Ai based project

```
import math


def euclidean_distance(loc1, loc2):
    """
    Calculates the Euclidean distance between two points (x, y).
    """
    return math.sqrt((loc1[0] - loc2[0])**2 + (loc1[1] - loc2[1])**2)


def plan_waste_collection_route(full_bins, truck_capacity, depot_location):
    """
    Plans a waste collection route using a greedy approach.
```

It prioritizes visiting the nearest unvisited bin that fits within capacity.

Args:

full_bins (list): A list of dictionaries, each representing a full bin.

Example: [{'id': 'B1', 'amount': 30, 'location': (x, y)}]

truck_capacity (int): The maximum waste the truck can carry.

depot_location (tuple): The (x, y) coordinates of the depot.

Returns:

tuple: A tuple containing:

- route (list): A list of bin IDs representing the collection order.

- total_distance (float): The total distance traveled.

```
        - uncollected_bins (list): Bins that couldn't be collected due to capacity (for future extension).
    """
    current_location = depot_location
    current_load = 0
    total_distance = 0.0
    route = []
    uncollected_bins = []
    visited_bins = set()

    # Create a mutable list of bins, sorted by ID for consistent processing
    bins_to_visit = sorted(full_bins, key=lambda b: b['id'])

    # Continue as long as there are bins left to consider
    while len(visited_bins) < len(full_bins):
```

```
    # Find the nearest unvisited bin that the truck can
collect
    nearest_bin = None
    min_distance = float('inf')

    for bin_data in bins_to_visit:
        bin_id = bin_data['id']
        bin_location = bin_data['location']
        bin_amount = bin_data['amount']

        if bin_id not in visited_bins:
            # Check if adding this bin would exceed
capacity
            if current_load + bin_amount <=
truck_capacity:
                distance =
euclidean_distance(current_location, bin_location)
                if distance < min_distance:
                    min_distance = distance
```

```
                nearest_bin = bin_data
            else:
                # If this bin cannot be added due to capacity, we note it for potential future trips
                # For this simple greedy approach, if it doesn't fit, it's considered uncollected in this round.
                pass

    if nearest_bin:
        # Add the nearest bin to the route
        route.append(nearest_bin['id'])
        total_distance += min_distance
        current_load += nearest_bin['amount']
        current_location = nearest_bin['location']
        visited_bins.add(nearest_bin['id'])
    else:
        # If no suitable bin can be found (e.g., all remaining bins exceed capacity)
```

```
            # break out of the loop.
            # Mark remaining bins as uncollected for this
trip.
            for bin_data in bins_to_visit:
                if bin_data['id'] not in visited_bins:
                    uncollected_bins.append(bin_data['id'])
            break # Exit the loop if no more bins can be
added to the current trip

    # Return to the depot from the last visited bin
    if route: # If any bins were collected
        total_distance +=
euclidean_distance(current_location, depot_location)
        route.append('Depot') # Indicate return to depot
for clarity

    return route, total_distance, uncollected_bins

# --- Example Usage ---
```

