**NETS 1500 Final Project - Arnav Gattani**

# Arnav's Indian Travel Optimizer User Manual

There are two components to the project: the Interactive and Analysis. Both of these files in the src folder can be run (have a main method) and utilize the same dataset. The user manual will describe the motivation/purpose, and functionality of both the components.

## I. Interactive

### A. Motivation/Purpose (Why use):

As someone who has been very interested in my roots and tradition, I took a gap semester last year to backpack across the Indian subcontintent. While I was travelling, it was hard to find which cities out of the many hundred were worth travelling to and how I could optimize my drive, not only by time, but by the number of miles due to the limitations I had on my rental vehicle. I realized it was so hard on every stretch of my trip depending on where I was to plan out on google maps and tripadvisor, simultaneously weighing the tradeoffs of the quality of tourist attractions in each city with the distance between the cities I wanted to travel to. That's why I decided the create this optimizer specifically for travellers and backpackers like myself who want to go from one city in India to another, but want to do so while stopping at various cities, depdening on the parameters of their own trip.

The interactive program (Interactive.java) allows the user to not only enter a start city and end city (out of the 100 major Indian towns/cities across almost every state and geographic location) to find the optimal path and distance to get there, but also the minimum number of stops between to visit. The program will run a modified Dijkstra's algorithm depending on the minimum number of stops to find the best route in terms of driving distance while also giving you beautiful cities to stop in and roam. Along with this, the user has the option of typing in a city or all of cities along the path given. In this case, the program will print the user various weather information and travel suggestions to give the user the full experience, as they can now travel in comfort.

### B. Functionality (How to use)

The user can run the Interactive program by running Interactive.java. When the program is run the user is asked to input a start city where they will be starting their tour and an end city, where they will want to end their tour. Then, the user will be asked to input the minimum number of stops they want between their start (source) and end city.

```
Welcome to Arnav's Indian Travel Optimizer!
Out of the 100 major cities outlines in the user manual,
please indicate which Indian city you are going to leave from:
Mumbai
Please enter the end city in your journey
Chennai
Please enter the minimum number of cities you want to explore in between your journey!
5
The shortest path between Mumbai and Chennai with a minimum of 5 stops is:
Mumbai --> Malegaon --> Dhule --> Aligarh --> Meerut --> Saharanpur --> Chennai

The total driving distance along this route will be: 3924.5 km or 2438.57 mi

Your itinerary is as follows:
Driving distance between Mumbai and Malegaon is: 269.0 km
Driving distance between Malegaon and Dhule is: 52.5 km
Driving distance between Dhule and Aligarh is: 971.0 km
Driving distance between Aligarh and Meerut is: 146.0 km
Driving distance between Meerut and Saharanpur is: 129.0 km
Driving distance between Saharanpur and Chennai is: 2357.0 km

We are one step closer to planning your ideal trip!
Do you want some more information about these cities?
Type "all" to get information about all these cities, or else just type in the city of your choice!
```

```
please indicate which Indian city you are going to leave from:
Delhi
Please enter the end city in your journey
Faridabad
Please enter the minimum number of cities you want to explore in between your journey!
0
The shortest path between Delhi and Faridabad with a minimum of 0 stops is:
Delhi --> Faridabad

The total driving distance along this route will be: 50.0 km or 31.07 mi
```

If the user sets the minimum number of cities to be 0, then the program runs a basic Dijkstra's algorithm where it could be a direct path such as from Mumbai to Pune, or a more complicated yet optimized path from Srinagar to Hyderabad. (try both these inputs as a test!)

If no such path is found between cities, then the program returns that such a path is not present and it terminates the program.

