Appendix

Code for Exploratory Data Analysis

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

# Load the dataset
file_path = 'song_data.csv'
data = pd.read_csv(file_path)

# Step 1: Data Cleaning

# Drop unnecessary columns
if 'song_name' in data.columns:
    data = data.drop(columns=['song_name'])

# Display basic information about the dataset
print("Basic Information about the Dataset:")
print(data.info())

# Display descriptive statistics
print("\nDescriptive Statistics:")
print(data.describe())
```

```
Basic Information about the Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12963 entries, 0 to 12962
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   song_popularity   12963 non-null  int64
 1   song_duration_ms  12963 non-null  int64
 2   acousticness      12963 non-null  float64
 3   danceability      12963 non-null  float64
 4   energy            12963 non-null  float64
 5   instrumentalness  12963 non-null  float64
 6   key               12963 non-null  int64
 7   liveness          12963 non-null  float64
 8   loudness          12963 non-null  float64
 9   audio_mode        12963 non-null  int64
 10  speechiness       12963 non-null  float64
 11  tempo             12963 non-null  float64
 12  time_signature    12963 non-null  int64
 13  audio_valence     12963 non-null  float64
 14  Hit               12963 non-null  int64
dtypes: float64(9), int64(6)
memory usage: 1.5 MB
None

Descriptive Statistics:
       song_popularity  song_duration_ms  acousticness
danceability  \
```

count	12963.000000	1.296300e+04	12963.000000	12963.000000
mean	48.501196	2.186847e+05	0.277869	0.625162
std	20.098640	6.352215e+04	0.301970	0.159137
min	0.000000	1.200000e+04	0.000001	0.000000
25%	37.000000	1.830000e+05	0.025000	0.525000
50%	51.000000	2.114850e+05	0.147000	0.637000
75%	63.000000	2.445995e+05	0.480000	0.741000
max	100.000000	1.799346e+06	0.996000	0.987000

	energy	instrumentalness	key	liveness \
count	12963.000000	12963.000000	12963.000000	12963.000000
mean	0.635872	0.096927	5.320219	0.180644
std	0.224070	0.246870	3.584777	0.145507
min	0.001070	0.000000	0.000000	0.011900
25%	0.490000	0.000000	2.000000	0.093300
50%	0.668000	0.000023	5.000000	0.122000
75%	0.815000	0.005965	8.000000	0.223000
max	0.999000	0.997000	11.000000	0.986000

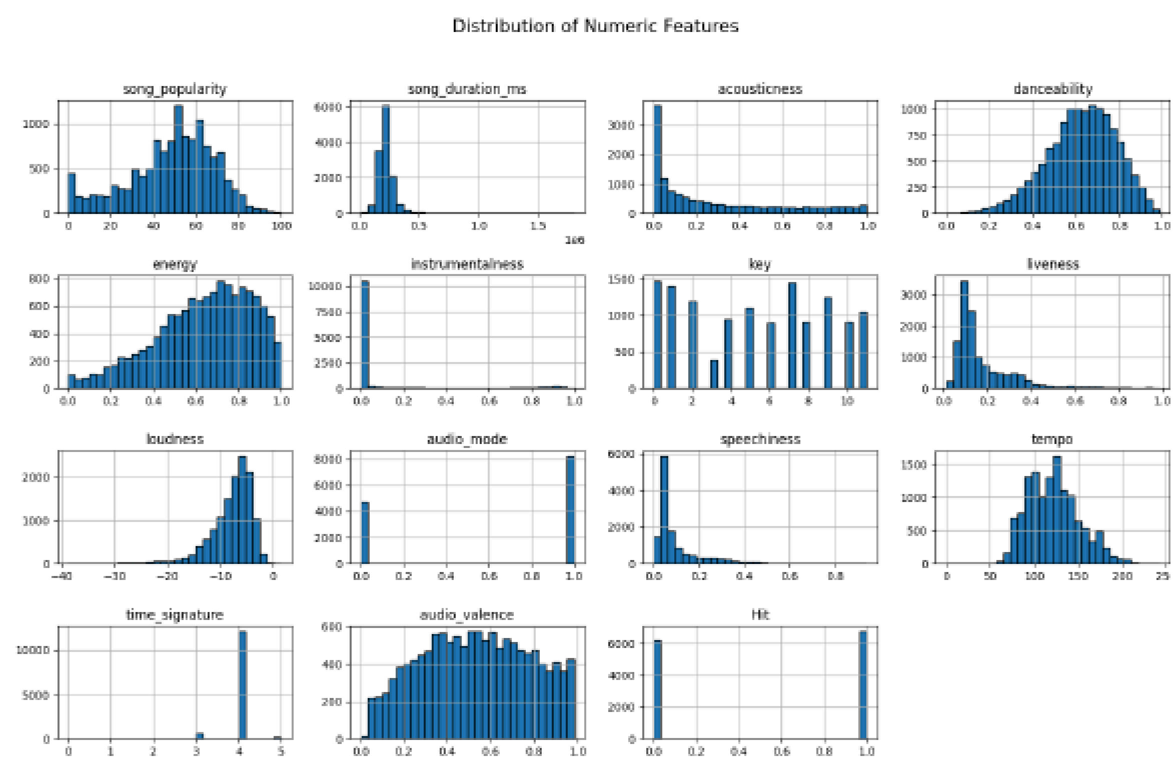
	loudness	audio_mode	speechiness	tempo	time_signature \
count	12963.000000	12963.000000	12963.000000	12963.000000	12963.000000
mean	-7.792560	0.633650	0.100789	121.157971	3.951246
std	4.116712	0.481825	0.105185	29.104165	0.319744
min	-38.768000	0.000000	0.000000	0.000000	0.000000
25%	-9.538500	0.000000	0.037300	98.072500	4.000000
50%	-6.859000	1.000000	0.054400	120.024000	4.000000
75%	-5.040500	1.000000	0.115000	139.955500	4.000000
max	1.585000	1.000000	0.941000	242.318000	5.000000

	audio_valence	Hit
count	12963.000000	12963.000000
mean	0.528709	0.520019
std	0.248780	0.499618

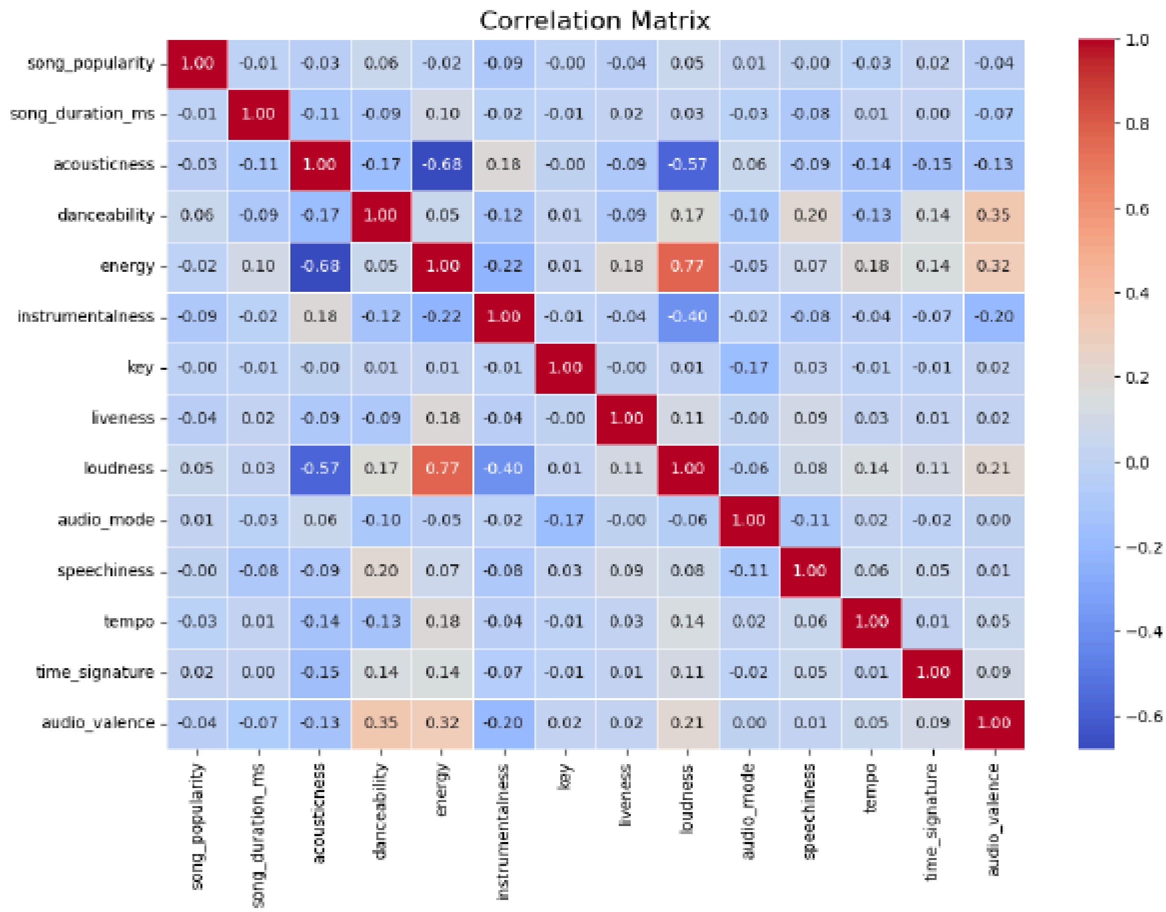
```
min          0.000000      0.000000
25%          0.334000      0.000000
50%          0.529000      1.000000
75%          0.731000      1.000000
max          0.984000      1.000000
```

