# Milestone -3

Dataset Preparation

The objective is to preprocess a hotel reviews dataset, add user-related information, and classify reviews according to certain preference categories, such as Wellness, Entertainment, Dining, and Social Activities.

**Steps Performed:**

The initial dataset contains reviews and rating with 20000 entries

**1. Adding Fake User Information :** A new column `reviewed\_by` was added to represent the name of the user who reviewed the hotel.

- Each row generated random fake names with the usage of the Faker library.

**2. Random Unique User E-mail :** Added a new column to this dataset: "user\_email," where each of the users had different unique email addresses for them.

**3. Adding Preference Columns**

New columns added: `Wellness`, `Entertainment`, `Dining`, `Social Activities`.

The columns were initialized to `None` values to later be populated as preferences from the reviews themselves.

**4. Defining Preference Keywords**

- A dictionary of keywords was defined for each preference category. For example:

- Wellness: Keywords like "spa," "massage," and "fitness."

- Entertainment: Keywords like "live music," "DJ," and "karaoke."

- Dining: Keywords like "restaurant," "bar," and "breakfast."

- Social Activities: words including "live music," "social events, "and "group activities."

**5 . Preference-Based Classifications of Reviews**

For each review in the database, look for the selected words.

- If a keyword for a particular category was present in the review, the corresponding column was filled in with a specific description. For example, if "spa" was found, then "spa treatments" was updated in the column. If no specific keyword was found, the column was marked as "Not Specified."

**6. Saving the Enhanced Dataset**

- The updated dataset with the `reviewed\_by`, `user\_email`, and preference columns was saved to a new CSV file called `**mydataset.csv**` for further analysis.

Recommendations System

The Recommendations System is a web-based system for hotel guests. The system generates personal activity recommendations from user preferences and past behaviors, incorporates mechanisms to collect user feedback, conduct sentiment analysis, and notify the management to resolve complaints. Here is the rewritten text, including the logic modification for sentiment analysis as well as the integration of this logic with the rest of the text:

**Implementation Logic for the Recommendations System**

**1. Personalized Recommendations**

- User Profile Matching:

Guest preferences (categories of interest) and historical activity data (e.g., ratings, time spent) are stored in a database.

- Similarity between users is calculated using the cosine similarity algorithm. This identifies users with overlapping preferences and behaviours.

- Logic:

- Each user is represented as a vector of activity preferences and ratings.

- Cosine similarity is computed as:

**[Similarity(A, B) = dot(A, B) / ||A|| ||B|| ]**

- Activities highly rated or frequently engaged in by similar users are recommended.

- Output:

- A ranked list of activities based on the aggregated ratings and time spent by similar users.

**2. Submission of Feedback and Sentiment Analysis**

- Feedback Collection:

Guests fill a form that submits their feedback to the database.

- Sentiment Analysis Logic:

Feedback text is passed through sentiment analysis using the OpenAI API.

Implementation Details:

- The `analyze\_sentiment` function will pass the text from feedback to the API to determine if the feedback is positive, negative, or neutral.

A retry mechanism has been added to use the `tenacity` library, to deal with a failed call. This can attempt up to 3 times, using exponential backoff.

- Introduce a delay between requests to avoid over-API rate limits.

- Logic:

- In GUEST Feedback text

- Process

- The system sends a chat completion request to the OpenAI API with a prompt tailored for sentiment analysis.

- The API returns a sentiment label, such as "positive," "neutral," or "negative."

- Output: The sentiment label is stored along with the feedback for further analysis.

- Category Detection:

- Feed back text is pre-processed through keyword extraction or Named Entity Recognition (NER)

- A predefined keyword is mapped with phrases that belongs to activity category; for instance, "spa" or "massage" → "Wellness".

- Logic:

- Tokenize the feed back text.

- Map tokens or phrases to categories using a dictionary or a machine-learning classifier.

- Assign the detected categories to the feedback record.

**3. Notification Mechanisms**

Slack Alerts:

Upon the event of detection of negative feedback, a webhook sends a notification to Slack, which includes:

- Feedback text.

- Sentiment label.

- Detected categories.

- Email Alerts:

- Email notifications are generated using SMTP or third-party APIs (e.g., SendGrid).

- Emails include:

- Guest name and feedback.

- Results of sentiment analysis

- Recommended action according to the feedback

- Logic for raising alerts:

- When the sentiment is "negative":

- Send Slack notification

- Draft and send an email.

**4. Recommendation Algorithm**

- *Step 1: Pull Guest Data*

- Retrieve guest's preferences and activity history from the database.

- *Step 2: Pull Similar User Data*

- Employ the cosine similarity algorithm to pull up users that share similar preferences.

*- Step 3: Aggregate Data*

- For every activity, compute

- Average rating by similar users.

- Average time spent by similar users.

- *Step 4: Rank and Filter*

- Rank activities by aggregated ratings and engagement.

- Filter out activities the guest has already experienced.

*- Step 5: Display Recommendations*

- Rank the activities and format into a user-friendly display (e.g., cards or list).

- `feedback\_id`, `user\_id`, `text`, `sentiment\_label`, `categories`.

User Interface:  
  
A screenshot of a hotel

Description automatically generated

Slack Notification:

A text on a white background

Description automatically generated

Email Notification:

A close-up of a text

Description automatically generated