomap (n={n}, neighbors={neighbors})'] = data_isomap
```

```
            reduction_errors[f'Isomap (n={n}, neighbors={neighbors})'] =␣
 ↪distortion_error


        # MDS métrico y no métrico
        for metric in [True, False]:
            mds = MDS(n_components=n, metric=metric, n_init=10, max_iter=500,␣
 ↪n_jobs=-1)
            data_mds = mds.fit_transform(data)
            # Guardar el estrés para MDS
            stress = mds.stress_
            metric_type = 'Métrico' if metric else 'No Métrico'
            results[f'MDS (n={n}, {metric_type})'] = data_mds
            reduction_errors[f'MDS (n={n}, {metric_type})'] = stress

# Aplicar reduccion de dimensionalidad
apply_dimensionality_reduction(data_normalized, n_components,␣
 ↪n_neighbors_isomap)
```

[44]:
```
import matplotlib.pyplot as plt

def visualize_dimensionality_reduction_results(results, n_components,␣
 ↪n_neighbors_isomap, figsize=(15, 10)):
    """
    Visualiza los resultados de reducción de dimensionalidad usando Isomap y MDS
    """
    fig, axs = plt.subplots(len(n_components), len(n_neighbors_isomap) + 2,␣
 ↪figsize=figsize)

    for i, n in enumerate(n_components):
        for j, neighbors in enumerate(n_neighbors_isomap):
            ax = axs[i, j]
            data_isomap = results[f'Isomap (n={n}, neighbors={neighbors})']
            ax.scatter(data_isomap[:, 0], data_isomap[:, 1])
            ax.set_title(f'Isomap: n={n}, neighbors={neighbors}')

        # Visualizacion del MDS Metrico
        ax = axs[i, -2]
        data_mds_metric = results[f'MDS (n={n}, Métrico)']
        ax.scatter(data_mds_metric[:, 0], data_mds_metric[:, 1])
        ax.set_title(f'MDS Métrico: n={n}')

        # Visualización de MDS No Métrico
        ax = axs[i, -1]
        data_mds_nonmetric = results[f'MDS (n={n}, No Métrico)']
        ax.scatter(data_mds_nonmetric[:, 0], data_mds_nonmetric[:, 1])
        ax.set_title(f'MDS No Métrico: n={n}')
```

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

visualize_dimensionality_reduction_results(results, n_components,␣
 ↪n_neighbors_isomap)
```



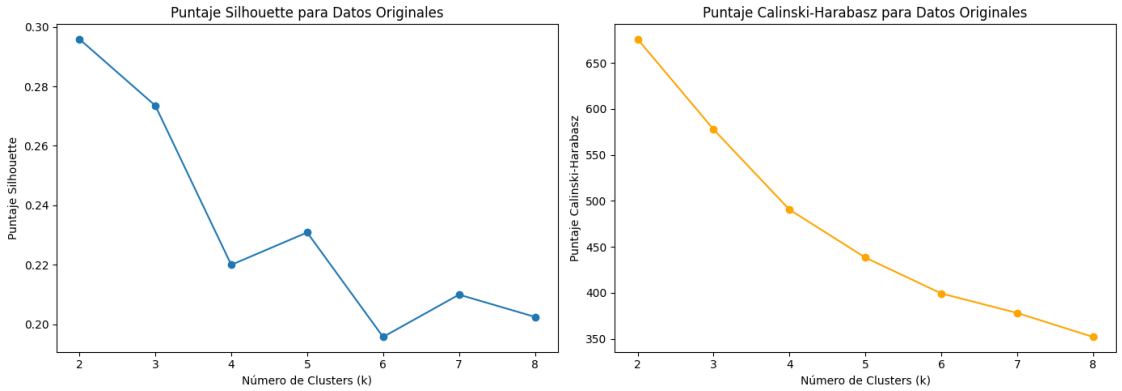
5 KMeans

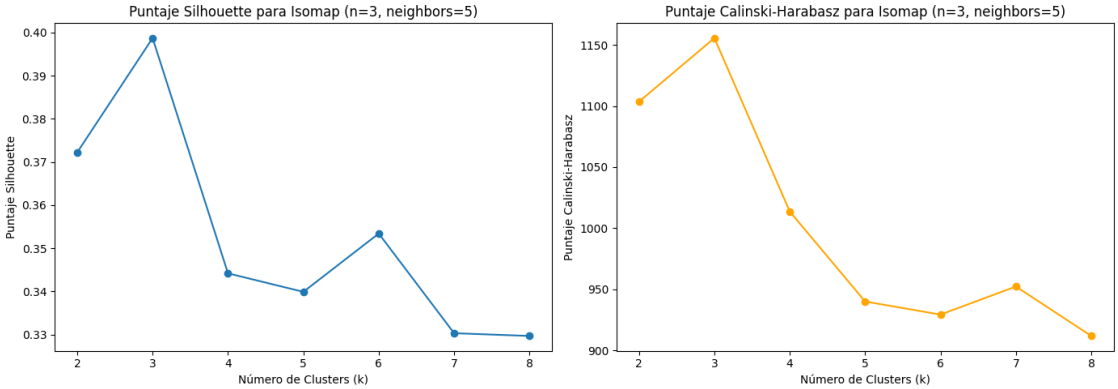
[51]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, calinski_harabasz_score
import warnings

warnings.filterwarnings("ignore")

#Diccionaro para almacenar métricas de clustering para cada configuración
clustering_metrics = {}

# Función para aplicar K-Means y calcular métricas
def apply_clustering(data, n_clusters, reduction_key, random_state=42):
```

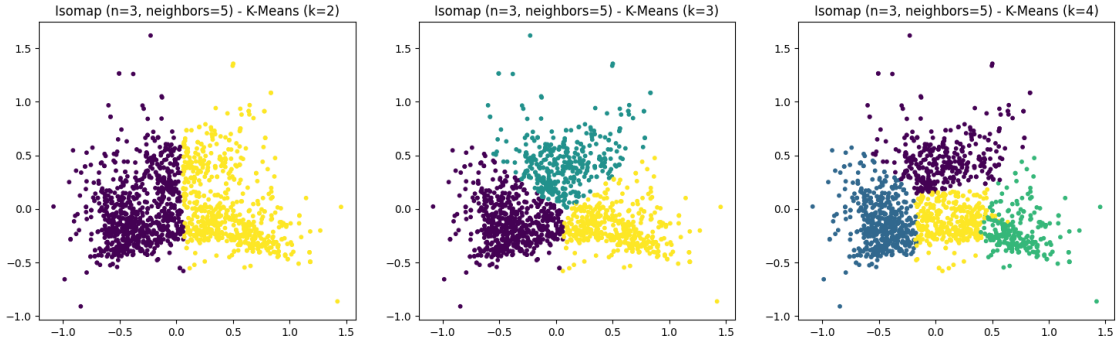
```
    clustering_results = {}
    for k in n_clusters:
        model = KMeans(n_clusters=k, init='k-means++',␣
 ↪random_state=random_state)
        labels = model.fit_predict(data)
        silhouette_avg = silhouette_score(data, labels)
        calinski_harabasz = calinski_harabasz_score(data, labels)
        clustering_results[k] = (labels, silhouette_avg, calinski_harabasz)

        # Almacenar las métricas en el diccionario global
        if reduction_key not in clustering_metrics:
            clustering_metrics[reduction_key] = {}
        clustering_metrics[reduction_key][k] = (silhouette_avg,␣
 ↪calinski_harabasz)

    return clustering_results

# Función para graficar las métricas de clusterización
def plot_clustering_metrics(clustering_results, description):
    k_values = list(clustering_results.keys())
    silhouette_scores = [clustering_results[k][1] for k in k_values]
    calinski_harabasz_scores = [clustering_results[k][2] for k in k_values]

    # Crear gráficos para Silhouette y Calinski-Harabasz
    fig, axs = plt.subplots(1, 2, figsize=(14, 5))

    # Gráfico de Silhouette Score
    axs[0].plot(k_values, silhouette_scores, marker='o')
    axs[0].set_xlabel("Número de Clusters (k)")
    axs[0].set_ylabel("Puntaje Silhouette")
    axs[0].set_title(f"Puntaje Silhouette para {description}")

    # Gráfico de Calinski-Harabasz Score
    axs[1].plot(k_values, calinski_harabasz_scores, marker='o', color='orange')
    axs[1].set_xlabel("Número de Clusters (k)")
    axs[1].set_ylabel("Puntaje Calinski-Harabasz")
    axs[1].set_title(f"Puntaje Calinski-Harabasz para {description}")

    plt.tight_layout()
    plt.show()

# Función para graficar los resultados de la clusterización en el espacio␣
 ↪reducido
def plot_clustering_results(ax, data_reduced, labels, title):
    ax.scatter(data_reduced[:, 0], data_reduced[:, 1], c=labels,␣
 ↪cmap='viridis', s=10)
    ax.set_title(title)
```

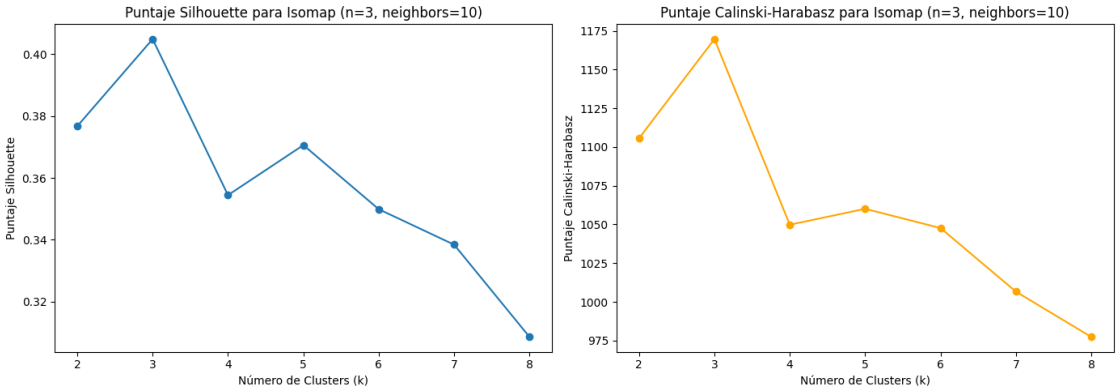
```
# Función principal para aplicar reducción de dimensionalidad, clustering y␣
 ↪visualizar resultados
def dimensionality_reduction_clustering_with_visuals(results, n_components,␣
 ↪n_neighbors_isomap, n_clusters, data_normalized):
    # Aplicar clusterización a los datos originales
    original_clustering_results = apply_clustering(data_normalized, n_clusters,␣
 ↪"Datos Originales")
    plot_clustering_metrics(original_clustering_results, "Datos Originales")

    # Graficar los resultados para cada configuración de reducción de␣
 ↪dimensionalidad
    for n in n_components:
        for neighbors in n_neighbors_isomap:
            key = f'Isomap (n={n}, neighbors={neighbors})'
            data_reduced = results[key]
            reduced_clustering_results = apply_clustering(data_reduced,␣
 ↪n_clusters, key)

            # Gráficos de métricas para cada configuración
            plot_clustering_metrics(reduced_clustering_results, key)

            # Visualización de clustering en el espacio reducido para 3 valores␣
 ↪de k (por ejemplo, k=2, k=3, y k=4)
            fig, axs = plt.subplots(1, 3, figsize=(18, 5))
            k_values_for_visuals = [2, 3, 4]
            for i, k in enumerate(k_values_for_visuals):
                labels = reduced_clustering_results[k][0]
                plot_clustering_results(axs[i], data_reduced, labels, f'{key} -␣
 ↪K-Means (k={k})')
            plt.show()

        # MDS Métrico
        key = f'MDS (n={n}, Métrico)'
        data_reduced = results[key]
        reduced_clustering_results = apply_clustering(data_reduced, n_clusters,␣
 ↪key)

        # Gráficos de métricas para MDS Métrico
        plot_clustering_metrics(reduced_clustering_results, key)

        # Visualización de clustering en el espacio reducido para 3 valores de␣
 ↪k (k=2, k=3, y k=4)
        fig, axs = plt.subplots(1, 3, figsize=(18, 5))
        for i, k in enumerate(k_values_for_visuals):
            labels = reduced_clustering_results[k][0]
```

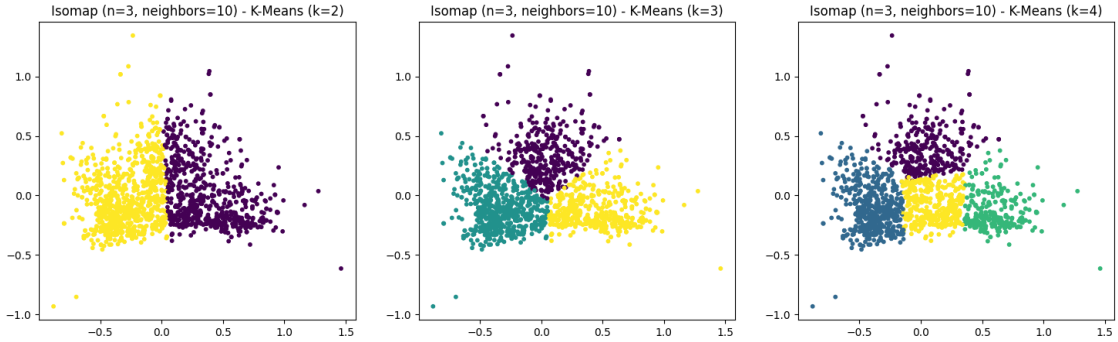
```
            plot_clustering_results(axs[i], data_reduced, labels, f'{key} -␣
 ↪K-Means (k={k})')
        plt.show()

        # MDS No Métrico
        key = f'MDS (n={n}, No Métrico)'
        data_reduced = results[key]
        reduced_clustering_results = apply_clustering(data_reduced, n_clusters,␣
 ↪key)

        # Gráficos de métricas para MDS No Métrico
        plot_clustering_metrics(reduced_clustering_results, key)

        # Visualización de clustering en el espacio reducido para 3 valores de␣
 ↪k (k=2, k=3, y k=4)
        fig, axs = plt.subplots(1, 3, figsize=(18, 5))
        for i, k in enumerate(k_values_for_visuals):
            labels = reduced_clustering_results[k][0]
            plot_clustering_results(axs[i], data_reduced, labels, f'{key} -␣
 ↪K-Means (k={k})')
        plt.show()

# Parámetros de número de clusters
n_clusters = [2, 3, 4, 5, 6, 7, 8]  # Número de clusters para K-Means (puedes␣
 ↪ajustar estos valores)

