This notebook contains codes executed for Assignment-3 (**Descriptive Data Mining**) for the course **CSIT558_01SP25** by:

- Chaudhari, Amishaben Natavarbhai (CWID:50129394)
- Javed, Syed Mehrose (CWID: 50128848)
- Subhani, Muhammad Kamal (CWID: 50136417)
- Thangavel Sathiyamoorthy, Punithan (CWID: 50135877)
- Venkata Appala Manoj Muvvala (CWID: 50122981)


```
# Importing the required libaries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA

from google.colab import files

# Uploading the file
dataset = files.upload()

<IPython.core.display.HTML object>

Saving Spotify_Youtube.csv to Spotify_Youtube.csv

# Loading the data into a dataframe
df = pd.read_csv("Spotify_Youtube.csv")

df.shape

(20718, 28)

print(list(df.columns))

['Unnamed: 0', 'Artist', 'Url_spotify', 'Track', 'Album',
'Album_type', 'Uri', 'Danceability', 'Energy', 'Key', 'Loudness',
'Speechiness', 'Acousticness', 'Instrumentalness', 'Liveness',
'Valence', 'Tempo', 'Duration_ms', 'Url_youtube', 'Title', 'Channel',
'Views', 'Likes', 'Comments', 'Description', 'Licensed',
'official_video', 'Stream']

df.head()

{"type":"dataframe","variable_name":"df"}

# Checking the missing values
print(df.isnull().sum())
```

```
# Very few records with null values
# Removing missing values
df = df.dropna()
```

```
Unnamed: 0             0
Artist                 0
Url_spotify            0
Track                  0
Album                  0
Album_type             0
Uri                    0
Danceability           2
Energy                 2
Key                    2
Loudness               2
Speechiness            2
Acousticness           2
Instrumentalness       2
Liveness               2
Valence                2
Tempo                  2
Duration_ms            2
Url_youtube          470
Title                470
Channel              470
Views                470
Likes                541
Comments             569
Description          876
Licensed             470
official_video       470
Stream               576
dtype: int64
```

```
print("Before dropping duplicates: ", df.shape)
# Dropping the duplicate rows
df.drop_duplicates(inplace=True)
print("After dropping duplicates: ", df.shape)
```

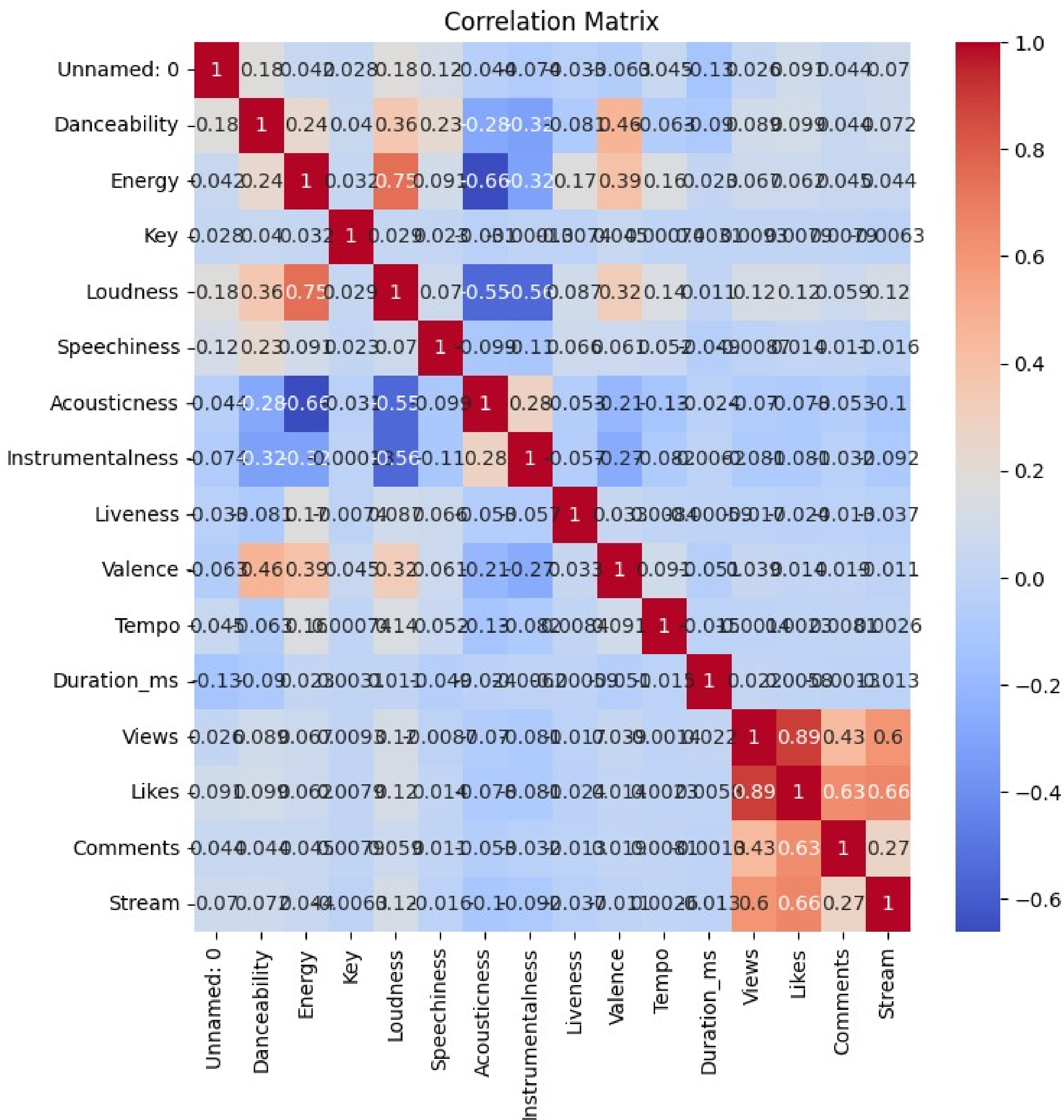
```
Before dropping duplicates:  (20718, 28)
After dropping duplicates:  (20718, 28)
```

```
# Selecting the numerical values to analyze the correlation
df_num = df.select_dtypes(include=["number"])
df_num.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 19170 entries, 0 to 20717
Data columns (total 16 columns):
```

```
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Unnamed: 0        19170 non-null  int64
 1   Danceability      19170 non-null  float64
 2   Energy            19170 non-null  float64
 3   Key               19170 non-null  float64
 4   Loudness          19170 non-null  float64
 5   Speechiness       19170 non-null  float64
 6   Acousticness      19170 non-null  float64
 7   Instrumentalness  19170 non-null  float64
 8   Liveness          19170 non-null  float64
 9   Valence           19170 non-null  float64
 10  Tempo             19170 non-null  float64
 11  Duration_ms       19170 non-null  float64
 12  Views             19170 non-null  float64
 13  Likes             19170 non-null  float64
 14  Comments          19170 non-null  float64
 15  Stream            19170 non-null  float64
dtypes: float64(15), int64(1)
memory usage: 2.5 MB
```

```python
# Analyrizing the correlation between the quantitative attributes
correlation_matrix = df_num.corr()
plt.figure(figsize=(8, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



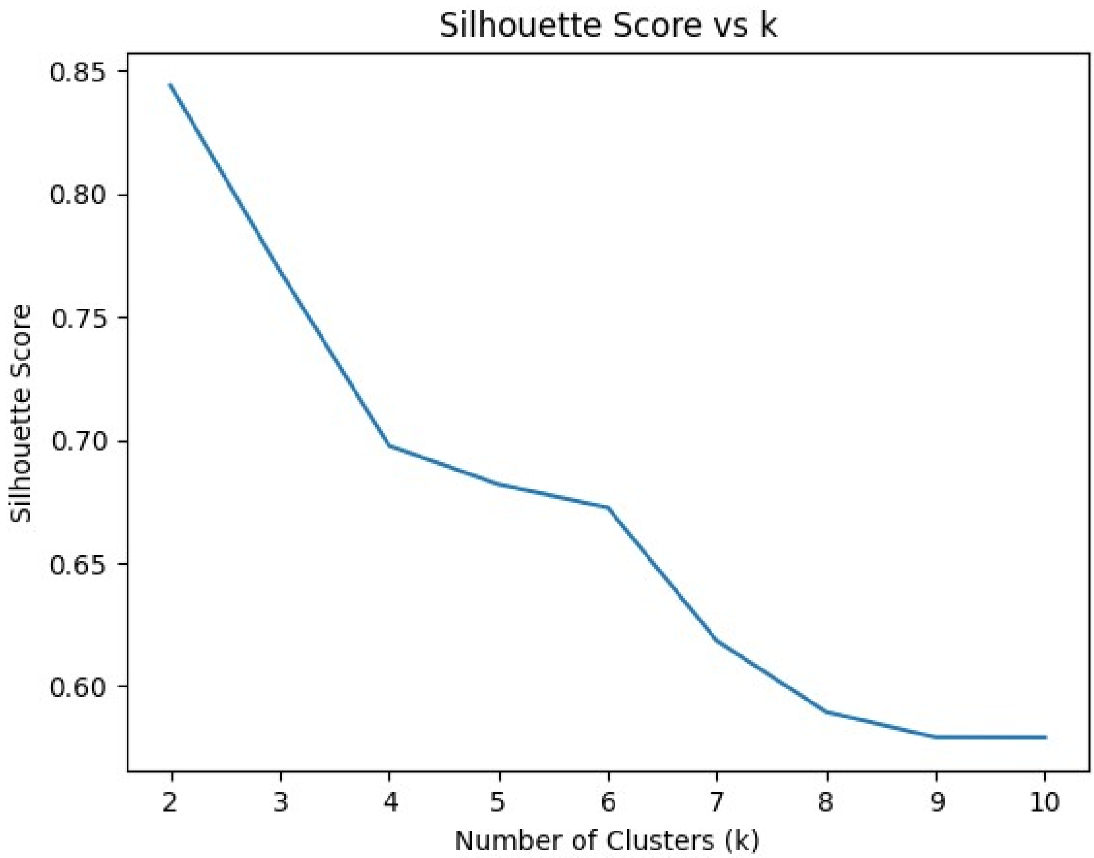

