

Trabalho_Final

July 11, 2025

1 Trabalho Final de Computação Científica e Análise de Dados

Nome: Davi dos Santos Mattos

DRE: 119133049

1.1 Tema: Agrupamento e Recomendação de música por similaridade

O objetivo do trabalho é, através de um dataset contendo informações de músicas, onde cada linha representa uma música, e cada colunas representa características da música (ex: Nome, Intérprete, popularidade, duração, etc.), tentar agrupar as músicas com base em suas similaridades e fazer uma recomendação com base em uma música escolhida

Para isso utilizaremos o seguinte dataset [Spotify Tracks Dataset](#), que contém informações de 114 mil músicas.

Vamos começar carregando dataset e dando uma breve olhada nele...

```
[1]: import pandas as pd

file_path = r'D:\REPOSITÓRIOS\obsidian_notebook\Computação Científica e Análise_
de Dados\Trabalho Final\dataset.csv'
df_musicas = pd.read_csv(file_path, index_col=0, sep=',')
df_musicas.head()
```

```
[1]:
```

	track_id	artists	\
0	5Su0ikwiRyPMVoIQDJUGSV	Gen Hoshino	
1	4qPNDBW1i3p13qLCtOKi3A	Ben Woodward	
2	1iJBSr7s7jYXzM8EGcbK5b	Ingrid Michaelson;ZAYN	
3	6lfxq3CG4xtTiEg7opyCyx	Kina Grannis	
4	5vjLSffimiIP26QG5WcN2K	Chord Overstreet	

	album_name	\
0	Comedy	
1	Ghost (Acoustic)	
2	To Begin Again	
3	Crazy Rich Asians (Original Motion Picture Sou...	
4	Hold On	

	track_name	popularity	duration_ms	explicit	\
--	------------	------------	-------------	----------	---

0	Comedy	73	230666	False
1	Ghost - Acoustic	55	149610	False
2	To Begin Again	57	210826	False
3	Can't Help Falling In Love	71	201933	False
4	Hold On	82	198853	False

	danceability	energy	key	loudness	mode	speechiness	acousticness	\
0	0.676	0.4610	1	-6.746	0	0.1430	0.0322	
1	0.420	0.1660	1	-17.235	1	0.0763	0.9240	
2	0.438	0.3590	0	-9.734	1	0.0557	0.2100	
3	0.266	0.0596	0	-18.515	1	0.0363	0.9050	
4	0.618	0.4430	2	-9.681	1	0.0526	0.4690	

	instrumentalness	liveness	valence	tempo	time_signature	track_genre
0	0.000001	0.3580	0.715	87.917	4	acoustic
1	0.000006	0.1010	0.267	77.489	4	acoustic
2	0.000000	0.1170	0.120	76.332	4	acoustic
3	0.000071	0.1320	0.143	181.740	3	acoustic
4	0.000000	0.0829	0.167	119.949	4	acoustic

Verificando o tamanho do dataset

```
[2]: df_musicas.shape
```

```
[2]: (114000, 20)
```

Vamos analisar o que compõem cada coluna e de que tipo são

```
[3]: df_musicas.dtypes.to_frame('types')
```

```
[3]:
```

	types
track_id	object
artists	object
album_name	object
track_name	object
popularity	int64
duration_ms	int64
explicit	bool
danceability	float64
energy	float64
key	int64
loudness	float64
mode	int64
speechiness	float64
acousticness	float64
instrumentalness	float64
liveness	float64
valence	float64

tempo	float64
time_signature	int64
track_genre	object

1.2 Colunas

Coluna	Descrição
track_id	ID do Spotify para a faixa.
artists	Nome(s) dos artistas que performaram a faixa. Se houver mais de um, são separados por ponto e vírgula (;).
album_name	Nome do álbum/playlist onde a faixa aparece.
track_name	Nome da faixa.
popularity	Popularidade da faixa (0 a 100). Calculada por algoritmo com base no número total de execuções e quão recentes essas execuções são. Faixas duplicadas são avaliadas separadamente.
duration_ms	Duração da faixa em milissegundos.
explicit	Indica se a faixa possui conteúdo explícito (True) ou não/desconhecido (False).
danceability	Indica quão dançante é a faixa (0.0 a 1.0). Leva em conta tempo, estabilidade do ritmo, força da batida, entre outros.
energy	Mede a intensidade e atividade da faixa (0.0 a 1.0). Faixas rápidas, altas e barulhentas têm maior energia. (Ex: Heavy Metal possui alta energia enquanto Musica Clássica possui meno energia)
key	Tom da faixa em notação Pitch Class (ex: 0 = C, 1 = C /D , 2 = D, ...), se o tom não for detectado seu valor é -1.
loudness	Volume geral da faixa em decibéis (dB).
mode	Modalidade da faixa: 1 para maior (major), 0 para menor (minor).
speechiness	Mede a presença de fala na faixa (0.0 a 1.0). Valores [> 0.66] = fala pura; [0.33~0.66] = fala + música (ex: rap) e [< 0.33] = predominantemente música.
acousticness	Confiança de que a faixa é acústica (0.0 a 1.0). Quanto mais próximo de 1.0, maior a probabilidade de ser acústica.
instrumentalness	Indica se a faixa é instrumental (sem vocais). Quanto mais próximo de 1.0, maior a chance de não conter voz.
liveness	Detecta a presença de audiência (música ao vivo). Valores > 0.8 indicam forte chance de gravação ao vivo.
valence	Medida de positividade emocional da música (0.0 = negativa, 1.0 = positiva).
tempo	Tempo estimado da faixa em batidas por minuto (BPM).
time_signature	Tempo estimado da música (valores entre 3 e 7 indicando 3/4 até 7/4).
track_genre	Gênero musical ao qual a faixa pertence.

Antes de seguir com o processo de clusterização, precisamos nos certificar de que o dataset está o mais limpo possível

```
[4]: df_musicas[df_musicas.duplicated()]
```

```
[4]:
```

	track_id	artists
1925	0CDucx9lKxuCZplLXUz0iX	Buena Onda Reggae Club

2155	2aibwv5hGXSgw7Yru8IYT0	Red Hot Chili Peppers
3738	7mULVp0DJrI2Nd6GesLvxn	Joy Division
4648	6d3RIvHfVko0tW1WHXmbX3	Little Symphony
5769	481beimUiUnMUzSb0AFcUT	SUPER BEAVER

...
111246	0sSjIvTvd6fUSZZ5rnTPDW	Everything But The Girl
111362	2zg3iJW4fK7KZgH0vJU67z	Faithless
111980	46FPub2Fewe7XrgM0smTYI	Morcheeba
112968	6qVA1MqDrDKfk9144bhoKp	Acil Servis
113345	5Wai0elSGekDk3UNQy8zaw	Matt Redman

	album_name \
1925	Disco 2
2155	Stadium Arcadium
3738	Timeless Rock Hits
4648	Serenity
5769	/

...	...
111246	Eden (Deluxe Edition)
111362	Faithless 2.0
111980	Parts of the Process
112968	Küçük Adam
113345	Sing Like Never Before: The Essential Collection

	track_name	popularity	duration_ms	explicit \
1925	Song for Rollins	16	219346	False
2155	Snow (Hey Oh)	80	334666	False
3738	Love Will Tear Us Apart	0	204621	False
4648	Margot	27	45714	False
5769		54	255080	False

...
111246	Another Bridge - 2012 Remaster	26	132826
111362	Tarantula	21	398152
111980	Undress Me Now	17	203773
112968	Bebek	38	319933
113345	Our God - New Recording	34	265373

	danceability	energy	key	loudness	mode	speechiness	acousticness \
1925	0.841	0.577	0	-7.544	1	0.0438	0.238000
2155	0.427	0.900	11	-3.674	1	0.0499	0.116000
3738	0.524	0.902	2	-8.662	1	0.0368	0.000989
4648	0.269	0.142	0	-23.695	1	0.0509	0.866000
5769	0.472	0.994	8	-1.786	1	0.1140	0.025900
...
111246	0.480	0.853	0	-6.276	1	0.0734	0.030600
111362	0.622	0.816	6	-11.095	0	0.0483	0.009590
111980	0.576	0.352	7	-10.773	0	0.0268	0.700000

112968	0.486	0.485	5	-12.391	0	0.0331	0.004460
113345	0.487	0.895	11	-5.061	1	0.0413	0.000183

	instrumentalness	liveness	valence	tempo	time_signature	\
1925	0.860000	0.0571	0.843	90.522	4	
2155	0.000017	0.1190	0.599	104.655	4	
3738	0.695000	0.1370	0.907	146.833	4	
4648	0.904000	0.1140	0.321	67.872	3	
5769	0.000000	0.0535	0.262	103.512	4	
...	
111246	0.000001	0.3200	0.775	85.181	4	
111362	0.578000	0.0991	0.427	136.007	4	
111980	0.270000	0.1600	0.360	95.484	4	
112968	0.000017	0.3690	0.353	120.095	4	
113345	0.000000	0.3590	0.384	105.021	4	

	track_genre
1925	afrobeat
2155	alt-rock
3738	alternative
4648	ambient
5769	anime
...	...
111246	trip-hop
111362	trip-hop
111980	trip-hop
112968	turkish
113345	world-music

[450 rows x 20 columns]

```
[5]: print(df_musicas.duplicated().sum())
```

450

Existem 450 faixas que possuem cópias idênticas dentro do dataset, vamos investigar essas duplicatas

```
[6]: df_musicas[df_musicas['track_id']=='2aibwv5hGXSgw7Yru8IYT0']
```

	track_id	artists	album_name	\
2109	2aibwv5hGXSgw7Yru8IYT0	Red Hot Chili Peppers	Stadium Arcadium	
2155	2aibwv5hGXSgw7Yru8IYT0	Red Hot Chili Peppers	Stadium Arcadium	
3259	2aibwv5hGXSgw7Yru8IYT0	Red Hot Chili Peppers	Stadium Arcadium	
37216	2aibwv5hGXSgw7Yru8IYT0	Red Hot Chili Peppers	Stadium Arcadium	
71158	2aibwv5hGXSgw7Yru8IYT0	Red Hot Chili Peppers	Stadium Arcadium	
91854	2aibwv5hGXSgw7Yru8IYT0	Red Hot Chili Peppers	Stadium Arcadium	

	track_name	popularity	duration_ms	explicit	danceability	energy	\
2109	Snow (Hey Oh)	80	334666	False	0.427	0.9	
2155	Snow (Hey Oh)	80	334666	False	0.427	0.9	
3259	Snow (Hey Oh)	80	334666	False	0.427	0.9	
37216	Snow (Hey Oh)	80	334666	False	0.427	0.9	
71158	Snow (Hey Oh)	80	334666	False	0.427	0.9	
91854	Snow (Hey Oh)	80	334666	False	0.427	0.9	

	key	loudness	mode	speechiness	acousticness	instrumentalness	\
2109	11	-3.674	1	0.0499	0.116	0.000017	
2155	11	-3.674	1	0.0499	0.116	0.000017	
3259	11	-3.674	1	0.0499	0.116	0.000017	
37216	11	-3.674	1	0.0499	0.116	0.000017	
71158	11	-3.674	1	0.0499	0.116	0.000017	
91854	11	-3.674	1	0.0499	0.116	0.000017	

	liveness	valence	tempo	time_signature	track_genre
2109	0.119	0.599	104.655	4	alt-rock
2155	0.119	0.599	104.655	4	alt-rock
3259	0.119	0.599	104.655	4	alternative
37216	0.119	0.599	104.655	4	funk
71158	0.119	0.599	104.655	4	metal
91854	0.119	0.599	104.655	4	rock

Vamos remover essas duplicatas idênticas

```
[7]: df_musicas = df_musicas.drop_duplicates()
df_musicas.shape
```

```
[7]: (113550, 20)
```

Passamos a ter 113550 músicas no dataset, vamos verificar se ainda há duplicatas

```
[8]: df_musicas[df_musicas['track_name'] == "Snow (Hey Oh)"]
```

```
[8]:
```

	track_id	artists	album_name	\
2109	2aibwv5hGXSgw7Yru8IYT0	Red Hot Chili Peppers	Stadium Arcadium	
3259	2aibwv5hGXSgw7Yru8IYT0	Red Hot Chili Peppers	Stadium Arcadium	
23630	4dIZsI7RjIHeNgBJInhpli	Goldbird;Offmind	Snow (Hey Oh)	
37216	2aibwv5hGXSgw7Yru8IYT0	Red Hot Chili Peppers	Stadium Arcadium	
71158	2aibwv5hGXSgw7Yru8IYT0	Red Hot Chili Peppers	Stadium Arcadium	
91854	2aibwv5hGXSgw7Yru8IYT0	Red Hot Chili Peppers	Stadium Arcadium	

	track_name	popularity	duration_ms	explicit	danceability	energy	\
2109	Snow (Hey Oh)	80	334666	False	0.427	0.900	
3259	Snow (Hey Oh)	80	334666	False	0.427	0.900	
23630	Snow (Hey Oh)	59	164221	False	0.603	0.714	
37216	Snow (Hey Oh)	80	334666	False	0.427	0.900	

71158	Snow (Hey Oh)	80	334666	False	0.427	0.900
91854	Snow (Hey Oh)	80	334666	False	0.427	0.900

	key	loudness	mode	speechiness	acousticness	instrumentalness	\
2109	11	-3.674	1	0.0499	0.116	0.000017	
3259	11	-3.674	1	0.0499	0.116	0.000017	
23630	8	-9.071	0	0.1080	0.419	0.002050	
37216	11	-3.674	1	0.0499	0.116	0.000017	
71158	11	-3.674	1	0.0499	0.116	0.000017	
91854	11	-3.674	1	0.0499	0.116	0.000017	

	liveness	valence	tempo	time_signature	track_genre
2109	0.119	0.599	104.655	4	alt-rock
3259	0.119	0.599	104.655	4	alternative
23630	0.713	0.736	113.917	4	deep-house
37216	0.119	0.599	104.655	4	funk
71158	0.119	0.599	104.655	4	metal
91854	0.119	0.599	104.655	4	rock

Ao investigar uma musica que tenha uma duplicata, podemos perceber que provavelmente é por causa de que uma música pode pertencer a dois “álbuns” (Album ou EP) ou então a mais de um gênero

Vamos começar retirando as músicas com o mesmo `track_id` e ver se todas as duplicatas foram retiradas

```
[9]: df_musicas = df_musicas.drop_duplicates(subset=['track_id'], keep='first')
df_musicas[df_musicas['track_name'] == "Snow (Hey Oh)"]
```

```
[9]:
```

	track_id	artists	album_name	\
2109	2aibwv5hGXSgw7Yru8IYTO	Red Hot Chili Peppers	Stadium Arcadium	
23630	4dIZsI7RjIHeNgBJInhpli	Goldbird;Offmind	Snow (Hey Oh)	

	track_name	popularity	duration_ms	explicit	danceability	energy	\
2109	Snow (Hey Oh)	80	334666	False	0.427	0.900	
23630	Snow (Hey Oh)	59	164221	False	0.603	0.714	

	key	loudness	mode	speechiness	acousticness	instrumentalness	\
2109	11	-3.674	1	0.0499	0.116	0.000017	
23630	8	-9.071	0	0.1080	0.419	0.002050	

	liveness	valence	tempo	time_signature	track_genre
2109	0.119	0.599	104.655	4	alt-rock
23630	0.713	0.736	113.917	4	deep-house

Aparentemente remover duplicatas de `track_id`, foi o suficiente para tratar das duplicatas

```
[10]: print(df_musicas.shape)
```

(89741, 20)

Vamos verificar se há também valores nulos

```
[11]: df_musicas[df_musicas.isnull().any(axis=1)]
```

```
[11]:
```

	track_id	artists	album_name	track_name	popularity	\
65900	1kR4gIb7nGxHPI3D2ifs59	NaN	NaN	NaN	0	

	duration_ms	explicit	danceability	energy	key	loudness	mode	\
65900	0	False	0.501	0.583	7	-9.46	0	

	speechiness	acousticness	instrumentalness	liveness	valence	\
65900	0.0605	0.69	0.00396	0.0747	0.734	

	tempo	time_signature	track_genre
65900	138.391	4	k-pop

```
[12]: df_musicas = df_musicas.dropna(axis=0)
```

Após remover todas as duplicatas e valores nulos do dataset, vamos conferir quantos dados sobraram.

```
[13]: df_musicas.shape
```

```
[13]: (89740, 20)
```

Com isso nosso dataset que havia 114 mil músicas, na verdade tem 89740 músicas.

Com isso vamos pegar apenas as colunas que possuem valores numéricos

```
[14]: df_dados = df_musicas.select_dtypes(include='number')
df_dados
```

```
[14]:
```

	popularity	duration_ms	danceability	energy	key	loudness	mode	\
0	73	230666	0.676	0.4610	1	-6.746	0	
1	55	149610	0.420	0.1660	1	-17.235	1	
2	57	210826	0.438	0.3590	0	-9.734	1	
3	71	201933	0.266	0.0596	0	-18.515	1	
4	82	198853	0.618	0.4430	2	-9.681	1	
...	
113995	21	384999	0.172	0.2350	5	-16.393	1	
113996	22	385000	0.174	0.1170	0	-18.318	0	
113997	22	271466	0.629	0.3290	0	-10.895	0	
113998	41	283893	0.587	0.5060	7	-10.889	1	
113999	22	241826	0.526	0.4870	1	-10.204	0	

	speechiness	acousticness	instrumentalness	liveness	valence	\
0	0.1430	0.0322	0.000001	0.3580	0.7150	
1	0.0763	0.9240	0.000006	0.1010	0.2670	

2	0.0557	0.2100	0.000000	0.1170	0.1200
3	0.0363	0.9050	0.000071	0.1320	0.1430
4	0.0526	0.4690	0.000000	0.0829	0.1670
...
113995	0.0422	0.6400	0.928000	0.0863	0.0339
113996	0.0401	0.9940	0.976000	0.1050	0.0350
113997	0.0420	0.8670	0.000000	0.0839	0.7430
113998	0.0297	0.3810	0.000000	0.2700	0.4130
113999	0.0725	0.6810	0.000000	0.0893	0.7080

	tempo	time_signature
0	87.917	4
1	77.489	4
2	76.332	4
3	181.740	3
4	119.949	4
...
113995	125.995	5
113996	85.239	4
113997	132.378	4
113998	135.960	4
113999	79.198	4

[89740 rows x 14 columns]

Que pertencem as seguintes colunas

```
[15]: df_dados.columns
```

```
[15]: Index(['popularity', 'duration_ms', 'danceability', 'energy', 'key',
        'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness',
        'liveness', 'valence', 'tempo', 'time_signature'],
        dtype='object')
```

Agora sim, podemos seguir para o processo de Clusterização

1.3 Funções Necessárias

Porém antes vamos definir algumas funções que vão nos ajudar posteriormente

`calcular_soma_erros()` - soma dos quadrados intra-clusters

`optimal_number_of_clusters()` - a quantidade ótima de clusters baseado na distância entre um ponto e uma reta

Referência: [Como definir o número de clusters para o seu KMeans](#)

`tamanho_amostra()` - Faz o calculo do tamanho da amostra necessária com grau de confiança de 95% e margem de erro de 5% (Para que seja computacionalmente viável)

recomendar_musicas() - Recomenda 10 músicas com base na faixa escolhida levando em conta o cluster em que ela se encontra e o cosseno do ângulo da mesma para as demais faixas.

```
[16]: from math import sqrt
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

def calcular_soma_erros(matrix):

    soma_erros = []
    for k in range(1, 20):
        kmeans = KMeans(n_clusters=k, random_state=14)
        kmeans.fit(matrix)
        soma_erros.append(kmeans.inertia_)
    return soma_erros

def optimal_number_of_clusters(wcss):
    x1, y1 = 1, wcss[0]
    x2, y2 = 20, wcss[len(wcss)-1]

    distances = []
    for i in range(len(wcss)):
        x0 = i+2
        y0 = wcss[i]
        numerator = abs((y2-y1)*x0 - (x2-x1)*y0 + x2*y1 - y2*x1)
        denominator = sqrt((y2 - y1)**2 + (x2 - x1)**2)
        distances.append(numerator/denominator)

    return distances.index(max(distances)) + 2

[17]: def tamanho_amostra(populacao):
import math
z = 1.96 # para 95% de confiança
e = 0.05 # margem de erro
p = 0.5 # proporção mais conservadora

n0 = (z**2 * p * (1 - p)) / (e**2)
n = (populacao * n0) / (populacao + n0 - 1)
return math.ceil(n)

[18]: from scipy.spatial.distance import euclidean, cosine

def recomendar_musicas(track_id, df_dados_normalizados, df_musicas):
    idx = df_musicas.index[df_musicas['track_id'] == track_id][0]
    cluster_id = df_musicas.loc[idx, 'cluster']
    print(f"Cluster: {cluster_id}")
    musica_base = df_dados_normalizados.drop(columns='cluster').iloc[idx].values
```

```

subset = df_dados_normalizados[df_dados_normalizados['cluster'] ==
↳ cluster_id].copy()
subset['distancia'] = subset.drop(columns='cluster').apply(lambda row:
↳ cosine(row.values, musica_base), axis=1)

resultado_idx = subset.sort_values('distancia').index
recomendacoes = df_musicas.iloc[resultado_idx][['track_name',
↳ 'artists', 'track_genre']]
recomendacoes = recomendacoes[recomendacoes.index != idx]

return recomendacoes.head(10)

```

Além de fazer o agrupamento por similaridade e a recomendação de músicas parecidas, vamos também analisar o impacto no agrupamento e consequentemente na recomendação de acordo com o modelo de normalização que utilizarmos.

Para exemplificar, vamos usar 4 tipos de normalização de dados - Robust Scale

Usa mediana e intervalo interquartil (IQR) para escalonar, ignorando outliers. Reduzindo o impacto de valores extremos sem removê-los. *Recomendado quando: Há outliers que distorcem a média ou o desvio padrão.*

- Min Max Scale

Reescala os dados para um intervalo específico (padrão: 0 a 1). Útil quando é importante preservar proporções relativas dos dados. *Recomendado quando: Os dados estão em escalas distintas e não há outliers relevantes.*

- Standart Scale

Transforma os dados para terem média 0 e desvio padrão 1. Ideal para algoritmos que assumem distribuição normal (ex: PCA, SVM, regressão logística). *Recomendado quando: Há presença de variáveis gaussianas ou com variâncias diferentes.*

- Max Abs Scale

Escala os dados pela magnitude máxima absoluta de cada atributo. Mantém sinais (positivo/negativo) e é útil em dados esparsos. *Recomendado quando: Você usa dados com muitos zeros (sparse) e não quer centrar em zero.*

De acordo com as recomendações as melhores normalizações são as *Robust Scale* e *Standart Scale*

2 Robust Scale

Primeiro normalizaremos os dados com Robust Scale

```

[19]: from sklearn.preprocessing import robust_scale
      escala1 = robust_scale(df_dados)
      df_dados_normalizados1 = pd.DataFrame(escala1, columns=df_dados.columns[:])
      df_dados_normalizados1

```

```
[19]:
```

	popularity	duration_ms	danceability	energy	key	loudness	\
0	1.333333	0.190355	0.413223	-0.542929	-0.666667	0.084192	
1	0.733333	-0.697900	-0.644628	-1.287879	-0.666667	-1.927410	
2	0.800000	-0.027062	-0.570248	-0.800505	-0.833333	-0.488853	
3	1.266667	-0.124516	-1.280992	-1.556566	-0.833333	-2.172892	
4	1.633333	-0.158269	0.173554	-0.588384	-0.500000	-0.478688	
...	
89735	-0.400000	1.881620	-1.669421	-1.113636	0.000000	-1.765930	
89736	-0.366667	1.881631	-1.661157	-1.411616	-0.833333	-2.135111	
89737	-0.366667	0.637464	0.219008	-0.876263	-0.833333	-0.711512	
89738	0.266667	0.773646	0.045455	-0.429293	0.333333	-0.710361	
89739	-0.366667	0.312653	-0.206612	-0.477273	-0.666667	-0.578990	
...	
	mode	speechiness	acousticness	instrumentalness	liveness	valence	\
0	-1.0	1.885772	-0.256292	-0.000584	1.250000	0.595843	
1	0.0	0.549098	1.210725	-0.000537	-0.171460	-0.438799	
2	0.0	0.136273	0.036190	-0.000594	-0.082965	-0.778291	
3	0.0	-0.252505	1.179470	0.000130	0.000000	-0.725173	
4	0.0	0.074148	0.462247	-0.000594	-0.271571	-0.669746	
...	
89735	0.0	-0.134269	0.743543	9.505168	-0.252765	-0.977136	
89736	-1.0	-0.176353	1.325876	9.996845	-0.149336	-0.974596	
89737	-1.0	-0.138277	1.116960	-0.000594	-0.266040	0.660508	
89738	0.0	-0.384770	0.317486	-0.000594	0.763274	-0.101617	
89739	-1.0	0.472946	0.810989	-0.000594	-0.236173	0.579677	
...	
	tempo	time_signature					
0	-0.835395	0.0					
1	-1.090893	0.0					
2	-1.119241	0.0					
3	1.463386	-1.0					
4	-0.050571	0.0					
...					
89735	0.097564	1.0					
89736	-0.901009	0.0					
89737	0.253955	0.0					
89738	0.341719	0.0					
89739	-1.049021	0.0					

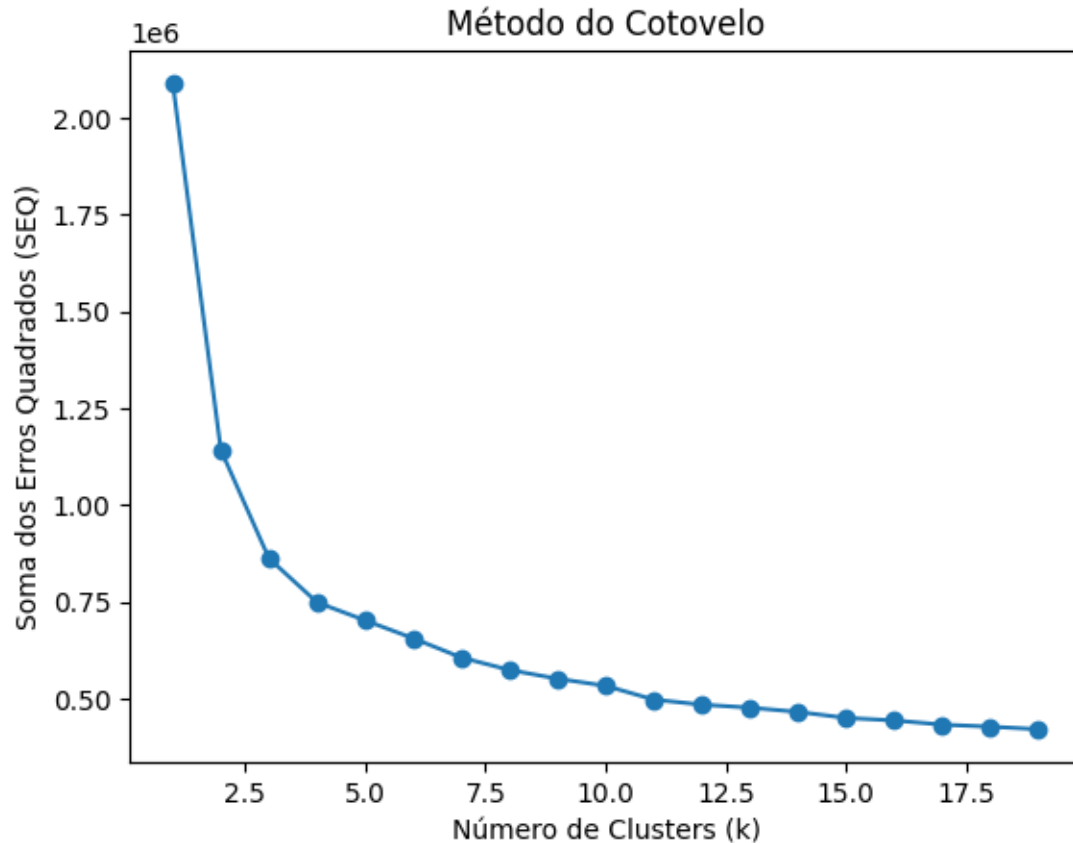
[89740 rows x 14 columns]

Agora precisamos descobrir qual é a melhor quantidade de clusters para agrupar os dados, e para isso vamos utilizar o método do cotovelo e usando as funções `calcular_soma_erros` e `optimal_number_of_clusters`

```
[20]: soma_e1 = calcular_soma_erros(escala1)

plt.plot(range(1, 20), soma_e1, marker='o')
```

```
plt.xlabel('Número de Clusters (k)')
plt.ylabel('Soma dos Erros Quadrados (SEQ)')
plt.title('Método do Cotovelo')
plt.show()
```



Se tentarmos escolher o número de cluster a olho nu pelo gráfico, aparentemente um número bom de clusters seria $k = 9$, porém temos uma função que descobre o k ideal para nós

```
[21]: ncluster1 = optimal_number_of_clusters(soma_e1)
ncluster1
```

[21]: 5

Rodando a função, vemos que o k ideal é $k = 5$

Portanto agora vamos agrupar as faixas, `df_musicas_rs` para agrupar o dataset original, `df_dados` para agrupar os dados sem serem normalizados, e `df_dados_normalizados1` para agrupar os dados normalizados

```
[22]: kmeans = KMeans(n_clusters=ncluster1, random_state=14)
clusters = kmeans.fit_predict(escala1)
```

```
df_musicas_rs = df_musicas.copy()
df_musicas_rs['cluster'] = clusters
df_dados['cluster'] = clusters
df_dados_normalizados1['cluster'] = clusters
```

Criei 3 variáveis pelos seguintes motivos, fazer a plotagem de gráficos e rodar a função `recomendar_musicas()`

2.1 Plotagem 2D e 3D dos clusters

Vamos utilizar o PCA para ter visualização dos clusters em 2D e 3D

```
[23]: from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

pca = PCA(n_components=2)
X_pca = pca.fit_transform(escala1)

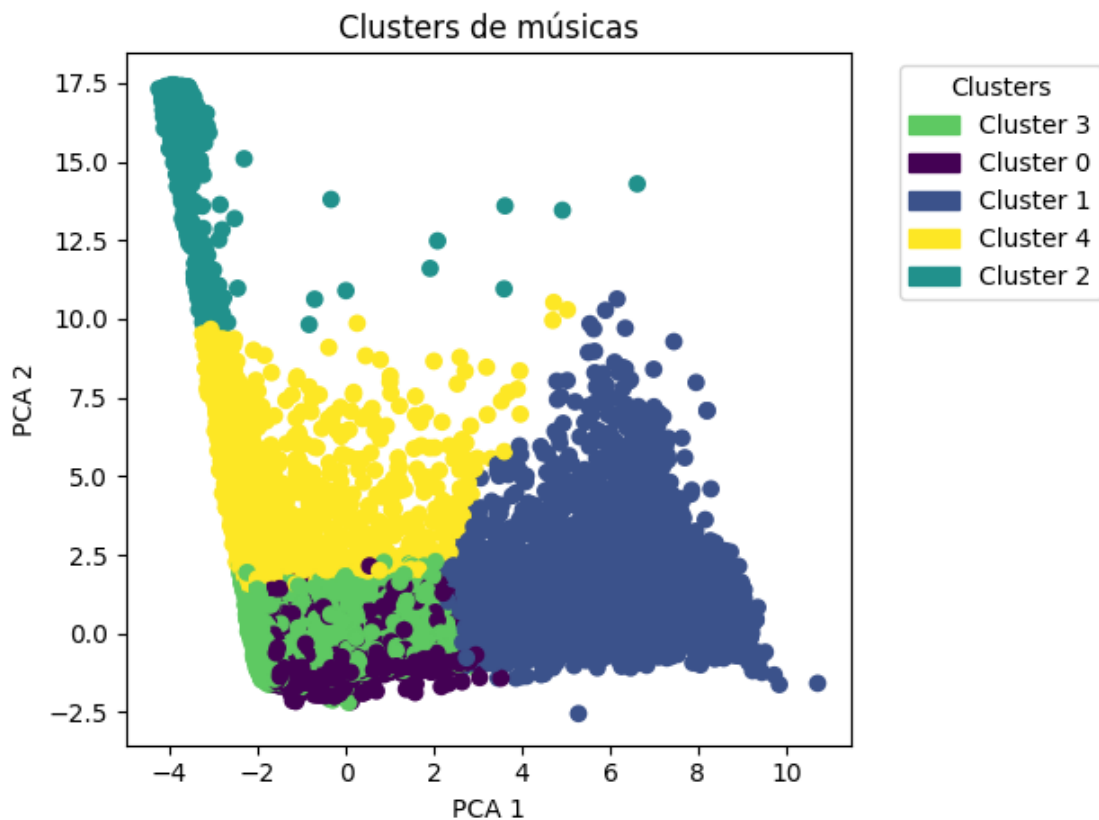
# Plot
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1],
    ↪c=df_dados_normalizados1['cluster'], cmap='viridis')

# Legenda manual
clusters = df_dados_normalizados1['cluster'].unique()
colors = scatter.cmap(scatter.norm(clusters))

handles = [mpatches.Patch(color=colors[i], label=f'Cluster {clusters[i]}') for
    ↪i in range(len(clusters))]

plt.legend(handles=handles, title='Clusters', bbox_to_anchor=(1.05, 1),
    ↪loc='upper left')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.title('Clusters de músicas')

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

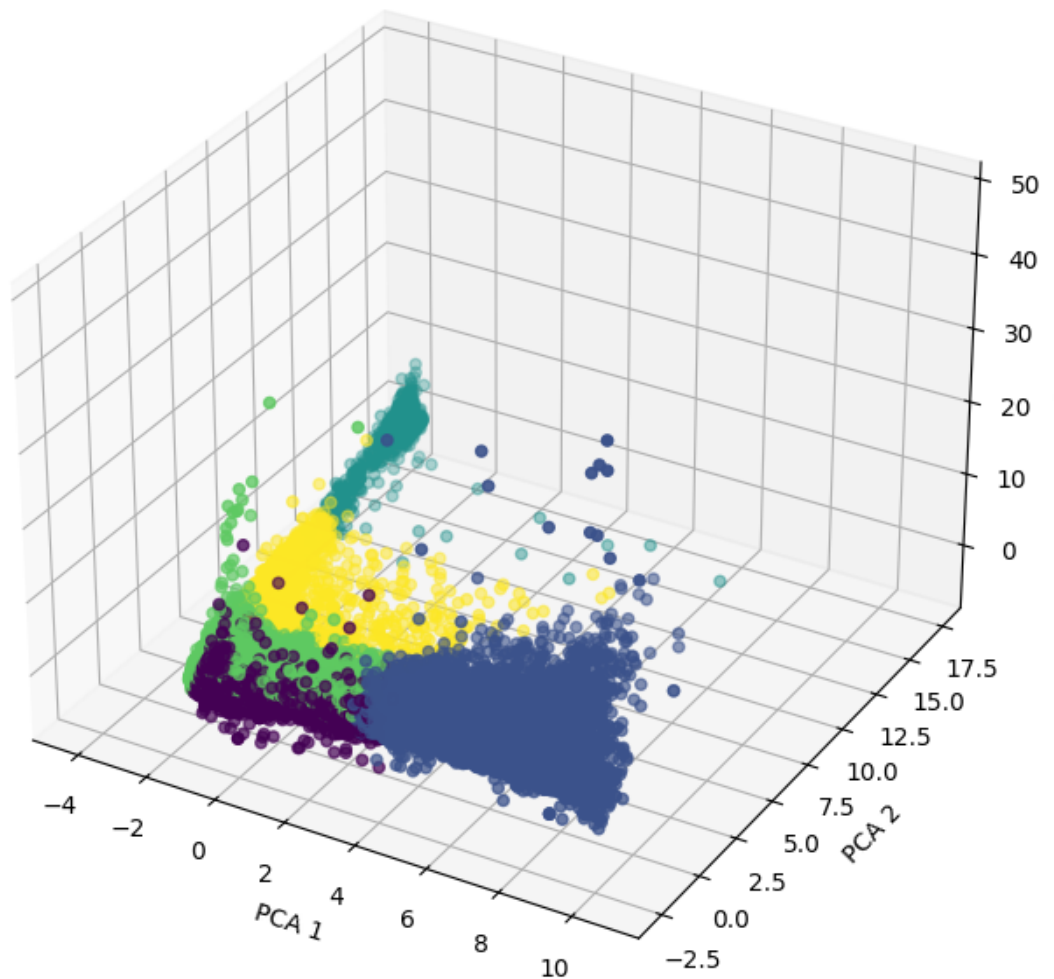


```
[24]: from mpl_toolkits.mplot3d import Axes3D

pca = PCA(n_components=3)
X_pca_3d = pca.fit_transform(escala1)

fig = plt.figure(figsize=(10,8))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X_pca_3d[:, 0], X_pca_3d[:, 1], X_pca_3d[:, 2],
                    c=df_dados_normalizados1['cluster'], cmap='viridis')
ax.set_xlabel('PCA 1')
ax.set_ylabel('PCA 2')
ax.set_zlabel('PCA 3')
plt.title('Clusters em 3D com PCA')
plt.show()
```

Clusters em 3D com PCA



2.2 Análise de agrupamento

Vamos ver quantas músicas foram agrupadas em cada cluster, e verificar se todas as variáveis possuem a mesma quantidade por cluster

```
[25]: df_musicas_rs['cluster'].value_counts()
```

```
[25]: cluster
3      41235
0      22472
1      17371
4       7733
```



```
2      929
Name: count, dtype: int64
```

Vamos tentar fazer uma análise bruta, calculando a média das características para cada cluster e comparar com as médias agrupadas por gênero da música do dataset original

2.2.1 Dados Originais

```
[26]: df_dados_media = df_dados.groupby('cluster').mean(numeric_only=True)
df_dados_media
```

```
[26]:
```

	popularity	duration_ms	danceability	energy	key	\
cluster						
0	32.960306	219468.585662	0.534000	0.398945	5.051175	
1	28.088020	256630.304531	0.497905	0.547896	5.252605	
2	23.758881	222023.057051	0.567002	0.694920	5.067815	
3	35.539299	228428.393961	0.587537	0.784274	5.382733	
4	34.026251	200192.427906	0.652508	0.707178	5.525152	

	loudness	mode	speechiness	acousticness	instrumentalness	\
cluster						
0	-10.854193	0.730109	0.043434	0.598251	0.021228	
1	-12.773717	0.589200	0.064965	0.386202	0.810877	
2	-11.209023	0.696448	0.884701	0.762481	0.005928	
3	-5.673428	0.620783	0.065279	0.161312	0.020798	
4	-6.793617	0.552826	0.288229	0.251861	0.017640	

	liveness	valence	tempo	time_signature
cluster				
0	0.157000	0.418517	113.080880	3.815148
1	0.184031	0.303307	118.951770	3.830868
2	0.696433	0.429245	100.385352	3.578041
3	0.250684	0.555456	128.028073	3.963793
4	0.227870	0.537177	125.893784	3.970516

Para os dados originais sem estarem normalizados, vemos o seguinte: - **Cluster 0:**

Músicas populares,
Menos intensas,
Menos barulhentas,
Com pouca ou nenhuma fala ,
Mais acústicas, Gravadas em estúdio,

- **Cluster 1:**

Baixa popularidade, Mais longas, Não muito dançante, Não tão barulhentas, Mais calmas, com pouca ou nenhuma voz, Faixas instrumentais,

- **Cluster 2:**

Mais popular, alta energia, som alto, dançante e de humor positivo.

- **Cluster 3:**

Extremamente “falado”, alta “liveness” (ao vivo), muito acústico, baixíssima popularidade.

- **Cluster 4:**

Muito dançante, alta energia, nível de “fala” típico de rap.

2.2.2 Dados Normalizados

```
[27]: df_dados_normalizados1_media = df_dados_normalizados1.groupby('cluster').mean()
      ↪ # Para gráfico de médias
      df_dados_normalizados1_media
```

```
[27]:
```

	popularity	duration_ms	danceability	energy	key	loudness	\
cluster							
0	-0.001323	0.067648	-0.173553	-0.699635	0.008529	-0.703686	
1	-0.163733	0.474886	-0.322708	-0.323496	0.042101	-1.071816	
2	-0.308037	0.095641	-0.037181	0.047778	0.011302	-0.771736	
3	0.084643	0.165834	0.047672	0.273419	0.063789	0.289892	
4	0.034208	-0.143591	0.316147	0.078732	0.087525	0.075060	

	mode	speechiness	acousticness	instrumentalness	liveness	\
cluster						
0	-0.269891	-0.109545	0.674866	0.216852	0.138276	
1	-0.410800	0.321935	0.326044	8.305441	0.287780	
2	-0.303552	16.749514	0.945025	0.060125	3.121861	
3	-0.379217	0.328231	-0.043903	0.212444	0.656441	
4	-0.447174	4.796180	0.105052	0.180096	0.530256	

	valence	tempo	time_signature
cluster			
0	-0.088875	-0.218848	-0.184852
1	-0.354950	-0.075004	-0.169132
2	-0.064100	-0.529904	-0.421959
3	0.227381	0.147377	-0.036207
4	0.185167	0.095084	-0.029484

Analisando os dados normalizados podemos ver que ele, esta casando com os dados originais, porém ficando mais claro distinguir cada grupo

Cluster 0: Mais calmas, menos fala, gravadas em estúdio

Cluster 1: Maior duração, mais dançantes, menos barulhentas, mais faixas instrumentais e atmosféricas

Cluster 2: Maior popularidade, Mais barulhentas, mais animadas, com maior bpm e predominantemente músicas acústicas

Cluster 3: Menos populares, mais enérgicas, mais faladas, acústicas, ao vivo, menor bpm

Cluster 4: Sem faixa instrumental, menos dançantes, menor duração

```
[28]: df_musicas_rs[df_musicas_rs['cluster']==  
↳2][['artists','track_name','track_genre']].head(10)
```

```
[28]:
```

	artists \	track_name	track_genre
1321	Afrocidade	Vivão (vinheta)	afrobeat
1492	Jorge Drexler	Fractura de escafoides tarsiano derecho - Cara B	afrobeat
1574	Afrocidade;Nildes Bomfim	Canoeiro (vinheta)	afrobeat
4073	Mohan;Murali;Chandiran	Varraaru Vaarraaru Yaaru Varraaru	ambient
9152	Sant;Stau	O Que Separa os Homens dos Meninos	brazil
9769	Racionais MC's	De Volta À Cena	brazil
10886	Noisia;Trolley Snatcha;Foreign Beggars	Contact - Trolley Snatcha Remix	breakbeat
14146	The Laurie Berkner Band	Down Down Baby	children
14427	WowKidz	Aao Bhai Aao Kyon Bhai Kyon	children
14587	Caillou	Cours, papa!	children

