

Projeto de Disciplina de Validação de modelos de clusterização [24E4_3]

Aluna: Rachel Reuters

PARTE 1- INFRAESTRUTURA

```
In [58]: import platform
import sys

print("Python version:", sys.version)
print("Platform:", platform.platform())
print("Architecture:", platform.architecture())
print("Processor:", platform.processor())
print("System:", platform.system())
print("Machine:", platform.machine())
print("Release:", platform.release())
print("Node:", platform.node())
```

Python version: 3.9.20 (main, Oct 3 2024, 07:38:01) [MSC v.1929 64 bit (AMD64)]
Platform: Windows-10-10.0.22621-SP0
Architecture: ('64bit', 'WindowsPE')
Processor: Intel64 Family 6 Model 158 Stepping 10, GenuineIntel
System: Windows
Machine: AMD64
Release: 10
Node: DESKTOP-4U6C9N4

```
In [59]: import subprocess

def get_conda_info():
    result = subprocess.run(['conda', 'info'], stdout=subprocess.PIPE)
    print(result.stdout.decode('utf-8'))

print("Conda Info:")
get_conda_info()
```

Conda Info:

```
active environment : py39
active env location : C:\Users\belch\anaconda3\envs\py39
shell level : 1
user config file : C:\Users\belch\.condarc
populated config files : C:\Users\belch\.condarc
    conda version : 24.5.0
    conda-build version : 24.5.1
    python version : 3.12.4.final.0
        solver : libmamba (default)
    virtual packages : __archspec=1=skylake
                        __conda=24.5.0=0
                        __cuda=12.6=0
                        __win=0=0
base environment : C:\Users\belch\anaconda3 (writable)
conda av data dir : C:\Users\belch\anaconda3\etc\conda
conda av metadata url : None
channel URLs : https://repo.anaconda.com/pkgs/main/win-64
                https://repo.anaconda.com/pkgs/main/noarch
                https://repo.anaconda.com/pkgs/r/win-64
                https://repo.anaconda.com/pkgs/r/noarch
                https://repo.anaconda.com/pkgs/msys2/win-64
                https://repo.anaconda.com/pkgs/msys2/noarch
                https://conda.anaconda.org/conda-forge/win-64
                https://conda.anaconda.org/conda-forge/noarch
package cache : C:\Users\belch\anaconda3\pkgs
                C:\Users\belch\.conda\pkgs
                C:\Users\belch\AppData\Local\conda\conda\pkgs
envs directories : C:\Users\belch\anaconda3\envs
                  C:\Users\belch\.conda\envs
                  C:\Users\belch\AppData\Local\conda\conda\envs
platform : win-64
user-agent : conda/24.5.0 requests/2.32.2 CPython/3.12.4 Windows/11 Windows/10.0.22631 solver/libmamba conda-libmamba-solver/24.1.0 libmambapy/1.5.8 aau/0.4.4 c/. s/. e/.
administrator : False
netrc file : None
offline mode : False
```

```
In [246]: with open('requirements.txt', 'r') as file:
    requirements = file.read()

print(requirements)
```

```
matplotlib==3.9.2
numpy==1.26.4
pandas==2.2.1
scikit-learn==1.5.2
scipy==1.13.1
seaborn==0.13.2
scikit-learn-extra==0.3.0
data_science_utils==1.8.0
hdbscan==0.8.40
tslearn==0.6.3
```

PARTE 2 - ESCOLHA DA BASE DE DADOS

```
In [ ]: #2.1. Escolha uma base de dados para realizar o trabalho. Essa base será usada em u
# Escolhi utilizar o dataset do Spotify https://www.kaggle.com/datasets/sanjanchaud
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

dataset_original = pd.read_csv('cleaned_dataset.csv', sep=',', decimal='.')

print("Numero de Musicas do dataset: ", dataset_original.shape[0])

#2.2. Escreva a justificativa para a escolha de dados, dando sua motivação e objetivo
# Gostaria de fazer um sistema de recomendação de músicas. Onde eu possa colocar um
# e gostaria que o programa me falasse quais músicas são parecidas.
# Se não for possível utilizar o conjunto de dados com os algoritmos propostos , pe
# uma análise encima dos agrupamentos gerados.

#2.3. Mostre através de gráficos a faixa dinâmica das variáveis que serão usadas na
# Analise os resultados mostrados. O que deve ser feito com os dados antes da etapa

#Irei remover algumas colunas que não são relevantes para a análise sendo feita
dataset_original.drop(['Licensed','official_video','Stream', 'most_playedon', 'Chan

print("Tipos de dados:")
print(dataset_original.dtypes)

#2.4. Realize o pré-processamento adequado dos dados. Descreva os passos necessário

#Para a análise vamos utilizar apenas os dados numéricos :

dataset_numerico = dataset_original.drop(['Artist', 'Track', 'Album', 'Title' ], axis=1

print("Quantidade de colunas para análise:")
print(dataset_numerico.shape[1])

#Depois verificar se tem duplicados
print("Dados duplicados:")
print(dataset_original.duplicated().sum())
```

```
#Verificar se tem nulos
print("Dados faltantes:")
print(dataset_numerico.isna().sum())

#Como sao apenas 2 linhas e nao acredito que colocar um valor sem saber a origem co
dataset_numerico = dataset_numerico.dropna(subset=['EnergyLiveness'], axis=0 )

dataset_original = dataset_original.dropna(subset=[ 'EnergyLiveness'], axis=0 )

print("Novo Numero de Musicas do dataset: ", dataset_numerico.shape[0])

plt.figure(figsize=(40, 14))

for index, value in enumerate(dataset_numerico.columns):
    plt.subplot(3, 5,index+1)
    sns.histplot(dataset_numerico[value], kde=True)
```

Numero de Musicas do dataset: 20594

Tipos de dados:

```
Artist          object
Track          object
Album          object
Danceability   float64
Energy         float64
Loudness       float64
Speechiness    float64
Acousticness   float64
Instrumentalness float64
Liveness        float64
Valence         float64
Tempo           float64
Duration_min   float64
Title           object
Views           float64
Likes           float64
Comments        float64
EnergyLiveness float64
dtype: object
```

Quantidade de colunas para analise:

14

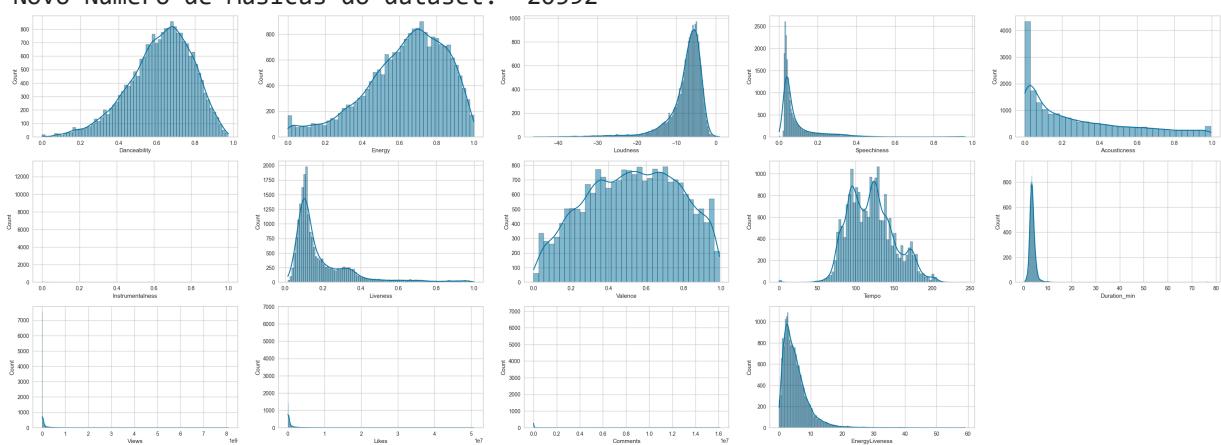
Dados duplicados:

0

Dados faltantes:

```
Danceability    0
Energy          0
Loudness        0
Speechiness     0
Acousticness    0
Instrumentalness 0
Liveness        0
Valence         0
Tempo           0
Duration_min    0
Views           0
Likes           0
Comments        0
EnergyLiveness  2
dtype: int64
```

Novo Numero de Musicas do dataset: 20592



In [153]:

```
#Alem disso podemos verificar que os dados estao com escalas diferentes. Precisamos
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import FunctionTransformer
from sklearn.preprocessing import QuantileTransformer, StandardScaler, MinMaxScaler
from scipy import stats

def calcular_indice_medio_correlacao(data):
    correlation_matrix = data.corr()
    correlation_values = correlation_matrix.values
    correlation_values_no_diag = correlation_values[np.triu_indices_from(correlation_matrix)]
    return np.mean(correlation_values_no_diag)

transformacoes = ['Log', 'Sqrt', 'QuantileTransformer', 'MinMax', 'Standard']
resultados = []

# Loop sobre cada transformação
for trans in transformacoes:
    if trans == 'Log':
        data_trans = np.log1p(dataset_numerico)
    elif trans == 'Sqrt':
        data_trans = np.sqrt(dataset_numerico)
    elif trans == 'QuantileTransformer':
        pt = QuantileTransformer(output_distribution='uniform')
        data_trans = pd.DataFrame(pt.fit_transform(dataset_numerico), columns=dataset_numerico.columns)
    elif trans == 'MinMax':
        scaler = MinMaxScaler()
        data_trans = pd.DataFrame(scaler.fit_transform(dataset_numerico), columns=dataset_numerico.columns)
    elif trans == 'Standard':
        scaler = StandardScaler()
        data_trans = pd.DataFrame(scaler.fit_transform(dataset_numerico), columns=dataset_numerico.columns)

    indice_medio_correlacao = calcular_indice_medio_correlacao(data_trans)
    resultados.append({"Metodo Transformacao": trans, "Correlacao Media": indice_medio_correlacao})

resultados_comparados = pd.DataFrame(resultados)

print(resultados_comparados)

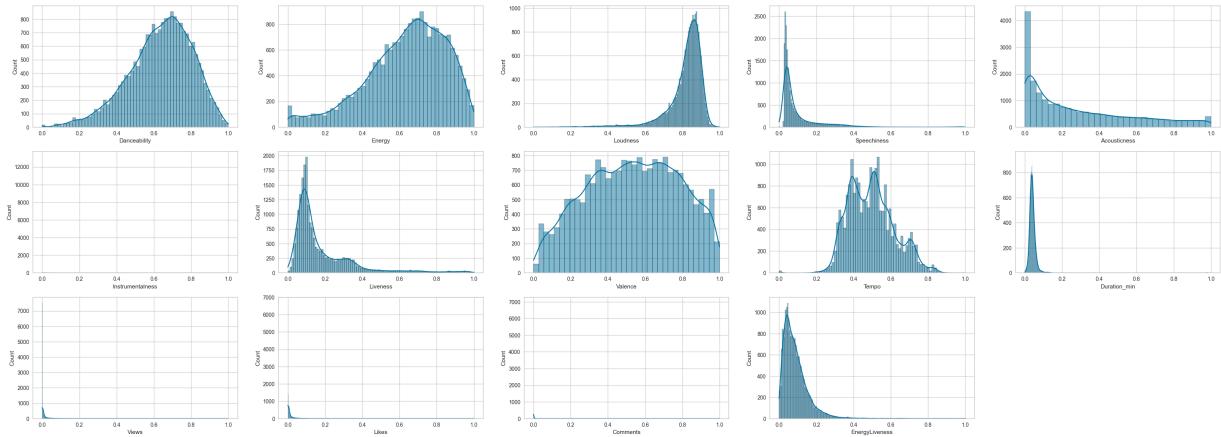
scaler = MinMaxScaler()
data_trans = pd.DataFrame(scaler.fit_transform(dataset_numerico), columns=dataset_numerico.columns)
dataset_normalizado = data_trans
plt.figure(figsize=(40, 14))

for index, value in enumerate(dataset_normalizado.columns):
    plt.subplot(3, 5, index+1)
    sns.histplot(dataset_normalizado[value], kde=True)
    plt.xlabel(dataset_numerico.columns[index])
```

```

c:\Users\belch\anaconda3\envs\py39\lib\site-packages\pandas\core\internals\blocks.p
y:393: RuntimeWarning: invalid value encountered in log1p
    result = func(self.values, **kwargs)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\pandas\core\internals\blocks.p
y:393: RuntimeWarning: invalid value encountered in sqrt
    result = func(self.values, **kwargs)
    Metodo Transformacao Correlacao Media
0              Log      0.026319
1            Sqrt      0.022626
2  QuantileTransformer      0.058447
3        MinMax      0.029524
4       Standard      0.029524

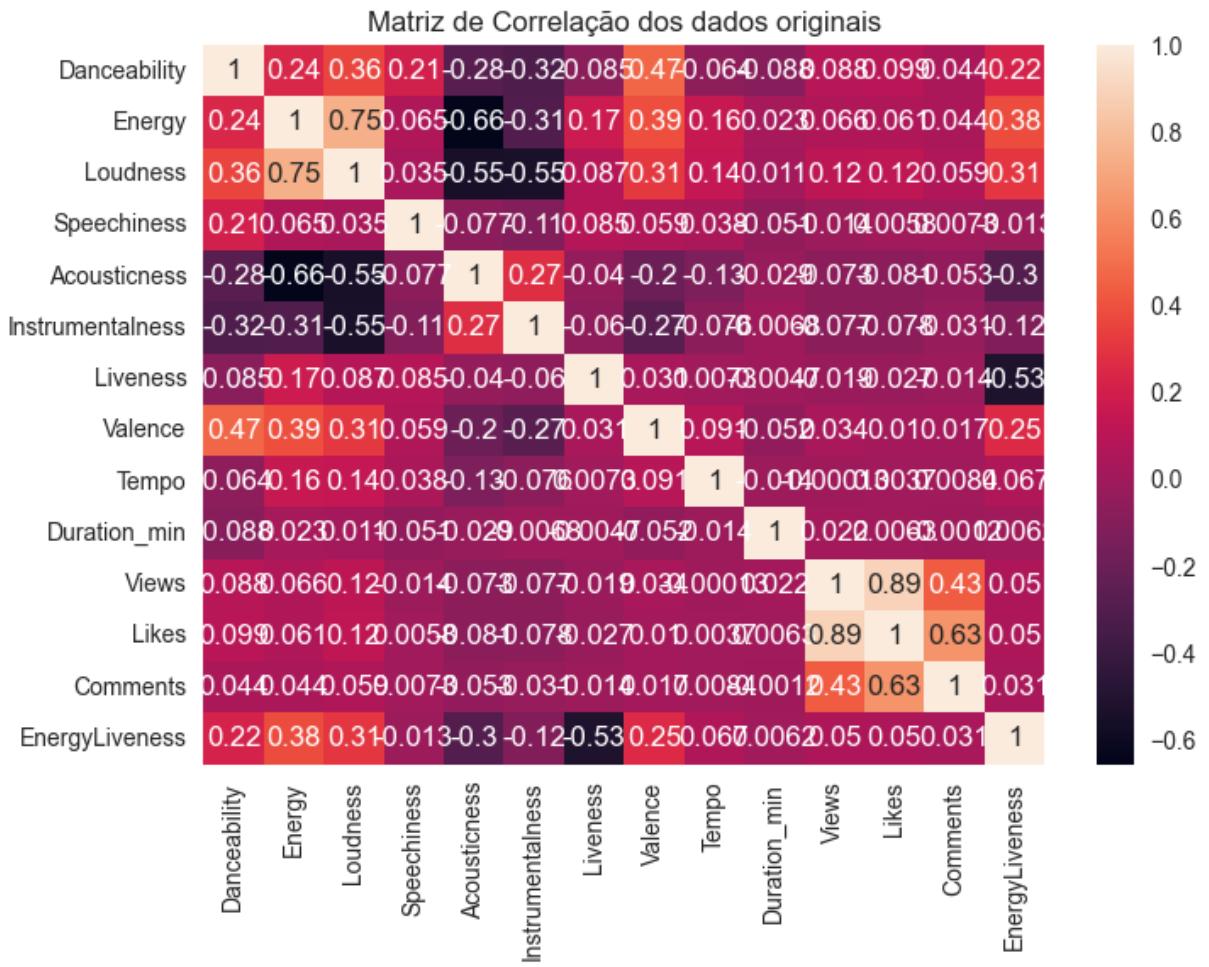
```



```

In [145... correlation_matrix = dataset_numerico.corr()
sns.heatmap(correlation_matrix, annot=True)
plt.title('Matriz de Correlação dos dados originais')
plt.show()
#Podemos observar que na matriz de correlacao, existem muitas correlacoes negativas
#Sugere que as propriedades sao independentes ou a relacao nao e linear.

