Spotify Data Analytics

1. Import Libraries

In [1]:
```
#import all libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

✅ Task 2.1 — Load the Dataset

In [2]:
```
df_tracks = pd.read_csv("data.csv")
df_artist = pd.read_csv("data_by_artist.csv")
df_genres = pd.read_csv("data_by_genres.csv")
df_year = pd.read_csv("data_by_year.csv")
df_w_genres = pd.read_csv("data_w_genres.csv")

df_tracks.head()
```

Out[2]:

	valence	year	acousticness	artists	danceability	duration_ms	energ
0	0.0594	1921	0.982	['Sergei Rachmaninoff', 'James Levine', 'Berli...	0.279	831667	0.21
1	0.9630	1921	0.732	['Dennis Day']	0.819	180533	0.34
2	0.0394	1921	0.961	['KHP Kridhamardawa Karaton Ngayogyakarta Hadi...	0.328	500062	0.16
3	0.1650	1921	0.967	['Frank Parker']	0.275	210000	0.30
4	0.2530	1921	0.957	['Phil Regan']	0.418	166693	0.19

✅ Task 2.2 —Data Inspection & Cleaning

In [3]:
```
#Basic Inspection
df_tracks.head()
df_tracks.info()
df_tracks.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 170653 entries, 0 to 170652
Data columns (total 19 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   valence           170653 non-null  float64
 1   year              170653 non-null  int64  
 2   acousticness      170653 non-null  float64
 3   artists           170653 non-null  object 
 4   danceability      170653 non-null  float64
 5   duration_ms       170653 non-null  int64  
 6   energy            170653 non-null  float64
 7   explicit          170653 non-null  int64  
 8   id                170653 non-null  object 
 9   instrumentalness  170653 non-null  float64
 10  key               170653 non-null  int64  
 11  liveness          170653 non-null  float64
 12  loudness          170653 non-null  float64
 13  mode              170653 non-null  int64  
 14  name              170653 non-null  object 
 15  popularity        170653 non-null  int64  
 16  release_date      170653 non-null  object 
 17  speechiness       170653 non-null  float64
 18  tempo             170653 non-null  float64
dtypes: float64(9), int64(6), object(4)
memory usage: 24.7+ MB
```

Out[3]:

	valence	year	acousticness	danceability	duration_m
count	170653.000000	170653.000000	170653.000000	170653.000000	1.706530e+0
mean	0.528587	1976.787241	0.502115	0.537396	2.309483e+0
std	0.263171	25.917853	0.376032	0.176138	1.261184e+0
min	0.000000	1921.000000	0.000000	0.000000	5.108000e+0
25%	0.317000	1956.000000	0.102000	0.415000	1.698270e+0
50%	0.540000	1977.000000	0.516000	0.548000	2.074670e+0
75%	0.747000	1999.000000	0.893000	0.668000	2.624000e+0
max	1.000000	2020.000000	0.996000	0.988000	5.403500e+0

In [4]:
```
#Check for Missing Values
df_tracks.isnull().sum()
```

Out[4]:

	0
valence	0
year	0
acousticness	0
artists	0
danceability	0
duration_ms	0
energy	0
explicit	0
id	0
instrumentalness	0
key	0
liveness	0
loudness	0
mode	0
name	0
popularity	0
release_date	0
speechiness	0
tempo	0

dtype: int64

In [5]:
```
#If missing values exist,clean them:
df_tracks = df_tracks.dropna()
print("Missing values removed.")
```

```
Missing values removed.
```

In [6]:
```
#Check for Duplicate Rows
df_tracks.duplicated().sum()
```

Out[6]:
```
np.int64(0)
```

In [7]:
```
#Remove duplicate
df_tracks = df_tracks.drop_duplicates()

#Other Datasets are also cleaned Similarly
df_artist = df_artist.dropna().drop_duplicates()
```

```
df_genres = df_genres.dropna().drop_duplicates()
df_year = df_year.dropna().drop_duplicates()
df_w_genres = df_w_genres.dropna().drop_duplicates()

print("Duplicate rows removed.")
```

```
Duplicate rows removed.
```

✅ Task 2.3 — Perform Exploratory Data Analysis (Basic)

1.Check the Shape of Each Dataset

In [8]:
```
print("Tracks dataset shape:", df_tracks.shape)
print("Artist dataset shape:", df_artist.shape)
print("Genres dataset shape:", df_genres.shape)
print("Year dataset shape:", df_year.shape)
print("Tracks-with-genres dataset shape:", df_w_genres.shape)
```

```
Tracks dataset shape: (170653, 19)
Artist dataset shape: (28680, 15)
Genres dataset shape: (2973, 14)
Year dataset shape: (100, 14)
Tracks-with-genres dataset shape: (28680, 16)
```

In [9]:
```
#View Summary Statistics
df_tracks.describe()
```

Out[9]:

	valence	year	acousticness	danceability	duration_m
count	170653.000000	170653.000000	170653.000000	170653.000000	1.706530e+0
mean	0.528587	1976.787241	0.502115	0.537396	2.309483e+0
std	0.263171	25.917853	0.376032	0.176138	1.261184e+0
min	0.000000	1921.000000	0.000000	0.000000	5.108000e+0
25%	0.317000	1956.000000	0.102000	0.415000	1.698270e+0
50%	0.540000	1977.000000	0.516000	0.548000	2.074670e+0
75%	0.747000	1999.000000	0.893000	0.668000	2.624000e+0
max	1.000000	2020.000000	0.996000	0.988000	5.403500e+0

In [10]:
```
#check feature types
df_tracks.dtypes
```

```
Out[10]:
```

	0
valence	float64
year	int64
acousticness	float64
artists	object
danceability	float64
duration_ms	int64
energy	float64
explicit	int64
id	object
instrumentalness	float64
key	int64
liveness	float64
loudness	float64
mode	int64
name	object
popularity	int64
release_date	object
speechiness	float64
tempo	float64

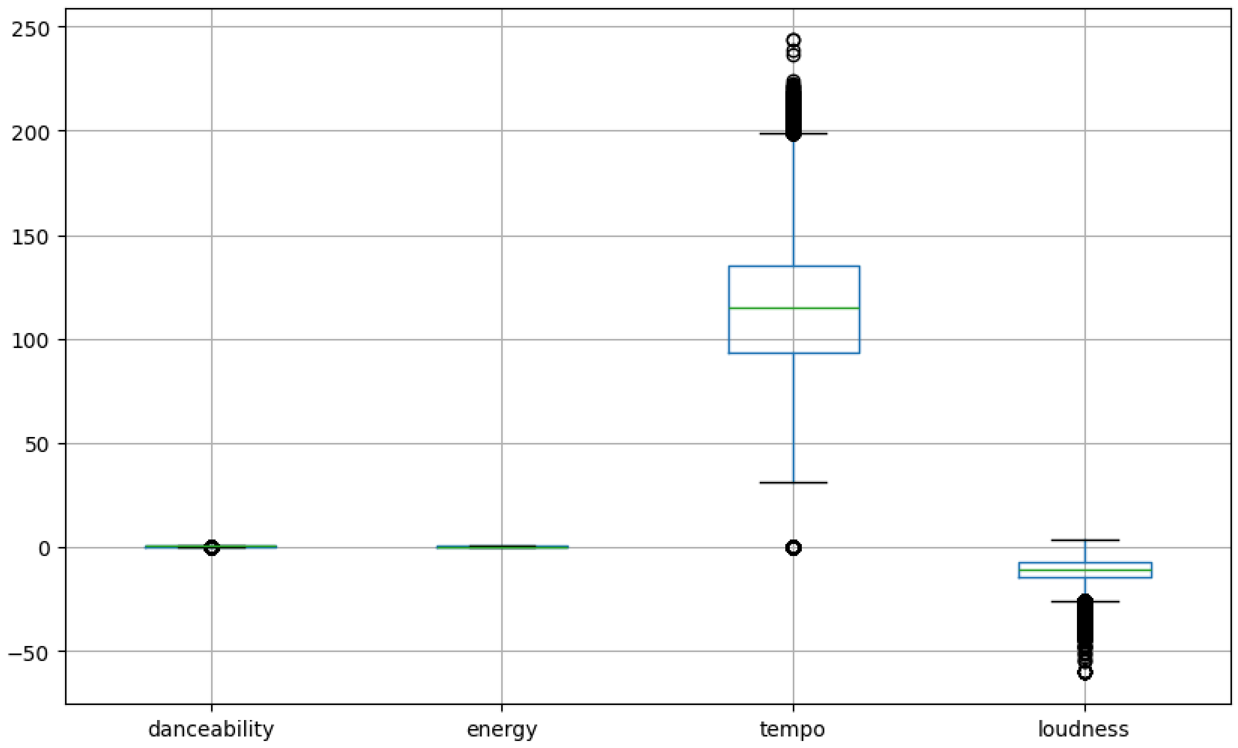
dtype: object

```
In [11]: #count unique values
         print("Unique artists:", df_tracks['artists'].nunique())
         print("Unique genres:", df_genres['genres'].nunique())
         print("Years covered:", df_year['year'].min(), "to", df_year['year'].max())
```