```python
# Selecting numerical features for clustering
features = ['Loudness', 'Energy', 'Danceability', 'Valence',
'Tempo','Speechiness',
            'Acousticness', 'Instrumentalness', 'Liveness', 'Views',
'Likes', 'Stream']

df_selected = df[features]
```

ALGORITHM EXECUTION

Value of k - Silhouette Scores

```
silhouette_list = []
# Checking for k between 2 and 10
k_range = range(2, 11)

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(df_selected)
    silhouette_list.append(silhouette_score(df_selected,
kmeans.labels_))

# Printing the silhouette scores
print(silhouette_list)

# Plotting the silhouette scores for different k values
plt.plot(k_range, silhouette_list)
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score vs k')
plt.show()
```

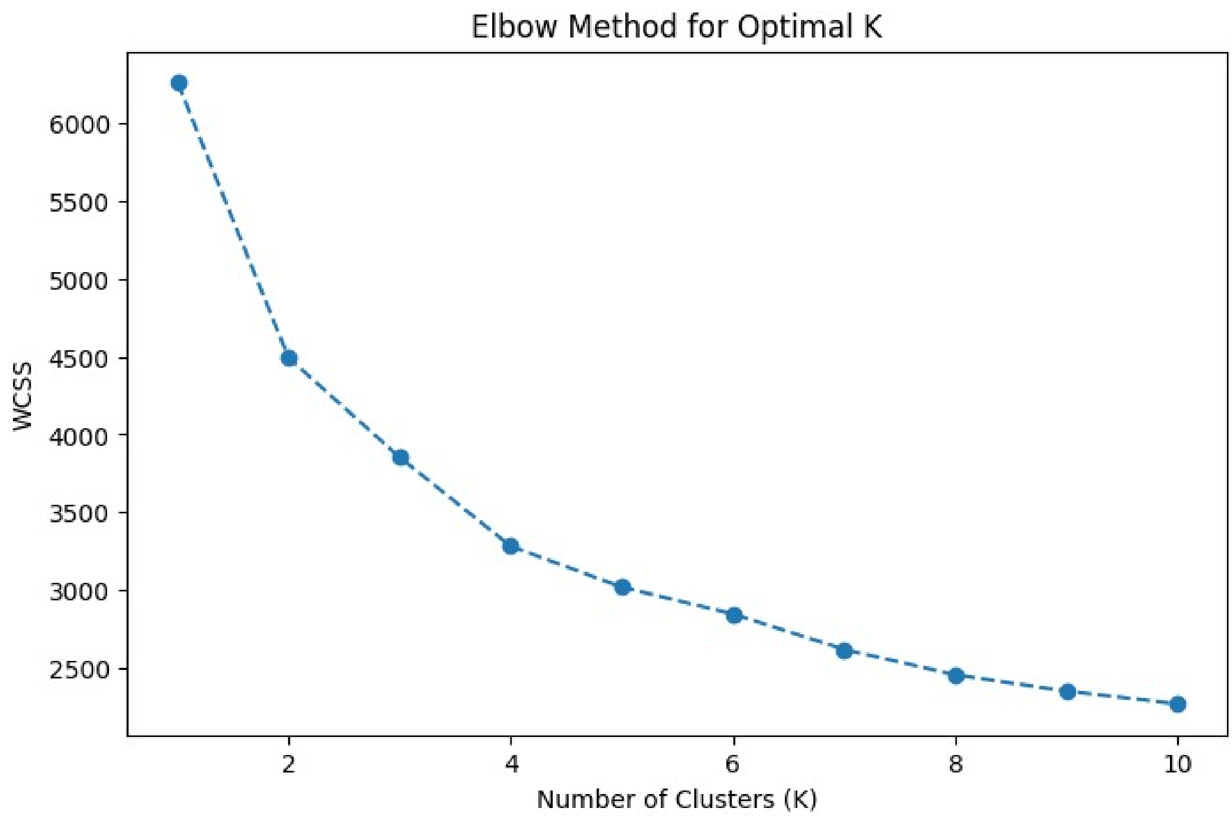
```
[0.843987884130402, 0.7686954086867869, 0.6975975600927531,
0.681949666560278, 0.6724521919555116, 0.618290878120718,
0.5894443917009909, 0.5791376172678511, 0.5790619254658479]
```



Value of k - Elbow Method

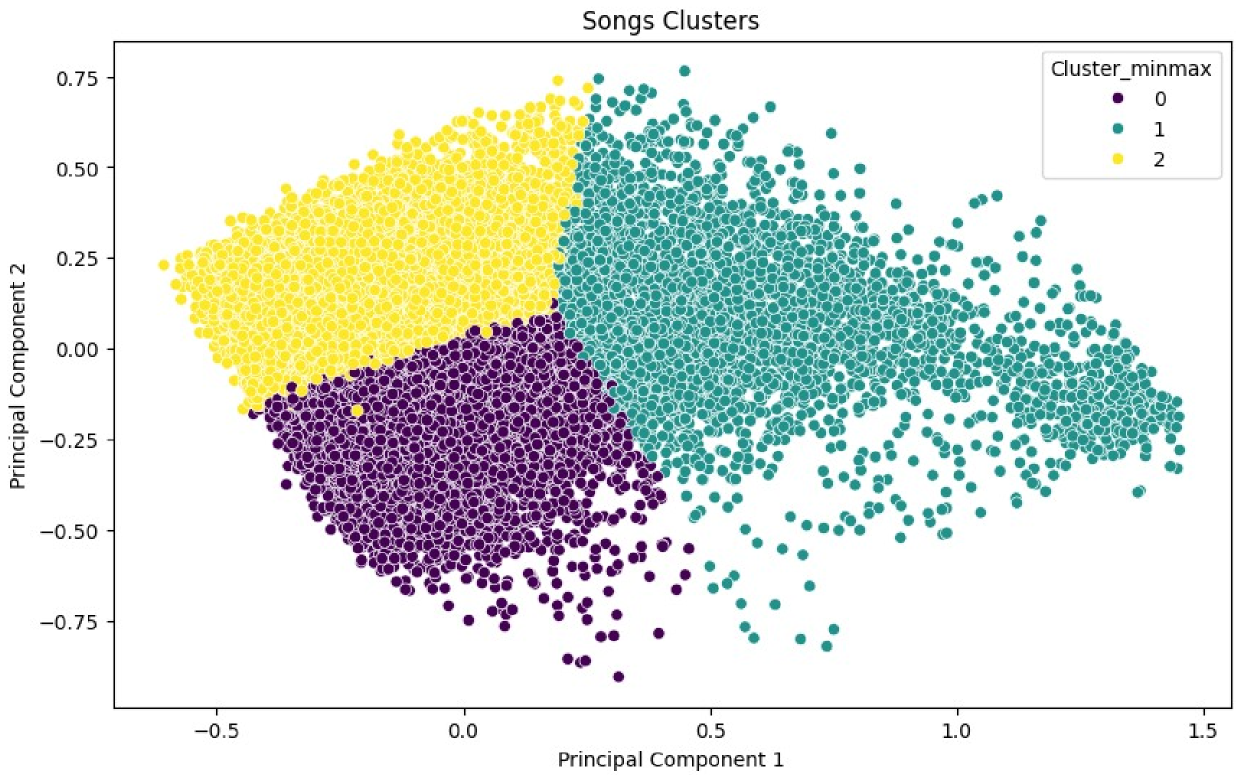
MinMaxScaler

