main

May 8, 2025

1 Music Recommendation System with Spotify Data

1.1 1. Data Loading

Load the Spotify dataset from a CSV file and inspect its shape and first few rows to understand the structure.

[1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import MinMaxScaler

def load_data(file_path):
    df = pd.read_csv(file_path)
    print("Original data shape:", df.shape)
    print("Original data head:\n", df.head(3))
    return df

df_raw = load_data('data/spotify_data.csv')
```

```
Original data shape: (1159764, 20)
Original data head:
    Unnamed: 0    artist_name        track_name                track_id  \
0           0     Jason Mraz   I Won't Give Up  53QF56cjZA9RTuuMZDrSA6
1           1     Jason Mraz  93 Million Miles  1s8tP3jP4GZcyHDsjvw218
2           2  Joshua Hyslop  Do Not Let Me Go  7BRCa8MPiyuvr2VU3O9W0F

   popularity  year     genre  danceability  energy  key  loudness  mode  \
0          68  2012  acoustic         0.483   0.303    4   -10.058     1
1          50  2012  acoustic         0.572   0.454    3   -10.286     1
2          57  2012  acoustic         0.409   0.234    3   -13.711     1

   speechiness  acousticness  instrumentalness  liveness  valence    tempo  \
0       0.0429         0.694          0.000000    0.1150    0.139  133.406
1       0.0258         0.477          0.000014    0.0974    0.515  140.182
2       0.0323         0.338          0.000050    0.0895    0.145  139.832

   duration_ms  time_signature
```

```
0       240166              3
1       216387              4
2       158960              4
```

1.1.1 2. Initial Cleaning & Preprocessing :

[2]:
```python
def clean_data(df):
    df_clean = df.dropna().copy()
    print("Missing values before dropping:\n", df.isnull().sum())
    print("\nShape before dropping NA:", df.shape)
    print("\nShape after dropping NA:", df_clean.shape)
    print("\nDuplicate rows:", df_clean.duplicated().sum())
    return df_clean
df_clean = clean_data(df_raw)
```

```
Missing values before dropping:
 Unnamed: 0           0
artist_name         15
track_name           1
track_id             0
popularity           0
year                 0
genre                0
danceability         0
energy               0
key                  0
loudness             0
mode                 0
speechiness          0
acousticness         0
instrumentalness     0
liveness             0
valence              0
tempo                0
duration_ms          0
time_signature       0
dtype: int64

Shape before dropping NA: (1159764, 20)

Shape after dropping NA: (1159748, 20)

Duplicate rows: 0
```

1.1.2 3. Feature Selection and Overview

The dataset contains various audio features from Spotify. For our similarity model, we'll focus on intrinsic audio characteristics rather than metadata.

Selected Features for Modeling:

Feature	Description	Range
Danceability	Suitability for dancing based on rhythm and beat	0.0-1.0
Energy	Intensity and activity level	0.0-1.0
Key	Musical key (requires one-hot encoding)	-1 to 11
Loudness	Overall loudness (requires scaling)	-60 to 0 dB
Mode	Major (1) or Minor (0) key (requires one-hot encoding)	0 or 1
Speechiness	Presence of spoken words	0.0-1.0
Acousticness	Likelihood of being acoustic	0.0-1.0
Instrumentalness	Likelihood of having no vocals	0.0-1.0
Liveness	Presence of audience/live recording	0.0-1.0
Valence	Musical positiveness/mood	0.0-1.0
Tempo	Speed in BPM (requires scaling)	Varies

We exclude metadata (artist_name, track_name, track_id, genre, year), popularity, duration_ms, and time_signature as they're less relevant for audio similarity.

[3]:
```
features = ['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness',␣
↪'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo']

df = df_clean[features]
print("\nSelected features shape:", df.shape)

# Set the index to track_id from the original dataframe
df.index = df_clean['track_id'].values
df.index.name = 'track_id'
```

```
Selected features shape: (1159748, 11)
```

1.1.3 4. Feature Engineering & Transformation:

[4]:
```
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

# Apply min-max scaling to tempo
print("\nTempo statistics before scaling:")
print(df['tempo'].describe())
scaler_tempo = MinMaxScaler()
df['tempo'] = scaler_tempo.fit_transform(df[['tempo']])
print("\nTempo statistics after scaling:")
print(df['tempo'].describe())

# One-hot encode key
```

```
key_dummies = pd.get_dummies(df['key'], prefix='key', drop_first=False) # Keep␣
 ↪all keys
df = pd.concat([df, key_dummies], axis=1)

# One-hot encode mode
mode_dummies = pd.get_dummies(df['mode'], prefix='mode', drop_first=False) #␣
 ↪Keep both 0 and 1
df = pd.concat([df, mode_dummies], axis=1)

# Drop original categorical columns
df.drop(['key', 'mode'], axis=1, inplace=True)

print("\nProcessed features head (after OHE):\n", df.head(2))
print("Processed features shape (after OHE):", df.shape)
print("Index check:", df.index.name)

# Create some music-specific Meaningful composite features
df['energy_to_acousticness_ratio'] = df['energy'] / (df['acousticness'] + 0.01)
df['vocal_character'] = df['speechiness'] * (1 - df['instrumentalness'])
```

```
Tempo statistics before scaling:
count    1.159748e+06
mean     1.213775e+02
std      2.977964e+01
min      0.000000e+00
25%      9.879800e+01
50%      1.219310e+02
75%      1.399030e+02
max      2.499930e+02
Name: tempo, dtype: float64

Tempo statistics after scaling:
count    1.159748e+06
mean     4.855236e-01
std      1.191219e-01
min      0.000000e+00
25%      3.952031e-01
50%      4.877377e-01
75%      5.596277e-01
max      1.000000e+00
Name: tempo, dtype: float64

