AI ASSISTED CODING

LAB TEST - 03

NAME : B.SRISHANTH ROLL NO : 2403A510G3

BATCH : 06

Q1:

PROMPT :

A retail company wants to recommend similar products to customers based on what they are currently viewing.

Design an algorithm (with AI assistance) that can automatically recommend similar items based on product descriptions.

CODE :

task1.py > ...

```python
# AI-assisted Product Recommendation System (Retail Sector)

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd

# ----------------------------
# Sample Product Dataset
# ----------------------------
data = {
    'Product_ID': [1, 2, 3, 4, 5],
    'Product_Name': [
        'Red Cotton T-Shirt',
        'Blue Denim Jeans',
        'Cotton Casual Shirt',
        'Leather Wallet',
        'Formal Black Shoes'
    ],
    'Description': [
        'Comfortable red cotton t-shirt for daily wear',
        'Stylish blue denim jeans for men',
        'Soft cotton shirt perfect for casual outings',
        'Premium leather wallet with multiple card slots',
        'Elegant black shoes for formal occasions'
    ]
}

# Create DataFrame
df = pd.DataFrame(data)

# ----------------------------
# Step 1: Convert text into TF-IDF vectors
# ----------------------------
vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = vectorizer.fit_transform(df['Description'])

# ----------------------------
```

task1.py > ...

```python
df = pd.DataFrame(data)

# ----------------------------
# Step 1: Convert text into TF-IDF vectors
# ----------------------------
vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = vectorizer.fit_transform(df['Description'])

# ----------------------------
# Step 2: Compute cosine similarity
# ----------------------------
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)

# ----------------------------
# Step 3: Recommend similar products
# ----------------------------
def recommend(product_name):
    idx = df[df['Product_Name'] == product_name].index[0]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:3]  # Top 2 similar products
    product_indices = [i[0] for i in sim_scores]
    return df[['Product_ID', 'Product_Name']].iloc[product_indices]

# ----------------------------
# Step 4: Test the recommendation system
# ----------------------------
print("Product Recommendations for 'Red Cotton T-Shirt':")
print(recommend('Red Cotton T-Shirt'))
```

OUTPUT :

```
Lotion - Predicted Next Stock: 72.00
PS C:\Users\keerthi priya\Desktop\ai lab test - 3> & "C:/Users/keerthi priya/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/keerthi priya/D
esktop/ai lab test - 3/task1.py"
Product Recommendations for 'Red Cotton T-Shirt':
   Product_ID        Product_Name
2           3  Cotton Casual Shirt
1           2     Blue Denim Jeans
PS C:\Users\keerthi priya\Desktop\ai lab test - 3>
```

OBSERVATION :

1. AI model converts text into numerical vectors (embeddings).
2. Cosine similarity measures closeness between products.
3. Similar items (like "headphones") score higher for earphones query.
4. AI assistance improves accuracy without manual rule creation.
5. This algorithm can scale for large product catalogs easily.

Q2.

PROMPT :

An education company wants to identify top-performing students efficiently from large data using **data structures.**
AI is used to analyze patterns and predict performance trends

Generate a Python program that uses data structures (like heap and dictionary) to analyze student scores and predict future marks using AI regression model.

CODE :

task2.py > ...

```python
# AI-Assisted Student Performance Analysis using Data Structures

import heapq
from sklearn.linear_model import LinearRegression
import numpy as np

# Step 1: Store student data using dictionary
students = {
    "Amit": [80, 85, 90],
    "Priya": [92, 89, 95],
    "Ravi": [78, 74, 70],
    "Keerthi": [88, 91, 84],
    "Rahul": [65, 70, 68]
}

# Step 2: Compute average marks and store in a list
averages = []
for name, marks in students.items():
    avg = sum(marks) / len(marks)
    averages.append((avg, name))

# Step 3: Use Heap (data structure) to find top 3 students
top_students = heapq.nlargest(3, averages)

print("Top 3 Students:")
for avg, name in top_students:
    print(f"{name} - Average Marks: {avg}")

# Step 4: AI Model (Linear Regression) to predict next exam marks
print("\nAI Predicted Performance:")
for name, marks in students.items():
    X = np.arange(len(marks)).reshape(-1, 1)
    y = np.array(marks)
    model = LinearRegression()
    model.fit(X, y)
    next_score = model.predict([[len(marks)]])[0]
    print(f"{name} - Predicted Next Score: {next_score:.2f}")
```

Ln

OUTPUT :

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                                    Python

PS C:\Users\keerthi priya\Desktop\ai lab test - 3> & "C:/Users/keerthi priya/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/keerthi priya/Desktop/ai lab test - 3/task2.py"
Top 3 Students:
Priya - Average Marks: 92.0
Keerthi - Average Marks: 87.66666666666667
Amit - Average Marks: 85.0

AI Predicted Performance:
Amit - Predicted Next Score: 95.00
Priya - Predicted Next Score: 95.00
Ravi - Predicted Next Score: 66.00
Keerthi - Predicted Next Score: 83.67
Rahul - Predicted Next Score: 70.67
PS C:\Users\keerthi priya\Desktop\ai lab test - 3>
```

OBSERVATION :

1. Student records are stored efficiently using a dictionary data structure.
2. Heap data structure is used to quickly retrieve top-performing students.
3. AI regression model predicts student's next score automatically.
4. Data structures improve efficiency in handling multiple student records.
5. AI helps discover patterns and future performance trends.
6. The combination of heap and regression gives both ranking and prediction.
7. The algorithm can be scaled for thousands of students easily.
8. Useful for institutions to identify high achievers and students needing help.
9. Reduces manual effort in analyzing large performance datasets.