```
[29]: df_musicas_rs[df_musicas_rs['cluster']==  
↳3][['artists','track_name','track_genre']].head(10)
```

```
[29]:
```

	artists \	track_name	track_genre
0	Gen Hoshino	Comedy	acoustic
5	Tyrone Wells	Days I Will Remember	acoustic
14	Chord Overstreet;Deepend	Hold On - Remix	acoustic
24	Jason Mraz	Unlonely	acoustic
27	Jason Mraz		
40	Eddie Vedder		
42	Brandi Carlile;Lucius		
43	Brandi Carlile;Lucius		
44	Brandi Carlile		
45	Brandi Carlile;Lucius		

27		If It Kills Me	acoustic
40		The Haves	acoustic
42		You and Me on the Rock	acoustic
43		You and Me on the Rock	acoustic
44	Speak Your Mind (From the Netflix Series "We T...		acoustic
45		You and Me on the Rock	acoustic

Aparentemente está batendo o que observamos

2.2.3 Gênero de músicas por cluster

Vamos tentar validar o que vimos acima com os gêneros mais presentes em cada cluster

```
[30]: genero_por_cluster = df_musicas_rs.groupby('cluster').apply(
      lambda x: x['track_genre'].value_counts().head(10),
      include_groups=False
    )
      genero_por_cluster
```

```
[30]: cluster  track_genre
0          honky-tonk      834
          cantopop       784
          romance        767
          acoustic       684
          opera          663
          tango          659
          mandopop       580
          show-tunes     555
          rock-n-roll    532
          bluegrass     518
1          study        905
          sleep         803
          new-age       785
          minimal-techno 783
          idm           772
          ambient       763
          detroit-techno 762
          iranian       676
          grindcore     609
          classical     576
2          comedy       786
          kids           17
          show-tunes     14
          children       12
          funk           9
          jazz           7
          iranian        7
          chill          5
```

	german	4
	garage	4
3	forro	931
	j-idol	868
	heavy-metal	816
	alt-rock	781
	dance	750
	party	731
	salsa	725
	pagode	723
	grunge	710
	power-pop	681
4	dancehall	441
	j-dance	426
	kids	299
	hardcore	280
	funk	262
	french	233
	hip-hop	219
	turkish	197
	emo	188
	hardstyle	181

Name: count, dtype: int64

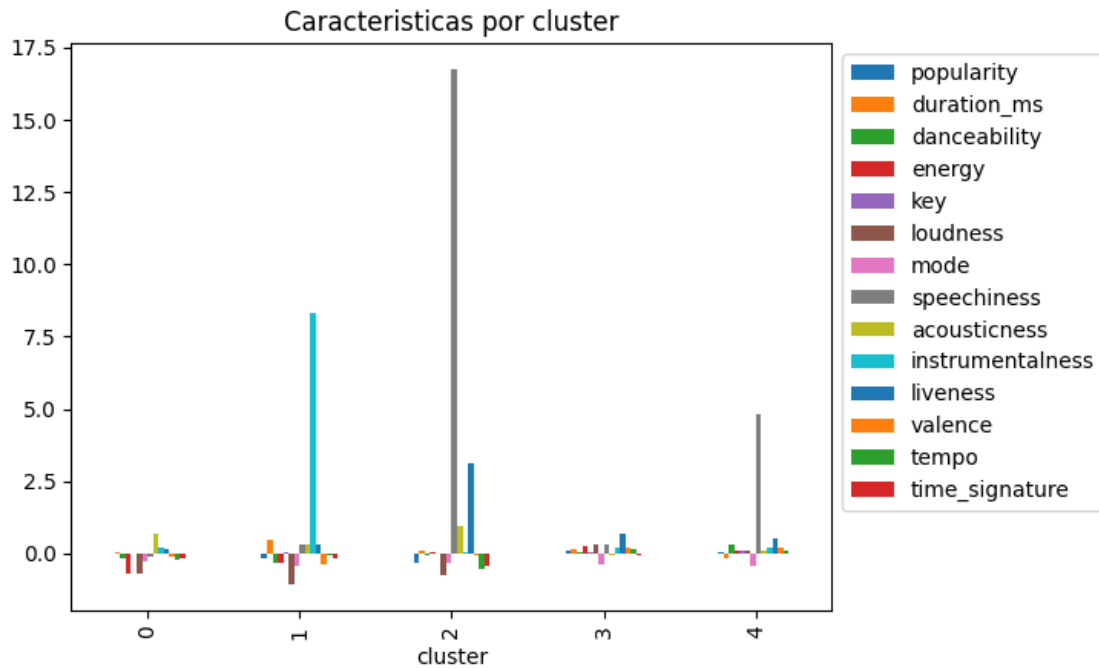
2.3 Gráficos

Vamos plotar os gráficos e ver se está condizente com a classificação feita anteriormente

2.3.1 Características X Cluster

Utilizando as médias das características dos dados normalizados

```
[31]: df_dados_normalizados1_media.plot(kind='bar')
plt.title('Características por cluster')
plt.legend(bbox_to_anchor=(1.0, 1.0))
plt.show()
```

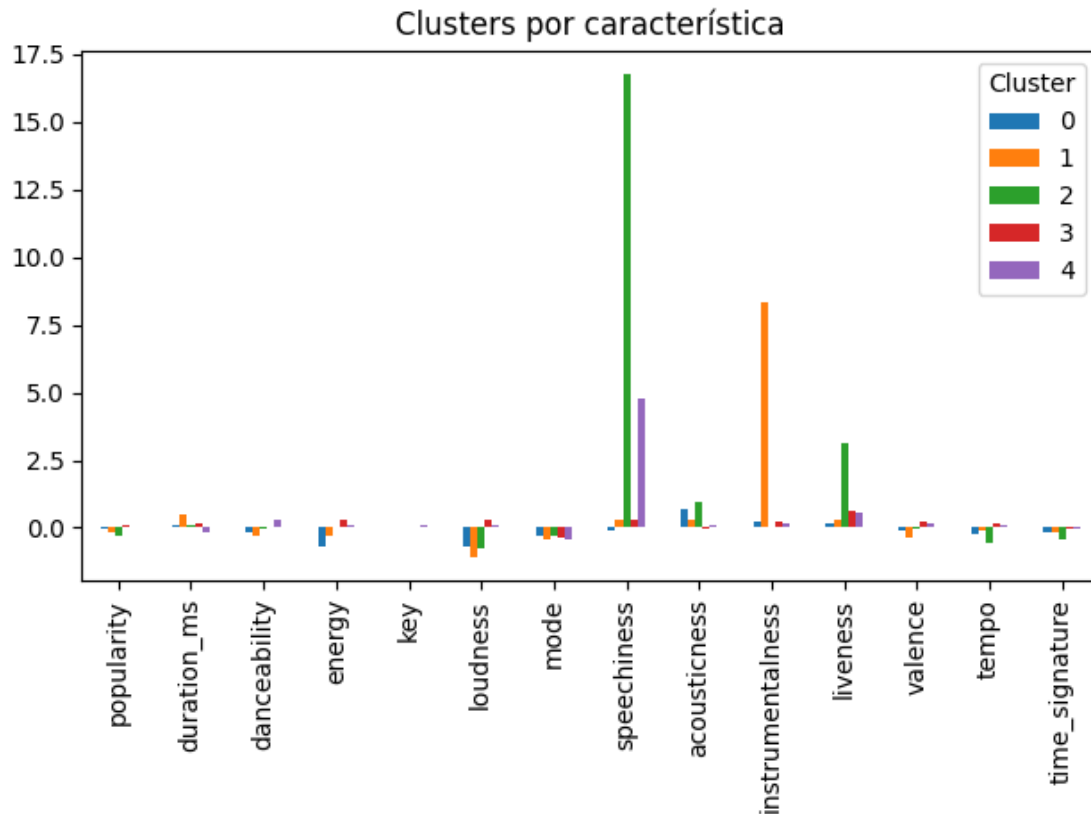


Olhando o gráfico acima vamos tentar classificar cada cluster

- Cluster 0: Menos 'energy' e 'loudness', e mais 'acousticness'
- Cluster 1: Maior 'instrumentalness' e 'valence' e menor 'loudness'
- Cluster 2: Alto 'liveness', 'speechiness', 'loudness' e 'energy'
- Cluster 3: Maior 'instrumentalness', 'liveness' e 'acousticness'
- Cluster 4: 'Speechiness', 'danceability' e 'liveness' altos, menor duração

2.3.2 Cluster X Características

```
[32]: df_dados_normalizados1_media.T.plot(kind='bar')
plt.title('Clusters por característica')
plt.legend(title='Cluster', bbox_to_anchor=(1.0, 1.0))
plt.tight_layout()
plt.show()
```



2.3.3 Histogramas

Utilizando os dados reais agrupados vamos analisar a frequência por característica, mas para ser computacionalmente viável, vamos pegar uma amostra usando a função `tamanho_amostra`, para um intervalo com 95% de confiança e 5% de margem de erro.

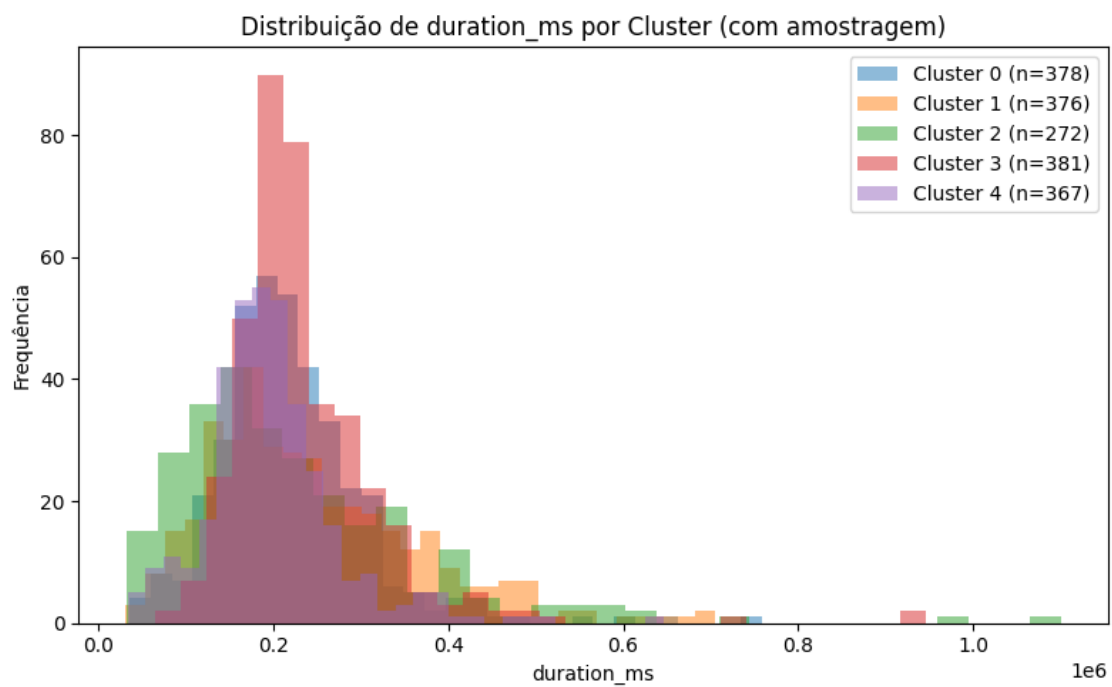
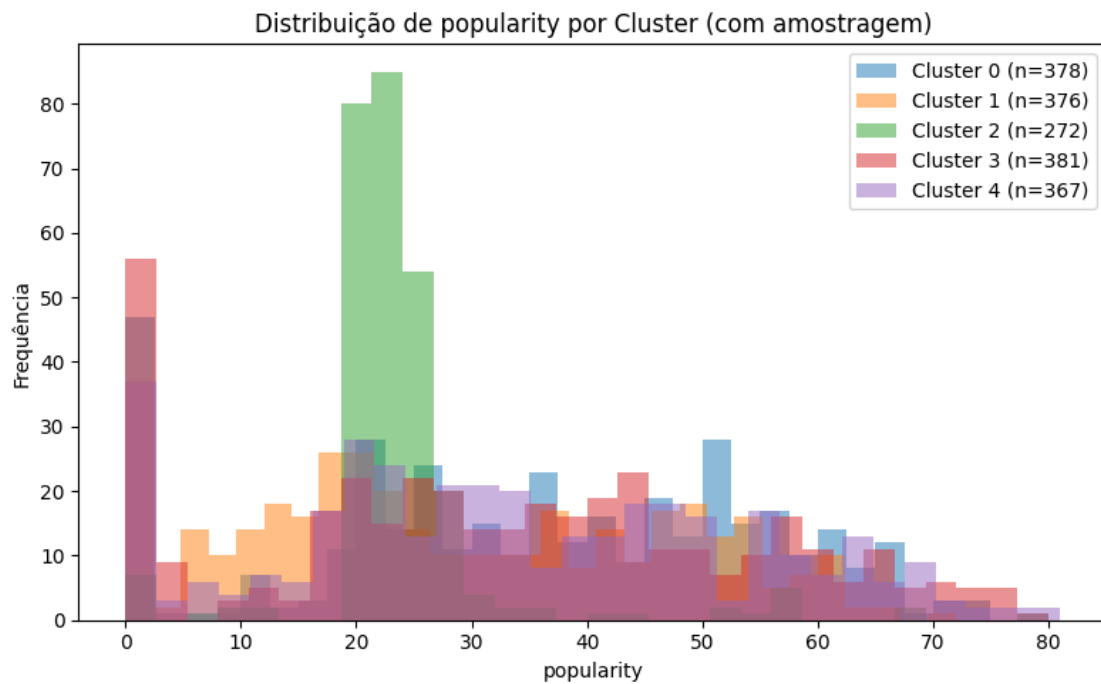
```
[33]: import matplotlib.pyplot as plt

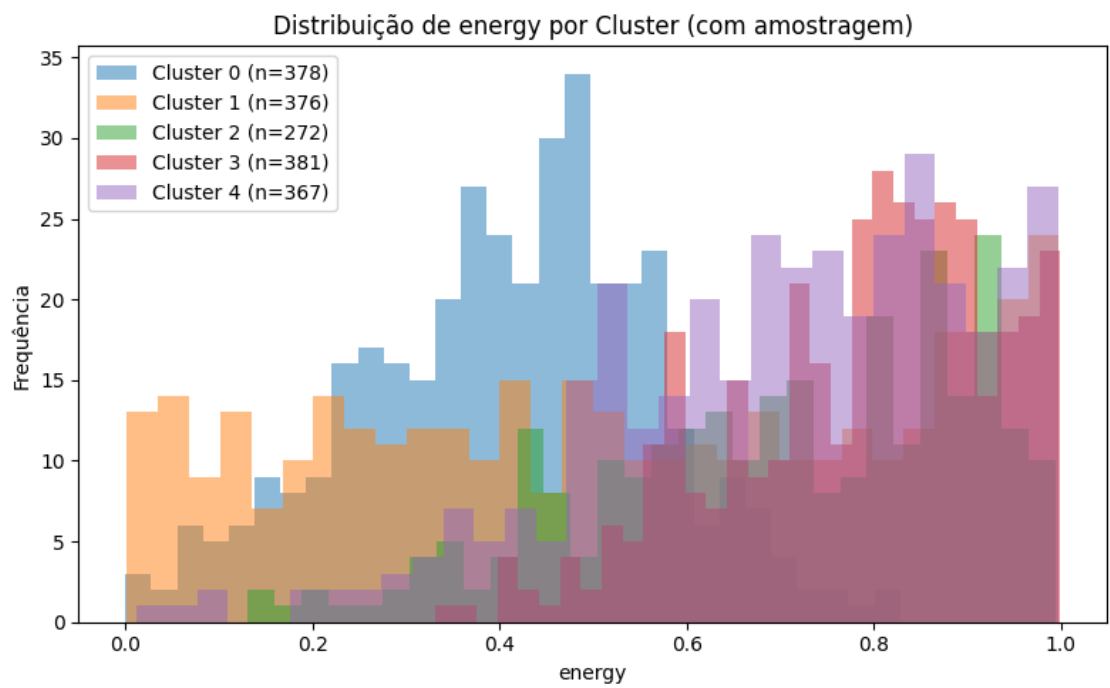
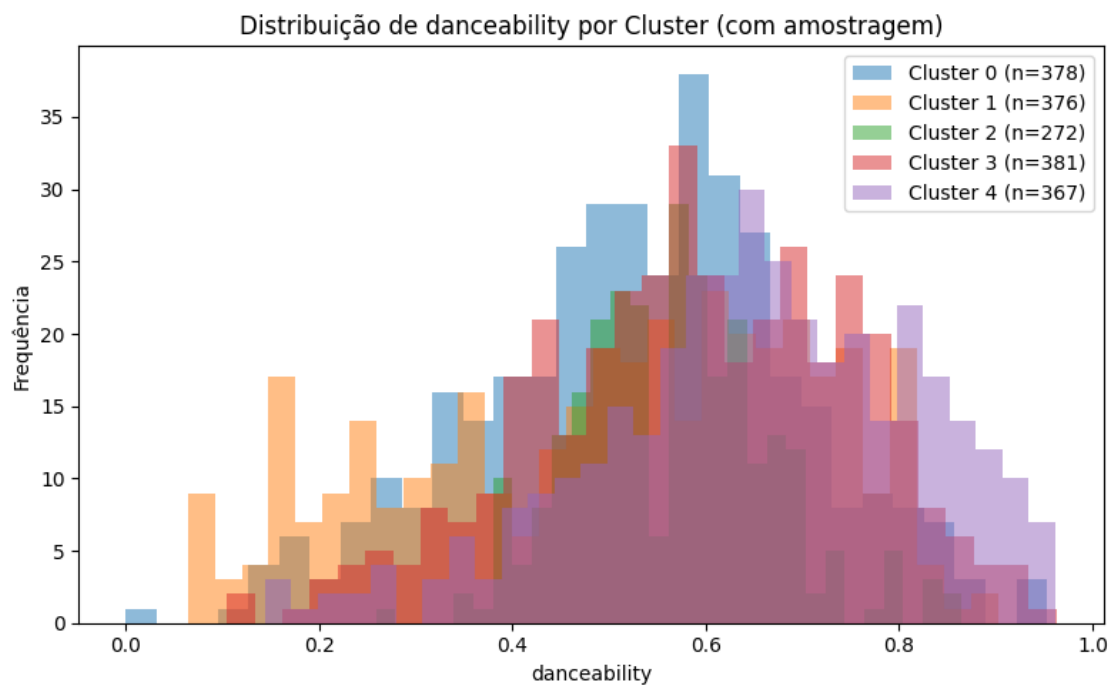
for feature in df_dados.columns.drop('cluster'):
    plt.figure(figsize=(8, 5))

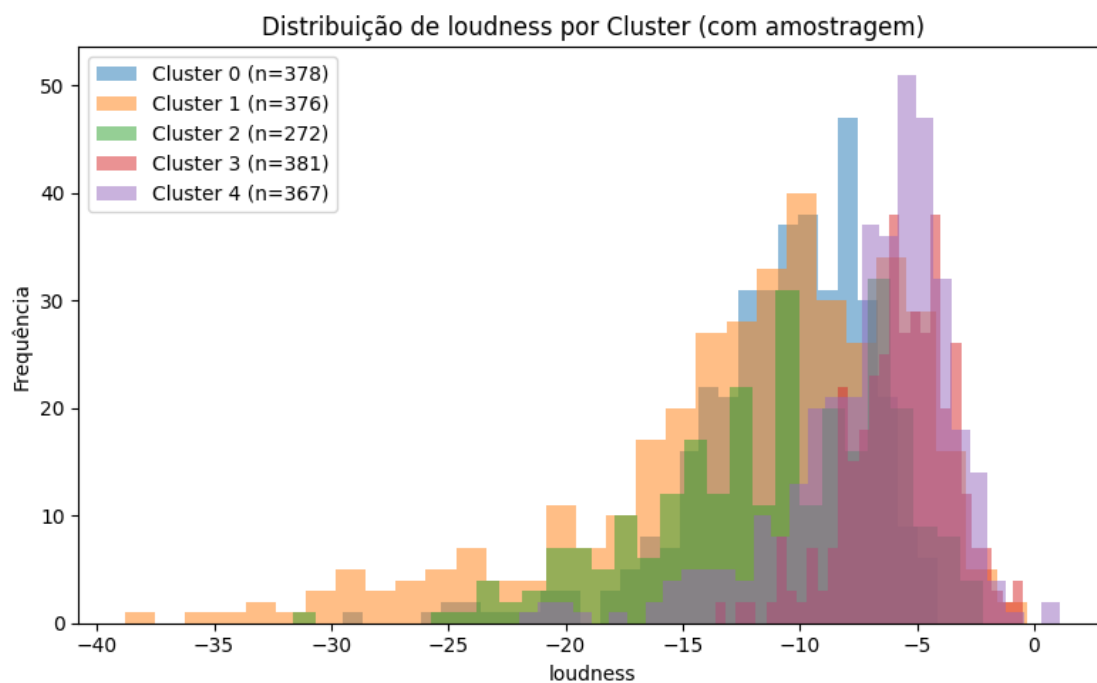
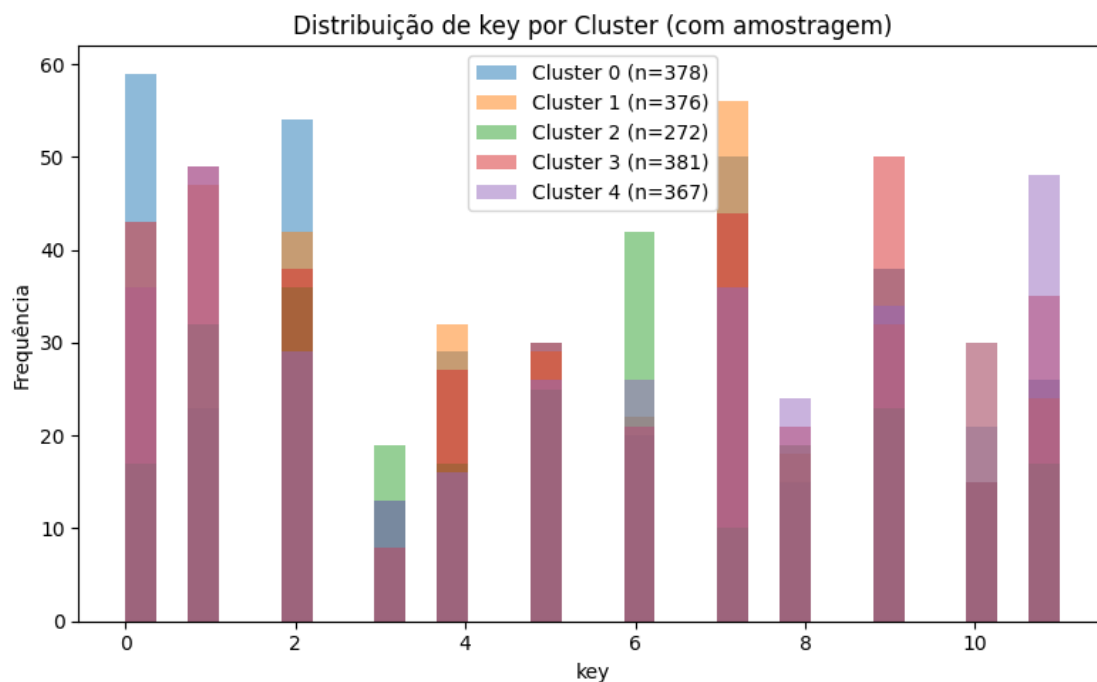
    for cluster_id in sorted(df_dados['cluster'].unique()):
        subset = df_dados[df_dados['cluster'] == cluster_id]
        tamanho = tamanho_amostra(len(subset))
        amostra = subset.sample(n=min(tamanho, len(subset)), random_state=42)
        # Garante que não passa o tamanho real
        plt.hist(amostra[feature], bins=30, alpha=0.5, label=f'Cluster_{cluster_id} (n={len(amostra)})')

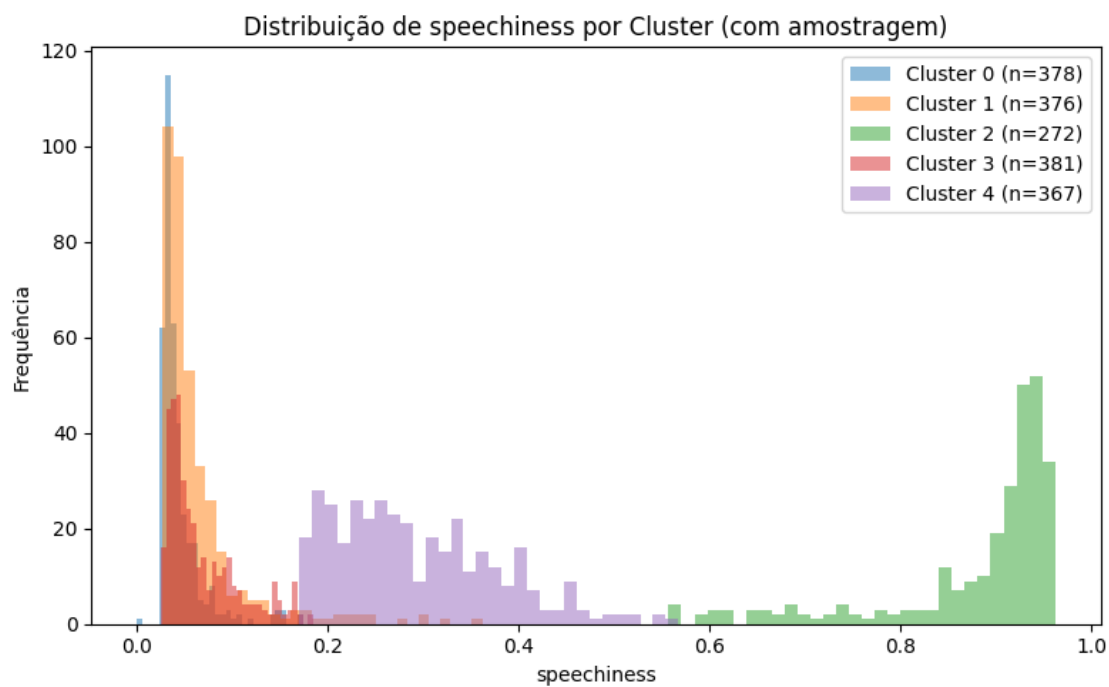
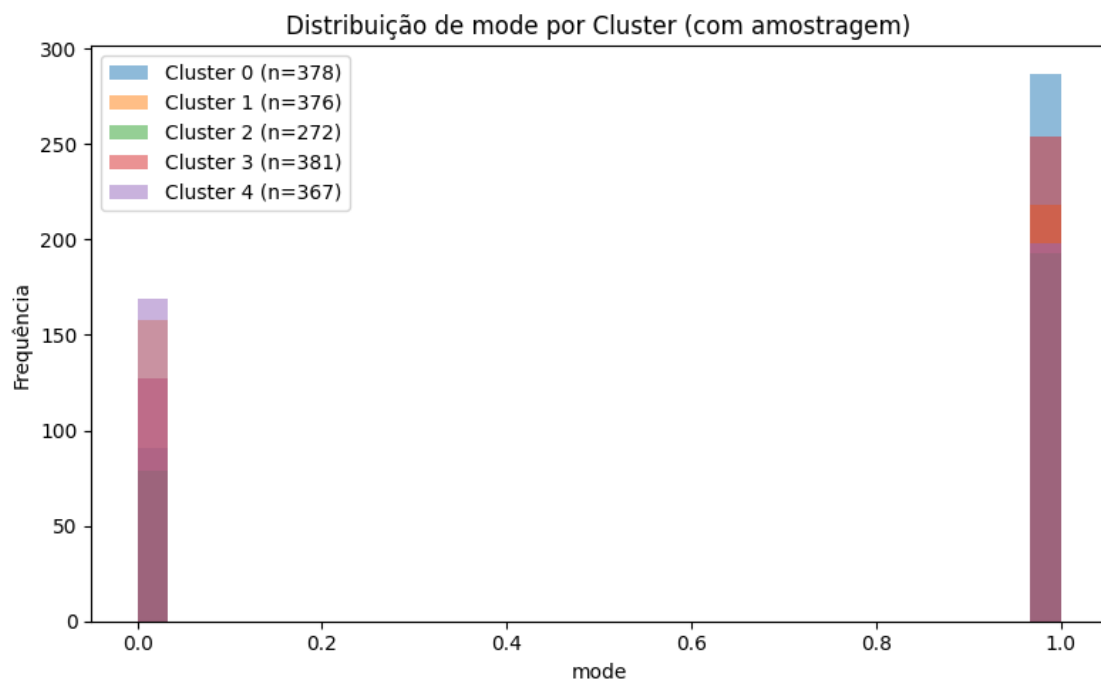
    plt.title(f'Distribuição de {feature} por Cluster (com amostragem)')
    plt.xlabel(feature)
```

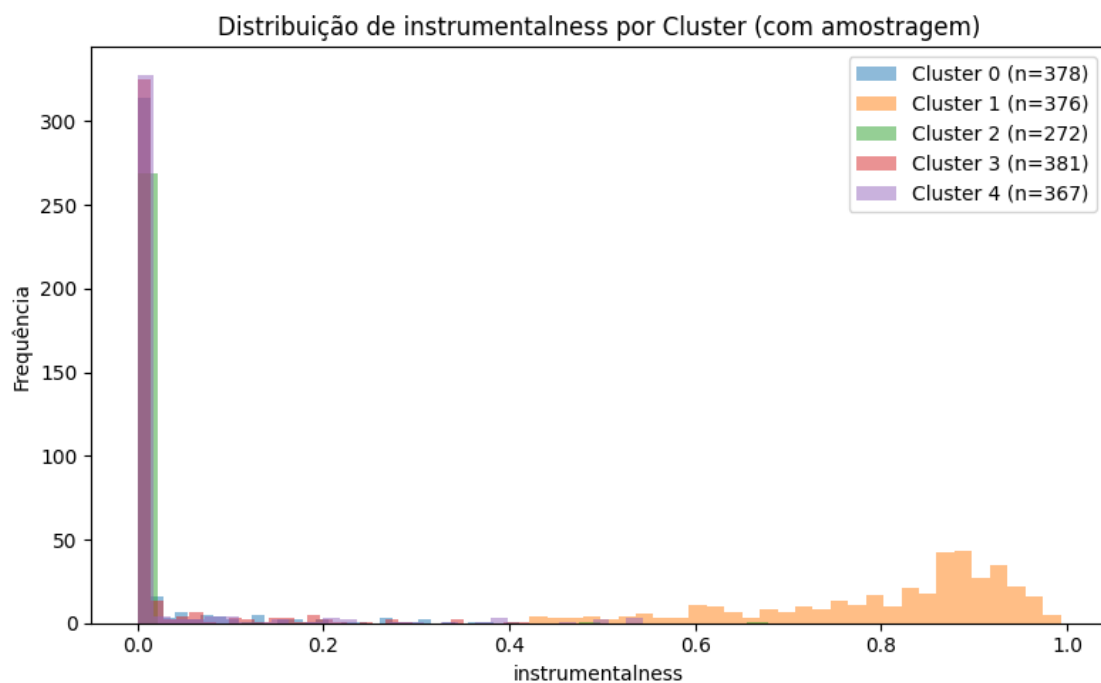
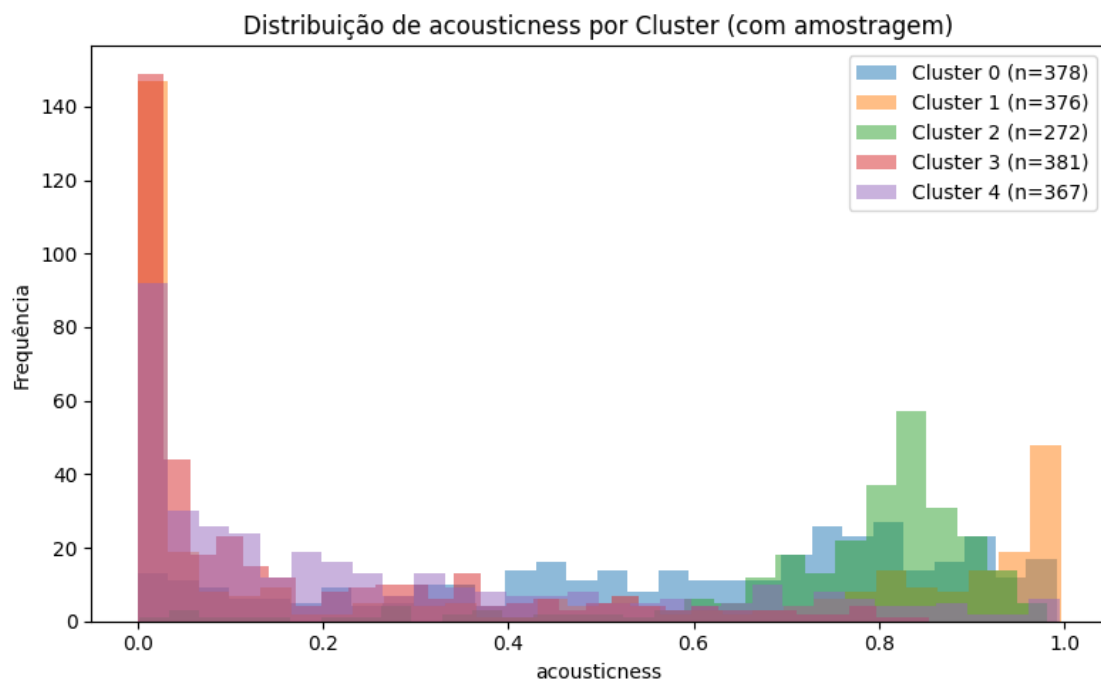
```
plt.ylabel('Frequência')
plt.legend()
plt.tight_layout()
plt.show()
```

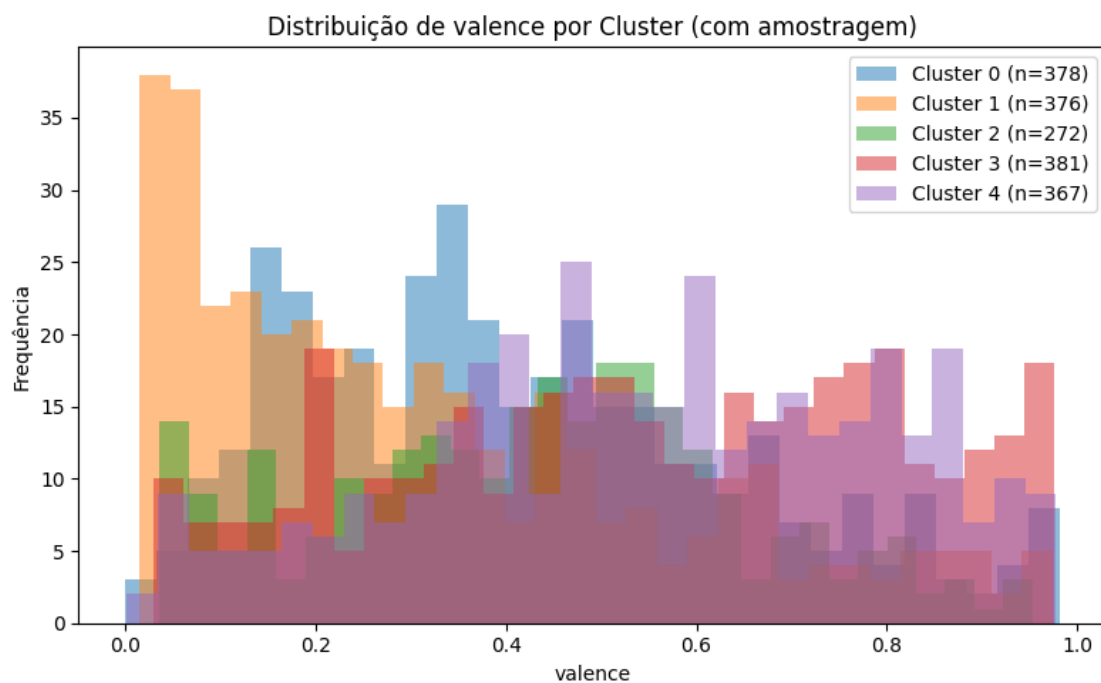
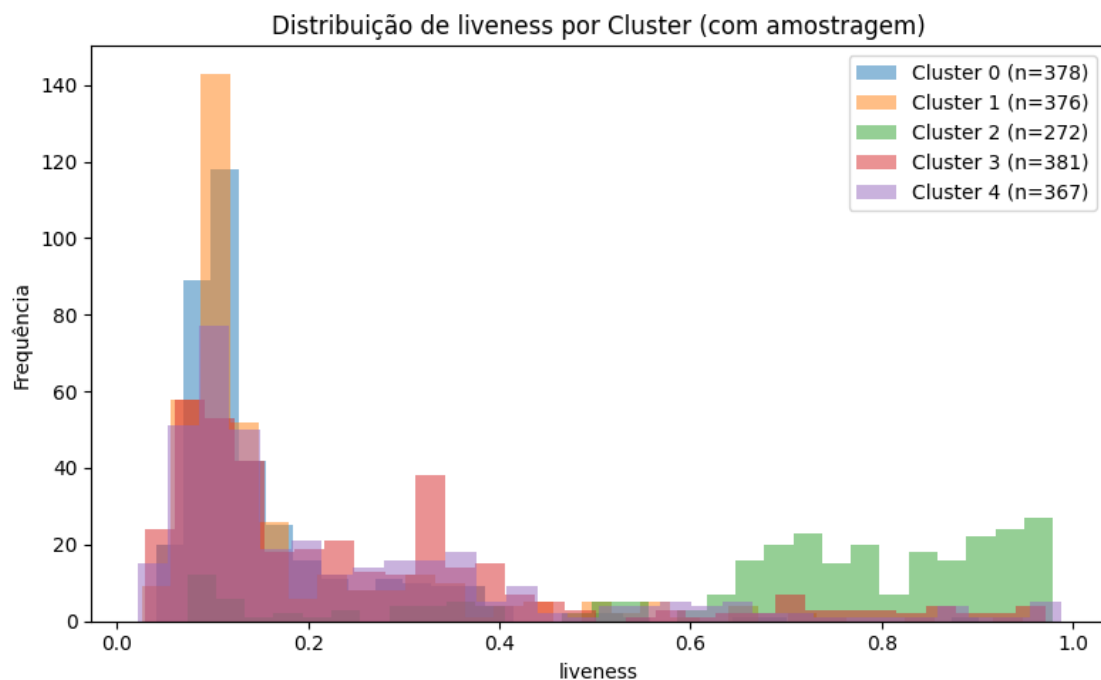


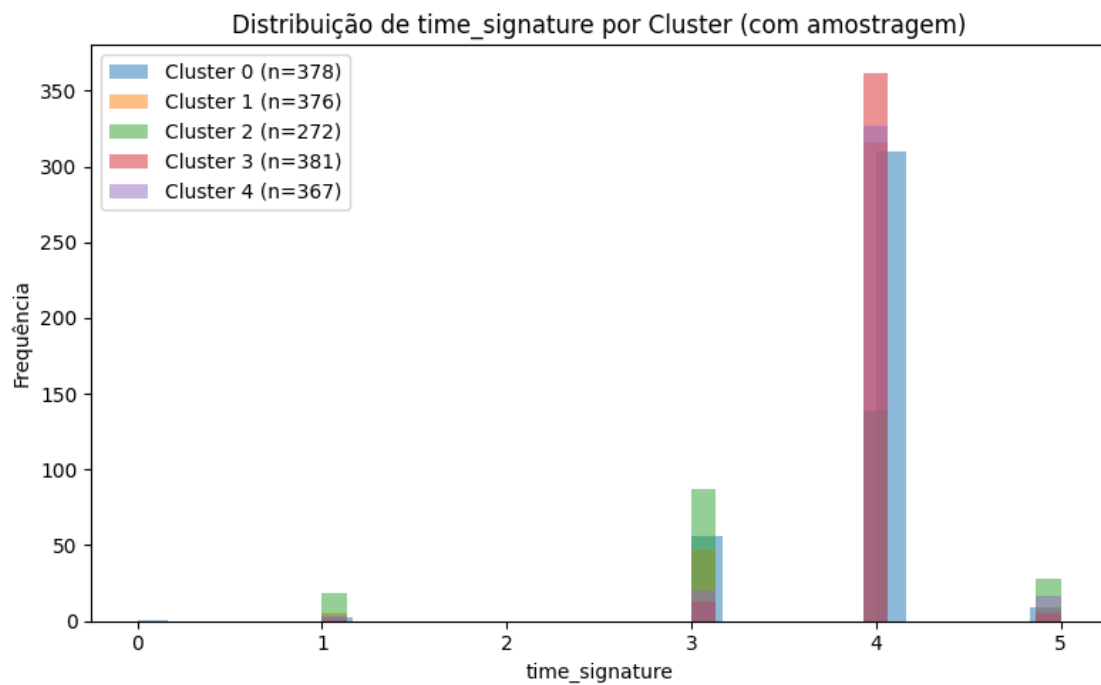
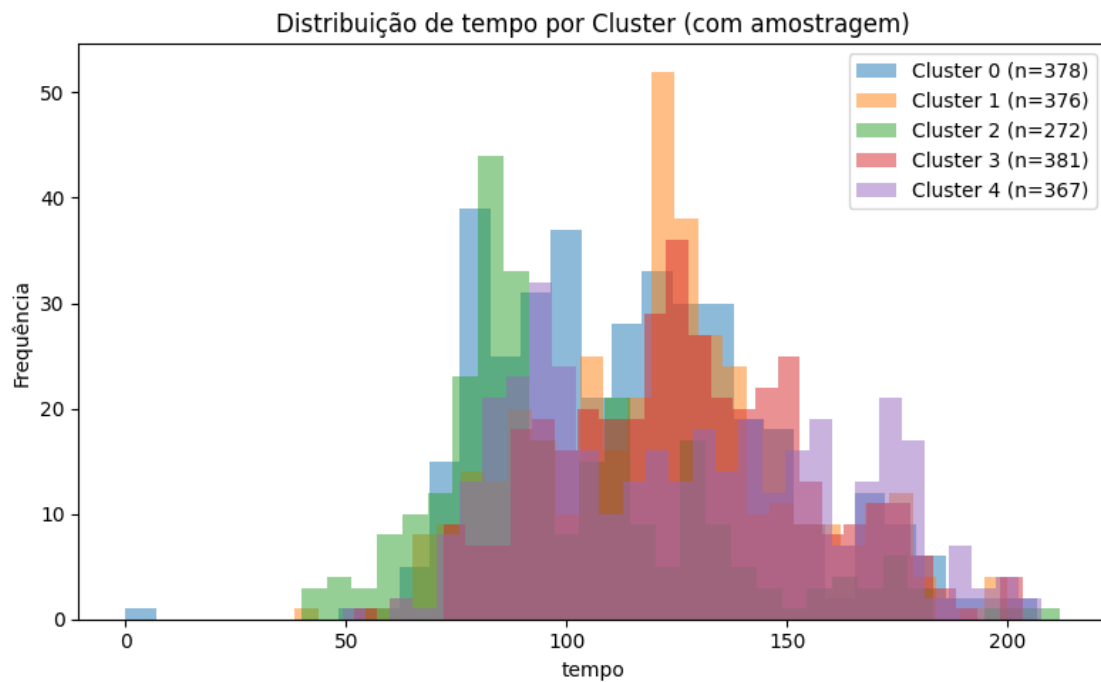












2.4 Musica

Agora que temos as musicas agrupadas por similaridade podemos recomendar 10 músicas mais parecidas dentro do grupo com base na distância

Vamos usar como exemplo a seguinte musica:

```
[34]: df_musicas_rs[df_musicas['track_id'] ==  
      ↪ '2aibwv5hGXSgw7Yru8IYT0'][['artists', 'track_name', 'track_genre', 'cluster']]
```

```
[34]:
```

	artists	track_name	track_genre	cluster
2109	Red Hot Chili Peppers	Snow (Hey Oh)	alt-rock	3

```
[35]: recomendar_musicas('2aibwv5hGXSgw7Yru8IYT0', df_dados_normalizados1,  
      ↪ df_musicas_rs)
```

Cluster: 3

```
[35]:
```

	track_name \
2110	Monster
44195	Fake It
11266	cotton candy
83624	Runaway - Radio Edit
53762	Under Control (feat. Hurts)
80561	Ek Toh Kum Zindagani (From "Marjaavaan")
8256	Ex's & Oh's
30764	What Would You Do?
30562	Head & Heart (feat. MNEK)
56383	SUPERMODEL

	artists	track_genre
2110	Skillet	alt-rock
44195	Seether	grunge
11266	YUNGBLUD	british
83624	Pierce Fulton	progressive-house
53762	Calvin Harris;Alesso;Hurts	house
80561	Neha Kakkar;Yash Narvekar;Tanishk Bagchi	pop-film
8256	Elle King	blues
30764	Joel Corry;David Guetta;Bryson Tiller	edm
30562	Joel Corry;MNEK	edm
56383	Måneskin	indie-pop

E agora vamos repetir o mesmo processo para os outros modelos de normalização, e continuaremos utilizando a música *Red Hot Chili Peppers - Snow (Hey Oh)* como exemplo

3 Min Max Scale

```
[36]: from sklearn.preprocessing import minmax_scale
escala2 = minmax_scale(df_dados)
df_dados_normalizados2 = pd.DataFrame(escala2, columns=df_dados.columns[:])
df_dados_normalizados2
```

```
[36]:
```

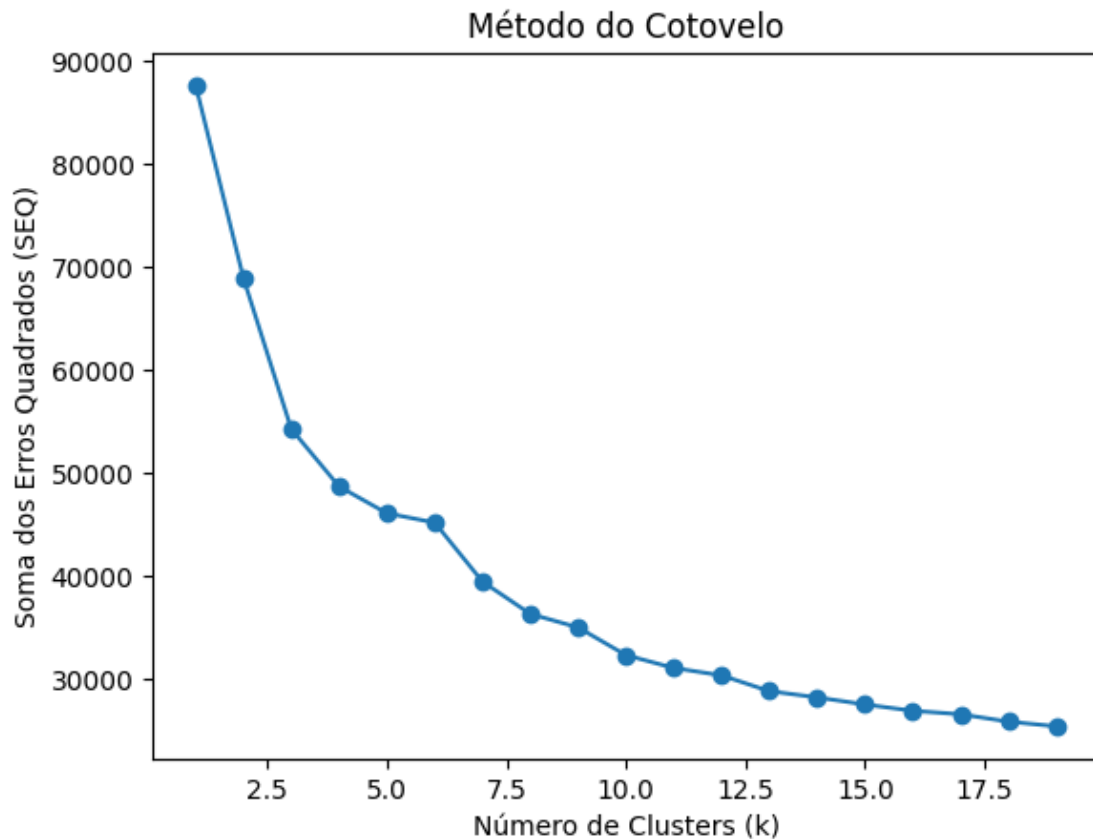
	popularity	duration_ms	danceability	energy	key	loudness	\
0	0.73	0.042473	0.686294	0.4610	0.090909	0.791392	
1	0.55	0.026971	0.426396	0.1660	0.090909	0.597377	
2	0.57	0.038679	0.444670	0.3590	0.000000	0.736123	
3	0.71	0.036978	0.270051	0.0596	0.000000	0.573701	
4	0.82	0.036389	0.627411	0.4430	0.181818	0.737103	
...	
89735	0.21	0.071990	0.174619	0.2350	0.454545	0.612952	
89736	0.22	0.071990	0.176650	0.1170	0.000000	0.577345	
89737	0.22	0.050276	0.638579	0.3290	0.000000	0.714648	
89738	0.41	0.052653	0.595939	0.5060	0.636364	0.714759	
89739	0.22	0.044608	0.534010	0.4870	0.090909	0.727429	
...	
	mode	speechiness	acousticness	instrumentalness	liveness	valence	\
0	0.0	0.148187	0.032329	0.000001	0.3580	0.718593	
1	1.0	0.079067	0.927711	0.000006	0.1010	0.268342	
2	1.0	0.057720	0.210843	0.000000	0.1170	0.120603	
3	1.0	0.037617	0.908635	0.000071	0.1320	0.143719	
4	1.0	0.054508	0.470884	0.000000	0.0829	0.167839	
...	
89735	1.0	0.043731	0.642570	0.928000	0.0863	0.034070	
89736	0.0	0.041554	0.997992	0.976000	0.1050	0.035176	
89737	0.0	0.043523	0.870482	0.000000	0.0839	0.746734	
89738	1.0	0.030777	0.382530	0.000000	0.2700	0.415075	
89739	0.0	0.075130	0.683735	0.000000	0.0893	0.711558	
...	
	tempo	time_signature	cluster				
0	0.361245	0.8	0.75				
1	0.318397	0.8	0.00				
2	0.313643	0.8	0.00				
3	0.746758	0.6	0.00				
4	0.492863	0.8	0.00				
...				
89735	0.517705	1.0	0.25				
89736	0.350242	0.8	0.25				
89737	0.543933	0.8	0.00				
89738	0.558651	0.8	0.00				
89739	0.325420	0.8	0.00				

[89740 rows x 15 columns]

Aplicando o método do cotovelo

```
[37]: soma_e2 = calcular_soma_erros(escala2)

[38]: plt.plot(range(1, 20), soma_e2, marker='o')
plt.xlabel('Número de Clusters (k)')
plt.ylabel('Soma dos Erros Quadrados (SEQ)')
plt.title('Método do Cotovelo')
plt.show()
```



Melhor número de clusters

```
[39]: nc2 = optimal_number_of_clusters(soma_e2)
nc2

[39]: 5

[40]: from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=nc2, random_state=14)
clusters = kmeans.fit_predict(escala2)
```

```
df_musicas_minmax = df_musicas.copy()
df_musicas_minmax['cluster'] = clusters
df_dados['cluster'] = clusters
df_dados_normalizados2['cluster'] = clusters
```

3.1 Plotagem 2D e 3D dos clusters

```
[41]: from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

pca = PCA(n_components=2)
X_pca = pca.fit_transform(escala2)

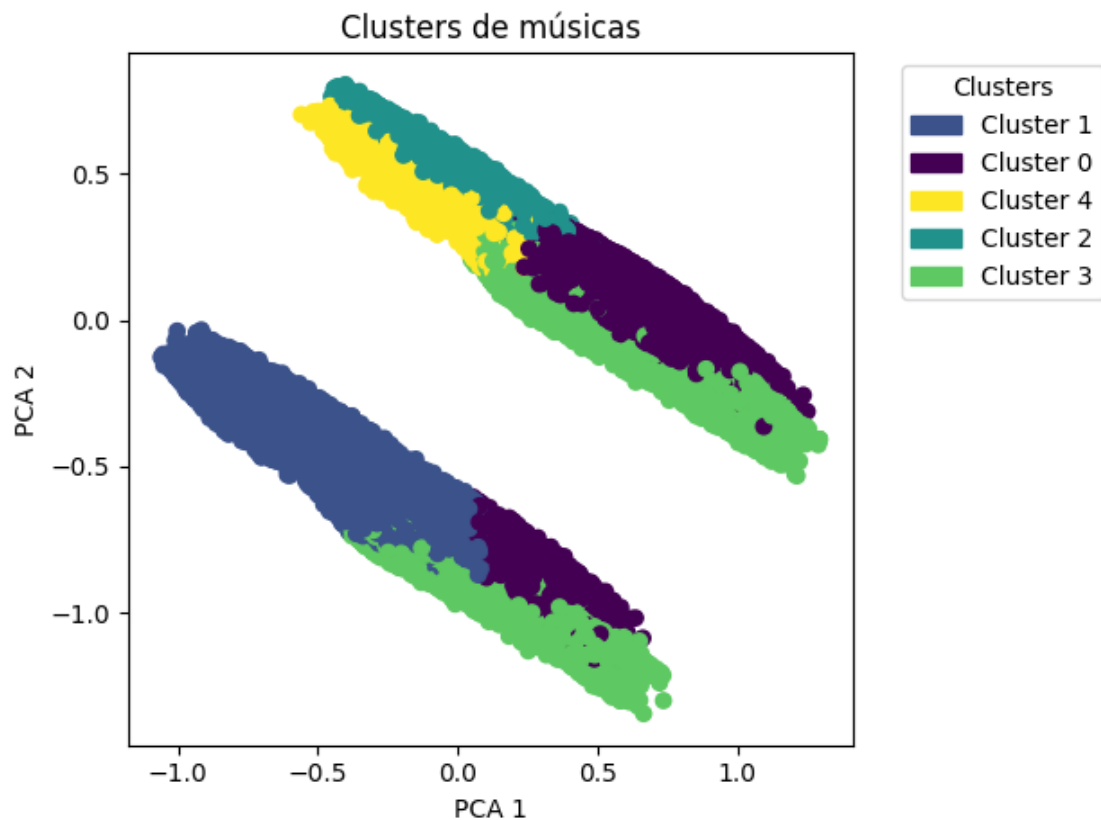
# Plot
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1],
    ↪c=df_dados_normalizados2['cluster'], cmap='viridis')

# Legenda manual
clusters = df_dados_normalizados2['cluster'].unique()
colors = scatter.cmap(scatter.norm(clusters))

handles = [mpatches.Patch(color=colors[i], label=f'Cluster {clusters[i]}') for
    ↪i in range(len(clusters))]

plt.legend(handles=handles, title='Clusters', bbox_to_anchor=(1.05, 1),
    ↪loc='upper left')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.title('Clusters de músicas')

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

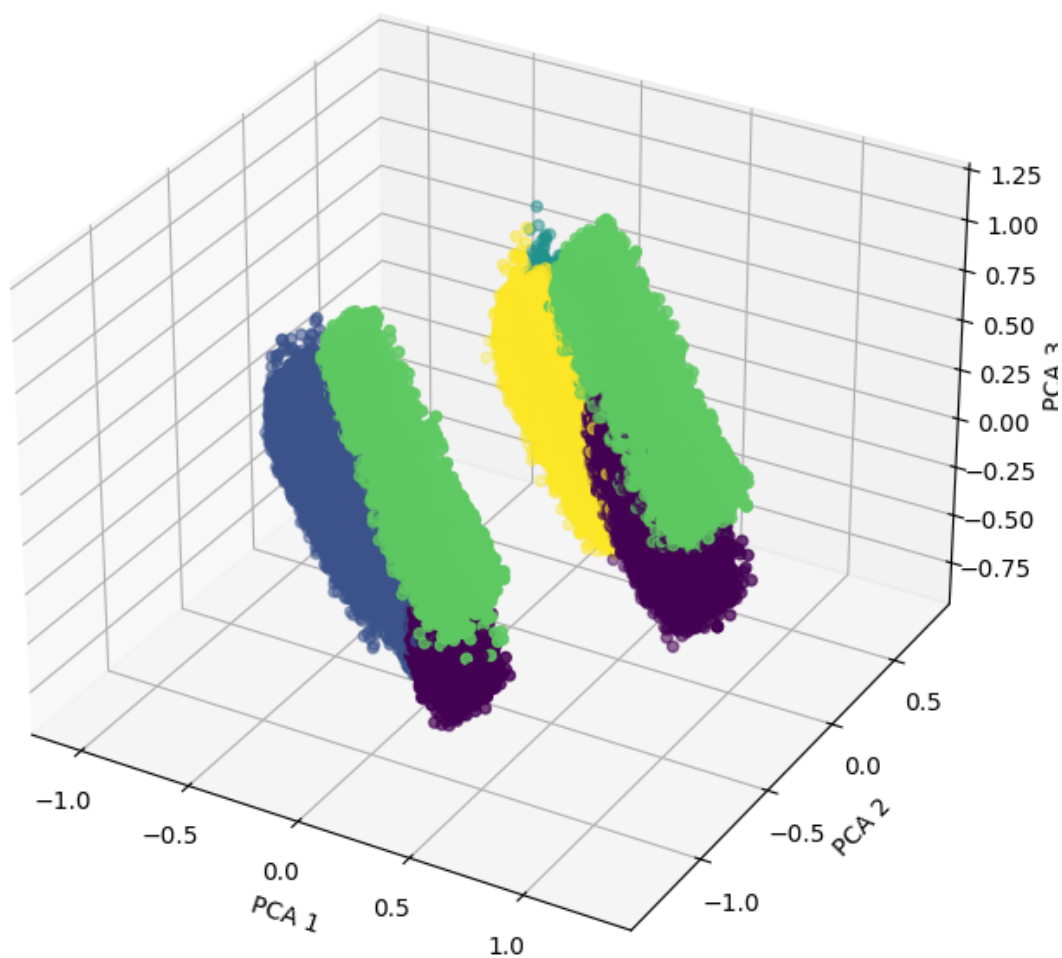


```
[42]: from mpl_toolkits.mplot3d import Axes3D

pca = PCA(n_components=3)
X_pca_3d = pca.fit_transform(escala2)

fig = plt.figure(figsize=(10,8))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X_pca_3d[:, 0], X_pca_3d[:, 1], X_pca_3d[:, 2],
                    c=df_dados_normalizados2['cluster'], cmap='viridis')
ax.set_xlabel('PCA 1')
ax.set_ylabel('PCA 2')
ax.set_zlabel('PCA 3')
plt.title('Clusters em 3D com PCA')
plt.show()
```

Clusters em 3D com PCA



3.2 Analisando o agrupamento

Quantas músicas tem em cada cluster

```
[43]: df_dados_normalizados2['cluster'].value_counts()
```

```
[43]: cluster
1    24439
0    19594
4    17207
3    14531
2    13969
Name: count, dtype: int64
```

3.2.1 Dados Originais

```
[44]: df_dados_media = df_dados.groupby('cluster').mean(numeric_only=True) #Para
      ↪ histogramas
      df_dados_media
```

```
[44]:
```

	popularity	duration_ms	danceability	energy	key	\
cluster						
0	32.383893	217019.983617	0.525607	0.380379	4.886904	
1	35.350996	232453.544376	0.608186	0.741494	6.141331	
2	34.556303	224643.552724	0.587936	0.774527	1.430310	
3	28.161448	252224.914321	0.477336	0.498026	5.004886	
4	34.221945	222413.414250	0.589153	0.773267	7.880281	

	loudness	mode	speechiness	acousticness	instrumentalness	\
cluster						
0	-11.055943	0.845106	0.048566	0.646504	0.022783	
1	-6.552397	0.000000	0.105628	0.184148	0.080628	
2	-5.977167	1.000000	0.110073	0.182374	0.032919	
3	-13.827622	0.648751	0.064213	0.444474	0.841625	
4	-5.899418	1.000000	0.107127	0.190973	0.026497	

	liveness	valence	tempo	time_signature
cluster				
0	0.162928	0.412569	113.253678	3.801572
1	0.225470	0.520812	124.491413	3.955031
2	0.266191	0.554182	127.931355	3.954972
3	0.179173	0.263937	116.950708	3.806896
4	0.258402	0.566165	128.173123	3.954495

3.2.2 Dados Normalizados

```
[45]: df_dados_normalizados2_media = df_dados_normalizados2.groupby('cluster').mean()
      ↪ # Para gráfico de médias
      df_dados_normalizados2_media
```

```
[45]:
```

	popularity	duration_ms	danceability	energy	key	loudness	\
cluster							
0	0.323839	0.039863	0.533612	0.380379	0.444264	0.711671	
1	0.353510	0.042815	0.617447	0.741494	0.558303	0.794973	
2	0.345563	0.041321	0.596890	0.774527	0.130028	0.805613	
3	0.281614	0.046596	0.484605	0.498026	0.454990	0.660403	
4	0.342219	0.040895	0.598125	0.773267	0.716389	0.807051	

	mode	speechiness	acousticness	instrumentalness	liveness	\
cluster						
0	0.845106	0.050328	0.649100	0.022783	0.162928	
1	0.000000	0.109459	0.184887	0.080628	0.225470	

2	1.000000	0.114065	0.183106	0.032919	0.266191
3	0.648751	0.066542	0.446259	0.841625	0.179173
4	1.000000	0.111013	0.191740	0.026497	0.258402

	valence	tempo	time_signature
cluster			
0	0.414642	0.465352	0.760314
1	0.523430	0.511527	0.791006
2	0.556967	0.525662	0.790994
3	0.265263	0.480543	0.761379
4	0.569010	0.526655	0.790899

3.2.3 Gênero de músicas por cluster

```
[46]: genero_por_cluster = df_musicas_minmax.groupby('cluster').apply(
        lambda x: x['track_genre'].value_counts().head(6),
        include_groups=False
    )
genero_por_cluster
```

```
[46]: cluster  track_genre
0           honky-tonk      836
           romance        737
           cantopop        707
           acoustic        658
           tango           647
           opera           647
1           turkish        494
           dancehall       471
           hardstyle       467
           deep-house      445
           dance           440
           afrobeat        423
2           j-idol         285
           power-pop       279
           comedy          275
           kids            272
           forro           261
           sertanejo       249
3           study          867
           sleep           801
           new-age         776
           ambient         749
           idm             695
           minimal-techno  662
4           j-idol         382
           party           363
```

```

    forro          362
    sertanejo      326
    country        304
    alt-rock       303
Name: count, dtype: int64

```

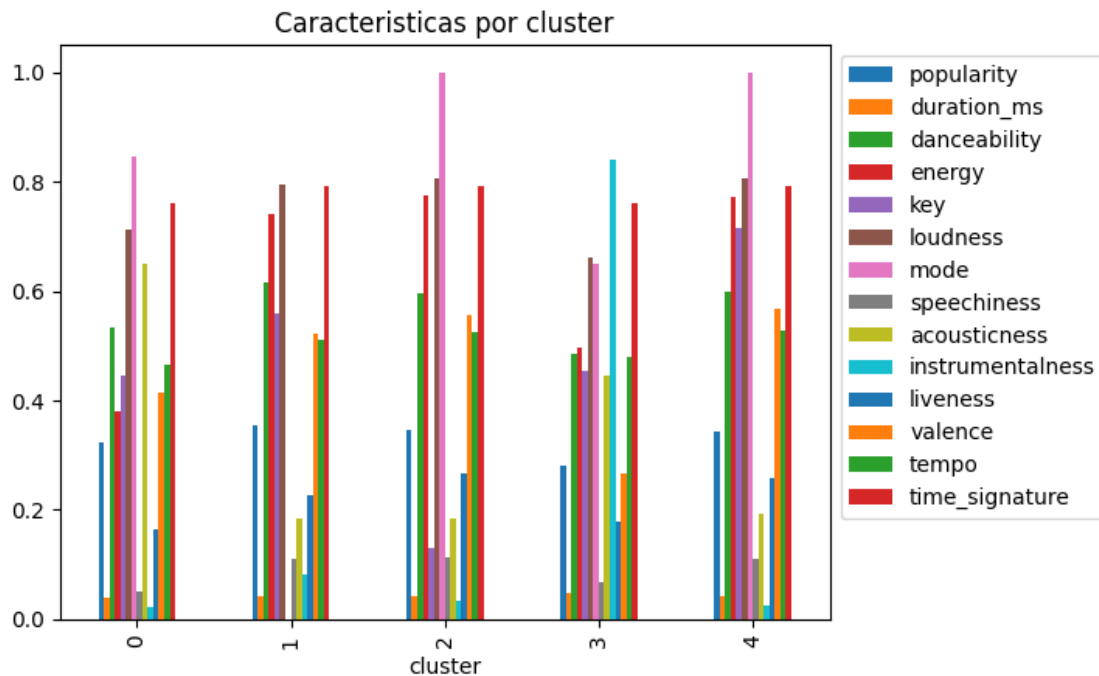
3.3 Gráficos

3.3.1 Características x Cluster

```

[47]: df_dados_normalizados2_media.plot(kind='bar')
plt.title('Características por cluster')
plt.legend(bbox_to_anchor=(1.0, 1.0))
plt.show()

```



Olhando os gráficos acima, podemos perceber que os cluster 0,1,3,4 possuem 'mode' = 1, e os clusters 2,5,6 tendo mode = 0.

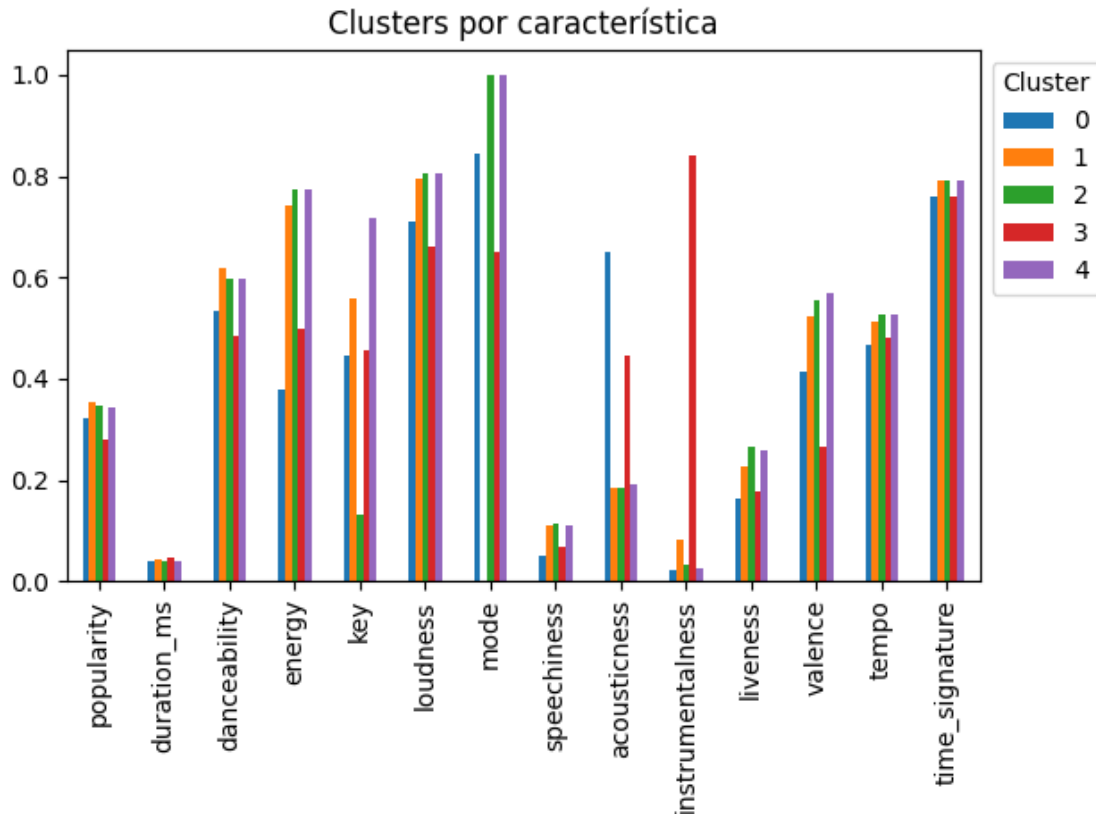
Os clusters 2 e 4 possuem 'instrumentalness' alto, indicando serem músicas clássica, ambiente, etc

Os clusters 0 e 5 possuem 'acousticness' alto e 'energy' baixo, indicando ter faixas predominantemente acusticas.

Os cluster 1, 3, 6 possuem 'valence' alto, indicando músicas animadas e de sentimento positivo

3.3.2 Cluster x Características

```
[48]: df_dados_normalizados2_media.T.plot(kind='bar')
plt.title('Clusters por característica')
plt.legend(title='Cluster', bbox_to_anchor=(1.0, 1.0))
plt.tight_layout()
plt.show()
```



Analisando o gráfico acima

Os clusters 2,4 possuem menos populares e com maior duração

Os clusters 0,5 possuem músicas menos dançantes

Os clusters 0,5 possuem músicas mais calmas, menos barulhentas, mais músicas acústicas, e com menor bpm

Os clusters 1,3,6 possuem músicas que tem mais fala

Os clusters 2,4 possuem mais músicas instrumentais e ambiente

Os clusters 1,3,6 possuem mais músicas ao vivo e mais felizes

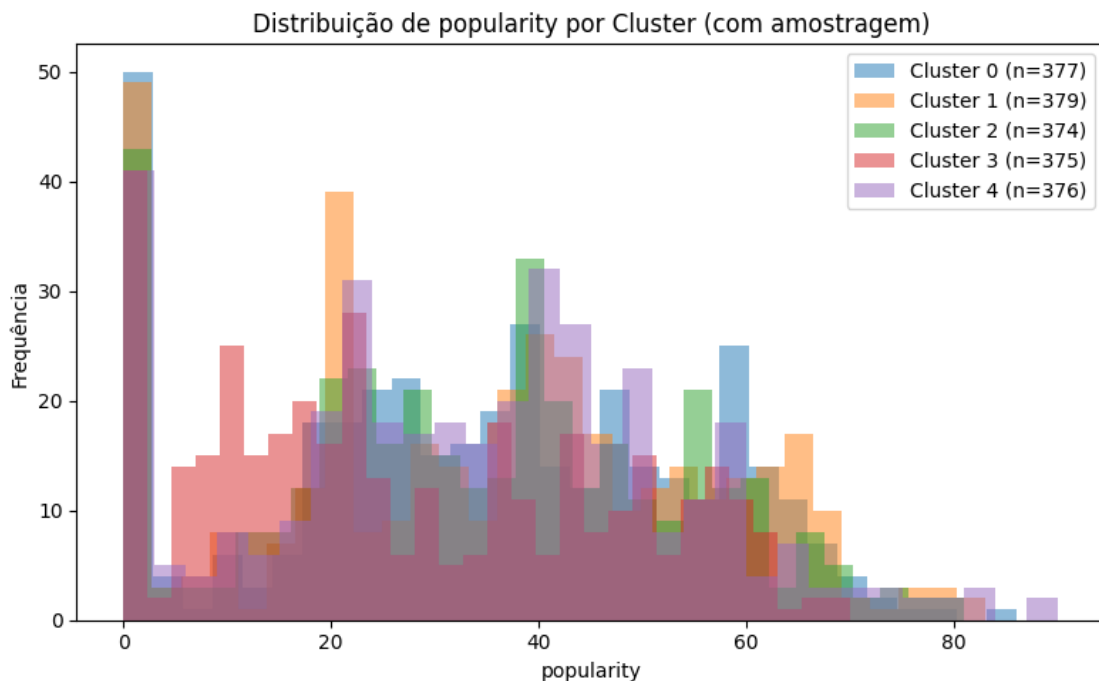
3.3.3 Histogramas

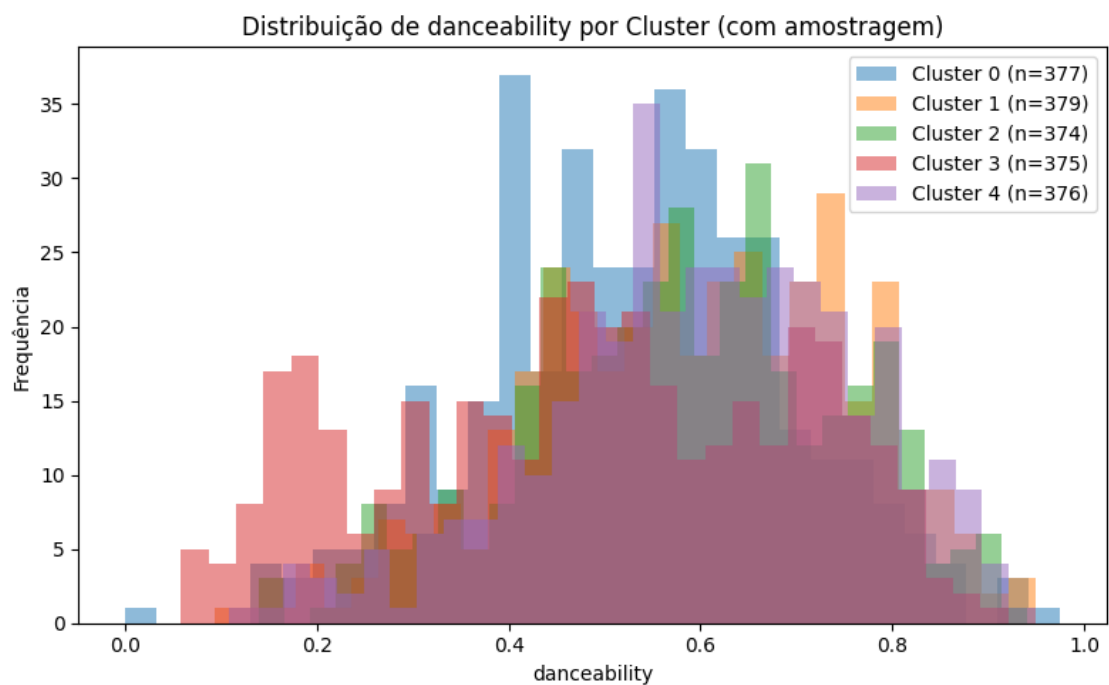
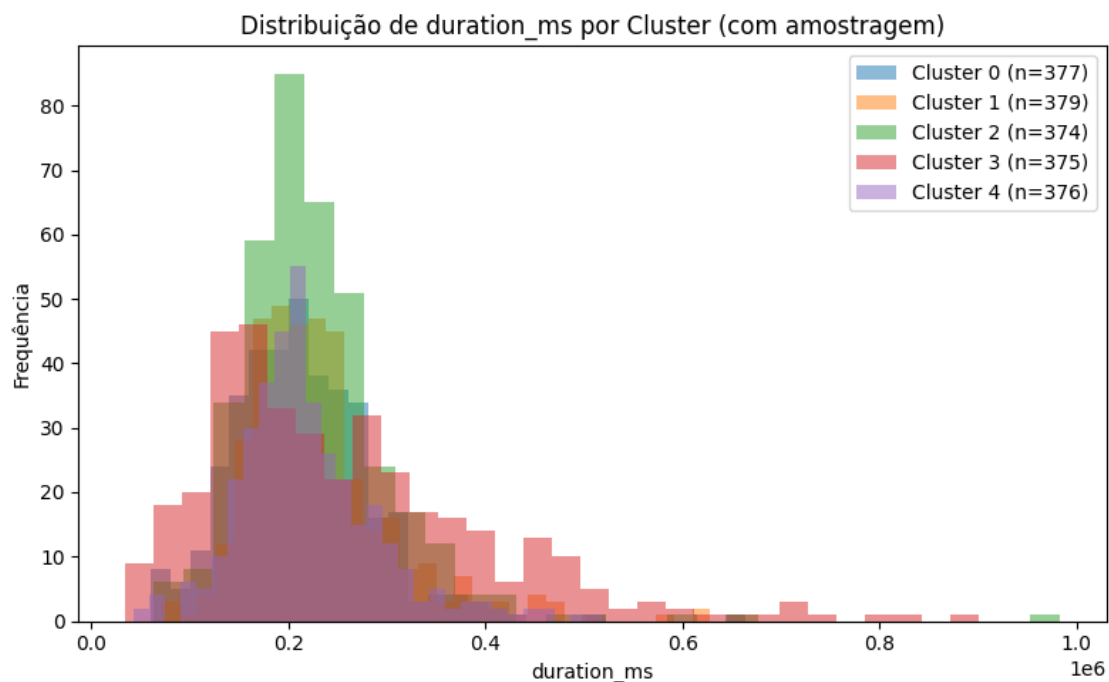
```
[49]: import matplotlib.pyplot as plt

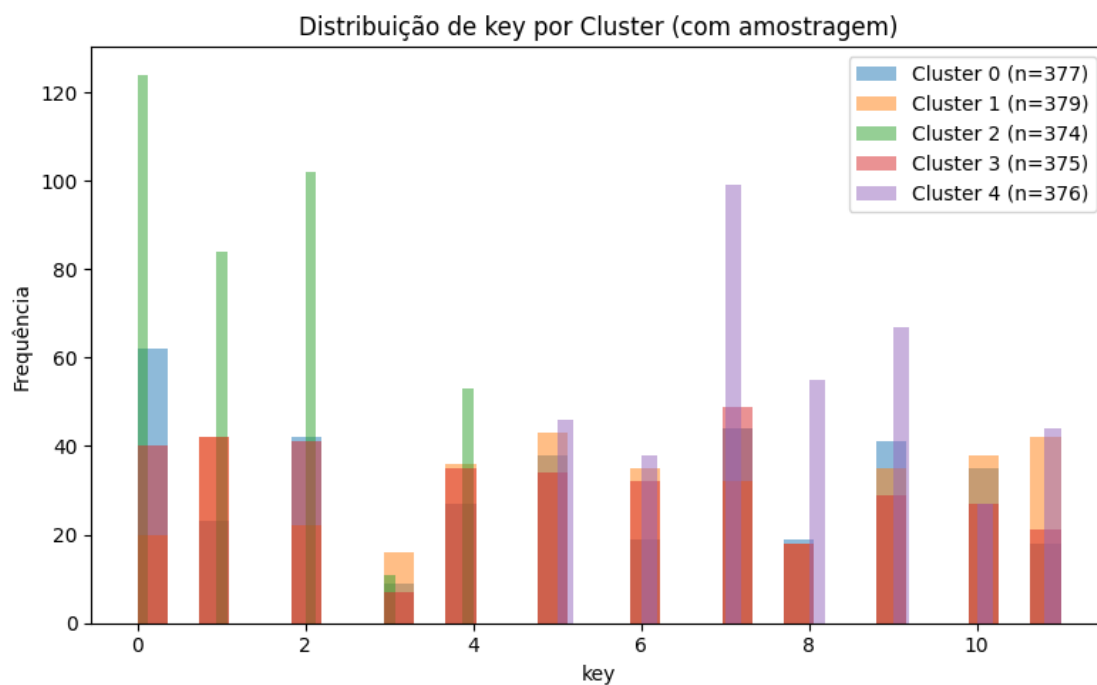
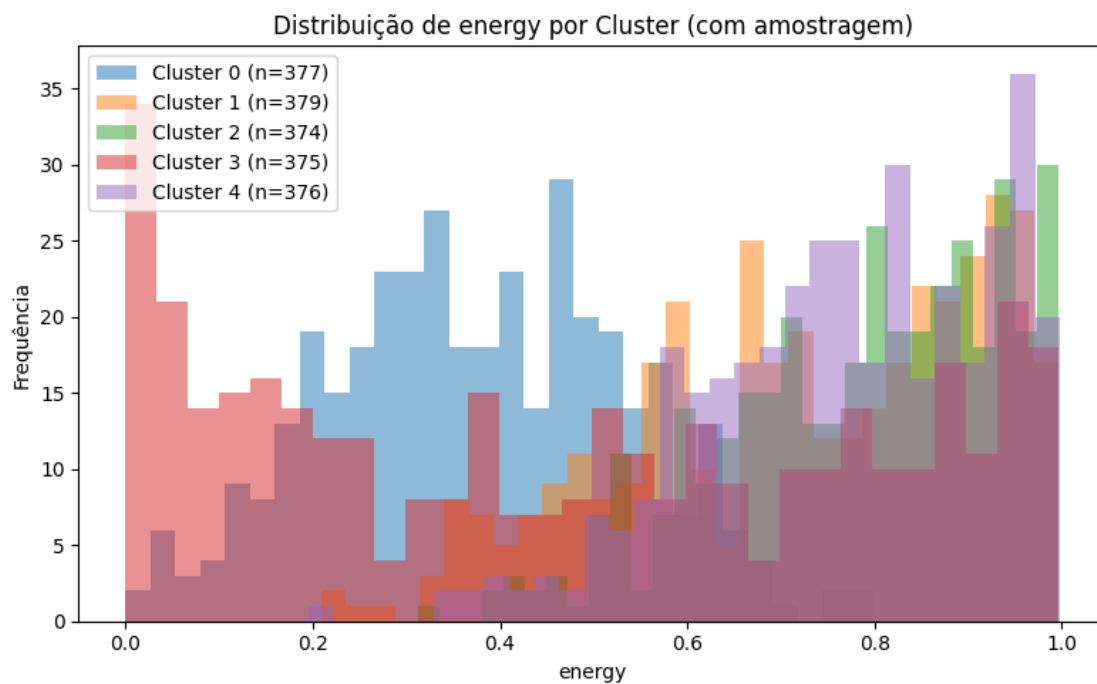
for feature in df_dados.columns.drop('cluster'):
    plt.figure(figsize=(8, 5))

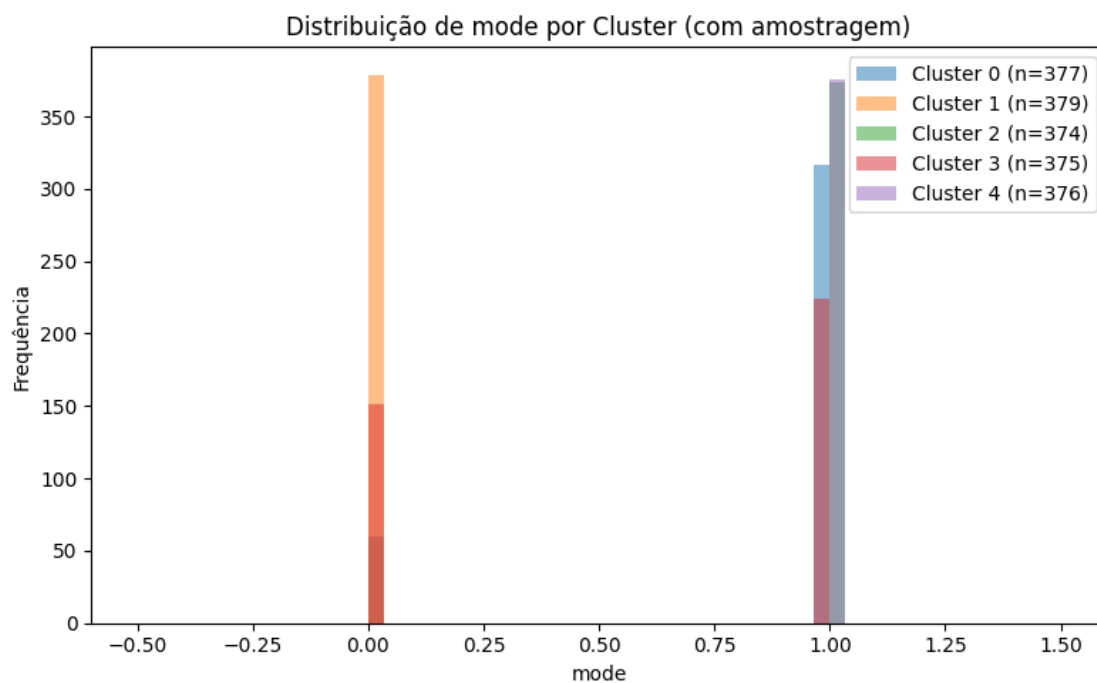
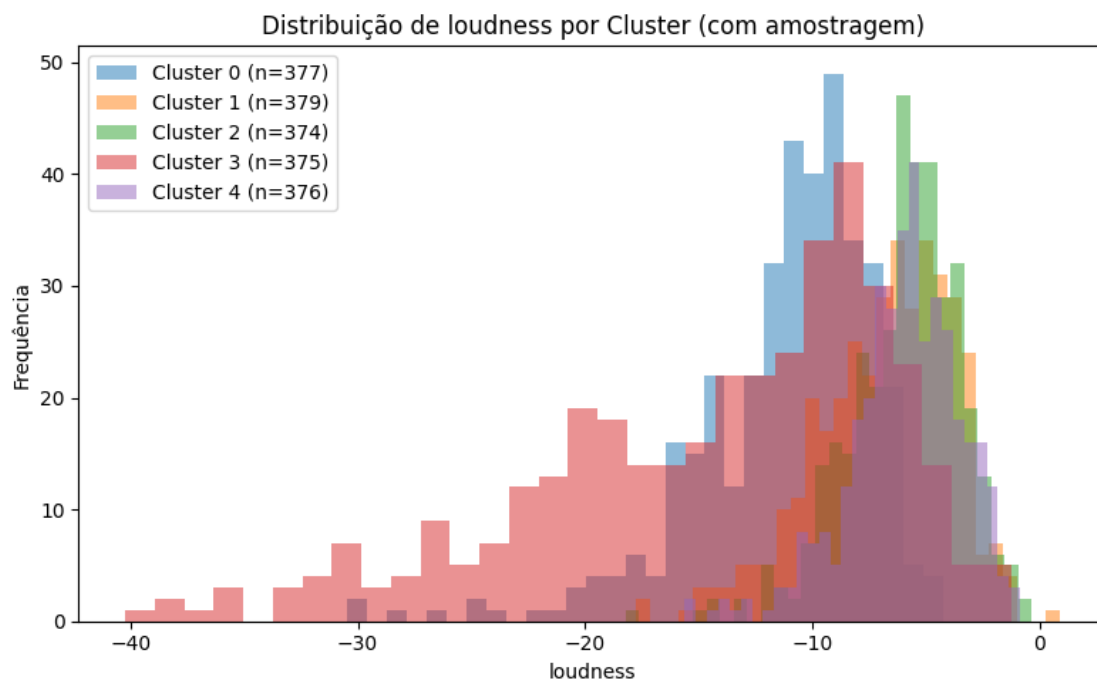
    for cluster_id in sorted(df_dados['cluster'].unique()):
        subset = df_dados[df_dados['cluster'] == cluster_id]
        tamanho = tamanho_amostra(len(subset))
        amostra = subset.sample(n=min(tamanho, len(subset)), random_state=42)
        ↪ # Garante que não passa o tamanho real
        plt.hist(amostra[feature], bins=30, alpha=0.5, label=f'Cluster_{cluster_id} (n={len(amostra)})')

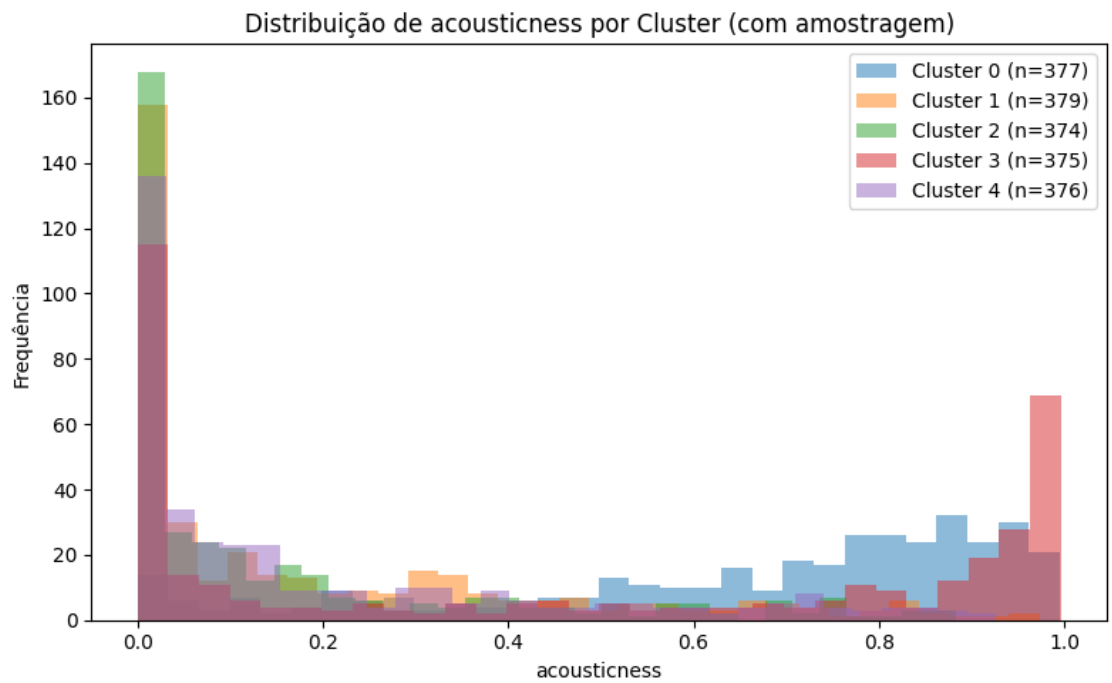
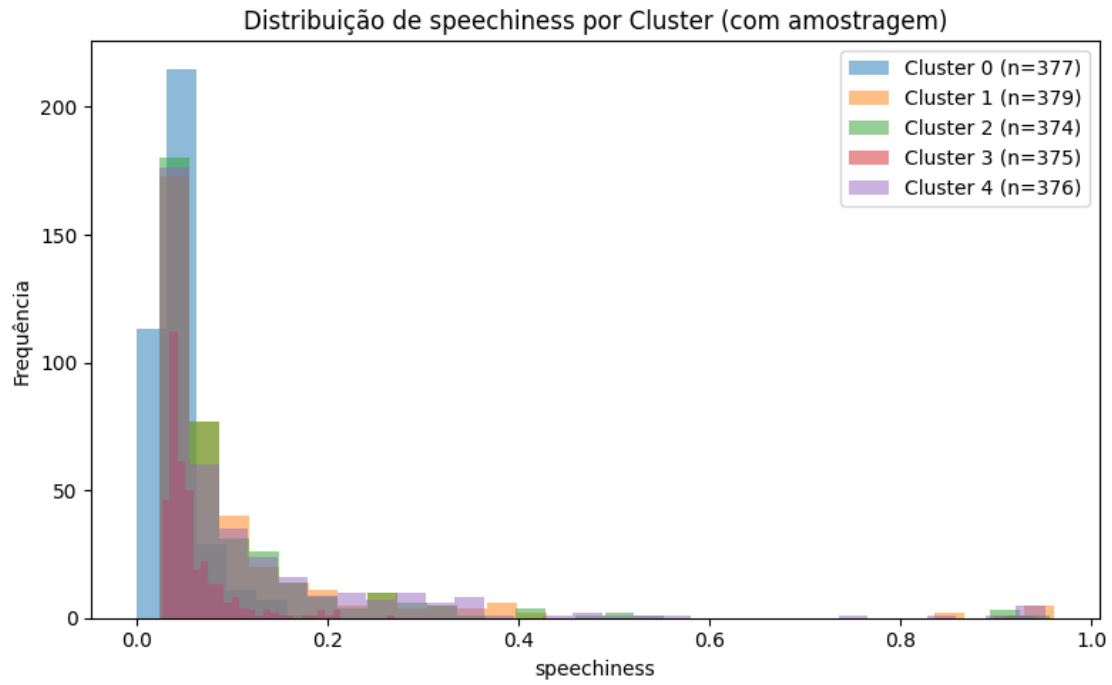
    plt.title(f'Distribuição de {feature} por Cluster (com amostragem)')
    plt.xlabel(feature)
    plt.ylabel('Frequência')
    plt.legend()
    plt.tight_layout()
    plt.show()
```