```python
# Step 2: Distribution of Numeric Columns
plt.figure(figsize=(12, 8))

# Determine the number of numeric columns
num_numeric_columns = data.select_dtypes(include=['number']).shape[1]

# Calculate the layout based on the number of numeric columns
rows = (num_numeric_columns // 4) + 1
cols = min(num_numeric_columns, 4)

data.hist(bins=30, figsize=(15, 10), layout=(rows, cols),
edgecolor='black')
plt.suptitle('Distribution of Numeric Features', fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

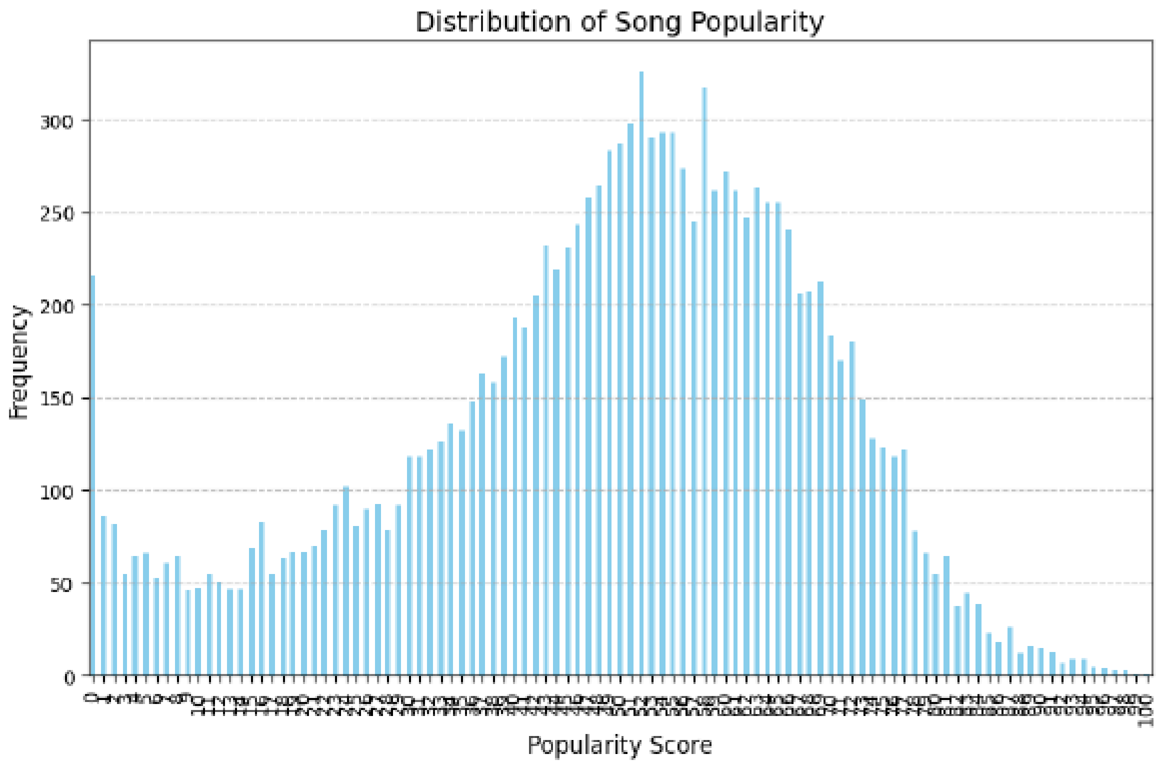
```
<Figure size 1200x800 with 0 Axes>
```



