**PROJECT: Smart City IoT Data Platform and Analytics Hub**

**Project Summary:**

The Smart City IoT Data Platform and Analytics Hub project aims to develop a robust system for real-time data collection, processing, and visualization from urban sensors, using AWS services. Python scripts will simulate sensor data for traffic, air quality, energy consumption, and waste management. AWS IoT Core and Kinesis will handle connectivity and data streaming, while AWS Lambda functions will process the data, storing it in Amazon RDS with PostgreSQL. The project incorporates a custom graph for city infrastructure and algorithms for traffic optimization and pollution detection. A RESTful API with AWS API Gateway and Lambda will expose data insights with secure controls. Visualization via Tableau and Power BI will provide real-time insights.

**Technical setup:**

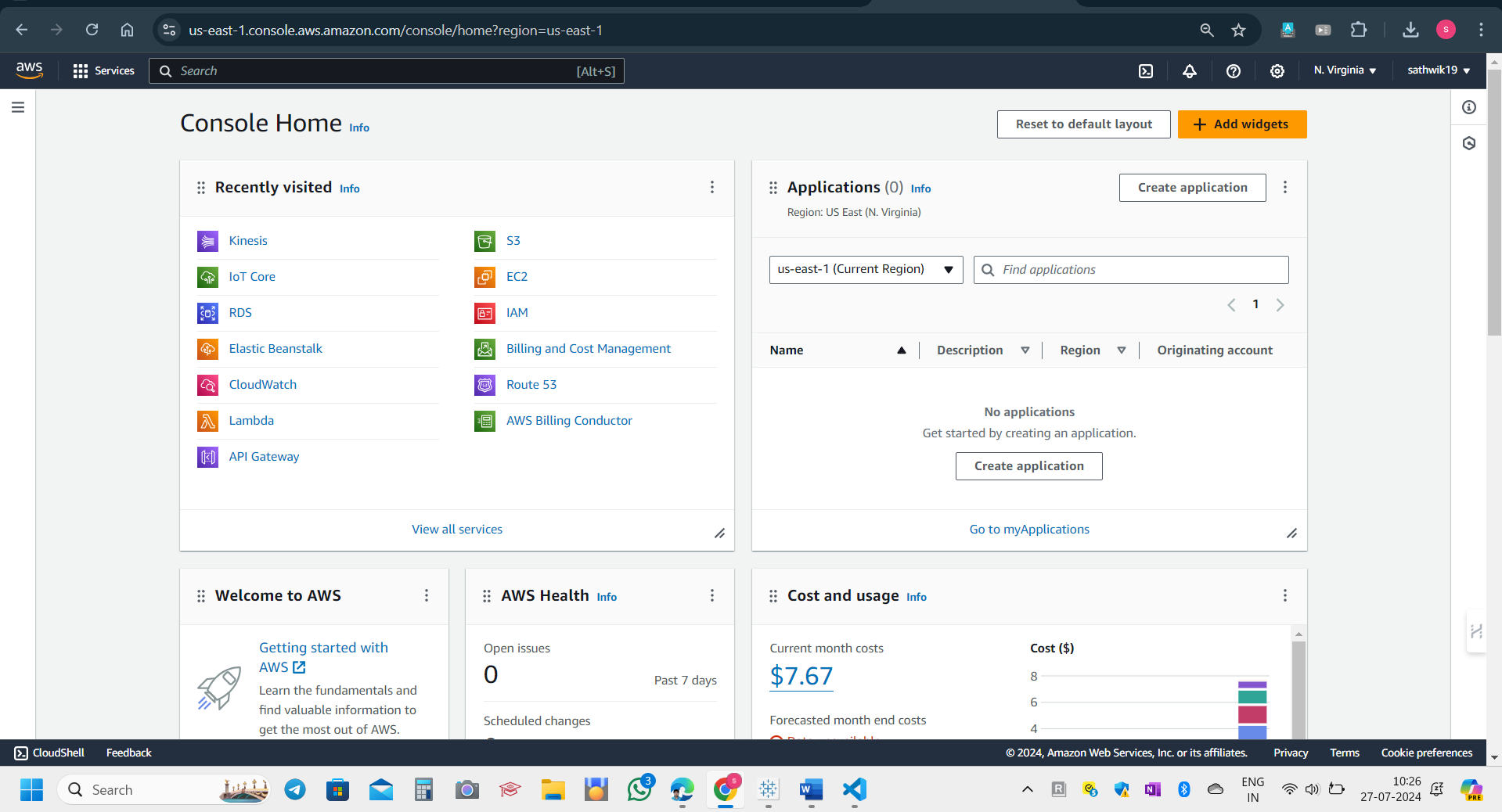
The setup for the Smart City IoT Data Platform and Analytics Hub involves configuring AWS services like IAM for access control, VPC for networking, and EC2 for computing. AWS IoT Core and Kinesis handle device connectivity and real-time data streaming. Data processing is managed by AWS Lambda functions, which clean, normalize, and enrich data before storing it in PostgreSQL on Amazon RDS. Visualization is achieved through Tableau and Power BI dashboards, with real-time monitoring via AWS CloudWatch. Advanced data structures and algorithms enhance traffic optimization and pollution detection.

1. **AWS Setup and configuration:**

The initial task in the "Smart City IoT Data Platform and Analytics Hub" project involves setting up and configuring the necessary AWS environment. This step is crucial as it lays the foundation for all subsequent tasks. The setup includes creating and configuring several AWS services: IAM (Identity and Access Management) to manage user permissions and secure access, VPC (Virtual Private Cloud) for network isolation, EC2 (Elastic Compute Cloud) for scalable computing resources, RDS (Relational Database Service) for managing the PostgreSQL database.

Each service is configured to ensure seamless data flow, security, and high availability. Additionally, PostgreSQL is installed and configured on Amazon RDS to facilitate efficient data storage and management, forming the backbone of the smart city data platform. This comprehensive AWS setup ensures that the project has infrastructure to support the development and deployment of the platform.

I have created an AWS free tier account and set up all the required web services



This is the AWS console and dashboard home page.

