In [1]:
```python
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics.pairwise import cosine_similarity
from scipy.sparse import csr_matrix
from sklearn.preprocessing import StandardScaler
from datetime import datetime

from math import sqrt
from sklearn.preprocessing import normalize
from scipy.spatial.distance import cosine, euclidean, hamming

from sklearn.model_selection import train_test_split
from sklearn.metrics import average_precision_score
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error

import pandas as pd
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.probability import FreqDist

# Download stopwords (if not already downloaded)
nltk.download('stopwords', quiet=True)

import warnings

# Ignore specific warning by category
warnings.filterwarnings("ignore")
```

In [2]:
```python
data = pd.read_csv(r"C:\Users\Tanmayee\OneDrive\Documents\Personal\September 2023\Bhanu\dataset.cs
```

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 170653 entries, 0 to 170652
Data columns (total 22 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   Unnamed: 0        114000 non-null  float64
 1   track_id          114000 non-null  object 
 2   artists           113999 non-null  object 
 3   album_name        113999 non-null  object 
 4   track_name        113999 non-null  object 
 5   popularity        114000 non-null  float64
 6   duration_ms       114000 non-null  float64
 7   explicit          114000 non-null  object 
 8   danceability      114000 non-null  float64
 9   energy            114000 non-null  float64
 10  key               114000 non-null  float64
 11  loudness          114000 non-null  float64
 12  mode              114000 non-null  float64
 13  speechiness       114000 non-null  float64
 14  acousticness      114000 non-null  float64
 15  instrumentalness  114000 non-null  float64
 16  liveness          114000 non-null  float64
 17  valence           114000 non-null  float64
 18  tempo             114000 non-null  float64
 19  time_signature    114000 non-null  float64
 20  track_genre       114000 non-null  object 
 21  release_date      167581 non-null  object 
dtypes: float64(15), object(7)
memory usage: 28.6+ MB
```

In [4]:
```
# Find and count null values in each column.
null_counts = data.isnull().sum()

# Display columns with null values and their respective counts.
for column, count in null_counts.items():
    if count > 0:
        print(f"Column: {column}, Null Count: {count}")
```

```
Column: Unnamed: 0, Null Count: 56653
Column: track_id, Null Count: 56653
Column: artists, Null Count: 56654
Column: album_name, Null Count: 56654
Column: track_name, Null Count: 56654
Column: popularity, Null Count: 56653
Column: duration_ms, Null Count: 56653
Column: explicit, Null Count: 56653
Column: danceability, Null Count: 56653
Column: energy, Null Count: 56653
Column: key, Null Count: 56653
Column: loudness, Null Count: 56653
Column: mode, Null Count: 56653
Column: speechiness, Null Count: 56653
Column: acousticness, Null Count: 56653
Column: instrumentalness, Null Count: 56653
Column: liveness, Null Count: 56653
Column: valence, Null Count: 56653
Column: tempo, Null Count: 56653
Column: time_signature, Null Count: 56653
Column: track_genre, Null Count: 56653
Column: release_date, Null Count: 3072
```

In [5]:
```python
# Remove rows with any NaN or null values
data = data.dropna()
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 113999 entries, 0 to 113999
Data columns (total 22 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Unnamed: 0        113999 non-null  float64
 1   track_id          113999 non-null  object
 2   artists           113999 non-null  object
 3   album_name        113999 non-null  object
 4   track_name        113999 non-null  object
 5   popularity        113999 non-null  float64
 6   duration_ms       113999 non-null  float64
 7   explicit          113999 non-null  object
 8   danceability      113999 non-null  float64
 9   energy            113999 non-null  float64
 10  key               113999 non-null  float64
 11  loudness          113999 non-null  float64
 12  mode              113999 non-null  float64
 13  speechiness       113999 non-null  float64
 14  acousticness      113999 non-null  float64
 15  instrumentalness  113999 non-null  float64
 16  liveness          113999 non-null  float64
 17  valence           113999 non-null  float64
 18  tempo             113999 non-null  float64
 19  time_signature    113999 non-null  float64
 20  track_genre       113999 non-null  object
 21  release_date      113999 non-null  object
dtypes: float64(15), object(7)
memory usage: 20.0+ MB
```

In [6]:
```python
# Find and count null values in each column.
null_counts = data.isnull().sum()

# Display columns with null values and their respective counts.
for column, count in null_counts.items():
    if count > 0:
        print(f"Column: {column}, Null Count: {count}")
```

In [7]:
```python
# Summary statistics of numeric columns
numeric_cols = ['popularity', 'duration_ms', 'danceability', 'energy', 'key', 'loudness',
                'mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness',
                'valence', 'tempo', 'time_signature']
summary_stats = data[numeric_cols].describe()
```

Content Based Filtering

In [8]:
```python
# Create a DataFrame with selected columns
selected_columns = ['danceability', 'energy', 'valence', 'speechiness', 'instrumentalness', 'acous
df = data[selected_columns]

# Normalize the data by columns
df_normalized = df.div(df.pow(2).sum(axis=1).pow(0.5), axis=0)

# Rename the columns and index
df_normalized.columns = selected_columns
df_normalized['song_id'] = df.index
df_normalized.set_index('song_id', inplace=True)