```



PARTE 3 - CLUSTERIZAÇÃO

```
In [ ]: #3.1. Realizar o agrupamento dos dados, escolhendo o número ótimo de clusters.
# Para tal, use o índice de silhueta e as técnicas:
#K-Médias
#DBScan

# Atualmente, a lib Yellowbrick fornece dois visualizadores para avaliar mecanismo
# o K-Means, e ajuda a descobrir um K ideal na métrica de clustering:

# O KELbowVisualizer visualiza os clusters de acordo com uma função de pontuação, p
# SilhouetteVisualizer visualiza as pontuações de silhueta de cada cluster em um ún

# Por padrão, a métrica do parâmetro de pontuação é definida como distorção, que ca
# de cada ponto até seu centro atribuído. No entanto, duas outras métricas também p
# silhueta e calinski_harabaz. A pontuação da silhueta é o coeficiente médio da sil
# enquanto a pontuação calinski_harabaz calcula a proporção de dispersão entre e de

from sklearn.cluster import KMeans

from yellowbrick.cluster import KElbowVisualizer

model = KMeans(random_state=22)
```

```

fig, axes = plt.subplots(1, 3, figsize=(12, 6))

visualizer = KElbowVisualizer(model, ax=axes[0], k=(2,20), timings=False)

visualizer.fit(dataset_normalizado)
visualizer.finalize()

visualizer2 = KElbowVisualizer(model, ax=axes[1], k=(2,20), metric='silhouette', tim

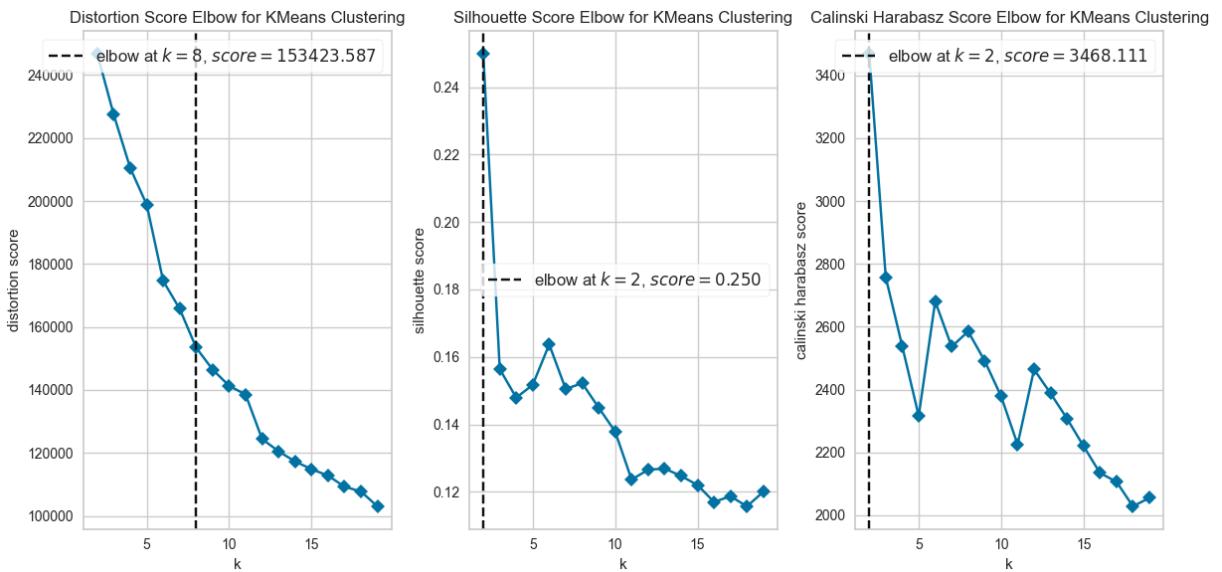
visualizer2.fit(dataset_normalizado)
visualizer2.finalize()

visualizer3 = KElbowVisualizer(model, ax=axes[2], k=(2,20), metric='calinski_harabas

visualizer3.fit(dataset_normalizado)
visualizer3.finalize()

plt.tight_layout()
plt.show()

```



```

In [154]: from sklearn.metrics import silhouette_score
# Define a range of cluster numbers to evaluate
range_n_clusters = range(2, 10)

# List to store silhouette scores
silhouette_scores = []

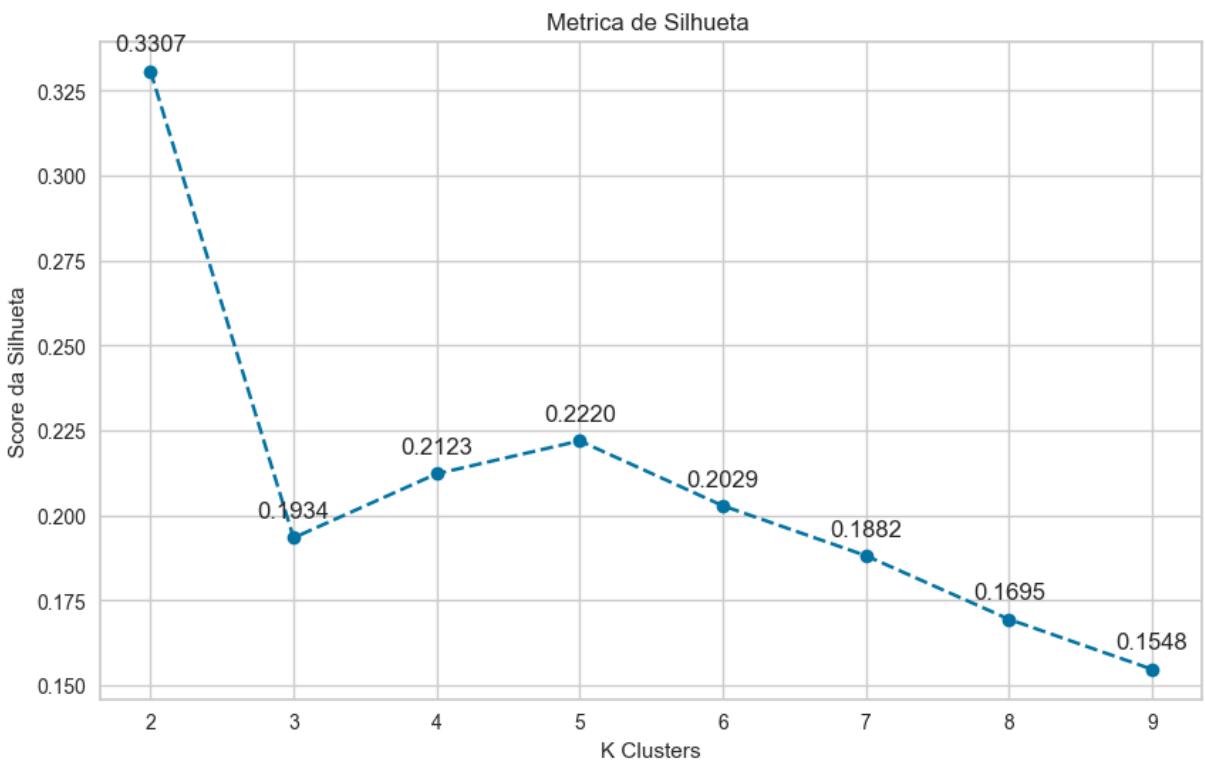
for n_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=n_clusters, random_state=22)
    cluster_labels = kmeans.fit_predict(dataset_normalizado)
    silhouette_avg = silhouette_score(dataset_normalizado, cluster_labels)
    silhouette_scores.append(silhouette_avg)

plt.figure(figsize=(10, 6))
plt.plot(range_n_clusters, silhouette_scores, marker='o', linestyle='--', color='b'

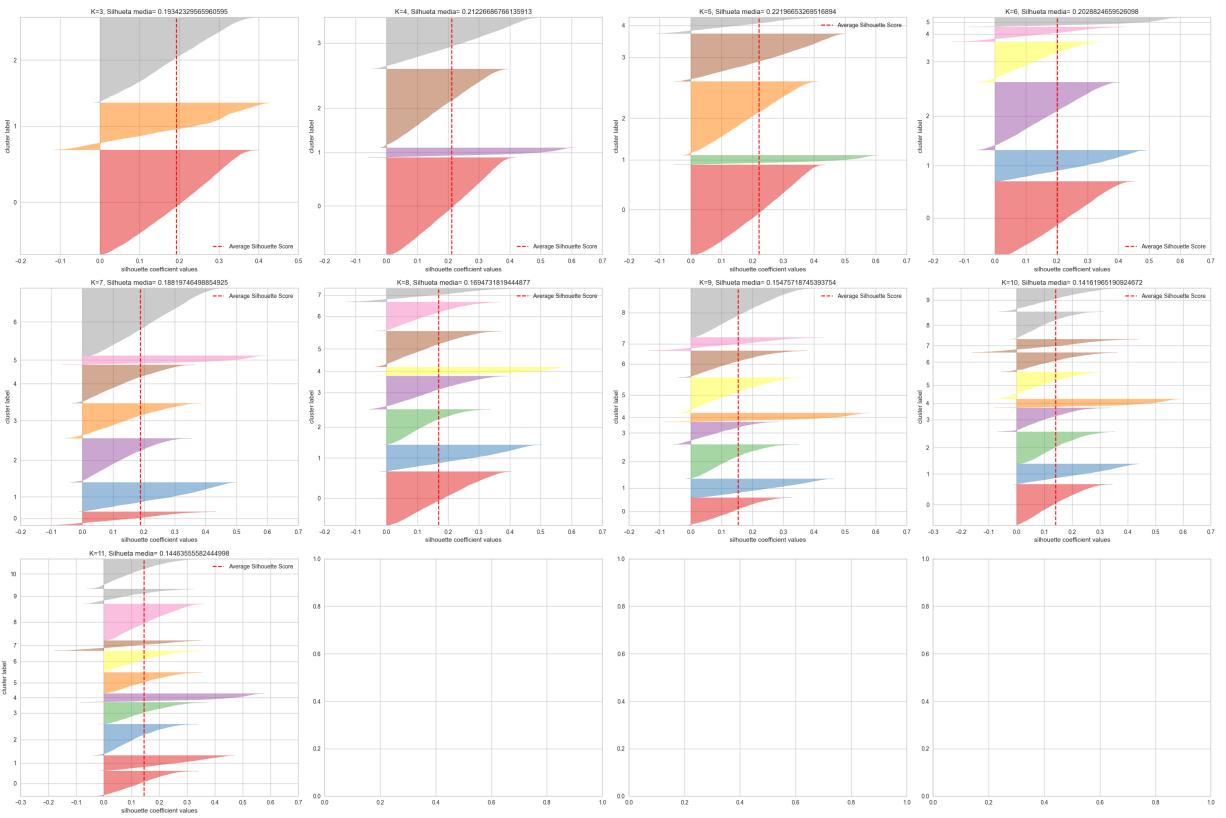
```

```
for i, txt in enumerate(silhouette_scores):
    plt.annotate(f'{txt:.4f}', (range_n_clusters[i], silhouette_scores[i]), textcoo

plt.title('Metrica de Silhueta ')
plt.xlabel('K Clusters')
plt.ylabel('Score da Silhueta')
plt.xticks(range_n_clusters)
plt.grid(True)
plt.show()
```



```
In [155...]:  
from yellowbrick.cluster import SilhouetteVisualizer  
from sklearn import metrics  
  
model = KMeans()  
fig, axes = plt.subplots(3, 4, figsize=(30, 20))  
axes = axes.flatten()  
for i, k in enumerate(range(3, 12)):  
    model = KMeans(n_clusters=k, random_state=22)  
    visualizer = SilhouetteVisualizer(model, ax=axes[i])  
    visualizer.fit(dataset_normalizado)  
    visualizer.finalize()  
    silhouette_samples = metrics.silhouette_samples(dataset_normalizado, model.labels_)  
    axes[i].set_title(f"K={k}, Silhueta media= {silhouette_samples.mean()}")  
  
plt.tight_layout()  
plt.show()
```



In []: #3.2 Com os resultados em mão, descreva o processo de mensuração do índice de silhu
Mostre o gráfico e justifique o número de clusters escolhidos.

```
#0 indice de silhueta serve para avaliar o quao bem cada ponto se encaixa no seu cl
# Esse indice 'e calculado da seguinte forma: Para cada ponto no dataset 'e calcula
# distancia media entre o ponto e todos os outros pontos no mesmo cluster
# distancia média entre o ponto e todos os pontos no cluster mais próximo
#0 indice de silhueta 'e uma formula encima desses dois parametros.
#Para o indice de silhueta global de todo dataset, basta calcular a media entre tod
```

```
#Fazendo um panorama entre todos as metricas utilizadas, o melhor 'e agrupar em 5 c
# Porem podemos observar no plot de silhueta que os dados nao estao muito bem dife
# e poucas musicas em 2 grupos. Tanto 'e que o indice de silhueta deu apenas 0.22
```

In []: #Apenas para fazer um teste, escolhi uma musica e quero entender quais 5 musicas sa

```
linha_dataset_musica = dataset_original[dataset_original['Track'] == "What I've Don
indice_musica_escolhida = dataset_original.index[dataset_original['Track'] == "What
print("Indice da musica")
print(indice_musica_escolhida)
```

```
kmeans = KMeans(n_clusters=5, random_state=22)
model = kmeans.fit(dataset_normalizado)

distancias = kmeans.transform(dataset_normalizado)

cluster_labels_dbSCAN_dbSCAN_dbSCAN_dbSCAN_dbSCAN_dbSCAN_dbSCAN_dbSCAN_dbSCAN_dbSCA
```

```

dataset_normalizado_com_cluster = dataset_normalizado.copy()
dataset_normalizado_com_cluster['cluster'] = cluster_labels

cluster_da_minha_musica = cluster_labels[indice_musica_escolhida]

dataset_do_cluster_da_minha_musica = dataset_normalizado[dataset_normalizado_com_cl

distancias = pd.DataFrame(distancias)

distancias["Track"] = dataset_original['Track']
distancias["Artist"] = dataset_original['Artist']

distancias["cluster"] = cluster_labels
distancias = distancias.sort_values(by=['cluster', cluster_da_minha_musica])

mais_proximos_centro = distancias.head()

Musicas_Selecionadas = mais_proximos_centro[['Track', 'Artist']].values.tolist()
print("Musicas mais parecidas de acordo com a clusterizacao ")
print(Musicas_Selecionadas)