```
Unique artists: 34088
Unique genres: 2973
Years covered: 1921 to 2020
```

```
In [12]: #identify outliers
         df_tracks[['danceability','energy','tempo','loudness']].boxplot(figsize=(10,6)
```

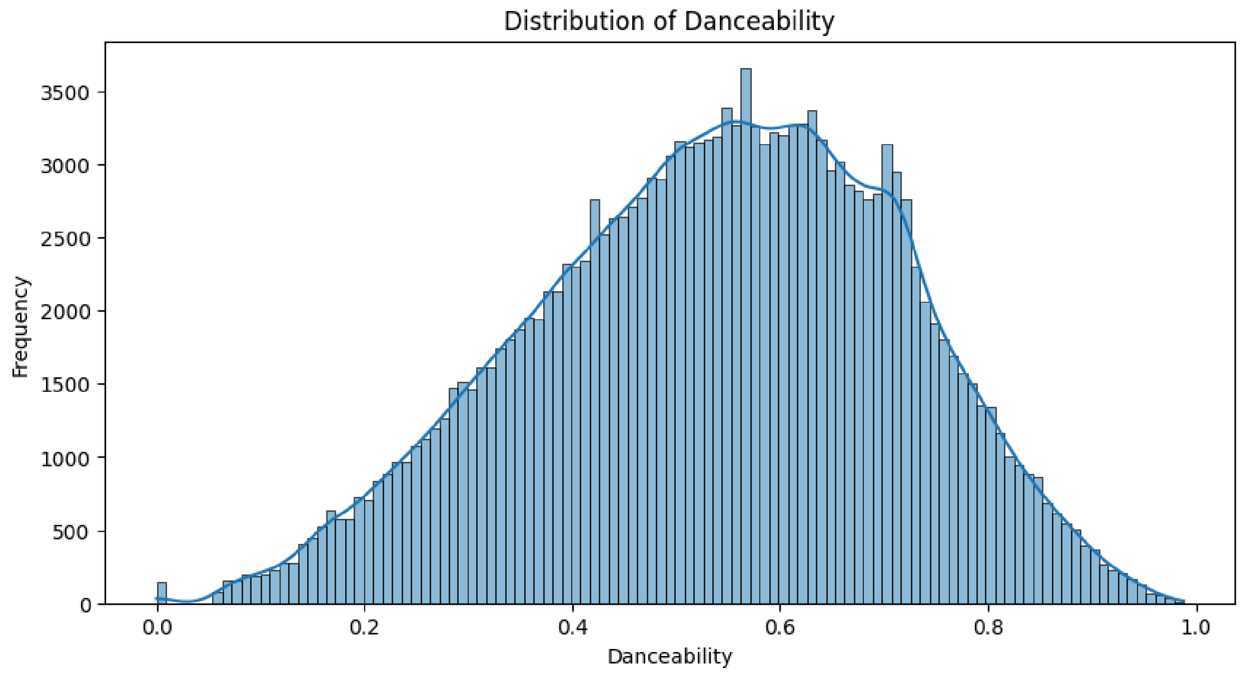
```
Out[12]: <Axes: >
```

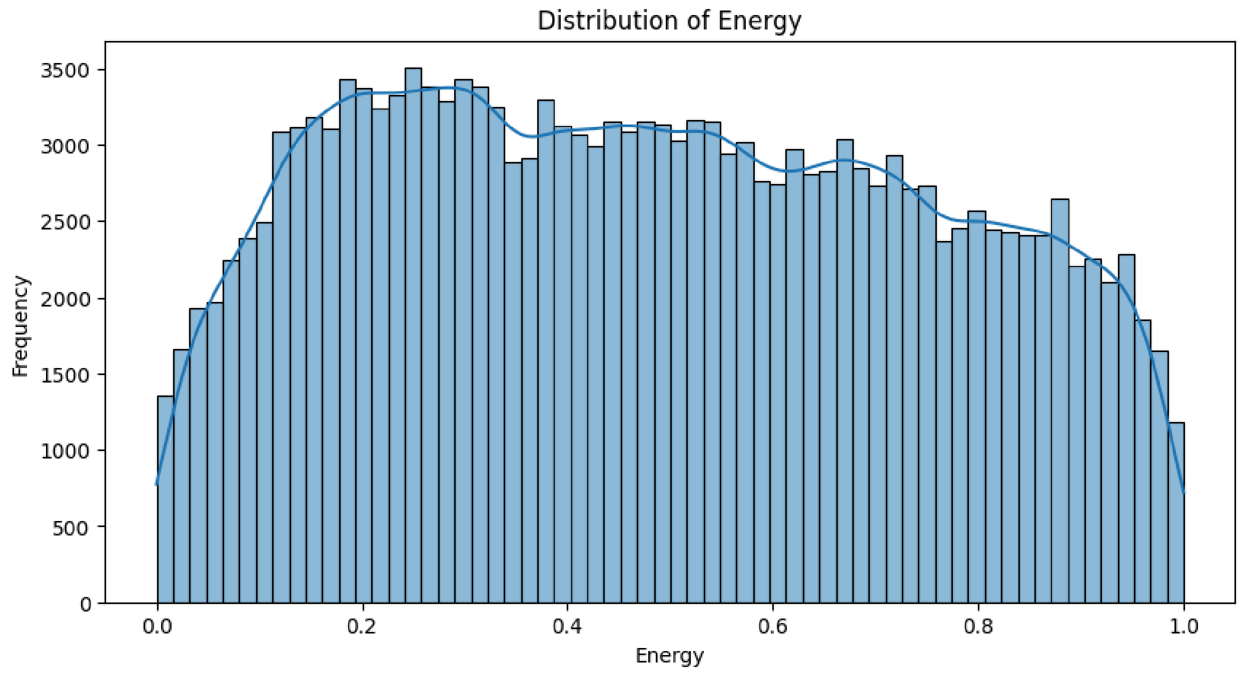


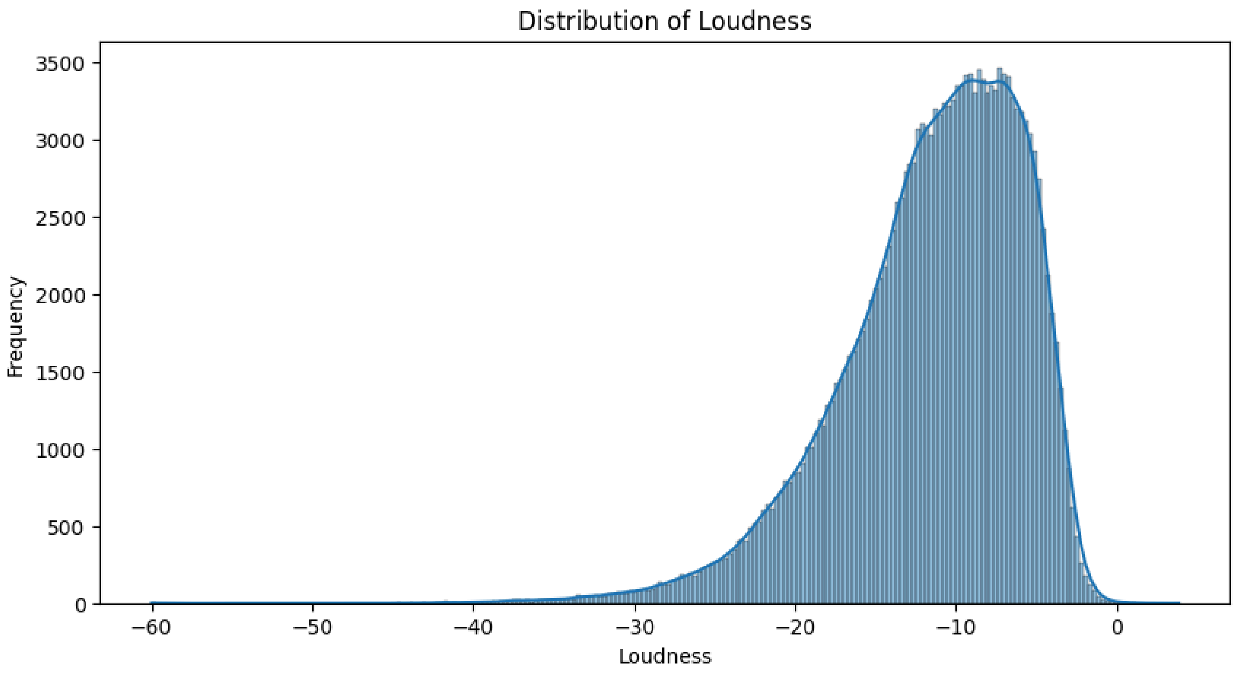
Data AnalysisVisualization

Task 3.1: Analyze the distribution of various features (e.g., danceability, energy, tempo).

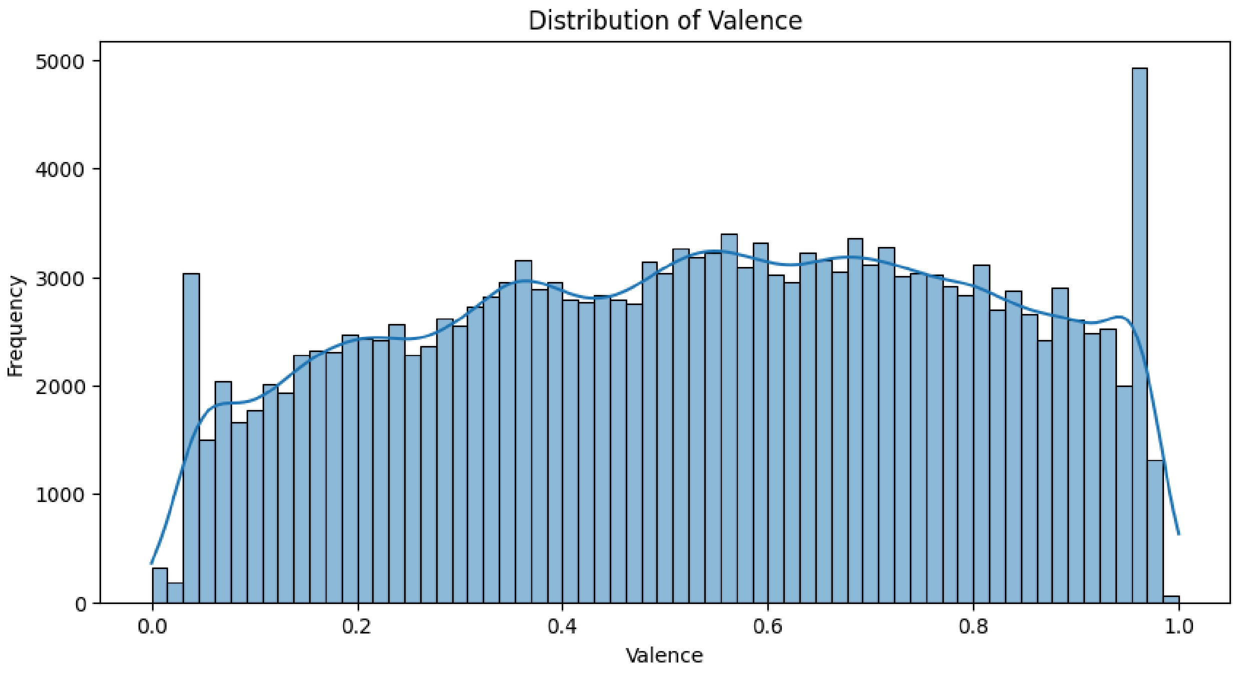
In [13]:
```
import matplotlib.pyplot as plt
import seaborn as sns

audio_features = ["danceability", "energy", "loudness", "valence", "tempo"]