```
# Bringing all the values of the selected features into a common
standard
scaler_minmax = MinMaxScaler()
df_scaled_minmax = scaler_minmax.fit_transform(df_selected)

wcss_minmax = []  # Within-cluster sum of squares
for k in range(1, 11):
    kmeans_minmax = KMeans(n_clusters=k, random_state=42)
    kmeans_minmax.fit(df_scaled_minmax)
    wcss_minmax.append(kmeans_minmax.inertia_)  # Inertia is the sum
of squared distances to the nearest centroid

# Plot the elbow curve
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), wcss_minmax, marker='o', linestyle='--')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS')
plt.title('Elbow Method for Optimal K')
plt.show()
```



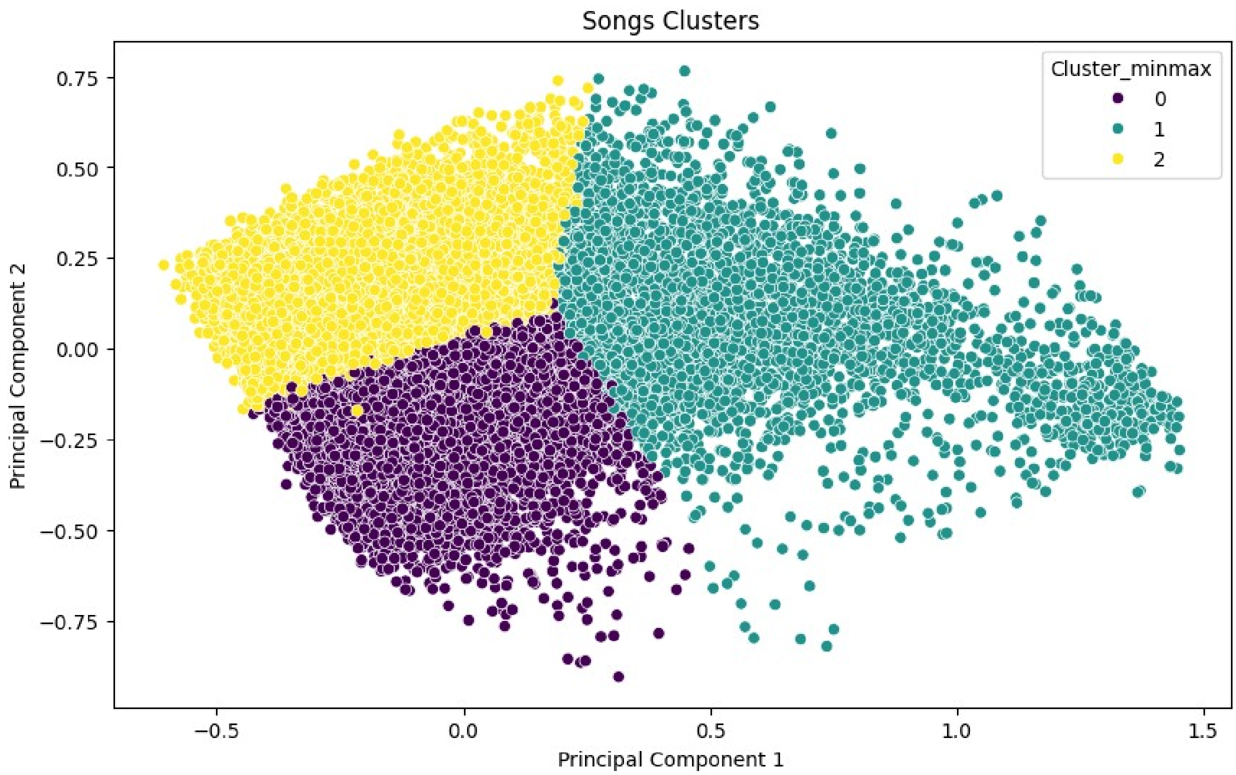
KMeans When k=3 and algorithm = 'elkan'

```
k = 3  # Update based on elbow method
algorithm='elkan' # Testing with 'elkan' algortihm

# Creating the K-Means clustering
kmeans_minmax = KMeans(n_clusters=k, algorithm=algorithm,
random_state=42)
df['Cluster_minmax'] = kmeans_minmax.fit_predict(df_scaled_minmax)

# PCA
pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_scaled_minmax)
df['PCA1'] = df_pca[:, 0]
df['PCA2'] = df_pca[:, 1]

# Visulaizing the clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x='PCA1', y='PCA2', hue='Cluster_minmax', data=df,
palette='viridis')
plt.title("Songs Clusters")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```