Processed features head (after OHE):
                         danceability  energy  loudness  speechiness  \
track_id
53QF56cjZA9RTuuMZDrSA6         0.483   0.303   -10.058       0.0429
```

```
1s8tP3jP4GZcyHDsjvw218         0.572   0.454   -10.286       0.0258

                        acousticness  instrumentalness  liveness  valence  \
track_id
53QF56cjZA9RTuuMZDrSA6         0.694          0.000000    0.1150    0.139
1s8tP3jP4GZcyHDsjvw218         0.477          0.000014    0.0974    0.515

                           tempo  key_0  …  key_4  key_5  key_6  key_7  \
track_id                                 …
53QF56cjZA9RTuuMZDrSA6  0.533639  False  …   True  False  False  False
1s8tP3jP4GZcyHDsjvw218  0.560744  False  …  False  False  False  False

                        key_8  key_9  key_10  key_11  mode_0  mode_1
track_id
53QF56cjZA9RTuuMZDrSA6  False  False   False   False   False    True
1s8tP3jP4GZcyHDsjvw218  False  False   False   False   False    True

[2 rows x 23 columns]
Processed features shape (after OHE): (1159748, 23)
Index check: track_id
```

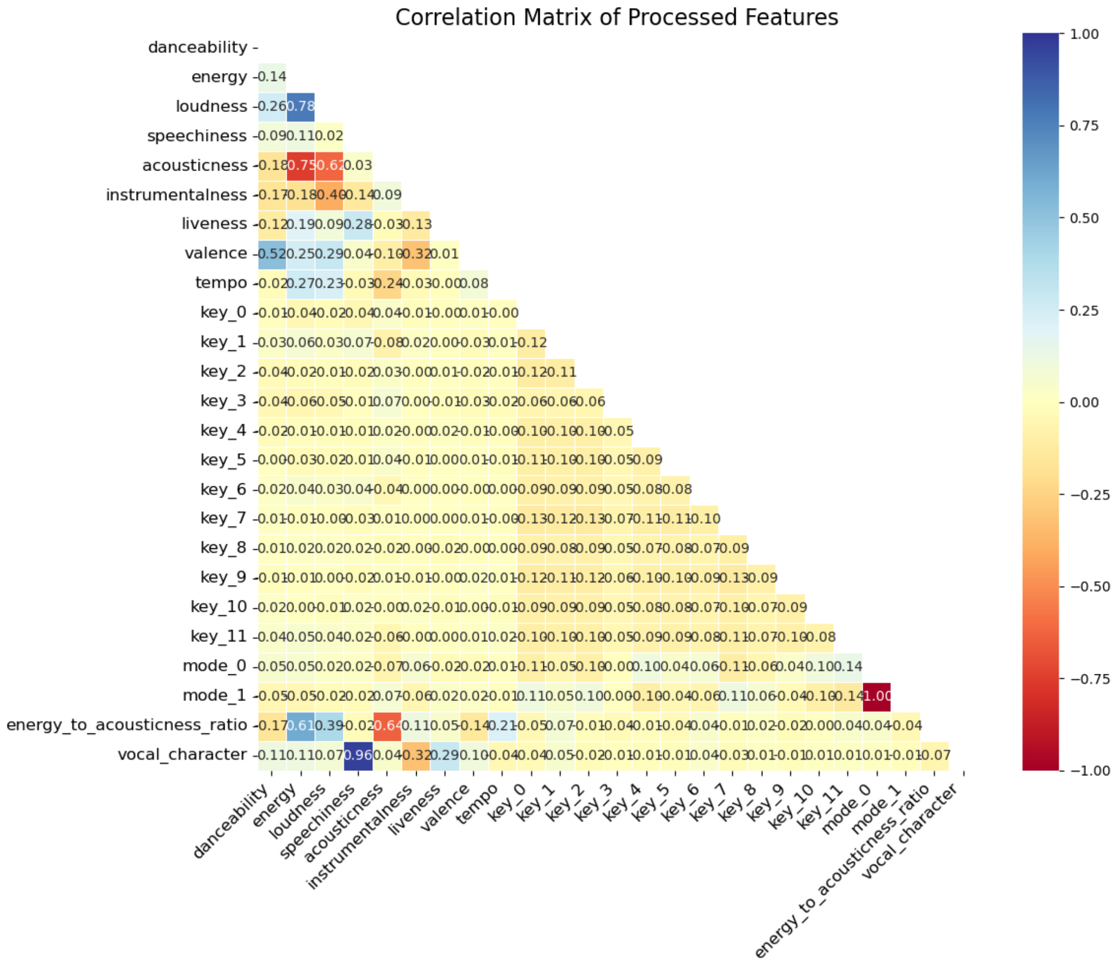
```
/var/folders/81/vw07xvx93g58v6nwmy2n02b00000gn/T/ipykernel_23341/4059478908.py:8
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['tempo'] = scaler_tempo.fit_transform(df[['tempo']])
```

1.1.4 5. Exploratory Data Analysis :

[5]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

print("\nCalculating Correlation Matrix...")
correlation_matrix = df.corr()

plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='RdYlBu', fmt='.2f',
            linewidths=0.5, vmin=-1, vmax=1, mask=np.triu(np.
 ↪ones_like(correlation_matrix, dtype=bool)),
            annot_kws={"size": 10})
plt.title('Correlation Matrix of Processed Features', fontsize=16)
plt.xticks(rotation=45, ha='right', fontsize=12)
plt.yticks(fontsize=12)
plt.tight_layout()
```

```
plt.show()
```

```
Calculating Correlation Matrix…
```



Correlation Analysis Findings The correlation matrix reveals several highly correlated feature pairs:

Feature Pair	Correlation Coefficient
energy and loudness	0.781
energy and acousticness	-0.753
speechiness and vocal_character	0.965
mode_0 and mode_1	-1.000

These strong correlations suggest potential redundancy in our feature set, which may impact model performance.