A screenshot of a computer

Description automatically generated

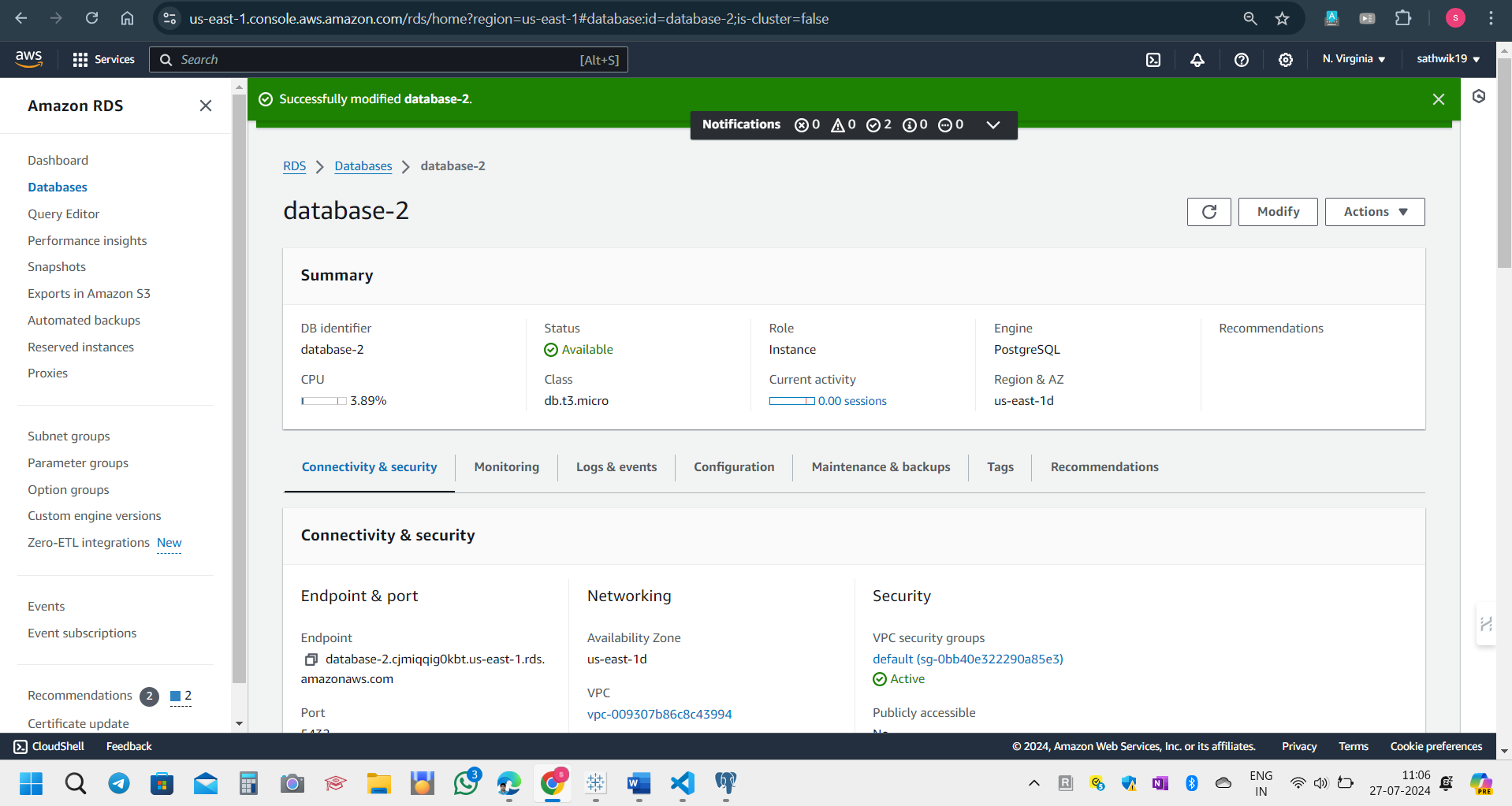
EC2 Instance is running now.

A screenshot of a computer

Description automatically generated

I have created IAM users to allow access to certain activities.

This makes the security of the system and web services effective.A screenshot of a computer

Description automatically generatedz 

I have created a RDS database named **final project**

This gives an end point URL.

I will connect to the PG admin in PostgreSQL to process queries and view data.

**Endpoint connection to pgadmin4:**

A screenshot of a computer

Description automatically generated

I have taken the endpoint URL to workbench and obtained the connection in workbench.

