Data Analysis Project

Theodoro Gasperin Terra Camargo
R0974221
I0P16A
Prof. Jan Aerts
December 11, 2024

1. Research Question

Understanding the diverse behaviors and preferences of game reviewers is crucial for tailoring game development, marketing strategies, and community engagement. By clustering user review data, and review content, alongside game interaction metrics, this analysis aims to uncover distinct reviewer personas. These personas may highlight how different groups of users engage with games and contribute to the review ecosystem, offering valuable insights into their motivations, interests, and levels of activity.

2. Data selection and preprocessing

The dataset used in this analysis was sourced from GitHub (https://github.com/mulhod/steam_reviews) and consists of numerous user reviews for popular games on the Steam platform. Given the large size of the dataset, this analysis focuses on a specific game: Grand Theft Auto 5 (GTA5). The dataset includes various valuable columns, such as information about the reviews themselves and detailed data about the reviewers. Reviewer-specific information includes metrics such as the total number of hours spent playing the game, the number of workshop items bought, and achievements unlocked. Whereas, review specific columns provide information about the number of people that found the review helpful/unhelpful/funny etc.

To reveal distinct reviewer personas, the dataset was sectioned to focus on a selection of valuable columns that offer both quantitative and qualitative insights. These columns include: 'total_game_hours', 'rating', 'num_reviews', 'num_games_owned', 'num_friends', 'num_found_funny', 'num_found_helpful', 'num_found_unhelpful', 'num_comments'. Together, these variables offer a multidimensional perspective on user activity, engagement, and influence within the gaming platform.

To ensure the data was suitable for analysis, a preprocessing step was undertaken. This involved scaling the variables to normalize their ranges, thereby preventing features with larger numeric values from disproportionately influencing the results. Additionally, missing values were identified and removed to maintain the integrity of the dataset and avoid biases in the analysis. These preprocessing steps were crucial for preparing the data for clustering, enabling a more accurate identification of patterns and relationships among users.

3. Multivariate analysis strategy

Several methods were used to analyze the dataset, focusing on clustering algorithms, dimensionality reduction techniques, and parameter exploration to ensure robust insights. The clustering algorithms applied included K-means, hierarchical clustering, DBSCAN, and Spectral Clustering. K-means was selected as an initial approach due to its simplicity and efficiency in identifying spherical clusters, with the optimal number of clusters (k=4) determined using the elbow method. Hierarchical clustering was explored to assess its ability to reveal nested data structures and was tested with two linkage methods (Ward, Average) and distance metrics (Euclidean, Cosine, Manhattan, Correlation, Canberra). DBSCAN was chosen for its ability to identify non-spherical clusters and handle noise effectively, while Spectral Clustering was included to capture potential nonlinear relationships in the data. Among these, DBSCAN demonstrated the most promising clustering results.

Dimensionality reduction techniques such as PCA, t-SNE, Isomap, and UMAP were employed to visualize the clustering outcomes in two and three dimensional spaces. These methods were selected for their complementary strengths in preserving different aspects of the data's structure. PCA was used for its efficiency in capturing linear variance, while t-SNE and UMAP were applied for their effectiveness in representing non-linear structures and maintaining local data relationships. Isomap was included to evaluate its capability in preserving geodesic distances in the reduced dimensions.

The methods were evaluated based on stress values, which measure how well the reduced-dimensional representations preserve the original data's pairwise distances. The stress values were PCA (0.42), t-SNE (18.29), UMAP (2.15), and Isomap (1.98). Despite its higher stress value, t-SNE provided the clearest visual separation of clusters, making it the preferred choice for visualization.

To refine the analysis, different parameter settings were explored for each method. For K-means, the value of k was varied to identify the optimal number of clusters. In hierarchical clustering, different linkage methods and distance metrics were tested to understand their impact on clustering structure. For DBSCAN, parameters such as the minimum number of samples and the epsilon distance threshold were adjusted to optimize cluster identification while minimizing noise. Similarly, dimensionality reduction techniques were tuned by varying parameters like perplexity in t-SNE and the number of neighbors in UMAP. This parameter exploration ensured that the chosen methods were well-suited for the dataset and its underlying characteristics.

4. Analysis and results

The HDBSCAN cluster analysis identified four major clusters (9, 10, 12, and 13) and several smaller clusters (0 to 8, and 11), as shown in Figure 1. The major clusters represented the majority of the dataset's samples. Among them, clusters 12 and 13 predominantly consist of users who recommended the game, while clusters 9 and 10 comprise those who did not recommend it.

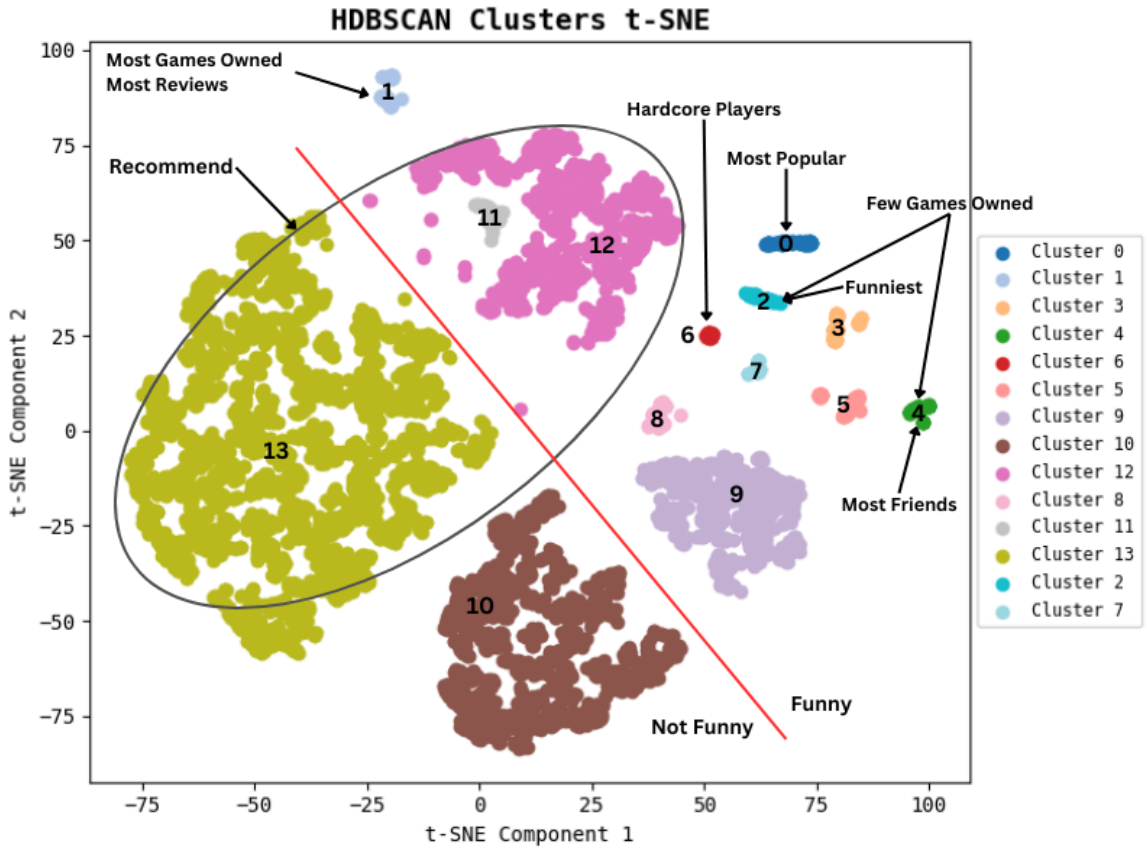


Figure 1: t-SNE HDBSCAN Clusters plot with annotations.

In terms of user behavior, clusters 12 and 13 exhibited a higher average number of friends and total game hours compared to clusters 9 and 10 (Figure 2). Within these groups, further subdivision emerged based on the humor perceived in their reviews: clusters 9 and 12 were associated with reviews some users found funny, while clusters 10 and 13 contained reviews considered not funny (red line in the Figure 1). Interestingly, clusters 9 and 10, despite their non-recommendations, showed average behavior across other metrics, indicating typical player patterns without strong community influence. These four major clusters reflect the archetypal player personas on the platform, providing a useful foundation for understanding broader user behavior trends.

On the other hand, the smaller clusters revealed several intriguing patterns, each showcasing unique user behaviors. Cluster 0 emerged as the most popular, with its reviews receiving the highest number of both "helpful" and "unhelpful" votes, suggesting that the reviews from this group sparked significant engagement, whether positive or negative. Despite its popularity, this cluster exhibited the shortest total game hours, and all individuals in this group unanimously chose not to recommend the game, indicating dissatisfaction despite their influential reviews. Additionally, it was the second funniest cluster, hinting that users in this group might rely on humor to engage their audience despite their lack of playtime.

In contrast, Cluster 1 comprised users with the highest number of owned games and reviews written, coupled with the second-lowest total game hours and friend count, and similarly, no recommendations for the game were observed. This unusual combination of very high number of reviews and game ownership, paired with low hours in gameplay and few social connections, strongly suggests that Cluster 1 may primarily consist of bots or automated accounts rather than genuine users. The identification of such unusual patterns underscores the value of clustering analysis in uncovering anomalies and distinguishing between genuine users and potentially automated accounts. In addition, it raises the question whether companies or individuals are trying to negatively impact GTA5 reviews on the Steam platform.

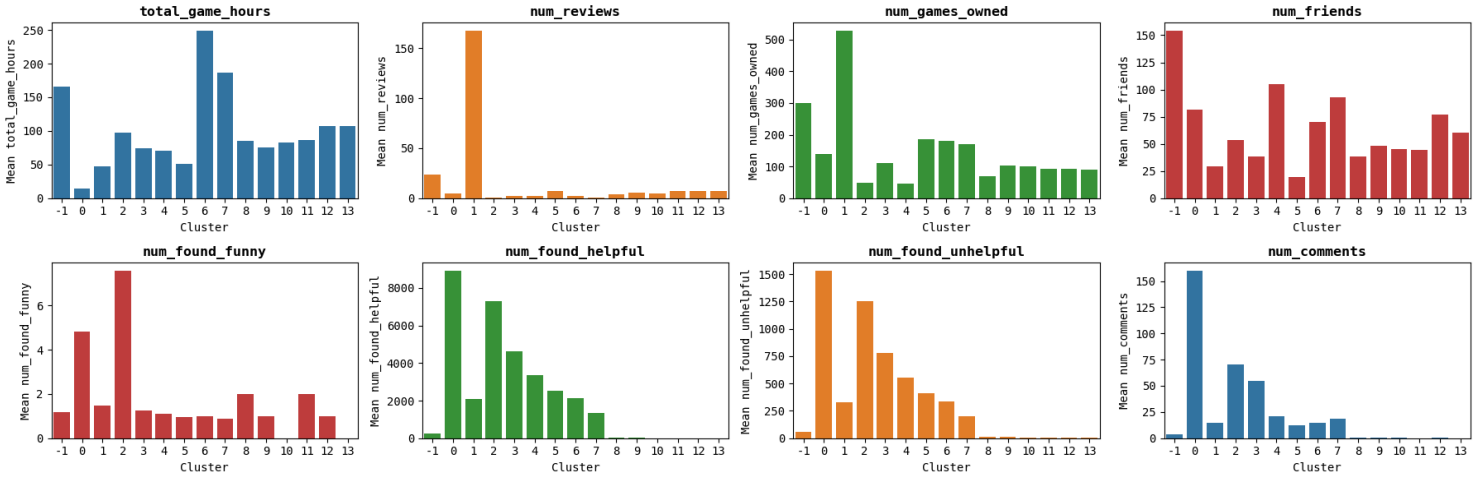


Figure 2: Review and User Variables Clusters' Means.

Cluster 6 exhibited the highest total game hours, closely followed by Cluster 7, indicating heavy gameplay among its members (Figure 2, top left). This level of gameplay suggests that these clusters are composed of dedicated gamers who spend a significant amount of time immersed in the game. In addition, both these clusters showed elevated numbers of friends on the platform, further reinforcing the idea that these users are socially active and highly engaged, forming connections within the gaming community and likely participating in multiplayer or collaborative activities.

Cluster 4 stood out as the most social group, boasting the largest number of friends on the Steam platform and the lowest number of games owned (Figure 2). This combination suggests that these players are community-oriented individuals who prioritize interactions with other players over accumulating or deeply engaging with games themselves. These users may derive their enjoyment primarily from the social aspects of the platform rather than the games they play, potentially acting as key influencers in building and maintaining player networks.

Meanwhile, Cluster 2 distinguished itself as the funniest cluster, receiving the highest number of "funny" votes, and it was the second most popular, trailing only behind Cluster 0 in terms of engagement (Figure 2). This suggests that users in this cluster likely have a talent for humor, using entertaining and relatable reviews to connect with the broader community. Their ability to elicit engagement through humor could indicate a deeper understanding of the audience and a preference for sharing creative and funny content. Additionally, the active participation of this cluster suggests they may play an essential role in shaping and influencing users' opinions on the Steam platform by means of their reviews.

5. Interpretation and insights

The analysis uncovered distinct reviewer personas on the Steam platform. Four major clusters represented the dominant player types, with clusters 12 and 13 comprising socially engaged users who recommended the game, and clusters 9 and 10 reflecting average players who did not. Smaller clusters revealed unique behaviors: Cluster 0 combined humor and popularity despite low playtime and non-recommendations, while Cluster 1 suggested bot-like activity. Other clusters, such as 6, 7, and 4, highlighted socially active gamers and community-focused users, providing valuable insights into varying engagement styles.

These findings have practical implications for game developers and marketers, enabling targeted strategies for engagement and addressing dissatisfaction in key groups. However, limitations exist, including the focus on a single game, potential dataset biases, and methodological constraints associated with the clustering algorithm used. Specifically, HDBSCAN's tendency to hide outliers may have influenced our analysis by isolating data points that could potentially belong to meaningful subgroups. This approach, while effective for identifying noise, could result in the exclusion of valuable patterns within these outliers. Expanding the analysis to include more variables, multiple games, and alternative clustering techniques could enhance the robustness of the findings and provide a broader understanding of reviewer behaviors on Steam.

6. Reflection

This analysis enhanced my understanding of various clustering algorithms and demonstrated the effectiveness of unsupervised learning in extracting valuable real-world insights.

Generative AI Acknowledgement

I would like to acknowledge the use of ChatGPT, developed by OpenAI, for assisting with the analysis and writing of this report.

Appendix

All the code used in this report can be found in:
https://github.com/tgaspe/Clustering_Analysis

The data used in this analysis can be found in:
https://github.com/mulhod/steam_reviews

Table 1: Clusters and number of recommendations. Cluster -1 represents noise classified by the HDBSCAN clustering algorithm.

HDBSCAN_Cluster	Recommended	Not_Recommended
-1	1087	659
0	0	70
1	0	62
2	0	45
3	0	72
4	0	51
5	0	72
6	0	29
7	0	33
8	0	39
9	0	638
10	0	1340
11	31	0
12	1072	0
13	3089	0

Multivarite Data Analysis

- **Student:** Theodoro Gasperin Terra Camargo
- **Number:** r0974221

Dataset can be found at: https://github.com/mulhod/steam_reviews

Reserch Questions:

Can I uncover distinct reviewer personas by means of clustering.

Importing Dependencies

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import sqlite3
```

Data Preprocessing:

Converting our data into a pandas df

In [2]:
```
df = pd.read_json('Grand_Theft_Auto_V.jsonlines', lines=True)
```

In [3]:
```
df.head(1)
```

Out[3]:

	num_found_funny	review_url	num_guides	tot
0	5	http://steamcommunity.com/profiles/76561198010...	0	

1 rows × 27 columns

In [4]:
```
df.shape
```

Out[4]:
```
(13349, 27)
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['num_found_funny', 'review_url', 'num_guides', 'total_game_hours',
       'num_workshop_items', 'num_found_unhelpful', 'steam_id_number',
       'username', 'num_found_helpful', 'total_game_hours_last_two_weeks',
       'date_posted', 'num_comments', 'profile_url', 'rating', 'num_review
s',
       'orig_url', 'num_groups', 'num_games_owned', 'friend_player_level',
       'date_updated', 'found_helpful_percentage', 'num_screenshots', 'revi
ew',
       'achievement_progress', 'num_voted_helpfulness', 'num_badges',
       'num_friends'],
      dtype='object')
```

In [6]:
```
columns_drop = ['review_url', 'steam_id_number', 'username', 'date_posted',
```

In [7]:
```
# Drop columns
df.drop(columns=columns_drop, inplace=True)
```

In [8]:
```
df.shape
```

Out[8]:
```
(13349, 20)
```

Changes rating to numerical categories

In [9]:
```
print(df['rating'].unique())
df['rating'] = df['rating'].apply(lambda x: 1 if x == 'Recommended' else 0)
print(df['rating'].unique())
```

```
['Not Recommended' 'Recommended']
[0 1]
```

Extracting achievenments

In [10]:
```
# Normalize the 'achievement_progress' column
achievement_df = pd.json_normalize(df['achievement_progress'])

# Concatenate the new dataframe with the original dataframe
df = pd.concat([df, achievement_df], axis=1)

# Drop the original 'achievement_progress' column
df.drop(columns=['achievement_progress', 'num_achievements_possible', 'num_a
#df.drop(columns=['achievement_progress'], inplace=True)
```

In [11]:
```
# Display the columns types
print(df.dtypes)
```

```
num_found_funny                    int64
num_guides                         int64
total_game_hours                 float64
num_workshop_items                 int64
num_found_unhelpful                int64
num_found_helpful                  int64
total_game_hours_last_two_weeks  float64
num_comments                       int64
rating                             int64
num_reviews                        int64
num_groups                       float64
num_games_owned                    int64
friend_player_level              float64
found_helpful_percentage         float64
num_screenshots                    int64
review                            object
num_voted_helpfulness              int64
num_badges                       float64
num_friends                      float64
num_achievements_percentage      float64
dtype: object
```

In [12]:
```
df.shape
```

Out[12]: (13349, 20)

Removing missing values

In [13]:
```
missing_values = df.isnull().sum()
print(missing_values)
```