```

Indice da musica

62

Musicas mais parecidas de acordo com a clusterizacao

[['Lovers Rock', 'TV Girl'], ['VALENTINO', '24kGoldn'], ['React', 'The Pussycat Dolls'], ['Love Generation', 'Bob Sinclair'], ['Mama Told Me Not To Come', 'Tom Jones']]

In [156]: #Considerando o K=5, que apresentou o melhor resultado, significa que as musicas podem ser usadas para representar cada grupo...

```

kmeans = KMeans(n_clusters=5, random_state=22)
model= kmeans.fit(dataset_normalizado)
y_kmeans = kmeans.predict(dataset_normalizado)

distances = kmeans.transform(dataset_normalizado)
distancias = pd.DataFrame(distances)

resultado_k_means = pd.DataFrame(y_kmeans)
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
resultado_k_means = resultado_k_means.rename(columns={0: 'Categoria'})
resultado_k_means['Musica'] = dataset_original['Track'].to_list()
resultado_k_means['Artista'] = dataset_original['Artist'].to_list()

resultado_k_means = resultado_k_means.sort_values(by='Categoria')

mais_proximos = []
mais_proximos = pd.DataFrame(mais_proximos)
mais_proximos['Categoria']=list(set(resultado_k_means['Categoria']))

linha =0
for center in range(len(kmeans.cluster_centers_)):
    distancia_coluna = distancias[linha].tolist()
    _index = np.argmin(distancia_coluna)

```

```

    mais_proximos.at[linha, 'Distancia'] = distancias[linha][_index]
    for coluna_original in dataset_original.columns:
        mais_proximos.at[linha, coluna_original] = dataset_original[coluna_original]

    linha+=1

print("Pontos mais próximos dos centroides:\n", mais_proximos)

```

Pontos mais próximos dos centroides:

	Categoria	Distancia	Artist	Track	Album
0	0	0.070099	A\$AP Rocky	Sundress	Sundress
1	1	0.167428	Sir Neville Marriner	Il barbiere di Siviglia: Overture (Sinfonia)	Rossini: Complete Overtures
2	2	0.105202	Usher	Love in This Club (feat. Young Jeezy)	Here I Stand
3	3	0.088591	Afgan	Terima Kasih Cinta	Confession No.1
4	4	0.145163	Rionegro & Solimões	Vida de Cão	Vida de Cão

	Danceability	Energy	Loudness	Speechiness	Acousticness	\
0	0.721	0.707	-6.364	0.0595	0.1810	
1	0.322	0.140	-20.747	0.0489	0.9060	
2	0.573	0.712	-5.976	0.0732	0.0572	
3	0.586	0.439	-9.096	0.0296	0.6860	
4	0.622	0.798	-3.705	0.0508	0.2730	

	Instrumentalness	Liveness	Valence	Tempo	Duration_min	\
0	0.000004	0.143	0.743	125.005	2.636767	
1	0.759000	0.125	0.285	92.122	7.083333	
2	0.000000	0.167	0.346	140.012	4.328667	
3	0.000028	0.103	0.428	120.032	4.190917	
4	0.000000	0.635	0.627	133.107	3.710517	

	Title	Views	Likes	\
0	A\$AP Rocky - Sundress (Official Video)	38120377.0	553624.0	
1	Rossini: Il barbiere di Siviglia (overture) - ...	8420613.0	37592.0	
2	Usher - Love in This Club (Official Music Vide...)	220352721.0	798143.0	
3	Afgan - Terima Kasih Cinta (Official Music Video)	15329041.0	52084.0	
4	Rionegro & Solimões - Peão Apaixonado (Ao Vivo)	7993850.0	47718.0	

	Comments	Energy	Liveness
0	13638.0	4.944056	
1	1245.0	1.120000	
2	25149.0	4.263473	
3	3683.0	4.262136	
4	746.0	1.256693	

In [157]: #Para a analise do DBscan , vou realizar uma iteracao com alguns valores possiveis

```
# Para os possiveis valores de eps, vou realizar uma analise com KNN usando o valor
# Encontrar o cotovelo do plot
```

```

from sklearn.neighbors import NearestNeighbors

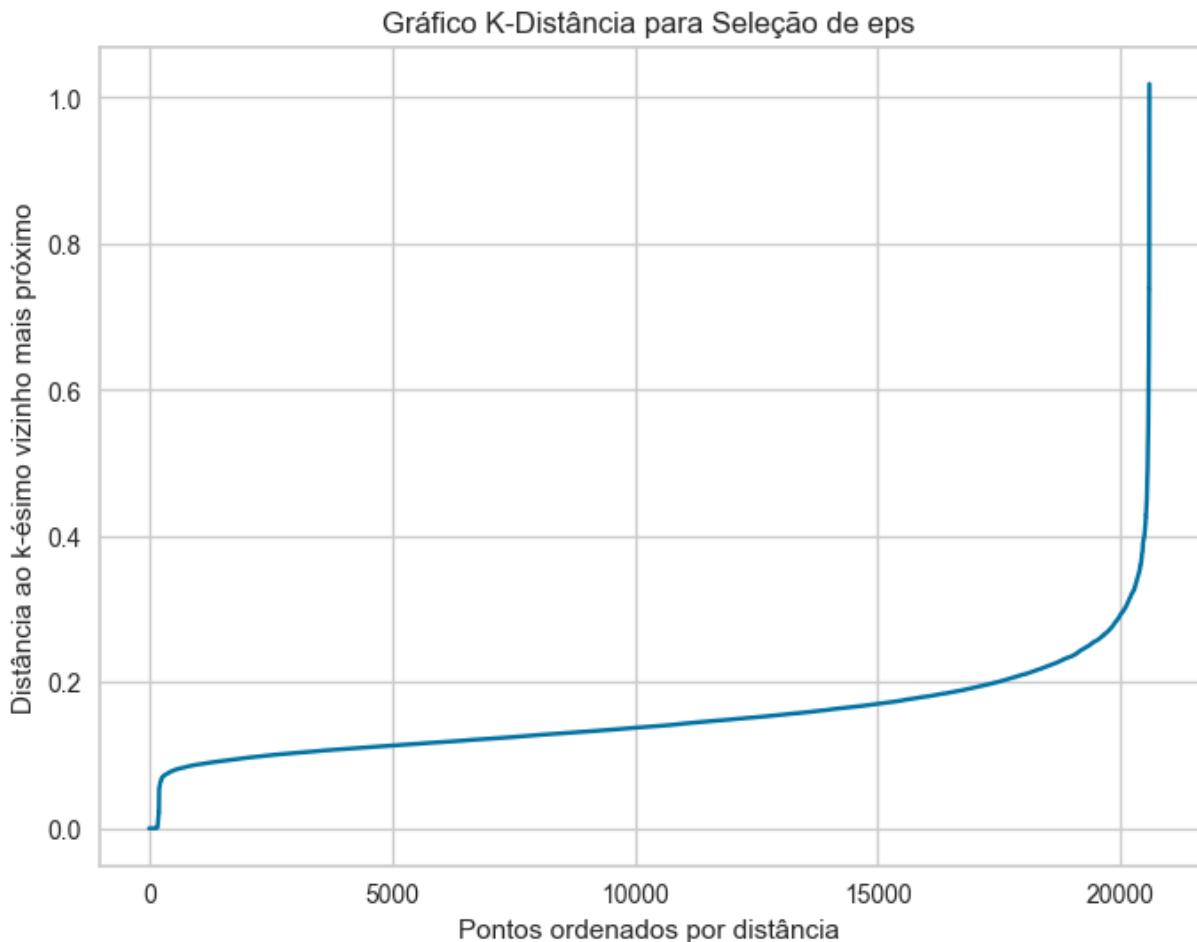
k = 5

# Calcular as distâncias k-ésimas mais próximas
neighbors = NearestNeighbors(n_neighbors=k)
neighbors_fit = neighbors.fit(dataset_normalizado)
distances, indices = neighbors_fit.kneighbors(dataset_normalizado)

# Ordenar as distâncias
distances = np.sort(distances[:, k-1])

# Plotar o gráfico K-Distância
plt.figure(figsize=(8, 6))
plt.plot(distances)
plt.xlabel('Pontos ordenados por distância')
plt.ylabel('Distância ao k-ésimo vizinho mais próximo')
plt.title('Gráfico K-Distância para Seleção de eps')
plt.show()

```



In []:

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
import matplotlib.cm as cm

#Pelo plot acima o valor possivel para eps esta proximo dos 0.2 onde acontece um a

```

```

#Realizando um loop como se fosse um grid search para encontrar os melhores valores
#Para o range do eps foram escolhidos valores próximos ao elbow como mostrado anteriormente

from sklearn.model_selection import ParameterGrid

param_grid = {
    'min_samples': range(5, 30, 2),
    'metric': ["euclidean", "jaccard"],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
}

final_result = []

for params in ParameterGrid(param_grid):
    eps= 0.2
    min_samples=params['min_samples']
    metric = params['metric']
    algorithm= params['algorithm']
    try:
        dbscan = DBSCAN(eps=eps, min_samples=min_samples, metric=metric, algorithm=algorithm)
        labels = dbscan.fit_predict(dataset_normalizado)

        if len(set(labels)) > 1:
            silhouette_avg = silhouette_score(dataset_normalizado, labels)
            n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
            final_result.append({"n_clusters": n_clusters, "silhueta": silhouette_avg})
        else:
            silhouette_avg = 0
            final_result.append({"n_clusters": 0, "silhueta": -1, "eps": eps, "min_samples": min_samples})
    except Exception as ex:
        final_result.append({"n_clusters": 0, "silhueta": -1, "eps": eps, "min_samples": min_samples})

final_result_df = pd.DataFrame(final_result)

plt.figure(figsize=(10, 6))
plt.plot(final_result_df['n_clusters'], final_result_df['silhueta'], marker='o', linestyle='solid')

for i, row in final_result_df.iterrows():
    plt.annotate(f'{row["silhueta"]:.4f}', (row['n_clusters'], row['silhueta']), textcoords="offset points", xytext=(-10, 10), ha='right', va='bottom')

plt.title('Metrica de Silhueta ')
plt.xlabel('K Clusters')
plt.ylabel('Score da Silhueta')
plt.xticks(final_result_df['n_clusters'])
plt.grid(True)
plt.show()

```

```
{'n_clusters': 33, 'silhueta': -0.28746858732142033, 'eps': 0.2, 'min_samples': 5, 'metric': 'euclidean', 'algorithm': 'auto'}  
{'n_clusters': 14, 'silhueta': 0.04133839369954966, 'eps': 0.2, 'min_samples': 7, 'metric': 'euclidean', 'algorithm': 'auto'}  
{'n_clusters': 11, 'silhueta': -0.09913628964313172, 'eps': 0.2, 'min_samples': 9, 'metric': 'euclidean', 'algorithm': 'auto'}  
{'n_clusters': 3, 'silhueta': 0.11242121102409232, 'eps': 0.2, 'min_samples': 11, 'metric': 'euclidean', 'algorithm': 'auto'}  
{'n_clusters': 6, 'silhueta': 0.1330269693910383, 'eps': 0.2, 'min_samples': 13, 'metric': 'euclidean', 'algorithm': 'auto'}  
{'n_clusters': 5, 'silhueta': 0.12504827473950672, 'eps': 0.2, 'min_samples': 15, 'metric': 'euclidean', 'algorithm': 'auto'}  
{'n_clusters': 7, 'silhueta': 0.03115474339846546, 'eps': 0.2, 'min_samples': 17, 'metric': 'euclidean', 'algorithm': 'auto'}  
{'n_clusters': 2, 'silhueta': 0.22654098717745025, 'eps': 0.2, 'min_samples': 19, 'metric': 'euclidean', 'algorithm': 'auto'}  
{'n_clusters': 2, 'silhueta': 0.22282268907423544, 'eps': 0.2, 'min_samples': 21, 'metric': 'euclidean', 'algorithm': 'auto'}  
{'n_clusters': 2, 'silhueta': 0.21897202567844815, 'eps': 0.2, 'min_samples': 23, 'metric': 'euclidean', 'algorithm': 'auto'}  
{'n_clusters': 2, 'silhueta': 0.2146899055763944, 'eps': 0.2, 'min_samples': 25, 'metric': 'euclidean', 'algorithm': 'auto'}  
{'n_clusters': 2, 'silhueta': 0.21110474782554256, 'eps': 0.2, 'min_samples': 27, 'metric': 'euclidean', 'algorithm': 'auto'}  
{'n_clusters': 2, 'silhueta': 0.20733056562784555, 'eps': 0.2, 'min_samples': 29, 'metric': 'euclidean', 'algorithm': 'auto'}  
{'n_clusters': 2, 'silhueta': 0.39718939732928416, 'eps': 0.2, 'min_samples': 5, 'metric': 'jaccard', 'algorithm': 'auto'}  
{'n_clusters': 2, 'silhueta': 0.39718939732928416, 'eps': 0.2, 'min_samples': 7, 'metric': 'jaccard', 'algorithm': 'auto'}  
{'n_clusters': 2, 'silhueta': 0.39718939732928416, 'eps': 0.2, 'min_samples': 9, 'metric': 'jaccard', 'algorithm': 'auto'}  
{'n_clusters': 2, 'silhueta': 0.39718939732928416, 'eps': 0.2, 'min_samples': 11, 'metric': 'jaccard', 'algorithm': 'auto'}  
{'n_clusters': 2, 'silhueta': 0.39718939732928416, 'eps': 0.2, 'min_samples': 13, 'metric': 'jaccard', 'algorithm': 'auto'}  
{'n_clusters': 2, 'silhueta': 0.39718939732928416, 'eps': 0.2, 'min_samples': 15, 'metric': 'jaccard', 'algorithm': 'auto'}  
{'n_clusters': 2, 'silhueta': 0.39718939732928416, 'eps': 0.2, 'min_samples': 17, 'metric': 'jaccard', 'algorithm': 'auto'}  
{'n_clusters': 1, 'silhueta': 0.5240089210282395, 'eps': 0.2, 'min_samples': 19, 'metric': 'jaccard', 'algorithm': 'auto'}  
{'n_clusters': 1, 'silhueta': 0.5240089210282395, 'eps': 0.2, 'min_samples': 21, 'metric': 'jaccard', 'algorithm': 'auto'}  
{'n_clusters': 1, 'silhueta': 0.5240089210282395, 'eps': 0.2, 'min_samples': 23, 'metric': 'jaccard', 'algorithm': 'auto'}  
{'n_clusters': 1, 'silhueta': 0.5240089210282395, 'eps': 0.2, 'min_samples': 25, 'metric': 'jaccard', 'algorithm': 'auto'}  
{'n_clusters': 1, 'silhueta': 0.5240089210282395, 'eps': 0.2, 'min_samples': 27, 'metric': 'jaccard', 'algorithm': 'auto'}  
{'n_clusters': 1, 'silhueta': 0.5240089210282395, 'eps': 0.2, 'min_samples': 29, 'metric': 'jaccard', 'algorithm': 'auto'}  
{'n_clusters': 33, 'silhueta': -0.28746858732142033, 'eps': 0.2, 'min_samples': 5, 'metric': 'euclidean', 'algorithm': 'ball_tree'}  
{'n_clusters': 14, 'silhueta': 0.04133839369954966, 'eps': 0.2, 'min_samples': 7, 'metric': 'euclidean', 'algorithm': 'ball_tree'}
```

