

# Projeto 2 Astrometria

André Almeida Trovello

**Estudo de regiões de formação estelar (Taurus) com métodos de Machine Learning**

## 1. Query

**Importando Bibliotecas**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from astroquery.gaia import Gaia
from sklearn.cluster import KMeans, DBSCAN, HDBSCAN
from sklearn.metrics import silhouette_score, silhouette_samples
from astropy.table import Table
```

**Criando Dataframe**

```
data = pd.read_csv('galli_2019_table1.csv')
main_table = pd.DataFrame(data)
```

**Tabela do Vizier**

```

# Garanta que a variável 'main_table' existe.
try:
    # Se já tiver 'main_table' na memória (ex: num notebook), pula
    main_table
    print("Variável 'main_table' já existe na memória.")
except NameError:
    try:
        # Tente carregar do CSV
        main_table = Table.read('galli_2019_table1.csv')
        print("Arquivo 'galli_2019_table1.csv' carregado com sucesso.")
    except FileNotFoundError:
        print("Erro: A variável 'main_table' não foi encontrada na memória.")
        print("E o arquivo 'galli_2019_table1.csv' não \
              foi encontrado no disco.")
        print("Por favor, rode o script do VizieR \
              primeiro ou verifique o nome do arquivo CSV.")
    raise

```

Variável 'main\_table' já existe na memória.

### Limpar os IDs

Remove “Gaia DR2” dos nomes dos objetos mantendo apenas seus IDs

```

print("Iniciando a limpeza dos IDs...")
try:
    dr2_id_strings = main_table['GaiaDR2'].tolist()

    dr2_ids_cleaned_list = []
    ids_pulados = 0

    for s in dr2_id_strings:
        try:
            id_str = s.split(' ')[-1]
            numeric_id = np.int64(id_str)
            dr2_ids_cleaned_list.append(numeric_id)
        except (AttributeError, IndexError, ValueError):
            ids_pulados += 1

    print(f"Encontrada e limpa a lista de \
          {len(dr2_ids_cleaned_list)} IDs (DR2) válidos.")

```

```

if ids_pulados > 0:
    print(f'{ids_pulados} linhas foram puladas por não \
        terem um ID válido)")

except (KeyError) as e:
    print(f"Erro: A coluna 'GaiaDR2' não foi encontrada na 'main_table'.")
    raise

```

Iniciando a limpeza dos IDs...  
Encontrada e limpa a lista de 458 IDs (DR2) válidos.  
(61 linhas foram puladas por não terem um ID válido)

### Cria a tabela para upload e a query organizada

```

upload_table = Table({'dr2_source_id_list': dr2_ids_cleaned_list})

# 1. Começa com a sua tabela (user_table)
# 2. Usa o nome correto da tabela (gaiadr3.dr2_neighbourhood)
# para cruzar os IDs
query_dr3 = """
SELECT
    dr3.source_id, dr3.ra, dr3.dec, dr3.parallax, dr3.pmra,
    dr3.pmdec, xmatch.angular_distance,
    dr3.l, dr3.b, xmatch.magnitude_difference, dr3.ruwe,
    dr3.phot_g_mean_mag, dr3.bp_rp
FROM
    tap_upload.my_table AS user_table
JOIN
    gaiadr3.dr2_neighbourhood AS xmatch
    ON user_table.dr2_source_id_list = xmatch.dr2_source_id
JOIN
    gaiadr3.gaia_source AS dr3
    ON xmatch.dr3_source_id = dr3.source_id
"""

```

### Executa a busca

```

print("Iniciando a busca no Gaia DR3 (com query otimizada)...")


try:
    job = Gaia.launch_job_async(
        query=query_dr3,
        upload_resource=upload_table,
        upload_table_name="my_table",
        verbose=True # Adiciona mais informações de debug
    )

    results_dr3_members = job.get_results()

    print(f"\nBusca concluída!")
    print(f"Foram encontrados dados no DR3 para {len(results_dr3_members)} \
          das {len(dr2_ids_cleaned_list)} estrelas.")

    print("\n--- 5 primeiras linhas dos membros de Taurus (dados do DR3) ---")
    print(results_dr3_members.to_pandas().head())

    # Salva os resultados
    results_dr3_members.write('taurus_membros_dr3.csv',
                               format='csv', overwrite=True)

except Exception as e:
    print(f"\nOcorreu um erro durante a busca no Gaia:")
    print(e)

```

Iniciando a busca no Gaia DR3 (com query otimizada)...

Launched query:

```

SELECT
    dr3.source_id, dr3.ra, dr3.dec, dr3.parallax, dr3.pmra,
    dr3.pmdec, xmatch.angular_distance,
    dr3.l, dr3.b, xmatch.magnitude_difference, dr3.rupe,
    dr3.phot_g_mean_mag, dr3.bp_rp
FROM
    tap_upload.my_table AS user_table
JOIN
    gaiadr3.dr2_neighbourhood AS xmatch
    ON user_table.dr2_source_id_list = xmatch.dr2_source_id
JOIN
    gaiadr3.gaia_source AS dr3

```

```

ON xmatch.dr3_source_id = dr3.source_id
'
----->https
host = gea.esac.esa.int:443
context = /tap-server/tap/async
Content-type = multipart/form-data; boundary=====1763100740753===
303 303
[('Date', 'Fri, 14 Nov 2025 06:12:22 GMT'), ('Server', 'Apache/2.4.6 (SLES Expanded Support
job 17631007420830, at: https://gea.esac.esa.int/tap-server/tap/async/17631007420830
Retrieving async. results...
INFO: Query finished. [astroquery.utils.tap.core]

Busca concluída!
Foram encontrados dados no DR3 para 499 das 458 estrelas.

--- 5 primeiras linhas dos membros de Taurus (dados do DR3) ---
      source_id          ra        dec   parallax     pmra     pmdec \
0  162535413750345856  60.955653  26.181023  6.758039  20.853242 -30.277735
1  162535413754166528  60.955305  26.180727  7.803457  20.133753 -27.911397
2  162541942104406784  60.958322  26.343862  6.971044  14.435075 -19.268191
3  162535345034688768  60.961924  26.181260  7.731206  19.547795 -30.009057
4  53092775104124288  61.164068  21.971754  8.147688  3.802709 -15.247629

      angular_distance         l         b magnitude_difference      ruwe \
0           5.860794  168.015225 -19.404779          0.033466  21.986296
1          1542.932129  168.015210 -19.405209          0.448453  1.159977
2            0.180421  167.895871 -19.287135         -0.025953  1.049943
3            0.095608  168.019296 -19.400658         -0.016413  2.134250
4            0.127694  171.356321 -22.234012         -0.015548  5.566028

      phot_g_mean_mag      bp_rp
0           13.836937  2.874246
1           14.251924      NaN
2           16.958754  3.813750
3           13.161151  2.514164
4           13.681566  2.646513

```

## 2. Seleção da Amostra

### Carregando tabelas

```
data = pd.read_csv('taurus_membros_dr3.csv')
df = pd.DataFrame(data)
#X = df[['ra', 'dec', 'parallax', 'pmra', 'pmdec']]
#print(X['ra'])
#print(df)
# 1. Calcula as contagens de cada source_id
counts = df['source_id'].value_counts()

# 2. Usa .map() para criar uma nova série do mesmo tamanho do df,
#     onde cada valor é a contagem do seu respectivo source_id.
#     Depois, compara com 2 para criar o filtro correto.
full_table = df[df['source_id'].map(counts) == 2]

full_table
```

	source_id	ra	dec	parallax	pmra	pmdec	angular_distance
34	163179006011625088	63.717943	28.099839	7.486891	8.195294	-22.940540	0.256296
35	163179006011625216	63.718513	28.099846	7.183761	9.408134	-24.033823	1809.979248
36	163179006011625088	63.717943	28.099839	7.486891	8.195294	-22.940540	1810.137573
37	163179006011625216	63.718513	28.099846	7.183761	9.408134	-24.033823	0.157825
98	152416436443721728	65.289006	27.843380	7.676996	7.989072	-26.709809	0.740511
99	152416436441091584	65.288916	27.843576	NaN	NaN	NaN	749.300537
100	152416436443721728	65.289006	27.843380	7.676996	7.989072	-26.709809	762.492981
101	152416436441091584	65.288916	27.843576	NaN	NaN	NaN	13.096863
159	152180827420224128	67.177628	27.234390	7.770386	7.754825	-25.039691	0.163206
160	152180831716415488	67.177664	27.234215	7.105773	9.080319	-26.201142	638.086182
161	152180827420224128	67.177628	27.234390	7.770386	7.754825	-25.039691	638.122437
162	152180831716415488	67.177664	27.234215	7.105773	9.080319	-26.201142	0.387675
188	146361013591547776	67.623567	24.445747	NaN	NaN	NaN	24.795189
189	146361013590374144	67.623320	24.445807	6.368969	8.908057	-24.076122	825.430115
190	146361013591547776	67.623567	24.445747	NaN	NaN	NaN	838.572693
191	146361013590374144	67.623320	24.445807	6.368969	8.908057	-24.076122	8.087403
195	145921994918655488	67.709547	23.002367	8.029621	11.949303	-14.773370	0.341585
196	145921999215653632	67.709932	23.002332	6.827561	10.848324	-16.615883	1280.887939
197	145921994918655488	67.709547	23.002367	8.029621	11.949303	-14.773370	1280.096558
198	145921999215653632	67.709932	23.002332	6.827561	10.848324	-16.615883	1.260170

	source_id	ra	dec	parallax	pmra	pmdec	angular_distance
231	3314132593934981248	68.126512	17.524747	6.752014	12.825682	-19.916297	1453.149902
232	3314132593936245248	68.126214	17.525034	6.873723	13.179530	-20.461011	0.183572
234	3314132593934981248	68.126512	17.524747	6.752014	12.825682	-19.916297	0.224493
235	3314132593936245248	68.126214	17.525034	6.873723	13.179530	-20.461011	1453.461304
312	3313386605360382720	68.982060	17.127651	6.948271	11.938078	-19.025502	1881.099121
313	3313386609655429248	68.981703	17.127255	6.674840	13.216349	-19.185871	0.187654
314	3313386605360382720	68.982060	17.127651	6.948271	11.938078	-19.025502	0.238448
315	3313386609655429248	68.981703	17.127255	6.674840	13.216349	-19.185871	1881.066895
353	148378033311467904	69.837256	25.750439	NaN	NaN	NaN	20.975664
354	148378033312276864	69.837141	25.750508	NaN	NaN	NaN	432.957581
355	148378033311467904	69.837256	25.750439	NaN	NaN	NaN	445.195953
356	148378033312276864	69.837141	25.750508	NaN	NaN	NaN	4.853233
362	148386898124771328	70.007363	25.941406	NaN	NaN	NaN	556.125671
363	148386898125118464	70.007242	25.941323	NaN	NaN	NaN	76.230293
364	148386898124771328	70.007363	25.941406	NaN	NaN	NaN	10.010338
365	148386898125118464	70.007242	25.941323	NaN	NaN	NaN	498.875671
454	155744074024631296	74.714207	28.523221	6.877693	6.051624	-29.397043	0.249245
455	155744074023050368	74.714590	28.523333	6.975570	6.545382	-31.430755	1276.910889
456	155744074024631296	74.714207	28.523221	6.877693	6.051624	-29.397043	1277.517456
457	155744074023050368	74.714590	28.523333	6.975570	6.545382	-31.430755	0.885773
479	156430822114424576	76.956081	30.401205	6.406249	2.640643	-27.397583	0.205192
480	156430817820015232	76.956544	30.401316	5.454096	4.906034	-24.577535	1491.435303
481	156430822114424576	76.956081	30.401205	6.406249	2.640643	-27.397583	1488.743896
482	156430817820015232	76.956544	30.401316	5.454096	4.906034	-24.577535	2.974982
486	3419115128091798272	77.034043	24.454042	NaN	NaN	NaN	26.817190
487	3419115132386033280	77.034161	24.454282	NaN	NaN	NaN	953.412903
488	3419115128091798272	77.034043	24.454042	NaN	NaN	NaN	933.809082
489	3419115132386033280	77.034161	24.454282	NaN	NaN	NaN	24.234728
495	3415706130944329216	78.114951	22.896915	5.714827	5.981752	-18.593437	6.100539
496	3415706130945884416	78.115074	22.897052	NaN	NaN	NaN	635.710510
497	3415706130944329216	78.114951	22.896915	5.714827	5.981752	-18.593437	643.074097
498	3415706130945884416	78.115074	22.897052	NaN	NaN	NaN	13.766609