```
num_found_funny                       0
num_guides                            0
total_game_hours                      0
num_workshop_items                    0
num_found_unhelpful                   0
num_found_helpful                     0
total_game_hours_last_two_weeks       0
num_comments                          0
rating                                0
num_reviews                           0
num_groups                         3267
num_games_owned                       0
friend_player_level                2003
found_helpful_percentage           2119
num_screenshots                       0
review                                0
num_voted_helpfulness                 0
num_badges                         2003
num_friends                        2124
num_achievements_percentage        2160
dtype: int64
```

In [14]:
```
df.dropna(inplace=True)
df.shape
```

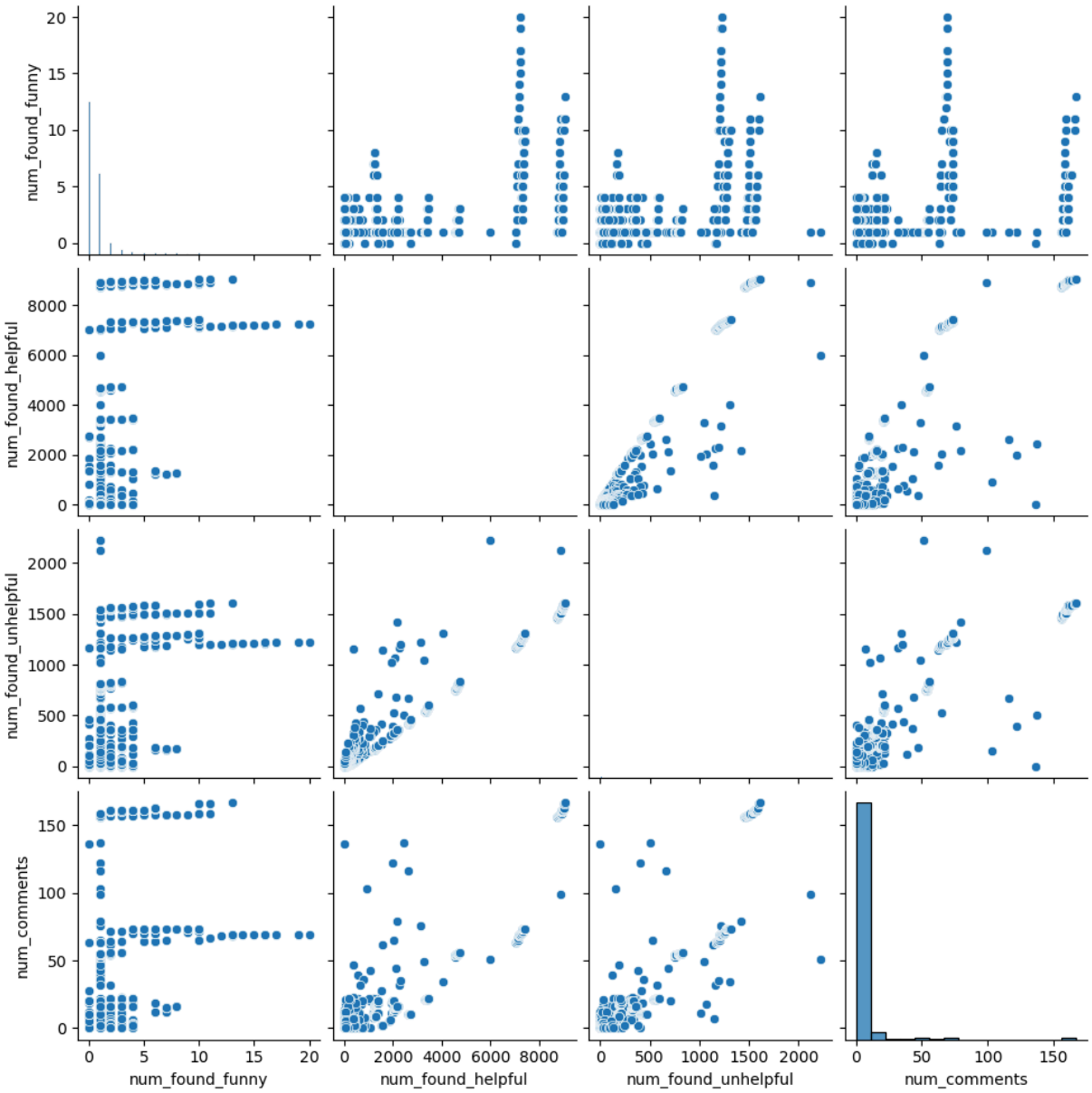
Out[14]: (8389, 20)

Multivarite Analysis

Review data pairplot

In [15]:
```
# Review data pairplot
columns = [
        'num_found_funny', 'num_found_helpful', 'num_found_unhelpful',
        'num_comments'
    ]
df_query = df[columns]
sns.pairplot(df_query)
```

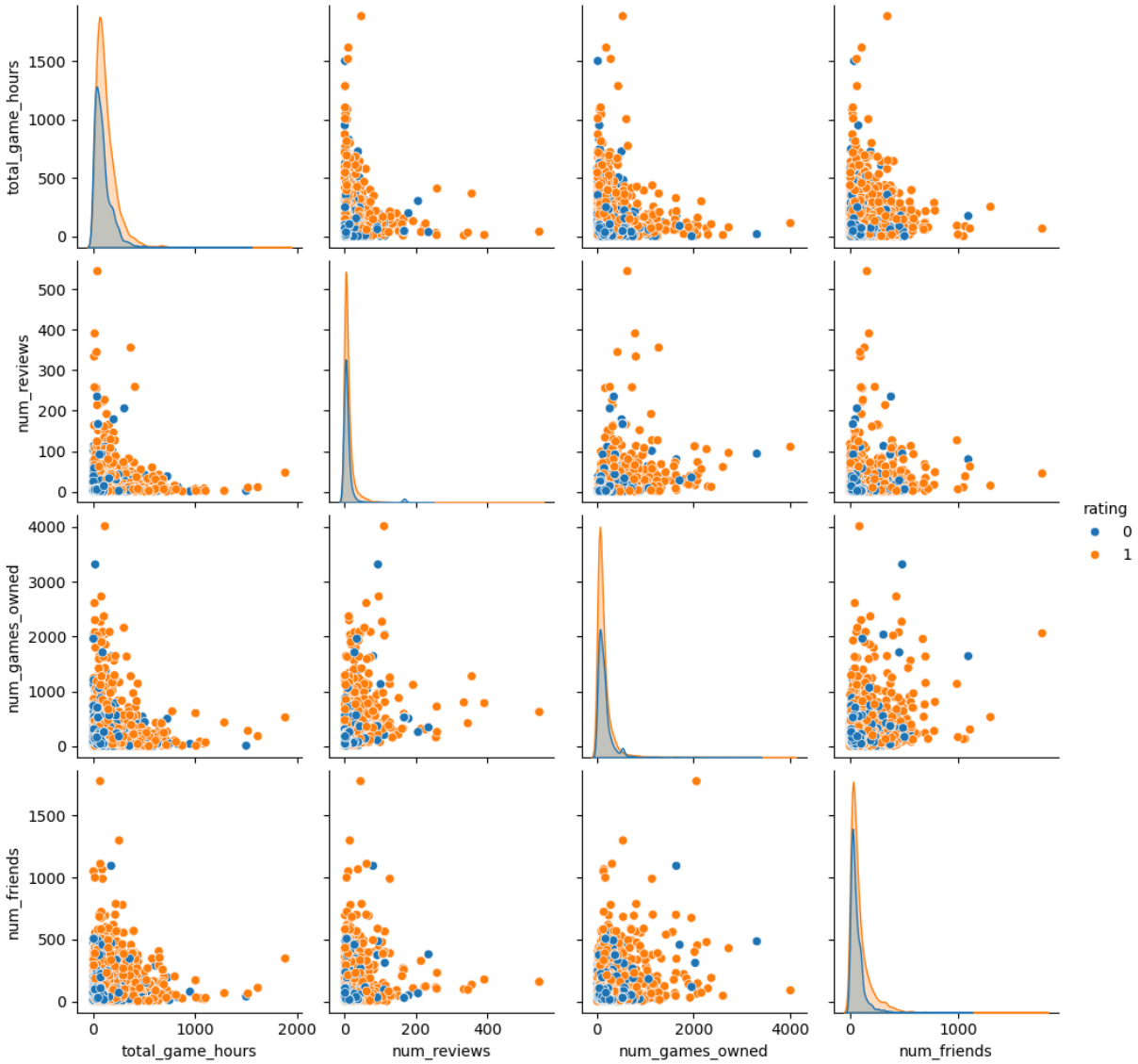
Out[15]: <seaborn.axisgrid.PairGrid at 0x1692eceb0>



Reviewer data pairplot

In [20]:
```
# Reviewer data pairplot
columns = [
        'total_game_hours', 'rating', 'num_reviews',
        'num_games_owned', 'num_friends'
    ]
df_query = df[columns]

sns.pairplot(df_query, hue='rating')
```

Out[20]: <seaborn.axisgrid.PairGrid at 0x173fbbc40>



In [18]:
```
df.columns
```

Out[18]:
```
Index(['num_found_funny', 'num_guides', 'total_game_hours',
       'num_workshop_items', 'num_found_unhelpful', 'num_found_helpful',
       'total_game_hours_last_two_weeks', 'num_comments', 'rating',
       'num_reviews', 'num_groups', 'num_games_owned', 'friend_player_leve
l',
       'found_helpful_percentage', 'num_screenshots', 'review',
       'num_voted_helpfulness', 'num_badges', 'num_friends',
       'num_achievements_percentage'],
      dtype='object')
```

In [16]:
```
# Select relevant features for clustering
features = ['total_game_hours', 'rating', 'num_reviews',
       'num_games_owned', 'num_friends', 'num_found_funny', 'num_found_help
       'num_comments']
```

In [20]:
```
# Select relevant features for clustering
# features = ['num_found_funny', 'total_game_hours', 'num_friends', 'num_com
#             'found_helpful_percentage']
```

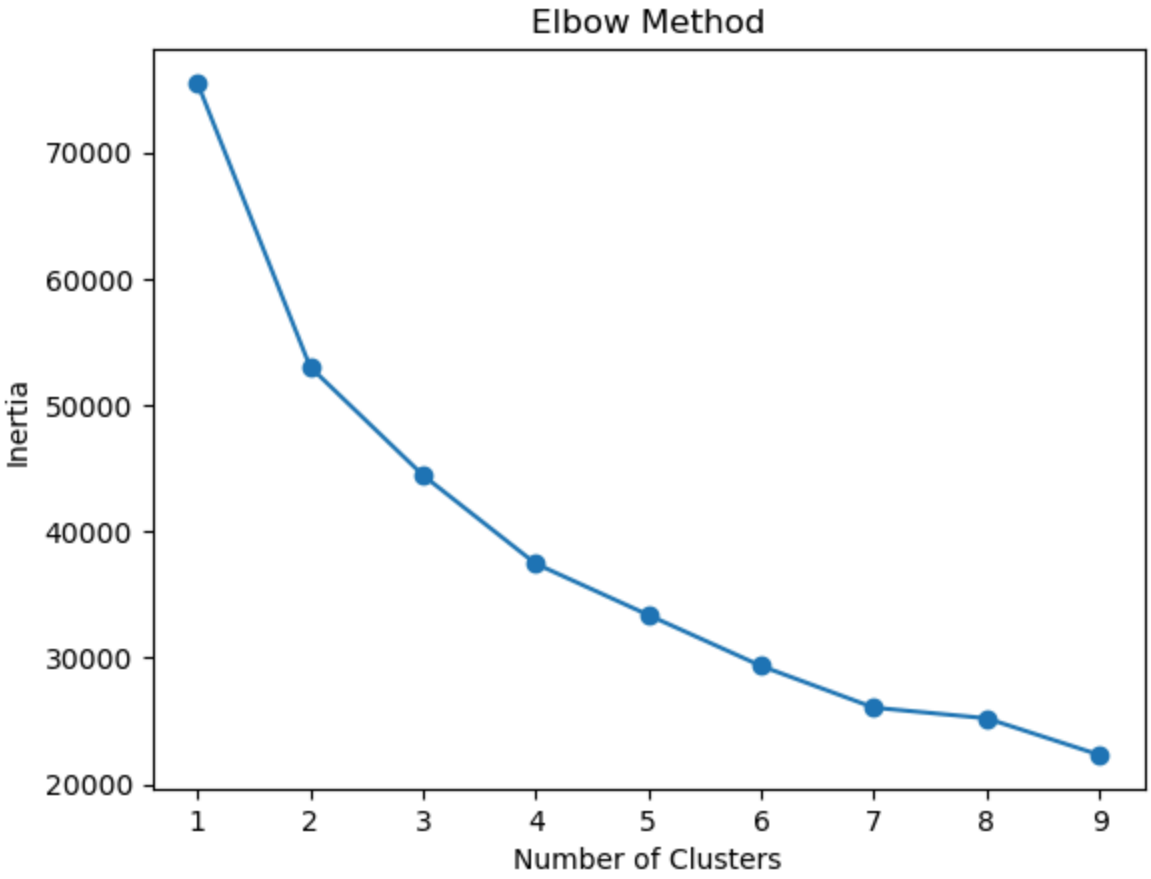
Scaling the data

In [17]:
```
from sklearn.preprocessing import StandardScaler

# Standardize features
scaler = StandardScaler()
data_scaled = scaler.fit_transform(df[features])
```

K Means Clustering

Determine the optimal number of clusters using the elbow method

In [18]:
```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Determine the optimal number of clusters using the elbow method
inertia = []
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data_scaled)
    inertia.append(kmeans.inertia_)

# Plot the elbow curve
plt.plot(range(1, 10), inertia, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.show()
```



Add K-Means clusters to df

In [19]:
```
# Apply K-Means with optimal k value
k = 4
kmeans = KMeans(n_clusters=k, random_state=42)
df['cluster'] = kmeans.fit_predict(data_scaled)
```

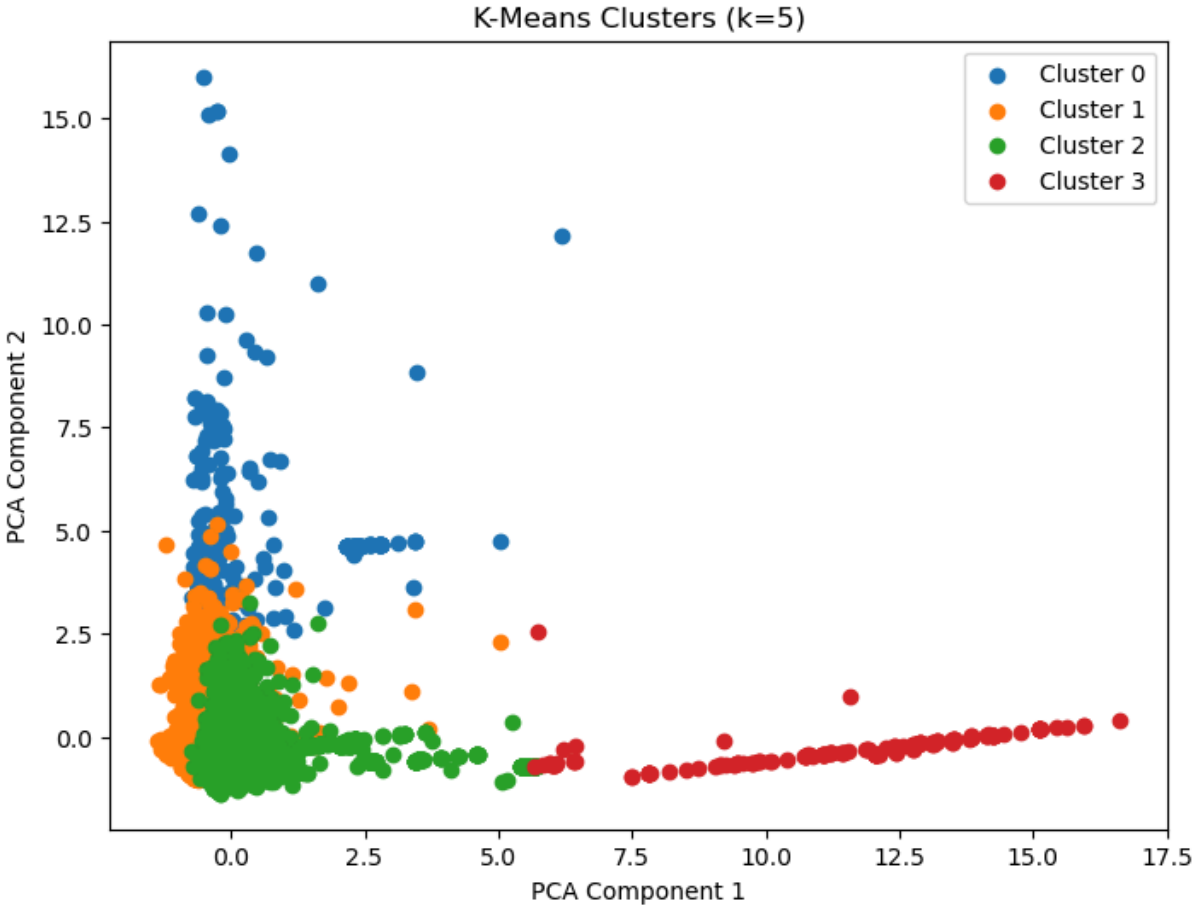
PCA

2 principle components

In [20]:
```
from sklearn.decomposition import PCA

# Reduce dimensions with PCA
pca = PCA(n_components=2)
data_pca = pca.fit_transform(data_scaled)
```

In [21]:
```
import matplotlib.pyplot as plt

# Plot the clusters
plt.figure(figsize=(8, 6))
for cluster in range(k):
    plt.scatter(
        data_pca[df['cluster'] == cluster, 0],
        data_pca[df['cluster'] == cluster, 1],
```

