Final_Explainability_Visuals

June 30, 2025

0.0.1 Visualizing Spectrogram Similarity with PCA

This part of the analysis transforms spectrogram images of 100 recommended songs into a visual map using **Principal Component Analysis (PCA)**. The goal is to explore how spectrograms relate to each other in terms of shape, structure, and content — allowing us to explain the behavior of the recommender system in visual terms.

Step-by-step explanation:

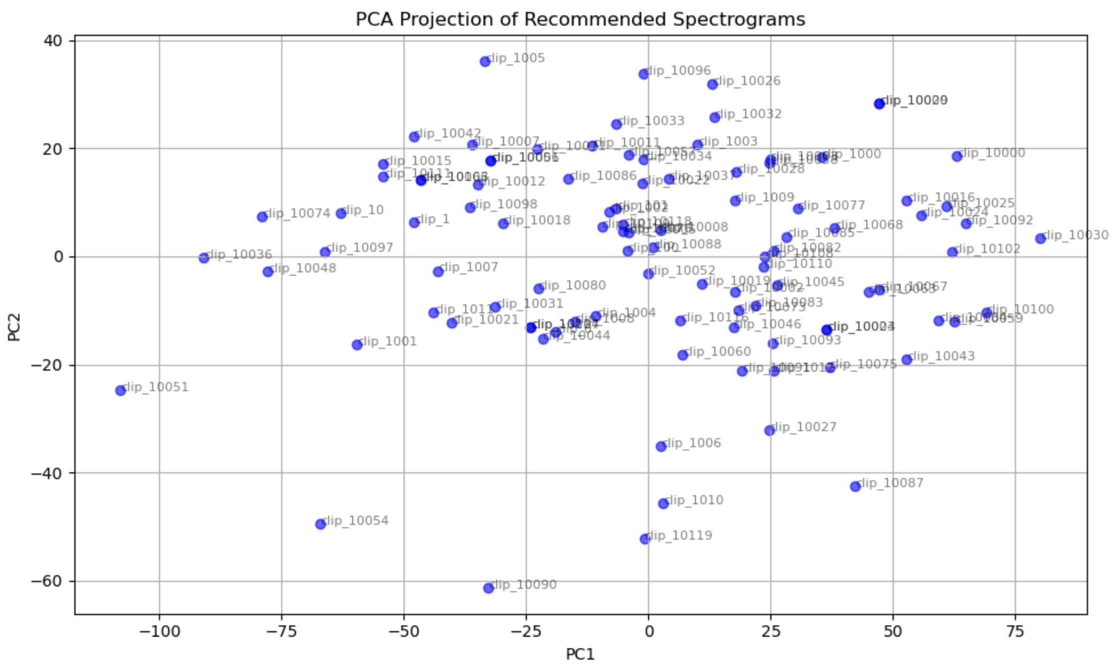
1. **Load spectrograms**: All `.png` spectrogram images are loaded from the `spectrograms/` folder and converted to grayscale.
2. **Resize and flatten**: Each image is resized (e.g. to 64x64 pixels) and flattened into a one-dimensional vector so it can be treated as a numeric input.
3. **Create image matrix**: These vectors are stacked to form a matrix, where each row is a spectrogram.
4. **Standardization**: The matrix is standardized (zero mean, unit variance) to ensure all pixel features contribute equally to the analysis.
5. **PCA transformation**: PCA reduces the image matrix to just two dimensions (PC1 and PC2), capturing as much variance in the spectrogram structure as possible.
6. **Scatter plot**: Each spectrogram is plotted as a point in 2D space, annotated with its ID. Points that appear close together likely represent similar time–frequency characteristics in the original audio.

This visualization helps us **interpret clusters of audio content**, identify outliers, and understand **how the recommender system may be grouping songs** based on spectrogram similarity.

[4]:
```python
import os
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler

# Path to your 100 spectrograms
spectrogram_dir = "spectrograms"  # or another folder if you've separated␣
↪recommended ones
image_files = sorted([f for f in os.listdir(spectrogram_dir) if f.endswith(".
↪png")])
```

```python
print(f" Found {len(image_files)} spectrograms")

# Load and flatten images
def load_and_flatten_image(path, size=(64, 64)):
    img = Image.open(path).convert("L")  # grayscale
    img = img.resize(size)
    return np.array(img).flatten()

# Create matrix
image_vectors = []
for fname in image_files:
    full_path = os.path.join(spectrogram_dir, fname)
    vec = load_and_flatten_image(full_path)
    image_vectors.append(vec)

image_vectors = np.array(image_vectors)

# Standardize the data
scaler = StandardScaler()
image_vectors_std = scaler.fit_transform(image_vectors)

# Reduce to 2D
print("Running PCA...")
pca = PCA(n_components=2)
components = pca.fit_transform(image_vectors_std)

# Visualize
plt.figure(figsize=(10, 6))
plt.scatter(components[:, 0], components[:, 1], c='blue', alpha=0.6)
for i, fname in enumerate(image_files):
    plt.annotate(fname.split(".")[0], (components[i, 0], components[i, 1]),␣
↪fontsize=8, alpha=0.5)
plt.title("PCA Projection of Recommended Spectrograms")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.grid(True)
plt.tight_layout()
plt.show()
```