A screenshot of a computer

Description automatically generated

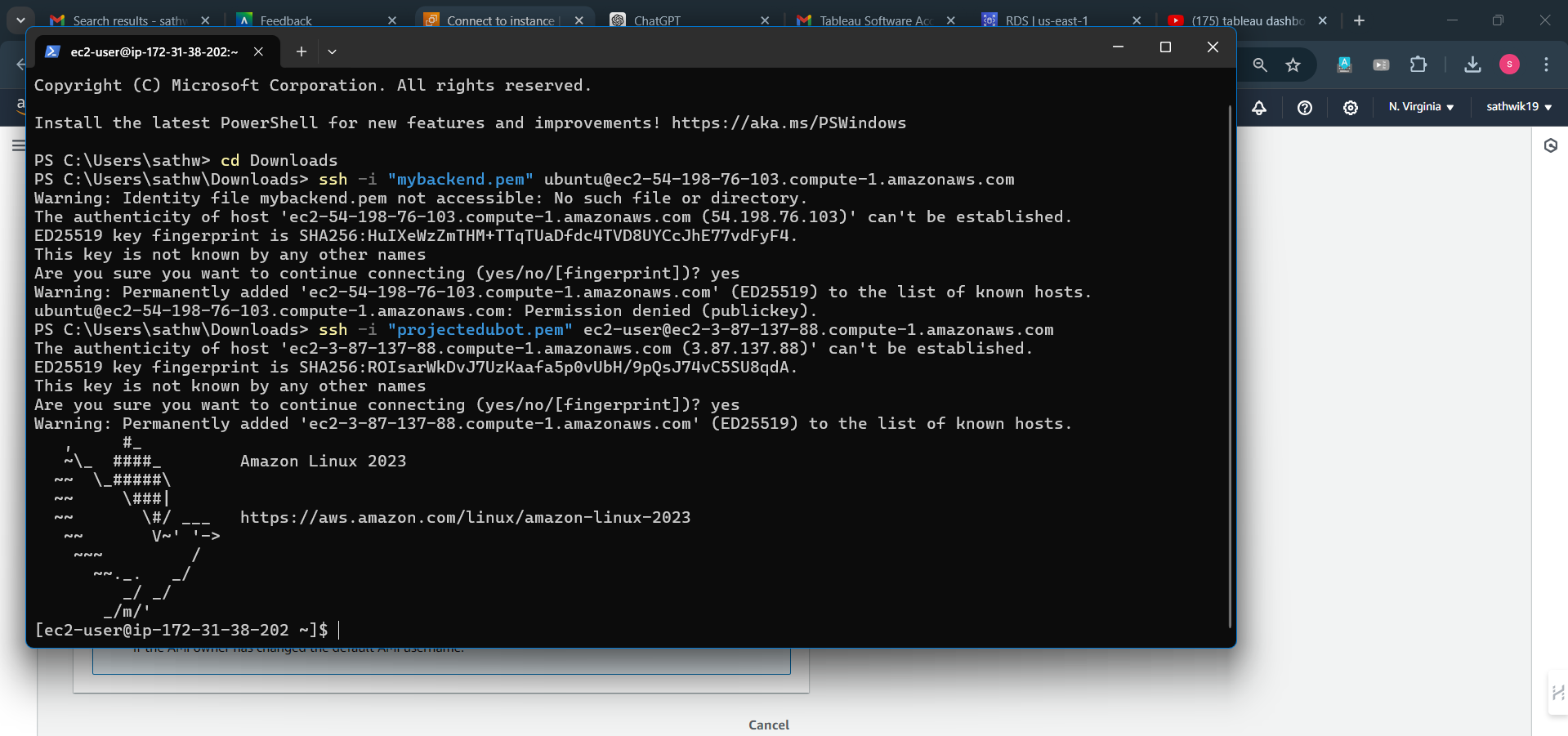
I have connected RDS with pgadmin4 and now this allows to view data and process queries.

1. **Data processing:**

To achieve direct insertion of simulated IoT data into an AWS RDS database, I created a Python script that generates realistic data for traffic flow, air quality, energy consumption, and waste management. The script connects to the RDS instance using pymysql, generates the data, and inserts it into the appropriate tables. I set up the necessary tables in the RDS database using a PostgreSQL script, then executed the Python script on an EC2 instance to automate the data insertion process.

**Simulating real time data:**

To achieve real-time data generation and insertion into an AWS RDS database, I modified the Python script to continuously generate realistic data for traffic flow, air quality, energy consumption, and waste management. The script establishes a connection to the RDS instance, creates the necessary tables, and generates data at regular intervals. Each generated data record is then inserted into the appropriate table in the RDS database. This setup was deployed on an EC2 instance.



I have connected the EC2 instance in the local machine.

I can now run this python script in EC2 instance.

A computer screen with white text

Description automatically generated

I have installed the necessary packages .

sql.connector library allows users to insert data from instances to tables in schema designs.

I have executed the program and data began to insert in the schema of the database.

I have installed pgadmin4 to connect to the PostgreSQL database and view or edit data.

I ran the python script to simuulate the real time data ingestion.

A screenshot of a computer

Description automatically generated

I have copied the endpoint to connect to pgadmin4.

A black background with white text

Description automatically generated

After running the program ,

The script started generating data of four tables.

This script creates complex data fields in the tables and makes the schema a normalized database.

A computer screen shot of a number

Description automatically generated

1. **Database Design and Implementation:**

For the Smart City Data Platform, we designed a normalized database schema to efficiently manage city data, focusing on locations for different areas. Each table includes primary keys, foreign keys, indexes, and constraints for data integrity and performance. Implemented in PostgreSQL, this schema ensures efficient data storage and retrieval, supporting real-time analytics for the city's operations.

The first step is to design a normalized database schema that accurately represents the various data entities and their relationships within the smart city environment, such as traffic flow, air quality, energy consumption, and waste management. This schema includes creating tables, defining primary and foreign keys, and establishing indexes and constraints to ensure data integrity and optimize query performance.

**Database Schema:**

**A screenshot of a computer

Description automatically generated**

This is the database schema for the project connected to pgadmin4.

I can view and process data from this pgadmin4 and that update will be reflected in RDS.

**Database Schema:**

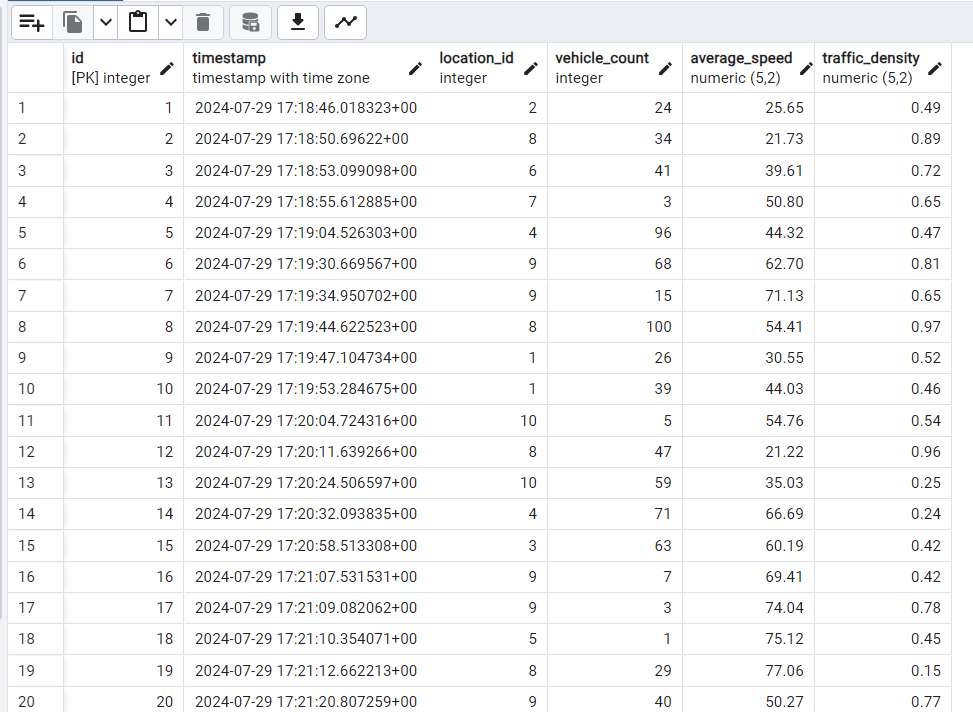
1. **Traffic Flow**

The traffic\_flow table stores data on vehicle movement within the city. It helps in analyzing traffic patterns and managing congestion.