```

import numpy as np

# A maneira mais eficiente, correta e idiomática (Pythonic) de fazer isso:

df_filtered = df.sort_values(
    by='source_id',                      # Critério de agrupamento (primeira ordenação)
    ascending=True
)

```

```

).sort_values(
    # 1º Critério: Mínimo do valor absoluto da diferença angular
    # Usamos o .abs() diretamente na coluna.
    by=['angular_distance', 'magnitude_difference'],
    key=lambda x: np.abs(x), # APlica np.abs() (módulo) a AMBAS as colunas
    ascending=True,
    kind='stable' # Mantém a ordem da ordenação anterior ('source_id')
).drop_duplicates(
    subset=['source_id'], # Coluna que define o que é 'único'
    keep='first'          # Mantém a linha que ficou no topo após a ordenação
)

print(df_filtered.head())

```

	source_id	ra	dec	parallax	pmra	\
484	156430577302380800	77.026518	30.439698	6.373135	4.030340	
291	3313414750283302400	68.804933	17.430409	6.626949	11.212731	
461	156725353793541504	75.012883	30.018997	6.246707	4.695674	
459	156732977357829888	74.762711	30.049984	6.380320	4.532889	
119	150501362066641664	65.568559	25.819861	6.687325	13.830570	

	pmdec	angular_distance	l	b	magnitude_difference	\
484	-25.723114	0.016792	174.209208	-5.931590		-0.035551
291	-17.987286	0.017127	180.091289	-19.810103		-0.027006
461	-23.607685	0.019499	173.495959	-7.575423		-0.067565
459	-24.640575	0.020147	173.338055	-7.728314		-0.044252
119	-20.020702	0.026451	171.310121	-16.665225		-0.033823

	ruwe	phot_g_mean_mag	bp_rp
484	1.076504	14.621793	3.228045
291	1.461531	14.222324	3.057713
461	1.615348	15.443042	2.167835
459	1.241720	13.402600	2.318685
119	0.978744	17.583151	4.717266

### Exclui valores de RUWE <= 1.4

```

# Aplicar o filtro RUWE, como no artigo
df_cleaned = df_filtered[df_filtered['ruwe'] <= 1.4].copy()

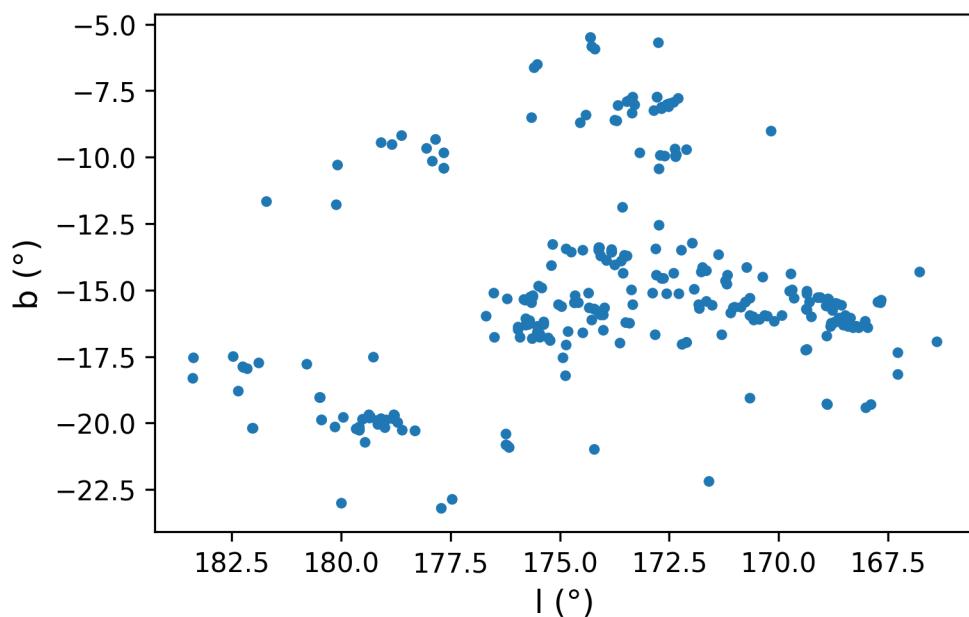
```

```
print(f"Amostra original: {len(df_filtered)} estrelas")
print(f"Amostra limpa (RUWE <= 1.4): {len(df_cleaned)} estrelas")
```

Amostra original: 473 estrelas  
Amostra limpa (RUWE <= 1.4): 285 estrelas

## Plota dados

```
plt.figure()
plt.plot(df_cleaned['l'],df_cleaned['b'], '.')
plt.gca().invert_xaxis()
plt.xlabel('l (°)', fontsize=12)
plt.ylabel('b (°)', fontsize=12)
plt.show()
```



### 3. Clustering

#### K-means

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score, silhouette_samples

# --- 1. PREPARAÇÃO DOS DADOS ---
# (Assumindo que 'df_cleaned' existe da célula anterior)

features = ['ra', 'dec', 'parallax', 'pmra', 'pmdec']
X = df_cleaned[features]

# Normaliza os dados
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print("Iniciando análise K-Means com dados NORMALIZADOS...")

# --- 2. ANÁLISE K-MEANS (A parte que faltava) ---

range_n_clusters = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
avg_silhouettes = []

for n_clusters in range_n_clusters:
    clusterer = KMeans(n_clusters=n_clusters, random_state=10, n_init=10)

    # Usa os dados normalizados (X_scaled)
    cluster_labels_loop = clusterer.fit_predict(X_scaled)

    # Usa os dados normalizados (X_scaled)
    silhouette_avg = silhouette_score(X_scaled, cluster_labels_loop)
    print(
        "For n_clusters =",
        n_clusters,
        "The average silhouette_score is :",
    )
```

```

        silhouette_avg,
    )
avg_silhouettes.append(silhouette_avg)

# Encontra o melhor cluster
best_cluster_index = np.argmax(avg_silhouettes)
best_n_clusters = range_n_clusters[best_cluster_index]
best_silhouette_score = avg_silhouettes[best_cluster_index]

print(f"highest value = {best_silhouette_score}, n_clusters = {best_n_clusters}")

# Roda o K-Means final com o melhor n_clusters
clusterer = KMeans(n_clusters=best_n_clusters, random_state=10, n_init=10)
cluster_labels = clusterer.fit_predict(X_scaled) # Esta é a variável que o plot usa

# Adiciona os rótulos ao DataFrame ( bom para outras análises)
df_cleaned['cluster_kmeans'] = cluster_labels

# --- 3. PLOTAGEM (Dividida em duas figuras) ---

# --- Gráfico 1: Pontuação de Silhueta ---
plt.figure(figsize=(10, 6)) # Define um tamanho de figura mais padrão
plt.plot(range_n_clusters, avg_silhouettes, '-', marker='o')
plt.xlabel('k (Número de Clusters)', fontsize=12)
plt.ylabel('Average Silhouette Score', fontsize=12)
plt.title('Método da Silhueta (com dados Normalizados)')
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()

# --- Gráfico 2: Mapa de Clusters (COM LEGENDA) ---
fig, ax = plt.subplots(figsize=(12, 10)) # Aumentei um pouco para caber a legenda

# Encontrar os labels únicos
unique_labels = np.unique(cluster_labels)
n_clusters = len(unique_labels)

# Criar uma paleta de cores
colors = cm.rainbow(np.linspace(0, 1, n_clusters))

# Iterar sobre cada cluster e plotá-lo separadamente
for k, col in zip(unique_labels, colors):

```

```

# Criar uma máscara para selecionar apenas as estrelas deste cluster
mask = (cluster_labels == k)
cluster_data = df_cleaned[mask]

# Plotar este cluster com uma cor e um 'label'
ax.scatter(
    cluster_data['l'],
    cluster_data['b'],
    c=[col], # A cor para este cluster
    s=50,    # Tamanho do ponto
    label=f'Cluster {k}', # O texto que aparecerá na legenda
    edgecolor='black',
    linewidths=0.5
)

# Inverte o eixo x (aplicado ao 'ax' correto)
ax.invert_xaxis()

# Configura os rótulos e título
ax.set_xlabel('l (°)', fontsize=12)
ax.set_ylabel('b (°)', fontsize=12)
ax.set_title(f'Clusters K-Means ({best_n_clusters} clusters)')
ax.grid(True, linestyle='--', alpha=0.6)

# --- COMANDO PARA ADICIONAR A LEGENDA ---
ax.legend(loc='best', title="Clusters") # 'loc='best'' encontra o melhor lugar

plt.show()

```

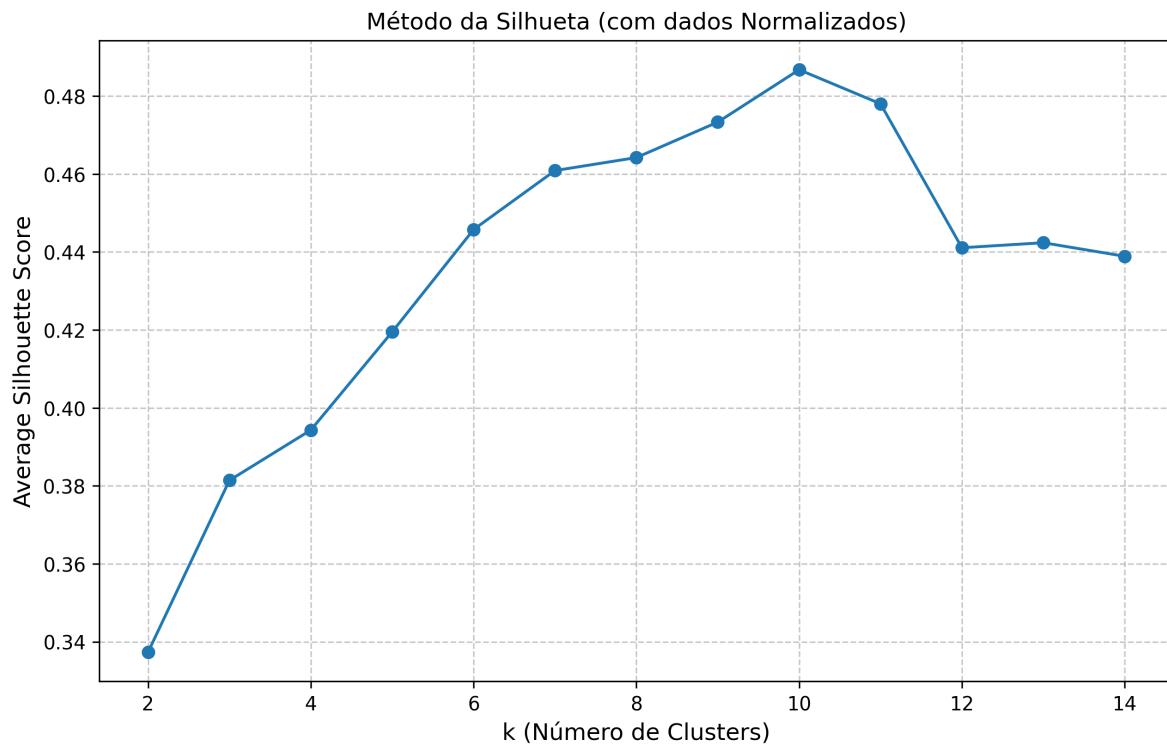
Iniciando análise K-Means com dados NORMALIZADOS...

