PROFILE 1, PROJECT 2

2. Music Recommendation System (ML, Difficulty: 5)

Description:

This project focuses on creating an intelligent system that suggests music to users based on their individual preferences and listening history. The recommendation system analyzes user behavior, song characteristics, and historical data to provide personalized music suggestions.

Abstract:

The Music Recommendation System project aims to enhance the user experience in music streaming platforms by offering tailored song suggestions. By leveraging machine learning algorithms, particularly k-Nearest Neighbors (kNN) and Collaborative Filtering, the system analyzes patterns in user listening habits and song attributes to generate accurate and diverse recommendations. The primary goal is to improve music discovery for users, making it more efficient and enjoyable. This project addresses the challenge of information overload in vast music libraries and aims to increase user engagement and satisfaction with music streaming services.

Proposed Algorithm:

Two main algorithms are proposed for this project:

1. k-Nearest Neighbors (kNN): This algorithm will be used to find similar songs based on their features. It works by identifying the k most similar songs to a given song in the feature space.

2. Collaborative Filtering: This technique will be employed to generate recommendations based on user behavior. It analyzes patterns in user preferences and finds similarities between users to suggest songs that similar users have enjoyed.

Dataset:

The dataset mentioned in the project description is the Spotify dataset available on Kaggle. Based on the search results provided, the link for this dataset is:

https://www.kaggle.com/datasets/yamaerenay/spotify-dataset-19212020-600k-tracks

This dataset includes audio features of over 600,000 tracks and popularity metrics for more than 1 million artists. It appears to be a comprehensive collection of Spotify data that would be suitable for the Music Recommendation System project described, as it contains the necessary information about songs, including audio features and popularity metrics, which are crucial for implementing both the k-Nearest Neighbors and Collaborative Filtering algorithms proposed in the project.

Citations:
[1] https://www.kaggle.com/datasets/yamaerenay/spotify-dataset-19212020-600k-tracks

Useful links:

Here are 20 useful links related to the Music Recommendation System project, with explanations of their relevance and summaries in plain text format:

1. https://www.restack.io/p/music-recommendation-answer-system-cat-ai
Relevance: Provides an overview of music recommendation systems and techniques.
Summary: Discusses collaborative filtering, deep learning, and other approaches used in music recommendation systems. Explains user-user and item-item collaborative filtering methods.

2. https://github.com/Gokul-Raja84/Spotify-Music-Recommendation-and-Data-Analysis
Relevance: Contains a music recommendation system implementation using Spotify data.
Summary: GitHub repository with code for a music recommendation system using Spotify data. Includes data analysis, feature engineering, and recommendation algorithms.

3. https://github.com/SwathyMM/Top-10-song-recommendation-using-collaborative-filtering-and-KNN
Relevance: Implements a song recommendation system using collaborative filtering and KNN.
Summary: GitHub repository with code for recommending top 10 songs to users based on collaborative filtering and K-Nearest Neighbors algorithm.

4. https://github.com/sathishprasad/Music-Recommendation-System
Relevance: Provides a music recommendation system implementation using the Spotify dataset.
Summary: GitHub repository with code for a music recommendation system using machine learning techniques and the Spotify dataset.

5. https://github.com/ABSounds/MusicRecommenderCF
Relevance: Implements a music recommender system using collaborative filtering.
Summary: GitHub repository with code for a music recommendation system based on collaborative filtering, using the ListenBrainz dataset.

6. https://github.com/AyaKhaledSaif/Music-Recommendation-using-Kmeans-KNN
Relevance: Implements a music recommendation system using K-means and KNN algorithms.

Summary: GitHub repository with code for a music recommendation system using K-means clustering and K-Nearest Neighbors algorithms.

7. https://pyimagesearch.com/2023/10/30/spotify-music-recommendation-systems/
Relevance: Explains Spotify's approach to music recommendation systems.
Summary: Article discussing various techniques used by Spotify for music recommendations, including matrix factorization, RNNs, and reinforcement learning.