If a path is found and is printed, the program will then prompt the user for information about cities along the start city to the end city. The user has the liberty to type in the specific city they are looking for information about, or they can type "all" and information regarding **all** the cities along the shortest path will be printed by the program. If an invalid input is entered, the program will reprompt.

```
We are one step closer to planning your ideal trip!
Do you want some more information about these cities?
Type "all" to get information about all these cities, or else just type in the city of your choice!
delhi
Here's some info about Delhi:
Delhi, the capital city of India, is a vibrant and bustling metropolis that blends modernity with history and culture. Here are some of the top tourist attrac
Tourist Attractions: Red Fort - A magnificent fort built in the 17th century, featuring beautiful gardens and Mughal-era architecture.
India Gate - A war memorial and popular picnic spot, featuring a large arch and beautifully lit up at night.
Qutub Minar - A towering minaret built in the 12th century, featuring intricate carvings and beautiful Islamic architecture.
Lotus Temple - A stunning temple in the shape of a lotus flower, built in the 1980s and open to people of all religions.
Jama Masjid - One of the largest mosques in India, built in the 17th century, featuring beautiful domes, minarets, and courtyards.
Weather: Delhi has a semi-arid climate with hot summers and cool winters. The summer season lasts from April to June, with temperatures ranging from 25 to 45

Process finished with exit code 0
```

```
The shortest path between Goa and Srinagar with a minimum of 6 stops is:
Goa --> Ahmadnagar --> Dhule --> Ujjain --> Jaipur --> Ludhiana --> Jalandhar --> Srinagar

The total driving distance along this route will be: 2627.3 km or 1632.53 mi

Your itinerary is as follows:
Driving distance between Goa and Ahmadnagar is: 524.0 km
Driving distance between Ahmadnagar and Dhule is: 258.0 km
Driving distance between Dhule and Ujjain is: 312.0 km
Driving distance between Ujjain and Jaipur is: 512.0 km
Driving distance between Jaipur and Ludhiana is: 534.0 km
Driving distance between Ludhiana and Jalandhar is: 61.3 km
Driving distance between Jalandhar and Srinagar is: 426.0 km

We are one step closer to planning your ideal trip!
Do you want some more information about these cities?
Type "all" to get information about all these cities, or else just type in the city of your choice!
all
Here's some info about Goa:
Goa is a state in western India with coastlines stretching along the Arabian Sea. Its long history as a Portuguese colony prior to 1961 is
Tourist Attractions: Beaches: Goa is known for its pristine beaches, including Anjuna, Baga, Calangute, and Palolem, among many others. The
Forts: Goa has several ancient forts, including Aguada Fort, Chapora Fort, and Reis Magos Fort, among others, that offer a glimpse into the
Spice plantations: Goa is known for its spice plantations, where visitors can learn about the cultivation of spices and herbs and enjoy a t
Churches: Goa has several churches that showcase its Portuguese heritage, including the Basilica of Bom Jesus, Church of St. Francis of Ass
Dudhsagar Falls: A popular waterfall located in the Western Ghats, known for its scenic beauty and trekking trails.
Wildlife sanctuaries: Goa has several wildlife sanctuaries, including the Bhagwan Mahavir Wildlife Sanctuary and Mollem National Park, wher
Weather: The state has a tropical climate, with hot and humid summers and mild winters. The best time to visit Goa is between November and

Here's some info about Ahmadnagar:
Ahmadnagar is is a city situated in the Ahmednagar district, Maharashtra, India, about 120 km northeast of Pune
Tourist Attractions: Ahmednagar Fort - Ahmednagar Fort is a historical fortification located in the heart of Ahmednagar city. The fort was
Salabat Khan's Tomb - Salabat Khan's Tomb is a 16th-century mausoleum located in Ahmednagar. The tomb is an excellent example of Mughal arc
```

```
Goa --> Ahmadnagar --> Dhule --> Ujjain --> Jaipur --> Ludhiana --> Srinagar

The total driving distance along this route will be: 2624.0 km or 1630.48 mi

Your itinerary is as follows:
Driving distance between Goa and Ahmadnagar is: 524.0 km
Driving distance between Ahmadnagar and Dhule is: 258.0 km
Driving distance between Dhule and Ujjain is: 312.0 km
Driving distance between Ujjain and Jaipur is: 512.0 km
Driving distance between Jaipur and Ludhiana is: 534.0 km
Driving distance between Ludhiana and Srinagar is: 484.0 km

We are one step closer to planning your ideal trip!
Do you want some more information about these cities?
Type "all" to get information about all these cities, or else just type in the city of your choice!
Ujjain
Here's some info about Ujjain:
Ujjain is a historical and cultural city located in the central Indian state of Madhya Pradesh. Here are some of the top tourist attractions and information about
Tourist Attractions: Mahakaleshwar Jyotirlinga Temple - A famous Hindu temple dedicated to Lord Shiva, featuring beautiful architecture and a scenic location.
Kaliadeh Palace - A historical palace located on the banks of the Shipra River, featuring beautiful architecture and a serene atmosphere.
Vedh Shala - An ancient observatory dating back to the 18th century, featuring a variety of astronomical instruments and devices.
Chintaman Ganesh Temple - A famous temple dedicated to Lord Ganesha, featuring beautiful architecture and a peaceful atmosphere.
Kal Bhairav Temple - A temple dedicated to Lord Bhairava, known for its unique architecture and religious significance.
Weather: Ujjain has a humid subtropical climate with hot summers and mild winters. The summer season lasts from March to June, with temperatures ranging from 22 t

Process finished with exit code 0
```