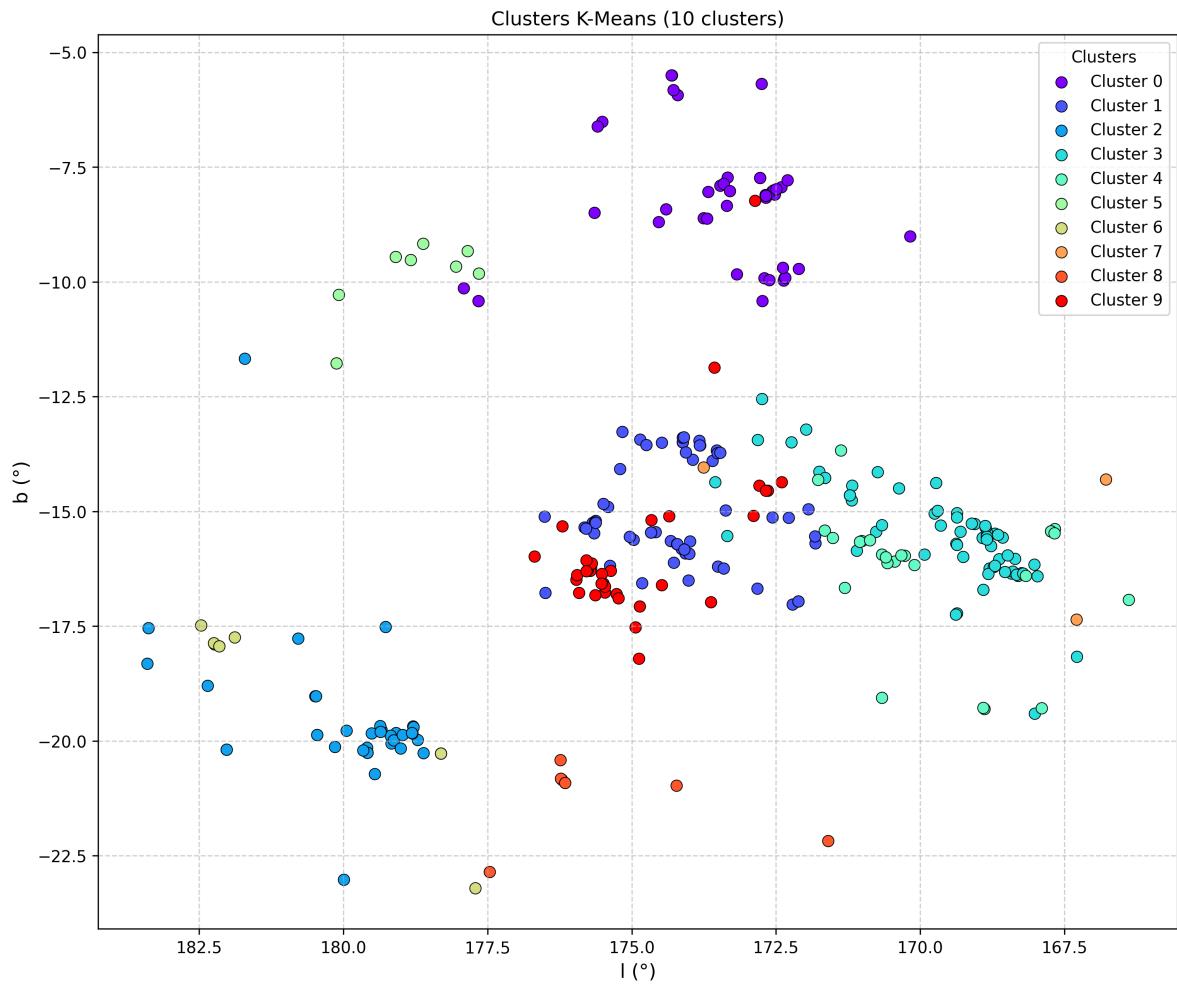
```

For n_clusters = 2 The average silhouette_score is : 0.3373296702417714
For n_clusters = 3 The average silhouette_score is : 0.3814150328040933
For n_clusters = 4 The average silhouette_score is : 0.3943169562212462
For n_clusters = 5 The average silhouette_score is : 0.41950995781089334
For n_clusters = 6 The average silhouette_score is : 0.4457576640466933
For n_clusters = 7 The average silhouette_score is : 0.46089055065593015
For n_clusters = 8 The average silhouette_score is : 0.46422409129822173
For n_clusters = 9 The average silhouette_score is : 0.47332121201209165
For n_clusters = 10 The average silhouette_score is : 0.48676031361765937
For n_clusters = 11 The average silhouette_score is : 0.4780272143713448
For n_clusters = 12 The average silhouette_score is : 0.44109201520643415

```

```
For n_clusters = 13 The average silhouette_score is : 0.4423785601516473
For n_clusters = 14 The average silhouette_score is : 0.4389041185968071
highest value = 0.48676031361765937, n_clusters = 10
```





## DBSCAN

```

import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import NearestNeighbors

# Assumindo que 'df_cleaned' é o seu DataFrame com o filtro RUWE já aplicado

# 1. Selecionar os dados para o clustering

```

```

features = ['ra', 'dec', 'parallax', 'pmra', 'pmdec']
X = df_cleaned[features]

# 2. Normalizar os dados (O PASSO MAIS IMPORTANTE)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. ESCOLHER OS PARÂMETROS PARA OS DADOS NORMALIZADOS
# O eps=0.5 não faz mais sentido. Precisamos encontrar um novo valor.

# --- Como escolher min_samples ---
# Uma boa regra é usar min_samples >= 2 * n_dimensões
# Para 5 dimensões, um bom começo é min_samples = 10 a 15.
min_samples = 10

# --- Como escolher eps (Método do "Joelho" ou K-distance) ---
# Vamos plotar a distância de cada ponto para o seu 15º vizinho mais próximo
# e procurar por um "joelho" (cotovelo) no gráfico.
print("Calculando o gráfico k-distance para encontrar o melhor 'eps'...")
neighbors = NearestNeighbors(n_neighbors=min_samples)
neighbors_fit = neighbors.fit(X_scaled)
distances, indices = neighbors_fit.kneighbors(X_scaled)

# Pega a distância para o k-ésimo vizinho (k = min_samples) e ordena
distances = np.sort(distances[:, min_samples-1], axis=0)

# Plota o gráfico
plt.figure(figsize=(10, 6))
plt.plot(distances)
plt.title('Gráfico K-distance para Escolha do Epsilon (eps)')
plt.xlabel('Pontos (ordenados por distância)')
plt.ylabel(f'Distância para o {min_samples}º Vizinho Mais Próximo')
plt.grid(True)
plt.show()

# OLHE O GRÁFICO! Onde a curva começa a subir abruptamente?
# Esse é o seu valor ideal para 'eps'. Digamos que seja 1.2, por exemplo.
# Você precisará ajustar este valor com base no seu gráfico.
eps_ideal = 1 # <-- AJUSTE ESTE VALOR COM BASE NO SEU GRÁFICO

# 4. Rodar o DBSCAN com os dados normalizados e parâmetros corretos
print(f"\nRodando DBSCAN com eps={eps_ideal} e min_samples={min_samples}")

```

```

db = DBSCAN(eps=eps_ideal, min_samples=min_samples).fit(X_scaled)
labels = db.labels_

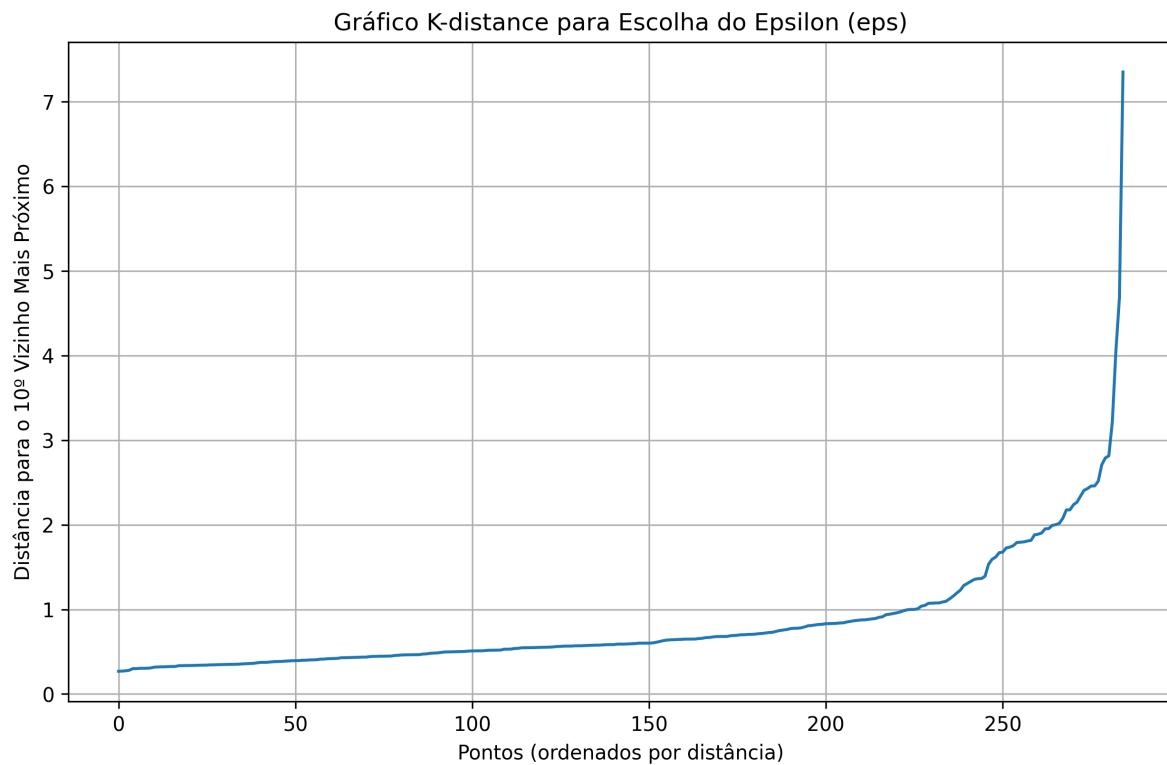
# 5. Analisar os resultados
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

print(f"\nNúmero estimado de clusters: {n_clusters_}")
print(f"Número estimado de pontos de ruído: {n_noise_}")

# Adicionar os labels dos clusters de volta ao DataFrame para análise
df_cleaned['cluster_dbSCAN'] = labels
print("\nDistribuição de estrelas por cluster:")
print(df_cleaned['cluster_dbSCAN'].value_counts())

