RS-MINIPROJECT2

Group 1

ZWE MIN MAW	6238135
SAW ZWE WAI YAN	6318013
THANARIT KANJANAMATAWAT	6410322

CONTENTS

PART 1	PART 2
USER-BASED WITH COSINE SIMILARITY	ITEM-BASES WITH ADJUSTED COSINE MEASURE

PART1

1.1 : Read train set file

```
rating_trainset = pd.read_csv("/kaggle/input/miniproject2/rating_trainset.csv")
rating_trainset.head()
```

1.2 : Read test set file

```
rating_testset = pd.read_csv("/kaggle/input/miniproject2/rating_testset.csv")
rating_testset.head()
```

RATING_TRAINSET

	userID	placeID	rating
0	U1001	132825	2
1	U1001	132830	1
2	U1001	135025	2
3	U1001	135033	1
4	U1001	135039	1

RATING_TESTSET

	userID	PlaceID	Rating
0	U1003	132825	2
1	U1003	135079	2
2	U1006	132825	1
3	U1006	135079	1
4	U1009	132834	2

2 : Create pivoted data for train dataset

+ Code + Markdown

```python
# Read the data into a pandas dataframe
data = rating_trainset
data.columns = ['user', 'place', 'rating']

# Pivot the data to transform it into a table with unique users and their ratings for each place
pivoted_data = data.pivot(index='user', columns='place', values='rating')

# Rename the columns to include 'Place' before the place id
pivoted_data.columns = ['Place ' + str(col) for col in pivoted_data.columns]

# Save the pivoted data to a new csv file
# pivoted_data.to_csv('pivoted_data.csv')

# Replace Nan Values with 0
# pivoted_data.fillna(0, inplace=True)
pivoted_data
```

PIVOTED_DATA

	Place 132560	Place 132561	Place 132564	Place 132572	Place 132583	Place 132584	Place 132594	Place 132608	Place 132609	Place 132613	...	Place 135080	Place 135081	Place 135082	Place 135085	Place 135086
user																
U1001	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	0.0	NaN
U1002	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	1.0	NaN
U1003	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2.0	NaN	NaN	NaN	NaN
U1004	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
U1005	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
U1134	NaN	NaN	NaN	0.0	NaN	NaN	NaN	NaN	NaN	NaN	...	1.0	NaN	NaN	2.0	NaN
U1135	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	0.0	NaN
U1136	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
U1137	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	2.0	NaN
U1138	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN

3 : Calcualte cosine similarity matrix between users

]:

```
# Similarity Matrix without function
pivoted_data_copy = pivoted_data.copy()
pivoted_data_copy.fillna(0, inplace=True)

# Compute cosine similarity matrix
cos_sim_matrix = np.zeros((len(pivoted_data_copy), len(pivoted_data_copy)))

for i in range(len(pivoted_data_copy)):
    for j in range(i, len(pivoted_data_copy)):
        dot_product = np.dot(pivoted_data_copy.iloc[i], pivoted_data_copy.iloc[j])
        norm_i = np.linalg.norm(pivoted_data_copy.iloc[i])
        norm_j = np.linalg.norm(pivoted_data_copy.iloc[j])
        cos_sim = dot_product / (norm_i * norm_j)
        cos_sim_matrix[i, j] = cos_sim
        cos_sim_matrix[j, i] = cos_sim

# Convert the matrix to a dataframe with user IDs as index and columns
cos_sim_df = pd.DataFrame(cos_sim_matrix, index=pivoted_data_copy.index, columns=pivoted_data_copy.index)

# Replace NaN values with 0
cos_sim_df.fillna(0, inplace=True)
cos_sim_df.to_csv('Group1_Part1_COSINE_11.csv', index=False)
# Print the resulting dataframe
print(cos_sim_df)
```