```
{'n_clusters': 11, 'silhueta': -0.09913628964313172, 'eps': 0.2, 'min_samples': 9, 'metric': 'euclidean', 'algorithm': 'ball_tree'}
{'n_clusters': 3, 'silhueta': 0.11242121102409232, 'eps': 0.2, 'min_samples': 11, 'metric': 'euclidean', 'algorithm': 'ball_tree'}
{'n_clusters': 6, 'silhueta': 0.1330269693910383, 'eps': 0.2, 'min_samples': 13, 'metric': 'euclidean', 'algorithm': 'ball_tree'}
{'n_clusters': 5, 'silhueta': 0.12504827473950672, 'eps': 0.2, 'min_samples': 15, 'metric': 'euclidean', 'algorithm': 'ball_tree'}
{'n_clusters': 7, 'silhueta': 0.03115474339846546, 'eps': 0.2, 'min_samples': 17, 'metric': 'euclidean', 'algorithm': 'ball_tree'}
{'n_clusters': 2, 'silhueta': 0.22654098717745025, 'eps': 0.2, 'min_samples': 19, 'metric': 'euclidean', 'algorithm': 'ball_tree'}
{'n_clusters': 2, 'silhueta': 0.22282268907423544, 'eps': 0.2, 'min_samples': 21, 'metric': 'euclidean', 'algorithm': 'ball_tree'}
{'n_clusters': 2, 'silhueta': 0.21897202567844815, 'eps': 0.2, 'min_samples': 23, 'metric': 'euclidean', 'algorithm': 'ball_tree'}
{'n_clusters': 2, 'silhueta': 0.2146899055763944, 'eps': 0.2, 'min_samples': 25, 'metric': 'euclidean', 'algorithm': 'ball_tree'}
{'n_clusters': 2, 'silhueta': 0.21110474782554256, 'eps': 0.2, 'min_samples': 27, 'metric': 'euclidean', 'algorithm': 'ball_tree'}
{'n_clusters': 2, 'silhueta': 0.20733056562784555, 'eps': 0.2, 'min_samples': 29, 'metric': 'euclidean', 'algorithm': 'ball_tree'}
{'n_clusters': 2, 'silhueta': 0.39718939732928416, 'eps': 0.2, 'min_samples': 5, 'metric': 'jaccard', 'algorithm': 'ball_tree'}
{'n_clusters': 2, 'silhueta': 0.39718939732928416, 'eps': 0.2, 'min_samples': 7, 'metric': 'jaccard', 'algorithm': 'ball_tree'}
{'n_clusters': 2, 'silhueta': 0.39718939732928416, 'eps': 0.2, 'min_samples': 9, 'metric': 'jaccard', 'algorithm': 'ball_tree'}
{'n_clusters': 2, 'silhueta': 0.39718939732928416, 'eps': 0.2, 'min_samples': 11, 'metric': 'jaccard', 'algorithm': 'ball_tree'}
{'n_clusters': 2, 'silhueta': 0.39718939732928416, 'eps': 0.2, 'min_samples': 13, 'metric': 'jaccard', 'algorithm': 'ball_tree'}
{'n_clusters': 2, 'silhueta': 0.39718939732928416, 'eps': 0.2, 'min_samples': 15, 'metric': 'jaccard', 'algorithm': 'ball_tree'}
{'n_clusters': 2, 'silhueta': 0.39718939732928416, 'eps': 0.2, 'min_samples': 17, 'metric': 'jaccard', 'algorithm': 'ball_tree'}
{'n_clusters': 1, 'silhueta': 0.5240089210282395, 'eps': 0.2, 'min_samples': 19, 'metric': 'jaccard', 'algorithm': 'ball_tree'}
{'n_clusters': 1, 'silhueta': 0.5240089210282395, 'eps': 0.2, 'min_samples': 21, 'metric': 'jaccard', 'algorithm': 'ball_tree'}
{'n_clusters': 1, 'silhueta': 0.5240089210282395, 'eps': 0.2, 'min_samples': 23, 'metric': 'jaccard', 'algorithm': 'ball_tree'}
{'n_clusters': 1, 'silhueta': 0.5240089210282395, 'eps': 0.2, 'min_samples': 25, 'metric': 'jaccard', 'algorithm': 'ball_tree'}
{'n_clusters': 1, 'silhueta': 0.5240089210282395, 'eps': 0.2, 'min_samples': 27, 'metric': 'jaccard', 'algorithm': 'ball_tree'}
{'n_clusters': 1, 'silhueta': 0.5240089210282395, 'eps': 0.2, 'min_samples': 29, 'metric': 'jaccard', 'algorithm': 'ball_tree'}
{'n_clusters': 33, 'silhueta': -0.28746858732142033, 'eps': 0.2, 'min_samples': 5, 'metric': 'euclidean', 'algorithm': 'kd_tree'}
{'n_clusters': 14, 'silhueta': 0.04133839369954966, 'eps': 0.2, 'min_samples': 7, 'metric': 'euclidean', 'algorithm': 'kd_tree'}
{'n_clusters': 11, 'silhueta': -0.09913628964313172, 'eps': 0.2, 'min_samples': 9, 'metric': 'euclidean', 'algorithm': 'kd_tree'}
{'n_clusters': 3, 'silhueta': 0.11242121102409232, 'eps': 0.2, 'min_samples': 11, 'metric': 'euclidean', 'algorithm': 'kd_tree'}
```

```
{'n_clusters': 6, 'silhueta': 0.1330269693910383, 'eps': 0.2, 'min_samples': 13, 'metric': 'euclidean', 'algorithm': 'kd_tree'}
{'n_clusters': 5, 'silhueta': 0.12504827473950672, 'eps': 0.2, 'min_samples': 15, 'metric': 'euclidean', 'algorithm': 'kd_tree'}
{'n_clusters': 7, 'silhueta': 0.03115474339846546, 'eps': 0.2, 'min_samples': 17, 'metric': 'euclidean', 'algorithm': 'kd_tree'}
{'n_clusters': 2, 'silhueta': 0.22654098717745025, 'eps': 0.2, 'min_samples': 19, 'metric': 'euclidean', 'algorithm': 'kd_tree'}
{'n_clusters': 2, 'silhueta': 0.22282268907423544, 'eps': 0.2, 'min_samples': 21, 'metric': 'euclidean', 'algorithm': 'kd_tree'}
{'n_clusters': 2, 'silhueta': 0.21897202567844815, 'eps': 0.2, 'min_samples': 23, 'metric': 'euclidean', 'algorithm': 'kd_tree'}
{'n_clusters': 2, 'silhueta': 0.2146899055763944, 'eps': 0.2, 'min_samples': 25, 'metric': 'euclidean', 'algorithm': 'kd_tree'}
{'n_clusters': 2, 'silhueta': 0.21110474782554256, 'eps': 0.2, 'min_samples': 27, 'metric': 'euclidean', 'algorithm': 'kd_tree'}
{'n_clusters': 2, 'silhueta': 0.20733056562784555, 'eps': 0.2, 'min_samples': 29, 'metric': 'euclidean', 'algorithm': 'kd_tree'}
{'n_clusters': 33, 'silhueta': -0.28746858732142033, 'eps': 0.2, 'min_samples': 5, 'metric': 'euclidean', 'algorithm': 'brute'}
{'n_clusters': 14, 'silhueta': 0.04133839369954966, 'eps': 0.2, 'min_samples': 7, 'metric': 'euclidean', 'algorithm': 'brute'}
{'n_clusters': 11, 'silhueta': -0.09913628964313172, 'eps': 0.2, 'min_samples': 9, 'metric': 'euclidean', 'algorithm': 'brute'}
{'n_clusters': 3, 'silhueta': 0.11242121102409232, 'eps': 0.2, 'min_samples': 11, 'metric': 'euclidean', 'algorithm': 'brute'}
{'n_clusters': 6, 'silhueta': 0.1330269693910383, 'eps': 0.2, 'min_samples': 13, 'metric': 'euclidean', 'algorithm': 'brute'}
{'n_clusters': 5, 'silhueta': 0.12504827473950672, 'eps': 0.2, 'min_samples': 15, 'metric': 'euclidean', 'algorithm': 'brute'}
{'n_clusters': 7, 'silhueta': 0.03115474339846546, 'eps': 0.2, 'min_samples': 17, 'metric': 'euclidean', 'algorithm': 'brute'}
{'n_clusters': 2, 'silhueta': 0.22654098717745025, 'eps': 0.2, 'min_samples': 19, 'metric': 'euclidean', 'algorithm': 'brute'}
{'n_clusters': 2, 'silhueta': 0.22282268907423544, 'eps': 0.2, 'min_samples': 21, 'metric': 'euclidean', 'algorithm': 'brute'}
{'n_clusters': 2, 'silhueta': 0.21897202567844815, 'eps': 0.2, 'min_samples': 23, 'metric': 'euclidean', 'algorithm': 'brute'}
{'n_clusters': 2, 'silhueta': 0.2146899055763944, 'eps': 0.2, 'min_samples': 25, 'metric': 'euclidean', 'algorithm': 'brute'}
{'n_clusters': 2, 'silhueta': 0.21110474782554256, 'eps': 0.2, 'min_samples': 27, 'metric': 'euclidean', 'algorithm': 'brute'}
{'n_clusters': 2, 'silhueta': 0.20733056562784555, 'eps': 0.2, 'min_samples': 29, 'metric': 'euclidean', 'algorithm': 'brute'}
```

```
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
```

```
{'n_clusters': 2, 'silhueta': 0.39718939732928416, 'eps': 0.2, 'min_samples': 5, 'metric': 'jaccard', 'algorithm': 'brute'}
```

```
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
```

```
{'n_clusters': 2, 'silhueta': 0.39718939732928416, 'eps': 0.2, 'min_samples': 7, 'metric': 'jaccard', 'algorithm': 'brute'}
```

```
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
```

```
{'n_clusters': 2, 'silhueta': 0.39718939732928416, 'eps': 0.2, 'min_samples': 9, 'metric': 'jaccard', 'algorithm': 'brute'}
```

```
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
```

```
{'n_clusters': 2, 'silhueta': 0.39718939732928416, 'eps': 0.2, 'min_samples': 11, 'metric': 'jaccard', 'algorithm': 'brute'}
```



```
{'n_clusters': 1, 'silhueta': 0.5240089210282395, 'eps': 0.2, 'min_samples': 19, 'metric': 'jaccard', 'algorithm': 'brute'}
```

```
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
{'n_clusters': 1, 'silhueta': 0.5240089210282395, 'eps': 0.2, 'min_samples': 21, 'metric': 'jaccard', 'algorithm': 'brute'}
```

```
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
{'n_clusters': 1, 'silhueta': 0.5240089210282395, 'eps': 0.2, 'min_samples': 23, 'metric': 'jaccard', 'algorithm': 'brute'}
```

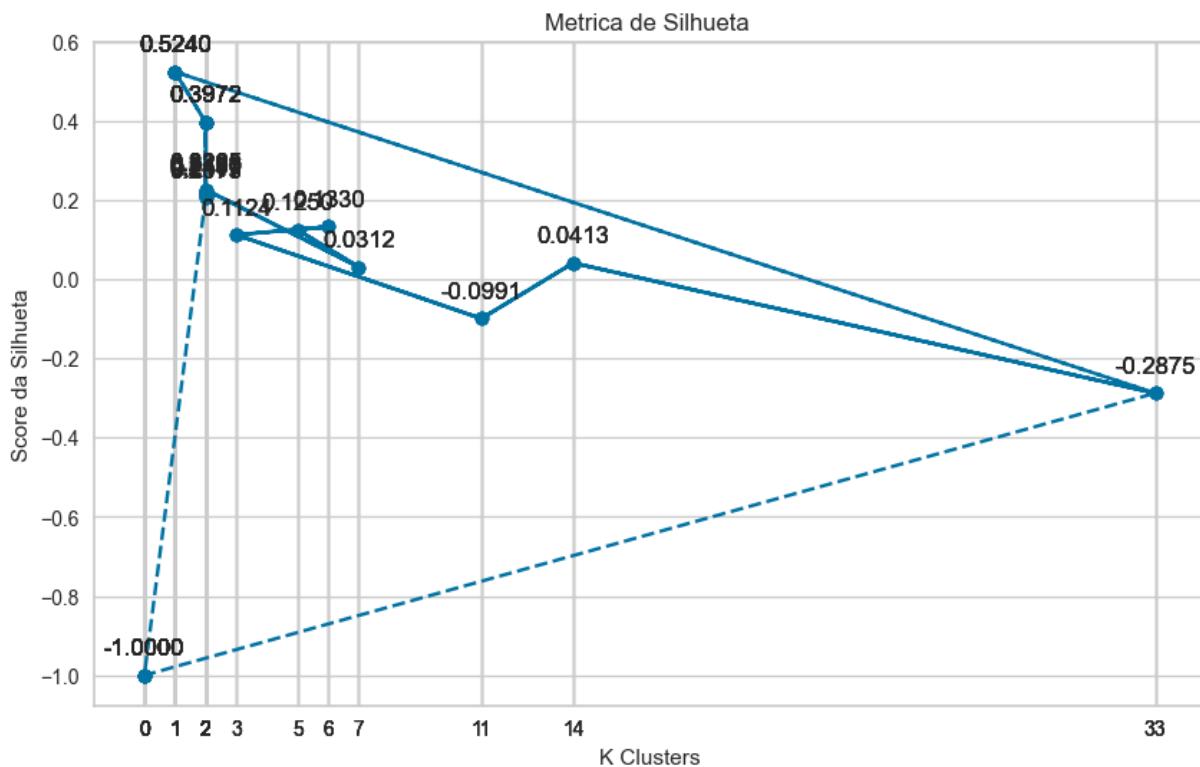
```
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
{'n_clusters': 1, 'silhueta': 0.5240089210282395, 'eps': 0.2, 'min_samples': 25, 'metric': 'jaccard', 'algorithm': 'brute'}
```

```

c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
{'n_clusters': 1, 'silhueta': 0.5240089210282395, 'eps': 0.2, 'min_samples': 27, 'metric': 'jaccard', 'algorithm': 'brute'}

c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
c:\Users\belch\anaconda3\envs\py39\lib\site-packages\sklearn\metrics\pairwise.py:236
1: DataConversionWarning: Data was converted to boolean for metric jaccard
    warnings.warn(msg, DataConversionWarning)
{'n_clusters': 1, 'silhueta': 0.5240089210282395, 'eps': 0.2, 'min_samples': 29, 'metric': 'jaccard', 'algorithm': 'brute'}

```



```
In [159]: final_result_df = final_result_df.sort_values(by=['silhueta'], ascending=False)

#Nao faz muito sentido dividir em apenas 2 grupos.
```

```
filtered_data = final_result_df.loc[final_result_df['n_clusters'] > 2]

filtered_data.head(1)['min_samples'].values[0]
filtered_data.head(1)['metric'].values[0]

filtered_data
```

Out[159]:

	n_clusters	silhueta	eps	min_samples	metric	algorithm
4	6	0.133027	0.2	13	euclidean	auto
30	6	0.133027	0.2	13	euclidean	ball_tree
82	6	0.133027	0.2	13	euclidean	brute
56	6	0.133027	0.2	13	euclidean	kd_tree
83	5	0.125048	0.2	15	euclidean	brute
31	5	0.125048	0.2	15	euclidean	ball_tree
5	5	0.125048	0.2	15	euclidean	auto
57	5	0.125048	0.2	15	euclidean	kd_tree
81	3	0.112421	0.2	11	euclidean	brute
55	3	0.112421	0.2	11	euclidean	kd_tree
29	3	0.112421	0.2	11	euclidean	ball_tree
3	3	0.112421	0.2	11	euclidean	auto
79	14	0.041338	0.2	7	euclidean	brute
27	14	0.041338	0.2	7	euclidean	ball_tree
53	14	0.041338	0.2	7	euclidean	kd_tree
1	14	0.041338	0.2	7	euclidean	auto
6	7	0.031155	0.2	17	euclidean	auto
58	7	0.031155	0.2	17	euclidean	kd_tree
84	7	0.031155	0.2	17	euclidean	brute
32	7	0.031155	0.2	17	euclidean	ball_tree
2	11	-0.099136	0.2	9	euclidean	auto
80	11	-0.099136	0.2	9	euclidean	brute
54	11	-0.099136	0.2	9	euclidean	kd_tree
28	11	-0.099136	0.2	9	euclidean	ball_tree
0	33	-0.287469	0.2	5	euclidean	auto
78	33	-0.287469	0.2	5	euclidean	brute
26	33	-0.287469	0.2	5	euclidean	ball_tree
52	33	-0.287469	0.2	5	euclidean	kd_tree

In [247...]