KMeans When k=3 and algorithm = 'llyod'

```
k = 3  # Update based on elbow method
algorithm='lloyd' # Testing with 'lloyd' algortihm

# Creating the K-Means clustering
kmeans_minmax = KMeans(n_clusters=k, algorithm=algorithm,
random_state=42)
df['Cluster_minmax'] = kmeans_minmax.fit_predict(df_scaled_minmax)

# PCA
pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_scaled_minmax)
df['PCA1'] = df_pca[:, 0]
df['PCA2'] = df_pca[:, 1]

# Visulaizing the clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x='PCA1', y='PCA2', hue='Cluster_minmax', data=df,
palette='viridis')
plt.title("Songs Clusters")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```



KMeans When k=4 and algorithm = 'elkan'

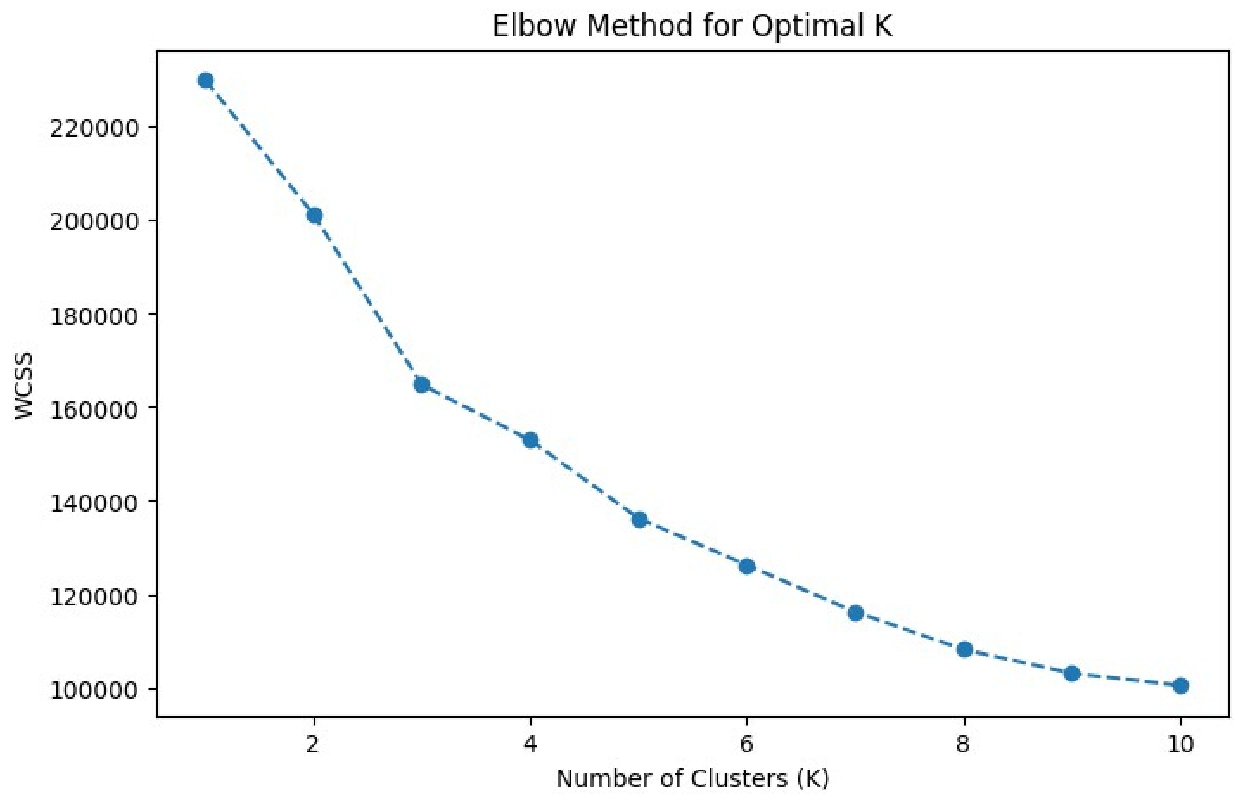
```
# Testing with k=4
k = 4
algorithm='elkan' # Testing with 'elkan' algortihm

# Creating the K-Means clustering
kmeans_minmax = KMeans(n_clusters=k, random_state=42)
df['Cluster_minmax'] = kmeans_minmax.fit_predict(df_scaled_minmax)

# Visualizing k=4
# PCA
pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_scaled_minmax)
df['PCA1'] = df_pca[:, 0]
df['PCA2'] = df_pca[:, 1]

# Visulaizing the clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x='PCA1', y='PCA2', hue='Cluster_minmax', data=df,
palette='viridis')
plt.title("Songs Clusters")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```

KMeans When k=3 and algorithm = 'llyod'

```
# Testing with k=4
k = 4
algorithm='lloyd' # Testing with 'lloyd' algortihm

# Creating the K-Means clustering
kmeans_minmax = KMeans(n_clusters=k, random_state=42)
df['Cluster_minmax'] = kmeans_minmax.fit_predict(df_scaled_minmax)

# Visualizing k=4
# PCA
pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_scaled_minmax)
df['PCA1'] = df_pca[:, 0]
df['PCA2'] = df_pca[:, 1]

# Visulaizing the clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x='PCA1', y='PCA2', hue='Cluster_minmax', data=df,
palette='viridis')
plt.title("Songs Clusters")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```

StandardScaler

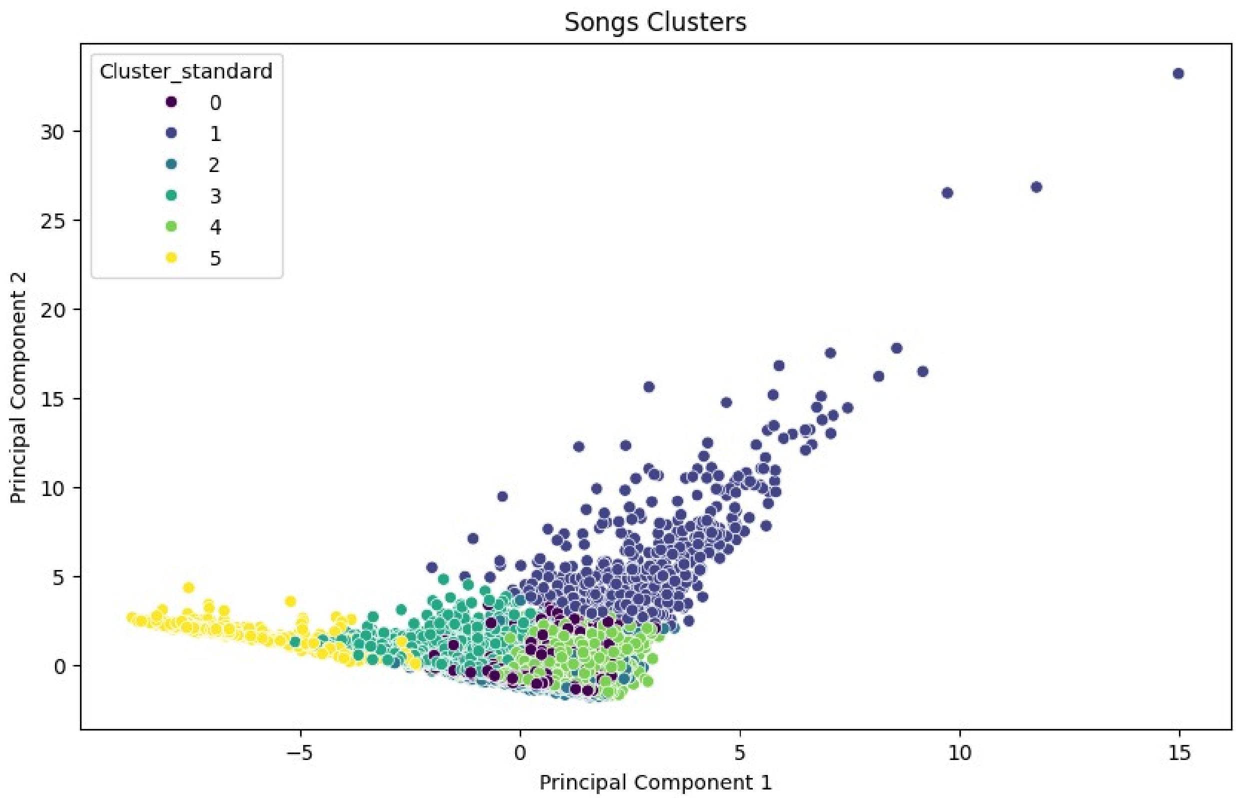
```
# Bringing all the values of the selected features into a common standard
scaler_standard = StandardScaler()
df_scaled_standard = scaler_standard.fit_transform(df_selected)

