MINI-PROJECT 1 PRESENTATION

CSX4207 sec541

by Group 4

CONTENTS

MEMBER

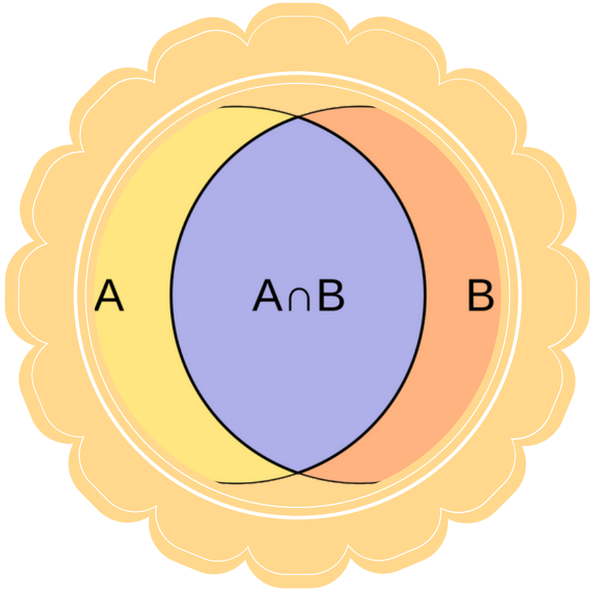
Member :

- AHAD RAHMAN 6411247
- Thit Lwin Win Thant 6540122
- Kaung Khant Lin 6540131
- Thust Thongsricharoen 6714508

INTRODUCTION

- Develop personalized hotel recommendations using datasets
- Two approaches:
  - Similarity-based (Jaccard)
  - Content-based filtering (Cosine Similarity)
- Build user profiles & analyze hotel attributes
- Hands-on practice in:
- Data preprocessing
- Algorithm design
- Evaluation

OBJECTIVES



PART 1 : Simple Recommendation Algorithm (Jaccard similarity)

- Preprocess hotel attributes
- Create profiles for 5 users
- Compute Jaccard similarity between user profiles and unvisited hotels
- Recommend top 5 hotels per user based on similarity

JACCARD SIMILARITY

- also known as the Jaccard index, is a metric used to measure the similarity between two sets of data. It is calculated by dividing the size of the intersection of the two sets by the size of their union.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

- J = Jaccard distance
- A = set 1
- B= set 2

PREPROSE THE DATA

hotelname	address	city	country	propertytype	starrating	roomamenities	roomtype	ratedescrip
The Harmonie Vienna BW Premier Collection	Harmoniegasse 5-7	Vienna	Austria	Hotels	4	Air conditioning ;Alarm clock ;Bathrobes ;Clos...	Comfort Double Room	Room siz m_square291 ft_sq City viev
The Harmonie Vienna BW Premier Collection	Harmoniegasse 5-7	Vienna	Austria	Hotels	4	Air conditioning ;Alarm clock ;Bathrobes ;Clos...	Classic Double or Twin Room	Room siz m_square215 ft_sq City viev
The Harmonie Vienna BW Premier Collection	Harmoniegasse 5-7	Vienna	Austria	Hotels	4	Air conditioning ;Alarm clock ;Bathrobes ;Clos...	Superior Double Room	Room siz m_square355 ft_sq City viev
The Harmonie Vienna BW Premier Collection	Harmoniegasse 5-7	Vienna	Austria	Hotels	4	Air conditioning ;Bathrobes ;Blackout curtains...	Suite	Room siz m_square377 ft_sq Bathtub
The Harmonie Vienna BW Premier Collection	Harmoniegasse 5-7	Vienna	Austria	Hotels	4	Air conditioning ;Blackout curtains ;Coffeetea...	Classic Double or Twin Room	Free Breakfast nothing until Noven 8

OBJECTIVES

- **Check hotel attributes**

hotel attributes :

- Item ID
- Hotel ID
- Hotel Name
- Address
- City
- Country
- Proper type
- Starting

hotel attributes :

- Room Amenities
- Room Type
- Rate Descirption
- Price Average

OBJECTIVES

- **Check hotel attributes**

```
import pandas as pd
df = pd.read_csv('HOTEL_OUTDATASET.csv')
allproper = df['propertytype']
uniquep = sorted(set(proer for proer in allproper))
for i in uniquep:
    print(i)
```

```
Hotels
Resorts
```