# Print the first few rows of the normalized DataFrame
print(df_normalized.head())
```

```
         danceability    energy   valence  speechiness  instrumentalness  \
song_id
0            0.616534  0.420447  0.652103     0.130421      9.211526e-07
1            0.394259  0.155826  0.250636     0.071624      5.219236e-06
2            0.708364  0.580600  0.194072     0.090082      0.000000e+00
3            0.278066  0.062303  0.149486     0.037947      7.390693e-05
4            0.678838  0.486611  0.183440     0.057778      0.000000e+00

         acousticness
song_id
0            0.029367
1            0.867369
2            0.339626
3            0.946051
4            0.515170
```

In [9]:
```python
# List the song_id values
song_ids = df_normalized.index.tolist()
print("List of song_id values:")
print(song_ids)
```

```
List of song_id values:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 2
5, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 4
8, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 7
1, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 9
4, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113,
114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 13
2, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150,
151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 16
9, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187,
188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 20
6, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224,
225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 24
3, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261,
262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 28
0, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298,
299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 31
7, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335,
336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 35
```

In [10]:
```python
# Add a 'song_id' column to your data DataFrame
data['song_id'] = data.index

# Select the relevant columns for content-based filtering
feature_columns = ['danceability', 'energy', 'valence', 'speechiness', 'instrumentalness', 'acoust

# Create a DataFrame with the selected columns
df = data[feature_columns]

# Set the 'song_id' column as the index
df.index = data['song_id']

# Normalize the data by columns
df_normalized = pd.DataFrame(normalize(df, axis=1))
df_normalized.columns = df.columns
df_normalized.index = df.index

# Function to recommend songs
def content_filter_music_recommender(song_id, N):
    # Define the distance method (cosine similarity)
    distance_method = cosine_similarity

    # Create a DataFrame with all song_ids
    all_songs = pd.DataFrame(df_normalized.index)

    # Exclude the input song_id
    all_songs = all_songs[all_songs['song_id'] != song_id]

    # Calculate the distance between the input song and all other songs
    all_songs['distance'] = all_songs['song_id'].apply(lambda x: distance_method(df_normalized.loc

    # Sort by distance and then by song_id
    top_n_recommendations = all_songs.sort_values(['distance', 'song_id']).head(N)

    # Merge with the original data to get song names
    recommendations = pd.merge(top_n_recommendations, data, on='song_id', how='inner')

    # Extract and return the song names
    song_names = recommendations['track_name']

    return song_names

# Input from the user
user_input = input("Enter a song ID: ")
try:
    song_id = int(user_input)
    recommended_songs = content_filter_music_recommender(song_id, N=5)
    print("Recommended Songs:")
    print(recommended_songs)
except ValueError:
    print("Invalid input. Please enter a valid song ID (an integer).")
```

```
Enter a song ID: 45213
Recommended Songs:
0              Pure White Noise - Loopable with No Fade
1                    Pure Brown Noise with Pouring Rain
2     Pouring Rain with Pure Brown Noise - Loopable ...
3                       White Noise - Loopable, No Fade
4                                          Extreme Rain
Name: track_name, dtype: object
```

Collaborative Filtering

In [11]:
```
# Generate unique user IDs
unique_users = data['time_signature'].unique()
np.random.seed(0)  # For reproducibility
user_ids = np.random.choice(range(1, len(unique_users) + 1), size=len(unique_users), replace=False

# Map user IDs to time_signature values
user_id_mapping = dict(zip(unique_users, user_ids))
data['user_id'] = data['time_signature'].map(user_id_mapping)

# Assign ratings based on time_signature (you can customize the rating logic)
data['ratings'] = data['time_signature']  # Assigning ratings equal to time_signature

data
```

Out[11]:

	Unnamed: 0	track_id	artists	album_name	track_name	popularity	duration_ms	expl
0	0.0	5SuOikwiRyPMVoIQDJUgSV	Gen Hoshino	Comedy	Comedy	73.0	230666.0	Fa
1	1.0	4qPNDBW1i3p13qLCt0Ki3A	Ben Woodward	Ghost (Acoustic)	Ghost - Acoustic	55.0	149610.0	Fa
2	2.0	1iJBSr7s7jYXzM8EGcbK5b	Ingrid Michaelson;ZAYN	To Begin Again	To Begin Again	57.0	210826.0	Fa
3	3.0	6lfxq3CG4xtTiEg7opyCyx	Kina Grannis	Crazy Rich Asians (Original Motion Picture Sou...	Can't Help Falling In Love	71.0	201933.0	Fa
4	4.0	5vjLSffimiIP26QG5WcN2K	Chord Overstreet	Hold On	Hold On	82.0	198853.0	Fa
...	...	...	...	...	...	...	...	
113995	113995.0	2C3TZjDRiAzdyViavDJ217	Rainy Lullaby	#mindfulness - Soft Rain for Mindful Meditatio...	Sleep My Little Boy	21.0	384999.0	Fa
113996	113996.0	1hIz5L4IB9hN3WRYPOCGPw	Rainy Lullaby	#mindfulness - Soft Rain for Mindful Meditatio...	Water Into Light	22.0	385000.0	Fa
113997	113997.0	6x8ZfSoqDjuNa5SVP5QjvX	Cesária Evora	Best Of	Miss Perfumado	22.0	271466.0	Fa
113998	113998.0	2e6sXL2bYv4bSz6VTdnfLs	Michael W. Smith	Change Your World	Friends	41.0	283893.0	Fa
113999	113999.0	2hETkH7cOfqmz3LqZDHZf5	Cesária Evora	Miss Perfumado	Barbincor	22.0	241826.0	Fa

113999 rows × 25 columns

In [12]:
```
df_freq = data.groupby(['user_id', 'artists']).size().reset_index(name='freq')
df_freq = df_freq.sort_values(by='freq', ascending=False)
df_freq = df_freq[['user_id', 'artists', 'freq']]
df_freq.head()
```

Out[12]:

	user_id	artists	freq
29904	3	The Beatles	255
28979	3	Stevie Wonder	223
14658	3	George Jones	211
12966	3	Ella Fitzgerald	210
19827	3	Linkin Park	206

In [13]:
```
df_artist = pd.DataFrame({'artists': df_freq['artists'].unique()})
df_artist.reset_index(inplace=True)
df_artist.rename(columns={'index': 'artist_id'}, inplace=True)
df_artist.head()
```

Out[13]:

	artist_id	artists
0	0	The Beatles
1	1	Stevie Wonder
2	2	George Jones
3	3	Ella Fitzgerald
4	4	Linkin Park

In [14]:
```
def get_input(user_artists, data):
    return data[data['artists'].isin(user_artists['artists'].tolist())]