wcss_standard = []  # Within-cluster sum of squares
for k in range(1, 11):
    kmeans_standard = KMeans(n_clusters=k, random_state=42)
    kmeans_standard.fit(df_scaled_standard)
    wcss_standard.append(kmeans_standard.inertia_)  # Inertia is the sum of squared distances to the nearest centroid

# Plot the elbow curve
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), wcss_standard, marker='o', linestyle='--')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS')
plt.title('Elbow Method for Optimal K')
plt.show()
```



KMeans When k=3

```
#                    TEST
k = 6  # Update based on elbow method

# Creating the K-Means clustering
kmeans_standard = KMeans(n_clusters=k, random_state=42)
df['Cluster_standard'] =
kmeans_standard.fit_predict(df_scaled_standard)

k = 3  # Update based on elbow method

# Creating the K-Means clustering
kmeans_standard = KMeans(n_clusters=k, random_state=42)
df['Cluster_standard'] =
kmeans_standard.fit_predict(df_scaled_standard)

# PCA
pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_scaled_standard)
df['PCA1'] = df_pca[:, 0]
df['PCA2'] = df_pca[:, 1]

# Visulaizing the clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x='PCA1', y='PCA2', hue='Cluster_standard', data=df,
```

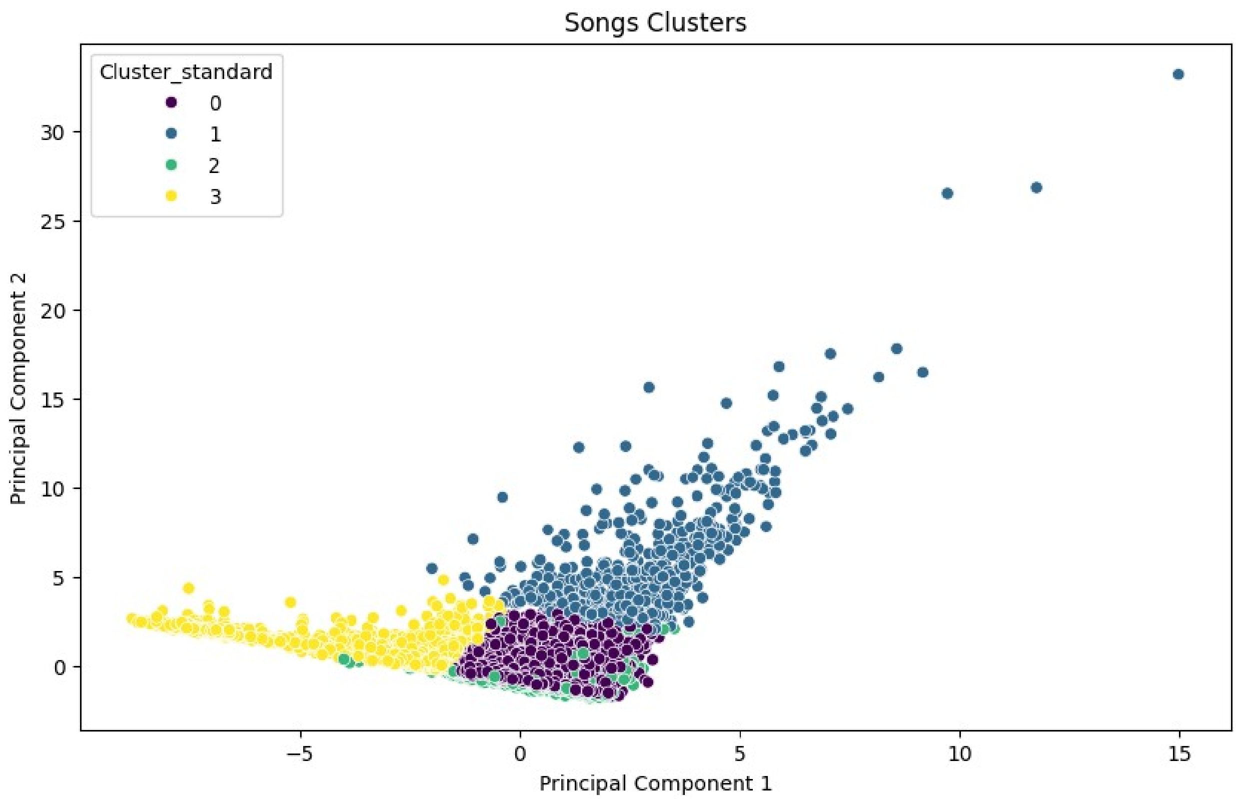
```
palette='viridis')
plt.title("Songs Clusters")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```



KMeans When k=4

```
# Testing k=4
k = 4

# Creating the K-Means clustering
kmeans_standard = KMeans(n_clusters=k, random_state=42)
df['Cluster_standard'] =
kmeans_standard.fit_predict(df_scaled_standard)

#Visualizing k=4
# PCA
pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_scaled_standard)
df['PCA1'] = df_pca[:, 0]
df['PCA2'] = df_pca[:, 1]

# Visulaizing the clusters
plt.figure(figsize=(10, 6))
```

```python
sns.scatterplot(x='PCA1', y='PCA2', hue='Cluster_standard', data=df,
palette='viridis')
plt.title("Songs Clusters")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```



ANALYSIS

MinMaxScaler Analysis

```python
# Analyzing the clusters through Mean
df.groupby("Cluster_minmax")[features].mean()
```

{"summary":"{\n \"name\": \"df\",\n \"rows\": 3,\n \"fields\": [\n
{\n   \"column\": \"Cluster_minmax\",\n   \"properties\": {\n
\"dtype\": \"int32\",\n   \"num_unique_values\": 3,\n
\"samples\": [\n   0,\n   1,\n   2\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Loudness\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
3.743019842598732,\n   \"min\": -12.830534926958832,\n
\"max\": -6.258135404315865,\n   \"num_unique_values\": 3,\n
\"samples\": [\n   -6.440584851527758,\n   -