- hotel attributes :Proper type
- Hotels
- Resorts

SELECTION

- We selected Room Amenities & Room Type

OBJECTIVES

- **Preprocess hotel attributes**
  - Cleaning the Data

```
raw_mapping = {
    'free wifi in all rooms': 'wifi',
    'in-room tablet': 'tablet',
    'mini bar': 'minibar',
    'in-room safe box': 'safebox',
    'tv flat screen': 'tv',
    'tv in bathroom': 'tv',
    'wifi in public areas': 'public wifi',
    'pets allowed in room':'petallow',
    'interconnecting rooms available':'interconnecting roo
    'satellitecable channels':'satellitecable'
}
mapping = {
    k.lower().strip(): v.lower().strip()
    for k, v in raw_mapping.items()
```

amenity_vector
{'hair dryer', 'alarm clock', 'closet', 'laptop safe box', 'safebox', 'wake-up ser
{'hair dryer', 'alarm clock', 'closet', 'laptop safe box', 'safebox', 'wake-up ser
{'hair dryer', 'alarm clock', 'closet', 'laptop safe box', 'safebox', 'wake-up ser
{'hair dryer', 'slippers', 'closet', 'laptop safe box', 'safebox', 'wake-up service
{'safebox', 'coffeetea maker', 'hair dryer', 'slippers', 'toiletries', 'blackout curt
{'safebox', 'coffeetea maker', 'hair dryer', 'slippers', 'toiletries', 'blackout curt
{'safebox', 'coffeetea maker', 'hair dryer', 'slippers', 'toiletries', 'blackout curt
{'safebox', 'coffeetea maker', 'hair dryer', 'slippers', 'toiletries', 'blackout curt
{'safebox', 'coffeetea maker', 'hair dryer', 'slippers', 'toiletries', 'blackout curt
{'safebox', 'shower', 'tv', 'air conditioning', 'free wi-fi in all rooms'}
{'safebox', 'telephone', 'wake-up service', 'hair dryer', 'toiletries', 'heating', '
{'safebox', 'hair dryer', 'toiletries', 'free bottled water', 'satellitecable', 'showe

OBJECTIVES

- **Preprocess roomtype attributes**
  - Step 1: Preprocess roomtype

```python
# --- Step 1: Preprocessing for 'roomtype' column ---
def preprocess_roomtype(roomtype_str):
    if pd.isna(roomtype_str):
        return set() # Return an empty set for missing values
    # 2. Lowercase
    tokens = str(roomtype_str).lower()
    # 3. Replace spaces with underscores
    tokens = tokens.replace(' ', '_')
    # 4. Remove leading/trailing punctuation and split into potential tokens
    potential_tokens = re.split(r'[;,/_\-]+', tokens)

    processed_tokens = set()
    for token in potential_tokens:
        cleaned_token = re.sub(r'^[^\w]+|[^\w]+$', '', token)
        if cleaned_token == 'room':
            pass

        elif cleaned_token == 'double':
            processed_tokens.add('double')

        elif cleaned_token and len(cleaned_token) > 1 and cleaned_token not in ['with', 'only', 'use', 'or', 'of']:
            processed_tokens.add(cleaned_token)
```

OBJECTIVES

- **Preprocess roomtype attributes**
  - Step 1: Preprocess roomtype

```
    final_tokens = set()
    has_meaningful_token = False
    for token in processed_tokens:
        if token not in ['with', 'only']: # Re-apply filtering for safety
            if token == 'room':
                if len(processed_tokens) == 1:
                    final_tokens.add('single')
                    has_meaningful_token = True
            elif token == 'double':
                final_tokens.add('double')
                has_meaningful_token = True
            else:
                final_tokens.add(token)
                has_meaningful_token = True

    if not has_meaningful_token and any(t.lower() == 'room' for t in str(roomtype_str).lower().split()):
        final_tokens.add('single')
    return final_tokens

df['roomtype_tokens'] = df['roomtype'].apply(preprocess_roomtype)
```