The perfect negative correlation between mode_0 and mode_1 is expected since they are one-hot encoded from the same categorical variable.

[6]:
```
# Drop highly correlated features to reduce redundancy
print("\nDropping highly correlated features...")
features_to_drop = ['mode_1', 'vocal_character', 'acousticness', 'loudness']
df = df.drop(columns=features_to_drop)

print(f"Features dropped: {features_to_drop}")
print(f"Remaining features: {df.columns.tolist()}")
print("Original dataframe shape:", df_raw.shape)
print(f"New dataframe shape: {df.shape}")
```

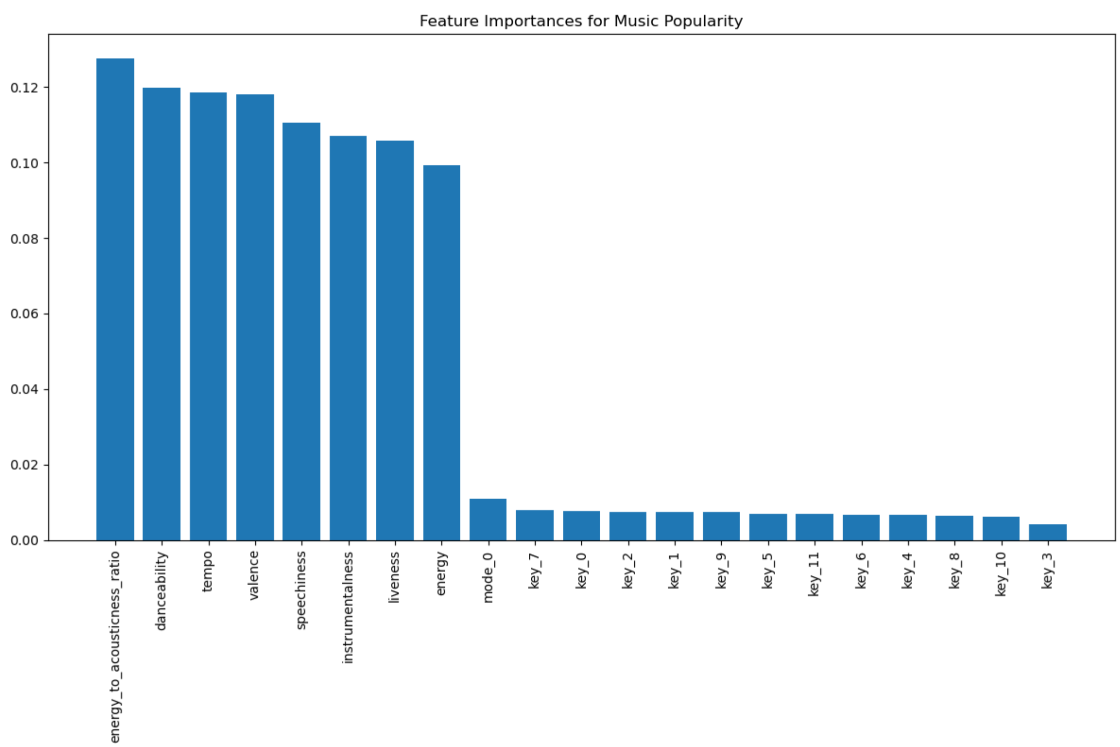
```
Dropping highly correlated features…
Features dropped: ['mode_1', 'vocal_character', 'acousticness', 'loudness']
Remaining features: ['danceability', 'energy', 'speechiness',
'instrumentalness', 'liveness', 'valence', 'tempo', 'key_0', 'key_1', 'key_2',
'key_3', 'key_4', 'key_5', 'key_6', 'key_7', 'key_8', 'key_9', 'key_10',
'key_11', 'mode_0', 'energy_to_acousticness_ratio']
Original dataframe shape: (1159764, 20)
New dataframe shape: (1159748, 21)
```

Feature Importance Analysis

[7]:
```
from sklearn.ensemble import RandomForestRegressor
import multiprocessing
print("Evaluating feature importance using Random Forest...")

# Use a simple target like popularity
target = df_clean['popularity']

# Train a random forest model to get feature importance with parallel processing
rf = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)
rf.fit(df, target)

# Get feature importances
importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]

# Plot feature importances
plt.figure(figsize=(12, 8))
plt.title('Feature Importances for Music Popularity')
plt.bar(range(df.shape[1]), importances[indices], align='center')
plt.xticks(range(df.shape[1]), [df.columns[i] for i in indices], rotation=90)
plt.tight_layout()
plt.show()

# Keep only top features
```

```
importance_threshold = 0.05  # This will capture the most significant features␣
 ↪based on your graph
important_indices = [i for i, imp in enumerate(importances) if imp >␣
 ↪importance_threshold]
top_features = [df.columns[i] for i in indices if importances[i] >␣
 ↪importance_threshold]
print(f"Features with importance > {importance_threshold}: {top_features}")
df_important = df[top_features]

# Let's also keep the custom ratio features since they're highly important
if 'energy_to_acousticness_ratio' not in df_important.columns and␣
 ↪'energy_to_acousticness_ratio' in df.columns:
    df_important['energy_to_acousticness_ratio'] =␣
 ↪df['energy_to_acousticness_ratio']

if 'vocal_character' not in df_important.columns and 'vocal_character' in df.
 ↪columns:
    df_important['vocal_character'] = df['vocal_character']

print(f"Refined feature set shape: {df_important.shape}")
```

