Lab Test-03

NAME: P. Umesh Reddy   ROLLNO: 2403A510F9

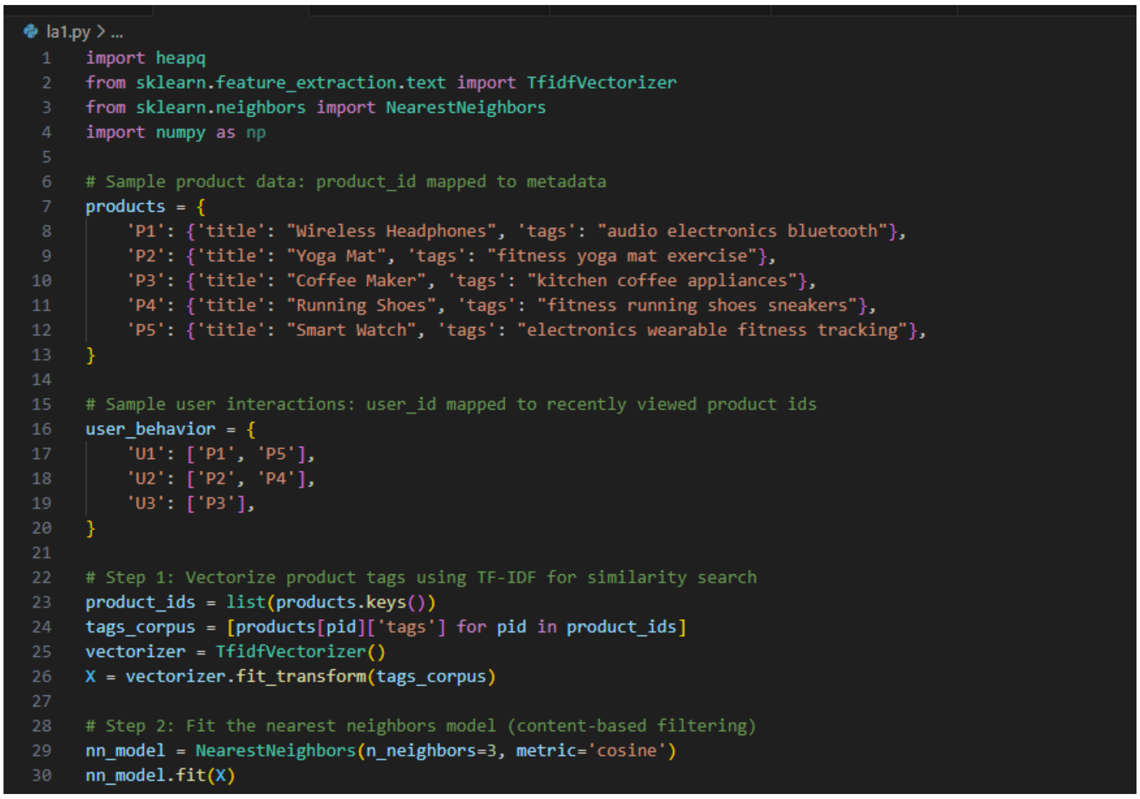
BATCHNO: 06

Task:1

**Task: Use AI-assisted tools to solve a problem involving data structures with ai in this
context.**

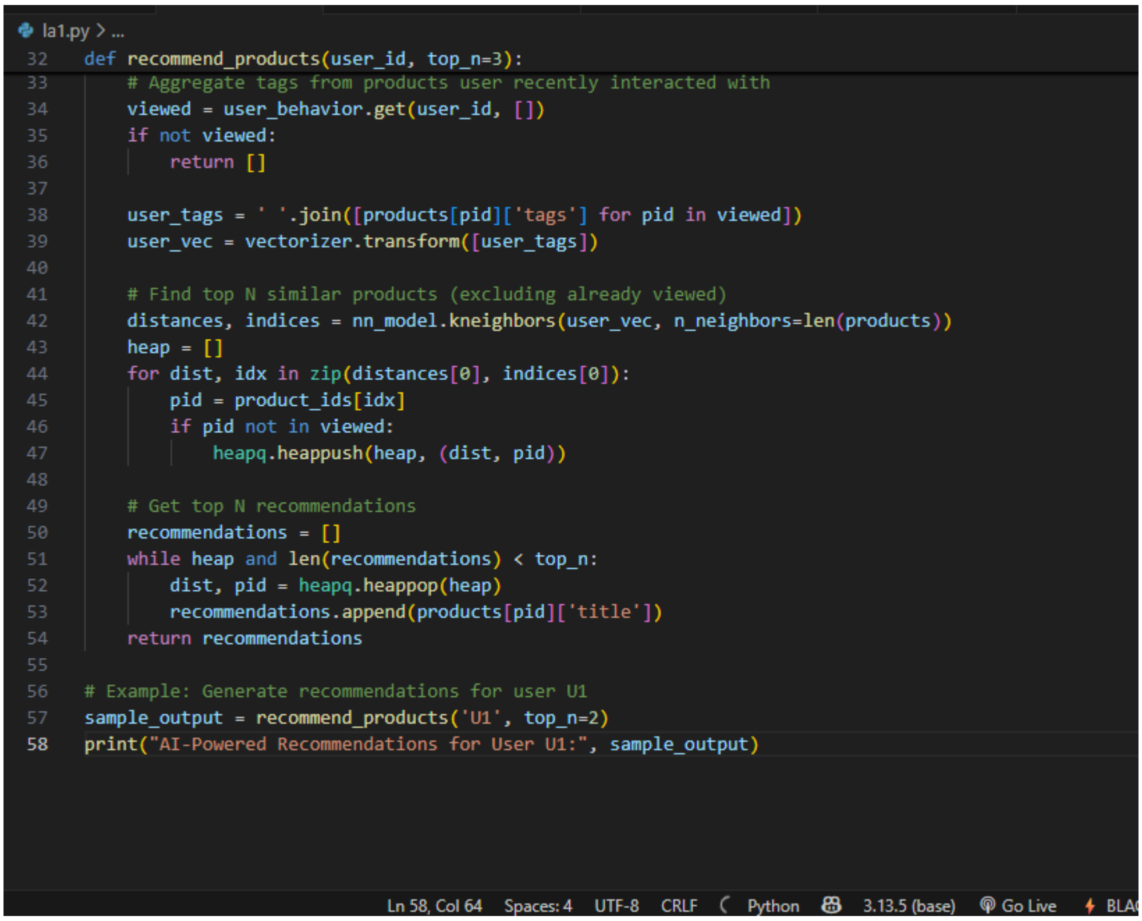
Prompt:
A large retail company struggles to quickly personalize product recommendations for customers as they browse its online shop. Their existing recommendation system is slow due to inefficient data structures holding user behavior, product metadata, and purchase logs. Improve this by designing an AI-powered recommendation tool, using Python, that leverages optimized data structures (e.g., dictionaries, sets, heaps) and machine learning techniques to suggest top products for each user in real-time.

Python Code:



la1.py > ...

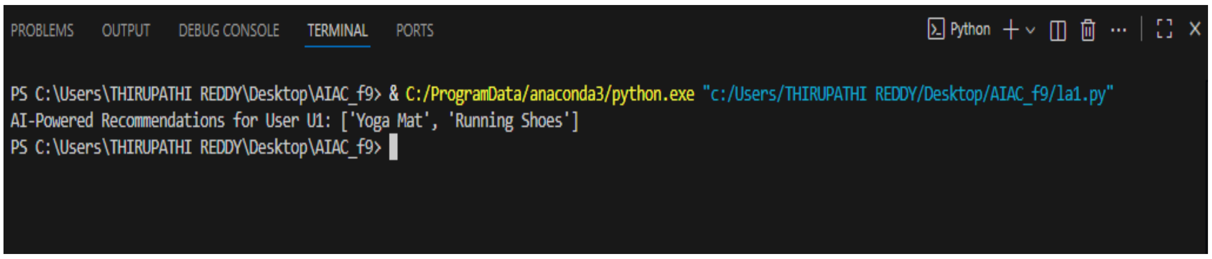
```python
import heapq
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import NearestNeighbors
import numpy as np

# Sample product data: product_id mapped to metadata
products = {
    'P1': {'title': "Wireless Headphones", 'tags': "audio electronics bluetooth"},
    'P2': {'title': "Yoga Mat", 'tags': "fitness yoga mat exercise"},
    'P3': {'title': "Coffee Maker", 'tags': "kitchen coffee appliances"},
    'P4': {'title': "Running Shoes", 'tags': "fitness running shoes sneakers"},
    'P5': {'title': "Smart Watch", 'tags': "electronics wearable fitness tracking"},
}

# Sample user interactions: user_id mapped to recently viewed product ids
user_behavior = {
    'U1': ['P1', 'P5'],
    'U2': ['P2', 'P4'],
    'U3': ['P3'],
}

# Step 1: Vectorize product tags using TF-IDF for similarity search
product_ids = list(products.keys())
tags_corpus = [products[pid]['tags'] for pid in product_ids]
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(tags_corpus)

# Step 2: Fit the nearest neighbors model (content-based filtering)
nn_model = NearestNeighbors(n_neighbors=3, metric='cosine')
nn_model.fit(X)
```