OBJECTIVES

- **Preprocess roomtype attributes**
  - Step 2: Common Token Filtering

```
# --- Step 2: Common Token Filtering (for roomtype tokens) ---
token_counts = {}
for tokens_set in df['roomtype_tokens']:
    for token in tokens_set:
        token_counts[token] = token_counts.get(token, 0) + 1

total_hotels = len(df)

common_token_threshold = 0.70
tokens_to_remove = {token for token, count in token_counts.items() if count / total_hotels >= common_token_threshold}

df['roomtype_tokens_filtered'] = df['roomtype_tokens'].apply(
    lambda tokens_set: {token for token in tokens_set if token not in tokens_to_remove}
)
```

OBJECTIVES

- **Preprocess roomtype attributes**

```
# --- Step 3: Output for Step 1 (for roomtype) ---
df['roomtype_tokens_str'] = df['roomtype_tokens_filtered'].apply(lambda x: '_'.join(sorted(list(x))))

output_columns = ['itemid', 'hotelid', 'roomtype', 'roomtype_tokens', 'roomtype_tokens_filtered', 'roomtype_tokens_str']
df_output = df[output_columns]

df_output.to_csv('Group4_Part1_preprocessed_roomtype.csv', index=False)
print("Saved preprocessed data for 'roomtype' (selected columns) to 'Group4_Part1_preprocessed_roomtype.csv'")

df_output_mapping = df_output[['itemid', 'roomtype_tokens_str']]
df_output_mapping = df_output_mapping.rename(columns={'roomtype_tokens_str': 'tokens'})
df_output_mapping.to_csv('Group4_Part1_id_tokens_roomtype.csv', index=False)

print("Saved minimal mapping for 'roomtype' to 'Group4_Part1_id_tokens_roomtype.csv'")
print("\nPreprocessing for 'roomtype' column completed.")
print("\nFirst 5 rows of the preprocessed data (with selected columns):")
print(df_output[['itemid', 'roomtype', 'roomtype_tokens', 'roomtype_tokens_filtered', 'roomtype_tokens_str']].head())
```

OBJECTIVES

- **Preprocess roomtype attributes**

itemid	hotelid	roomtype	roomtype_tokens	roomtype_tokens_filtered	roomtype_tokens_str
1	31	Comfort Double Room	{'double'}	{'double'}	double
2	31	Classic Double or Twin Room	{'double', 'twin'}	{'double', 'twin'}	double
3	31	Superior Double Room	{'double'}	{'double'}	double
4	31	Suite	{'suite'}	{'suite'}	suite
5	31	Classic Double or Twin Room	{'double', 'twin'}	{'double', 'twin'}	double
6	31	Comfort Room	{'single'}	{'single'}	single
7	31	Run of House	{'run', 'house'}	{'run', 'house'}	house
8	31	Suite-1 Double Bed Living Room	{'double'}	{'double'}	double
9	31	Superior Room	{'single'}	{'single'}	single
10	31	Double Non Smoking	{'double'}	{'double'}	double
11	1429	Superior Double Room	{'double'}	{'double'}	double
12	1429	Run of House	{'run', 'house'}	{'run', 'house'}	house
13	1429	Suite with Balcony	{'suite'}	{'suite'}	suite
14	1429	Premium Double Room	{'double'}	{'double'}	double
15	1429	Comfort Double Room	{'double'}	{'double'}	double
16	1429	Double Or Twin Standard	{'double', 'twin'}	{'double', 'twin'}	double

OBJECTIVES

- **Preprocess hotel attributes**
  - Result after all cleaning datas

id	hotelname	hotelid	roomtype_to	amenity_vector
1	The Harmoni	31	double	{'hair dryer', 'alarm clock', 'closet', 'laptop safe box', 'saf
2	The Harmoni	31	double	{'hair dryer', 'alarm clock', 'closet', 'laptop safe box', 'saf
3	The Harmoni	31	double	{'hair dryer', 'alarm clock', 'closet', 'laptop safe box', 'saf
4	The Harmoni	31	suite	{'hair dryer', 'slippers', 'closet', 'laptop safe box', 'safebo
5	The Harmoni	31	double	{'safebox', 'coffeetea maker', 'hair dryer', 'slippers', 'toile
6	The Harmoni	31	single	{'safebox', 'coffeetea maker', 'hair dryer', 'slippers', 'toile
7	The Harmoni	31	house	{'safebox', 'coffeetea maker', 'hair dryer', 'slippers', 'toile