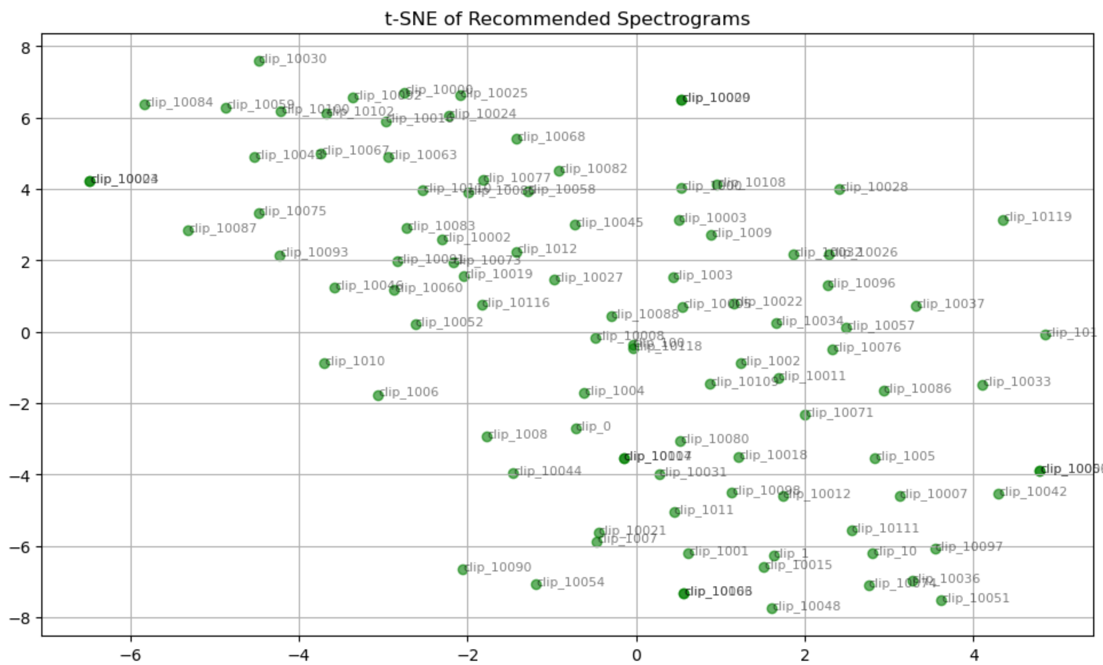
```
 Found 100 spectrograms
Running PCA…
```



[5]:

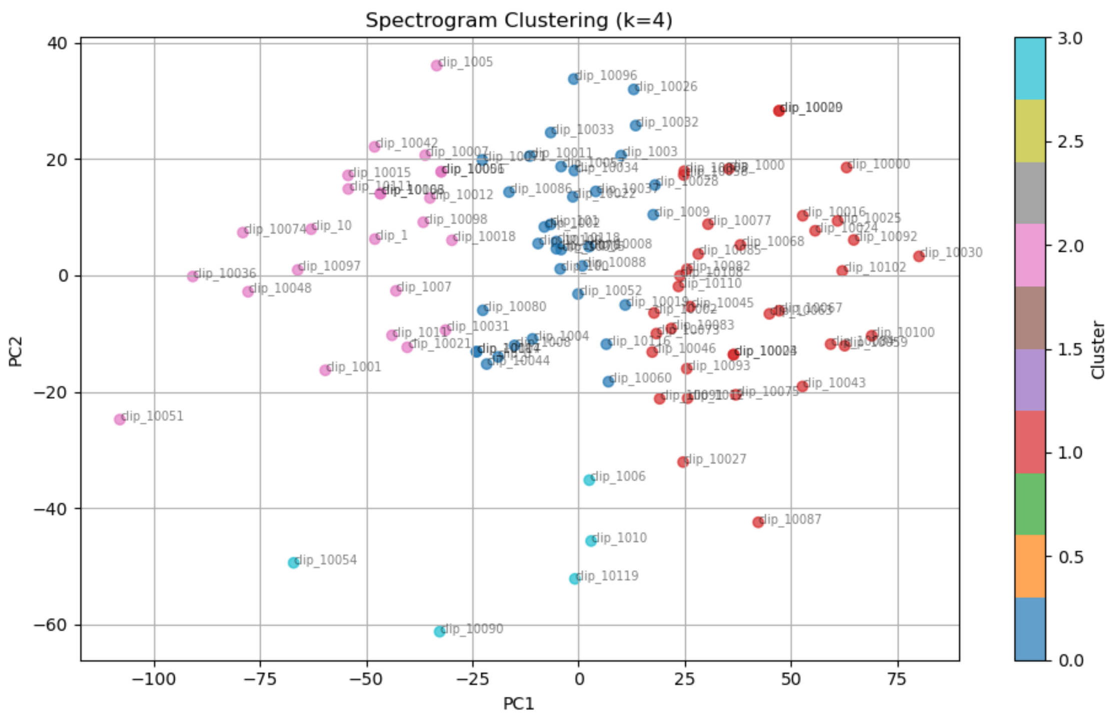
```
print("Running t-SNE (this may take 30-60 sec)...")
tsne = TSNE(n_components=2, perplexity=30, random_state=42)
components_tsne = tsne.fit_transform(image_vectors_std)

plt.figure(figsize=(10, 6))
plt.scatter(components_tsne[:, 0], components_tsne[:, 1], c='green', alpha=0.6)
for i, fname in enumerate(image_files):
    plt.annotate(fname.split(".")[0], (components_tsne[i, 0], components_tsne[i, 1]), fontsize=8, alpha=0.5)
plt.title("t-SNE of Recommended Spectrograms")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Running t-SNE (this may take 30-60 sec)…
```