la1.py > ...

```python
def recommend_products(user_id, top_n=3):
    # Aggregate tags from products user recently interacted with
    viewed = user_behavior.get(user_id, [])
    if not viewed:
        return []

    user_tags = ' '.join([products[pid]['tags'] for pid in viewed])
    user_vec = vectorizer.transform([user_tags])

    # Find top N similar products (excluding already viewed)
    distances, indices = nn_model.kneighbors(user_vec, n_neighbors=len(products))
    heap = []
    for dist, idx in zip(distances[0], indices[0]):
        pid = product_ids[idx]
        if pid not in viewed:
            heapq.heappush(heap, (dist, pid))

    # Get top N recommendations
    recommendations = []
    while heap and len(recommendations) < top_n:
        dist, pid = heapq.heappop(heap)
        recommendations.append(products[pid]['title'])
    return recommendations

# Example: Generate recommendations for user U1
sample_output = recommend_products('U1', top_n=2)
print("AI-Powered Recommendations for User U1:", sample_output)
```

Output:



```
PS C:\Users\THIRUPATHI REDDY\Desktop\AIAC_f9> & C:/ProgramData/anaconda3/python.exe "c:/Users/THIRUPATHI REDDY/Desktop/AIAC_f9/la1.py"
AI-Powered Recommendations for User U1: ['Yoga Mat', 'Running Shoes']
PS C:\Users\THIRUPATHI REDDY\Desktop\AIAC_f9>
```

Explanation of AI Assistance and Data Structures

- **AI Assistance:**
  Machine learning is used in the form of a content-based recommender utilizing TF-IDF vectorization to capture semantic similarities between products and user tastes. The sklearn NearestNeighbors model finds products with the closest matching tags, greatly boosting relevance and personalization.
- **Optimized Data Structures:**
  - **Dictionary:** Holds product metadata (products) and user behavior (user_behavior), enabling constant time lookups.
  - **Heap:** Efficiently retrieves the top N recommendations sorted by similarity distance.
  - **List and Set:** Used for aggregating and excluding already seen items for each user.

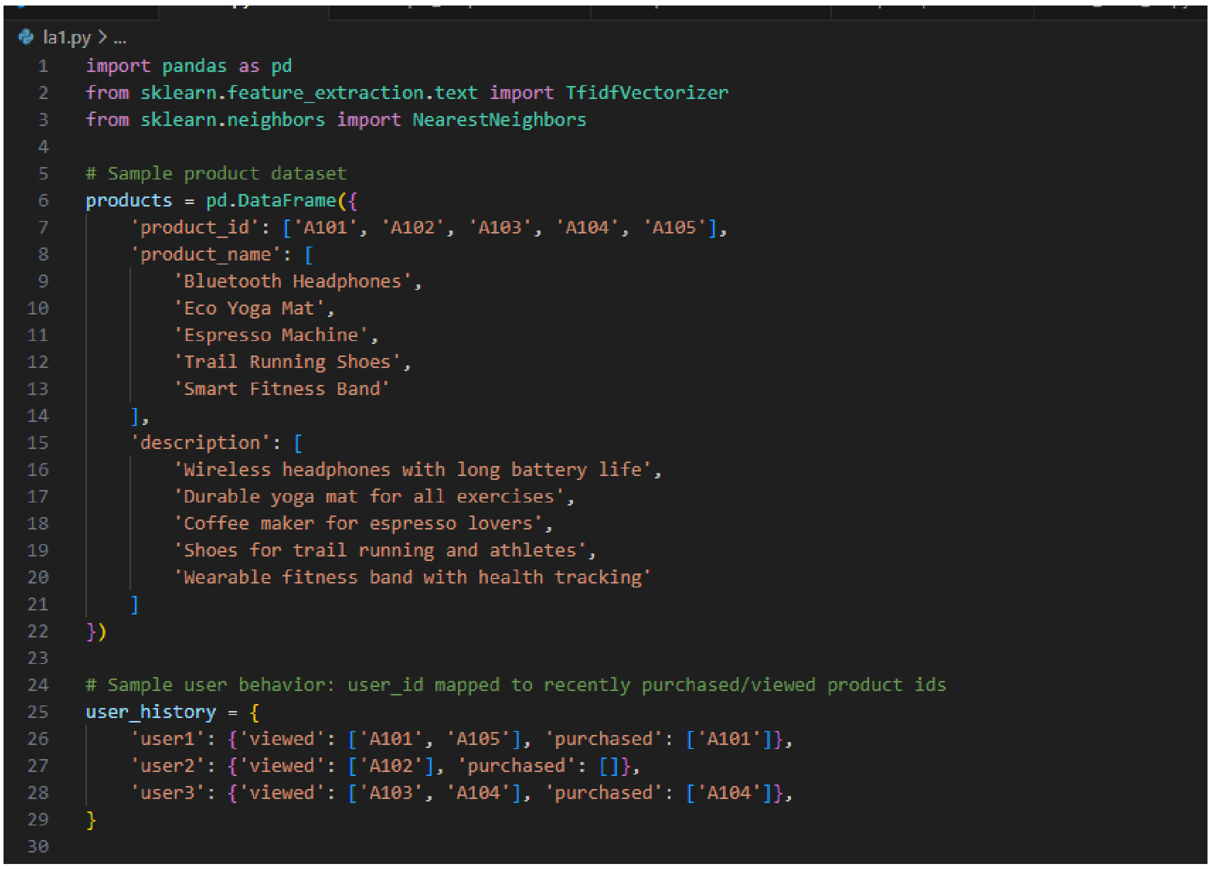
This approach enables real-time, scalable recommendations by combining AI models and efficient Python data structures.