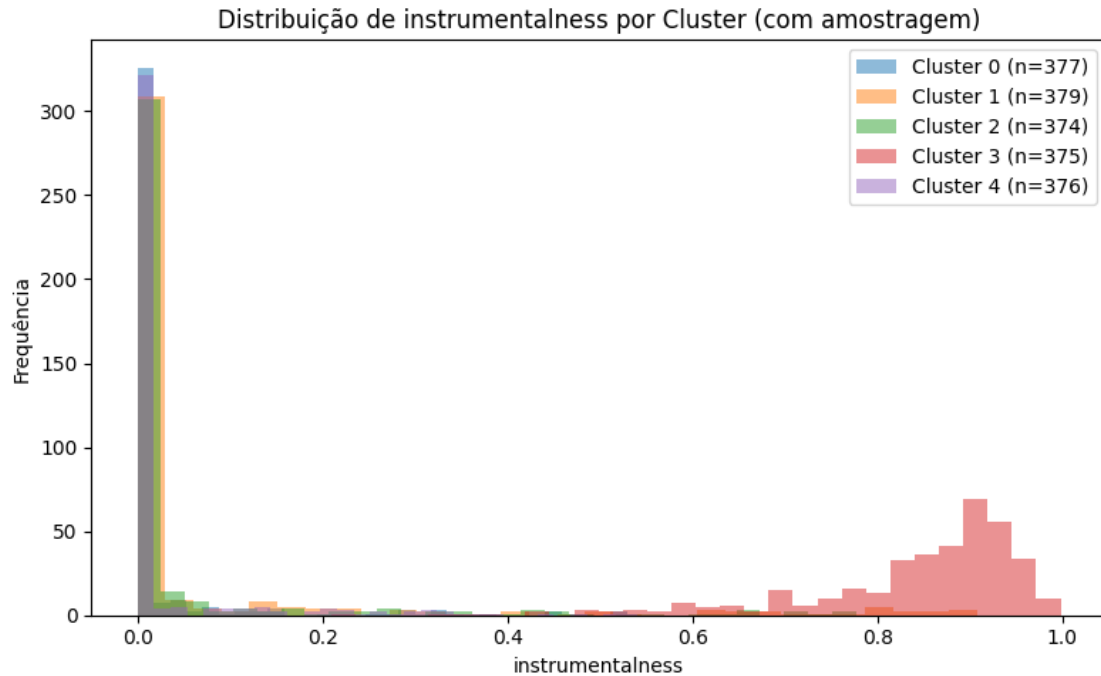




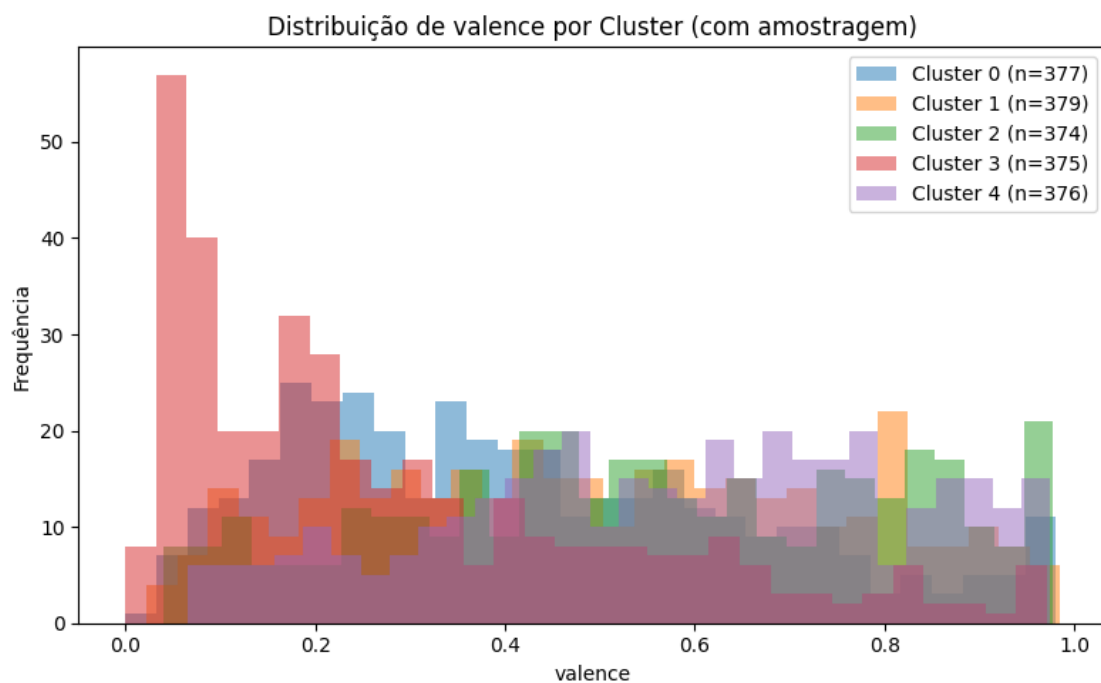
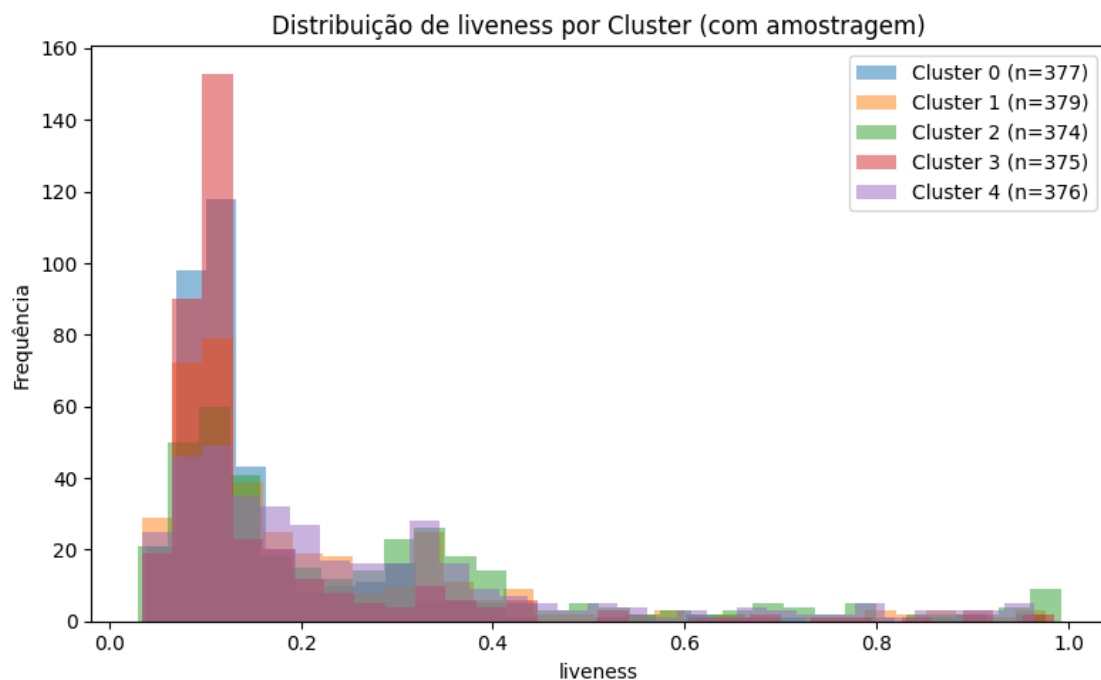


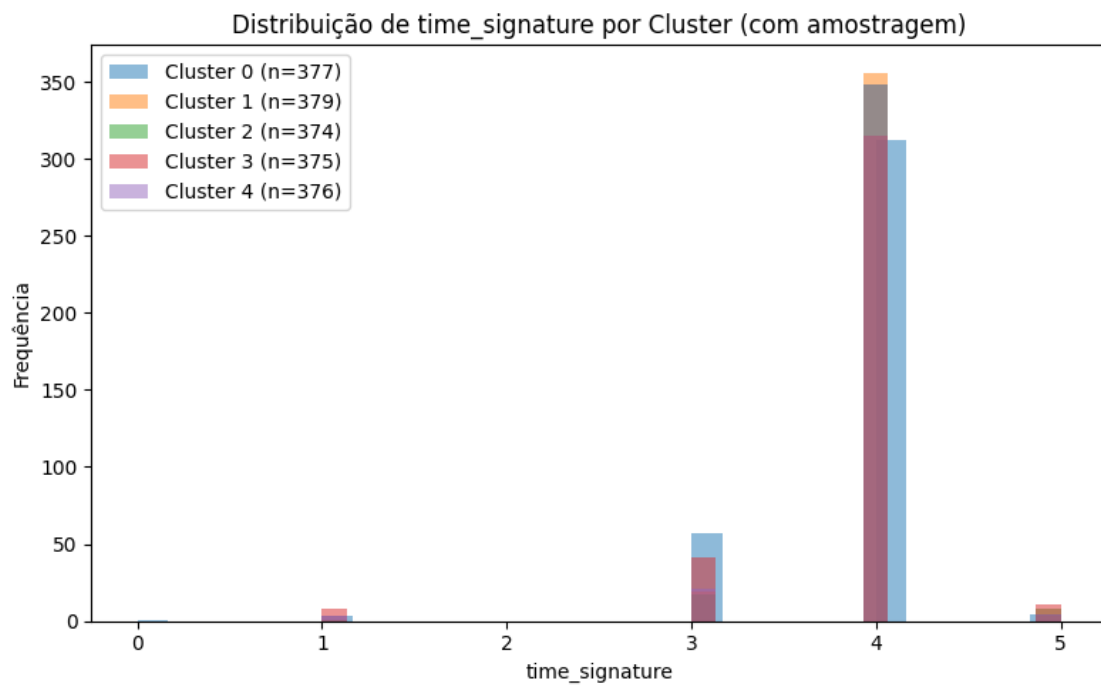
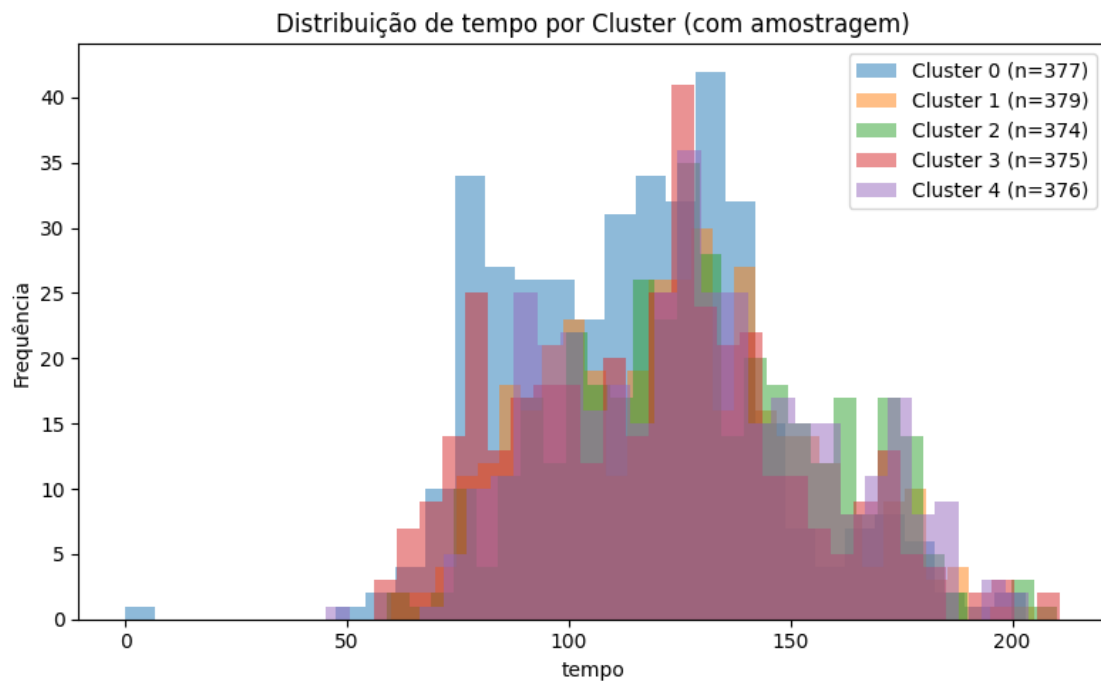






```
C:\Users\conta\AppData\Local\Temp\ipykernel_6744\430233715.py:16: UserWarning:
Creating legend with loc="best" can be slow with large amounts of data.
  plt.tight_layout()
C:\Users\conta\AppData\Roaming\Python\Python313\site-
packages\IPython\core\pylabtools.py:170: UserWarning: Creating legend with
loc="best" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```





3.4 Musica

```
[50]: recomendar_musicas('2aibwv5hGXSgw7Yru8IYT0', df_dados_normalizados2,
↳ df_musicas_minmax)
```

Cluster: 4

```
[50]:
```

	track_name	artists	track_genre
2110	Monster	Skillet	alt-rock
44195	Fake It	Seether	grunge
56827	I'm Born To Run	American Authors	indie-pop
82063	Buddy Holly	Weezer	power-pop
2357	HandClap	Fitz and The Tantrums	alt-rock
65364	Love Maze	BTS	k-pop
65464	Alcohol-Free	TWICE	k-pop
30764	What Would You Do?	Joel Corry;David Guetta;Bryson Tiller	edm
44284	Semi-Charmed Life	Third Eye Blind	grunge
38090	Teddy Picker	Arctic Monkeys	garage

4 Standart Scale

```
[51]: from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
escala3 = scaler.fit_transform(df_dados)

df_dados_normalizados3 = pd.DataFrame(escala3, columns=df_dados.columns[:])
df_dados_normalizados3
```

```
[51]:
```

	popularity	duration_ms	danceability	energy	key	loudness	\
0	1.933925	0.013472	0.644253	-0.675975	-1.203275	0.335727	
1	1.059312	-0.704186	-0.804604	-1.825602	-1.203275	-1.673087	
2	1.156491	-0.162188	-0.702731	-1.073473	-1.484183	-0.236524	
3	1.836746	-0.240925	-1.676182	-2.240247	-1.484183	-1.918228	
4	2.371232	-0.268195	0.315996	-0.746122	-0.922368	-0.226373	
...	
89735	-0.592735	1.379914	-2.208184	-1.556706	-0.079646	-1.511831	
89736	-0.544146	1.379923	-2.196865	-2.016557	-1.484183	-1.880499	
89737	-0.544146	0.374710	0.378251	-1.190384	-1.484183	-0.458874	
89738	0.379057	0.484736	0.140548	-0.500608	0.482169	-0.457725	
89739	-0.544146	0.112281	-0.204687	-0.574652	-1.203275	-0.326536	
...	
	mode	speechiness	acousticness	instrumentalness	liveness	\	
0	-1.324621	0.490458	-0.875166	-0.535482	0.723656		
1	0.754933	-0.098364	1.760810	-0.535468	-0.595078		
2	0.754933	-0.280219	-0.349626	-0.535485	-0.512978		
3	0.754933	-0.451480	1.704650	-0.535266	-0.436009		
4	0.754933	-0.307585	0.415925	-0.535485	-0.687954		
...		

89735	0.754933	-0.399395	0.921365	2.330062	-0.670508
89736	-1.324621	-0.417934	1.967716	2.478280	-0.574553
89737	-1.324621	-0.401161	1.592330	-0.535485	-0.682823
89738	0.754933	-0.509744	0.155815	-0.535485	0.272105
89739	-1.324621	-0.131910	1.042553	-0.535485	-0.655114

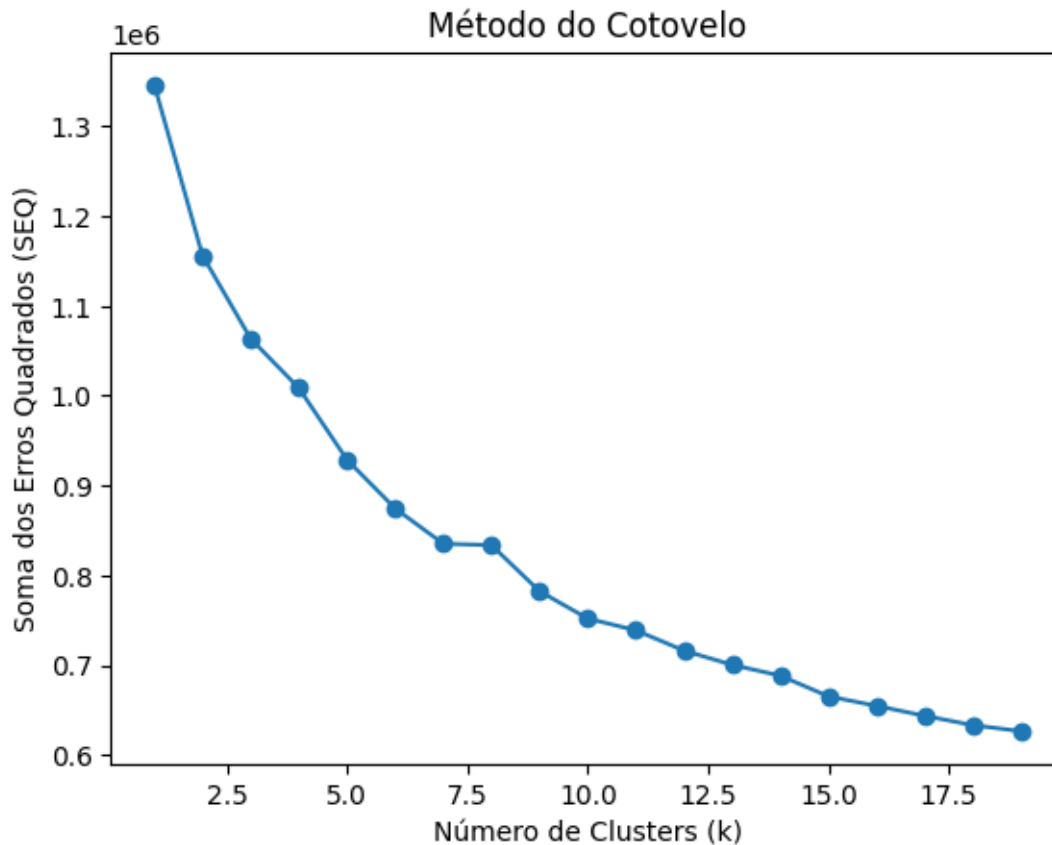
	valence	tempo	time_signature	cluster
0	0.934047	-1.133599	0.226216	-0.584473
1	-0.770269	-1.479843	0.226216	-1.283274
2	-1.329497	-1.518259	0.226216	-1.283274
3	-1.241999	1.981635	-1.979174	-1.283274
4	-1.150696	-0.070030	0.226216	-1.283274
...
89735	-1.657046	0.130717	2.431606	0.813129
89736	-1.652861	-1.222517	0.226216	0.813129
89737	1.040567	0.342654	0.226216	-1.283274
89738	-0.214844	0.461588	0.226216	-1.283274
89739	0.907417	-1.423098	0.226216	-1.283274

[89740 rows x 15 columns]

Aplicando o método do cotovelo

```
[52]: soma_e3 = calcular_soma_erros(escala3)
```

```
[53]: plt.plot(range(1, 20), soma_e3, marker='o')
plt.xlabel('Número de Clusters (k)')
plt.ylabel('Soma dos Erros Quadrados (SEQ)')
plt.title('Método do Cotovelo')
plt.show()
```



Diferentemente do método anterior, neste caso conseguimos ver claramente o ‘cotovelo’, por isso definimos o **número de clusters = 7**

```
[54]: nc3=7
```

```
[55]: from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=nc3, random_state=14)
clusters = kmeans.fit_predict(escala3)

df_musicas_stand = df_musicas.copy()
df_musicas_stand['cluster'] = clusters
df_dados['cluster'] = clusters
df_dados_normalizados3['cluster'] = clusters
```

4.1 Plotagem 2D e 3D dos clusters

```
[56]: from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
```

```

pca = PCA(n_components=2)
X_pca = pca.fit_transform(escala3)

# Plot
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1],
    ↪c=df_dados_normalizados3['cluster'], cmap='viridis')

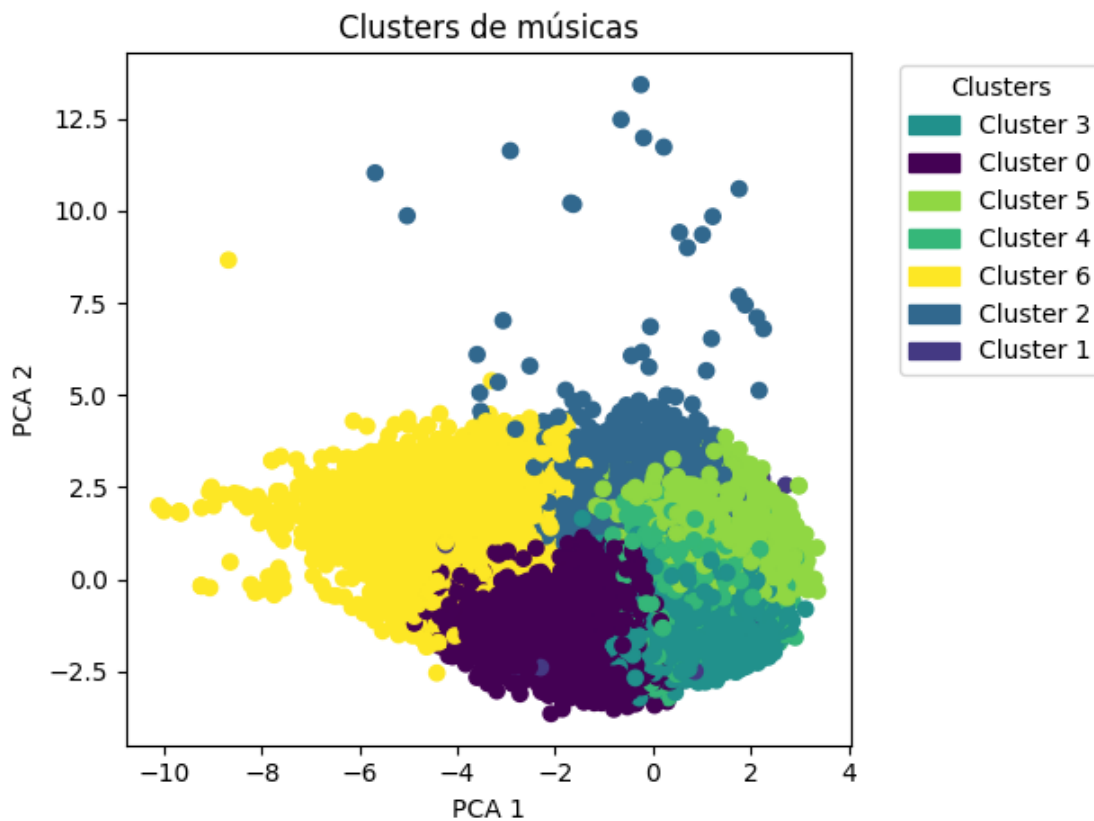
# Legenda manual
clusters = df_dados_normalizados3['cluster'].unique()
colors = scatter.cmap(scatter.norm(clusters))

handles = [mpatches.Patch(color=colors[i], label=f'Cluster {clusters[i]}') for
    ↪i in range(len(clusters))]

plt.legend(handles=handles, title='Clusters', bbox_to_anchor=(1.05, 1),
    ↪loc='upper left')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.title('Clusters de músicas')

plt.tight_layout()
plt.show()

```

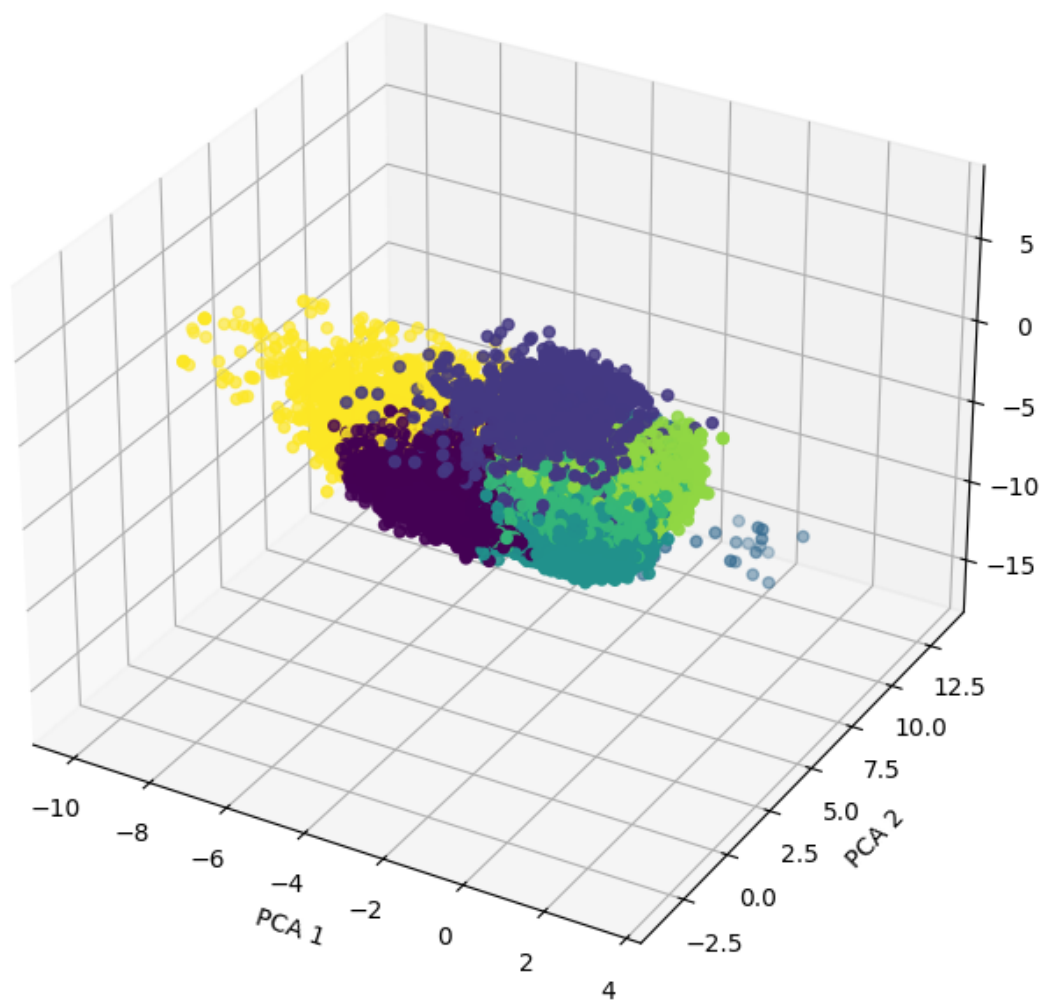


```
[57]: from mpl_toolkits.mplot3d import Axes3D

pca = PCA(n_components=3)
X_pca_3d = pca.fit_transform(escala3)

fig = plt.figure(figsize=(10,8))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X_pca_3d[:, 0], X_pca_3d[:, 1], X_pca_3d[:, 2],
                    c=df_dados_normalizados3['cluster'], cmap='viridis')
ax.set_xlabel('PCA 1')
ax.set_ylabel('PCA 2')
ax.set_zlabel('PCA 3')
plt.title('Clusters em 3D com PCA')
plt.show()
```

Clusters em 3D com PCA



4.2 Analisando os dados

```
[58]: df_dados_normalizados3['cluster'].value_counts()
```

```
[58]: cluster
3      22104
0      18773
5      16778
4      14447
2      10029
6       6544
1       1065
```

Name: count, dtype: int64

4.2.1 Dados Originais

```
[59]: df_dados_media = df_dados.groupby('cluster').mean(numeric_only=True) #Para
      ↪ histogramas
      df_dados_media
```

```
[59]:
```

	popularity	duration_ms	danceability	energy	key	\
cluster						
0	32.504927	215846.004688	0.526570	0.374973	5.092260	
1	24.567136	218449.107981	0.570402	0.675837	5.081690	
2	25.833782	301005.254163	0.565043	0.739642	5.467345	
3	36.422774	226008.865590	0.612985	0.745666	6.071571	
4	35.043608	220436.050253	0.591548	0.765430	1.403682	
5	34.544761	219725.037490	0.590372	0.774489	7.881392	
6	29.468062	212870.119957	0.349699	0.187127	4.826406	

	loudness	mode	speechiness	acousticness	instrumentalness	\
cluster						
0	-10.868613	0.817024	0.046370	0.662169	0.020890	
1	-11.196860	0.693897	0.836988	0.740297	0.006451	
2	-8.582128	0.558879	0.073359	0.123740	0.797845	
3	-6.278753	0.000181	0.097442	0.178467	0.040702	
4	-6.050017	0.999931	0.089257	0.168797	0.032542	
5	-5.792718	1.000000	0.093049	0.181285	0.026267	
6	-20.979222	0.649756	0.052707	0.851916	0.817720	

	liveness	valence	tempo	time_signature
cluster				
0	0.164118	0.410774	113.427991	3.806105
1	0.673429	0.444778	100.919011	3.583099
2	0.188640	0.327106	127.400637	3.936484
3	0.220076	0.531866	124.859598	3.972901
4	0.246794	0.560779	127.820266	3.966221
5	0.251183	0.570707	128.765935	3.963643
6	0.173681	0.188213	102.686851	3.574114

4.2.2 Dados Normalizados

```
[60]: df_dados_normalizados3_media = df_dados_normalizados3.groupby('cluster').mean()
      ↪ # Para gráfico de médias
      df_dados_normalizados3_media
```

```
[60]:
```

	popularity	duration_ms	danceability	energy	key	loudness	\
cluster							
0	-0.033715	-0.117742	-0.201459	-1.011227	-0.053729	-0.453821	

1	-0.419410	-0.094694	0.046610	0.161255	-0.056698	-0.516685
2	-0.357864	0.636246	0.016281	0.409905	0.051635	-0.015922
3	0.156651	-0.027761	0.287614	0.433381	0.221366	0.425212
4	0.089638	-0.077102	0.166287	0.510402	-1.089878	0.469019
5	0.065399	-0.083397	0.159632	0.545703	0.729759	0.518296
6	-0.181276	-0.144090	-1.202478	-1.743270	-0.128410	-2.390167

	mode	speechiness	acousticness	instrumentalness	liveness	\
cluster						
0	0.374426	-0.362579	0.986892	-0.470978	-0.271203	
1	0.118375	6.616923	1.217822	-0.515566	2.342207	
2	-0.162401	-0.124328	-0.604592	1.928159	-0.145373	
3	-1.324244	0.088274	-0.442832	-0.409803	0.015934	
4	0.754789	0.016024	-0.471413	-0.434999	0.153032	
5	0.754933	0.049499	-0.434500	-0.454376	0.175548	
6	0.026581	-0.306639	1.547745	1.989530	-0.222131	

	valence	tempo	time_signature
cluster			
0	-0.223312	-0.286549	-0.201399
1	-0.093950	-0.701889	-0.693214
2	-0.541607	0.177389	0.086139
3	0.237356	0.093018	0.166452
4	0.347346	0.191322	0.151721
5	0.385116	0.222721	0.146034
6	-1.069995	-0.643191	-0.713029

4.2.3 Gênero de músicas por cluster

```
[61]: genero_por_cluster = df_musicas_stand.groupby('cluster').apply(
        lambda x: x['track_genre'].value_counts().head(6),
        include_groups=False
    )
genero_por_cluster
```

```
[61]: cluster  track_genre
0             honky-tonk      833
           romance          747
           tango            719
           cantopop          694
           acoustic          650
           opera            571
1             comedy          806
           show-tunes         32
           kids              22
           children          19
           french            15
```

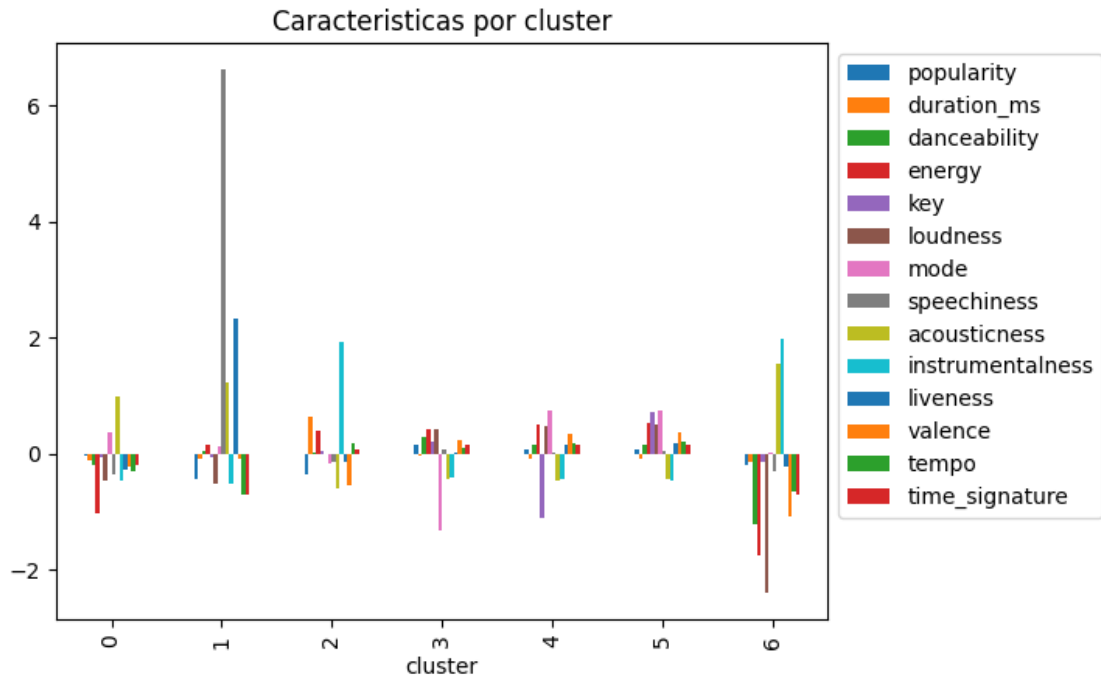

	funk	14
2	minimal-techno	747
	detroit-techno	742
	grindcore	582
	study	573
	chicago-house	502
	black-metal	497
3	turkish	479
	dancehall	467
	dance	439
	hardstyle	436
	hip-hop	419
	k-pop	407
4	power-pop	303
	j-idol	286
	kids	279
	forro	266
	sertanejo	253
	heavy-metal	252
5	j-idol	381
	party	363
	forro	362
	sertanejo	325
	country	304
	alt-rock	304
6	sleep	814
	new-age	751
	ambient	660
	classical	608
	piano	418
	disney	415

Name: count, dtype: int64

4.3 Gráficos

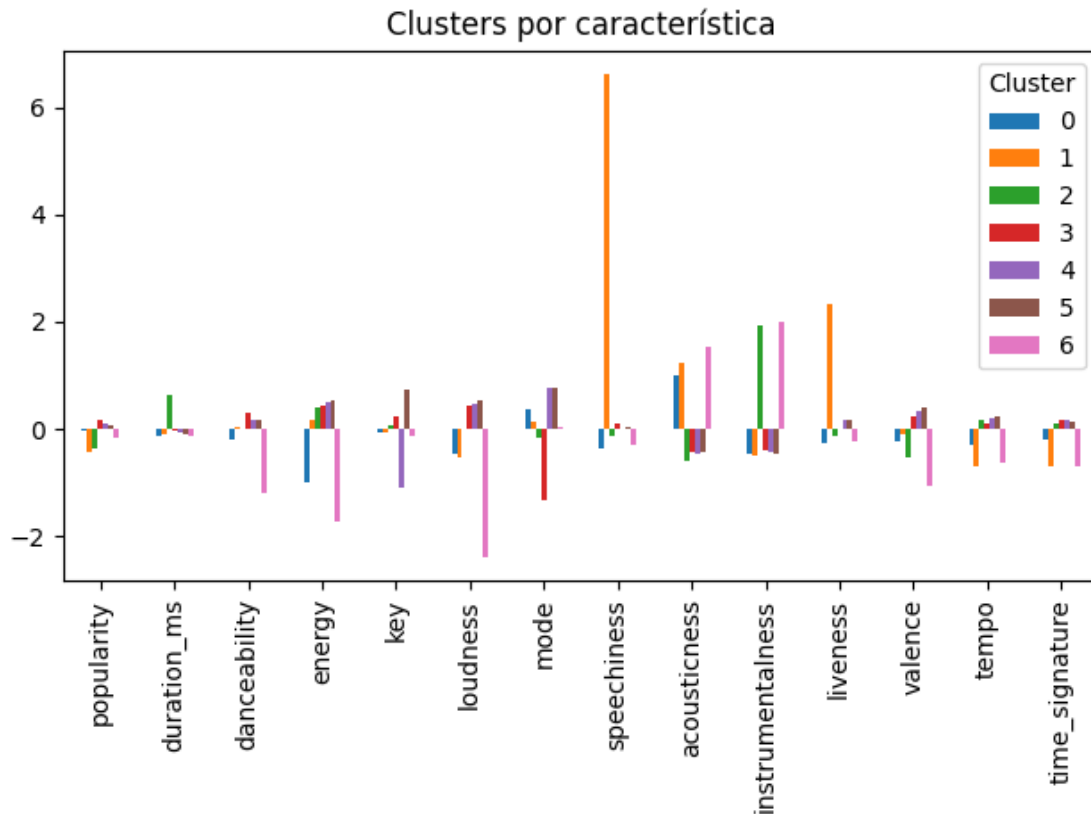
4.3.1 Características x Cluster

