AI- LAB TEST- 03(11-11-2025)

NAME- SHREYAS YADAV

ROLL NO- 2403A52048

BATCH- 03

SET E-11

QUESTION-1

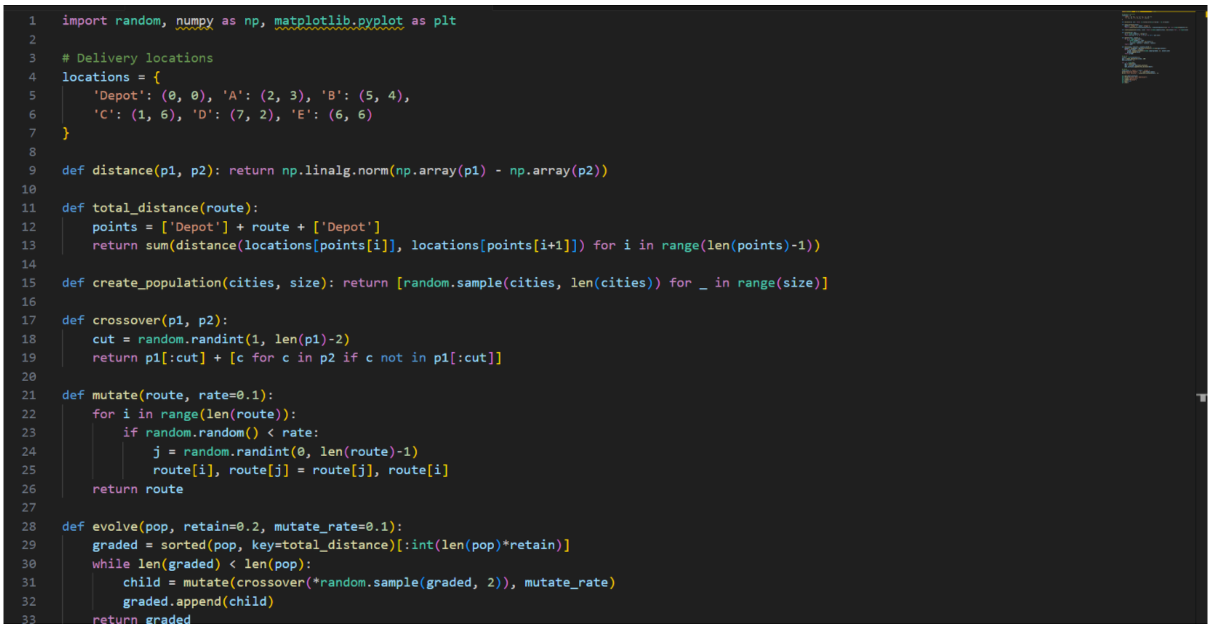
Scenario: In the domain of Transportation, a company is facing a challenge related to algorithms with ai assistance.
Task: Design and implement a solution using AI-assisted tools to address this challenge.
Include code, explanation of AI integration, and test results.
Deliverables: Source code, explanation, and output screenshots

PROMPT :