OBJECTIVES

- **Create profiles for 5 users**
  - Aggregate profile

Pro

- Simplified Decision-Making: making it easier to identify trends and make strategic decisions.
- Deeper User Understanding: Can reveal patterns and behaviors that might be hidden in individual

Cons

- Data Quality Issues.
- Ecological Fallacy: can lead to inaccurate inferences and false conclusions.

userid	itemid
1	21
1	70
1	160
1	440
2	444
2	579
2	770
3	751
3	771
3	779
4	45
4	84
4	155
5	81
5	124

OBJECTIVES

- **Create profiles for 5 users**
  - Aggregate profile

```
user_items = {
    1: [21, 70, 160, 440],
    2: [444, 579, 770],
    3: [751, 771, 779],
    4: [45, 84, 155],
    5: [81, 124, 702]
}
```

```
results = []
for user_id, items in user_items.items():
    roomtypes = set()
    amenities = set()

    for item in items:
        matches = df[df["itemid"] == item]
        for _, row in matches.iterrows():
            roomtypes.add(str(row["roomtype_tokens_str"]).stri
            amenities.update(a.strip(" '") for a in str(row["a

    results.append({
        "userID": user_id,
        "Itemids":", ".join(map(str,items)),
        "roomtypes": ", ".join(sorted(roomtypes)),
        "amenities": ", ".join(sorted(amenities))
    })
```

OBJECTIVES

- **Create profiles for 5 users**
  - Aggregate profile

	A	B	C	D	E
		userID	itemid	roomtype	roomameni
0		1	21, 70, 160,	executive, s	air conditio
1		2	444, 579, 77	double, fan	additional b
2		3	751, 771, 77	apartment,	additional b
3		4	45, 84, 155	double, sing	air conditio
4		5	81, 124, 702	double, fan	additional t

OBJECTIVES

- **Compute Jaccard similarity between user profiles and “Unvisited” hotels**
  - Import CSV file

```
import pandas as pd

user_df = pd.read_csv("user_roomtypes_amenities.csv")
hotel_df = pd.read_csv("Group4_Part1_preprocessed.csv")
```

OBJECTIVES

- **Compute Jaccard similarity between user profiles and “Unvisited” hotels**
  - Ensure every file is in correct type and ready to use

```python
def parse_itemid_list(x):
    if pd.isna(x):
        return []
    return [int(tok.strip()) for tok in str(x).split(",") if tok.strip().isdigit()]
```

```python
def parse_roomtype_tokens(s):
    if pd.isna(s):
        return set()
    return set(t.strip().lower() for t in str(s).replace(";", ",").replace("|", ",").split(",") if t.strip())
```

OBJECTIVES

- **Compute Jaccard similarity between user profiles and “Unvisited” hotels**
  - Ensure every file is in correct type and ready to use

```
def parse_amenities_vector(s):
    if pd.isna(s):
        return set()
    s = str(s).strip()
    if s.startswith("{") and s.endswith("}"):
        s = s[1:-1]   # remove curly braces
    toks = [t.strip().strip("'").strip('"').lower() for t in s.split(",")]
    return set(t for t in toks if t)
```

OBJECTIVES

- **Compute Jaccard similarity between user profiles and “Unvisited” hotels**
  - Mapping the each user & The “Visited” Hotel

```
item_to_features = dict(zip(hotel_df["itemid"].astype(int), hotel_df["feature_set"]))
item_to_hotelid = dict(zip(hotel_df["itemid"].astype(int), hotel_df["hotelid"].astype(int)))

users = user_df["userID"].tolist()[:5]
#user_to_items = {row["userID"]: parse_itemid_list(row["itemid"]) for _, row in user_df.iterrows()}
user_to_items = {row["userID"]:[ int(tok.strip())  for tok in str(row["itemid"]).split(",") if tok.strip().isdigit()] if not pd.isna(row["itemid"])
```

OBJECTIVES

- **Compute Jaccard similarity between user profiles and “Unvisited” hotels**
  - Mapping the each user & The “Visited” Hotel

```
user_to_item_feature_sets = {}
user_to_visited_hotelids = {}
for uid, items in user_to_items.items():
    item_feature_map = {}
    visited_hotelids = set()
    for it in items:
        if it in item_to_features:
            item_feature_map[it] = item_to_features[it]
            visited_hotelids.add(item_to_hotelid[it])
    user_to_item_feature_sets[uid] = item_feature_map
    user_to_visited_hotelids[uid] = visited_hotelids
```