```
        label=f'Cluster {cluster}'
    )
# plt.scatter(
#     kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
#     s=200, c='red', marker='X', label='Centroids'
# )
plt.title('K-Means Clusters (k=5)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend()
plt.show()
```

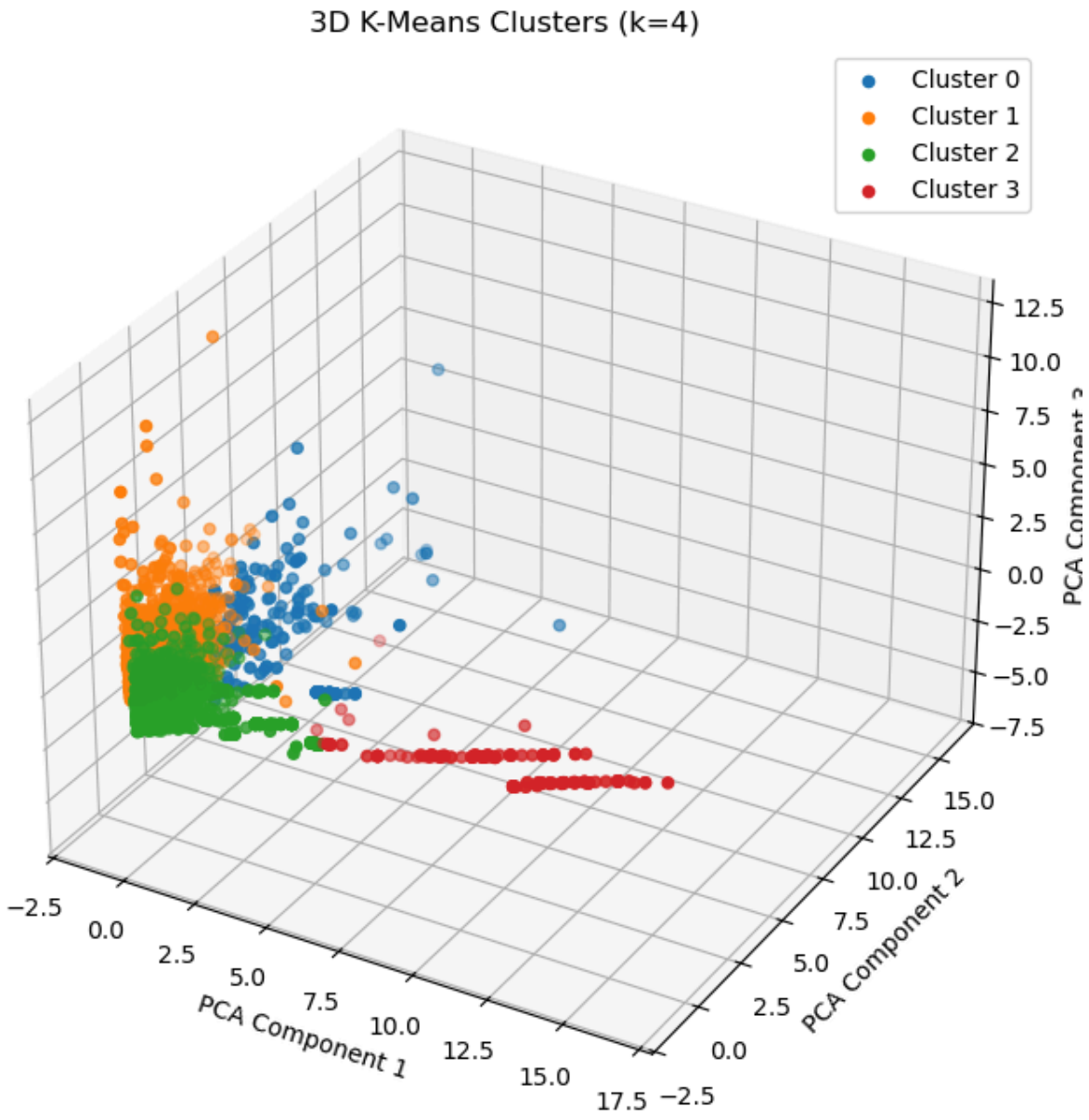


3 principle components

In [22]:
```
from sklearn.decomposition import PCA

# Reduce dimensions with PCA
pca = PCA(n_components=3)
data_pca = pca.fit_transform(data_scaled)
```

In [24]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Assuming `data_pca` has three components (PCA Component 1, 2, and 3)
# Modify for 3D visualization
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
```

```
for cluster in range(k):
    ax.scatter(
        data_pca[df['cluster'] == cluster, 0],
        data_pca[df['cluster'] == cluster, 1],
        data_pca[df['cluster'] == cluster, 2],
        label=f'Cluster {cluster}'
    )

# Add labels and title
ax.set_title('3D K-Means Clusters (k=4)')
ax.set_xlabel('PCA Component 1')
ax.set_ylabel('PCA Component 2')
ax.set_zlabel('PCA Component 3')
ax.legend()

plt.show()
```



INTERACTIVE 3D PLOTTING PCA

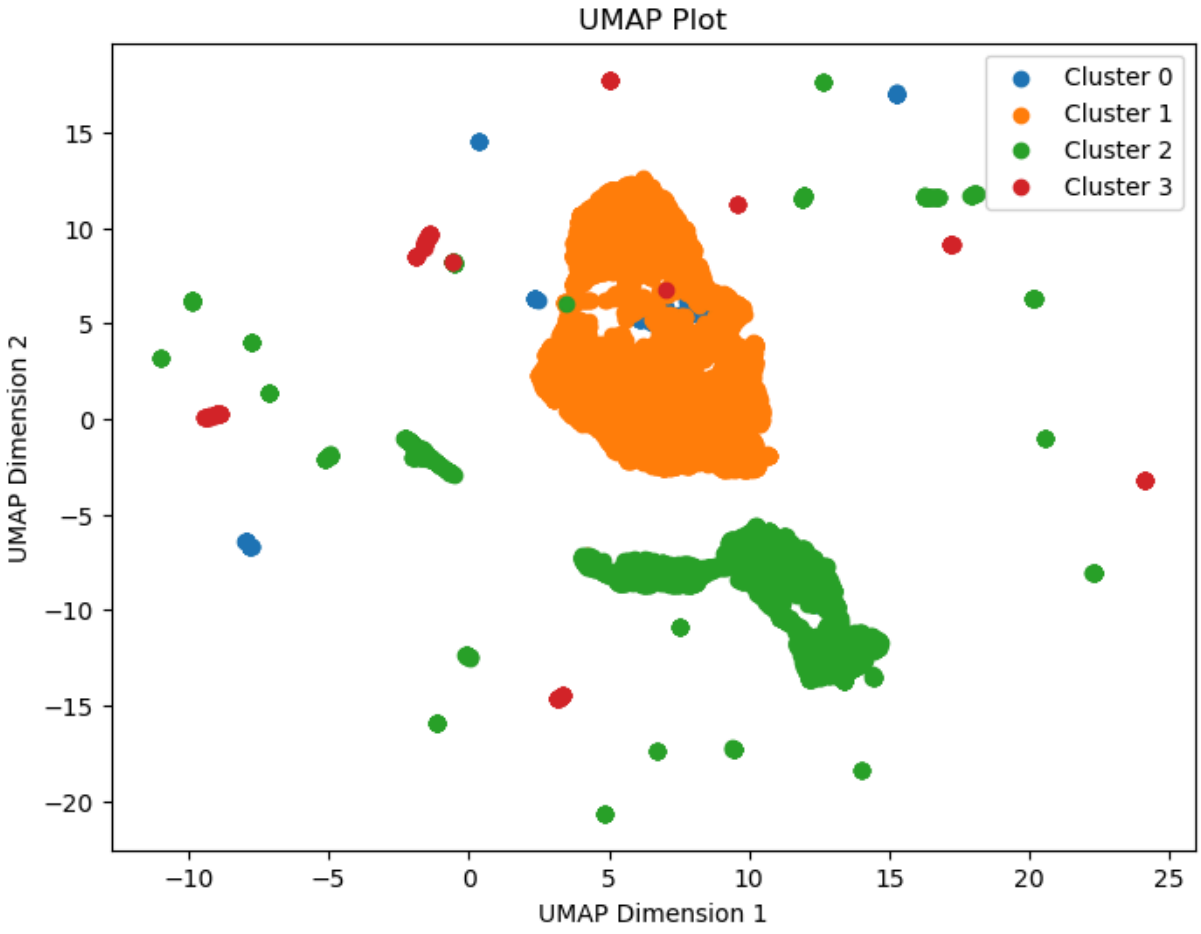
In [ ]:
```
from mayavi import mlab
from sklearn.decomposition import PCA

# Reduce dimensions with PCA
pca = PCA(n_components=3)
data_pca = pca.fit_transform(data_scaled)

x, y, z = data_pca[:, 0], data_pca[:, 1], data_pca[:, 2]
labels = df['cluster']

# Map labels to a colormap for visualization

categories = labels.unique()
color_map = {
    category: i for i, category in enumerate(categories)
}
colors = np.array([color_map[label] for label in labels])  # Map labels to i

# 5. Create 3D scatter plot using Mayavi
mlab.figure(size=(800, 600), bgcolor=(1, 1, 1))  # White background

# Create the scatter plot
points = mlab.points3d(
    x, y, z,
    colors,
    scale_factor=0.5,
    colormap='blue-red', # Choose a colormap
    scale_mode='none'
)

# Add labels for the axes
mlab.axes(
    xlabel='PCA Component 1',
    ylabel='PCA Component 2',
    zlabel='PCA Component 3'
)
mlab.colorbar(points, title='Label', orientation='vertical')  # Add colorbar
mlab.title('3D PCA Plot')

# Show the plot
mlab.show()
```

UMAP

In [26]:
```
import umap

# Apply UMAP
umap_reducer = umap.UMAP(n_components=2, random_state=42)
data_umap = umap_reducer.fit_transform(data_scaled)


# Plot the clusters
```

```
plt.figure(figsize=(8, 6))
for cluster in range(k):
    plt.scatter(
        data_umap[df['cluster'] == cluster, 0],
        data_umap[df['cluster'] == cluster, 1],
        label=f'Cluster {cluster}'
    )
# plt.scatter(
#     kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
#     s=200, c='red', marker='X', label='Centroids'
# )
plt.title('UMAP Plot')
plt.xlabel('UMAP Dimension 1')
plt.ylabel('UMAP Dimension 2')
plt.legend()
plt.show()
```

```
/Users/theo/miniforge3/envs/data_project/lib/python3.10/site-packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(
```



INTERACTIVE 3D PLOTTING UMAP

In [ ]:

```
from mayavi import mlab
import umap

# Apply UMAP
```

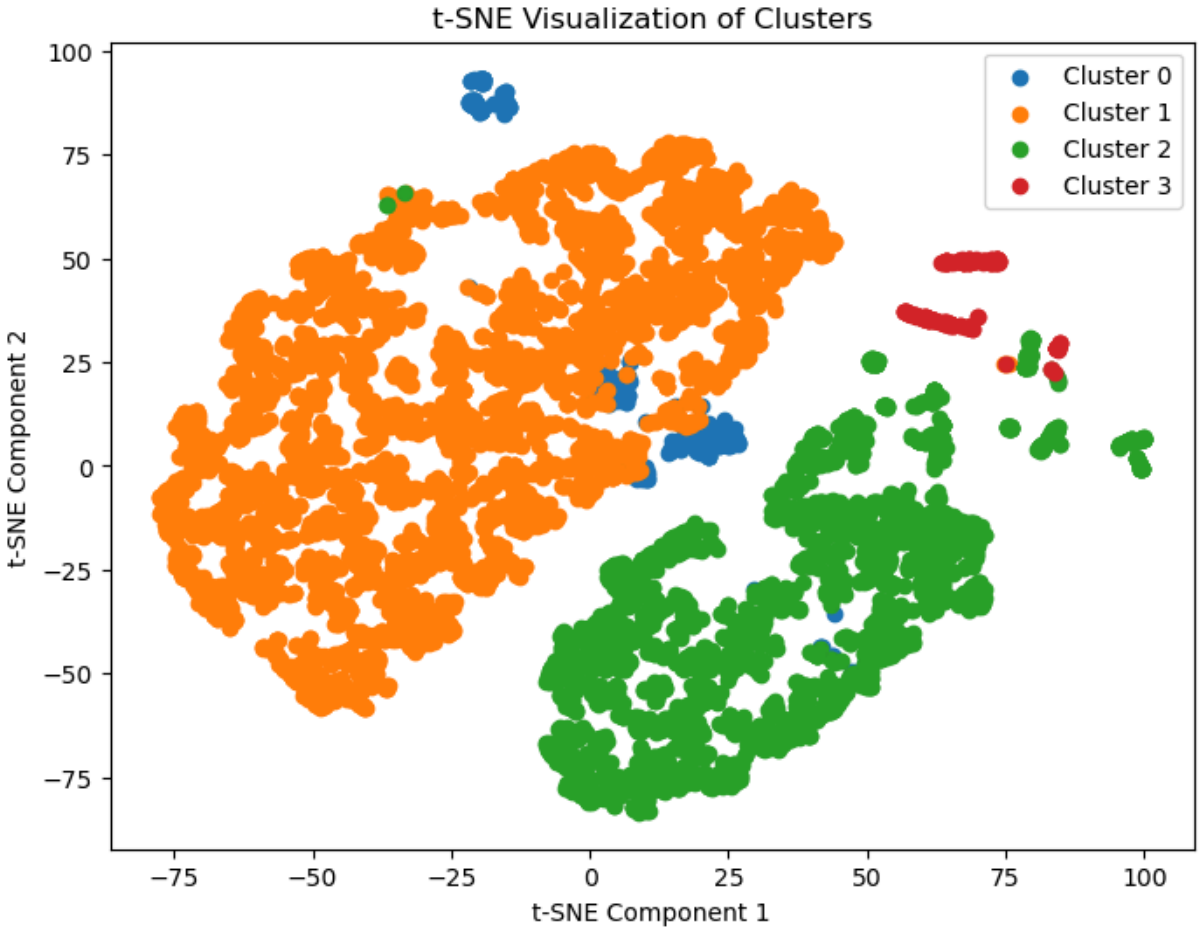
```
umap_reducer = umap.UMAP(n_components=3, random_state=42)
data_umap = umap_reducer.fit_transform(data_scaled)

x, y, z = data_umap[:, 0], data_umap[:, 1], data_umap[:, 2]
labels = df['cluster']

# Map labels to a colormap for visualization

categories = labels.unique()
color_map = {
    category: i for i, category in enumerate(categories)
}
colors = np.array([color_map[label] for label in labels])  # Map labels to i

# 5. Create 3D scatter plot using Mayavi
mlab.figure(size=(800, 600), bgcolor=(200,200,200))

# Create the scatter plot
points = mlab.points3d(
    x, y, z,
    colors,
    scale_factor=0.5,
    colormap='tab20', # Choose a colormap
    scale_mode='none'
)

# Add labels for the axes
mlab.axes(
    xlabel='UMAP 1',
    ylabel='UMAP 2',
    zlabel='UMAP 3'
)

mlab.title('3D UMAP Plot')

# Show the plot
mlab.show()
```

t-SNE

In [27]:

```
from sklearn.manifold import TSNE

# Reduce dimensions with t-SNE
tsne = TSNE(n_components=2, random_state=42)
data_tsne = tsne.fit_transform(data_scaled)

# Plot the clusters
plt.figure(figsize=(8, 6))
for cluster in range(k):
    plt.scatter(
        data_tsne[df['cluster'] == cluster, 0],
        data_tsne[df['cluster'] == cluster, 1],
        label=f'Cluster {cluster}'
    )
plt.title('t-SNE Visualization of Clusters')
```

```
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.legend()
plt.show()
```



INTERACTIVE 3D PLOTTING TSNE

In [ ]:
```
from mayavi import mlab
from sklearn.manifold import TSNE

# Reduce dimensions with t-SNE
tsne = TSNE(n_components=3, random_state=42)
data_tsne = tsne.fit_transform(data_scaled)

x, y, z = data_tsne[:, 0], data_tsne[:, 1], data_tsne[:, 2]
labels = df['cluster']

# Map labels to a colormap for visualization

categories = labels.unique()
color_map = {
    category: i for i, category in enumerate(categories)
}
colors = np.array([color_map[label] for label in labels])  # Map labels to i

# 5. Create 3D scatter plot using Mayavi
mlab.figure(size=(800, 600), bgcolor=(1, 1, 1))  # White background
```

```
# Create the scatter plot
points = mlab.points3d(
    x, y, z,
    colors,
    scale_factor=0.5,
    colormap='blue-red', # Choose a colormap
    scale_mode='none'
)

# Add labels for the axes
mlab.axes(
    xlabel='TSNE 1',
    ylabel='TSNE 2',
    zlabel='TSNE 3'
)
mlab.colorbar(points, title='Label', orientation='vertical')  # Add colorbar
mlab.title('3D TSNE Plot')

# Show the plot
mlab.show()
```

Stress Calculation for embedders

In [31]:
```
from scipy.stats import spearmanr
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import pairwise_distances

# Function to calculate stress
def calculate_stress(orig_dist, embedded_dist):
    return np.sqrt(np.sum((orig_dist - embedded_dist)**2)) / np.sqrt(np.sum(

# Function to create Shepard diagram
def plot_shepard_diagram(ax, original_distances, embedded_distances, method_
    # Flatten the distance matrices
    orig_dist_flat = original_distances[np.triu_indices(original_distances.s
    emb_dist_flat = embedded_distances[np.triu_indices(embedded_distances.sh

    # Calculate stress
    stress = calculate_stress(orig_dist_flat, emb_dist_flat)

    # Calculate Spearman correlation
    correlation, _ = spearmanr(orig_dist_flat, emb_dist_flat)

    # Create scatter plot
    ax.scatter(orig_dist_flat, emb_dist_flat, alpha=0.5, s=20)

    # Add perfect correlation line
    min_dist = min(orig_dist_flat.min(), emb_dist_flat.min())
    max_dist = max(orig_dist_flat.max(), emb_dist_flat.max())
    ax.plot([min_dist, max_dist], [min_dist, max_dist], 'r--', alpha=0.8)

    ax.set_xlabel('Original Distances')
```

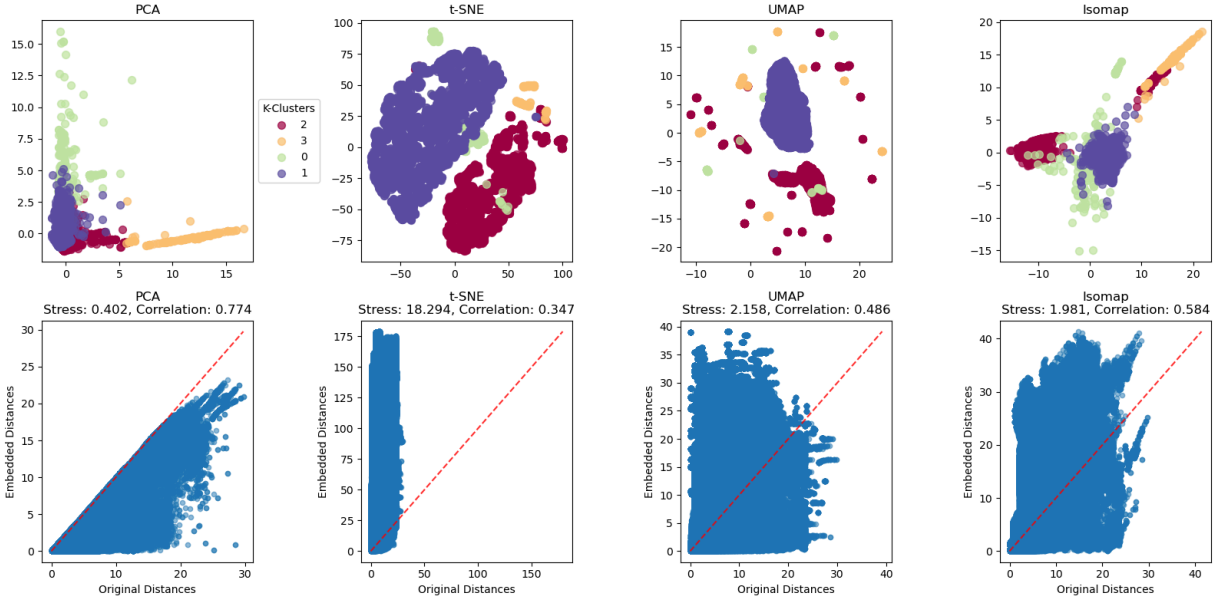
```
    ax.set_ylabel('Embedded Distances')
    ax.set_title(f'{method_name}\nStress: {stress:.3f}, Correlation: {correl
```