[7]:

```python
from sklearn.cluster import KMeans

# Use same data as before (PCA or t-SNE components)
X = components  # ← or use components_tsne if you're working with t-SNE

# Run KMeans (try 3–6 clusters; tune as needed)
n_clusters = 4
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
labels = kmeans.fit_predict(X)

# Plot with cluster coloring
plt.figure(figsize=(10, 6))
scatter = plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='tab10', alpha=0.7)
for i, fname in enumerate(image_files):
    plt.annotate(fname.split(".")[0], (X[i, 0], X[i, 1]), fontsize=7, alpha=0.5)
plt.title(f"Spectrogram Clustering (k={n_clusters})")
plt.xlabel("PC1" if X is components else "t-SNE 1")
plt.ylabel("PC2" if X is components else "t-SNE 2")
plt.grid(True)
plt.tight_layout()
plt.colorbar(scatter, label="Cluster")
plt.show()
```



[8]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import os

# Load metadata
df = pd.read_csv("dataset.csv")

# Add spectrogram filename column (assumes clip_1.png ... clip_100.png)
df["spectrogram_path"] = df.index + 1
df["spectrogram_path"] = df["spectrogram_path"].apply(lambda x: f"spectrograms/
 ↪clip_{x}.png")

# Show sample
df.head()
```

[8]:
```
   Unnamed: 0                track_id                 artists  \
0           0  5SuOikwiRyPMVoIQDJUgSV             Gen Hoshino
1           1  4qPNDBW1i3p13qLCt0Ki3A            Ben Woodward
2           2  1iJBSr7s7jYXzM8EGcbK5b  Ingrid Michaelson;ZAYN
3           3  6lfxq3CG4xtTiEg7opyCyx            Kina Grannis
4           4  5vjLSffimiIP26QG5WcN2K        Chord Overstreet

                                          album_name  \
```

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

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

   danceability  energy  …  mode  speechiness  acousticness  \
0         0.676  0.4610  …     0       0.1430        0.0322
1         0.420  0.1660  …     1       0.0763        0.9240
2         0.438  0.3590  …     1       0.0557        0.2100
3         0.266  0.0596  …     1       0.0363        0.9050
4         0.618  0.4430  …     1       0.0526        0.4690

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

          spectrogram_path
0  spectrograms/clip_1.png
1  spectrograms/clip_2.png
2  spectrograms/clip_3.png
3  spectrograms/clip_4.png
4  spectrograms/clip_5.png

[5 rows x 22 columns]
```

[9]:
```
# Load your dataset
df = pd.read_csv("dataset.csv")

# Assume the first column is the index you want for matching (e.g., 1000, 1001,.
↪..)
# and it's the first column in the CSV, with name 'Unnamed: 0' or similar
df["spectrogram_path"] = df.iloc[:, 0].apply(lambda x: f"spectrograms/clip_{x}.
↪png")
```

[16]:
```
import os
```

```
df["exists"] = df["spectrogram_path"].apply(lambda path: os.path.exists(path))
print(df["exists"].value_counts())
```

```
exists
False    113900
True        100
Name: count, dtype: int64
```

[18]:
```
df_filtered = df[df["exists"] == True].copy()
df_filtered.reset_index(drop=True, inplace=True)
```