COSINE_SIMILARITY_MATRIX

```
user      U1001     U1002     U1003     U1004     U1005     U1006     U1007  \
user
U1001  1.000000  0.227921  0.000000  0.000000  0.059761  0.000000  0.188982
U1002  0.227921  1.000000  0.148454  0.158362  0.095346  0.000000  0.075378
U1003  0.000000  0.148454  1.000000  0.000000  0.000000  0.227921  0.000000
U1004  0.000000  0.158362  0.000000  1.000000  0.166091  0.081044  0.131306
U1005  0.059761  0.095346  0.000000  0.166091  1.000000  0.000000  0.237171
...         ...       ...       ...       ...       ...       ...       ...
U1134  0.083478  0.199778  0.380609  0.000000  0.000000  0.068160  0.110432
U1135  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
U1136  0.000000  0.322329  0.065795  0.280745  0.000000  0.041239  0.000000
U1137  0.161165  0.385695  0.472377  0.000000  0.000000  0.197386  0.106600
U1138  0.000000  0.355335  0.232104  0.000000  0.000000  0.290957  0.000000

user   U1008     U1009     U1010  ...  U1129  U1130  U1131     U1132  U1133  \
user                                ...
U1001    0.0  0.129641  0.000000  ...    0.0    0.0    0.0  0.353553    0.0
U1002    0.0  0.517088  0.000000  ...    0.0    0.0    0.0  0.402911    0.0
U1003    0.0  0.337760  0.000000  ...    0.0    0.0    0.0  0.000000    0.0
U1004    0.0  0.045038  0.000000  ...    0.0    0.0    0.0  0.350931    0.0
U1005    0.0  0.000000  0.447214  ...    0.0    0.0    0.0  0.084515    0.0
...      ...       ...       ...  ...    ...    ...    ...       ...    ...
U1134    0.0  0.303022  0.156174  ...    0.0    0.0    0.0  0.177084    0.0
U1135    0.0  0.000000  0.000000  ...    0.0    0.0    0.0  0.000000    0.0
U1136    0.0  0.183340  0.000000  ...    0.0    0.0    0.0  0.142857    0.0
U1137    0.0  0.365636  0.000000  ...    0.0    0.0    0.0  0.227921    0.0
U1138    0.0  0.000000  0.000000  ...    0.0    0.0    0.0  0.000000    0.0
```

4 : Create 7-K nearest neighbour for each user

```
similarity_matrix = cos_sim_df.copy()

# create a new DataFrame to store the NN information
nn_data = pd.DataFrame(index=similarity_matrix.index, columns=[f'{i}thNNUserID' for i in range(1, 8)])

# for each row in the similarity matrix, find the 7 most similar users
for user in similarity_matrix.index:
    sim_series = similarity_matrix.loc[user].sort_values(ascending=False)
    nn_series = sim_series.iloc[1:8]
    nn_tuples = [(idx, round(nn_series.loc[idx], 2)) for idx in nn_series.index]
    nn_data.loc[user] = nn_tuples

nn_data
```

NEAREST_NEIGHBOR_DATA_FRAME

	1thNNUserID	2thNNUserID	3thNNUserID	4thNNUserID	5thNNUserID	6thNNUserID	7thNNUserID
user							
U1001	(U1054, 0.47)	(U1036, 0.45)	(U1024, 0.43)	(U1071, 0.42)	(U1092, 0.4)	(U1116, 0.4)	(U1055, 0.4)
U1002	(U1029, 0.58)	(U1009, 0.52)	(U1090, 0.44)	(U1045, 0.43)	(U1027, 0.4)	(U1132, 0.4)	(U1078, 0.4)
U1003	(U1029, 0.47)	(U1137, 0.47)	(U1061, 0.39)	(U1134, 0.38)	(U1009, 0.34)	(U1048, 0.3)	(U1108, 0.3)
U1004	(U1132, 0.35)	(U1016, 0.35)	(U1061, 0.31)	(U1097, 0.28)	(U1136, 0.28)	(U1078, 0.26)	(U1104, 0.26)
U1005	(U1075, 0.63)	(U1014, 0.58)	(U1125, 0.56)	(U1010, 0.45)	(U1053, 0.38)	(U1018, 0.35)	(U1016, 0.35)
...	...	...	...	...	...	...	...
U1134	(U1036, 0.52)	(U1059, 0.51)	(U1108, 0.4)	(U1003, 0.38)	(U1029, 0.38)	(U1122, 0.37)	(U1136, 0.35)
U1135	(U1095, 0.0)	(U1089, 0.0)	(U1090, 0.0)	(U1091, 0.0)	(U1092, 0.0)	(U1093, 0.0)	(U1094, 0.0)
U1136	(U1033, 0.41)	(U1134, 0.35)	(U1112, 0.34)	(U1089, 0.33)	(U1002, 0.32)	(U1038, 0.31)	(U1057, 0.3)
U1137	(U1003, 0.47)	(U1029, 0.45)	(U1002, 0.39)	(U1009, 0.37)	(U1045, 0.34)	(U1116, 0.34)	(U1077, 0.33)
U1138	(U1086, 0.41)	(U1029, 0.4)	(U1002, 0.36)	(U1090, 0.34)	(U1027, 0.32)	(U1046, 0.31)	(U1112, 0.3)