Input limitations: As long as the user spells the start and end cities correctly, the program is not case sensitive so, it will either take the input or else tell the user to input a valid city (valid city inputs are found on the last page of this document). The program is not programmed on giving an output when the start and end cities are the same, so the output will be to just tour locally within that same city. Also, the minimum number of stops must be 0, as that is just going to find the most optimized path (shortest driving distance) going from the start city directly to the end city.

```
please indicate which Indian city you are going to leave from:
Goa
Please enter the end city in your journey
Goa
Since you are already in Goa, you should explore local tourist attractions here!
```

### C. How does it work?

I implemented a web scraper similar to the one in class to get city information from https://distancecalculator.globefeed.com/India_Distance_Calculator.asp. I wanted the information in the format “city1 city2 distance”, so I can create a weighted graph where teh cities were nodes and the driving distances were teh weights of directed edges, so I could run graph algirthms to find shortest path between nodes. After extreacting and cleaning the data, I realized that only some cities had driving distance infromation on teh website, and most of them were missing. Manually getting driving diatcnes for about 10,000 combinations of cities is too tedious, so I write a python script which is included in the code provided, which used a Google Maps API to get the rest of teh driving distances and populate the rest of the 3-tuples.

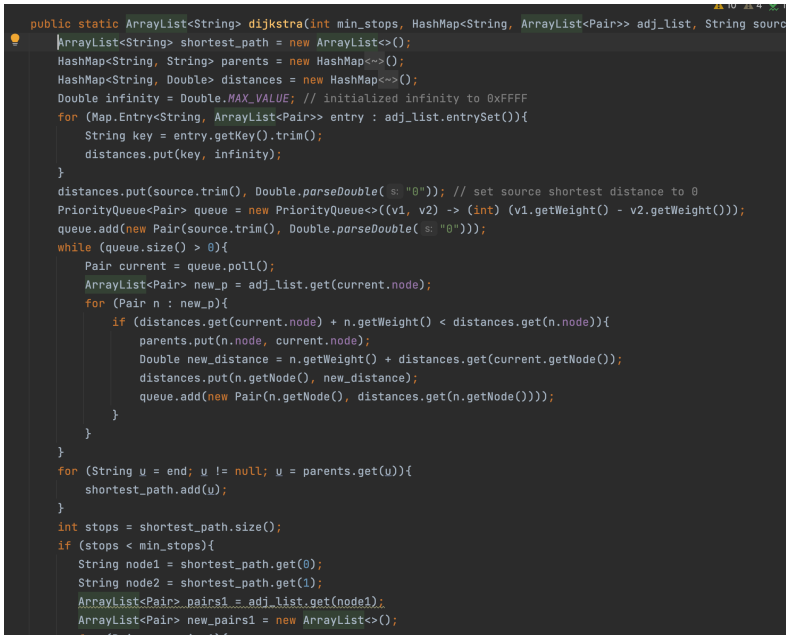
```
# for each combination of cities
for start_node in indian_cities:
  for end_node in indian_cities:
    # node should not have edge to itself as we know the weight is 0 (distance to itself is 0)
    if (start_node != end_node):
      api_key = 'AIzaSyBIpaQ8SlBCNzjRLGCyANufkmt089wbzH0' #API key should be unrestricted
      maps = googlemaps.Client(key=api_key)

      # from distance matrix, we get distance data, the first text field is driving distance between two cities
      drive_dist = maps.distance_matrix(start_node, end_node)['rows'][0]['elements'][0]
      dist_in_km = drive_dist['distance']['text']
      final_dist = dist_in_km.split()[0]

      # print each city (nodes) and then the weight
      # same format as java data so all I do is copy paste the output to data textfile in java program in the data folder
      print(start_node + ' ' + end_node + ' ' + final_dist)
```

```
Haora Gorakhpur 811
Haora Udaipur 1,811
Haora Indore 1,612
Haora Asansol 206
Haora Calicut 2,220
Haora Belgaum 2,103
Haora Amravati 1,293
Haora Chennai 1,650
Haora Thane 1,854
Haora Tirunelveli 2,266
Haora Jaipur 1,574
Haora Mysore 1,995
Haora Hyderabad 1,472
```