for feature in audio_features:
    plt.figure(figsize=(10,5))
    sns.histplot(df_tracks[feature], kde=True)
    plt.title(f"Distribution of {feature.capitalize()}")
    plt.xlabel(feature.capitalize())
    plt.ylabel("Frequency")
    plt.show()
```


Distribution of Danceability
Frequency
Danceability
0
500
1000
1500
2000
2500
3000
3500
0.0
0.2
0.4
0.6
0.8
1.0

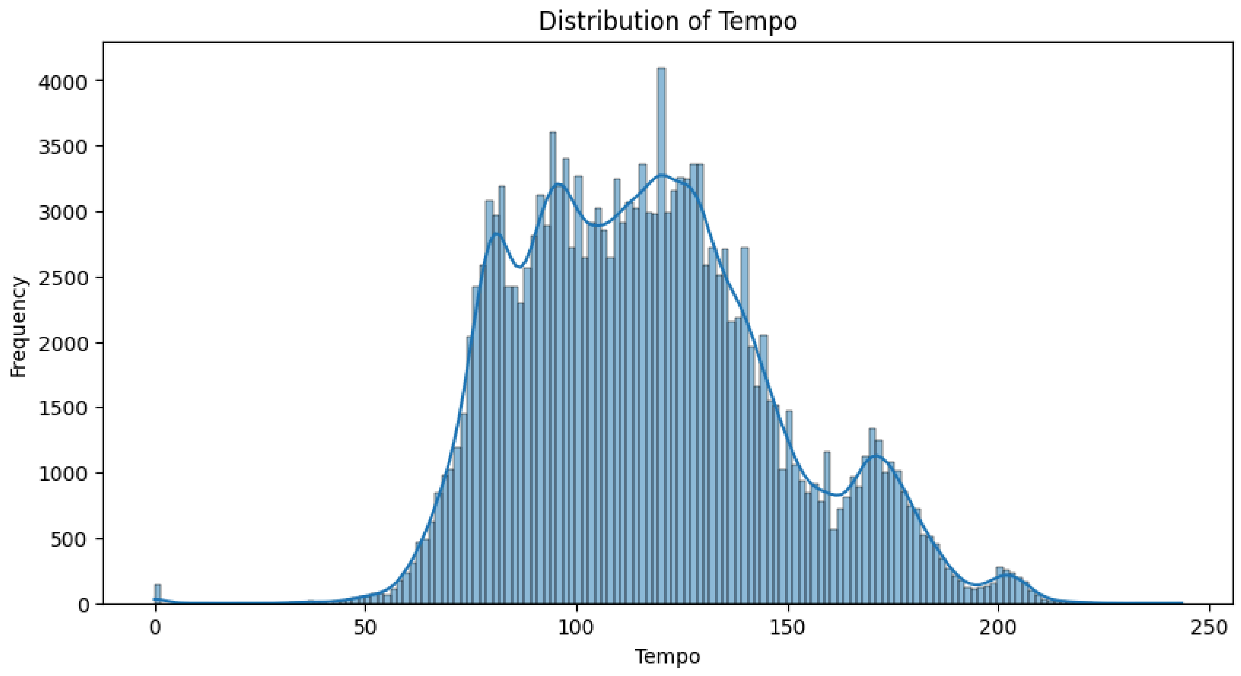

Distribution of Energy
Frequency
Energy
0
500
1000
1500
2000
2500
3000
3500
0.0
0.2
0.4
0.6
0.8
1.0



Distribution of Loudness
Frequency
Loudness
0
500
1000
1500
2000
2500
3000
3500
−60
−50
−40
−30
−20
−10
0

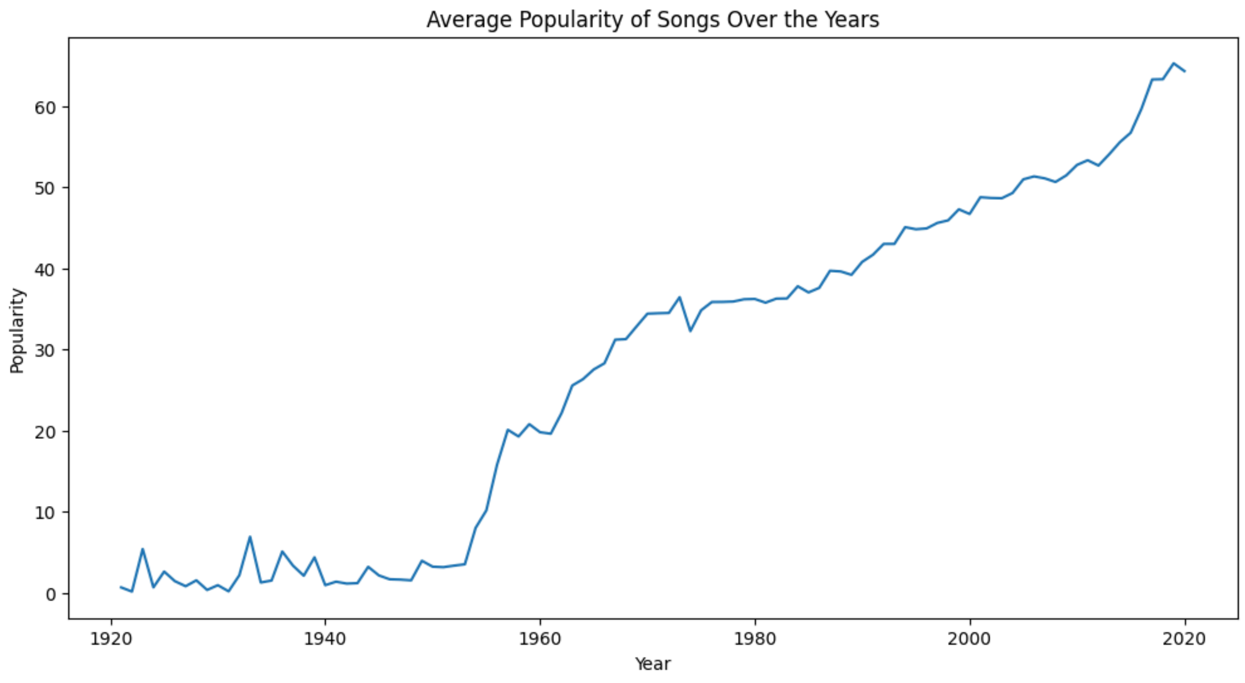


Distribution of Valence
Frequency
Valence
0
1000
2000
3000
4000
5000
0.0
0.2
0.4
0.6
0.8
1.0



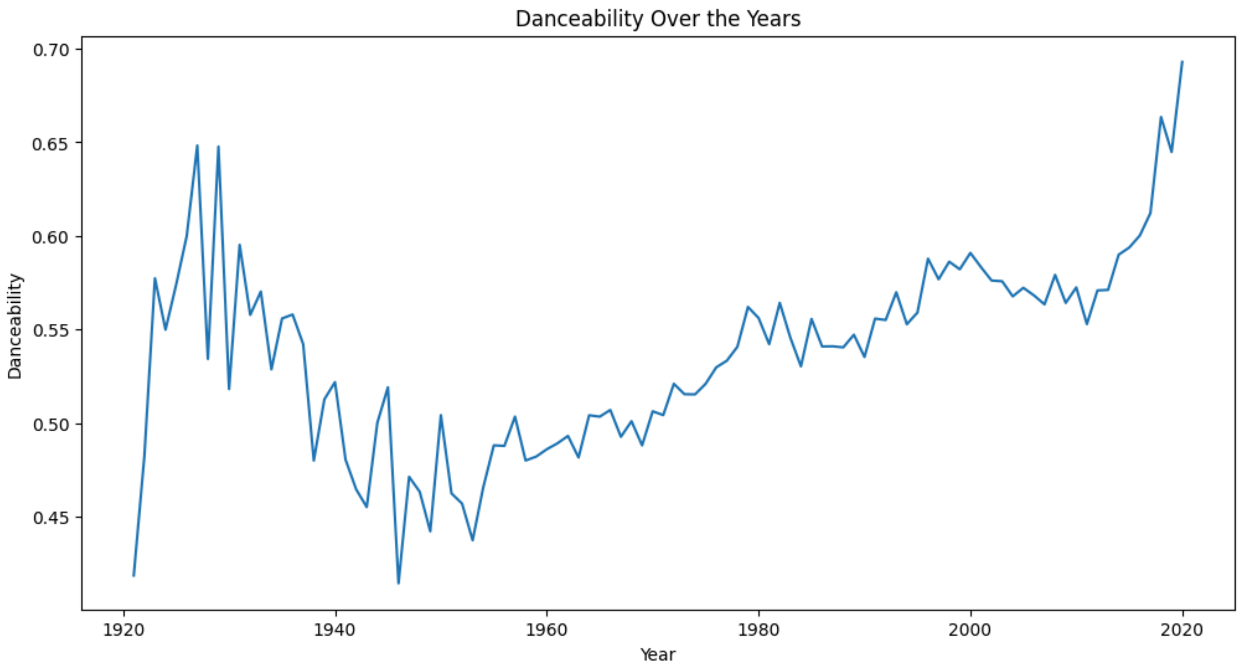
Task 3.2: Identify trends over time (e.g., popularity ofgenres, changes in song features).

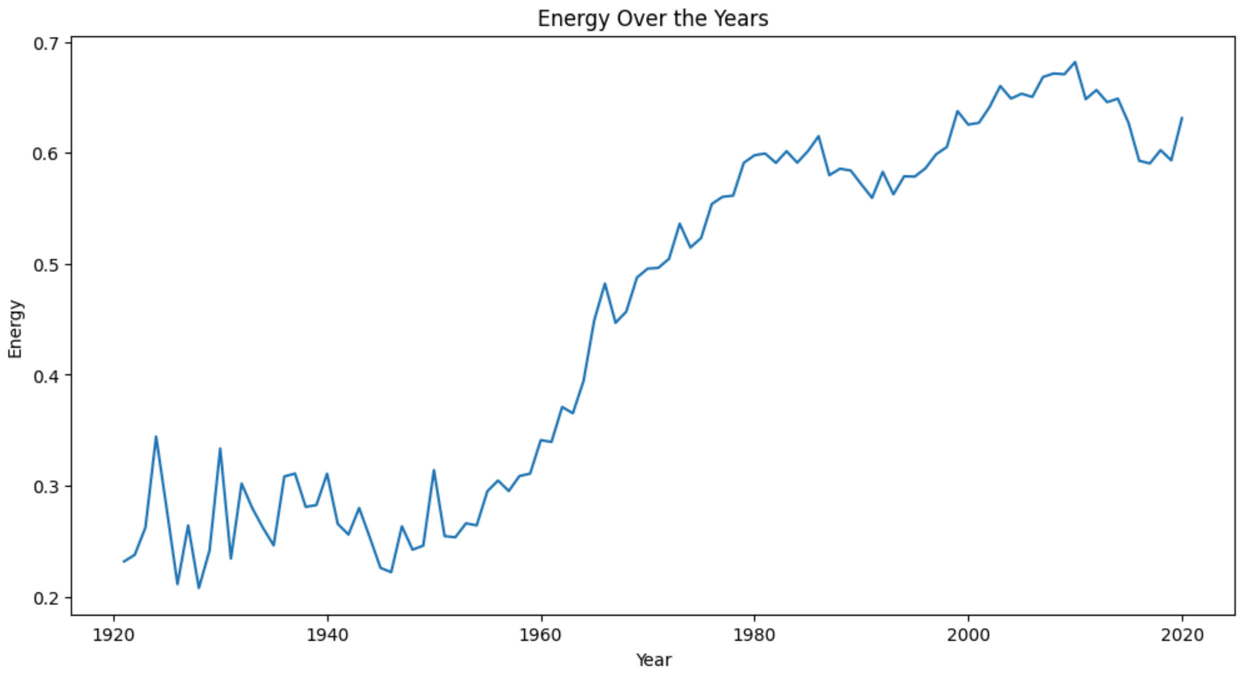
In [14]:
```python
# Trend of Average Popularity Over the Years
plt.figure(figsize=(12,6))
sns.lineplot(data=df_year, x="year", y="popularity")
plt.title("Average Popularity of Songs Over the Years")
plt.xlabel("Year")
plt.ylabel("Popularity")
plt.show()
```

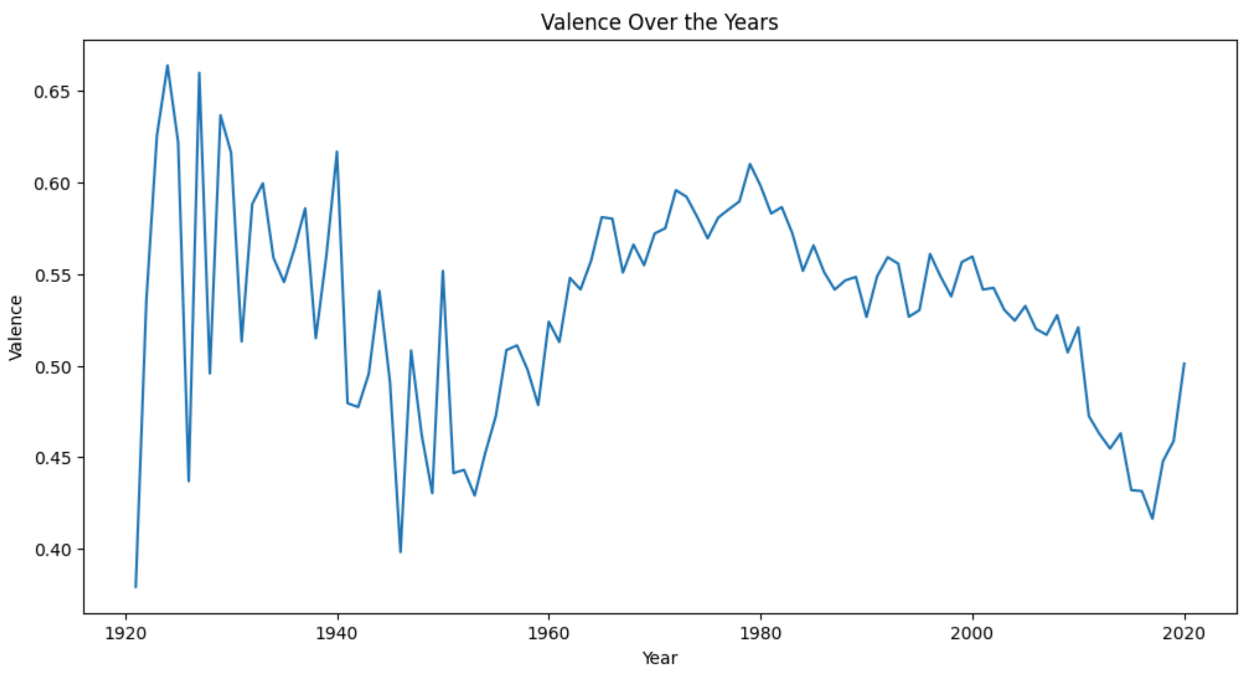


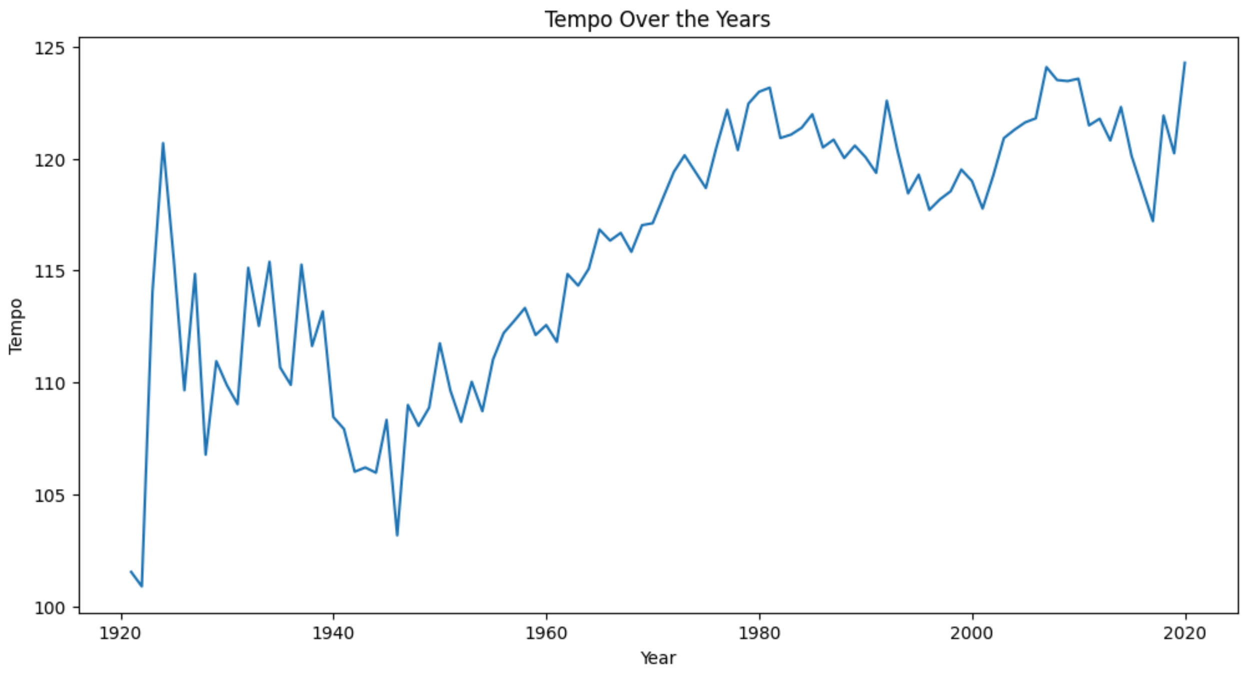
Audio Feature Trends Over Time

In [15]:
```
yearly_features = ["danceability", "energy", "valence", "tempo"]