# Ejecuta la función con los resultados de reducción de dimensionalidad y␣
 ↪clustering
dimensionality_reduction_clustering_with_visuals(results, n_components,␣
 ↪n_neighbors_isomap, n_clusters, data_normalized)
```

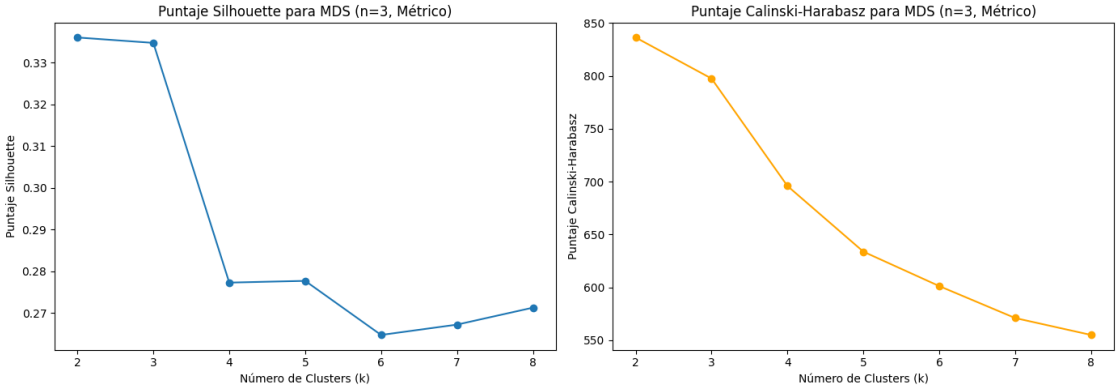


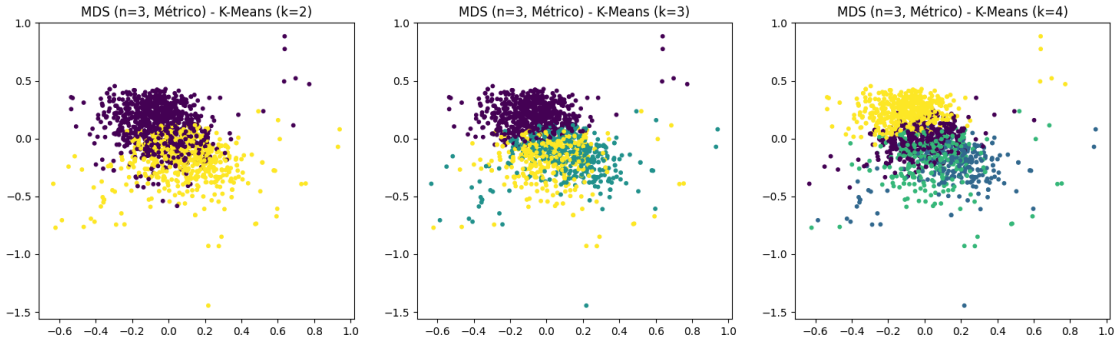

Puntaje Silhouette para Isomap (n=3, neighbors=5)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para Isomap (n=3, neighbors=5)
Puntaje Calinski-Harabasz
Número de Clusters (k)

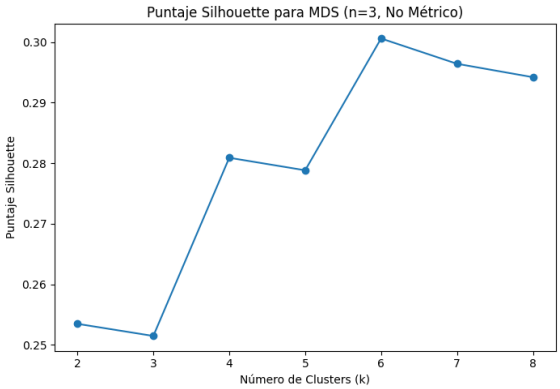

Isomap (n=3, neighbors=5) - K-Means (k=2)
Isomap (n=3, neighbors=5) - K-Means (k=3)
Isomap (n=3, neighbors=5) - K-Means (k=4)

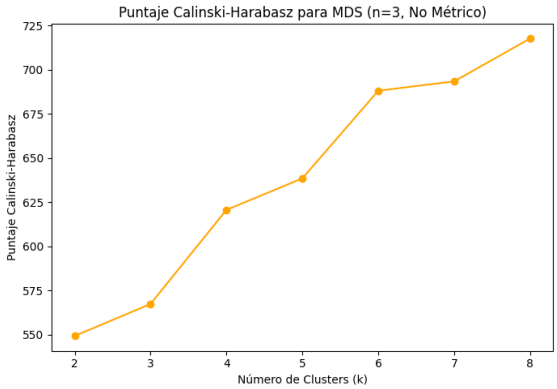

Puntaje Silhouette para Isomap (n=3, neighbors=10)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para Isomap (n=3, neighbors=10)
Puntaje Calinski-Harabasz
Número de Clusters (k)

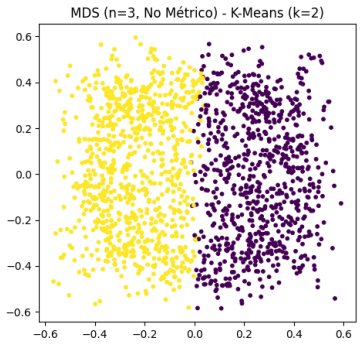

Isomap (n=3, neighbors=10) - K-Means (k=2)
Isomap (n=3, neighbors=10) - K-Means (k=3)
Isomap (n=3, neighbors=10) - K-Means (k=4)

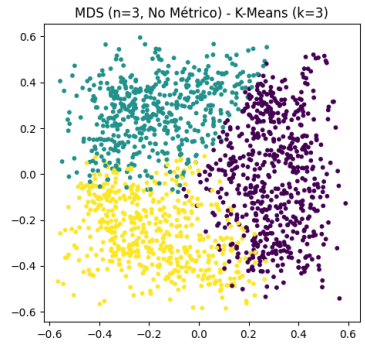

Puntaje Silhouette para MDS (n=3, Métrico)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para MDS (n=3, Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

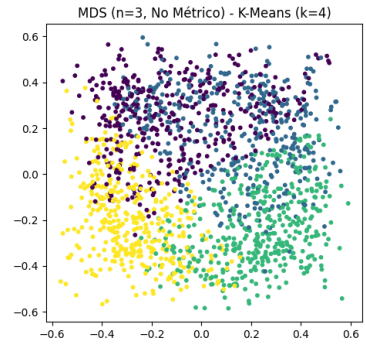

MDS (n=3, Métrico) - K-Means (k=2)
MDS (n=3, Métrico) - K-Means (k=3)
MDS (n=3, Métrico) - K-Means (k=4)

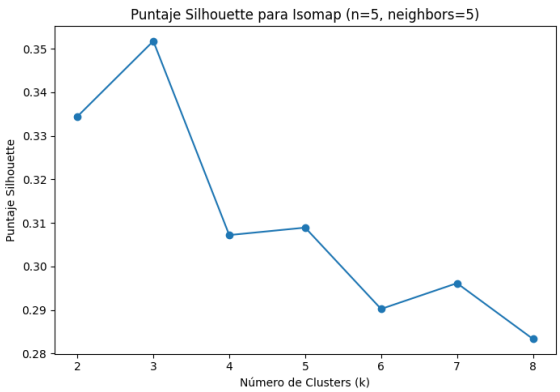

Puntaje Silhouette para MDS (n=3, No Métrico)
Puntaje Silhouette
Número de Clusters (k)

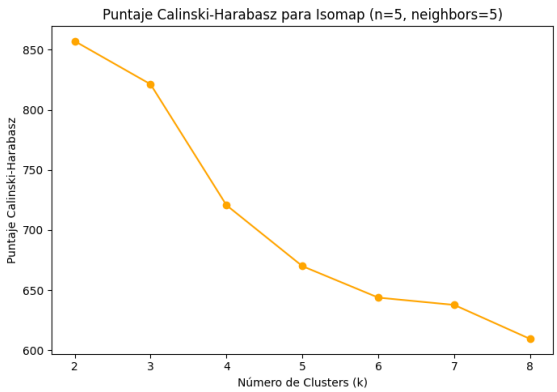

Puntaje Calinski-Harabasz para MDS (n=3, No Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

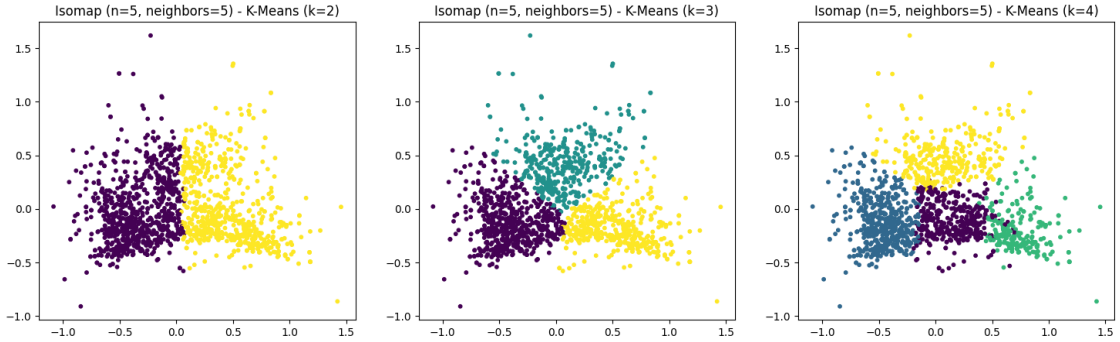

MDS (n=3, No Métrico) - K-Means (k=2)

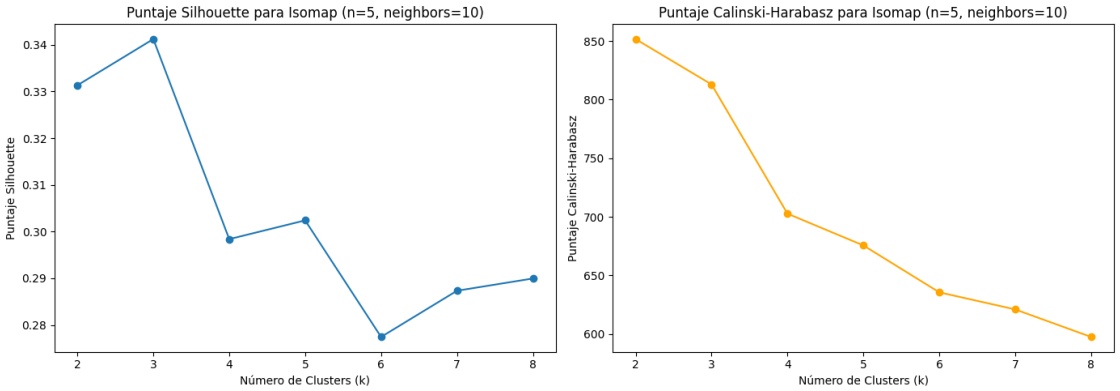

MDS (n=3, No Métrico) - K-Means (k=3)

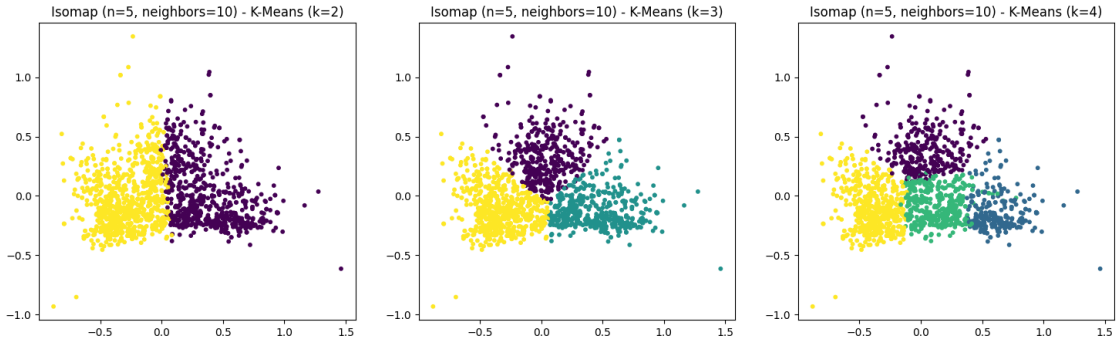

MDS (n=3, No Métrico) - K-Means (k=4)

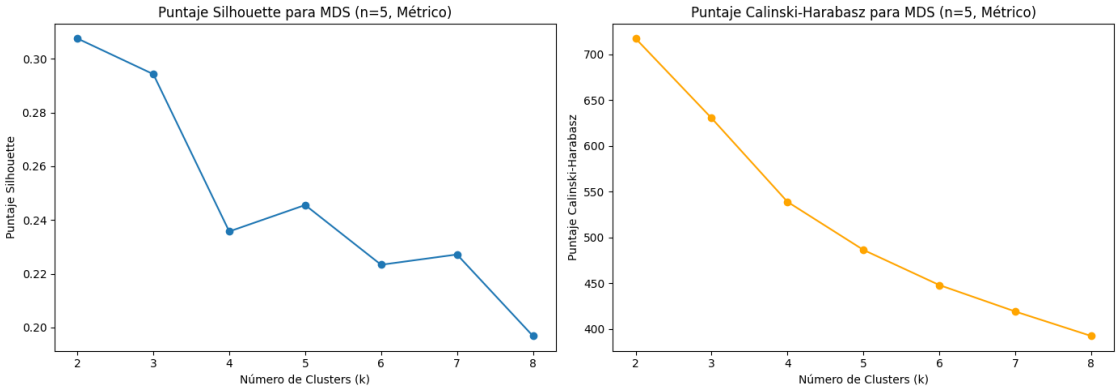

Puntaje Silhouette para Isomap (n=5, neighbors=5)
Puntaje Silhouette
Número de Clusters (k)

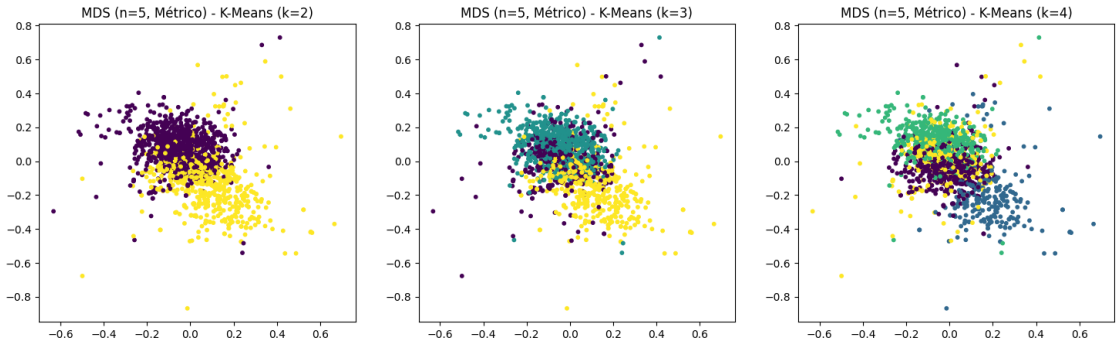

Puntaje Calinski-Harabasz para Isomap (n=5, neighbors=5)
Puntaje Calinski-Harabasz
Número de Clusters (k)

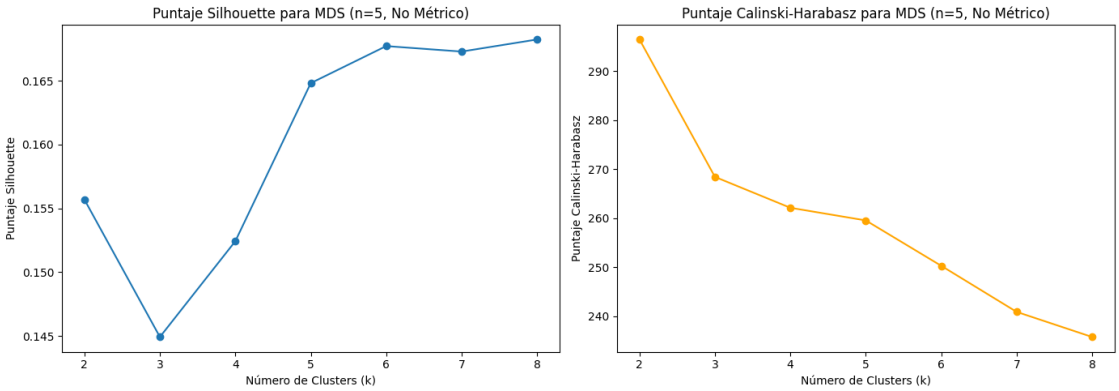

Isomap (n=5, neighbors=5) - K-Means (k=2)
Isomap (n=5, neighbors=5) - K-Means (k=3)
Isomap (n=5, neighbors=5) - K-Means (k=4)

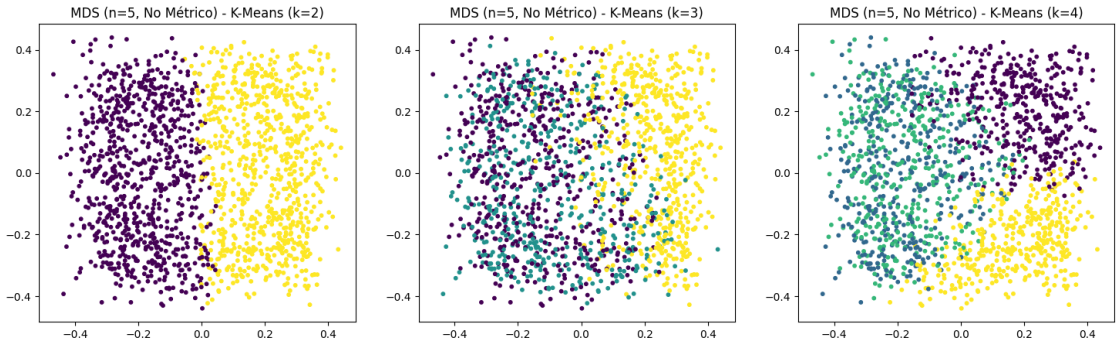

Puntaje Silhouette para Isomap (n=5, neighbors=10)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para Isomap (n=5, neighbors=10)
Puntaje Calinski-Harabasz
Número de Clusters (k)

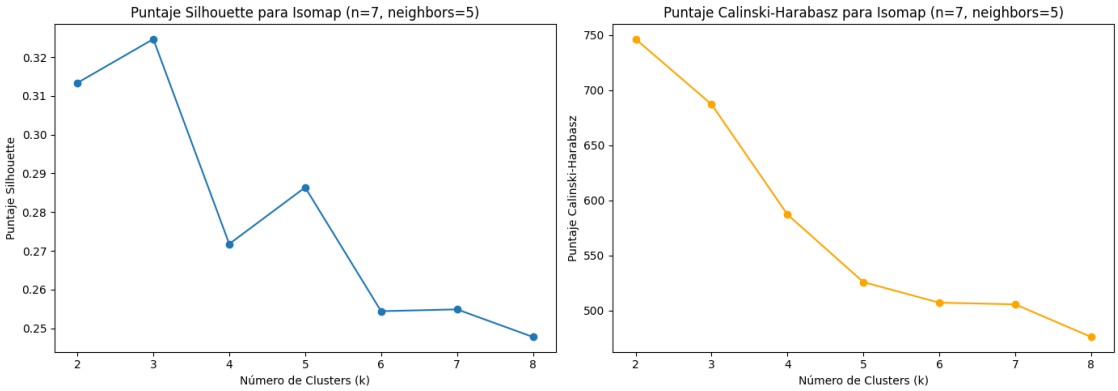

Isomap (n=5, neighbors=10) - K-Means (k=2)
Isomap (n=5, neighbors=10) - K-Means (k=3)
Isomap (n=5, neighbors=10) - K-Means (k=4)

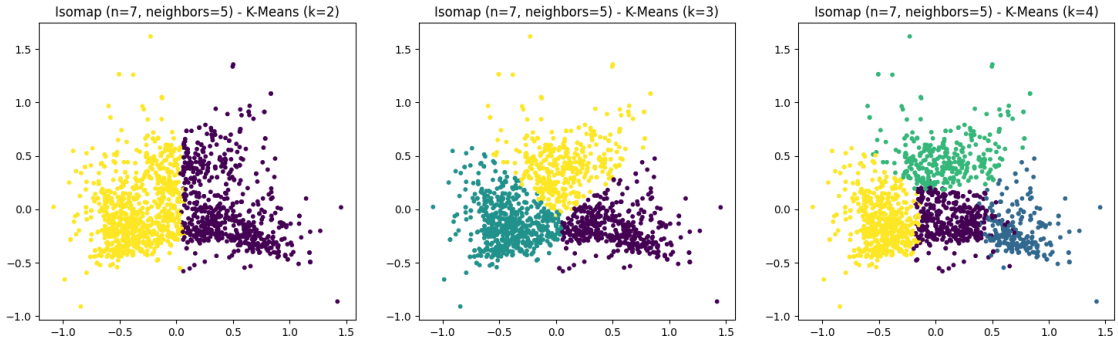

Puntaje Silhouette para MDS (n=5, Métrico)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para MDS (n=5, Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

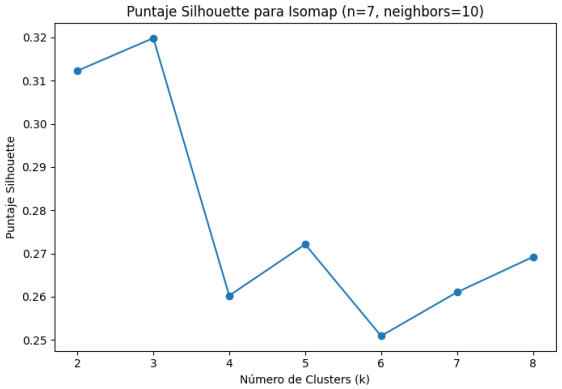

MDS (n=5, Métrico) - K-Means (k=2)
MDS (n=5, Métrico) - K-Means (k=3)
MDS (n=5, Métrico) - K-Means (k=4)

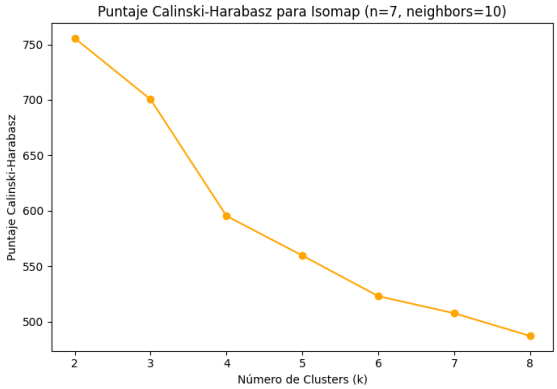

Puntaje Silhouette para MDS (n=5, No Métrico)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para MDS (n=5, No Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

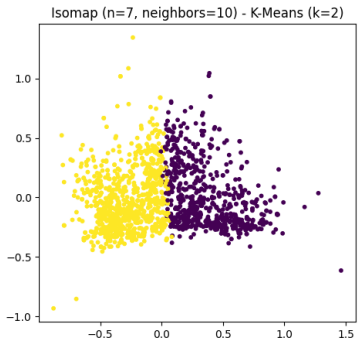

MDS (n=5, No Métrico) - K-Means (k=2)
MDS (n=5, No Métrico) - K-Means (k=3)
MDS (n=5, No Métrico) - K-Means (k=4)

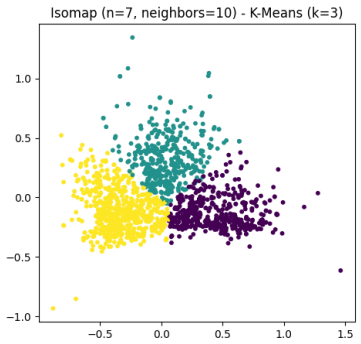

Puntaje Silhouette para Isomap (n=7, neighbors=5)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para Isomap (n=7, neighbors=5)
Puntaje Calinski-Harabasz
Número de Clusters (k)

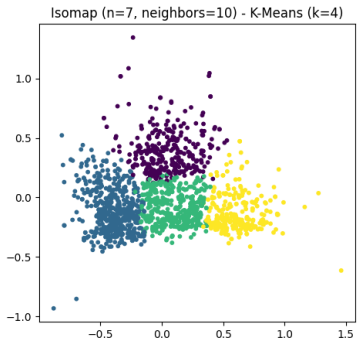

Isomap (n=7, neighbors=5) - K-Means (k=2)
Isomap (n=7, neighbors=5) - K-Means (k=3)
Isomap (n=7, neighbors=5) - K-Means (k=4)

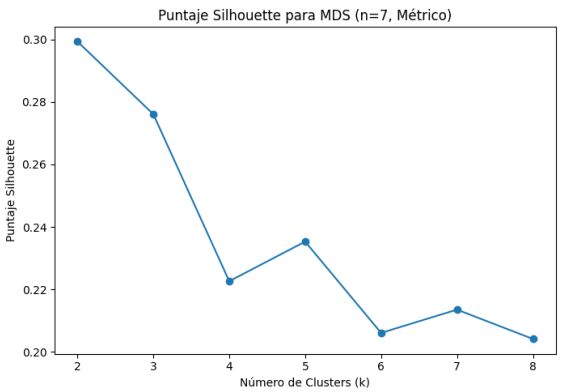

Puntaje Silhouette para Isomap (n=7, neighbors=10)
Puntaje Silhouette
Número de Clusters (k)

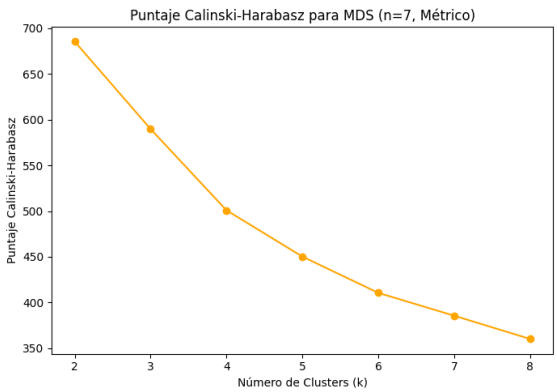

Puntaje Calinski-Harabasz para Isomap (n=7, neighbors=10)
Puntaje Calinski-Harabasz
Número de Clusters (k)

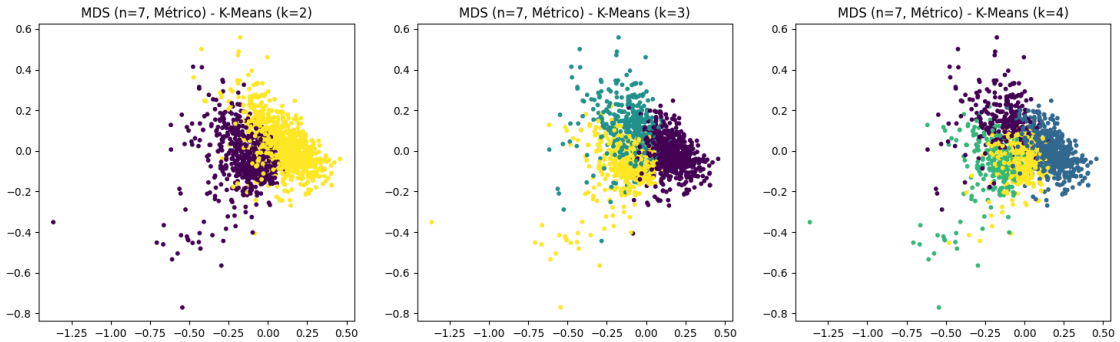

Isomap (n=7, neighbors=10) - K-Means (k=2)

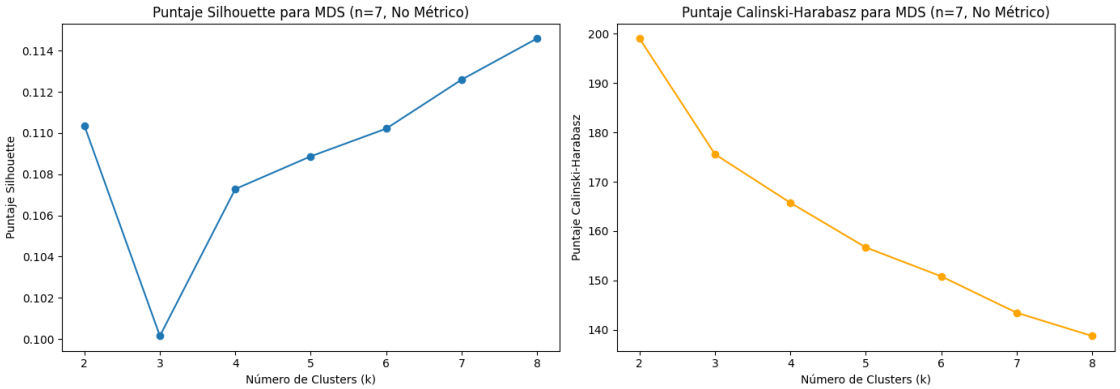

Isomap (n=7, neighbors=10) - K-Means (k=3)

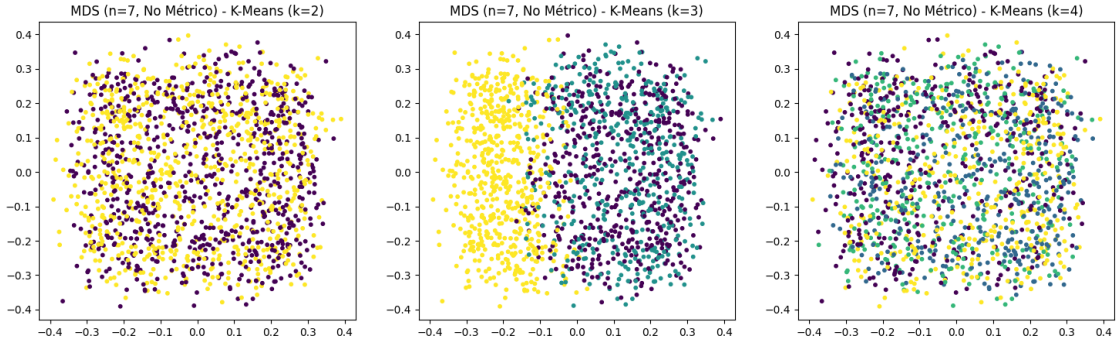

Isomap (n=7, neighbors=10) - K-Means (k=4)


Puntaje Silhouette para MDS (n=7, Métrico)
Puntaje Silhouette
Número de Clusters (k)


Puntaje Calinski-Harabasz para MDS (n=7, Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)







#Mejor Esquema de Clasificacion (KMeans)

[52]:
```
# Función para encontrar el mejor esquema de clustering
def find_best_scheme_kmeans(clustering_metrics, reduction_errors):
    best_scheme = None
    best_score = -np.inf

    for reduction_key, metrics in clustering_metrics.items():
```

```
        # Obtiene el error de reducción de dimensionalidad
        reduction_error = reduction_errors.get(reduction_key, 0)

        for k, (silhouette_avg, calinski_harabasz) in metrics.items():
            # Calcular un puntaje combinado para cada esquema (mayor es mejor)
            score = (silhouette_avg + calinski_harabasz) / 2 - reduction_error