```
[62]: df_dados_normalizados3_media.plot(kind='bar')
plt.title('Características por cluster')
plt.legend(bbox_to_anchor=(1.0, 1.0))
plt.show()
```



4.3.2 Cluster x Características

```
[63]: df_dados_normalizados3_media.T.plot(kind='bar')
plt.title('Clusters por característica')
plt.legend(title='Cluster', bbox_to_anchor=(1.0, 1.0))
plt.tight_layout()
plt.show()
```



4.3.3 Histogramas

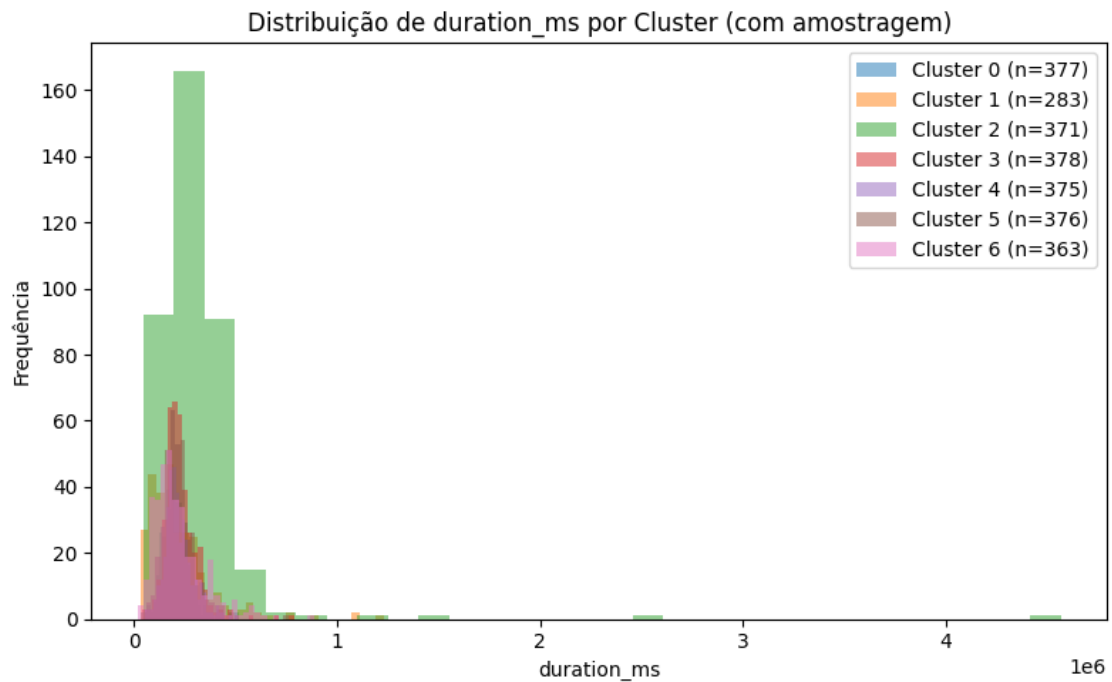
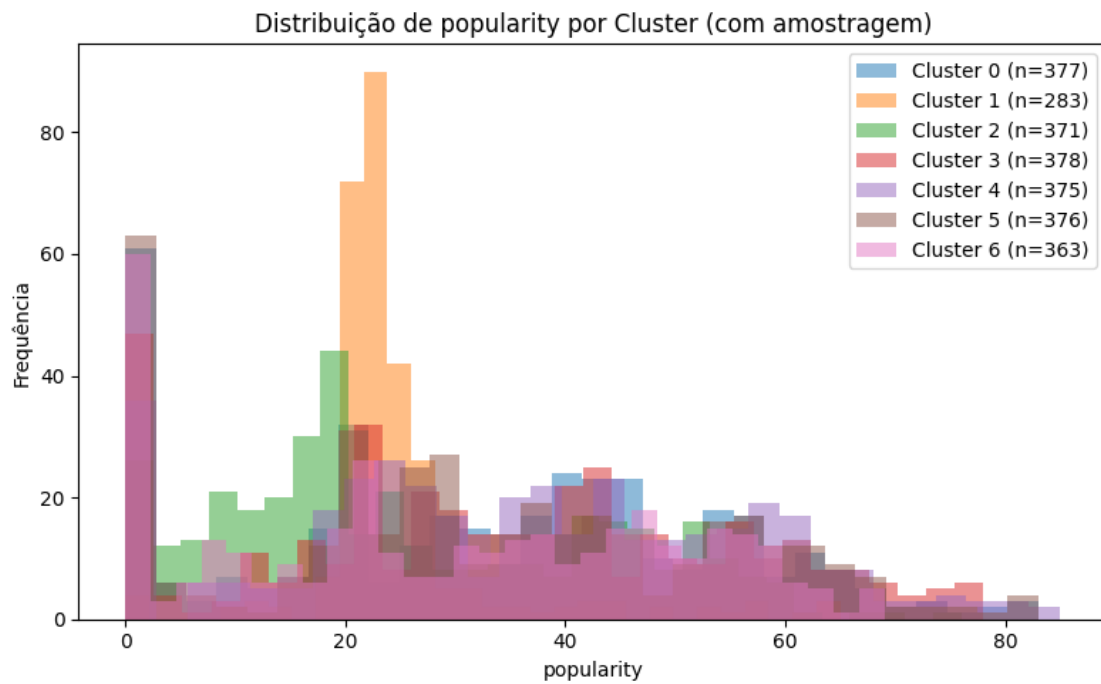
```
[64]: import matplotlib.pyplot as plt

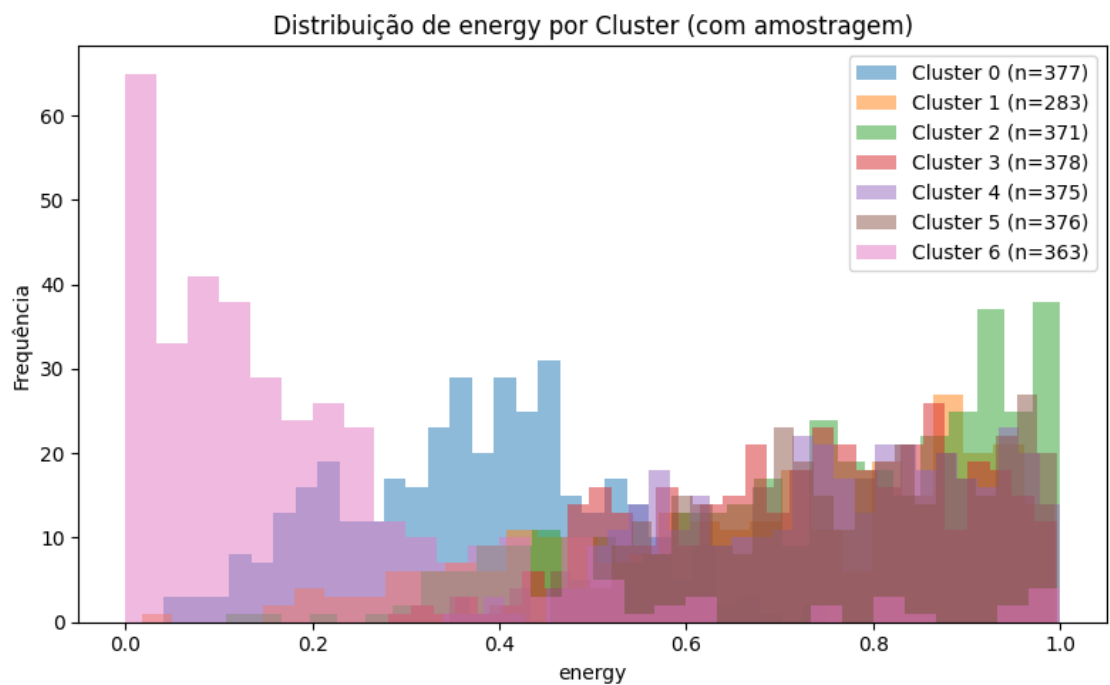
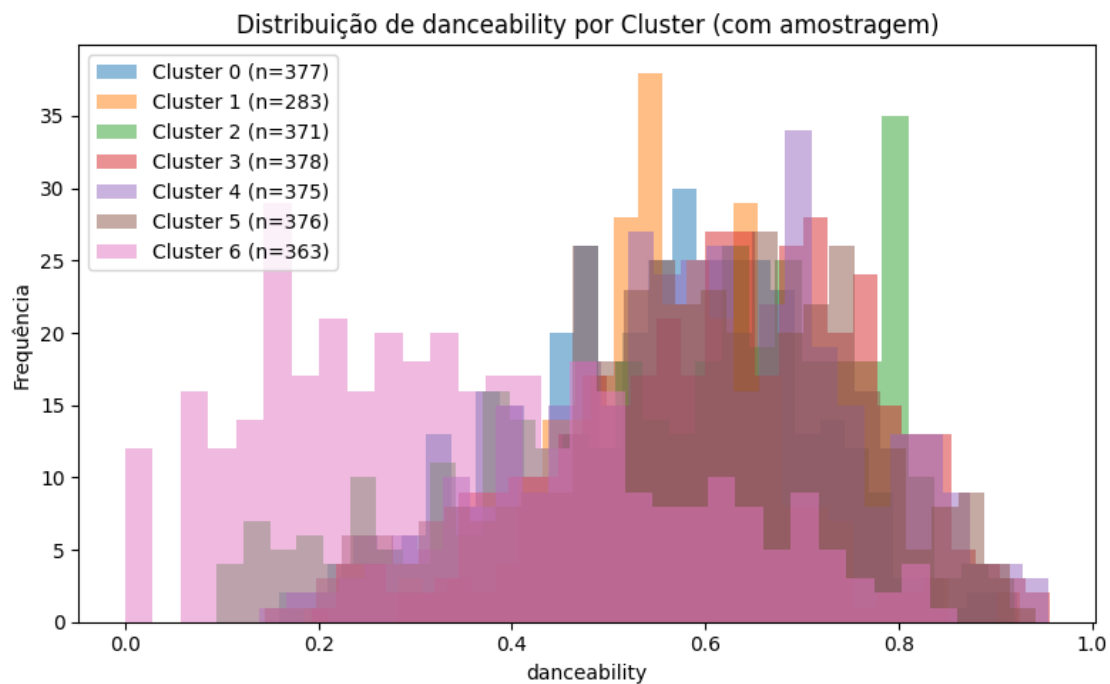
for feature in df_dados.columns.drop('cluster'):
    plt.figure(figsize=(8, 5))

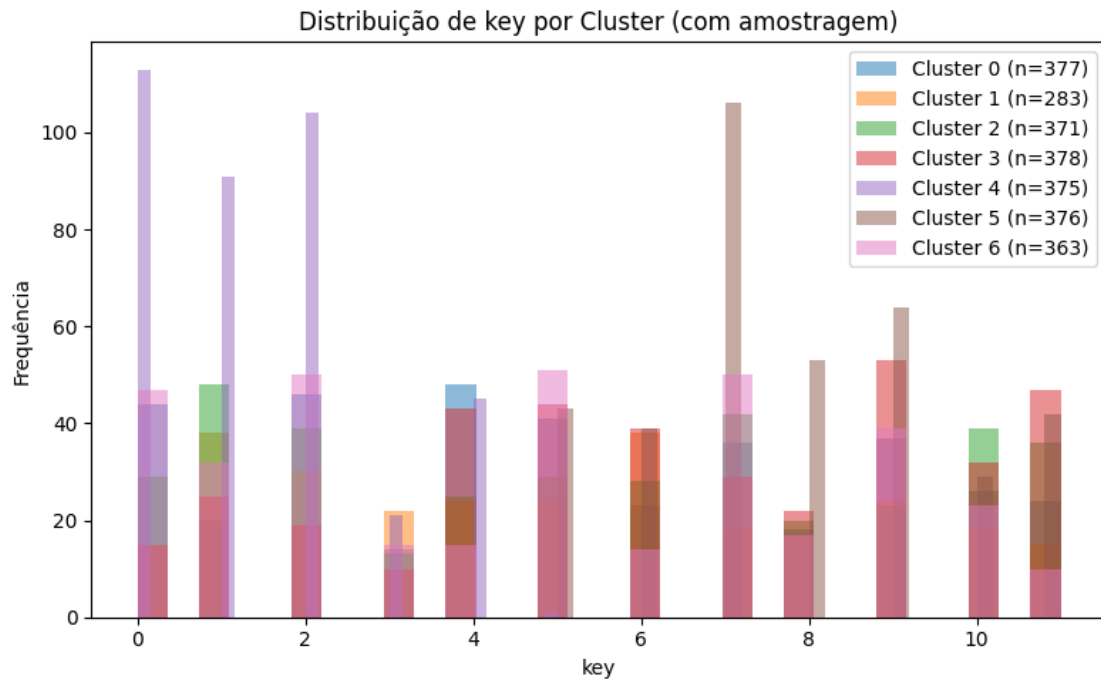
    for cluster_id in sorted(df_dados['cluster'].unique()):
        subset = df_dados[df_dados['cluster'] == cluster_id]
        tamanho = tamanho_amostra(len(subset))
        amostra = subset.sample(n=min(tamanho, len(subset)), random_state=42)
        # Garante que não passa o tamanho real
        plt.hist(amostra[feature], bins=30, alpha=0.5, label=f'Cluster_{cluster_id} (n={len(amostra)})')

    plt.title(f'Distribuição de {feature} por Cluster (com amostragem)')
    plt.xlabel(feature)
    plt.ylabel('Frequência')
    plt.legend()
    plt.tight_layout()
```

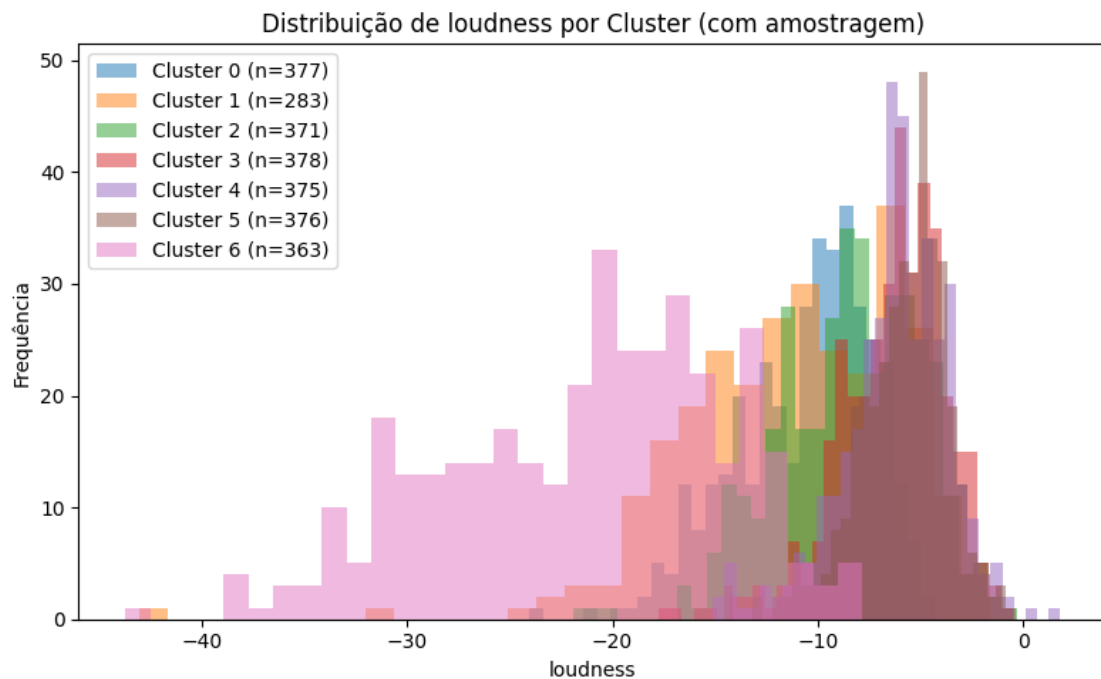
```
plt.show()
```

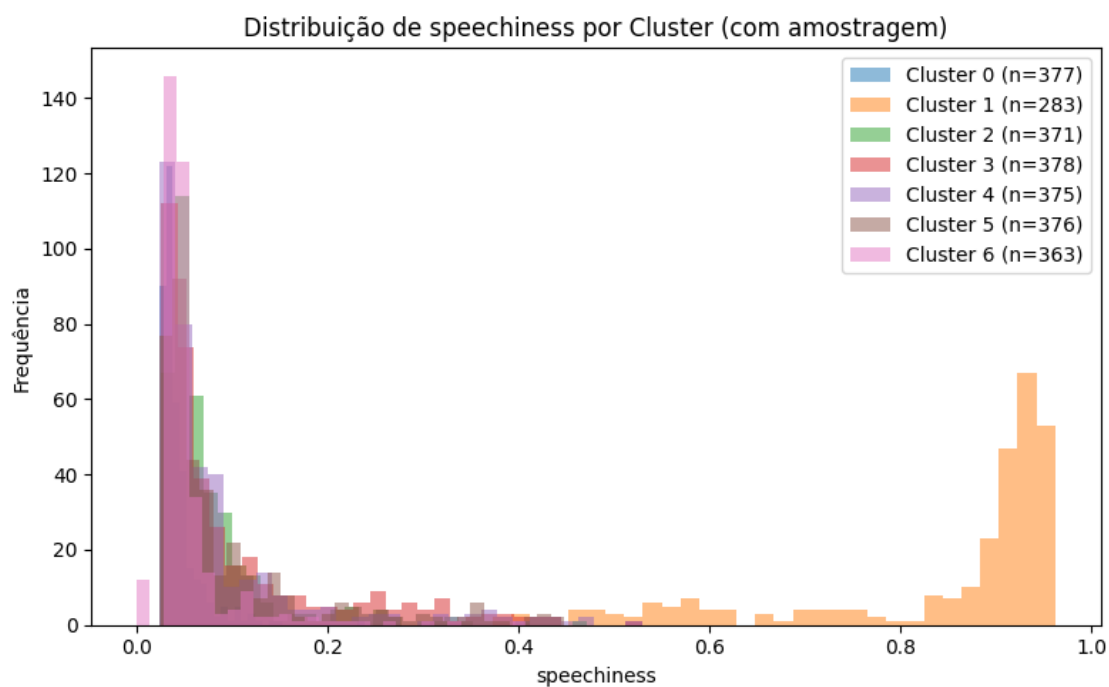
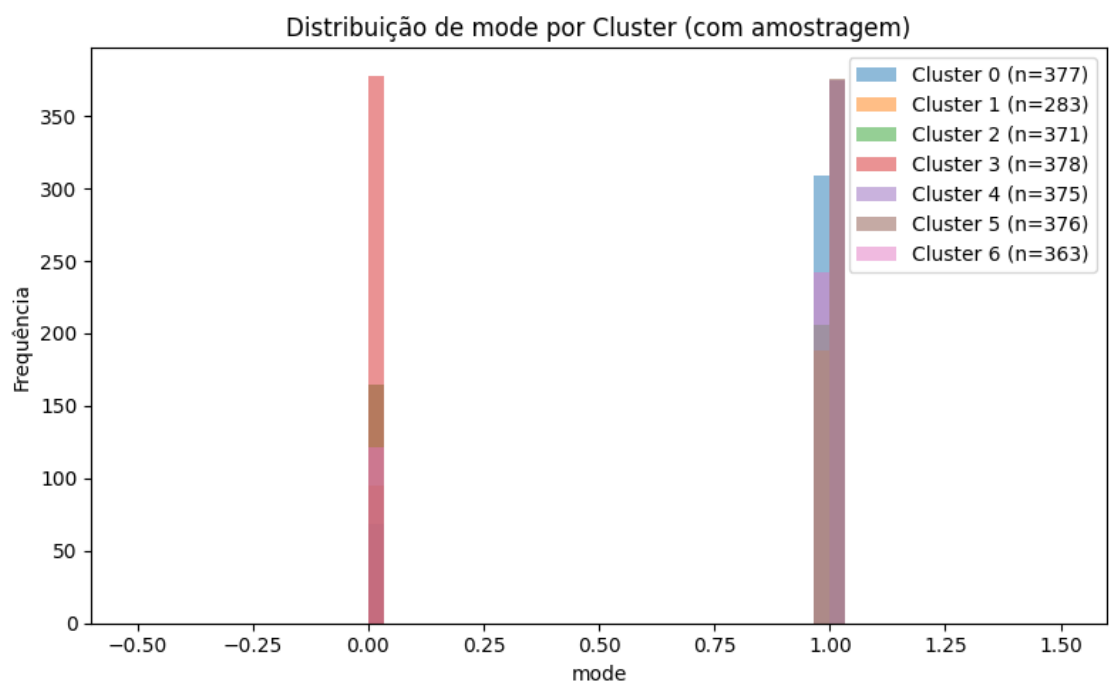


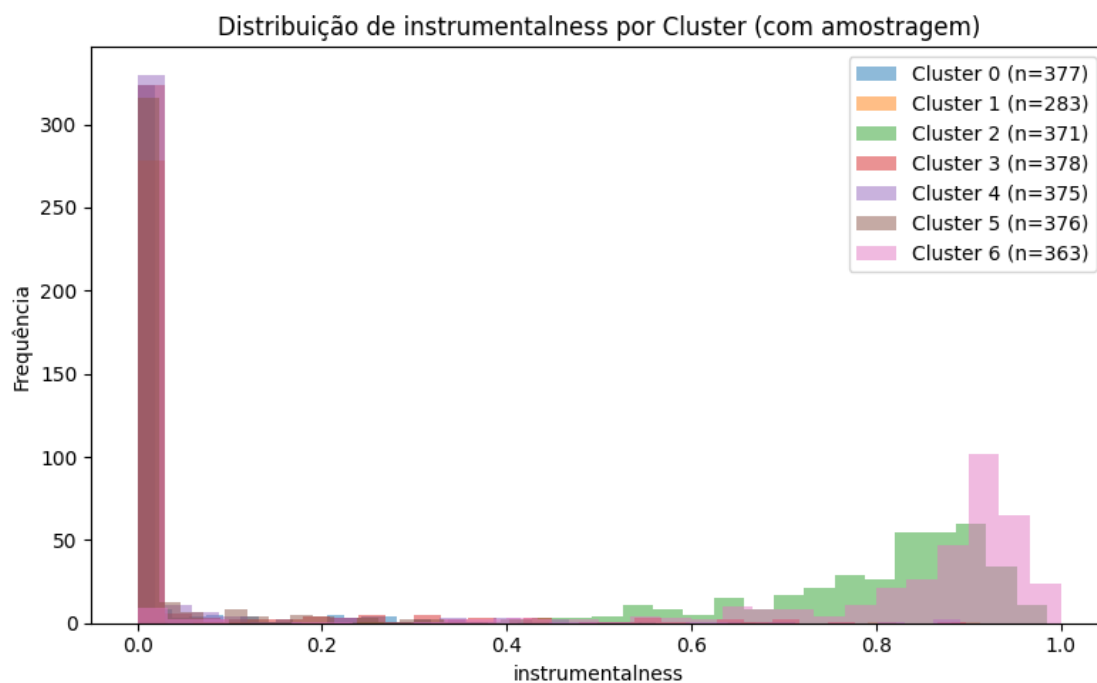
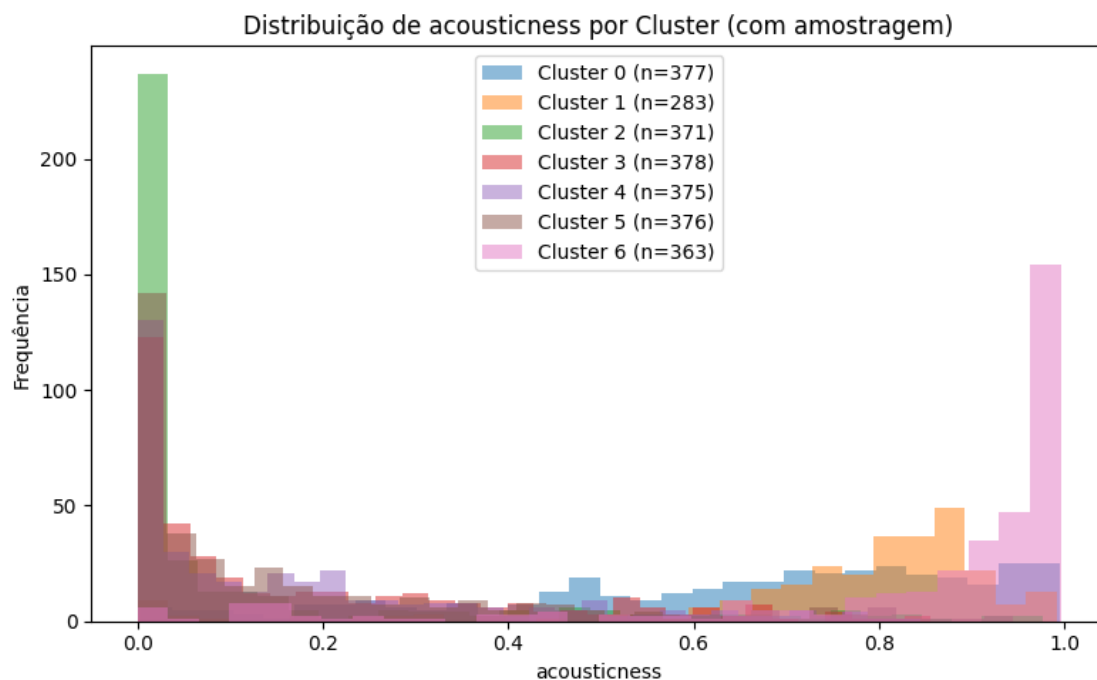


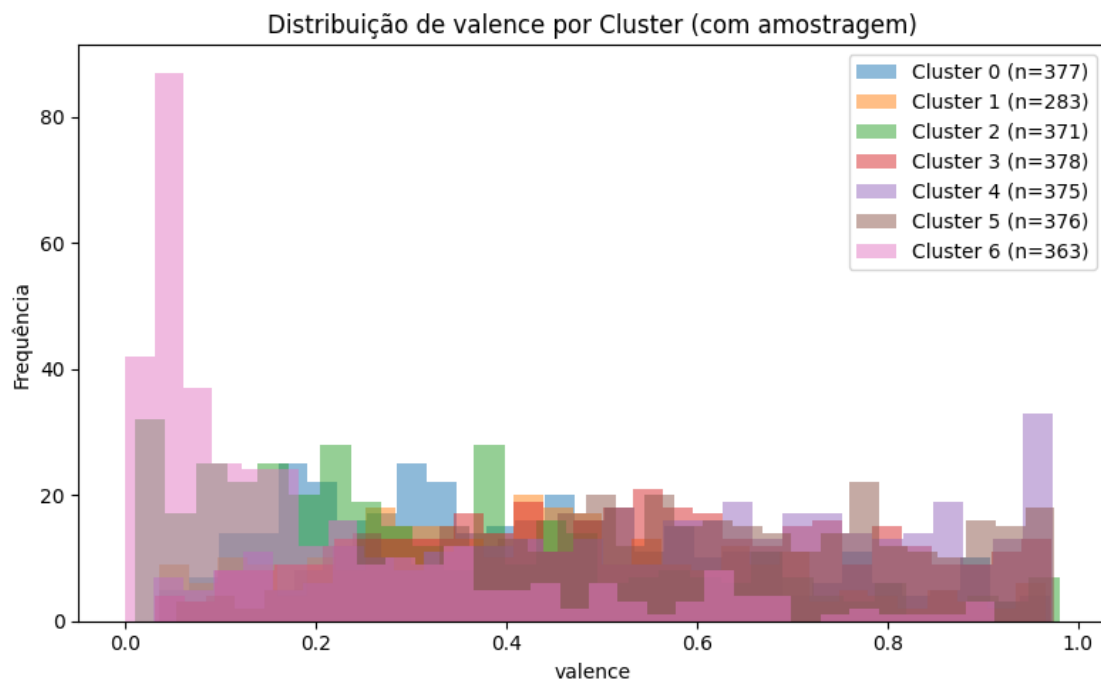
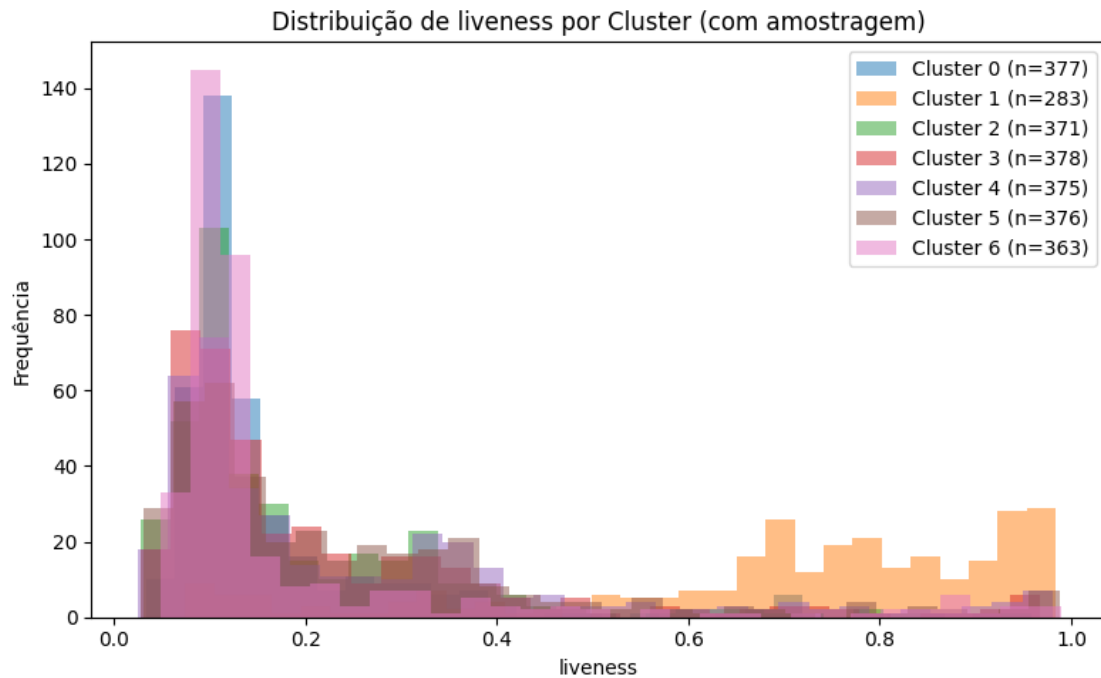


```
C:\Users\conta\AppData\Roaming\Python\Python313\site-
packages\IPython\core\pylabtools.py:170: UserWarning: Creating legend with
loc="best" can be slow with large amounts of data.
fig.canvas.print_figure(bytes_io, **kw)
```

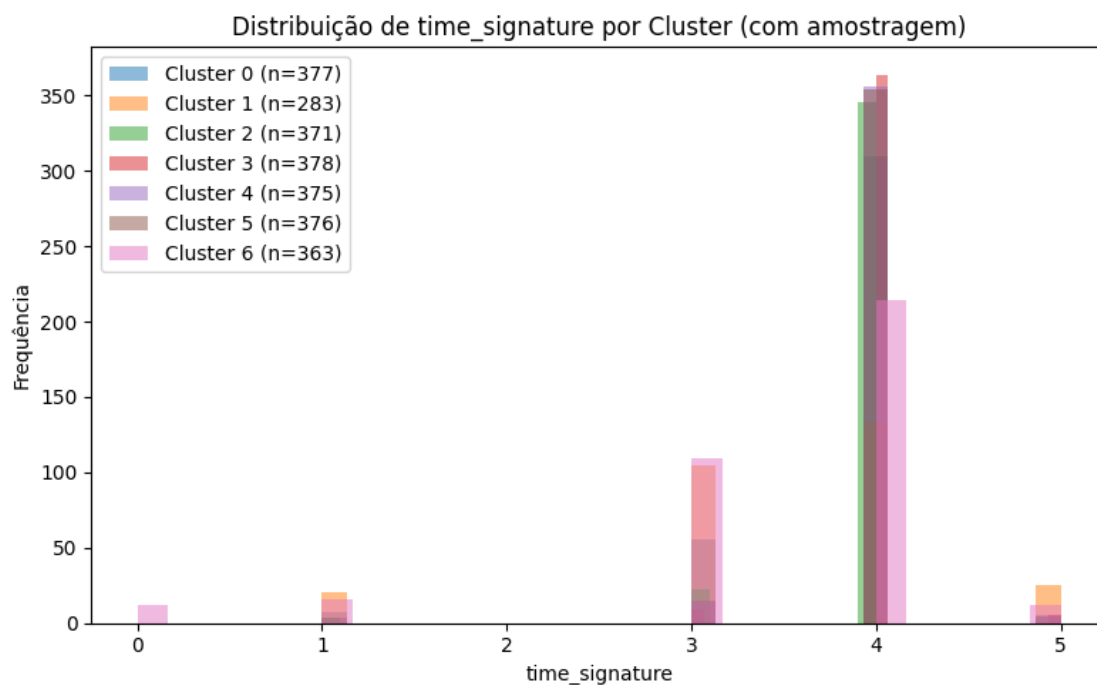
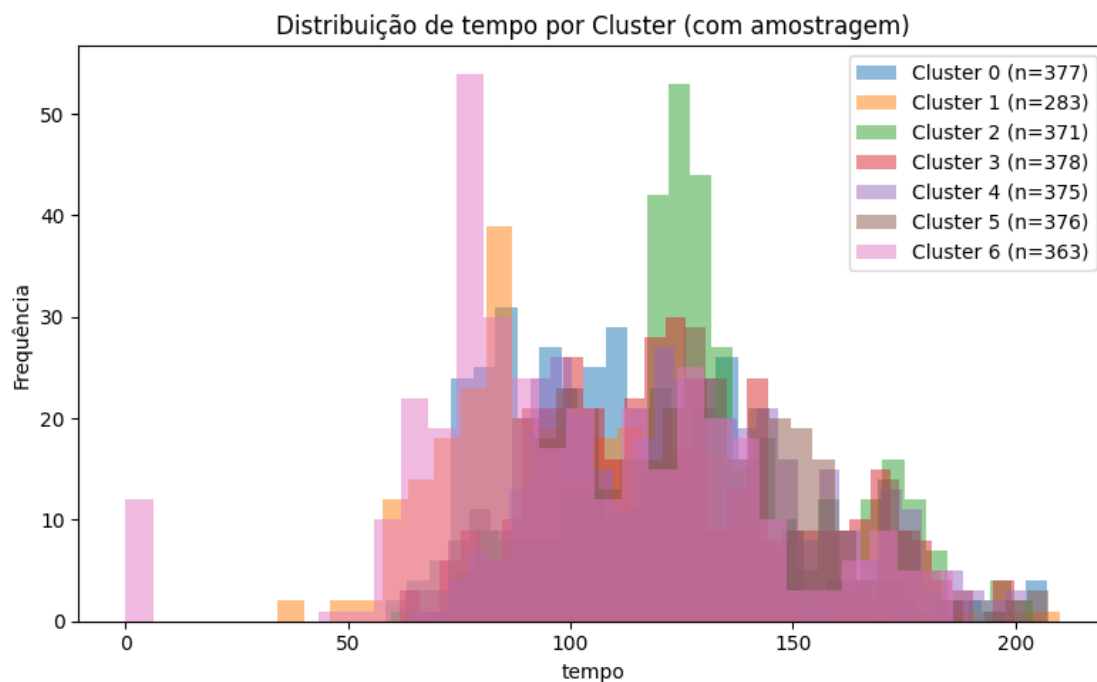








C:\Users\conta\AppData\Local\Temp\ipykernel_6744\430233715.py:16: UserWarning:
 Creating legend with loc="best" can be slow with large amounts of data.
 plt.tight_layout()



4.4 Musica

```
[65]: recomendar_musicas('2aibwv5hGXSgw7Yru8IYT0', df_dados_normalizados3,
    ↪ df_musicas_stand)
```

Cluster: 5

```
[65]:
```

	track_name \		artists	track_genre
2110	Monster		Skillet	alt-rock
44195	Fake It		Seether	grunge
30764	What Would You Do?		Joel Corry;David Guetta;Bryson Tiller	edm
2357	HandClap		Fitz and The Tantrums	alt-rock
56827	I'm Born To Run		American Authors	indie-pop
11266	cotton candy		YUNGBLUD	british
56383	SUPERMODEL		Måneskin	indie-pop
80561	Ek Toh Kum Zindagani (From "Marjaavaan")		Neha Kakkar;Yash Narvekar;Tanishk Bagchi	pop-film
38090	Teddy Picker		Arctic Monkeys	garage
53762	Under Control (feat. Hurts)		Calvin Harris;Alesso;Hurts	house

5 Max Abs Scale

```
[66]: from sklearn.preprocessing import MaxAbsScaler
scaler = MaxAbsScaler()
escala4 = scaler.fit_transform(df_dados)
df_dados_normalizados4 = pd.DataFrame(escala4, columns=df_dados.columns[:])
df_dados_normalizados4
```

```
[66]:
```

	popularity	duration_ms	danceability	energy	key	loudness \
0	0.73	0.044043	0.686294	0.4610	0.090909	-0.136198
1	0.55	0.028566	0.426396	0.1660	0.090909	-0.347964
2	0.57	0.040255	0.444670	0.3590	0.000000	-0.196523
3	0.71	0.038557	0.270051	0.0596	0.000000	-0.373806
4	0.82	0.037969	0.627411	0.4430	0.181818	-0.195453
...
89735	0.21	0.073511	0.174619	0.2350	0.454545	-0.330964
89736	0.22	0.073511	0.176650	0.1170	0.000000	-0.369829

89737	0.22	0.051833	0.638579	0.3290	0.000000	-0.219963
89738	0.41	0.054206	0.595939	0.5060	0.636364	-0.219842
89739	0.22	0.046174	0.534010	0.4870	0.090909	-0.206012

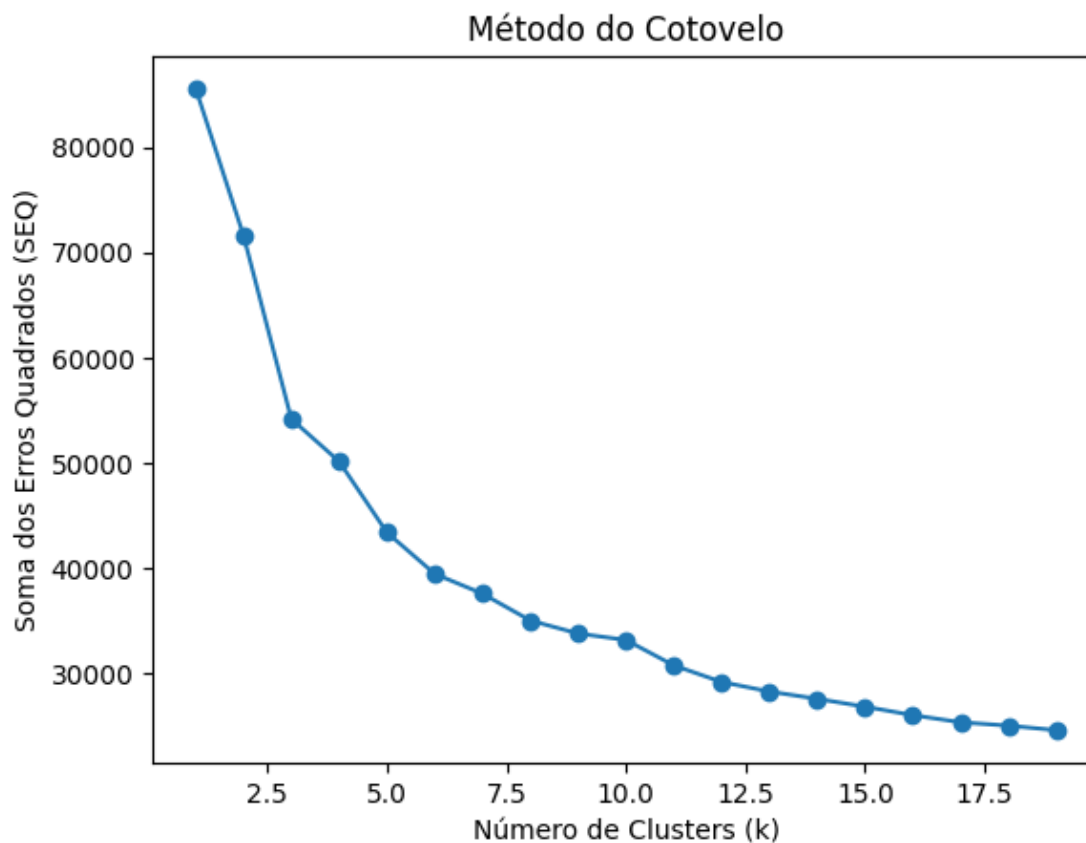
	mode	speechiness	acousticness	instrumentalness	liveness	valence	\
0	0.0	0.148187	0.032329	0.000001	0.3580	0.718593	
1	1.0	0.079067	0.927711	0.000006	0.1010	0.268342	
2	1.0	0.057720	0.210843	0.000000	0.1170	0.120603	
3	1.0	0.037617	0.908635	0.000071	0.1320	0.143719	
4	1.0	0.054508	0.470884	0.000000	0.0829	0.167839	
...	
89735	1.0	0.043731	0.642570	0.928000	0.0863	0.034070	
89736	0.0	0.041554	0.997992	0.976000	0.1050	0.035176	
89737	0.0	0.043523	0.870482	0.000000	0.0839	0.746734	
89738	1.0	0.030777	0.382530	0.000000	0.2700	0.415075	
89739	0.0	0.075130	0.683735	0.000000	0.0893	0.711558	

	tempo	time_signature	cluster
0	0.361245	0.8	0.5
1	0.318397	0.8	0.0
2	0.313643	0.8	0.0
3	0.746758	0.6	0.0
4	0.492863	0.8	0.0
...
89735	0.517705	1.0	1.0
89736	0.350242	0.8	1.0
89737	0.543933	0.8	0.0
89738	0.558651	0.8	0.0
89739	0.325420	0.8	0.0

[89740 rows x 15 columns]