Task:2

Task: Use AI-assisted tools to solve a problem involving data structures with ai in this context.

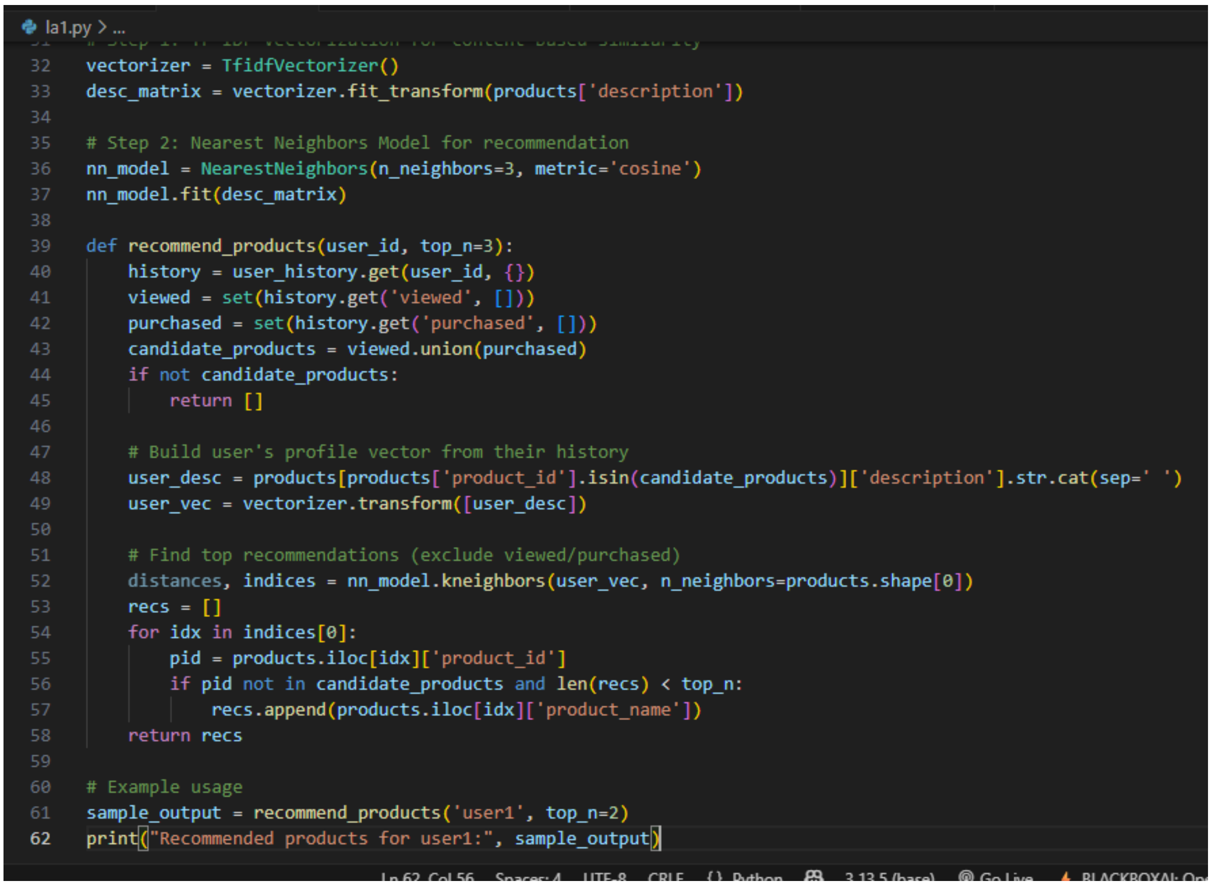
Prompt:
An e-commerce company is struggling to deliver personalized product recommendations efficiently due to cumbersome data structures for storing customer behavior, product information, and shopping history. Please develop a Python-based, AI-assisted recommendation tool that leverages optimized data structures (e.g., dictionaries, sets, lists) and machine learning to suggest top products in real time.