            # Actualiza el mejor esquema si encuentra un puntaje más alto
            if score > best_score:
                best_score = score
                best_scheme = (reduction_key, k, silhouette_avg,␣
 ↪calinski_harabasz, reduction_error)

    return best_scheme

# Encuentra el mejor esquema de clasificación
best_scheme_kmeans = find_best_scheme_kmeans(clustering_metrics,␣
 ↪reduction_errors)

# Muestra el mejor esquema encontrado
print("Mejor esquema de clasificación (KMeans):")
print(f"Reducción de Dimensionalidad: {best_scheme_kmeans[0]}")
print(f"Número de Clusters (k): {best_scheme_kmeans[1]}")
print(f"Puntaje Silhouette Promedio: {best_scheme_kmeans[2]:.2f}")
print(f"Puntaje Calinski-Harabasz Promedio: {best_scheme_kmeans[3]:.2f}")
print(f"Error de Reducción: {best_scheme_kmeans[4]:.2f}")
```

```
Mejor esquema de clasificación (KMeans):
Reducción de Dimensionalidad: Isomap (n=3, neighbors=10)
Número de Clusters (k): 3
Puntaje Silhouette Promedio: 0.40
Puntaje Calinski-Harabasz Promedio: 1169.67
Error de Reducción: 0.17
```

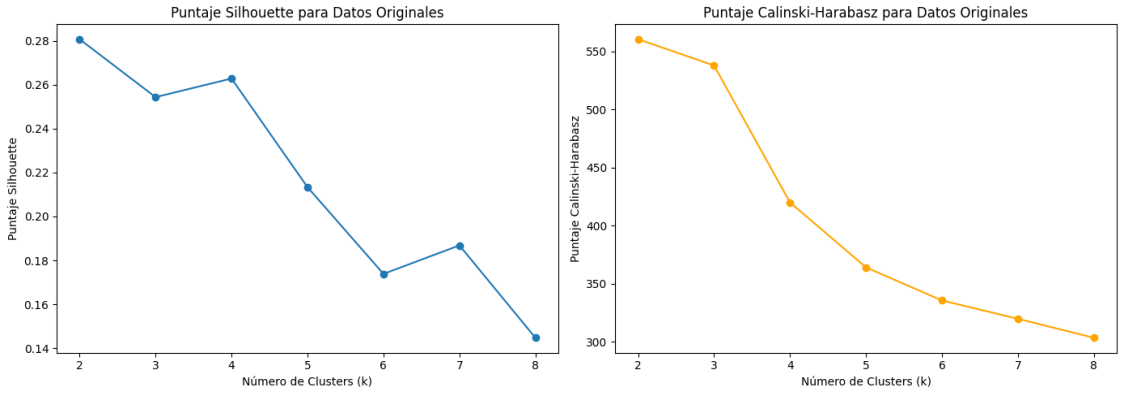
#Hierarchical Clustering

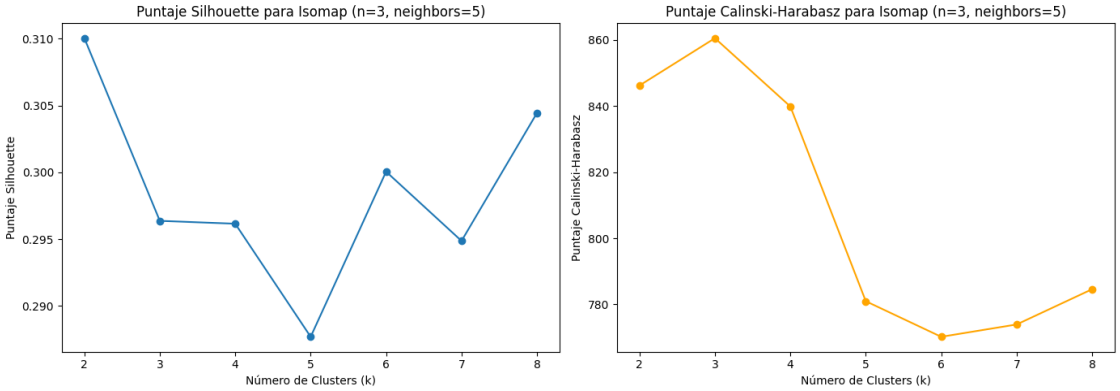
[53]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score, calinski_harabasz_score
import warnings

warnings.filterwarnings("ignore")

# Diccionario para almacenar métricas de clustering para cada configuración
clustering_metrics_hierarchical = {}

# Función para aplicar Hierarchical Clustering y calcular métricas
```

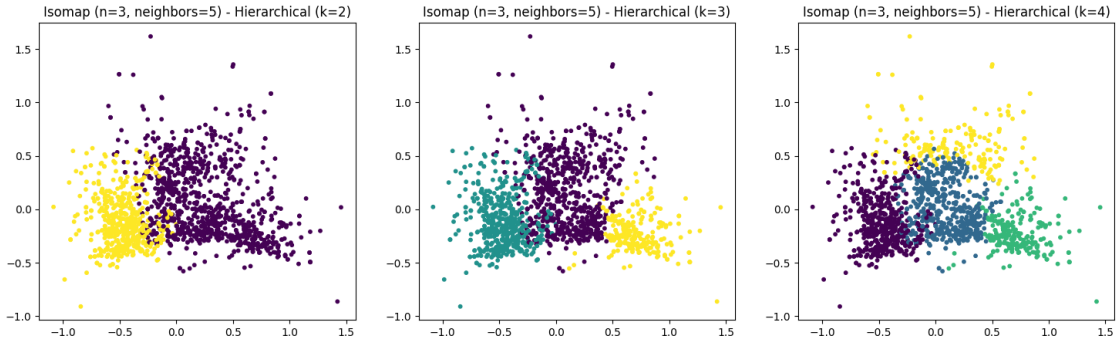
```
def apply_hierarchical_clustering(data, n_clusters, reduction_key):
    clustering_results = {}
    for k in n_clusters:
        model = AgglomerativeClustering(n_clusters=k)
        labels = model.fit_predict(data)
        silhouette_avg = silhouette_score(data, labels)
        calinski_harabasz = calinski_harabasz_score(data, labels)
        clustering_results[k] = (labels, silhouette_avg, calinski_harabasz)

        # Almacenar las métricas en el diccionario global
        if reduction_key not in clustering_metrics_hierarchical:
            clustering_metrics_hierarchical[reduction_key] = {}
        clustering_metrics_hierarchical[reduction_key][k] = (silhouette_avg,␣
 ↪calinski_harabasz)

    return clustering_results

# Función para graficar las métricas de clusterización
def plot_clustering_metrics(clustering_results, description):
    k_values = list(clustering_results.keys())
    silhouette_scores = [clustering_results[k][1] for k in k_values]
    calinski_harabasz_scores = [clustering_results[k][2] for k in k_values]

    # Crear gráficos para Silhouette y Calinski-Harabasz
    fig, axs = plt.subplots(1, 2, figsize=(14, 5))

    # Gráfico de Silhouette Score
    axs[0].plot(k_values, silhouette_scores, marker='o')
    axs[0].set_xlabel("Número de Clusters (k)")
    axs[0].set_ylabel("Puntaje Silhouette")
    axs[0].set_title(f"Puntaje Silhouette para {description}")

    # Gráfico de Calinski-Harabasz Score
    axs[1].plot(k_values, calinski_harabasz_scores, marker='o', color='orange')
    axs[1].set_xlabel("Número de Clusters (k)")
    axs[1].set_ylabel("Puntaje Calinski-Harabasz")
    axs[1].set_title(f"Puntaje Calinski-Harabasz para {description}")

    plt.tight_layout()
    plt.show()

# Función para graficar los resultados de la clusterización en el espacio␣
 ↪reducido
def plot_clustering_results(ax, data_reduced, labels, title):
    ax.scatter(data_reduced[:, 0], data_reduced[:, 1], c=labels,␣
 ↪cmap='viridis', s=10)
    ax.set_title(title)
```

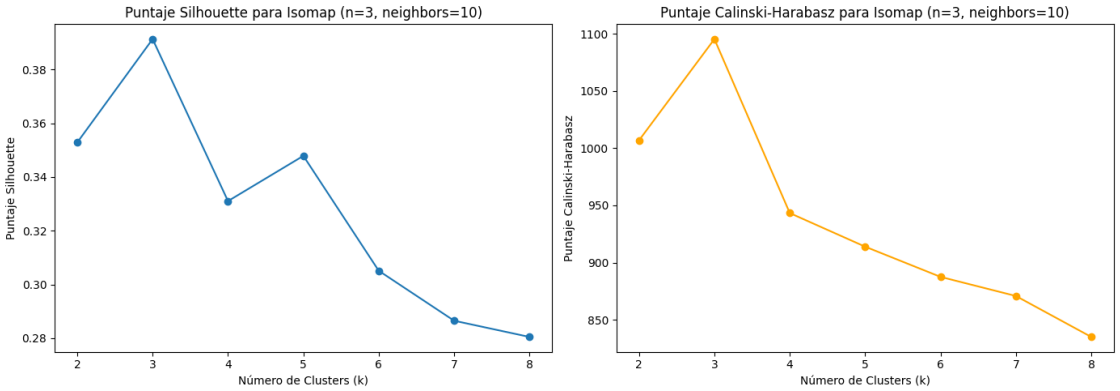
```
# Función principal para aplicar reducción de dimensionalidad, clustering y visualizar resultados
def dimensionality_reduction_clustering_with_visuals(results, n_components, n_neighbors_isomap, n_clusters, data_normalized):
    # Aplicar clusterización a los datos originales
    original_clustering_results = apply_hierarchical_clustering(data_normalized, n_clusters, "Datos Originales")
    plot_clustering_metrics(original_clustering_results, "Datos Originales")

    # Graficar los resultados para cada configuración de reducción de dimensionalidad
    for n in n_components:
        for neighbors in n_neighbors_isomap:
            key = f'Isomap (n={n}, neighbors={neighbors})'
            data_reduced = results[key]
            reduced_clustering_results = apply_hierarchical_clustering(data_reduced, n_clusters, key)

            # Gráficos de métricas para cada configuración
            plot_clustering_metrics(reduced_clustering_results, key)

            # Visualización de clustering en el espacio reducido para 3 valores de k (por ejemplo, k=2, k=3, y k=4)
            fig, axs = plt.subplots(1, 3, figsize=(18, 5))
            k_values_for_visuals = [2, 3, 4]
            for i, k in enumerate(k_values_for_visuals):
                labels = reduced_clustering_results[k][0]
                plot_clustering_results(axs[i], data_reduced, labels, f'{key} - Hierarchical (k={k})')
            plt.show()

        # MDS Métrico
        key = f'MDS (n={n}, Métrico)'
        data_reduced = results[key]
        reduced_clustering_results = apply_hierarchical_clustering(data_reduced, n_clusters, key)

        # Gráficos de métricas para MDS Métrico
        plot_clustering_metrics(reduced_clustering_results, key)

        # Visualización de clustering en el espacio reducido para 3 valores de k (k=2, k=3, y k=4)
        fig, axs = plt.subplots(1, 3, figsize=(18, 5))
        for i, k in enumerate(k_values_for_visuals):
```

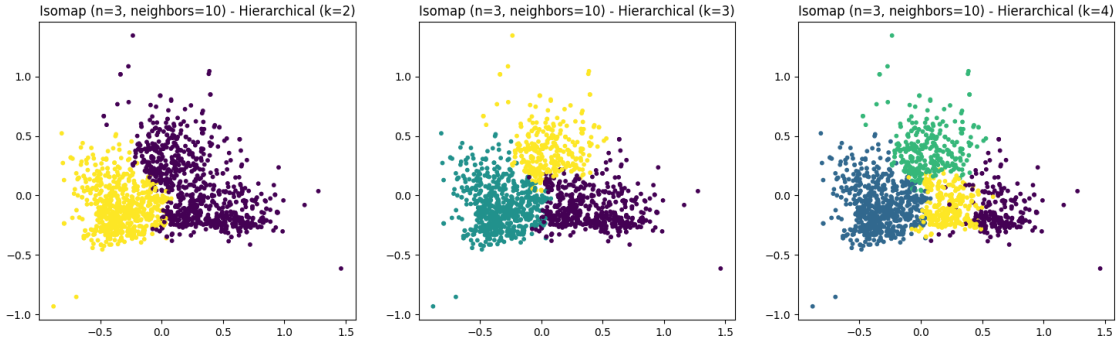
```
            labels = reduced_clustering_results[k][0]
            plot_clustering_results(axs[i], data_reduced, labels, f'{key} -␣
 ↪Hierarchical (k={k})')
        plt.show()

        # MDS No Métrico
        key = f'MDS (n={n}, No Métrico)'
        data_reduced = results[key]
        reduced_clustering_results =␣
 ↪apply_hierarchical_clustering(data_reduced, n_clusters, key)

        # Gráficos de métricas para MDS No Métrico
        plot_clustering_metrics(reduced_clustering_results, key)

        # Visualización de clustering en el espacio reducido para 3 valores de␣
 ↪k (k=2, k=3, y k=4)
        fig, axs = plt.subplots(1, 3, figsize=(18, 5))
        for i, k in enumerate(k_values_for_visuals):
            labels = reduced_clustering_results[k][0]
            plot_clustering_results(axs[i], data_reduced, labels, f'{key} -␣
 ↪Hierarchical (k={k})')
        plt.show()

# Parámetros de número de clusters
n_clusters = [2, 3, 4, 5, 6, 7, 8]  # Número de clusters para Agglomerative␣
 ↪Clustering

# Ejecuta la función con los resultados de reducción de dimensionalidad y␣
 ↪clustering
dimensionality_reduction_clustering_with_visuals(results, n_components,␣
 ↪n_neighbors_isomap, n_clusters, data_normalized)
```

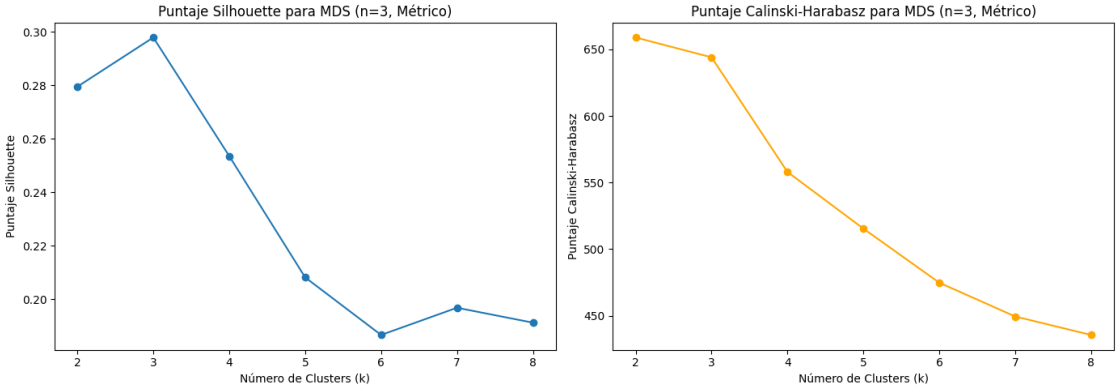


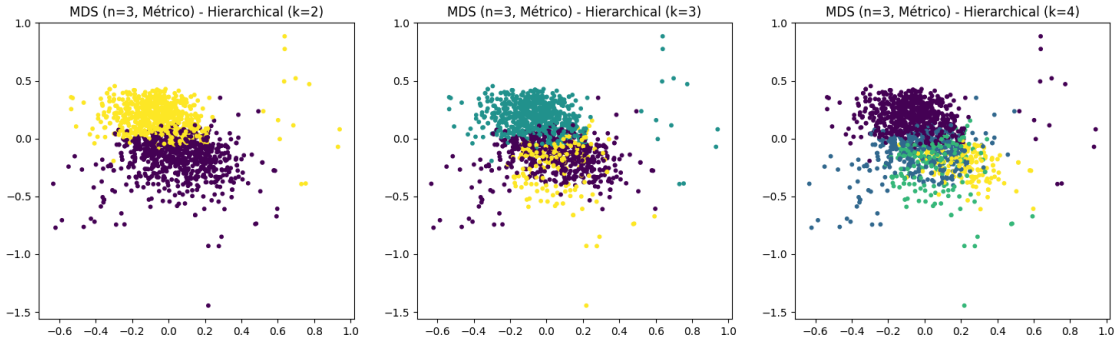

Puntaje Silhouette para Isomap (n=3, neighbors=5)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para Isomap (n=3, neighbors=5)
Puntaje Calinski-Harabasz
Número de Clusters (k)

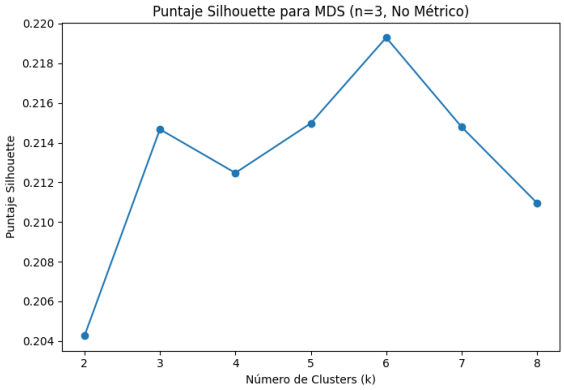

Isomap (n=3, neighbors=5) - Hierarchical (k=2)
Isomap (n=3, neighbors=5) - Hierarchical (k=3)
Isomap (n=3, neighbors=5) - Hierarchical (k=4)

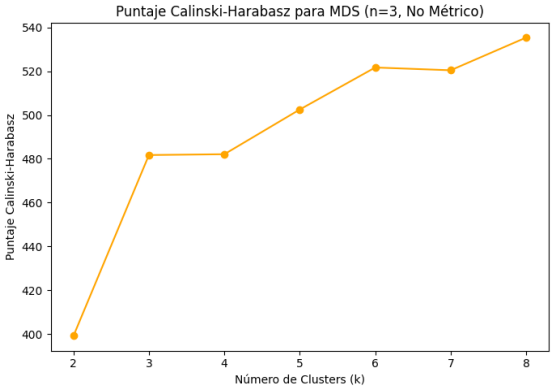

Puntaje Silhouette para Isomap (n=3, neighbors=10)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para Isomap (n=3, neighbors=10)
Puntaje Calinski-Harabasz
Número de Clusters (k)

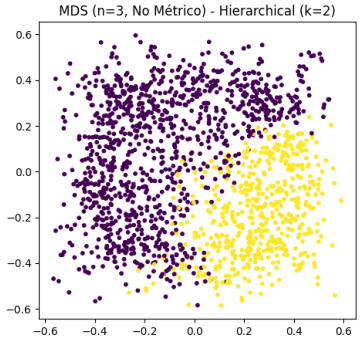

Isomap (n=3, neighbors=10) - Hierarchical (k=2)
Isomap (n=3, neighbors=10) - Hierarchical (k=3)
Isomap (n=3, neighbors=10) - Hierarchical (k=4)

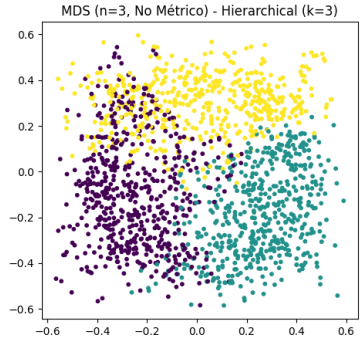

Puntaje Silhouette para MDS (n=3, Métrico)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para MDS (n=3, Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

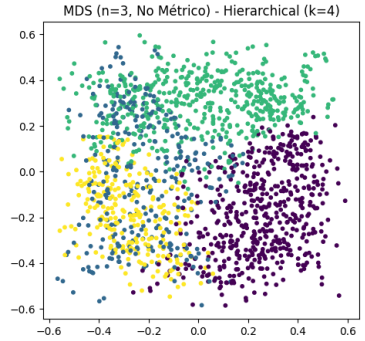

MDS (n=3, Métrico) - Hierarchical (k=2)
MDS (n=3, Métrico) - Hierarchical (k=3)
MDS (n=3, Métrico) - Hierarchical (k=4)


Puntaje Silhouette para MDS (n=3, No Métrico)
Puntaje Silhouette
Número de Clusters (k)