OBJECTIVES

- **Compute Jaccard similarity between user profiles and “Unvisited” hotels**
  - Jaccard

```
def jaccard(a, b):
    if not a and not b:
        return 0.0
    return len(a & b) / len(a | b)
```

OBJECTIVES

- **Compute Jaccard similarity between user profiles and “Unvisited” hotels**
  - Calculation Part

```
for uid in users:
    visited_items = set(user_to_items.get(uid, []))
    visited_hotels = user_to_visited_hotelids.get(uid, set())
    for _, row in hotel_df.iterrows():
        item_id = int(row["itemid"])
        hotel_id = int(row["hotelid"])
        if item_id in visited_items or hotel_id in visited_hotels:
            matrix.loc[item_id, uid] = "Visited"
        else:
            hotel_set = row["feature_set"]
            sims = [jaccard(hotel_set, fset) for fset in user_to_item_feature_sets[uid].values()]
            matrix.loc[item_id, uid] = round(max(sims), 3) if sims else 0.0
```

OBJECTIVES

- **Recommend top 5 hotels per user based on similarity**
  - Set up file

```
:  import pandas as pd
   import io
   import numpy as np

   # --- Step 1: Load and Preprocess Hotel Data ---
   print("--- Step 1: Loading and Preprocessing Hotel Data ---")

   hotel_data_url = 'Group4_Part1_preprocessed.csv'
```

OBJECTIVES

- **Recommend top 5 hotels per user based on similarity**
  - Set up file

```
try:
    df_hotels = pd.read_csv(hotel_data_url)
    print(f"Successfully loaded hotel data from: {hotel_data_url}")

    df_hotels = df_hotels[['itemid', 'hotelname', 'hotelid']].rename(
        columns={'itemid': 'Item ID', 'hotelname': 'Hotel Name'}
    )

    df_hotels['Hotel ID'] = df_hotels['hotelid'].astype(str)

    df_hotels = df_hotels[['Item ID', 'Hotel Name', 'Hotel ID']]
    df_hotels = df_hotels.drop_duplicates(subset=['Item ID', 'Hotel Name', 'Hotel ID']).reset_index(drop=True)
```

OBJECTIVES

- **Recommend top 5 hotels per user based on similarity**
  - Process Similarity data

```
df_similarity = pd.read_csv(io.StringIO(similarity_data_string))

df_similarity = df_similarity.replace('Visited', np.nan)

user_columns = ['User 1', 'User 2', 'User 3', 'User 4', 'User 5']
for col in user_columns:
    df_similarity[col] = pd.to_numeric(df_similarity[col], errors='coerce')
```

OBJECTIVES

- **Recommend top 5 hotels per user based on similarity**
  - Process Similarity data

```
df_melted_similarity = df_similarity.melt(
    id_vars=['Item ID'],
    value_vars=user_columns,
    var_name='User',
    value_name='Similarity'
)

df_melted_similarity['User'] = df_melted_similarity['User'].str.replace(' ', '')

df_melted_similarity.dropna(subset=['Similarity'], inplace=True)

print(f"Similarity data processed. Shape: {df_melted_similarity.shape}")
print("First 5 rows of processed similarity data (long format):")
print(df_melted_similarity.head())
```

OBJECTIVES

- **Recommend top 5 hotels per user based on similarity**
  - Merger & Saving

```
try:
    df_hotels
    df_melted_similarity
except NameError:
    print("Error: df_hotels or df_melted_similarity not found. Please ensure Steps 1 and 2 have been executed.")
    df_hotels = pd.DataFrame(columns=['Item ID', 'Hotel Name', 'Hotel ID'])
    df_melted_similarity = pd.DataFrame(columns=['Item ID', 'User', 'Similarity'])

df_merged = pd.merge(
    df_melted_similarity,
    df_hotels,
    on='Item ID',
    how='inner'
)
```

