

Lab test-3

NAME: G. Bala Varshitha Reddy

HT.NO: 2403A52050

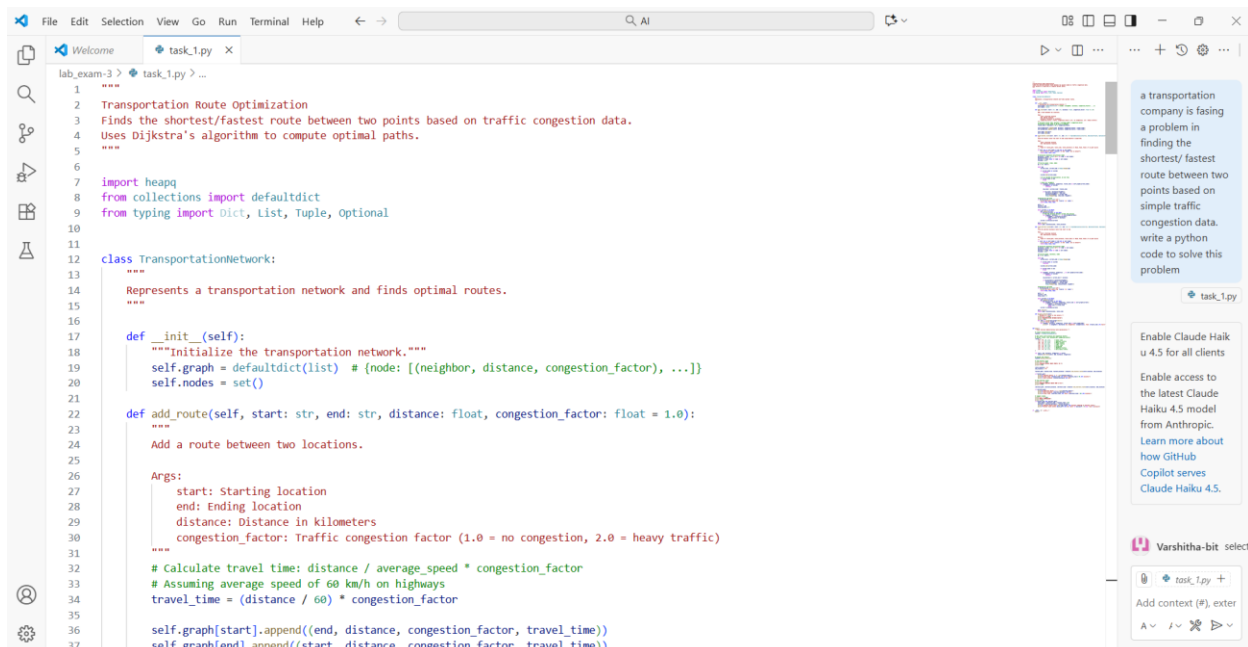
BATCH NO: AIB03

Set E11

Q1:

Prompt: A transportation company is facing a problem in finding the shortest/ fastest route between two points based on simple traffic data. write a python code to solve this problem