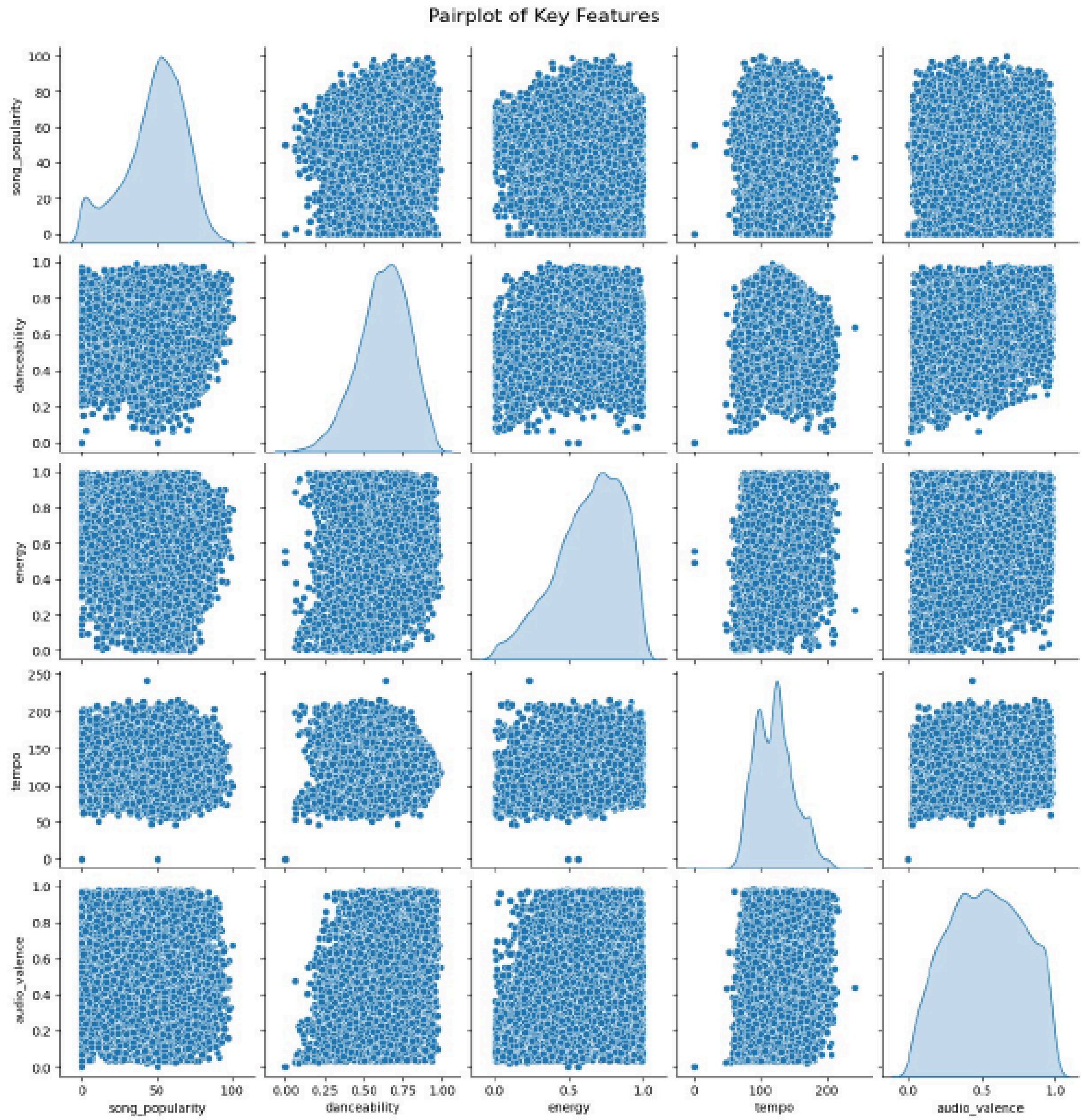
```python
# Step 3: Correlation Matrix
plt.figure(figsize=(12, 8))
corr_matrix = data.corr()
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm',
linewidths=0.5)
plt.title('Correlation Matrix', fontsize=16)
plt.show()
```



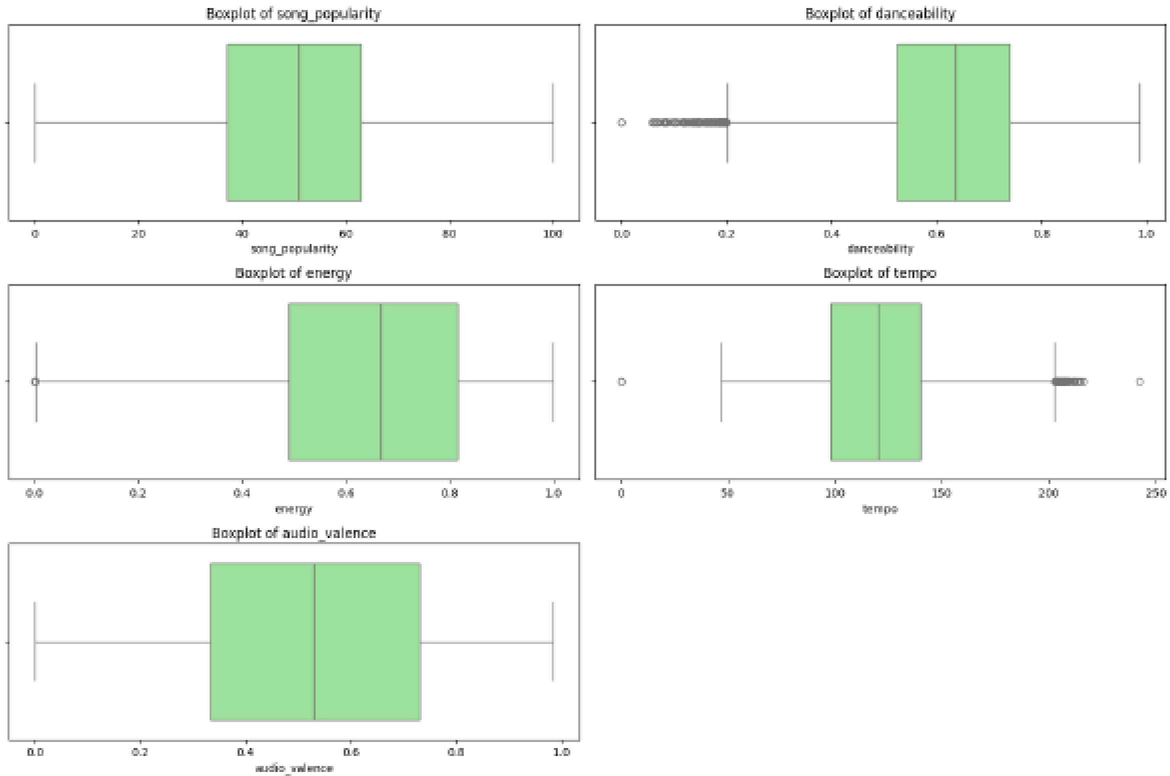

```python
# Step 4: Distribution of Song Popularity
plt.figure(figsize=(10, 6))
data['song_popularity'].value_counts().sort_index().plot(kind='bar',
color='skyblue')
plt.title('Distribution of Song Popularity', fontsize=14)
plt.xlabel('Popularity Score', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```