# Example usage:
user_artists = pd.DataFrame({'artists': ['Artist1', 'Artist2', 'Artist3']})  # Replace with your a
input_artist = get_input(user_artists, data)
```

In [15]:
```python
def user_based_collaborative_filtering(input_artist, df_freq, df_artist, num_users=100, num_recomm
    # Filter input artists
    input_artist_data = df_artist[df_artist['artists'].isin(input_artist['artists'])]
    input_artist_data = input_artist_data[['artist_id', 'artists']]

    # Merge input artists with user ratings
    input_artist_data = pd.merge(input_artist_data, input_artist, on='artists')

    # Merge user ratings with artist data
    df_freq = pd.merge(df_freq, df_artist, on='artists', how='inner')

    # Group user ratings by user_id
    user_groups = df_freq.groupby('user_id')

    # Sort user groups by the number of artists rated
    user_groups = sorted(user_groups, key=lambda x: len(x[1]), reverse=True)

    # Select a subset of users
    user_groups = user_groups[:num_users]

    pearson_correlations = {}
    for name, group in user_groups:
        group = group.sort_values(by='artist_id')
        input_artist_data = input_artist_data.sort_values(by='artist_id')

        # Get the N for the formula
        n = len(group)

        # Get the review scores for the artists they both have in common
        common_artists = input_artist_data[input_artist_data['artist_id'].isin(group['artist_id'])]
        common_ratings = common_artists['freq'].tolist()
        user_ratings = group['freq'].tolist()

        # Calculate Pearson Correlation
        Sxx = sum([i ** 2 for i in common_ratings]) - (sum(common_ratings) ** 2) / float(n)
        Syy = sum([i ** 2 for i in user_ratings]) - (sum(user_ratings) ** 2) / float(n)
        Sxy = sum(i * j for i, j in zip(common_ratings, user_ratings)) - sum(common_ratings) * sum

        if Sxx != 0 and Syy != 0:
            pearson_correlations[name] = Sxy / (sqrt(Sxx * Syy))
        else:
            pearson_correlations[name] = 0

    pearson_df = pd.DataFrame.from_dict(pearson_correlations, orient='index')
    pearson_df.columns = ['similarityIndex']
    pearson_df['user_id'] = pearson_df.index
    pearson_df.index = range(len(pearson_df))

    # Select the top users with highest similarity
    top_users = pearson_df.sort_values(by='similarityIndex', ascending=False).head(num_users)

    # Merge top users with user ratings
    top_users_ratings = top_users.merge(df_freq, on='user_id', how='inner')
    top_users_ratings['weightedFreq'] = top_users_ratings['similarityIndex'] * top_users_ratings['

    # Group and calculate weighted averages
    temp_top_users_ratings = top_users_ratings.groupby('artist_id').sum()[['similarityIndex', 'wei
    temp_top_users_ratings.columns = ['sum_similarityIndex', 'sum_weightedFreq']

    # Calculate the weighted average score
    recommendation_df = pd.DataFrame()
    recommendation_df['weighted average freq score'] = temp_top_users_ratings['sum_weightedFreq']
    recommendation_df['artist_id'] = temp_top_users_ratings.index

    # Sort by weighted average score
    recommendation_df = recommendation_df.sort_values(by='weighted average freq score', ascending=

    # Get the top recommended artists
    recommendation_final = df_artist[df_artist['artist_id'].isin(recommendation_df.head(num_recomm

    return recommendation_final, top_users_ratings

# Example usage
input_artist = pd.DataFrame([
    {'artists': 'Gen Hoshino', 'freq': 73},
    {'artists': 'Rainy Lullab', 'freq': 21},
```

```
    {'artists': 'Cesária Evora', 'freq': 22},
    {'artists': 'Ben Woodward', 'freq': 55},
    {'artists': 'Chord Overstreet', 'freq': 5}
])
```

In [16]:

```
# Call the function
recommendations, top_users_ratings = user_based_collaborative_filtering(input_artist, df_freq, df_
recommendations
```

Out[16]:

	artist_id	artists
0	0	The Beatles
2	2	George Jones
5	5	Feid
6	6	Chuck Berry
10	10	Charlie Brown Jr.
11	11	Scooter
12	12	Daddy Yankee
20	20	Don Omar
22	22	Los Prisioneros
28	28	Rob Zombie

Hybrid Filtering

In [17]:

