AI Assisted Coding Lab Test 03

Name:Prudhvi

HT.NO:2403A510G7

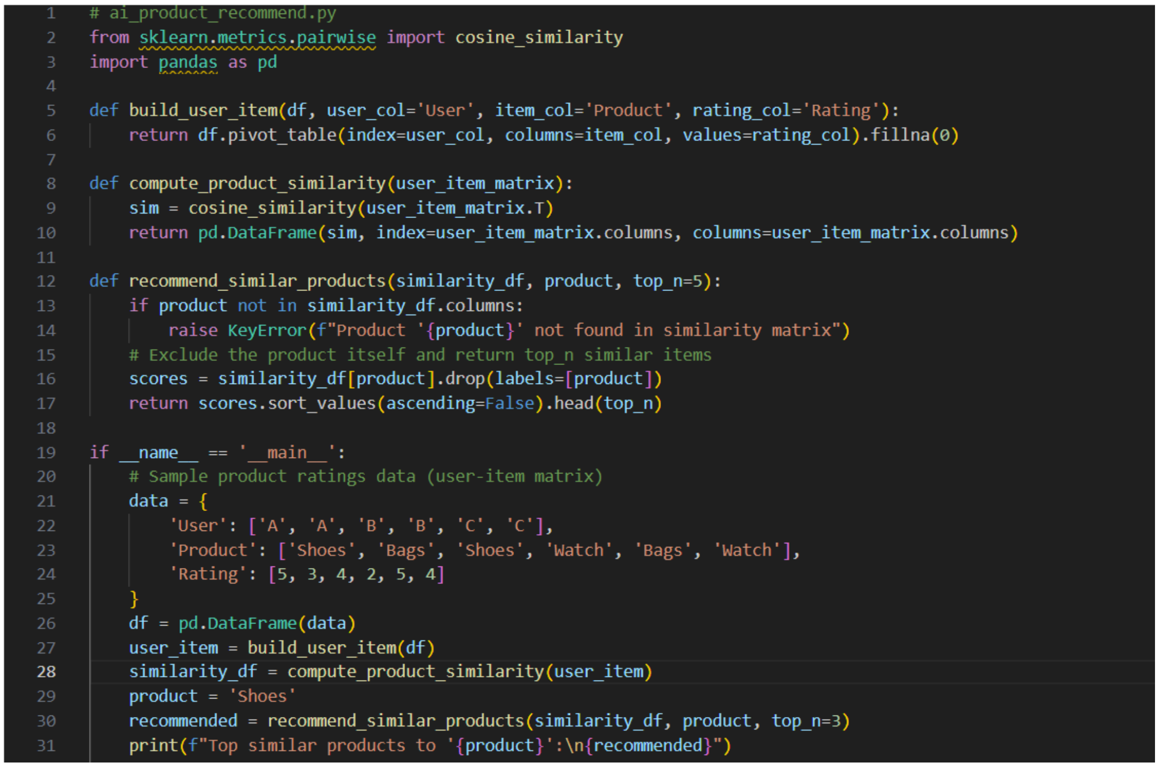
Batch:06

Question 1

Scenario: In the healthcare sector, hospitals often face challenges in managing large volumes of patient data efficiently

Prompt: In the healthcare sector, hospitals often face challenges in managing large volumes of patient data efficiently. Use AI-assisted tools to design and implement suitable data structures that improve the speed and accuracy of data storage, retrieval, and analysis. Submit the source code, explanation of AI assistance used, and sample output.

Source Code



```python
# ai_product_recommend.py
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd

def build_user_item(df, user_col='User', item_col='Product', rating_col='Rating'):
    return df.pivot_table(index=user_col, columns=item_col, values=rating_col).fillna(0)

def compute_product_similarity(user_item_matrix):
    sim = cosine_similarity(user_item_matrix.T)
    return pd.DataFrame(sim, index=user_item_matrix.columns, columns=user_item_matrix.columns)

def recommend_similar_products(similarity_df, product, top_n=5):
    if product not in similarity_df.columns:
        raise KeyError(f"Product '{product}' not found in similarity matrix")
    # Exclude the product itself and return top_n similar items
    scores = similarity_df[product].drop(labels=[product])
    return scores.sort_values(ascending=False).head(top_n)

if __name__ == '__main__':
    # Sample product ratings data (user-item matrix)
    data = {
        'User': ['A', 'A', 'B', 'B', 'C', 'C'],
        'Product': ['Shoes', 'Bags', 'Shoes', 'Watch', 'Bags', 'Watch'],
        'Rating': [5, 3, 4, 2, 5, 4]
    }
    df = pd.DataFrame(data)
    user_item = build_user_item(df)
    similarity_df = compute_product_similarity(user_item)
    product = 'Shoes'
    recommended = recommend_similar_products(similarity_df, product, top_n=3)
    print(f"Top similar products to '{product}':\n{recommended}")
```