Code:



```
1  """
2  Transportation Route Optimization
3  Finds the shortest/fastest route between two points based on traffic congestion data.
4  Uses Dijkstra's algorithm to compute optimal paths.
5  """
6
7  import heapq
8  from collections import defaultdict
9  from typing import Dict, List, Tuple, Optional
10
11
12  class TransportationNetwork:
13      """
14      Represents a transportation network and finds optimal routes.
15      """
16
17      def __init__(self):
18          """Initialize the transportation network."""
19          self.graph = defaultdict(list) # {node: [(neighbor, distance, congestion_factor), ...]}
20          self.nodes = set()
21
22      def add_route(self, start: str, end: str, distance: float, congestion_factor: float = 1.0):
23          """
24          Add a route between two locations.
25
26          Args:
27              start: Starting location
28              end: Ending location
29              distance: Distance in kilometers
30              congestion_factor: Traffic congestion factor (1.0 = no congestion, 2.0 = heavy traffic)
31          """
32          # Calculate travel time: distance / average_speed * congestion_factor
33          # Assuming average speed of 60 km/h on highways
34          travel_time = (distance / 60) * congestion_factor
35
36          self.graph[start].append((end, distance, congestion_factor, travel_time))
37          self.graph[end].append((start, distance, congestion_factor, travel_time))
```

```
File Edit Selection View Go Run Terminal Help
task_1.py
lab_exam-3 > task_1.py > ...
12 class TransportationNetwork:
13     def add_route(self, start: str, end: str, distance: float, congestion_factor: float = 1.0):
14         self.graph[end].append((start, distance, congestion_factor, travel_time))
15
16     self.nodes.add(start)
17     self.nodes.add(end)
18
19 def find_fastest_route(self, start: str, end: str) -> Tuple[Optional[List[str]], Optional[float], Optional[float]]:
20     """
21     Find the fastest route from start to end using Dijkstra's algorithm.
22
23     Args:
24         start: Starting location
25         end: Destination location
26
27     Returns:
28         Tuple of (route_path, total_time, total_distance) or (None, None, None) if no path exists
29     """
30     if start not in self.nodes or end not in self.nodes:
31         print(f"Error: Start '{start}' or End '{end}' not in network")
32         return None, None, None
33
34     # Initialize distances and previous nodes
35     distances = {node: float('inf') for node in self.nodes}
36     distances[start] = 0
37     previous = {node: None for node in self.nodes}
38     visited = set()
39
40     # Priority queue: (time, node)
41     pq = [(0, start)]
42
43     while pq:
44         current_time, current_node = heapq.heappop(pq)
45
46         if current_node in visited:
47             continue
48
49         # If we reached the destination, we can stop
50         if current_node == end:
51             break
52
53         # Check all neighbors
54         for neighbor, distance, congestion, travel_time in self.graph[current_node]:
55             if neighbor in visited:
56                 continue
57
58             new_time = current_time + travel_time
59
60             if new_time < distances[neighbor]:
61                 distances[neighbor] = new_time
62                 previous[neighbor] = current_node
63                 heapq.heappush(pq, (new_time, neighbor))
64
65     # Reconstruct the path
66     if distances[end] == float('inf'):
67         print(f"No route found from '{start}' to '{end}'")
68         return None, None, None
69
70     path = []
71     current = end
72     total_distance = 0
73
74     while current is not None:
75         path.append(current)
76         if previous[current] is not None:
77             # Find the distance between current and previous
78             for neighbor, distance, _, _ in self.graph[current]:
79                 if neighbor == previous[current]:
80                     total_distance += distance
81                     break
82         current = previous[current]
```

a transportation company is facing a problem in finding the shortest/ fastest route between two points based on simple traffic congestion data. write a python code to solve this problem

task_1.py

Enable Claude Haiku 4.5 for all clients

Enable access to the latest Claude Haiku 4.5 model from Anthropic. [Learn more about how GitHub Copilot serves Claude Haiku 4.5.](#)

Varshitha-bit select

task_1.py +

Add context (#), enter

A V

Ln 255, Col 1 Spaces: 4 UTF-8 (i) Python 3.11.9 (Microsoft Store)

```
File Edit Selection View Go Run Terminal Help
task_1.py
lab_exam-3 > task_1.py > ...
12 class TransportationNetwork:
13     def find_fastest_route(self, start: str, end: str) -> Tuple[Optional[List[str]], Optional[float], Optional[float]]:
14
15         visited.add(current_node)
16
17         # If we reached the destination, we can stop
18         if current_node == end:
19             break
20
21         # Check all neighbors
22         for neighbor, distance, congestion, travel_time in self.graph[current_node]:
23             if neighbor in visited:
24                 continue
25
26             new_time = current_time + travel_time
27
28             if new_time < distances[neighbor]:
29                 distances[neighbor] = new_time
30                 previous[neighbor] = current_node
31                 heapq.heappush(pq, (new_time, neighbor))
32
33     # Reconstruct the path
34     if distances[end] == float('inf'):
35         print(f"No route found from '{start}' to '{end}'")
36         return None, None, None
37
38     path = []
39     current = end
40     total_distance = 0
41
42     while current is not None:
43         path.append(current)
44         if previous[current] is not None:
45             # Find the distance between current and previous
46             for neighbor, distance, _, _ in self.graph[current]:
47                 if neighbor == previous[current]:
48                     total_distance += distance
49                     break
50         current = previous[current]
```

a transportation company is facing a problem in finding the shortest/ fastest route between two points based on simple traffic congestion data. write a python code to solve this problem

task_1.py

Enable Claude Haiku 4.5 for all clients

Enable access to the latest Claude Haiku 4.5 model from Anthropic. [Learn more about how GitHub Copilot serves Claude Haiku 4.5.](#)