```
12.830534926958832,\n          -6.258135404315865\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Energy\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.20683997931350273,\n        \"min\": 0.347898999814077,\n
\"max\": 0.7211589992885938,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.6889830727298809,\n
0.347898999814077,\n          0.7211589992885938\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Danceability\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.10423178376969981,\n        \"min\": 0.4969739176626826,\n
\"max\": 0.7048240455299977,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.587059274135705,\n
0.4969739176626826,\n          0.7048240455299977\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Valence\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.226767993446099,\n        \"min\": 0.34511065604249663,\n
\"max\": 0.74656153663742,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.3630819954095539,\n
0.34511065604249663,\n          0.74656153663742\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Tempo\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 5.561508175967272,\n
\"min\": 112.9524358565737,\n        \"max\": 123.11102955099699,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
123.11102955099699,\n          112.9524358565737,\n
121.95516872184018\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Speechiness\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.023294504332633657,\n        \"min\":
0.06272990703851261,\n        \"max\": 0.10660993597344083,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
0.0982275426768039,\n          0.06272990703851261,\n
0.10660993597344083\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Acousticness\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.33319397164761444,\n        \"min\":
0.12491309022234974,\n        \"max\": 0.7433287014900398,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
0.12491309022234974,\n          0.7433287014900398,\n
0.21911794928977946\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Instrumentalness\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.08820459029153568,\n        \"min\":
0.017096223883092246,\n        \"max\": 0.17827469157237716,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
0.035586541388609956,\n          0.17827469157237716,\n
```

0.017096223883092246\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Liveness\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0.020177175307000346,\n   \"min\":
0.16410788844621516,\n   \"max\": 0.2023756276000574,\n
\"num_unique_values\": 3,\n   \"samples\": [\n
0.2023756276000574,\n   0.16410788844621516,\n
0.19433411192791084\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Views\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 31359758.22461054,\n   \"min\": 53920413.45232404,\n
\"max\": 113310754.43644771,\n   \"num_unique_values\": 3,\n
\"samples\": [\n   101076793.9843638,\n
53920413.45232404,\n   113310754.43644771\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Likes\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\": 199401.9112288662,\n
\"min\": 404759.10464807437,\n   \"max\": 750828.494427318,\n
\"num_unique_values\": 3,\n   \"samples\": [\n
749433.9746090948,\n   404759.10464807437,\n
750828.494427318\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Stream\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 32505248.823813245,\n   \"min\": 97260257.17556441,\n
\"max\": 161640483.11217904,\n   \"num_unique_values\": 3,\n
\"samples\": [\n   161640483.11217904,\n
97260257.17556441,\n   137271078.84301636\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

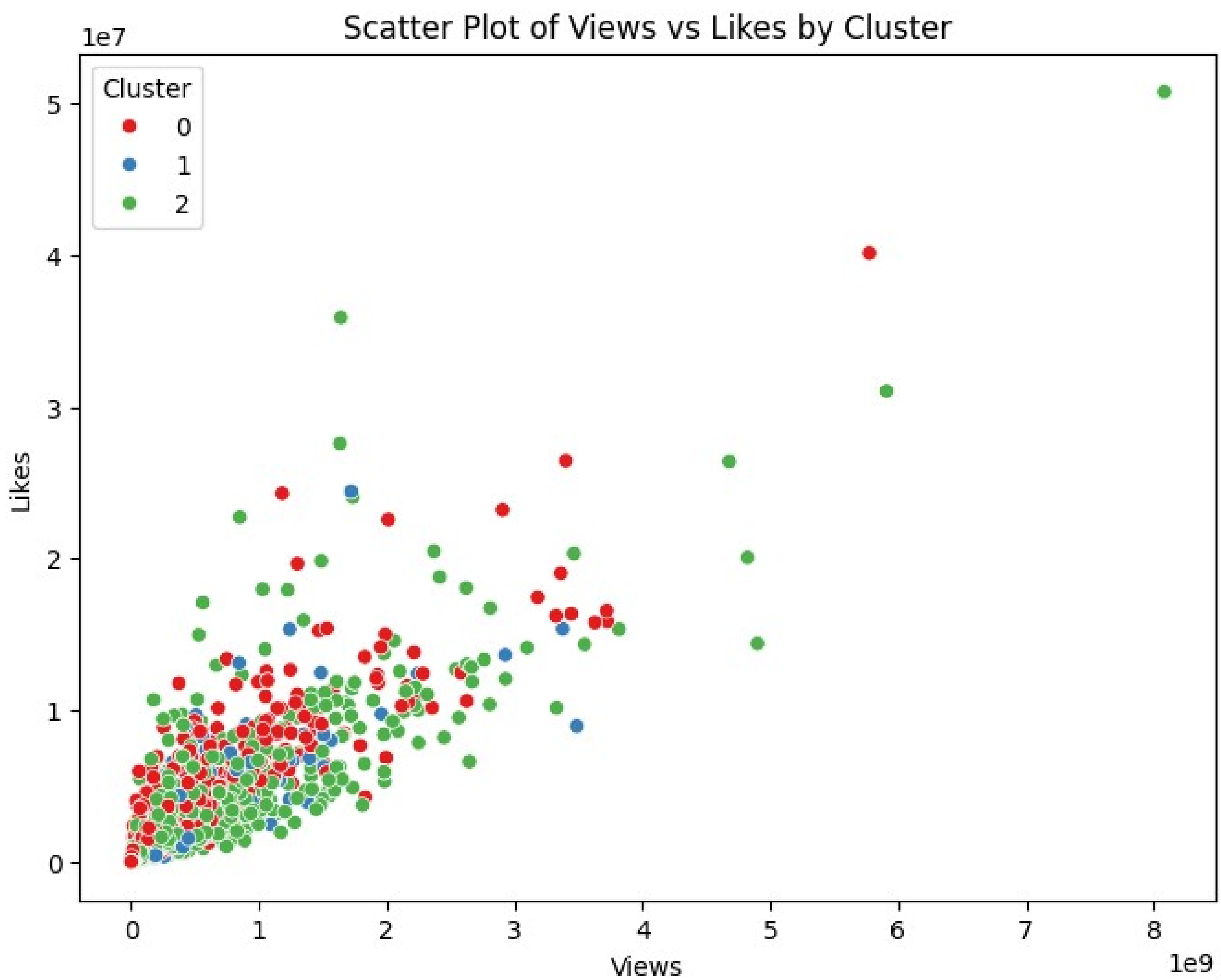
StandardScaler Analysis