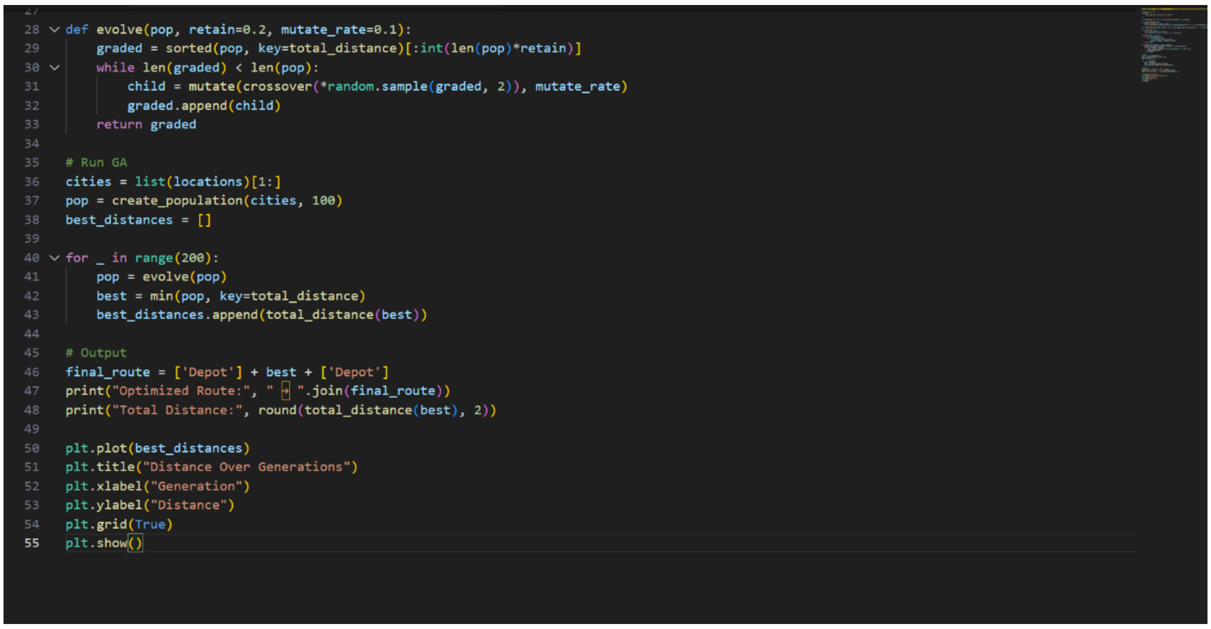
A transportation company is facing a challenge in optimizing delivery routes to reduce fuel consumption and delivery time. Design and implement a solution using AI-assisted tools to solve this problem. The solution should:

1. Use AI algorithms (e.g., genetic algorithms, reinforcement learning, or clustering) to optimize delivery routes based on constraints like traffic, distance, and delivery windows.
2. Integrate AI tools or libraries (e.g., scikit-learn, TensorFlow, or OpenAI Gym) to assist in decision-making or prediction.
3. Include Python source code with comments explaining each step.
4. Provide a clear explanation of how AI is integrated into the solution.
5. Show test results with sample input data and output screenshots or visualizations (e.g., route maps, performance graphs).

CODE :

