```
import pandas as pd
import numpy as np

song_df_1 = pd.read_csv('triplets_file.csv')
song_df_1.head()
```

	user_id	song_id	listen_count
0	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOAKIMP12A8C130995	1
1	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBBMDR12A8C13253B	2
2	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBXHDL12A81C204C0	1
3	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBYHAJ12A6701BF1D	1
4	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SODACBL12A8C13C273	1

```
song_df_2 = pd.read_csv('song_data.csv')
song_df_2.head()
```

	song_id	title	release	artist_name	year
0	SOQMMHC12AB0180CB8	Silent Night	Monster Ballads X-Mas	Faster Pussy cat	2003
1	SOVFVAK12A8C1350D9	Tanssi vaan	Karkuteillä	Karkkiautomaatti	1995
2	SOGTUKN12AB017F4F1	No One Could Ever	Butter	Hudson Mohawke	2006
3	SOBNYVR12A8C13558C	Si Vos Querés	De Culo	Yerba Brava	2003
4	SOHSBXH12A8C13B0DF	Tangle Of	Rene Ablaze Presents Winter	Der Mystic	0

```
song_df = pd.merge(song_df_1, song_df_2.drop_duplicates(['song_id']), on='song_id', how='left')
song_df.head()
```

	user_id	song_id	listen_count	titl
0	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOAKIMP12A8C130995	1	The Cov
1	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBBMDR12A8C13253B	2	Entre Do Agua
2	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBXHDL12A81C204C0	1	Stronge

```
print(len(song_df_1), len(song_df_2))
len(song_df)
```

```
2000000 1000000
2000000
```

```
# song_df['song'] = song_df['title']+' - '+song_df['artist_name']
song_df.head()
```

	user_id	song_id	listen_count	titl
0	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOAKIMP12A8C130995	1	The Cov
1	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBBMDR12A8C13253B	2	Entre Do Agua
2	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBXHDL12A81C204C0	1	Stronge

```
import numpy as np
import pandas as pd
from difflib import get_close_matches
```

```
class PopularityRecommender:

    def __init__(self):
      self.train_data = None
      self.user_id = None
      self.item_id = None
      self.popularity_recommendations = None

    def create(self, train_data, user_id, item_id):
        self.train_data = train_data
        self.user_id = user_id
        self.item_id = item_id

        train_data_grouped = train_data.groupby([self.item_id]).agg({self.user_id: 'count'}).reset_index()
        train_data_grouped.rename(columns={'user_id': 'score'}, inplace=True)
        train_data_sort = train_data_grouped.sort_values(['score', self.item_id], ascending=[0, 1])
        train_data_sort['Rank'] = train_data_sort['score'].rank(ascending=0, method='first')

        self.popularity_recommendations = train_data_sort.head(10)

    def recommend(self):
        return self.popularity_recommendations


class ItemSimilarityRecommender:
    def __init__(self):
        self.train_data = None
        self.user_id = None
        self.item_id = None
        self.cooccurence_matrix = None
        self.songs_dict = None
        self.rev_songs_dict = None
        self.item_similarity_recommendations = None

    # This method takes a user identifier and returns the items (songs) that the user has interacted with in the training data.  It looks for
    def get_user_items(self, user):

        user_data = self.train_data[self.train_data[self.user_id] == user]
        user_items = list(user_data[self.item_id].unique())
        return user_items

    #This method takes an item (song) identifier and returns the set of users who have interacted with that item in the training data
    def get_item_users(self, item):
        item_data = self.train_data[self.train_data[self.item_id] == item]
        item_users = set(item_data[self.user_id].unique())
        return item_users

    # This method returns a list of all unique item identifiers (songs) in the training data. It retrieves all the unique values from the ite
    def get_all_items_train_data(self):
        all_items = list(self.train_data[self.item_id].unique())
        return all_items


    #The method iterates over all items in all_songs and all items in user_songs, calculating the co-occurrence between each pair of items ar
    def construct_cooccurence_matrix(self, user_songs, all_songs):
        user_songs_users = []
        for i in range(len(user_songs)):
            user_songs_users.append(self.get_item_users(user_songs[i]))

        cooccurence_matrix = np.matrix(np.zeros(shape=(len(user_songs), len(all_songs))), float)

        for i in range(len(all_songs)):
            songs_i_data = self.train_data[self.train_data[self.item_id] == all_songs[i]]
            users_i = set(songs_i_data[self.user_id].unique())

            for j in range(len(user_songs)):
                users_j = user_songs_users[j]
                users_intersection = users_i.intersection(users_j)

                if len(users_intersection) != 0:
                    users_union = users_i.union(users_j)
                    cooccurence_matrix[j, i] = float(len(users_intersection)) / float(len(users_union))
                else:
                    cooccurence_matrix[j, i] = 0

        return cooccurence_matrix

    #This method takes a user, the co-occurrence matrix, all unique items, and user-interacted items to generate the top 10 song recommendati
```

```
        #This method takes a user, the co-occurrence matrix, all unique items, and user-interacted items to generate the top 10 song recommendati
        def generate_top_recommendations(self, user, cooccurence_matrix, all_songs, user_songs):
            user_sim_scores = cooccurence_matrix.sum(axis=0) / float(cooccurence_matrix.shape[0])
            user_sim_scores = np.array(user_sim_scores)[0].tolist()

            sort_index = sorted(((e, i) for i, e in enumerate(list(user_sim_scores))), reverse=True)
            columns = ['user_id', 'song', 'score', 'rank']
            df = pd.DataFrame(columns=columns)

            rank = 1
            for i in range(len(sort_index)):
                if ~np.isnan(sort_index[i][0]) and all_songs[sort_index[i][1]] not in user_songs and rank <= 10:
                    df.loc[len(df)] = [user, all_songs[sort_index[i][1]], sort_index[i][0], rank]
                    rank += 1

            if df.shape[0] == 0:
                print("The current user has no songs for training the item similarity based recommendation model.")
                return -1
            else:
                return df

        def create(self, train_data, user_id, item_id):
            self.train_data = train_data
            self.user_id = user_id
            self.item_id = item_id

        def recommend(self, user):
            user_songs = self.get_user_items(user)
            all_songs = self.get_all_items_train_data()
            cooccurence_matrix = self.construct_cooccurence_matrix(user_songs, all_songs)
            df_recommendations = self.generate_top_recommendations(user, cooccurence_matrix, all_songs, user_songs)
            return df_recommendations

        def get_artist_songs(self, artist_name):

            artist_songs_data = self.train_data[self.train_data['artist_name'] == artist_name]
            artist_songs = list(artist_songs_data['title'].unique())
            return artist_songs

        def recommend(self, user):
            user_songs = self.get_user_items(user)
            all_songs = self.get_all_items_train_data()
            cooccurence_matrix = self.construct_cooccurence_matrix(user_songs, all_songs)
            df_recommendations = self.generate_top_recommendations(user, cooccurence_matrix, all_songs, user_songs)
            return df_recommendations

    def get_dataset():

        return song_df

    def main():
        dataset = get_dataset()

        # Create and train popularity-based recommender
        pop_recommender = PopularityRecommender()
        pop_recommender.create(dataset, 'user_id', 'title')

        # Create and train item similarity-based recommender
        item_recommender = ItemSimilarityRecommender()
        item_recommender.create(dataset, 'user_id', 'title')

        # Interactive user input
        while True:
            user_input = input("Enter a song name or 'popular' for popular songs (or 'quit' to exit): ")
            if user_input.lower() == 'quit':
                break

            if user_input.lower() == 'popular':
                # Get popular recommendations
                popular_recommendations = pop_recommender.recommend()
                print("Top 10 Popular Songs:")
                print(popular_recommendations)
            else:
                # Get similar song recommendations or artist songs
                artist_name_data = dataset[dataset['title'] == user_input]['artist_name']
                if not artist_name_data.empty:
                    artist_name = artist_name_data.values[0]
                    artist_songs = item_recommender.get_artist_songs(artist_name)
```

```
                print("Songs by Artist '{}':".format(artist_name))
                print(artist_songs)
            else:
                print("Song not found in the dataset.")

if __name__ == "__main__":
    main()
```

```
Enter a song name or 'popular' for popular songs (or 'quit' to exit): popular
Top 10 Popular Songs:
                                                 title  score  Rank
6836                                     Sehr kosmisch   8277   1.0
8725                                              Undo   7032   2.0
1964                     Dog Days Are Over (Radio Edit)   6949   3.0
9496                                   You're The One   6729   4.0
6498                                           Revelry   6145   5.0
6825                                           Secrets   5841   6.0
3437  Horn Concerto No. 4 in E flat K495: II. Romanc...   5385   7.0
2595                                         Fireflies   4795   8.0
3322                                 Hey_ Soul Sister   4758   9.0
8494                                          Tive Sim   4548  10.0
Enter a song name or 'popular' for popular songs (or 'quit' to exit): Revelty
Song not found in the dataset.
Enter a song name or 'popular' for popular songs (or 'quit' to exit): Revelry
Songs by Artist 'Kings Of Leon':
['Trani', 'Revelry', 'Use Somebody', 'Camaro', 'Manhattan', "Joe's Head", 'Ragoo', 'Trunk', 'I Want You', 'Arizona', 'McFearless', 'The
Enter a song name or 'popular' for popular songs (or 'quit' to exit): quit
```