**Table Name:** traffic\_flow

**Columns:**

* + id: Unique identifier for each traffic flow record.
  + timestamp: Date and time of the recorded traffic data.
  + location\_id: Identifier for the location where data is collected.
  + vehicle\_count: Number of vehicles counted in the given time frame.
  + average\_speed: Average speed of the vehicles.
  + traffic\_density: Measure of traffic density in the location.



This is the structureof **traffic\_flow** table.

1. **Air Quality**

The air\_quality table contains data on various air pollutants and the overall air quality index. It is used to monitor and manage air pollution levels in the city.

**Table Name:** air\_quality

**Columns:**

* id: Unique identifier for each air quality record.
* timestamp: Date and time of the recorded air quality data.
* location: Identifier for the location where data is collected.
* PM2\_5: Particulate matter with diameter less than 2.5 micrometers.
* PM10: Particulate matter with diameter less than 10 micrometers.
* NO2: Concentration of Nitrogen dioxide.
* SO2: Concentration of Sulphur dioxide.
* CO: Concentration of Carbon monoxide.
* O3: Concentration of Ozone.
* AQI: Air Quality Index.

A screenshot of a computer

Description automatically generated

This is the structure of **air\_quality** table.

1. **Energy Consumption**

The energy\_consumption table records energy usage data across different sectors. It is useful for analyzing energy consumption patterns and optimizing energy resources.

**Table Name:** energy\_consumption

**Columns:**

* + id: Unique identifier for each energy consumption record.
  + timestamp: Date and time of the recorded energy usage data.
  + location\_id: Identifier for the location where data is collected.
  + household\_energy\_usage: Energy usage in households.
  + commercial\_energy\_usage: Energy usage in commercial establishments.
  + industrial\_energy\_usage: Energy usage in industrial sectors.
  + solar\_energy\_generated: Amount of solar energy generated.
  + wind\_energy\_generated: Amount of wind energy generated.

A screenshot of a computer

Description automatically generated

This is the structure of **energy\_consumption** table.

1. **Waste Management**

The waste\_management table captures data on waste collection and disposal. It aids in tracking waste management efficiency and recycling efforts.

* **Table Name:** waste\_management
* **Columns:**
  + id: Unique identifier for each waste management record.
  + timestamp: Date and time of the recorded waste management data.
  + location\_id: Identifier for the location where data is collected.
  + waste\_collected: Total amount of waste collected.
  + recycled\_waste: Amount of waste recycled.
  + organic\_waste: Amount of organic waste collected.
  + hazardous\_waste: Amount of hazardous waste collected.

A screenshot of a computer

Description automatically generated

This is the structure of **waste\_management** table.

1. **Advanced Data Structures and Algorithms:**

To create a graoh based data structure,I have installed all the required packages to build a complex data structure.I have used these libraries.

**Libraries Used**

**networkx:**

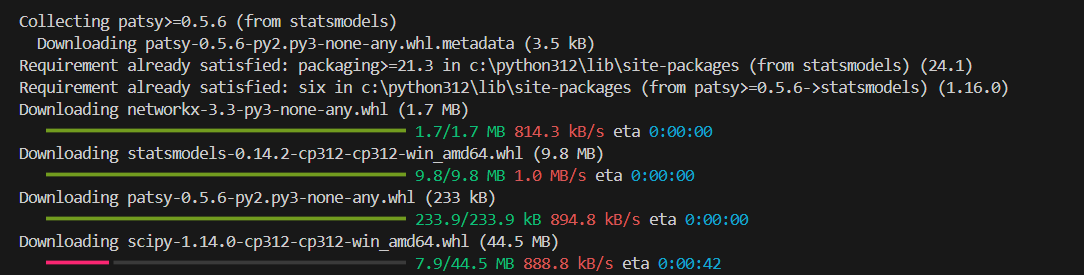
Used for creating and managing the city's graph-based infrastructure, and implementing shortest path algorithms like Dijkstra's.

**pandas:**

Utilized for handling and preprocessing time series data, such as historical air quality data.

**statsmodels:**

Employed for time series forecasting using the SARIMAX model to predict future air quality indices.



**Explanation of the Script**

**1. Graph-Based Data Structure and Sensor Integration**

**Graph Creation:**

* The script utilizes the networkx library to create a directed graph (DiGraph) that models the city's infrastructure. This graph comprises nodes (representing intersections) and edges (representing roads connecting these intersections). Using a graph allows efficient representation and manipulation of the city's network.

**Adding Roads:**

* The add\_road method is responsible for adding edges between nodes in the graph. Each edge has attributes such as length, capacity, speed\_limit, and flow. These attributes are crucial for simulating realistic traffic conditions:
  + length represents the distance of the road segment.
  + capacity indicates the maximum number of vehicles the road can handle.
  + speed\_limit specifies the maximum allowed speed on the road.
  + flow denotes the current traffic flow on the road.

**Adding Sensors:**

* The add\_sensor method attaches sensor data to specific nodes in the graph. This data includes traffic data (like flow and speed) and air quality data (like AQI). Sensors provide real-time data essential for monitoring and managing city infrastructure:
  + Traffic sensors help in tracking vehicle flow and speed.
  + Air quality sensors monitor pollution levels, represented by AQI.

**Updating Sensor Data:**

* The update\_sensor\_data method allows the sensor data to be updated dynamically. If the sensor data at a node changes over time, this method updates the node's sensor attributes, ensuring the graph always has the latest data.

**2. Optimized Path Calculation**

**Travel Time Calculation:**

* The calculate\_travel\_time function computes the travel time for each road segment. It adjusts the speed based on the traffic flow relative to the road’s capacity, reducing the speed if the flow exceeds capacity:
  + If the flow is less than the capacity, the speed remains at the speed limit.
  + If the flow exceeds the capacity, the speed is proportionally reduced, simulating congestion.