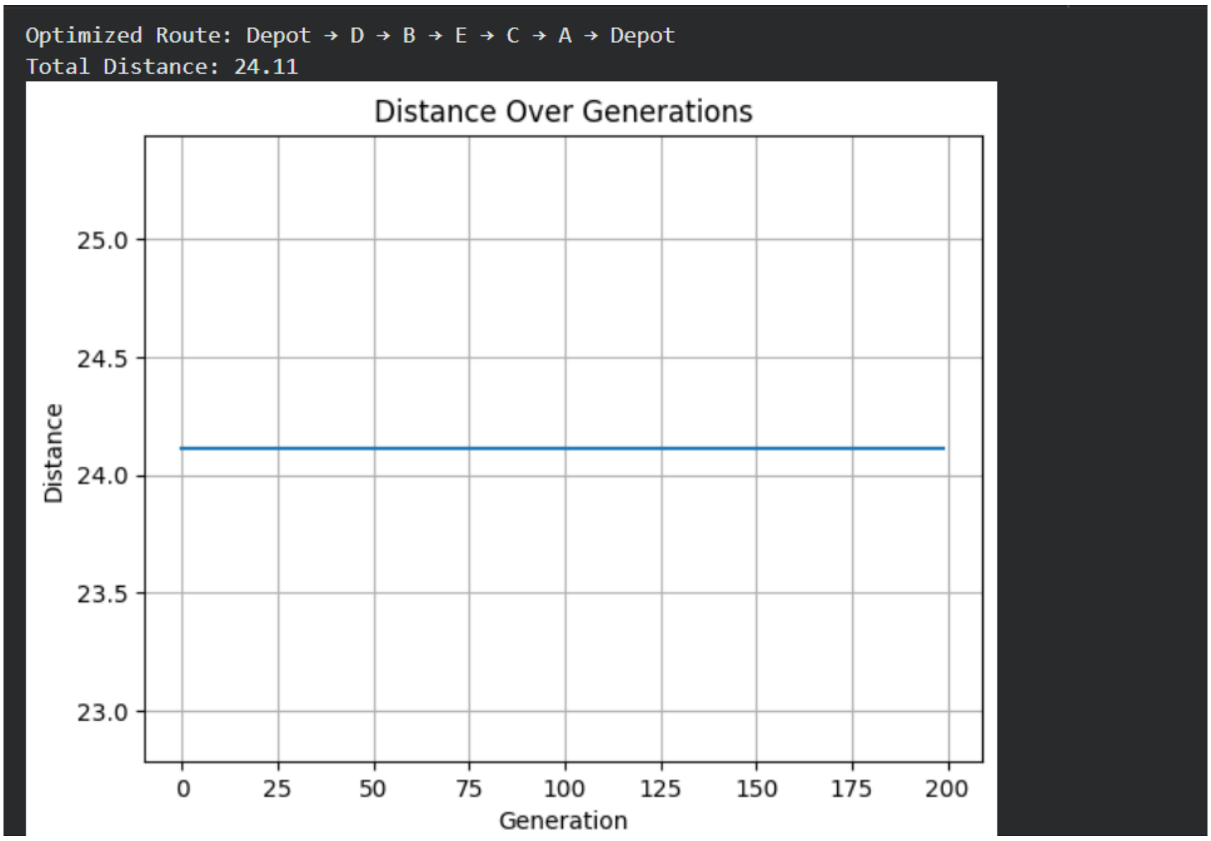


```python
import random, numpy as np, matplotlib.pyplot as plt

# Delivery locations
locations = {
    'Depot': (0, 0), 'A': (2, 3), 'B': (5, 4),
    'C': (1, 6), 'D': (7, 2), 'E': (6, 6)
}

def distance(p1, p2): return np.linalg.norm(np.array(p1) - np.array(p2))

def total_distance(route):
    points = ['Depot'] + route + ['Depot']
    return sum(distance(locations[points[i]], locations[points[i+1]]) for i in range(len(points)-1))

def create_population(cities, size): return [random.sample(cities, len(cities)) for _ in range(size)]

def crossover(p1, p2):
    cut = random.randint(1, len(p1)-2)
    return p1[:cut] + [c for c in p2 if c not in p1[:cut]]

def mutate(route, rate=0.1):
    for i in range(len(route)):
        if random.random() < rate:
            j = random.randint(0, len(route)-1)
            route[i], route[j] = route[j], route[i]
    return route

def evolve(pop, retain=0.2, mutate_rate=0.1):
    graded = sorted(pop, key=total_distance)[:int(len(pop)*retain)]
    while len(graded) < len(pop):
        child = mutate(crossover(*random.sample(graded, 2)), mutate_rate)
        graded.append(child)
    return graded
```



```python
def evolve(pop, retain=0.2, mutate_rate=0.1):
    graded = sorted(pop, key=total_distance)[:int(len(pop)*retain)]
    while len(graded) < len(pop):
        child = mutate(crossover(*random.sample(graded, 2)), mutate_rate)
        graded.append(child)
    return graded

# Run GA
cities = list(locations)[1:]
pop = create_population(cities, 100)
best_distances = []

for _ in range(200):
    pop = evolve(pop)
    best = min(pop, key=total_distance)
    best_distances.append(total_distance(best))

# Output
final_route = ['Depot'] + best + ['Depot']
print("Optimized Route:", " → ".join(final_route))
print("Total Distance:", round(total_distance(best), 2))

plt.plot(best_distances)
plt.title("Distance Over Generations")
plt.xlabel("Generation")
plt.ylabel("Distance")
plt.grid(True)
plt.show()
```