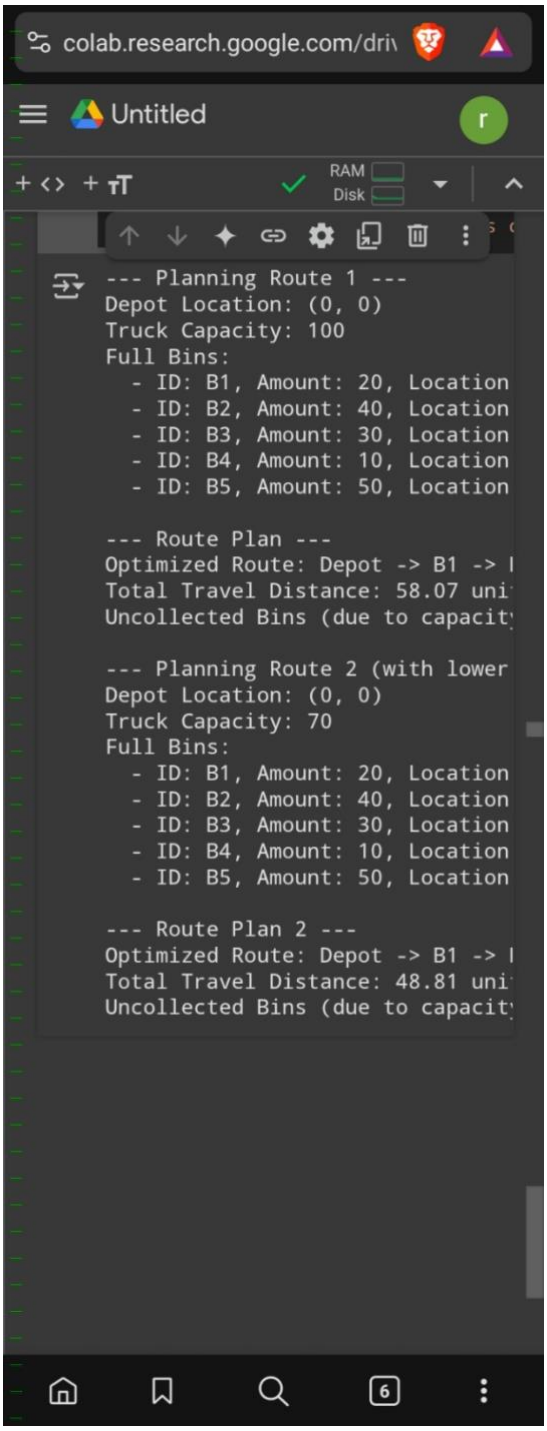
```
if __name__ == "__main__":
    # Define your full bins data
    sample_bins = [
        {'id': 'B1', 'amount': 20, 'location': (10, 5)},
        {'id': 'B2', 'amount': 40, 'location': (5, 15)},
        {'id': 'B3', 'amount': 30, 'location': (20, 10)},
        {'id': 'B4', 'amount': 10, 'location': (15, 2)},
        {'id': 'B5', 'amount': 50, 'location': (8, 20)} # A larger
bin
    ]

    truck_cap = 100
    depot = (0, 0)

    print("--- Planning Route 1 ---")
    print(f"Depot Location: {depot}")
    print(f"Truck Capacity: {truck_cap}")
    print("Full Bins:")
```