8. https://github.com/asrinutku/music-recommendation
Relevance: Provides an item-based music recommender using KNN algorithm.
Summary: GitHub repository with code for an item-based music recommendation system using the K-Nearest Neighbors algorithm.

9. https://web-ainf.aau.at/pub/jannach/files/Workshop_RecSys_Challenge_2018.pdf
Relevance: Academic paper on effective nearest-neighbor music recommendations.
Summary: Research paper discussing nearest-neighbor techniques for music recommendation, including collaborative filtering and session-based approaches.

10. https://developer.spotify.com/documentation/web-api/
Relevance: Official Spotify Web API documentation.
Summary: Provides information on accessing Spotify's data and features, which can be useful for integrating real-world data into the project.

11. https://www.kaggle.com/datasets/vatsalmavani/spotify-dataset
Relevance: Provides access to a Spotify dataset for music recommendation.
Summary: Kaggle dataset containing Spotify track data, including audio features and popularity metrics, useful for training recommendation models.

12. https://towardsdatascience.com/how-to-build-a-simple-song-recommender-296fcbc8c85
Relevance: Tutorial on building a simple song recommender system.
Summary: Step-by-step guide on creating a basic music recommendation system using Python and collaborative filtering techniques.

13. https://github.com/microsoft/recommenders
Relevance: Microsoft's repository of recommendation system algorithms.
Summary: Comprehensive collection of recommendation algorithms, including those applicable to music recommendation, with example implementations.

14. https://arxiv.org/abs/1901.04555
Relevance: Research paper on deep learning for music recommendation.
Summary: Academic paper discussing the application of deep learning techniques in music recommendation systems.

15. https://www.tensorflow.org/recommenders
Relevance: TensorFlow Recommenders library documentation.
Summary: Official documentation for TensorFlow Recommenders, a library useful for building recommendation systems, including music recommenders.

16. https://surprise.readthedocs.io/en/stable/
Relevance: Documentation for the Surprise library, designed for building and analyzing recommender systems.
Summary: Python scikit for building and analyzing recommender systems, including algorithms relevant to music recommendation.

17. https://www.cs.cornell.edu/~shuochen/lme/data_io/song_data.html
Relevance: Provides access to the Million Song Dataset.
Summary: Information about and access to the Million Song Dataset, a large dataset of audio features and metadata for contemporary music tracks.

18. https://github.com/ybayle/awesome-deep-learning-music
Relevance: Curated list of deep learning resources for music-related tasks.
Summary: GitHub repository containing a comprehensive list of resources, including papers and implementations, related to deep learning in music, including recommendation systems.

19. https://www.sciencedirect.com/science/article/pii/S0950705118301679
Relevance: Academic paper on hybrid techniques for music recommendation.
Summary: Research paper discussing hybrid approaches combining content-based and collaborative filtering techniques for music recommendation.

20. https://github.com/guoguibing/librec
Relevance: Java library for recommender systems.
Summary: Open-source Java library for recommender systems that can be adapted for music recommendation tasks.

Citations:

[1] https://www.restack.io/p/music-recommendation-answer-system-cat-ai
[2] https://github.com/Gokul-Raja84/Spotify-Music-Recommendation-and-Data-Analysis
[3] https://github.com/SwathyMM/Top-10-song-recommendation-using-collaborative-filtering-and-KNN
[4] https://github.com/sathishprasad/Music-Recommendation-System
[5] https://github.com/ABSounds/MusicRecommenderCF
[6] https://github.com/AyaKhaledSaif/Music-Recommendation-using-Kmeans-KNN
[7] https://pyimagesearch.com/2023/10/30/spotify-music-recommendation-systems/
[8] https://github.com/asrinutku/music-recommendation
[9] https://web-ainf.aau.at/pub/jannach/files/Workshop_RecSys_Challenge_2018.pdf