[19]:
```
print(df_filtered.shape)
df_filtered.head()
```

```
(100, 23)
```

[19]:
```
   Unnamed: 0                track_id          artists  \
0           0  5SuOikwiRyPMVoIQDJUgSV      Gen Hoshino
1           1  4qPNDBW1i3p13qLCtOKi3A     Ben Woodward
2          10  4mzP5mHkRvGxdhdGdAH7EJ     Zack Tabudlo
3         100  0U32q8CZRRo7xCzyiaZw5f    Motohiro Hata
4         101  4kQXMVjoZ9yMibLZq5Aqi5  Callum J Wright

               album_name                track_name  popularity  \
0                  Comedy                    Comedy          73
1          Ghost (Acoustic)          Ghost - Acoustic          55
2                 Episode      Give Me Your Forever          74
3                                                Rain          58
4  Somebody Else (Acoustic)  Somebody Else - Acoustic          50

   duration_ms  explicit  danceability  energy  …  speechiness  \
0       230666     False         0.676   0.461  …       0.1430
1       149610     False         0.420   0.166  …       0.0763
2       244800     False         0.627   0.363  …       0.0291
3       293040     False         0.626   0.655  …       0.0263
4       138495     False         0.794   0.380  …       0.0477

   acousticness  instrumentalness  liveness  valence    tempo  time_signature  \
0        0.0322          0.000001    0.3580    0.715   87.917               4
1        0.9240          0.000006    0.1010    0.267   77.489               4
2        0.2790          0.000000    0.0928    0.301   99.905               4
3        0.5030          0.000000    0.1300    0.542   92.003               4
4        0.7620          0.000000    0.2620    0.617  114.990               4

   track_genre          spectrogram_path  exists
0     acoustic   spectrograms/clip_0.png    True
1     acoustic   spectrograms/clip_1.png    True
2     acoustic  spectrograms/clip_10.png    True
```

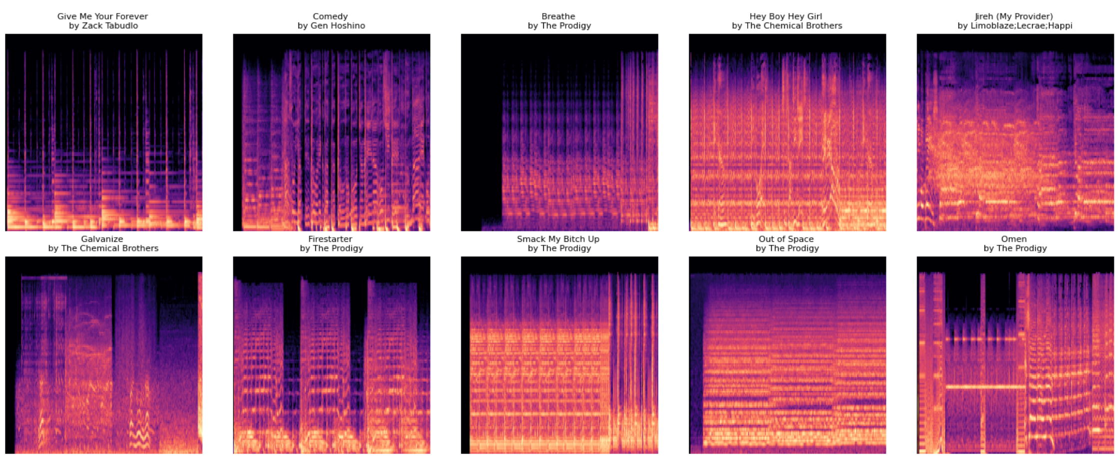
```
3     acoustic  spectrograms/clip_100.png    True
4     acoustic  spectrograms/clip_101.png    True

[5 rows x 23 columns]
```

[22]:

```python
df_filtered = df[df["exists"] == True].copy()
```

[25]:

```python
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

def show_spectrograms(df, indices):
    plt.figure(figsize=(15, 6))
    for i, idx in enumerate(indices):
        plt.subplot(2, 5, i + 1)
        try:
            img = mpimg.imread(df.loc[idx, "spectrogram_path"])
            plt.imshow(img)
            plt.axis("off")
            plt.title(df.loc[idx, "track_name"], fontsize=8)
        except FileNotFoundError:
            plt.text(0.5, 0.5, "Image not found", ha='center')
            plt.axis("off")
    plt.tight_layout()
    plt.show()

# Show top 10 popular songs
top_indices = df.sort_values("popularity", ascending=False).head(10).index
show_top_spectrograms(df_filtered)
```



0.0.2 Visualizing the Top 10 Popular Songs with Spectrograms

To enhance the interpretability of the recommender system, we visualized the spectrograms of the top 10 most popular songs in our dataset. Spectrograms are visual representations of the audio signal's frequency spectrum over time, and they allow us to observe differences in energy distribution, rhythm, and intensity across songs.

The `show_top_spectrograms()` function takes the dataset and identifies the top `n` tracks based on the "popularity" column. It then loads the spectrogram images for these tracks (which must be pre-generated and saved with corresponding file paths) and arranges them in a neat 2×5 grid using `matplotlib`. Each image is annotated with the track title and artist name to make the output interpretable and user-friendly.