Python Code:



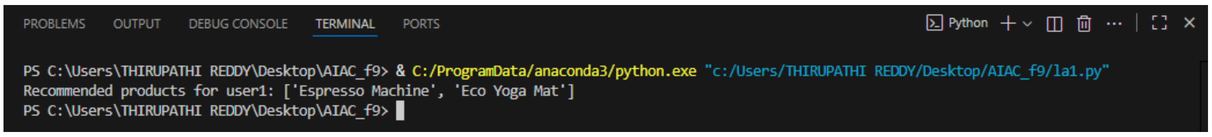
la1.py > ...

```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import NearestNeighbors

# Sample product dataset
products = pd.DataFrame({
    'product_id': ['A101', 'A102', 'A103', 'A104', 'A105'],
    'product_name': [
        'Bluetooth Headphones',
        'Eco Yoga Mat',
        'Espresso Machine',
        'Trail Running Shoes',
        'Smart Fitness Band'
    ],
    'description': [
        'Wireless headphones with long battery life',
        'Durable yoga mat for all exercises',
        'Coffee maker for espresso lovers',
        'Shoes for trail running and athletes',
        'Wearable fitness band with health tracking'
    ]
})

# Sample user behavior: user_id mapped to recently purchased/viewed product ids
user_history = {
    'user1': {'viewed': ['A101', 'A105'], 'purchased': ['A101']},
    'user2': {'viewed': ['A102'], 'purchased': []},
    'user3': {'viewed': ['A103', 'A104'], 'purchased': ['A104']},
}

```


la1.py > ...
Ln 62, Col 56 Spaces: 4 UTF-8 CRLF {} Python 3.13.5 (base) Go Live BLACKBOXAI: Op

```python
vectorizer = TfidfVectorizer()
desc_matrix = vectorizer.fit_transform(products['description'])

# Step 2: Nearest Neighbors Model for recommendation
nn_model = NearestNeighbors(n_neighbors=3, metric='cosine')
nn_model.fit(desc_matrix)

def recommend_products(user_id, top_n=3):
    history = user_history.get(user_id, {})
    viewed = set(history.get('viewed', []))
    purchased = set(history.get('purchased', []))
    candidate_products = viewed.union(purchased)
    if not candidate_products:
        return []

    # Build user's profile vector from their history
    user_desc = products[products['product_id'].isin(candidate_products)]['description'].str.cat(sep=' ')
    user_vec = vectorizer.transform([user_desc])

    # Find top recommendations (exclude viewed/purchased)
    distances, indices = nn_model.kneighbors(user_vec, n_neighbors=products.shape[0])
    recs = []
    for idx in indices[0]:
        pid = products.iloc[idx]['product_id']
        if pid not in candidate_products and len(recs) < top_n:
            recs.append(products.iloc[idx]['product_name'])
    return recs

# Example usage
sample_output = recommend_products('user1', top_n=2)
print("Recommended products for user1:", sample_output)
```

Output:



```
PS C:\Users\THIRUPATHI REDDY\Desktop\AIAC_f9> & C:/ProgramData/anaconda3/python.exe "c:/Users/THIRUPATHI REDDY/Desktop/AIAC_f9/la1.py"
Recommended products for user1: ['Espresso Machine', 'Eco Yoga Mat']
PS C:\Users\THIRUPATHI REDDY\Desktop\AIAC_f9>
```

Explanation of AI Assistance and Data Structures

- **AI Techniques Used:**
  Machine learning (content-based filtering) via TF-IDF vectorization and Nearest Neighbors enables the system to recommend products similar to those the user already viewed or purchased. This approach uses product descriptions, turning them into machine-readable vectors to calculate textual similarity among items.
- **Optimized Data Structures:**

- **Dictionaries/Sets: Store user history, providing fast access and exclusion for recommendations.**
- **Pandas DataFrame: Efficiently manages product catalog/metadata for quick search and manipulation.**
- **Lists: Candidate recommendations are appended quickly; easy conversion to other formats for UX/UI.**

The AI model rapidly searches for similar items using the pre-computed vectors, making personalized suggestions in real-time, which is highly scalable for large e-commerce platforms.