Python code sample:

```python
# Music Recommendation System
# Author: [Your Name]
```

```
# Date: [Current Date]

# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.neighbors import NearestNeighbors
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import precision_score, recall_score, f1_score, ndcg_score
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA, TruncatedSVD
from scipy.sparse import csr_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from typing import List, Tuple, Dict
import warnings
from tqdm import tqdm
import joblib
from surprise import SVD, Dataset, Reader
from surprise.model_selection import cross_validate

# Suppress warnings for cleaner output
warnings.filterwarnings('ignore')

# Set up matplotlib for inline plotting in Jupyter
%matplotlib inline

# 1. Data Loading and Preprocessing

print("1. Data Loading and Preprocessing")

# Load the Spotify dataset
# Note: Replace 'spotify_dataset.csv' with the actual path to your dataset
spotify_data = pd.read_csv('spotify_dataset.csv')

print("Dataset loaded. Shape:", spotify_data.shape)
spotify_data.head()

# Check for missing values
missing_values = spotify_data.isnull().sum()
print("\nMissing values:\n", missing_values)

# Handle missing values
```

```
# For numerical columns, we'll impute with median
# For categorical columns, we'll impute with mode
numerical_columns = spotify_data.select_dtypes(include=[np.number]).columns
categorical_columns = spotify_data.select_dtypes(exclude=[np.number]).columns

for col in numerical_columns:
    spotify_data[col].fillna(spotify_data[col].median(), inplace=True)

for col in categorical_columns:
    spotify_data[col].fillna(spotify_data[col].mode()[0], inplace=True)

print("Shape after handling missing values:", spotify_data.shape)

# Select relevant features for recommendation
features = ['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness',
            'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo']

# Normalize the features using StandardScaler
scaler = StandardScaler()
spotify_data_normalized = pd.DataFrame(scaler.fit_transform(spotify_data[features]),
columns=features)

# 2. Exploratory Data Analysis (EDA)

print("\n2. Exploratory Data Analysis")

# Visualize feature distributions
plt.figure(figsize=(15, 10))
spotify_data[features].boxplot()
plt.title("Distribution of Audio Features")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Visualize correlations between features
plt.figure(figsize=(12, 10))
sns.heatmap(spotify_data[features].corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap of Audio Features")
plt.tight_layout()
plt.show()

# 3. Implement k-Nearest Neighbors (kNN) Algorithm

print("\n3. Implementing k-Nearest Neighbors (kNN) Algorithm")
```

```
def knn_recommendation(song_id: str, k: int = 5) -> List[str]:
    """
    Recommend songs using k-Nearest Neighbors algorithm.

    Args:
    song_id (str): ID of the song to base recommendations on
    k (int): Number of neighbors to consider

    Returns:
    List[str]: List of recommended song IDs
    """
    # Find the index of the given song
    song_idx = spotify_data[spotify_data['id'] == song_id].index[0]

    # Create and fit the kNN model
    knn_model = NearestNeighbors(n_neighbors=k+1, metric='euclidean')
    knn_model.fit(spotify_data_normalized)

    # Find k nearest neighbors
    distances, indices =
knn_model.kneighbors(spotify_data_normalized.iloc[song_idx].values.reshape(1, -1))

    # Get the IDs of recommended songs (excluding the input song)
    recommended_song_ids = spotify_data.iloc[indices[0][1:]]['id'].tolist()

    return recommended_song_ids

# Optimize kNN parameters using GridSearchCV
param_grid = {'n_neighbors': [3, 5, 7, 9, 11], 'metric': ['euclidean', 'manhattan', 'cosine']}
knn_model = NearestNeighbors()
grid_search = GridSearchCV(knn_model, param_grid, cv=5,
scoring='neg_mean_squared_error')
grid_search.fit(spotify_data_normalized)

print("Best kNN parameters:", grid_search.best_params_)