```
# Analyzing the clusters through Mean
df.groupby("Cluster_standard")[features].mean()
```

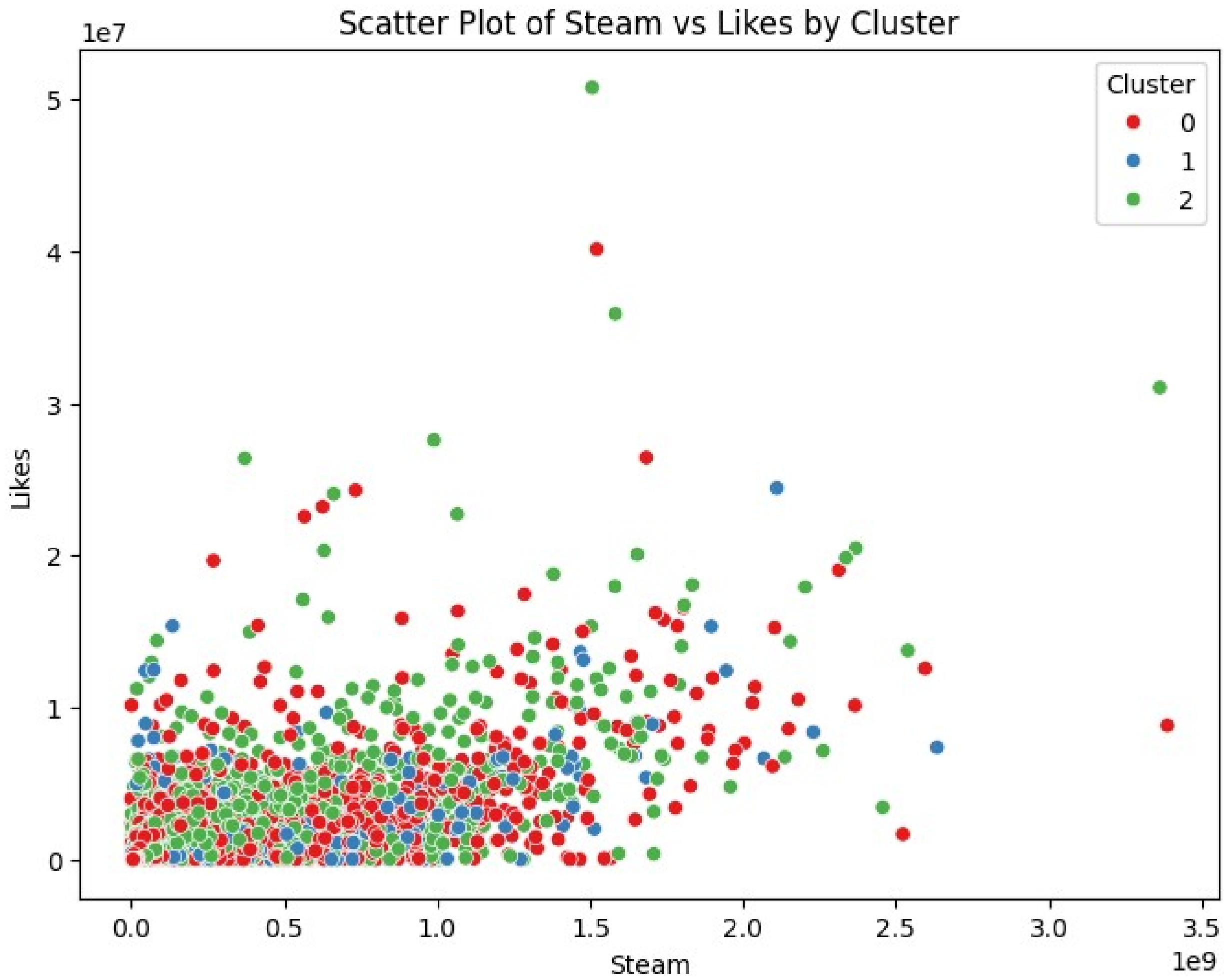
{"summary":"{\n \"name\": \"df\",\n \"rows\": 3,\n \"fields\": [\n
{\n   \"column\": \"Cluster_standard\",\n   \"properties\": {\n
\"dtype\": \"int32\",\n   \"num_unique_values\": 3,\n
\"samples\": [\n   0,\n   1,\n   2\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Loudness\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
4.745604364128136,\n   \"min\": -14.249233238004448,\n
\"max\": -5.750179586563307,\n   \"num_unique_values\": 3,\n
\"samples\": [\n   -14.249233238004448,\n   -
5.750179586563307,\n   -6.340902288674667\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Energy\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":

```
0.22113463065173242,\n        \"min\": 0.3068938145217668,\n
\"max\": 0.7020525936126959,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.3068938145217668,\n
0.6764869509043928,\n          0.7020525936126959\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Danceability\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.11895498414603495,\n        \"min\": 0.4601414045122339,\n
\"max\": 0.6782325581395349,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.4601414045122339,\n
0.6782325581395349,\n          0.6515159026821431\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Valence\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.14733088832635305,\n        \"min\": 0.3011736924054655,\n
\"max\": 0.5749202177191947,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.3011736924054655,\n
0.5324731266149871,\n          0.5749202177191947\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Tempo\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 6.025900650334015,\n
\"min\": 111.07260851604703,\n        \"max\": 122.61352409994097,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
111.07260851604703,\n          119.84970671834624,\n
122.61352409994097\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Speechiness\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.03054088819177807,\n        \"min\":
0.04715923101366381,\n        \"max\": 0.10488816315823989,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
0.04715923101366381,\n          0.09330852713178295,\n
0.10488816315823989\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Acousticness\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.2997279870893155,\n        \"min\":
0.19557125775193798,\n        \"max\": 0.7186737976835081,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
0.7186737976835081,\n          0.19557125775193798,\n
0.20358081858154634\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Instrumentalness\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.1269994152651009,\n        \"min\":
0.00581588488372093,\n        \"max\": 0.23313273834763265,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
0.23313273834763265,\n          0.00581588488372093,\n
0.021332370879401926\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Liveness\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.023020826107442504,\n        \"min\":
```

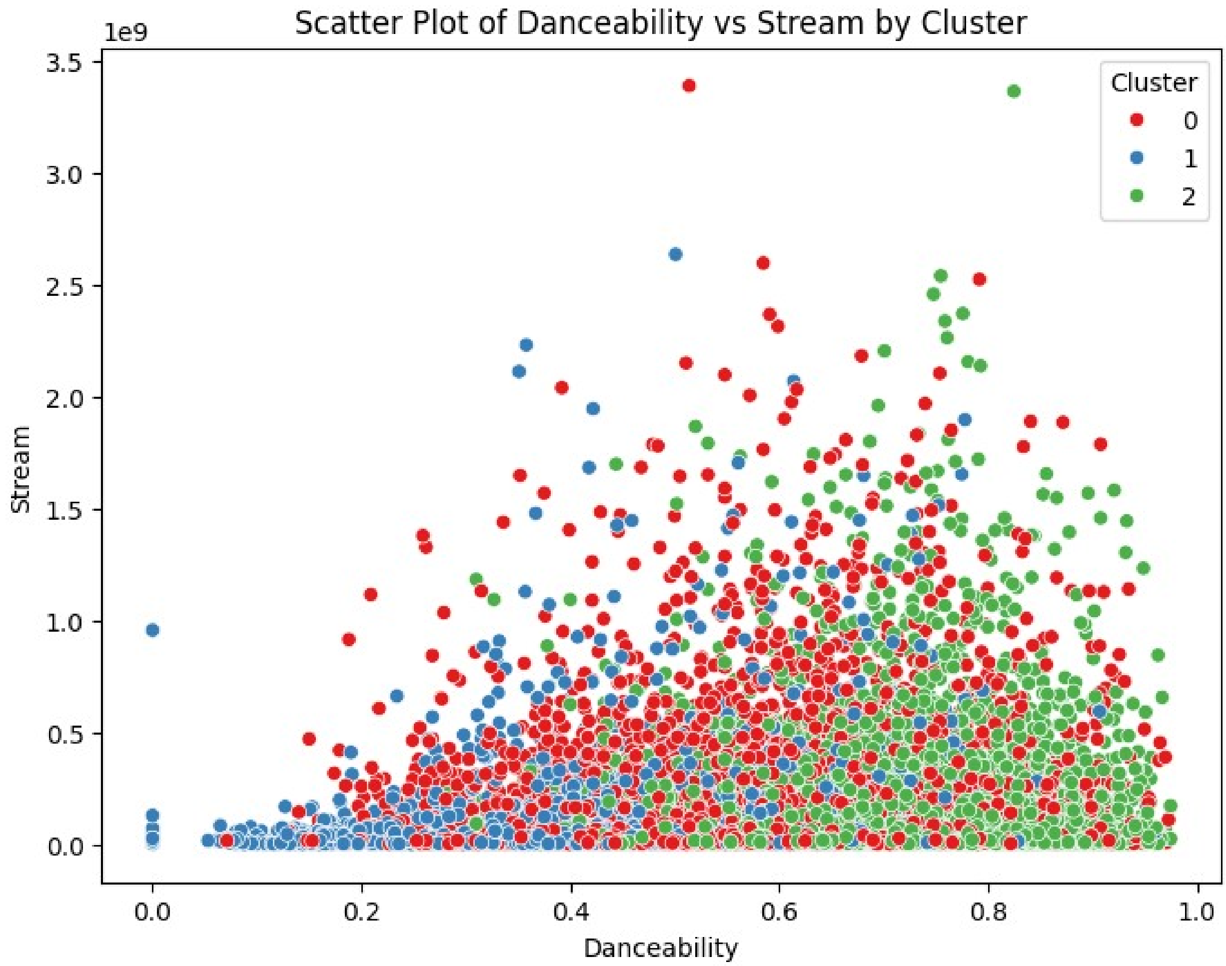
```
0.15374995233555766,\n        \"max\": 0.19978753360876125,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
0.15374995233555766,\n          0.17729896640826873,\n
0.19978753360876125\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Views\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 582724517.9804786,\n        \"min\": 29672744.90085796,\n
\"max\": 1055002643.4573643,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          29672744.90085796,\n
1055002643.4573643,\n          62517372.69270116\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Likes\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 3892237.1099447547,\
n        \"min\": 232029.01239275502,\n        \"max\":
7080185.662790698,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          232029.01239275502,\n
7080185.662790698,\n          450550.9071414519\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Stream\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 501589538.514698,\
n        \"min\": 79533331.95074674,\n        \"max\":
962463209.007752,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          79533331.95074674,\n
962463209.007752,\n          108563645.85100663\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}