Evaluating feature importance using Random Forest…



Features with importance > 0.05: ['energy_to_acousticness_ratio',

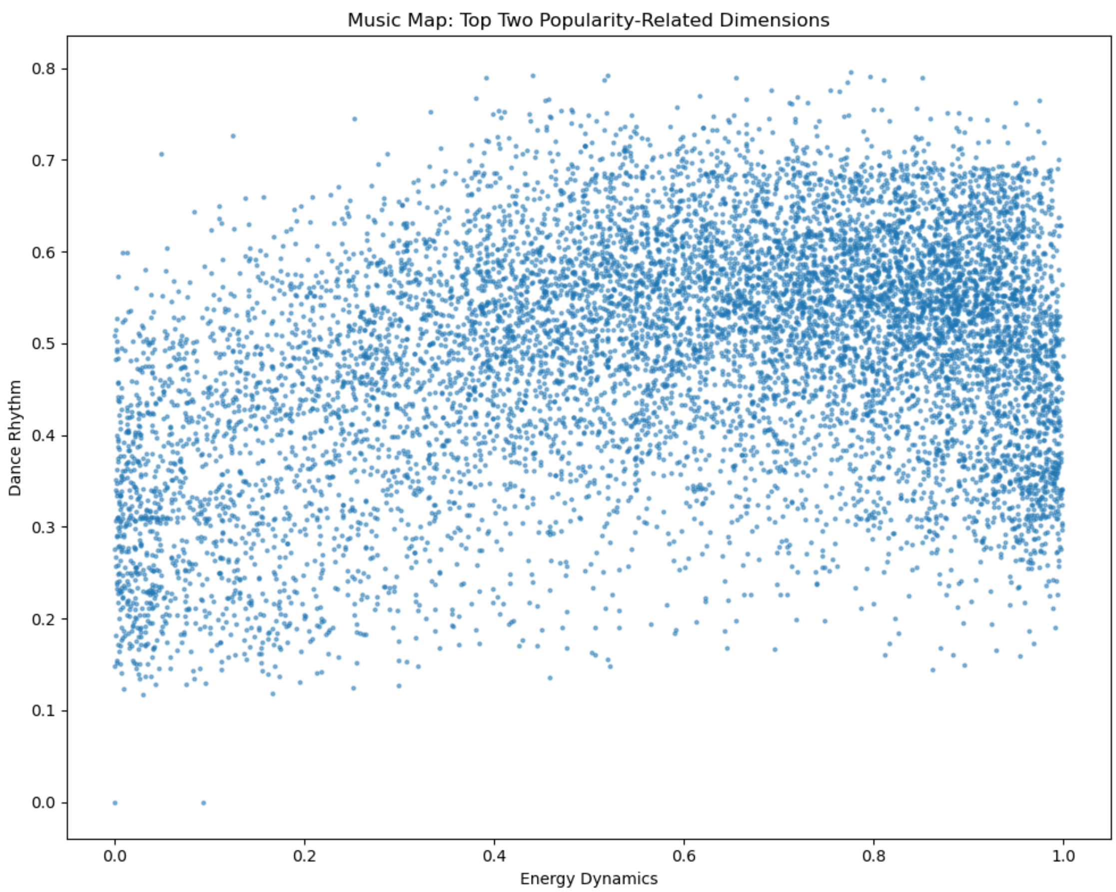
```
'danceability', 'tempo', 'valence', 'speechiness', 'instrumentalness',
'liveness', 'energy']
Refined feature set shape: (1159748, 8)
```

[8]:
```python
# Define more musically meaningful components based on feature importance
df_important = df_important.copy()
df_important.loc[:, 'energy_dynamics'] = df['energy']
df_important.loc[:, 'dance_rhythm'] = 0.6*df['danceability'] + 0.4*df['tempo']
df_important.loc[:, 'emotional_content'] = df['valence']
df_important.loc[:, 'vocal_presence'] = df['speechiness'] - 0.
 ↪5*df['instrumentalness']
df_important.loc[:, 'performance_style'] = df['liveness']

print("Improved music-specific components (first 5 rows):")
print(df_important.head())

# Visualize top dimensions
plt.figure(figsize=(10, 8))
sample_size = min(10000, len(df_important))
plt.scatter(
    df_important['energy_dynamics'].iloc[:sample_size],
    df_important['dance_rhythm'].iloc[:sample_size],
    alpha=0.5, s=5
)
plt.xlabel('Energy Dynamics')
plt.ylabel('Dance Rhythm')
plt.title('Music Map: Top Two Popularity-Related Dimensions')
plt.tight_layout()
plt.show()
```

```
Improved music-specific components (first 5 rows):
                        energy_to_acousticness_ratio  danceability     tempo  \
track_id
53QF56cjZA9RTuuMZDrSA6                      0.430398         0.483  0.533639
1s8tP3jP4GZcyHDsjvw218                      0.932238         0.572  0.560744
7BRCa8MPiyuvr2VU3O9W0F                      0.672414         0.409  0.559344
63wsZUhUZLlh1OsyrZq7sz                      0.307222         0.392  0.819867
6nXIYClvJAfi6ujLiKqEq8                      9.576271         0.430  0.687475

                        valence  speechiness  instrumentalness  liveness  \
track_id
53QF56cjZA9RTuuMZDrSA6    0.139       0.0429          0.000000    0.1150
1s8tP3jP4GZcyHDsjvw218    0.515       0.0258          0.000014    0.0974
7BRCa8MPiyuvr2VU3O9W0F    0.145       0.0323          0.000050    0.0895
63wsZUhUZLlh1OsyrZq7sz    0.508       0.0363          0.000000    0.0797
6nXIYClvJAfi6ujLiKqEq8    0.217       0.0302          0.019300    0.1100

                        energy  energy_dynamics  dance_rhythm  \
```

track_id			
53QF56cjZA9RTuuMZDrSA6	0.303	0.303	0.503256
1s8tP3jP4GZcyHDsjvw218	0.454	0.454	0.567497
7BRCa8MPiyuvr2VU309W0F	0.234	0.234	0.469137
63wsZUhUZLlh1OsyrZq7sz	0.251	0.251	0.563147
6nXIYClvJAfi6ujLiKqEq8	0.791	0.791	0.532990

	emotional_content	vocal_presence	performance_style
track_id			
53QF56cjZA9RTuuMZDrSA6	0.139	0.042900	0.1150
1s8tP3jP4GZcyHDsjvw218	0.515	0.025793	0.0974
7BRCa8MPiyuvr2VU309W0F	0.145	0.032275	0.0895
63wsZUhUZLlh1OsyrZq7sz	0.508	0.036300	0.0797
6nXIYClvJAfi6ujLiKqEq8	0.217	0.020550	0.1100



1.1.5 6. Standard Scaling:

We'll initially scale all features to have zero mean and unit variance. This is important for distance-based algorithms like K-Means and PCA..