# Update knn_recommendation function with best parameters
best_k = grid_search.best_params_['n_neighbors']
best_metric = grid_search.best_params_['metric']

def optimized_knn_recommendation(song_id: str, k: int = best_k) -> List[str]:
    song_idx = spotify_data[spotify_data['id'] == song_id].index[0]
    knn_model = NearestNeighbors(n_neighbors=k+1, metric=best_metric)
```

```
    knn_model.fit(spotify_data_normalized)
    distances, indices = 
knn_model.kneighbors(spotify_data_normalized.iloc[song_idx].values.reshape(1, -1))
    return spotify_data.iloc[indices[0][1:]]['id'].tolist()

# Test optimized kNN recommendation
test_song_id = spotify_data['id'].iloc[0]
knn_recommendations = optimized_knn_recommendation(test_song_id)
print(f"Optimized kNN Recommendations for song {test_song_id}:")
print(knn_recommendations)

# 4. Implement Collaborative Filtering

print("\n4. Implementing Collaborative Filtering")

# For this example, we'll create a dummy user-item interaction matrix
# In a real scenario, this would come from actual user listening history
n_users = 1000
user_item_matrix = pd.DataFrame(np.random.randint(0, 5, size=(n_users, len(spotify_data))),
                                columns=spotify_data['id'])

# Convert the user-item matrix to a sparse matrix for efficiency
user_item_sparse = csr_matrix(user_item_matrix.values)

def collaborative_filtering_recommendation(user_id: int, n_recommendations: int = 5) -> List[str]:
    """
    Recommend songs using Collaborative Filtering with matrix factorization.

    Args:
    user_id (int): ID of the user to recommend songs for
    n_recommendations (int): Number of songs to recommend

    Returns:
    List[str]: List of recommended song IDs
    """
    # Perform matrix factorization using TruncatedSVD
    svd = TruncatedSVD(n_components=50, random_state=42)
    user_factors = svd.fit_transform(user_item_sparse)
    item_factors = svd.components_.T

    # Compute user-item similarities
    user_vector = user_factors[user_id]
    item_similarities = np.dot(item_factors, user_vector)
```

```python
    # Get top N recommendations
    top_n_indices = item_similarities.argsort()[::-1][:n_recommendations]
    recommended_song_ids = spotify_data['id'].iloc[top_n_indices].tolist()

    return recommended_song_ids

# Test collaborative filtering recommendation
test_user_id = 0
cf_recommendations = collaborative_filtering_recommendation(test_user_id)
print(f"\nCollaborative Filtering Recommendations for user {test_user_id}:")
print(cf_recommendations)

# 5. Advanced Collaborative Filtering using Surprise library

print("\n5. Advanced Collaborative Filtering using Surprise library")

# Prepare data for Surprise
reader = Reader(rating_scale=(0, 5))
data = Dataset.load_from_df(user_item_matrix.melt(id_vars=['user_id'], var_name='item_id',
value_name='rating'), reader)

# Use SVD algorithm
svd = SVD()

# Perform cross-validation
cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)

# Train the model on the entire dataset
trainset = data.build_full_trainset()
svd.fit(trainset)

def surprise_cf_recommendation(user_id: int, n_recommendations: int = 5) -> List[str]:
    """
    Recommend songs using Surprise's SVD algorithm.

    Args:
    user_id (int): ID of the user to recommend songs for
    n_recommendations (int): Number of songs to recommend

    Returns:
    List[str]: List of recommended song IDs
    """
    # Get all items the user hasn't interacted with
    user_items = user_item_matrix.loc[user_id]
```

```
    unrated_items = user_items[user_items == 0].index

    # Predict ratings for all unrated items
    predictions = [svd.predict(user_id, item_id).est for item_id in unrated_items]

    # Get top N recommendations
    top_n_indices = np.argsort(predictions)[::-1][:n_recommendations]
    recommended_song_ids = unrated_items[top_n_indices].tolist()

    return recommended_song_ids

# Test Surprise-based collaborative filtering recommendation
surprise_cf_recommendations = surprise_cf_recommendation(test_user_id)