```python
# Step 5: Pairplot for Key Features
key_features = ['song_popularity', 'danceability', 'energy', 'tempo',
'audio_valence']
sns.pairplot(data[key_features], diag_kind='kde')
plt.suptitle('Pairplot of Key Features', y=1.02, fontsize=16)
plt.show()
```




```python
# Step 6: Outlier Detection using Boxplots
plt.figure(figsize=(15, 10))
for i, column in enumerate(key_features, 1):
    plt.subplot(3, 2, i)
    sns.boxplot(data=data, x=column, color='lightgreen')
    plt.title(f'Boxplot of {column}', fontsize=12)
    plt.tight_layout()
plt.show()
```

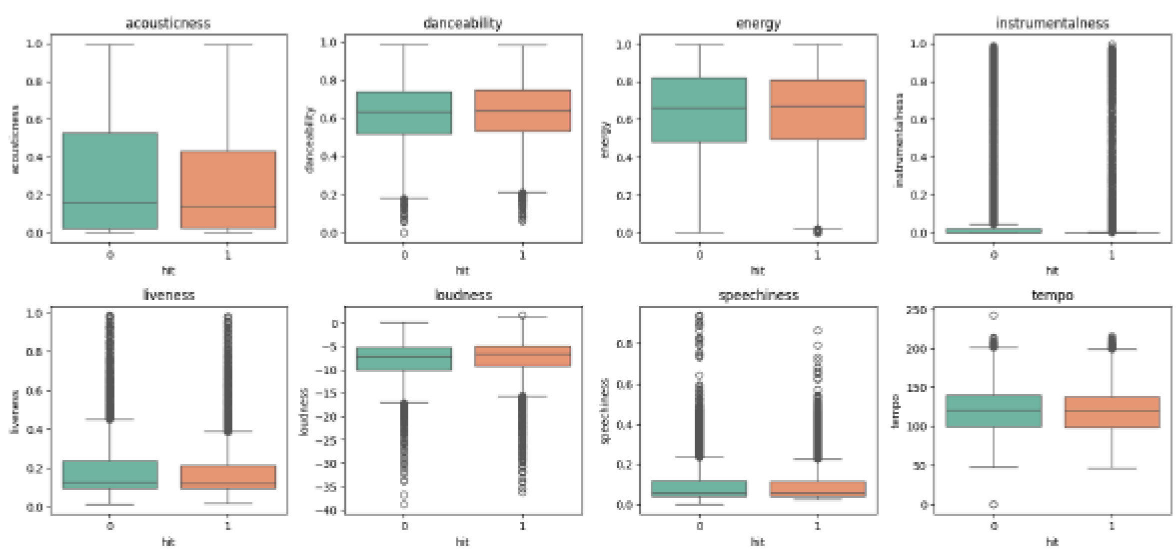


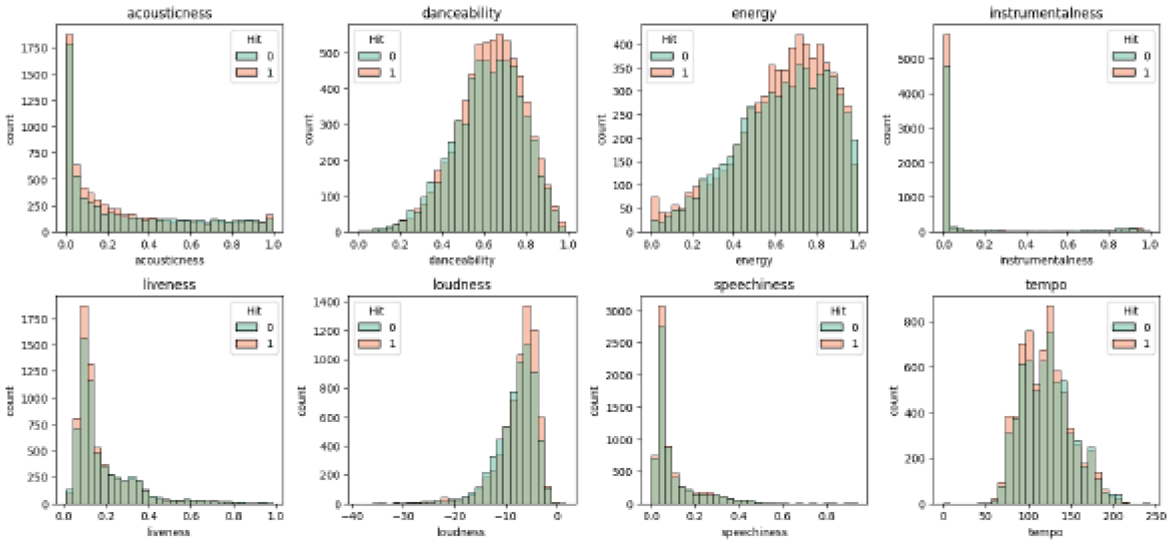

```
# Define numeric columns and the "hit" column
numeric_columns = ['acousticness', 'danceability', 'energy',
'instrumentalness',
                   'liveness', 'loudness', 'speechiness', 'tempo',
'valence', 'artist_popularity']
hit_column = 'Hit'

# Check if all columns in numeric_columns exist in the dataframe
missing_columns = [col for col in numeric_columns if col not in
data.columns]
if missing_columns:
    print(f"Warning: The following columns are missing in the
dataframe: {missing_columns}")

# Box plots for numeric features grouped by "hit"
plt.figure(figsize=(15, 10))
for i, col in enumerate(numeric_columns, 1):
    if col in data.columns:
        plt.subplot(3, 4, i)
        sns.boxplot(data=data, x=hit_column, y=col, palette="Set2")
        plt.title(col)
        plt.xlabel("hit")
        plt.ylabel(col)
plt.tight_layout()
plt.show()
```