OBJECTIVES

- **Recommend top 5 hotels per user based on similarity**
  - Merger & Saving

```
df_sorted = df_merged.sort_values(by=['User', 'Similarity'], ascending=[True, False])

top_n = 5
df_top_recommendations = df_sorted.groupby('User').head(n=top_n).reset_index(drop=True)

df_top_recommendations.rename(columns={'User': 'User ID'}, inplace=True)

df_final_recommendations = df_top_recommendations[[
    'User ID',
    'Item ID',
    'Hotel ID',
    'Hotel Name',
    'Similarity'
]]

output_filename = 'Group4_Part1_Recommendation13.csv'
```

OBJECTIVES

PART 2 : Content-Based Filtering

We used Cosine SImilarity for this part

OBJECTIVES

- Using the same user profile from Part 1

Group4_Part1_Profile11.csv

userid	roomtypes	amenities
1	executive, suite, vip	air conditioning, bathrobes, blackout curtains, carpeting, closet, clothes rack, coffeetea maker, daily housekeeping, desk, dvdcd player, fan, free bottled wat
2	double, family	additional bathroom, air conditioning, alarm clock, bathrobes, carpeting, cleaning products, closet, clothes dryer, clothes rack, coffeetea maker, desk, dressi
3	apartment, single, standard	additional bathroom, air conditioning, alarm clock, bathrobes, carpeting, cleaning products, closet, clothes dryer, clothes rack, coffeetea maker, desk, dressi
4	double, single	air conditioning, blackout curtains, carpeting, closet, coffeetea maker, desk, fan, free bottled water, free wi-fi in all rooms, hair dryer, heating, ironing faciliti
5	double, family	additional toilet, air conditioning, blackout curtains, carpeting, cleaning products, closet, coffeetea maker, daily housekeeping, dart board, desk, extra long l

PREPROCESSING DATA

```
# Load user preferences
user_df = pd.read_csv('user_roomtypes_amenities.csv')
# Load hotel data
hotels_df = pd.read_csv('Group4_Part1_preprocessed_roomtype 4 + amen.csv')

# Create mapping from itemid to all hotelids
item_to_hotels = {}
for _, row in hotels_df.iterrows():
    item_id = row['itemid']
    hotel_id = row['hotelid']
    if item_id not in item_to_hotels:
        item_to_hotels[item_id] = set()
    item_to_hotels[item_id].add(hotel_id)

# Expand visited items to visited hotel IDs for each user
visited_hotels_by_user = {}
for user_id, item_list in visited_items.items():
    visited_hotels_by_user[user_id] = set()
    for item_id in item_list:
        if item_id in item_to_hotels:
            visited_hotels_by_user[user_id].update(item_to_hotels[item_id])
```

IMPLEMENTING COSINE SIMILARITY

```python
def cosine_similarity(vec1, vec2):
    # Common vocabulary unioned in both sets
    common_vocab = list(vec1.union(vec2))
    # Abuse space complexity
    vec1_binary = [1 if item in vec1 else 0 for item in common_vocab]
    vec2_binary = [1 if item in vec2 else 0 for item in common_vocab]
    # Calculate dot product
    dot_product = sum(v1 * v2 for v1, v2 in zip(vec1_binary, vec2_binary))
    # Calculate magnitudes
    mag1 = math.sqrt(sum(v * v for v in vec1_binary))
    mag2 = math.sqrt(sum(v * v for v in vec2_binary))
    if mag1 * mag2 == 0:
        return 0.0
    return dot_product / (mag1 * mag2)
```

OUTPUT FILE

```
for hotel_idx, hotel_row in hotels_df.iterrows():
    item_id = hotel_row['itemid']
    hotel_id = hotel_row['hotelid']
    hotel_amenities = hotel_row['amenity_set']
    # Initialize row with item ID and hotel ID
    row_data = {'itemid': item_id, 'hotelid': hotel_id}
    # Calculate similarity with each user
    for user_idx, user_row in user_df.iterrows():
        user_id = user_idx + 1
        user_amenities = user_row['amenity_set']
        # Check if this hotel was visited by this user (any item from same h
        if hotel_id in visited_hotels_by_user[user_id]:
            similarity = "VISITED"
        else:
            similarity = cosine_similarity(user_amenities, hotel_amenities)
        row_data[f'user_{user_id}'] = similarity
    output_data.append(row_data)
# Create output dataframe
output_df = pd.DataFrame(output_data)
```