print(f"\nSurprise-based Collaborative Filtering Recommendations for user {test_user_id}:")
print(surprise_cf_recommendations)

# 6. Evaluation Metrics

print("\n6. Evaluation Metrics")

def evaluate_recommendations(true_likes: List[str], recommendations: List[str]) -> Dict[str, float]:
    """
    Evaluate the recommendations using precision, recall, F1 score, and NDCG.

    Args:
    true_likes (List[str]): List of songs actually liked by the user
    recommendations (List[str]): List of recommended song IDs

    Returns:
    Dict[str, float]: Dictionary containing evaluation metrics
    """
    true_set = set(true_likes)
    rec_set = set(recommendations)

    true_positives = len(true_set.intersection(rec_set))
    precision = true_positives / len(rec_set) if len(rec_set) > 0 else 0
    recall = true_positives / len(true_set) if len(true_set) > 0 else 0
    f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0

    # Calculate NDCG
    relevance = [1 if item in true_set else 0 for item in recommendations]
    ideal_relevance = sorted(relevance, reverse=True)
    ndcg = ndcg_score([ideal_relevance], [relevance])
```

```
    return {
        'precision': precision,
        'recall': recall,
        'f1_score': f1,
        'ndcg': ndcg
    }

# Simulate true likes for evaluation
true_likes = spotify_data['id'].sample(10).tolist()

# Evaluate kNN recommendations
knn_metrics = evaluate_recommendations(true_likes, knn_recommendations)
print("\nkNN Evaluation:")
print(f"Precision: {knn_metrics['precision']:.2f}, Recall: {knn_metrics['recall']:.2f}, "
      f"F1 Score: {knn_metrics['f1_score']:.2f}, NDCG: {knn_metrics['ndcg']:.2f}")

# Evaluate Collaborative Filtering recommendations
cf_metrics = evaluate_recommendations(true_likes, cf_recommendations)
print("\nCollaborative Filtering Evaluation:")
print(f"Precision: {cf_metrics['precision']:.2f}, Recall: {cf_metrics['recall']:.2f}, "
      f"F1 Score: {cf_metrics['f1_score']:.2f}, NDCG: {cf_metrics['ndcg']:.2f}")

# Evaluate Surprise-based Collaborative Filtering recommendations
surprise_cf_metrics = evaluate_recommendations(true_likes, surprise_cf_recommendations)
print("\nSurprise-based Collaborative Filtering Evaluation:")
print(f"Precision: {surprise_cf_metrics['precision']:.2f}, Recall: {surprise_cf_metrics['recall']:.2f}, "
      f"F1 Score: {surprise_cf_metrics['f1_score']:.2f}, NDCG: {surprise_cf_metrics['ndcg']:.2f}")

# 7. Advanced Analysis: Clustering Songs

print("\n7. Advanced Analysis: Clustering Songs")

# Perform K-means clustering
n_clusters = 5
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
spotify_data['cluster'] = kmeans.fit_predict(spotify_data_normalized)

# Visualize clusters (using PCA for dimensionality reduction)
pca = PCA(n_components=2)
pca_result = pca.fit_transform(spotify_data_normalized)

plt.figure(figsize=(10, 8))
scatter = plt.scatter(pca_result[:, 0], pca_result[:, 1], c=spotify_data['cluster'], cmap='viridis')
plt.colorbar(scatter)
```

```
plt.title("Song Clusters Visualization")
plt.xlabel("First Principal Component")
plt.ylabel("Second Principal Component")
plt.show()

# Analyze cluster characteristics
cluster_means = spotify_data.groupby('cluster')[features].mean()
print("\nCluster Characteristics:")
print(cluster_means)

# 8. Hybrid Recommendation System

print("\n8. Hybrid Recommendation System")

def hybrid_recommendation(song_id: str, user_id: int, alpha: float = 0.5) -> List[str]:
    """
    Generate recommendations using a hybrid approach combining kNN and Collaborative
Filtering.

    Args:
    song_id (str): ID of the song to base content-based recommendations on
    user_id (int): ID of the user to base collaborative filtering recommendations on
    alpha (float): Weight for kNN recommendations (1-alpha for CF recommendations)

    Returns:
    List[str]: List of recommended song IDs
    """
    knn_recs = optimized_knn_recommendation(song_id)
    cf_recs = surprise_cf_recommendation(user_id)

    # Combine recommendations
    hybrid_recs = list(set(knn_recs + cf_recs))