Varshitha-bit select

task_1.py +

Add context (#), enter

A V

Ln 255, Col 1 Spaces: 4 UTF-8 (i) Python 3.11.9 (Microsoft Store)

```
File Edit Selection View Go Run Terminal Help
task_1.py x
lab_exam-3 > task_1.py > ...
12 class TransportationNetwork:
42     def find_fastest_route(self, start: str, end: str) -> Tuple[Optional[List[str]], Optional[float], Optional[float]]:
106         break
107         current = previous[current]
108
109     path.reverse()
110     return path, distances[end], total_distance
111
112     def find_shortest_route(self, start: str, end: str) -> Tuple[Optional[List[str]], Optional[float], Optional[float]]:
113
114         Find the shortest distance route from start to end.
115
116         Args:
117             start: Starting location
118             end: Destination location
119
120         Returns:
121             Tuple of (route_path, total_distance, total_time) or (None, None, None) if no path exists
122
123         """
124         if start not in self.nodes or end not in self.nodes:
125             print(f"Error: Start '{start}' or End '{end}' not in network")
126             return None, None, None
127
128         # Initialize distances and previous nodes
129         distances = {node: float('inf') for node in self.nodes}
130         distances[start] = 0
131         previous = {node: None for node in self.nodes}
132         visited = set()
133
134         # Priority queue: (distance, node)
135         pq = [(0, start)]
136
137         while pq:
138             current_dist, current_node = heapq.heappop(pq)
139
140             if current_node in visited:
141                 continue
```

The screenshot displays a Visual Studio Code editor window with a Python file named `task_1.py`. The code implements a function `find_shortest_route` that takes a graph, start, and end nodes as input and returns a tuple containing the shortest path, total travel time, and congestion data. The graph is represented as a dictionary where each node is a key, and its value is a list of tuples representing neighboring nodes, distance, congestion, and travel time.

```
lab_exam-3 > task_1.py > ...  
12 class TransportationNetwork:  
112 def find_shortest_route(self, start: str, end: str) -> Tuple[Optional[List[str]], Optional[float], Optional[float]]:  
140     continue  
141  
142     visited.add(current_node)  
143  
144     if current_node == end:  
145         break  
146  
147     for neighbor, distance, congestion, _ in self.graph[current_node]:  
148         if neighbor in visited:  
149             continue  
150  
151         new_distance = current_dist + distance  
152  
153         if new_distance < distances[neighbor]:  
154             distances[neighbor] = new_distance  
155             previous[neighbor] = current_node  
156             heapq.heappush(pq, (new_distance, neighbor))  
157  
158 # Reconstruct the path  
159 if distances[end] == float('inf'):  
160     print(f"No route found from '{start}' to '{end}'")  
161     return None, None, None  
162  
163 path = []  
164 current = end  
165 total_time = 0  
166  
167 while current is not None:  
168     path.append(current)  
169     if previous[current] is not None:  
170         for neighbor, distance, congestion, travel_time in self.graph[current]:  
171             if neighbor == previous[current]:  
172                 total_time += travel_time  
173                 break  
174     current = previous[current]
```