for feature in yearly_features:
    plt.figure(figsize=(12,6))
    sns.lineplot(data=df_year, x="year", y=feature)
    plt.title(f"{feature.capitalize()} Over the Years")
    plt.xlabel("Year")
    plt.ylabel(feature.capitalize())
    plt.show()
```



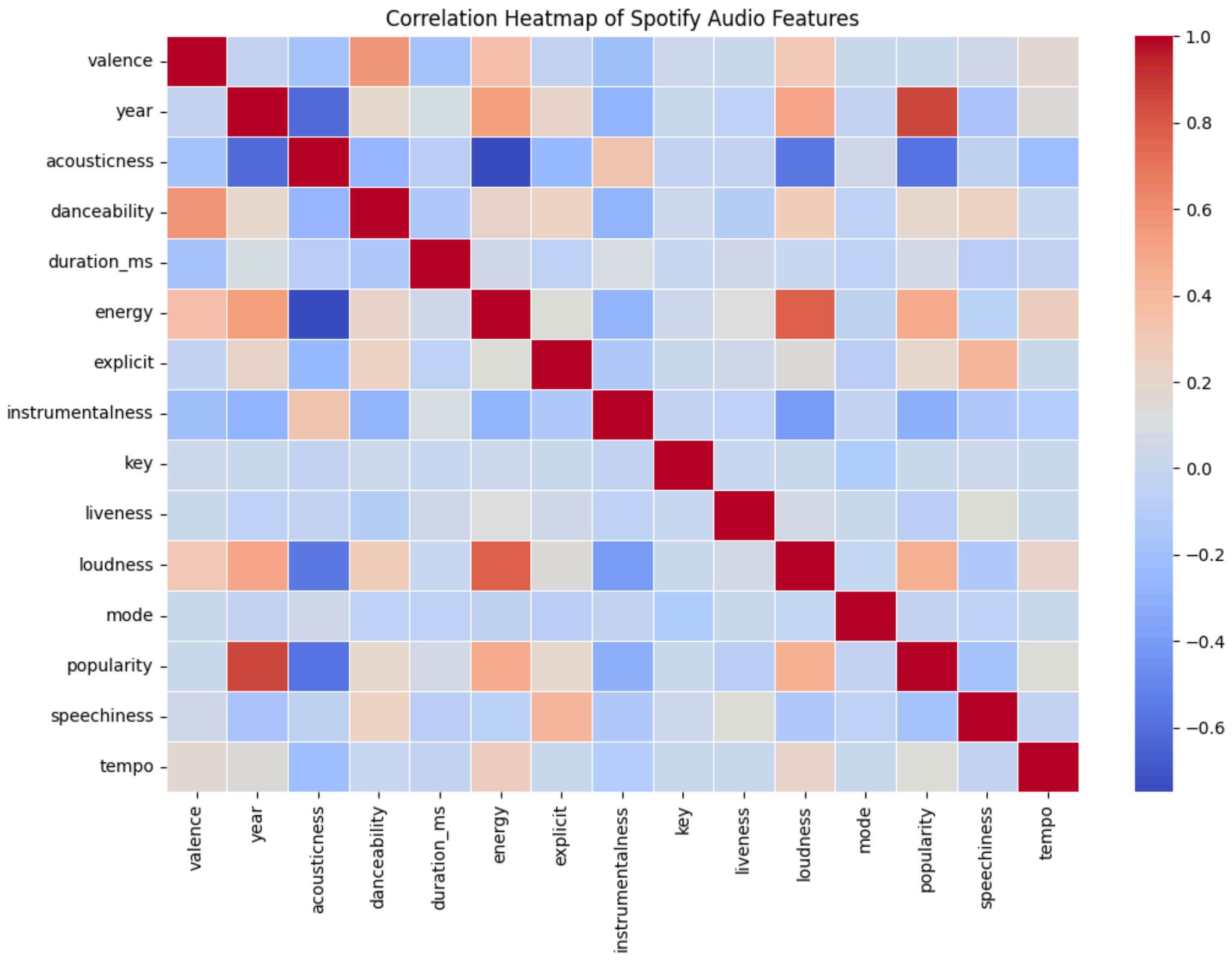






Task 3.3: Correlation Between Audio Features

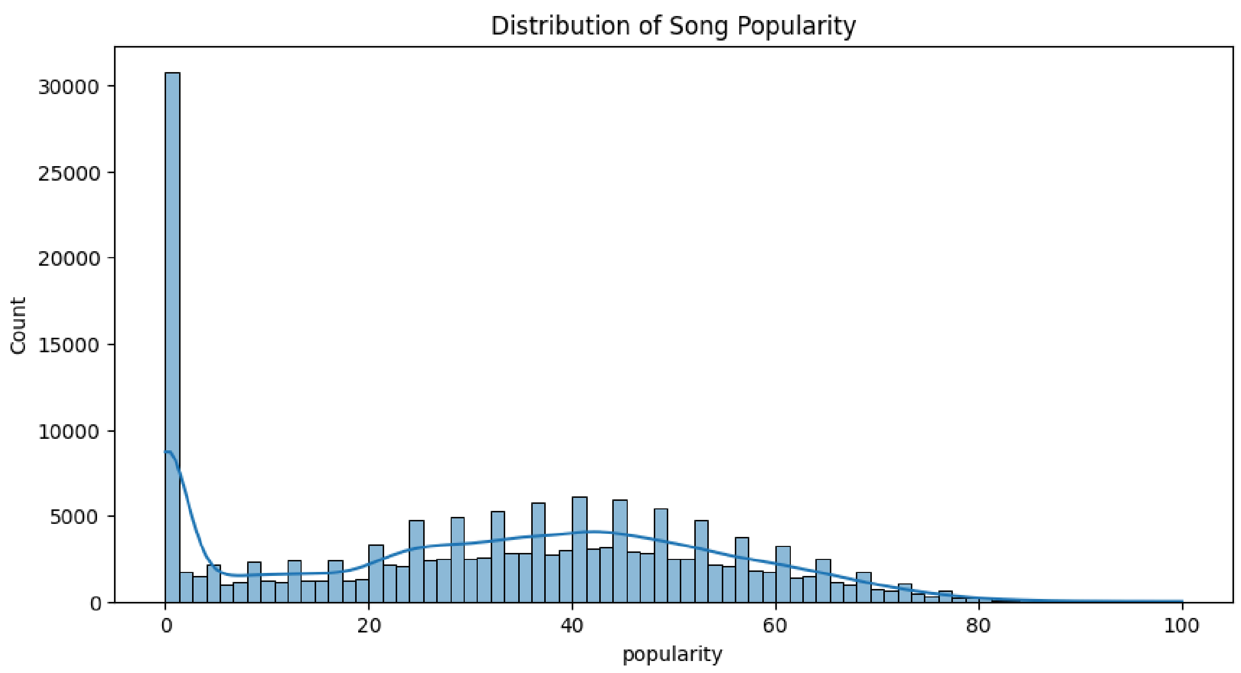
In [16]:
```
#Correlation analysis helps identify how different musical attributes relate t
plt.figure(figsize=(12,8))
corr = df_tracks.corr(numeric_only=True)
sns.heatmap(corr, annot=False, cmap="coolwarm",linewidth=0.5)
plt.title("Correlation Heatmap of Spotify Audio Features")
plt.show()
```



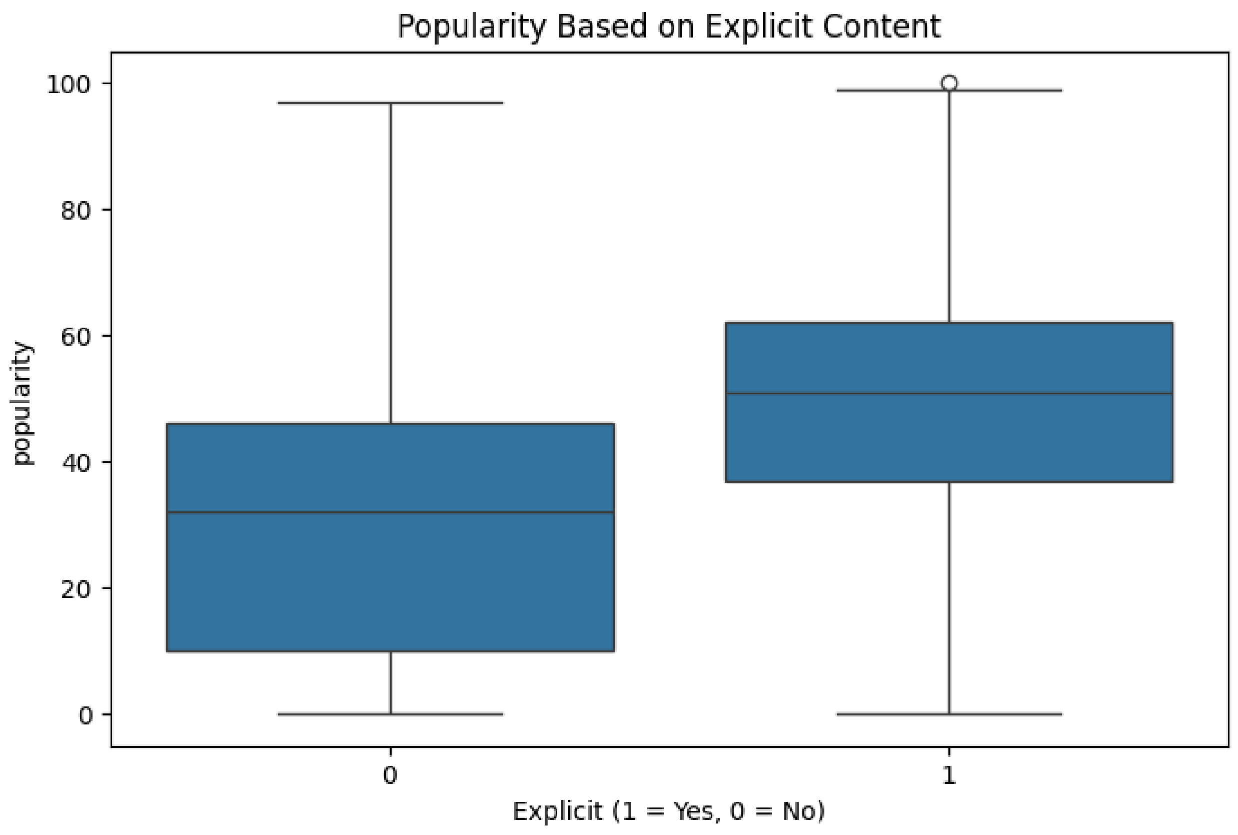
Task 3.4: Analyze user preferences and listening habits.

Popularity reflects listener engagement. This section uncovers what kinds of songs listeners prefer.

In [17]:
```
#1. Popularity Distribution
plt.figure(figsize=(10,5))
sns.histplot(df_tracks["popularity"], kde=True)
plt.title("Distribution of Song Popularity")
plt.show()
```

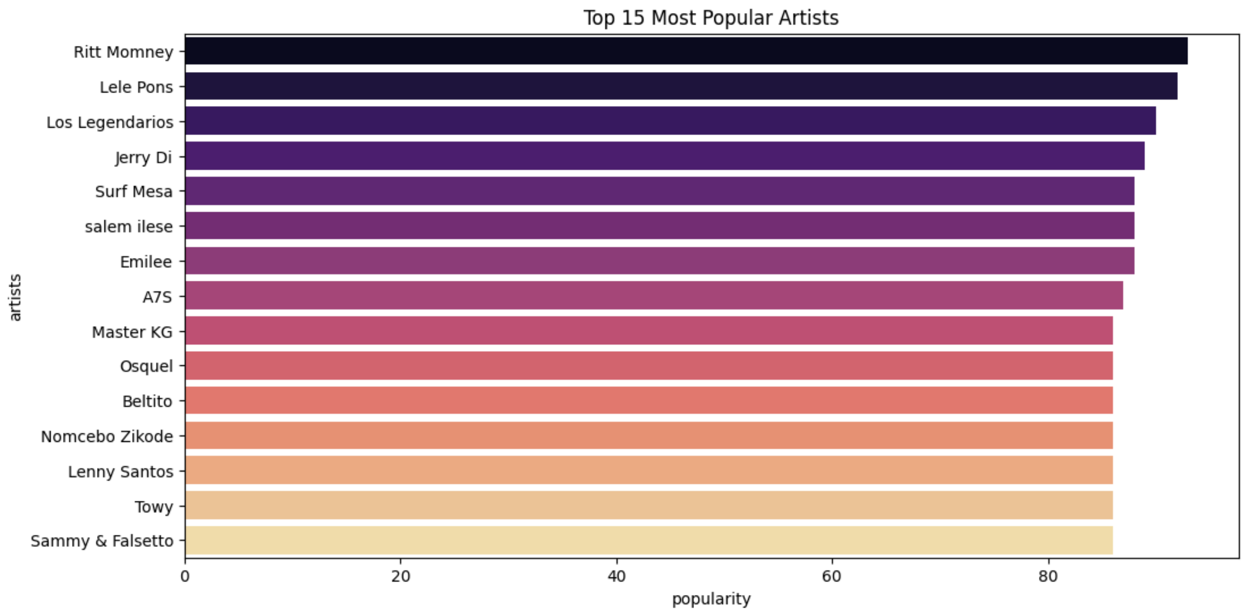


```
In [18]: #2. Popularity by Explicit vs Non-Explicit Songs
         plt.figure(figsize=(8,5))
         sns.boxplot(data=df_tracks, x="explicit", y="popularity")
         plt.title("Popularity Based on Explicit Content")
         plt.xlabel("Explicit (1 = Yes, 0 = No)")
         plt.show()
```



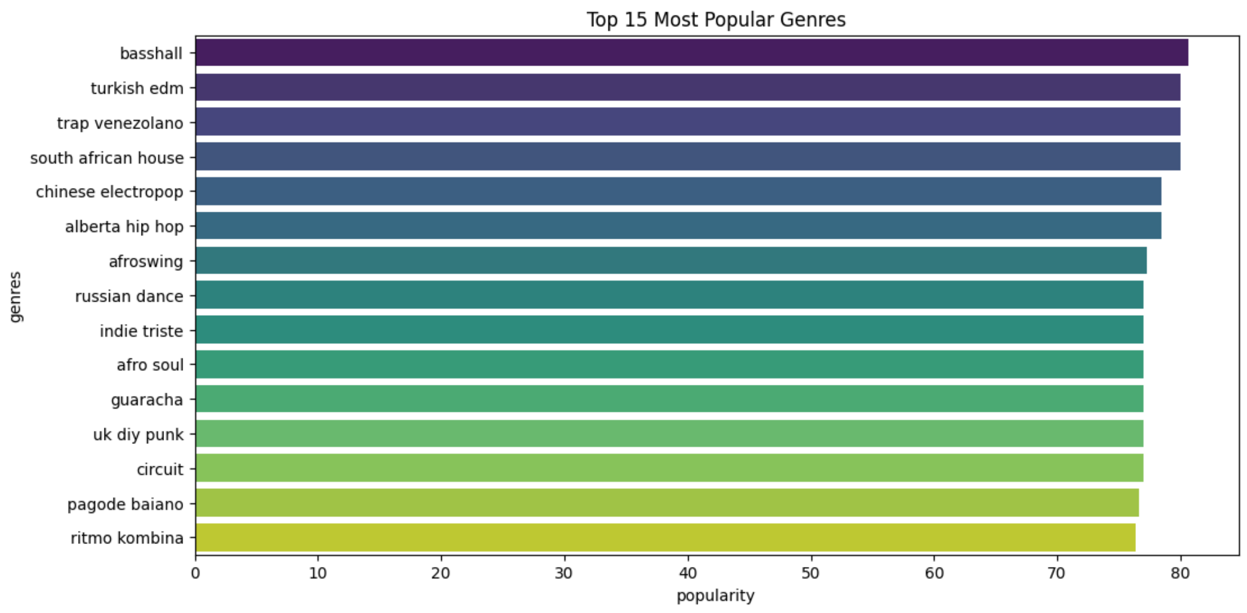
```
In [19]: #3. Which Artists Are Most Popular?
         top_artists = df_artist.sort_values("popularity", ascending=False).head(15)

         plt.figure(figsize=(12,6))
         sns.barplot(data=top_artists, x="popularity", y="artists",palette="magma")
         plt.title("Top 15 Most Popular Artists")
         plt.show()
```

