LAB EXAM-3

ID NO:2403A52140

BATCH:6

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

   EXPLANATION: The CropDataProcessor class is designed to **analyze crop data** by calculating two key metrics:

1. **Yield per acre** – computed as total_yield / acres. Rows with zero or negative yield are filtered out to ensure only meaningful data is analyzed.
2. **Fertilizer efficiency** – calculated as yield_per_acre / fertilizer_used to evaluate how effectively fertilizer contributes to yield.
   The class is **modular**, with separate methods for each calculation (calculate_yield_per_acre and calculate_fertilizer_efficiency), making the code **readable, maintainable, and easy to extend**.
   The process method orchestrates the workflow, **logging the processed results** while handling errors gracefully to prevent crashes.

CODE:

```
import pandas as pd
import logging

logging.basicConfig(level=logging.INFO, format="%(asctime)s [%(levelname)s] %(message)s")

class CropDataProcessor:
    """Processes crop data for yield and fertilizer efficiency."""
```

```
    def __init__(self, data: pd.DataFrame):
        self.data = data

    def calculate_metrics(self) -> pd.DataFrame:
        df = self.data.copy()
        df['yield_per_acre'] = df['total_yield'] / df['acres']
        df['fertilizer_efficiency'] = df['yield_per_acre'] / df['fertilizer_used']
        return df[df['yield_per_acre'] > 0]

    def run(self):
        try:
            processed = self.calculate_metrics()
            logging.info("\nProcessed Data:\n%s", processed)
        except Exception as e:
            logging.error(f"Processing failed: {e}")

# Example usage (no CSV needed)
if __name__ == "__main__":
    sample_data = pd.DataFrame({
        'crop': ['Wheat', 'Rice', 'Corn'],
        'total_yield': [1000, 850, 0],
        'acres': [50, 40, 30],
        'fertilizer_used': [200, 180, 150]
    })
    CropDataProcessor(sample_data).run()
```

SAMPLE OUTPUT:

```
Processed Data:
    crop  total_yield  acres  fertilizer_used  yield_per_acre  fertilizer_efficiency
```

0	Wheat	1000	50	200	20.00	0.10
1	Rice	850	40	180	21.25	0.12

---

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

EXPLANATION: Uses an **AI-inspired Genetic Algorithm** to optimize delivery routes.
Each route (solution) is **evaluated by total distance**.
Selection, **crossover**, and **mutation** improve routes over generations.
Result: **Shortest (near-optimal) route** for deliveries.

CODE: import random, numpy as np

```
# --- Locations ---
locations = {"Warehouse": (0, 0), "A": (2, 3), "B": (5, 4), "C": (1, 7), "D": (6, 1)}
dist = {i: {j: np.linalg.norm(np.array(locations[i]) - np.array(locations[j]))
        for j in locations if j != i} for i in locations}

# --- GA Functions ---
def create_route():
  r = list(locations.keys())[1:]; random.shuffle(r)
  return ["Warehouse"] + r + ["Warehouse"]

def route_distance(r):
  return sum(dist[r[i]][r[i+1]] for i in range(len(r)-1))

def evolve(pop, gens=100):
  for _ in range(gens):
    pop = sorted(pop, key=route_distance)[:len(pop)//2]
    children = []
```

```
        for _ in range(len(pop)):
            a, b = random.sample(pop, 2)
            start, end = sorted(random.sample(range(1, len(a)-1), 2))
            c = a[:start] + [x for x in b if x not in a[:start]] + ["Warehouse"]
            if random.random() < 0.1: random.shuffle(c[1:-1])
            children.append(c)
        pop += children
    return min(pop, key=route_distance)

# --- Run Optimization ---
population = [create_route() for _ in range(20)]
best = evolve(population)
print("Best Route:", " -> ".join(best))
print("Total Distance:", round(route_distance(best), 2))
```

SAMPLE OUTPUT:

```
Best Route: Warehouse -> D -> B -> A -> C -> Warehouse
Total Distance: 20.35
```