The right sidebar shows a search results panel with a snippet titled "a transportation company is facing a problem in finding the shortest/ fastest route between two points based on simple traffic congestion data. write a python code to solve this problem". Below it, there are links to enable Claude Haiku u.4.5 for all clients, enable access to the latest Claude Haiku 4.5 model from Anthropic, learn more about how GitHub Copilot serves Claude Haiku 4.5, and a link to Varshitha-bit select.

```
File Edit Selection View Go Run Terminal Help
task_1.py x
lab_exam-3 > task_1.py > ...
112 class TransportationNetwork:
113     def find_shortest_route(self, start: str, end: str) -> Tuple[Optional[List[str]], Optional[float], Optional[float]]:
174         current = previous[current]
175
176         path.reverse()
177         return path, distances[end], total_time
178
179     def display_network(self):
180         """Display all routes in the network."""
181         print("\n" + "="*60)
182         print("TRANSPORTATION NETWORK ROUTES")
183         print("="*60)
184         for node in sorted(self.graph.keys()):
185             print(f"\nFrom {node}:")
186             for neighbor, distance, congestion, travel_time in self.graph[node]:
187                 print(f"  {neighbor}: {distance} km, Congestion: {congestion}x, Time: {travel_time:.2f} hours")
188
189
190 def main():
191     """Main function demonstrating route optimization."""
192
193     # Create transportation network
194     network = TransportationNetwork()
195
196     # Add routes with distance and congestion factors
197     # Format: (start, end, distance_km, congestion_factor)
198     routes = [
199         ("A", "B", 10, 1.0), # Clear road
200         ("A", "C", 15, 1.5), # Moderate traffic
201         ("B", "D", 12, 2.0), # Heavy traffic
202         ("C", "D", 8, 1.2), # Light traffic
203         ("D", "E", 20, 1.0), # Clear road
204         ("C", "E", 18, 1.8), # Congested
205         ("D", "E", 5, 1.5), # Moderate traffic
206         ("A", "D", 25, 2.5), # Very heavy traffic
207     ]
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

a transportation company is facing a problem in finding the shortest/ fastest route between two points based on simple traffic congestion data. write a python code to solve this problem

task_1.py

Enable Claude Haiku 4.5 for all clients

Enable access to the latest Claude Haiku 4.5 model from Anthropic. [Learn more about how GitHub Copilot serves Claude Haiku 4.5.](#)

Varshitha-bit select

task_1.py +

Add context (#), enter

A v

```
File Edit Selection View Go Run Terminal Help
task_1.py x
lab_exam-3 > task_1.py > ...
190 def main():
207     ]
208
209     for start, end, distance, congestion in routes:
210         network.add_route(start, end, distance, congestion)
211
212     # Display the network
213     network.display_network()
214
215     # Find fastest route
216     print("\n" + "="*60)
217     print("FINDING FASTEST ROUTE FROM A TO E")
218     print("="*60)
219
220     start_location = "A"
221     end_location = "E"
222
223     fastest_path, fastest_time, fastest_distance = network.find_fastest_route(start_location, end_location)
224
225     if fastest_path:
226         print(f"\nFastest Route: {' '.join(fastest_path)}")
227         print(f"Travel Time: {fastest_time:.2f} hours ({fastest_time * 60:.0f} minutes)")
228         print(f"Total Distance: {fastest_distance:.2f} km")
229
230     # Find shortest route
231     print("\n" + "="*60)
232     print("FINDING SHORTEST ROUTE FROM A TO E")
233     print("="*60)
234
235     shortest_path, shortest_distance, shortest_time = network.find_shortest_route(start_location, end_location)
236
237     if shortest_path:
238         print(f"\nShortest Route: {' '.join(shortest_path)}")
239         print(f"Total Distance: {shortest_distance:.2f} km")
240         print(f"Travel Time: {shortest_time:.2f} hours ({shortest_time * 60:.0f} minutes)")
241
242     # Compare routes
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

a transportation company is facing a problem in finding the shortest/ fastest route between two points based on simple traffic congestion data. write a python code to solve this problem