In [ ]:
```
from sklearn.manifold import MDS, Isomap
# Features and labels
labels = df['cluster']  # Second column contains labels (species)
features = data_scaled  # All other columns except column 2


# Unique species and colors
unique_species = labels.unique()
colors = plt.cm.Spectral(np.linspace(0, 1, len(unique_species)))

# Original pairwise distances
original_distances = pairwise_distances(features)

# Dimensionality reduction methods
embeddings = {
    'PCA': PCA(n_components=2).fit_transform(features),
    't-SNE': TSNE(n_components=2, random_state=42).fit_transform(features),
    'UMAP': umap.UMAP(n_components=2, random_state=42).fit_transform(feature
    #'MDS': MDS(n_components=2, random_state=42).fit_transform(features),
    'Isomap': Isomap(n_components=2).fit_transform(features)
}

# Pairwise distances in embedding spaces
embedded_distances = {
    method: pairwise_distances(embedding)
    for method, embedding in embeddings.items()
}
```

In [44]:
```
# Visualization
fig = plt.figure(figsize=(16, 8))
n_methods = len(embeddings)

for idx, (method, embedding) in enumerate(embeddings.items()):
    # Top row: Projection plot
    ax_proj = plt.subplot(2, n_methods, idx + 1)
    for i, species in enumerate(unique_species):
        species_mask = labels == species
        ax_proj.scatter(
            embedding[species_mask, 0],
            embedding[species_mask, 1],
            c=[colors[i]],
            label=species,
            s=50,
            alpha=0.7
        )
    ax_proj.set_title(method)
    if idx == 0:
        ax_proj.legend(title="K-Clusters", loc="center left", bbox_to_anchor

    # Bottom row: Shepard diagram
    ax_shep = plt.subplot(2, n_methods, idx + n_methods + 1)
    plot_shepard_diagram(ax_shep, original_distances, embedded_distances[met
```

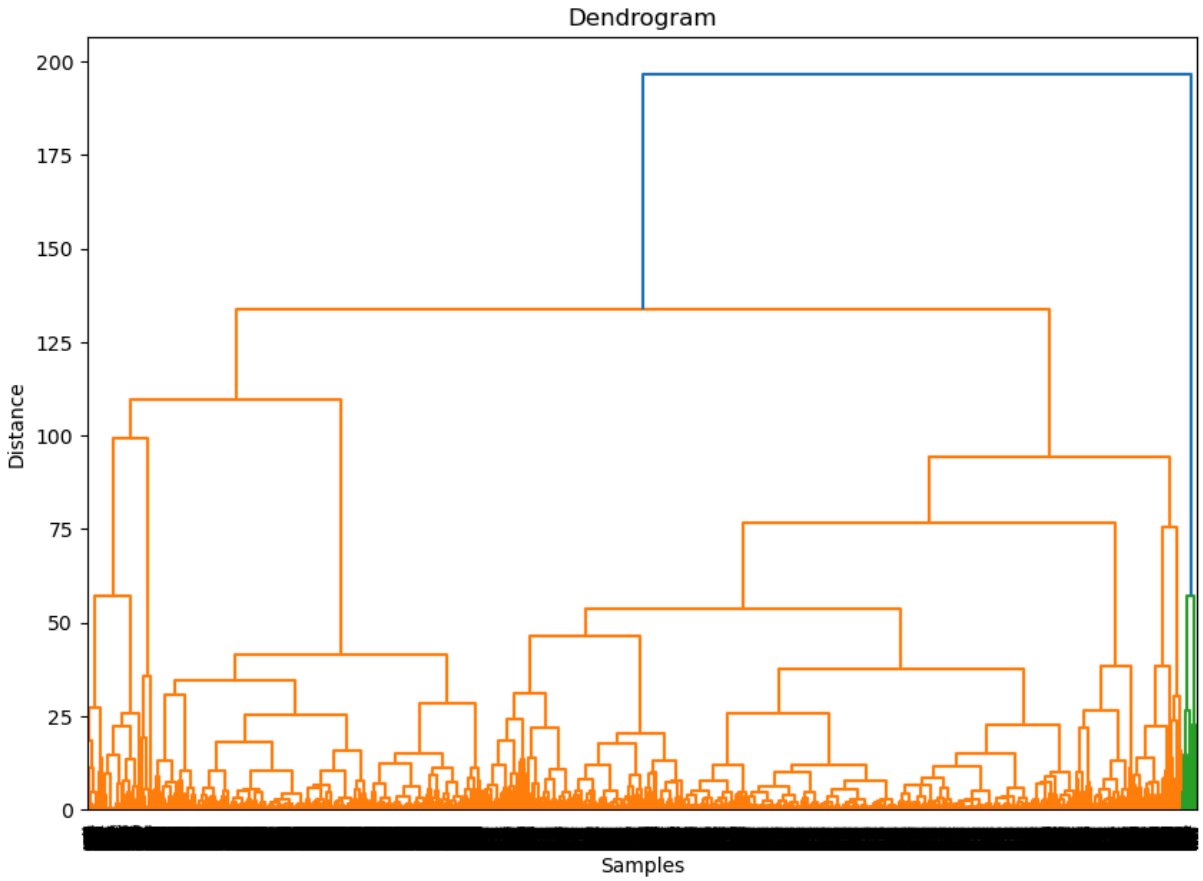
```
# Print stress values
print("\nStress values:")
print("-" * 50)
for method, distances in embedded_distances.items():
    stress = calculate_stress(original_distances, distances)
    print(f"{method}: {stress:.3f}")

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

```
Stress values:
--------------------------------------------------
PCA: 0.402
t-SNE: 18.294
UMAP: 2.158
Isomap: 1.981
```

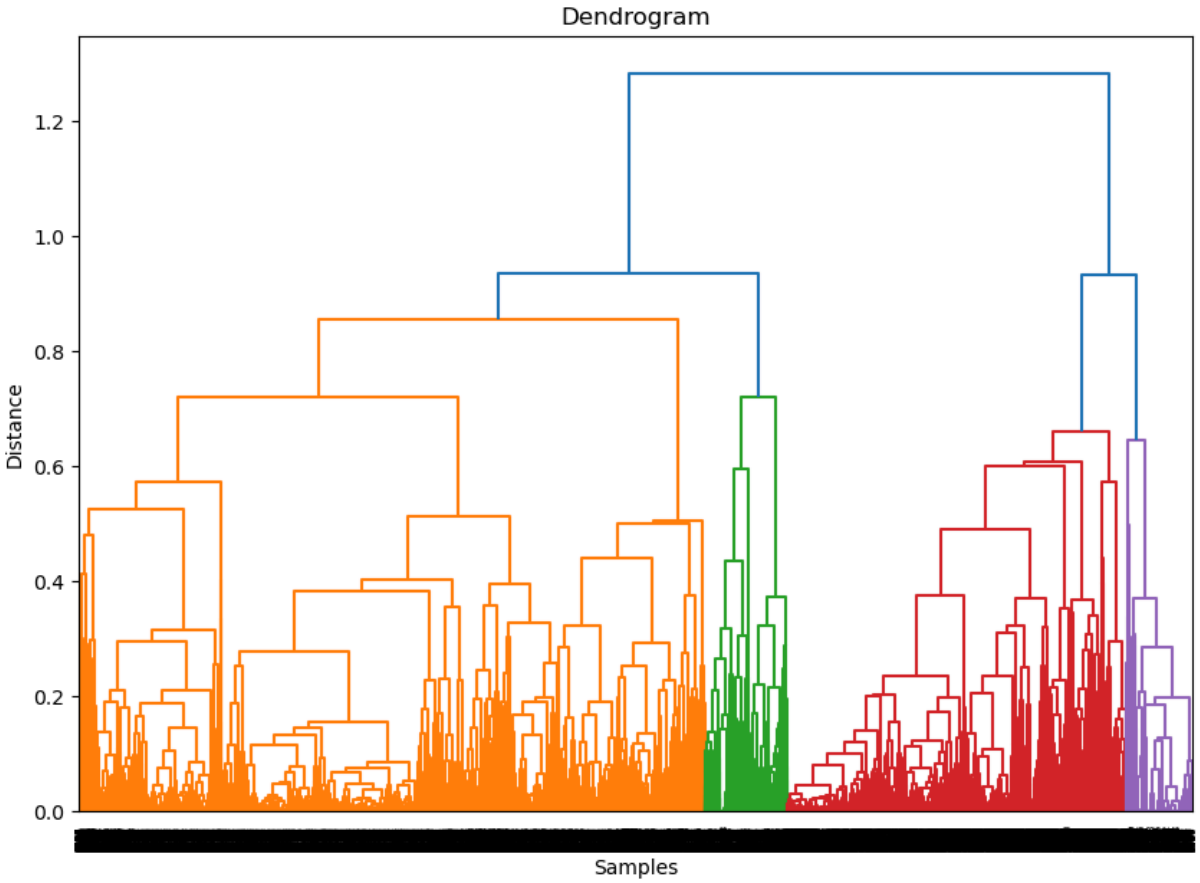


Hierarchical Clustering

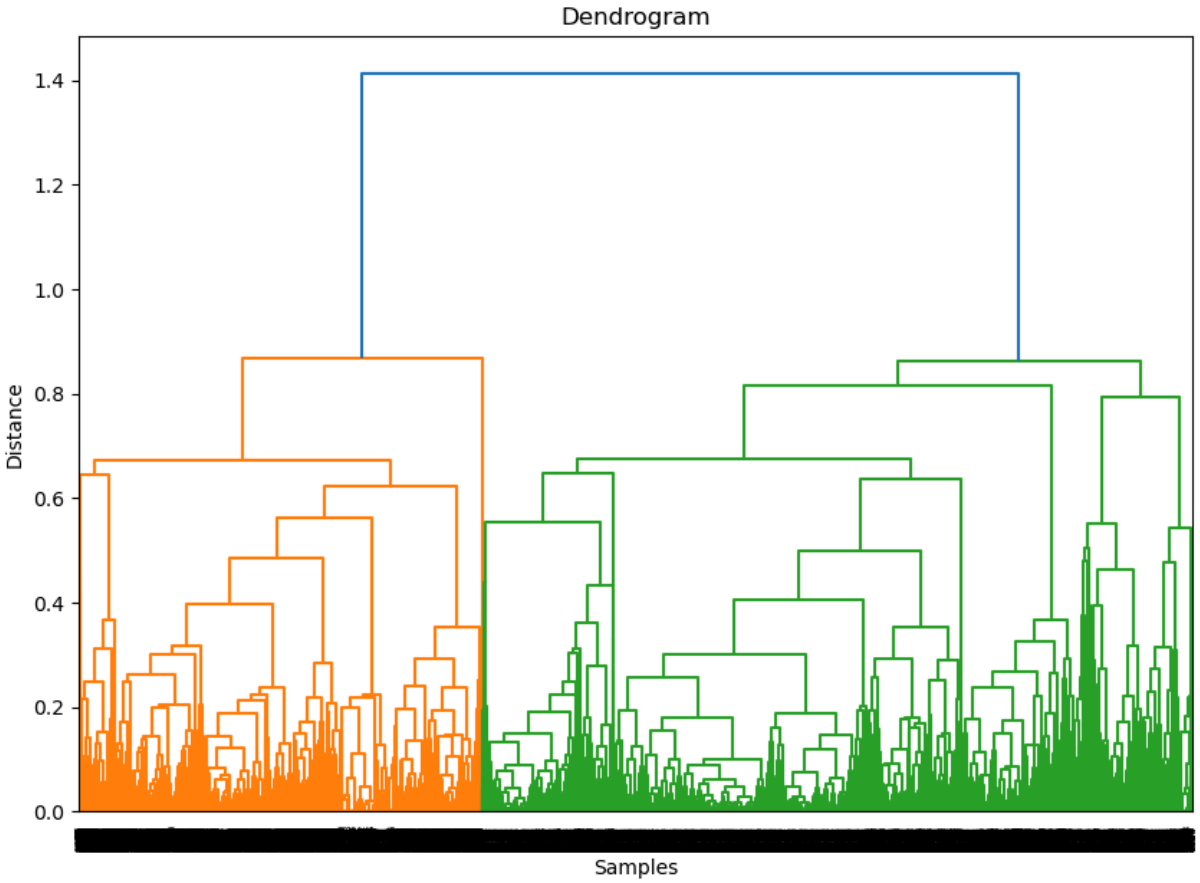
In [45]:
```
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

# Perform hierarchical clustering
linked = linkage(data_scaled, method='ward', metric='euclidean')  # 'ward' m

# Plot dendrogram
plt.figure(figsize=(10, 7))
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_
plt.title('Dendrogram')
plt.xlabel('Samples')
plt.ylabel('Distance')
plt.show()
```



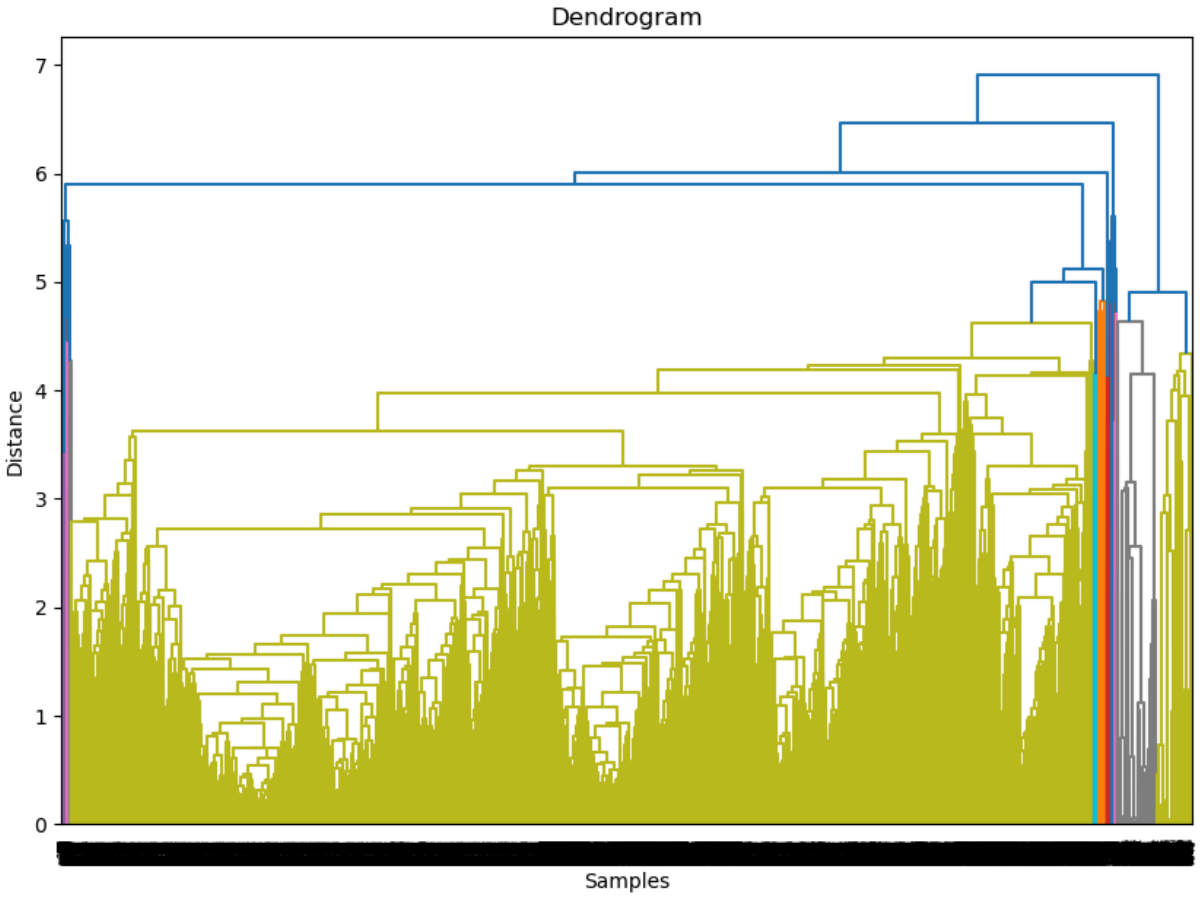
In [51]:
```
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

# Perform hierarchical clustering
linked = linkage(data_scaled, method='average', metric='cosine')

# Plot dendrogram
plt.figure(figsize=(10, 7))
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_
plt.title('Dendrogram')
plt.xlabel('Samples')
plt.ylabel('Distance')
plt.show()
```



In [54]:
```
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