```python
# Histograms for numeric features grouped by "hit"
plt.figure(figsize=(15, 10))
for i, col in enumerate(numeric_columns, 1):
    if col in data.columns:
        plt.subplot(3, 4, i)
        sns.histplot(data=data, x=col, hue=hit_column, kde=False,
palette="Set2", bins=30)
        plt.title(col)
        plt.xlabel(col)
        plt.ylabel("count")
plt.tight_layout()
plt.show()
```

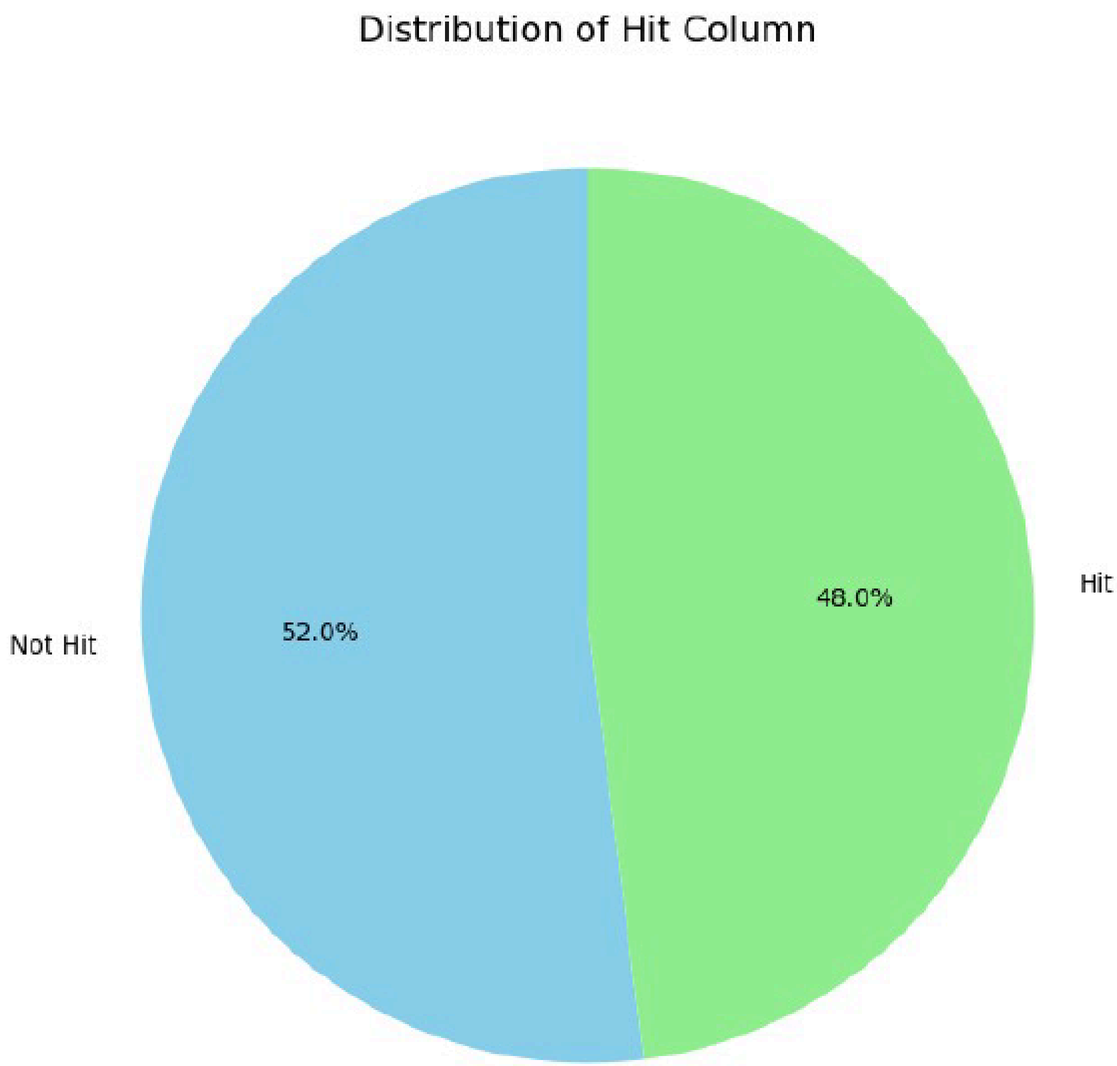
```
Warning: The following columns are missing in the dataframe:
['valence', 'artist_popularity']

C:\Users\Shyam\AppData\Local\Temp\ipykernel_32864\3363145177.py:16:
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 effect.

  sns.boxplot(data=data, x=hit_column, y=col, palette="Set2")
C:\Users\Shyam\AppData\Local\Temp\ipykernel_32864\3363145177.py:16:
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 effect.

  sns.boxplot(data=data, x=hit_column, y=col, palette="Set2")
C:\Users\Shyam\AppData\Local\Temp\ipykernel_32864\3363145177.py:16:
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 effect.

  sns.boxplot(data=data, x=hit_column, y=col, palette="Set2")
C:\Users\Shyam\AppData\Local\Temp\ipykernel_32864\3363145177.py:16:
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 effect.

  sns.boxplot(data=data, x=hit column, y=col, palette="Set2")
C:\Users\Shyam\AppData\Local\Temp\ipykernel_32864\3363145177.py:16:
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 effect.

  sns.boxplot(data=data, x=hit_column, y=col, palette="Set2")
C:\Users\Shyam\AppData\Local\Temp\ipykernel_32864\3363145177.py:16:
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 effect.

  sns.boxplot(data=data, x=hit_column, y=col, palette="Set2")
C:\Users\Shyam\AppData\Local\Temp\ipykernel_32864\3363145177.py:16:
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 effect.

  sns.boxplot(data=data, x=hit_column, y=col, palette="Set2")
C:\Users\Shyam\AppData\Local\Temp\ipykernel_32864\3363145177.py:16:
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 effect.