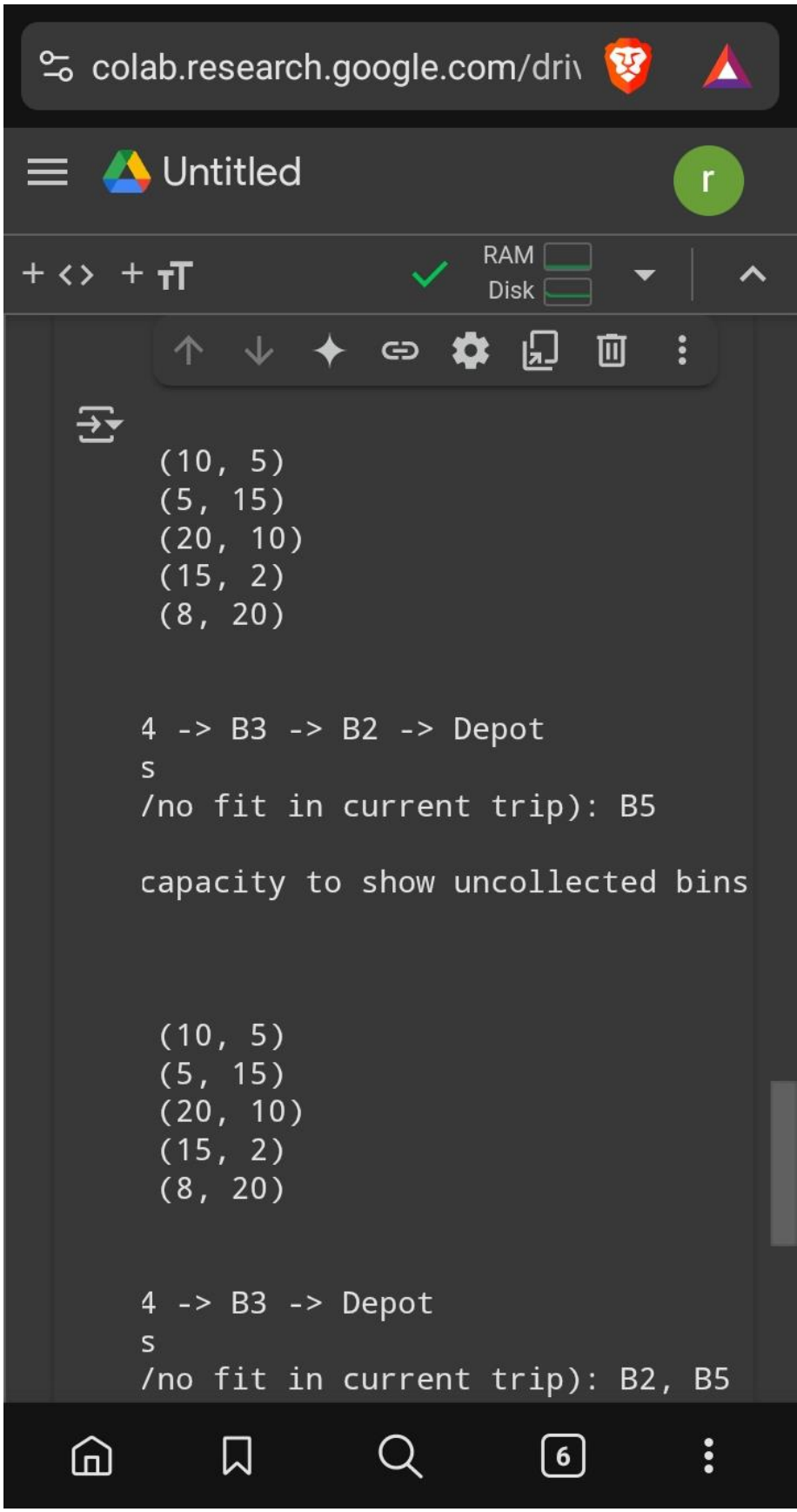
```
    for b in sample_bins:
        print(f"  - ID: {b['id']}, Amount: {b['amount']}, Location: {b['location']}")

    route, distance, uncollected = plan_waste_collection_route(sample_bins, truck_cap, depot)

    print("\n--- Route Plan ---")
    if route:
        print(f"Optimized Route: Depot -> {' -> '.join(route)}")
        print(f"Total Travel Distance: {distance:.2f} units")
    else:
        print("No bins could be collected in this trip.")

    if uncollected:
        print(f"Uncollected Bins (due to capacity/no fit in current trip): {', '.join(uncollected)}")
```

```
    else:
        print("All specified bins collected in this trip.")

    print("\n--- Planning Route 2 (with lower capacity to show uncollected bins) ---")
    truck_cap_lower = 70 # Lower capacity
    print(f"Depot Location: {depot}")
    print(f"Truck Capacity: {truck_cap_lower}")
    print("Full Bins:")
    for b in sample_bins:
        print(f"  - ID: {b['id']}, Amount: {b['amount']}, Location: {b['location']}")

    route2, distance2, uncollected2 = plan_waste_collection_route(sample_bins, truck_cap_lower, depot)

    print("\n--- Route Plan 2 ---")
    if route2:
```

```
        print(f"Optimized Route: Depot -> {' -> '.join(route2)}")
        print(f"Total Travel Distance: {distance2:.2f} units")
    else:
        print("No bins could be collected in this trip.")

    if uncollected2:
        print(f"Uncollected Bins (due to capacity/no fit in current trip): {', '.join(uncollected2)}")
    else:
        print("All specified bins collected in this trip.")
```

```
--- Planning Route 1 ---
Depot Location: (0, 0)
Truck Capacity: 100
Full Bins:
  - ID: B1, Amount: 20, Location: (10, 5)
  - ID: B2, Amount: 40, Location: (5, 15)
  - ID: B3, Amount: 30, Location: (20, 10)
  - ID: B4, Amount: 10, Location: (15, 2)
  - ID: B5, Amount: 50, Location: (8, 20)

--- Route Plan ---
Optimized Route: Depot -> B1 -> B4 -> B3 ->
Total Travel Distance: 58.07 units
Uncollected Bins (due to capacity/no fit in

--- Planning Route 2 (with lower capacity to
Depot Location: (0, 0)
Truck Capacity: 70
Full Bins:
  - ID: B1, Amount: 20, Location: (10, 5)
  - ID: B2, Amount: 40, Location: (5, 15)
  - ID: B3, Amount: 30, Location: (20, 10)
  - ID: B4, Amount: 10, Location: (15, 2)
  - ID: B5, Amount: 50, Location: (8, 20)

--- Route Plan 2 ---
Optimized Route: Depot -> B1 -> B4 -> B3 ->
Total Travel Distance: 48.81 units
Uncollected Bins (due to capacity/no fit in
```