# Perform hierarchical clustering
linked = linkage(data_scaled, method='average', metric='correlation')

# Plot dendrogram
plt.figure(figsize=(10, 7))
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_
plt.title('Dendrogram')
plt.xlabel('Samples')
plt.ylabel('Distance')
plt.show()
```



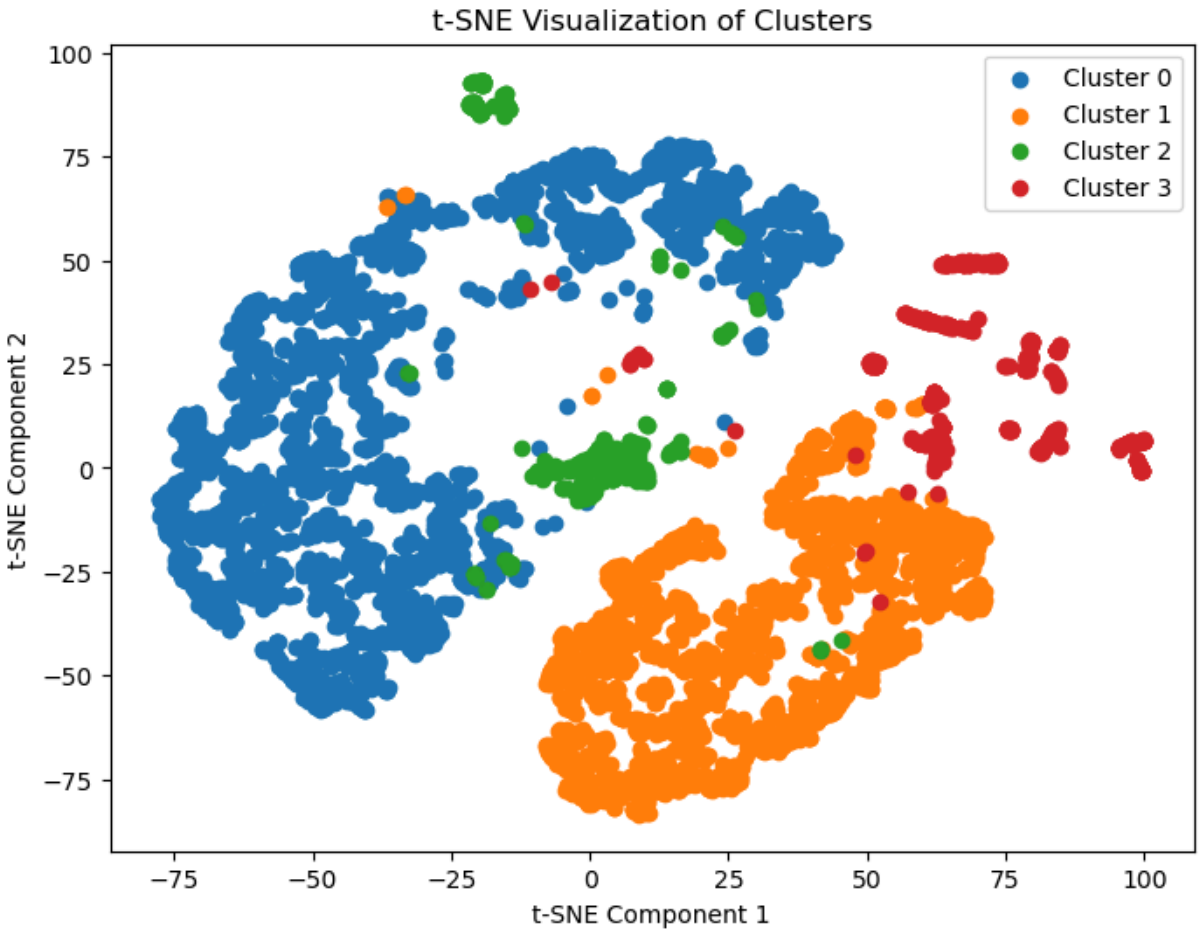
In [55]:
```
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

# Perform hierarchical clustering
linked = linkage(data_scaled, method='average', metric='canberra')

# Plot dendrogram
plt.figure(figsize=(10, 7))
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_
plt.title('Dendrogram')
plt.xlabel('Samples')
plt.ylabel('Distance')
plt.show()
```



In [56]:
```
from sklearn.cluster import AgglomerativeClustering

# Cut the dendrogram to form clusters (e.g., 5 clusters)
# hierarchical = AgglomerativeClustering(n_clusters=5, metric='euclidean', l
# hierarchical = AgglomerativeClustering(n_clusters=6, metric='correlation',
# hierarchical = AgglomerativeClustering(n_clusters=4, metric='correlation',
hierarchical = AgglomerativeClustering(n_clusters=6, metric='cosine', linkag
df['hierarchical_cluster'] = hierarchical.fit_predict(data_scaled)
```

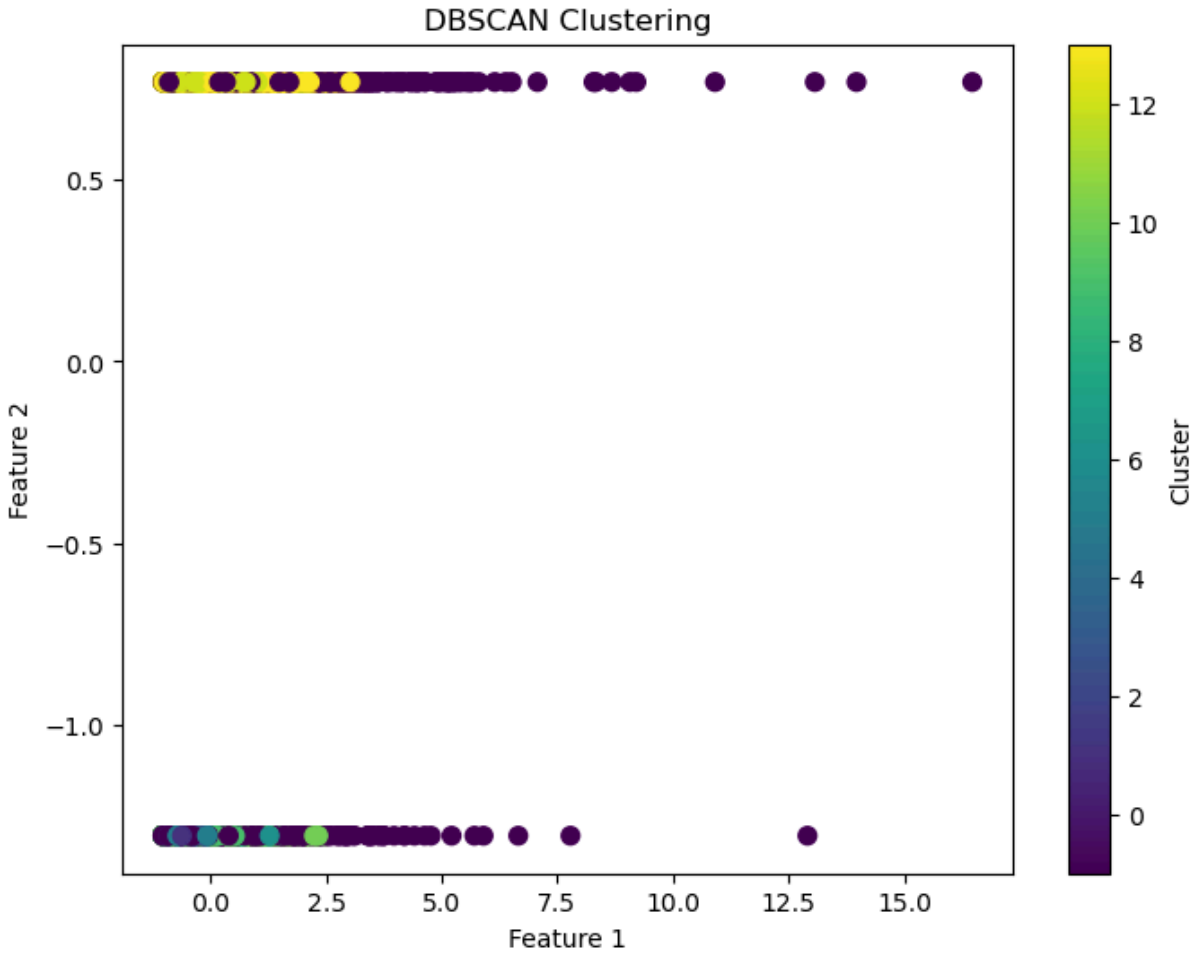
In [57]:
```
from sklearn.manifold import TSNE

# Reduce dimensions with t-SNE
tsne = TSNE(n_components=2, random_state=42)
data_tsne = tsne.fit_transform(data_scaled)

# Plot the clusters
plt.figure(figsize=(8, 6))
for cluster in range(k):
    plt.scatter(
        data_tsne[df['hierarchical_cluster'] == cluster, 0],
        data_tsne[df['hierarchical_cluster'] == cluster, 1],
        label=f'Cluster {cluster}'
    )
plt.title('t-SNE Visualization of Clusters')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
```

```
plt.legend()
plt.show()
```



HDBSCAN Cluster

In [28]:
```
import hdbscan

clusterer = hdbscan.HDBSCAN(min_cluster_size=25, gen_min_span_tree=True)
df['hdbscan_cluster'] = clusterer.fit_predict(data_scaled)

# Check cluster labels (-1 indicates noise)
#print(df['hdbscan_cluster'].value_counts())

# Plot DBSCAN clusters
plt.figure(figsize=(8, 6))
plt.scatter(data_scaled[:, 0], data_scaled[:, 1], c=df['hdbscan_cluster'], c
plt.title('DBSCAN Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar(label='Cluster')
plt.show()
```



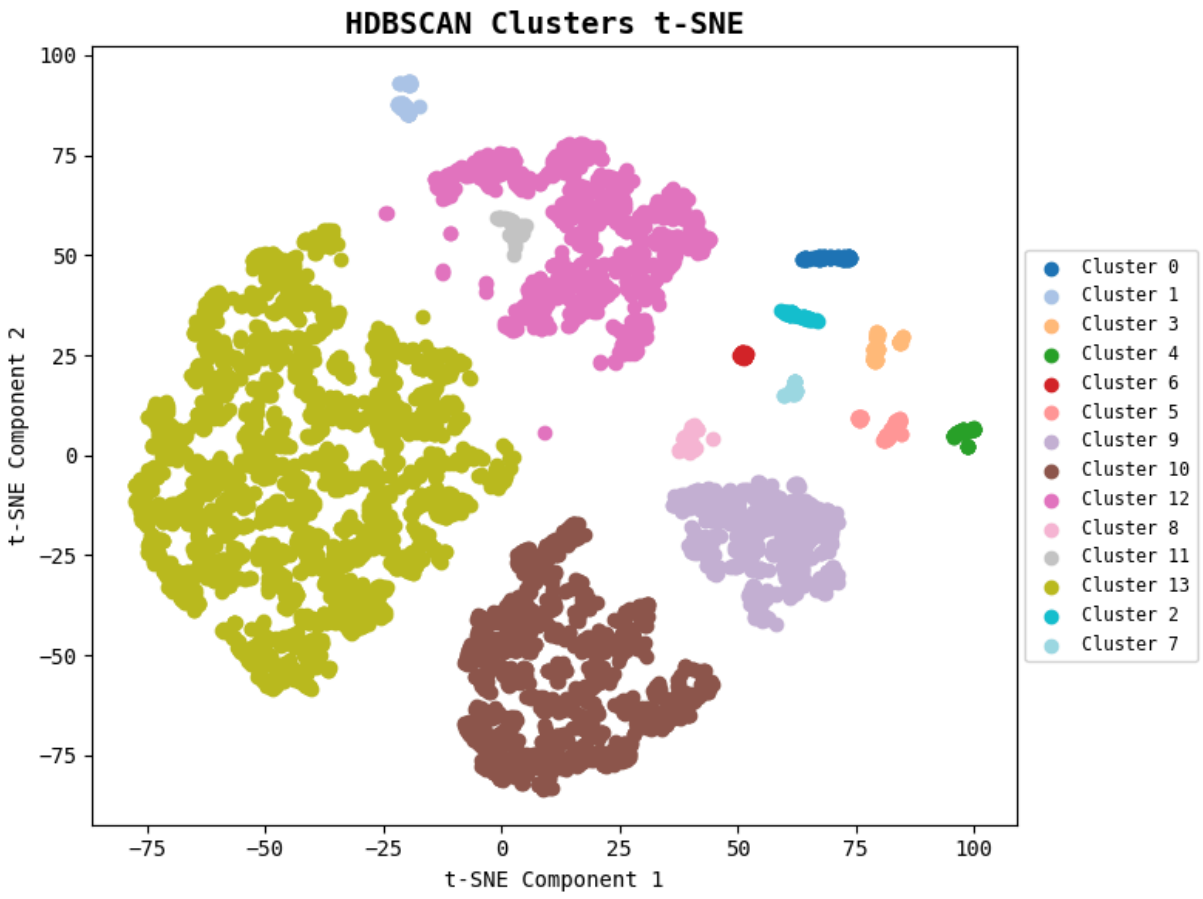
2D t-SNE

In [29]:
```
from sklearn.manifold import TSNE
# Reduce dimensions with t-SNE
tsne = TSNE(n_components=2, random_state=42)
data_tsne = tsne.fit_transform(data_scaled)
```

In [142…
```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.cm as cm

# Define the number of clusters and a colormap
unique_clusters = df['hdbscan_cluster'].unique()
unique_clusters = [cluster for cluster in unique_clusters if cluster != -1]
k = len(unique_clusters)
colormap = cm.get_cmap('tab20', k)  # Choose a colormap with k colors
# Set the font to monospace
plt.rc('font', family='monospace')

# Plot the clusters
plt.figure(figsize=(8, 6))
for idx, cluster in enumerate(unique_clusters):
    # Get the color from the colormap
    color = colormap(idx)
    cluster_mask = df['hdbscan_cluster'] == cluster
```

```
    # Scatter plot for each cluster
    plt.scatter(
        data_tsne[cluster_mask, 0],
        data_tsne[cluster_mask, 1],
        color=color,
        label=f'Cluster {cluster}'
    )

# Add plot labels and legend
plt.title('HDBSCAN Clusters t-SNE ', fontsize=14, fontweight='bold')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')

# Place the legend outside the plot on the right
plt.legend(loc='upper left', bbox_to_anchor=(1, 0.75), fontsize='small')

# Adjust the layout to ensure the legend fits
plt.tight_layout()

plt.show()
```

```
/var/folders/78/h5q_zlz123d11y9zm52f2p4c0000gn/T/ipykernel_3573/2495304927.py:9: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be removed in 3.11. Use ``matplotlib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap()`` or ``pyplot.get_cmap()`` instead.
  colormap = cm.get_cmap('tab20', k)  # Choose a colormap with k colors
```

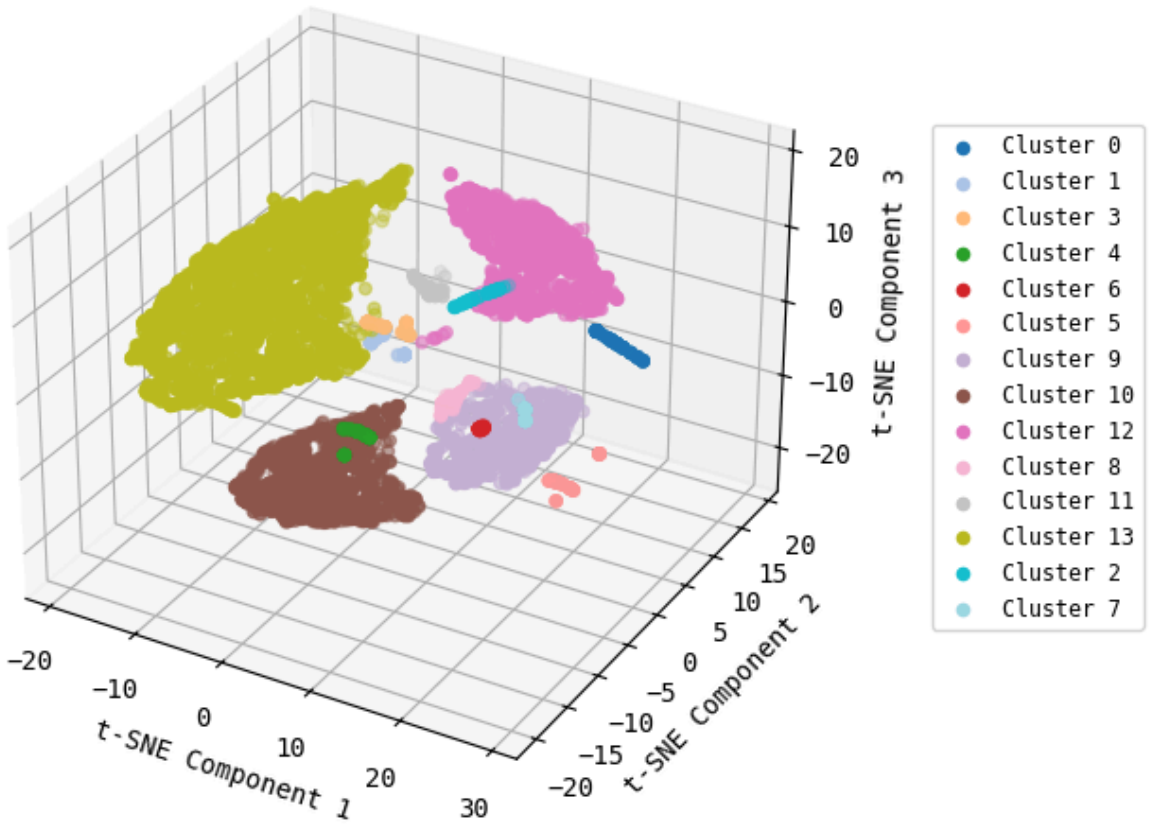


3D t-SNE

In [48]:
```
from sklearn.manifold import TSNE
# Reduce dimensions with t-SNE to 3D
tsne = TSNE(n_components=3, random_state=42)
data_tsne = tsne.fit_transform(data_scaled)
```

In [132…
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.cm as cm

# Set the font to monospace
plt.rc('font', family='monospace')

# Define the number of clusters and a colormap
unique_clusters = df['hdbscan_cluster'].unique()
unique_clusters = [cluster for cluster in unique_clusters if cluster != -1]
k = len(unique_clusters)
colormap = cm.get_cmap('tab20', k)  # Choose a colormap with k colors

# Add white space to the right for the legend
fig = plt.figure(figsize=(16, 6))
ax = fig.add_subplot(111, projection='3d')

for idx, cluster in enumerate(unique_clusters):
    color = colormap(idx)
    cluster_mask = df['hdbscan_cluster'] == cluster
    ax.scatter(
        data_tsne[cluster_mask, 0],
        data_tsne[cluster_mask, 1],
        data_tsne[cluster_mask, 2],
        color=color,
        label=f'Cluster {cluster}'
    )

# Adjust the layout to add more space on the right
plt.subplots_adjust(right=0.9)  # Adjust this value to increase space
ax.set_title('3D t-SNE HDBSCAN Clusters', pad=0, fontsize=14, y=0.998, loc='
ax.set_xlabel('t-SNE Component 1')
ax.set_ylabel('t-SNE Component 2')
ax.set_zlabel('t-SNE Component 3')

# Place the legend on the right
ax.legend(loc='center left', bbox_to_anchor=(1.1, 0.5), fontsize='small')  #

plt.show()
```

```
/var/folders/78/h5q_zlz123d11y9zm52f2p4c0000gn/T/ipykernel_3573/1548657583.p
y:13: MatplotlibDeprecationWarning: The get_cmap function was deprecated in
Matplotlib 3.7 and will be removed in 3.11. Use ``matplotlib.colormaps[name]
`` or ``matplotlib.colormaps.get_cmap()`` or ``pyplot.get_cmap()`` instead.
  colormap = cm.get_cmap('tab20', k)  # Choose a colormap with k colors
```