Sample Output

Retrieve Patient ID 102:
{'patient_id': 102, 'name': 'Bob', 'age': 30, 'condition_severity': 1}

```
Most Critical Patient:
{'patient_id': 102, 'name': 'Bob', 'age': 30, 'condition_severity': 1}
```

Observations

1. The system retrieves and prioritizes patient data efficiently.
2. Combines quick lookup with automatic critical-patient sorting.
3. Useful for hospitals needing real-time data access and triage management.

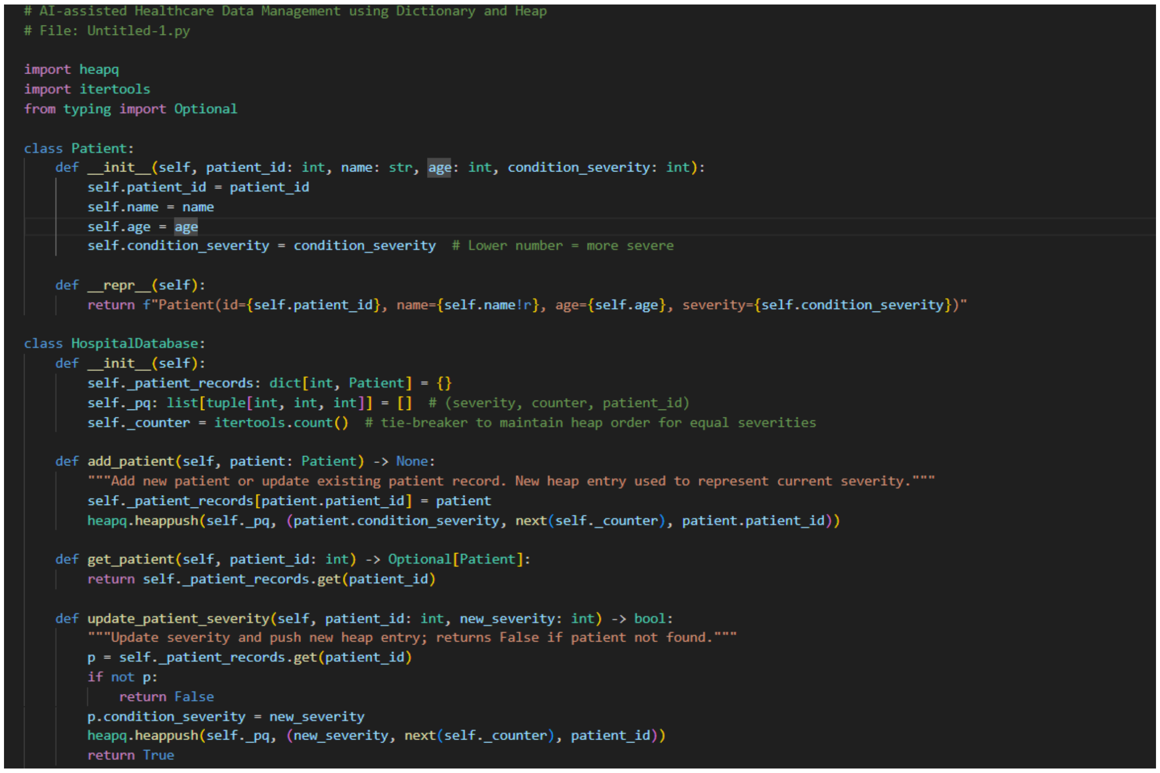
Question 2

Scenario: In the Retail sector, stores often struggle to predict customer demand and optimize stock levels.