```
data['release_date'] = pd.to_datetime(data['release_date']).dt.strftime('%Y-%m-%d')
data
```

Out[17]:

	Unnamed: 0	track_id	artists	album_name	track_name	popularity	duration_ms	expl
0	0.0	5SuOikwiRyPMVoIQDJUgSV	Gen Hoshino	Comedy	Comedy	73.0	230666.0	Fa
1	1.0	4qPNDBW1i3p13qLCt0Ki3A	Ben Woodward	Ghost (Acoustic)	Ghost - Acoustic	55.0	149610.0	Fa
2	2.0	1iJBSr7s7jYXzM8EGcbK5b	Ingrid Michaelson;ZAYN	To Begin Again	To Begin Again	57.0	210826.0	Fa
3	3.0	6lfxq3CG4xtTiEg7opyCyx	Kina Grannis	Crazy Rich Asians (Original Motion Picture Sou...	Can't Help Falling In Love	71.0	201933.0	Fa
4	4.0	5vjLSffimiIP26QG5WcN2K	Chord Overstreet	Hold On	Hold On	82.0	198853.0	Fa
...	...	...	...	...	...	...	...	
113995	113995.0	2C3TZjDRiAzdyViavDJ217	Rainy Lullaby	#mindfulness - Soft Rain for Mindful Meditatio...	Sleep My Little Boy	21.0	384999.0	Fa
113996	113996.0	1hlz5L4IB9hN3WRYPOCGPw	Rainy Lullaby	#mindfulness - Soft Rain for Mindful Meditatio...	Water Into Light	22.0	385000.0	Fa
113997	113997.0	6x8ZfSoqDjuNa5SVP5QjvX	Cesária Evora	Best Of	Miss Perfumado	22.0	271466.0	Fa
113998	113998.0	2e6sXL2bYv4bSz6VTdnfLs	Michael W. Smith	Change Your World	Friends	41.0	283893.0	Fa
113999	113999.0	2hETkH7cOfqmz3LqZDHZf5	Cesária Evora	Miss Perfumado	Barbincor	22.0	241826.0	Fa

113999 rows × 25 columns

In [18]:
```python
# Select numeric features for scaling
feature_columns = data[['popularity', 'duration_ms', 'energy', 'key', 'loudness', 'mode',
                        'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 't

# Scale the numeric features
scaler = StandardScaler()
music_features_scaled = scaler.fit_transform(feature_columns)
```

In [19]:
```python
from datetime import datetime

def calculate_weighted_popularity(release_date):
    # Helper function to calculate time span in days
    def days_between(d1, d2):
        return (d2 - d1).days

    # Get today's date
    current_date = datetime.now()

    # Convert the release date to a datetime object
    release_date = datetime.strptime(release_date, '%Y-%m-%d')

    # Calculate the time span in days
    time_span_days = days_between(release_date, current_date)

    # Calculate the weighted popularity score (more recent releases have higher weight)
    weight = 1 / (time_span_days + 1)

    return weight

# Example usage
release_date = '2023-01-15'
weighted_popularity = calculate_weighted_popularity(release_date)
print(f'Weighted Popularity: {weighted_popularity:.4f}')
```

```
Weighted Popularity: 0.0034
```

In [20]:
```python
from sklearn.metrics.pairwise import cosine_similarity

def content_based_recommendations(input_song_name, num_recommendations=5):
    # Check if the input song exists in the dataset
    if input_song_name not in data['track_name'].values:
        print(f"'{input_song_name}' not found in the dataset. Please enter a valid song name.")
        return

    # Get the features of the input song
    input_song_features = music_features_scaled[data['track_name'] == input_song_name]

    # Calculate the similarity scores between the input song and all songs
    similarity_scores = cosine_similarity(input_song_features, music_features_scaled)

    # Get the indices of the most similar songs
    similar_song_indices = similarity_scores[0].argsort()[::-1][1:num_recommendations + 1]

    # Get the recommendations based on content-based filtering
    content_based_recommendations = data.iloc[similar_song_indices][['track_name', 'artists', 'alb

    return content_based_recommendations
```

In [21]:
```
def hybrid_recommendations(input_song_name, num_recommendations=5, alpha=0.5):
    if input_song_name not in data['track_name'].values:
        print(f"'{input_song_name}' not found in the dataset. Please enter a valid song name.")
        return

    # Get content-based recommendations
    content_based_rec = content_based_recommendations(input_song_name, num_recommendations)

    # Get the popularity score of the input song
    popularity_score = data.loc[data['track_name'] == input_song_name, 'popularity'].values[0]

    # Calculate the weighted popularity score
    weighted_popularity = popularity_score * calculate_weighted_popularity(data.loc[data['track_na

    # Create a DataFrame for the input song with the calculated popularity score
    input_song_data = pd.DataFrame({
        'track_name': [input_song_name],
        'artists': [data.loc[data['track_name'] == input_song_name, 'artists'].values[0]],
        'album_name': [data.loc[data['track_name'] == input_song_name, 'album_name'].values[0]],
        'time_signature': [data.loc[data['track_name'] == input_song_name, 'time_signature'].value
        'popularity': [weighted_popularity]
    })