[9]:
```python
# Import StandardScaler from sklearn.preprocessing
from sklearn.preprocessing import StandardScaler

# Apply standard scaling to the reduced feature set
scaler_opt = StandardScaler()
df_important_scaled = scaler_opt.fit_transform(df_important)
df_important_scaled = pd.DataFrame(df_important_scaled, index=df.index,␣
↪columns=df_important.columns)
```

1.1.6 7. Dimensionality Reduction:

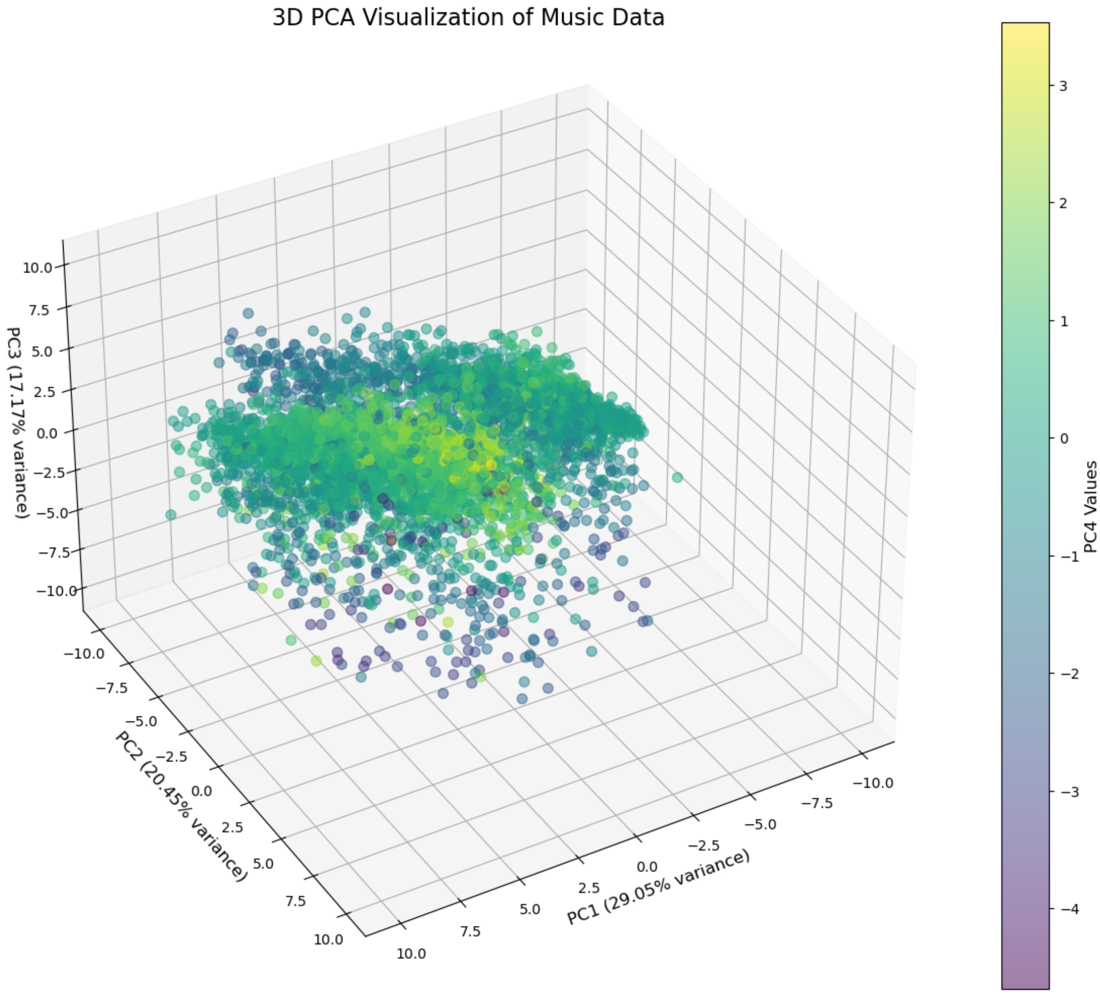
We'll use PCA on the scaled data to reduce the number of features while retaining most of the information (variance).

[10]:
```python
from sklearn.decomposition import PCA

# Apply PCA to reduce dimensionality while retaining most variance
n_components = 6
pca = PCA(n_components=n_components)
df_pca = pca.fit_transform(df_important_scaled)
df_pca = pd.DataFrame(df_pca, index=df_important.index,
                      columns=[f'PC{i+1}' for i in range(n_components)])
# Print explained variance to evaluate PCA effectiveness
explained_variance = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance)
print(f"Using {n_components} components for dimensionality reduction")
print(f"Total explained variance: {cumulative_variance[-1]:.4f}")
```

```
Using 6 components for dimensionality reduction
Total explained variance: 0.9199
```