```

Calculando o gráfico k-distance para encontrar o melhor 'eps'...



```
Rodando DBSCAN com eps=1 e min_samples=10
```

```
Número estimado de clusters: 4
Número estimado de pontos de ruído: 44
```

```
Distribuição de estrelas por cluster:
```

```
cluster_dbSCAN
```

```
3    113
2     50
-1    44
0     41
1     37
```

```
Name: count, dtype: int64
```

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

# Assuma que 'df_cleaned' é o seu DataFrame final com os dados e os rótulos do DBSCAN
# Ex: df_cleaned['cluster_dbSCAN'] contém os resultados do DBSCAN

# Pega os rótulos e encontra quantos clusters únicos existem (sem contar o ruído)
labels = df_cleaned['cluster_dbSCAN']
unique_labels = set(labels)
n_clusters = len(unique_labels) - (1 if -1 in labels else 0)

# Cria uma paleta de cores para os clusters
# Usamos cm.rainbow para gerar cores distintas
colors = cm.rainbow(np.linspace(0, 1, len(unique_labels)))

# Inicia a figura do gráfico
fig, ax = plt.subplots(figsize=(12, 10))

# Itera sobre cada cluster único e o plota com uma cor diferente
for k, col in zip(unique_labels, colors):
    # --- Estilo para os pontos de ruído (label = -1) ---
    if k == -1:
        # Ruído será plotado como pequenos 'x' pretos e semi-transparentes
        col = [0, 0, 0, 0.3] # RGBA: Preto com 30% de opacidade
        marker = 'x'
        size = 20
```

```

    plot_label = f'Ruído ({list(labels).count(-1)} estrelas)'
    # --- Estilo para os clusters reais (label >= 0) ---
    else:
        # Clusters serão plotados como círculos coloridos
        marker = 'o'
        size = 50
        # --- CORREÇÃO DE LABEL ---
        plot_label = f'Cluster {k}' # Mais claro que k+1

    # Cria uma máscara para selecionar apenas as estrelas do cluster atual
    class_member_mask = (labels == k)

    # Seleciona os dados do cluster atual
    xy = df_cleaned[class_member_mask]

    # Plota os pontos no gráfico
    ax.scatter(xy['l'], xy['b'], s=size, c=[col], marker=marker, label=plot_label, edgecolor='black')

    # --- Finalização e customização do gráfico ---
    # --- CORREÇÃO NO TÍTULO ---
    ax.set_title(f'DBSCAN Região de Taurus ({n_clusters-1} clusters)', fontsize=14) # Removido o
    ax.set_xlabel('l(°)', fontsize=12)
    ax.set_ylabel('b(°)', fontsize=12)

    # Inverte o eixo X, como é comum em mapas celestes
    ax.invert_xaxis()

    # --- LINHA DESCOMENTADA ---
    ax.legend(loc='best', title="Clusters") # 'loc='best'' encontra o melhor lugar

    ax.grid(True, linestyle='--', alpha=0.6)
    plt.show()

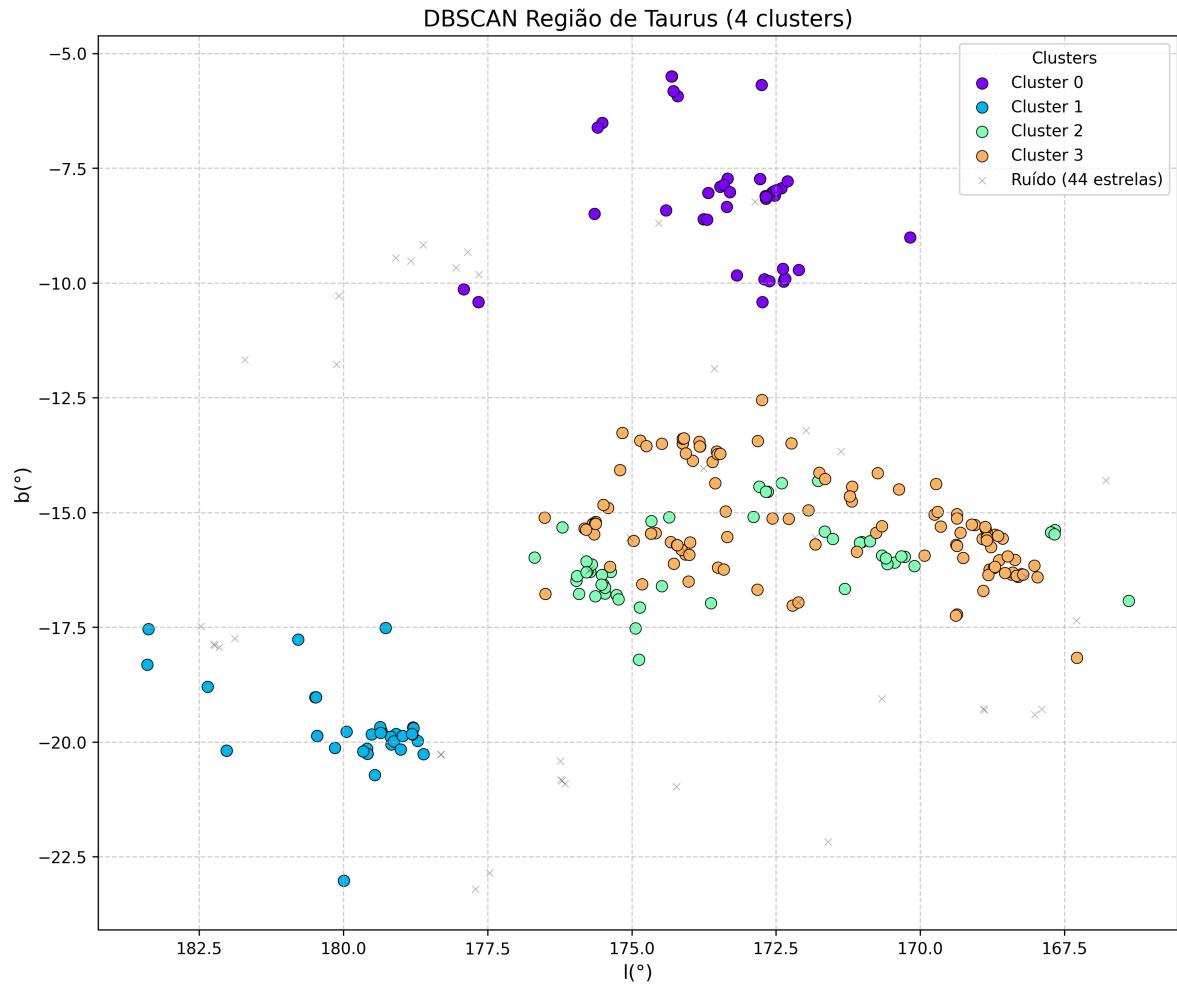
```

C:\Users\dedet\AppData\Local\Temp\ipykernel\_20764\155810959.py:45: UserWarning: You passed a

```

ax.scatter(xy['l'], xy['b'], s=size, c=[col], marker=marker, label=plot_label, edgecolor='black')

```



## HDBSCAN

```

features = ['ra', 'dec', 'parallax', 'pmra', 'pmdec']
X = df_cleaned[features]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 2. Implementar o HDBSCAN do Scikit-learn
print("Rodando HDBSCAN (do Scikit-learn)...") 
clusterer = HDBSCAN(
    min_cluster_size=10, # <-- O seu parâmetro principal
)

```

```

    min_samples=10           # <-- Parâmetro de ajuste fino (opcional)
)

clusterer.fit(X_scaled)

# 3. Analisar os resultados (Igual)
labels = clusterer.labels_

n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

print(f"\nNúmero estimado de clusters: {n_clusters_}")
print(f"Número estimado de pontos de ruído: {n_noise_}")

# Adicionar os labels dos clusters de volta ao DataFrame
df_cleaned['cluster_hdbscan'] = labels
print("\nDistribuição de estrelas por cluster:")
print(df_cleaned['cluster_hdbscan'].value_counts())

```

Rodando HDBSCAN (do Scikit-learn)...

Número estimado de clusters: 5  
 Número estimado de pontos de ruído: 52

Distribuição de estrelas por cluster:

```

cluster_hdbscan
4      57
-1     52
3      52
2      48
0      39
1      37
Name: count, dtype: int64

```

```

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

# Assuma que 'df_cleaned' é o seu DataFrame com a coluna 'cluster_hdbscan'

```

```

# Pega os rótulos e encontra quantos clusters únicos existem (sem contar o ruído)
labels = df_cleaned['cluster_hdbscan']
unique_labels = set(labels)
n_clusters = len(unique_labels) - (1 if -1 in labels else 0)

# Cria uma paleta de cores para os clusters
colors = cm.rainbow(np.linspace(0, 1, len(unique_labels)))

# Inicia a figura do gráfico
fig, ax = plt.subplots(figsize=(12, 10))

# Itera sobre cada cluster único e o plota com uma cor diferente
for k, col in zip(unique_labels, colors):
    # --- Estilo para os pontos de ruído (label = -1) ---
    if k == -1:
        col = [0, 0, 0, 0.3] # RGBA: Preto com 30% de opacidade
        marker = 'x'
        size = 20
        plot_label = f'Ruído ({list(labels).count(-1)} estrelas)'
    # --- Estilo para os clusters reais (label >= 0) ---
    else:
        marker = 'o'
        size = 50
        plot_label = f'Cluster {k}'

    # Cria uma máscara para selecionar apenas as estrelas do cluster atual
    class_member_mask = (labels == k)

    # Seleciona os dados do cluster atual
    xy = df_cleaned[class_member_mask]

    # Plota os pontos no gráfico
    ax.scatter(xy['l'], xy['b'], s=size, c=[col], marker=marker, label=plot_label, edgecolor='black')

# --- Finalização e customização do gráfico ---
# --- CORREÇÃO NO TÍTULO ---
ax.set_title(f'HDBSCAN Região de Taurus ({n_clusters-1} clusters)', fontsize=14)
ax.set_xlabel('l(°)', fontsize=12)
ax.set_ylabel('b(°)', fontsize=12)
# Inverte o eixo X, como é comum em mapas celestes
ax.invert_xaxis()

```

```

# --- LINHA DESCOMENTADA ---
ax.legend(loc='best', title="Clusters") # 'loc='best'' encontra o melhor lugar

ax.grid(True, linestyle='--', alpha=0.6)
plt.show()