**Shortest Path Algorithm:**

* The dijkstra\_optimized\_path function uses Dijkstra's algorithm (provided by networkx) to find the shortest path between two intersections. The weight of each path is determined by the calculated travel time, which is dynamically adjusted based on real-time traffic conditions:
  + The function calculates the travel time for each road segment in the path.
  + It then sums these times to determine the total travel time for the path, optimizing the route based on current traffic data.

**3. Air Quality Prediction**

**Time Series Analysis:**

**Forecasting:**

* The function fits the SARIMAX model to the historical data and generates predictions for a specified number of future periods. This can help city planners anticipate air quality changes and implement mitigation strategies:
  + The model is trained on the historical data.
  + Predictions are made for the future periods, providing a forecast of AQI level.

1. import networkx as nx
2. import pandas as pd
3. import matplotlib.pyplot as plt
4. from statsmodels.tsa.statespace.sarimax import SARIMAX
5. class CityGraph:
6. def \_\_init\_\_(self):
7. self.graph = nx.DiGraph()
8. def add\_road(self, from\_node, to\_node, length, capacity, speed\_limit, flow=0):
9. self.graph.add\_edge(from\_node, to\_node, length=length, capacity=capacity, speed\_limit=speed\_limit, flow=flow)
11. def add\_sensor(self, node, sensor\_type, sensor\_data):
12. if node not in self.graph:
13. self.graph.add\_node(node)
14. self.graph.nodes[node][sensor\_type] = sensor\_data
15. def update\_sensor\_data(self, node, sensor\_type, sensor\_data):
16. if node in self.graph and sensor\_type in self.graph.nodes[node]:
17. self.graph.nodes[node][sensor\_type].update(sensor\_data)
18. else:
19. self.add\_sensor(node, sensor\_type, sensor\_data)
21. def get\_graph(self):
22. return self.graph
23. def calculate\_travel\_time(u, v, edge\_data):
24. flow = edge\_data.get('flow', 0)
25. capacity = edge\_data.get('capacity', 1)
26. if flow < capacity:
27. speed = edge\_data['speed\_limit']
28. else:
29. speed = edge\_data['speed\_limit'] \* (capacity / flow)
30. travel\_time = edge\_data['length'] / speed
31. return travel\_time
32. def dijkstra\_optimized\_path(graph, start\_node, end\_node):
33. path = nx.shortest\_path(graph, start\_node, end\_node, weight=calculate\_travel\_time)
34. travel\_times = [calculate\_travel\_time(u, v, graph[u][v]) for u, v in zip(path[:-1], path[1:])]
35. total\_travel\_time = sum(travel\_times)
36. return path, total\_travel\_time
37. def predict\_air\_quality(sensor\_data, periods=5):
38. air\_quality\_series = pd.Series(sensor\_data)
40. model = SARIMAX(air\_quality\_series, order=(1, 1, 1), seasonal\_order=(1, 1, 1, 12))
41. try:
42. model\_fit = model.fit(disp=False)
43. forecast = model\_fit.forecast(steps=periods)
44. except ValueError as e:
45. print(f"Error fitting model: {e}")
46. forecast = pd.Series([None]\*periods)
47. return forecast
48. def visualize\_graph(graph):
49. pos = nx.spring\_layout(graph)
50. plt.figure(figsize=(10, 8))
51. nx.draw(graph, pos, with\_labels=True, node\_size=700, node\_color="skyblue", font\_size=10, font\_weight="bold", edge\_color="gray")
53. edge\_labels = {(u, v): f"{d['length']}km" for u, v, d in graph.edges(data=True)}
54. nx.draw\_networkx\_edge\_labels(graph, pos, edge\_labels=edge\_labels, font\_color="red")
56. plt.title("Traffic intersection graph")
57. plt.show()
58. if \_\_name\_\_ == "\_\_main\_\_":
59. city\_graph = CityGraph()
60. roads = [
61. ('Intersection\_1', 'Intersection\_2', 2.5, 2000, 60, 1500),
62. ('Intersection\_2', 'Intersection\_3', 3.0, 1800, 50, 1600),
63. ('Intersection\_3', 'Intersection\_4', 1.2, 1500, 40, 1400),
64. ('Intersection\_4', 'Intersection\_5', 4.0, 2200, 70, 2000),
65. ('Intersection\_5', 'Intersection\_1', 3.5, 2000, 60, 1800),
66. ('Intersection\_2', 'Intersection\_5', 2.8, 1900, 55, 1700),
67. ('Intersection\_1', 'Intersection\_4', 3.6, 2100, 65, 1900),
68. ('Intersection\_3', 'Intersection\_5', 1.5, 1600, 45, 1500)
69. ]
70. for road in roads:
71. city\_graph.add\_road(\*road)
73. sensors = [
74. ('Intersection\_1', 'traffic', {'flow': 150, 'speed': 50}),
75. ('Intersection\_2', 'air\_quality', {'AQI': 45}),
76. ('Intersection\_3', 'traffic', {'flow': 300, 'speed': 40}),
77. ('Intersection\_4', 'air\_quality', {'AQI': 60}),
78. ('Intersection\_5', 'traffic', {'flow': 250, 'speed': 55}),
79. ('Intersection\_2', 'traffic', {'flow': 200, 'speed': 48}),
80. ('Intersection\_4', 'traffic', {'flow': 350, 'speed': 38}),
81. ('Intersection\_1', 'air\_quality', {'AQI': 50})
82. ]
83. for sensor in sensors:
84. city\_graph.add\_sensor(\*sensor)
85. G = city\_graph.get\_graph()
86. print("Graph Nodes:", G.nodes(data=True))
87. print("Graph Edges:", G.edges(data=True))
89. visualize\_graph(G)
91. shortest\_path, total\_travel\_time = dijkstra\_optimized\_path(G, 'Intersection\_1', 'Intersection\_4')
92. print("Optimized path from Intersection\_1 to Intersection\_4:", shortest\_path)
93. print("Total travel time:", total\_travel\_time)
95. air\_quality\_data = [45, 48, 50, 47, 49, 52, 54, 55, 58, 60, 63, 65, 67, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90]
96. predicted\_aqi = predict\_air\_quality(air\_quality\_data)
97. print("Predicted AQI for next periods:", predicted\_aqi)