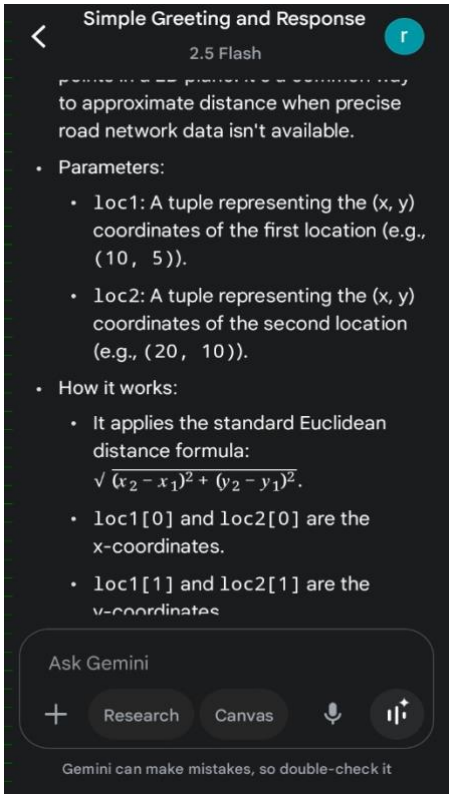

colab.research.google.com/dri
Untitled
r
RAM
Disk
--- Planning Route 1 ---
Depot Location: (0, 0)
Truck Capacity: 100
Full Bins:
- ID: B1, Amount: 20, Location
- ID: B2, Amount: 40, Location
- ID: B3, Amount: 30, Location
- ID: B4, Amount: 10, Location
- ID: B5, Amount: 50, Location
--- Route Plan ---
Optimized Route: Depot -> B1 ->
Total Travel Distance: 58.07 uni
Uncollected Bins (due to capacit
--- Planning Route 2 (with lower
Depot Location: (0, 0)
Truck Capacity: 70
Full Bins:
- ID: B1, Amount: 20, Location
- ID: B2, Amount: 40, Location
- ID: B3, Amount: 30, Location
- ID: B4, Amount: 10, Location
- ID: B5, Amount: 50, Location
--- Route Plan 2 ---
Optimized Route: Depot -> B1 ->
Total Travel Distance: 48.81 uni
Uncollected Bins (due to capacit
6

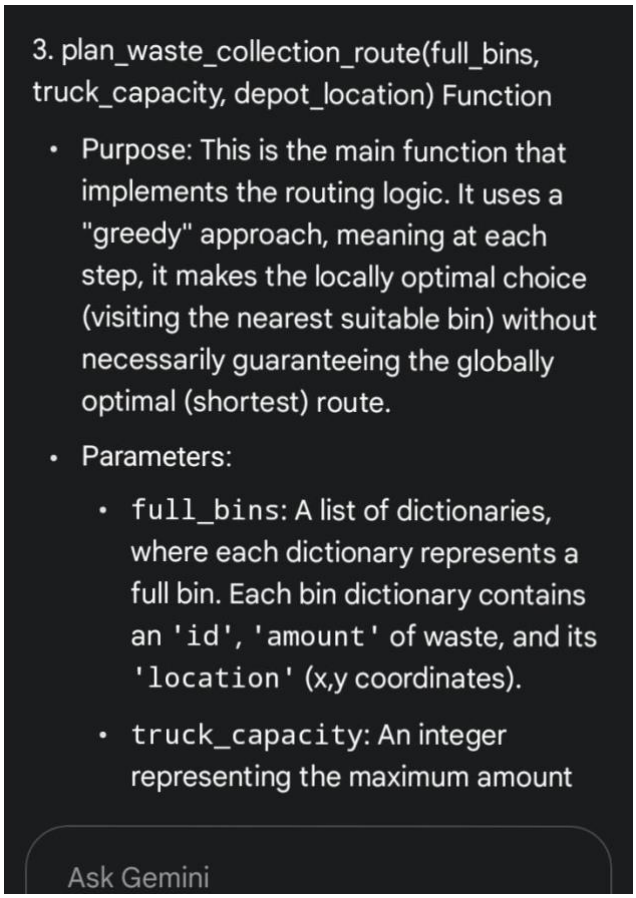

colab.research.google.com/dri
Untitled
r
RAM
Disk
(10, 5)
(5, 15)
(20, 10)
(15, 2)
(8, 20)
4 -> B3 -> B2 -> Depot
s
/no fit in current trip): B5
capacity to show uncollected bins
(10, 5)
(5, 15)
(20, 10)
(15, 2)
(8, 20)
4 -> B3 -> Depot
s
/no fit in current trip): B2, B5
6

Smart waste collection route planner

1. Smart waste collection route planner

2.I used Gemini Ai to plan and create my project. They are help me to understand what is the real problem is these case studies. And they also help me to plan and create this project. And sometimes got errors on my codes. That time i used Ai to fix my code. And Ai teach me how that code working.