5 : Calculate the mean ratings of users

```
# create a new DataFrame to store the means
mean_data = pd.DataFrame(columns=['R_Mean'])

# for each user, calculate the mean rating
for user, row in pivoted_data_copy.iterrows():
    mean = row.sum() / row.count()
    mean_data.loc[user] = {'R_Mean': mean}
mean_data
```

MEAN_RATING OF USERS

	R_Mean
U1001	0.076923
U1002	0.107692
U1003	0.130769
U1004	0.115385
U1005	0.092308
...	...
U1134	0.176923
U1135	0.000000
U1136	0.123077
U1137	0.169231
U1138	0.038462

6 : Create Prediciton Matrix

```
rating_data = pivoted_data.copy()
K_NNUsers = nn_data
R_Mean = mean_data

# create a copy of rating_data that replaces NaN with 0
rating_data_copy = rating_data.fillna(0)

# create an empty DataFrame for the predictions
Prediction_Matrix = pd.DataFrame(columns=rating_data.columns, index=rating_data.index)

# for each user and place, calculate the predicted rating
for user in rating_data.index:
    for place in rating_data.columns:
        if pd.isna(rating_data.loc[user, place]):
            r_mean = R_Mean.loc[user, 'R_Mean']
            knn_data = K_NNUsers.loc[user]
            num = 0
            den = 0
            for i in range(1, 8):
                nn_user, sim = knn_data[f'{i}thNNUserID']
                nn_rating = rating_data_copy.loc[nn_user, place]
                nn_mean = R_Mean.loc[nn_user, 'R_Mean']
                if nn_rating != 0:
                    num += sim * (nn_rating - nn_mean)
                    den += sim
            if den != 0:
                pred_rating = r_mean + num / den
            else:
                pred_rating = r_mean
            Prediction_Matrix.loc[user, place] = pred_rating
        else:
            Prediction_Matrix.loc[user, place] = rating_data.loc[user, place]
```

PREDICTION MATRIX

user	Place 132560	Place 132561	Place 132564	Place 132572	Place 132583	Place 132584	Place 132594	Place 132608	Place 132609	Place 132613	...	Place 135080	Place 135081
U1001	0.076923	0.076923	0.076923	1.484615	0.076923	0.076923	0.076923	0.076923	0.076923	0.076923	...	0.076923	0.076923
U1002	0.107692	0.107692	0.107692	0.969231	0.107692	0.107692	0.107692	0.107692	0.107692	0.107692	...	0.107692	1.030769
U1003	0.130769	0.130769	0.130769	0.95786	0.130769	0.130769	0.130769	0.130769	0.130769	0.130769	...	2.0	0.130769
U1004	0.115385	0.115385	0.115385	0.892308	0.115385	0.115385	0.115385	0.115385	0.115385	0.115385	...	1.892308	1.038462
U1005	0.092308	0.092308	0.092308	0.092308	0.092308	0.092308	0.092308	0.092308	0.092308	0.092308	...	0.092308	1.475456
...	...	...	...	...	...	...	...	...	...	...	...	...	...
U1134	0.176923	0.176923	0.176923	0.0	0.176923	0.176923	0.176923	0.176923	0.176923	0.176923	...	1.0	0.176923
U1135	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
U1136	0.123077	0.123077	0.123077	1.0	0.123077	0.123077	0.123077	0.123077	0.123077	0.123077	...	0.992308	0.123077
U1137	0.169231	0.169231	0.169231	0.169231	0.169231	0.169231	0.169231	0.169231	0.169231	0.169231	...	2.038462	0.169231
U1138	0.038462	0.038462	0.038462	0.922012	0.038462	0.038462	0.038462	0.038462	0.038462	0.038462	...	0.038462	0.038462

7.1 : Recommend 5 not-yet visted to user with place and predicited ratings

+ Code + Markdown

```
# recommended_places

# sort the predicted rating in descending order with its corresponding place
Prediction_Sorted = Prediction_Matrix.apply(lambda x: pd.Series(x.sort_values(ascending=False)
                                                        .iloc[:5].index.tolist()), axis=1)

# compare the place id with test data and skip the ones that are already in the test data
test_data_places = set(rating_testset["PlaceID"].unique().tolist())

Recommendations = Prediction_Sorted.apply(lambda x: [(place, Prediction_Matrix.loc[user, place]) for place in x
                                                     if place not in test_data_places][:5], axis=1)


Recommendations
```

7.2 : Seperate the Data Series to be each columns

```
# create an empty list to store the data
data = []

# iterate through the rows of the Series
for user, ratings in Recommendations.iteritems():
    # iterate through the ratings for each user
    for place, rating in ratings:
        # append the data as a tuple to the list
        data.append((user, place, rating))

# create a DataFrame from the list of tuples
Recommendations_Seperated = pd.DataFrame(data, columns=['user', 'PlaceID', 'predicted_rating'])

# output the DataFrame
Recommendations_Seperated
```