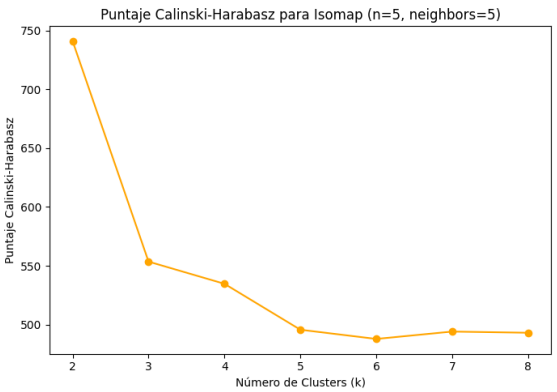

Puntaje Calinski-Harabasz para MDS (n=3, No Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

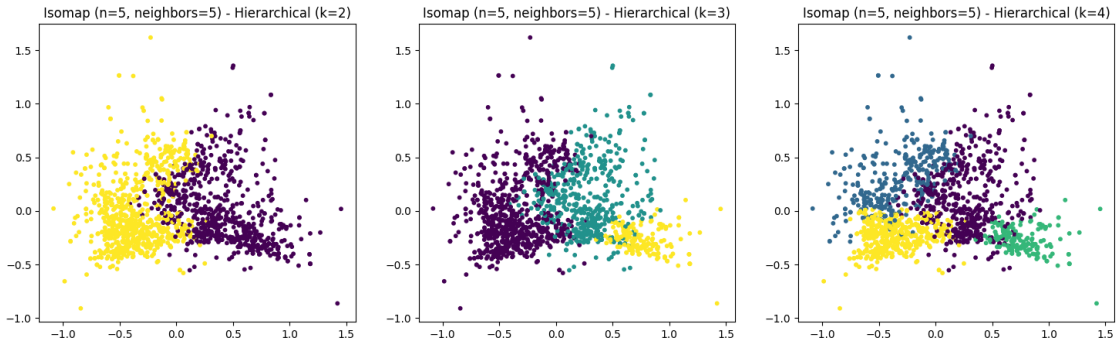

MDS (n=3, No Métrico) - Hierarchical (k=2)

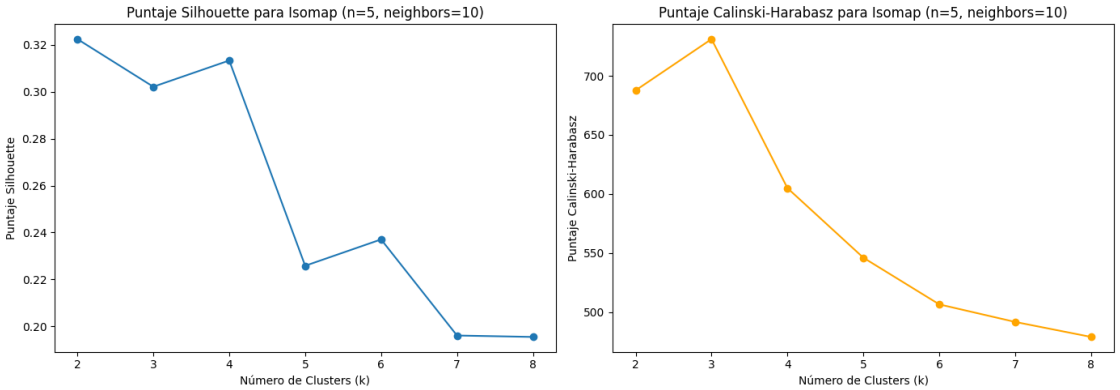

MDS (n=3, No Métrico) - Hierarchical (k=3)

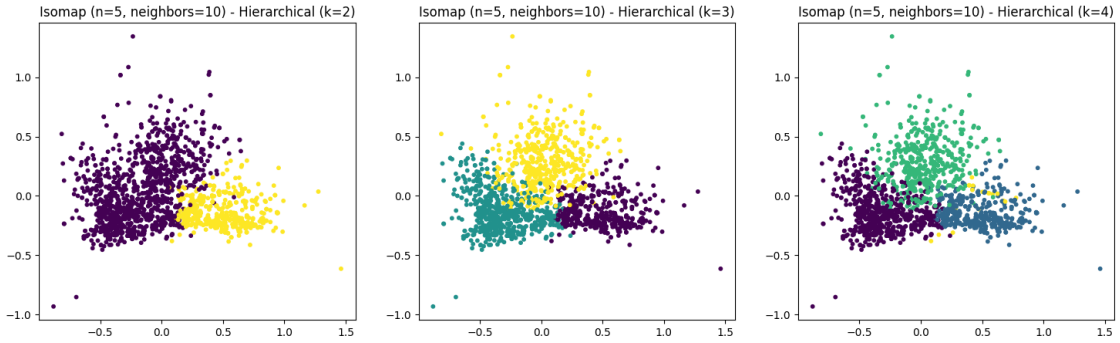

MDS (n=3, No Métrico) - Hierarchical (k=4)

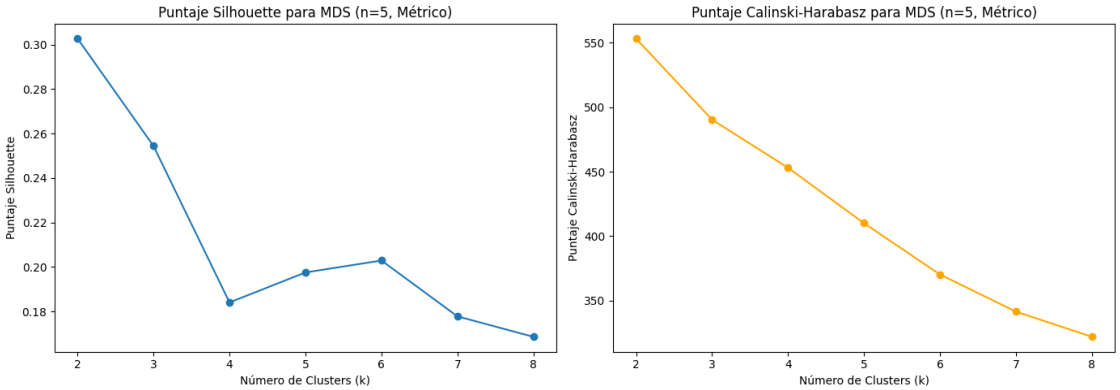

Puntaje Silhouette para Isomap (n=5, neighbors=5)
Puntaje Silhouette
Número de Clusters (k)

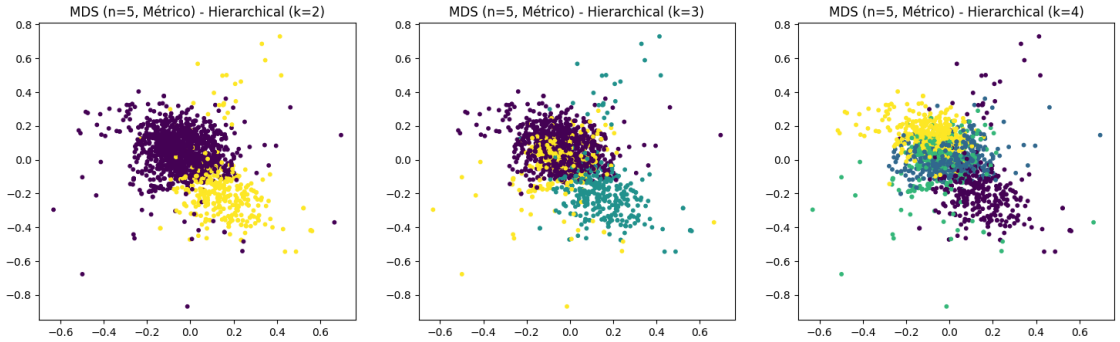

Puntaje Calinski-Harabasz para Isomap (n=5, neighbors=5)
Puntaje Calinski-Harabasz
Número de Clusters (k)

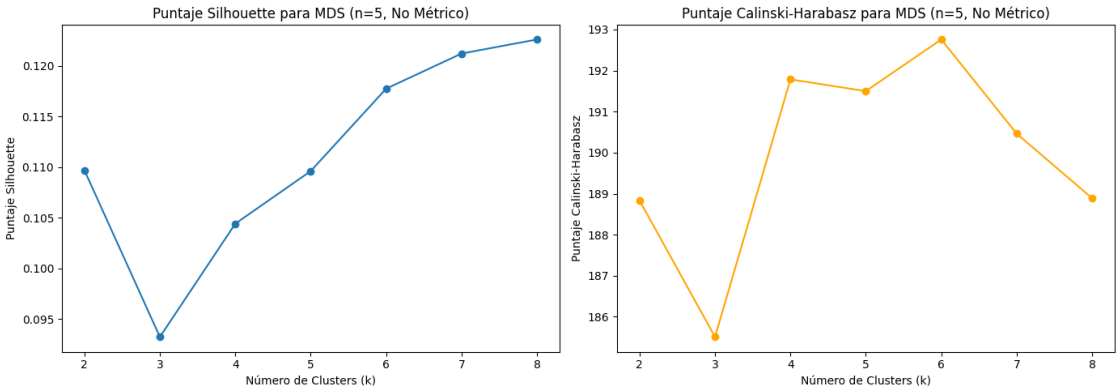

Isomap (n=5, neighbors=5) - Hierarchical (k=2)
Isomap (n=5, neighbors=5) - Hierarchical (k=3)
Isomap (n=5, neighbors=5) - Hierarchical (k=4)

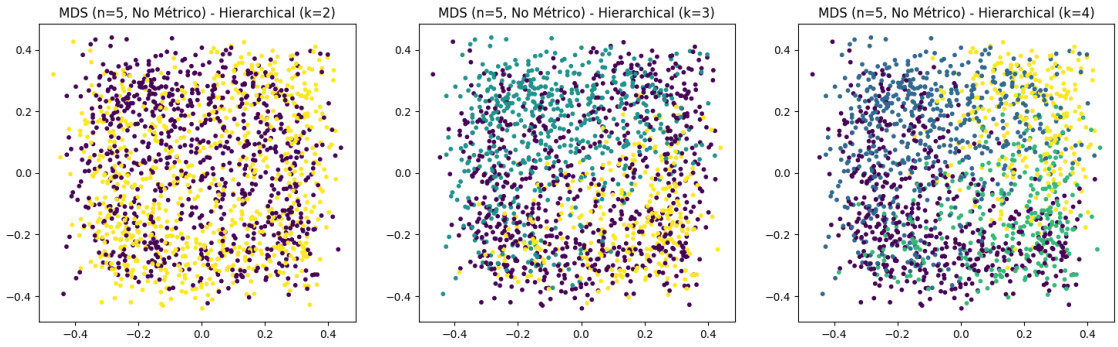

Puntaje Silhouette para Isomap (n=5, neighbors=10)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para Isomap (n=5, neighbors=10)
Puntaje Calinski-Harabasz
Número de Clusters (k)

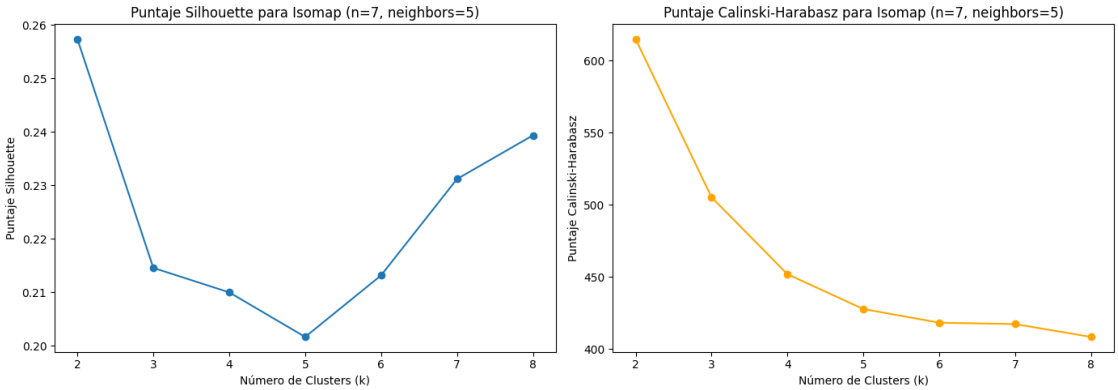

Isomap (n=5, neighbors=10) - Hierarchical (k=2)
Isomap (n=5, neighbors=10) - Hierarchical (k=3)
Isomap (n=5, neighbors=10) - Hierarchical (k=4)

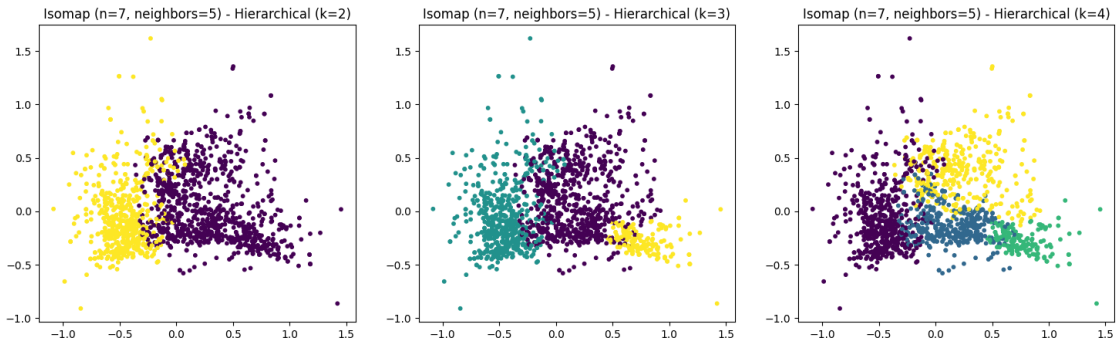

Puntaje Silhouette para MDS (n=5, Métrico)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para MDS (n=5, Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)


MDS (n=5, Métrico) - Hierarchical (k=2)
MDS (n=5, Métrico) - Hierarchical (k=3)
MDS (n=5, Métrico) - Hierarchical (k=4)

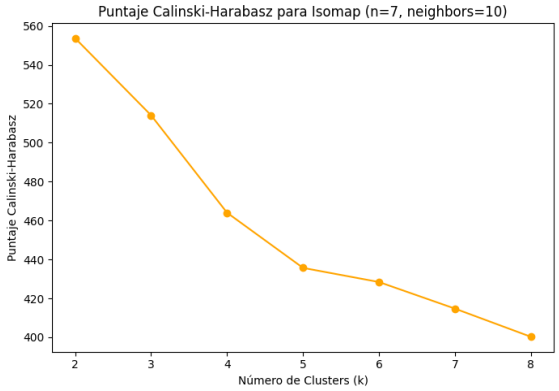

Puntaje Silhouette para MDS (n=5, No Métrico)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para MDS (n=5, No Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

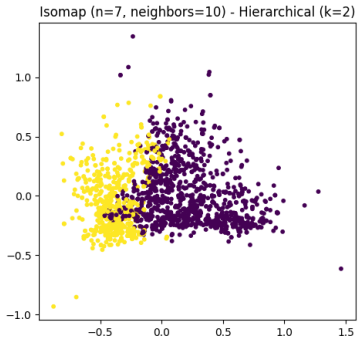

MDS (n=5, No Métrico) - Hierarchical (k=2)
MDS (n=5, No Métrico) - Hierarchical (k=3)
MDS (n=5, No Métrico) - Hierarchical (k=4)

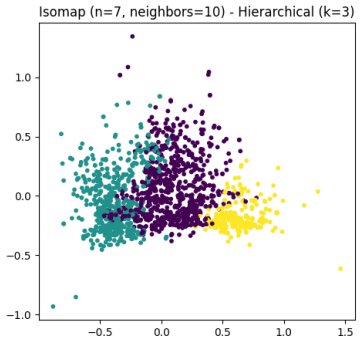

Puntaje Silhouette para Isomap (n=7, neighbors=5)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para Isomap (n=7, neighbors=5)
Puntaje Calinski-Harabasz
Número de Clusters (k)

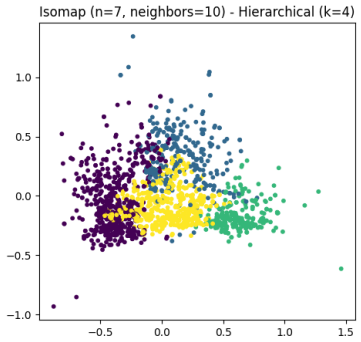

Isomap (n=7, neighbors=5) - Hierarchical (k=2)
Isomap (n=7, neighbors=5) - Hierarchical (k=3)
Isomap (n=7, neighbors=5) - Hierarchical (k=4)

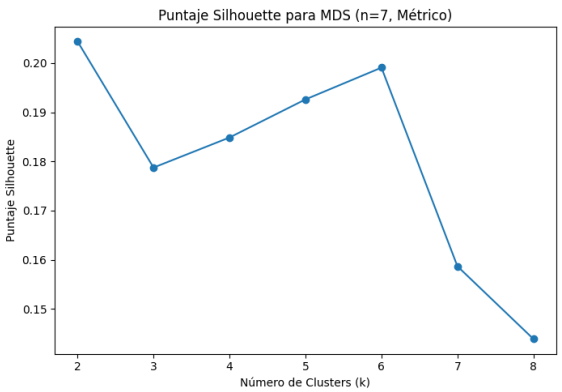

Puntaje Silhouette para Isomap (n=7, neighbors=10)
Puntaje Silhouette
Número de Clusters (k)

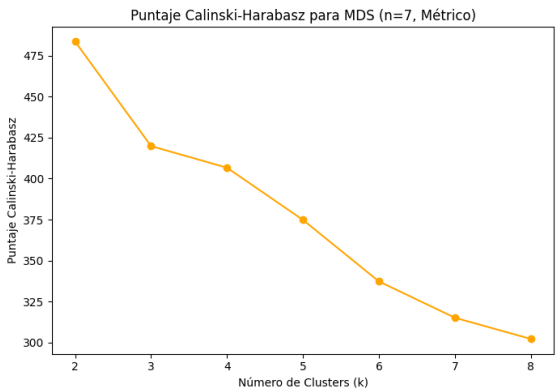

Puntaje Calinski-Harabasz para Isomap (n=7, neighbors=10)
Puntaje Calinski-Harabasz
Número de Clusters (k)

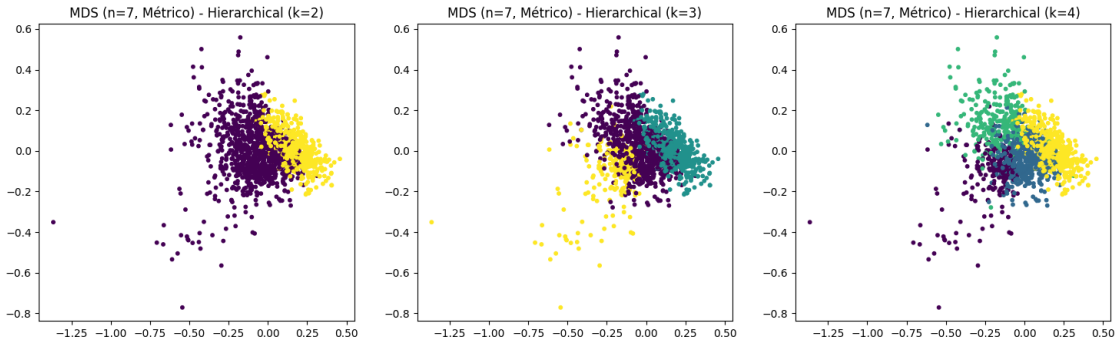

Isomap (n=7, neighbors=10) - Hierarchical (k=2)

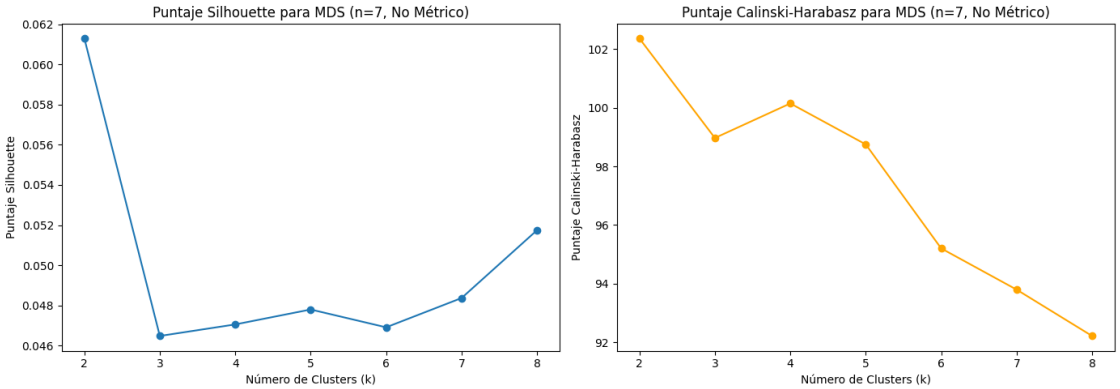

Isomap (n=7, neighbors=10) - Hierarchical (k=3)

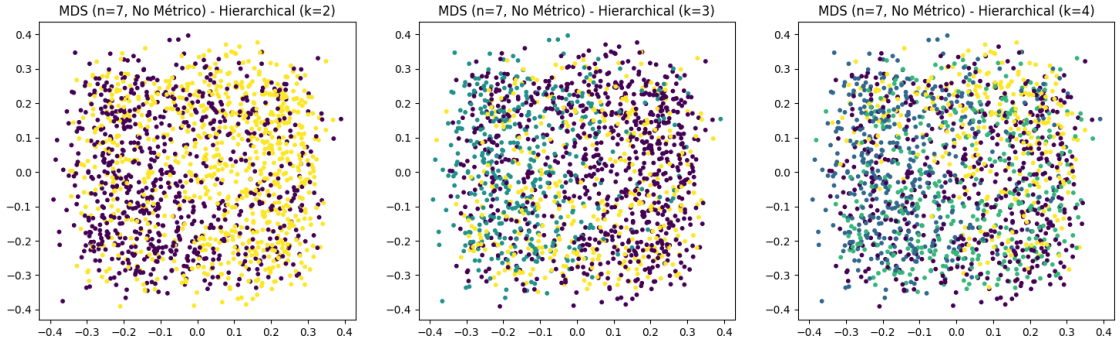

Isomap (n=7, neighbors=10) - Hierarchical (k=4)