```
/tmp/ipython-input-2754866546.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same e
ffect.

  sns.barplot(data=top_artists, x="popularity", y="artists",palette="magma")
```



```
In [20]: #4. Most Popular Genres
         top_genres = df_genres.sort_values("popularity", ascending=False).head(15)

         plt.figure(figsize=(12,6))
         sns.barplot(data=top_genres, x="popularity", y="genres",palette="viridis")
         plt.title("Top 15 Most Popular Genres")
         plt.show()
```

```
/tmp/ipython-input-1427288677.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same e
ffect.

  sns.barplot(data=top_genres, x="popularity", y="genres",palette="viridis")
```

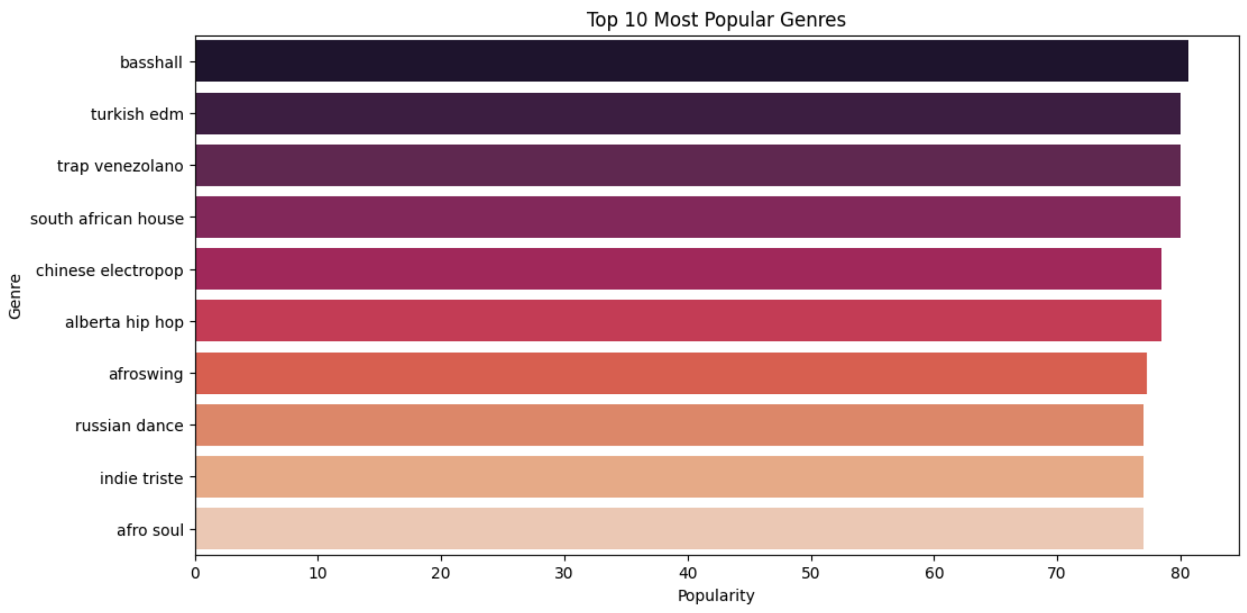


Data Analysis Visualization

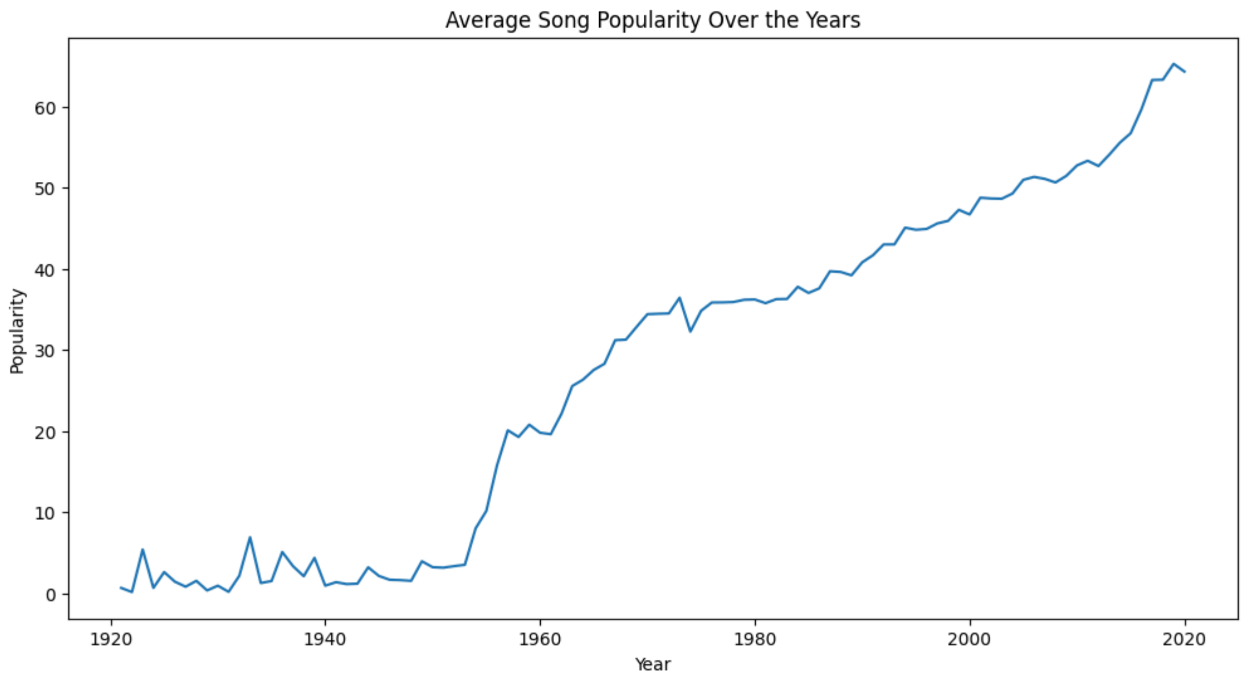
Task 4.1: Create visualizations to represent keyfindings (e.g., bar charts, line graphs, scatter plots).

In [21]:
```
#1. Bar Chart – Top 10 Most Popular Genres
top_genres = df_genres.sort_values("popularity", ascending=False).head(10)

plt.figure(figsize=(12,6))
sns.barplot(data=top_genres, x="popularity", y="genres",palette="rocket")
plt.title("Top 10 Most Popular Genres")
plt.xlabel("Popularity")
plt.ylabel("Genre")
plt.show()
```

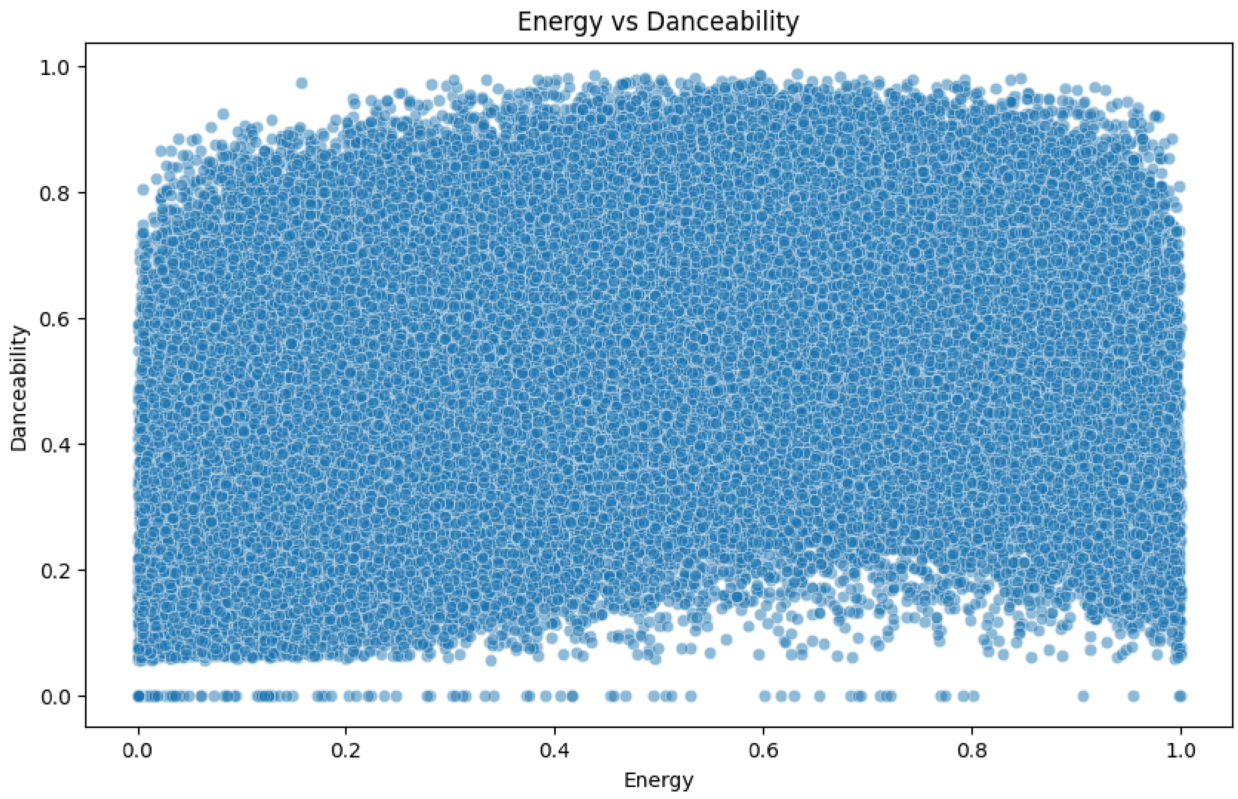
```
/tmp/ipython-input-387303681.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same e
ffect.

  sns.barplot(data=top_genres, x="popularity", y="genres",palette="rocket")
```



In [22]:
```
#2. Line Chart – Popularity Trend Over the Years
plt.figure(figsize=(12,6))
sns.lineplot(data=df_year, x='year', y='popularity')
plt.title("Average Song Popularity Over the Years")
plt.xlabel("Year")
plt.ylabel("Popularity")
plt.show()
```

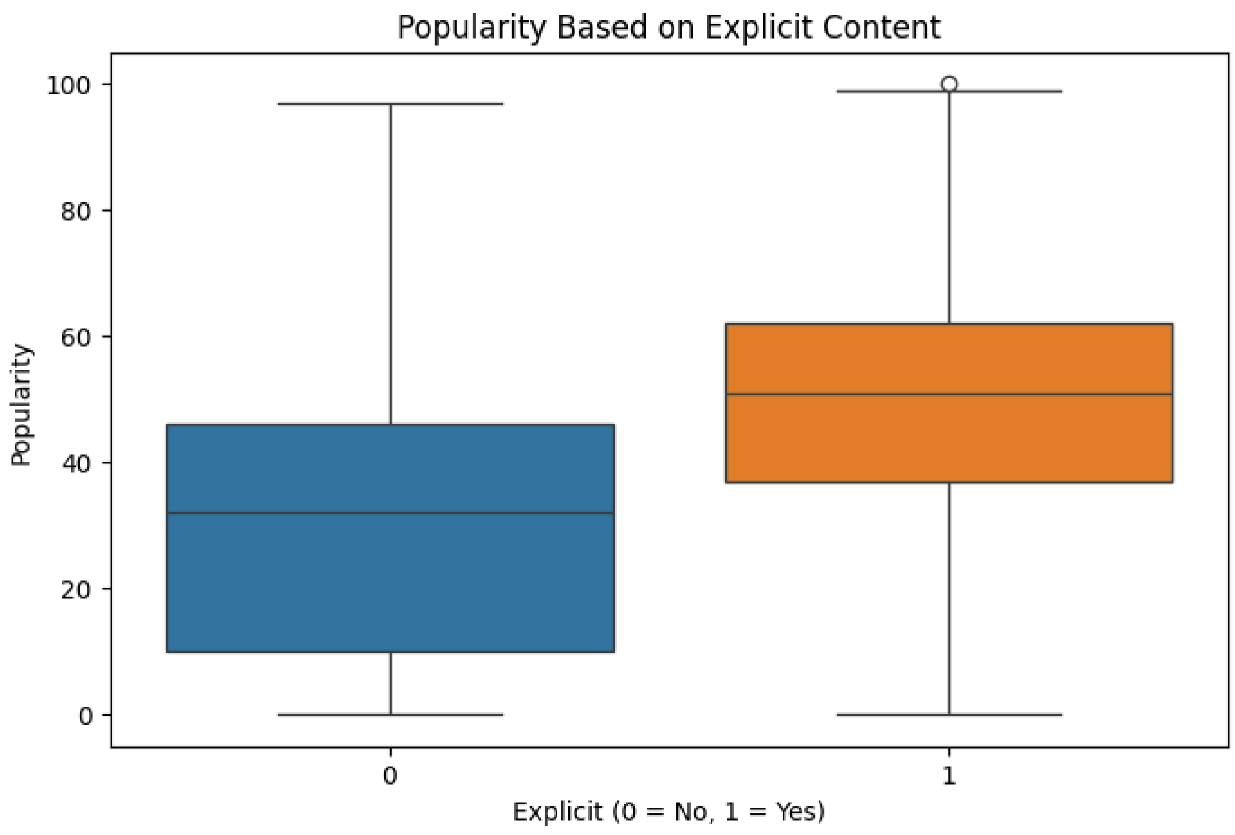


In [23]:
```
#Scatter Plot – Energy vs. Danceability
plt.figure(figsize=(10,6))
sns.scatterplot(data=df_tracks, x='energy', y='danceability', alpha=0.5)
plt.title("Energy vs Danceability")
plt.xlabel("Energy")
plt.ylabel("Danceability")
plt.show()
```



In [24]:
```
#4. Boxplot — Popularity by Explicit Content
plt.figure(figsize=(8,5))
sns.boxplot(
    data=df_tracks,
    x='explicit',
    y='popularity',
    palette=["#1f77b4", "#ff7f0e"]   # custom colors
)
plt.title("Popularity Based on Explicit Content")
plt.xlabel("Explicit (0 = No, 1 = Yes)")
plt.ylabel("Popularity")
plt.show()
```