  sns.boxplot(data=data, x=hit_column, y=col, palette="Set2")
```






```python
# Step 7: Pie Chart for Hit Column
hit_counts = data['Hit'].value_counts()
plt.figure(figsize=(8, 8))
hit_counts.plot.pie(autopct='%1.1f%%', startangle=90,
colors=['skyblue', 'lightgreen'], labels=['Not Hit', 'Hit'])
plt.title('Distribution of Hit Column', fontsize=14)
plt.ylabel('')
plt.show()
```


Distribution of Hit Column
Not Hit
52.0%
48.0%
Hit

Code for Multilinear Regression

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Load the dataset
file_path = 'song_data.csv'
data = pd.read_csv(file_path)

# Step 1: Data Cleaning

# Drop the 'song_name' column if it exists
if 'song_name' in data.columns:
    data = data.drop(columns=['song_name'])

# Check for null values and handle them (drop rows with NaNs)
data = data.dropna()

# Step 2: Define Features and Target
X = data.drop(columns=['Hit'])  # Features
y = data['Hit']  # Target variable

# Ensure all features are numeric
X = pd.get_dummies(X, drop_first=True)

# Step 3: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Step 4: Fit the Linear Regression Model
model = LinearRegression()
model.fit(X_train, y_train)

LinearRegression()

# Step 5: Predictions and Evaluation
y_pred = model.predict(X_test)

# Evaluate Training Data
y_train_pred = model.predict(X_train)

# Calculate Evaluation Metrics for Training Data
train_mse = mean_squared_error(y_train, y_train_pred)
train_r2 = r2_score(y_train, y_train_pred)

print("Training Mean Squared Error (MSE):", train_mse)
print("Training R-squared (R²):", train_r2)

Training Mean Squared Error (MSE): 0.09387354454122344
Training R-squared (R²): 0.623859081452601
```

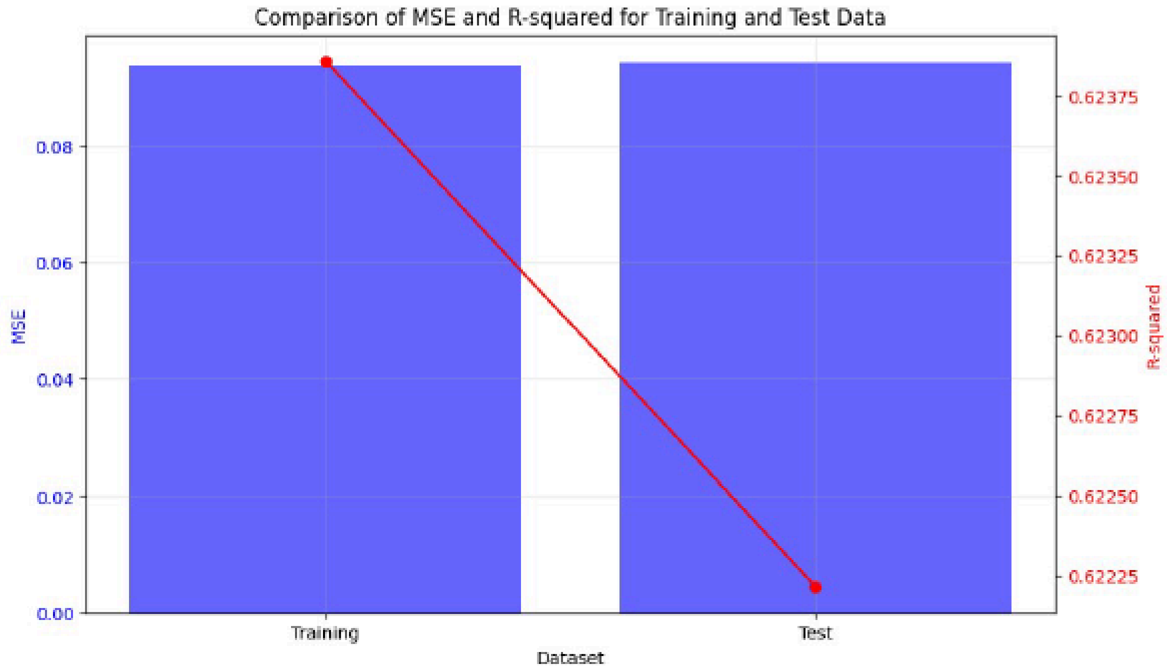
```python
# Evaluation Metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error (MSE):", mse)
print("R-squared (R²):", r2)
```

```
Mean Squared Error (MSE): 0.0943349801587489
R-squared (R²): 0.6222150182321906
```

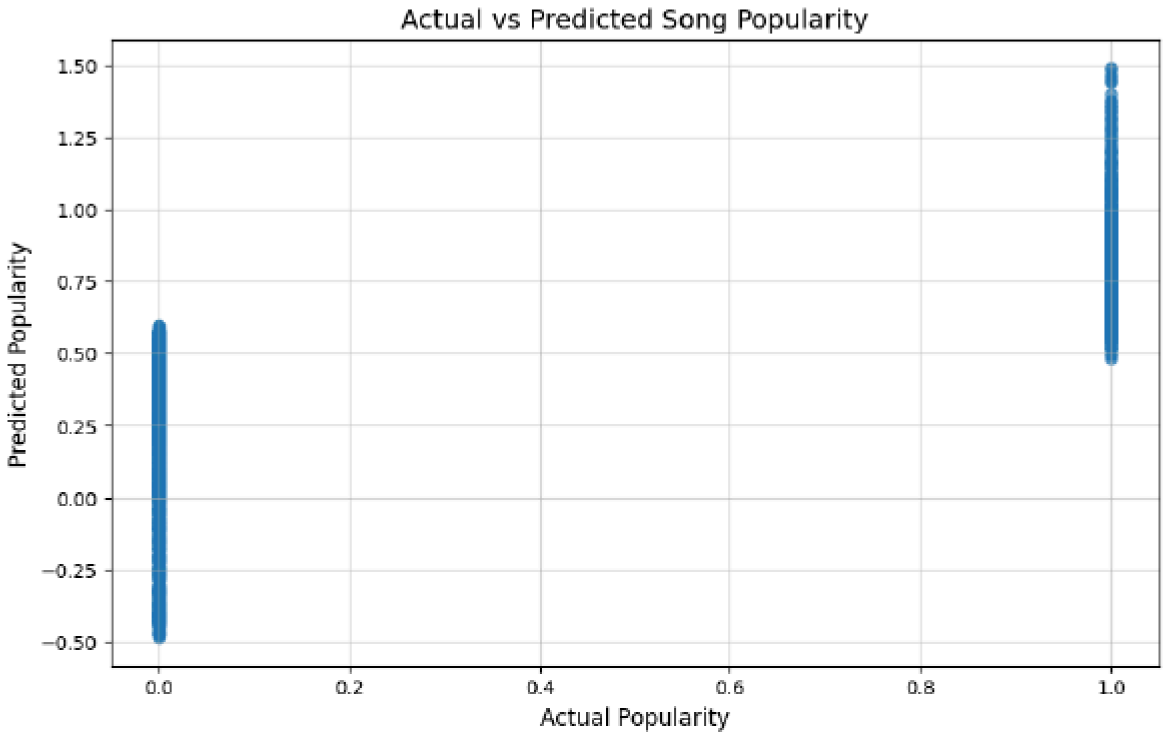
```python
# Calculate Model Accuracy
accuracy = model.score(X_test, y_test) * 100
print("Model Accuracy: {:.2f}%".format(accuracy))
```

```
Model Accuracy: 62.22%
```

```python
# Create a DataFrame for the metrics
metrics_df = pd.DataFrame({
    'Dataset': ['Training', 'Test'],
    'MSE': [train_mse, mse],
    'R-squared': [train_r2, r2]
})

# Plot the metrics
fig, ax1 = plt.subplots(figsize=(10, 6))

# Plot MSE
ax1.bar(metrics_df['Dataset'], metrics_df['MSE'], color='b',
alpha=0.6, label='MSE')
ax1.set_xlabel('Dataset')
ax1.set_ylabel('MSE', color='b')
ax1.tick_params(axis='y', labelcolor='b')

# Create a second y-axis to plot R-squared
ax2 = ax1.twinx()
ax2.plot(metrics_df['Dataset'], metrics_df['R-squared'], color='r',
marker='o', label='R-squared')
ax2.set_ylabel('R-squared', color='r')
ax2.tick_params(axis='y', labelcolor='r')

# Add title and grid
plt.title('Comparison of MSE and R-squared for Training and Test
Data')
ax1.grid(alpha=0.3)

# Show plot
plt.show()
```




```python
# Step 6: Visualize Actual vs Predicted
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.7)
plt.title('Actual vs Predicted Song Popularity', fontsize=14)
plt.xlabel('Actual Popularity', fontsize=12)
plt.ylabel('Predicted Popularity', fontsize=12)
plt.grid(alpha=0.5)
plt.show()
```