RECOMMENDATION_SEPARATED

	user	PlaceID	predicted_rating
0	U1001	Place 135025	1.900000
1	U1001	Place 132825	1.424696
2	U1001	Place 135052	1.263499
3	U1001	Place 135062	1.263499
4	U1001	Place 135047	0.969231
...	...	...	...
685	U1138	Place 132921	2.000000
686	U1138	Place 132922	2.000000
687	U1138	Place 135058	1.946154
688	U1138	Place 135065	1.946154
689	U1138	Place 135045	1.930769

8 : Merge nn-data and recommendation for each users

```
# merge nn_data and Recommendations_Seperated on 'user' column
merged_df = pd.merge(nn_data.reset_index(), Recommendations_Seperated, on='user')

# rename 'user' column to 'userID'
merged_df = merged_df.rename(columns={'user': 'userID'})

# select only the required columns
merged_df = merged_df[['userID', '1thNNUserID', '2thNNUserID', '3thNNUserID', '4thNNUserID', '5thNNUserID',
                       '6thNNUserID', '7thNNUserID', 'PlaceID', 'predicted_rating']]

# output the merged dataframe
merged_df
```

9 : Merge and Recommend users in test set

```
# Find unique users in test data
unique_users = rating_testset['userID'].unique()
unique_users_df = pd.DataFrame({'userID': unique_users})

result = pd.merge(unique_users_df.reset_index(), merged_df, on='userID')
result = result.drop(['index'], axis=1)
result.to_csv('Group1_Part1_RECOMMEND_12.csv', index=False)
result
```

RECOMMEDING TOP-5 NOT YET VISITED PLACES WITH RATINGS

	userID	1thNNUserID	2thNNUserID	3thNNUserID	4thNNUserID	5thNNUserID	6thNNUserID	7thNNUserID	PlaceID	predicted_rating
0	U1003	(U1029, 0.47)	(U1137, 0.47)	(U1061, 0.39)	(U1134, 0.38)	(U1009, 0.34)	(U1048, 0.3)	(U1108, 0.3)	Place 135052	1.263499
1	U1003	(U1029, 0.47)	(U1137, 0.47)	(U1061, 0.39)	(U1134, 0.38)	(U1009, 0.34)	(U1048, 0.3)	(U1108, 0.3)	Place 135032	0.038462
2	U1003	(U1029, 0.47)	(U1137, 0.47)	(U1061, 0.39)	(U1134, 0.38)	(U1009, 0.34)	(U1048, 0.3)	(U1108, 0.3)	Place 132937	1.315929
3	U1003	(U1029, 0.47)	(U1137, 0.47)	(U1061, 0.39)	(U1134, 0.38)	(U1009, 0.34)	(U1048, 0.3)	(U1108, 0.3)	Place 135059	1.477328
4	U1003	(U1029, 0.47)	(U1137, 0.47)	(U1061, 0.39)	(U1134, 0.38)	(U1009, 0.34)	(U1048, 0.3)	(U1108, 0.3)	Place 132755	0.038462
...	...	...	...	...	...	...	...	...	...	...
70	U1137	(U1003, 0.47)	(U1029, 0.45)	(U1002, 0.39)	(U1009, 0.37)	(U1045, 0.34)	(U1116, 0.34)	(U1077, 0.33)	Place 135038	0.038462
71	U1137	(U1003, 0.47)	(U1029, 0.45)	(U1002, 0.39)	(U1009, 0.37)	(U1045, 0.34)	(U1116, 0.34)	(U1077, 0.33)	Place 135051	0.900000
72	U1137	(U1003, 0.47)	(U1029, 0.45)	(U1002, 0.39)	(U1009, 0.37)	(U1045, 0.34)	(U1116, 0.34)	(U1077, 0.33)	Place 135025	1.900000
73	U1137	(U1003, 0.47)	(U1029, 0.45)	(U1002, 0.39)	(U1009, 0.37)	(U1045, 0.34)	(U1116, 0.34)	(U1077, 0.33)	Place 135080	0.038462
74	U1137	(U1003, 0.47)	(U1029, 0.45)	(U1002, 0.39)	(U1009, 0.37)	(U1045, 0.34)	(U1116, 0.34)	(U1077, 0.33)	Place 135075	0.038462