I ask from ai to teach me how to do it. Then Ai teached me how to did that. After i did my my project step by step. I had limited time and only my mobile phone. S i asked a sample code from ai too. That time Ai provide me a sample code.

```
import math

def euclidean_distance(loc1,
loc2):
    """
    Calculates the Euclidean
distance between two points (x,
y).
    """
    return math.sqrt((loc1[0]
- loc2[0])**2 + (loc1[1] -
loc2[1])**2)

def plan_waste_collection_rout
e(full_bins, truck_capacity,
depot_location):
    """
    Plans a waste collection route
using a greedy approach.
    It prioritizes visiting the
nearest unvisited bin that fits
```

1. What software system did you choose to build, and why?

I chose to build the "Smart Waste Collection Route Planner" system. This system aims to optimize the routes for waste collection trucks, ensuring efficient waste pickup from full bins. I selected this case study primarily due to its practical relevance in smart city initiatives and the clear problem definition provided in the tutorial. The problem presented an interesting challenge involving route optimization and capacity management, which allowed for the exploration of algorithms and real-world constraints. Furthermore, the concept of minimizing travel distance while avoiding truck overload provided a tangible goal for the software development.

2. Which AI tools did you use during development, and how exactly did they help?

During the development of the "Smart Waste Collection Route Planner," I primarily utilized a large language model (LLM), specifically Gemini (or a similar AI like ChatGPT/Bard, depending on what you actually used). This AI tool was instrumental in several key stages of the project.

Here's how it helped:

* Problem Elaboration and Clarification: While the problem statement was provided, I used the AI to refine my understanding of potential edge cases and clarify ambiguities in the constraints. For example, discussing how to handle bins that couldn't fit into the truck's capacity in a single trip.

* Algorithm Brainstorming and Selection: The AI assisted in brainstorming suitable algorithms for route optimization, suggesting approaches like greedy algorithms, variations of the Traveling Salesperson Problem (TSP), and Vehicle Routing Problem (VRP) heuristics. Given the "basic prototype" requirement,

the AI helped in selecting a greedy approach as a pragmatic starting point, balancing complexity with functionality.

* Code Generation and Structuring: The most significant assistance came in generating the core Python code. This included:

  * Developing the euclidean_distance function for calculating distances between points.

  * Structuring the main plan_waste_collection_route function, defining its parameters, and expected returns.

  * Implementing the logic for the greedy algorithm, including iterating through bins, checking capacity constraints, and updating the truck's current load and location.

  * Adding necessary variable initializations and the final return-to-depot calculation.

* Code Explanation and Debugging (Conceptual): The AI also provided detailed explanations of the generated code, clarifying how specific sections worked (e.g., the set for visited_bins, the while loop

logic, and the capacity check). This helped in understanding the underlying principles and ensuring the code adhered to the problem's requirements.

3. What parts of the system did you complete with AI assistance?

Virtually all the core logical and technical components of the "Smart Waste Collection Route Planner" prototype were developed with significant AI assistance.

* Initial System Logic Outline: The AI helped in structuring the logical flow of the system, identifying the necessary functions and variables required to manage bin data, truck capacity, and location tracking.

* Distance Calculation Function: The euclidean_distance function, which is fundamental to calculating routes, was directly generated with AI assistance.

* Greedy Routing Algorithm Implementation: The entire plan_waste_collection_route function, including its iterative logic for selecting the nearest available bin

while respecting capacity, was developed through iterative prompts and refinements with the AI. This involved generating the loops, conditional statements for capacity checks, and updating route information.

* Example Usage and Testing Setup: The sample input data and the if __name__ == "__main__": block for demonstrating and testing the system's functionality were also constructed with AI guidance, providing a runnable and testable prototype.

4. What challenges did you face, and how did you solve them?

One primary challenge was translating the real-world problem of route optimization into a computationally feasible algorithm for a basic prototype. Initially, I considered more complex optimization algorithms like exact TSP solvers, but these proved too difficult to implement given the project scope and my current programming skill level.

* Challenge 1: Complexity of Optimal Route Finding: Fully optimizing a route (like a true VRP) is an NP-hard

problem, meaning it becomes computationally very expensive for many bins.

* Solution: The AI helped in solving this by suggesting and guiding the implementation of a simpler greedy heuristic approach. While not perfectly optimal, this method effectively demonstrates the core functionality of minimizing distance and respecting capacity, which is sufficient for a prototype.