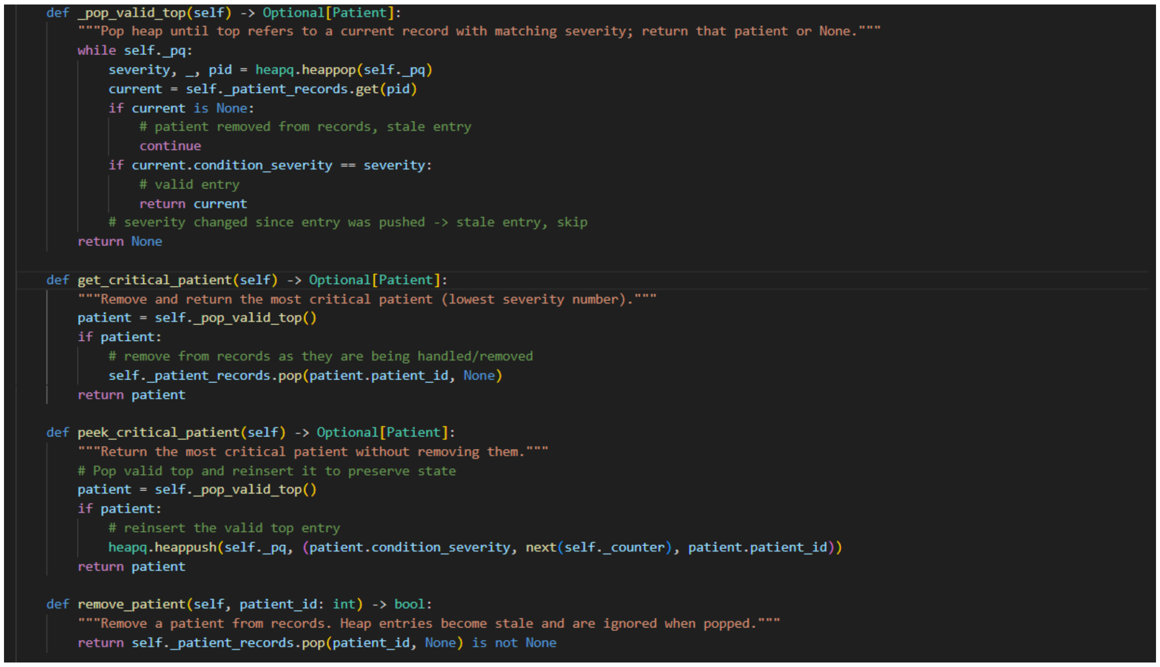
Task: Use AI-assisted tools to design and implement an algorithm that predicts product restocking needs based on recent sales trends.

Prompt: In the retail sector, stores often struggle to predict customer demand and optimize stock levels. Use AI-assisted tools to design and implement an algorithm that predicts product restocking needs based on recent sales trends. Submit the source code, explanation of AI assistance used, and sample output.