```
[67]: soma_e4 = calcular_soma_erros(escala4)
```

```
[68]: plt.plot(range(1, 20), soma_e4, marker='o')
plt.xlabel('Número de Clusters (k)')
plt.ylabel('Soma dos Erros Quadrados (SEQ)')
plt.title('Método do Cotovelo')
plt.show()
```



Número de Clusters

```
[69]: nc4 = optimal_number_of_clusters(soma_e4)
      nc4
```

```
[69]: 7
```

```
[70]: from sklearn.cluster import KMeans
      kmeans = KMeans(n_clusters=nc4, random_state=14)
      clusters = kmeans.fit_predict(escala4)

      df_musicas_maxabs = df_musicas.copy()
      df_musicas_maxabs['cluster'] = clusters

      df_dados['cluster'] = clusters

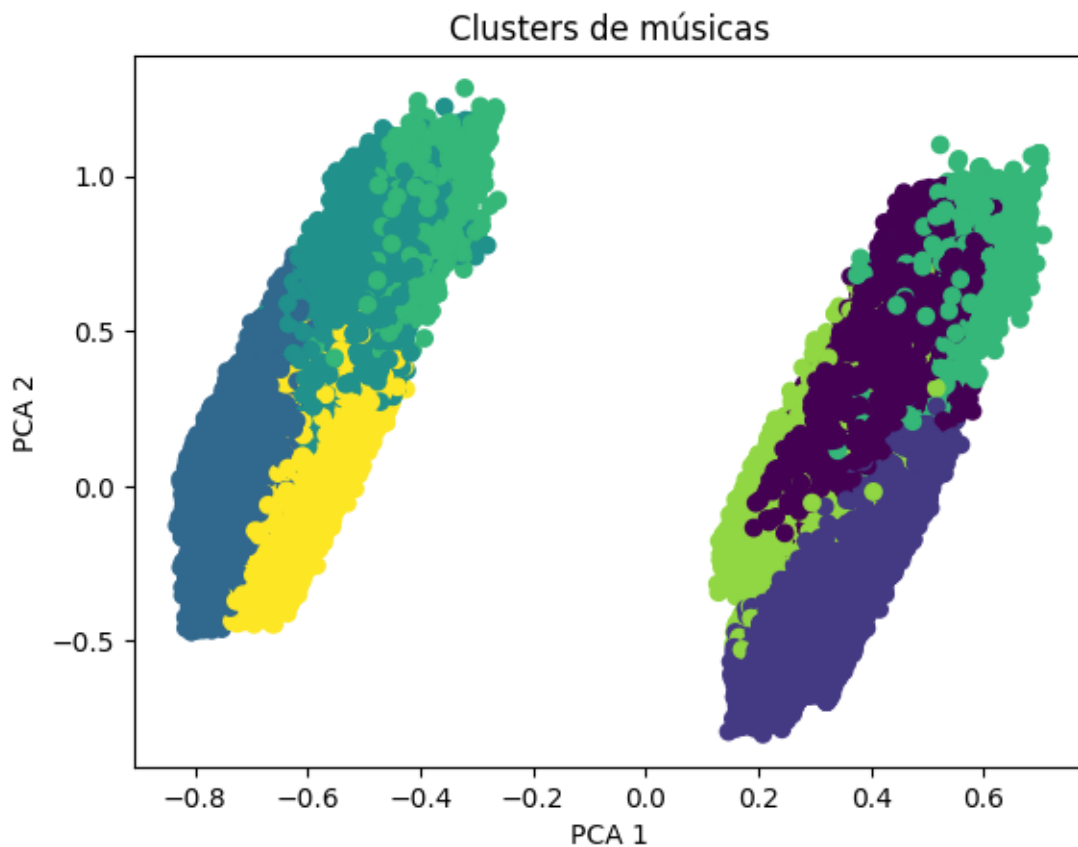
      df_dados_normalizados4['cluster'] = clusters
```

5.1 Plotagem 2D e 3D dos clusters

```
[71]: from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

pca = PCA(n_components=2)
X_pca = pca.fit_transform(escala4)

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df_dados_normalizados4['cluster'],
            cmap='viridis')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.title('Clusters de músicas')
plt.show()
```



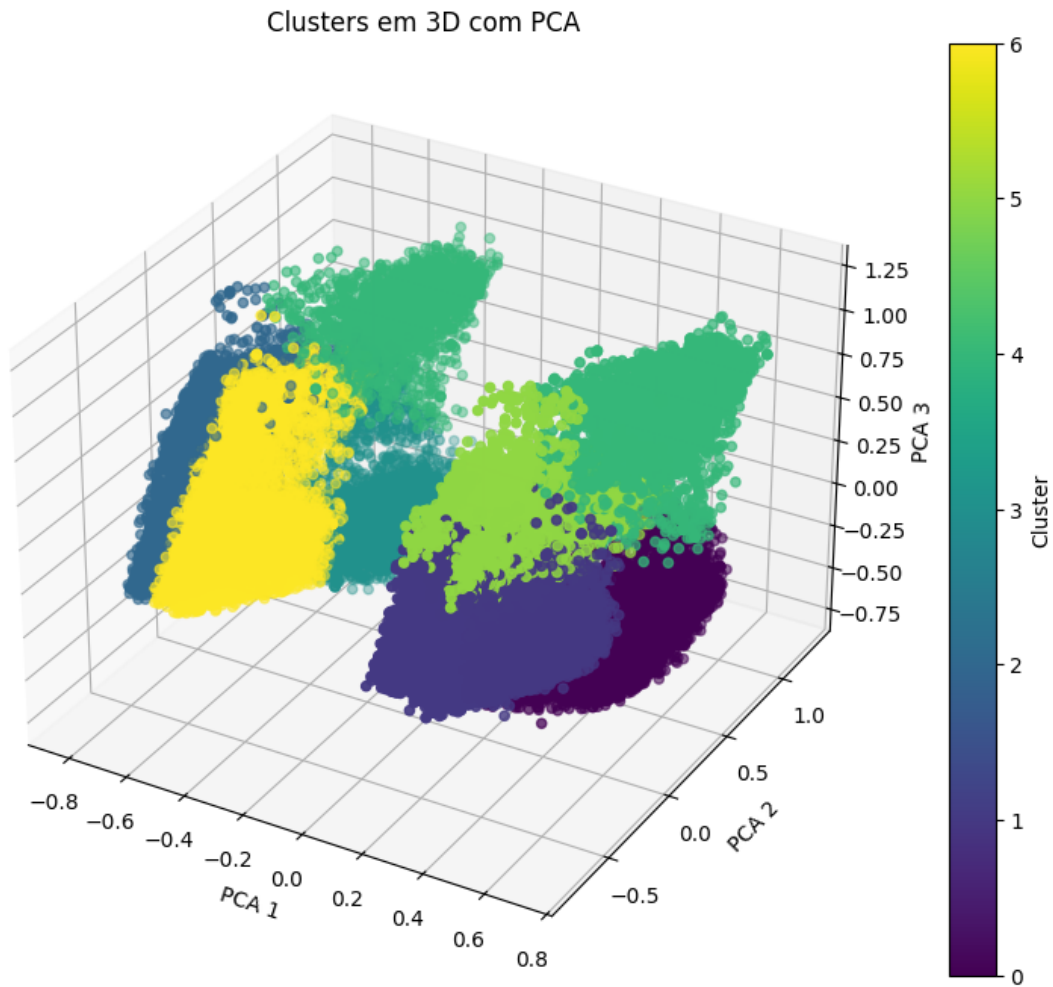
```
[72]: from mpl_toolkits.mplot3d import Axes3D

pca = PCA(n_components=3)
X_pca_3d = pca.fit_transform(escala4)
```

```

fig = plt.figure(figsize=(10,8))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X_pca_3d[:, 0], X_pca_3d[:, 1], X_pca_3d[:, 2],
                    c=df_dados_normalizados4['cluster'], cmap='viridis')
ax.set_xlabel('PCA 1')
ax.set_ylabel('PCA 2')
ax.set_zlabel('PCA 3')
plt.title('Clusters em 3D com PCA')
fig.colorbar(scatter, ax=ax, label='Cluster')
plt.show()

```



5.2 Analisando os Dados

```
[73]: df_dados_normalizados4['cluster'].value_counts()
```

```
[73]: cluster
      1    30831
      0    16102
      6    12653
      2    12354
      4     6349
      5     6044
      3     5407
      Name: count, dtype: int64
```

5.2.1 Dados Originais

```
[74]: df_dados_media = df_dados.groupby('cluster').mean(numeric_only=True) #Paradigm
      ↪ histograms
      df_dados_media
```

```
[74]:
```

	popularity	duration_ms	danceability	energy	key \
cluster					
0	32.795864	218357.337846	0.534535	0.397805	5.056018
1	34.794914	221842.710486	0.590766	0.769514	4.910869
2	33.316254	246772.990853	0.607742	0.755957	9.165129
3	30.463658	217480.252820	0.537491	0.407466	5.525985
4	29.563396	215031.307135	0.355796	0.178936	4.813829
5	26.485771	275757.100099	0.555490	0.760994	4.643779
6	35.907374	233251.716431	0.600431	0.753038	3.128902

	loudness	mode	speechiness	acousticness	instrumentalness \
cluster					
0	-10.693765	1.000000	0.080258	0.645617	0.020292
1	-5.917412	1.000000	0.091659	0.175628	0.021109
2	-6.848869	0.000000	0.094254	0.129465	0.225609
3	-10.909479	0.000000	0.112988	0.749975	0.078030
4	-20.899163	0.659159	0.052027	0.871596	0.829377
5	-8.450949	1.000000	0.076642	0.101583	0.782472
6	-6.378263	0.000000	0.091674	0.146014	0.109119

	liveness	valence	tempo	time_signature
cluster				
0	0.184727	0.421555	113.042340	3.808782
1	0.249172	0.566335	128.199835	3.963673
2	0.212826	0.487327	125.634418	3.956694
3	0.222964	0.422468	113.871474	3.803773
4	0.163899	0.194033	103.489860	3.601827
5	0.200676	0.335155	127.462723	3.923891
6	0.215442	0.499468	125.308435	3.966648

5.2.2 Dados Normalizados

```
[75]: df_dados_normalizados4_media = df_dados_normalizados4.groupby('cluster').mean()
      ↪ # Para gráfico de médias
      df_dados_normalizados4_media
```

```
[75]:
```

	popularity	duration_ms	danceability	energy	key	loudness	\
cluster							
0	0.327959	0.041693	0.542675	0.397805	0.459638	-0.215900	
1	0.347949	0.042358	0.599762	0.769514	0.446443	-0.119469	
2	0.333163	0.047118	0.616997	0.755957	0.833194	-0.138274	
3	0.304637	0.041525	0.545676	0.407466	0.502362	-0.220256	
4	0.295634	0.041058	0.361214	0.178936	0.437621	-0.421941	
5	0.264858	0.052653	0.563950	0.760994	0.422162	-0.170619	
6	0.359074	0.044537	0.609575	0.753038	0.284446	-0.128773	

	mode	speechiness	acousticness	instrumentalness	liveness	\
cluster						
0	1.000000	0.083168	0.648210	0.020292	0.184727	
1	1.000000	0.094983	0.176334	0.021109	0.249172	
2	0.000000	0.097672	0.129985	0.225609	0.212826	
3	0.000000	0.117086	0.752987	0.078030	0.222964	
4	0.659159	0.053914	0.875096	0.829377	0.163899	
5	1.000000	0.079422	0.101991	0.782472	0.200676	
6	0.000000	0.094999	0.146601	0.109119	0.215442	

	valence	tempo	time_signature
cluster			
0	0.423673	0.464484	0.761756
1	0.569181	0.526765	0.792735
2	0.489776	0.516224	0.791339
3	0.424591	0.467891	0.760755
4	0.195008	0.425233	0.720365
5	0.336840	0.523736	0.784778
6	0.501978	0.514884	0.793330

5.2.3 Gênero de músicas por cluster

```
[76]: genero_por_cluster = df_musicas_maxabs.groupby('cluster').apply(
      lambda x: x['track_genre'].value_counts().head(7),
      include_groups=False
    )
      genero_por_cluster
```

```
[76]: cluster  track_genre
0           honky-tonk      819
           comedy         604
           cantopop       588
```

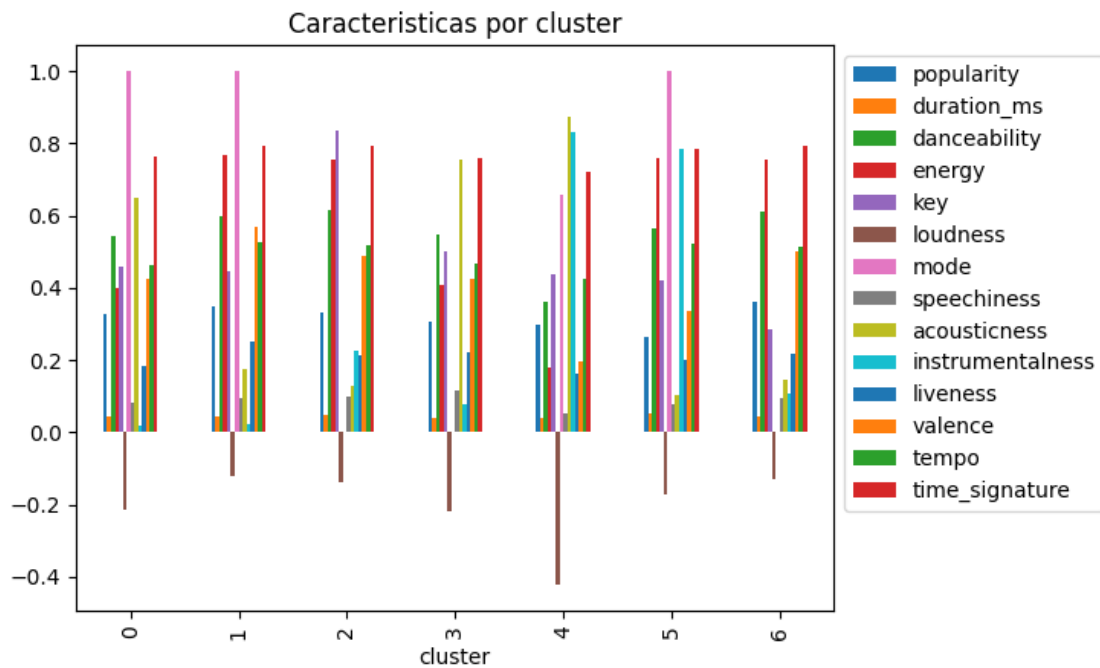
	acoustic	556
	show-tunes	481
	opera	466
	mandopop	445
1	j-idol	661
	forro	628
	party	600
	sertanejo	577
	power-pop	571
	kids	551
	pagode	523
2	chicago-house	327
	detroit-techno	322
	drum-and-bass	301
	breakbeat	271
	minimal-techno	263
	hardstyle	258
	deep-house	247
3	romance	503
	tango	447
	comedy	250
	chill	150
	study	144
	malay	137
	opera	132
4	new-age	741
	sleep	693
	ambient	657
	classical	604
	piano	418
	disney	415
	iranian	400
5	minimal-techno	419
	detroit-techno	388
	grindcore	383
	black-metal	311
	study	281
	breakbeat	273
	chicago-house	269
6	dancehall	265
	deep-house	263
	dance	244
	metalcore	239
	turkish	236
	k-pop	231
	hardstyle	229

Name: count, dtype: int64

5.3 Gráficos

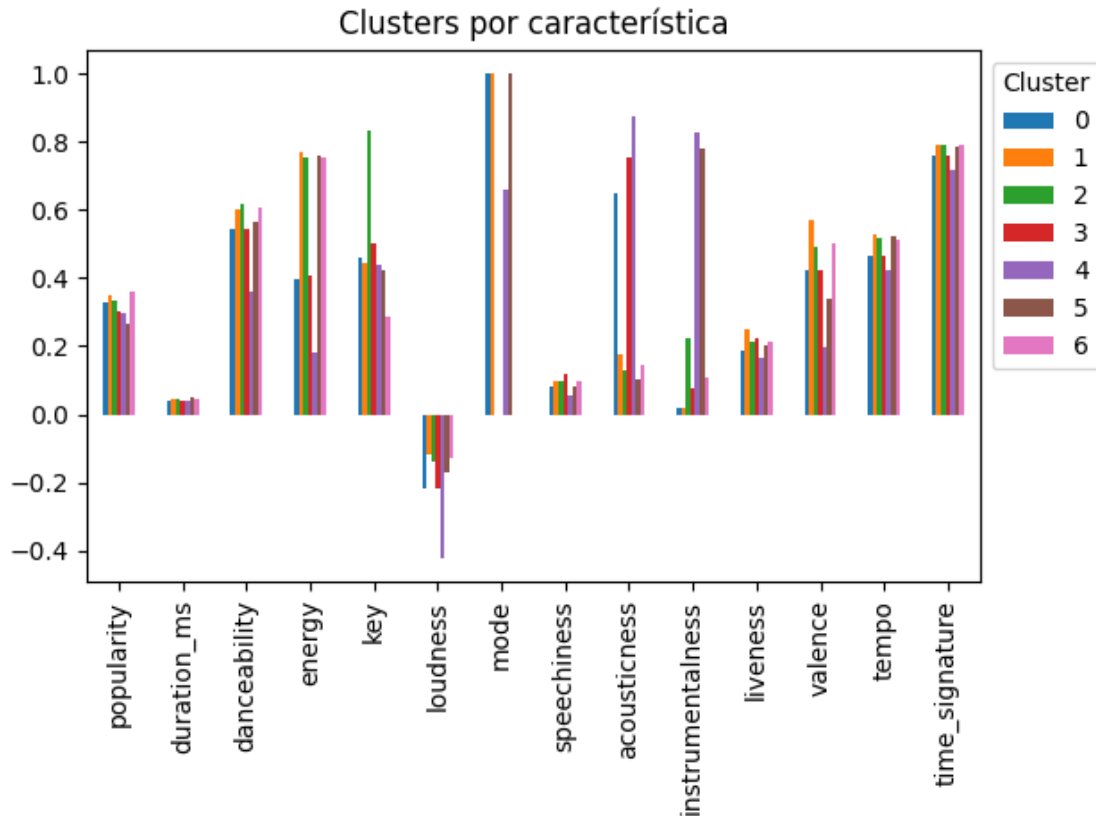
5.3.1 Características x Cluster

```
[77]: df_datos_normalizados4_media.plot(kind='bar')
plt.title('Características por cluster')
plt.legend(bbox_to_anchor=(1.0, 1.0))
plt.show()
```



5.3.2 Cluster x Características

```
[78]: df_datos_normalizados4_media.T.plot(kind='bar')
plt.title('Clusters por característica')
plt.legend(title='Cluster', bbox_to_anchor=(1.0, 1.0))
plt.tight_layout()
plt.show()
```



5.3.3 Histogramas

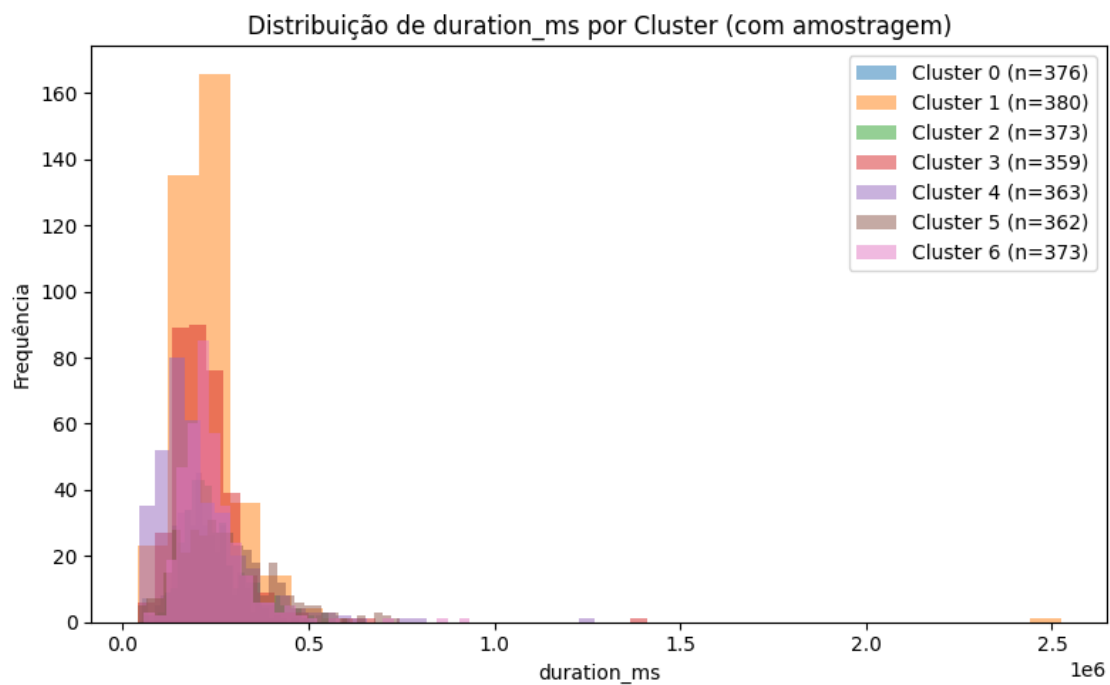
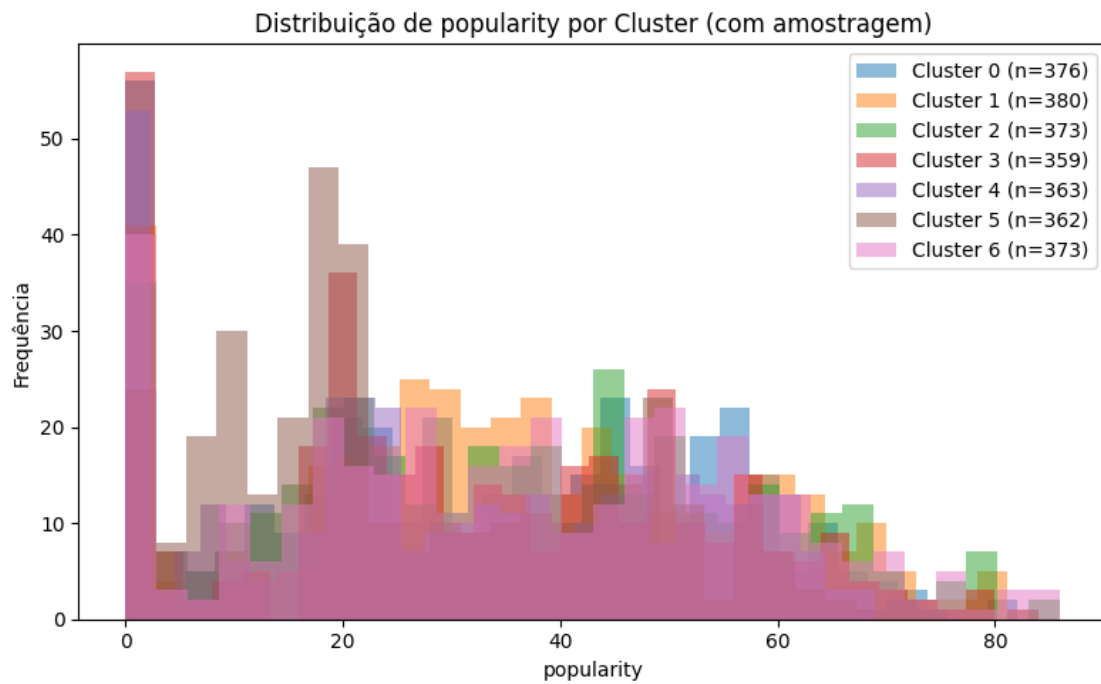
```
[79]: import matplotlib.pyplot as plt

for feature in df_dados.columns.drop('cluster'):
    plt.figure(figsize=(8, 5))

    for cluster_id in sorted(df_dados['cluster'].unique()):
        subset = df_dados[df_dados['cluster'] == cluster_id]
        tamanho = tamanho_amostra(len(subset))
        amostra = subset.sample(n=min(tamanho, len(subset)), random_state=42)
        ↪ # Garante que não passa o tamanho real
        plt.hist(amostra[feature], bins=30, alpha=0.5, label=f'Cluster_{cluster_id} (n={len(amostra)})')

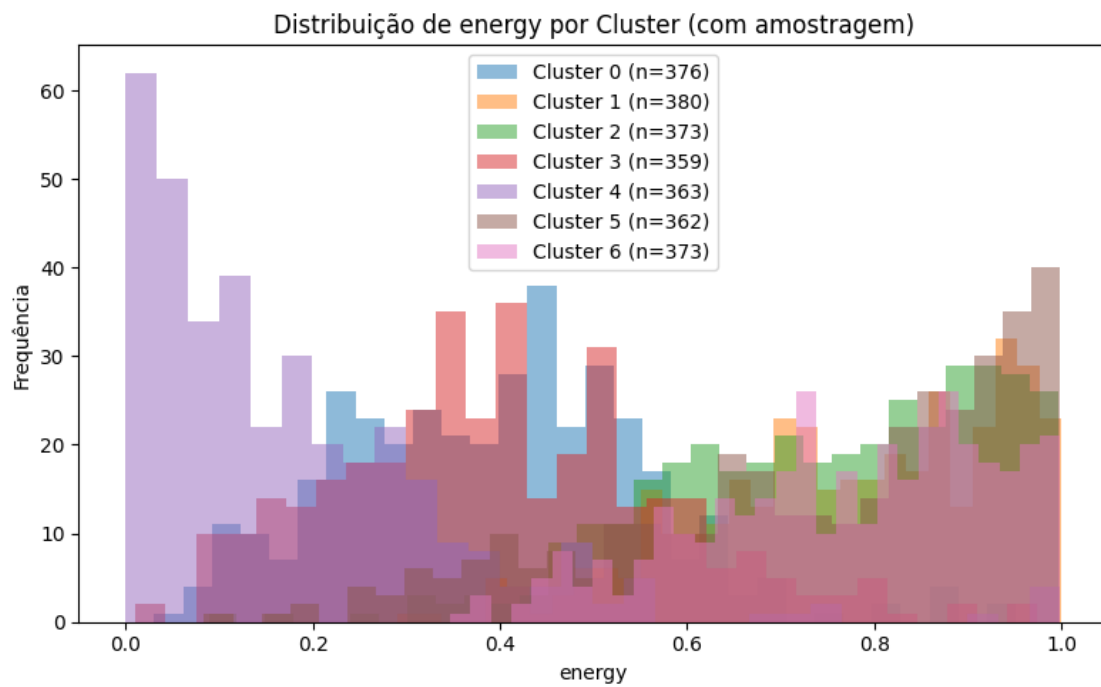
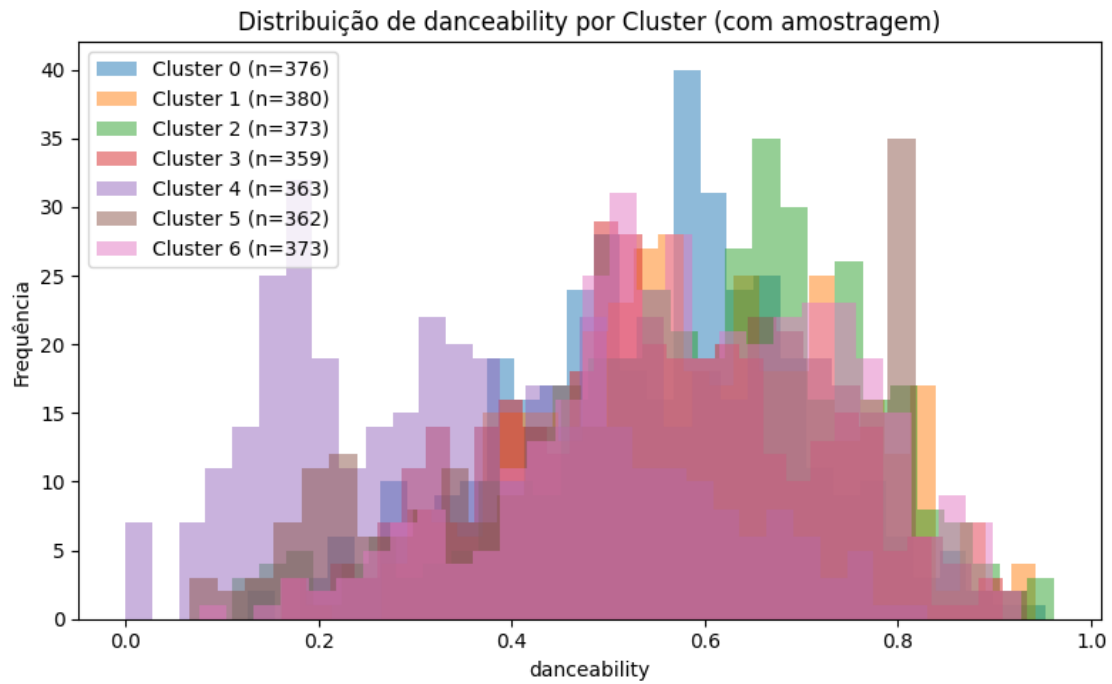
    plt.title(f'Distribuição de {feature} por Cluster (com amostragem)')
    plt.xlabel(feature)
    plt.ylabel('Frequência')
    plt.legend()
    plt.tight_layout()
```

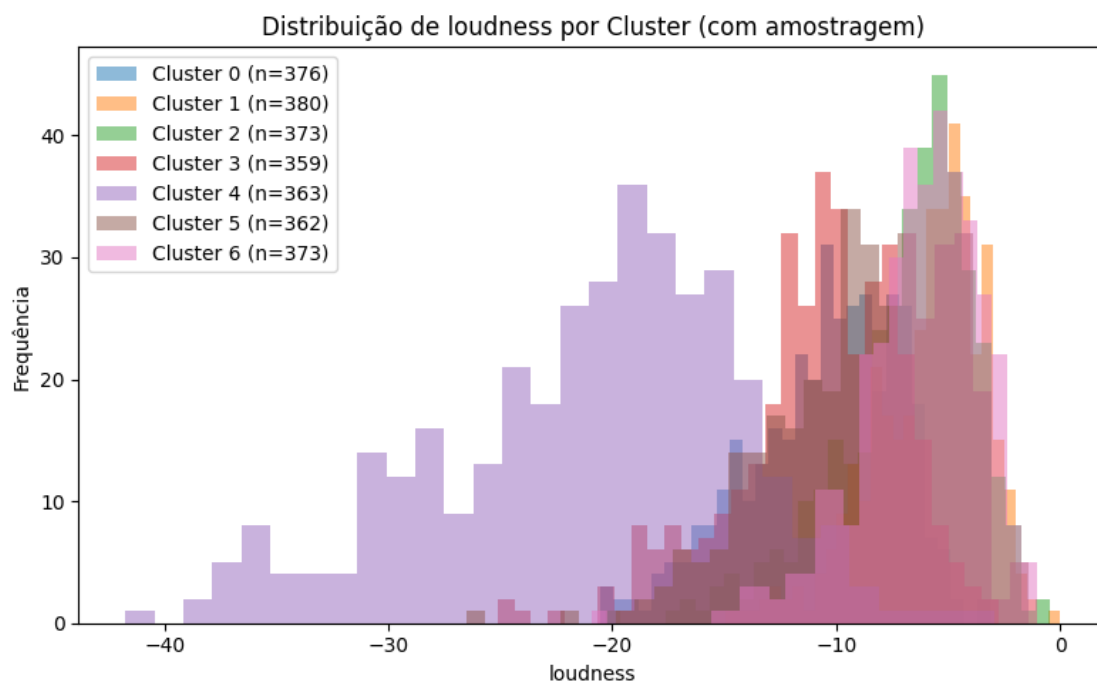
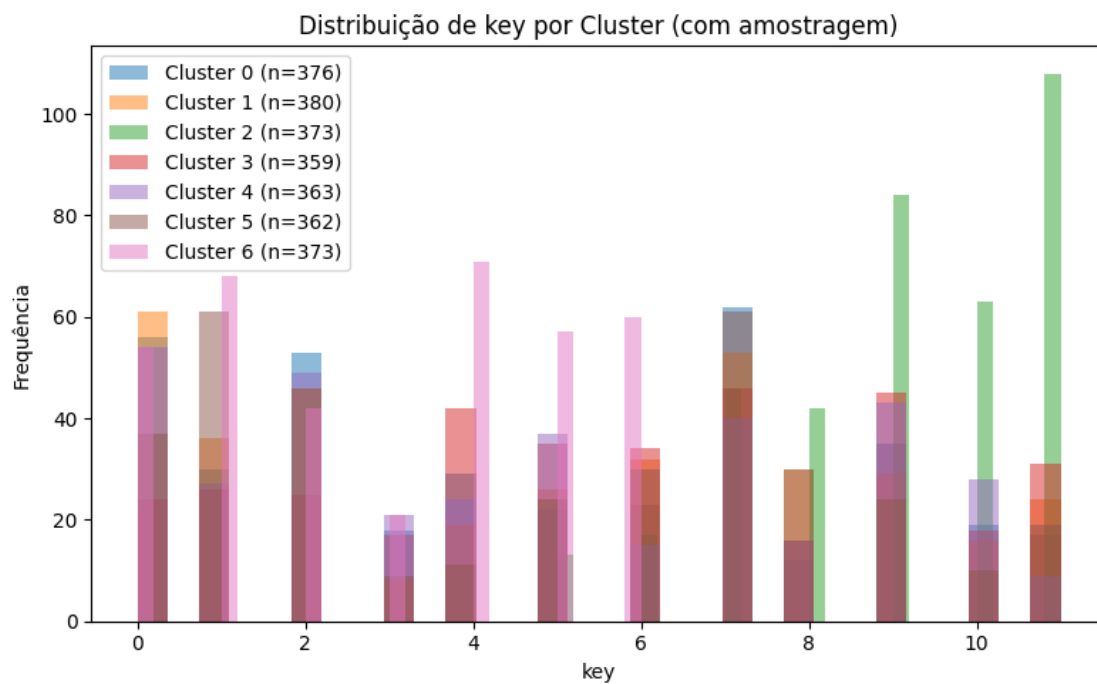
```
plt.show()
```

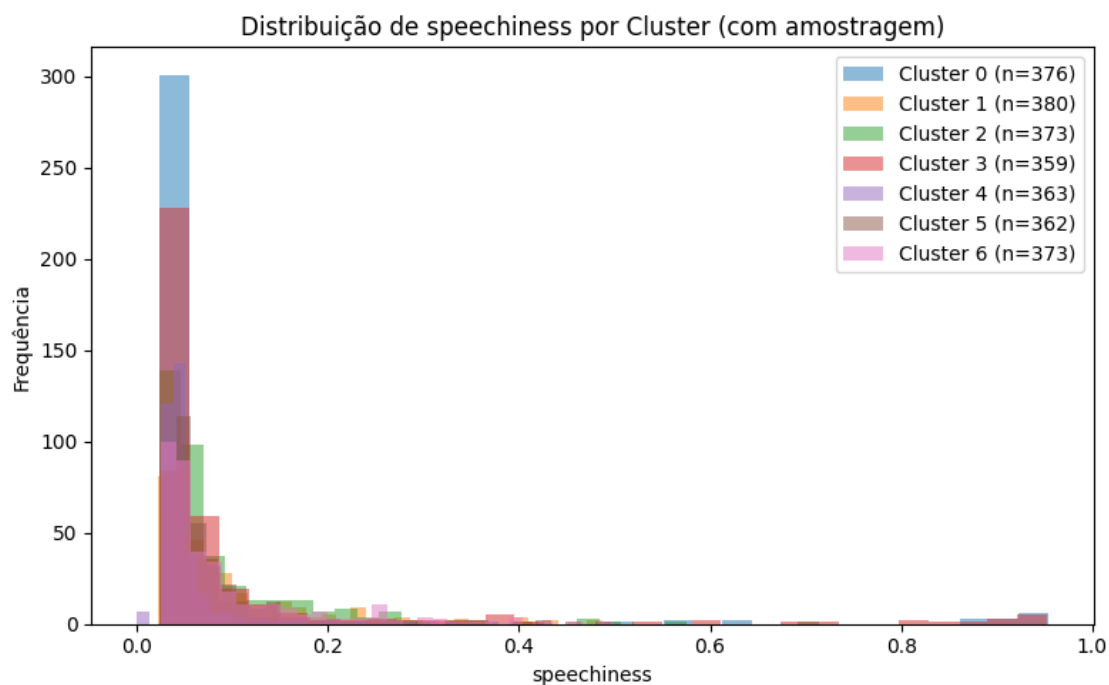
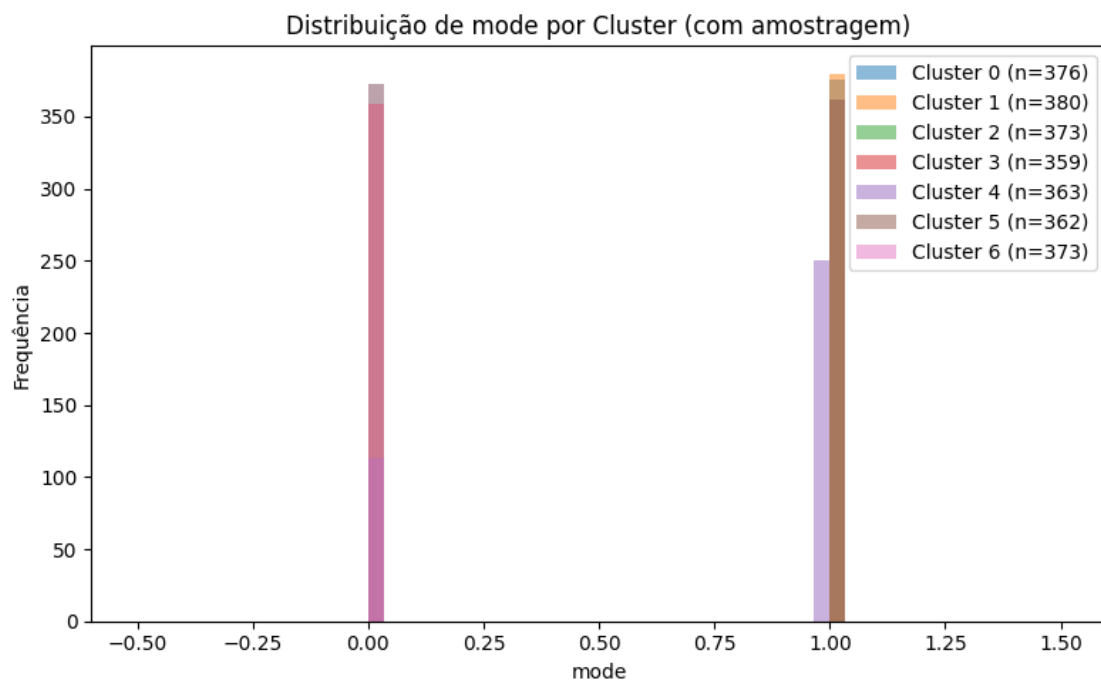


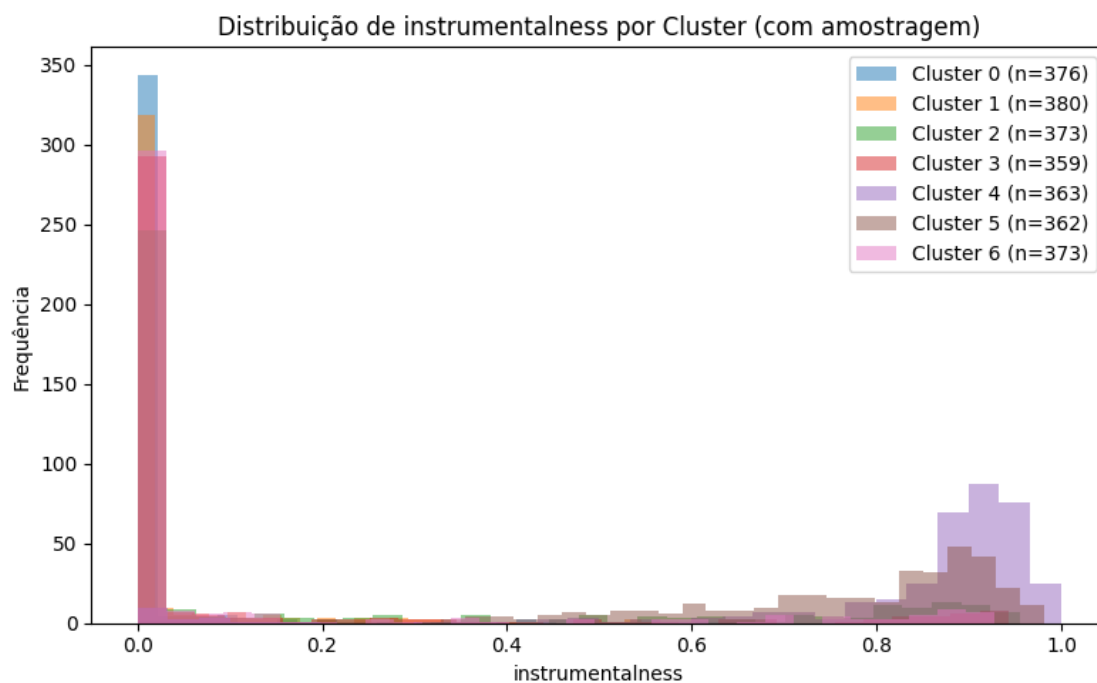
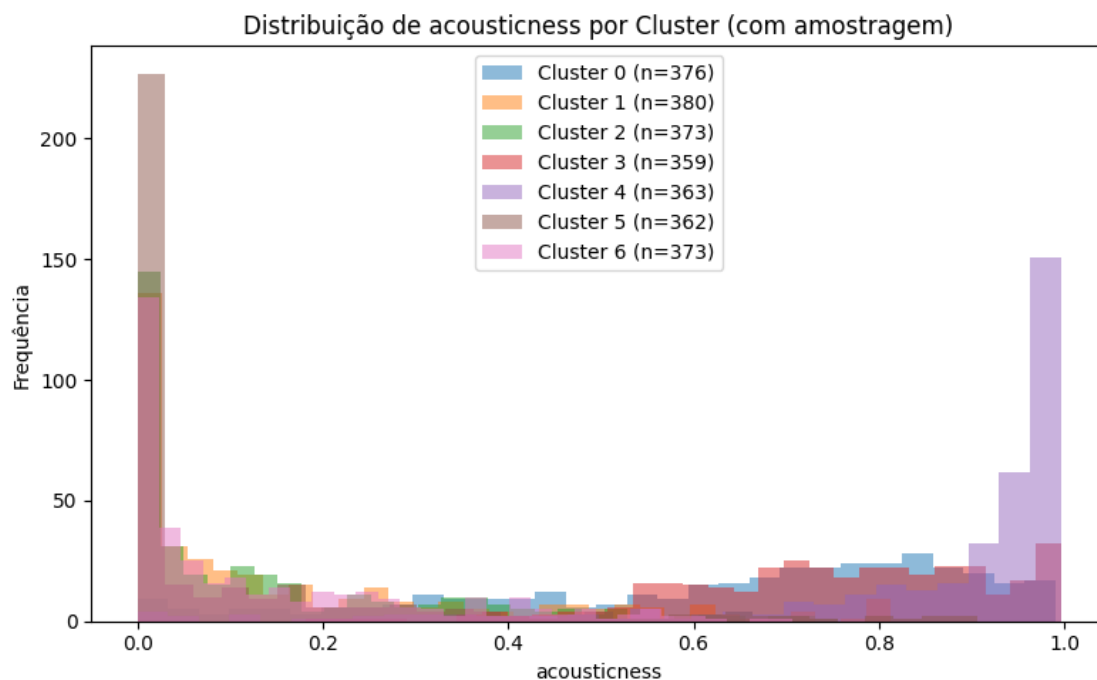
C:\Users\conta\AppData\Local\Temp\ipykernel_6744\430233715.py:16: UserWarning:

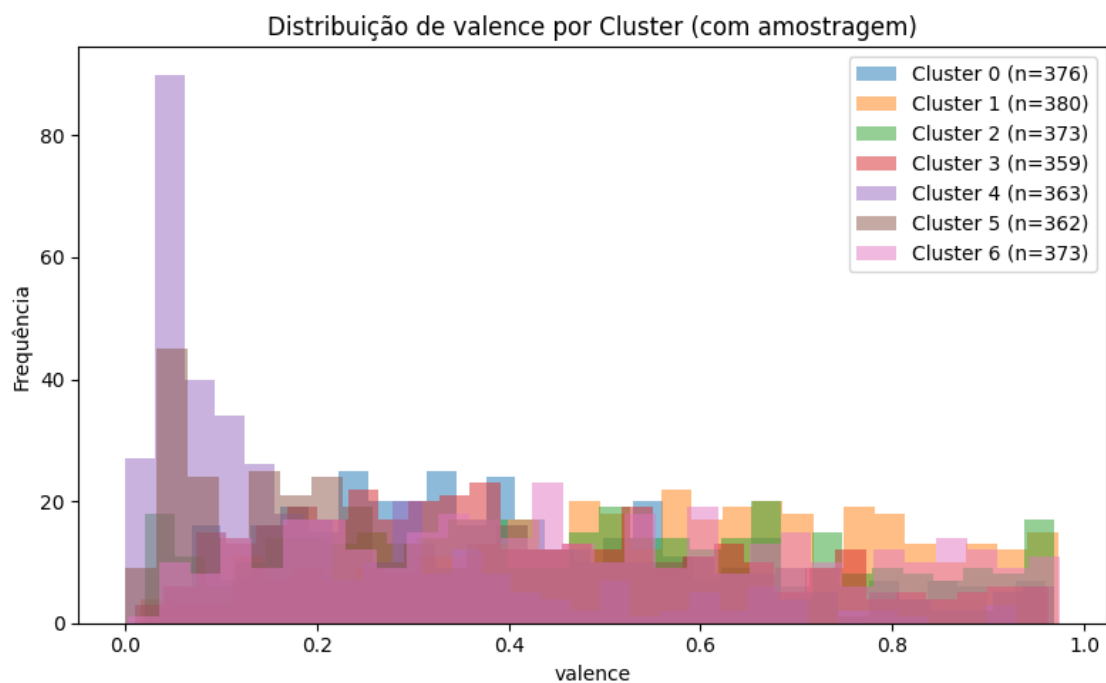
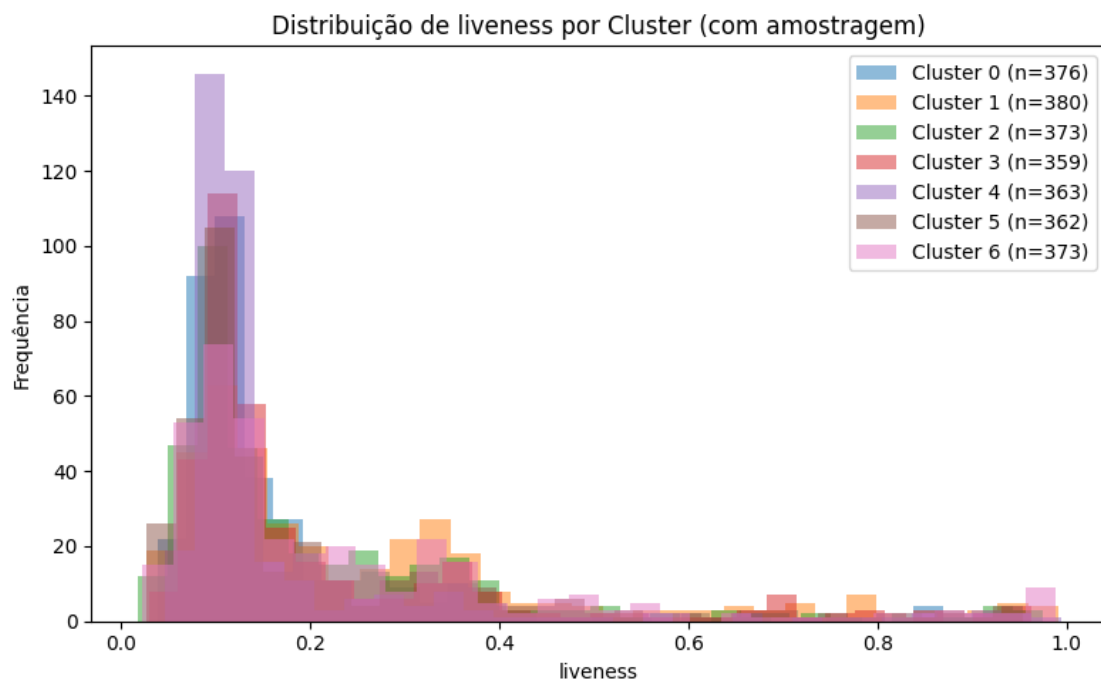
Creating legend with `loc="best"` can be slow with large amounts of data.
`plt.tight_layout()`

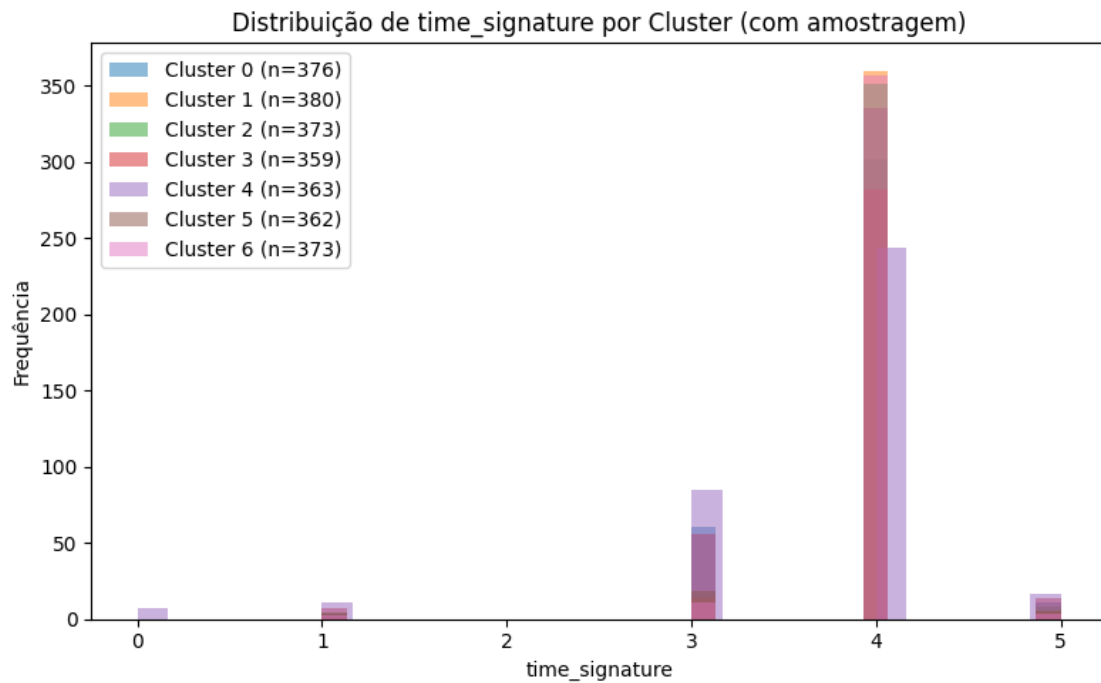
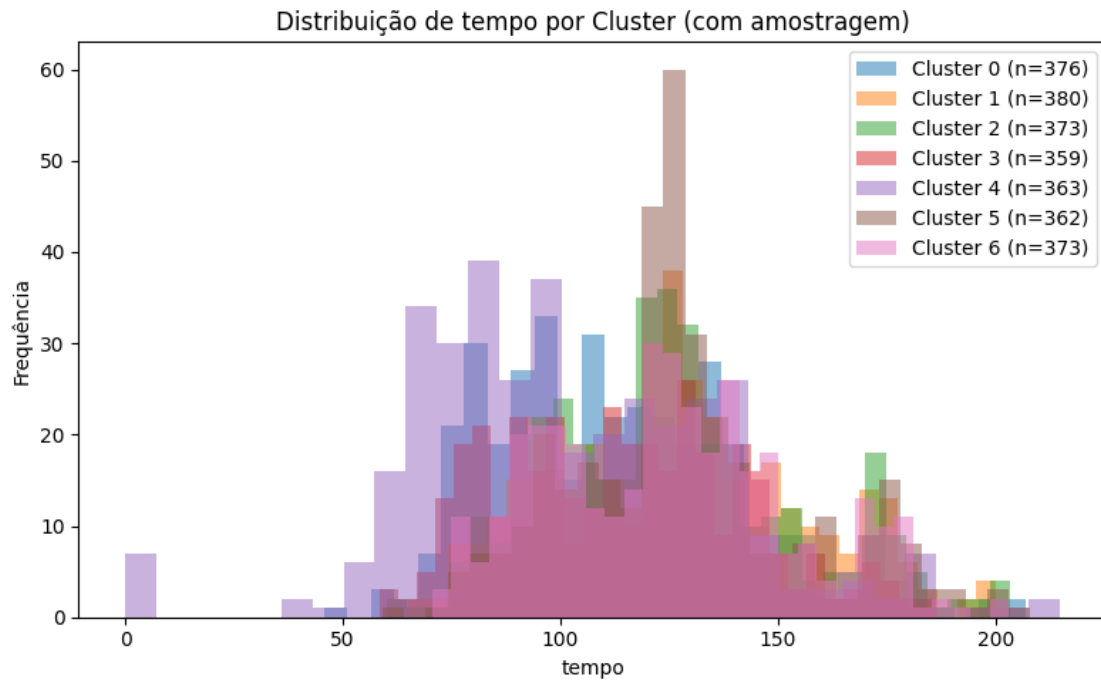












5.4 Musica