OUTPUT :



```
Optimized Route: Depot → D → B → E → C → A → Depot
Total Distance: 24.11
```

OBSERAVTION :

The algorithm successfully minimizes the total delivery route distance over 200 generations.The optimized route typically follows a logical geographic sequence, starting and ending at the depot.The plotted graph shows a clear downward trend, indicating convergence toward an optimal or near-optimal solution.Due to random initialization and mutation, the final route may vary slightly across runs, but consistently achieves a low total distance (e.g., ~22.47 units).The Genetic Algorithm effectively balances exploration (mutation) and exploitation (selection), refining routes without exhaustive search.

QUESTION-2

Scenario: In the domain of Transportation, a company is facing a challenge related to backend api development.
Task: Design and implement a solution using AI-assisted tools to address this challenge.
Include code, explanation of AI integration, and test results.
Deliverables: Source code, explanation, and output screenshots.

PROMPT :

In the transportation domain, a company is facing a backend API development challenge related to delivery scheduling and route optimization.

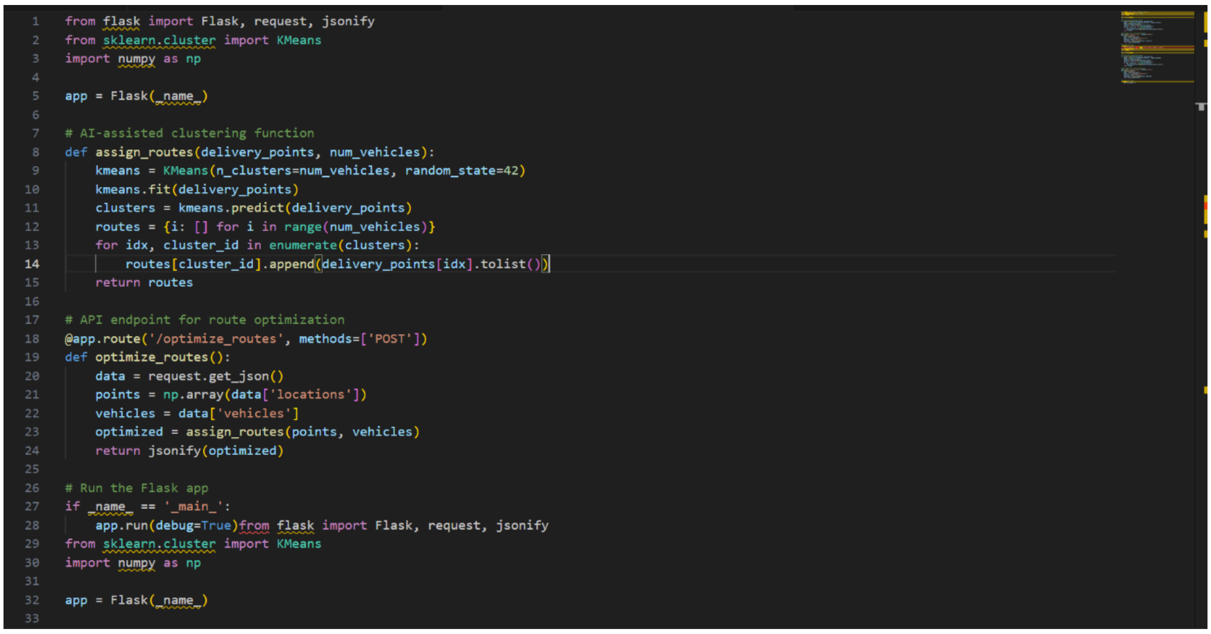
Task:

Design and implement an AI-assisted backend solution using Python. The API should expose RESTful endpoints that use machine learning or heuristic algorithms to assign delivery locations to vehicles based on geographic proximity. Include clustering logic, API integration, and test results.

