Spotify recommendation system

```
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import NearestNeighbors

# Load song dataset
# Dataset should have columns: 'song_id', 'title', 'artist', 'genre', 'features', and 'user_id' for
collaborative filtering
# Example CSV format:
# song_id,title,artist,genre,features (a list of numerical features like tempo, energy, danceability),
user_id
song_data = pd.read_csv("songs.csv")
user_data = pd.read_csv("user_ratings.csv")

# Collaborative Filtering - Recommend songs based on similar users' ratings

# Step 1: Create a pivot table of users and their song ratings
user_song_matrix = user_data.pivot_table(index='user_id', columns='song_id',
values='rating').fillna(0)

# Step 2: Calculate cosine similarity between users
user_similarity = cosine_similarity(user_song_matrix)
user_similarity_df = pd.DataFrame(user_similarity, index=user_song_matrix.index,
columns=user_song_matrix.index)

# Step 3: Function to get song recommendations for a user based on similar users
def recommend_songs_collaborative(user_id, num_recommendations=5):
    # Find similar users
    similar_users =
user_similarity_df[user_id].sort_values(ascending=False).index[1:num_recommendations + 1]

    # Get the songs these users like that the target user hasn't rated yet
    recommended_songs = pd.Series(dtype='float64')
    for similar_user in similar_users:
        songs = user_song_matrix.loc[similar_user]
        recommended_songs = recommended_songs.append(songs[songs > 0])

    # Sort and recommend top songs not yet rated by the target user
    recommended_songs =
recommended_songs[~recommended_songs.index.isin(user_song_matrix.loc[user_id][user_so
ng_matrix.loc[user_id] > 0].index)]
    return recommended_songs.nlargest(num_recommendations)

# Content-Based Filtering - Recommend songs with similar content features

# Step 1: Prepare feature data
```

```
# Combine genre, title, and artist into a single feature for content-based similarity (could use
more sophisticated features)
song_data['content_features'] = song_data['genre'] + " " + song_data['title'] + " " +
song_data['artist']
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(song_data['content_features'])

# Step 2: Calculate cosine similarity based on content features
song_similarity = cosine_similarity(tfidf_matrix, tfidf_matrix)
song_similarity_df = pd.DataFrame(song_similarity, index=song_data['song_id'],
columns=song_data['song_id'])

# Step 3: Function to get song recommendations based on content similarity
def recommend_songs_content(song_id, num_recommendations=5):
    # Get the similarity scores for the given song_id
    similar_songs =
song_similarity_df[song_id].sort_values(ascending=False).index[1:num_recommendations + 1]
    return song_data[song_data['song_id'].isin(similar_songs)][['title', 'artist']]

# Hybrid Recommendation - Combining both methods
def hybrid_recommendation(user_id, song_id, num_recommendations=5):
    # Get collaborative recommendations
    collab_recommendations = recommend_songs_collaborative(user_id,
num_recommendations)
    # Get content-based recommendations
    content_recommendations = recommend_songs_content(song_id, num_recommendations)

    # Combine results, prioritize collaborative recommendations
    combined = pd.concat([collab_recommendations,
content_recommendations]).drop_duplicates()
    return combined.head(num_recommendations)

# Test the recommendation functions
user_id = 1  # Specify a test user ID
song_id = 101  # Specify a test song ID

print("Collaborative Filtering Recommendations for User", user_id)
print(recommend_songs_collaborative(user_id))

print("\nContent-Based Recommendations for Song ID", song_id)
print(recommend_songs_content(song_id))

print("\nHybrid Recommendations for User", user_id, "and Song ID", song_id)
print(hybrid_recommendation(user_id, song_id))
```