[11]:
```python
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

# Create a 3D plot for the first 3 principal components
sample_size = min(5000, len(df_pca))  # Smaller sample for clarity
scaler = MinMaxScaler(feature_range=(-10, 10)) # Scale data for better␣
↪visualization
viz_data = scaler.fit_transform(df_pca[['PC1', 'PC2', 'PC3']].iloc[:
↪sample_size])
fig = plt.figure(figsize=(12, 10))
ax = fig.add_subplot(projection='3d')
scatter = ax.scatter(viz_data[:, 0], viz_data[:, 1], viz_data[:, 2],
                    c=df_pca['PC4'].iloc[:sample_size], cmap='viridis', alpha=0.
↪5, s=50)
plt.colorbar(scatter).set_label('PC4 Values', fontsize=12)
ax.set_xlabel(f'PC1 ({explained_variance[0]*100:.2f}% variance)', fontsize=12)
ax.set_ylabel(f'PC2 ({explained_variance[1]*100:.2f}% variance)', fontsize=12)
```

```
ax.set_zlabel(f'PC3 ({explained_variance[2]*100:.2f}% variance)', fontsize=12)
plt.title('3D PCA Visualization of Music Data', fontsize=16)
ax.grid(True, alpha=0.3)
ax.view_init(elev=35, azim=60)
plt.tight_layout()
plt.show()
```



1.1.7 8. Clustering using K means :

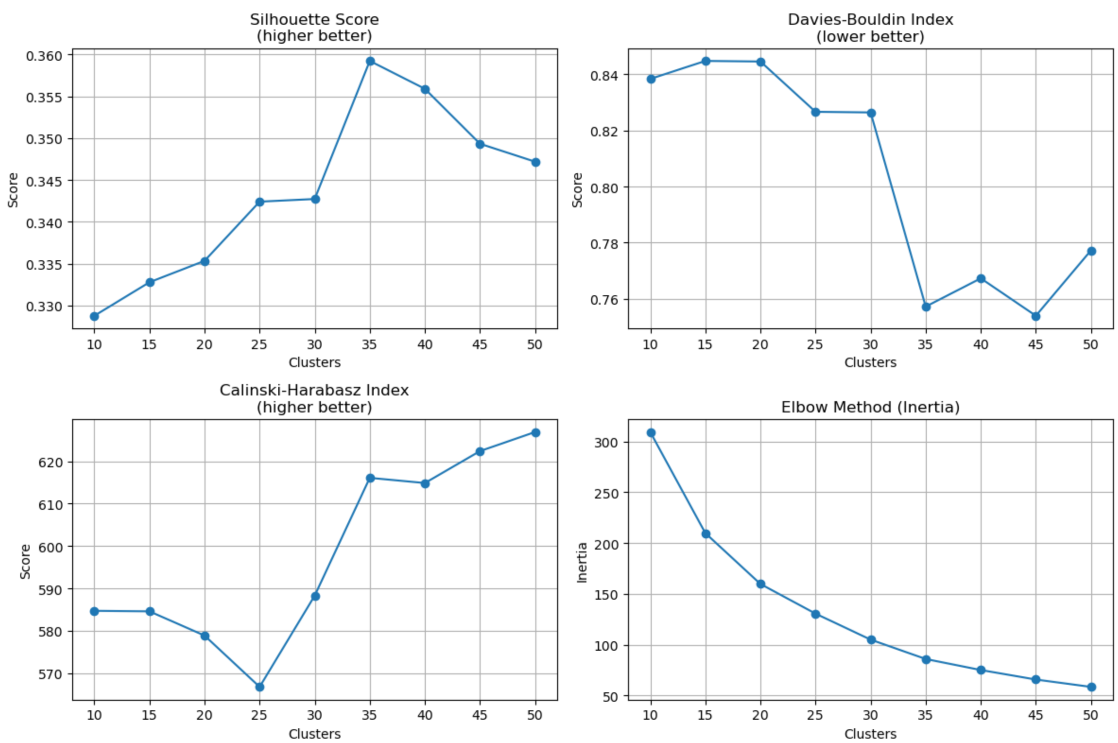
[12]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, davies_bouldin_score,
    ↪calinski_harabasz_score
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from tqdm import tqdm
```

```python
import time

def evaluate_clusters(data, min_clusters=10, max_clusters=100, step=10):
    print(f"Evaluating clusters ({min_clusters}-{max_clusters}, step: {step})...")
    results = {'n_clusters': [], 'silhouette': [], 'davies_bouldin': [],
               'calinski_harabasz': [], 'inertia': [], 'time': []}

    sample_size = min(50000, len(data))
    sampled_data = data.sample(n=sample_size, random_state=42) if sample_size < len(data) else data

    for n in tqdm(range(min_clusters, max_clusters + 1, step)):
        start_time = time.time()
        kmeans = KMeans(n_clusters=n, random_state=42, n_init=10)
        labels = kmeans.fit_predict(sampled_data)

        results['n_clusters'].append(n)
        results['silhouette'].append(silhouette_score(sampled_data, labels, sample_size=10000))
        results['davies_bouldin'].append(davies_bouldin_score(sampled_data, labels))
        results['calinski_harabasz'].append(calinski_harabasz_score(sampled_data, labels))
        results['inertia'].append(kmeans.inertia_)
        results['time'].append(time.time() - start_time)

        print(f"Clusters: {n}, Silhouette: {results['silhouette'][-1]:.4f}, "
              f"Davies-Bouldin: {results['davies_bouldin'][-1]:.4f}, "
              f"Calinski-Harabasz: {results['calinski_harabasz'][-1]:.2f}, "
              f"Time: {results['time'][-1]:.2f}s")

    return results

def plot_metrics(results):
    fig, axes = plt.subplots(2, 2, figsize=(12, 8))

    metrics = [
        ('silhouette', 'Silhouette Score\n(higher better)', 0, 0),
        ('davies_bouldin', 'Davies-Bouldin Index\n(lower better)', 0, 1),
        ('calinski_harabasz', 'Calinski-Harabasz Index\n(higher better)', 1, 0),
        ('inertia', 'Elbow Method (Inertia)', 1, 1)
    ]

    for metric, title, row, col in metrics:
        axes[row, col].plot(results['n_clusters'], results[metric], 'o-')
        axes[row, col].set_title(title)
```