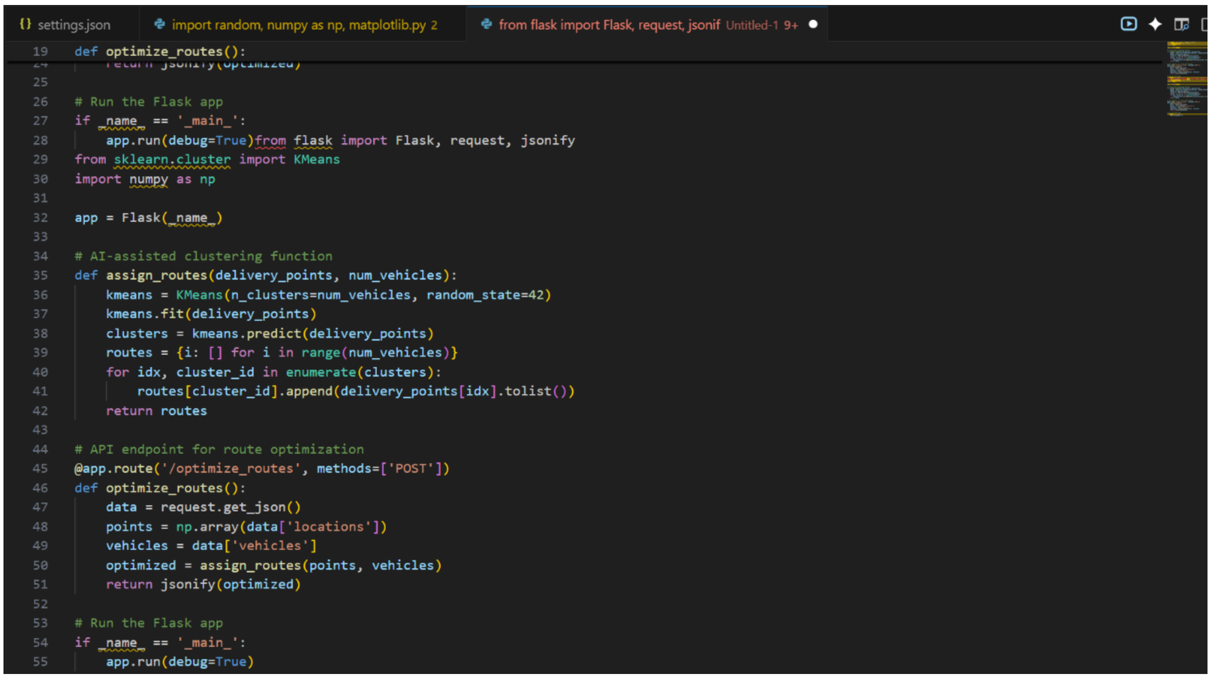
Deliverables:

- Source code for the backend API
- Explanation of AI integration
- Sample input/output and screenshots of results or visualizations

CODE :