```
from sklearn.metrics import silhouette_samples, silhouette_score
```

#Um resultado que aparentemente tem uma boa metrica 'e' o de 6 clusters.
Porem o resultado com 5 clusters foi bem proximo. Para fins comparativos com o Km

```

# Valor de eps: 0.2 e min_samples: 15

# Compute DBSCAN
dbSCAN = DBSCAN(0.2, min_samples=15, algorithm = "ball_tree")
labels_dbSCAN = dbSCAN.fit_predict(dataset_normalizado)
core_samples_mask = np.zeros_like(labels_dbSCAN, dtype=bool)
core_samples_mask[dbSCAN.core_sample_indices_] = True
realClusterNum=len(set(labels_dbSCAN)) - (1 if -1 in labels_dbSCAN else 0)
clusterNum = len(set(labels_dbSCAN))

silhouette_avg = silhouette_score(dataset_normalizado, labels_dbSCAN)
sample_silhouette_values = silhouette_samples(dataset_normalizado, labels_dbSCAN)

print(f"Silhouette Score médio: {silhouette_avg}")

fig, ax1 = plt.subplots(1, 1, figsize=(10, 6))

# Set the range for the silhouette plot
ax1.set_xlim([-0.1, 1])
ax1.set_ylim([0, len(dataset_normalizado) + (realClusterNum + 1) * 10])

y_lower = 10
for i in range(realClusterNum):
    ith_cluster_silhouette_values = sample_silhouette_values[labels_dbSCAN == i]
    ith_cluster_silhouette_values.sort()

    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i

    color = cm.nipy_spectral(float(i) / realClusterNum)
    ax1.fill_betweenx(np.arange(y_lower, y_upper), 0, ith_cluster_silhouette_values,
                      facecolor=color, edgecolor=color, alpha=0.7)

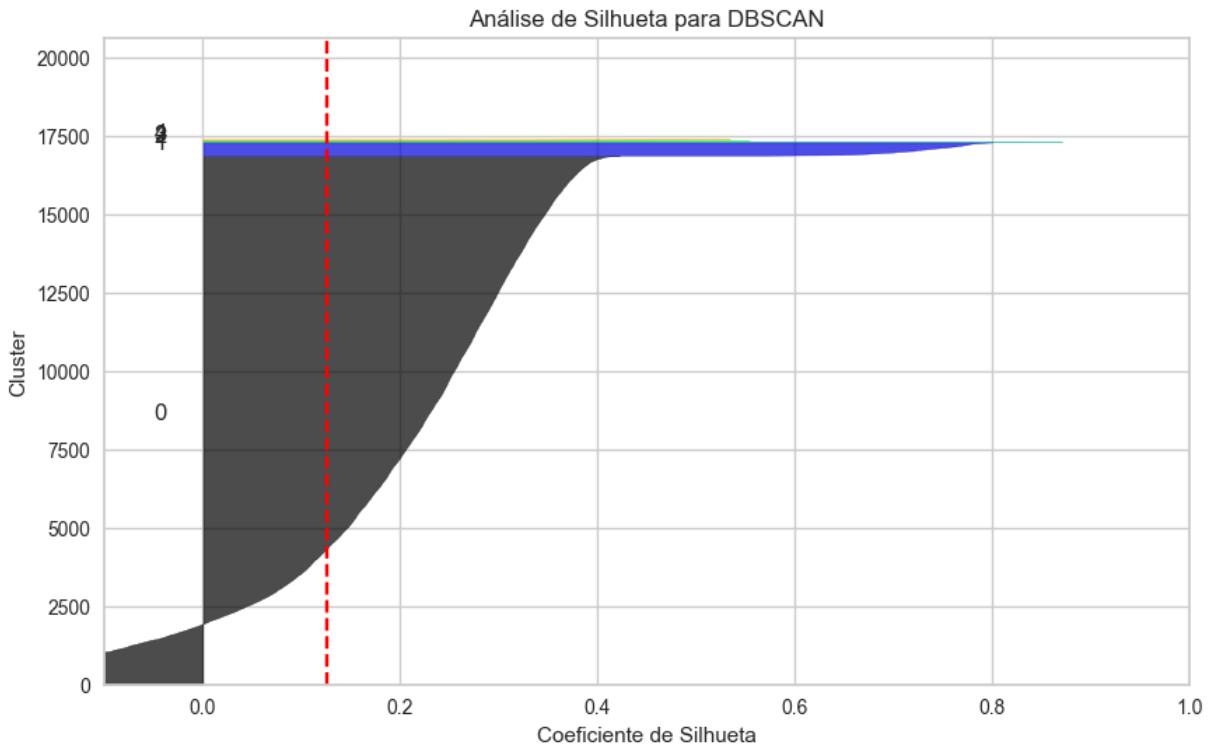
    ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
    y_lower = y_upper + 10

ax1.set_title("Análise de Silhueta para DBSCAN")
ax1.set_xlabel("Coeficiente de Silhueta")
ax1.set_ylabel("Cluster")
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

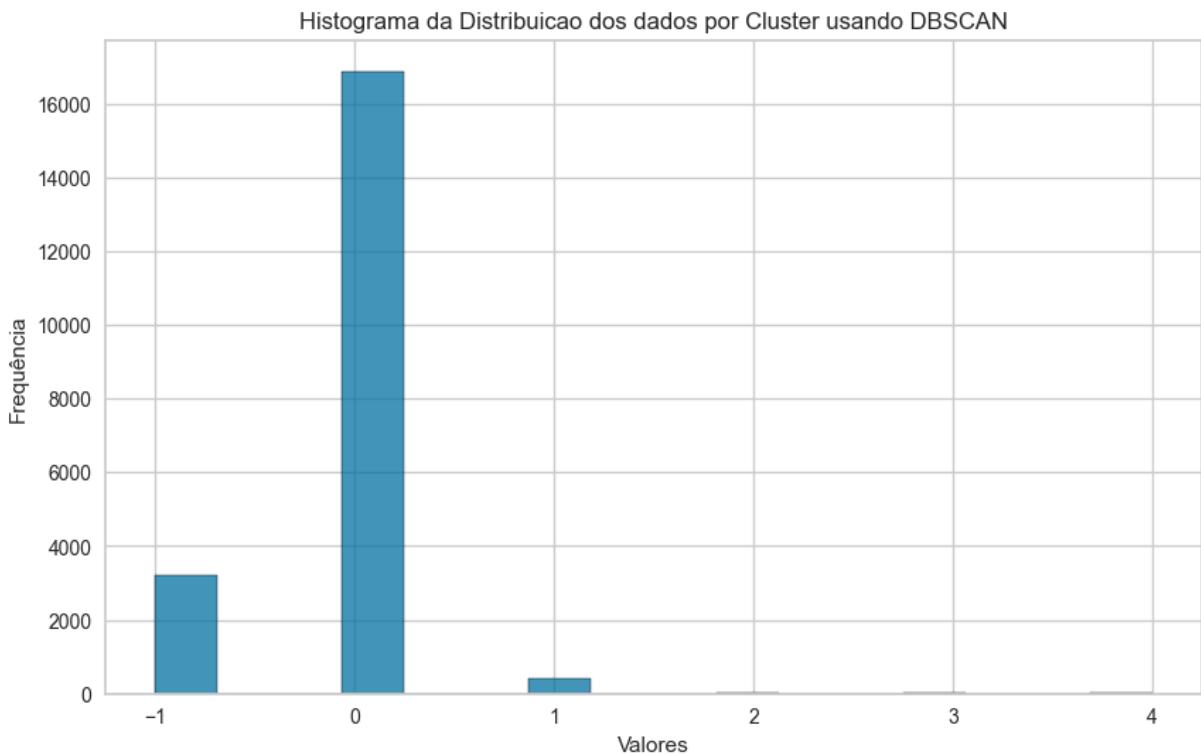
plt.show()

```

Silhouette Score médio: 0.12504827473950672



```
In [248]: # Criar o histograma
plt.figure(figsize=(10, 6))
sns.histplot(labels_dbSCAN)
plt.xlabel('Valores')
plt.ylabel('Frequência')
plt.title('Histograma da Distribuição dos dados por Cluster usando DBSCAN')
plt.show()
```



```
In [162]: from sklearn.cluster import AgglomerativeClustering
```

```
from scipy.cluster.hierarchy import dendrogram

def plot_dendrogram(model, **kwargs):
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1 # Leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

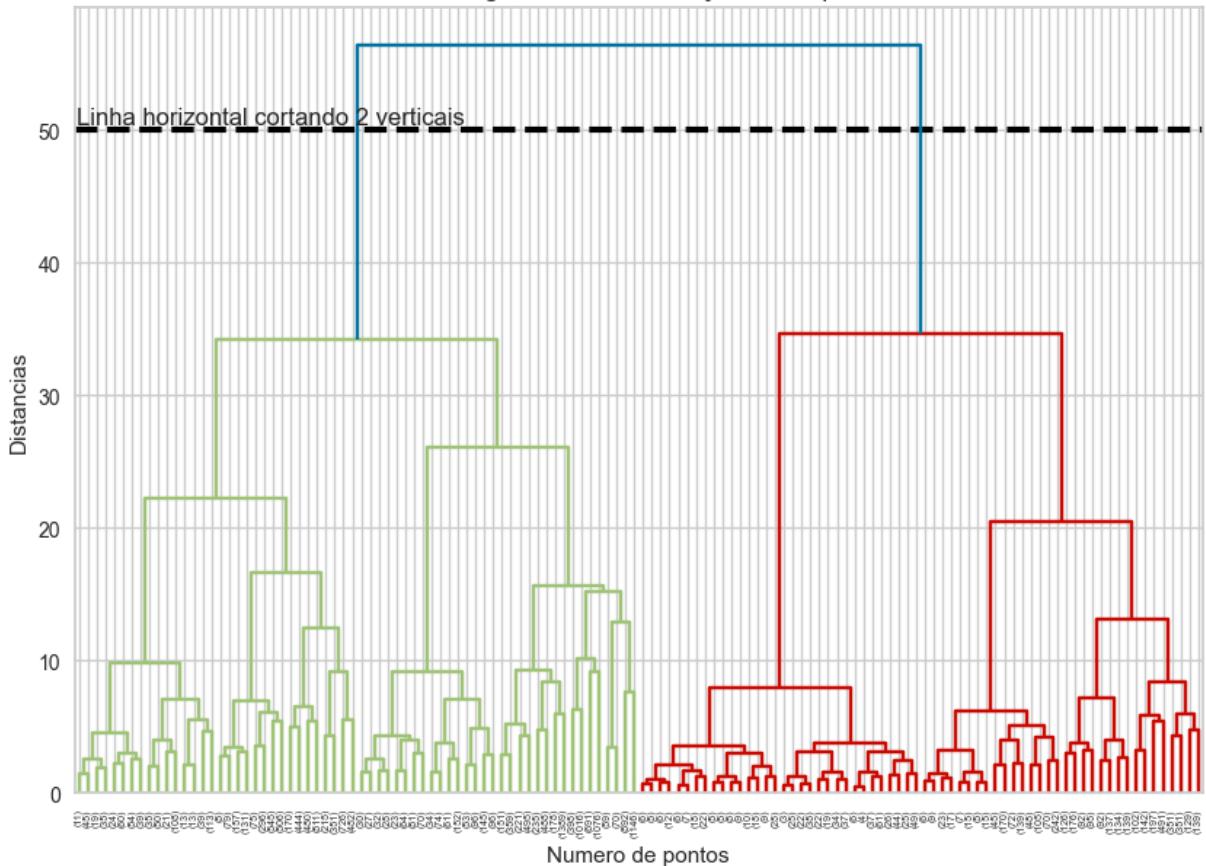
    linkage_matrix = np.column_stack([model.children_, model.distances_,
                                      counts]).astype(float)

    return dendrogram(linkage_matrix, **kwargs)

cluster = AgglomerativeClustering(distance_threshold=0, n_clusters=None)
model = cluster.fit(dataset_normalizado)

plt.figure(figsize=(10, 7))
plt.title('Dendrograma da Clusterização Hierárquica')
plt.ylabel('Distancias')
plt.hlines(y=50,xmin=0,xmax=2000,lw=3,linestyles='--',colors='black' )
plt.text(x=2,y=50.5,s='Linha horizontal cortando 2 verticais',fontsize=12)
#plt.grid(True)
dendro=plot_dendrogram(model, truncate_mode='level', p=6)
plt.xlabel("Numero de pontos")
plt.show()
```

Dendrograma da Clusterização Hierárquica



```
In [ ]: #3.3 Compare os dois resultados, aponte as semelhanças e diferenças e interprete.
# O indice de silhueta e o melhor numero de cluster resultou em valores muito proximamente iguais.
# DBScan cluster = 5, silhueta = 0.125
# Kmeans cluster = 5, silhueta = 0.222