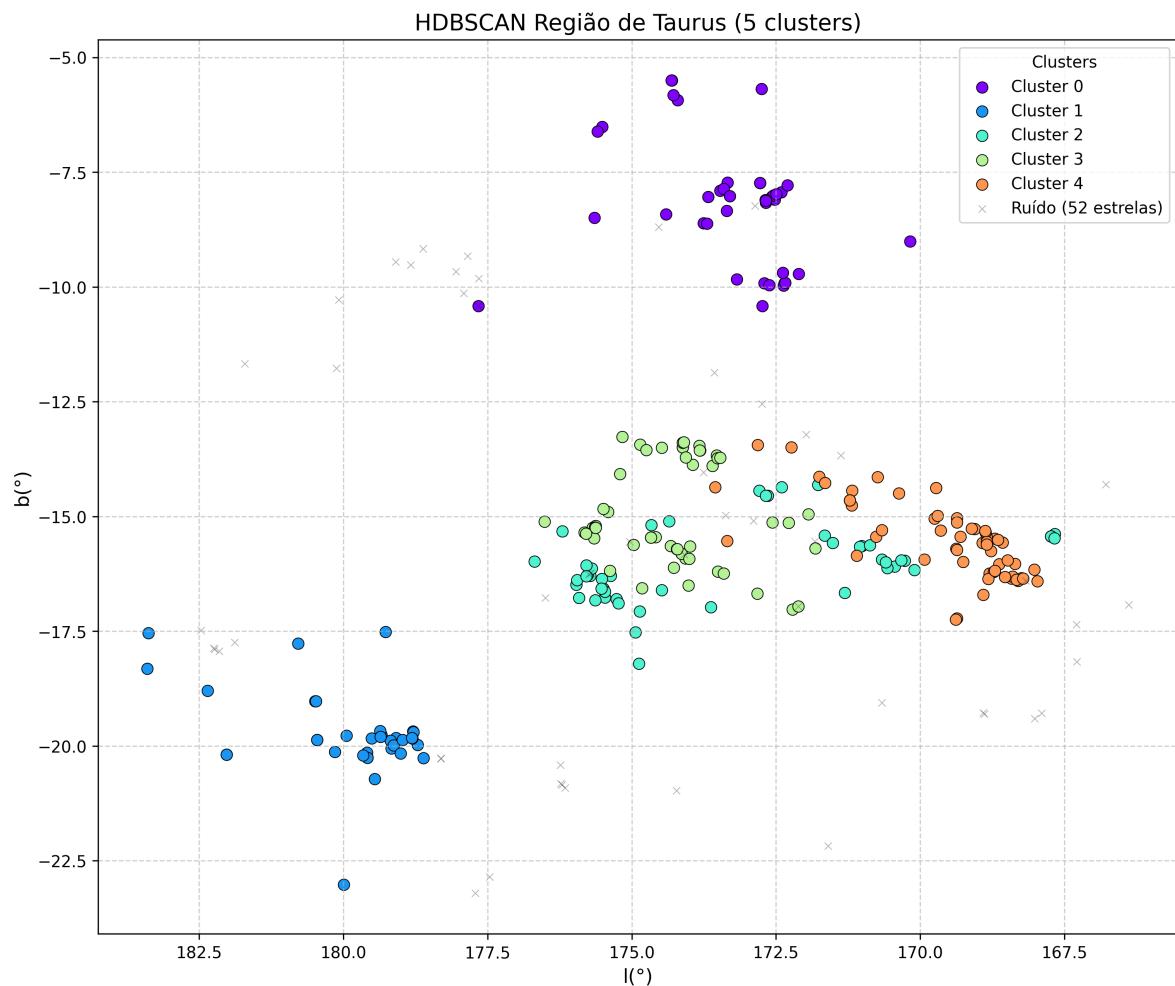
```

C:\Users\dedet\AppData\Local\Temp\ipykernel\_20764\2685425413.py:40: UserWarning: You passed a

```

ax.scatter(xy['l'], xy['b'], s=size, c=[col], marker=marker, label=plot_label, edgecolor='black')

```



## CMDs

```
import matplotlib.pyplot as plt
import pandas as pd
import math

# --- Configuração dos Dados (Igual ao seu código original) ---
labels = df_cleaned['cluster_hdbscan']
noise_data = df_cleaned[labels == -1]
cluster_ids = sorted([c for c in labels.unique() if c >= 0])
n_clusters = len(cluster_ids)

print(f"Gerando mosaico para {n_clusters} clusters...")

# --- Configuração do Grid (Mosaico) ---
# Define quantas colunas você quer (ex: 3 ou 4)
n_cols = 3
# Calcula quantas linhas são necessárias (arredonda para cima)
n_rows = math.ceil(n_clusters / n_cols)

# Cria a figura gigante e os eixos (axs)
# figsize ajustado: largura fixa, altura cresce com o número de linhas
fig, axs = plt.subplots(n_rows, n_cols, figsize=(15, 5 * n_rows), constrained_layout=True)

# Transforma a matriz de eixos em uma lista linear para facilitar o loop
# (Trata o caso de haver apenas 1 cluster para evitar erro)
if n_clusters > 1:
    axs = axs.flatten()
else:
    axs = [axs]

# --- Loop de Plotagem ---
for i, ax in enumerate(axs):
    # Verifica se ainda temos clusters para plotar
    if i < n_clusters:
        cluster_id = cluster_ids[i]
        cluster_data = df_cleaned[labels == cluster_id]

        # 1. Plota o "campo" de ruído
        # DICAS: rasterized=True ajuda a deixar o PDF/SVG mais leve se tiver muitos pontos
        ax.scatter(
            noise_data['bp_rp'],
```

```

        noise_data['phot_g_mean_mag'],
        c='gray',
        s=1,
        alpha=0.05, # Reduzi a opacidade pois o gráfico fica menor
        label='Ruído' if i == 0 else "", # Legenda só no primeiro para não poluir
        rasterized=True
    )

# 2. Plota os membros do cluster
ax.scatter(
    cluster_data['bp_rp'],
    cluster_data['phot_g_mean_mag'],
    c='orange',
    s=10,
    label=f'ID {cluster_id} (N={len(cluster_data)})',
    edgecolor='black',
    linewidth=0.3
)

# --- Customização Específica do Eixo (ax) ---
ax.set_title(f'Cluster {cluster_id}', fontsize=12, fontweight='bold')
ax.set_xlabel('Cor ($G_{BP} - G_{RP}$)', fontsize=10)
ax.set_ylabel('Magnitude G', fontsize=10)

ax.invert_yaxis() # Inverte Y (Fundamental para CMD)
ax.grid(True, linestyle='--', alpha=0.5)
ax.legend(loc='best', fontsize='small', frameon=True)

else:
    # Se acabaram os clusters mas sobraram "quadrinhos" no grid, desligue-os
    ax.axis('off')

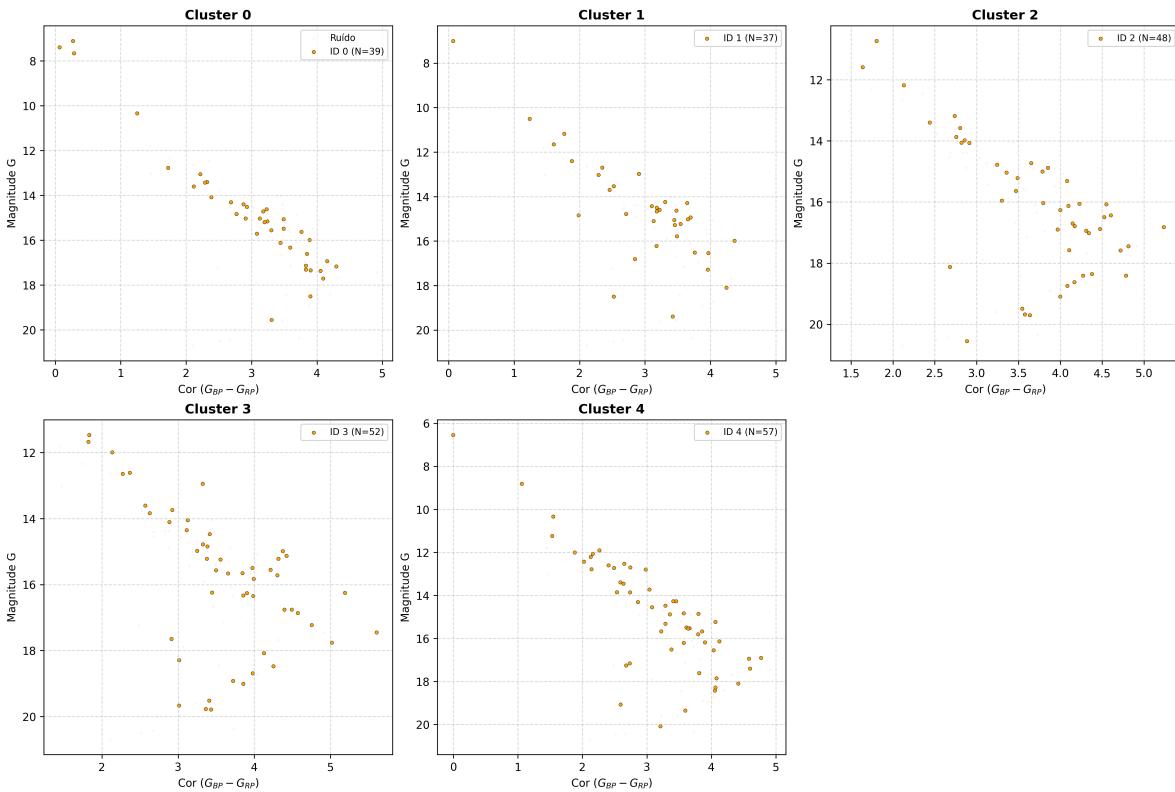
# Título geral da figura (opcional)
fig.suptitle(f'Análise de Clusters HDBSCAN - Total: {n_clusters}', fontsize=20, y=1.06)

plt.show()

```

Gerando mosaico para 5 clusters...

### Análise de Clusters HDBSCAN - Total: 5



```

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

# --- Configuração dos Dados ---
# Assumindo que 'df_cleaned' é o seu DataFrame
labels = df_cleaned['cluster_dbSCAN'] # <-- MUDANÇA AQUI
noise_data = df_cleaned[labels == -1]
cluster_ids = sorted([c for c in labels.unique() if c >= 0])
n_clusters = len(cluster_ids)

print(f"Gerando mosaico de CMDs para {n_clusters} clusters do DBSCAN...")

# --- Configuração do Grid (Mosaico) ---
n_cols = 3 # 3 colunas
n_rows = math.ceil(n_clusters / n_cols) # Calcula as linhas

```

```

fig, axs = plt.subplots(n_rows, n_cols, figsize=(15, 5 * n_rows), constrained_layout=True)

if n_clusters > 1:
    axs = axs.flatten()
else:
    axs = [axs]

# --- Loop de Plotagem ---
for i, ax in enumerate(axs):
    if i < n_clusters:
        cluster_id = cluster_ids[i]
        cluster_data = df_cleaned[labels == cluster_id]

        # 1. Plota o "campo" de ruído
        ax.scatter(
            noise_data['bp_rp'],
            noise_data['phot_g_mean_mag'],
            c='gray',
            s=1,
            alpha=0.05,
            label='Ruído' if i == 0 else "",
            rasterized=True
        )

        # 2. Plota os membros do cluster
        ax.scatter(
            cluster_data['bp_rp'],
            cluster_data['phot_g_mean_mag'],
            c='orange',
            s=10,
            label=f'ID {cluster_id} (N={len(cluster_data)})',
            edgecolor='black',
            linewidth=0.3
        )