Puntaje Silhouette para MDS (n=7, Métrico)
Puntaje Silhouette
Número de Clusters (k)


Puntaje Calinski-Harabasz para MDS (n=7, Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)







#Mejor Esquema (Hierarchical)

[54]:
```
# Función para encontrar el mejor esquema de clustering
def find_best_scheme_hierarchical(clustering_metrics_hierarchical,␣
 ↪reduction_errors):
    best_scheme = None
    best_score = -np.inf
```

```
    for reduction_key, metrics in clustering_metrics_hierarchical.items():
        # Obtiene el error de reducción de dimensionalidad
        reduction_error = reduction_errors.get(reduction_key, 0)

        for k, (silhouette_avg, calinski_harabasz) in metrics.items():
            # Calcular un puntaje combinado para cada esquema (mayor es mejor)
            score = (silhouette_avg + calinski_harabasz) / 2 - reduction_error

            # Actualiza el mejor esquema si encuentra un puntaje más alto
            if score > best_score:
                best_score = score
                best_scheme = (reduction_key, k, silhouette_avg,␣
 ↪calinski_harabasz, reduction_error)

    return best_scheme

# Encuentra el mejor esquema de clasificación
best_scheme_hierarchical =␣
 ↪find_best_scheme_hierarchical(clustering_metrics_hierarchical,␣
 ↪reduction_errors)

# Muestra el mejor esquema encontrado
print("Mejor esquema de clasificación (Hierarchical):")
print(f"Reducción de Dimensionalidad: {best_scheme_hierarchical[0]}")
print(f"Número de Clusters (k): {best_scheme_hierarchical[1]}")
print(f"Puntaje Silhouette Promedio: {best_scheme_hierarchical[2]:.2f}")
print(f"Puntaje Calinski-Harabasz Promedio: {best_scheme_hierarchical[3]:.2f}")
print(f"Error de Reducción: {best_scheme_hierarchical[4]:.2f}")
```

```
Mejor esquema de clasificación (Hierarchical):
Reducción de Dimensionalidad: Isomap (n=3, neighbors=10)
Número de Clusters (k): 3
Puntaje Silhouette Promedio: 0.39
Puntaje Calinski-Harabasz Promedio: 1095.33
Error de Reducción: 0.17
```

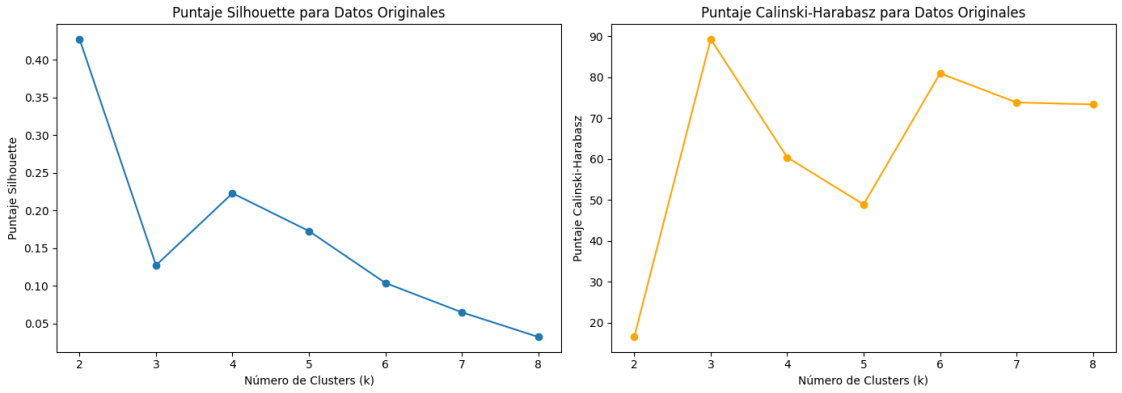
#GMM

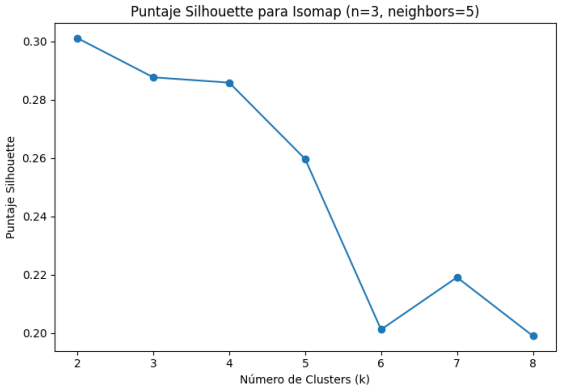
[55]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score, calinski_harabasz_score
import warnings

warnings.filterwarnings("ignore")

# Diccionario para almacenar métricas de clustering para cada configuración
clustering_metrics_gmm = {}
```

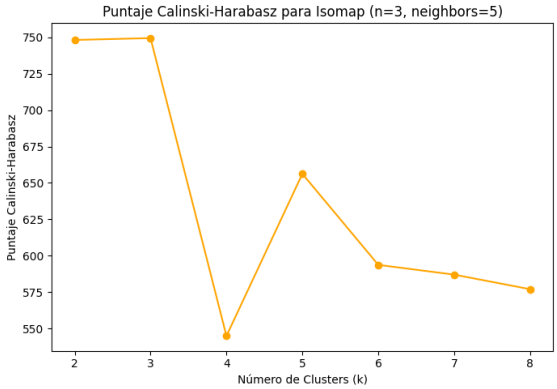
```python
# Función para aplicar Gaussian Mixture Models (GMM) y calcular métricas
def apply_gmm_clustering(data, n_clusters, reduction_key, random_state=42):
    clustering_results = {}
    for k in n_clusters:
        model = GaussianMixture(n_components=k, random_state=random_state)
        labels = model.fit_predict(data)
        silhouette_avg = silhouette_score(data, labels)
        calinski_harabasz = calinski_harabasz_score(data, labels)
        clustering_results[k] = (labels, silhouette_avg, calinski_harabasz)

        # Almacenar las métricas en el diccionario global
        if reduction_key not in clustering_metrics_gmm:
            clustering_metrics_gmm[reduction_key] = {}
        clustering_metrics_gmm[reduction_key][k] = (silhouette_avg,␣
 ↪calinski_harabasz)

    return clustering_results

# Función para graficar las métricas de clusterización
def plot_clustering_metrics(clustering_results, description):
    k_values = list(clustering_results.keys())
    silhouette_scores = [clustering_results[k][1] for k in k_values]
    calinski_harabasz_scores = [clustering_results[k][2] for k in k_values]

    # Crear gráficos para Silhouette y Calinski-Harabasz
    fig, axs = plt.subplots(1, 2, figsize=(14, 5))

    # Gráfico de Silhouette Score
    axs[0].plot(k_values, silhouette_scores, marker='o')
    axs[0].set_xlabel("Número de Clusters (k)")
    axs[0].set_ylabel("Puntaje Silhouette")
    axs[0].set_title(f"Puntaje Silhouette para {description}")

    # Gráfico de Calinski-Harabasz Score
    axs[1].plot(k_values, calinski_harabasz_scores, marker='o', color='orange')
    axs[1].set_xlabel("Número de Clusters (k)")
    axs[1].set_ylabel("Puntaje Calinski-Harabasz")
    axs[1].set_title(f"Puntaje Calinski-Harabasz para {description}")

    plt.tight_layout()
    plt.show()

# Función para graficar los resultados de la clusterización en el espacio␣
 ↪reducido
def plot_clustering_results(ax, data_reduced, labels, title):
```

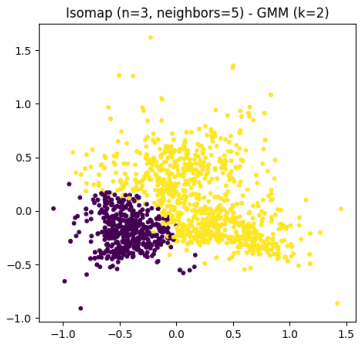
```
    ax.scatter(data_reduced[:, 0], data_reduced[:, 1], c=labels,␣
 ↪cmap='viridis', s=10)
    ax.set_title(title)

# Función principal para aplicar reducción de dimensionalidad, clustering y␣
 ↪visualizar resultados
def dimensionality_reduction_clustering_with_visuals(results, n_components,␣
 ↪n_neighbors_isomap, n_clusters, data_normalized):
    # Aplicar clusterización a los datos originales
    original_clustering_results = apply_gmm_clustering(data_normalized,␣
 ↪n_clusters, "Datos Originales")
    plot_clustering_metrics(original_clustering_results, "Datos Originales")

    # Graficar los resultados para cada configuración de reducción de␣
 ↪dimensionalidad
    for n in n_components:
        for neighbors in n_neighbors_isomap:
            key = f'Isomap (n={n}, neighbors={neighbors})'
            data_reduced = results[key]
            reduced_clustering_results = apply_gmm_clustering(data_reduced,␣
 ↪n_clusters, key)

            # Gráficos de métricas para cada configuración
            plot_clustering_metrics(reduced_clustering_results, key)

            # Visualización de clustering en el espacio reducido para 3 valores␣
 ↪de k (por ejemplo, k=2, k=3, y k=4)
            fig, axs = plt.subplots(1, 3, figsize=(18, 5))
            k_values_for_visuals = [2, 3, 4]
            for i, k in enumerate(k_values_for_visuals):
                labels = reduced_clustering_results[k][0]
                plot_clustering_results(axs[i], data_reduced, labels, f'{key} -␣
 ↪GMM (k={k})')
            plt.show()

        # MDS Métrico
        key = f'MDS (n={n}, Métrico)'
        data_reduced = results[key]
        reduced_clustering_results = apply_gmm_clustering(data_reduced,␣
 ↪n_clusters, key)

        # Gráficos de métricas para MDS Métrico
        plot_clustering_metrics(reduced_clustering_results, key)

        # Visualización de clustering en el espacio reducido para 3 valores de␣
 ↪k (k=2, k=3, y k=4)
```

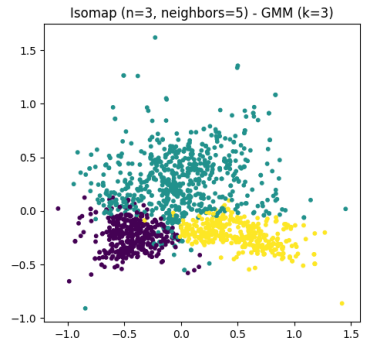
```
        fig, axs = plt.subplots(1, 3, figsize=(18, 5))
        for i, k in enumerate(k_values_for_visuals):
            labels = reduced_clustering_results[k][0]
            plot_clustering_results(axs[i], data_reduced, labels, f'{key} - GMM␣
 ↪(k={k})')
        plt.show()

        # MDS No Métrico
        key = f'MDS (n={n}, No Métrico)'
        data_reduced = results[key]
        reduced_clustering_results = apply_gmm_clustering(data_reduced,␣
 ↪n_clusters, key)

        # Gráficos de métricas para MDS No Métrico
        plot_clustering_metrics(reduced_clustering_results, key)

        # Visualización de clustering en el espacio reducido para 3 valores de␣
 ↪k (k=2, k=3, y k=4)
        fig, axs = plt.subplots(1, 3, figsize=(18, 5))
        for i, k in enumerate(k_values_for_visuals):
            labels = reduced_clustering_results[k][0]
            plot_clustering_results(axs[i], data_reduced, labels, f'{key} - GMM␣
 ↪(k={k})')
        plt.show()