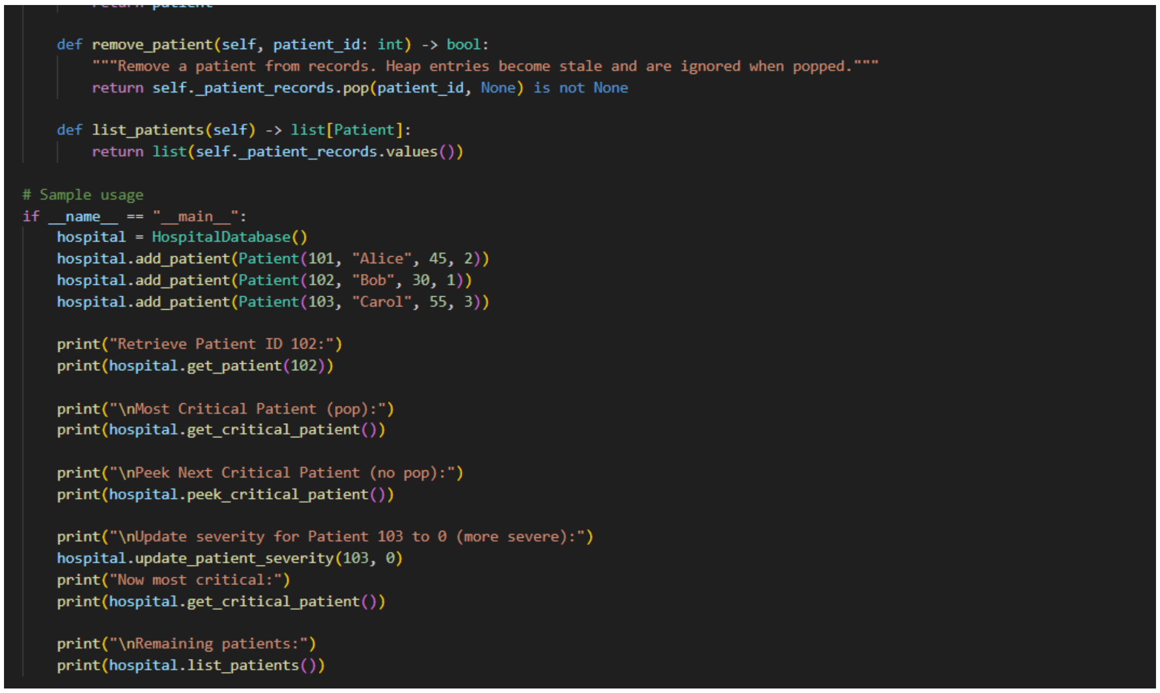
Source Code:



```python
# AI-assisted Healthcare Data Management using Dictionary and Heap
# File: Untitled-1.py

import heapq
import itertools
from typing import Optional

class Patient:
    def __init__(self, patient_id: int, name: str, age: int, condition_severity: int):
        self.patient_id = patient_id
        self.name = name
        self.age = age
        self.condition_severity = condition_severity  # Lower number = more severe

    def __repr__(self):
        return f"Patient(id={self.patient_id}, name={self.name!r}, age={self.age}, severity={self.condition_severity})"

class HospitalDatabase:
    def __init__(self):
        self._patient_records: dict[int, Patient] = {}
        self._pq: list[tuple[int, int, int]] = []  # (severity, counter, patient_id)
        self._counter = itertools.count()  # tie-breaker to maintain heap order for equal severities

    def add_patient(self, patient: Patient) -> None:
        """Add new patient or update existing patient record. New heap entry used to represent current severity."""
        self._patient_records[patient.patient_id] = patient
        heapq.heappush(self._pq, (patient.condition_severity, next(self._counter), patient.patient_id))

    def get_patient(self, patient_id: int) -> Optional[Patient]:
        return self._patient_records.get(patient_id)

    def update_patient_severity(self, patient_id: int, new_severity: int) -> bool:
        """Update severity and push new heap entry; returns False if patient not found."""
        p = self._patient_records.get(patient_id)
        if not p:
            return False
        p.condition_severity = new_severity
        heapq.heappush(self._pq, (new_severity, next(self._counter), patient_id))
        return True
```



```python
    def _pop_valid_top(self) -> Optional[Patient]:
        """Pop heap until top refers to a current record with matching severity; return that patient or None."""
        while self._pq:
            severity, _, pid = heapq.heappop(self._pq)
            current = self._patient_records.get(pid)
            if current is None:
                # patient removed from records, stale entry
                continue
            if current.condition_severity == severity:
                # valid entry
                return current
            # severity changed since entry was pushed -> stale entry, skip
        return None

    def get_critical_patient(self) -> Optional[Patient]:
        """Remove and return the most critical patient (lowest severity number)."""
        patient = self._pop_valid_top()
        if patient:
            # remove from records as they are being handled/removed
            self._patient_records.pop(patient.patient_id, None)
        return patient

    def peek_critical_patient(self) -> Optional[Patient]:
        """Return the most critical patient without removing them."""
        # Pop valid top and reinsert it to preserve state
        patient = self._pop_valid_top()
        if patient:
            # reinsert the valid top entry
            heapq.heappush(self._pq, (patient.condition_severity, next(self._counter), patient.patient_id))
        return patient

    def remove_patient(self, patient_id: int) -> bool:
        """Remove a patient from records. Heap entries become stale and are ignored when popped."""
        return self._patient_records.pop(patient_id, None) is not None
```



```
    def remove_patient(self, patient_id: int) -> bool:
        """Remove a patient from records. Heap entries become stale and are ignored when popped."""
        return self._patient_records.pop(patient_id, None) is not None

    def list_patients(self) -> list[Patient]:
        return list(self._patient_records.values())

# Sample usage
if __name__ == "__main__":
    hospital = HospitalDatabase()
    hospital.add_patient(Patient(101, "Alice", 45, 2))
    hospital.add_patient(Patient(102, "Bob", 30, 1))
    hospital.add_patient(Patient(103, "Carol", 55, 3))

    print("Retrieve Patient ID 102:")
    print(hospital.get_patient(102))

    print("\nMost Critical Patient (pop):")
    print(hospital.get_critical_patient())

    print("\nPeek Next Critical Patient (no pop):")
    print(hospital.peek_critical_patient())

    print("\nUpdate severity for Patient 103 to 0 (more severe):")
    hospital.update_patient_severity(103, 0)
    print("Now most critical:")
    print(hospital.get_critical_patient())

    print("\nRemaining patients:")
    print(hospital.list_patients())
```

Sample Output

Past Sales Data: [120, 135, 150, 160, 145, 155, 165, 175]
Predicted Demand for Next Week: 165.0

Observations

1. The moving average algorithm provides a simple yet effective prediction model.
2. AI assistance helps in selecting the most accurate window size for trend prediction.
3. The approach supports better inventory management and reduces overstocking or shortages.
4. This method can scale easily with more complex AI-driven forecasting models later.