        # Customização do eixo
        ax.set_title(f'Cluster {cluster_id}', fontsize=12, fontweight='bold')
        ax.set_xlabel('Cor ($G_{BP} - G_{RP}$)', fontsize=10)
        ax.set_ylabel('Magnitude G', fontsize=10)
        ax.invert_yaxis()
        ax.grid(True, linestyle='--', alpha=0.5)
        ax.legend(loc='best', fontsize='small', frameon=True)

```

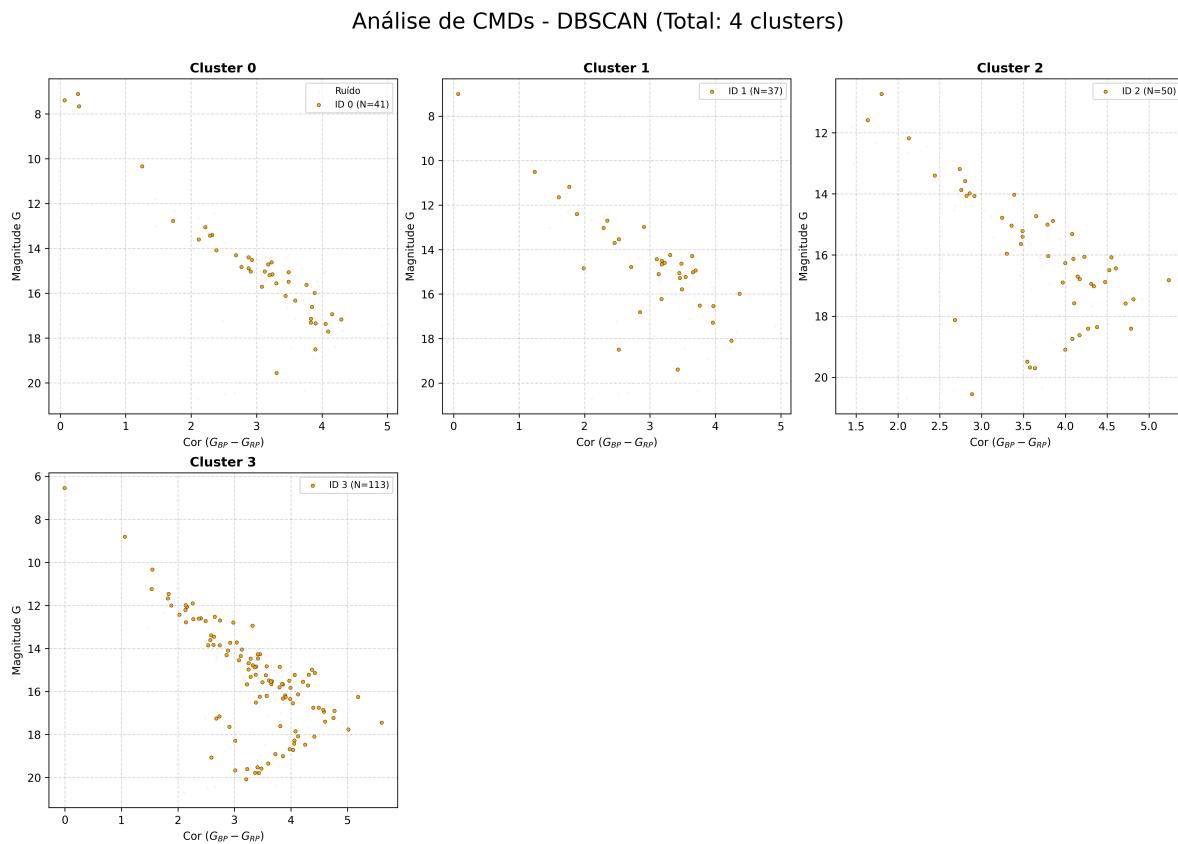
```

else:
    # Desliga eixos extras
    ax.axis('off')

fig.suptitle(f'Análise de CMDs - DBSCAN (Total: {n_clusters} clusters)', fontsize=20, y=1.06)
plt.show()

```

Gerando mosaico de CMDs para 4 clusters do DBSCAN...



```

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import matplotlib.cm as cm
import math
from sklearn.cluster import KMeans

```

```

# --- 1. GARANTIR QUE O K-MEANS CORRETO FOI EXECUTADO ---
# (Este bloco já está no seu notebook)

# features = ['ra', 'dec', 'parallax', 'pmra', 'pmdec']
# X = df_cleaned[features]
# scaler = StandardScaler()
# X_scaled = scaler.fit_transform(X)
# n_clusters_kmeans = 8
# kmeans = KMeans(n_clusters=n_clusters_kmeans, random_state=10, n_init=10)
# kmeans_labels = kmeans.fit_predict(X_scaled)
# df_cleaned['cluster_kmeans'] = kmeans_labels

# --- 2. GERAR O MOSAICO DE PLOTS (Seu código da pág. 32) ---

labels = df_cleaned['cluster_kmeans']
cluster_ids = sorted(labels.unique()) # K-Means não tem -1
n_clusters = len(cluster_ids)

print(f"Gerando mosaico de CMDs para {n_clusters} clusters do K-Means...")

# --- Configuração do Grid (Mosaico) ---
n_cols = 5
n_rows = math.ceil(n_clusters / n_cols)
fig, axs = plt.subplots(n_rows, n_cols, figsize=(15, 3 * n_rows), constrained_layout=True)
axs = axs.flatten()

# --- Loop de Plotagem ---
for i, ax in enumerate(axs):
    if i < n_clusters:
        cluster_id = cluster_ids[i]

        # Dados deste cluster (para plotar em laranja)
        cluster_data = df_cleaned[labels == cluster_id]

        # Dados de "fundo" (TODOS os outros clusters)
        background_data = df_cleaned[labels != cluster_id]

        # 1. Plota o "fundo" (outros clusters)
        ax.scatter(
            background_data['bp_rp'],
            background_data['phot_g_mean_mag'],
            c='gray',

```

```

        s=1,
        alpha=0.05,
        label='Outros Clusters' if i == 0 else "",
        rasterized=True
    )

# 2. Plota os membros do cluster
ax.scatter(
    cluster_data['bp_rp'],
    cluster_data['phot_g_mean_mag'],
    c='orange',
    s=10,
    label=f'ID {cluster_id} (N={len(cluster_data)})',
    edgecolor='black',
    linewidth=0.3
)

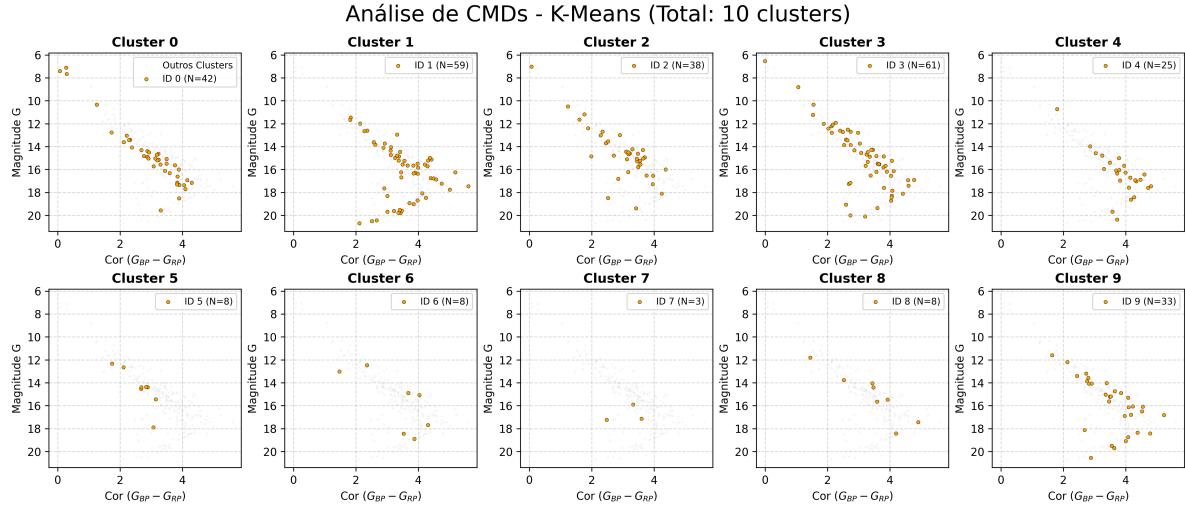
# Customização do eixo
ax.set_title(f'Cluster {cluster_id}', fontsize=12, fontweight='bold')
ax.set_xlabel('Cor ($G_{BP} - G_{RP}$)', fontsize=10)
ax.set_ylabel('Magnitude G', fontsize=10)
ax.invert_yaxis()
ax.grid(True, linestyle='--', alpha=0.5)
ax.legend(loc='best', fontsize='small', frameon=True)

else:
    # Desliga eixos extras
    ax.axis('off')

fig.suptitle(f'Análise de CMDs - K-Means (Total: {n_clusters} clusters)', fontsize=20, y=1.00)
plt.show()

```

Gerando mosaico de CMDs para 10 clusters do K-Means...



## pmra x pmdec

### 1. K-means

```

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

# --- 1. CONFIGURAÇÃO DOS DADOS ---
# Assumindo que 'df_cleaned' é o seu DataFrame
labels = df_cleaned['cluster_kmeans']
cluster_ids = sorted(labels.unique())
n_clusters = len(cluster_ids)

print(f"Gerando mosaico de Movimento Próprio para {n_clusters} clusters do K-Means...")

# --- 2. CONFIGURAÇÃO DO GRID ---
n_cols = 5
n_rows = math.ceil(n_clusters / n_cols)
fig, axs = plt.subplots(n_rows, n_cols, figsize=(15, 3 * n_rows), constrained_layout=True)
axs = axs.flatten()

# --- 3. LOOP DE PLOTAGEM ---

```

```

for i, ax in enumerate(axes):
    if i < n_clusters:
        cluster_id = cluster_ids[i]

        # Dados deste cluster (para plotar em laranja)
        cluster_data = df_cleaned[labels == cluster_id]

        # Dados de "fundo" (TODOS os outros clusters)
        background_data = df_cleaned[labels != cluster_id]

        # 1. Plota o "fundo" (outros clusters)
        ax.scatter(
            background_data['pmra'],
            background_data['pmdec'],
            c='gray',
            s=1,
            alpha=0.05,
            label='Outros Clusters' if i == 0 else '',
            rasterized=True
        )

        # 2. Plota os membros do cluster
        ax.scatter(
            cluster_data['pmra'],
            cluster_data['pmdec'],
            c='orange',
            s=10,
            label=f'ID {cluster_id} (N={len(cluster_data)})',
            edgecolor='black',
            linewidth=0.3
        )