# Parámetros de número de clusters
n_clusters = [2, 3, 4, 5, 6, 7, 8]  # Número de clusters para GMM

# Ejecuta la función con los resultados de reducción de dimensionalidad y␣
 ↪clustering
dimensionality_reduction_clustering_with_visuals(results, n_components,␣
 ↪n_neighbors_isomap, n_clusters, data_normalized)
```

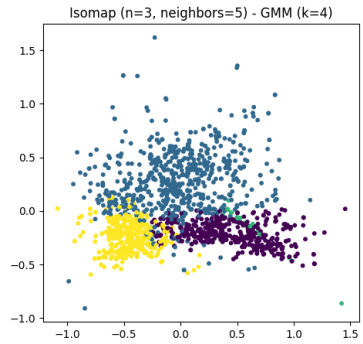



Puntaje Silhouette para Isomap (n=3, neighbors=5)
Puntaje Silhouette
Número de Clusters (k)

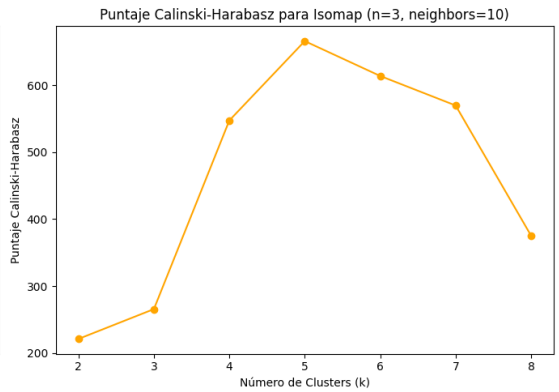

Puntaje Calinski-Harabasz para Isomap (n=3, neighbors=5)
Puntaje Calinski-Harabasz
Número de Clusters (k)

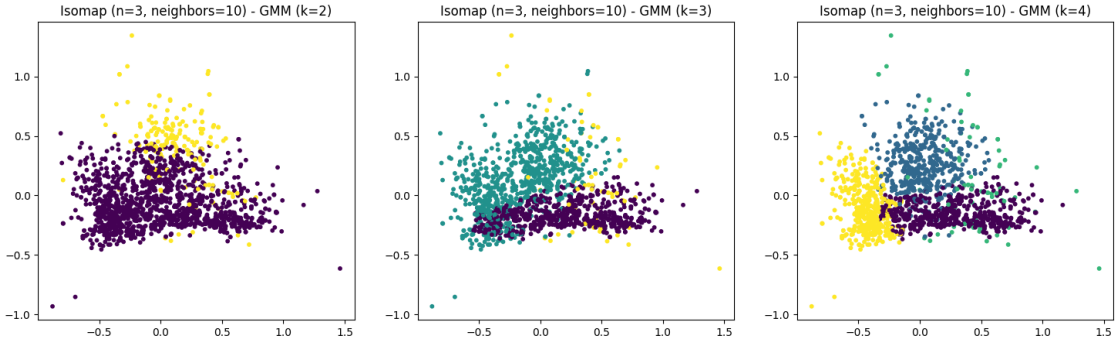

Isomap (n=3, neighbors=5) - GMM (k=2)

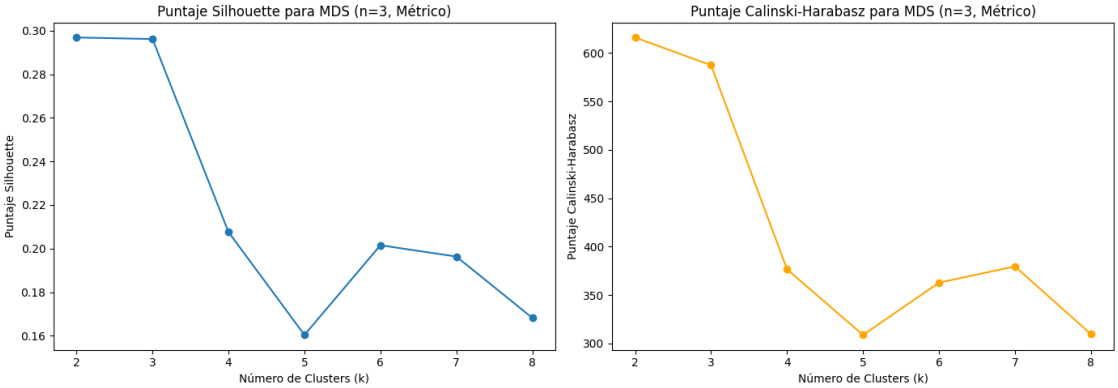

Isomap (n=3, neighbors=5) - GMM (k=3)

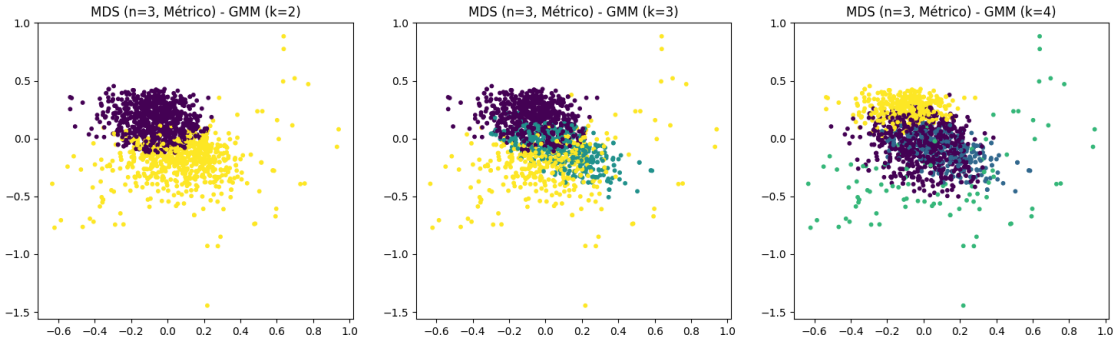

Isomap (n=3, neighbors=5) - GMM (k=4)

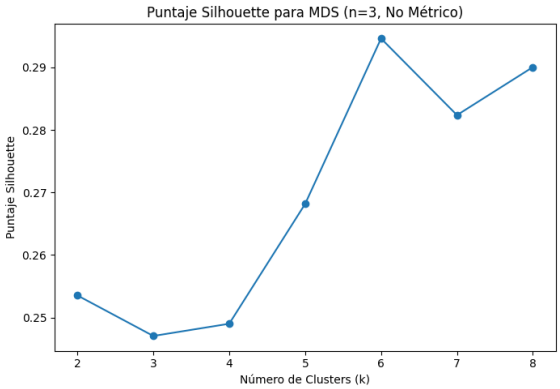

Puntaje Silhouette para Isomap (n=3, neighbors=10)
Puntaje Silhouette
Número de Clusters (k)

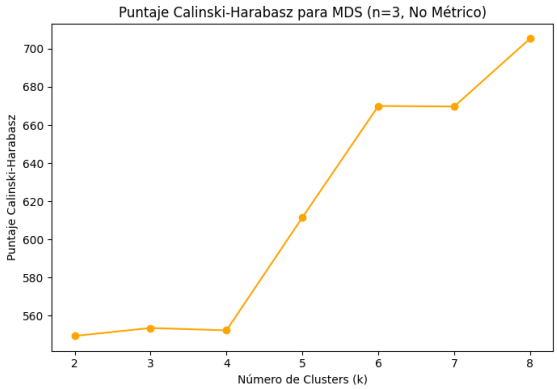

Puntaje Calinski-Harabasz para Isomap (n=3, neighbors=10)
Puntaje Calinski-Harabasz
Número de Clusters (k)

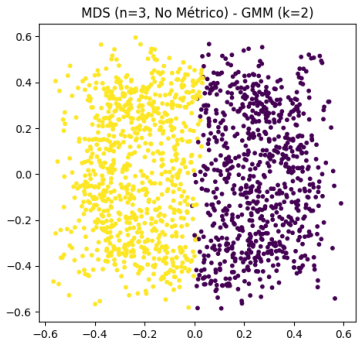

Isomap (n=3, neighbors=10) - GMM (k=2)
Isomap (n=3, neighbors=10) - GMM (k=3)
Isomap (n=3, neighbors=10) - GMM (k=4)

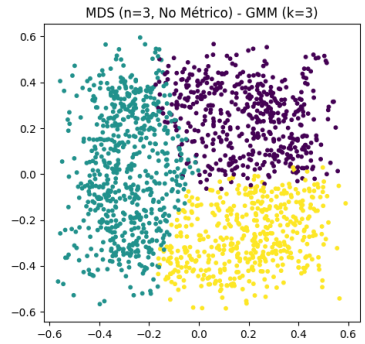

Puntaje Silhouette para MDS (n=3, Métrico)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para MDS (n=3, Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

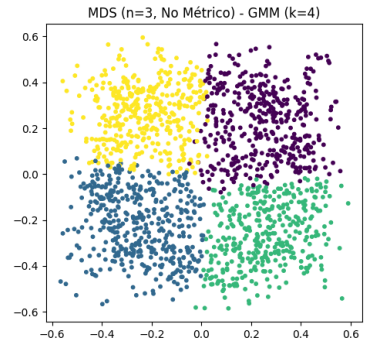

MDS (n=3, Métrico) - GMM (k=2)
MDS (n=3, Métrico) - GMM (k=3)
MDS (n=3, Métrico) - GMM (k=4)

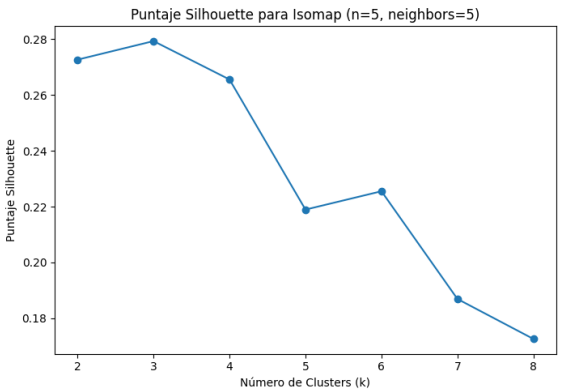

Puntaje Silhouette para MDS (n=3, No Métrico)
Puntaje Silhouette
Número de Clusters (k)

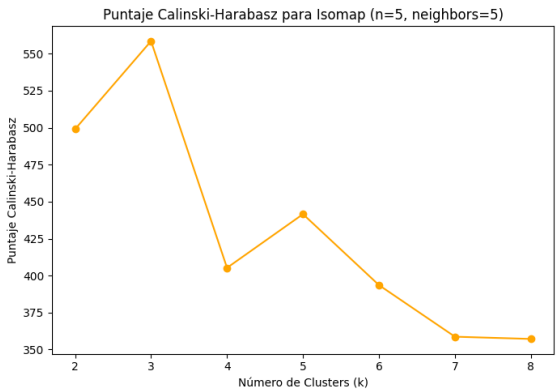

Puntaje Calinski-Harabasz para MDS (n=3, No Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

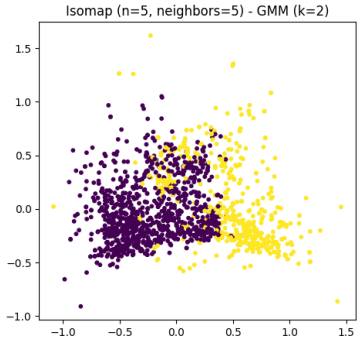

MDS (n=3, No Métrico) - GMM (k=2)

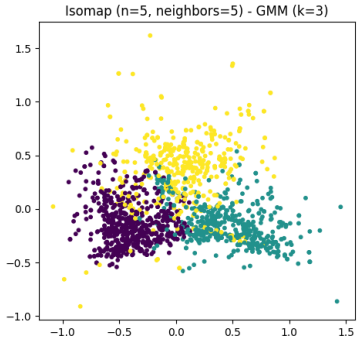

MDS (n=3, No Métrico) - GMM (k=3)

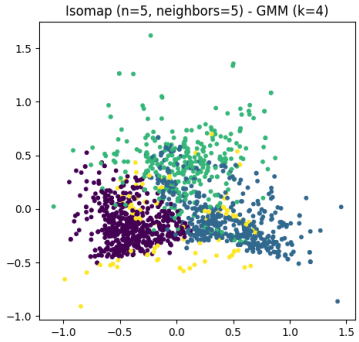

MDS (n=3, No Métrico) - GMM (k=4)

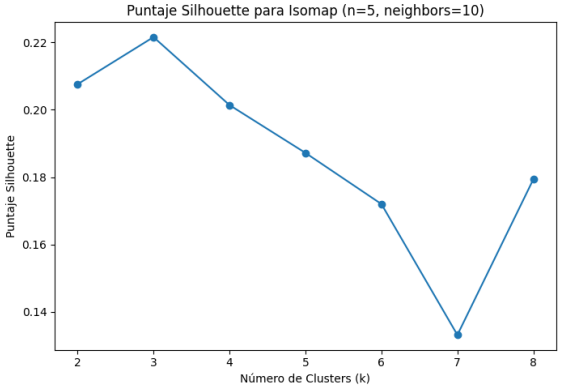

Puntaje Silhouette para Isomap (n=5, neighbors=5)
Puntaje Silhouette
Número de Clusters (k)

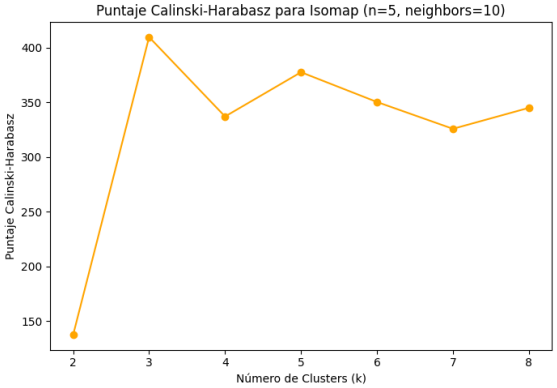

Puntaje Calinski-Harabasz para Isomap (n=5, neighbors=5)
Puntaje Calinski-Harabasz
Número de Clusters (k)

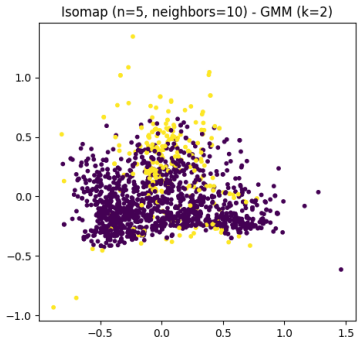

Isomap (n=5, neighbors=5) - GMM (k=2)

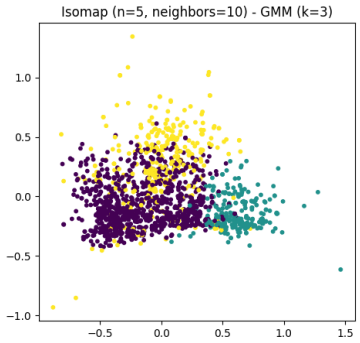

Isomap (n=5, neighbors=5) - GMM (k=3)

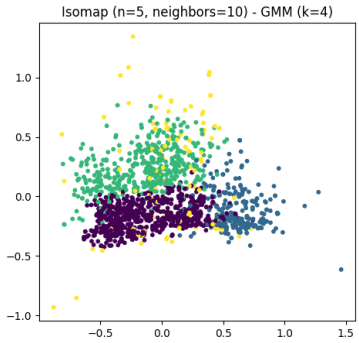

Isomap (n=5, neighbors=5) - GMM (k=4)

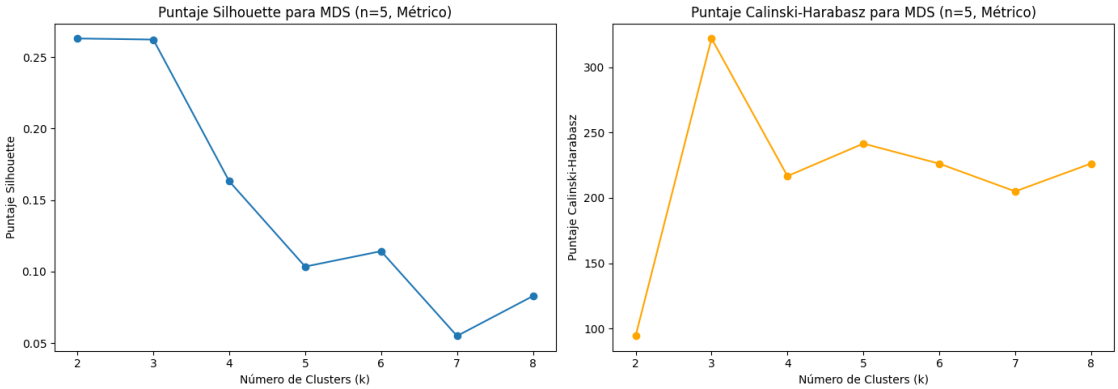

Puntaje Silhouette para Isomap (n=5, neighbors=10)
Puntaje Silhouette
Número de Clusters (k)

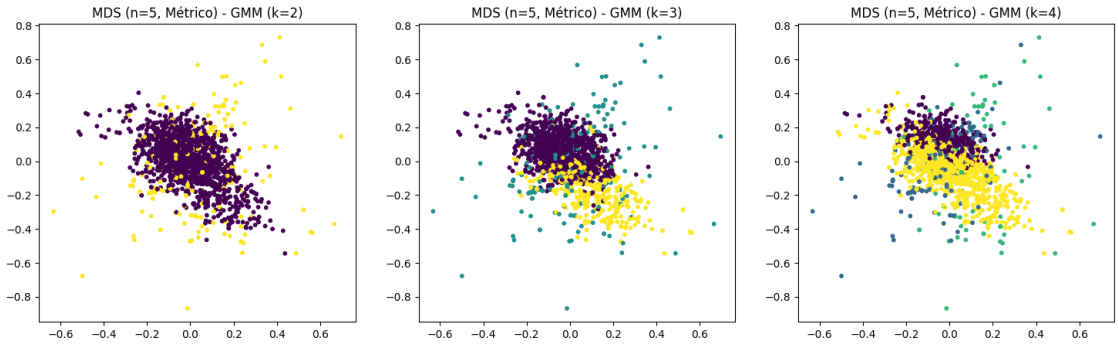

Puntaje Calinski-Harabasz para Isomap (n=5, neighbors=10)
Puntaje Calinski-Harabasz
Número de Clusters (k)

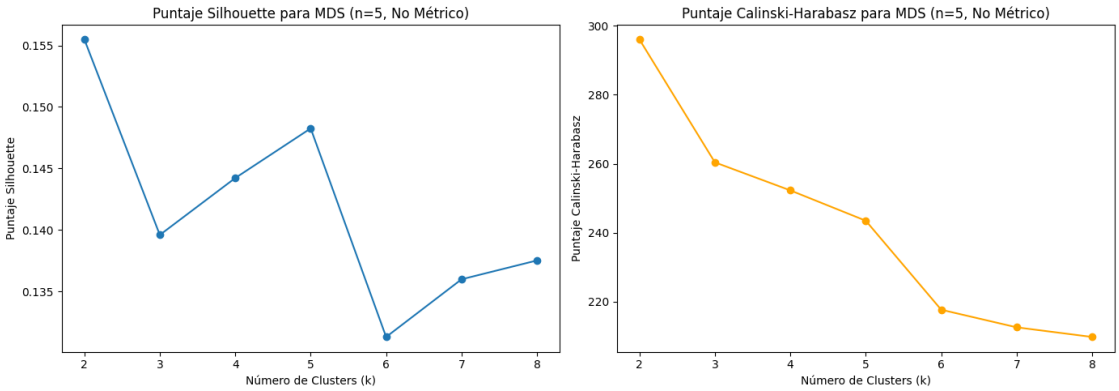

Isomap (n=5, neighbors=10) - GMM (k=2)

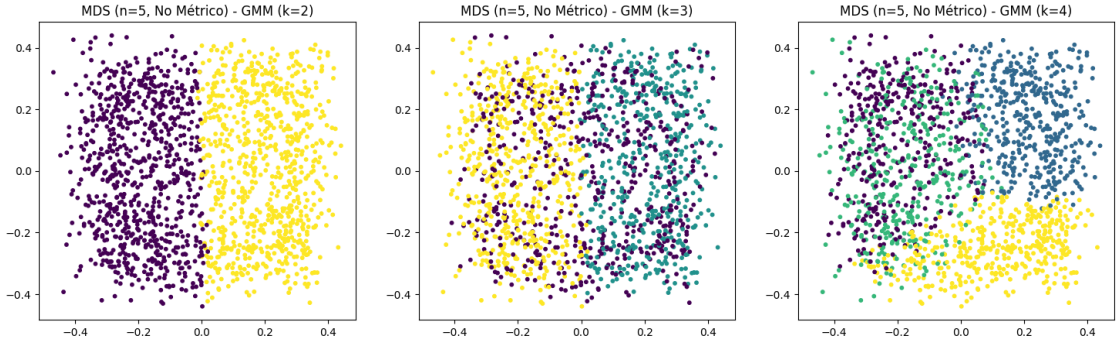

Isomap (n=5, neighbors=10) - GMM (k=3)

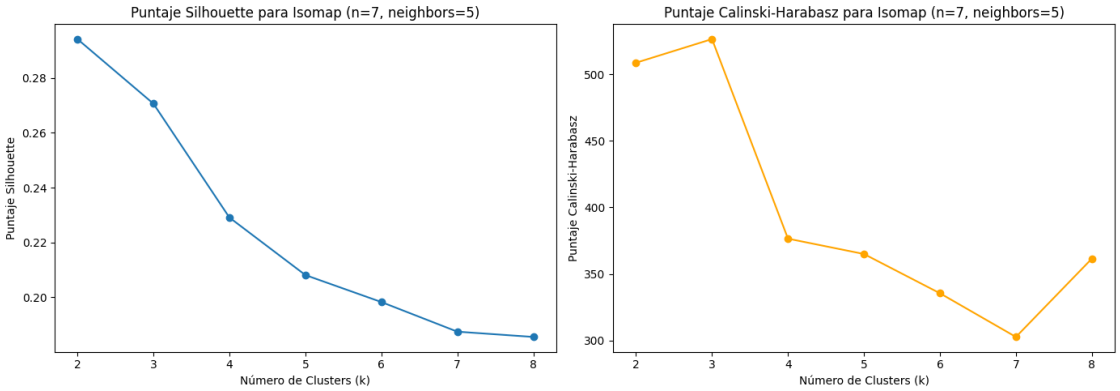

Isomap (n=5, neighbors=10) - GMM (k=4)

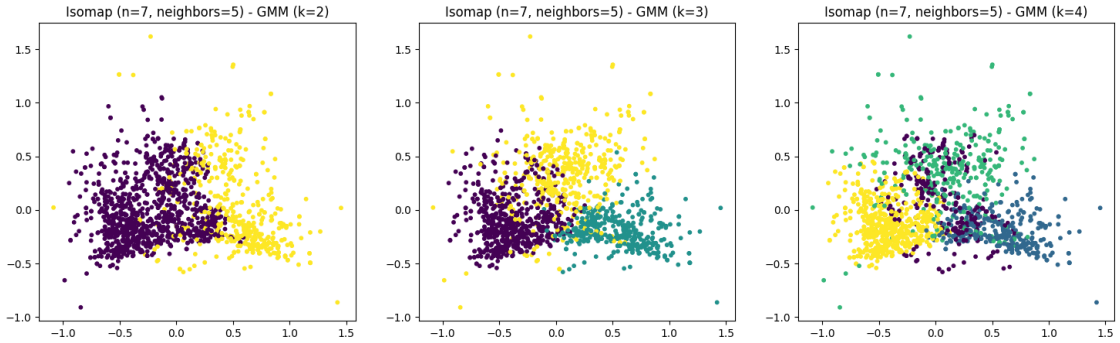

Puntaje Silhouette para MDS (n=5, Métrico)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para MDS (n=5, Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)


MDS (n=5, Métrico) - GMM (k=2)
MDS (n=5, Métrico) - GMM (k=3)
MDS (n=5, Métrico) - GMM (k=4)

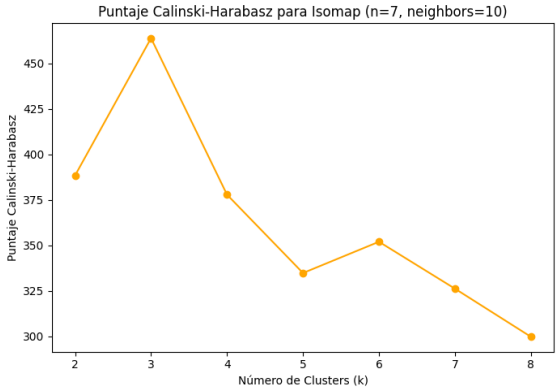

Puntaje Silhouette para MDS (n=5, No Métrico)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para MDS (n=5, No Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)

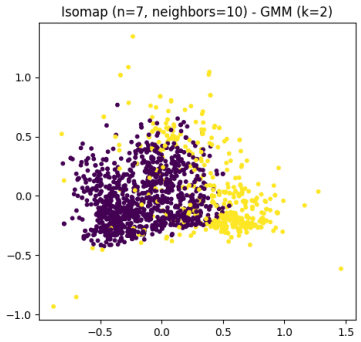

MDS (n=5, No Métrico) - GMM (k=2)
MDS (n=5, No Métrico) - GMM (k=3)
MDS (n=5, No Métrico) - GMM (k=4)

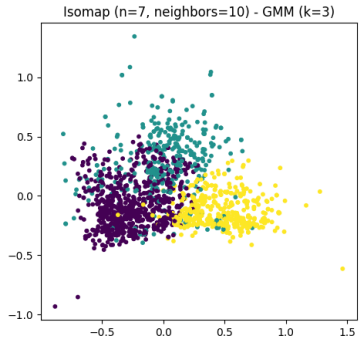

Puntaje Silhouette para Isomap (n=7, neighbors=5)
Puntaje Silhouette
Número de Clusters (k)
Puntaje Calinski-Harabasz para Isomap (n=7, neighbors=5)
Puntaje Calinski-Harabasz
Número de Clusters (k)

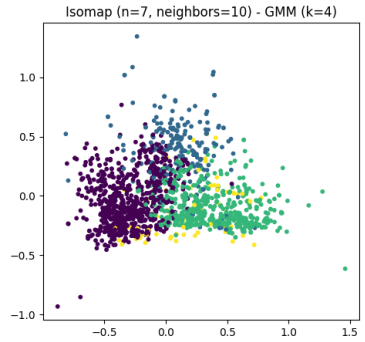

Isomap (n=7, neighbors=5) - GMM (k=2)
Isomap (n=7, neighbors=5) - GMM (k=3)
Isomap (n=7, neighbors=5) - GMM (k=4)

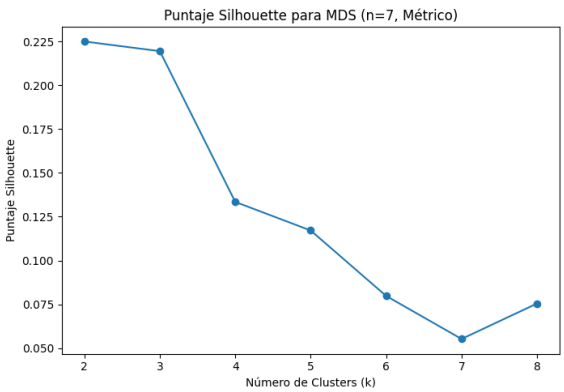

Puntaje Silhouette para Isomap (n=7, neighbors=10)
Puntaje Silhouette
Número de Clusters (k)

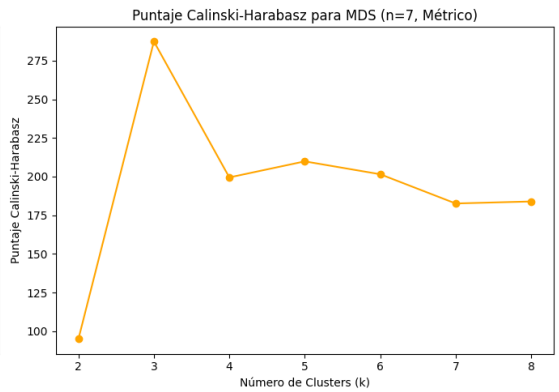

Puntaje Calinski-Harabasz para Isomap (n=7, neighbors=10)
Puntaje Calinski-Harabasz
Número de Clusters (k)

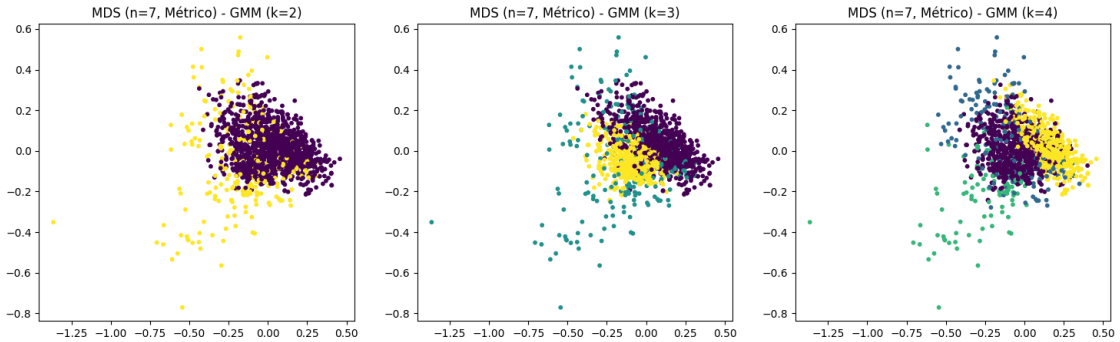

Isomap (n=7, neighbors=10) - GMM (k=2)

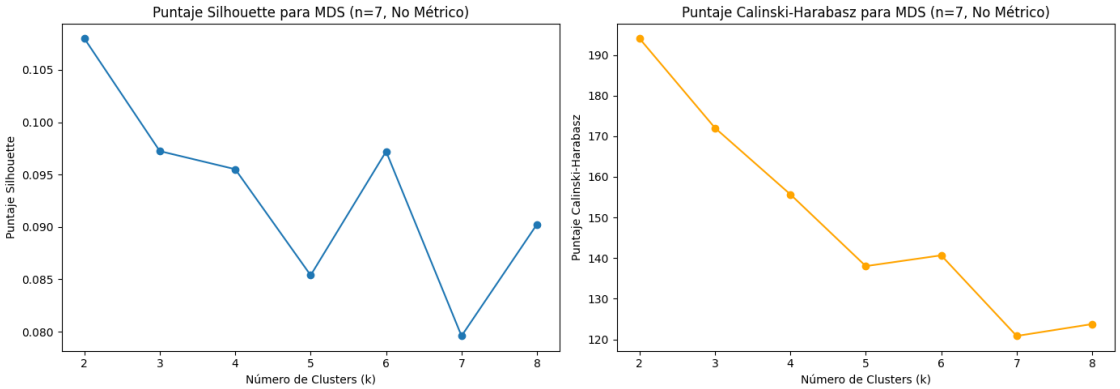

Isomap (n=7, neighbors=10) - GMM (k=3)

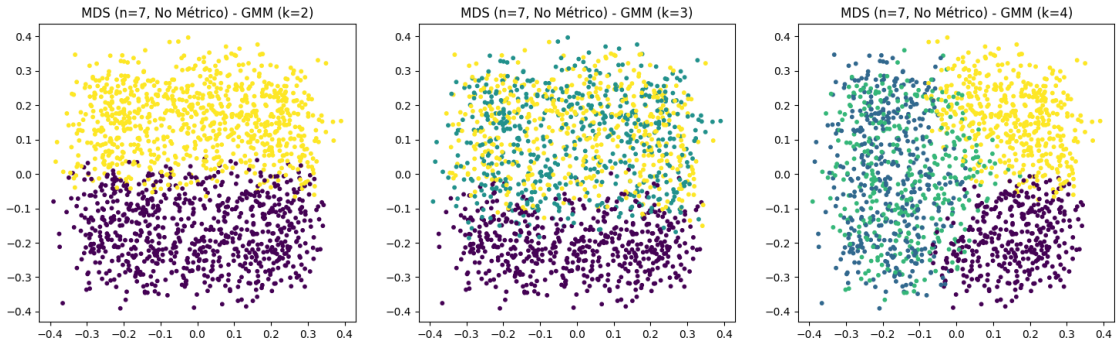

Isomap (n=7, neighbors=10) - GMM (k=4)


Puntaje Silhouette para MDS (n=7, Métrico)
Puntaje Silhouette
Número de Clusters (k)


Puntaje Calinski-Harabasz para MDS (n=7, Métrico)
Puntaje Calinski-Harabasz
Número de Clusters (k)







#Mejor Esquema (GMM)

[56]:
```
# Función para encontrar el mejor esquema de clustering para GMM
def find_best_scheme_gmm(clustering_metrics_gmm, reduction_errors):
    best_scheme = None
    best_score = -np.inf

    for reduction_key, metrics in clustering_metrics_gmm.items():
```

```
        # Obtiene el error de reducción de dimensionalidad
        reduction_error = reduction_errors.get(reduction_key, 0)

        for k, (silhouette_avg, calinski_harabasz) in metrics.items():
            # Calcular un puntaje combinado para cada esquema (mayor es mejor)
            score = (silhouette_avg + calinski_harabasz) / 2 - reduction_error