```
[80]: recomendar_musicas(track_id='2aibwv5hGXSgw7Yru8IYT0',df_dados_normalizados=df_dados_normalizados)
```

Cluster: 1

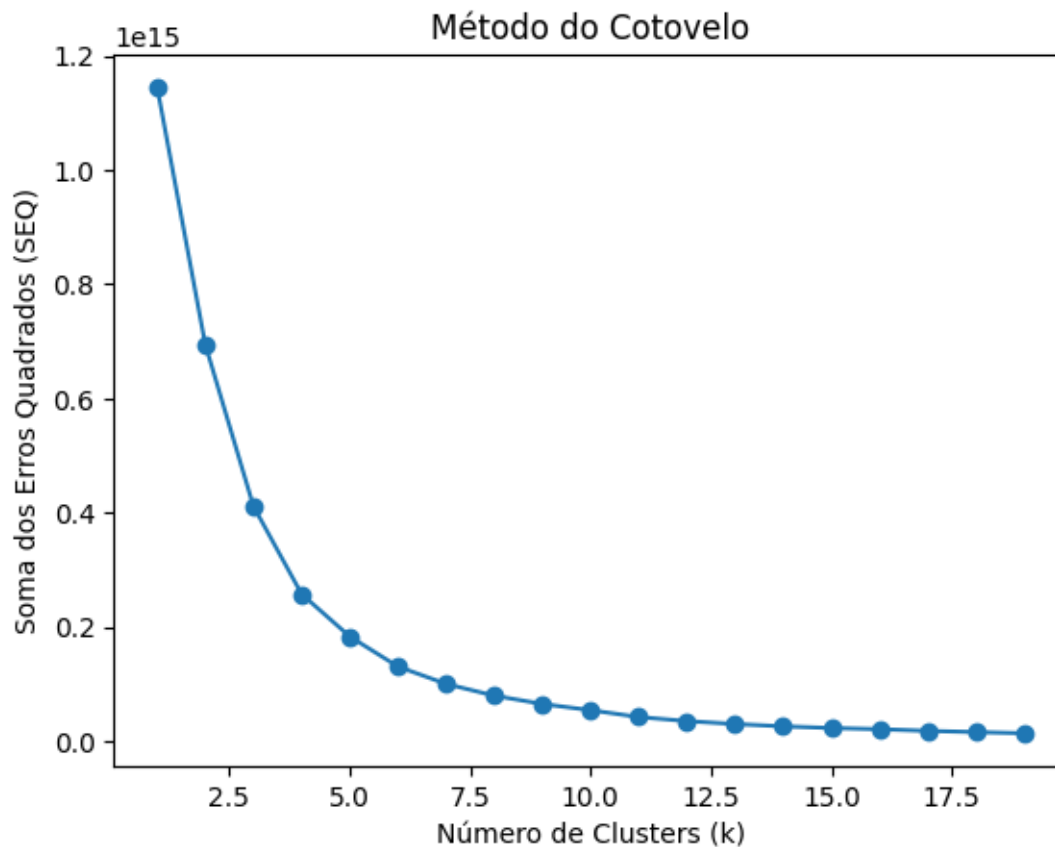
```
[80]:
```

	track_name	artists	track_genre
2110	Monster	Skillet	alt-rock
44195	Fake It	Seether	grunge
56827	I'm Born To Run	American Authors	indie-pop
82063	Buddy Holly	Weezer	power-pop
2357	HandClap	Fitz and The Tantrums	alt-rock
65364	Love Maze	BTS	k-pop
65464	Alcohol-Free	TWICE	k-pop
30764	What Would You Do?	Joel Corry;David Guetta;Bryson Tiller	edm
38090	Teddy Picker	Arctic Monkeys	garage
65823	DOMINO	Stray Kids	k-pop

6 Sem normalizar os dados

```
[81]: df_dados = df_musicas.select_dtypes(include='number')
soma_erros = calcular_soma_erros(df_dados)
```

```
[82]: plt.plot(range(1, 20), soma_erros, marker='o')
plt.xlabel('Número de Clusters (k)')
plt.ylabel('Soma dos Erros Quadrados (SEQ)')
plt.title('Método do Cotovelo')
plt.show()
```



```
[83]: numero_clusters = optimal_number_of_clusters(soma_erros)
      numero_clusters
```

```
[83]: 6
```

```
[84]: from sklearn.cluster import KMeans
      kmeans = KMeans(n_clusters=numero_clusters, random_state=14)
      clusters = kmeans.fit_predict(df_dados)

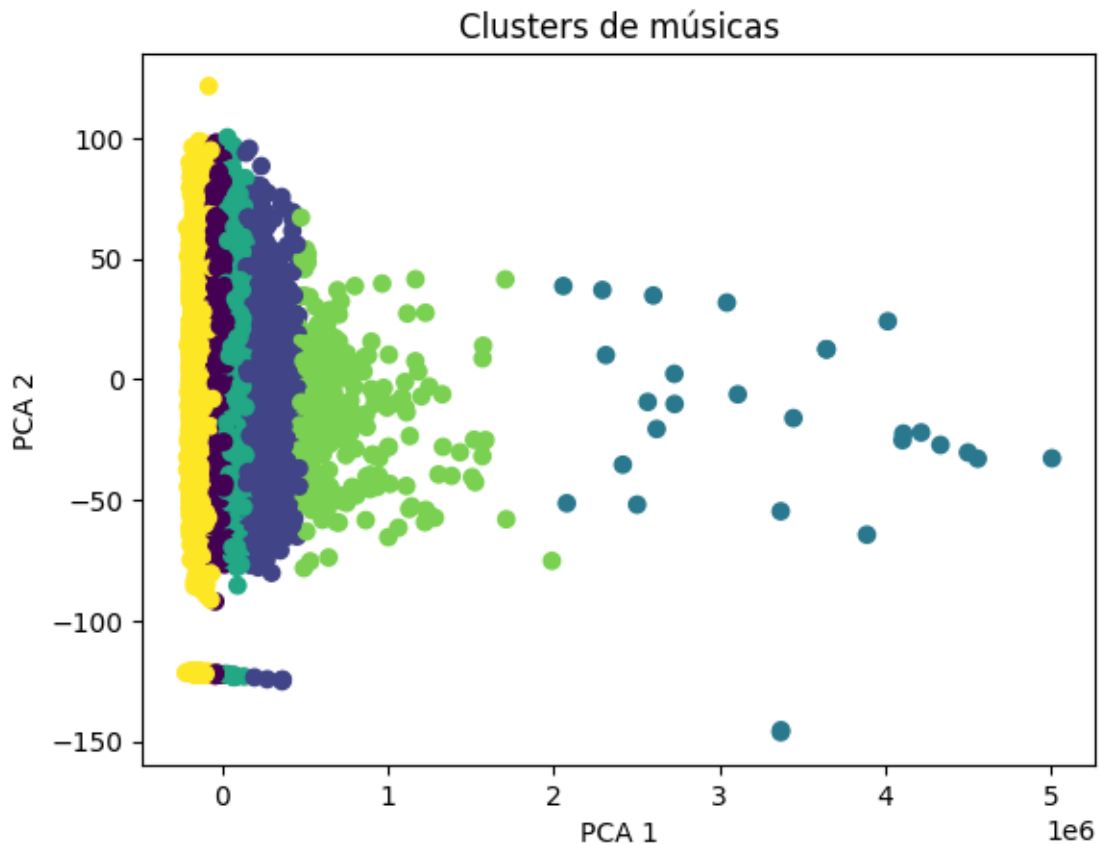
      df_musicas_ = df_musicas.copy()
      df_musicas_['cluster'] = clusters
      df_dados['cluster'] = clusters
```

6.1 Plotagem 2D e 3D

```
[85]: from sklearn.decomposition import PCA
      import matplotlib.pyplot as plt

      pca = PCA(n_components=2)
      X_pca = pca.fit_transform(df_dados)
```

```
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df_dados['cluster'], cmap='viridis')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.title('Clusters de músicas')
plt.show()
```

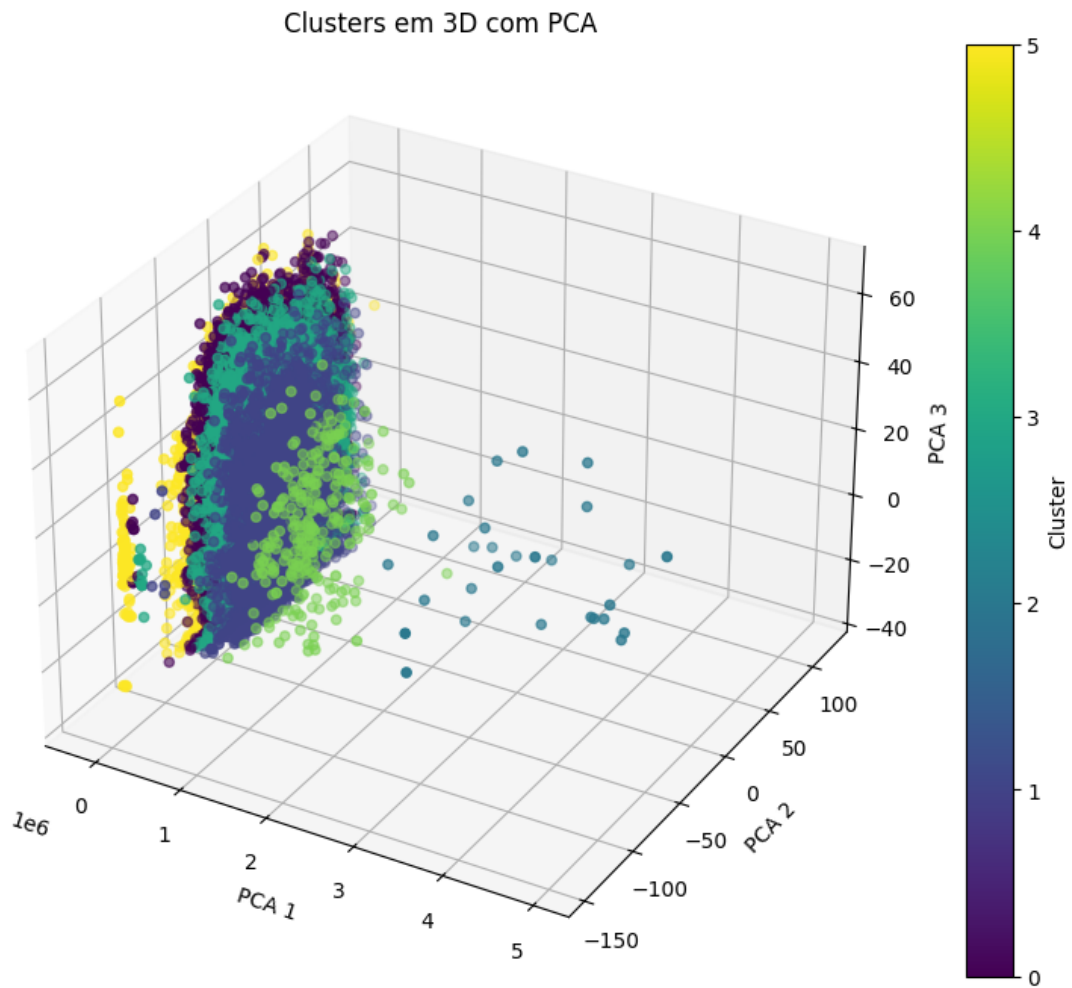


```
[86]: from mpl_toolkits.mplot3d import Axes3D

pca = PCA(n_components=3)
X_pca_3d = pca.fit_transform(df_dados)

fig = plt.figure(figsize=(10,8))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X_pca_3d[:, 0], X_pca_3d[:, 1], X_pca_3d[:, 2],
                    c=df_dados['cluster'], cmap='viridis')
ax.set_xlabel('PCA 1')
ax.set_ylabel('PCA 2')
ax.set_zlabel('PCA 3')
```

```
plt.title('Clusters em 3D com PCA')
fig.colorbar(scatter, ax=ax, label='Cluster')
plt.show()
```



6.2 Analisando os Dados

```
[87]: df_dados['cluster'].value_counts()
```

```
[87]: cluster
0    40722
5    22719
3    20455
1     5527
4       289
2         28
Name: count, dtype: int64
```

6.2.1 Média por colunas em cada cluster

```
[88]: df_dados_media = df_dados.groupby('cluster').mean(numeric_only=True) #Para ↵  
      ↵histogramas  
df_dados_media
```

```
[88]:
```

	popularity	duration_ms	danceability	energy	key	\
cluster						
0	34.816856	2.111165e+05	0.572026	0.661627	5.293674	
1	27.684096	4.471515e+05	0.542549	0.637954	5.438755	
2	22.642857	3.539151e+06	0.535950	0.669855	4.821429	
3	33.698802	2.946916e+05	0.538147	0.660156	5.307993	
4	24.716263	9.428537e+05	0.417203	0.511397	4.989619	
5	31.310929	1.362483e+05	0.572769	0.563295	5.209868	

	loudness	mode	speechiness	acousticness	instrumentalness	\
cluster						
0	-7.619783	0.633441	0.083476	0.299293	0.105262	
1	-10.198387	0.583318	0.081148	0.243171	0.452481	
2	-9.679250	0.642857	0.054343	0.304207	0.357049	
3	-8.196155	0.620093	0.075459	0.288332	0.187837	
4	-12.935190	0.626298	0.115398	0.453128	0.409456	
5	-9.876261	0.671684	0.106557	0.435370	0.211471	

	liveness	valence	tempo	time_signature
cluster				
0	0.215343	0.488386	122.882838	3.920583
1	0.223716	0.335568	122.529896	3.894156
2	0.166929	0.396290	123.613500	3.607143
3	0.216945	0.422715	123.558110	3.908726
4	0.290933	0.295293	115.700588	3.820069
5	0.217392	0.512559	119.193605	3.847881

6.3 Gêneros de músicas por cluster

```
[89]: genero_por_cluster = df_musicas_.groupby('cluster').apply(  
      lambda x: x['track_genre'].value_counts().head(7),  
      include_groups=False  
)  
genero_por_cluster
```

```
[89]: cluster  track_genre  
0          country      647  
          dancehall      635  
          dance        622  
          alt-rock      599  
          spanish      595
```

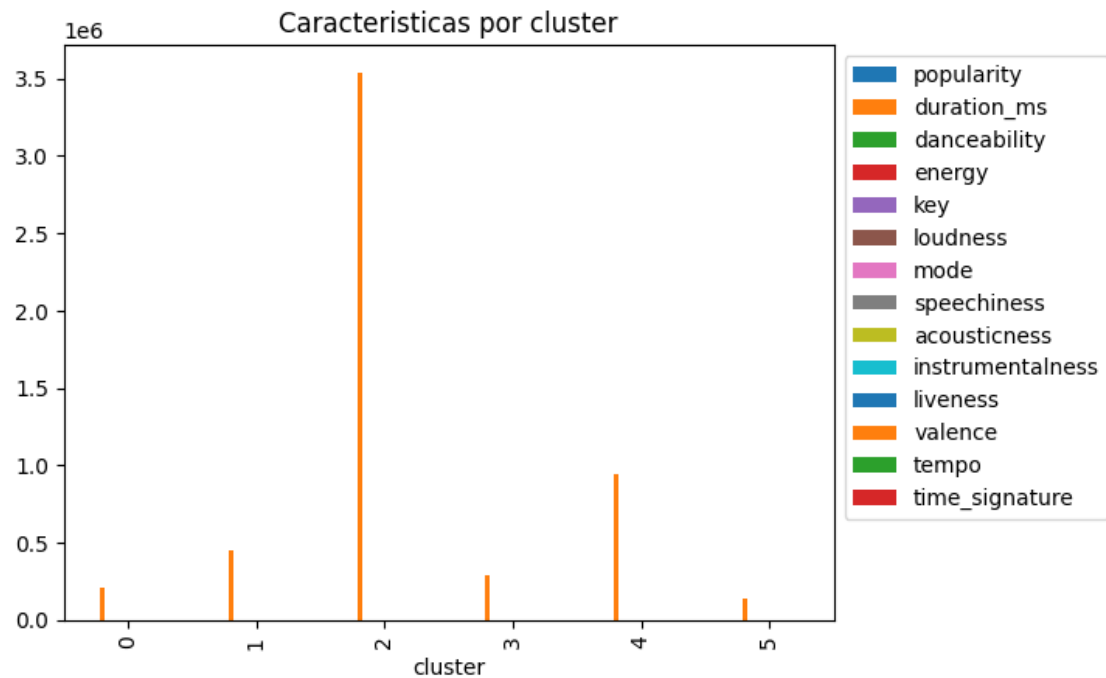

	party	594
	disco	586
1	minimal-techno	439
	detroit-techno	416
	chicago-house	405
	iranian	256
	trance	216
	black-metal	210
	new-age	181
2	breakbeat	7
	chicago-house	3
	sleep	3
	classical	2
	detroit-techno	2
	guitar	2
	k-pop	2
3	salsa	611
	j-idol	520
	drum-and-bass	517
	pop-film	489
	breakbeat	440
	mandopop	423
	malay	416
4	iranian	27
	new-age	27
	folk	24
	detroit-techno	22
	black-metal	21
	classical	20
	comedy	12
5	study	856
	children	710
	grindcore	707
	honky-tonk	689
	disney	620
	kids	600
	sleep	568

Name: count, dtype: int64

6.4 Gráficos

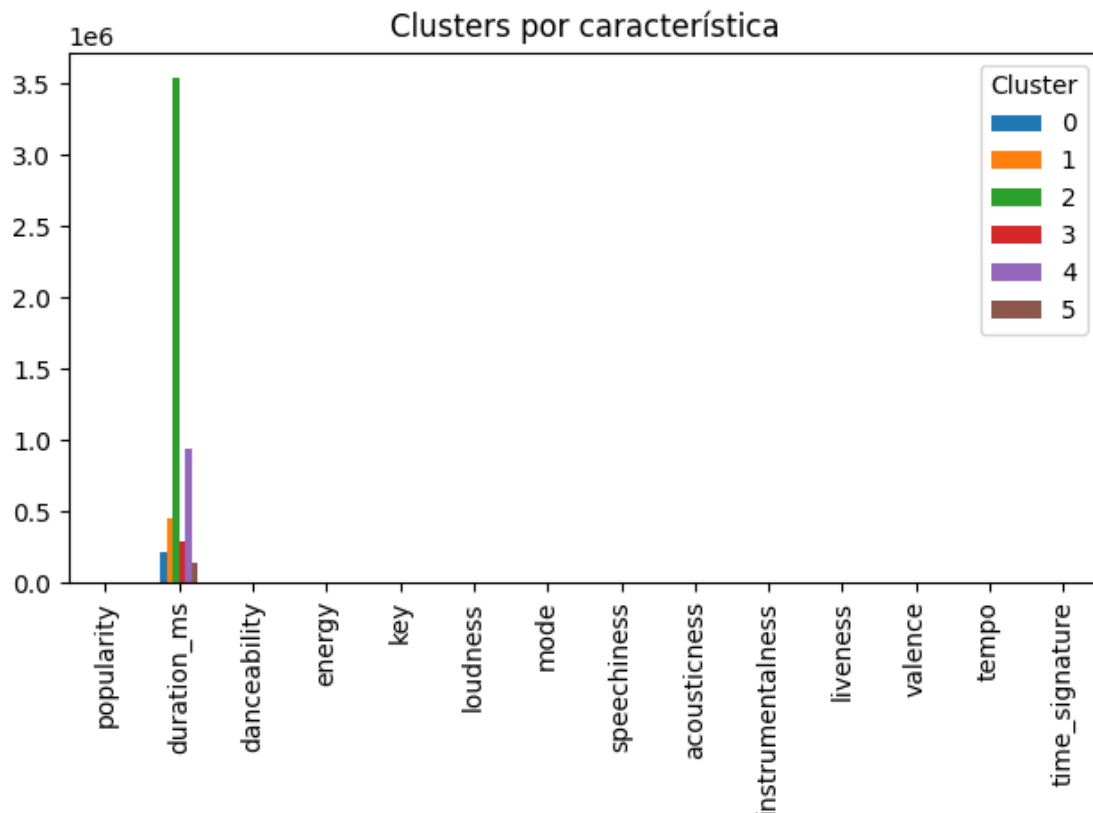
6.4.1 Características x Cluster

```
[90]: df_datos_media.plot(kind='bar')
plt.title('Características por cluster')
plt.legend(bbox_to_anchor=(1.0, 1.0))
plt.show()
```



6.4.2 Cluster x Características

```
[91]: df_datos_media.T.plot(kind='bar')
plt.title('Clusters por característica')
plt.legend(title='Cluster', bbox_to_anchor=(1.0, 1.0))
plt.tight_layout()
plt.show()
```



6.4.3 Histogramas

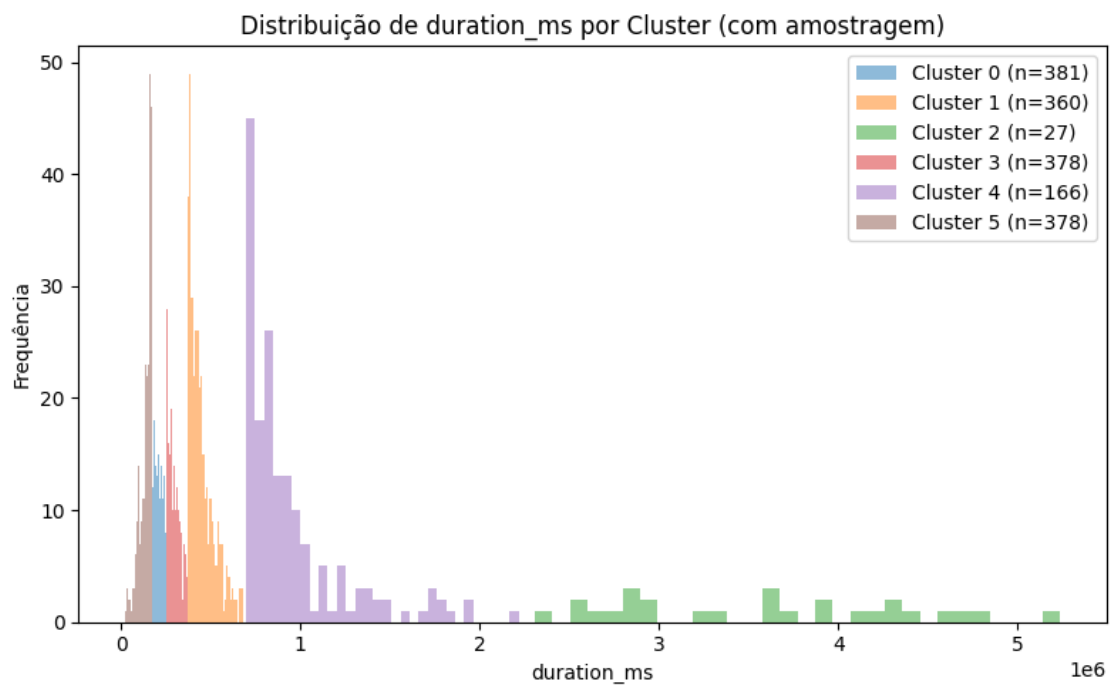
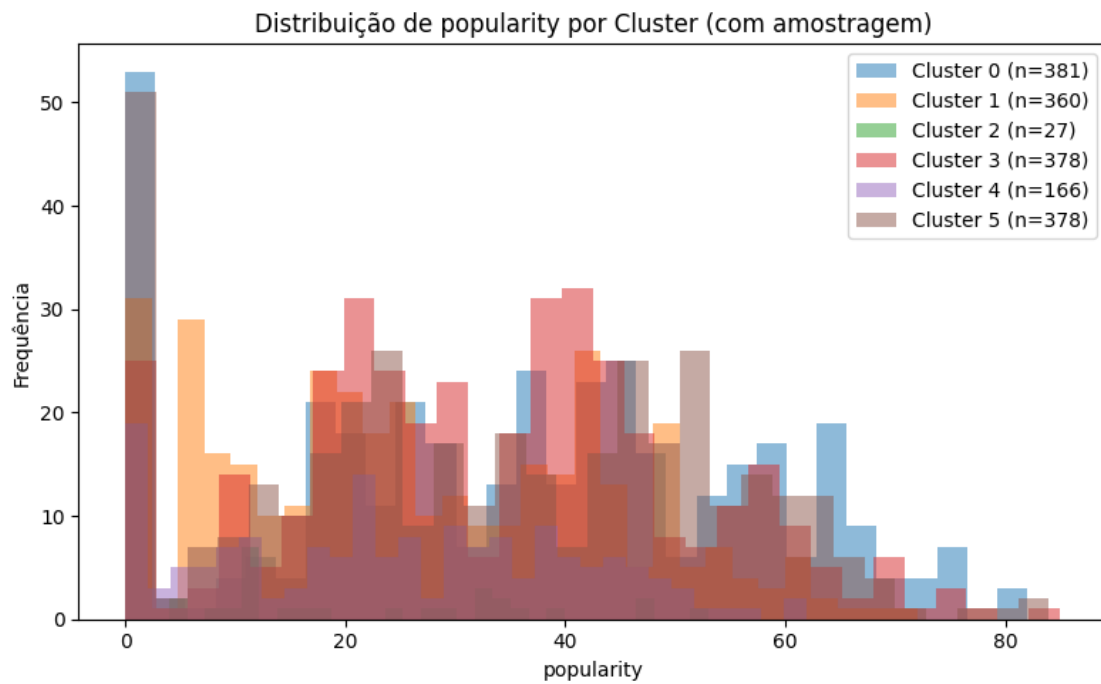
```
[92]: import matplotlib.pyplot as plt

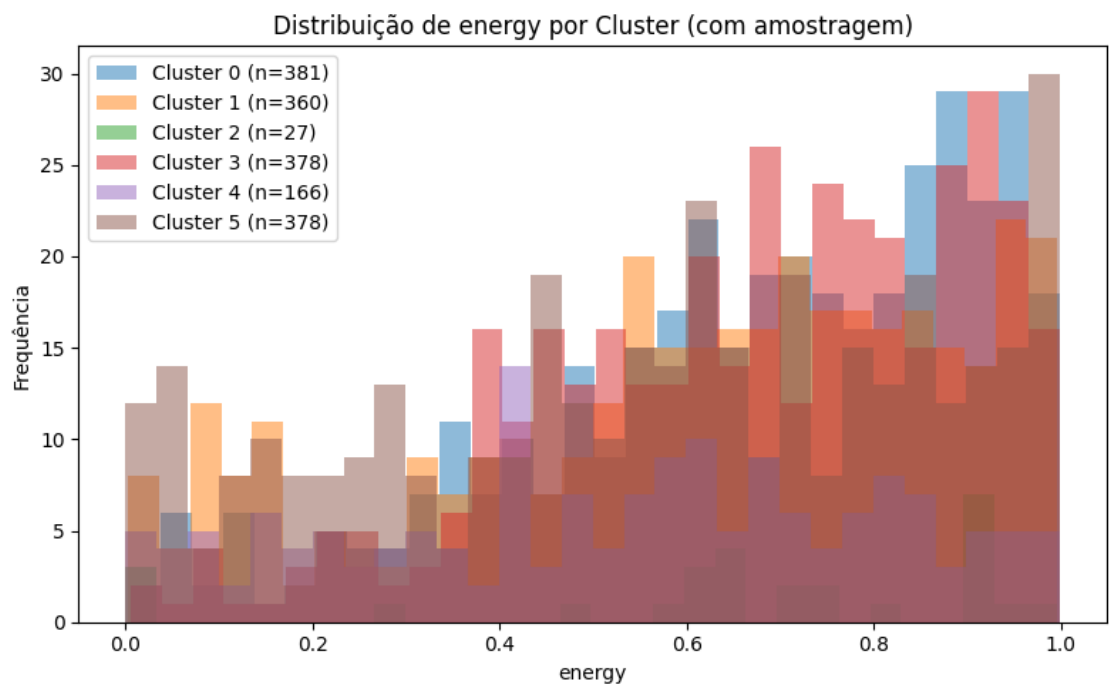
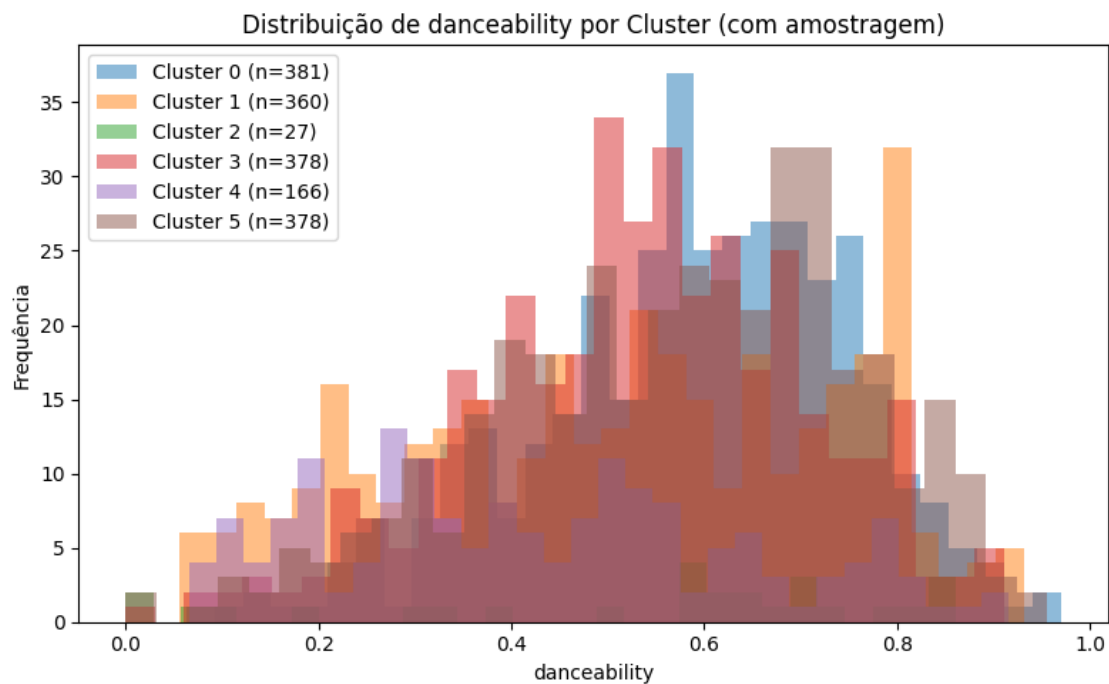
for feature in df_dados.columns.drop('cluster'):
    plt.figure(figsize=(8, 5))