    # Sort based on a weighted score
    def get_score(song):
        knn_score = 1 / (knn_recs.index(song) + 1) if song in knn_recs else 0
        cf_score = 1 / (cf_recs.index(song) + 1) if song in cf_recs else 0
        return alpha * knn_score + (1 - alpha) * cf_score

    hybrid_recs.sort(key=get_score, reverse=True)

    return hybrid_recs[:5]  # Return top 5 recommendations

# Test hybrid recommendation
```

```
hybrid_recommendations = hybrid_recommendation(test_song_id, test_user_id)
print("\nHybrid Recommendations:")
print(hybrid_recommendations)

# Evaluate hybrid recommendations
hybrid_metrics = evaluate_recommendations(true_likes, hybrid_recommendations)
print("\nHybrid Recommendation Evaluation:")
print(f"Precision: {hybrid_metrics['precision']:.2f}, Recall: {hybrid_metrics['recall']:.2f}, "
    f"F1 Score: {hybrid_metrics['f1_score']:.2f}, NDCG: {hybrid_metrics['ndcg']:.2f}")

# 9. Time-based Analysis

print("\n9. Time-based Analysis")

# Assuming we have a 'release_date' column in our dataset
spotify_data['release_date'] = pd.to_datetime(spotify_data['release_date'], errors='coerce')
spotify_data['release_year'] = spotify_data['release_date'].dt.year

# Analyze popularity trends over time
plt.figure(figsize=(12, 6))
spotify_data.groupby('release_year')['popularity'].mean().plot()
plt.title("Average Song Popularity by Release Year")
plt.xlabel("Release Year")
plt.ylabel("Average Popularity")
plt.show()

# 10. Genre Analysis

print("\n10. Genre Analysis")

# Assuming we have a 'genre' column in our dataset
genre_popularity =
spotify_data.groupby('genre')['popularity'].mean().sort_values(ascending=False)

plt.figure(figsize=(12, 6))
genre_popularity.head(10).plot(kind='bar')
plt.title("Top 10 Most Popular Genres")
plt.xlabel("Genre")
plt.ylabel("Average Popularity")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

# 11. Artist Collaboration Network
```

```
print("\n11. Artist Collaboration Network")

import networkx as nx

# Create a graph of artist collaborations
G = nx.Graph()

# Assuming we have 'artist' and 'featured_artists' columns
for _, row in spotify_data.iterrows():
    artists = [row['artist']] + row['featured_artists'].split(',') if pd.notna(row['featured_artists']) else
[row['artist']]
    for i in range(len(artists)):
        for j in range(i+1, len(artists)):
            G.add_edge(artists[i].strip(), artists[j].strip())

# Analyze the network
print(f"Number of artists: {G.number_of_nodes()}")
print(f"Number of collaborations: {G.number_of_edges()}")

# Find the most collaborative artists
top_collaborative_artists = sorted(G.degree, key=lambda x: x[1], reverse=True)[:10]
print("\nTop 10 Most Collaborative Artists:")
for artist, degree in top_collaborative_artists:
    print(f"{artist}: {degree} collaborations")

# Visualize a subgraph of the most collaborative artists
subgraph = G.subgraph([artist for artist, _ in top_collaborative_artists])
pos = nx.spring_layout(subgraph)
plt.figure(figsize=(12, 8))
nx.draw(subgraph, pos, with_labels=True, node_color='lightblue', node_size=1000, font_size=8,
font_weight='bold')
plt.title("Collaboration Network of Top 10 Most Collaborative Artists")
plt.axis('off')
plt.tight_layout()
plt.show()

# 12. Advanced Feature Engineering

print("\n12. Advanced Feature Engineering")

# Create new features
spotify_data['energy_danceability_ratio'] = spotify_data['energy'] / spotify_data['danceability']
spotify_data['valence_energy_interaction'] = spotify_data['valence'] * spotify_data['energy']
```

```
spotify_data['loudness_normalized'] = (spotify_data['loudness'] - spotify_data['loudness'].min()) /
(spotify_data['loudness'].max() - spotify_data['loudness'].min())

# Analyze the impact of new features on popularity
new_features = ['energy_danceability_ratio', 'valence_energy_interaction',
'loudness_normalized']
correlation_with_popularity = spotify_data[new_features +
['popularity']].corr()['popularity'].sort_values(ascending=False)

print("Correlation of new features with popularity:")