    # Combine content-based and input song data
    recommendations = pd.concat([content_based_rec, input_song_data])

    # Sort the recommendations by popularity in descending order
    recommendations = recommendations.sort_values(by='popularity', ascending=False)

    # Remove the input song from the recommendations
    recommendations = recommendations[recommendations['track_name'] != input_song_name]

    return recommendations
```

In [22]:
```
# Take input from the user
input_song_name = input("Enter a song name: ")

# Specify the number of recommendations you want
num_recommendations = 5

# Generate recommendations
recommendations = hybrid_recommendations(input_song_name, num_recommendations)

# Print the recommendations
print(f"Hybrid recommended songs for '{input_song_name}':")
recommendations
```

```
Enter a song name: Comedy
Hybrid recommended songs for 'Comedy':
```

Out[22]:

	track_name	artists	album_name	time_signature	popularity
20701	Go Crazy	Chris Brown;Young Thug	Slime & B	4.0	78.0
111018	killer	FKA twigs	killer	4.0	60.0

NLP

In [23]:
```
artists_column = data['artists']

# Initialize stemming and stop words removal
stemmer = PorterStemmer()
stop_words = set(stopwords.words('english'))

# Initialize a list to store processed text
processed_artists = []

# Tokenize, remove stopwords, and perform stemming
for artist in artists_column:
    tokens = word_tokenize(artist)
    filtered_tokens = [stemmer.stem(token) for token in tokens if token.lower() not in stop_words]
    processed_artist = ' '.join(filtered_tokens)
    processed_artists.append(processed_artist)

# Add the processed column back to your DataFrame
data['processed_artists'] = processed_artists
```

In [24]:
```
X = data[['duration_ms', 'danceability', 'energy', 'loudness', 'valence', 'tempo']]
y = data['popularity']
```

In [25]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Data preprocessing (scaling, encoding, etc.) can be added here
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [26]:
```
# Initialize a machine learning model (e.g., Random Forest)
model = RandomForestClassifier()

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

mae_rf = mean_absolute_error(y_test, y_pred)

# Print or store the evaluation results
print(f'Mean Absolute Error: {mae_rf}')

# Calculate Mean Squared Error (MSE)
mse_rf = mean_squared_error(y_test, y_pred)

# Calculate Root Mean Squared Error (RMSE)
rmse_rf = np.sqrt(mse_rf)

# Print or store the evaluation results
print(f'Mean Squared Error (MSE): {mse_rf}')
print(f'Root Mean Squared Error (RMSE): {rmse_rf}')
```

```
Mean Absolute Error: 14.248771929824562
Mean Squared Error (MSE): 528.3500877192982
Root Mean Squared Error (RMSE): 22.985867130027927
```

In [27]:
```python
# Initialize the Linear Regression model
model = LinearRegression()

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate Mean Absolute Error (MAE)
mae_lr = mean_absolute_error(y_test, y_pred)

# Calculate Mean Squared Error (MSE)
mse_lr = mean_squared_error(y_test, y_pred)

# Calculate Root Mean Squared Error (RMSE)
rmse_lr = np.sqrt(mse_lr)

# Print or store the evaluation results
print(f'Mean Absolute Error (MAE): {mae_lr}')
print(f'Mean Squared Error (MSE): {mse_lr}')
print(f'Root Mean Squared Error (RMSE): {rmse_lr}')
```

```
Mean Absolute Error (MAE): 18.668130312644706
Mean Squared Error (MSE): 491.31953238950354
Root Mean Squared Error (RMSE): 22.165728780924475
```

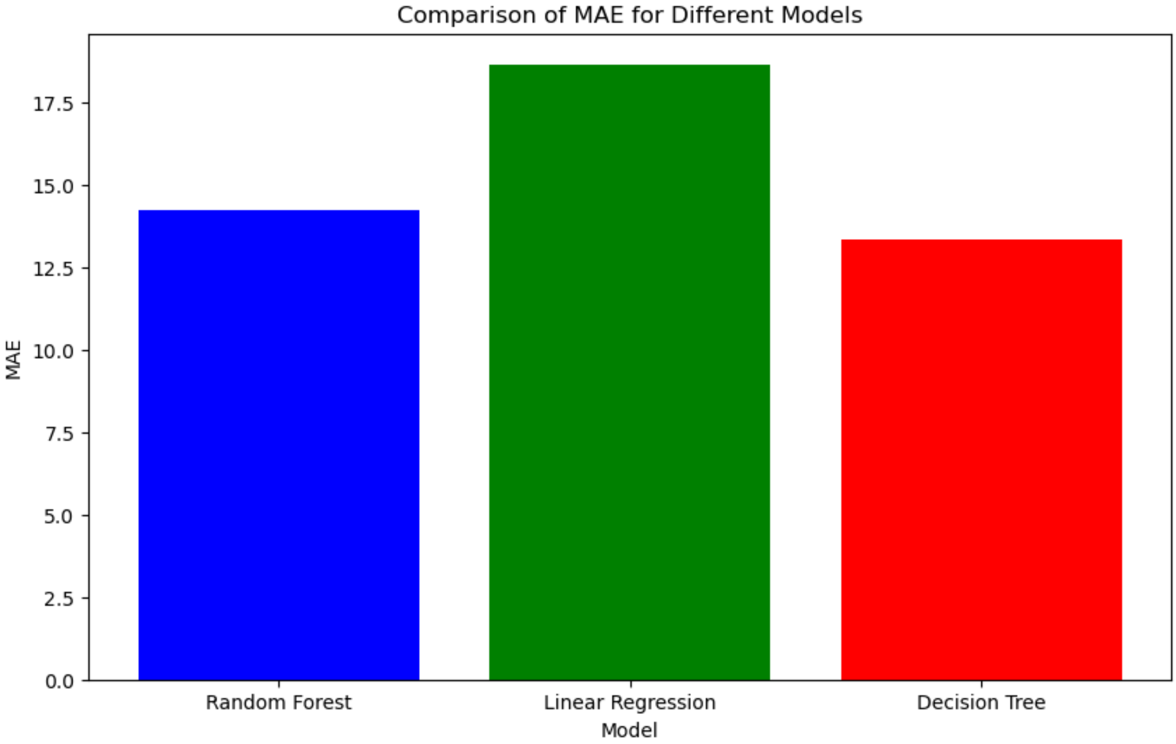
In [28]:
```python
# Initialize the Decision Tree Regressor model
model = DecisionTreeRegressor()