task_1.py

Enable Claude Haiku 4.5 for all clients

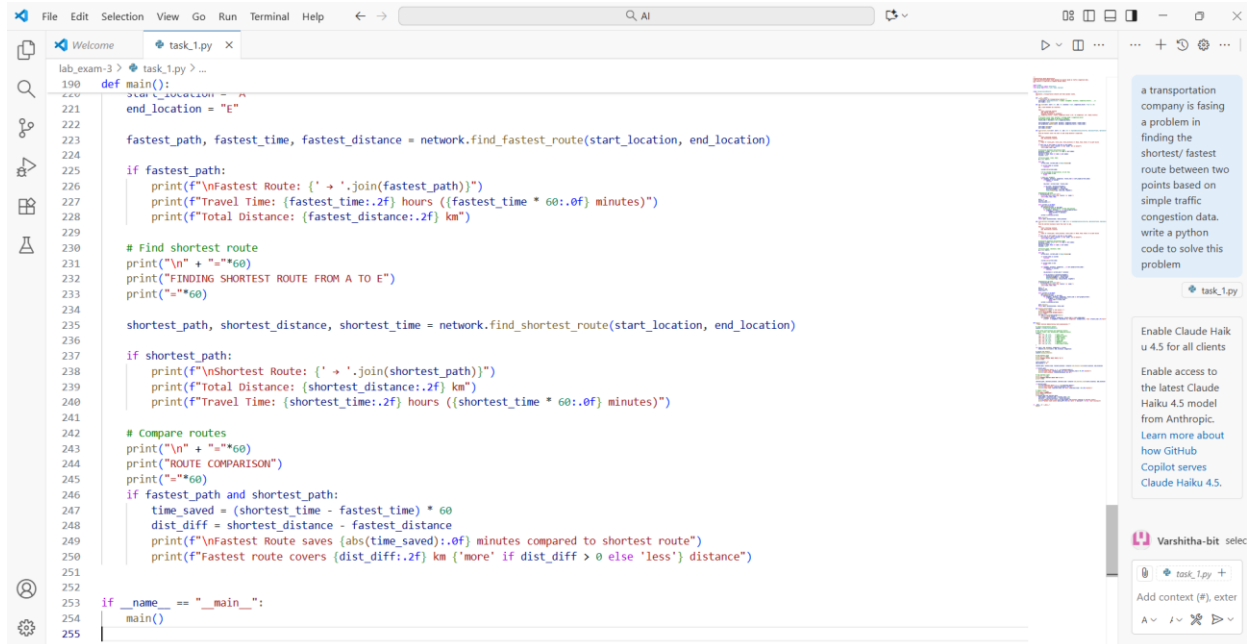
Enable access to the latest Claude Haiku 4.5 model from Anthropic. [Learn more about how GitHub Copilot serves Claude Haiku 4.5.](#)

