In this problem, you will implement your own version of a collaborative filtering recommender system that will use user-based CF strategy to predict users' ratings on items. You will use a modified version of a joke ratings dataset Download joke ratings dataset. This is a modified subset of the data from the Jester Online Joke Recommender System.

The data set contains two files. The file "modified_jester_data.csv" contains the ratings on 100 jokes by 1000 users (each row is a user profile). The ratings have been normalized to be between 1 and 21 (a 20-point scale), with 1 being the lowest rating. A zero indicates a missing rating. The file "jokes.csv" contains the joke ids mapped to the actual text of the jokes.

Given: a user-item ratings matrix with rows corresponding to user records (ratings given by a user to various items), and columns corresponding to all items in the database. This is the training data. The target user u_t may be a new user or an existing user and is represented as a vector of ratings over all items (a row in the ratings matrix).

The task is to generate a list of jokes for a user based on similar user profiles ("Users who are similar to you also liked these jokes …").

Limit your response to 1-2 pages. [5 points]

```
import pyterrier as pt
if not pt.started():
  pt.init()

# User-based collaborative filtering
import pandas as pd
import numpy as np
```

```
C:\Users\Fortu\AppData\Local\Temp\ipykernel_22032\1874076957.py:2: DeprecationWarning: Call to
deprecated function (or staticmethod) started. (use pt.java.started() instead) -- Deprecated
since version 0.11.0.
  if not pt.started():
```

```
jokes = pd.read_csv('./Assignment9_data/Assignment7_data/jokes.csv')
jokes.head()
```

	jokeid	joke
0	1	A man visits the doctor. The doctor says "I ha...
1	2	This couple had an excellent relationship goin...
2	3	Q. What's 200 feet long and has 4 teeth? A. Th...
3	4	Q. What's the difference between a man and a t...
4	5	Q. What's O. J. Simpson's Internet address? A....

```
jokes.shape
```

```
(100, 2)
```

```
ratings = pd.read_csv('./Assignment9_data/Assignment7_data/ratings.csv')
ratings.head()
```

	jokeid	userid	rating
0	1	1	3.18
1	1	2	15.08
2	1	3	0.00
3	1	4	0.00
4	1	5	19.50

```
ratings.shape
```

```
(100000, 3)
```

```
# data normalization
from sklearn.preprocessing import MinMaxScaler
```

```
col = 'rating'
ratings[col] = MinMaxScaler().fit_transform(ratings[col].values.reshape(-1, 1))
```

```
# Create user matrix: contains relationships among users and items
user_matrix = pd.pivot_table(ratings, values='rating', index='userid', columns='jokeid')
```

```
user_matrix.head()
```

jokeid	1	2	3	4	5	6	7	8	9	10	...	91
userid												
1	0.152153	0.946890	0.064115	0.135885	0.166507	0.119617	0.055024	0.725837	0.096651	0.298565	...	0.66124
2	0.721531	0.512440	0.830622	0.735407	0.412440	0.064115	0.491388	0.270813	0.951196	0.967464	...	0.66124
3	0.000000	0.000000	0.000000	0.000000	0.958373	0.969856	0.958373	0.969856	0.000000	0.000000	...	0.00000
4	0.000000	0.925837	0.000000	0.000000	0.612440	0.916746	0.391388	0.823445	0.000000	0.614354	...	0.00000
5	0.933014	0.746890	0.326794	0.268421	0.591388	0.602871	0.863158	0.746890	0.505263	0.800478	...	0.77464

5 rows × 100 columns

```
user_matrix = user_matrix.fillna(0)
```

```
user_matrix.head()
```

jokeid	1	2	3	4	5	6	7	8	9	10	...	91
userid												
1	0.152153	0.946890	0.064115	0.135885	0.166507	0.119617	0.055024	0.725837	0.096651	0.298565	...	0.66124
2	0.721531	0.512440	0.830622	0.735407	0.412440	0.064115	0.491388	0.270813	0.951196	0.967464	...	0.66124
3	0.000000	0.000000	0.000000	0.000000	0.958373	0.969856	0.958373	0.969856	0.000000	0.000000	...	0.00000
4	0.000000	0.925837	0.000000	0.000000	0.612440	0.916746	0.391388	0.823445	0.000000	0.614354	...	0.00000
5	0.933014	0.746890	0.326794	0.268421	0.591388	0.602871	0.863158	0.746890	0.505263	0.800478	...	0.77464

5 rows × 100 columns

```
user_matrix.shape
```

```
(1000, 100)
```

```
from sklearn.metrics.pairwise import cosine_similarity
import operator
```

```
def similar_users(user_id, matrix, k=1):
  """
  Find the most similar users to a given user based on cosine similarity.
  Parameters:
  user_id (int or str): The ID of the user for whom to find similar users.
  matrix (pd.DataFrame): A pandas DataFrame where rows represent users and columns represent featu
  k (int, optional): The number of similar users to return. Default is 1.
  Returns:
  list: A list of user IDs that are most similar to the given user.
  """
  # Create a df of just the current user
  user = matrix[matrix.index == user_id]
  # And one of all other users
  other_users = matrix[matrix.index != user_id]
  # calculate cosine similarity between user and each other user one by one
  similarities = cosine_similarity(user, other_users)[0].tolist()
  # Create list of indices of these users to get the data quickly
  indices = other_users.index.tolist()
  # Create key/values pairs of user index and their similarity
  index_similarity = dict(zip(indices, similarities))
  # Sort by similarity
```

```
  index_similarity_sorted = sorted(index_similarity.items(), key=operator.itemgetter(1))
  index_similarity_sorted.reverse()
  # Get the top k highest similarities
  top_users_similarities = index_similarity_sorted[:k]
  simusers = [u[0] for u in top_users_similarities]

  return simusers
```

```
# This is what we want to do
bc = 1
sm = similar_users(bc, user_matrix, k=5)
print(sm)
```

```
[245, 983, 471, 158, 588]
```

```
# Create a function to recommend jokes to a user
def recommend_items(userid, similar_user_indices, matrix, items):
  # Load vector for the similar users
  similar_users = matrix[matrix.index.isin(similar_user_indices)]
  # Calculate the mean rating for each joke
  similar_users = similar_users.mean(axis=0)
  # Convert to dataframe so it's easy to sort and filter
  similar_users_df = pd.DataFrame(similar_users, columns=['mean'])
  # Load vector for the current user
  user = matrix[matrix.index == userid]
  # Transpose it so it's easier to filter
  jokes_rated = user.transpose()
  jokes_rated.columns = ['rating']
  # Select jokes that the user has not rated
  jokes_rated = jokes_rated[jokes_rated['rating'] == 0]
  # Generate a list of jokes the user has not rated
  jokes_nl = jokes_rated.index.tolist()
  # Score those jokes by filtering the similar users vector to just the jokes the current user has
  scores = similar_users_df[similar_users_df.index.isin(jokes_nl)]
  # Sort the scores and return the top items
  recommendations = scores.sort_values('mean', ascending=False)
  top_jokes = recommendations.head(items)
  top_n_jokes_indices = recommendations.index.tolist()
  joke_information = jokes[jokes['jokeid'].isin(top_n_jokes_indices)]

  return top_jokes, joke_information
```

```
# This is what we want to do
x, y = recommend_items(1,sm,user_matrix,10)

x.join(y.set_index('jokeid'))
```

jokeid	mean	joke
84	0.148421	Q: What is the difference between Mechanical E...
88	0.141914	A Czechoslovakian man felt his eyesight was gr...
77	0.089952	If pro- is the opposite of con- then congress ...
74	0.075024	Q: How many stalkers does it take to change a ...
80	0.075024	Hillary Bill Clinton and the Pope are sitting ...
73	0.072727	Q: What is the difference between George Wash...
76	0.000000	There once was a man and a woman that both go...
75	0.000000	Q: Do you know the difference between an intel...
72	0.000000	On the first day of college the Dean addressed...
71	0.000000	At a recent Sacramento PC Users Group meeting ...

A website has collected ratings from 25 past users (user1-user25) on a selection of recent games. The ratings range from 1 = worst to 5 = best. Two new users (NU1 and NU2) have recently visited the site and rated some of the games(" " represents missing ratings). The two new users' ratings are given in the last two rows.

```
# Define the data
data = {
  'User': ['user1', 'user2', 'user3', 'user4', 'user5', 'user6', 'user7', 'user8', 'user9', 'user
  'Minecraft': [1, 5, 3, np.nan, 2, 5, 1, 2, np.nan, 3, np.nan, 4, np.nan, np.nan, np.nan, np.nan
  'Fortnite': [5, 4, np.nan, 3, 4, np.nan, 4, 1, np.nan, 5, np.nan, 4, np.nan, 5, 2, 3, 5, np.nan
  'Animal Crossing': [np.nan, np.nan, 1, np.nan, 3, np.nan, 5, np.nan, 3, 1, 2, np.nan, 2, 3, np.
  'Call of Duty: Modern Warfare': [3, np.nan, 2, np.nan, 3, 5, np.nan, 2, np.nan, 1, 2, np.nan, 3
  'Madden NFL 20': [np.nan, 3, 2, 4, np.nan, 1, 2, 4, 2, np.nan, np.nan, np.nan, 4, 2, 3, 1, np.na
  'Mortal Kombat 11': [np.nan, 2, 2, 1, np.nan, 5, np.nan, np.nan, 2, 1, np.nan, 3, np.nan, 4, 3,
  'Super Smash Bros': [3, 1, 2, np.nan, 3, np.nan, 1, np.nan, 4, np.nan, 1, 4, 1, np.nan, 4, np.na
  'Kingdom Hearts III': [5, np.nan, 5, 3, np.nan, 1, 4, np.nan, 3, 5, 4, 2, 4, 4, 5, 1, 5, 5, 5, 4
}

# Create the DataFrame
df = pd.DataFrame(data)

df
```

	User	Minecraft	Fortnite	Animal Crossing	Call of Duty: Modern Warfare	Madden NFL 20	Mortal Kombat 11	Super Smash Bros	Kingdom Hearts III
0	user1	1.0	5.0	NaN	3.0	NaN	NaN	3.0	5.0
1	user2	5.0	4.0	NaN	NaN	3.0	2.0	1.0	NaN

	User	Minecraft	Fortnite	Animal Crossing	Call of Duty: Modern Warfare	Madden NFL 20	Mortal Kombat 11	Super Smash Bros	Kingdom Hearts III
2	user3	3.0	NaN	1.0	2.0	2.0	2.0	2.0	5.0
3	user4	NaN	3.0	NaN	NaN	4.0	1.0	NaN	3.0
4	user5	2.0	4.0	3.0	3.0	NaN	NaN	3.0	NaN
5	user6	5.0	NaN	NaN	5.0	1.0	5.0	NaN	1.0
6	user7	1.0	4.0	5.0	NaN	2.0	NaN	1.0	4.0
7	user8	2.0	1.0	NaN	2.0	4.0	NaN	NaN	NaN
8	user9	NaN	NaN	3.0	NaN	2.0	2.0	4.0	3.0
9	user10	3.0	5.0	1.0	1.0	NaN	1.0	NaN	5.0
10	user11	NaN	NaN	2.0	2.0	NaN	NaN	1.0	4.0
11	user12	4.0	4.0	NaN	NaN	NaN	3.0	4.0	2.0
12	user13	NaN	NaN	2.0	3.0	4.0	NaN	1.0	4.0
13	user14	NaN	5.0	3.0	NaN	2.0	4.0	NaN	4.0
14	user15	NaN	2.0	NaN	1.0	3.0	3.0	4.0	5.0
15	user16	NaN	3.0	2.0	2.0	1.0	5.0	NaN	1.0
16	user17	1.0	5.0	1.0	2.0	NaN	NaN	4.0	5.0
17	user18	5.0	NaN	4.0	NaN	4.0	3.0	3.0	5.0
18	user19	NaN	4.0	NaN	2.0	NaN	5.0	1.0	5.0
19	user20	2.0	5.0	1.0	1.0	5.0	3.0	NaN	4.0
20	user21	NaN	3.0	NaN	4.0	NaN	NaN	2.0	4.0
21	user22	5.0	NaN	NaN	2.0	NaN	2.0	NaN	1.0
22	user23	4.0	NaN	3.0	NaN	4.0	1.0	1.0	NaN
23	user24	1.0	3.0	2.0	1.0	NaN	NaN	NaN	5.0
24	user25	NaN	NaN	3.0	5.0	NaN	1.0	NaN	1.0
25	NU1	3.0	NaN	5.0	4.0	2.0	3.0	NaN	5.0
26	NU2	NaN	5.0	2.0	2.0	4.0	NaN	1.0	3.0

Using the item-based CF predict the ratings of these new users for each of the games they have not yet rated. You can use any similarity measures discussed in the class.

Be sure to show the intermediate steps in your work (or provide a short explanation of how you computed the predictions). Limit your response to 1-2 pages. [5 points]

Submit a single PDF with both these parts.

```
# Create user DataFrame
user_df = df[['User']].copy()
user_df['UserID'] = user_df.index + 1
```

```
# Create game DataFrame
game_df = pd.DataFrame({
  'Game': df.columns[1:],
  'GameID': range(1, len(df.columns[1:]) + 1)
})

# Create ratings DataFrame
ratings_df = df.melt(id_vars=['User'], var_name='Game', value_name='Rating')
ratings_df = ratings_df.merge(user_df, on='User').merge(game_df, on='Game')
ratings_df = ratings_df.drop(columns=['User', 'Game'])
```

```
# Display the DataFrames
print("User DataFrame:")
user_df
```

```
User DataFrame:
```

	User	UserID
0	user1	1
1	user2	2
2	user3	3
3	user4	4
4	user5	5
5	user6	6
6	user7	7
7	user8	8
8	user9	9
9	user10	10
10	user11	11
11	user12	12
12	user13	13
13	user14	14
14	user15	15
15	user16	16
16	user17	17
17	user18	18
18	user19	19
19	user20	20
20	user21	21

	User	UserID
21	user22	22
22	user23	23
23	user24	24
24	user25	25
25	NU1	26
26	NU2	27

```
print("\nGame DataFrame:")
game_df
```

```
Game DataFrame:
```

	Game	GameID
0	Minecraft	1
1	Fortnite	2
2	Animal Crossing	3
3	Call of Duty: Modern Warfare	4
4	Madden NFL 20	5
5	Mortal Kombat 11	6
6	Super Smash Bros	7
7	Kingdom Hearts III	8

```
print("\nRatings DataFrame:")
ratings_df
```

```
Ratings DataFrame:
```

	Rating	UserID	GameID
0	1.0	1	1
1	5.0	2	1
2	3.0	3	1
3	NaN	4	1
4	2.0	5	1
...	...	...	...
211	NaN	23	8

	Rating	UserID	GameID
212	5.0	24	8
213	1.0	25	8
214	5.0	26	8
215	3.0	27	8

216 rows × 3 columns

```
# Create pivot table for ratings
ratings_pivot = ratings_df.pivot_table(index='GameID', columns='UserID', values='Rating')

# Display the pivot table
ratings_pivot.head()
```

UserID	1	2	3	4	5	6	7	8	9	10	...	18	19	20	21	22	23	24	25	26
GameID																				
1	1.0	5.0	3.0	NaN	2.0	5.0	1.0	2.0	NaN	3.0	...	5.0	NaN	2.0	NaN	5.0	4.0	1.0	NaN	3.0
2	5.0	4.0	NaN	3.0	4.0	NaN	4.0	1.0	NaN	5.0	...	NaN	4.0	5.0	3.0	NaN	NaN	3.0	NaN	NaN
3	NaN	NaN	1.0	NaN	3.0	NaN	5.0	NaN	3.0	1.0	...	4.0	NaN	1.0	NaN	NaN	3.0	2.0	3.0	5.0
4	3.0	NaN	2.0	NaN	3.0	5.0	NaN	2.0	NaN	1.0	...	NaN	2.0	1.0	4.0	2.0	NaN	1.0	5.0	4.0
5	NaN	3.0	2.0	4.0	NaN	1.0	2.0	4.0	2.0	NaN	...	4.0	NaN	5.0	NaN	NaN	4.0	NaN	NaN	2.0

5 rows × 27 columns

```
ratings_pivot.shape
```

```
(8, 27)
```

```
ratings_pivot = ratings_pivot.fillna(0)
ratings_pivot.head()
```

UserID	1	2	3	4	5	6	7	8	9	10	...	18	19	20	21	22	23	24	25	26	27
GameID																					
1	1.0	5.0	3.0	0.0	2.0	5.0	1.0	2.0	0.0	3.0	...	5.0	0.0	2.0	0.0	5.0	4.0	1.0	0.0	3.0	0.0
2	5.0	4.0	0.0	3.0	4.0	0.0	4.0	1.0	0.0	5.0	...	0.0	4.0	5.0	3.0	0.0	0.0	3.0	0.0	0.0	5.0
3	0.0	0.0	1.0	0.0	3.0	0.0	5.0	0.0	3.0	1.0	...	4.0	0.0	1.0	0.0	0.0	3.0	2.0	3.0	5.0	2.0
4	3.0	0.0	2.0	0.0	3.0	5.0	0.0	2.0	0.0	1.0	...	0.0	2.0	1.0	4.0	2.0	0.0	1.0	5.0	4.0	2.0
5	0.0	3.0	2.0	4.0	0.0	1.0	2.0	4.0	2.0	0.0	...	4.0	0.0	5.0	0.0	0.0	4.0	0.0	0.0	2.0	4.0

5 rows × 27 columns

```
# Normalize the rating by adjusting it around the mean for a user.
# axis=1 means the operation should be performed on the columns.
ratings_norm = ratings_pivot.apply(lambda x: x - np.nanmean(x), axis=1)
ratings_norm.head()
```

UserID	1	2	3	4	5	6	7	8	9	10
GameID										
1	-0.740741	3.259259	1.259259	-1.740741	0.259259	3.259259	-0.740741	0.259259	-1.740741	1.259259
2	2.592593	1.592593	-2.407407	0.592593	1.592593	-2.407407	1.592593	-1.407407	-2.407407	2.592593
3	-1.592593	-1.592593	-0.592593	-1.592593	1.407407	-1.592593	3.407407	-1.592593	1.407407	-0.592593
4	1.259259	-1.740741	0.259259	-1.740741	1.259259	3.259259	-1.740741	0.259259	-1.740741	-0.740741
5	-1.740741	1.259259	0.259259	2.259259	-1.740741	-0.740741	0.259259	2.259259	0.259259	-1.740741

5 rows × 27 columns

```
item_sim_df = pd.DataFrame(cosine_similarity(ratings_norm,ratings_norm),index=ratings_norm.index,
```

```
item_sim_df.head()
```

GameID	1	2	3	4	5	6	7	8
GameID								
1	1.000000	-0.243442	-0.074327	-0.101845	0.072004	0.234278	-0.036273	-0.332424
2	-0.243442	1.000000	-0.223634	-0.272126	-0.138072	-0.071492	0.089981	0.199555
3	-0.074327	-0.223634	1.000000	-0.074839	0.139828	-0.087104	-0.069765	0.103614
4	-0.101845	-0.272126	-0.074839	1.000000	-0.379093	-0.043893	-0.192719	-0.080452
5	0.072004	-0.138072	0.139828	-0.379093	1.000000	0.024564	-0.142640	-0.083416

```
item_sim_df.shape
```

```
(8, 8)
```

```
# Given a game, find the similar games
def get_similar_games(gameid):
  if gameid not in ratings_norm.index:
    print(f"{gameid} item not available in ratings_norm.index")
```

```
    return None, None
  else:
    sim_games = item_sim_df.sort_values(by=gameid, ascending=False).index[1:]
    sim_scores = item_sim_df.sort_values(by=gameid, ascending=False).loc[:, gameid].tolist()[1:]
    return sim_games, sim_scores
```

```
# What we would like to do
games, scores = get_similar_games(8)
```

```
for x, y in zip(games, scores):
  print("{} with similarity of {}".format(x,y))
```

```
7 with similarity of 0.21592724477570088
2 with similarity of 0.1995550367869258
3 with similarity of 0.1036142251373073
6 with similarity of -0.024626278791953508
4 with similarity of -0.08045246286742387
5 with similarity of -0.0834161873957046
1 with similarity of -0.33242358705791364
```

```
# Join user_df with ratings_pivot to include UserID and User columns
ratings_pivot_with_user = ratings_pivot.T.join(user_df.set_index('UserID')).T

# Define the new users to predict for
new_users = [26, 27]

# Define the predictions dictionary to store predictions for each user
for user in new_users:
    # Fetch the ratings for the current user
    user_ratings = ratings_pivot[user]  # This is the user ratings from the ratings pivot table

    # Find the games that the user has not rated yet
    unrated_games = user_ratings[user_ratings == 0].index
    predictions = {}

    # Iterate through each unrated game
    for game in unrated_games:
        # Get similar games for the current game
        similar_games, similar_scores = get_similar_games(game)

        # Identify the games that the user has rated
        user_rated_games = user_ratings[user_ratings > 0].index

        # Filter the similarity scores to include only those for the games the user has rated
        sim_scores = [score for gm, score in zip(similar_games, similar_scores) if gm in user_rat
        rated_scores = user_ratings[user_rated_games].values

        # If the user has rated similar games, compute the predicted rating
        if sum(sim_scores) != 0:
```

```
            predicted_rating = np.dot(sim_scores, rated_scores) / sum(sim_scores)
            predictions[game] = predicted_rating

    # Sort predictions by predicted ratings in descending order
    sorted_predictions = sorted(predictions.items(), key=lambda x: x[1], reverse=True)

    # Output the predicted ratings for the current user
    user_name = user_df[user_df['UserID'] == user]['User'].values[0]
    print(f"Predicted ratings for {user_name}:")
    for game, rating in sorted_predictions:
        game_name = game_df[game_df['GameID'] == game]['Game'].values[0]
        print(f"{game_name}: {rating:.2f}")
    print()
```

```
Predicted ratings for NU1:
Super Smash Bros: 4.15
Fortnite: 3.80

Predicted ratings for NU2:
Mortal Kombat 11: 2.33
Minecraft: 2.11
```