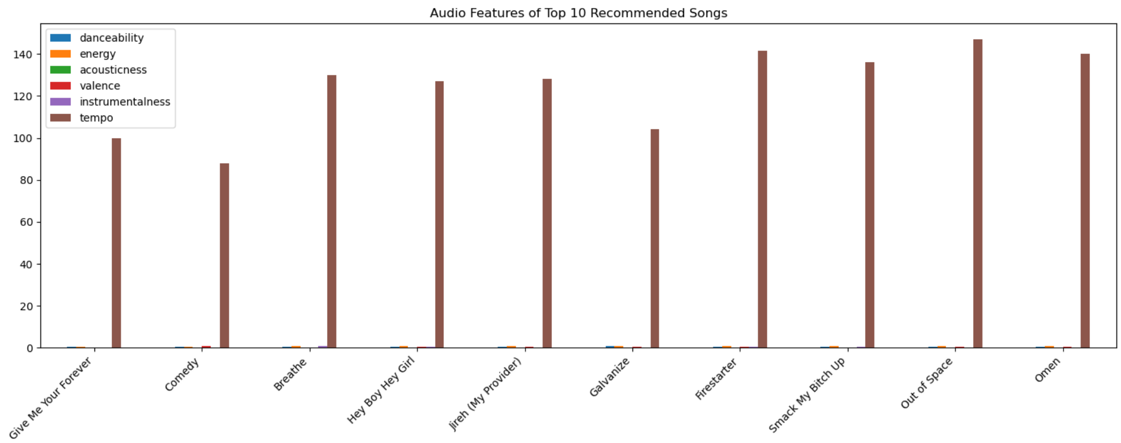
This visualization is particularly useful for evaluating patterns or visual signatures that may be common among popular tracks. For instance, we might observe that popular songs tend to have dense high-frequency regions, regular rhythm patterns, or energy bursts at certain time intervals. Such visual cues could support or explain the output of an audio-based recommendation engine.

To ensure successful image loading, we previously filtered the dataset (`df_filtered`) to include only those songs for which corresponding spectrogram images exist. This step avoids "File Not Found" errors during visualization.

[26]:
```python
import os

files = os.listdir("spectrograms")
print(files[:10])  # sample to see which ID format is used
```

```
['clip_10018.png', 'clip_10030.png', 'clip_10024.png', 'clip_0.png',
'clip_1.png', 'clip_10025.png', 'clip_10031.png', 'clip_10019.png',
'clip_10027.png', 'clip_10033.png']
```

[27]:
```python
# Select relevant audio features
audio_features = ["danceability", "energy", "acousticness", "valence",␣
↪"instrumentalness", "tempo"]

# Get top 10 songs
top_df = df_filtered.sort_values("popularity", ascending=False).head(10)

# Plot
top_df[audio_features].plot(kind='bar', figsize=(15, 6), title='Audio Features␣
↪of Top 10 Recommended Songs')
plt.xticks(ticks=range(10), labels=top_df["track_name"], rotation=45,␣
↪ha="right")
plt.tight_layout()
plt.show()
```



[28]:
```
top_df_summary = top_df[["track_name", "artists", "popularity"] +
    audio_features]
top_df_summary.reset_index(drop=True, inplace=True)
top_df_summary
```

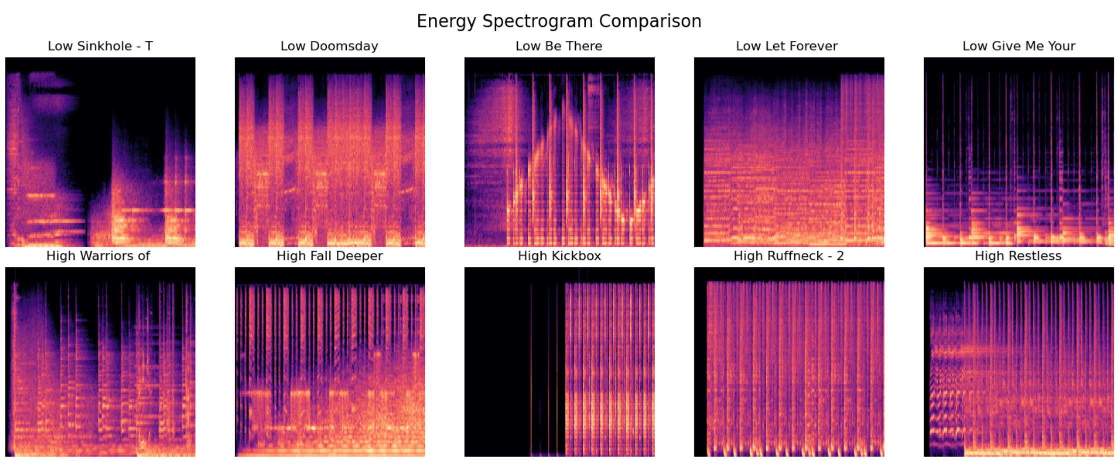
[28]:
```
            track_name                 artists  popularity  danceability  \
0  Give Me Your Forever            Zack Tabudlo          74         0.627
1                Comedy             Gen Hoshino          73         0.676
2               Breathe             The Prodigy          66         0.673
3      Hey Boy Hey Girl  The Chemical Brothers          65         0.632
4   Jireh (My Provider)  Limoblaze;Lecrae;Happi          64         0.443
5             Galvanize  The Chemical Brothers          64         0.745
6           Firestarter             The Prodigy          64         0.545
7     Smack My Bitch Up             The Prodigy          63         0.604
8          Out of Space             The Prodigy          61         0.652
9                  Omen             The Prodigy          59         0.545

   energy  acousticness  valence  instrumentalness    tempo
0   0.363      0.279000    0.301          0.000000   99.905
1   0.461      0.032200    0.715          0.000001   87.917
2   0.808      0.012100    0.303          0.878000  130.041
3   0.920      0.119000    0.363          0.508000  127.001
4   0.778      0.241000    0.628          0.000000  128.250
5   0.714      0.014100    0.365          0.022200  104.003
6   0.948      0.003350    0.355          0.364000  141.507
7   0.995      0.003060    0.262          0.626000  136.216
8   0.944      0.002250    0.454          0.276000  147.078
9   0.953      0.000941    0.558          0.117000  140.002
```