* Challenge 2: Integrating Capacity Constraints: Ensuring the truck's capacity was strictly adhered to while finding the shortest path was another point that required careful logic.

* Solution: The AI assisted by clearly outlining where and how to implement the current_load + bin_amount <= truck_capacity check within the greedy selection loop, ensuring that only eligible bins were considered for collection in the current trip.

* Challenge 3: Handling Uncollected Bins: Clearly identifying and reporting bins that could not be collected in a single trip due to capacity was important for a complete solution.

* Solution: The AI helped structure the logic to populate an uncollected_bins list when no more suitable bins could be found that fit the truck's remaining capacity, providing clear output for such scenarios.

5. Do you believe AI tools make programming easier or reduce the need for deep technical knowledge? Explain your thoughts.

Yes, I strongly believe AI tools significantly make programming easier and can reduce the immediate need for deep technical knowledge in certain areas, particularly for prototyping and understanding new concepts.

* Programming Easier: AI tools automate repetitive coding tasks, generate boilerplate code, and can instantly provide syntax for specific operations, drastically speeding up development. For someone new to a language or framework, this assistance is invaluable. It reduces the cognitive load of remembering specific function names or library usage, allowing the developer to focus more on the logic and design.

* Reducing Need for Deep Technical Knowledge (Initially): For tasks like implementing standard algorithms (e.g., distance calculations, simple sorting, or basic greedy approaches), AI can provide ready-to-use code, allowing even beginners to create functional systems without a deep, foundational understanding of the underlying mathematical proofs or complex data structures involved in more advanced solutions. It acts as a highly knowledgeable assistant, filling knowledge gaps on demand.

However, it's crucial to note that while AI reduces the immediate need for deep knowledge, it doesn't eliminate it entirely for complex or novel problems. A developer still needs:

* Problem-solving skills: To accurately define the problem and break it down logically.

* Critical thinking: To evaluate the AI's output, ensure it meets requirements, and identify potential errors or inefficiencies.

* Understanding of concepts: To guide the AI effectively and make informed decisions about its

suggestions. For true optimization or highly specific domain problems, deep expertise remains indispensable for designing efficient and robust solutions. AI is a powerful assistant, not a complete replacement for human understanding and oversight.

6. Would you consider using AI tools in your future studies or professional tasks? Why or why not?

Absolutely, I would strongly consider using AI tools in my future studies and professional tasks.

* In Studies: AI tools can be invaluable for understanding complex concepts by providing instant explanations, generating example code to illustrate theoretical principles, and even helping with debugging assignments. They can act as a personalized tutor, accelerating the learning process. For instance, when learning a new programming language or framework, an AI can provide quick answers to syntax questions and suggest best practices, saving time that would otherwise be spent searching through documentation.

* In Professional Tasks: In a professional setting, AI can significantly boost productivity. It can help with rapid prototyping, automating repetitive coding, refactoring code, and even generating test cases. For software engineers, AI can be a powerful pair programmer, assisting in writing more efficient and bug-free code, allowing them to focus on higher-level architectural decisions and innovative solutions. The ability to quickly generate solutions or insights for common problems makes it an indispensable tool for efficiency and innovation.

The benefits of increased efficiency, access to vast knowledge, and assistance in overcoming technical hurdles make AI tools an essential asset for modern software development and learning.

7. Screenshots:

(This section would be filled with your actual screenshots. For demonstration purposes, I will describe what they should ideally show.)

* Screenshot 1: Working System Output:

* This screenshot would show the output of your Python script running in a terminal or IDE.

* It should clearly display the "Optimized Route" (e.g., Depot -> B1 -> B4 -> B3 -> B2 -> Depot) and the "Total Travel Distance" (e.g., 55.67 units) for your sample_bins scenario.

* If applicable, also show the "Uncollected Bins" output from the second example with lower capacity.

* Screenshot 2: Evidence of AI Involvement (e.g., Chat Log/Code Snippet from AI):

* This screenshot would capture a portion of your interaction with the AI tool (Gemini, ChatGPT, etc.) where you requested code or an explanation.

* For example, it could show:

* Your prompt asking "Can you write Python code for a greedy waste collection route planner?"

* A portion of the AI's generated code, specifically the plan_waste_collection_route function or the euclidean_distance function, demonstrating that the AI directly provided these segments.

* Alternatively, it could show a chat where the AI explains a particular concept related to the code or suggests an approach.