# Para o DBScan quase todos os pontos ficaram agrupados no mesmo cluster (cluster 0)
# foi um pouco mais distribuido entre os clusters.
```

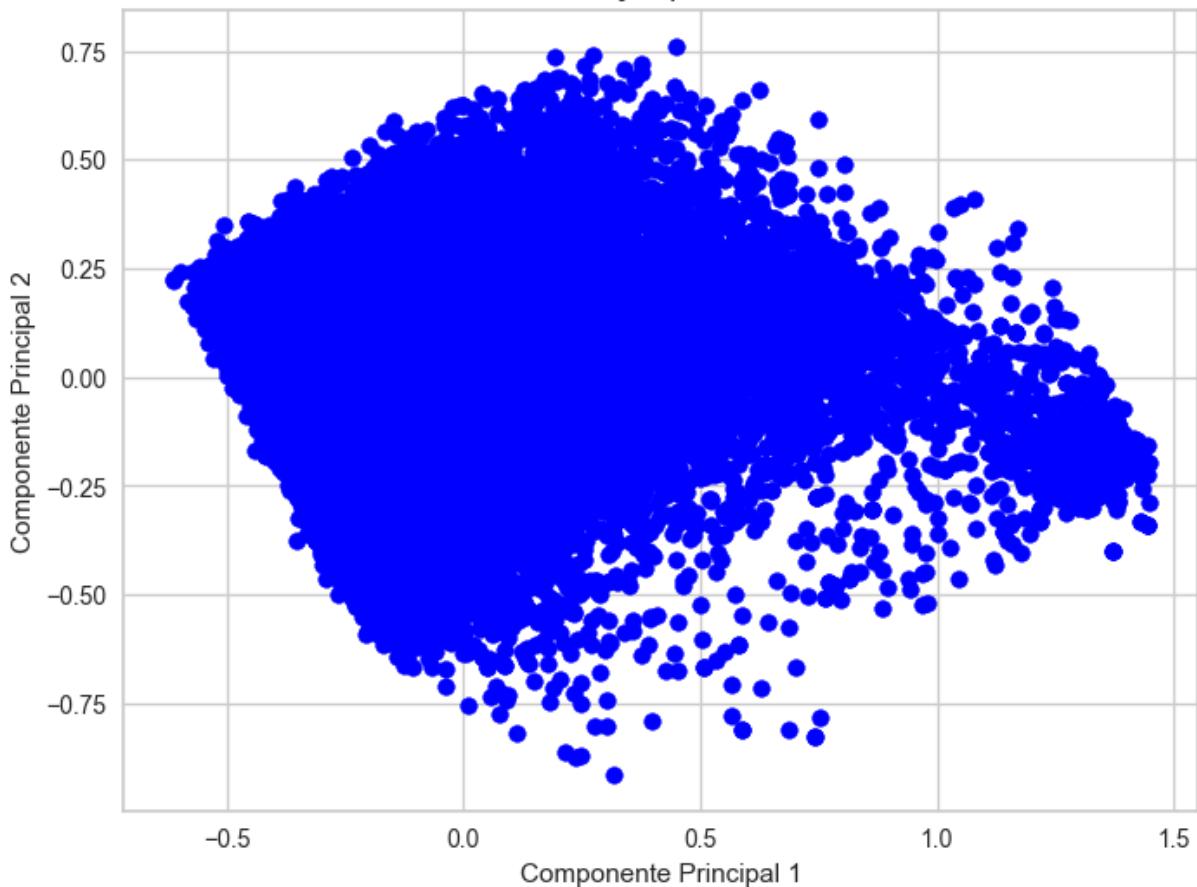
```
In [163...]: #Aplicando o PCA para duas variaveis, vamos tentar identificar a distribuicao.
# Aparentemente esta com uma densidade extremamente alta.
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
data_pca = pca.fit_transform(dataset_normalizado)

# Criar um DataFrame com os resultados do PCA
pca_df = pd.DataFrame(data_pca, columns=['Componente Principal 1', 'Componente Principal 2'])

# Visualizar os resultados do PCA
plt.figure(figsize=(8, 6))
plt.scatter(pca_df['Componente Principal 1'], pca_df['Componente Principal 2'], c='red')
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.title('PCA com Redução para 2 Dimensões')
plt.show()
```

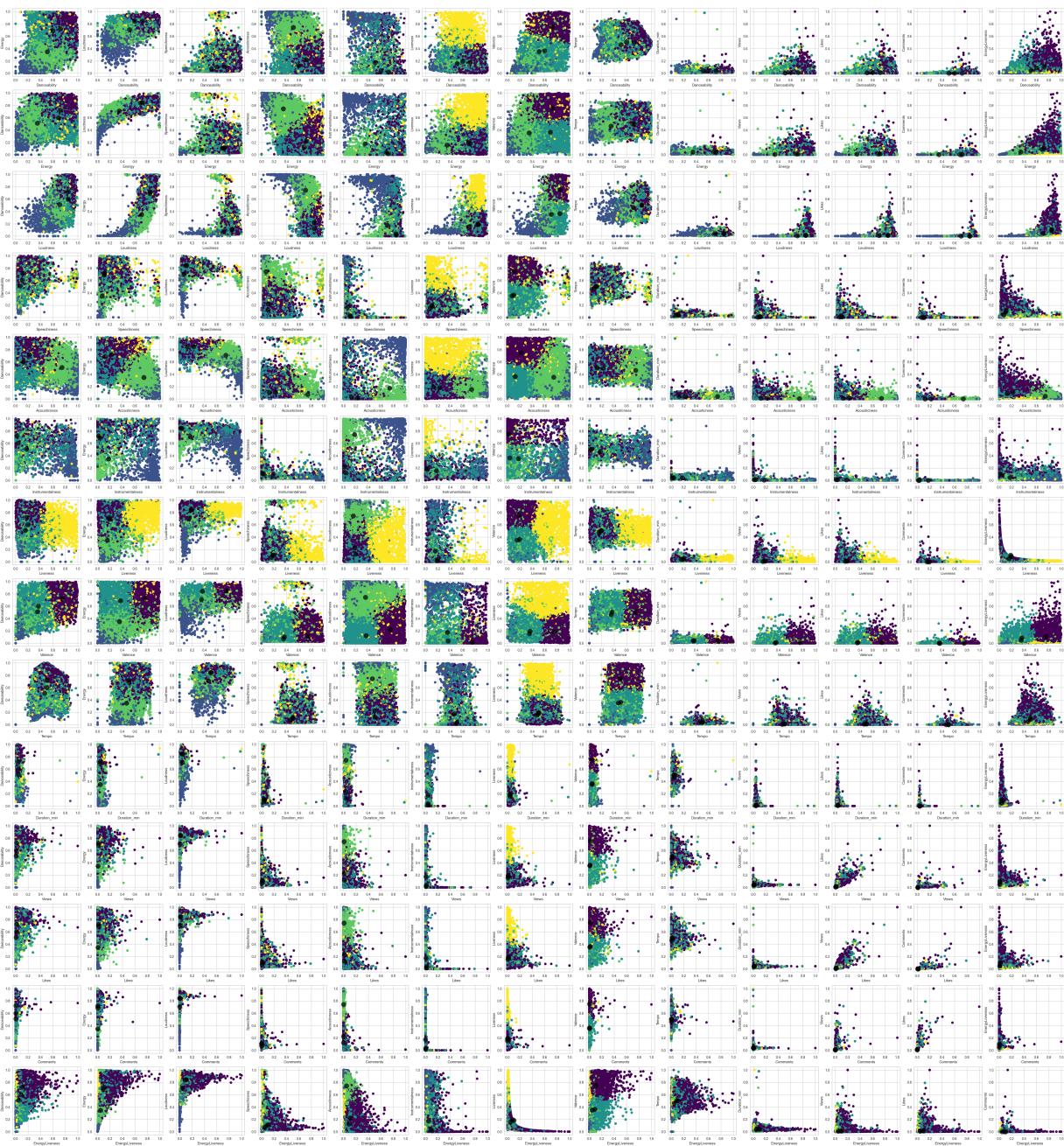
PCA com Redução para 2 Dimensões



```
In [ ]: centers = kmeans.cluster_centers_

plt.figure(figsize=(60, 60))
index=0
for col1 in dataset_normalizado.columns:
    for col2 in dataset_normalizado.columns:
        if col1 != col2:
            index+=1
            plt.subplot(14,14,index)
            plt.scatter(dataset_normalizado[col1], dataset_normalizado[col2], c=y_k
            plt.xlabel(col1)
            plt.ylabel(col2)
            plt.scatter(centers[:, dataset_normalizado.columns.get_loc(col1)], cent
index+=1

#Fazendo uma analise breve dos plots 2 a 2 podemos ver que os dados estao MUITO agl
```

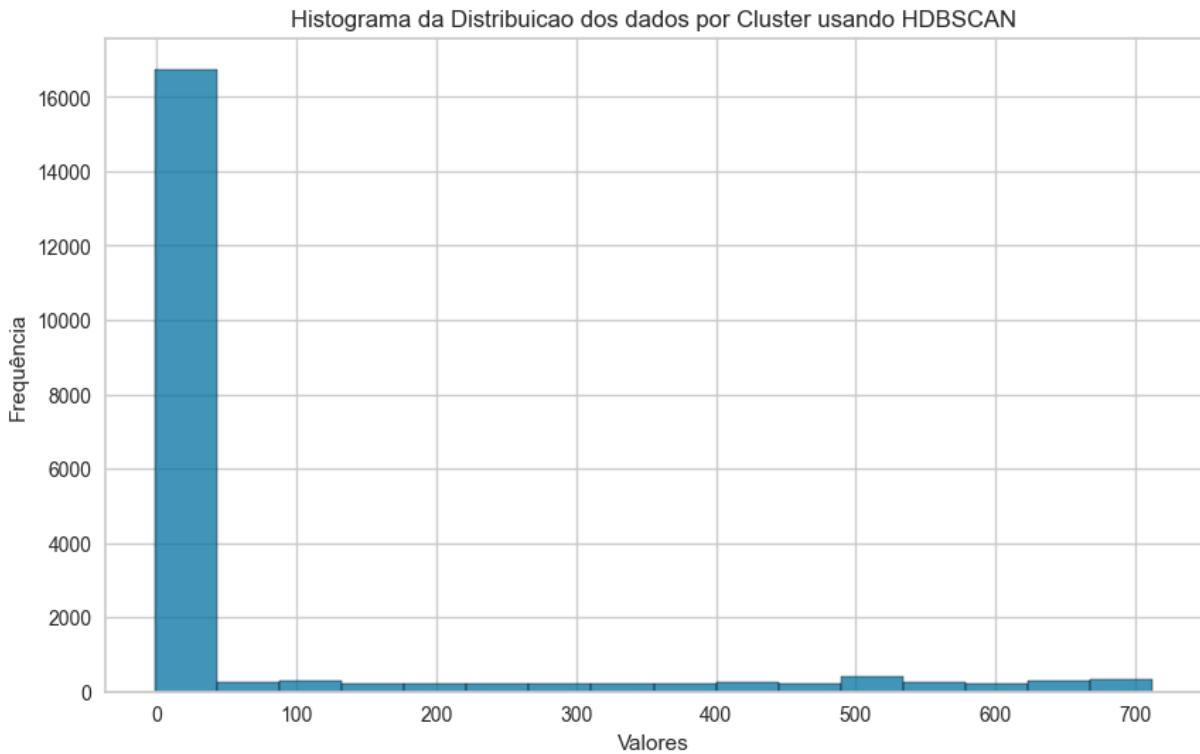


In []: `import hdbscan`

```
hdbscan_clusterer = hdbscan.HDBSCAN(min_cluster_size=3)
labels_hdbscan = hdbscan_clusterer.fit_predict(dataset_normalizado)
```

```
# Criar o histograma
plt.figure(figsize=(10, 6))
sns.histplot(labels_hdbscan)
plt.xlabel('Valores')
plt.ylabel('Frequência')
plt.title('Histograma da Distribuição dos dados por Cluster usando HDBSCAN')
plt.show()
```

#Assim como os demais algoritmos , o HDBSCAN também agrupou muitos dados num mesmo



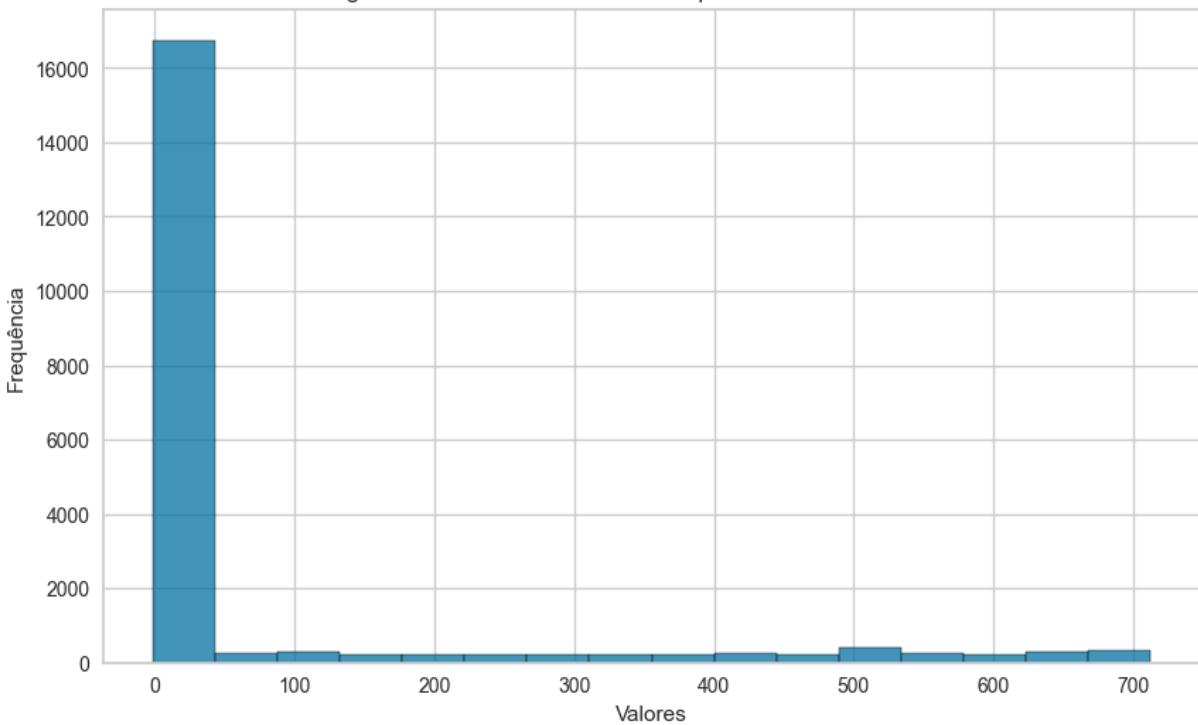
```
In [ ]: from sklearn.cluster import MeanShift

#Mean Shift é um algoritmo de clusterização baseado em densidade que não requer esp
# prévia do número de clusters. Ele identifica os modos (picos) na distribuição de

mean_shift = MeanShift()
labels_mean_shift = mean_shift.fit_predict(dataset_normalizado)

plt.figure(figsize=(10, 6))
sns.histplot(labels_hdbscan)
plt.xlabel('Valores')
plt.ylabel('Frequência')
plt.title('Histograma da Distribuição dos dados por Cluster usando MeanShift')
plt.show()
#Assim como os demais algoritmos , o MeanShift também agrupou muitos dados num mesm
```

Histograma da Distribuição dos dados por Cluster usando MeanShift



```
In [ ]: #3.4. Escolha mais duas medidas de validação para comparar com o índice de silhueta
# analise os resultados encontrados. Observe, para a escolha, medidas adequadas aos
#Índice Davies-Bouldin (Davies-Bouldin Index):
#Avalia a média da razão entre as distâncias intracluster e intercluster. Valores m
#Índice Calinski-Harabasz
# Ele se baseia na dispersão dentro dos clusters (intra-cluster) e na dispersão ent
from sklearn.metrics import calinski_harabasz_score,davies_bouldin_score

# Aplicar K-Means
kmeans = KMeans(n_clusters=3)
labels_kmeans = kmeans.fit_predict(dataset_normalizado)

# Aplicar DBSCAN
dbSCAN = DBSCAN(eps=0.2, min_samples=19,algorithm = "ball_tree" )
labels_dbSCAN = dbSCAN.fit_predict(dataset_normalizado)

# Calcular métricas de validação para K-Means
calinski_harabasz_kmeans = calinski_harabasz_score(dataset_normalizado, labels_kmea
davies_bouldin_kmeans = davies_bouldin_score(dataset_normalizado, labels_kmeans)

print(f"K-Means - Coeficiente de Calinski Harabasz: {calinski_harabasz_kmeans}")
print(f"K-Means - Índice Davies-Bouldin: {davies_bouldin_kmeans}")

# Calcular métricas de validação para DBSCAN (se houver mais de um cluster)
if len(set(labels_dbSCAN)) > 1:
    calinski_harabasz_dbSCAN = calinski_harabasz_score(dataset_normalizado, labels_
    davies_bouldin_dbSCAN = davies_bouldin_score(dataset_normalizado, labels_dbSCAN)
```