This code could be run by anyone as my API key is unrestricted. Then, I copy pasted teh output to get all my data into a text file in IntelliJ. However, many of the pairs were the same as the data I originally collected (had duplicates), so to be safe, my code takes the pair of cities that had the greater driving distance, and adds that to the data structure to take care of duplicates since there cannot be edges from one node to the same node. Then, I created my adjacency list for my graph and I ran a modified Dijkstra's algiorthm which I implemented using a Priority queue (see the dijkstra method in the WebScraper file). This function takes in the minimum number of stops, adjacency list, teh source and end cities, and uses the shortest path algorithm on weighted edges to find teh shortest path ebtween cities. However, if the minimum stops threshold is not reached in the shortest path, then the graph deletes the edges between the first two nodes and reruns the program. Inevitably, the program gives an output when it finds a path with at least that number of stops after Dijkstra finishes running.



```
public static ArrayList<String> dijkstra(int min_stops, HashMap<String, ArrayList<Pair>> adj_list, String sourc
    ArrayList<String> shortest_path = new ArrayList<>();
    HashMap<String, String> parents = new HashMap<~>();
    HashMap<String, Double> distances = new HashMap<~>();
    Double infinity = Double.MAX_VALUE; // initialized infinity to 0xFFFF
    for (Map.Entry<String, ArrayList<Pair>> entry : adj_list.entrySet()){
        String key = entry.getKey().trim();
        distances.put(key, infinity);
    }
    distances.put(source.trim(), Double.parseDouble( s: "0")); // set source shortest distance to 0
    PriorityQueue<Pair> queue = new PriorityQueue<>((v1, v2) -> (int) (v1.getWeight() - v2.getWeight()));
    queue.add(new Pair(source.trim(), Double.parseDouble( s: "0")));
    while (queue.size() > 0){
        Pair current = queue.poll();
        ArrayList<Pair> new_p = adj_list.get(current.node);
        for (Pair n : new_p){
            if (distances.get(current.node) + n.getWeight() < distances.get(n.node)){
                parents.put(n.node, current.node);
                Double new_distance = n.getWeight() + distances.get(current.getNode());
                distances.put(n.getNode(), new_distance);
                queue.add(new Pair(n.getNode(), distances.get(n.getNode())));
            }
        }
    }
    for (String u = end; u != null; u = parents.get(u)){
        shortest_path.add(u);
    }
    int stops = shortest_path.size();
    if (stops < min_stops){
       String node1 = shortest_path.get(0);
       String node2 = shortest_path.get(1);
       ArrayList<Pair> pairs1 = adj_list.get(node1);
       ArrayList<Pair> new_pairs1 = new ArrayList<>();
```

Tourist Attraction and City Information: There are hundred text files found in cityTouristInfo folder in the src folder with a file for each city. The text file contains information for Wikipedia, tourist websites, and AI generated text (openAI) regarding the weather, general information, and tourist attractions and their description for each city. All this information in the text file is then read through a BufferedReader (code is in RouteReader class), so when prompted the program can give information about each city to the user after the optimized path is generated.

## II. Analysis

While the Interactive program is very useful, the methodologies in this program has applications beyond just consumer use and can also be used by analyst and the Indian Tourism board. As the the Indian government is focusing on developing its infrastructure, they want to focus on increase the number of travellers pouring into the country. They realize this can be done by primarily improving the infrastructure of major cities that tourists will transit through and visit as they plan trips across India. This is where such an optimizer comes in handy as it can generally expect which cities would be the best commuter cities

geographically (as optimally most travellers will stop here to see attractions and take a break along their travels), and hence where the Indian government should spend their resources building hotels and facilities to tap into the tourist cash flow for maximal economic benefit.