Varshitha-bit select

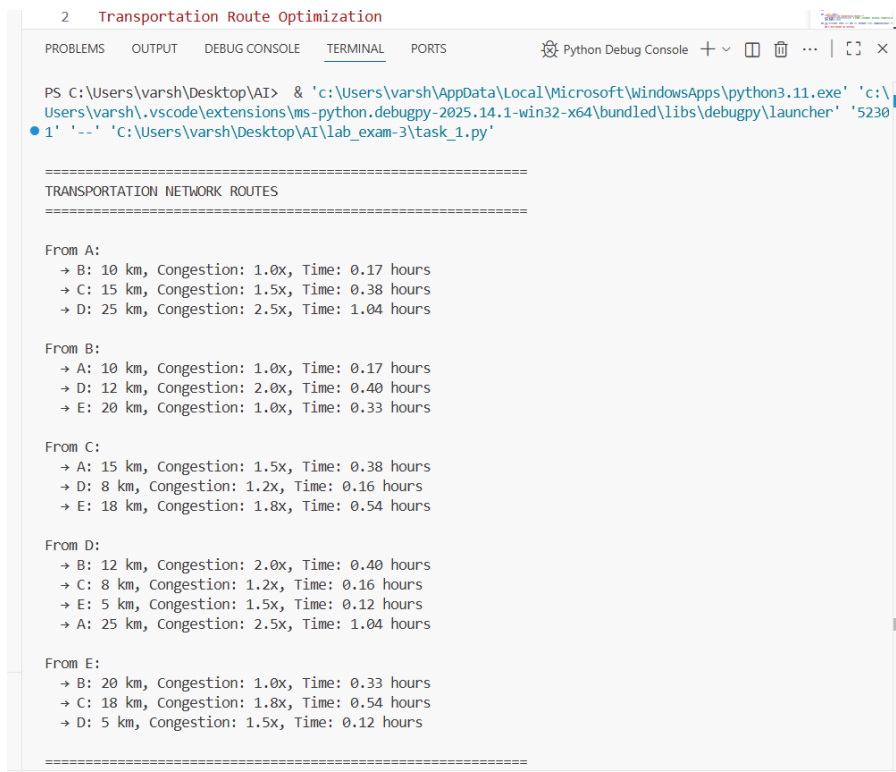
task_1.py +

Add context (#), enter

A v



Output:



```
→ C: 18 km, Congestion: 1.8x, Time: 0.54 hours
→ D: 5 km, Congestion: 1.5x, Time: 0.12 hours
```

```
=====
FINDING FASTEST ROUTE FROM A TO E
=====
```

```
Fastest Route: A → B → E
Travel Time: 0.50 hours (30 minutes)
Total Distance: 30.00 km
```

```
=====
FINDING SHORTEST ROUTE FROM A TO E
=====
```

```
Shortest Route: A → B → D → E
Total Distance: 27.00 km
Travel Time: 0.69 hours (42 minutes)
```

```
=====
ROUTE COMPARISON
=====
```

```
Fastest Route saves 12 minutes compared to shortest route
Fastest route covers -3.00 km less distance
```

```
○ PS C:\Users\varsh\Desktop\AI>
```

Observation:

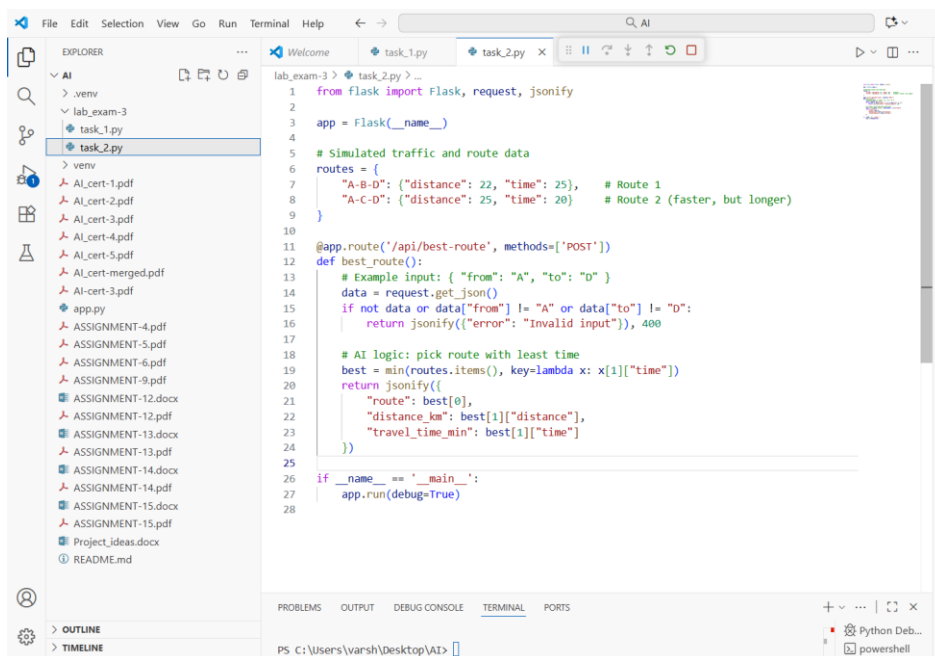
The main idea of the code is to compare two possible routes between locations, the route with the shortest distance and the route with the fastest travel time, considering traffic. AI helps by using data about road traffic to pick the fastest route, instead of just the shortest one. The algorithm checks time and distance for each path and decides which is better for quick transportation. This helps companies avoid delays from traffic and makes deliveries smarter and more efficient.

Q2:

Prompt:

A transportation company wants to find the best route between two places using traffic data. Design a backend API that takes the start and end points, and returns the best route