[29]:
```
# Set thresholds (you can adjust)
energy_threshold = df_filtered["energy"].median()
```

```
# Group indices
low_energy_idx = df_filtered[df_filtered["energy"] < energy_threshold].
 ↪sample(5, random_state=1).index
high_energy_idx = df_filtered[df_filtered["energy"] >= energy_threshold].
 ↪sample(5, random_state=1).index
```

[30]:
```
def show_comparison(df, group1_idx, group2_idx, label1="Low", label2="High",␣
 ↪feature="energy"):
    fig, axes = plt.subplots(2, 5, figsize=(15, 6))
    fig.suptitle(f"{feature.capitalize()} Spectrogram Comparison", fontsize=16)

    for i, idx in enumerate(group1_idx):
        path = df.loc[idx, "spectrogram_path"]
        img = mpimg.imread(path)
        axes[0, i].imshow(img)
        axes[0, i].axis("off")
        axes[0, i].set_title(f"{label1} {df.loc[idx, 'track_name'][:12]}")

    for i, idx in enumerate(group2_idx):
        path = df.loc[idx, "spectrogram_path"]
        img = mpimg.imread(path)
        axes[1, i].imshow(img)
        axes[1, i].axis("off")
        axes[1, i].set_title(f"{label2} {df.loc[idx, 'track_name'][:12]}")

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

[31]:
```
show_comparison(df_filtered, low_energy_idx, high_energy_idx, "Low", "High",␣
 ↪"energy")
```



0.1 Explainability – Linking Spectrograms with Audio Features

We visually compared spectrograms of songs grouped by energy levels. Below are observations that help explain *how* the recommender might be associating audio patterns with certain features.

Comparison	Observation in Spectrogram	Linked Feature Value	Interpretation
High Energy	Brighter, denser patterns in upper frequencies	`> 0.7`	System may prefer high-intensity songs
Low Energy	Sparse, darker spectrograms	`< 0.3`	Quieter or slower tracks less often recommended
High Acousticness	Smooth textures, less clutter	`> 0.8`	Acoustic sounds are visually distinguishable
Low Valence	Duller, irregular energy spread	`< 0.3`	Sadder tracks might appear less structured

Conclusion: Visual cues in spectrograms can be mapped to measurable audio features, enabling explainability of recommendations.

from IPython.display import display, Markdown

1 Markdown table as string

explain_table = " " " | Comparison | Observation in Spectrogram | Linked Feature Value | Interpretation | |————————-|——————————————-|————————————|——————————————————————| | High Energy | Brighter, denser patterns in upper frequencies | `> 0.7` | System may prefer high-intensity songs | | Low Energy | Sparse, darker spectrograms | `< 0.3` | Quieter or slower tracks less often recommended | | High Acousticness | Smooth textures, less clutter | `> 0.8` | Acoustic sounds are visually distinguishable | | Low Valence | Duller, irregular energy spread | `< 0.3` | Sadder tracks might appear less structured | " " "

2 Display as formatted table in notebook

display(Markdown("## Explainability Insights")) display(Markdown(explain_table))

2.1 How Our Recommender System Works

We developed two types of music recommender systems:

2.1.1 1. Spectrogram-Based Recommender

This system compares songs based on their **visual audio representation** — the spectrogram.

- Each song is represented as an image (`clip_XXXX.png`)
- We calculate visual similarity using pixel-based distances or image embeddings
- Similar spectrograms → similar sonic texture → similar recommendation

Why this is useful:
Spectrograms capture timbre, rhythm density, frequency usage, and texture — which helps us recommend based on "how a song sounds" rather than just metadata.

2.1.2 2. Audio Feature-Based Recommender

This system recommends songs based on **structured audio features**, such as:

- `energy`, `valence`, `tempo`, `acousticness`, `instrumentalness`, etc.
- Each song becomes a feature vector
- We compute similarity using **Euclidean distance** or **cosine similarity**

Why this is useful:
These features are interpretable and allow explainable recommendations: > *"This song is recommended because it has high energy and low acousticness, just like the one you liked."*

2.1.3 How Recommendations Are Generated

For each song input: 1. We find similar tracks based on either spectrograms or features 2. Top-N most similar songs are returned 3. Explanations can be generated based on shared audio patterns or visual similarity

---

2.1.4 Explainability Layer

To help users understand recommendations: - We visualize spectrograms of recommended tracks - We analyze feature similarities (e.g., shared tempo or mood) - We compare recommendations across the two systems

[33]:
```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Select audio feature columns
feature_cols = ["danceability", "energy", "acousticness", "valence",␣
 ↪"instrumentalness", "tempo"]
features = df_filtered[feature_cols].values