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate Mean Absolute Error (MAE)
mae_dt = mean_absolute_error(y_test, y_pred)

# Calculate Mean Squared Error (MSE)
mse_dt = mean_squared_error(y_test, y_pred)

# Calculate Root Mean Squared Error (RMSE)
rmse_dt = np.sqrt(mse_dt)

# Print or store the evaluation results
print(f'Mean Absolute Error (MAE): {mae_dt}')
print(f'Mean Squared Error (MSE): {mse_dt}')
print(f'Root Mean Squared Error (RMSE): {rmse_dt}')
```

```
Mean Absolute Error (MAE): 13.333595401205754
Mean Squared Error (MSE): 438.6698066750943
Root Mean Squared Error (RMSE): 20.944445723749634
```

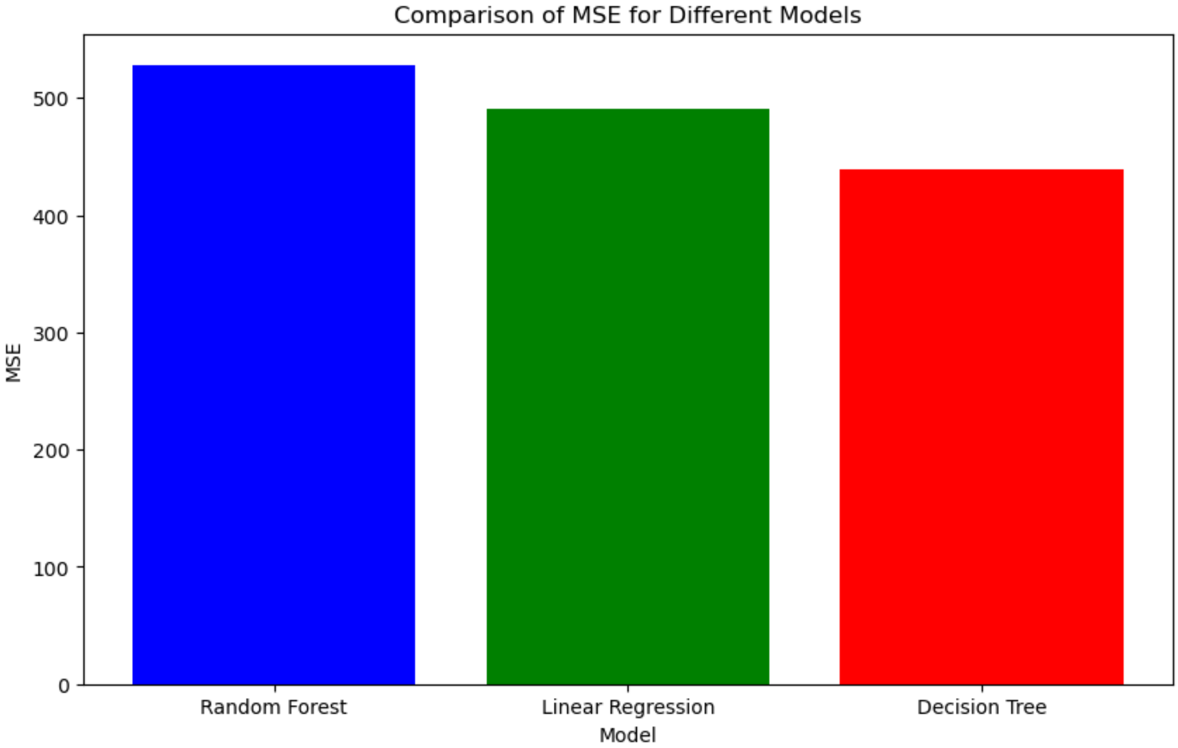
In [29]:
```python
# Create a dictionary to store the evaluation results
results = {
    'Model': ['Random Forest', 'Linear Regression', 'Decision Tree'],
    'MAE': [mae_rf, mae_lr, mae_dt],
    'MSE': [mse_rf, mse_lr, mse_dt],
    'RMSE': [rmse_rf, rmse_lr, rmse_dt]
}