```
/tmp/ipython-input-1009819500.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same e
ffect.

  sns.boxplot(
```

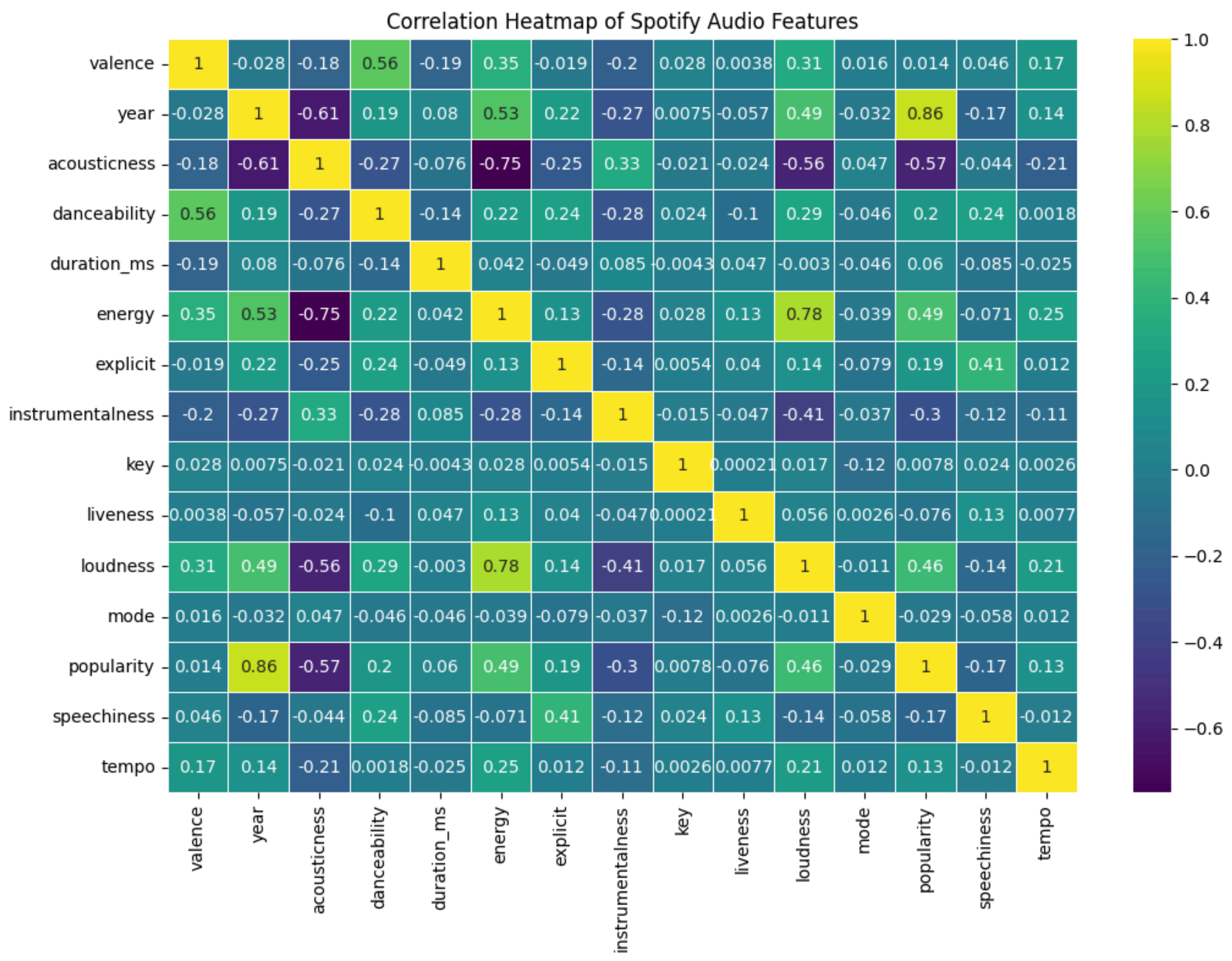


Task 4.2 — Advanced Visualization Techniques

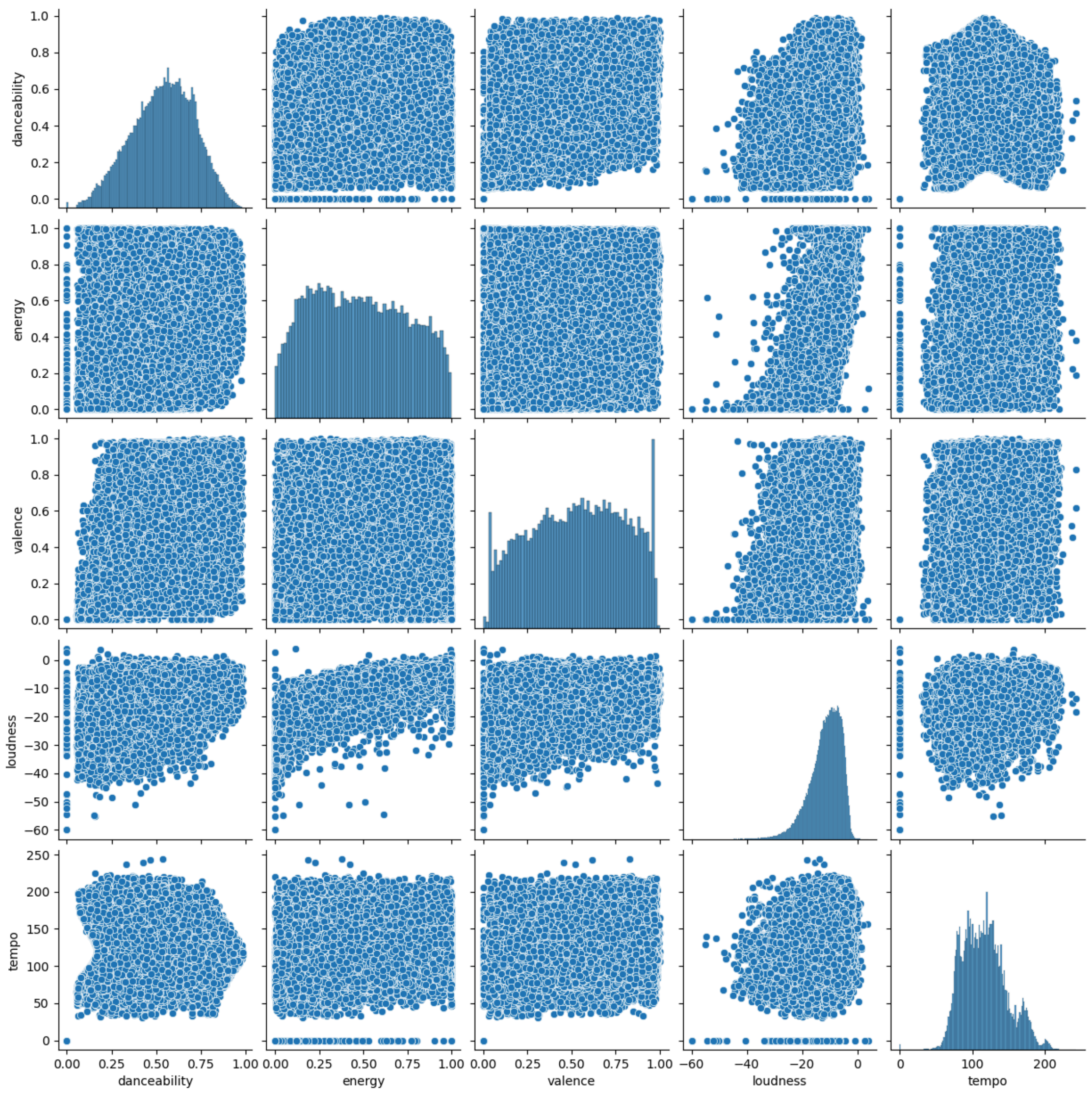
Task 3.2 Popularity Over the Years

Identify trends over time (e.g., popularity ofgenres, changes in song features).

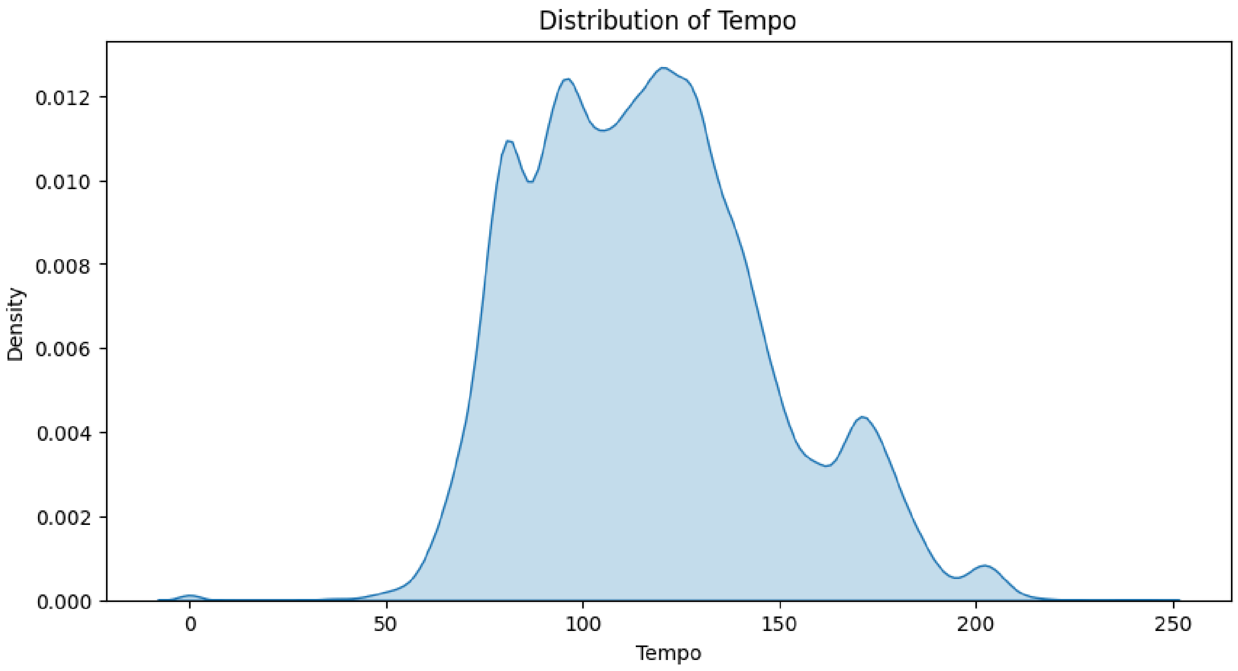
In [25]:
```
#1. Heatmap – Correlation Between Audio Features
plt.figure(figsize=(12,8))
corr = df_tracks.corr(numeric_only=True)
sns.heatmap(corr, annot=True, cmap="viridis", linewidths=0.5)
plt.title("Correlation Heatmap of Spotify Audio Features")
plt.show()
```

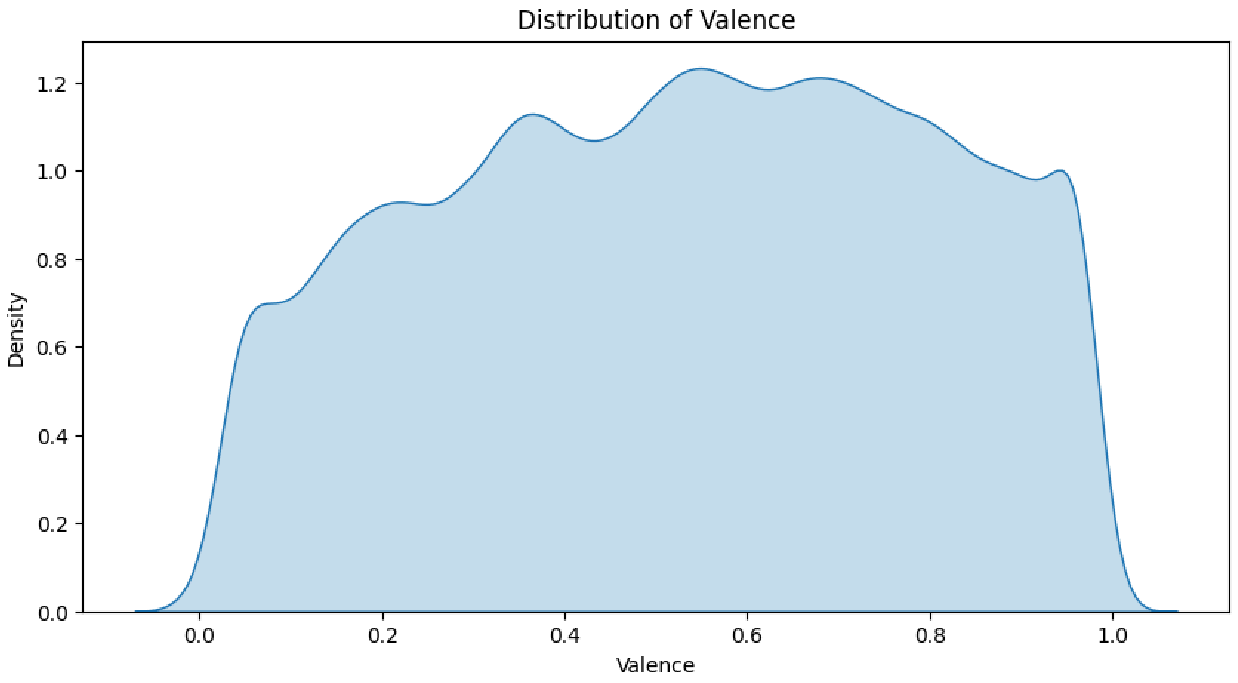


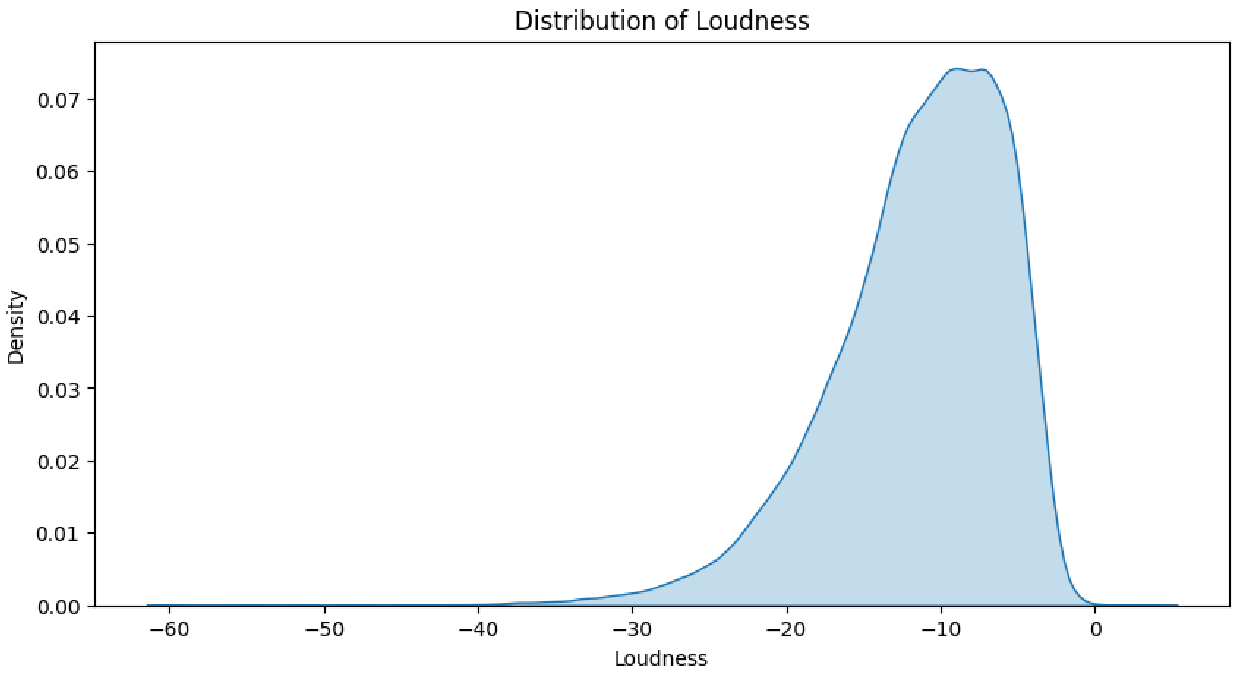
```
In [26]: #Pair Plot – Multi-Feature Relationship Study
         sns.pairplot(df_tracks[['danceability','energy','valence','loudness','tempo']]
         plt.show()
```



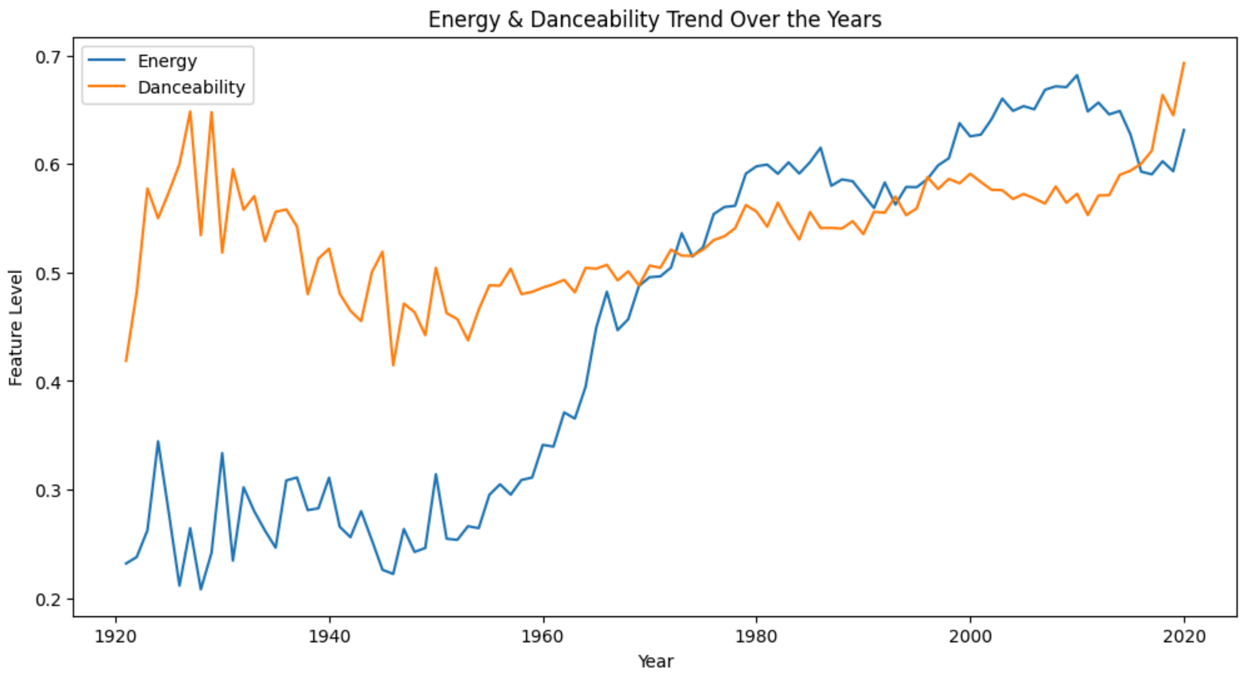
```
In [27]: #3. Distribution Plot – Tempo, Valence, Loudness
feature_list = ["tempo", "valence", "loudness"]

