CODE:

```
import numpy as np
import random

# Define the problem: Delivery locations and their coordinates
def euclidean_distance(city1, city2):
    """Calculate Euclidean distance between two cities."""
    return np.sqrt((city1[0] - city2[0])**2 + (city1[1] - city2[1])**2)

class AntColonyOptimization:
    def __init__(self, distance_matrix, num_ants, num_iterations, alpha, beta, rho,
pheromone_initial):
        self.distance_matrix = distance_matrix
        self.num_cities = len(distance_matrix)
        self.num_ants = num_ants
        self.num_iterations = num_iterations
        self.alpha = alpha  # Importance of pheromone
        self.beta = beta    # Importance of heuristic information (1/distance)
        self.rho = rho      # Pheromone evaporation rate
        self.pheromone = np.full((self.num_cities, self.num_cities), pheromone_initial)
        self.best_route = None
        self.best_distance = float('inf')

    def run(self):
        for iteration in range(self.num_iterations):
            all_routes = []
            all_distances = []
            for ant in range(self.num_ants):
                route, distance = self.construct_solution()
                all_routes.append(route)
                all_distances.append(distance)
                if distance < self.best_distance:
                    self.best_distance = distance
                    self.best_route = route
            self.update_pheromones(all_routes, all_distances)
            print(f"Iteration {iteration + 1}/{self.num_iterations}, Best Distance:
{self.best_distance}")
        return self.best_route, self.best_distance
```

```python
def construct_solution(self):
    route = []
    unvisited = list(range(self.num_cities))
    current_city = random.choice(unvisited)
    route.append(current_city)
    unvisited.remove(current_city)

    while unvisited:
        probabilities = self.calculate_transition_probabilities(current_city, unvisited)
        next_city = random.choices(unvisited, weights=probabilities)[0]
        route.append(next_city)
        unvisited.remove(next_city)
        current_city = next_city

    route.append(route[0])  # Return to the starting city
    distance = self.calculate_route_distance(route)
    return route, distance

def calculate_transition_probabilities(self, current_city, unvisited):
    probabilities = []
    for next_city in unvisited:
        pheromone = self.pheromone[current_city][next_city] ** self.alpha
        heuristic = (1 / self.distance_matrix[current_city][next_city]) ** self.beta
        probabilities.append(pheromone * heuristic)
    total = sum(probabilities)
    probabilities = [p / total for p in probabilities]
    return probabilities

def calculate_route_distance(self, route):
    distance = 0
    for i in range(len(route) - 1):
        distance += self.distance_matrix[route[i]][route[i + 1]]
    return distance

def update_pheromones(self, all_routes, all_distances):
    self.pheromone *= (1 - self.rho)  # Evaporation
    for route, distance in zip(all_routes, all_distances):
        pheromone_contribution = 1 / distance
        for i in range(len(route) - 1):
            self.pheromone[route[i]][route[i + 1]] += pheromone_contribution
```

```
                self.pheromone[route[i + 1]][route[i]] += pheromone_contribution

# Supply chain management: Define delivery locations and distance matrix
locations = [
    (0, 0),  # Depot
    (2, 5),  # Location 1
    (5, 2),  # Location 2
    (6, 6),  # Location 3
    (8, 3)   # Location 4
]

# Create the distance matrix
num_cities = len(locations)
distance_matrix = np.zeros((num_cities, num_cities))
for i in range(num_cities):
    for j in range(num_cities):
        distance_matrix[i][j] = euclidean_distance(locations[i], locations[j])

# Parameters for ACO
num_ants = 10
num_iterations = 50
alpha = 1.0  # Importance of pheromone
beta = 5.0   # Importance of heuristic (1/distance)
rho = 0.5    # Pheromone evaporation rate
pheromone_initial = 0.1

# Run ACO
aco = AntColonyOptimization(distance_matrix, num_ants, num_iterations, alpha, beta, rho,
pheromone_initial)
best_route, best_distance = aco.run()

# Output the best route and distance
print("\nOptimal Delivery Route Found:")
print(f"Best Route: {best_route}")
print(f"Best Distance: {best_distance:.2f}")
```

OUTPUT:

```
Iteration 1/5, Best Distance: 24.880915804124726
Iteration 2/5, Best Distance: 21.661264175519037
Iteration 3/5, Best Distance: 21.661264175519037
Iteration 4/5, Best Distance: 21.661264175519037
Iteration 5/5, Best Distance: 21.661264175519037

Optimal Delivery Route Found:
Best Route: [1, 3, 4, 2, 0, 1]
Best Distance: 21.66
```