PART2

USER_PROFILE

user	Place 132560	Place 132561	Place 132564	Place 132572	Place 132583	Place 132584	Place 132594	Place 132608	Place 132609	Place 132613	...	Place 135080	Place 135081	Place 135082	Place 135085	Place 135086
U1001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
U1002	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0
U1003	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	2.0	0.0	0.0	0.0	0.0
U1004	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
U1005	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
U1134	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	2.0	0.0
U1135	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
U1136	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
U1137	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	2.0	0.0
U1138	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0

3 : User Profile

```
# Load the user profile data into a pandas DataFrame
user_profile = user_profile.copy()

# Load the R_Mean data into a pandas DataFrame
R_Mean = mean_data.copy()

# Subtract the R_Mean value from each column in the user profile
for user in user_profile.index:
    user_profile.loc[user] = user_profile.loc[user] - R_Mean.loc[user]['R_Mean']

user_profile.to_csv('Group1_Part2_PROFILE_21.csv', index=True)
# Output the modified user profile DataFrame
user_profile
```

ADJUSTED_MEAN USER PROFILE

```
      Place 132560  Place 132561  Place 132564  Place 132572  Place 132583  \
user
U1001    -0.076923     -0.076923     -0.076923     -0.076923     -0.076923
U1002    -0.107692     -0.107692     -0.107692     -0.107692     -0.107692
U1003    -0.130769     -0.130769     -0.130769     -0.130769     -0.130769
U1004    -0.115385     -0.115385     -0.115385     -0.115385     -0.115385
U1005    -0.092308     -0.092308     -0.092308     -0.092308     -0.092308
...            ...           ...           ...           ...           ...
U1134    -0.176923     -0.176923     -0.176923     -0.176923     -0.176923
U1135     0.000000      0.000000      0.000000      0.000000      0.000000
U1136    -0.123077     -0.123077     -0.123077     -0.123077     -0.123077
U1137    -0.169231     -0.169231     -0.169231     -0.169231     -0.169231
U1138    -0.038462     -0.038462     -0.038462     -0.038462     -0.038462

      Place 132584  Place 132594  Place 132608  Place 132609  Place 132613  \
user
U1001    -0.076923     -0.076923     -0.076923     -0.076923     -0.076923
U1002    -0.107692     -0.107692     -0.107692     -0.107692     -0.107692
U1003    -0.130769     -0.130769     -0.130769     -0.130769     -0.130769
U1004    -0.115385     -0.115385     -0.115385     -0.115385     -0.115385
U1005    -0.092308     -0.092308     -0.092308     -0.092308     -0.092308
...            ...           ...           ...           ...           ...
U1134    -0.176923     -0.176923     -0.176923     -0.176923     -0.176923
U1135     0.000000      0.000000      0.000000      0.000000      0.000000
U1136    -0.123077     -0.123077     -0.123077     -0.123077     -0.123077
U1137    -0.169231     -0.169231     -0.169231     -0.169231     -0.169231
U1138    -0.038462     -0.038462     -0.038462     -0.038462     -0.038462
```

4 : Similarity Matrix using adjusted cosine

+ Code + Markdown

```
# Compute mean adjusted ratings
mean_adjusted_ratings = user_profile.sub(user_profile.mean(axis=0), axis=1)

# Compute adjusted cosine similarity matrix
adj_cos_sim_matrix = np.zeros((len(mean_adjusted_ratings.columns), len(mean_adjusted_ratings.columns)))

for i in range(len(mean_adjusted_ratings.columns)):
    for j in range(i, len(mean_adjusted_ratings.columns)):
        dot_product = np.dot(mean_adjusted_ratings.iloc[:, i], mean_adjusted_ratings.iloc[:, j])
        norm_i = np.linalg.norm(mean_adjusted_ratings.iloc[:, i])
        norm_j = np.linalg.norm(mean_adjusted_ratings.iloc[:, j])
        adj_cos_sim = dot_product / (norm_i * norm_j)
        adj_cos_sim_matrix[i, j] = adj_cos_sim
        adj_cos_sim_matrix[j, i] = adj_cos_sim

# Convert the matrix to a dataframe with place IDs as index and columns
adj_cos_sim_df = pd.DataFrame(adj_cos_sim_matrix, index=mean_adjusted_ratings.columns, columns=mean_adjuste

# Replace NaN values with 0
adj_cos_sim_df.fillna(0, inplace=True)

adj_cos_sim_df.to_csv('Group1_Part2_SIMILARITY_22.csv', index=False)
# Print the resulting dataframe
adj_cos_sim_df
```

