# --- Core Imports ---

import pandas as pd

import re

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

from IPython.display import clear\_output

# --- Data Loading & Preprocessing ---

# Load datasets

users = pd.read\_csv('users.csv')

meals = pd.read\_csv('meal\_data(new).csv')

# Handle missing values

meals.fillna({'Common\_Allergens': 'None', 'Diet\_Preference': 'None'}, inplace=True)

users.fillna({'Allergens': 'None', 'Diet\_Preference': 'None', 'Health\_Restriction': 'None'}, inplace=True)

# Create meal tags for TF-IDF

meals['tags'] = meals['Diet\_Preference'] + ' ' + meals['food\_category'] + ' ' + meals['meal\_types']

# --- Recommendation Engine Setup ---

tfidf = TfidfVectorizer(stop\_words='english', lowercase=True)

tfidf\_matrix = tfidf.fit\_transform(meals['tags'])

def recommend\_meals(user\_id, n=5):

"""Core recommendation logic with safety filters"""

try:

user = users[users['UserID'] == user\_id].iloc[0]

except IndexError:

return pd.DataFrame()

filtered = meals.copy()

# Allergen filtering with regex boundaries

user\_allergens = [a.strip().lower() for a in user['Allergens'].split(', ') if a.strip().lower() != 'none']

if user\_allergens:

pattern = r'\b(' + '|'.join(re.escape(allergen) for allergen in user\_allergens) + r')\b'

filtered = filtered[~filtered['Common\_Allergens'].str.lower().str.contains(

pattern, na=False, regex=True

)]

# Health restrictions

health\_map = {'Heart': 'Heart\_Healthy', 'Diabetic': 'Diabetic\_Friendly'}

if user['Health\_Restriction'] != 'None':

health\_col = health\_map.get(user['Health\_Restriction'], user['Health\_Restriction'])

filtered = filtered[filtered[health\_col] == 1]

# Diet preferences

user\_diets = [d.strip().lower() for d in str(user['Diet\_Preference']).split(',') if d.strip().lower() != 'none']

if user\_diets:

filtered = filtered[filtered['Diet\_Preference'].str.lower().str.contains(

'|'.join(user\_diets), na=False, regex=False

)]

if filtered.empty:

return filtered

# Ranking system

if user\_diets:

user\_profile = ' '.join(user\_diets)

user\_vector = tfidf.transform([user\_profile])

cos\_sim = cosine\_similarity(user\_vector, tfidf\_matrix[filtered.index])

filtered['similarity'] = cos\_sim[0]

else:

filtered['similarity'] = 0 # Default for non-diet users

return filtered.sort\_values(

by=['similarity', 'Protein (g)'],

ascending=[False, False]

).head(n)

# --- Interactive Recommendation System ---

class MealRecommender:

def \_\_init\_\_(self):

self.sessions = {}

def \_get\_full\_recommendations(self, user\_id):

"""Get complete sorted meal list for user"""

recs = recommend\_meals(user\_id, n=len(meals))

return recs.reset\_index(drop=True)

def start\_session(self, user\_id):

"""Initialize new recommendation session"""

full\_recs = self.\_get\_full\_recommendations(user\_id)

self.sessions[user\_id] = {

'all\_meals': full\_recs,

'current\_pos': 0,

'shown': []

}

return f"Session started for user {user\_id} - {len(full\_recs)} meals available"

def get\_meal(self, user\_id, position=None):

"""Get specific meal from recommendations"""

if user\_id not in self.sessions:

self.start\_session(user\_id)

session = self.sessions[user\_id]

meals = session['all\_meals']

if meals.empty:

return None, "No meals available"

# Handle position requests

if position is None:

position = session['current\_pos']

else:

position = max(0, min(position, len(meals)-1))

meal = meals.iloc[position]

session['current\_pos'] = position

# Track shown meals

if position not in session['shown']:

session['shown'].append(position)

return meal, position+1 # Return 1-based index

def next\_meal(self, user\_id, step=1):

"""Move through recommendation list"""

session = self.sessions[user\_id]

new\_pos = (session['current\_pos'] + step) % len(session['all\_meals'])

return self.get\_meal(user\_id, new\_pos)

def interactive\_session(self, user\_id):

"""Command-line interface for meal browsing"""

self.start\_session(user\_id)

session = self.sessions[user\_id]

if session['all\_meals'].empty:

print("⚠️ No meals match your dietary requirements")

return

while True:

clear\_output(wait=True)

meal, pos = self.get\_meal(user\_id)

# Display current meal

print(f" Recommendation #{pos}/{len(session['all\_meals'])} ".center(40, '='))

print(f"🍽 {meal['FoodName']}")

print(f"🔹 Calories: {meal['Calories']}")

print(f"🔹 Protein: {meal['Protein (g)']}g")

print(f"🔹 Allergens: {meal['Common\_Allergens'] or 'None'}")

print(f"🔹 Diets: {meal['Diet\_Preference']}")

print(f"🔹 Similarity Score: {meal.get('similarity', 0):.2f}")

# Navigation controls

print("\nControls:")

print("1. Next meal ➡")

print("2. Previous meal ⬅")

print("3. Random new meal 🎲")

print("4. Restart session 🔄")

print("5. Exit ❌")

choice = input("\nYour choice (1-5): ").strip()

if choice == '1':

self.next\_meal(user\_id, 1)

elif choice == '2':

self.next\_meal(user\_id, -1)

elif choice == '3':

self.next\_meal(user\_id, len(session['all\_meals'])//2) # Example random jump

elif choice == '4':

self.start\_session(user\_id)

elif choice == '5':

print("👋 Ending recommendation session")

break

else:

print("⚠️ Invalid input, please try again")

# --- Usage Example ---

if \_\_name\_\_ == "\_\_main\_\_":

# Initialize recommender system

recommender = MealRecommender()

# Start interactive session for user 16

recommender.interactive\_session(16)