print(correlation_with_popularity)

# 13. Personalized Playlist Generation

print("\n13. Personalized Playlist Generation")

def generate_playlist(seed_songs: List[str], playlist_length: int = 20) -> List[str]:
    """
    Generate a personalized playlist based on seed songs.

    Args:
    seed_songs (List[str]): List of song IDs to base the playlist on
    playlist_length (int): Desired length of the playlist

    Returns:
    List[str]: List of song IDs for the generated playlist
    """
    playlist = set(seed_songs)
    while len(playlist) < playlist_length:
        for song in seed_songs:
            recommendations = optimized_knn_recommendation(song, k=3)
            playlist.update(recommendations)
            if len(playlist) >= playlist_length:
                break
    return list(playlist)[:playlist_length]

# Generate a playlist
seed_songs = spotify_data['id'].sample(3).tolist()
personalized_playlist = generate_playlist(seed_songs)

print("Personalized Playlist:")
print(personalized_playlist)

# 14. Model Persistence
```

```
print("\n14. Model Persistence")

# Save the trained models
joblib.dump(kmeans, 'kmeans_model.joblib')
joblib.dump(svd, 'svd_model.joblib')

print("Models saved successfully.")

# 15. Conclusion and Future Work

print("\n15. Conclusion and Future Work")

print("This notebook demonstrates the implementation of a comprehensive Music Recommendation System using various techniques:")
print("1. k-Nearest Neighbors for content-based recommendations")
print("2. Collaborative Filtering using matrix factorization")
print("3. Advanced Collaborative Filtering using the Surprise library")
print("4. Hybrid recommendation system combining multiple approaches")
print("5. Clustering analysis for song grouping")
print("6. Time-based and genre-based analyses")
print("7. Artist collaboration network analysis")
print("8. Advanced feature engineering")
print("9. Personalized playlist generation")

print("\nFuture work could include:")
print("1. Implementing more advanced algorithms like deep learning models (e.g., neural collaborative filtering)")
print("2. Incorporating natural language processing for lyric analysis")
print("3. Developing a real-time recommendation system with online learning capabilities")
print("4. Implementing context-aware recommendations (e.g., based on time of day, user mood)")
print("5. Conducting A/B testing to compare different recommendation strategies")
print("6. Integrating with external APIs (e.g., Spotify API) for real-world data and testing")
print("7. Developing a user interface for interactive recommendations and feedback")
print("8. Implementing privacy-preserving recommendation techniques")
print("9. Exploring multi-modal recommendations (combining audio features with visual and textual data)")
print("10. Investigating the impact of diversity and serendipity in recommendations")

# End of the notebook
print("\nEnd of the Music Recommendation System Jupyter Notebook")
```

This completes the Jupyter Notebook for the Music Recommendation System project. The notebook now includes:

1. Comprehensive data loading and preprocessing
2. Exploratory Data Analysis (EDA) with visualizations
3. Implementation of k-Nearest Neighbors with parameter optimization
4. Collaborative Filtering using matrix factorization
5. Advanced Collaborative Filtering using the Surprise library
6. Detailed evaluation metrics including NDCG
7. Clustering analysis of songs
8. Hybrid recommendation system
9. Time-based analysis of song popularity
10. Genre analysis
11. Artist collaboration network analysis
12. Advanced feature engineering
13. Personalized playlist generation
14. Model persistence for future use
15. Comprehensive conclusion and future work suggestions

This notebook provides a thorough implementation of the Music Recommendation System, incorporating various advanced techniques and analyses. It serves as a solid foundation for further development and experimentation in the field of music recommendation systems.