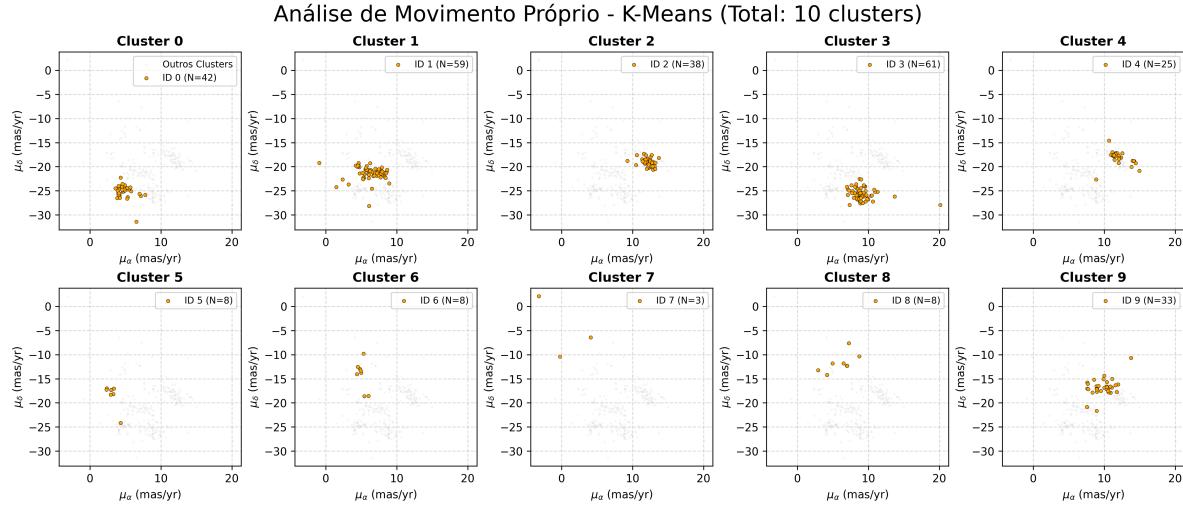
        # Customização do eixo
        ax.set_title(f'Cluster {cluster_id}', fontsize=12, fontweight='bold')
        ax.set_xlabel('$\mu_\alpha$ (mas/yr)', fontsize=10) # Label de PMRA
        ax.set_ylabel('$\mu_\delta$ (mas/yr)', fontsize=10) # Label de PMDEC
        ax.grid(True, linestyle='--', alpha=0.5)
        ax.legend(loc='best', fontsize='small', frameon=True)

    else:
        # Desliga eixos extras
        ax.axis('off')

```

```
fig.suptitle(f'Análise de Movimento Próprio - K-Means (Total: {n_clusters} clusters)', fontweight='bold')
plt.show()
```

Gerando mosaico de Movimento Próprio para 10 clusters do K-Means...



## 2. DBSCAN

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import matplotlib.cm as cm
import math

# --- 1. CONFIGURAÇÃO DOS DADOS ---
# Assumindo que 'df_cleaned' é o seu DataFrame
labels = df_cleaned['cluster_dbSCAN'] # <-- MUDANÇA AQUI
noise_data = df_cleaned[labels == -1]
cluster_ids = sorted([c for c in labels.unique() if c >= 0])
n_clusters = len(cluster_ids)

print(f"Gerando mosaico de Movimento Próprio para {n_clusters} clusters do DBSCAN...")

# --- 2. CONFIGURAÇÃO DO GRID ---
n_cols = 3
```

```

n_rows = math.ceil(n_clusters / n_cols)
fig, axs = plt.subplots(n_rows, n_cols, figsize=(15, 5 * n_rows), constrained_layout=True)

if n_clusters > 1:
    axs = axs.flatten()
else:
    axs = [axs]

# --- 3. LOOP DE PLOTAGEM ---
for i, ax in enumerate(axs):
    if i < n_clusters:
        cluster_id = cluster_ids[i]
        cluster_data = df_cleaned[labels == cluster_id]

        # 1. Plota o "campo" de ruído
        ax.scatter(
            noise_data['pmra'],
            noise_data['pmdec'],
            c='gray',
            s=1,
            alpha=0.05,
            label='Ruído' if i == 0 else "",
            rasterized=True
        )

        # 2. Plota os membros do cluster
        ax.scatter(
            cluster_data['pmra'],
            cluster_data['pmdec'],
            c='orange',
            s=10,
            label=f'ID {cluster_id} (N={len(cluster_data)})',
            edgecolor='black',
            linewidth=0.3
        )

        # Customização do eixo
        ax.set_title(f'Cluster {cluster_id}', fontsize=12, fontweight='bold')
        ax.set_xlabel('$\mu_\alpha$ (mas/yr)', fontsize=10)
        ax.set_ylabel('$\mu_\delta$ (mas/yr)', fontsize=10)
        ax.grid(True, linestyle='--', alpha=0.5)
        ax.legend(loc='best', fontsize='small', frameon=True)

```

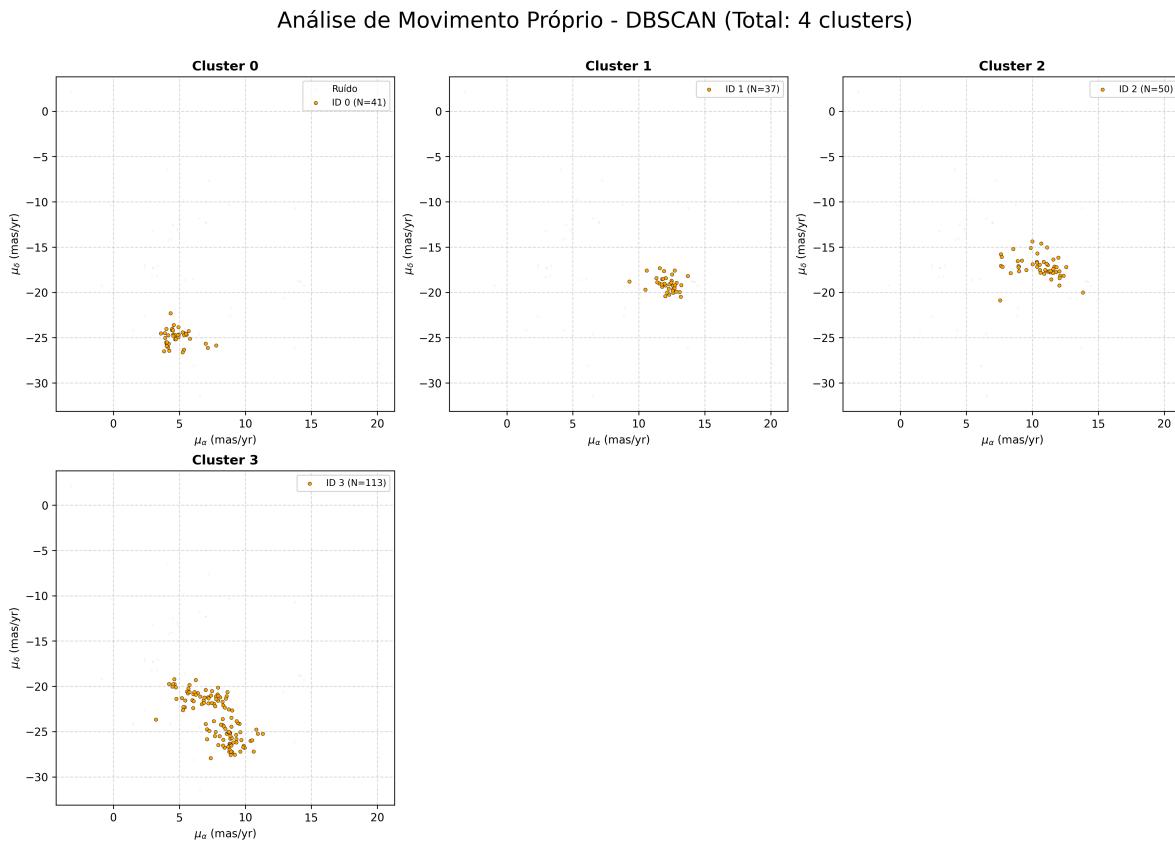
```

else:
    # Desliga eixos extras
    ax.axis('off')

fig.suptitle(f'Análise de Movimento Próprio - DBSCAN (Total: {n_clusters} clusters)', fontsize=14)
plt.show()

```

Gerando mosaico de Movimento Próprio para 4 clusters do DBSCAN...



## HDBSCAN

```

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

```

```

import matplotlib.cm as cm
import math

# --- 1. CONFIGURAÇÃO DOS DADOS ---
# Assumindo que 'df_cleaned' é o seu DataFrame
labels = df_cleaned['cluster_hdbscan'] # <-- MUDANÇA AQUI
noise_data = df_cleaned[labels == -1]
cluster_ids = sorted([c for c in labels.unique() if c >= 0])
n_clusters = len(cluster_ids)

print(f"Gerando mosaico de Movimento Próprio para {n_clusters} clusters do HDBSCAN...")

# --- 2. CONFIGURAÇÃO DO GRID ---
n_cols = 3
n_rows = math.ceil(n_clusters / n_cols)
fig, axs = plt.subplots(n_rows, n_cols, figsize=(15, 5 * n_rows), constrained_layout=True)

if n_clusters > 1:
    axs = axs.flatten()
else:
    axs = [axs]

# --- 3. LOOP DE PLOTAGEM ---
for i, ax in enumerate(axs):
    if i < n_clusters:
        cluster_id = cluster_ids[i]
        cluster_data = df_cleaned[labels == cluster_id]

        # 1. Plota o "campo" de ruído
        ax.scatter(
            noise_data['pmra'],
            noise_data['pmdec'],
            c='gray',
            s=1,
            alpha=0.05,
            label='Ruído' if i == 0 else "",
            rasterized=True
        )

        # 2. Plota os membros do cluster
        ax.scatter(
            cluster_data['pmra'],

```

```

        cluster_data['pmdec'],
        c='orange',
        s=10,
        label=f'ID {cluster_id} (N={len(cluster_data)})',
        edgecolor='black',
        linewidth=0.3
    )

    # Customização do eixo
    ax.set_title(f'Cluster {cluster_id}', fontsize=12, fontweight='bold')
    ax.set_xlabel('$\mu_\alpha$ (mas/ano)', fontsize=10)
    ax.set_ylabel('$\mu_\delta$ (mas/ano)', fontsize=10)
    ax.grid(True, linestyle='--', alpha=0.5)
    ax.legend(loc='best', fontsize='small', frameon=True)

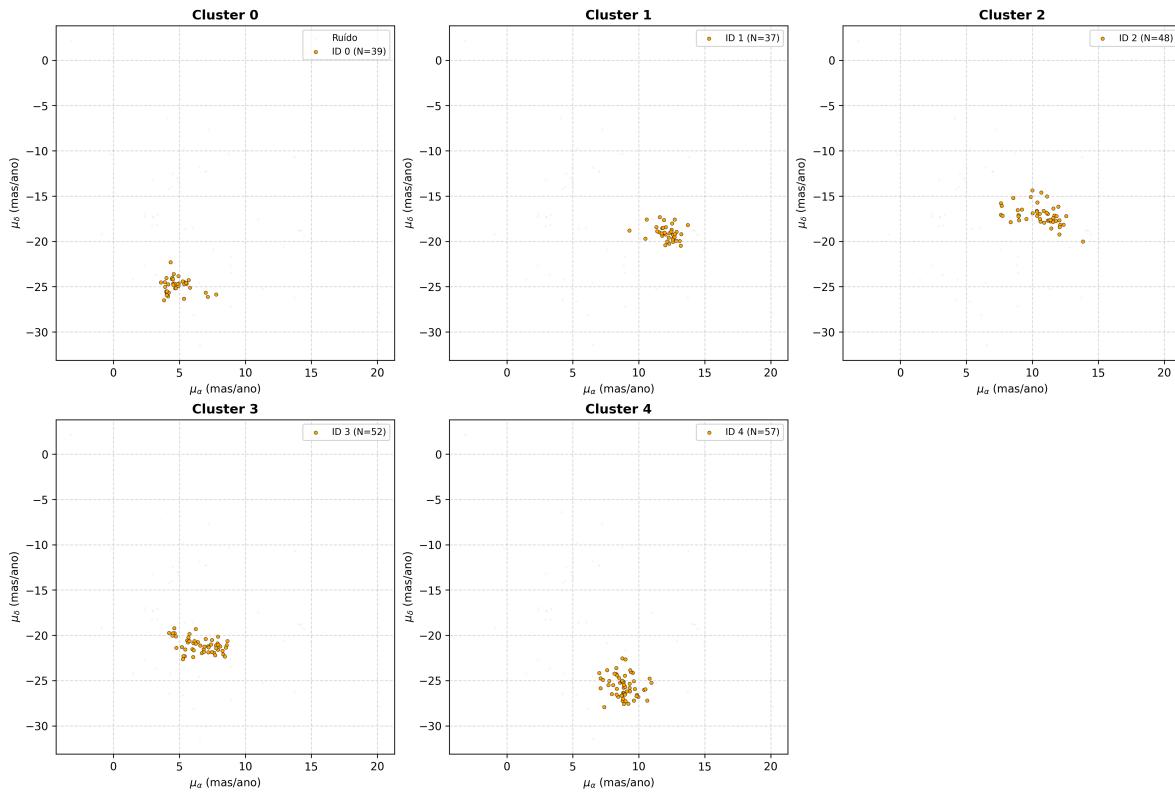
else:
    # Desliga eixos extras
    ax.axis('off')

fig.suptitle(f'Análise de Movimento Próprio - HDBSCAN (Total: {n_clusters} clusters)', font-size=16)
plt.show()

```

Gerando mosaico de Movimento Próprio para 5 clusters do HDBSCAN...