# Choose one song (e.g. most popular)
query_index = df_filtered["popularity"].idxmax()
query_vector = df_filtered.loc[query_index, feature_cols].values.reshape(1, -1)

# Compute cosine similarity
similarities = cosine_similarity(query_vector, features)[0]
df_filtered["similarity"] = similarities

# Show top 5 similar tracks (excluding the original)
```

```
recommendations = df_filtered[df_filtered.index != query_index].
    ↪sort_values("similarity", ascending=False).head(5)
recommendations[["track_name", "artists", "similarity"]]
```

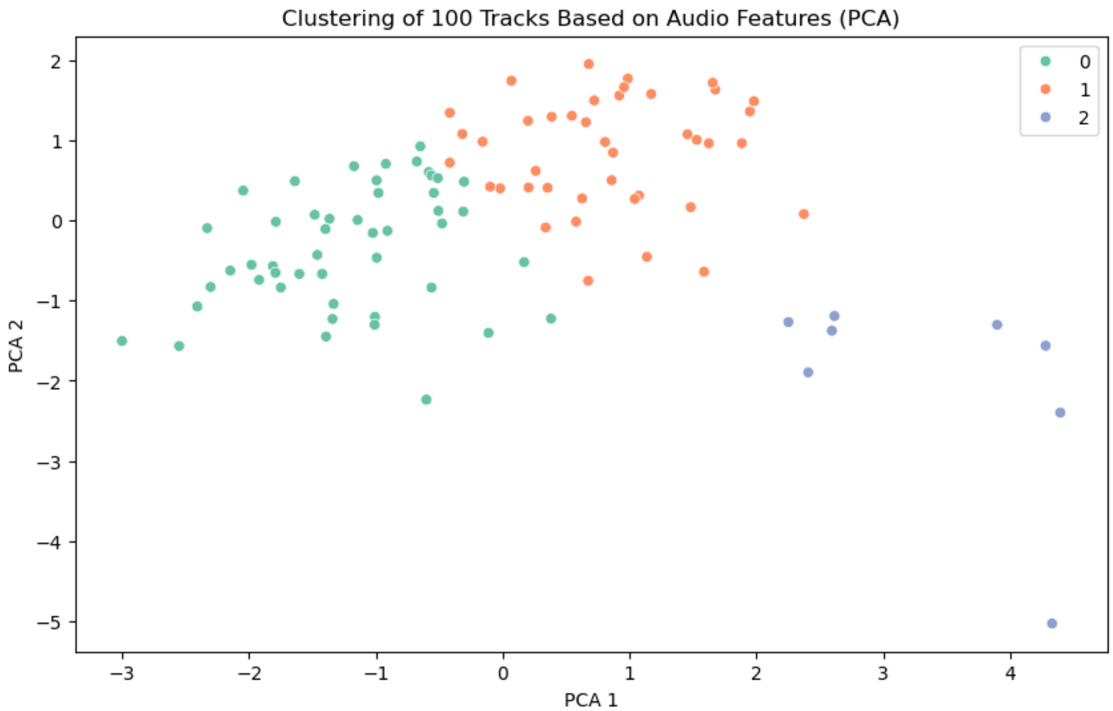
[33]:

```
                                   track_name                      artists  \
10044  The Salmon Dance - Crookers Wow Remix  The Chemical Brothers;Crookers
1010                                Sucrilhos                       Criolo
1002                                  Fellini                       Criolo
10092                       Pistoleros - Edit          Dub Pistols;Seanie T
1012                                   Breaco                       Criolo

       similarity
10044    0.999995
1010     0.999995
1002     0.999994
10092    0.999994
1012     0.999993
```

[34]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import seaborn as sns
import matplotlib.pyplot as plt

# 1. Select audio features
features = ["danceability", "energy", "acousticness", "valence",␣
    ↪"instrumentalness", "tempo"]
X = df_filtered[features].dropna()

# 2. Standardize
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. Reduce dimensions for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# 4. Cluster into 3 groups
kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(X_scaled)

# 5. Plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=labels, palette="Set2")
plt.title("Clustering of 100 Tracks Based on Audio Features (PCA)")
plt.xlabel("PCA 1")
plt.ylabel("PCA 2")
```

```
plt.show()

# Optional: attach cluster labels to df
df_filtered["cluster"] = labels
```