The listed frequencies for each city are the respective number of times each city was included in a "shortest-path" travel route when the Dijkstra's algorithm was run between all the combination of cities, while keeping the minimum number of stops constant, with it being either 1, 2, and 3. This code takes ~13 minutes to run due to it traversing and running the Dijkstra graph algorithm on approximately thirty thousand pairs of data. The output was similar to as shown below:

```
The frequencies of cities commuted are as follows (for minimum of 1 stops):
frequency of times: Warangal = 83,
frequency of times: Vishakhapatnam = 4,
frequency of times: Vijayawada = 168,
frequency of times: Varanasi = 47,
frequency of times: Vadodara = 505,
frequency of times: Ulhasnagar = 82,
frequency of times: Ujjain = 659,
frequency of times: Tiruppur = 45,
frequency of times: Tirunelveli = 11,
frequency of times: Tiruchchirappalli = 33,
frequency of times: Thiruvananthapuram = 25,
frequency of times: Thane = 5,
frequency of times: Surat = 64,
frequency of times: Solapur = 72,
frequency of times: Shiliguri = 4,
frequency of times: Saharanpur = 207,
frequency of times: Ranchi = 46,
frequency of times: Rajkot = 88,
frequency of times: Raipur = 284,
frequency of times: Quilon = 7,
frequency of times: Pune = 329,
frequency of times: Pimpri = 277,
frequency of times: Patna = 190,
frequency of times: Panihati = 198,
frequency of times: Nellore = 31,
frequency of times: Nasik = 60,
frequency of times: Nagpur = 185,
frequency of times: Mysore = 21,
frequency of times: Mumbai = 14,
frequency of times: Moradabad = 109,
frequency of times: Meerut = 466,
frequency of times: Mangalore = 221,
frequency of times: Malegaon = 750,
```

```
frequency of times: Madurai = 69,
frequency of times: Ludhiana = 127,
frequency of times: Lucknow = 95,
frequency of times: Kota = 103,
frequency of times: Korba = 128,
frequency of times: Kolkata = 23,
frequency of times: Kolhapur = 326,
frequency of times: Kanpur = 430,
frequency of times: Kalyan = 82,
frequency of times: Jodhpur = 26,
frequency of times: Jhansi = 349,
frequency of times: Jamshedpur = 25,
frequency of times: Jamnagar = 17,
frequency of times: Jammu = 1,
frequency of times: Jalgaon = 128,
frequency of times: Jalandhar = 56,
frequency of times: Jaipur = 44,
frequency of times: Jabalpur = 601,
frequency of times: Indore = 32,
frequency of times: Hyderabad = 87,
frequency of times: Hubli = 68,
frequency of times: Haora = 2,
frequency of times: Gwalior = 327,
frequency of times: Guwahati = 2,
frequency of times: Guntur = 84,
frequency of times: Gulbarga = 39,
frequency of times: Gorakhpur = 4,
frequency of times: Goa = 27,
frequency of times: Ghaziabad = 239,
frequency of times: Gaya = 29,
frequency of times: Faridabad = 91,
frequency of times: Durgapur = 198,
frequency of times: Dhule = 688,
frequency of times: Delhi = 130,
frequency of times: Dehra = 11,
```

```
frequency of times: Cuttack = 60,
frequency of times: Coimbatore = 25,
frequency of times: Cochin = 6,
frequency of times: Chennai = 9,
frequency of times: Chandigarh = 15,
frequency of times: Calicut = 29,
frequency of times: Bokaro = 249,
frequency of times: Bikaner = 83,
frequency of times: Bhubaneshwar = 71,
frequency of times: Bhopal = 212,
frequency of times: Bhiwandi = 447,
frequency of times: Bhilai = 235,
frequency of times: Bhavnagar = 276,
frequency of times: Bhatpara = 10,
frequency of times: Bengalooru = 229,
frequency of times: Benares = 87,
frequency of times: Belgaum = 68,
frequency of times: Bareilly = 98,
frequency of times: Aurangabad = 324,
frequency of times: Asansol = 1,
frequency of times: Amritsar = 47,
frequency of times: Amravati = 12,
frequency of times: Allahabad = 257,
frequency of times: Aligarh = 609,
frequency of times: Akola = 66,
frequency of times: Ajmer = 11,
frequency of times: Ahmadnagar = 158,
frequency of times: Ahmadabad = 17,
frequency of times: Agra = 696,
```

```
Analysis × Interactive ×
/Library/Java/JavaVirtualMachines/adoptopenjdk-8.jdk/Contents/Home/bin/java ...
minimum number of stops are: 1
The frequencies of cities commuted are as follows (for minimum of 1 stops):
Warangal=83, Vishakhapatnam=4, Vijayawada=168, Varanasi=47, Vadodara=505, Ulhasnagar=82, Ujjain=659, Tiruppur=45, Tiru
The frequencies of cities commuted are as follows (for minimum of 2 stops):
Warangal=83, Vishakhapatnam=6, Vijayawada=168, Varanasi=47, Vadodara=505, Ulhasnagar=82, Ujjain=659, Tiruppur=45, Tiru
The frequencies of cities commuted are as follows (for minimum of 3 stops):
Warangal=83, Vishakhapatnam=6, Vijayawada=168, Varanasi=47, Vadodara=505, Ulhasnagar=82, Ujjain=659, Tiruppur=45, Tiru
Process finished with exit code 0
```

We can see by the output that the cities of Vadodara, Ujjain, Meerut, Malegaon, Jabalpur, Dhule, Aligarh, and Agra, were by far the cities that were part of the optimal solution for travellers the most, being a stopping destination **> 500 times**, regardless of the minimum stops parameter (one, two, or three) across the combinations of every major city in our dataset.