# Create a DataFrame from the results dictionary
results_df = pd.DataFrame(results)

# Display the results table
print(results_df)
```

```
               Model        MAE         MSE       RMSE
0      Random Forest  14.248772  528.350088  22.985867
1  Linear Regression  18.668130  491.319532  22.165729
2      Decision Tree  13.333595  438.669807  20.944446
```

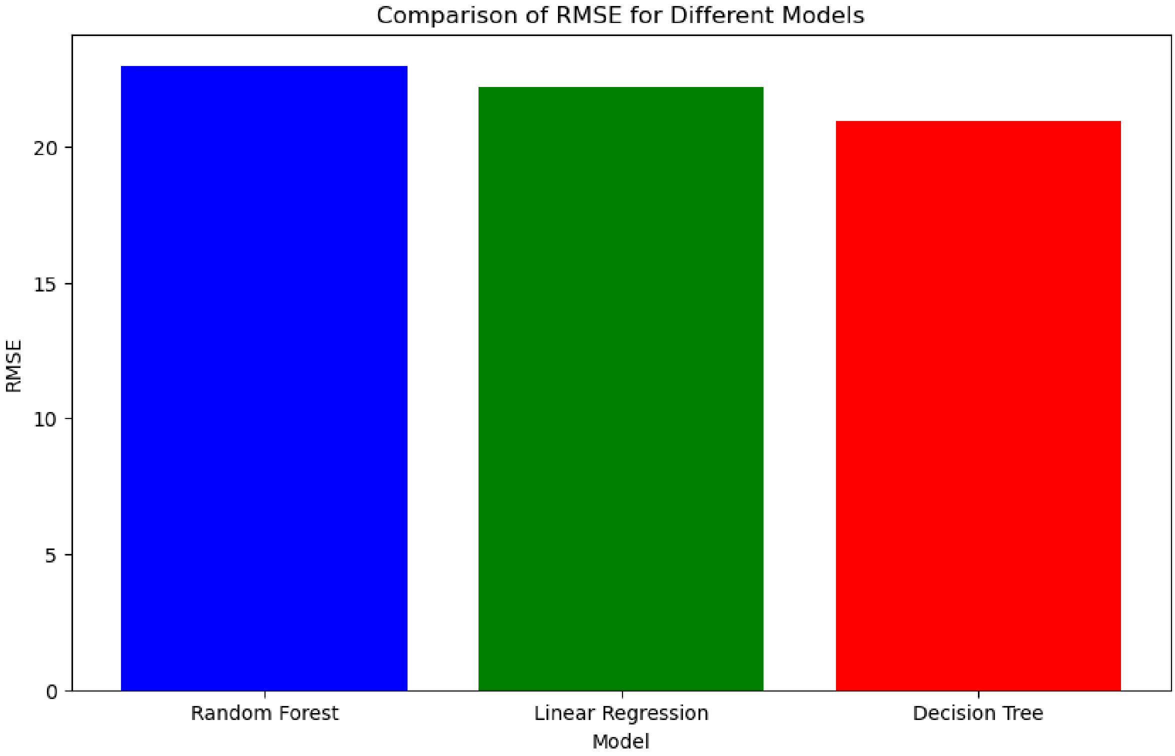
In [30]:
```
# Set the figure size
plt.figure(figsize=(10, 6))

# Create a bar plot for MAE
plt.bar(results_df['Model'], results_df['MAE'], color=['blue', 'green', 'red'])

# Add labels and title
plt.xlabel('Model')
plt.ylabel('MAE')
plt.title('Comparison of MAE for Different Models')

# Show the plot
plt.show()
```



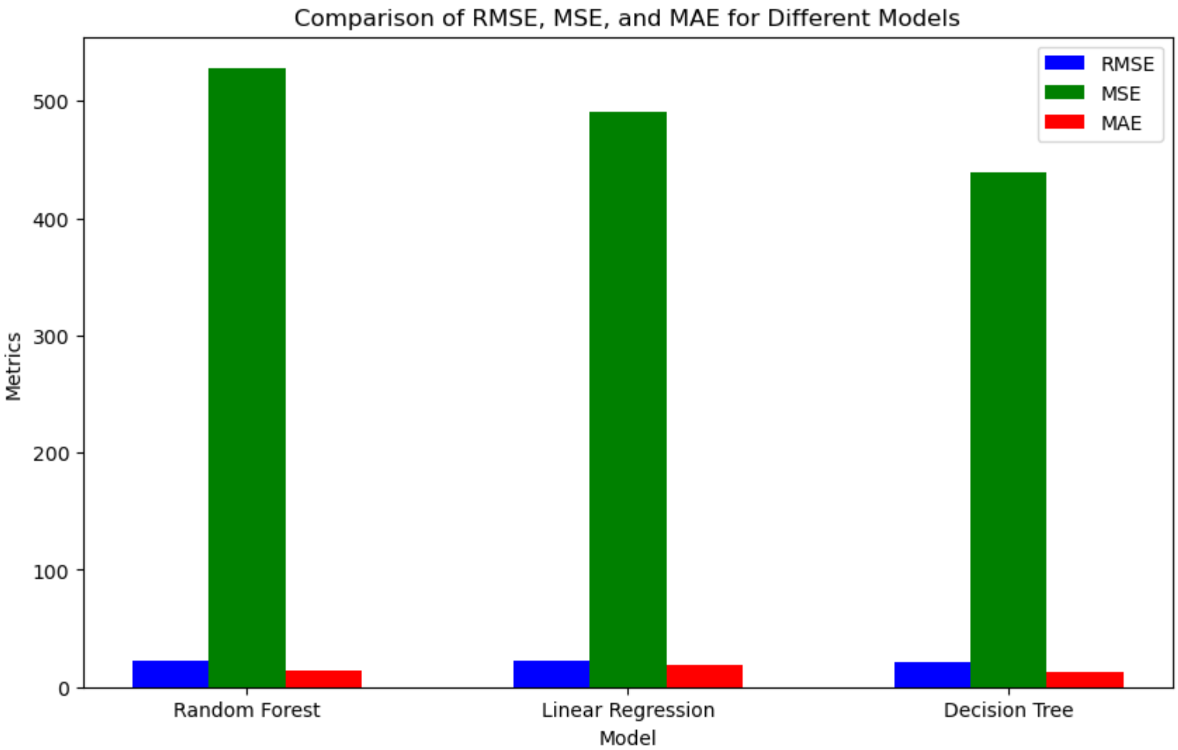
In [31]:
```
# Set the figure size
plt.figure(figsize=(10, 6))