```

print(f"DBSCAN - Coeficiente de Calinski Harabasz: {calinski_harabasz_dbSCAN}")
print(f"DBSCAN - Índice Davies-Bouldin: {davies_bouldin_dbSCAN}")

K-Means - Coeficiente de Calinski Harabasz: 6335.190950529577
K-Means - Índice Davies-Bouldin: 1.6481574396316956
DBSCAN - Coeficiente de Calinski Harabasz: 2011.4398627382725
DBSCAN - Índice Davies-Bouldin: 2.6424353278381543

```

In []: #3.5. Realizando a análise, responda: A silhueta é um o índice indicado para escolher os clusters para o algoritmo de DBScan?

Acredito que o indice de silhueta pode nao ser muito indicado para DBSCAN por conter clusters de tamanhos e formas variadas. O Coeficiente de Silhueta é mais adequado para DBSCAN.

PARTE 4 - MEDIDAS DE SIMILARIDADE

In [236]: #1. Um determinado problema, apresenta 10 séries temporais distintas.
Gostaríamos de agrupá-las em 3 grupos, de acordo com um critério de similaridade, baseado no valor máximo de correlação cruzada entre elas. Descreva em tópicos todos os passos necessários para resolver esse problema.

- Normalizar e tratar os dados (rotina de pre processamento)
- Para cada par de series temporais, calcular a correlação cruzada maxima, medindo a similaridade entre elas
- Construir uma matriz de similaridade onde cada entrada representa a correlação entre duas series temporais
- Converter a matriz de similaridade em uma matriz de distância. Uma abordagem comum é usar a distância euclidiana entre os vetores de similaridade
- Utilizar um algoritmo de clusterização adequado, como K-Means, Hierarchical Clustering ou DBSCAN
- Verificar a qualidade da clusterização aplicando as métricas corretas, por exemplo o Coeficiente de Calinski-Harabasz

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

np.random.seed(22)

num_samples = 10
length_of_series = 100

# Gerando 10 amostras de séries temporais
amostras_series = [np.abs(np.random.randn(length_of_series)).cumsum() for _ in range(num_samples)]

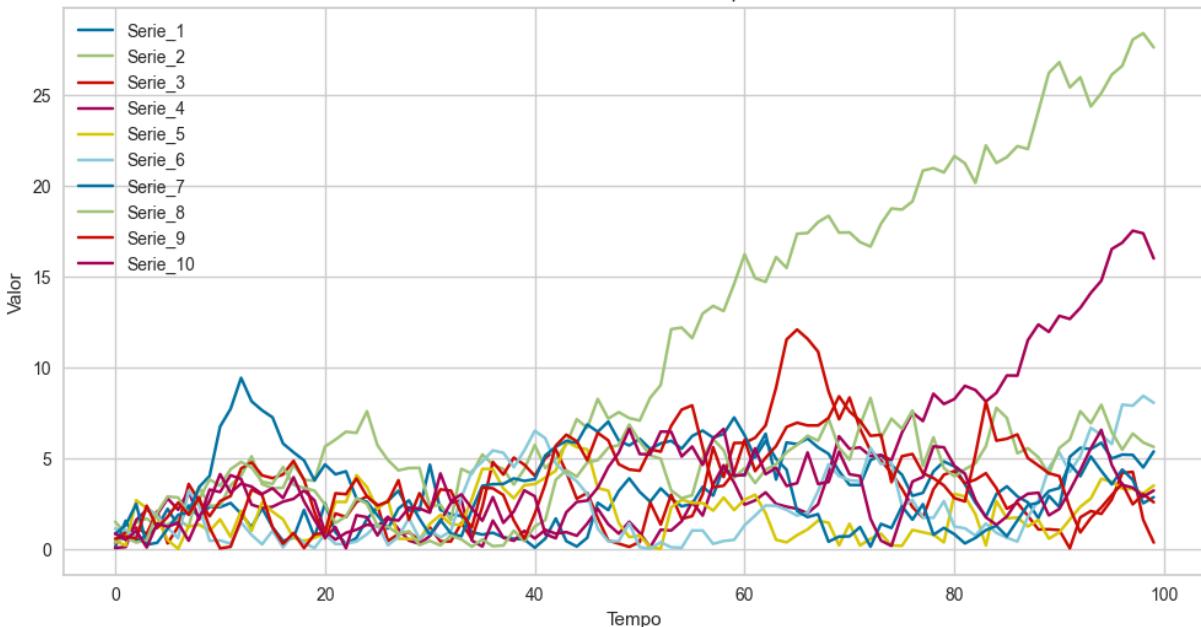
df_series_temporais = pd.DataFrame(amostras_series).T
df_series_temporais.columns = [f'Serie_{i+1}' for i in range(num_samples)]

plt.figure(figsize=(12, 6))
for column in df_series_temporais.columns:
    plt.plot(df_series_temporais.index, df_series_temporais[column], label=column)

plt.title('10 Amostras de Séries Temporais')
plt.xlabel('Tempo')
plt.ylabel('Valor')
plt.legend()
plt.show()

```

10 Amostras de Sérias Temporais



```
In [237...]: from scipy.signal import correlate

normalizer = MinMaxScaler()

series_temporais_normalizadas = [normalizer.fit_transform(serie.reshape(-1, 1)).flatten() for serie in series_temporais]

def max_cross_correlation(serie1, serie2):
    correlation = correlate(serie1, serie2, mode='full')
    return max(correlation)

matriz_similaridade = np.zeros((num_samples, num_samples))

for i in range(num_samples):
    for j in range(num_samples):
        if i != j:
            matriz_similaridade[i, j] = max_cross_correlation(series_temporais_norm[i], series_temporais_norm[j])
        else:
            matriz_similaridade[i, j] = 1

distance_matrix = 1 - matriz_similaridade

from sklearn.manifold import MDS

mds = MDS(n_components=num_samples, dissimilarity='precomputed', random_state=22)
mds_distancia_matrix = mds.fit_transform(distance_matrix)

kmeans = KMeans(n_clusters=3, random_state=22)
labels = kmeans.fit_predict(mds_distancia_matrix)

# Avaliar a qualidade dos clusters
silhouette_avg = silhouette_score(mds_distancia_matrix, labels)
calinski_harabasz = calinski_harabasz_score(mds_distancia_matrix, labels)
davies_bouldin = davies_bouldin_score(mds_distancia_matrix, labels)
```

```

print(f"Coeficiente de Silhueta: {silhouette_avg}")
print(f"Índice Calinski-Harabasz: {calinski_harabasz}")
print(f"Índice Davies-Bouldin: {davies_bouldin}")

# Visualização dos clusters (opcional)
import matplotlib.pyplot as plt

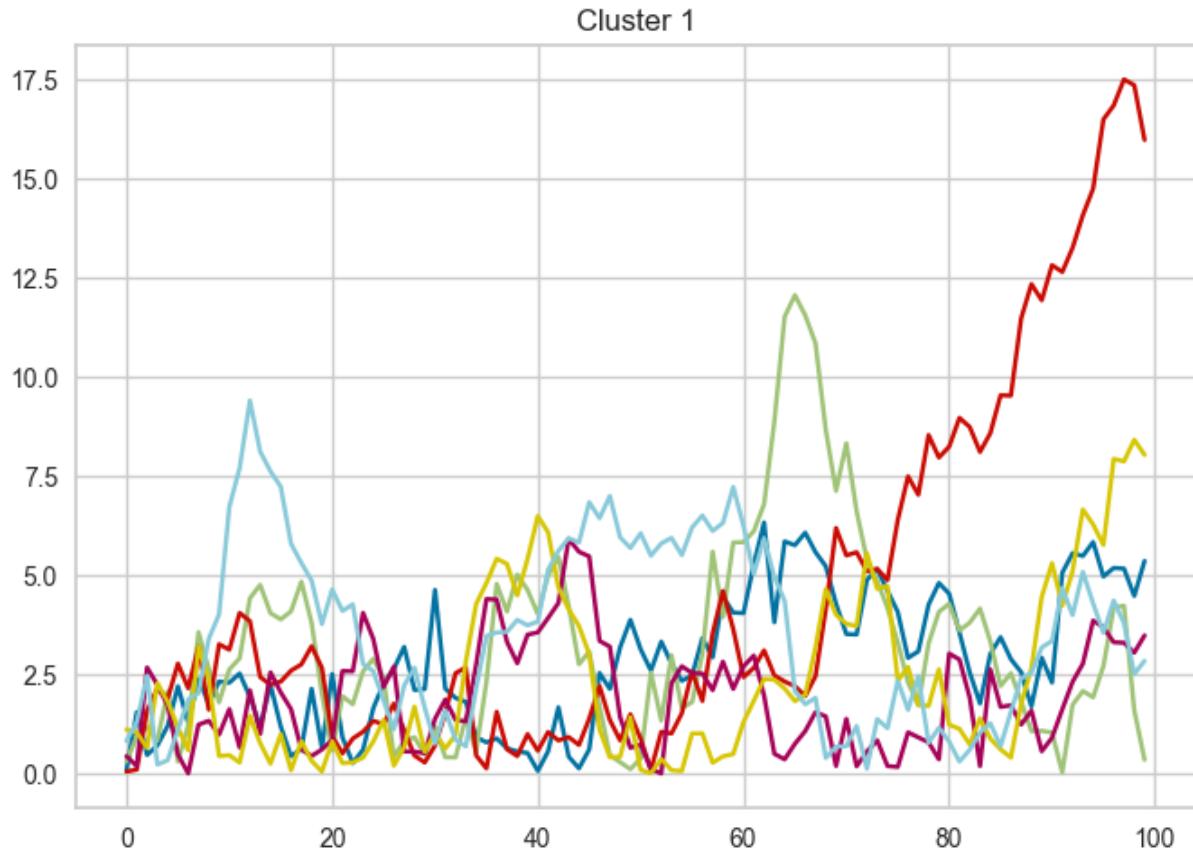
# Exemplo de visualização de uma série temporal em cada cluster
for cluster in range(3):
    plt.figure()
    for i in range(len(amostras_series)):
        if labels[i] == cluster:
            plt.plot(amostras_series[i])
    plt.title(f"Cluster {cluster + 1}")
    plt.show()

```

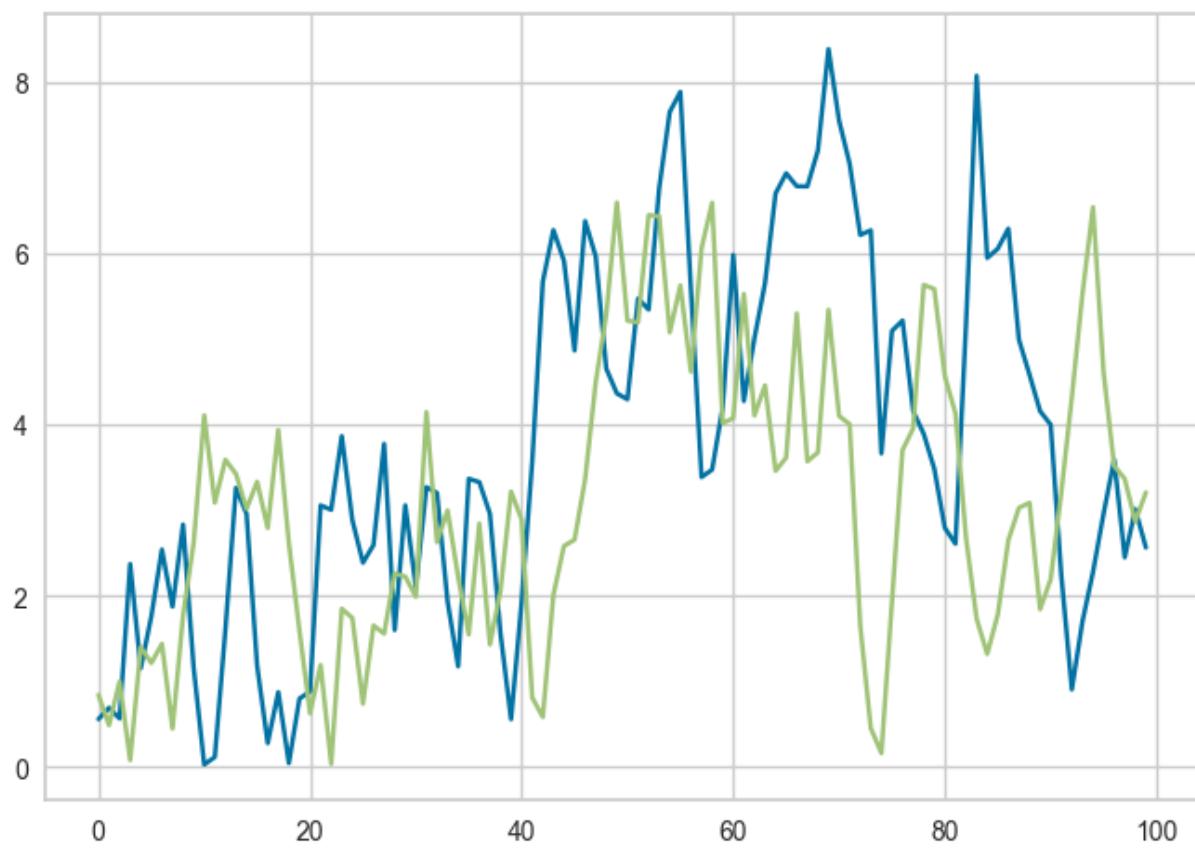
Coeficiente de Silhueta: 0.23895455147672084

Índice Calinski-Harabasz: 3.851692495863009

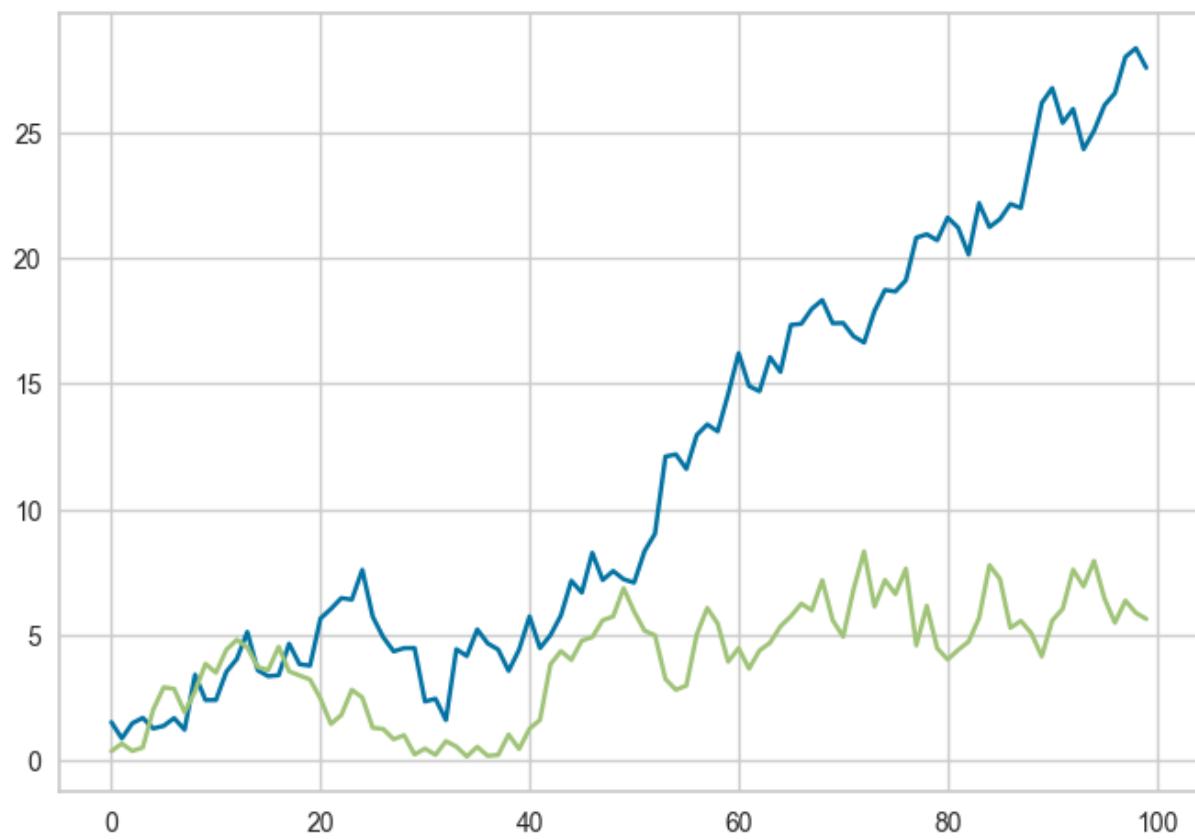
Índice Davies-Bouldin: 1.0716391387160202