# Scatter plot to show Views vs Likes, with different clusters as
colors
plt.figure(figsize=(8, 6))

sns.scatterplot(data=df, x='Views', y='Likes', hue='Cluster_minmax',
palette='Set1')

plt.title('Scatter Plot of Views vs Likes by Cluster')
plt.xlabel('Views')
plt.ylabel('Likes')
plt.legend(title='Cluster')
plt.show()
```



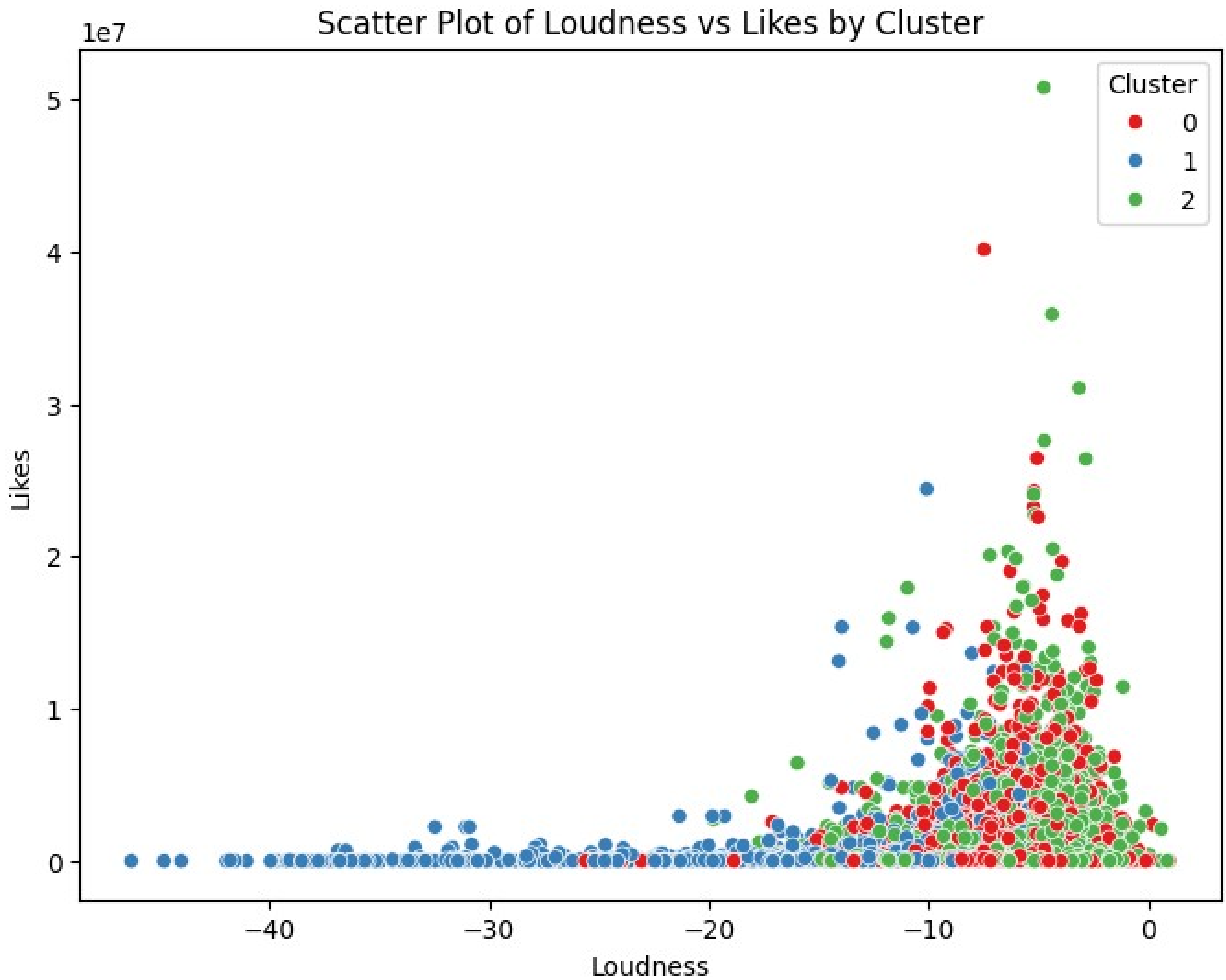

```python
# Scatter plot to show Stream vs Likes, with different clusters as colors
plt.figure(figsize=(8, 6))

sns.scatterplot(data=df, x='Stream', y='Likes', hue='Cluster_minmax', palette='Set1')

plt.title('Scatter Plot of Steam vs Likes by Cluster')
plt.xlabel('Steam')
plt.ylabel('Likes')
plt.legend(title='Cluster')
plt.show()
```




```
# Scatter plot to show Danceability vs Stream, with different clusters
as colors
plt.figure(figsize=(8, 6))

sns.scatterplot(data=df, x='Danceability', y='Stream',
hue='Cluster_minmax', palette='Set1')

plt.title('Scatter Plot of Danceability vs Stream by Cluster')
plt.xlabel('Danceability')
plt.ylabel('Stream')
plt.legend(title='Cluster')
plt.show()
```




```
# Scatter plot to show Loudness vs Likes, with different clusters as
colors
plt.figure(figsize=(8, 6))

sns.scatterplot(data=df, x='Loudness', y='Likes',
hue='Cluster_minmax', palette='Set1')

plt.title('Scatter Plot of Loudness vs Likes by Cluster')
plt.xlabel('Loudness')
plt.ylabel('Likes')
plt.legend(title='Cluster')
plt.show()
```


Scatter Plot of Loudness vs Likes by Cluster
1e7
Likes
Loudness
Cluster
0
1
2
5
4
3
2
1
0
−40
−30
−20
−10
0