```
        axes[row, col].set_xlabel('Clusters')
        axes[row, col].set_ylabel('Score' if metric != 'inertia' else 'Inertia')
        axes[row, col].grid(True)

    plt.tight_layout()
    plt.savefig('cluster_metrics.png')
    plt.show()
    return fig

def get_optimal_clusters(results):
    # Normalize metrics
    silhouette = (np.array(results['silhouette']) - min(results['silhouette']))␣
 ↪/ \
                 (max(results['silhouette']) - min(results['silhouette']))
    db = 1 - (np.array(results['davies_bouldin']) -␣
 ↪min(results['davies_bouldin'])) / \
             (max(results['davies_bouldin']) - min(results['davies_bouldin']))
    ch = (np.array(results['calinski_harabasz']) -␣
 ↪min(results['calinski_harabasz'])) / \
         (max(results['calinski_harabasz']) - min(results['calinski_harabasz']))

    # Composite score
    score = 0.4 * silhouette + 0.3 * db + 0.3 * ch
    optimal_n = results['n_clusters'][np.argmax(score)]

    print(f"Optimal clusters: {optimal_n}")
    return optimal_n

# Example usage with synthetic data
if __name__ == "__main__":
    # Generate sample data
    np.random.seed(42)
    data = pd.DataFrame(np.random.randn(1000, 2))

    # Call functions
    results = evaluate_clusters(data, min_clusters=10, max_clusters=50, step=5)
    plot_metrics(results)
    optimal_n = get_optimal_clusters(results)
```

```
Evaluating clusters (10-50, step: 5)…

 11%|                                          | 1/9 [00:00<00:01,  4.81it/s]

Clusters: 10, Silhouette: 0.3288, Davies-Bouldin: 0.8383, Calinski-Harabasz:
584.71, Time: 0.21s

 22%|                                        | 2/9 [00:00<00:01,  4.70it/s]

Clusters: 15, Silhouette: 0.3328, Davies-Bouldin: 0.8447, Calinski-Harabasz:
```

```
584.59, Time: 0.22s

 33%|                                        | 3/9 [00:00<00:01,  4.01it/s]

Clusters: 20, Silhouette: 0.3353, Davies-Bouldin: 0.8445, Calinski-Harabasz:
578.88, Time: 0.29s

 44%|                                      | 4/9 [00:00<00:01,  3.82it/s]

Clusters: 25, Silhouette: 0.3424, Davies-Bouldin: 0.8266, Calinski-Harabasz:
566.77, Time: 0.28s

 78%|                               | 7/9 [00:01<00:00,  5.69it/s]

Clusters: 30, Silhouette: 0.3427, Davies-Bouldin: 0.8264, Calinski-Harabasz:
588.28, Time: 0.29s
Clusters: 35, Silhouette: 0.3592, Davies-Bouldin: 0.7572, Calinski-Harabasz:
616.07, Time: 0.08s
Clusters: 40, Silhouette: 0.3559, Davies-Bouldin: 0.7673, Calinski-Harabasz:
614.85, Time: 0.08s

100%|                         | 9/9 [00:01<00:00,  5.30it/s]

Clusters: 45, Silhouette: 0.3493, Davies-Bouldin: 0.7540, Calinski-Harabasz:
622.37, Time: 0.12s
Clusters: 50, Silhouette: 0.3472, Davies-Bouldin: 0.7772, Calinski-Harabasz:
626.87, Time: 0.12s
```



Optimal clusters: 35

[13]:
```python
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

n_clusters = optimal_n
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
df_pca['cluster'] = kmeans.fit_predict(df_pca)
silhouette_avg = silhouette_score(df_pca.drop('cluster', axis=1),␣
↪df_pca['cluster'], sample_size=10000)
print("\nSilhouette Score for clustering:", silhouette_avg)

