

# 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
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
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.ruwe
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=====1762736084217===
303 303
[('Date', 'Mon, 10 Nov 2025 00:54:45 GMT'), ('Server', 'Apache/2.4.6 (SLES Expanded Support
job 17627360856790, at: https://gea.esac.esa.int/tap-server/tap/async/17627360856790
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

```

## 2. Clustering

### 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
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

	source_id	ra	dec	parallax	pmra	pmdec	angular_distance
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
484  1.076504
291  1.461531
461  1.615348
459  1.241720
119  0.978744

```

### 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

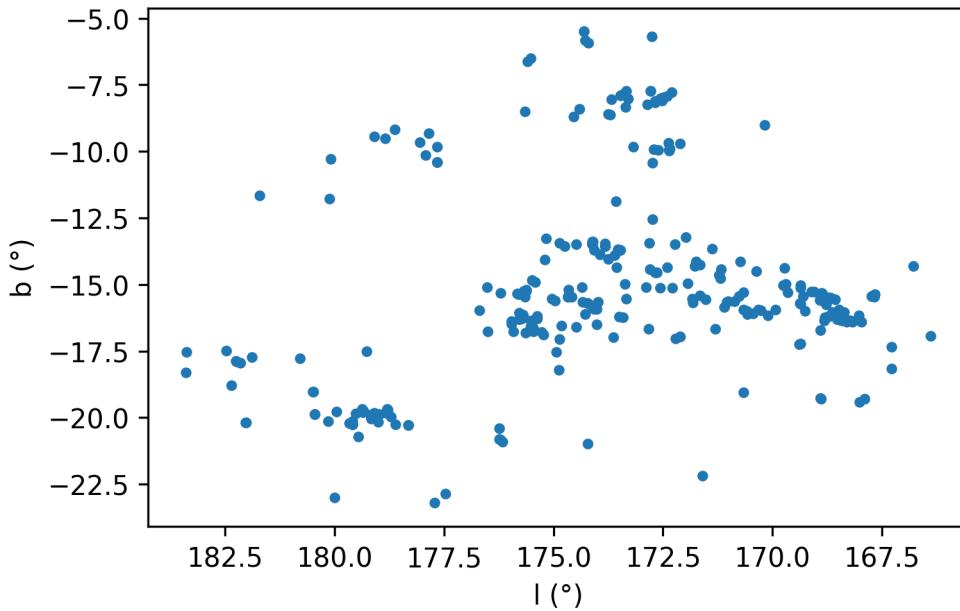
```

### Plota dados

```

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

```



```

X = df_cleaned[['ra', 'dec', 'parallax', 'pmra', 'pmdec']]
range_n_clusters = [2,3,4,5,6,7,8,9,10]
avg_silhouettes = []

for n_clusters in range_n_clusters:
    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(X, cluster_labels)
    print(
        "For n_clusters =",
        n_clusters,
        "The average silhouette_score is :",
        silhouette_avg,
    )
    avg_silhouettes.append(silhouette_avg)
# Compute the silhouette scores for each sample
sample_silhouette_values = silhouette_samples(X, cluster_labels)
best_cluster = avg_silhouettes.index(max(avg_silhouettes))

```

```

print(f"highest value = {max(avg_silhouettes)}, \
      n_clusters = {range_n_clusters[best_cluster]}")

clusterer = KMeans(n_clusters=best_cluster, random_state=10)
cluster_labels = clusterer.fit_predict(X)

fig, (ax1, ax2) = plt.subplots(2,1)
fig.set_size_inches(18, 18)
ax1.plot(range_n_clusters, avg_silhouettes, '-o')
ax1.set_xlabel('k')
ax1.set_ylabel('Average Silhouette Score')
ax2.scatter(df_cleaned['l'], df_cleaned['b'], c=cluster_labels)
plt.gca().invert_xaxis()
ax2.set_xlabel('l (°)')
ax2.set_ylabel('b (°)')
plt.show()

```

For n\_clusters = 2 The average silhouette\_score is : 0.3513435880167119  
 For n\_clusters = 3 The average silhouette\_score is : 0.3432501307716766  
 For n\_clusters = 4 The average silhouette\_score is : 0.3628889639560881  
 For n\_clusters = 5 The average silhouette\_score is : 0.4193704024583248  
 For n\_clusters = 6 The average silhouette\_score is : 0.3739106221513999  
 For n\_clusters = 7 The average silhouette\_score is : 0.41787443037455696  
 For n\_clusters = 8 The average silhouette\_score is : 0.4369431070834808  
 For n\_clusters = 9 The average silhouette\_score is : 0.4038321380146288  
 For n\_clusters = 10 The average silhouette\_score is : 0.4120131572727246  
 highest value = 0.4369431070834808, n\_clusters = 8