INTERACTIVE 3D PLOTTING TSNE

In [ ]:
```
from mayavi import mlab
from sklearn.manifold import TSNE

# Reduce dimensions with t-SNE
tsne = TSNE(n_components=3, random_state=42)
data_tsne = tsne.fit_transform(data_scaled)

x, y, z = data_tsne[:, 0], data_tsne[:, 1], data_tsne[:, 2]
labels = df['hdbscan_cluster']

# Map labels to a colormap for visualization

categories = labels.unique()
color_map = {
    category: i for i, category in enumerate(categories)
}
colors = np.array([color_map[label] for label in labels])  # Map labels to i

# 5. Create 3D scatter plot using Mayavi
mlab.figure(size=(800, 600), bgcolor=(0.5, 0.5, 0.5))  # White background

# Create the scatter plot
points = mlab.points3d(
    x, y, z,
    colors,
```

```
    scale_factor=0.5,
    colormap='Vega20c', # Choose a colormap
    scale_mode='none'
)

# Add labels for the axes
mlab.axes(
    xlabel='TSNE 1',
    ylabel='TSNE 2',
    zlabel='TSNE 3'
)
mlab.scalarbar(points, title='Clusters', orientation='vertical', nb_colors=1
mlab.title('3D TSNE Plot', size=0.3, height=0.85)

# Show the plot
mlab.show()
# mlab.view(azimuth=45, elevation=30, distance=10)
# mlab.savefig('custom_view.png')
```

INTERACTIVE 3D PLOTTING ISOMAP

In [ ]:

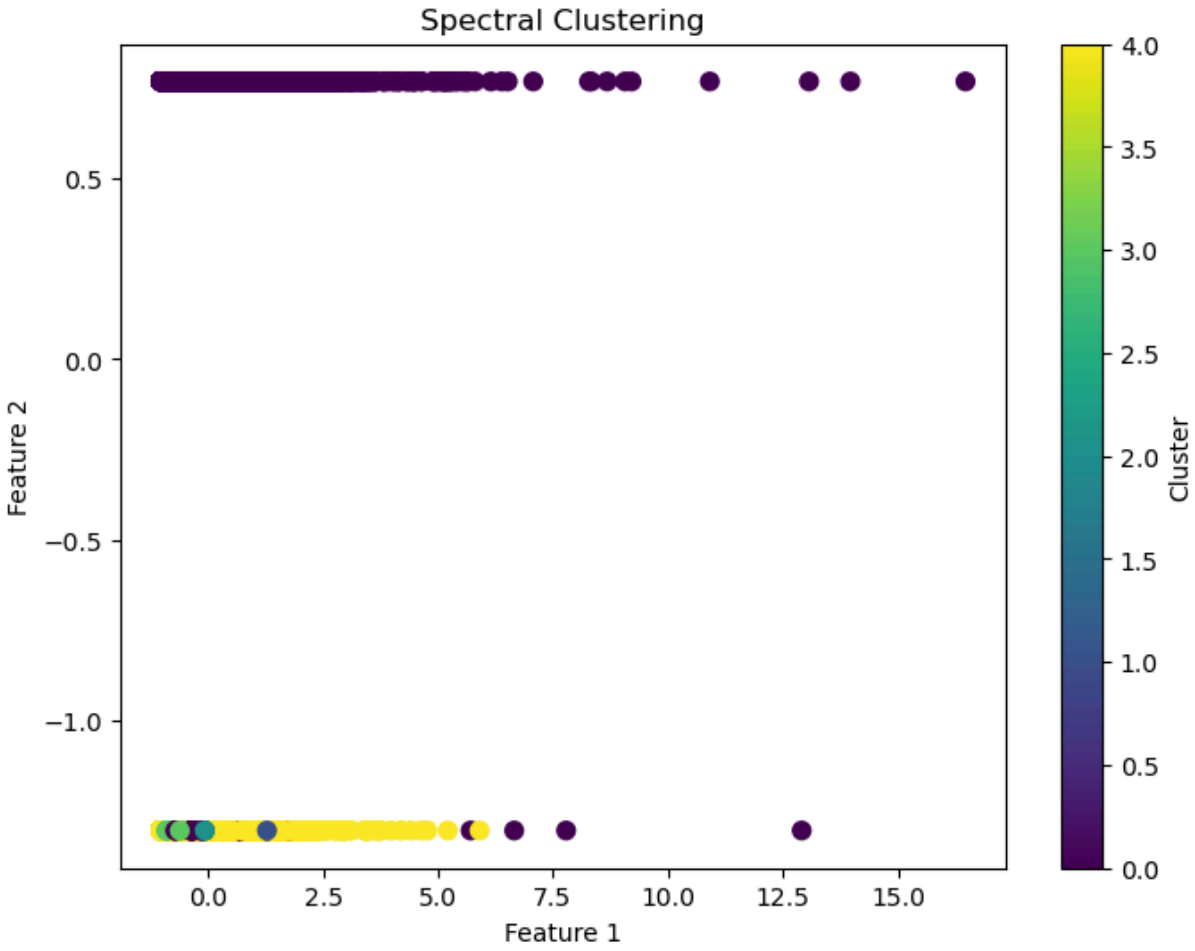
```
from mayavi import mlab
from sklearn.manifold import Isomap

# Reduce dimensions with IsoMap
isomap = Isomap(n_components=3, n_neighbors=10)
data_isomap = isomap.fit_transform(data_scaled)

x, y, z = data_isomap[:, 0], data_isomap[:, 1], data_isomap[:, 2]
labels = df['hdbscan_cluster']

# Map labels to a colormap for visualization

categories = labels.unique()
color_map = {
    category: i for i, category in enumerate(categories)
}
colors = np.array([color_map[label] for label in labels])  # Map labels to i

# 5. Create 3D scatter plot using Mayavi
mlab.figure(size=(800, 600), bgcolor=(1, 1, 1))  # White background

# Create the scatter plot
points = mlab.points3d(
    x, y, z,
    colors,
    scale_factor=0.5,
    colormap='blue-red', # Choose a colormap
    scale_mode='none'
)

# Add labels for the axes
mlab.axes(
    xlabel='Isomap 1',
    ylabel='Isomap 2',
```

```
    zlabel='Isomap 3'
)
mlab.colorbar(points, title='Label', orientation='vertical')  # Add colorbar
mlab.title('3D Isomap Plot')

# Show the plot
mlab.show()
```

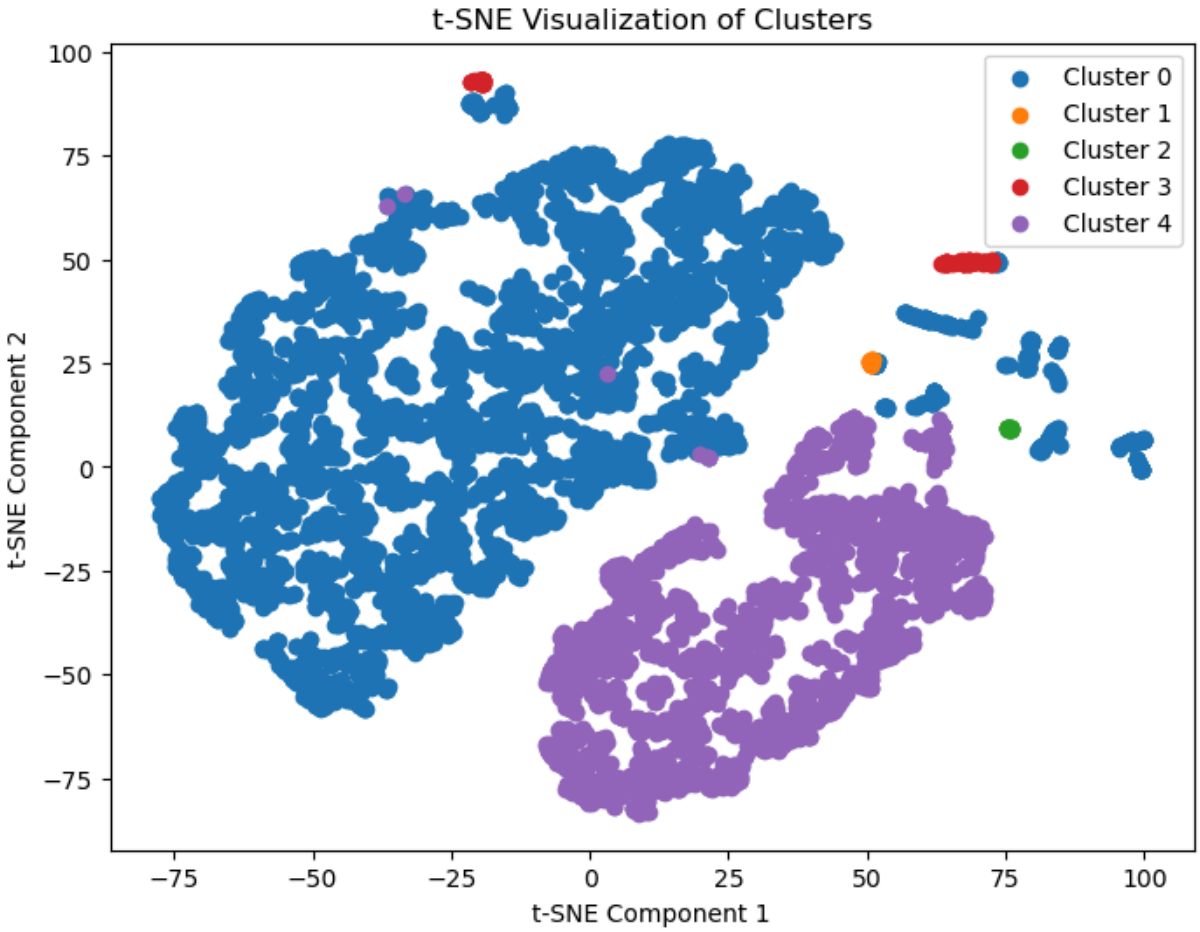
Spectral Clustering

In [63]:
```
from sklearn.cluster import SpectralClustering

# Perform Spectral Clustering
spectral = SpectralClustering(n_clusters=5, affinity='nearest_neighbors', ra
df['spectral_cluster'] = spectral.fit_predict(data_scaled)

# Visualize Spectral Clustering
plt.figure(figsize=(8, 6))
plt.scatter(data_scaled[:, 0], data_scaled[:, 1], c=df['spectral_cluster'],
plt.title('Spectral Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar(label='Cluster')
plt.show()
```

```
/Users/theo/miniforge3/envs/data_project/lib/python3.10/site-packages/sklear
n/manifold/_spectral_embedding.py:329: UserWarning: Graph is not fully conne
cted, spectral embedding may not work as expected.
  warnings.warn(
```



In [64]:
```
from sklearn.manifold import TSNE

# Reduce dimensions with t-SNE
tsne = TSNE(n_components=2, random_state=42)
data_tsne = tsne.fit_transform(data_scaled)

# Plot the clusters
plt.figure(figsize=(8, 6))
k = len(df['spectral_cluster'].unique())
for cluster in range(k):
    plt.scatter(
        data_tsne[df['spectral_cluster'] == cluster, 0],
        data_tsne[df['spectral_cluster'] == cluster, 1],
        label=f'Cluster {cluster}'
    )
plt.title('t-SNE Visualization of Clusters')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.legend()
plt.show()
```

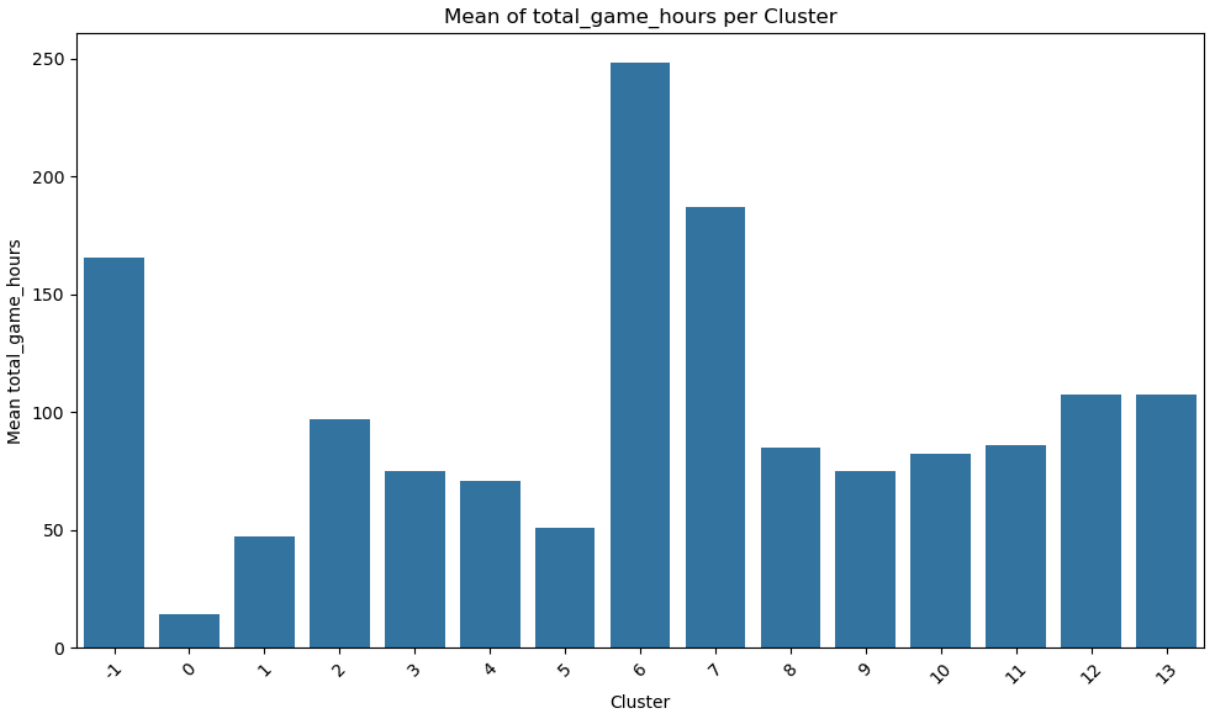


Analysing clusters

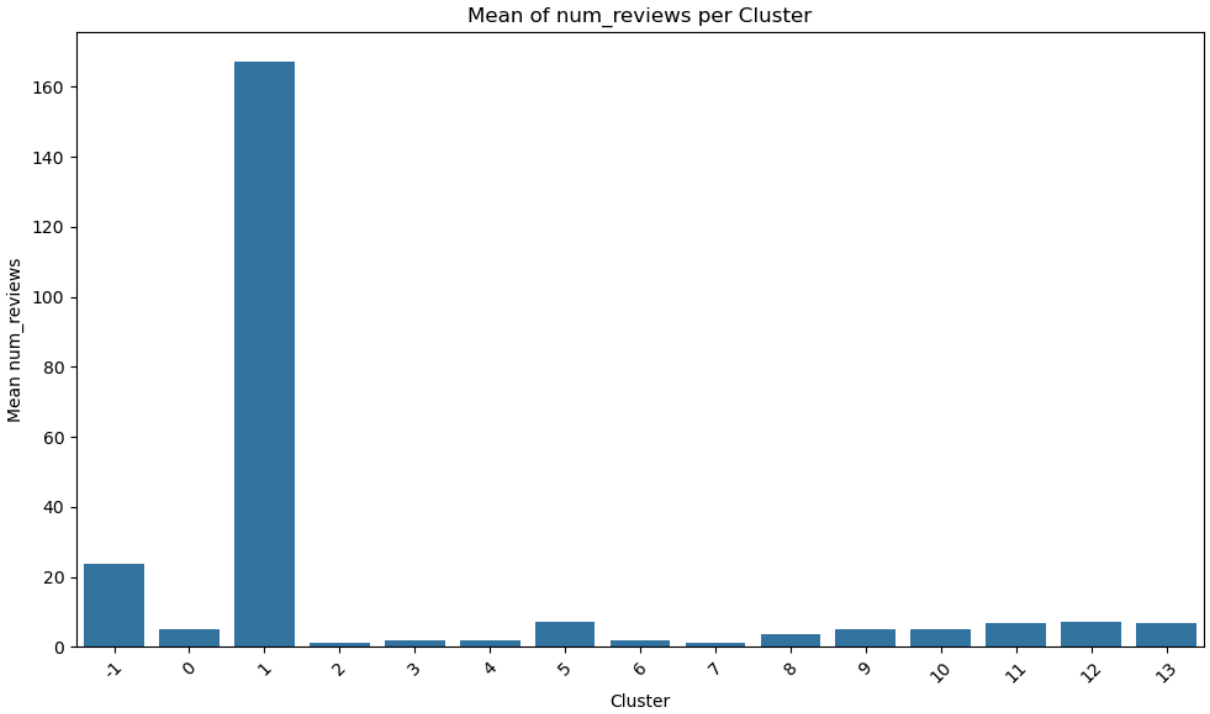
In [ ]:
```
# Assuming 'cluster' is the column representing cluster labels
for cluster_id in df['hdbscan_cluster'].unique():
    cluster_data = df[df['hdbscan_cluster'] == cluster_id]
    print(f"Cluster {cluster_id}:")
    print(cluster_data.describe())
```

In [67]:
```
# Specify the column you want to analyze
column_name = 'total_game_hours'  # Replace with the column you're intereste

# Group by clusters and calculate the mean for the specified column
cluster_means = df.groupby('hdbscan_cluster')[column_name].mean()

# Plot the bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x=cluster_means.index, y=cluster_means.values)

# Customize the plot
plt.title(f'Mean of {column_name} per Cluster')
plt.xlabel('Cluster')
plt.ylabel(f'Mean {column_name}')
plt.xticks(rotation=45)
plt.tight_layout()
```