itemid	hotelid	user_1	user_2	user_3	user_4	user_5
1	31	0.739940073	0.680009255	0.726273039	0.717108788	0.797381532
2	31	0.754593775	0.693476093	0.74065608	0.731310341	0.777817459
3	31	0.792118034	0.696310624	0.742392339	0.740740741	0.78247589
4	31	0.840840992	0.620878016	0.665750286	0.717108788	0.797381532
5	31	0.595854023	0.402911482	0.412393049	0.5143445	0.56694671
6	31	0.595854023	0.402911482	0.412393049	0.5143445	0.56694671
7	31	0.595854023	0.402911482	0.412393049	0.5143445	0.56694671
8	31	0.595854023	0.402911482	0.412393049	0.5143445	0.56694671
9	31	0.595854023	0.402911482	0.412393049	0.5143445	0.56694671
10	31	0.306785996	0.202259959	0.207019668	0.25819889	0.237170825
11	1429	0.643119694	0.489955935	0.501485887	0.673575314	0.618718434
12	1429	0.568796459	0.363636364	0.372193684	0.522232968	0.533001791
13	1429	0.61993042	0.506024314	0.517932397	0.645975193	0.547722558
14	1429	0.595854023	0.523784927	0.536110964	0.66864785	0.56694671
15	1429	0.641688948	0.523784927	0.536110964	0.66864785	0.56694671
16	1429	0.306785996	0.202259959	0.207019668	0.25819889	0.237170825
17	1429	0.306785996	0.202259959	0.207019668	0.25819889	0.237170825
18	1520	0.687184271	0.533001791	0.545544726	0.725774739	0.75
19	1520	0.595854023	0.44320263	0.453632354	0.6172134	0.661437828
20	1520	0.665512265	0.548454318	0.561360891	0.700140042	0.600245048
21	2243	VISITED	0.709574875	0.726273039	0.830336492	0.797381532
22	2243	VISITED	0.587957294	0.601793477	0.794719414	0.77055175
23	2243	VISITED	0.351763235	0.36004115	0.513200239	0.530330086
24	2243	VISITED	0.639602149	0.654653671	0.816496581	0.708333333
25	2243	VISITED	0.246182982	0.251976315	0.314269681	0.288675135
26	2243	VISITED	0.246182982	0.251976315	0.314269681	0.288675135
27	2243	VISITED	0.246182982	0.251976315	0.314269681	0.288675135
28	2243	VISITED	0.246182982	0.251976315	0.314269681	0.288675135

OBJECTIVES

- **Recommend top 10 hotels per user based on similarity**
  - Set up file

```python
import pandas as pd
import io
import numpy as np

# --- Step 1: Load and Preprocess Hotel Data ---
hotel_data_url = 'Group4_Part1_preprocessed.csv'
try:
    df_hotels = pd.read_csv(hotel_data_url)
    df_hotels = df_hotels[['itemid', 'hotelname', 'hotelid']].rename(
        columns={'itemid': 'Item ID', 'hotelname': 'Hotel Name'}
    )
    df_hotels['Hotel ID'] = df_hotels['hotelid'].astype(str)
    df_hotels = df_hotels[['Item ID', 'Hotel Name', 'Hotel ID']]
    df_hotels = df_hotels.drop_duplicates(subset=['Item ID', 'Hotel Name', 'Hotel ID']).reset_index(drop=True)
except Exception as e:
    print(f"Error in Step 1: {e}")
    df_hotels = pd.DataFrame()
```

OBJECTIVES

- **Recommend top 10 hotels per user based on similarity**
  - Process Similarity data

```
# --- Step 2: Load and Process the Similarity Data ---
similarity_data_url = 'Group1_Part2_Model22.csv'
try:
    df_similarity = pd.read_csv(similarity_data_url)
except Exception as e:
    print(f"Error loading similarity data from URL: {e}")
    df_similarity = pd.DataFrame()

df_similarity = df_similarity.replace('VISITED', np.nan)

user_columns = ['user_1', 'user_2', 'user_3', 'user_4', 'user_5']
for col in user_columns:
    if col in df_similarity.columns:
        df_similarity[col] = pd.to_numeric(df_similarity[col], errors='coerce')
    else:
        print(f"Warning: Column '{col}' not found in similarity data.")
```