ADJUSTED_CONSINE_SIMILARITY_MATRIX

	Place 132560	Place 132561	Place 132564	Place 132572	\
Place 132560	1.000000	0.147240	0.096986	-0.062044	
Place 132561	0.147240	1.000000	0.092690	-0.050964	
Place 132564	0.096986	0.092690	1.000000	-0.052590	
Place 132572	-0.062044	-0.050964	-0.052590	1.000000	
Place 132583	0.083539	0.084446	0.051224	-0.056133	
...	...	...	...	...	
Place 135088	0.096026	0.089743	0.052121	-0.055163	
Place 135104	0.301579	0.070812	0.038978	-0.064603	
Place 135106	-0.051166	-0.039499	-0.040441	0.101361	
Place 135108	-0.010036	-0.006806	-0.019357	-0.044024	
Place 135109	0.096790	0.095112	0.058616	-0.050722	

	Place 132583	Place 132584	Place 132594	Place 132608	\
Place 132560	0.083539	0.414840	0.477665	0.081799	
Place 132561	0.084446	0.061250	0.120870	0.077006	
Place 132564	0.051224	0.031907	0.076623	0.272583	
Place 132572	-0.056133	-0.056575	-0.064100	-0.051350	
Place 132583	1.000000	0.030425	0.066548	0.041387	
...	...	...	...	...	
Place 135088	0.324408	0.027987	0.074729	0.040856	
Place 135104	0.034748	0.482396	0.822200	0.246855	
Place 135106	-0.044905	-0.043187	-0.051842	-0.039293	
Place 135108	-0.021363	-0.029451	-0.018419	-0.022041	
Place 135109	0.052788	0.035825	0.077677	0.047656	

	Place 132609	Place 132613	...	Place 135080	Place 135081	\
Place 132560	0.137686	0.072318	...	-0.058338	-0.052866	
Place 132561	0.136406	0.067713	...	-0.036509	-0.041705	
Place 132564	0.087423	0.036677	...	-0.034364	-0.043572	
Place 132572	-0.055260	-0.058040	...	0.047636	-0.112799	
Place 132583	0.078581	0.034220	...	-0.044565	-0.047673	
...	...	...	...	...	...	
Place 135088	0.084821	0.032968	...	-0.033083	-0.045481	
Place 135104	0.253976	0.439303	...	-0.046046	-0.054689	
Place 135106	-0.043536	-0.044663	...	-0.091635	0.008619	
Place 135108	-0.010607	-0.028684	...	-0.073879	0.122511	

5 : K-NN for places

```
similarity_matrix = adj_cos_sim_df.copy()

# create a new DataFrame to store the NN information
nn_data = pd.DataFrame(index=similarity_matrix.index, columns=[f'{i}thNNPlaceID' for i in range(1, 8)])

# for each row in the similarity matrix, find the 7 most similar users
for user in similarity_matrix.index:
    sim_series = similarity_matrix.loc[user].sort_values(ascending=False)
    nn_series = sim_series.iloc[1:8]
    nn_tuples = [(idx, round(nn_series.loc[idx], 2)) for idx in nn_series.index]
    nn_data.loc[user] = nn_tuples
nn_data
```

7TH K-NEAREST-NEIGHBOR_DATA_FRAME

	1thNNPlaceID	2thNNPlaceID	3thNNPlaceID	4thNNPlaceID	5thNNPlaceID	6thNNPlaceID	7thNNPlaceID
Place 132560	(Place 132732, 0.81)	(Place 132667, 0.56)	(Place 132594, 0.48)	(Place 132663, 0.48)	(Place 132584, 0.41)	(Place 132630, 0.31)	(Place 132740, 0.31)
Place 132561	(Place 132665, 0.73)	(Place 132654, 0.66)	(Place 132626, 0.61)	(Place 132706, 0.57)	(Place 135040, 0.19)	(Place 132560, 0.15)	(Place 132766, 0.15)
Place 132564	(Place 132717, 0.81)	(Place 132733, 0.43)	(Place 132715, 0.35)	(Place 132740, 0.3)	(Place 132626, 0.29)	(Place 132608, 0.27)	(Place 132660, 0.23)
Place 132572	(Place 135075, 0.5)	(Place 135048, 0.43)	(Place 132884, 0.31)	(Place 135074, 0.31)	(Place 135046, 0.28)	(Place 132875, 0.28)	(Place 135034, 0.22)
Place 132583	(Place 134986, 0.58)	(Place 132768, 0.46)	(Place 135001, 0.38)	(Place 135088, 0.32)	(Place 135000, 0.29)	(Place 132773, 0.29)	(Place 135018, 0.16)
...	...	...	...	...	...	...	...
Place 135088	(Place 135109, 0.47)	(Place 132768, 0.45)	(Place 134986, 0.37)	(Place 135016, 0.36)	(Place 135018, 0.36)	(Place 132583, 0.32)	(Place 132773, 0.15)
Place 135104	(Place 132594, 0.82)	(Place 132667, 0.81)	(Place 132663, 0.63)	(Place 132584, 0.48)	(Place 132613, 0.44)	(Place 132733, 0.39)	(Place 132740, 0.39)
Place 135106	(Place 135041, 0.33)	(Place 135052, 0.32)	(Place 135062, 0.26)	(Place 135060, 0.23)	(Place 135028, 0.21)	(Place 135048, 0.2)	(Place 132885, 0.18)

