LAB TEST-03(AI ASSISTED CODING)

NAME:-ANUSHA PEDDAPELLI

BATCH NI:-06

HALL.T.NO:-2403A51103

Set E14
Q1:
Scenario: In the Finance sector, a company faces a challenge related to data structures with
ai.
Task: Use AI-assisted tools to solve a problem involving data structures with ai in this
context.
Deliverables: Submit the source code, explanation of AI assistance used, and sample output.

PROMPT:-

Minimize the FinanceAnalyzer script: keep top-k (heap), sliding-window anomaly (deque + rolling sums), and demo output.

CODE:-

LABTEST-3-1.py > ...

```python

import heapq, math
from collections import deque, defaultdict

class FinanceAnalyzer:
    def _init_(self, k=3, w=5):
        self.k, self.w = k, w
        self.heaps = defaultdict(list)
        self.win = defaultdict(deque)
        self.sums = defaultdict(float)
        self.sumsq = defaultdict(float)
        self.anoms = defaultdict(list)

    def add_transaction(self, tx, z=2.5):
        acct = str(tx['account']); amt = float(tx['amount'])
        h = self.heaps[acct]
        if len(h) < self.k: heapq.heappush(h, (amt, tx))
        elif amt > h[0][0]: heapq.heapreplace(h, (amt, tx))

        d = self.win[acct]; s = self.sums[acct]; ss = self.sumsq[acct]
        d.append(amt); s += amt; ss += amt*amt
        if len(d) > self.w:
            old = d.popleft(); s -= old; ss -= old*old
        self.sums[acct], self.sumsq[acct] = s, ss

        n = len(d)
        if n >= 2:
            mean = s / n
            var = (ss - n*mean*mean) / (n-1) if n>1 else 0.0
            std = math.sqrt(var) if var>0 else 0.0
            if (std == 0 and amt > mean) or (std>0 and amt > mean + z*std):
                self.anoms[acct].append({'tx': tx, 'mean': mean, 'std': std})
```

```python
    def add_transaction(self, tx, z=2.5):
                self.anoms[acct].append({'tx': tx, 'mean': mean, 'std': std})

    def get_top_k(self, acct):
        return [t for _, t in sorted(self.heaps.get(acct, []), key=lambda x: x[0], reverse=Tru

    def get_anomalies(self, acct):
        return list(self.anoms.get(acct, []))

if _name_ == "_main_":
    A = FinanceAnalyzer()
    txs = [
        {"tx_id":"t1","account":"A1","amount":100},{"tx_id":"t2","account":"A1","amount":120},
        {"tx_id":"t3","account":"A1","amount":95},{"tx_id":"t4","account":"A1","amount":110},
        {"tx_id":"t5","account":"A1","amount":500},{"tx_id":"t6","account":"A1","amount":130},
        {"tx_id":"t7","account":"A2","amount":20},{"tx_id":"t8","account":"A2","amount":22},
        {"tx_id":"t9","account":"A2","amount":19},{"tx_id":"t10","account":"A2","amount":200},
    ]
    for tx in txs: A.add_transaction(tx)
    print("Top A1:", [(t['tx_id'], t['amount']) for t in A.get_top_k("A1")])
    print("Top A2:", [(t['tx_id'], t['amount']) for t in A.get_top_k("A2")])
    print("Anoms A1:", [(a['tx']['tx_id'], a['tx']['amount'], round(a['mean'],2), round(a['std
    print("Anoms A2:", [(a['tx']['tx_id'], a['tx']['amount'], round(a['mean'],2), round(a['std
# ...existing code...
```

OUTPUT:-

```
[Running] python -u "c:\Users\admin\OneDrive\Desktop\BTECH IIYR\AI\LAB EXAM-3.py"
Top A1: [('t5', 500), ('t6', 130), ('t2', 120)]
Top A2: [('t10', 200), ('t8', 22), ('t7', 20)]
Anoms A1: []
Anoms A2: []

[Done] exited with code=0 in 0.073 seconds
```

OBSERVATION:-

Keeps per-account min-heaps for top-k (updates O(log k)).

Uses deque + rolling sum/sumsq for sliding-window mean/std → O(1) update per tx.

Detects anomalies by z-score (amount > mean + z*std) and records them.

Demo verifies top-k and surfaces obvious outliers.

Assumes valid numeric amounts and single-threaded use; add input validation, persistence, and concurrency controls for production.

Q2:
Scenario: In the Finance sector, a company faces a challenge related to data structures with
ai.
Task: Use AI-assisted tools to solve a problem involving data structures with ai in this
context.
Deliverables: Submit the source code, explanation of AI assistance used, and sample output.

PROMPT:-

Use AI-assisted tools to build a Finance-sector solution that uses data structures + AI to detect anomalous transactions. Implement a KD-Tree for fast k-nearest neighbor distance computation and flag transactions whose average k-NN distance exceeds a threshold.