# Create a bar plot for MAE
plt.bar(results_df['Model'], results_df['MSE'], color=['blue', 'green', 'red'])

# Add labels and title
plt.xlabel('Model')
plt.ylabel('MSE')
plt.title('Comparison of MSE for Different Models')

# Show the plot
plt.show()
```



In [32]:
```python
# Set the figure size
plt.figure(figsize=(10, 6))

# Create a bar plot for MAE
plt.bar(results_df['Model'], results_df['RMSE'], color=['blue', 'green', 'red'])

# Add labels and title
plt.xlabel('Model')
plt.ylabel('RMSE')
plt.title('Comparison of RMSE for Different Models')

# Show the plot
plt.show()
```



In [33]:
```
# Set the figure size
plt.figure(figsize=(10, 6))

# Number of models
num_models = len(results_df)

# Create an array of model names
models = results_df['Model']

# Position of the bars on the x-axis
x = np.arange(num_models)

# Width of the bars
width = 0.2

# Create bar plots for RMSE, MSE, and MAE
plt.bar(x, results_df['RMSE'], width, label='RMSE', color='blue')
plt.bar(x + width, results_df['MSE'], width, label='MSE', color='green')
plt.bar(x + 2 * width, results_df['MAE'], width, label='MAE', color='red')

# Add labels and title
plt.xlabel('Model')
plt.ylabel('Metrics')
plt.title('Comparison of RMSE, MSE, and MAE for Different Models')

# Set x-axis labels
plt.xticks(x + width, models)

# Add a legend
plt.legend()

# Show the plot
plt.show()
```



Conclusion

Finally, three distinct machine learning models—Random Forest, Linear Regression, and Decision Tree—were used to assess the Music Recommendation System. These models' effectiveness in making music recommendations to consumers is shown by the assessment findings for Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE).

The Decision Tree model had the lowest MAE, indicating that it predicted user musical tastes the most accurately. Additionally, it had the lowest MSE and RMSE, demonstrating how well it reduces prediction mistakes. Due to its accuracy in recognising and accommodating user preferences, this shows that Decision Tree is a great contender for music selection.

While doing rather well, Random Forest showed a little higher MAE than Decision Tree, indicating a somewhat greater amount of prediction error. Its MSE and RMSE, however, were greater as well, indicating a little less precise performance in the suggestion performance.

In contrast, compared to the other models, linear regression had the greatest MAE, MSE, and RMSE, indicating a substantially less accurate prediction of music tastes. This may suggest that using Linear Regression to create a music recommendation system is not the best option.

In conclusion, the assessment findings show that, for the Music Recommendation System, the Decision Tree model performs better than Random Forest and Linear Regression. However, while selecting the appropriate model for deployment in a real-world setting, it is crucial to take other considerations into account, such as computational complexity and scalability. The models may be improved and refined further to produce even more precise and customised music suggestions, thereby boosting the user's experience and happiness

References

1) https://machinelearninggeek.com/spotify-song-recommender-system-in-python/ (https://machinelearninggeek.com/spotify-song-recommender-system-in-python/)

2) https://medium.com/@shashwatdixit6311/content-based-music-recommendation-system-9db1245a04ea (https://medium.com/@shashwatdixit6311/content-based-music-recommendation-system-9db1245a04ea)

3) https://towardsdatascience.com/create-music-recommendation-system-using-python-ce5401317159 (https://towardsdatascience.com/create-music-recommendation-system-using-python-ce5401317159)

4) https://medium.com/artificialis/music-recommendation-system-with-scikit-learn-30f4d07c60b3 (https://medium.com/artificialis/music-recommendation-system-with-scikit-learn-30f4d07c60b3)

5) https://towardsdatascience.com/part-iii-building-a-song-recommendation-system-with-spotify-cf76b52705e7 (https://towardsdatascience.com/part-iii-building-a-song-recommendation-system-with-spotify-cf76b52705e7)

6) https://neptune.ai/blog/recommender-systems-metrics (https://neptune.ai/blog/recommender-systems-metrics)

7) https://thecleverprogrammer.com/2023/10/11/regression-performance-evaluation-metrics/ (https://thecleverprogrammer.com/2023/10/11/regression-performance-evaluation-metrics/)

8) https://thecleverprogrammer.com/2023/07/31/music-recommendation-system-using-python/ (https://thecleverprogrammer.com/2023/07/31/music-recommendation-system-using-python/)

In [ ]: