Nouran_Khattab_A8.ipynb used functions.

1- get_similar_movies(movie_id)

```
# Function to get the top 10 similar movies
def get_similar_movies(movie_id: int) -> Union[pd.DataFrame, str]:
    # Check if the movie ID exists in the similarity matrixIf it doesn't, return
    a message indicating that the movie ID was not found.
    if movie_id in similarity_df.index:
        # If the movie ID exists, get the most similar movies from the
        similarity matrix by sorting the movies in the similarity matrix in
        descending order of similarity and selecting the top 10 movies
        # Starting from index 1 instead of 0 to exclude the movie itself (since a
        movie is always most similar to itself)
        similar_movies =
        similarity_df[movie_id].sort_values(ascending=False)[1:11]
        # Return the details of the similar movies by selecting the rows in the
        movies DataFrame that match the IDs of the similar movies
        return movies[movies['movieId'].isin(similar_movies.index)]
    else:
        # If the movie ID does not exist in the similarity matrix, return a
        message indicating that the movie ID was not found.
        return 'Movie ID not found in the similarity matrix.'
```

Where:

movie_id -> The ID of target movie to find most similar movies to it.

similarity_df -> is a dataframe, it rows and columns are the movies and its cells values are the cosine similarity between the two movies represented by the cell's row and column.

This function sorts all the movies in the row of the target movie descendingly, this creates a pandas series, its indices is the indices of the most similar movies and the values are the cosine similarity values that shows to what degree the the most similar movies are similar to the target movie.

This is the similar_movies series of "Toy Story (1995)"

```
804     0.558823
316     0.557919
661     0.554750
500     0.533452
423     0.533289
296     0.533147
1270    0.525359
2761    0.519465
1206    0.513769
1213    0.510691
```

It then gets the movies details from all movies dataframe to return more understandable results.

2- recommend_movies(user_id)

```python
# Function to recommend 3 movies for a user
def recommend_movies(user_id):
    # Make sure that user_id is in the training data
    if user_id not in  user_movie_matrix.index:
        return 'User ID not found in the data.'
    # First, let's get the ratings of the user to the movies that they have
     watched before
    # This is a pandas series where, as we have seen before, the index is
    movieId and the value is the rating of this user to the movies used in
    the training, some of them the user didn't watch before
    user_ratings = user_movie_matrix.loc[user_id]
    # To dentify the movies that the user has already watched, we select the
    movies that have a rating greater than 0.
    watched_movies = user_ratings[user_ratings > 0].index

    # Identify the movies that the user hasn't watched yet and we know other
    user's reaction to these movies, so we choose them out of
    user_movie_matrix not from all movies
    unwatched_movies = [movie for movie in user_movie_matrix.columns if movie
    not in watched_movies]



    # Initialize a dictionary to store the estimated ratings for unwatched
    movies, the keys will be the movieIds and the values will be the
    estimated ratings
```

```
    estimated_ratings = {}

    # Loop over each unwatched movieFor each movie, we calculate an estimated
    rating based on the ratings of similar movies that the user has watched
    for movie in unwatched_movies:
        # Get the top 10 movies that are most similar to the current movie by
        sorting the movies in the similarity matrix in descending order and
        selecting the top 10
        similar_movies =
        similarity_df[movie].sort_values(ascending=False)[1:20] # Changing
        the number of top most similar movies affects the results, using most
        similar ones seems to be a better idea but this will limit the search
        scope for the next step

        # Filter the similar movies to include only those that the user has
        watched by checking if the movieId of the similar movie is in the
        watched_movies list
        similar_movies_watched =
        similar_movies[similar_movies.index.isin(watched_movies)]

        # If the sum of similarities is zero (no similar movies have been
        watched by the user), skip this movieThis is to avoid division by
        zero in the next step
        if similar_movies_watched.sum() == 0:
            continue

        # Estimate the user's rating for the current movie by taking a
        weighted average of the user's ratings for the similar movies,
        # where the weights are the similarities between the current movie
        and the similar moviesThis is the key step in collaborative filtering
        estimated_ratings[movie] =
        (user_ratings[similar_movies_watched.index] *
        similar_movies_watched).sum() / similar_movies_watched.sum()

    # Get the 3 unwatched movies with the highest estimated ratings by
    converting the estimated_ratings dictionary to a pandas Series, to be
    used in rows selection,
    # then sorting it in descending order, and selecting the top 3 movies
    recommended_movies =
    pd.Series(estimated_ratings).sort_values(ascending=False)[:3].index


    # Return the details of the recommended movies by selecting the rows in
    the movies DataFrame that match the recommended movieIds
    return movies[movies['movieId'].isin(recommended_movies)]
```

Where:

user_id -> The ID of the target user to recommend movies for them.

user_movie_matrix -> the matrix that contains users IDs as indices/rows and the 200 movies used for training as columns, the ratings of the user to all the movies as cells' values. It's zero filled for non-watched movies.

This function recommends movies based on:

1- most similar watched movies to the movies that the user did not watch.
2- The ratings of this user to the previous list of moves.

For each movie in the unwatched movies, we get an estimated rating to the movie by doing weighted average to the ratings in point 2. Then sorting all the estimated ratings and getting the top 3.

Notes and other trials:

- We can make sure that the selected users and movies are population representative somehow, my ideas include getting an equal number of movies in each genre (they are 19, the 20^{th} is 'not spicified') or getting top movies that are rated by lots of users, but they represent the trend more than the population, choosing some of top movies and some of the non-trendy ones may solve this issue.

- Zero filling the non-watched movies may indicate that the users hate that movie, while it's not necessarily true. Searching led to using SVDs (Funk SVD or SVD++) to learn latent preferences of users and the latent attributes of items.