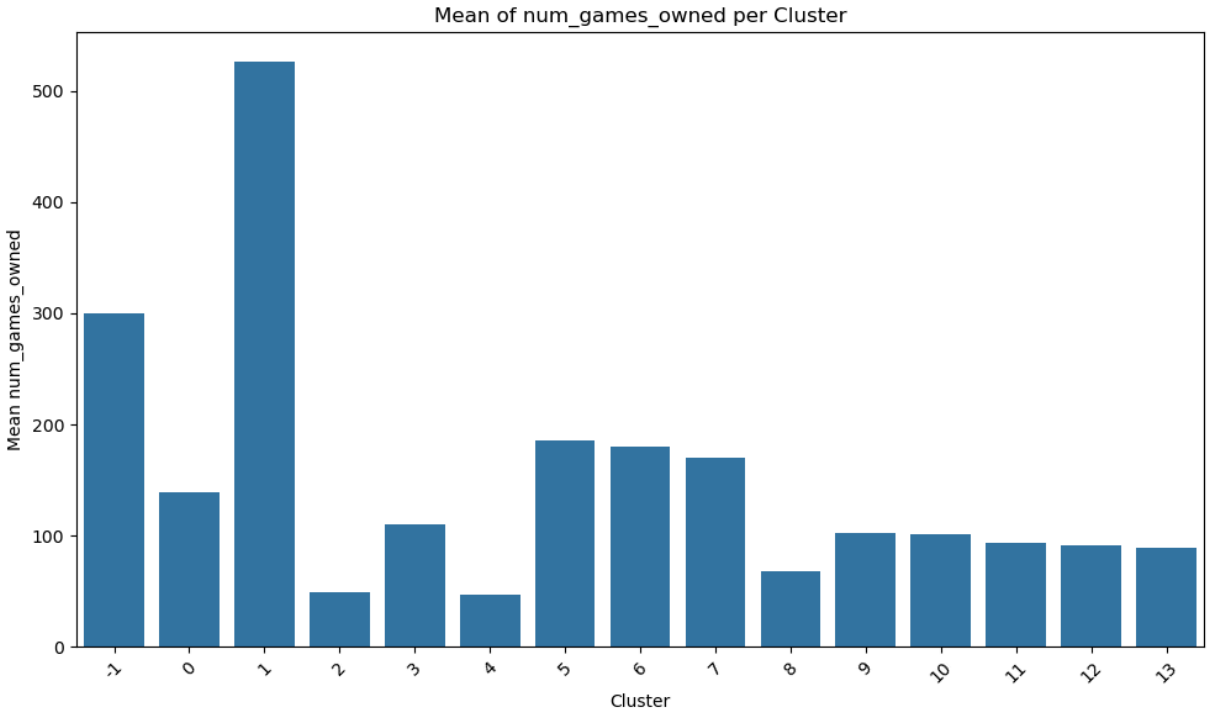
```
# Show the plot
plt.show()
```



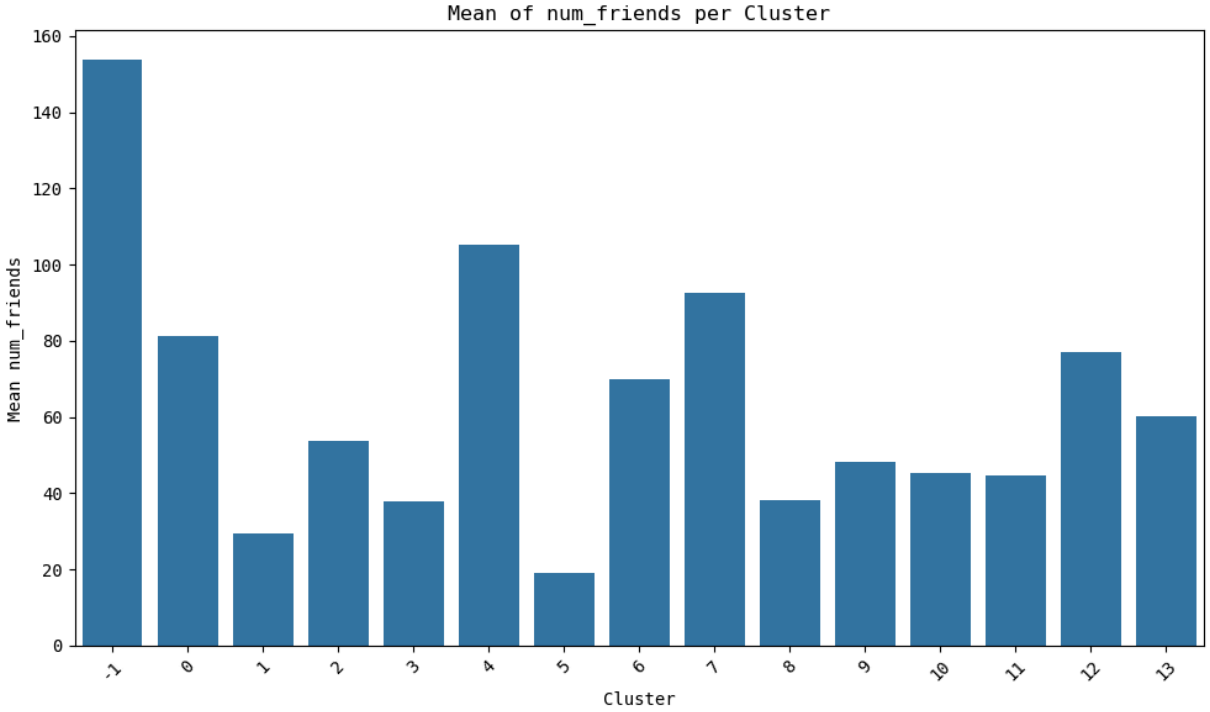
In [74]:
```
# Specify the column you want to analyze
column_name = 'num_reviews'  # Replace with the column you're interested in

# Group by clusters and calculate the mean for the specified column
cluster_means = df.groupby('hdbscan_cluster')[column_name].mean()

# Plot the bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x=cluster_means.index, y=cluster_means.values)

# Customize the plot
plt.title(f'Mean of {column_name} per Cluster')
plt.xlabel('Cluster')
plt.ylabel(f'Mean {column_name}')
plt.xticks(rotation=45)
plt.tight_layout()

# Show the plot
plt.show()
```



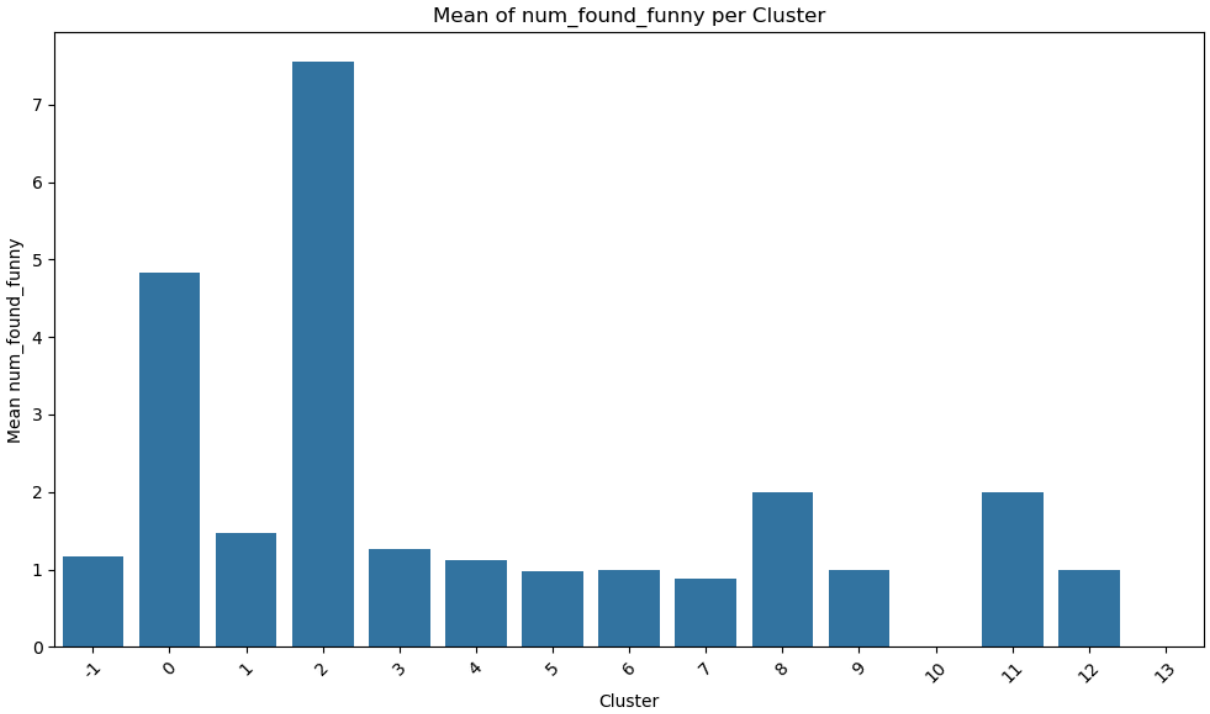
In [73]:
```
# Specify the column you want to analyze
column_name = 'num_games_owned'  # Replace with the column you're interested

# Group by clusters and calculate the mean for the specified column
cluster_means = df.groupby('hdbscan_cluster')[column_name].mean()

# Plot the bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x=cluster_means.index, y=cluster_means.values)

# Customize the plot
plt.title(f'Mean of {column_name} per Cluster')
plt.xlabel('Cluster')
plt.ylabel(f'Mean {column_name}')
plt.xticks(rotation=45)
plt.tight_layout()

# Show the plot
plt.show()
```



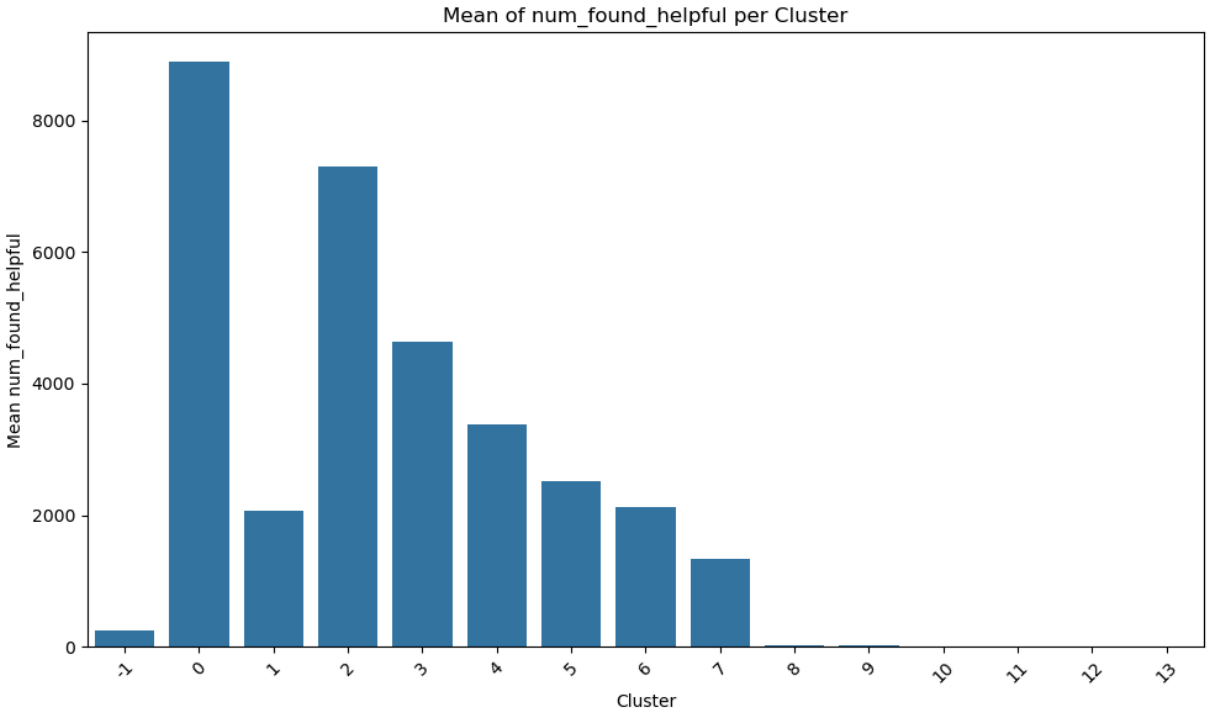
In [93]:
```
# Specify the column you want to analyze
column_name = 'num_friends'  # Replace with the column you're interested in

# Group by clusters and calculate the mean for the specified column
cluster_means = df.groupby('hdbscan_cluster')[column_name].mean()

# Plot the bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x=cluster_means.index, y=cluster_means.values)

# Customize the plot
plt.title(f'Mean of {column_name} per Cluster')
plt.xlabel('Cluster')
plt.ylabel(f'Mean {column_name}')
plt.xticks(rotation=45)
plt.tight_layout()

# Show the plot
plt.show()
```



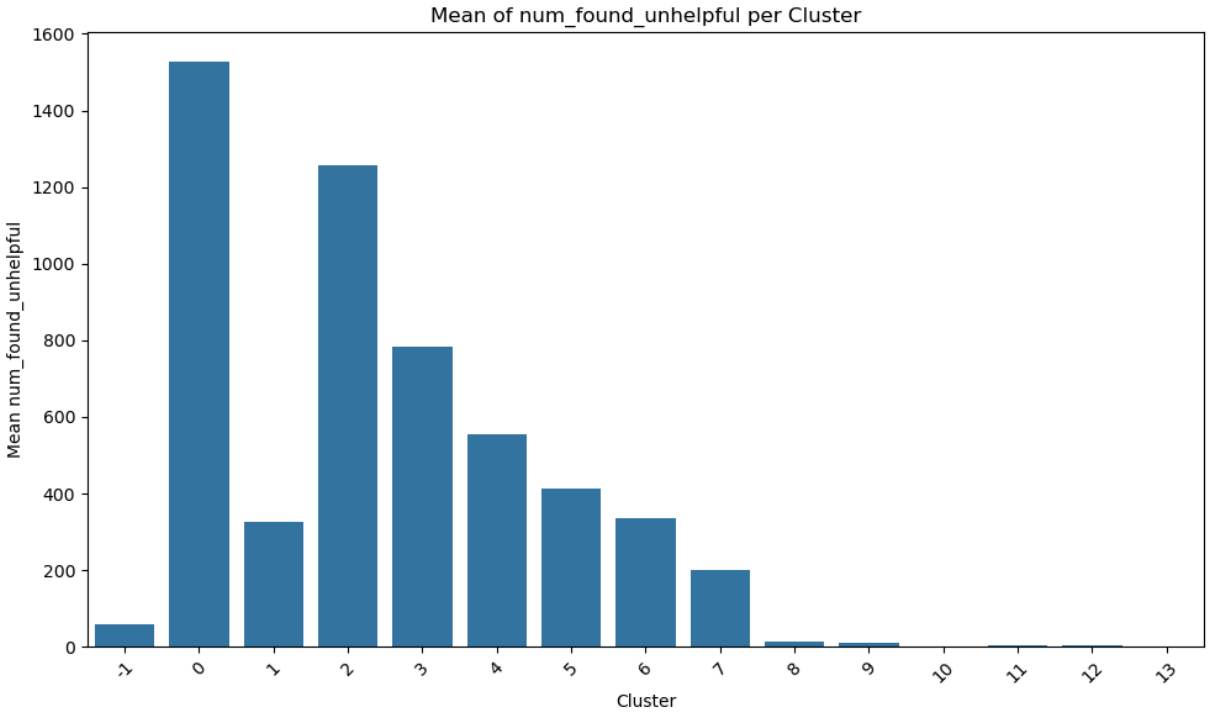
In [72]:
```
# Specify the column you want to analyze
column_name = 'num_found_funny'  # Replace with the column you're interested

# Group by clusters and calculate the mean for the specified column
cluster_means = df.groupby('hdbscan_cluster')[column_name].mean()

# Plot the bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x=cluster_means.index, y=cluster_means.values)

# Customize the plot
plt.title(f'Mean of {column_name} per Cluster')
plt.xlabel('Cluster')
plt.ylabel(f'Mean {column_name}')
plt.xticks(rotation=45)
plt.tight_layout()

# Show the plot
plt.show()
```



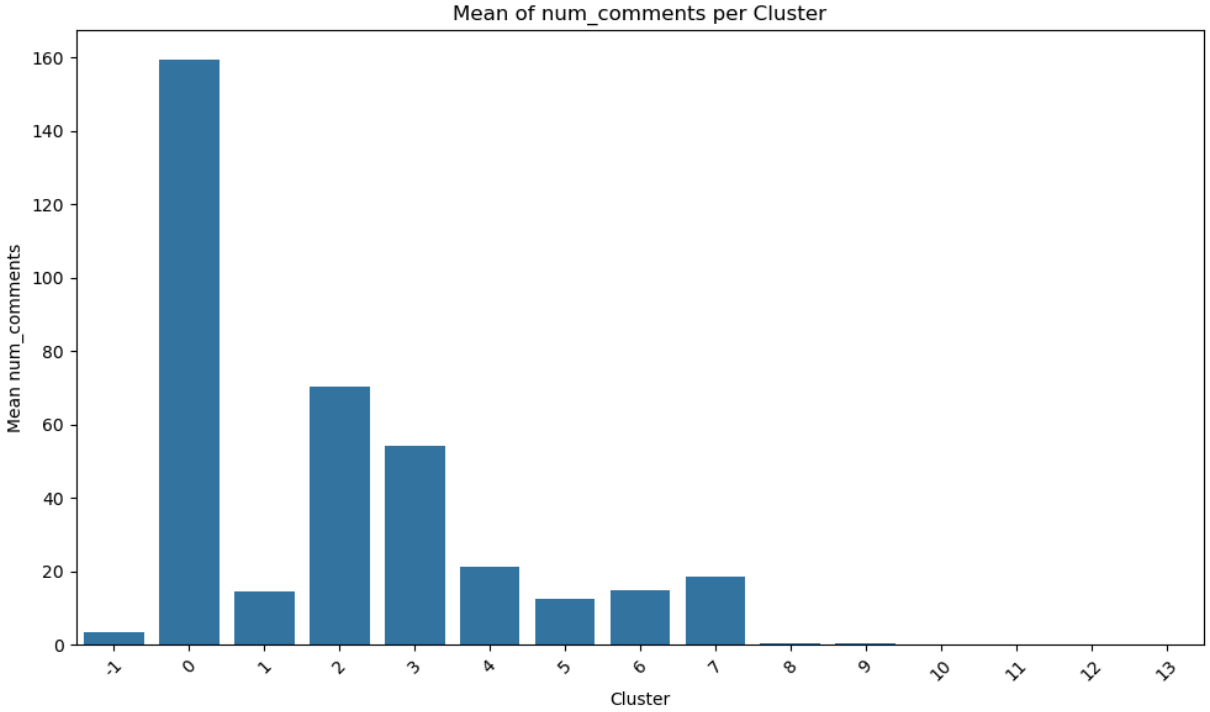
In [68]:
```
# Specify the column you want to analyze
column_name = 'num_found_helpful'  # Replace with the column you're interest

# Group by clusters and calculate the mean for the specified column
cluster_means = df.groupby('hdbscan_cluster')[column_name].mean()

# Plot the bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x=cluster_means.index, y=cluster_means.values)

# Customize the plot
plt.title(f'Mean of {column_name} per Cluster')
plt.xlabel('Cluster')
plt.ylabel(f'Mean {column_name}')
plt.xticks(rotation=45)
plt.tight_layout()

# Show the plot
plt.show()
```



In [71]:
```
# Specify the column you want to analyze
column_name = 'num_found_unhelpful'  # Replace with the column you're intere

# Group by clusters and calculate the mean for the specified column
cluster_means = df.groupby('hdbscan_cluster')[column_name].mean()

# Plot the bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x=cluster_means.index, y=cluster_means.values)

# Customize the plot
plt.title(f'Mean of {column_name} per Cluster')
plt.xlabel('Cluster')
plt.ylabel(f'Mean {column_name}')
plt.xticks(rotation=45)
plt.tight_layout()

# Show the plot
plt.show()
```