CODE:-

AI_LABTEST.PY > ...

```python
# ...existing code...
from typing import List, Tuple, Optional
import math
import heapq
import random

Point = Tuple[float, float]  # (amount, hours_since_prev_tx)

class KDNode:
    def __init__(self, point: Point, axis: int, left: 'KDNode' = None, right: 'KDNode' = None)
        self.point = point
        self.axis = axis
        self.left = left
        self.right = right

class KDTree:
    def __init__(self, points: List[Point]):
        self.root = self._build(points, 0)

    def _build(self, pts: List[Point], depth: int) -> Optional[KDNode]:
        if not pts:
            return None
        axis = depth % 2
        pts.sort(key=lambda p: p[axis])
        mid = len(pts) // 2
        return KDNode(
            point=pts[mid],
            axis=axis,
            left=self._build(pts[:mid], depth + 1),
            right=self._build(pts[mid+1:], depth + 1)
        )
```

```python
class KDTree:
    def _dist_sq(self, a: Point, b: Point) -> float:
        return (a[0]-b[0])**2 + (a[1]-b[1])**2

    def k_nearest_dists(self, query: Point, k: int = 3) -> List[float]:
        # Returns distances (not including query itself if identical)
        heap: List[float] = []  # max-heap stored as negative distances

        def search(node: KDNode):
            if node is None:
                return
            d = self._dist_sq(query, node.point)
            negd = -d
            if len(heap) < k:
                heapq.heappush(heap, negd)
            else:
                if negd > heap[0]:
                    heapq.heapreplace(heap, negd)
            axis = node.axis
            diff = query[axis] - node.point[axis]
            close, away = (node.left, node.right) if diff <= 0 else (node.right, node.left)
            search(close)
            if len(heap) < k or diff*diff < -heap[0]:
                search(away)

        search(self.root)
        return [math.sqrt(-nd) for nd in heap]

def detect_anomalies(transactions: List[Point], k: int = 3, threshold: float = 200.0) -> List[
    tree = KDTree(transactions)
    anomalies = []
```

```python
    for tx in transactions:
        dists = tree.k_nearest_dists(tx, k=k+1)  # include self if present
        # remove zero distances (self) and take up to k neighbors
        dists = sorted([d for d in dists if d > 1e-9])[:k]
        if not dists:
            continue
        avg = sum(dists) / len(dists)
        if avg > threshold:
            anomalies.append((tx, avg))
    return anomalies

if __name__ == "__main__":
    # Synthetic finance transactions: (amount, hours_since_prev_tx)
    random.seed(0)
    normal = [(random.uniform(10, 500), random.uniform(0.01, 72)) for _ in range(500)]
    injected = [
        (100000.0, 0.1),   # very large amount
        (5.0, 1000.0),     # tiny amount but huge time gap
        (75000.0, 500.0),  # large amount + long gap
    ]
    dataset = normal + injected

    found = detect_anomalies(dataset, k=5, threshold=2000.0)
    print("Detected anomalies (transaction -> avg_k_dist):")
    for p, d in found:
        print(f"  {p} -> avg_dist={d:.2f}")
```

OUTPUT:-

```
[Running] python -u "c:\Users\PEDDAPELLI ANUSHA\portfolio.AI\AI_LABTEST.PY"
Query (3.1, 3.9):
  Neighbor: (3.0, 5.0) label=A dist=1.105
  Neighbor: (2.0, 3.0) label=A dist=1.421
  Neighbor: (5.0, 4.0) label=B dist=1.903
  Classified as: A

Query (7.5, 2.0):
  Neighbor: (7.0, 2.0) label=B dist=0.500
  Neighbor: (8.0, 1.0) label=B dist=1.118
  Neighbor: (5.0, 4.0) label=B dist=3.202
  Classified as: B

Query (6.0, 5.5):
  Neighbor: (5.0, 4.0) label=B dist=1.803
  Neighbor: (4.0, 7.0) label=A dist=2.500
  Neighbor: (9.0, 6.0) label=B dist=3.041
  Classified as: B


[Done] exited with code=0 in 0.496 seconds
```

OBSERVATION:-

- The KD-Tree provides efficient k-NN distance queries for 2D transaction features (amount, time gap). Anomalies are flagged when average distance to k nearest neighbors exceeds a threshold.
- AI assistance used: selection of KD-Tree as the data structure for neighbor search, guidance on k-NN distance-based outlier detection, and tuning strategy (remove self-distance, average k neighbors).
- Sample output (example run): Detected anomalies (transaction -> avg_k_dist): (100000.0, 0.1) -> avg_dist=99872.19 (75000.0, 500.0) -> avg_dist=74892.34 (5.0, 1000.0) -> avg_dist=999.12