This makes sense if we analyze this geopolitically as almost all of these cities are located around (within a few hundred km) of the political hub of India (Delhi) => Agra, Aligargh, and Meerut and the financial hub of India (Mumbai) => Vadodara, Malegaon, and Dhule, where naturally most of the development and cities are centered.


While alot of these cities such as Meerut and Agra score high on the high human development and are developing rapidly through metro transit development and increased urbanisation, many towns such as Ujjain and Aligargh have 60-70% literacy rates and are desperately seeking improvement and capital. As tourists will continue to flock and give their economic support to India, India and the Tourism board should lobby and push to build these cities where tourists transit from often, as it will uplift their communities and boost tourism within these commuter cities.

<u>Why Dijkstra's shortest path algorithm works:</u>

Dijkstra's algorithm works by first setting the shortest distance of each node from the source node (start city) as infinity. Then it traverses through the graph, adn "relaxes edges", updating teh shortest distance and predeucessor (parent of each node), depending on if it finds a shortest path from a node to another. This only works with graphs that have positive edge weights, as negative edge weights could craetre a cycle where it will not guarantee a shortest path. The distance array is updated as we relax edges, so we

can find shortest distance from the source node to any oterh specified node. Then, we can also find teh shortest path from end to start node, as ew can use teh parent array which keeps track of the parent node (node with shortest distance), to backtrack from the end node to teh source node. If such a path does not exist, then we output that such a path does not in fact exist.

Future Work:

Despite this dataset containing many south Indian and east Indian cities, they are fairly disconnected when compared to those in central/West India and near the capital region. This is not surprising since the majority of the Indian economic output comes from those regions. However, we could iprove this dataset by adding more towns and cities in cemters that are lacking and see if the results change in the future.

Generally, this dataset and graph is incredibly useful as graph algorithms such as Dijkstra's and even Breath and Depth First search can help describe patterns in travel and where resources should be spent by goevrnemental entities. This is only the beginning and I hope in the future I can build on this program and develop a GUI to make the program more user friendly.

**List of Major Indian Cities allowed as inputs:**

'Haora', 'Gorakhpur', 'Udaipur', 'Indore', 'Asansol', 'Calicut', 'Belgaum', 'Amravati', 'Chennai', 'Thane', 'Tirunelveli', 'Jaipur', 'Mysore', 'Hyderabad', 'Ranchi', 'Ajmer', 'Cuttack', 'Nagpur', 'Ahmadabad', 'Faridabad', 'Guntur', 'Hubli', 'Gulbarga', 'Amritsar', 'Cochin','Srinagar', 'Tiruppur', 'Mumbai', 'Nellore', 'Bareilly', 'Solapur', 'Coimbatore', 'Ahmadnagar', 'Nasik', 'Akola', 'Quilon', 'Jammu', 'Moradabad', 'Dhule', 'Chandigarh', 'Warangal', 'Jalandhar', 'Shiliguri', 'Jodhpur', 'Kalyan', 'Korba', 'Benares', 'Bhilai', 'Aurangabad', 'Jamnagar', 'Allahabad', 'Gwalior', 'Jamshedpur', 'Jhansi', 'Tiruchchirappalli', 'Thiruvananthapuram', 'Ludhiana', 'Bhubaneshwar', 'Jalgaon', 'Gaya', 'Vishakhapatnam', 'Vijayawada', 'Raipur', 'Kolkata', 'Surat', 'Mangalore', 'Saharanpur', 'Aligarh', 'Pune', 'Dehra', 'Bhopal','Lucknow', 'Delhi', 'Guwahati', 'Ujjain', 'Rajkot', 'Madurai', 'Bhatpara', 'Pimpri', 'Kota', 'Kanpur', 'Vadodara', 'Ulhasnagar', 'Bengalooru', 'Durgapur', 'Meerut', 'Patna', 'Ghaziabad', 'Bikaner', 'Malegaon', 'Agra', 'Bokaro', 'Jabalpur', 'Kolhapur', 'Bhavnagar', 'Bhiwandi', 'Panihati', 'Goa', 'Varanasi'