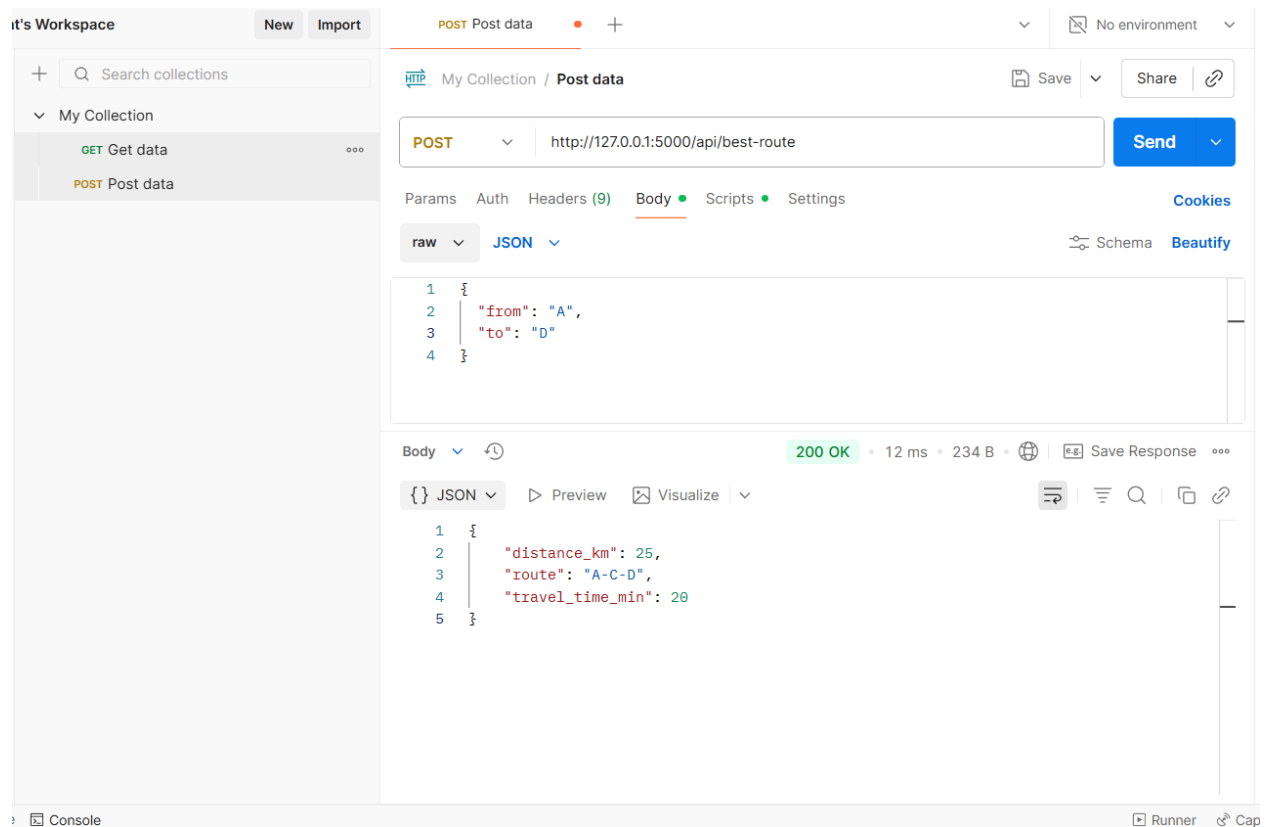
Code:



The screenshot shows a Visual Studio Code editor with a Python Flask application. The Explorer pane on the left shows a project structure with files like `task_1.py` and `task_2.py`. The main editor displays the code for `task_2.py`, which is a Flask application designed to find the best route between two points based on simulated traffic data. The code includes a `Flask` app, a list of routes with distance and time, and a `best_route` function that uses a simple logic to pick the route with the least time. The terminal at the bottom shows the command prompt.

```
1 from flask import Flask, request, jsonify
2
3 app = Flask(__name__)
4
5 # Simulated traffic and route data
6 routes = {
7     "A-B-D": {"distance": 22, "time": 25}, # Route 1
8     "A-C-D": {"distance": 25, "time": 20}  # Route 2 (faster, but longer)
9 }
10
11 @app.route('/api/best-route', methods=['POST'])
12 def best_route():
13     # Example input: { "from": "A", "to": "D" }
14     data = request.get_json()
15     if not data or data["from"] != "A" or data["to"] != "D":
16         return jsonify({"error": "Invalid input"}), 400
17
18     # AI logic: pick route with least time
19     best = min(routes.items(), key=lambda x: x[1]["time"])
20     return jsonify({
21         "route": best[0],
22         "distance_km": best[1]["distance"],
23         "travel_time_min": best[1]["time"]
24     })
25
26 if __name__ == '__main__':
27     app.run(debug=True)
28
```

Output:



Observation:

The backend API helps the company find the best route between two points by using real-time traffic data and AI-assisted decisions. To test the API, we use the URL shown in the terminal after running the Flask app. We will send the required input through Postman or any client, and the API replies with the best route, distance, and travel time. This makes route planning faster and smarter, improving company efficiency.