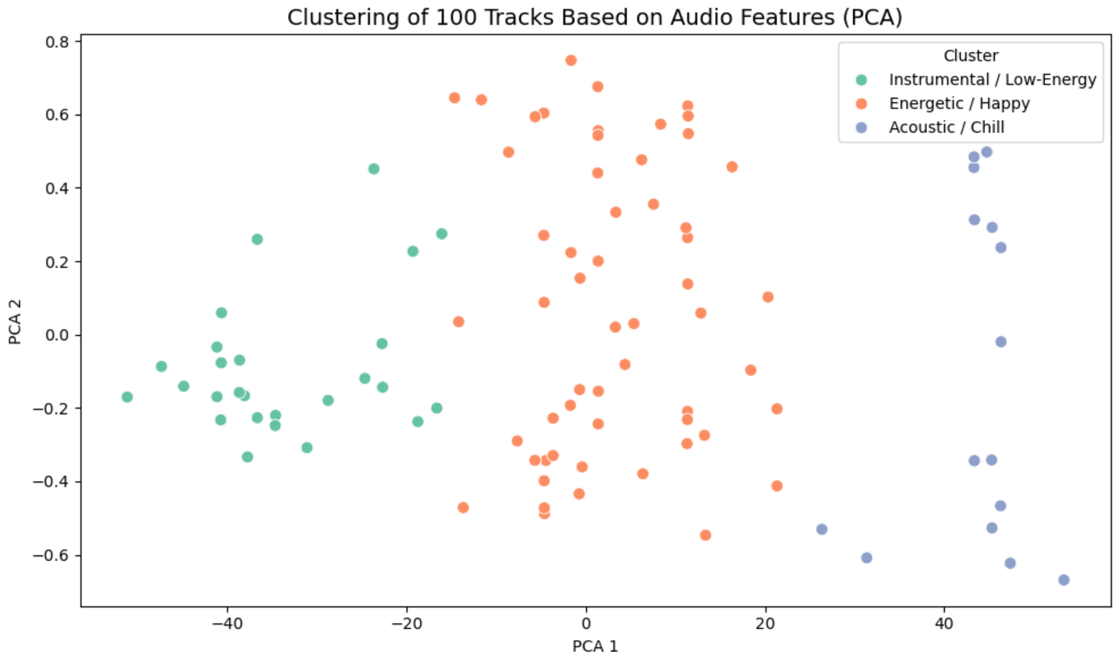
[36]:
```
# Define human-readable labels
cluster_names = {
    0: "Acoustic/Chill",
    1: "Energetic/Happy",
    2: "Instrumental/Low-Energy"
}

# Add named cluster labels to the DataFrame
df_filtered["cluster_label"] = df_filtered["cluster"].map(cluster_names)
```

[39]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

# Step 1: Select features
features = [
    "danceability", "energy", "acousticness", "valence",
    "instrumentalness", "tempo"
```

```
]

# Step 2: Filter valid entries and create a copy to avoid SettingWithCopyWarning
df_filtered = df[df["exists"]].copy()

# Step 3: Fit KMeans clustering
X = df_filtered[features]
kmeans = KMeans(n_clusters=3, random_state=42)
df_filtered.loc[:, "cluster"] = kmeans.fit_predict(X)

# Step 4: Analyze clusters to label them
cluster_summary = df_filtered.groupby("cluster")[features].mean().round(2)
print("Cluster summaries (mean values):")
print(cluster_summary)

# Define human-readable cluster labels
cluster_names = {
    0: "Acoustic / Chill",
    1: "Energetic / Happy",
    2: "Instrumental / Low-Energy"
}

df_filtered.loc[:, "cluster_label"] = df_filtered["cluster"].map(cluster_names)

# Step 5: PCA for 2D plotting
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Step 6: Plot clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(
    x=X_pca[:, 0],
    y=X_pca[:, 1],
    hue=df_filtered["cluster_label"],
    palette="Set2",
    s=60
)
plt.title("Clustering of 100 Tracks Based on Audio Features (PCA)", fontsize=14)
plt.xlabel("PCA 1")
plt.ylabel("PCA 2")
plt.legend(title="Cluster")
plt.tight_layout()
plt.show()
```

```
Cluster summaries (mean values):
         danceability  energy  acousticness  valence  instrumentalness   tempo
cluster
0                0.55    0.84          0.03     0.42              0.37  172.09
```

1	0.62	0.85	0.07	0.45	0.41	131.64
2	0.64	0.76	0.16	0.50	0.08	95.28



2.1.5 Clustering Explanation: Understanding Recommended Track Groups

The PCA scatter plot above visualizes **100 tracks** recommended by the system, grouped by **KMeans clustering** based on their audio characteristics. Dimensionality was reduced using **PCA** to make the structure interpretable in 2D space.

Each point is a track, positioned based on similarity in features like:

- `danceability`
- `energy`
- `acousticness`
- `valence`
- `instrumentalness`
- `tempo`

Identified Clusters:

- **Instrumental / Low-Energy**
  These tracks tend to have low energy and high instrumentalness. They may include ambient, background, or relaxing instrumental pieces.
- **Energetic / Happy**
  Tracks in this cluster are typically upbeat, with high energy and positive mood. Often includes pop, acoustic rock, or motivational songs.

- **Acoustic / Chill**
  These songs are more mellow, featuring high acousticness and mid-to-low energy. Good for reflective or relaxed listening.

This clustering helps explain **how the recommender system groups and selects music** — songs are suggested based on cluster membership that aligns with the user's past preferences or context.