            # Actualiza el mejor esquema si encuentra un puntaje más alto
            if score > best_score:
                best_score = score
                best_scheme = (reduction_key, k, silhouette_avg,␣
 ↪calinski_harabasz, reduction_error)

    return best_scheme

# Encuentra el mejor esquema de clasificación en GMM
best_scheme_gmm = find_best_scheme_gmm(clustering_metrics_gmm, reduction_errors)

# Muestra el mejor esquema encontrado
print("Mejor esquema de clasificación (GMM):")
print(f"Reducción de Dimensionalidad: {best_scheme_gmm[0]}")
print(f"Número de Clusters (k): {best_scheme_gmm[1]}")
print(f"Puntaje Silhouette Promedio: {best_scheme_gmm[2]:.2f}")
print(f"Puntaje Calinski-Harabasz Promedio: {best_scheme_gmm[3]:.2f}")
print(f"Error de Reducción: {best_scheme_gmm[4]:.2f}")
```

```
Mejor esquema de clasificación (GMM):
Reducción de Dimensionalidad: Isomap (n=3, neighbors=5)
Número de Clusters (k): 3
Puntaje Silhouette Promedio: 0.29
Puntaje Calinski-Harabasz Promedio: 749.33
Error de Reducción: 0.30
```

#Mejor Esquema (Overall)

[57]:
```
# Función para encontrar el mejor esquema entre los tres algoritmos de␣
 ↪clustering
def find_overall_best_scheme(best_scheme_kmeans, best_scheme_hierarchical,␣
 ↪best_scheme_gmm):
    # Inicializa variables para el mejor esquema general
    best_overall_scheme = None
    best_score = -np.inf

    # Diccionario con los resultados de los mejores esquemas para cada algoritmo
    best_schemes = {
        "KMeans": best_scheme_kmeans,
        "Hierarchical": best_scheme_hierarchical,
```

```
        "GMM": best_scheme_gmm
    }

    # Itera sobre cada esquema y calcula el puntaje combinado para cada uno
    for algorithm, scheme in best_schemes.items():
        reduction_key, k, silhouette_avg, calinski_harabasz, reduction_error =␣
 ↪scheme
        score = (silhouette_avg + calinski_harabasz) / 2 - reduction_error

        # Verifica si este esquema es mejor que el actual mejor esquema
        if score > best_score:
            best_score = score
            best_overall_scheme = (algorithm, reduction_key, k, silhouette_avg,␣
 ↪calinski_harabasz, reduction_error)

    return best_overall_scheme

# Encuentra el mejor esquema entre los resultados de KMeans, Hierarchical y GMM
best_overall_scheme = find_overall_best_scheme(best_scheme_kmeans,␣
 ↪best_scheme_hierarchical, best_scheme_gmm)

# Muestra el mejor esquema general encontrado
print("Mejor esquema de clasificación entre los tres algoritmos:")
print(f"Algoritmo: {best_overall_scheme[0]}")
print(f"Reducción de Dimensionalidad: {best_overall_scheme[1]}")
print(f"Número de Clusters (k): {best_overall_scheme[2]}")
print(f"Puntaje Silhouette Promedio: {best_overall_scheme[3]:.2f}")
print(f"Puntaje Calinski-Harabasz Promedio: {best_overall_scheme[4]:.2f}")
print(f"Error de Reducción: {best_overall_scheme[5]:.2f}")
```

```
Mejor esquema de clasificación entre los tres algoritmos:
Algoritmo: KMeans
Reducción de Dimensionalidad: Isomap (n=3, neighbors=10)
Número de Clusters (k): 3
Puntaje Silhouette Promedio: 0.40
Puntaje Calinski-Harabasz Promedio: 1169.67
Error de Reducción: 0.17
```

5.1 Importar Librerías y Cargar Datos

[ ]:
```
import pandas as pd
import numpy as np
from joblib import Parallel, delayed
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.decomposition import PCA
```

```
from sklearn.manifold import Isomap, MDS
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score, calinski_harabasz_score
import matplotlib.pyplot as plt
from IPython.display import display
import pandas as pd

def load_data(filepath):
    """Carga los datos desde un archivo Excel"""
    df = pd.read_excel(filepath)
    return df

def scale_data(df):
    """Escala los datos utilizando StandardScaler"""
    scaler = StandardScaler()
    data_scaled = scaler.fit_transform(df)
    return data_scaled

def normalize_data(df):
    """Normaliza los datos utilizando MinMaxScaler para llevarlos al rango [0,␣
 ↪1]"""
    scaler = MinMaxScaler(feature_range=(0, 1))
    data_normalized = scaler.fit_transform(df)
    return data_normalized

filepath = 'dataset_wq.xlsx'
df = load_data(filepath)
data_scaled = scale_data(df)
data_normalized = normalize_data(data_scaled)
```

5.2 Aplicar Isomap y MDS

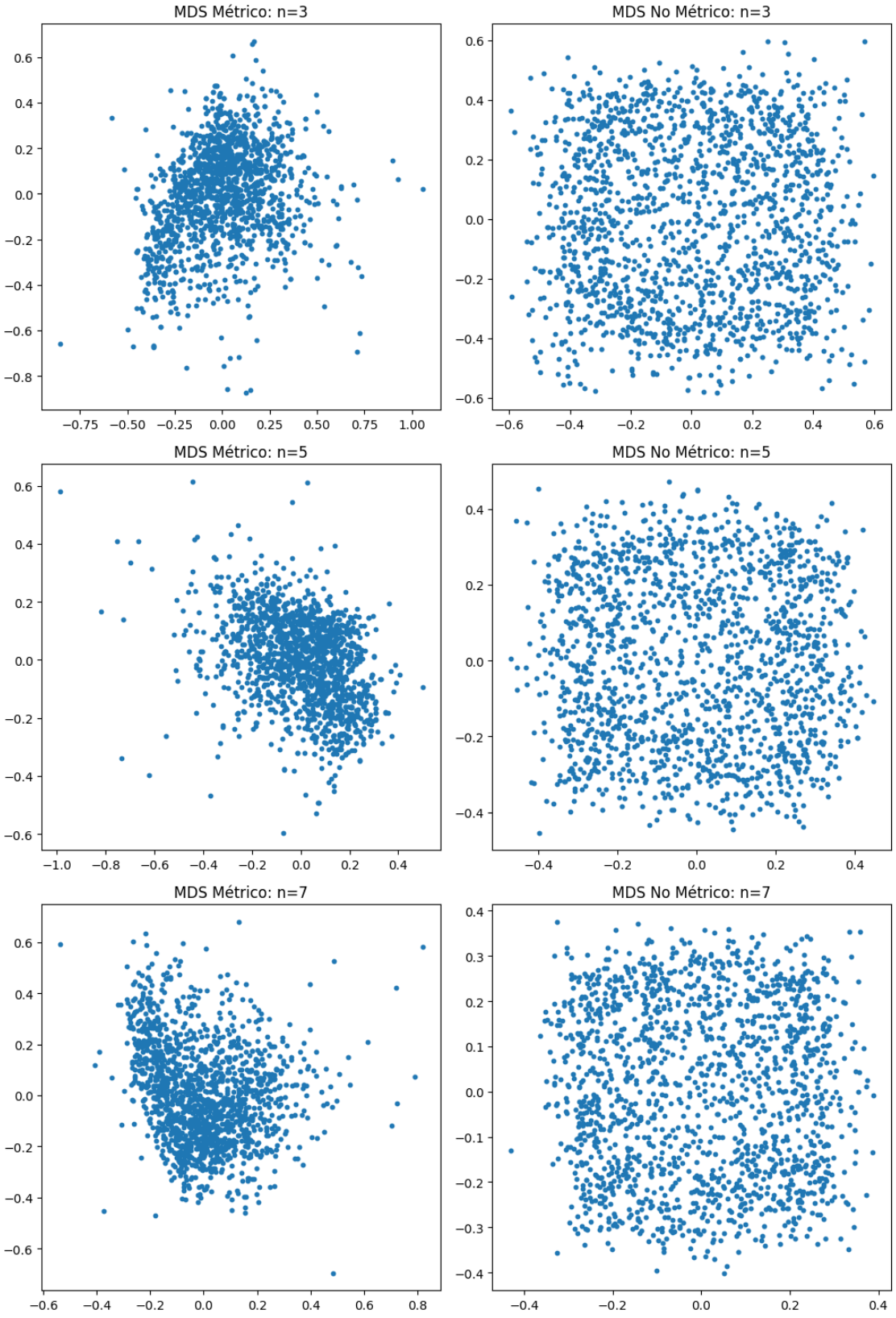
[ ]:
```
import warnings
warnings.filterwarnings("ignore")

def apply_isomap(data, n, neighbors, path_method='auto'):
    """Aplica Isomap con los parámetros dados"""
    isomap = Isomap(n_components=n, n_neighbors=neighbors,␣
 ↪path_method=path_method, n_jobs=1)
    return isomap.fit_transform(data), f'Isomap (n={n}, neighbors={neighbors},␣
 ↪path_method={path_method})'

def apply_mds(data, n, metric, n_init=3, max_iter=500):
    """Aplica MDS con los parámetros dados"""
```

```
    mds = MDS(n_components=n, metric=metric, n_init=n_init, max_iter=max_iter,␣
 ↪n_jobs=1)
    metric_type = 'Métrico' if metric else 'No Métrico'
    return mds.fit_transform(data), f'MDS {metric_type} (n={n})'

def run_isomap_parallel(data, n_components, n_neighbors_isomap,␣
 ↪path_methods=['auto']):
    """Aplica Isomap en paralelo con diferentes configuraciones de parámetros"""
    results = {}
    tasks = [(n, neighbors, path_method) for n in n_components for neighbors in␣
 ↪n_neighbors_isomap for path_method in path_methods]

    results_list = Parallel(n_jobs=1,␣
 ↪backend='threading')(delayed(apply_isomap)(data, n, neighbors, path_method)␣
 ↪for n, neighbors, path_method in tasks)

    for (n, neighbors, path_method), (data_reduced, label) in zip(tasks,␣
 ↪results_list):
        results[label] = data_reduced

    return results

def run_mds_parallel(data, n_components, metrics=[True, False], n_init=3,␣
 ↪max_iter=200):
    """Aplica MDS en paralelo con diferentes configuraciones de parámetros"""
    results = {}
    tasks = [(n, metric) for n in n_components for metric in metrics]

    results_list = Parallel(n_jobs=1,␣
 ↪backend='threading')(delayed(apply_mds)(data, n, metric, n_init, max_iter)␣
 ↪for n, metric in tasks)

    for (n, metric), (data_reduced, label) in zip(tasks, results_list):
        results[label] = data_reduced

    return results

n_components = [3, 5, 7]
n_neighbors_isomap = [5, 10]
path_methods = ['auto']
metrics = [True, False]

results_isomap = run_isomap_parallel(data_normalized, n_components,␣
 ↪n_neighbors_isomap, path_methods)
results_mds = run_mds_parallel(data_normalized, n_components, metrics,␣
 ↪n_init=3, max_iter=200)
```

5.3 Visualizar los resultados de reducción de dimensionalidad

```
[ ]: def visualize_dimensionality_reduction_results(results, n_components,␣
     ↪n_neighbors_isomap, figsize=(20, 15)):
         """
         Visualiza los resultados de reducción de dimensionalidad usando Isomap y MDS
         """
         fig, axs = plt.subplots(len(n_components), len(n_neighbors_isomap) + 2,␣
     ↪figsize=figsize)

         if len(n_components) == 1:
             axs = [axs]

         for i, n in enumerate(n_components):
             for j, neighbors in enumerate(n_neighbors_isomap):
                 ax = axs[i][j]
                 label = f'Isomap (n={n}, neighbors={neighbors},␣
     ↪path_method=dijkstra)'
                 if label in results:
                     data_isomap = results[label]
                     if data_isomap.shape[1] >= 2:
                         ax.scatter(data_isomap[:, 0], data_isomap[:, 1], s=10)
                         ax.set_title(f'Isomap: n={n}, neighbors={neighbors}')
                     else:
                         ax.set_visible(False)
                 else:
                     ax.set_visible(False)

             ax = axs[i][-2]
             label_metric = f'MDS Métrico (n={n})'
             if label_metric in results:
                 data_mds_metric = results[label_metric]
                 if data_mds_metric.shape[1] >= 2:
                     ax.scatter(data_mds_metric[:, 0], data_mds_metric[:, 1], s=10)
                     ax.set_title(f'MDS Métrico: n={n}')
                 else:
                     ax.set_visible(False)
             else:
                 ax.set_visible(False)

             ax = axs[i][-1]
             label_nonmetric = f'MDS No Métrico (n={n})'
             if label_nonmetric in results:
                 data_mds_nonmetric = results[label_nonmetric]
                 if data_mds_nonmetric.shape[1] >= 2:
                     ax.scatter(data_mds_nonmetric[:, 0], data_mds_nonmetric[:, 1],␣
     ↪s=10)
```

```
                ax.set_title(f'MDS No Métrico: n={n}')
            else:
                ax.set_visible(False)
        else:
            ax.set_visible(False)

    plt.tight_layout()
    plt.show()

results = {**results_isomap, **results_mds}
visualize_dimensionality_reduction_results(results, n_components,␣
 ↪n_neighbors_isomap)
```


MDS Métrico: n=3
MDS No Métrico: n=3
MDS Métrico: n=5
MDS No Métrico: n=5
MDS Métrico: n=7
MDS No Métrico: n=7

5.4 Aplicar K-Means

[ ]:
```
def apply_kmeans_parallel(data, k_range=range(2, 9), n_init=10):
    def kmeans_clustering(k):
        if data is not None and len(data) > 0 and data.shape[1] >= 2:
            kmeans = KMeans(n_clusters=k, init='k-means++', n_init=n_init,␣
 ↪max_iter=300, random_state=42)
            labels = kmeans.fit_predict(data)
            silhouette = silhouette_score(data, labels)
            ch_score = calinski_harabasz_score(data, labels)
            return f'K-Means (k={k})', labels, silhouette, ch_score
        else:
            return f'K-Means (k={k})', None, None, None

    results = Parallel(n_jobs=-1)(delayed(kmeans_clustering)(k) for k in␣
 ↪k_range)
    return {key: (data, labels, silhouette, ch_score) for key, labels,␣
 ↪silhouette, ch_score in results if labels is not None}
```

[ ]:
```
# Aplicar K-Means con Isomap
kmeans_results_isomap = {}
for label, data_isomap in results_isomap.items():
    kmeans_results = apply_kmeans_parallel(data_isomap)
    for key, value in kmeans_results.items():
        label_clean = label.replace(", path_method=dijkstra", "")
        kmeans_results_isomap[f'{label_clean} - {key}'] = value
        print(f'{label_clean} - {key}: Silhouette = {value[2]:.2f},␣
 ↪Calinski-Harabasz = {value[3]:.2f}')
```

```
Isomap (n=3, neighbors=5, path_method=auto) - K-Means (k=2): Silhouette = 0.37,
Calinski-Harabasz = 1103.56
Isomap (n=3, neighbors=5, path_method=auto) - K-Means (k=3): Silhouette = 0.40,
Calinski-Harabasz = 1155.56
Isomap (n=3, neighbors=5, path_method=auto) - K-Means (k=4): Silhouette = 0.34,
Calinski-Harabasz = 1013.66
Isomap (n=3, neighbors=5, path_method=auto) - K-Means (k=5): Silhouette = 0.34,
Calinski-Harabasz = 940.06
Isomap (n=3, neighbors=5, path_method=auto) - K-Means (k=6): Silhouette = 0.35,
Calinski-Harabasz = 929.39
Isomap (n=3, neighbors=5, path_method=auto) - K-Means (k=7): Silhouette = 0.33,
Calinski-Harabasz = 952.33
Isomap (n=3, neighbors=5, path_method=auto) - K-Means (k=8): Silhouette = 0.33,
Calinski-Harabasz = 911.90
Isomap (n=3, neighbors=10, path_method=auto) - K-Means (k=2): Silhouette = 0.38,
Calinski-Harabasz = 1105.64
```

```
Isomap (n=3, neighbors=10, path_method=auto) - K-Means (k=3): Silhouette = 0.40,
Calinski-Harabasz = 1169.67
Isomap (n=3, neighbors=10, path_method=auto) - K-Means (k=4): Silhouette = 0.35,
Calinski-Harabasz = 1049.80
Isomap (n=3, neighbors=10, path_method=auto) - K-Means (k=5): Silhouette = 0.37,
Calinski-Harabasz = 1060.09
Isomap (n=3, neighbors=10, path_method=auto) - K-Means (k=6): Silhouette = 0.35,
Calinski-Harabasz = 1047.66
Isomap (n=3, neighbors=10, path_method=auto) - K-Means (k=7): Silhouette = 0.34,
Calinski-Harabasz = 1006.61
Isomap (n=3, neighbors=10, path_method=auto) - K-Means (k=8): Silhouette = 0.31,
Calinski-Harabasz = 977.36
Isomap (n=5, neighbors=5, path_method=auto) - K-Means (k=2): Silhouette = 0.33,
Calinski-Harabasz = 857.21
Isomap (n=5, neighbors=5, path_method=auto) - K-Means (k=3): Silhouette = 0.35,
Calinski-Harabasz = 821.29
Isomap (n=5, neighbors=5, path_method=auto) - K-Means (k=4): Silhouette = 0.31,
Calinski-Harabasz = 720.64
Isomap (n=5, neighbors=5, path_method=auto) - K-Means (k=5): Silhouette = 0.31,
Calinski-Harabasz = 669.92
Isomap (n=5, neighbors=5, path_method=auto) - K-Means (k=6): Silhouette = 0.29,
Calinski-Harabasz = 643.73
Isomap (n=5, neighbors=5, path_method=auto) - K-Means (k=7): Silhouette = 0.30,
Calinski-Harabasz = 637.65
Isomap (n=5, neighbors=5, path_method=auto) - K-Means (k=8): Silhouette = 0.28,
Calinski-Harabasz = 609.33
Isomap (n=5, neighbors=10, path_method=auto) - K-Means (k=2): Silhouette = 0.33,
Calinski-Harabasz = 851.69
Isomap (n=5, neighbors=10, path_method=auto) - K-Means (k=3): Silhouette = 0.34,
Calinski-Harabasz = 813.00
Isomap (n=5, neighbors=10, path_method=auto) - K-Means (k=4): Silhouette = 0.30,
Calinski-Harabasz = 702.66
Isomap (n=5, neighbors=10, path_method=auto) - K-Means (k=5): Silhouette = 0.30,
Calinski-Harabasz = 675.54
Isomap (n=5, neighbors=10, path_method=auto) - K-Means (k=6): Silhouette = 0.28,
Calinski-Harabasz = 635.49
Isomap (n=5, neighbors=10, path_method=auto) - K-Means (k=7): Silhouette = 0.29,
Calinski-Harabasz = 620.90
Isomap (n=5, neighbors=10, path_method=auto) - K-Means (k=8): Silhouette = 0.29,
Calinski-Harabasz = 597.40
Isomap (n=7, neighbors=5, path_method=auto) - K-Means (k=2): Silhouette = 0.31,
Calinski-Harabasz = 746.27
Isomap (n=7, neighbors=5, path_method=auto) - K-Means (k=3): Silhouette = 0.32,
Calinski-Harabasz = 687.09
Isomap (n=7, neighbors=5, path_method=auto) - K-Means (k=4): Silhouette = 0.27,
Calinski-Harabasz = 587.12
Isomap (n=7, neighbors=5, path_method=auto) - K-Means (k=5): Silhouette = 0.29,
Calinski-Harabasz = 525.80
```

```
Isomap (n=7, neighbors=5, path_method=auto) - K-Means (k=6): Silhouette = 0.25,
Calinski-Harabasz = 507.14
Isomap (n=7, neighbors=5, path_method=auto) - K-Means (k=7): Silhouette = 0.25,
Calinski-Harabasz = 505.65
Isomap (n=7, neighbors=5, path_method=auto) - K-Means (k=8): Silhouette = 0.25,
Calinski-Harabasz = 476.05
Isomap (n=7, neighbors=10, path_method=auto) - K-Means (k=2): Silhouette = 0.31,
Calinski-Harabasz = 755.69
Isomap (n=7, neighbors=10, path_method=auto) - K-Means (k=3): Silhouette = 0.32,
Calinski-Harabasz = 700.44
Isomap (n=7, neighbors=10, path_method=auto) - K-Means (k=4): Silhouette = 0.26,
Calinski-Harabasz = 595.43
Isomap (n=7, neighbors=10, path_method=auto) - K-Means (k=5): Silhouette = 0.27,
Calinski-Harabasz = 559.69
Isomap (n=7, neighbors=10, path_method=auto) - K-Means (k=6): Silhouette = 0.25,
Calinski-Harabasz = 523.07
Isomap (n=7, neighbors=10, path_method=auto) - K-Means (k=7): Silhouette = 0.26,
Calinski-Harabasz = 507.72
Isomap (n=7, neighbors=10, path_method=auto) - K-Means (k=8): Silhouette = 0.27,
Calinski-Harabasz = 487.24
```

[ ]:
```python
# Mostrar los resultados de Isomap
dataframe_results_isomap = pd.DataFrame(
    [(label, f'{value[2]:.2f}', f'{value[3]:.2f}') for label, value in␣
 ↪kmeans_results_isomap.items() if value[2] is not None and value[3] is not␣
 ↪None],
    columns=['Método', 'Silhouette Score', 'Calinski-Harabasz Score']
)
display(dataframe_results_isomap)
```