Cluster 2



Cluster 3



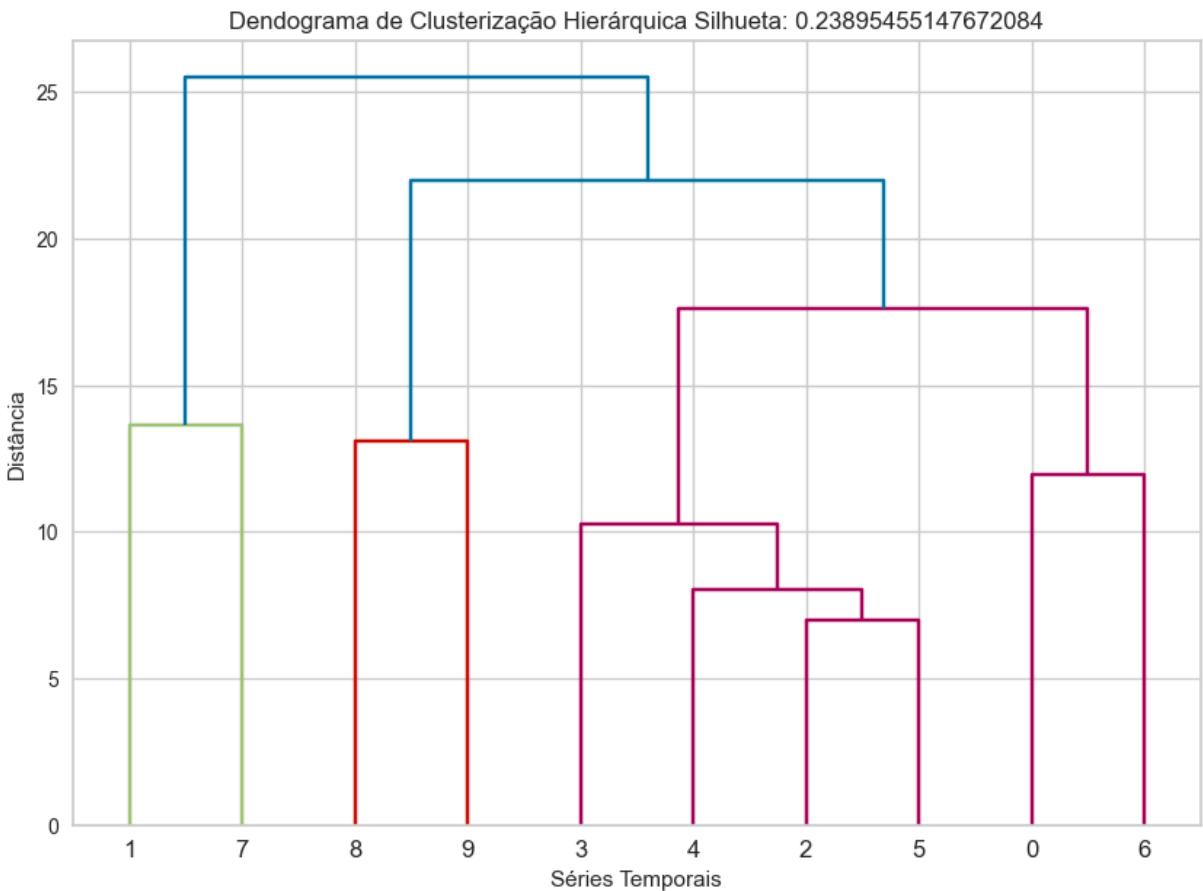
In [238]: `from scipy.cluster.hierarchy import linkage, dendrogram, fcluster`

```

linkage_matrix = linkage(mds_distancia_matrix, method='ward')
num_clusters = 3
labels = fcluster(linkage_matrix, num_clusters, criterion='maxclust')
silhueta_hierarquico_temporal = silhouette_score(mds_distancia_matrix, labels)

plt.figure(figsize=(10, 7))
dendrogram(linkage_matrix)
plt.title(f"Dendograma de Clusterização Hierárquica Silhueta: {silhueta_hierarquico_temporal}")
plt.xlabel("Séries Temporais")
plt.ylabel("Distância")
plt.show()

```



In []: #2. Para o problema da questão anterior, indique qual algoritmo de clusterização você escolhe para esse tipo de dado.

#A escolha do melhor algoritmo de clusterização para séries temporais depende dos dados e das necessidades do usuário. K-Means: Mais adequado para séries temporais com padrões bem definidos e sem muita variação. DBSCAN: Ideal para dados com densidades variáveis e para identificar clusters de alta densidade. Hierarchical Clustering: útil para visualizar a estrutura dos dados e determinar a similaridade entre amostras.

Para o exemplo de problema acima, como não percebi nenhum padrão, seria talvez mais adequado usar K-Means ou DBSCAN.

In []: #3. Indique um caso de uso para essa solução projetada.

Monitoramento de Equipamentos Industriais
Identificar padrões operacionais e detectar comportamentos anômalos em sensores de produção.
Clusterizar séries temporais de sensores para diferenciar modos normais de operação.
Detectar anomalias, como falhas iminentes ou condições de manutenção.

In [245...]

```
#4. Sugira outra estratégia para medir a similaridade entre séries temporais. Descr

#Distância Dinâmica de Time Warping (DTW): ele considera desalinhamentos no tempo,
# mesmo quando eventos ocorrem em momentos diferentes.
#Passos para medir com DTW:
# - Pre processamento dos dados
# - Calcular a distância DTW entre todas as combinações de series temporais e const
# - Aplicar algoritmo de clusterizacao (DBscan, Hierarquico, Kmeans... )
# - Validacao do algoritmo

from tslearn.metrics import dtw
from scipy.spatial.distance import euclidean

def verificar_unidimensionalidade(series):
    return [serie.flatten() for serie in series]

series_temporais_unidimensionais = verificar_unidimensionalidade(series_temporais_n

def calcular_matriz_dtw(series):
    n = len(series)
    matriz_distancias = np.zeros((n, n))

    for i in range(n):
        for j in range(i + 1, n):
            distancia = dtw(series[i], series[j])
            matriz_distancias[i, j] = distancia
            matriz_distancias[j, i] = distancia

    return matriz_distancias

# Calcular a matriz de distância DTW para as séries temporais
matriz_dtw = calcular_matriz_dtw(series_temporais_unidimensionais)

mds = MDS(n_components=num_samples, dissimilarity='precomputed', random_state=22)
mds_distancia_matrix = mds.fit_transform(distance_matrix)

kmeans = KMeans(n_clusters=3, random_state=22)
labels = kmeans.fit_predict(matriz_dtw)

# Avaliar a qualidade dos clusters
silhouette_avg = silhouette_score(matriz_dtw, labels)
calinski_harabasz = calinski_harabasz_score(matriz_dtw, labels)
davies_bouldin = davies_bouldin_score(matriz_dtw, labels)

print(f"Coeficiente de Silhueta: {silhouette_avg}")
print(f"Índice Calinski-Harabasz: {calinski_harabasz}")
print(f"Índice Davies-Bouldin: {davies_bouldin}")

# Visualização dos clusters (opcional)
import matplotlib.pyplot as plt

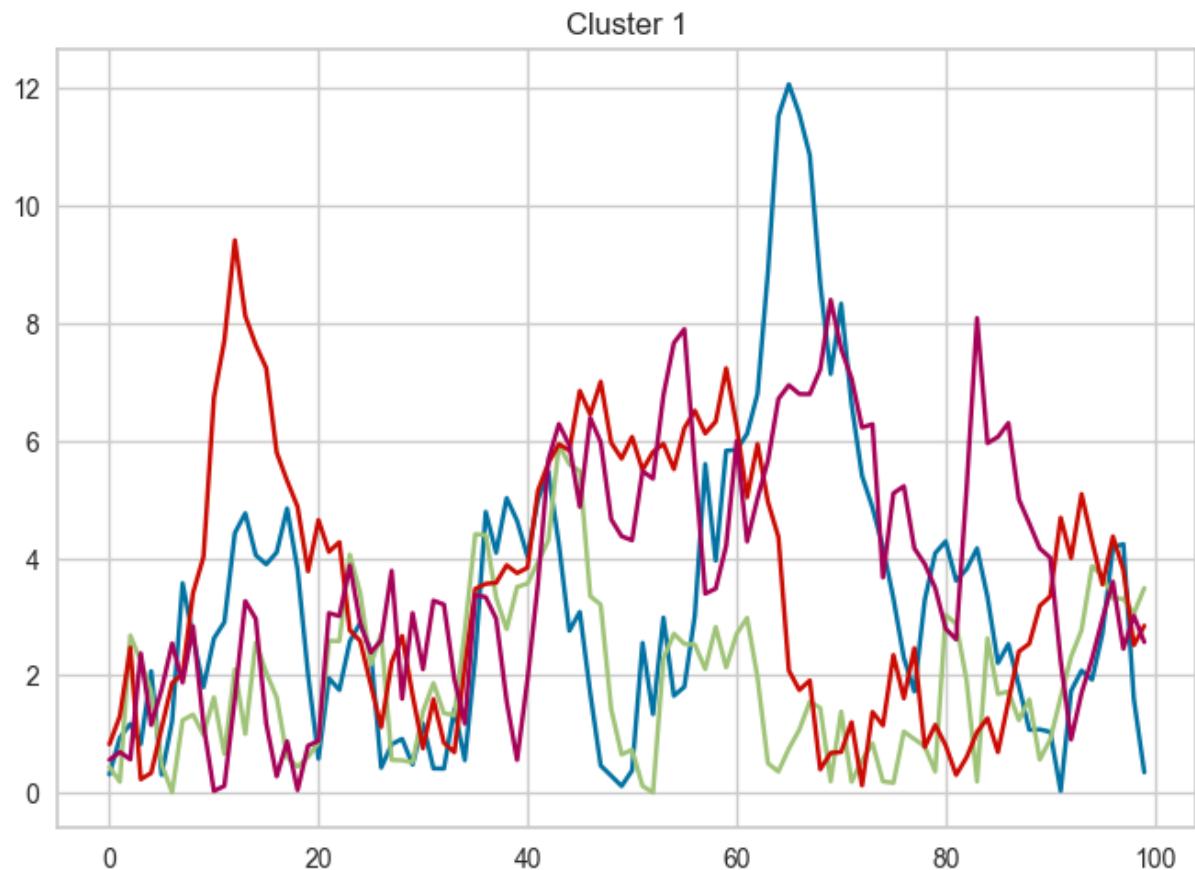
# Exemplo de visualização de uma série temporal em cada cluster
for cluster in range(3):
    plt.figure()
    for i in range(len(amostras_series)):
        if labels[i] == cluster:
            amostras_series[i].plot()
```

```
plt.plot(amostras_series[i])
plt.title(f"Cluster {cluster + 1}")
plt.show()
```

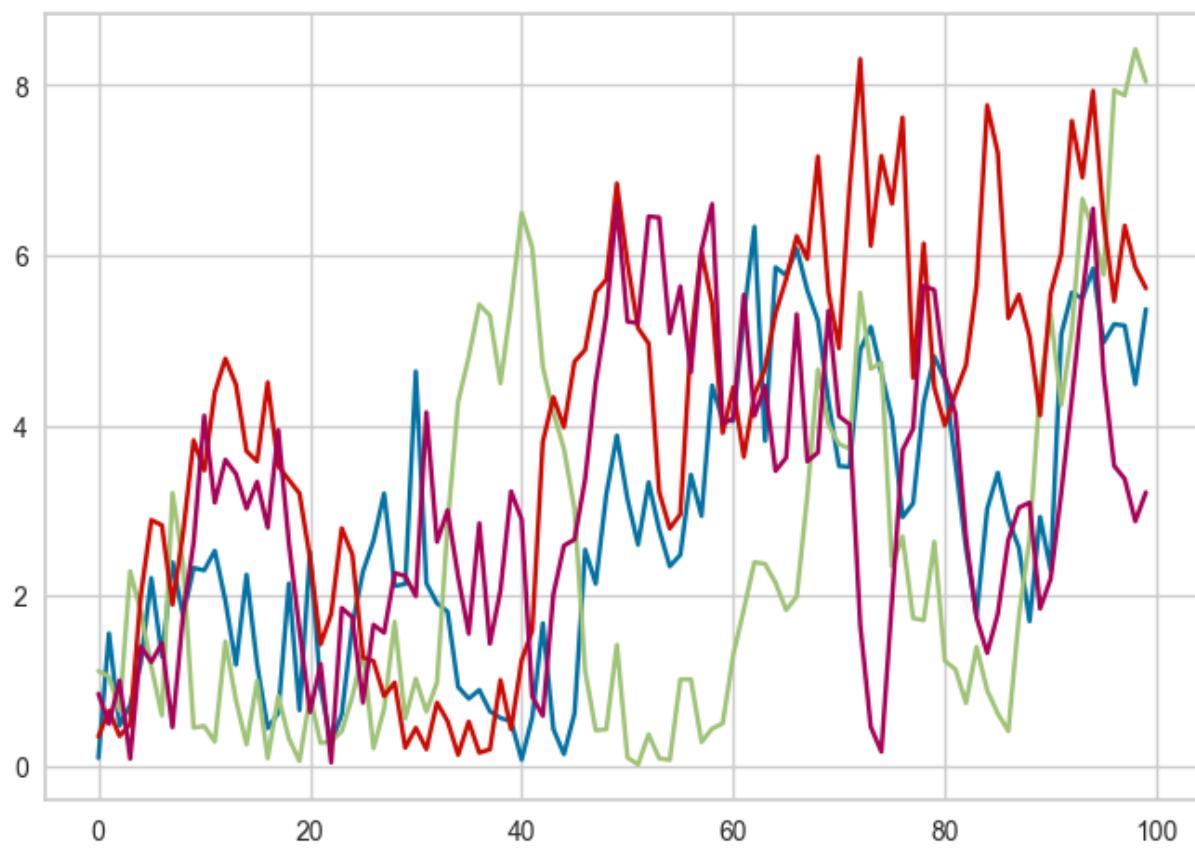
Coeficiente de Silhueta: 0.36384300887597587

Índice Calinski-Harabasz: 9.153177682880244

Índice Davies-Bouldin: 0.8658638370195421



Cluster 2



Cluster 3