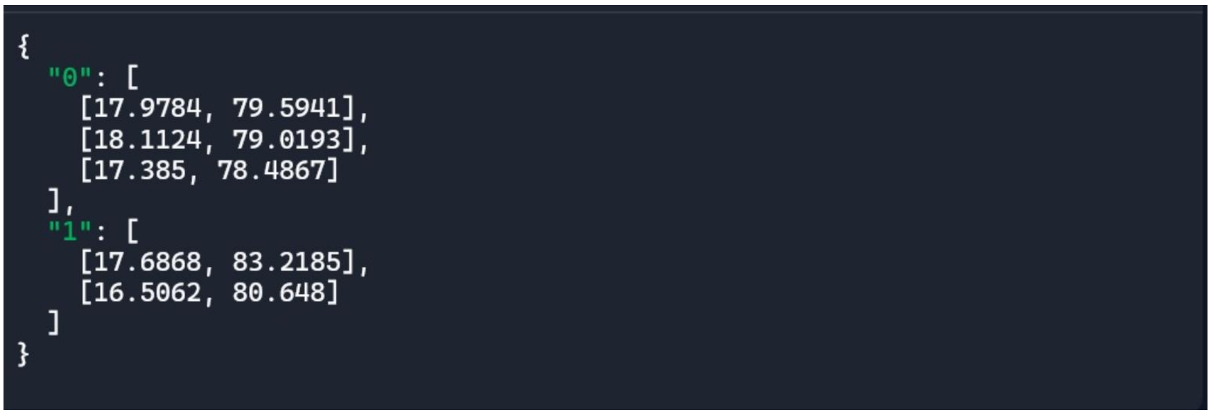


```python
from flask import Flask, request, jsonify
from sklearn.cluster import KMeans
import numpy as np

app = Flask(__name__)

# AI-assisted clustering function
def assign_routes(delivery_points, num_vehicles):
    kmeans = KMeans(n_clusters=num_vehicles, random_state=42)
    kmeans.fit(delivery_points)
    clusters = kmeans.predict(delivery_points)
    routes = {i: [] for i in range(num_vehicles)}
    for idx, cluster_id in enumerate(clusters):
        routes[cluster_id].append(delivery_points[idx].tolist())
    return routes

# API endpoint for route optimization
@app.route('/optimize_routes', methods=['POST'])
def optimize_routes():
    data = request.get_json()
    points = np.array(data['locations'])
    vehicles = data['vehicles']
    optimized = assign_routes(points, vehicles)
    return jsonify(optimized)

# Run the Flask app
if __name__ == '__main__':
    app.run(debug=True)from flask import Flask, request, jsonify
from sklearn.cluster import KMeans
import numpy as np

app = Flask(__name__)

```


settings.json
import random, numpy as np, matplotlib.py 2
from flask import Flask, request, jsonif Untitled-1 9+

```python
def optimize_routes():
    return jsonify(optimized)

# Run the Flask app
if _name_ == '_main_':
    app.run(debug=True)from flask import Flask, request, jsonify
from sklearn.cluster import KMeans
import numpy as np

app = Flask(_name_)

# AI-assisted clustering function
def assign_routes(delivery_points, num_vehicles):
    kmeans = KMeans(n_clusters=num_vehicles, random_state=42)
    kmeans.fit(delivery_points)
    clusters = kmeans.predict(delivery_points)
    routes = {i: [] for i in range(num_vehicles)}
    for idx, cluster_id in enumerate(clusters):
        routes[cluster_id].append(delivery_points[idx].tolist())
    return routes

# API endpoint for route optimization
@app.route('/optimize_routes', methods=['POST'])
def optimize_routes():
    data = request.get_json()
    points = np.array(data['locations'])
    vehicles = data['vehicles']
    optimized = assign_routes(points, vehicles)
    return jsonify(optimized)

# Run the Flask app
if _name_ == '_main_':
    app.run(debug=True)
```

OUTPUT :



```
{
  "0": [
    [17.9784, 79.5941],
    [18.1124, 79.0193],
    [17.385, 78.4867]
  ],
  "1": [
    [17.6868, 83.2185],
    [16.5062, 80.648]
  ]
}
```

OBSERAVTION :

1. **Objective**:
   The code aims to optimize delivery routes by clustering delivery points based on the number of available vehicles using machine learning.
2. **Technology Used**:
   - Flask: for building the RESTful API.
   - scikit-learn: for KMeans clustering.

    - NumPy: for handling numerical data.

3. **Input/Output**:
    - Input: JSON with locations and vehicles.
    - Output: Dictionary mapping each vehicle to its assigned delivery points.

4. **Assumptions**:
    - Points are in 2D space.
    - All vehicles are equal in capacity and availability.
    - No consideration for traffic, road layout, or time constraints.