```
                                        Método Silhouette Score  \
0    Isomap (n=3, neighbors=5) - K-Means (k=2)              0.37
1    Isomap (n=3, neighbors=5) - K-Means (k=3)              0.40
2    Isomap (n=3, neighbors=5) - K-Means (k=4)              0.34
3    Isomap (n=3, neighbors=5) - K-Means (k=5)              0.34
4    Isomap (n=3, neighbors=5) - K-Means (k=6)              0.32
5    Isomap (n=3, neighbors=5) - K-Means (k=7)              0.33
6    Isomap (n=3, neighbors=5) - K-Means (k=8)              0.32
7   Isomap (n=3, neighbors=10) - K-Means (k=2)              0.38
8   Isomap (n=3, neighbors=10) - K-Means (k=3)              0.40
9   Isomap (n=3, neighbors=10) - K-Means (k=4)              0.35
10  Isomap (n=3, neighbors=10) - K-Means (k=5)              0.37
11  Isomap (n=3, neighbors=10) - K-Means (k=6)              0.35
12  Isomap (n=3, neighbors=10) - K-Means (k=7)              0.34
13  Isomap (n=3, neighbors=10) - K-Means (k=8)              0.31
14   Isomap (n=5, neighbors=5) - K-Means (k=2)              0.33
15   Isomap (n=5, neighbors=5) - K-Means (k=3)              0.35
16   Isomap (n=5, neighbors=5) - K-Means (k=4)              0.31
```

17	Isomap (n=5, neighbors=5) - K-Means (k=5)	0.31
18	Isomap (n=5, neighbors=5) - K-Means (k=6)	0.29
19	Isomap (n=5, neighbors=5) - K-Means (k=7)	0.30
20	Isomap (n=5, neighbors=5) - K-Means (k=8)	0.29
21	Isomap (n=5, neighbors=10) - K-Means (k=2)	0.33
22	Isomap (n=5, neighbors=10) - K-Means (k=3)	0.34
23	Isomap (n=5, neighbors=10) - K-Means (k=4)	0.30
24	Isomap (n=5, neighbors=10) - K-Means (k=5)	0.30
25	Isomap (n=5, neighbors=10) - K-Means (k=6)	0.28
26	Isomap (n=5, neighbors=10) - K-Means (k=7)	0.29
27	Isomap (n=5, neighbors=10) - K-Means (k=8)	0.29
28	Isomap (n=7, neighbors=5) - K-Means (k=2)	0.31
29	Isomap (n=7, neighbors=5) - K-Means (k=3)	0.32
30	Isomap (n=7, neighbors=5) - K-Means (k=4)	0.27
31	Isomap (n=7, neighbors=5) - K-Means (k=5)	0.27
32	Isomap (n=7, neighbors=5) - K-Means (k=6)	0.28
33	Isomap (n=7, neighbors=5) - K-Means (k=7)	0.26
34	Isomap (n=7, neighbors=5) - K-Means (k=8)	0.27
35	Isomap (n=7, neighbors=10) - K-Means (k=2)	0.31
36	Isomap (n=7, neighbors=10) - K-Means (k=3)	0.32
37	Isomap (n=7, neighbors=10) - K-Means (k=4)	0.26
38	Isomap (n=7, neighbors=10) - K-Means (k=5)	0.27
39	Isomap (n=7, neighbors=10) - K-Means (k=6)	0.25
40	Isomap (n=7, neighbors=10) - K-Means (k=7)	0.26
41	Isomap (n=7, neighbors=10) - K-Means (k=8)	0.27

	Calinski-Harabasz Score
0	1103.56
1	1155.56
2	1013.66
3	941.21
4	918.73
5	952.33
6	912.01
7	1105.64
8	1169.67
9	1049.76
10	1060.01
11	1047.66
12	1006.79
13	977.17
14	857.21
15	821.29
16	720.64
17	669.40
18	643.16
19	635.90
20	621.01

```
21                  851.67
22                  813.00
23                  702.66
24                  675.53
25                  635.49
26                  620.90
27                  597.57
28                  746.27
29                  687.09
30                  587.12
31                  538.85
32                  508.96
33                  496.18
34                  493.98
35                  755.69
36                  700.44
37                  595.32
38                  559.69
39                  523.13
40                  490.11
41                  487.81
```

[ ]:
```python
# Aplicar K-Means con MDS
kmeans_results_mds = {}
for label, data_mds in results_mds.items():
    kmeans_results = apply_kmeans_parallel(data_mds)
    for key, value in kmeans_results.items():
        label_clean = label.replace(", path_method=auto", "")
        kmeans_results_mds[f'{label_clean} - {key}'] = value
        print(f'{label_clean} - {key}: Silhouette = {value[2]:.2f},␣
 ↪Calinski-Harabasz = {value[3]:.4f}')
```

```
MDS Métrico (n=3) - K-Means (k=2): Silhouette = 0.30, Calinski-Harabasz =
418.2453
MDS Métrico (n=3) - K-Means (k=3): Silhouette = 0.30, Calinski-Harabasz =
410.7630
MDS Métrico (n=3) - K-Means (k=4): Silhouette = 0.31, Calinski-Harabasz =
422.3985
MDS Métrico (n=3) - K-Means (k=5): Silhouette = 0.32, Calinski-Harabasz =
391.0481
MDS Métrico (n=3) - K-Means (k=6): Silhouette = 0.23, Calinski-Harabasz =
386.5834
MDS Métrico (n=3) - K-Means (k=7): Silhouette = 0.24, Calinski-Harabasz =
389.0834
MDS Métrico (n=3) - K-Means (k=8): Silhouette = 0.24, Calinski-Harabasz =
382.7539
MDS Métrico (n=3) - K-Means (k=9): Silhouette = 0.25, Calinski-Harabasz =
381.0941
```

```
MDS Métrico (n=3) - K-Means (k=10): Silhouette = 0.25, Calinski-Harabasz =
380.3050
MDS Métrico (n=3) - K-Means (k=11): Silhouette = 0.25, Calinski-Harabasz =
383.3550
MDS Métrico (n=3) - K-Means (k=12): Silhouette = 0.24, Calinski-Harabasz =
399.4486
MDS Métrico (n=3) - K-Means (k=13): Silhouette = 0.25, Calinski-Harabasz =
402.2615
MDS Métrico (n=3) - K-Means (k=14): Silhouette = 0.25, Calinski-Harabasz =
408.4705
MDS No Métrico (n=3) - K-Means (k=2): Silhouette = 0.25, Calinski-Harabasz =
549.0039
MDS No Métrico (n=3) - K-Means (k=3): Silhouette = 0.26, Calinski-Harabasz =
576.4341
MDS No Métrico (n=3) - K-Means (k=4): Silhouette = 0.29, Calinski-Harabasz =
639.4523
MDS No Métrico (n=3) - K-Means (k=5): Silhouette = 0.28, Calinski-Harabasz =
651.4788
MDS No Métrico (n=3) - K-Means (k=6): Silhouette = 0.30, Calinski-Harabasz =
696.1136
MDS No Métrico (n=3) - K-Means (k=7): Silhouette = 0.30, Calinski-Harabasz =
712.1314
MDS No Métrico (n=3) - K-Means (k=8): Silhouette = 0.29, Calinski-Harabasz =
725.4777
MDS No Métrico (n=3) - K-Means (k=9): Silhouette = 0.29, Calinski-Harabasz =
690.0067
MDS No Métrico (n=3) - K-Means (k=10): Silhouette = 0.28, Calinski-Harabasz =
661.9997
MDS No Métrico (n=3) - K-Means (k=11): Silhouette = 0.28, Calinski-Harabasz =
643.7919
MDS No Métrico (n=3) - K-Means (k=12): Silhouette = 0.28, Calinski-Harabasz =
630.0304
MDS No Métrico (n=3) - K-Means (k=13): Silhouette = 0.27, Calinski-Harabasz =
618.6957
MDS No Métrico (n=3) - K-Means (k=14): Silhouette = 0.27, Calinski-Harabasz =
612.7554
MDS Métrico (n=5) - K-Means (k=2): Silhouette = 0.26, Calinski-Harabasz =
334.3856
MDS Métrico (n=5) - K-Means (k=3): Silhouette = 0.25, Calinski-Harabasz =
306.6343
MDS Métrico (n=5) - K-Means (k=4): Silhouette = 0.25, Calinski-Harabasz =
305.0451
MDS Métrico (n=5) - K-Means (k=5): Silhouette = 0.26, Calinski-Harabasz =
307.5611
MDS Métrico (n=5) - K-Means (k=6): Silhouette = 0.27, Calinski-Harabasz =
318.0356
MDS Métrico (n=5) - K-Means (k=7): Silhouette = 0.26, Calinski-Harabasz =
309.1949
```

```
MDS Métrico (n=5) - K-Means (k=8): Silhouette = 0.20, Calinski-Harabasz =
303.2761
MDS Métrico (n=5) - K-Means (k=9): Silhouette = 0.20, Calinski-Harabasz =
294.1950
MDS Métrico (n=5) - K-Means (k=10): Silhouette = 0.19, Calinski-Harabasz =
295.3729
MDS Métrico (n=5) - K-Means (k=11): Silhouette = 0.21, Calinski-Harabasz =
304.5388
MDS Métrico (n=5) - K-Means (k=12): Silhouette = 0.19, Calinski-Harabasz =
295.3322
MDS Métrico (n=5) - K-Means (k=13): Silhouette = 0.20, Calinski-Harabasz =
295.6707
MDS Métrico (n=5) - K-Means (k=14): Silhouette = 0.20, Calinski-Harabasz =
293.6794
MDS No Métrico (n=5) - K-Means (k=2): Silhouette = 0.15, Calinski-Harabasz =
295.2600
MDS No Métrico (n=5) - K-Means (k=3): Silhouette = 0.14, Calinski-Harabasz =
268.8787
MDS No Métrico (n=5) - K-Means (k=4): Silhouette = 0.15, Calinski-Harabasz =
261.4705
MDS No Métrico (n=5) - K-Means (k=5): Silhouette = 0.15, Calinski-Harabasz =
252.3063
MDS No Métrico (n=5) - K-Means (k=6): Silhouette = 0.16, Calinski-Harabasz =
248.4216
MDS No Métrico (n=5) - K-Means (k=7): Silhouette = 0.17, Calinski-Harabasz =
241.4909
MDS No Métrico (n=5) - K-Means (k=8): Silhouette = 0.17, Calinski-Harabasz =
239.7204
MDS No Métrico (n=5) - K-Means (k=9): Silhouette = 0.18, Calinski-Harabasz =
239.7975
MDS No Métrico (n=5) - K-Means (k=10): Silhouette = 0.19, Calinski-Harabasz =
237.8071
MDS No Métrico (n=5) - K-Means (k=11): Silhouette = 0.18, Calinski-Harabasz =
231.0783
MDS No Métrico (n=5) - K-Means (k=12): Silhouette = 0.18, Calinski-Harabasz =
224.8842
MDS No Métrico (n=5) - K-Means (k=13): Silhouette = 0.18, Calinski-Harabasz =
220.5155
MDS No Métrico (n=5) - K-Means (k=14): Silhouette = 0.19, Calinski-Harabasz =
216.9331
MDS Métrico (n=7) - K-Means (k=2): Silhouette = 0.26, Calinski-Harabasz =
304.9060
MDS Métrico (n=7) - K-Means (k=3): Silhouette = 0.26, Calinski-Harabasz =
286.6349
MDS Métrico (n=7) - K-Means (k=4): Silhouette = 0.24, Calinski-Harabasz =
295.9206
MDS Métrico (n=7) - K-Means (k=5): Silhouette = 0.25, Calinski-Harabasz =
285.4016
```

```
MDS Métrico (n=7) - K-Means (k=6): Silhouette = 0.25, Calinski-Harabasz =
285.5203
MDS Métrico (n=7) - K-Means (k=7): Silhouette = 0.24, Calinski-Harabasz =
277.6449
MDS Métrico (n=7) - K-Means (k=8): Silhouette = 0.18, Calinski-Harabasz =
283.2217
MDS Métrico (n=7) - K-Means (k=9): Silhouette = 0.17, Calinski-Harabasz =
274.4853
MDS Métrico (n=7) - K-Means (k=10): Silhouette = 0.19, Calinski-Harabasz =
285.6371
MDS Métrico (n=7) - K-Means (k=11): Silhouette = 0.18, Calinski-Harabasz =
281.9441
MDS Métrico (n=7) - K-Means (k=12): Silhouette = 0.17, Calinski-Harabasz =
274.5145
MDS Métrico (n=7) - K-Means (k=13): Silhouette = 0.16, Calinski-Harabasz =
267.3646
MDS Métrico (n=7) - K-Means (k=14): Silhouette = 0.18, Calinski-Harabasz =
258.2467
MDS No Métrico (n=7) - K-Means (k=2): Silhouette = 0.11, Calinski-Harabasz =
207.4957
MDS No Métrico (n=7) - K-Means (k=3): Silhouette = 0.10, Calinski-Harabasz =
181.8920
MDS No Métrico (n=7) - K-Means (k=4): Silhouette = 0.11, Calinski-Harabasz =
168.0652
MDS No Métrico (n=7) - K-Means (k=5): Silhouette = 0.11, Calinski-Harabasz =
160.3915
MDS No Métrico (n=7) - K-Means (k=6): Silhouette = 0.11, Calinski-Harabasz =
150.5397
MDS No Métrico (n=7) - K-Means (k=7): Silhouette = 0.11, Calinski-Harabasz =
146.0856
MDS No Métrico (n=7) - K-Means (k=8): Silhouette = 0.12, Calinski-Harabasz =
140.9663
MDS No Métrico (n=7) - K-Means (k=9): Silhouette = 0.12, Calinski-Harabasz =
134.6673
MDS No Métrico (n=7) - K-Means (k=10): Silhouette = 0.12, Calinski-Harabasz =
129.2413
MDS No Métrico (n=7) - K-Means (k=11): Silhouette = 0.12, Calinski-Harabasz =
126.4339
MDS No Métrico (n=7) - K-Means (k=12): Silhouette = 0.12, Calinski-Harabasz =
121.6855
MDS No Métrico (n=7) - K-Means (k=13): Silhouette = 0.12, Calinski-Harabasz =
119.5026
MDS No Métrico (n=7) - K-Means (k=14): Silhouette = 0.12, Calinski-Harabasz =
115.8280
```

[ ]:
```
dataframe_results_mds = pd.DataFrame(
```

```
    [(label, f'{value[2]:.2f}', f'{value[3]:.2f}') for label, value in␣
 ↪kmeans_results_mds.items() if value[2] is not None and value[3] is not None],
    columns=['Método', 'Silhouette Score', 'Calinski-Harabasz Score']
)
display(dataframe_results_mds)
```

	Método	Silhouette Score	\
0	MDS Métrico (n=3) - K-Means (k=2)	0.30	
1	MDS Métrico (n=3) - K-Means (k=3)	0.30	
2	MDS Métrico (n=3) - K-Means (k=4)	0.31	
3	MDS Métrico (n=3) - K-Means (k=5)	0.26	
4	MDS Métrico (n=3) - K-Means (k=6)	0.23	
5	MDS Métrico (n=3) - K-Means (k=7)	0.27	
6	MDS Métrico (n=3) - K-Means (k=8)	0.24	
7	MDS No Métrico (n=3) - K-Means (k=2)	0.25	
8	MDS No Métrico (n=3) - K-Means (k=3)	0.26	
9	MDS No Métrico (n=3) - K-Means (k=4)	0.29	
10	MDS No Métrico (n=3) - K-Means (k=5)	0.28	
11	MDS No Métrico (n=3) - K-Means (k=6)	0.30	
12	MDS No Métrico (n=3) - K-Means (k=7)	0.30	
13	MDS No Métrico (n=3) - K-Means (k=8)	0.29	
14	MDS Métrico (n=5) - K-Means (k=2)	0.26	
15	MDS Métrico (n=5) - K-Means (k=3)	0.25	
16	MDS Métrico (n=5) - K-Means (k=4)	0.25	
17	MDS Métrico (n=5) - K-Means (k=5)	0.26	
18	MDS Métrico (n=5) - K-Means (k=6)	0.27	
19	MDS Métrico (n=5) - K-Means (k=7)	0.26	
20	MDS Métrico (n=5) - K-Means (k=8)	0.20	
21	MDS No Métrico (n=5) - K-Means (k=2)	0.15	
22	MDS No Métrico (n=5) - K-Means (k=3)	0.14	
23	MDS No Métrico (n=5) - K-Means (k=4)	0.15	
24	MDS No Métrico (n=5) - K-Means (k=5)	0.15	
25	MDS No Métrico (n=5) - K-Means (k=6)	0.16	
26	MDS No Métrico (n=5) - K-Means (k=7)	0.16	
27	MDS No Métrico (n=5) - K-Means (k=8)	0.17	
28	MDS Métrico (n=7) - K-Means (k=2)	0.26	
29	MDS Métrico (n=7) - K-Means (k=3)	0.24	
30	MDS Métrico (n=7) - K-Means (k=4)	0.24	
31	MDS Métrico (n=7) - K-Means (k=5)	0.25	
32	MDS Métrico (n=7) - K-Means (k=6)	0.25	
33	MDS Métrico (n=7) - K-Means (k=7)	0.24	
34	MDS Métrico (n=7) - K-Means (k=8)	0.18	
35	MDS No Métrico (n=7) - K-Means (k=2)	0.11	
36	MDS No Métrico (n=7) - K-Means (k=3)	0.10	
37	MDS No Métrico (n=7) - K-Means (k=4)	0.11	
38	MDS No Métrico (n=7) - K-Means (k=5)	0.11	
39	MDS No Métrico (n=7) - K-Means (k=6)	0.11	

40	MDS No Métrico (n=7) - K-Means (k=7)	0.11
41	MDS No Métrico (n=7) - K-Means (k=8)	0.12

	Calinski-Harabasz Score
0	418.25
1	410.76
2	422.40
3	374.86
4	386.58
5	385.75
6	382.75
7	549.01
8	576.43
9	639.45
10	651.48
11	696.11
12	712.13
13	725.48
14	334.39
15	306.63
16	305.05
17	307.56
18	318.04
19	309.19
20	303.28
21	295.22
22	268.88
23	261.47
24	251.77
25	248.33
26	241.16
27	239.72
28	304.91
29	272.66
30	280.29
31	285.40
32	271.15
33	277.64
34	282.72
35	207.50
36	181.89
37	168.07
38	158.78
39	150.43
40	145.97
41	139.34

5.5 Aplicar Clustering Jerárquico

[ ]:
```python
def apply_hierarchical_parallel(data, k_range=range(2, 9)):
    """Aplica Clustering Jerárquico al conjunto de datos en paralelo y calcula␣
 ↪métricas de evaluación"""
    def hierarchical_clustering(k):
        hierarchical = AgglomerativeClustering(n_clusters=k)
        labels = hierarchical.fit_predict(data)
        silhouette = silhouette_score(data, labels)
        ch_score = calinski_harabasz_score(data, labels)
        return f'Hierarchical (k={k})', labels, silhouette, ch_score

    results = Parallel(n_jobs=-1)(delayed(hierarchical_clustering)(k) for k in␣
 ↪k_range)
    return {key: (labels, silhouette, ch_score) for key, labels, silhouette,␣
 ↪ch_score in results}

print("\nAplicando Clustering Jerárquico al espacio original de los datos...")
hierarchical_results_original = apply_hierarchical_parallel(data_scaled)
for key, value in hierarchical_results_original.items():
    print(f'{key}: Silhouette = {value[1]:.4f}, Calinski-Harabasz = {value[2]:.
 ↪4f}')
```

```
Aplicando Clustering Jerárquico al espacio original de los datos…
Hierarchical (k=2): Silhouette = 0.1951, Calinski-Harabasz = 241.3505
Hierarchical (k=3): Silhouette = 0.1938, Calinski-Harabasz = 228.2533
Hierarchical (k=4): Silhouette = 0.2021, Calinski-Harabasz = 242.4904
Hierarchical (k=5): Silhouette = 0.2067, Calinski-Harabasz = 269.0623
Hierarchical (k=6): Silhouette = 0.1675, Calinski-Harabasz = 287.9275
Hierarchical (k=7): Silhouette = 0.1856, Calinski-Harabasz = 317.2540
Hierarchical (k=8): Silhouette = 0.1872, Calinski-Harabasz = 328.2945
```

5.6 Aplicar GMM

[ ]:
```python
def apply_gmm_parallel(data, k_range=range(2, 9)):
    """Aplica GMM al conjunto de datos en paralelo y calcula métricas de␣
 ↪evaluación"""
    def gmm_clustering(k):
        gmm = GaussianMixture(n_components=k, random_state=42)
        labels = gmm.fit_predict(data)
        silhouette = silhouette_score(data, labels)
        ch_score = calinski_harabasz_score(data, labels)
        return f'GMM (k={k})', labels, silhouette, ch_score

    results = Parallel(n_jobs=-1)(delayed(gmm_clustering)(k) for k in k_range)
```

```
    return {key: (labels, silhouette, ch_score) for key, labels, silhouette,␣
 ↪ch_score in results}

print("\nAplicando GMM al espacio original de los datos...")
gmm_results_original = apply_gmm_parallel(data_scaled)
for key, value in gmm_results_original.items():
    print(f'{key}: Silhouette = {value[1]:.4f}, Calinski-Harabasz = {value[2]:.
 ↪4f}')
```

```
Aplicando GMM al espacio original de los datos…
GMM (k=2): Silhouette = 0.7071, Calinski-Harabasz = 76.1813
GMM (k=3): Silhouette = 0.5647, Calinski-Harabasz = 92.9668
GMM (k=4): Silhouette = 0.4245, Calinski-Harabasz = 72.3082
GMM (k=5): Silhouette = 0.1942, Calinski-Harabasz = 97.3379
GMM (k=6): Silhouette = 0.1161, Calinski-Harabasz = 123.8879
GMM (k=7): Silhouette = 0.1511, Calinski-Harabasz = 204.9765
GMM (k=8): Silhouette = 0.1608, Calinski-Harabasz = 277.8577
```