I have implemented the code in python and these are the results:

A screenshot of a computer

Description automatically generated

I have saved the file named **smart\_city.py.**

**Result Output**

1. **Graph Nodes and Edges**:
   * Displays the nodes and edges of the city's infrastructure graph, including traffic and air quality sensor data.
2. **Optimized Travel Path**:
   * Shows the shortest path between specified intersections, considering real-time traffic conditions.
3. **Air Quality Predictions**:
   * Provides future AQI values based on historical data, helping in proactive air quality management.
4. **API Development:**

**Purpose:**

The purpose of this PHP API script is to provide a secure, efficient way to access and retrieve data from a MySQL database on AWS RDS. It allows authorized users to query specific tables via HTTP GET requests, with validation for passkey and table names to ensure security. The API handles errors, returns data in JSON format, and offers a scalable solution for data management and integration with various applications**.**

I have created an API to see data insights and view the data in RDS.

I have written a PHP script to create API in elastic beanstalk.

A screenshot of a computer

Description automatically generated

I ran this script in local machine.

The API is created, and elastic beanstalk environment is deployed.

The API shows data by URL provided.

A screenshot of a computer

Description automatically generated

**Data Retrieval:**

When a valid request is made with the correct passkey and table name, the API will return data from the specified table in JSON format.

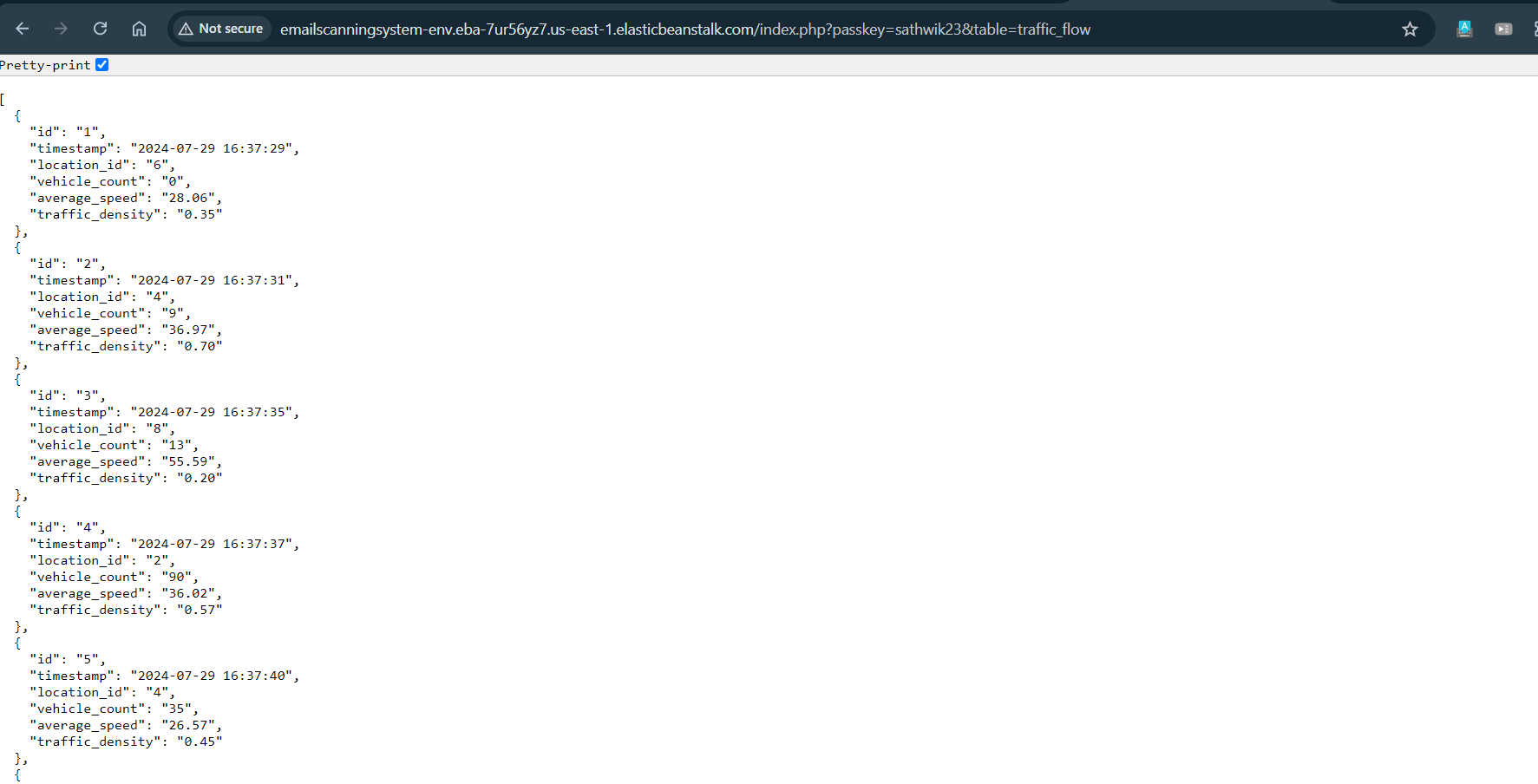
Implementation:



I have provided the passkey and table name to the API endpoint.

This will check if passkey is true and retrieves data from the RDS.

This will display data in json format.



This is the data of **traffic\_flow** table retrieved by API.

1. **Visualization and Reporting**

I have created dashboards in Tableau to effectively visualize and analyse complex data from my smart city project. These dashboards transform raw data into interactive, user-friendly visualizations, making it easier to understand and interpret key metrics.