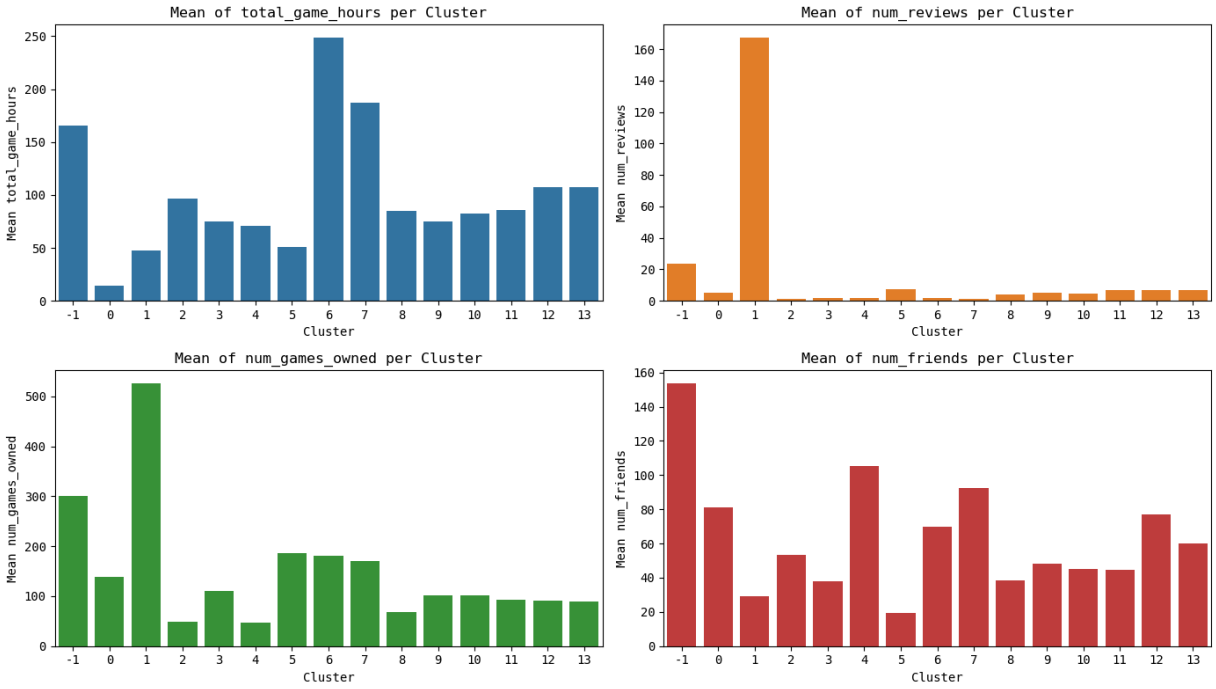
In [69]:
```
# Specify the column you want to analyze
column_name = 'num_comments'  # Replace with the column you're interested in

# Group by clusters and calculate the mean for the specified column
cluster_means = df.groupby('hdbscan_cluster')[column_name].mean()

# Plot the bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x=cluster_means.index, y=cluster_means.values)

# Customize the plot
plt.title(f'Mean of {column_name} per Cluster')
plt.xlabel('Cluster')
plt.ylabel(f'Mean {column_name}')
plt.xticks(rotation=45)
plt.tight_layout()

# Show the plot
plt.show()
```



In [107…

```
import matplotlib.pyplot as plt
import seaborn as sns

# Specify the columns you want to analyze
columns = ['total_game_hours', 'num_reviews', 'num_games_owned', 'num_friend

# Create a list of colors to be used for each subplot
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728']  # You can change thes

# Set the font to monospace
plt.rc('font', family='monospace')

# Create subplots (2 rows, 2 columns)
fig, axes = plt.subplots(2, 2, figsize=(14, 8))

# Flatten the axes for easy iteration
axes = axes.flatten()

# Loop over columns and create a bar plot for each
for idx, (column_name, color) in enumerate(zip(columns, colors)):
    # Group by clusters and calculate the mean for the specified column
    cluster_means = df.groupby('hdbscan_cluster')[column_name].mean()

    # Create a bar plot in the corresponding subplot
    sns.barplot(x=cluster_means.index, y=cluster_means.values, ax=axes[idx],

    # Customize the plot
    axes[idx].set_title(f'Mean of {column_name} per Cluster')
    axes[idx].set_xlabel('Cluster')
    axes[idx].set_ylabel(f'Mean {column_name}')
    axes[idx].tick_params(axis='x', rotation=0)  # Rotate x-axis labels

# Adjust layout to avoid overlap
```

```
plt.tight_layout()

# Show the combined plot
plt.show()
plt.savefig('user_stats.png')
```

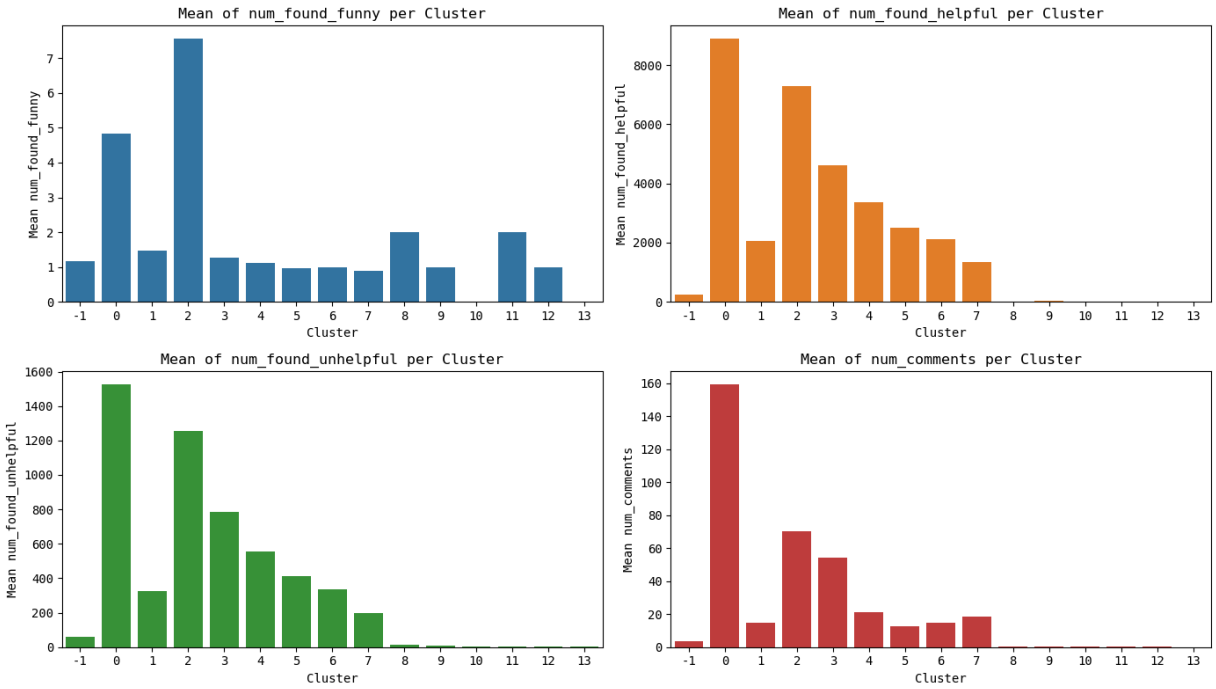


<Figure size 640x480 with 0 Axes>

In [108…

```
import matplotlib.pyplot as plt
import seaborn as sns

# Specify the columns you want to analyze
columns = ['num_found_funny', 'num_found_helpful', 'num_found_unhelpful',
        'num_comments']  # Add your columns here

# Create a list of colors to be used for each subplot
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728']  # You can change thes

# Set the font to monospace
plt.rc('font', family='monospace')

# Create subplots (2 rows, 2 columns)
fig, axes = plt.subplots(2, 2, figsize=(14, 8))

# Flatten the axes for easy iteration
axes = axes.flatten()

# Loop over columns and create a bar plot for each
for idx, (column_name, color) in enumerate(zip(columns, colors)):
    # Group by clusters and calculate the mean for the specified column
    cluster_means = df.groupby('hdbscan_cluster')[column_name].mean()

    # Create a bar plot in the corresponding subplot
    sns.barplot(x=cluster_means.index, y=cluster_means.values, ax=axes[idx],

    # Customize the plot
```

```
    axes[idx].set_title(f'Mean of {column_name} per Cluster')
    axes[idx].set_xlabel('Cluster')
    axes[idx].set_ylabel(f'Mean {column_name}')
    axes[idx].tick_params(axis='x', rotation=0)  # Rotate x-axis labels

# Adjust layout to avoid overlap
plt.tight_layout()

# Show the combined plot
plt.show()
plt.savefig('review_stats.png')
```

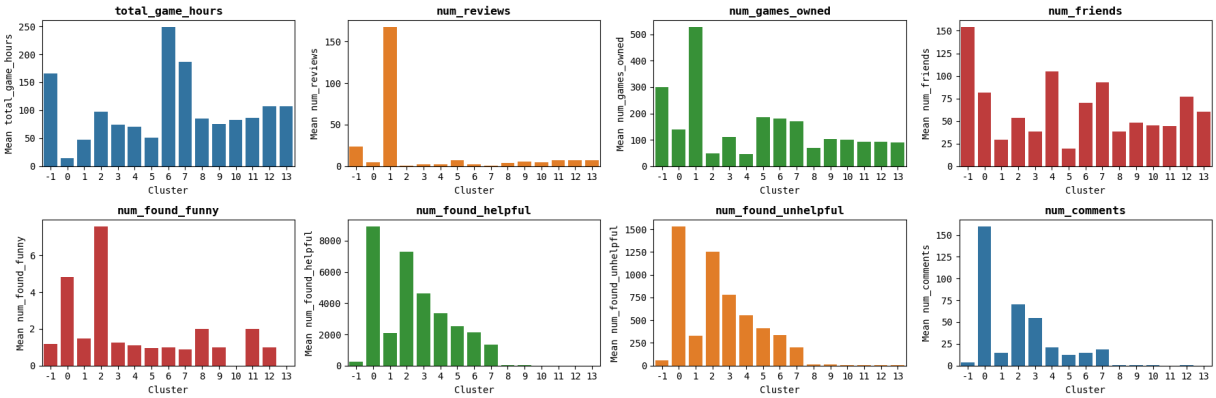


<Figure size 640x480 with 0 Axes>

In [158…

```
import matplotlib.pyplot as plt
import seaborn as sns

# Specify the columns you want to analyze
columns = ['total_game_hours', 'num_reviews', 'num_games_owned', 'num_frienc
        'num_comments']  # Add your columns here

# Create a list of colors to be used for each subplot
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#d62728',  '#2ca02c',

# Set the font to monospace
plt.rc('font', family='monospace')

# Create subplots (2 rows, 2 columns)
fig, axes = plt.subplots(2, 4, figsize=(18, 6))

# Flatten the axes for easy iteration
axes = axes.flatten()

# Loop over columns and create a bar plot for each
for idx, (column_name, color) in enumerate(zip(columns, colors)):
    # Group by clusters and calculate the mean for the specified column
```

```
    cluster_means = df.groupby('hdbscan_cluster')[column_name].mean()

    # Create a bar plot in the corresponding subplot
    sns.barplot(x=cluster_means.index, y=cluster_means.values, ax=axes[idx],

    # Customize the plot
    axes[idx].set_title(f'{column_name}', fontweight='bold')
    axes[idx].set_xlabel('Cluster')
    axes[idx].set_ylabel(f'Mean {column_name}')
    axes[idx].tick_params(axis='x', rotation=0)  # Rotate x-axis labels

# Adjust layout to avoid overlap
plt.tight_layout()

# Show the combined plot
plt.show()
plt.savefig('user_stats.png')
```




```
<Figure size 640x480 with 0 Axes>
```

SQLite Database for easy querying

In [30]:
```
# Creating a SQLite database
dbfile = os.path.join('database.sqlite')
db = sqlite3.connect(dbfile)
df.to_sql('gta5_reviews', db, if_exists='replace', index=False)
```

Out[30]:
```
8389
```

In [31]:
```
df.columns
```

Out[31]:
```
Index(['num_found_funny', 'num_guides', 'total_game_hours',
       'num_workshop_items', 'num_found_unhelpful', 'num_found_helpful',
       'total_game_hours_last_two_weeks', 'num_comments', 'rating',
       'num_reviews', 'num_groups', 'num_games_owned', 'friend_player_leve
l',
       'found_helpful_percentage', 'num_screenshots', 'review',
       'num_voted_helpfulness', 'num_badges', 'num_friends',
       'num_achievements_percentage', 'cluster', 'hdbscan_cluster'],
      dtype='object')
```

In [32]:
```
query = """
SELECT
```

```
    SUM(CASE WHEN rating = 1 THEN 1 ELSE 0 END) AS count_rating_1,
    SUM(CASE WHEN rating = 0 THEN 1 ELSE 0 END) AS count_rating_0
FROM gta5_reviews
WHERE hdbscan_cluster = 6;

"""
# saving the query result in a pandas df
df_query = pd.read_sql(query, db)
df_query
```

Out[32]:

	count_rating_1	count_rating_0
0	0	29

In [34]:

```
query = """
SELECT
    hdbscan_cluster,
    SUM(CASE WHEN rating = 1 THEN 1 ELSE 0 END) AS Recommended,
    SUM(CASE WHEN rating = 0 THEN 1 ELSE 0 END) AS Not_Recommended
FROM gta5_reviews
GROUP BY hdbscan_cluster;
"""
# saving the query result in a pandas df
df_query = pd.read_sql(query, db)
df_query
```

Out[34]:

	hdbscan_cluster	Recommended	Not_Recommended
0	-1	1087	659
1	0	0	70
2	1	0	62
3	2	0	45
4	3	0	72
5	4	0	51
6	5	0	72
7	6	0	29
8	7	0	33
9	8	0	39
10	9	0	638
11	10	0	1340
12	11	31	0
13	12	1072	0
14	13	3089	0

In [ ]:
```
# Save the dataframe to a CSV file
df_query.to_csv('cluster_recommendations.csv', index=False)
```

In [152…
```
query = """
SELECT
    hdbscan_cluster, AVG(num_games_owned) AS total_games_owned
FROM gta5_reviews
GROUP BY hdbscan_cluster;
"""
# saving the query result in a pandas df
df_query = pd.read_sql(query, db)
df_query
# Save the dataframe to a CSV file
#df_query.to_csv('cluster_recommendations.csv', index=False)
```

Out[152…

	hdbscan_cluster	total_games_owned
0	-1	300.339633
1	0	138.828571
2	1	526.387097
3	2	49.000000
4	3	110.000000
5	4	46.647059
6	5	185.500000
7	6	180.413793
8	7	170.000000
9	8	68.051282
10	9	102.456113
11	10	100.864925
12	11	93.193548
13	12	91.233209
14	13	89.291033

In [120…
```
import matplotlib.pyplot as plt
from matplotlib import cm

# List all available colormaps
print(sorted(cm._colormaps))
```

```
['Accent', 'Accent_r', 'Blues', 'Blues_r', 'BrBG', 'BrBG_r', 'BuGn', 'BuGn_
r', 'BuPu', 'BuPu_r', 'CMRmap', 'CMRmap_r', 'Dark2', 'Dark2_r', 'GnBu', 'GnB
u_r', 'Grays', 'Greens', 'Greens_r', 'Greys', 'Greys_r', 'OrRd', 'OrRd_r',
'Oranges', 'Oranges_r', 'PRGn', 'PRGn_r', 'Paired', 'Paired_r', 'Pastel1',
'Pastel1_r', 'Pastel2', 'Pastel2_r', 'PiYG', 'PiYG_r', 'PuBu', 'PuBuGn', 'Pu
BuGn_r', 'PuBu_r', 'PuOr', 'PuOr_r', 'PuRd', 'PuRd_r', 'Purples', 'Purples_
r', 'RdBu', 'RdBu_r', 'RdGy', 'RdGy_r', 'RdPu', 'RdPu_r', 'RdYlBu', 'RdYlBu_
r', 'RdYlGn', 'RdYlGn_r', 'Reds', 'Reds_r', 'Set1', 'Set1_r', 'Set2', 'Set2_
r', 'Set3', 'Set3_r', 'Spectral', 'Spectral_r', 'Wistia', 'Wistia_r', 'YlG
n', 'YlGnBu', 'YlGnBu_r', 'YlGn_r', 'YlOrBr', 'YlOrBr_r', 'YlOrRd', 'YlOrRd_
r', 'afmhot', 'afmhot_r', 'autumn', 'autumn_r', 'binary', 'binary_r', 'bon
e', 'bone_r', 'brg', 'brg_r', 'bwr', 'bwr_r', 'cividis', 'cividis_r', 'coo
l', 'cool_r', 'coolwarm', 'coolwarm_r', 'copper', 'copper_r', 'crest', 'cres
t_r', 'cubehelix', 'cubehelix_r', 'flag', 'flag_r', 'flare', 'flare_r', 'gis
t_earth', 'gist_earth_r', 'gist_gray', 'gist_gray_r', 'gist_grey', 'gist_hea
t', 'gist_heat_r', 'gist_ncar', 'gist_ncar_r', 'gist_rainbow', 'gist_rainbow
_r', 'gist_stern', 'gist_stern_r', 'gist_yarg', 'gist_yarg_r', 'gist_yerg',
'gnuplot', 'gnuplot2', 'gnuplot2_r', 'gnuplot_r', 'gray', 'gray_r', 'grey',
'hot', 'hot_r', 'hsv', 'hsv_r', 'icefire', 'icefire_r', 'inferno', 'inferno_
r', 'jet', 'jet_r', 'magma', 'magma_r', 'mako', 'mako_r', 'nipy_spectral',
'nipy_spectral_r', 'ocean', 'ocean_r', 'pink', 'pink_r', 'plasma', 'plasma_
r', 'prism', 'prism_r', 'rainbow', 'rainbow_r', 'rocket', 'rocket_r', 'seism
ic', 'seismic_r', 'spring', 'spring_r', 'summer', 'summer_r', 'tab10', 'tab1
0_r', 'tab20', 'tab20_r', 'tab20b', 'tab20b_r', 'tab20c', 'tab20c_r', 'terra
in', 'terrain_r', 'turbo', 'turbo_r', 'twilight', 'twilight_r', 'twilight_sh
ifted', 'twilight_shifted_r', 'viridis', 'viridis_r', 'vlag', 'vlag_r', 'win
ter', 'winter_r']
```