OBJECTIVES

- **Recommend top 10 hotels per user based on similarity**
  - Process Similarity data

```
df_melted_similarity = df_similarity.melt(
    id_vars=['itemid', 'hotelid'],
    value_vars=user_columns,
    var_name='User',
    value_name='Similarity'
)

df_melted_similarity.rename(columns={'itemid': 'Item ID', 'hotelid': 'Hotel ID_from_sim'}, inplace=True)
df_melted_similarity['User'] = df_melted_similarity['User'].str.replace('user_', 'User ').str.strip()

df_melted_similarity.dropna(subset=['Similarity'], inplace=True)
```

OBJECTIVES

- **Recommend top 10 hotels per user based on similarity**
  - Merger & Saving

```
# --- Step 3: Merging Data, Generating Top 10 Recommendations, and Saving ---
try:
    df_hotels
    df_melted_similarity
except NameError:
    print("Error: df_hotels or df_melted_similarity not defined. Ensure Steps 1 and 2 ran correctly.")
    df_hotels = pd.DataFrame(columns=['Item ID', 'Hotel Name', 'Hotel ID'])
    df_melted_similarity = pd.DataFrame(columns=['Item ID', 'User', 'Similarity'])

df_merged = pd.merge(
    df_melted_similarity,
    df_hotels,
    left_on='Item ID',
    right_on='Item ID',
    how='inner'
)

df_sorted = df_merged.sort_values(by=['User', 'Similarity'], ascending=[True, False])
```

OBJECTIVES

- **Recommend top 10 hotels per user based on similarity**
  - Merger & Saving

```
top_n = 10
df_top_recommendations = df_sorted.groupby('User').head(n=top_n).reset_index(drop=True)

df_top_recommendations.rename(columns={'User': 'User ID'}, inplace=True)

df_final_recommendations = df_top_recommendations[[
    'User ID',
    'Item ID',
    'Hotel ID',
    'Hotel Name',
    'Similarity'
]]

output_filename = 'Group4_Part2_Recommendation23.csv'
df_final_recommendations.to_csv(output_filename, index=False)

print(f"Process complete. Recommendations saved to '{output_filename}'.")
```

OBJECTIVES

- **Recommend top 10 hotels per user based on similarity**

User ID	Item ID	Hotel ID	Hotel Name	Similarity
User 1	140	90896	Hotel Altstadt Vienna	0.85813
User 1	4	31	The Harmonie Vienna BW Premier Collection	0.84084
User 1	83	70993	Star Inn Hotel Salzburg Airport-Messe by Comfort	0.84017
User 1	84	70993	Star Inn Hotel Salzburg Airport-Messe by Comfort	0.84017
User 1	531	381887	Landhotel Stockerwirt	0.84017
User 1	586	503028	Sporthotel Igls	0.84017
User 1	587	503028	Sporthotel Igls	0.84017
User 1	755	7229537	AMEDIA Hotel Linz	0.84017
User 1	456	261557	Alpenland Sporthotel St. Johann im Pongau	0.82512
User 1	202	129814	Sport- und Wellnesshotel Held	0.82319
User 2	747	6881583	Hotel Die Residenz Superior	0.85428
User 2	748	6881583	Hotel Die Residenz Superior	0.84092
User 2	749	6881583	Hotel Die Residenz Superior	0.84092

REFERENCE

- What are the pros and cons of using cosine similarity vs. Jaccard similarity for text analysis? : https://www.linkedin.com/advice/3/what-pros-cons-using-cosine-similarity-vs
- HOTEL_OUTDATASET.CSV : Class Material/miniproject 1
- RS_MiniProject1.pdf: Class Material/miniproject 1
- UserData.csv: Class Material/miniproject 1
- https://youtu.be/zcUGLp5vwaQ?si=UApTSlgeWfT7hRVS

REFERENCE

- Pros-and-cons-of-aggregation-methods :https://www.researchgate.net/figure/Pros-and-cons-of-aggregation-methods_tbl3_292643361

- An efficient recommendation generation using relevant Jaccard similarity : https://www.researchgate.net/publication/330171725_An_efficient_recommendation_generation_using_relevant_Jaccard_similarity


THANK YOU