They provide real-time insights into critical areas such as traffic conditions, air quality, and energy and waste management, enabling stakeholders to monitor performance, identify trends, and make informed decisions quickly. By supporting detailed analysis and uncovering patterns, these dashboards facilitate strategic planning and operational improvements, enhancing urban management and overall quality of life.

a. **Real-Time Traffic Flow Dashboard:**

Displays live traffic conditions and congestion levels across the city. It visualizes traffic flow and speed data from various intersections, highlighting congestion hotspots and enabling real-time traffic management.

b. **Air Quality Dashboard:**

Provides an overview of air quality across different regions. It shows current AQI levels, trends, and forecasts based on historical data, helping monitor pollution levels and assess air quality improvements or deteriorations.

c. **Energy Consumption Dashboard:**

Monitors energy consumption patterns. It visualizes data on energy usage and identifying areas for optimization and tracking the effectiveness of strategies.

**Advantages of visualization:**

* Dashboards transform complex data into interactive, easy-to-understand visuals, simplifying data interpretation.
* They provide immediate access to critical metrics, including traffic conditions, air quality, and resource management.
* By offering comprehensive views of various data aspects, the dashboards contribute to better urban management and an enhanced quality of life.

**Traffic flow dashboard:**

**A map of the world

Description automatically generated**

**Heat Map:**

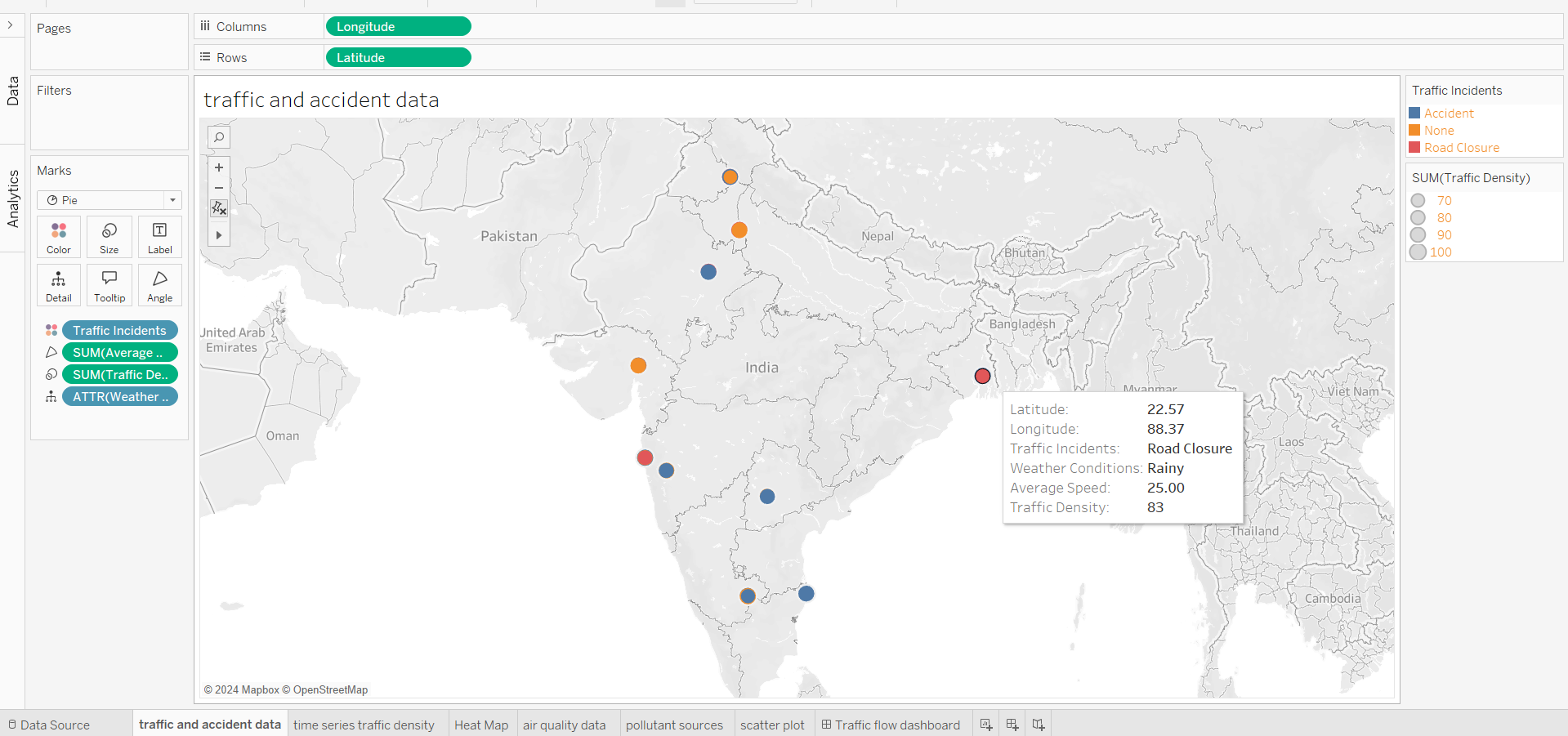
Displays a heat map highlighting areas with high traffic density. This visualization helps identify traffic congestion hotspots and patterns over time.

A screenshot of a computer

Description automatically generated

**Time Series Traffic Density:**

A time series chart displaying changes in traffic density over specific periods. This worksheet provides insights into daily, weekly, or seasonal traffic patterns.



**Traffic and Accident Data:**

Contains detailed data on traffic incidents and accidents, including location, severity, and frequency. This information is crucial for identifying dangerous areas and improving road safety.

**Dashboard:**

**A screenshot of a map

Description automatically generated**

**Air quality dashboard:**

**A map of the world

Description automatically generated**

**Pollution Levels by Region**:

Displays pollution levels across various regions. It helps identify areas with poor air quality and monitor changes over time.

A graph of data showing the average level of air comparison

Description automatically generated with medium confidence

**Air Quality Index (AQI) Trends:**

A time series chart showing AQI trends over time. This visualization helps track improvements or deteriorations in air quality.