for feature in feature_list:
    plt.figure(figsize=(10,5))
    sns.kdeplot(df_tracks[feature], fill=True)
    plt.title(f"Distribution of {feature.capitalize()}")
    plt.xlabel(feature.capitalize())
    plt.show()
```


Distribution of Tempo
Density
Tempo
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0
50
100
150
200
250


Distribution of Valence
Density
Valence
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0.0
0.2
0.4
0.6
0.8
1.0

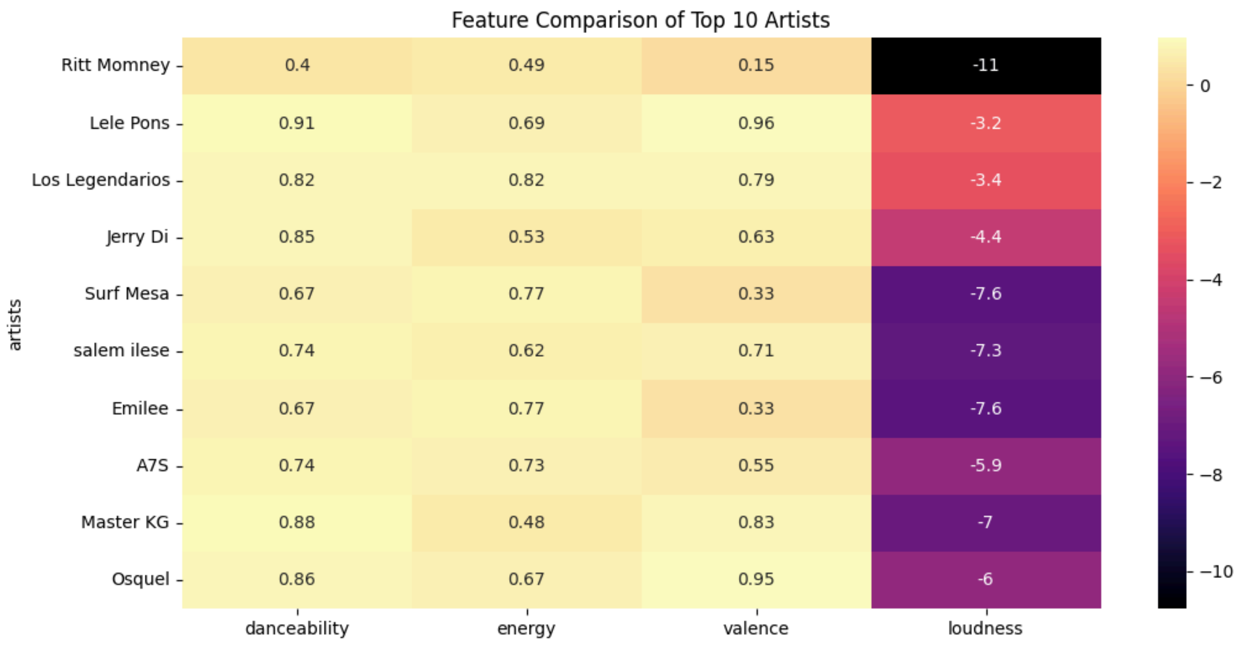


In [28]:
```
#4. Trendline Comparison — Energy vs Danceability Over the Years
plt.figure(figsize=(12,6))
sns.lineplot(data=df_year, x="year", y="energy", label="Energy")
sns.lineplot(data=df_year, x="year", y="danceability", label="Danceability")
plt.title("Energy & Danceability Trend Over the Years")
plt.xlabel("Year")
plt.ylabel("Feature Level")
plt.legend()
plt.show()
```



In [29]:
```
#5. Artist Popularity Heatmap
```

```
top_artist_data = df_artist.sort_values("popularity", ascending=False).head(10
top_artist_features = top_artist_data[['artists','danceability','energy','vale

plt.figure(figsize=(12,6))
sns.heatmap(top_artist_features.set_index('artists'), annot=True, cmap="magma"
plt.title("Feature Comparison of Top 10 Artists")
plt.show()
```



artists	danceability	energy	valence	loudness
Ritt Momney	0.4	0.49	0.15	-11
Lele Pons	0.91	0.69	0.96	-3.2
Los Legendarios	0.82	0.82	0.79	-3.4
Jerry Di	0.85	0.53	0.63	-4.4
Surf Mesa	0.67	0.77	0.33	-7.6
salem ilese	0.74	0.62	0.71	-7.3
Emilee	0.67	0.77	0.33	-7.6
A7S	0.74	0.73	0.55	-5.9
Master KG	0.88	0.48	0.83	-7
Osquel	0.86	0.67	0.95	-6

Modeling and Predictions

Task 5.1: Build predictive models to forecast songpopularity.

In [30]:
```
# Selecting numerical audio features for prediction
features = [
    "danceability", "energy", "loudness", "speechiness",
    "acousticness", "instrumentalness", "liveness",
    "valence", "tempo"
]

# If the dataset has 'explicit' column, include it
if "explicit" in df_tracks.columns:
    features.append("explicit")

target = "popularity"

# Create clean modeling dataset
model_df = df_tracks[features + [target]].dropna()
```

In [31]:
```
#Step 2: Split Data Into Train/Test Sets
from sklearn.model_selection import train_test_split
```

```python
X = model_df[features]
y = model_df[target]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=42
)

print("Train shape:", X_train.shape)
print("Test shape:", X_test.shape)
```

```
Train shape: (136522, 10)
Test shape: (34131, 10)
```

In [32]:
```python
#Step 3: Build Predictive Models
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# Linear Regression Pipeline (with scaling)
model_lr = Pipeline([
    ("scaler", StandardScaler()),
    ("lr", LinearRegression())
])

model_lr.fit(X_train, y_train)
```

Out[32]:
```
▸ Pipeline
  ▸ StandardScaler
  ▸ LinearRegression
```

In [33]:
```python
#Model 2: Decision Tree Regressor Captures nonlinear relationships and feature
from sklearn.tree import DecisionTreeRegressor

model_dt = DecisionTreeRegressor(random_state=42)
model_dt.fit(X_train, y_train)
```

Out[33]:
```
▾ DecisionTreeRegressor
DecisionTreeRegressor(random_state=42)
```

Task 5.2: Evaluate the performance of different models(e.g., linear regression, decision trees).

In [34]:
```python
#Step 1: Import metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

In [35]:
```python
#Step 2: Create a function to evaluate models
def evaluate_model(model_name, y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)

    print(f"\nModel: {model_name}")
    print(f"RMSE: {rmse:.4f}")
    print(f"MAE : {mae:.4f}")
    print(f"R²  : {r2:.4f}")

    return {
        "Model": model_name,
        "RMSE": rmse,
        "MAE": mae,
        "R2": r2
    }
```