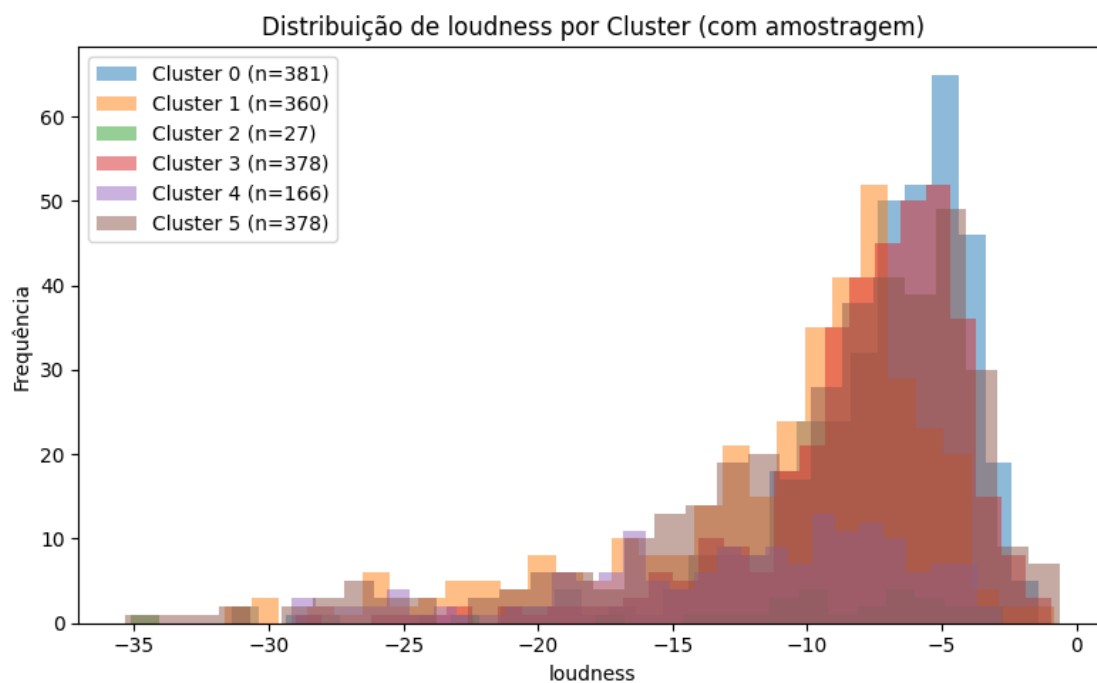
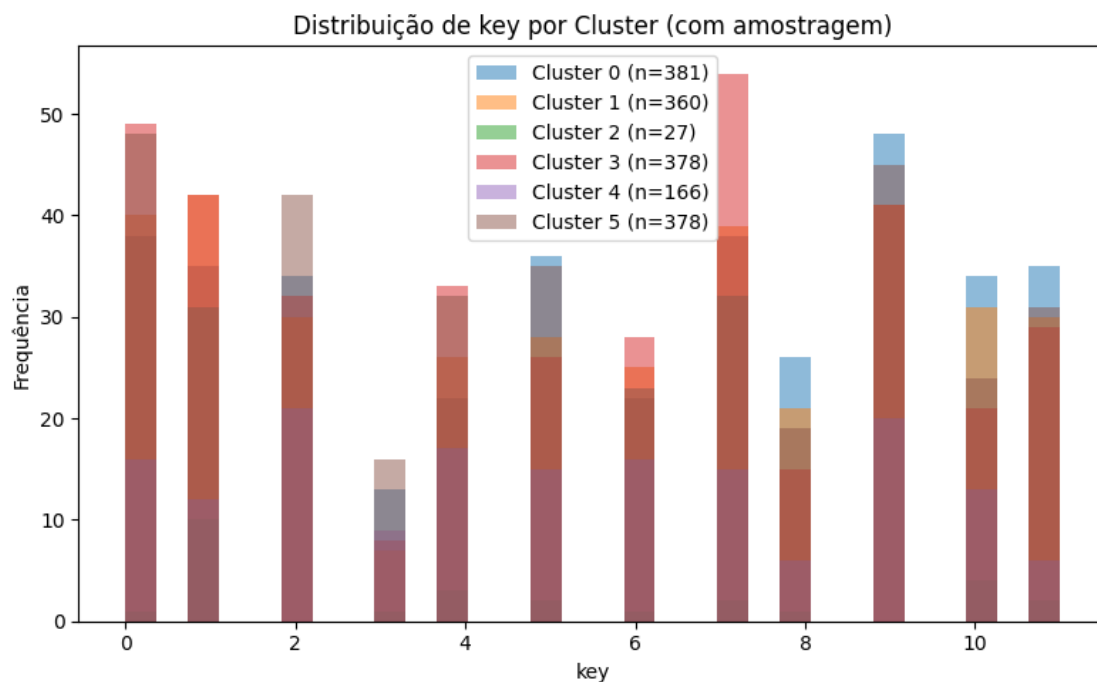
    for cluster_id in sorted(df_dados['cluster'].unique()):
        subset = df_dados[df_dados['cluster'] == cluster_id]
        tamanho = tamanho_amostra(len(subset))
        amostra = subset.sample(n=min(tamanho, len(subset)), random_state=42)
        # Garante que não passa o tamanho real
        plt.hist(amostra[feature], bins=30, alpha=0.5, label=f'Cluster {cluster_id} (n={len(amostra)})')

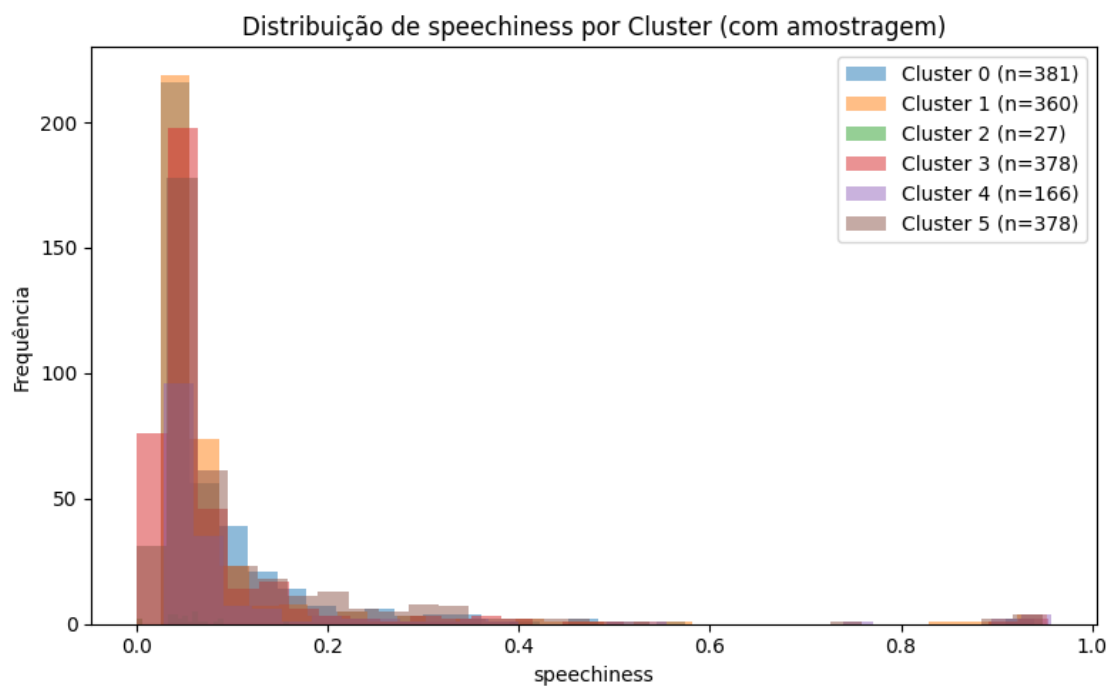
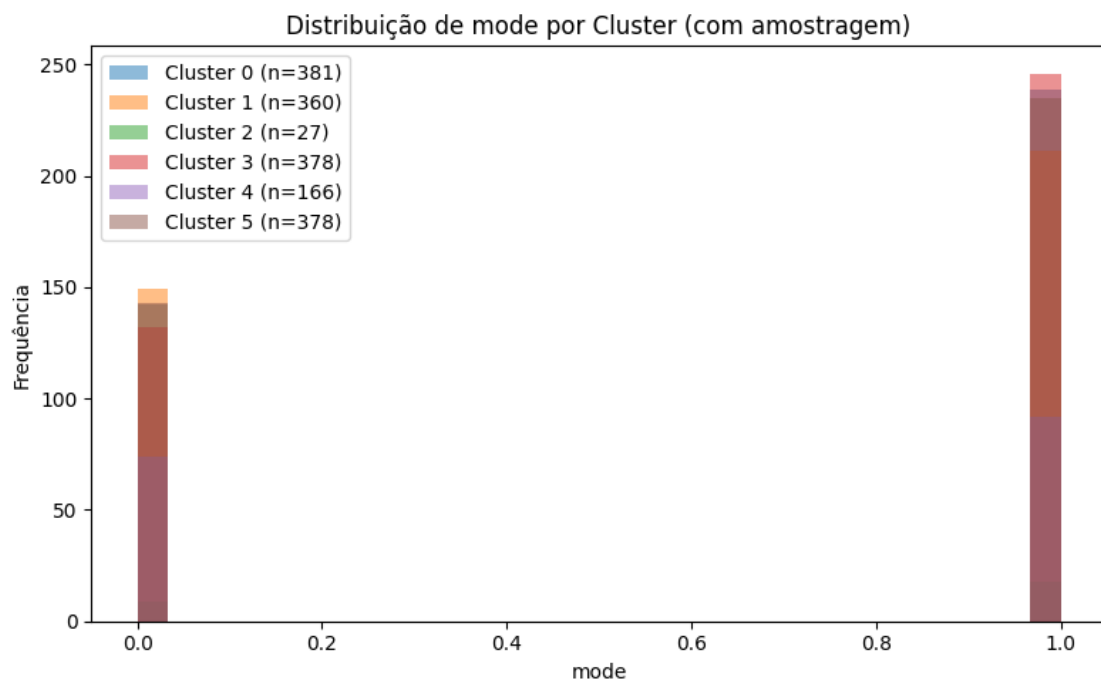
    plt.title(f'Distribuição de {feature} por Cluster (com amostragem)')
    plt.xlabel(feature)
    plt.ylabel('Frequência')
    plt.legend()
    plt.tight_layout()
```

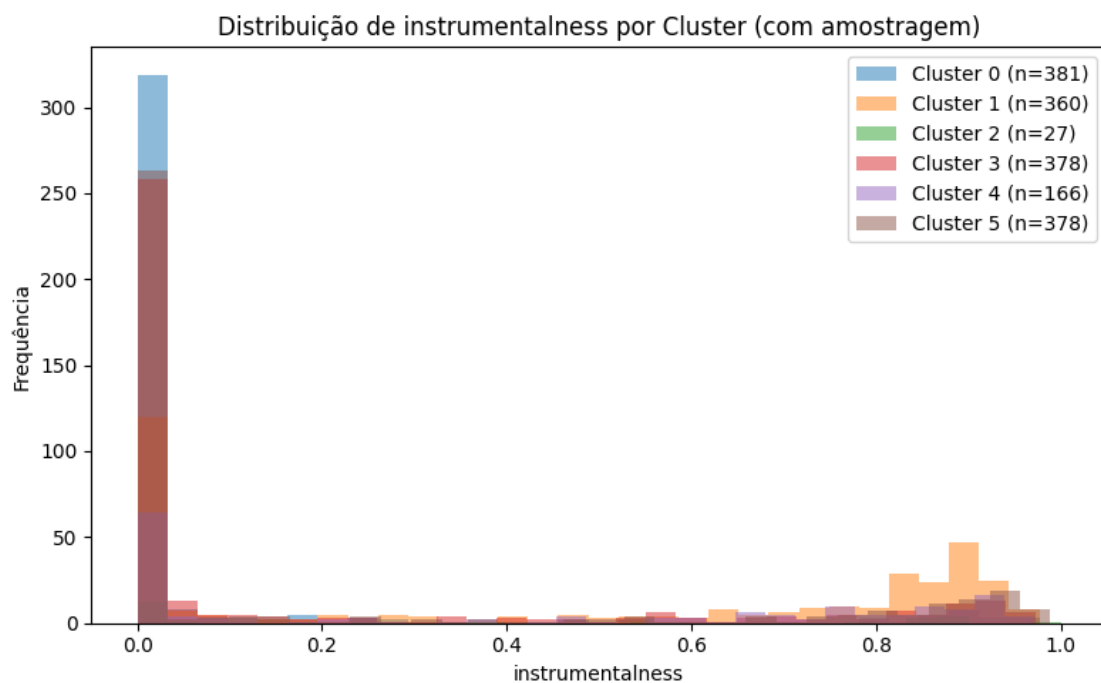
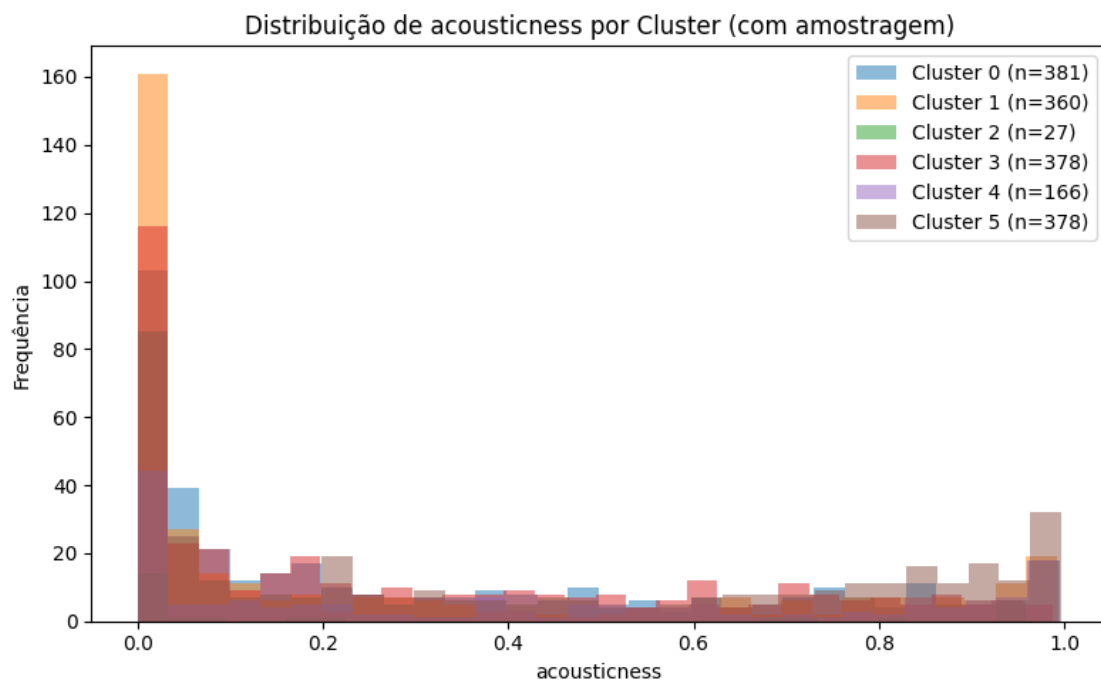
```
plt.show()
```

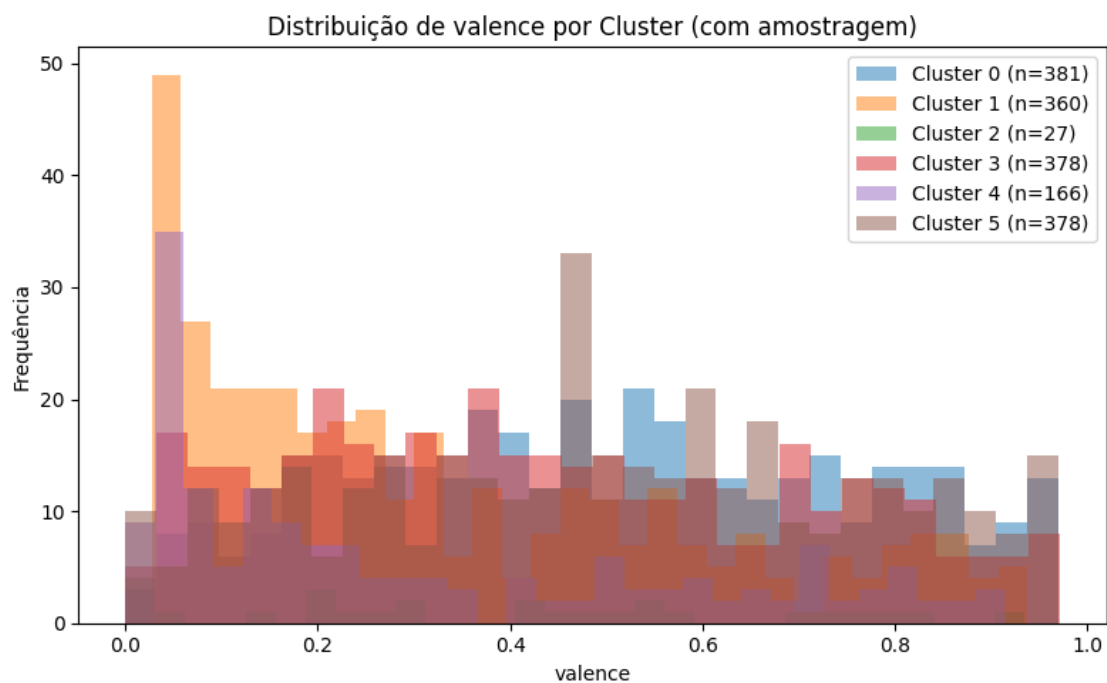
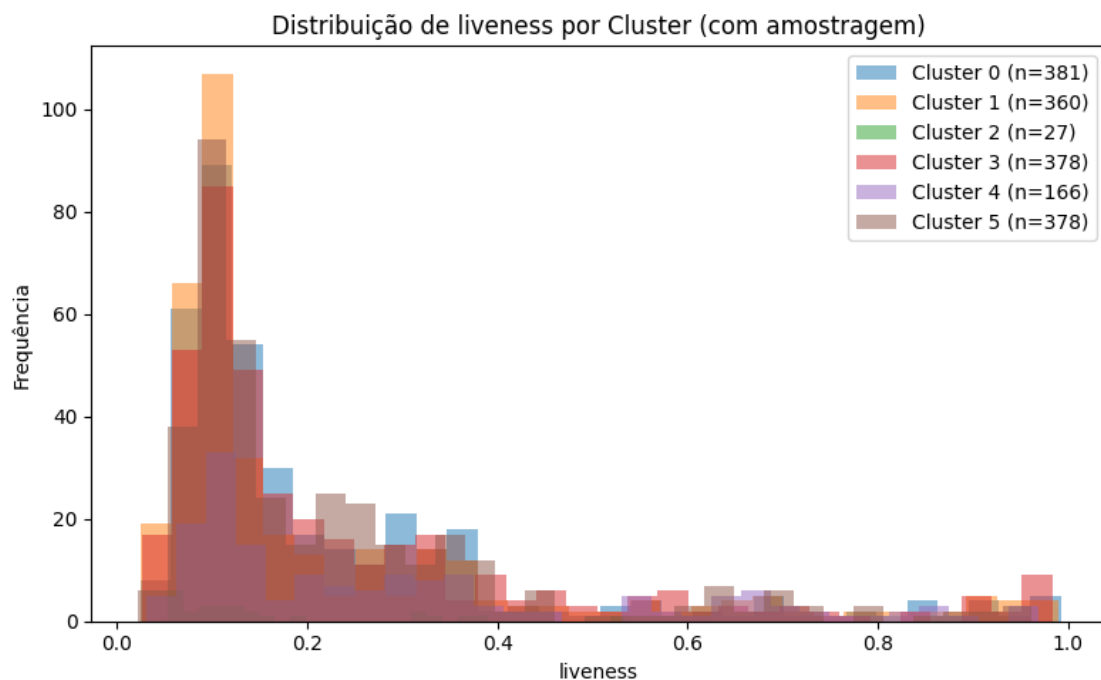


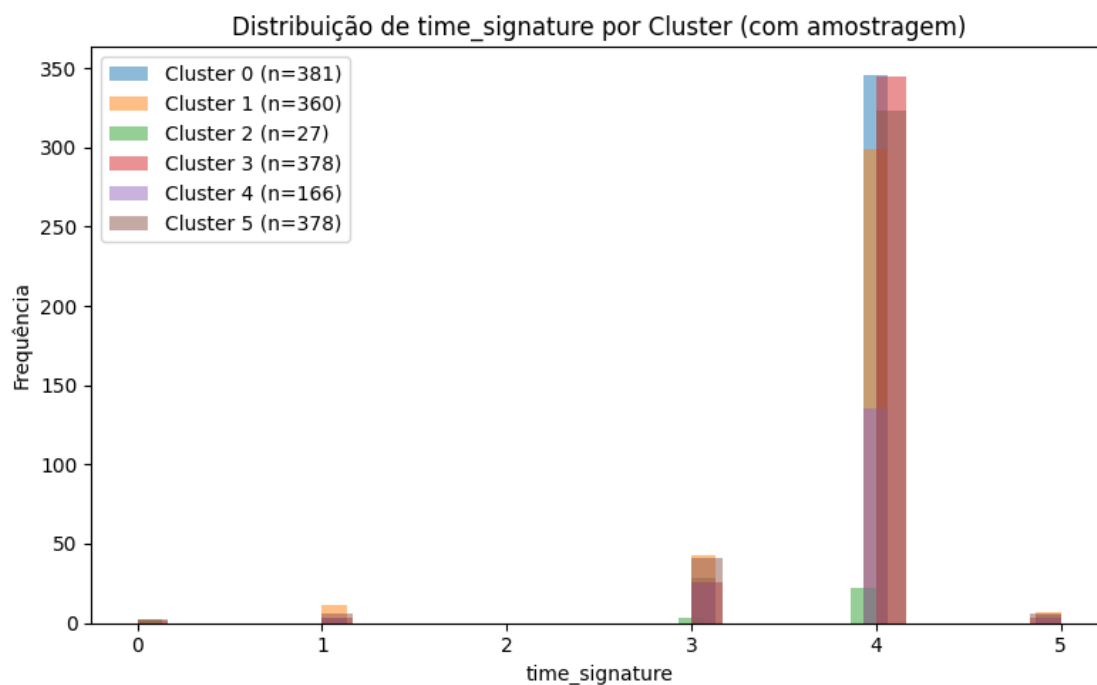
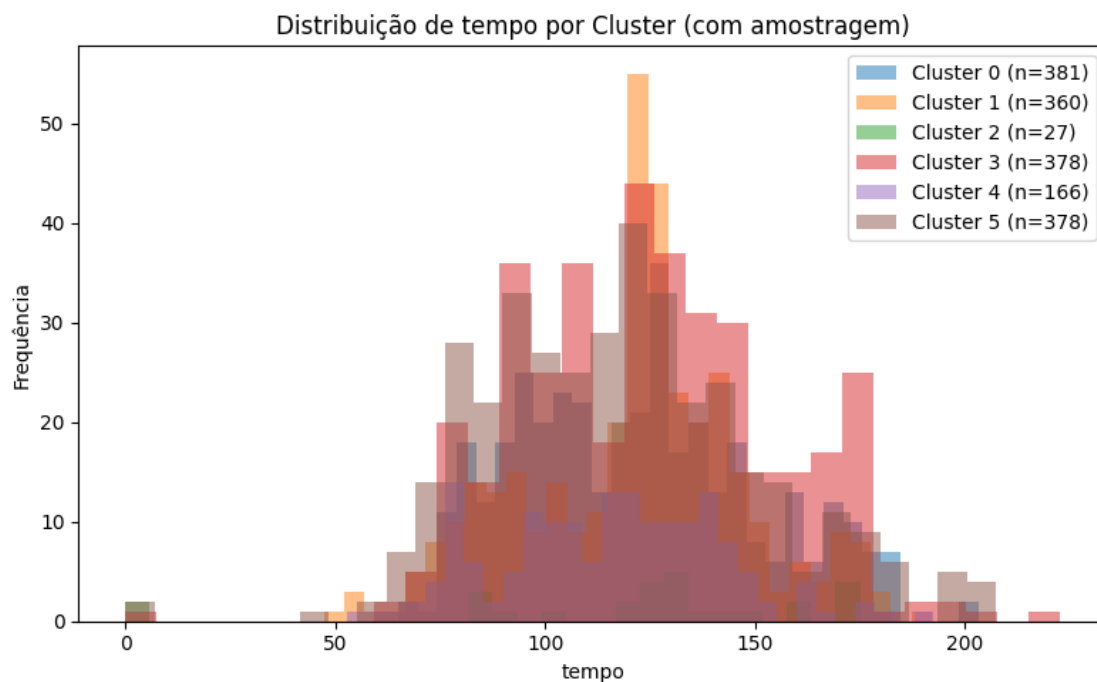












6.5 Música

```
[93]: from scipy.spatial.distance import euclidean, cosine

def recomendar_musicas(track_id, df_datos_normalizados, df_musicas):
    idx = df_musicas.index[df_musicas['track_id'] == track_id][0]
    cluster_id = df_musicas.loc[idx, 'cluster']
    print(f"Cluster: {cluster_id}")
    musica_base = df_datos_normalizados.drop(columns='cluster').iloc[idx].values

    subset = df_datos_normalizados[df_datos_normalizados['cluster'] ==
    ↪cluster_id].copy()
    subset['distancia'] = subset.drop(columns='cluster').apply(lambda row:
    ↪cosine(row.values, musica_base), axis=1)

    resultado_idx = subset.sort_values('distancia').index
    recomendacoes = df_musicas.loc[resultado_idx][['track_name',
    ↪'artists', 'track_genre']]
    recomendacoes = recomendacoes[recomendacoes.index != idx]

    return recomendacoes.head(10)
```

```
[94]: recomendar_musicas(track_id='2aibwv5hGXSgw7Yru8IYT0', df_datos_normalizados=df_datos, df_musicas=
```

Cluster: 3

```
[94]:
```

	track_name \		artists	track_genre
47014	Back In Black		AC/DC	hard-rock
68663	La Llevo Al Cielo (Ft. Ñengo Flow)		Chris Jedi;Anuel AA;Chencho Corleone;Ñengo Flow	latino
91018	Bad Liar		Imagine Dragons	rock
51032	Chidiya		Vilen	hip-hop
37950	I Got 5 On It		Luniz;Michael Marshall	funk
60	Pano		Zack Tabudlo	acoustic
91914	Wonderwall - Remastered		Oasis	rock
2259	Heart-Shaped Box		Nirvana	alt-rock
48125	Party Up		DMX	hardcore
68906	AM Remix			

7 Recomendação de Músicas

Vendo as os exemplos anteriores de recomendação de músicas, podemos perceber que o programa recomenda músicas com diversos gêneros para música (base) escolhida, mas será que é o que o usuário gostaria? Quais as chances do usuário escolher uma música, o programa recomendar 10 músicas com gêneros variados e o usuário se interessar em alguma música da recomendação e adicionar na sua playlist?

Pensando nisso, eu aprimorei a função `recomendar_musicas`, para levar em consideração o gênero da música e realizei um teste (pesquisa) com alguns amigos.

Utilizei as duas funções `recomendar_musicas` e `recomendar_musicas_por_genero`, para um determinada música do gosto deles, pedi para eles escutarem a música recomendada (top 1) de cada tabela, caso a primeira música já tenha aparecido anteriormente, passaríamos para a segunda, e assim por diante.

Com isso para cada par de tabela (ou seja, para cada método), eu pedi para que eles me informasse qual música teria mais chances deles adicionarem na playlist, assim descobriríamos se levar em conta o gênero o qual a música está associada para recomendação faz sentido ou não.

No fim teríamos 5 músicas (uma para cada método), então pedi para eles montarem um ranking das músicas que eles com certeza adicionariam na playlist para as músicas que eles talvez não adicionariam, então descobrimos qual método tem mais sucesso.

```
[106]: def recomendar_musicas_por_genero(track_id, df_dados_normalizados, df_musicas):  
    # Alinhar índices  
    df_dados_normalizados = df_dados_normalizados.reset_index(drop=True)  
    df_musicas = df_musicas.reset_index(drop=True)  
  
    idx = df_musicas.index[df_musicas['track_id'] == track_id][0]  
    cluster_id = df_musicas.loc[idx, 'cluster']  
    genero_base = df_musicas.loc[idx, 'track_genre']  
    print(f"Cluster: {cluster_id} | Gênero: {genero_base}")  
  
    musica_base = df_dados_normalizados.drop(columns='cluster').iloc[idx].values  
  
    # Filtra por cluster  
    subset = df_dados_normalizados[df_dados_normalizados['cluster'] ==  
    ↪ cluster_id].copy()  
  
    # Filtra por gênero  
    subset = subset[df_musicas.loc[subset.index, 'track_genre'] == genero_base]  
  
    # Calcula a distância  
    subset['distancia'] = subset.drop(columns='cluster').apply(  
        lambda row: cosine(row.values, musica_base), axis=1  
    )
```

```

resultado_idx = subset.sort_values('distancia').index
recomendacoes = df_musicas.iloc[resultado_idx][['track_name', 'artists', '
↳ 'track_genre']]
recomendacoes = recomendacoes[recomendacoes.index != idx]

return recomendacoes.head(10)

```

Busca de músicas por artistas

```

[107]: nome_artista= r"Red Hot Chili Peppers"
df_musicas[df_musicas['artists'] == nome_artista][['track_id', 'artists', '
↳ 'track_name', 'album_name']]

```

```

[107]:
      track_id      artists \
2013  6QvYmrqFKRSVrcVgvoA7Ar  Red Hot Chili Peppers
2014  75fT19n6ZQHR67Mkh86qfD  Red Hot Chili Peppers
2016  2AEdT2ZKd14kzOouzdnOUj  Red Hot Chili Peppers
2017  4xnB2yGAeXOuR3kJ9KTD3b  Red Hot Chili Peppers
2019  5xUhs6dlfkcxZM8zq7riYF  Red Hot Chili Peppers
2021  62pennq1pcVBFVW1b1CrFu  Red Hot Chili Peppers
2024  58yD1ZePlS6V4gImXGV8Ed  Red Hot Chili Peppers
2025  1tRpraJDpEU10HjnkMosqx  Red Hot Chili Peppers
2031  1c9MeAa2tLEoZl5osXf5Ts  Red Hot Chili Peppers
2034  0v8tmDMQrTMavluubqAJo4  Red Hot Chili Peppers
2035  47dUWYurHWefCpINwcvit2  Red Hot Chili Peppers
2042  5HNau9WX6OmX2PcBn3n24r  Red Hot Chili Peppers
2048  12cG3BUpk6PzLAzWK0mry0  Red Hot Chili Peppers
2105  48UPSzbZjgc449aqz8bxox  Red Hot Chili Peppers
2109  2aibwv5hGXSgw7Yru8IYT0  Red Hot Chili Peppers
2260  3Z0EytgrvLwQaqXreDs2Jx  Red Hot Chili Peppers
2309  3d9DChrdc6B0eFsbrZ3Is0  Red Hot Chili Peppers
2402  64BbK9SFKH2jk86U3dGj2P  Red Hot Chili Peppers
2404  1G391cbiT3v3Cygw8T7DM1  Red Hot Chili Peppers
2408  2oaK4JLVnmRGI09ytBE1bt  Red Hot Chili Peppers
2615  10Nmj3JCNoMeBQ87uw5j8k  Red Hot Chili Peppers
2716  5KnqauuLF6qyZA9RzfeKxG  Red Hot Chili Peppers
2812  7BCxIjRsqbc9Yiy2haBCIK  Red Hot Chili Peppers
2861  7cpC2EuFMexgy392K13RZb  Red Hot Chili Peppers
2863  0G1BEi8ByngMCJqcPnDACw  Red Hot Chili Peppers
2864  2dbA4sPwu3iK72c0mA6907  Red Hot Chili Peppers
2865  5kFWp8In0sIpeXPfPuJlYE  Red Hot Chili Peppers
2867  0g8gKLBfY1Y16Zyrc9fuKA  Red Hot Chili Peppers
2870  00B7SBwrjbycLM0gAmeIU8  Red Hot Chili Peppers
2911  5axys0a071B1dZdfnrxHvy  Red Hot Chili Peppers
2912  4cvngM7l4d0wldWzMOEP6c  Red Hot Chili Peppers
2915  4diMr8l7HCK10drQM3LAzq  Red Hot Chili Peppers

```

2918	1d6zb2JQZt9LpxOwU1E3y4	Red Hot Chili Peppers
2919	1TmBaKkDUu0akM9xzSxRia	Red Hot Chili Peppers
2920	2AHctrKhJekyp7wIykP83U	Red Hot Chili Peppers
2957	1f2V8U1BiWaC9aJWmpOARe	Red Hot Chili Peppers
2968	141GeeQQItnC1sH7gXUDlf	Red Hot Chili Peppers
37653	3W2reBya7FkjHCP1B2nmQr	Red Hot Chili Peppers
37759	05bNuygSyh9WlJTRepNiDp	Red Hot Chili Peppers
37851	5GWrhVfgP4szBtCOXbSki8	Red Hot Chili Peppers
37853	3a94TbZ0xhkI9xuNwYL53b	Red Hot Chili Peppers
71781	0uppYCG86ajpV2hSR3dJJ0	Red Hot Chili Peppers
71867	3s53zBEi1ZU6dJj19eF0xW	Red Hot Chili Peppers
71871	1ndGB6rvxKYN9seCY01dTF	Red Hot Chili Peppers
71923	7Gpr3kKk4BMgItz6UbI73q	Red Hot Chili Peppers
91095	5Cgxae0bSnra1V8laWLcCq	Red Hot Chili Peppers
91096	3FkFxMaunvWoOXr7oEM9Qj	Red Hot Chili Peppers
91098	2p3Wbdb4HdfS1jEJ1s5CoR	Red Hot Chili Peppers
91133	6fQ0IKOV2aVrPbpH4VFQsS	Red Hot Chili Peppers

	track_name	album_name
2013	Tippa My Tongue	All That Really Matters - Just Hits
2014	Tippa My Tongue	Rock Brandneu
2016	Californication	OK Boomers - Top Tunes
2017	Nerve Flip	Rock Brandneu
2019	Black Summer	Tek It - New Noise
2021	Nerve Flip	Top of the Rock
2024	Tippa My Tongue	Daily Pop Mix
2025	Black Summer	Top of the Rock
2031	Fire - Remastered	Kick It
2034	Higher Ground - Remastered	80s Bangers Vol. 2
2035	Black Summer	Pop Hits Now
2042	Black Summer	20's Rock
2048	By the Way	Alternative Rock Mixtape
2105	Californication	Californication (Deluxe Edition)
2109	Snow (Hey Oh)	Stadium Arcadium
2260	Can't Stop	By the Way (Deluxe Edition)
2309	Under the Bridge	Blood Sugar Sex Magik (Deluxe Edition)
2402	Otherside	Californication (Deluxe Edition)
2404	Scar Tissue	Californication (Deluxe Edition)
2408	Dark Necessities	The Getaway
2615	Dani California	Stadium Arcadium
2716	The Drummer	Return of the Dream Canteen
2812	Tippa My Tongue	Return of the Dream Canteen
2861	Fake as Fu@k	Return of the Dream Canteen
2863	Eddie	Return of the Dream Canteen
2864	Bella	Return of the Dream Canteen
2865	Roulette	Return of the Dream Canteen
2867	In the Snow	Return of the Dream Canteen

2870	Reach Out	Return of the Dream Canteen
2911	Carry Me Home	Return of the Dream Canteen
2912	La La La La La La La La	Return of the Dream Canteen
2915	Afterlife	Return of the Dream Canteen
2918	Handful	Return of the Dream Canteen
2919	Bag of Grins	Return of the Dream Canteen
2920	Shoot Me a Smile	Return of the Dream Canteen
2957	By the Way	By the Way (Deluxe Edition)
2968	Copperbelly	Return of the Dream Canteen
37653	Peace and Love	Return of the Dream Canteen
37759	My Cigarette	Return of the Dream Canteen
37851	Eddie	Eddie
37853	Black Summer	Black Summer
71781	Give It Away	Blood Sugar Sex Magik (Deluxe Edition)
71867	Tippa My Tongue	Tippa My Tongue
71871	The Zephyr Song	By the Way (Deluxe Edition)
71923	Road Trippin'	Californication
91095	My Friends	Session Acoustique
91096	Tippa My Tongue	Pop n' Fresh
91098	Eddie	This Is Why - This Is Rock
91133	Tippa My Tongue	This Is Why - This Is Rock

```
[108]: id_musica = "2aibwv5hGXsgw7Yru8IYT0"
```

7.1 Robusts Scale

```
[109]: recomendar_musicas_por_genero(id_musica,df_dados_normalizados1,df_musicas_rs)
```

Cluster: 3 | Gênero: alt-rock

```
[109]:
```

	track_name	artists	track_genre
2926	Man in the Box	Alice In Chains	alt-rock
2619	This Is Amazing Grace	Phil Wickham	alt-rock
2258	Can't Stop	Red Hot Chili Peppers	alt-rock
2874	La Ensalada	Tan Bionica	alt-rock
2414	Our God	Chris Tomlin	alt-rock
2719	Animal I Have Become	Three Days Grace	alt-rock
2104	Californication	Red Hot Chili Peppers	alt-rock
2557	Black Hole Sun	Soundgarden	alt-rock
2616	Be Yourself	Audioslave	alt-rock
2352	Happiness	The 1975	alt-rock

7.2 Min Max Scale

```
[110]: recomendar_musicas_por_genero(id_musica,df_dados_normalizados2,df_musicas_minmax)
```

Cluster: 4 | Gênero: alt-rock

```
[110]:
```

	track_name	artists	track_genre
2548	My Type	Saint Motel	alt-rock
2948	My Type	Saint Motel	alt-rock
2709	Human	The Killers	alt-rock
2719	Animal I Have Become	Three Days Grace	alt-rock
2450		BLUE ENCOUNTER	alt-rock
2002	Sweater Weather	The Neighbourhood	alt-rock
2403	How You Remind Me	Nickelback	alt-rock
2926	Man in the Box	Alice In Chains	alt-rock
2311	All Star	Smash Mouth	alt-rock
2651	Chocolate	The 1975	alt-rock

7.3 Standart Scale

```
[111]: recomendar_musicas_por_genero(id_musica,df_dados_normalizados3,df_musicas_stand)
```

Cluster: 5 | Gênero: alt-rock

```
[111]:
```

	track_name	artists	track_genre
2926	Man in the Box	Alice In Chains	alt-rock
2719	Animal I Have Become	Three Days Grace	alt-rock
2548	My Type	Saint Motel	alt-rock
2450		BLUE ENCOUNTER	alt-rock
2948	My Type	Saint Motel	alt-rock
2619	This Is Amazing Grace	Phil Wickham	alt-rock
2258	Can't Stop	Red Hot Chili Peppers	alt-rock
2002	Sweater Weather	The Neighbourhood	alt-rock
2871	Tornado Of Souls - Remastered 2004	Megadeth	alt-rock
2352	Happiness	The 1975	alt-rock

7.4 Max Abs Scale

```
[112]: recomendar_musicas_por_genero(id_musica,df_dados_normalizados4,df_musicas_maxabs)
```

Cluster: 1 | Gênero: alt-rock

```
[112]:
```

	track_name	artists	track_genre
2548	My Type	Saint Motel	alt-rock
2948	My Type	Saint Motel	alt-rock
2709	Human	The Killers	alt-rock
2719	Animal I Have Become	Three Days Grace	alt-rock
2450		BLUE ENCOUNTER	alt-rock
2311	All Star	Smash Mouth	alt-rock
2002	Sweater Weather	The Neighbourhood	alt-rock
2403	How You Remind Me	Nickelback	alt-rock
2651	Chocolate	The 1975	alt-rock
2352	Happiness	The 1975	alt-rock

7.5 Sem normalizar os dados

```
[113]: recomendar_musicas(id_musica, df_dados, df_musicas_)
```

Cluster: 3

```
[113]:
```

	track_name \	
47014	Back In Black	
68663	La Llevo Al Cielo (Ft. Ñengo Flow)	
91018	Bad Liar	
51032	Chidiya	
37950	I Got 5 On It	
60	Pano	
91914	Wonderwall - Remastered	
2259	Heart-Shaped Box	
48125	Party Up	
68906	AM Remix	

	artists	track_genre
47014	AC/DC	hard-rock
68663	Chris Jedi;Anuel AA;Chencho Corleone;Ñengo Flow	latino
91018	Imagine Dragons	rock
51032	Vilen	hip-hop
37950	Luniz;Michael Marshall	funk
60	Zack Tabudlo	acoustic
91914	Oasis	rock
2259	Nirvana	alt-rock
48125	DMX	hardcore
68906	Nio Garcia;J Balvin;Bad Bunny	latino

```
[114]: recomendar_musicas_por_genero(id_musica,df_dados_normalizados=df_dados,df_musicas=df_musicas_)
```

Cluster: 3 | Gênero: alt-rock

```
[114]:
```

	track_name	artists	track_genre
2901	Sweet Dreams (Are Made Of This)	Marilyn Manson	alt-rock
2406	Dark Necessities	Red Hot Chili Peppers	alt-rock
2104	Californication	Red Hot Chili Peppers	alt-rock
2557	Black Hole Sun	Soundgarden	alt-rock
2871	Tornado Of Souls - Remastered 2004	Megadeth	alt-rock
2555	Cult of Personality	Living Colour	alt-rock
2364	Killing In The Name	Rage Against The Machine	alt-rock
2810	Tippa My Tongue	Red Hot Chili Peppers	alt-rock
2299	About You	The 1975	alt-rock
2619	This Is Amazing Grace	Phil Wickham	alt-rock

8 Conclusão

Dependendo o método que utilizarmos para normalizar os dados, ele irá sim, tanto no agrupamento quanto na recomendação das músicas. Como podemos ver anteriormente na recomendação das músicas, a pesar de algumas músicas aparecerem em todas as listagens, elas aparecem em posições diferentes, com exceção de Min Max Scale e Max Abs Scale onde o top 5 é idêntico em ambas. Também temos que levar em consideração que algumas músicas pertencem a mais de um gênero, e isso acaba impactando diretamente na forma com que as músicas são recomendadas por gênero. Portanto, uma possível solução é tentar dividir as músicas em grupos maiores como por exemplo, Heavy Metal e Power Metal, ambos serem do gênero ‘Metal’.

Após testar as recomendações, o método que obteve maior sucesso foi Min Max Scale, seguido de MaxAbs e Standard Scale, ambos utilizando o sistema de recomendação por gênero, mas em alguns casos principalmente em músicas do gênero industrial, heavy metal, metal, etc. o sistema de recomendação sem levar em conta o gênero acabou se saindo surpreendentemente melhor nas recomendações, enquanto para músicas de outros gênero a recomendação levando em conta o gênero da música se saiu melhor.

9 Referências e Links

<https://www.kaggle.com/datasets/maharshipandya/-spotify-tracks-dataset/data>

<https://medium.com/pizzadedados/kmeans-e-metodo-do-cotovelo-94ded9fdf3a9>

<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html>

<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.maxabs_scale.html

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.minmax_scale.html

<https://scikit-learn.org/stable/index.html>