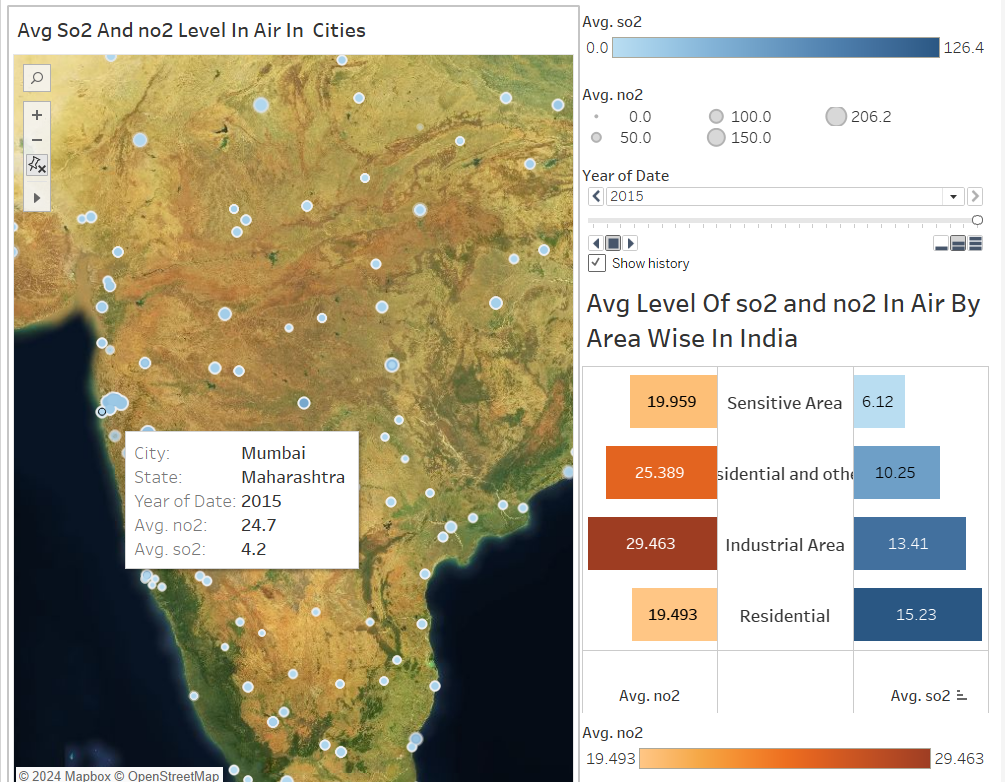
A screenshot of a computer

Description automatically generated

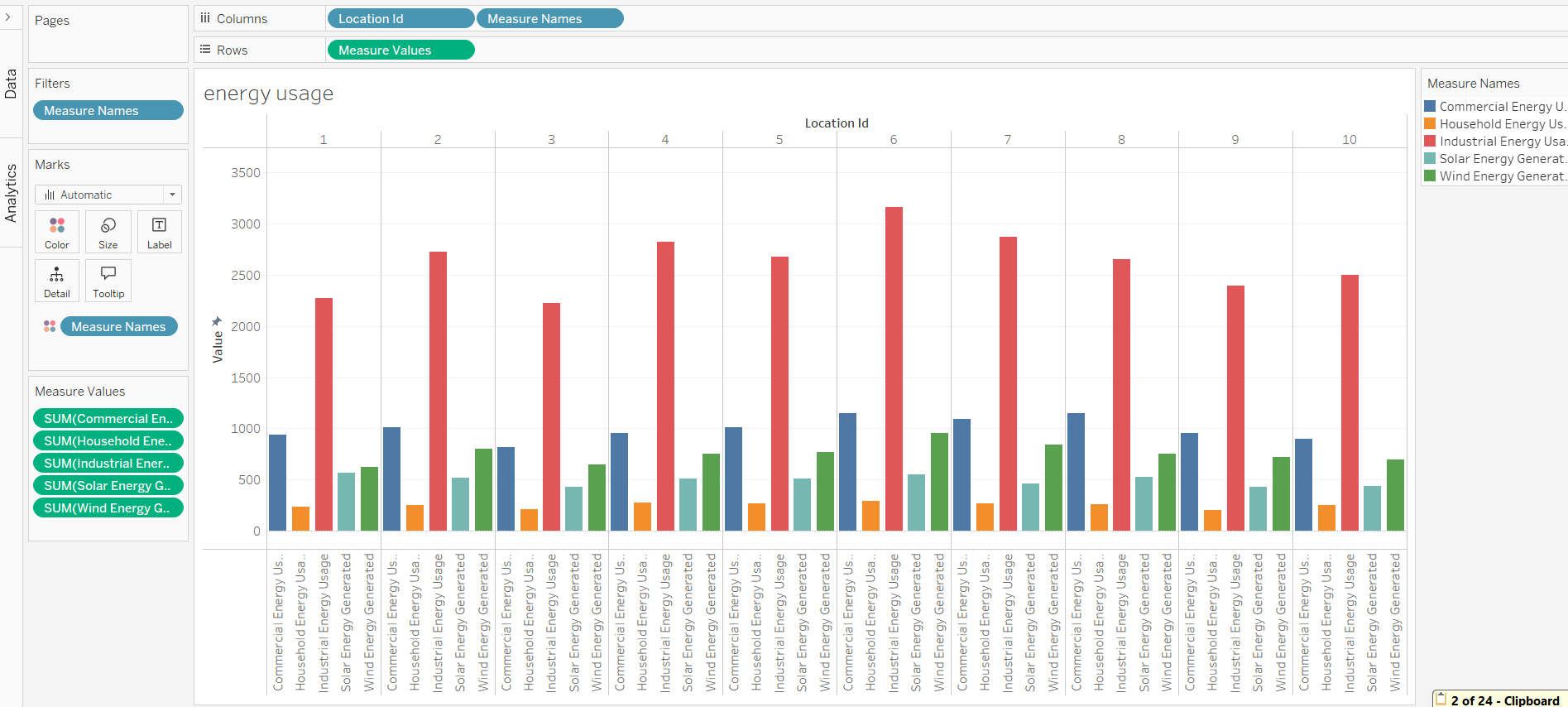
**Sources of Pollution:**

Lists and visualizes different sources of pollution, such as traffic, industry, and natural sources. It helps understand the impact of each source on overall air quality.

**Dashboard:**

****

**Energy Consumption Visualization:**

****

**Energy consumption and generation bar chart:**

It segments data across commercial, household, and industrial sectors, offering valuable insights into their respective energy usage patterns. Additionally, it highlights renewable energy contributions from solar and wind sources. With geographic categorization through location identifiers, this worksheet serves as a tool for understanding and optimizing energy consumption.