6 : Predicition Matrix

```
# Pred_Rating(u, j) = Σ sim(i, j) * Rating(u, i) / Σ sim(i, j)
rating_data = pivoted_data.copy()
K_NNPlaces = nn_data

# create a copy of rating_data that replaces NaN with 0
rating_data_copy = rating_data.fillna(0)

# create an empty DataFrame for the predictions
Prediction_Matrixp2 = pd.DataFrame(columns=rating_data.columns, index=rating_data.index)

# for each user and place, calculate the predicted rating
for user in rating_data.index:
    for place in rating_data.columns:
        if pd.isna(rating_data.loc[user, place]):
            knn_data = K_NNPlaces.loc[place]
            num = 0
            den = 0
            for i in range(1, 8):
                nn_place, sim = knn_data[f'{i}thNNPlaceID']
                nn_rating = rating_data_copy.loc[user, nn_place]
                if nn_rating != 0:
                    num += sim * nn_rating
                    den += sim
            if den != 0:
                pred_rating = num / den
            else:
                pred_rating = 0
            Prediction_Matrixp2.loc[user, place] = pred_rating
        else:
            Prediction_Matrixp2.loc[user, place] = rating_data.loc[user, place]

Prediction_Matrixp2
```

PREDICTION_MATRIX

user	Place 132560	Place 132561	Place 132564	Place 132572	Place 132583	Place 132584	Place 132594	Place 132608	Place 132609	Place 132613	...	Place 135080	Place 135081	Place 135082	Place 135085
U1001	0	1.0	0	0	0	0	0	0	0	0	...	0	0	0	0.0
U1002	0	0	0	0	0	0	0	0	0	0	...	1.0	0	0	1.0
U1003	0	0	0	2.0	0	0	0	0	0	0	...	2.0	0	0	1.509434
U1004	0	0	0	0	0	0	0	0	0	0	...	2.0	0	0	0
U1005	0	0	0	0	0	0	0	0	0	0	...	0	1.563291	1.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
U1134	0	0	0	0.0	0	0	0	0	0	0	...	1.0	0	0	2.0
U1135	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0.0
U1136	0	0	0	1.27451	0	0	0	0	0	0	...	1.0	0	0	1.490566
U1137	0	0	0	2.0	0	0	0	0	0	0	...	2.0	0	0	2.0
U1138	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0

7.1 : Recommend 10 not yet visited and predicited ratings

```
# sort the predicted rating in descending order with its corresponding place
Prediction_Sortedp2 = Prediction_Matrixp2.apply(lambda x: pd.Series(x.sort_values(ascending=False)
                                                        .iloc[:10].index.tolist()), axis=1)

# compare the place id with test data and skip the ones that are already in the test data
test_data_places = set(rating_testset["PlaceID"].unique().tolist())

Recommendationsp2 = Prediction_Sortedp2.apply(lambda x: [(place, Prediction_Matrix.loc[user, place]) for place in x
                                                     if place not in test_data_places and place not in rating_data.loc[user].dropna().index][:10], axis=
Recommendationsp2
```

7.2 : Seperate the Data Series to be each columns

```
# create an empty list to store the data
data = []

# iterate through the rows of the Series
for user, ratings in Recommendationsp2.iteritems():
    # iterate through the ratings for each user
    for place, rating in ratings:
        # append the data as a tuple to the list
        data.append((user, place, rating))

# create a DataFrame from the list of tuples
Recommendations_Seperated = pd.DataFrame(data, columns=['userID', 'PlaceID', 'predicted_rating'])

# output the DataFrame
Recommendations_Seperated
```

8 : Merge test and recommendation

```python
# Find unique users in test data
unique_users = rating_testset['userID'].unique()
unique_users_df = pd.DataFrame({'userID': unique_users})
result = pd.merge(unique_users_df.reset_index(), Recommendations_Seperated, on='userID')
result = result.drop(['index'], axis=1)
result.to_csv('Group1_Part2_RECOMMEND_23.csv', index=False)
result
```