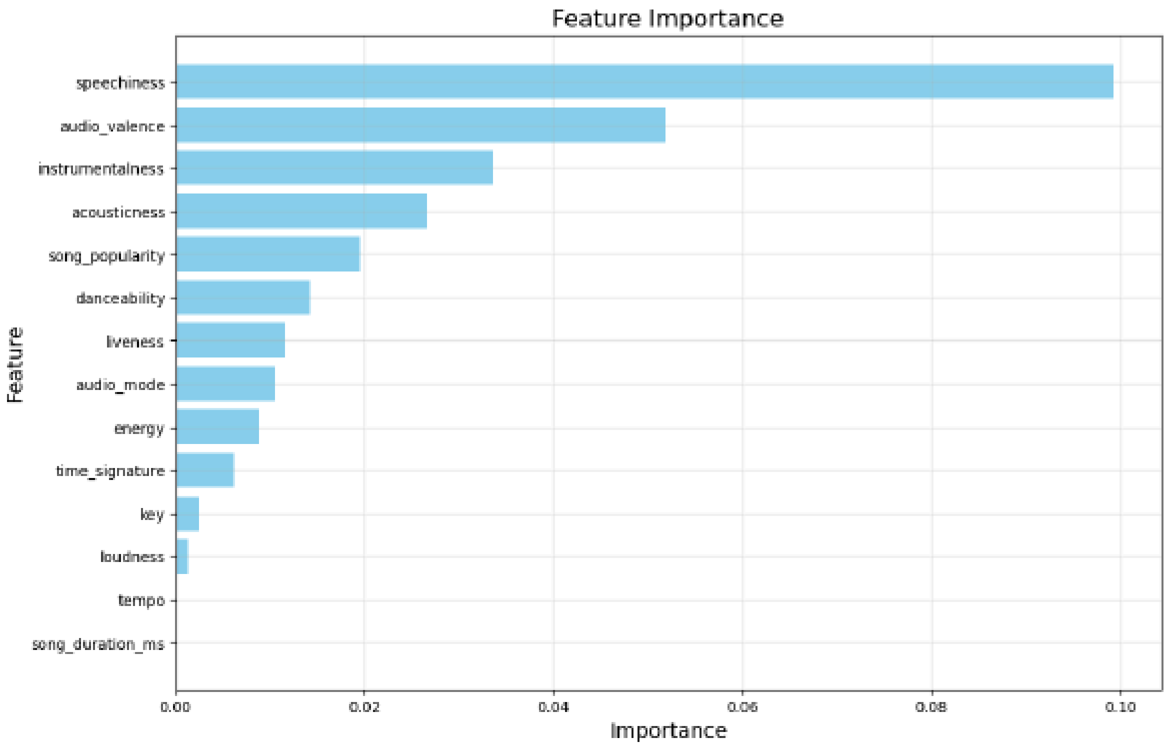



```
# Display regression coefficients
coefficients = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': model.coef_
}).sort_values(by='Coefficient', ascending=False)

print("\nRegression Coefficients:")
print(coefficients)


Regression Coefficients:
             Feature   Coefficient
0    song_popularity  1.951877e-02
9         audio_mode  1.054356e-02
12    time_signature  6.223634e-03
6                key  2.524986e-03
8           loudness  1.337141e-03
1   song_duration_ms  7.422640e-08
11             tempo -1.643269e-04
4             energy -8.837254e-03
7           liveness -1.163437e-02
3       danceability -1.427307e-02
2       acousticness -2.656792e-02
5   instrumentalness -3.364185e-02
13     audio_valence -5.179220e-02
10       speechiness -9.935116e-02
```

```
# Calculate the absolute values of the coefficients
coefficients['Importance'] = coefficients['Coefficient'].abs()

# Sort the features by importance
coefficients = coefficients.sort_values(by='Importance',
ascending=False)

print("\nFeature Importance:")
print(coefficients[['Feature', 'Importance']])
```

```
Feature Importance:
               Feature    Importance
10         speechiness  9.935116e-02
13       audio_valence  5.179220e-02
5     instrumentalness  3.364185e-02
2         acousticness  2.656792e-02
0      song_popularity  1.951877e-02
3         danceability  1.427307e-02
7             liveness  1.163437e-02
9           audio mode  1.054356e-02
4               energy  8.837254e-03
12      time_signature  6.223634e-03
6                  key  2.524986e-03
8             loudness  1.337141e-03
11               tempo  1.643269e-04
1     song_duration_ms  7.422640e-08
```

```
# Plot Feature Importance
plt.figure(figsize=(12, 8))
plt.barh(coefficients['Feature'], coefficients['Importance'],
color='skyblue')
plt.xlabel('Importance', fontsize=14)
plt.ylabel('Feature', fontsize=14)
plt.title('Feature Importance', fontsize=16)
plt.gca().invert_yaxis()  # Invert y-axis to have the most important
feature at the top
plt.grid(alpha=0.3)
plt.show()
```