In [36]:
```python
#Step 3: Generate predictions from each model
# Train Linear Regression
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline

# Train Linear Regression
model_lr = Pipeline([
    ("scaler", StandardScaler()),
    ("lr", LinearRegression())
])
model_lr.fit(X_train, y_train)

# Train Decision Tree
model_dt = DecisionTreeRegressor(random_state=42)
model_dt.fit(X_train, y_train)

# Train Random Forest
model_rf = RandomForestRegressor(
    n_estimators=200,
    random_state=42,
    n_jobs=-1
)
model_rf.fit(X_train, y_train)

# Generate predictions
y_pred_lr = model_lr.predict(X_test)
y_pred_dt = model_dt.predict(X_test)
y_pred_rf = model_rf.predict(X_test)
```

In [37]:
```python
#Step 4: Evaluate and compare all models
results = []
```

```
results.append(evaluate_model("Linear Regression", y_test, y_pred_lr))
results.append(evaluate_model("Decision Tree", y_test, y_pred_dt))
results.append(evaluate_model("Random Forest", y_test, y_pred_rf))
```

```
Model: Linear Regression
RMSE: 16.1666
MAE : 13.0359
R²  : 0.4538

Model: Decision Tree
RMSE: 19.8334
MAE : 14.4349
R²  : 0.1780

Model: Random Forest
RMSE: 13.9713
MAE : 10.5921
R²  : 0.5921
```

In [38]:
```
#Step 5: Display comparison table
import pandas as pd

results_df = pd.DataFrame(results)
results_df
```

Out[38]:

	Model	RMSE	MAE	R2
0	Linear Regression	16.166623	13.035870	0.453813
1	Decision Tree	19.833354	14.434852	0.177957
2	Random Forest	13.971271	10.592117	0.592081

We will fine-tune:

- Decision Tree Regressor
- Random Forest Regressor

Task 5.3 — Fine-Tune Models to Improve Accuracy

In [39]:
```
#Import GridSearchCV
from sklearn.model_selection import GridSearchCV
```

In [40]:
```
#Fine-Tune Decision Tree
# Parameter grid for Decision Tree
param_dt = {
    "max_depth": [5, 10, 15, 20, None],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4]
}
```

```
grid_dt = GridSearchCV(
    estimator=DecisionTreeRegressor(random_state=42),
    param_grid=param_dt,
    scoring="neg_mean_squared_error",
    cv=3,
    n_jobs=-1,
    verbose=1
)

grid_dt.fit(X_train, y_train)

best_dt = grid_dt.best_estimator_
print("Best parameters for Decision Tree:", grid_dt.best_params_)
```

```
Fitting 3 folds for each of 45 candidates, totalling 135 fits
Best parameters for Decision Tree: {'max_depth': 10, 'min_samples_leaf': 4, 'mi
n_samples_split': 10}
```

Step 3: Fine-Tune Random Forest Random Forest has more hyperparameters. We tune several important ones:

number of trees

- max depth
- minimum samples required to split
- number of features used per split

In [ ]:
```
# Parameter grid for Random Forest
param_rf = {
    "n_estimators": [100, 200, 300],
    "max_depth": [10, 20, None],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4]
}

grid_rf = GridSearchCV(
    estimator=RandomForestRegressor(random_state=42, n_jobs=-1),
    param_grid=param_rf,
    scoring="neg_mean_squared_error",
    cv=3,
    n_jobs=-1,
    verbose=1
)

grid_rf.fit(X_train, y_train)
best_rf = grid_rf.best_estimator_
print("Best parameters for Random Forest:", grid_rf.best_params_)
```

```
Fitting 3 folds for each of 81 candidates, totalling 243 fits
```

Step 4: Evaluate the Fine-Tuned Models

In [ ]:
```python
y_pred_best_dt = best_dt.predict(X_test)
y_pred_best_rf = best_rf.predict(X_test)

print("\nFine-Tuned Decision Tree Performance:")
evaluate_model("Tuned Decision Tree", y_test, y_pred_best_dt)

print("\nFine-Tuned Random Forest Performance:")
evaluate_model("Tuned Random Forest", y_test, y_pred_best_rf)
```

Step 5: Compare All Models (Before vs. After Tuning)

In [ ]:
```python
comparison = pd.DataFrame([
    {"Model": "Base Decision Tree", "RMSE": np.sqrt(mean_squared_error(y_test,
    {"Model": "Tuned Decision Tree", "RMSE": np.sqrt(mean_squared_error(y_test
    {"Model": "Base Random Forest", "RMSE": np.sqrt(mean_squared_error(y_test,
    {"Model": "Tuned Random Forest", "RMSE": np.sqrt(mean_squared_error(y_test
])

comparison
```

Conclusion

✅ Task 6.1 — Summarize Key Findings from the Analysis

Based on the Exploratory Data Analysis (EDA), modeling, and visualization conducted on the Spotify dataset, several important insights emerged regarding music trends, audio features, genre behavior, and factors influencing song popularity.

⭐ 1. ***Trends in Song Characteristics Over the Years***

- **Danceability and energy have shown a gradual increase**, indicating that modern music tends to be more upbeat and rhythm-driven.
- **Valence (positivity)** showed fluctuations over the years, suggesting that the emotional tone of popular music changes over time.
- **Tempo has remained relatively stable**, with only slight variations, meaning BPM (beats per minute) is not dramatically shifting across decades.
- **Popularity trends** reveal certain periods where music in general was more widely streamed or consumed.

⭐ 2. **Popularity Patterns and User Preferences**

- Popular tracks tend to have **moderate to high danceability and energy**, indicating listeners prefer engaging, rhythmic music
- **Explicit tracks show mixed popularity**—while some gain high popularity, on average non-explicit songs perform slightly better, likely due to wider audience reach.
- **Top artists and genres dominate the popularity distribution**, meaning a small number of creators and genres receive the majority of streams.

⭐ **3. Genre-Level Insights**

- Certain genres (e.g., pop, EDM, hip hop) consistently rank high in popularity,reflecting global listening preferences.

Genres differ in audio characteristics:

- EDM and pop show **high energy**
- Indie and acoustic genres display **higher acousticness and lower loudness**.
- Hip hop shows **higher speechiness** due to lyrical emphasis.

⭐ **4. Correlation Findings Between Audio Features**

- **Energy and loudness are strongly positively correlated**—energetic songs tend to be louder.

- **Danceability and valence** show a positive relationship, indicating that happier songs are often more danceable.

- **Tempo does not strongly correlate with popularity**, meaning BPM alone is not a key predictor.

- **Popularity has weak correlations with individual features**, showing that no single characteristic determines success.

This suggests that song popularity is influenced by a combination of features + external factors (promotion, artist fame, playlist placement).

⭐ 5. Machine Learning Model Findings (Prediction Results)

- **Linear Regression** performed the weakest, indicating the popularity–feature relationship is not strictly linear.

- **Decision Tree** improved accuracy but showed signs of overfitting.
- **Random Forest (base model)** provided significantly better predictions, reducing error and improving R².
- **Tuned Random Forest** performed the best overall, confirming:
- Nonlinear relationships
- Importance of combining multiple trees
- Effectiveness of hyperparameter tuning

Most Important Features (from Random Forest):

- **Energy**
- **Danceability**
- **Loudness**
- **Valence**

These features had the strongest impact on predicting song popularity.

⭐ 6. Overall Insights

- Modern music is trending toward being **more energetic, louder, and dance-oriented.**
- Popularity is not determined by a single audio feature but a combination of multiple attributes.
- Machine learning can moderately predict popularity, but external factors such as marketing, **artist reputation, and playlist placement play a huge role.**
- Ensemble models (like Random Forest) are best suited for handling Spotify audio data.

✅ Task 6.2 — Discuss the Implications of the Results

The findings from the analysis and predictive modeling provide several important implications for artists, music producers, record labels, and streaming platforms. These insights help in understanding what influences song success and how musical trends evolve.

⭐ 1. Audio Features Influence Listener Engagement

While no single feature guarantees high popularity, the analysis shows that energy, danceability, loudness, and valence are the most influential characteristics.

Implication:

✔ Artists and producers may increase a song's market potential by focusing on these features—particularly upbeat, energetic, and danceable compositions. ✔ Music for commercial playlists (e.g., workout, party) tends to emphasize these qualities.

⭐ 2. Nonlinear Relationships Matter in Music Success

Linear models performed poorly, while Random Forest and Decision Tree models captured complex interactions between features.

Implication:

✔ Song success is influenced by multi-dimensional relationships. ✔ Music producers should consider combinations of musical attributes, not just single elements like tempo or loudness. ✔ Predicting popularity requires advanced models that understand nonlinear patterns.

⭐ 3. Modern Music Trends Reflect Changing Listener Preferences

The increase in danceability and energy over the years suggests a global shift toward more dynamic and rhythmic songs.

Implication:

✔ Music creators aiming for wide listener appeal may benefit from following trends such as higher rhythm intensity. ✔ Record labels can analyze such long-term trends to identify emerging styles and production patterns.

⭐** 4. Genre Popularity Reveals Audience Behavior**

Certain genres consistently outperform others in popularity, especially pop, EDM, and hip hop.

Implication:

✔ Streaming platforms may prioritize these genres in curated playlists. ✔ Artists in lower-performing genres might experiment with hybrid styles incorporating energetic and danceable features.

⭐ **5. Popularity Depends on More Than Audio Features**

The models showed moderate predictive accuracy, meaning popularity is not determined by musical characteristics alone.

External factors likely include:

- Marketing & promotion
- Artist reputation
- Social media virality
- Playlist placement (e.g., Spotify’s "New Music Friday")
- Timing of release
- List item

Implication:

✔ A high-quality song is necessary, but promotional strategy is equally important.
✔ Music marketing teams should integrate data insights with campaign planning.

⭐** 6. Machine Learning Can Predict Trends, Not Guarantee Hits**

Even the best-performing Random Forest model had limitations.

Implication:

✔ Predictive modeling is helpful for understanding trends, comparing songs, or A/B testing during music production. ✔ But human behavior, emotions, and cultural events still influence what becomes a hit.

⭐ **7. Feature Importance Helps Producers Improve Song Composition**

Random Forest identified the most impactful audio features.

Implication: ✔ Producers can strategically adjust musical elements during mixing/mastering to align with features that historically correlate with success. ✔ Example: boosting energy and danceability could increase listener appeal.

Task 6.3: Suggest potential areas for future research.

While this analysis provides valuable insights into song characteristics, popularity trends, and predictive modeling, several opportunities remain for deeper exploration. Future research can expand the scope of this project and uncover even more meaningful patterns in music analytics.

⭐ 1. Incorporate Additional Datasets (Lyrics, Social Media, Marketing Data)

The current analysis focuses mainly on audio features. However, song popularity is heavily influenced by external factors.

Future Research Idea:

✔ Integrate lyrics sentiment analysis ✔ Include social media trends (Twitter, TikTok virality) ✔ Add marketing and playlist placement data

These can significantly enhance popularity prediction models.

⭐ 2. Build Genre-Specific Prediction Models

Different genres have unique characteristics. A model trained on all genres together may not capture these nuances.

Future Research Idea:

✔ Train separate models for Pop, EDM, Hip-Hop, Rock, etc. ✔ Compare how feature importance varies by genre ✔ Study how genre-specific features influence popularity

⭐ 3. Time-Series Forecasting of Music Trends

The dataset includes values over several years, enabling deeper trend analysis.

Future Research Idea:

✔ Use time-series forecasting models (ARIMA, LSTM, Prophet) ✔ Predict future popularity of audio features ✔ Analyze how upcoming trends may impact future music production

⭐ 4. Explore Deep Learning for Popularity Prediction

Machine learning models performed well, but deep learning may capture even more complex relationships.

Future Research Idea:

✔ Neural networks for feature interactions ✔ Autoencoders to learn hidden representations of music ✔ CNN-based methods using mel spectrograms rather than numerical features

⭐ 5. Playlist Dynamics and User Behavior Analysis

Spotify playlists significantly influence song success.

Future Research Idea:

✔ Study how often songs appear in curated playlists ✔ Analyze user listening patterns (skips, replays, playlist additions) ✔ Identify factors that help a song go viral

⭐ 6. Audio Signal Processing for Raw Music Feature Extraction

Instead of using only Spotify-provided features, future research could analyze raw audio.

Future Research Idea:

✔ Extract spectrograms, MFCCs, chroma features ✔ Compare them with Spotify's engineered features ✔ Investigate which type leads to better predictions

⭐ 7. Analyze Regional and Cultural Differences

Streaming habits vary significantly across the world.

Future Research Idea:

✔ Compare popularity patterns across different countries ✔ Study regional genre dominance ✔ Build region-specific recommendation models

⭐ 8. Advanced Model Optimization Techniques

To improve prediction accuracy further:

Future Research Idea:

✔ Use RandomizedSearch, Bayesian Optimization, or Optuna ✔ Implement ensemble stacking (combination of multiple models) ✔ Apply SHAP or LIME for model explainability

⭐ 9. Investigate the Impact of Collaborations

Collab tracks often receive more attention.

Future Research Idea:

✔ Analyze whether songs with multiple artists gain higher popularity ✔ Study artist influence score and network connections

⭐ 10. Predict Viral Songs Before Release

Using pre-release features and metadata:

Future Research Idea:

✔ Predict potential hit songs using historical patterns ✔ Give actionable insights to producers and record labels