# Recommendation Function
from sklearn.metrics.pairwise import cosine_similarity

def recommend_songs(track_id, df_pca, df_clean, n_recommendations=5):
    if track_id not in df_pca.index:
        return "Track ID not found."
    cluster = df_pca.loc[track_id, 'cluster']
    similar_songs = df_pca[df_pca['cluster'] == cluster].drop('cluster', axis=1)
    if len(similar_songs) <= 1:
        return "Not enough songs in the same cluster."
    # Calculate cosine similarity
    track_features = similar_songs.loc[track_id].values.reshape(1, -1)
    similarities = cosine_similarity(track_features, similar_songs)[0]
    # Get top similar songs
    similar_indices = similar_songs.index[np.argsort(similarities)[::-1][1:
↪n_recommendations+1]]
    recommendations = df_clean.loc[df_clean['track_id'].isin(similar_indices),
                                   ['track_id', 'track_name', 'artist_name']]
    return recommendations
```

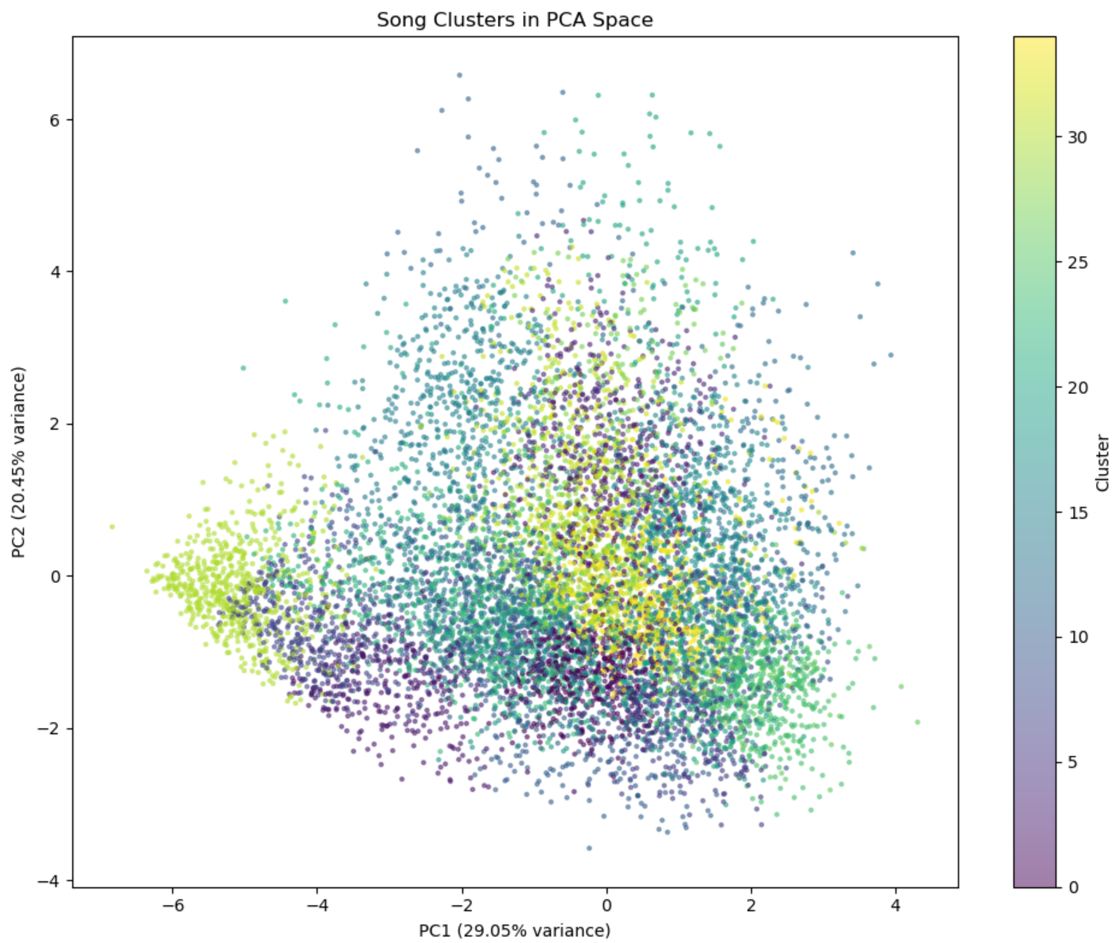
Silhouette Score for clustering: 0.16672112292655636

[250]:
```python
# Test recommendation with random song selection
import random

# Select a random song from the dataset
random_index = random.randint(0, len(df_pca) - 1)
sample_track_id = df_pca.index[random_index]

# Get information about the selected song
song_info = df_clean[df_clean['track_id'] == sample_track_id][['track_name',␣
↪'artist_name', 'genre']]
```

```
print(f"Selected random track: {song_info['track_name'].values[0]} by␣
 ↪{song_info['artist_name'].values[0]}")
print(f"Genre: {song_info['genre'].values[0]}")
print(f"Track ID: {sample_track_id}")

# Get recommendations
print("\nRecommendations:")
recommendations = recommend_songs(sample_track_id, df_pca, df_clean)
print(recommendations)

# Add popularity and genre information to recommendations if available
if 'popularity' in df_clean.columns:
    recommendations = pd.merge(
        recommendations,
        df_clean[['track_id', 'popularity', 'genre']],
        on='track_id',
        how='left'
    )
    print("\nRecommendations with additional info:")
    print(recommendations[['track_name', 'artist_name', 'genre', 'popularity']])

# Visualize Clusters
plt.figure(figsize=(10, 8))
sample_size = min(10000, len(df_pca))
plt.scatter(df_pca['PC1'].iloc[:sample_size], df_pca['PC2'].iloc[:sample_size],
            c=df_pca['cluster'].iloc[:sample_size], cmap='viridis', alpha=0.5,␣
 ↪s=5)
plt.xlabel(f'PC1 ({explained_variance[0]*100:.2f}% variance)')
plt.ylabel(f'PC2 ({explained_variance[1]*100:.2f}% variance)')
plt.title('Song Clusters in PCA Space')
plt.colorbar(label='Cluster')
plt.tight_layout()
plt.savefig('cluster_visualization.png')

# Mark the selected song in the PCA space (if it's in the sample)
if random_index < sample_size:
    plt.scatter(
        df_pca['PC1'].iloc[random_index],
        df_pca['PC2'].iloc[random_index],
        color='red',
        marker='*',
        s=200,
        edgecolor='black',
        label='Selected Song'
    )
    plt.legend()
    plt.savefig('cluster_visualization_with_selection.png')
```

```
plt.show()
```

```
Selected random track: Enough by Disturbed
Genre: metal
Track ID: 0lOonb8Xn49VXJ1Ukb2vgh

Recommendations:
                       track_id        track_name      artist_name
407552   2v9m7YE7C6G968RtjFAzgx      Play My Game       The Donnas
451454   6TBF0fYsDJMZo1fZdtNQM0  Live And Let Die  Bass Modulators
585113   0FUdXqHABSaQ6pTPK1w1Ax         30/30-150       Stone Sour
1021528  4Lf8XuVfGIiluB0unKDD9j            Sahara        Relient K
1089193  6Ec62rWJdqhBysAqKEJQGU            Area 1         All Ends

Recommendations with additional info:
         track_name      artist_name      genre  popularity
0      Play My Game       The Donnas  power-pop          15
1  Live And Let Die  Bass Modulators  hardstyle          25
2         30/30-150       Stone Sour   alt-rock           0
3            Sahara        Relient K   alt-rock          28
4            Area 1         All Ends       goth           8
```



1.1.8 9. Save and Version Outputs:

[106]:
```
df_pca.to_pickle(f'df_pca.pkl')
df_clean.to_pickle(f'df_clean.pkl')

print("End")
```

End