Feature Importance
speechiness
audio_valence
instrumentalness
acousticness
song_popularity
danceability
liveness
audio_mode
energy
time_signature
key
loudness
tempo
song_duration_ms
Feature
0.00
0.02
0.04
0.06
0.08
0.10
Importance

Code for Polynomial Regression

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Load the dataset
file_path = 'song_data.csv'
data = pd.read_csv(file_path)

# Step 1: Data Cleaning

# Drop the 'song_name' column if it exists
if 'song_name' in data.columns:
    data = data.drop(columns=['song_name'])

# Check for null values and handle them (drop rows with NaNs)
data = data.dropna()

# Step 2: Define Features and Target
X = data.drop(columns=['Hit'])  # Features
y = data['Hit']  # Target variable

# Ensure all features are numeric
X = pd.get_dummies(X, drop_first=True)

#X = data[['danceability', 'loudness', 'energy', 'acousticness',
'instrumentalness']]
#print(X.head(3))

# Step 3: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Step 4: Polynomial Feature Transformation
degree = 2  # Degree of the polynomial
poly = PolynomialFeatures(degree=degree)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

# Step 5: Standardization
scaler = StandardScaler()
X_train_poly = scaler.fit_transform(X_train_poly)
X_test_poly = scaler.transform(X_test_poly)

# Step 6: Fit the Polynomial Regression Model
model = LinearRegression()
model.fit(X_train_poly, y_train)
```

```
LinearRegression()
```

```python
# Step 7: Predictions and Evaluation
y_pred = model.predict(X_test_poly)

# Predictions on the training set
y_train_pred = model.predict(X_train_poly)

# Evaluation Metrics for the training set
train_mse = mean_squared_error(y_train, y_train_pred)
train_r2 = r2_score(y_train, y_train_pred)

print(f"Polynomial Regression (Degree {degree}) - Training Set")
print("Mean Squared Error (MSE):", train_mse)
print("R-squared (R²):", train_r2)
```

```
Polynomial Regression (Degree 2) - Training Set
Mean Squared Error (MSE): 0.08378420341603458
R-squared (R²): 0.6642859563182886
```

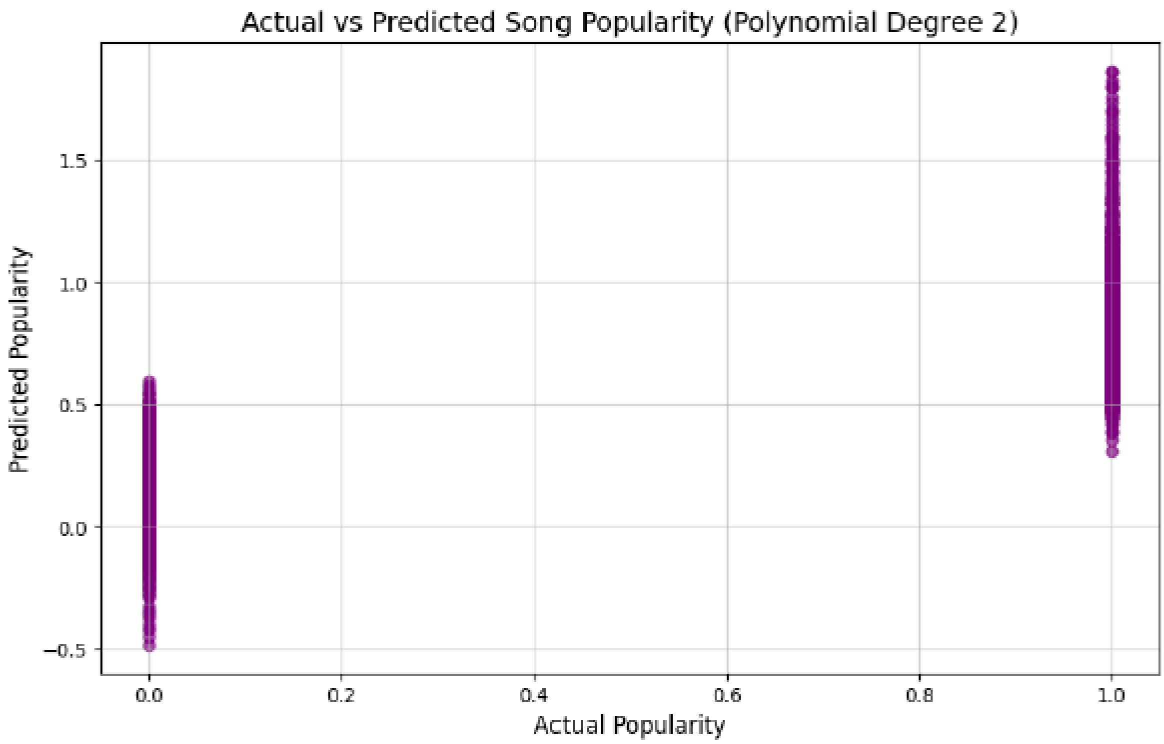
```python
# Evaluation Metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Polynomial Regression (Degree {degree})")
print("Mean Squared Error (MSE):", mse)
print("R-squared (R²):", r2)
```

```
Polynomial Regression (Degree 2)
Mean Squared Error (MSE): 0.08598448871420705
R-squared (R²): 0.6556563806284058
```

```python
# Calculate Model Accuracy
accuracy = model.score(X_test_poly, y_test) * 100
print("Model Accuracy: {:.2f}%".format(accuracy))
```

```
Model Accuracy: 65.57%
```

```python
# Step 8: Visualize Actual vs Predicted
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.7, color='purple')
plt.title(f'Actual vs Predicted Song Popularity (Polynomial Degree {degree})', fontsize=14)
plt.xlabel('Actual Popularity', fontsize=12)
plt.ylabel('Predicted Popularity', fontsize=12)
plt.grid(alpha=0.5)
plt.show()
```




```python
# Display regression coefficients (optional, as polynomial features
may not be directly interpretable)
coefficients = pd.DataFrame({
    'Feature': poly.get_feature_names_out(X.columns),
    'Coefficient': model.coef_
}).sort_values(by='Coefficient', ascending=False)

print("\nTop Features in Polynomial Regression:")
print(coefficients.head(10))
```

```
Top Features in Polynomial Regression:
                          Feature   Coefficient
105                  audio_mode^2  7.822887e+11
15               song_popularity^2  3.385074e-01
2                 song_duration_ms  9.524357e-02
4                     danceability  8.346122e-02
19          song_popularity energy  4.588825e-02
55             danceability energy  4.534912e-02
72                    energy tempo  4.257202e-02
51              acousticness tempo  4.177856e-02
28   song_popularity audio_valence  3.328451e-02
13                  time_signature  3.305153e-02
```