RECOMMENDATION MATRIX

```
user
U1001    [(Place 135047, 0.9692307692307692), (Place 13...
U1002    [(Place 135045, 1.9307692307692308), (Place 13...
U1003    [(Place 135065, 1.9461538461538463), (Place 13...
U1004    [(Place 135051, 0.9), (Place 135044, 0.9461538...
U1005    [(Place 132872, 0.9615384615384616), (Place 13...
                                 ...
U1134    [(Place 132862, 1.449120879120879), (Place 132...
U1135    [(Place 132560, 0.038461538461538464), (Place ...
U1136    [(Place 132885, 0.038461538461538464), (Place ...
U1137    [(Place 135025, 1.9000000000000004), (Place 13...
U1138    [(Place 132958, 0.038461538461538464), (Place ...
Length: 138, dtype: object
```

	userID	PlaceID	predicted_rating
0	U1003	Place 135065	1.946154
1	U1003	Place 135035	0.038462
2	U1003	Place 132754	0.038462
3	U1003	Place 132755	0.038462
4	U1003	Place 132955	0.038462
...	...	...	...
139	U1137	Place 135059	1.477328
140	U1137	Place 132825	1.424696
141	U1137	Place 132951	0.944822
142	U1137	Place 132834	1.000000
143	U1137	Place 132937	1.315929

9 : Calculate RSME, Precision, and Recall

```python
pivoted_data = user_profile
test_data = rating_testset
Prediction_Matrix = Prediction_Matrixp2

# create a list to store the predicted and actual ratings
predicted_ratings = []
actual_ratings = []
userlist = []
placelist = []

# iterate over the rows in the test dataset
for index, row in test_data.iterrows():
    user_id = row['userID']
    place_id = row['PlaceID']
    actual_rating = row['Rating']

    # get the predicted rating from the Prediction_Matrix
    predicted_rating = Prediction_Matrix.loc[user_id, f'Place {place_id}']

    # add the predicted and actual ratings to the list
    userlist.append(user_id)
    placelist.append(place_id)
    predicted_ratings.append(predicted_rating)
    actual_ratings.append(actual_rating)


# compute the MSE and RMSE
mse = np.mean(np.power(np.array(actual_ratings) - np.array(predicted_ratings), 2))
rmse = np.sqrt(mse)

# Set a range of threshold values for the predicted ratings
threshold_values = [1.5]
```

```python
# Initialize variables to store the best threshold and its corresponding precision and recal
best_threshold = None
best_precision = 0
best_recall = 0
# Initialize lists to store results
rmse_list = []
precision_list = []
recall_list = []

# Iterate over the threshold values
Test = pd.DataFrame({
    'userID': userlist,
    'PlaceID': placelist,
    'Actual Rating': actual_ratings,
    'Predicted Rating': predicted_ratings
})
for threshold in threshold_values:
    # Calculate the precision and recall for the current threshold
    predicted_labels = np.where(Test['Predicted Rating'] >= threshold, 1, 0)
    actual_labels = np.where(Test ['Actual Rating']>= threshold, 1, 0)

    true_positives = np.sum(np.logical_and(predicted_labels == 1, actual_labels == 1))
    false_positives = np.sum(np.logical_and(predicted_labels == 1, actual_labels == 0))
    false_negatives = np.sum(np.logical_and(predicted_labels == 0, actual_labels == 1))

    precision = true_positives / (true_positives + false_positives)
    recall = true_positives / (true_positives + false_negatives)

    # Update the best threshold and its corresponding precision and recall if applicable
    if precision + recall > best_precision + best_recall:
        best_threshold = threshold
        best_precision = precision
        best_recall = recall
rmse_list.append(rmse)
precision_list.append(precision)
recall_list.append(recall)

# Combine lists into a DataFrame
Evaluation = pd.DataFrame({
    'RMSE': rmse_list,
    'Precision': precision_list,
    'Recall': recall_list
})


Evaluation.to_csv("Group1_Part2_EVAL_24.csv",index=False)

Evaluation
```

TEST DATASET (WITH PREDICTED RATING)

	userID	PlaceID	Actual Rating	Predicted Rating
0	U1003	132825	2	2.000000
1	U1003	135079	2	2.000000
2	U1006	132825	1	0.000000
3	U1006	135079	1	0.000000
4	U1009	132834	2	1.323529
5	U1009	135038	2	1.000000
6	U1016	132834	2	2.000000
7	U1016	135060	2	2.000000
8	U1022	135038	2	1.515625
9	U1022	135062	1	2.000000
10	U1024	135032	2	1.000000

EVALUATION MATRIX

	RMSE	Precision	Recall
0	0.983406	0.666667	0.5

Thank You