### Análise de Movimento Próprio - HDBSCAN (Total: 5 clusters)



## Paralaxes

### 1. K-means

```

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

# --- 1. CONFIGURAÇÃO DOS DADOS ---
# Assumindo que 'df_cleaned' é o seu DataFrame
labels = df_cleaned['cluster_kmeans']
cluster_ids = sorted(labels.unique())
n_clusters = len(cluster_ids)

```

```

print(f"Gerando mosaico de histogramas de paralaxe para {n_clusters} clusters do K-Means...")

# --- 2. CONFIGURAÇÃO DO GRID ---
n_cols = 4
n_rows = math.ceil(n_clusters / n_cols)
fig, axs = plt.subplots(n_rows, n_cols, figsize=(15, 3 * n_rows), constrained_layout=True)
axs = axs.flatten()

# --- 3. LOOP DE PLOTAGEM ---
for i, ax in enumerate(axs):
    if i < n_clusters:
        cluster_id = cluster_ids[i]
        cluster_data = df_cleaned[labels == cluster_id]

        # Plota o histograma
        ax.hist(cluster_data['parallax'], bins=10, edgecolor='black', alpha=0.7)

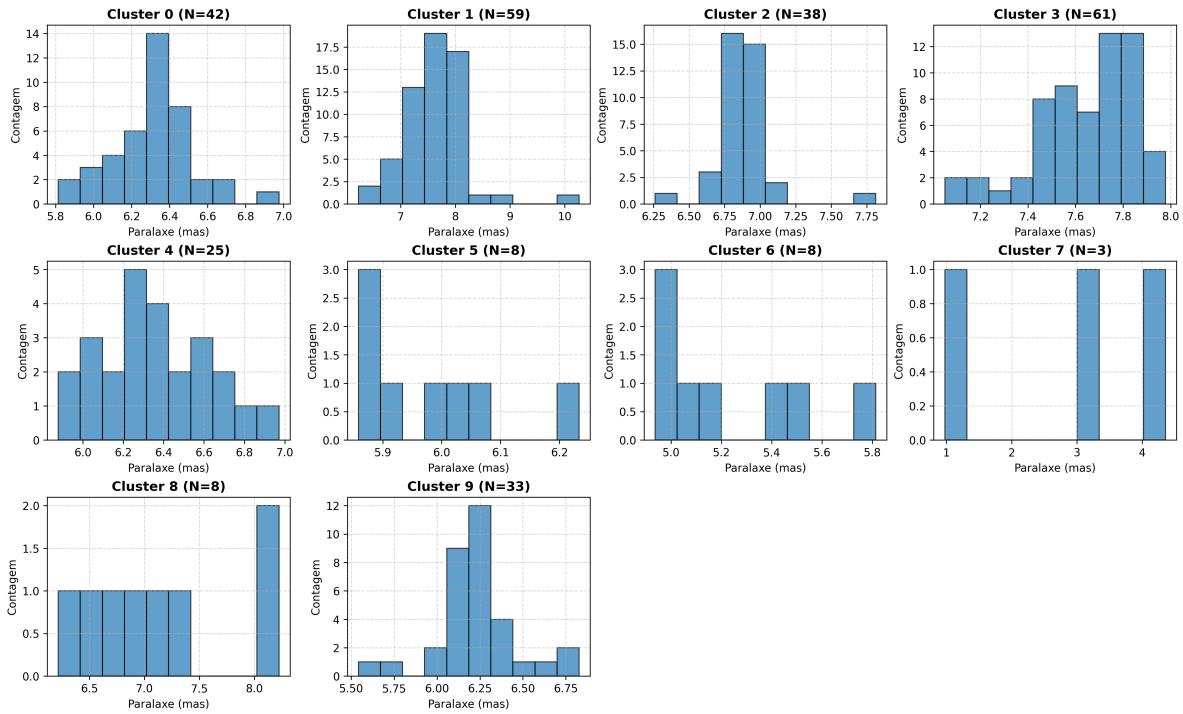
        # Customização
        ax.set_title(f'Cluster {cluster_id} (N={len(cluster_data)})', fontsize=12, fontweight='bold')
        ax.set_xlabel('Paralaxe (mas)', fontsize=10)
        ax.set_ylabel('Contagem', fontsize=10)
        ax.grid(True, linestyle='--', alpha=0.5)
    else:
        ax.axis('off')

fig.suptitle(f'Histogramas de Paralaxe - K-Means (Total: {n_clusters} clusters)', fontsize=20)
plt.show()

```

Gerando mosaico de histogramas de paralaxe para 10 clusters do K-Means...

Histogramas de Paralaxe - K-Means (Total: 10 clusters)



## 2. DBSCAN

```

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

# --- 1. CONFIGURAÇÃO DOS DADOS ---
labels = df_cleaned['cluster_dbSCAN']
cluster_ids = sorted(labels.unique()) # Inclui o -1 (ruído)
n_clusters = len(cluster_ids)

print(f"Gerando mosaico de histogramas de paralaxe para DBSCAN...")

# --- 2. CONFIGURAÇÃO DO GRID ---
n_cols = 4
n_rows = math.ceil(n_clusters / n_cols)
fig, axs = plt.subplots(n_rows, n_cols, figsize=(15, 4 * n_rows), constrained_layout=True)

```

```

axs = axs.flatten()

# --- 3. LOOP DE PLOTAGEM ---
for i, ax in enumerate(axs):
    if i < n_clusters:
        cluster_id = cluster_ids[i]
        cluster_data = df_cleaned[labels == cluster_id]

        plot_title = f'Cluster {cluster_id} (N={len(cluster_data)})'
        color = 'blue'

        if cluster_id == -1:
            plot_title = f'Ruído (N={len(cluster_data)})'
            color = 'gray'

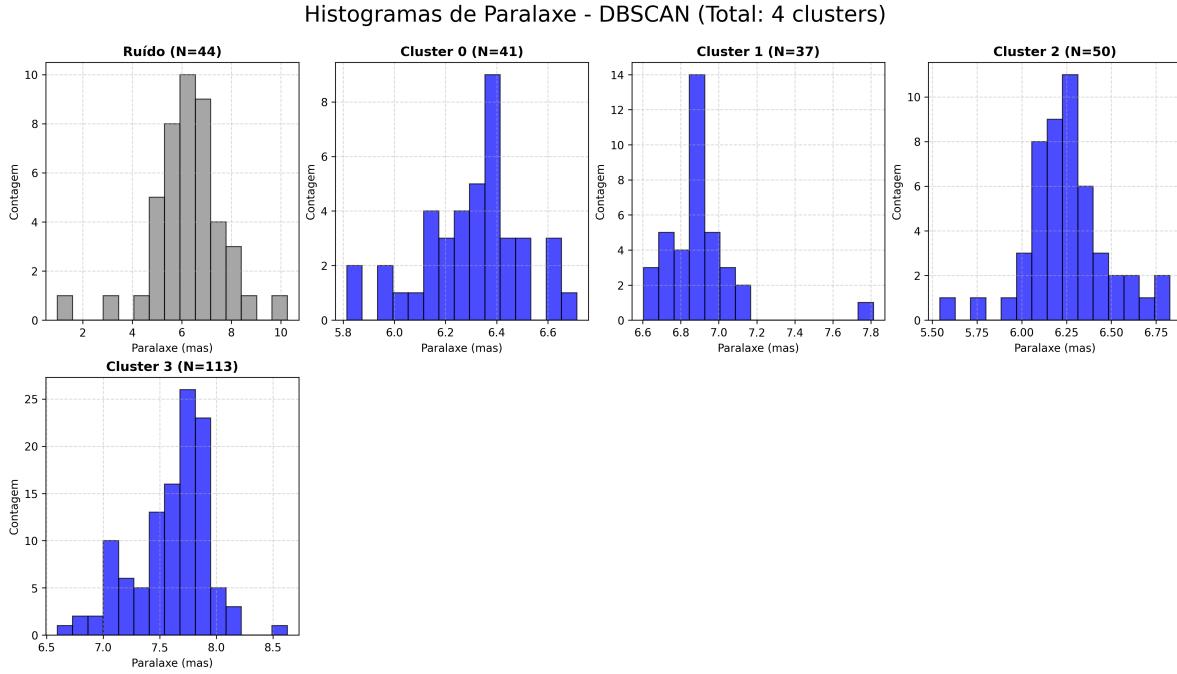
        # Plota o histograma
        ax.hist(cluster_data['parallax'], bins=15, edgecolor='black', alpha=0.7, color=color)

        # Customização
        ax.set_title(plot_title, fontsize=12, fontweight='bold')
        ax.set_xlabel('Paralaxe (mas)', fontsize=10)
        ax.set_ylabel('Contagem', fontsize=10)
        ax.grid(True, linestyle='--', alpha=0.5)
    else:
        ax.axis('off')

fig.suptitle(f'Histogramas de Paralaxe - DBSCAN (Total: {n_clusters-1} clusters)', fontsize=14)
plt.show()

```

Gerando mosaico de histogramas de paralaxe para DBSCAN...



### 3. HDBSCAN

```

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

# --- 1. CONFIGURAÇÃO DOS DADOS ---
labels = df_cleaned['cluster_hdbscan']
cluster_ids = sorted(labels.unique()) # Inclui o -1 (ruído)
n_clusters = len(cluster_ids)

print(f"Gerando mosaico de histogramas de paralaxe para HDBSCAN...")

# --- 2. CONFIGURAÇÃO DO GRID ---
n_cols = 3
n_rows = math.ceil(n_clusters / n_cols)
fig, axs = plt.subplots(n_rows, n_cols, figsize=(15, 4 * n_rows), constrained_layout=True)
axs = axs.flatten()

```

```

# --- 3. LOOP DE PLOTAGEM ---
for i, ax in enumerate(axes):
    if i < n_clusters:
        cluster_id = cluster_ids[i]
        cluster_data = df_cleaned[labels == cluster_id]

        plot_title = f'Cluster {cluster_id} (N={len(cluster_data)})'
        color = 'green'

        if cluster_id == -1:
            plot_title = f'Ruído (N={len(cluster_data)})'
            color = 'gray'

        # Plota o histograma
        ax.hist(cluster_data['parallax'], bins=15, edgecolor='black', alpha=0.7, color=color)

        # Customização
        ax.set_title(plot_title, fontsize=12, fontweight='bold')
        ax.set_xlabel('Paralaxe (mas)', fontsize=10)
        ax.set_ylabel('Contagem', fontsize=10)
        ax.grid(True, linestyle='--', alpha=0.5)
    else:
        ax.axis('off')

fig.suptitle(f'Histogramas de Paralaxe - HDBSCAN (Total: {n_clusters-1} clusters)', fontsize=16)
plt.show()

```

Gerando mosaico de histogramas de paralaxe para HDBSCAN...

Histogramas de Paralaxe - HDBSCAN (Total: 5 clusters)

